# Integrating Computer Science Into Middle School Mathematics

by

Chris Brown

under the supervision of Dr. Susan Rodger,

Department of Computer Science, Duke University

April 2013

## 1. Introduction and Background of Project

Computer science education is, as the Association for Computing Machinery (ACM) and the Computer Science Teachers Association (CSTA) put it, at a "crisis" in K-12 education [21, p. 6]. Since 2000, The NCWIT (National Center for Women and Information Technology) states that the percentage of students entering college who intend to major in computer science has decreased over 70% [10]. Even though the exposure to computers and technology in the classrooms is growing rapidly, very few middle and high schools offer computer science courses. This results in a lack of interest in the subject, which could be damaging to the United States in the future. The ACM predicts that America will be able to fill less than a third of the total technology jobs with American citizens in 2018 [21, p. 25]. This would result in more jobs going overseas, if they will even be filled at all.

Since most schools would be hard-pressed to create a new computer science class, we believe that integrating computer science skills and computational thinking into the classes and curriculum that already exist will be better for exposing the students to computing. Computational thinking refers to the thought process behind problem solving. Jeannette Wing of Carnegie Mellon writes about computational thinking, saying, "[It] is a fundamental skill for everyone, not just computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" [22, p. 33]. Computational thinking is important to help students think though a problem and asking "What's the best way to solve it?" [22, p. 33] Wing also provides many different practical examples of how we use it that don't involve programming: from finding a lost pair of mittens or choosing a line at the grocery store to authorizing humans on the Internet. The tool that we are using to implement computer science and computational thinking into K-12 education is called Alice. Alice [3] in particular allows students to be creative and build 3D animations to tell stories and play games. The Adventures in

Alice Programming Project at Duke has already created many Alice worlds, tutorials, and other educational materials that can be used in a variety of subjects including Math, English, Science, History, Foreign Languages, Technology and Computer Applications, Business, and more for students from elementary to high school. All of these are available for free on the Duke Adventures in Alice Programming website [11]. We have hosted summer workshops for teachers to learn how to use Alice, to show them the curriculum materials that we have created, and to have teachers create Alice lesson plans for their classrooms. Using a program such as Alice to integrate into K-12 education is one solution to increasing the interest in computer science and fixing the technology education crisis.

For this project specifically, we are looking at ways to integrate Alice into middle school math education. We created Alice worlds for middle school math concepts, made Alice tutorials to build specific math projects in Alice and to teach Alice programming concepts, created Math Challenges to help students practice math skills and program simultaneously, mapped our Alice materials to the Common Core Mathematics Standards for grades 5-12, mapped our Alice materials to the CSTA Computer Science Standards for Level 2 and Level 3 ($6^{th}$ – $8^{th}$ and $9^{th}$ – $10^{th}$ grades), visited a middle school to see how students responded to learning Alice, presented at the Association for Computing Machinery's Special Interest Group on Computer Science Education (SIGCSE) conference, and hosted an Alice Activity Day at Duke University for local $6^{th}$ grade students.

## 2. Related Work

### 2.1 Integrating Computer Science into K-12 Education

There are many other programs and curricula that are also trying to integrate computing into K-12 education. The National Science Foundation is working on the CS 10K Teachers Project, where they are trying to get 10,000 computer science teachers into 10,000 high schools

by 2015 [10]. Another example is Computer Science Unplugged [8], which strives to integrate programming techniques and algorithms without actually using computers, but by implementing other activities and concepts such as having students move around to learn a variety of searching and sorting algorithms. Their activities are very hands-on and interactive. Scratch is another program that works to integrate computer science into K-12 education. Scratch is similar to Alice in that it uses a drag-and-drop interface and uses commands to create animations and games, but different as it is in a 2D environment rather than a 3D space. Scratch was created at MIT to reach out to younger children from ages 8-16, and its purpose is to introduce programming to those with no previous experience [16, p. 2]. The Scratch website [18] allows users to share their programs, view tutorials and examples of other projects, receive and provide feedback on posted projects, and more. A third example of a program that helps introductory programmers is Greenfoot which was created by the Programming Education Tools Group at the University of Kent in the UK. Greenfoot [13] is meant to teach children 14 and older how to program and it also uses a visual and interactive interface like Alice and Scratch. It is more complicated than just dragging and dropping instructions into a method, but it is still designed to have a simple programming approach meant for beginners to help them eventually transfer into other programming environments.

Alice, Scratch, and Greenfoot are all similar in their purposes and goals in allowing younger students to explore computer programming, focusing on engaging the user and interactivity, making strides to integrate programming into K-12 education, as well as gaining interest among minority and female students [20]. The paper "Alice, Greenfoot, and Scratch – A Discussion" gathers the leaders of these three programs together to examine the common themes and differences between each programming environment.

Robotics has also been used to help integrate computer science into K-12 education to help students gain interest in programming and computing. Lego Mindstorm has exposed students to programming through robotics and created curriculum and educational materials to be integrated with projects in classrooms. The NXT Base software for Lego Mindstorm is specifically targeted to middle school students to help them learn programming and can be implemented into a variety of subjects such as math, science, technology, and engineering [15]. The Lego robotics approach allows students to creatively design and build robots and encode various instructions for the robot to complete.

### 2.2 Integrating Alice into K-12 Education

Alice is a 3D programming environment for beginner programmers to learn the basics of coding. It was created in 1995 at the University of Virginia by Randy Pausch, who moved to Carnegie Mellon in 1997. Alice was first used for virtual reality and was later adapted with the drag-and-drop interface to use with novice programmers [17]. This program is easy and fun to use because it contains a large library of 3D objects and characters. It uses many computer science programming concepts such as methods, objects, loops, conditional statements, functions, variables, parameters, and more. It has a drag-and-drop interface so no typing code is involved, and it is useful for building games, animations, and telling stories. Alice is a virtual programming world that appeals to younger children and allows them to learn about computer programming.

Several places are working to integrate Alice into pre-college education to help students get exposure to programming concepts. In the Virginia Beach School District in 2006, Alice was taught in their introductory computer science class and over the course of 4 years they saw the number of students taking this class triple [7]. In addition, the number of students taking the AP Computer Science class also tripled across the school district, including a 25% increase of women and 20% increase of minorities. For all of the sites, teachers who participated or interacted with

Alice enjoyed working with Alice and stated that it met their needs. 90% planned to continue using Alice in their classrooms with their students.

In San Jose, Alice has been used as a critical thinking model of TPACK (Technology, Pedagogy, and Content Knowledge) into the 9 core subjects and 4 interdisciplinary themes as stated by P21 (Partnership for 21st Century Skills) for middle school students from grades 5-8 [19]. These subjects are English, reading or language arts; world languages; arts; mathematics; economics; science; geography; history; and government and civics, and the four interdisciplinary themes are global awareness; financial, economic, business and entrepreneurial literacy; civic literacy; and health literacy. Alice can be integrated into these subjects by helping the students design projects and interactive Alice worlds dealing with these topics and using critical thinking to come up with possible solutions or applications of them.

The Duke University Alice team has been running one-week and two-week workshops for teachers in the summer every year since 2008. For these workshops, we have developed many tutorials and materials to help teachers learn Alice programming concepts. In 2009, Duke University hosted an Alice Symposium where many teachers from all over the United States came together to see what others were doing with Alice as well as present and how it could be used in the education field [1]. There were several presentations, papers, and other materials submitted to display how Alice could be used in various curricula. We will also be hosting another Alice Symposium this summer, June 17-21 2013. In summer 2012, we ran two teacher workshops over the summer. 25 teachers attended the beginner two-week workshop and 9 attended a week long follow-up workshop. At these workshops we introduced the teachers to Alice, showed them some of the materials we developed earlier in the summer, demoed other Alice worlds, taught the teachers how to program and build worlds in Alice by going through tutorials, and gave them an

opportunity to create and present possible lesson plans to show how they would use Alice in their classes.

### 2.3 Integrating Computer Science into K-12 Math

There are also many projects working on integrating computing into K-12 mathematics curriculum. One example of this is Bootstrap. Bootstrap is a curriculum that teaches students how to write code and program. It uses mathematics that students must learn based on their school's educational standards, i.e. algebra and geometry, to create animations and video games for them to play. Bootstrap focuses on integrating computer programming into math and technology classes around the world and can be applied to a variety of courses from grades 6 - 12. This not only helps the students learn the mathematic concepts in a fun and interactive way, but teaches them good coding skills and habits in the process. Most children really enjoy playing video games and watching animations, so they will be eager to participate in a course that uses the Bootstrap curriculum. Emmanuel Schanzer, the creator of Bootstrap, also provides various lesson plans and activities for students on the website and a table that maps these lesson plans to the Common Core standards as well as math educational standards for various states in the US [4]. This makes it much easier for teachers to implement this program into their classes without having to rush through other material or set aside extra time to use it. Most classes use Bootstrap online with the WeScheme IDE and server, so no downloads are necessary. The purpose of Bootstrap is very similar to the Alice project at Duke, where we are trying to integrate computational thinking and programming into middle school and high school curricula in a fun way so that more students will gain more interest in and exposure to computer science and the United States will avoid a crisis in computer science education.

Another example is iMPaCT-Math (Media Propelled Computational Thinking for Mathematics Classrooms), which is a program that allows students to learn and study math through graphical programming and computational thinking. This curriculum focuses on using video game design and project-based learning to help these students engage in mathematics, specifically algebra [14]. iMPaCT-Math originated at the University of Texas-El Paso. The group turned their attention to Algebra 1 because it is an essential class for all STEM (Science, Technology, Engineering, and Mathematics) subjects. The program is Python-based and brought great results when they first used it on college students at the university. The failure rates of the algebra and pre-calculus and the subsequent calculus classes were cut in half and the enrollment in the intro CS class doubled. In high schools, people who failed Algebra 1 multiple times were able to pass and 25% more students enrolled in AP Computer Science [12]. The research team at iMPaCT-Math is looking to take the feedback from teachers, building a better relationship with teachers, principals, and school districts, and reach a wider audience with their program.

**3. Duke University Alice Math Project**

The Adventures in Alice Programming Project is working on using the Alice programming tool to incorporate computer science into different subjects in K-12 education. Since 2008 the program has been implemented in various places around the US, including Durham, NC; Virginia Beach, VA; San Jose, CA; Charleston, SC; and Oxford, MI. It will allow the students to get a taste of programming and hopefully encourage them to want to learn more about it. The sites run teacher workshops to encourage K-12 teachers to use Alice in their classrooms with their students.

For our project at Duke, we are currently working on integrating Alice into middle school math education. We are doing this in various ways using tutorials, Alice worlds, activities

for students, a teacher survey, and reaching out to middle school students specifically. In this section, we will describe the teacher survey, where we asked teachers about what classes and grade level they taught, their opinions on the Common Core and CSTA standards, implementing Alice in their classes, and we asked for suggestions of any new Alice tools or resources that they would find useful. Chapter 4 will describe the Alice curriculum materials and resources that we have created for this project. In Chapter 5, we will discuss outreach events that we have hosted or presented our Alice project materials this year.

### 3.1 Teacher Survey

To collect information on how teachers felt about Alice, we sent out a survey to twenty-two math and science related teachers as well as media teachers who have attended a teacher workshop hosted at Duke University. The survey asked the teachers:

1. Name:
2. What school do you teach at?
3. What grade do you teach?
4. What is the name of the class that you teach?
5. How do you feel about the new core standards?
6. Are you familiar with the CSTA computer science standards?
7. Have you tried to implement Alice in your curriculum with your students?
8. If Alice could be mapped to requirements in the Common Core standards, would you be more likely to use it?
9. List all of the subjects that you will be teaching from November to February.
10. Are there any Alice worlds or Alice tools that you feel would be really useful for your class?

We received nine responses to this survey, and the teachers who responded taught grades 6 – 12 and taught various subjects including Geometry, Algebra 1, Algebra 2, Finance, Biology, Chemistry, Physical Science, AP Environment, Computer Programming, AP Statistics, and Pre-Calculus. The teachers provided us with feedback and ideas for the Alice project at Duke and different Alice worlds that we could create. None of the teachers surveyed had ever heard of the CSTA K-12 standards for computer science. They also had very mixed opinions about the new Common Core standards, the majority of them were negative or skeptical that they would work. All of the teachers except one replied that they would be more likely to use Alice if it was mapped to the Common Core standards, since that is what they are required to teach in their classes. The teacher who was not more likely to use Alice if it was mapped to the Common Core standards said that he would most likely not be able to use it in his classes because he taught calculus and advanced high school math that Alice does not support very well. We also had several ideas for Alice worlds to create that the teachers provided including an Alice world to help students structure and visualize word problems, more short challenges that help students focus on one particular topic, more games to help students learn the more difficult math topics, 3D and 2D shapes that can be manipulated and modified in Alice, and an Alice world to help students learn the importance of parameters and recursion for programming.

## 4. Curriculum Materials

### 4.1 Mapping

We have mapped Alice concepts and Alice worlds developed at Duke since 2008 to the math Common Core standards and the CSTA standards. The Common Core State Standards Initiative was designed to provide a clear understanding of topics that students should learn [6]. North Carolina and all but five of the US states have implemented the Common Core standards.

Eventually they will be used throughout the U.S. as a whole. Three American territories have also adopted the Common Core Standards into their curricula [5]. The standards will provide a benchmark for all students to graduate from high school and be able to succeed in college-level classes or the work force. We have created a spreadsheet that maps the Common Core standards for Mathematics from 5$^{th}$ grade through the high school requirements to different Alice worlds and materials that have been created by the Duke Alice team as well as teacher lesson plans from the teacher workshops. These standards have also given us ideas on worlds to create, such as the Boat Averages, and Fraction World. For example, standard 7.SP.5 says that students should understand the definition of probability as the chance that an event occurs between 0 and 1, and the Probability World mentioned earlier deals with this specific idea in statistics and probability. There are over 100 Alice worlds on the Duke Alice repository site that satisfy at least one of the math Common Core standards. We do not have a lot of resources available for the higher levels of math because most of the focus is around middle school students and Alice does not have the functionality to deal with Calculus, advanced trigonometry, etc. Appendix 1 Exhibit A contains the mappings of our Alice materials to the Common Core math standards from 5$^{th}$ grade through 12$^{th}$ grade.

We also mapped all of our Alice materials to the CSTA Computer Science standards that describe what students should know about computing and technology at a certain age [9]. These have been mapped for Level 2 (grades 6-9). The CSTA standards are a little different, because not only are our Alice materials mapped to certain requirements, but also to overarching Alice concepts. For example, CSTA standard 2.CPP.5 states that students should know how to implement programming solutions to problems including techniques such as loops, conditional statements, variables, logic expressions, and functions. It is possible for students to learn and

practice these programming concepts in Alice. Most of our tutorials and educational materials provide various ways to use all of these in different worlds and projects. Some CSTA standards, however, were very vague or just couldn't be completed in Alice because they deal with all of computer science and not just programming. An example of this is standard 2.CI.3 that requires students to analyze the positive and negative impacts of computing on human culture, which is a topic that is way too broad for Alice. The documents of these mappings can be seen in Appendix 1 Exhibit B.

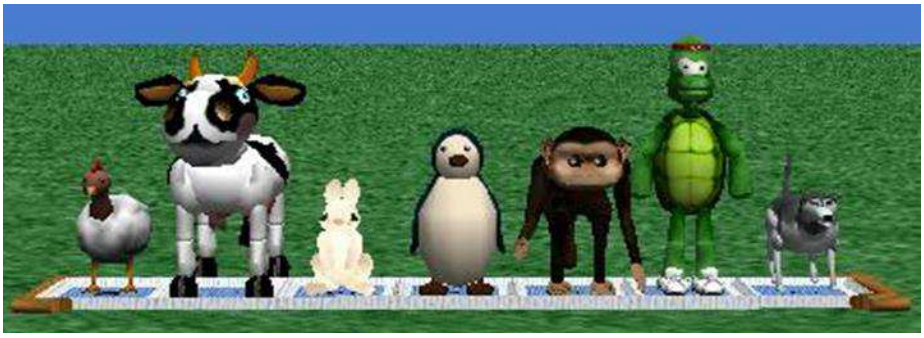
### 4.2 General Tutorials

Since starting with the Alice project in the summer of 2012, we have created several Alice worlds and tutorials that deal with teaching students how to program in Alice and using Alice to practice and program for mathematics. The programming concept tutorials that we created are Array Tutorial, Visual Lists, and we modified the Scene Change Tutorial and one of the Getting Started tutorials. The Array Tutorial teaches the user how to create arrays in Alice and has them create methods to demonstrate what that can be done with arrays (Appendix 2, Exhibit A). This tutorial uses a gym setting with various Alice characters in an array and has a coach who narrates the world.

#### 4.2.1 Array Tutorial

The Array Tutorial shows users how to iterate through each element of the array and make them execute a method in order and in reverse, several ways how to go through the elements of an array at a specified interval and have the characters perform different actions, how to randomly select elements from the objects in the array, and how to choose specific elements and make them swap places within the array by having Alice prompt the user to enter the two indices of the characters that he wants to switch, then animates the objects changing places.

### 4.2.2 Visual Lists



The Visual List Tutorial is very similar to the Array Tutorial, in that it shows the functionality of visual lists in Alice to compare and contrast the uses of arrays and lists to collect objects (Appendix 2, Exhibit B). This world has a visual list with various animal objects from the Local Gallery. In this tutorial, students will learn how to iterate through each of the objects in the list and give them an action, how to make all of the objects in the list complete an action at the same time, and how to cycle through each of the objects in the list so that each animal shifts through each position of the list until they arrive at the original position. When this tutorial is finished, the user is asked to try out the methods and make the animals do different methods in order, all together at the same time, and while cycling through the list.

### 4.2.3 Scene Change 2.0

Scene Change 2.0 is a modified and updated version of a tutorial called Scene Change that shows students how to change scenes in Alice. The newer version of Alice made scene changes and changing the ground texture for each scene easier because the textures were saved as objects. We changed this in the tutorial, as well as added how to move a character from scene to scene in the world and added the use of the "orient to" method (Appendix 2, Exhibit C). In this world there is a rabbit that starts off in the desert, then moves to an island in the ocean, and finally ends up on the moon. In each scene the rabbit does a different action, and each time the camera fades out to black and then back in so it appears that the scenes change seamlessly. At the end, the user is challenged to add more methods to each of the scenes and create a fourth scene with a different ground cover to conclude the world.

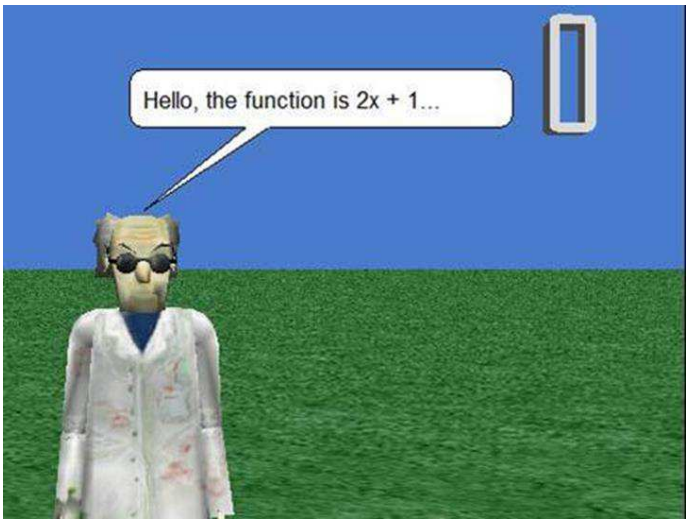
### 4.2.4 Shortened Astronaut-Humvee Introduction Tutorial

Finally, we created a shorter introductory tutorial modified from the Getting Started Space Tutorial that tells a story about an astronaut and a humvee. We shortened this tutorial to make it easier to do in one class period and took out some of the more complicated features such as moving the camera around. This tutorial shows users how to add objects, move objects around in the world, use methods, create an object method, create an event, and set an object's vehicle. (Appendix 2, Exhibit D). The setting is the Space environment with an astronaut and a humvee. The tutorial goes over how to create a new method to make the astronaut wave his arm, since there isn't one built in. It also goes over how to create a new event, where the user can drive the humvee around the world using the arrow keys on the keyboard after the astronaut has moved to the humvee. It also changes the astronaut's vehicle property from the world to the humvee, essentially gluing the astronaut to the humvee so that when the humvee is moved, the astronaut will move with it.

### 4.3 Tutorials Involving Math

We have also created several tutorials that deal with Alice programming concepts as well as topics in math. These tutorials are Nonvisual Arrays, Nonvisual Arrays and Recursion, and Probability World.

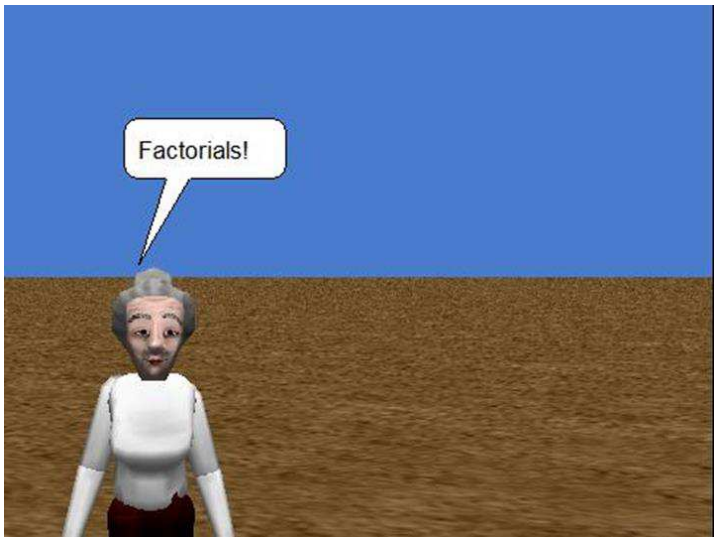
#### 4.3.1 Nonvisual Arrays




The Nonvisual Array tutorial shows users how to create nonvisual arrays in Alice and use them for their specific world. In this context, it is used to create a quiz for a given function that helps the user practice plugging numbers into algebraic formulas (Appendix 2, Exhibit E). In the teacher survey, teachers suggested that we create more quizzes and games for students to use and practice math skills. This tutorial not only provides students with a quiz to practice algebra, but they will learn how to create their own using Alice. In this example, the equation is 2x + 1, and the tutorial shows the user how to fill in an array with the solutions to the equation when x = 0 to when x = 50. To create the equation and produce the numbers, the student enters the equation into an Alice function, which automatically calculates the answer to their equation, which is 2x + 1 in this case. This function is called in a loop within the main method that iterates 51 times, starting

at x = 0 and incrementing the value of x. Every time a value is calculated, they add it into the appropriate index of the array. When the loop completes, the array will be filled with the solutions of the equation that the user chose up to 50. At the end of this tutorial, the student is challenged to complete the world by creating the quiz that asks them to calculate the solutions by hand, and then to create their own version that uses the equation of their choice.

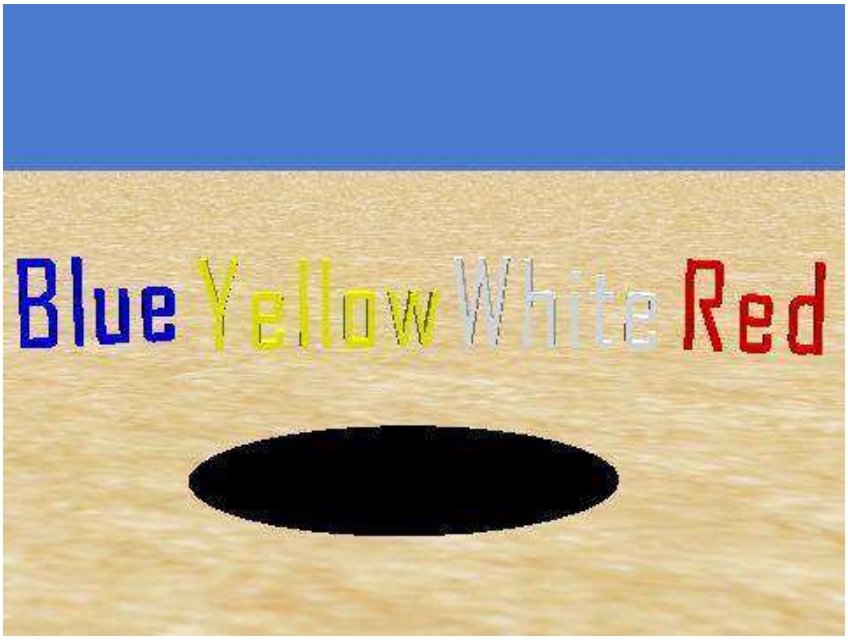
### 4.3.2 Nonvisual Arrays and Recursion




Nonvisual Arrays and Recursion is very similar to the Nonvisual Array Tutorial, but it also shows the users how to use the advanced computer programming technique of recursion in Alice and uses recursive mathematical formulas to practice using it (Appendix 2, Exhibit F). This tutorial goes over nonvisual lists in Alice and how to create them, and then shows the user how to create a recursive function. The user will also learn how to create a quiz in this tutorial. Here, the recursive function is Fibonacci's Sequence. The user will create an Alice function that makes a call to itself within the function and automatically computes Fibonacci's Sequence. The student will also need to figure out the base case, so that there program will not run forever. Once the function is completed, a loop in the main method is called to fill in the array with the first 10

Fibonacci numbers. They also learn how to create the quiz method in this world. The quiz starts when the world is run, and the user will be asked to calculate the Fibonacci number at a certain index and provide the correct answer. At the end of the tutorial, the student is asked to create another Alice world with factorials as the recursive function.

### 4.3.3 Probability World




Probability World, which also has a tutorial to go with it (Appendix 2, Exhibit G), is a game where certain colored balls are put into a hole and the player must correctly guess the probability of choosing a random colored ball. After each try, the number of balls is updated and the user must recalculate a new probability. My version has 4 blue balls, 3 yellow balls, 2 white balls, and 3 red balls, and the program asks the probability of choosing one of these colors each time. After the user gets the correct answer, the ball is taken out of the hole and the number of balls for that color and the total is decremented. There is also a challenge at the end of the tutorial, for the user to create their own version of the world using different colored balls, a different number of each colored ball, etc. to allow them to practice programming in Alice. This world

maps to the Common Core Math standards 7.SP.5, 7.SP.6, and 7.SP.7A that deal with understanding probability and chance in the 7th grade.

### 4.4 Alice Math Worlds

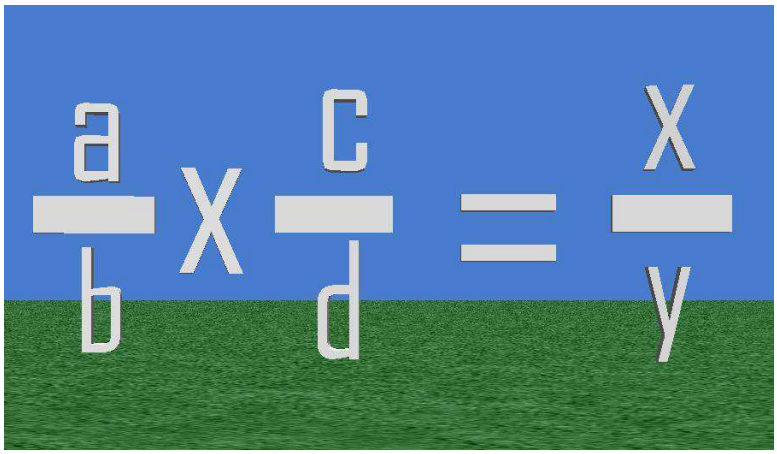
The Alice worlds that I have created that focus solely on math are Basketball Math, Fraction World, and Order of Operations. The CSTA Standard 2.CT.14 says that students should be able to examine the connections between mathematics and computer science. All of these worlds deal with helping students practice different math concepts using Alice.

#### 4.4.1 Basketball Math


Basketball Math is a world that helps students practice multiplication in a fun game. If they get the answer to the problem correct, then they make the basket and the score is incremented. Otherwise, they will miss the shot and have to try again until they provide the correct answer. There are two versions of this game, one that allows students to practice multiplying positive and negative integers up to 12 and the other one allows students to practice

mixed multiplication by multiplying integers and decimals together. The first version of this world fulfills standards 5.NBT.5, 6.NS.3, and 7.NS.2A. The version that practices mixed multiplication satisfies Common Core standard 5.NF.4A.

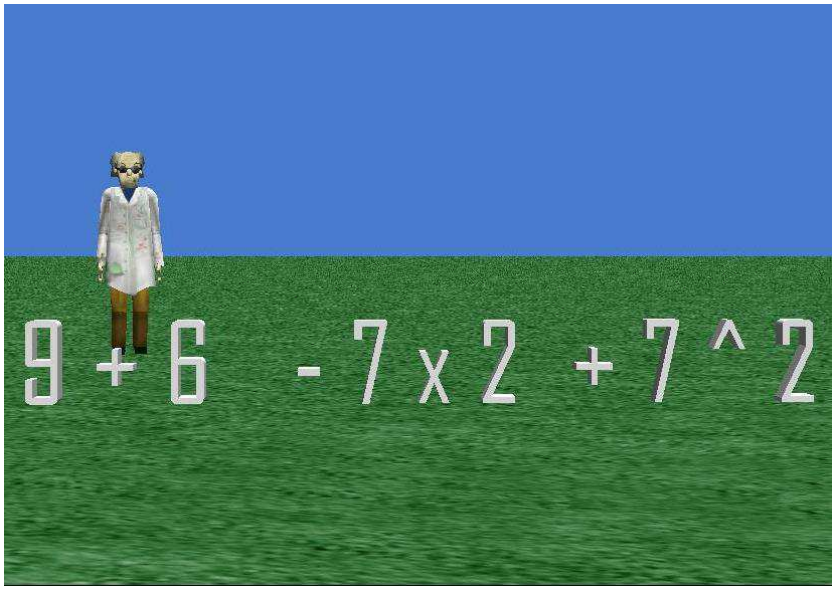
### 4.4.2 Fractions World



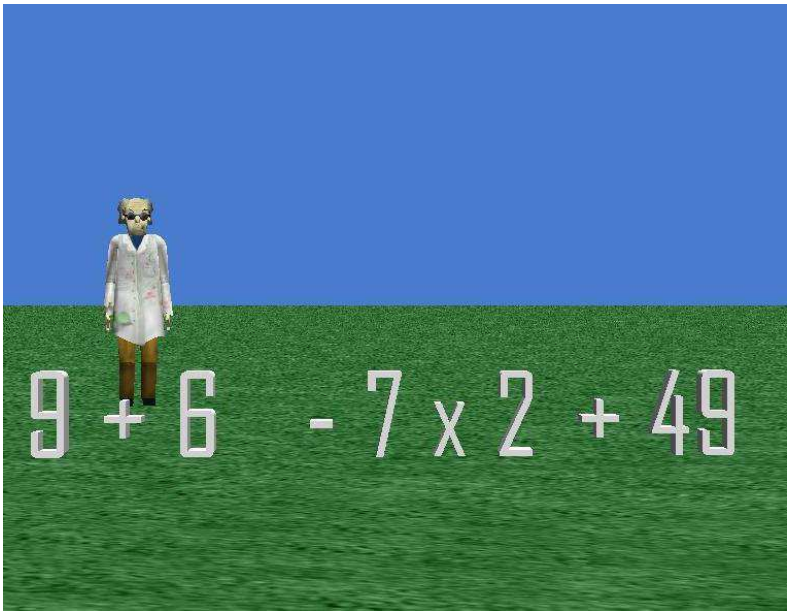

Fraction World is an Alice world where users can practice adding, subtracting, multiplying, and dividing fractions. The Common Core standards for middle school math have many different requirements for fractions, so this world will help students practice working with them in various ways. In this world, the students can choose which arithmetic expressions that they want to practice by pressing ‘a’ for addition, ‘s’ for subtraction, ‘d’ for division, and ‘m’ for multiplication. It randomly provides problems for them to figure out after the user decides which arithmetic operation they want to use, and then asks them, if necessary, to reduce the fraction for multiplication and division and to find the least common denominator of the fractions for addition and subtraction. For each type of arithmetic expression, the numerators are a random number chosen between 1 and 9 and the denominators are random numbers ranging from 2 to 12 using Alice’s random number generator. These values can be changed within the individual methods to make problems easier or more difficult. Then, the user must calculate the numerator and

denominator of the resulting fractions based on which math operation that they chose to practice. This world satisfies standards 5.NF.1, 5.NF.4A, 5.NF.5B, and 6.NS.1.

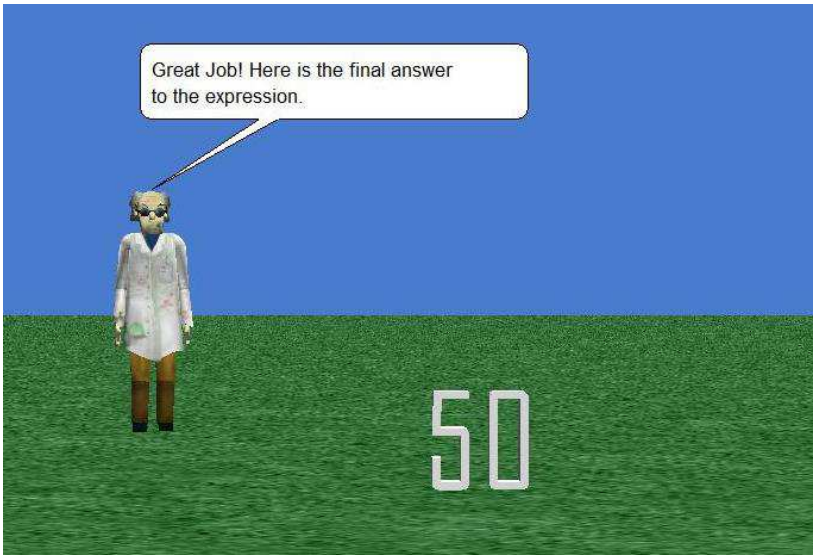
### 4.4.3 Order of Operations World




The Order of Operations World is an interactive program that helps students practice the order of operations by clicking on the operators that need to be calculated in the correct order, and then calculating each part to find the final answer. A game to help students with the Order of Operations was suggested by a specific teacher in our teacher survey, and this world was built for that. We already had an Alice world that dealt with the order of operations, but this world is a better, newer, and more interactive version to help students learn the concepts. In this world, the user should click on the exponent (^) in the expression "7 ^ 2" ($7^2$). Alice will then ask the user for the answer to this part of the problem and when they input the correct answer, the problem will update the expression with the new value.




The user continues to click on the appropriate symbol of each expression and enter the answers until they reach the last expression, and finally calculate the final answer to the numerical expression.



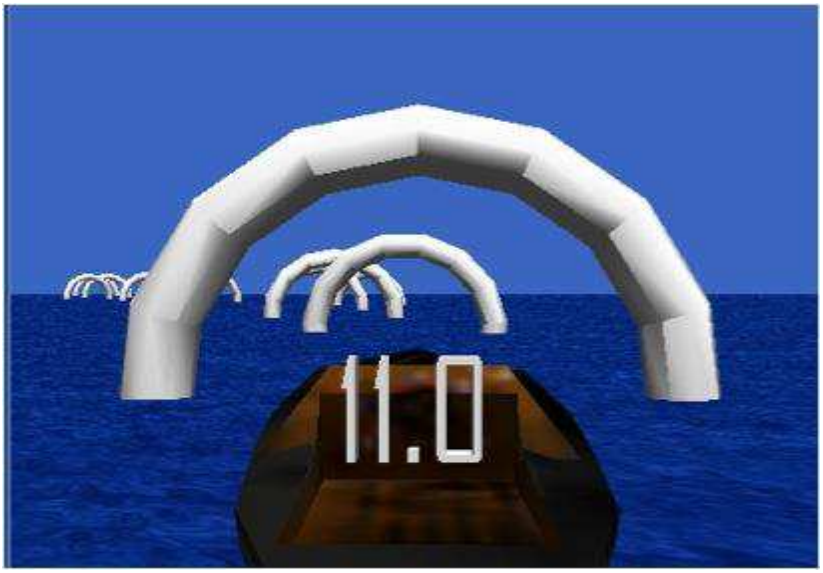

This world will help students learn the Order of Operations (parenthesis, exponents, multiplication/division, and addition/subtraction from left to right) and practice solving functions with them. There are nine examples to try with three different templates, and the world is

designed to get harder as the student goes on. All of the parts of the Order of Operations are practiced in this world, but addition/subtraction and multiplication/division are the most prevalent. We wanted to create a good mixture of equations for the students to try to make sure they understood the concept. It fulfills Common Core math standard 5.OA.1on the order of operations as well as any other standards dealing with simple addition, subtraction, multiplication, division, and exponents.

### 4.5 Math Challenges

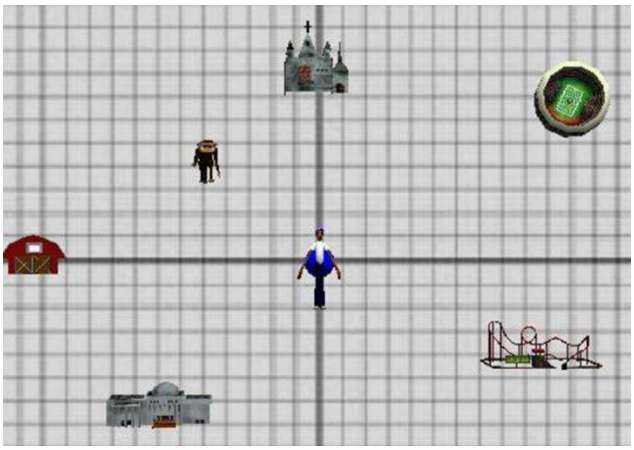
Challenges are Alice worlds that have been started, but it is the students' job to fill in a function or a method to finish it. These were created so that students could use Alice in their classes to learn about different math skills without teachers having to take the time to teach the students all about Alice. We have created several different math-related Alice challenges for students to complete, including the Boat Race Challenges, Calculator Challenge, and Distance Challenge.

#### 4.5.1 Boat Racing Challenges



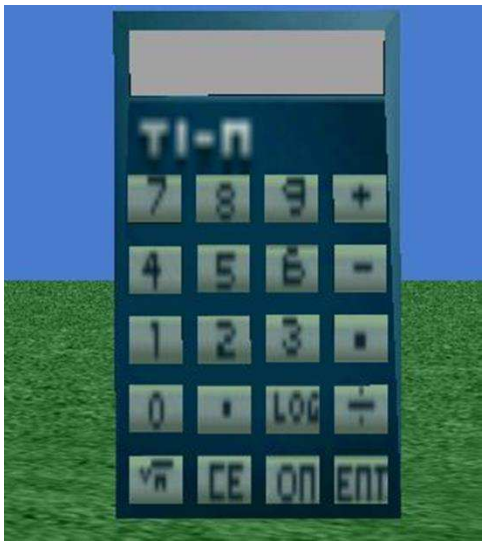
The Boat Racing Game is an Alice world where the user must play a game where they drive a boat through 10 arches. In the challenges, we modified this game so that the student creates their own data from the total time that it takes them to complete the game, and then calculates the average speed of the boat to finish the game based on that info. This world is an adjusted version of a boat game tutorial that shows users how to build a boat racing game, and it has now been applied to mathematics. This game allows the students to create their own data from playing the game, and then use it in a math context. There are four new forms of this challenge, each with a separate goal for the user. We have created a version that calculates the average time it takes the user to go between each arch, one that calculates the average distance between the arches, a version to calculate the speed over the entire game (meters/sec), and a version that calculates the average time it takes the user to win over multiple games. In the first two examples, the user must fill in a function called average, so that the correct average time or distance is returned in the problem. In the next example, the student must fill in the function speed to return the speed of the boat. Finally in the last example, the student must fill in the average function to calculate the average time for the games as well as the win method so that when one game finishes, the user has the option to play again or finish the world. In all of these challenges, the student is also encouraged to change the win method and add any animations they would like to the world. These challenges can be seen in Appendix 4 Exhibits A-D.

### 4.5.2 Distance Challenge

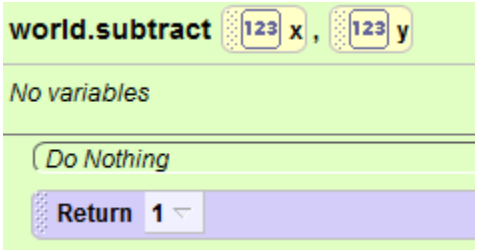


Another challenge that we have created is the Distance Challenge. In this world, we have a character named Jimmy who is visiting a new city for the first time. It is the student's job to fill in the distance function in this challenge (Appendix 3, Exhibit E), so that Jimmy will know how far he has to go to reach the different sites. This world was created simply to test students' knowledge of the distance formula and finding the distance between two points. A coordinate plane is dropped into the world as a billboard and the objects are placed in front of it to simulate points on the plane. From this, the distance formula can be derived from Pythagorean's Theorem based on the x-y position of the destination. Each object has a saved position in a variable x and y value, which is passed into the function when the user clicks on a specific place. In this world, when the user clicks on a location, if the distance to that point is correct, then Jimmy will move to visit that place and an animation will play. When he makes it to one place, the user then clicks on another location and Jimmy will continue to move from there if the distances are correct. If it is wrong, then Jimmy will notify the user by telling them that the answer is incorrect. The correct answer is provided by a function called solution, which provides the right answer to the problem.

### 4.5.3 Calculator Challenge




The last challenge we created is the Calculator Challenge, where students are given a calculator object in Alice and they must fill in all of the functions to make the calculator work. The starting world for this challenge already has the buttons working to enter numbers, so the students don't have to write that part of the code. All they need to do is fill in different Alice functions so that the calculator operates properly. Before the student begins, the functions pass in the appropriate number of parameters for the calculation, but they all simply return 1.




In the most basic challenge for this world, the students have to fill in all of the functions for buttons that are represented on the calculator (addition, subtraction, multiplication, division, and square root). There are also additional challenges for this world. One example is the Logarithm challenge, because there is a log button on the calculator, and Alice does have log base

10 as one of its built-in math functions, although it does have log base e (ln). The goal of this challenge is to input a formula that changes bases of the logarithm function. There were also more advanced challenges, such as creating a new button and function to calculate exponents, adding a button to simulate multiplication just using adding (loops), dropping in images as billboards and creating buttons for all of the advanced math functions that are Alice already has (cosine, sine, tangent, etc.), adding a button to calculate the factorial of a number, inserting a special function button where the user can choose any function to put in and the calculator will calculate the answer for any value of x that they put in, and more. The handout for this challenge is in Appendix 3 Exhibit F.

## 5. Outreach

### 5.1 School Visit

On November 30, 2012, we were able to visit Oak Grove Middle School in Winston-Salem, NC. We were invited by Mr. Mendenhall, a teacher who attended an Alice teacher workshop over the summer and expressed interest in having us visiting his school. While we were there, we taught three $6^{th}$ grade math classes. We showed Alice demonstrations and taught the students how to program in Alice by creating an Alice world with them. The world that they created was the shortened version of the Astronaut-Humvee introductory tutorial. Each class had roughly 25-30 students and we presented in three different classrooms where each student was given a laptop from a mobile lab. The media teacher of Oak Grove Middle School and two other information technology teachers who rotate around different schools in the district also joined us and observed our presentations to the classes. There were some problems that we ran into when showing the students various math demos, but this gave us more feedback and helped to debug

these worlds and make them better. Each class went increasingly better as we got used to how the students would learn the information.

In the first class, we started off by going over the Order of Operations world. After we spoke about it, we let them try it out for themselves on their individual laptops to help them practice learning the order of operations. This is where many kids became frustrated because of the way the text messed up in the world and it was taking up a lot of time, so we decided to move on to the tutorial. The students were very excited once they began to create their own world, but eventually we ran out of time and were not able to finish making the world. While they were packing their things up, we demoed another Alice world called the Princess and the Dragon world, which the students really enjoyed watching and seeing what could be done with Alice.

In the second class, we started by demoing the Princess-Dragon tutorial and the Fraction World. Once again, the students enjoyed the animation about the princess and the dragon and were involved in various problems with the Fraction World. We decided to go through it together as a class rather than let each child try it because that would be quicker and more efficient. In between class, the students broke for lunch. After lunch, we began working on the tutorial and we were able to finish it. The students were really excited about creating their own Alice world, and they were especially thrilled when they learned how to create a new event and drive the humvee around the Alice world using the arrow keys, which we didn't make it to with the first class. The students definitely became more animated and the volume of the class rose when this happened. At the end of the period, we showed some more demos of a Halloween Greeting Card world and a Spin the Bottle game in Alice, which the kids liked a lot.

For the last class, we started by demoing one of the Boat Averages games and the Fraction World. After this, we went through the Astronaut-Humvee tutorial with the students. Once again,

they were excited to drive the humvee. This class also finished the tutorial a lot earlier, so we gave them the opportunity to play around with the world and add anything they wanted to their story. We noticed that right away the students wanted to add more characters, objects, and sounds to their worlds. At the end, we were able to demo the Halloween world, an Alice world that simulated an Octopus rollercoaster ride, and a simulation of the Frogger game in Alice.

The students were overall very enthusiastic about Alice, with several asking questions such as "Can I do this at home?", "Is it free?", etc., and Mr. Mendenhall emailed us 4 days after our visit to thank us for coming and let us know that the students were still talking about their experiences with Alice and thinking of neat ideas. One student already made plans to create a football world before we left the school on our visit and told his teacher about it. We did not even show the students how to get any of the other objects of Alice other than what was needed for the tutorial, which showed that they ventured into the Local Gallery of objects on their own. The students really enjoyed the demos of different games and animations that we showed them. Some of them had trouble with the worlds they created because they were playing around with other things that we did not present such as moving the camera, tumbling objects, and more. The students were also able to see the errors in the world easily, because they could see if the object did something incorrectly or differently from how our world worked.

### 5.2 SIGCSE

We presented a poster and ran a workshop at the Association for Computing Machinery Special Interest Group on Computer Science Education (SIGCSE) conference this year in Denver, CO. This was a national conference where a large number of computer science professors, teachers, and students all come together to focus on the state of computer science education and to learn about different research projects, tools, programs, etc. on the topic. Our

poster with Susan Rodger was entitled “Integrating Computer Science into Middle School Mathematics”. It consisted of all of our work from the summer and the Alice project at Duke along with our more recent work on using Alice in middle school math. The poster that we presented can be seen in Appendix 4. The workshop was entitled “Experimenting With and Integrating Alice 2.3 Into Many Disciplines”. It was taught with Steve Cooper of Stanford University, Wanda Dann and Jacobo Carrasquel of Carnegie Mellon, and Susan Rodger, who have all done a lot of work with Alice. Dann presented on some of the changes in the newest edition of Alice 2.3, Carrasquel presented on the Spanish translation version of Alice 2.3, Cooper showed a new tutorial format for learning Alice, Rodger gave an overview of the work done at Duke, and we were able to present our materials that Duke has come up with for integrating Alice into math. Overall, the conference was a great experience and we were able to exhibit the research that we have been working on throughout this year.

### 5.3 Alice Activity Day

On March 23, 2013, we hosted an Alice Activity Day for local 6$^{th}$ grade students to come to Duke and learn about Alice. There were two sessions, one in the morning (9:00-12:00) and one in the afternoon (1:30-12:30). At each of the sessions, we started out by giving the students a pre-survey, to get their attitudes and views on computer science, test their Alice knowledge, and to obtain their demographic information (gender, age, race, and ethnicity) and career goals. After all of the students finished the survey, we showed some general demonstrations of Alice worlds. This was to introduce the students to Alice, since many of them had never used it before, and to get them excited about all of the different possible things that can be done in Alice. After showing demos for about 15-30 minutes, the students were able to try creating their own world. We

created another introduction to Alice world very similar to the shortened Astronaut-Humvee world, except this one featured a person on an island. For this world, the students learned how to:

- add and position 3D objects in an Alice world,
- make a character using the He/She-Builder
- use built-in Alice methods and create new methods in order to teach our characters a new action, in this case a backflip,
- create events so that they could interact with the world by pressing 'B' to make the character do a backflip and using the arrow keys to drive around a rowboat
- and utilize the "vehicle" property so that the character and the rowboat would move together.

After we completed this world, we gave the students some free time and a break before moving on to math Alice worlds. After about 30 minutes of free time, we showed a few more demos of Alice worlds related to school projects, and then had them run the Fraction world on their own individual computers to try a few problems. Then, they played the Order of Operations world and practiced several of those problems. Finally, we had them complete the basic part of the Calculator Challenge, after showing them one example. They had to fill in the add, subtract, multiply, divide, and square root functions in order to build a working calculator. After this, we gave the students some more free time before giving them the post-survey to see if using Alice changed their opinions at all on computer science, how they think Alice could be useful to them in their classes, and see if they learned about Alice coding since the pretest we gave them in the beginning. We noticed that the afternoon session went by much faster than the morning one, so the kids in the second activity were able to have more free time and play around with the demo worlds and games that we placed on each of the laptops.

At our activity day sessions we had 26 total students, 14 in the morning and 12 in the afternoon. Of these students, 25 students chose to fill out our survey. We had 13 girls and 12 boys show up to our activity day. The story-telling aspect and creativity aspect of Alice may have attracted more females and encouraged them to sign up for the session. The student population was also diverse racially with 9 Caucasian students, 6 Asian students, 3 African American students, 1 American Indian student, 1 White/American Indian student, 1 Turkish student, 2 students who selected Multi-Racial, and 2 students who chose not to respond. Also, all of the students were either 11 or 12 years old. Additionally, in the pre-survey, we asked the students what their career goals were in the future. 5 students replied that they wanted to be a doctor, 3 responded with veterinarians, 2 lawyers, 1 law enforcement agent, 6 scientists/engineers, 5 responded "Other" with an equestrian, a video game designer, a robotics [engineer], a graphic designer, and a crazy cat lady, and 6 students were undecided on their career goals. For the workshop, we found out that most of the students really enjoyed Alice and it had an overall positive affect on their views about computer science.

The surveys really showed us that Alice would be a successful tool to use in the middle school curriculum. Most of the students said that Alice did not really help them learn about math, because they already knew the math skills that we went over and we didn't teach them anything new about math. Many students, however, noted that Alice did help them learn about computer programming and coding. More than half of the students suggested that Alice would be fun and valuable to use in their classes for projects, book reports, games, presentations, etc. in math as well as in other subjects. One student commented about creating a Civil War or World War Alice world and several of the students mentioned using Alice to replace PowerPoint presentations. Most of the students also found Alice easy to use and did not find any part of it confusing other

than the Calculator Challenge that we gave them. While completing this challenge, many of the students had trouble clicking on the buttons and getting the expected result when they ran their world. This could have been due to the students clicking too many times, moving ahead and not paying attention when we showed them how to complete the first example, messing with other parts of the code, not wanting to try the challenge, etc. The version that we built worked fine along with several other students in the sessions, and everyone had the same calculatorStart.a2w file to start with. There were also 11 students who just left their Calculator Challenge world blank. In the section about the attitudes of computer science, we saw that there was a general increase in interest in computer science between the pre-test and the post-test. The two statements that had the most significant increase were "I hope that my future career will require the use of computer science concepts." and "I like to use computer science to solve problems.", that both went from an average answer of "Disagree" to "Agree" by the students. Alice is a good tool to increase students' interest in and exposure to computer science and would be easy to integrate into K-12 curricula to help solve the "crisis" in computer science education.

## 6. Future Plans

To continue this project in the future, the Duke Adventures in Alice Programming project should continue building new Alice worlds and materials that would be helpful for teachers to implement in their classrooms and for students to use. The worlds they should create will help to fill in the holes that we found between our Alice products and the CSTA and Common Core math standards. The goal is to eventually have an Alice world for each of the standards in both curricula. In our interactions with students, we found that the children really liked Alice and enjoyed it because it allowed them to use their creativity to create animations and games. Alice helps students learn computer programming skills, while having fun at the same time. From our teacher surveys, it was suggested that it would be good to create more Alice worlds that deal with

word problems and giving students practice and knowledge on how to solve word problems. Alice would be a good tool to use to help students visualize problems that they need to solve. The Alice team should also start switching their tutorial format to the new interactive tutorial system that Steve Cooper is implementing for learning Alice worlds.

Another way to improve on this project in the future is to create more Math Challenges for students to use that deal with more subjects. As of right now, the challenges that we have deal with averages, arithmetic, the distance formula, and Pythagorean's Theorem. These would also help to complete the map of the Duke Alice materials to the Common Core math standards, as well as give teachers more opportunities to use these challenges in their classes and get the students to complete them. The Alice Team should also create more Alice Math worlds for students to be able to play to practice developing certain math skills. The students have the ability to modify the code and change these worlds if they would like to as well. We would like to have an Alice world or challenge for each standard in the Common Core Math curriculum available for teachers and students to use on our Duke Alice Repository website.

## 7. Conclusion

We have concluded that Alice can increase the interest in computer science among middle school students. From the post-survey of our Alice Activity Day, over half of the students suggested that Alice should be used in their schools for projects, presentations, and games. This shows that Alice could be easily integrated into K-12 education, and that students would enjoy using it their classes. The students at Oak Grove Middle School were also very excited about using Alice in their classes and learning how to create animations. 7 students in the Alice Activity Day also noted that Alice helped them understand computers and programming better. In addition, the number of students who "Strongly Agreed" with liking to use computer science to

solve problems increased from 3 to 8 and nobody answered negatively after using Alice compared to 5 who "Disagreed" before. There was also a significant increase in the interest for using computer science in a future career. The average response grew from "Disagree" (2.88) to "Agree" (3.217) after the students spent the session learning about Alice and creating their own worlds. We also found that 14 students "Strongly Agreed" with voluntarily taking additional computer science classes after using Alice and only 8 did in the pre-survey. Overall, Alice increased the students' interest in computer science and their desire to learn more about it.

Computer science must be introduced to students at a younger age in order to help increase interest and exposure to the topic and help avoid the crisis in computing education. Since it is very difficult to add new classes for students to take because of standardized testing, curriculum, and not having enough room, we believe that it is best to integrate programming skills into the present curriculum that students must learn in order to help them gain interest and see the applications of computer science. We want to work with teachers as well as media/technology specialists in schools to encourage them to integrate Alice into their curriculum to introduce the students to programming at an earlier age. Many other programs and groups are also working on integrating computing into K-12 education, so that students can learn about computing and be more likely become interested in computer science when they are older.

## 6. References


[1] 2009 Alice Symposium. website, 2009. Retrieved April 22, 2013 from http://www.cs.duke.edu/csed/aliceSymposium2009/

[2] 2013 SIGSCE conference. website, 2013. Retrieved April 22, 2013 from http://www.sigcse.org/sigcse2013/

[3] Alice. website, 1999. Retrieved April 22, 2013 from http://www.alice.org/

[4] Bootstrap. website, 2013. Retrieved April 22, 2013 from http://www.bootstrapworld.org/

[5] Common Core In the States. website, 2012. Retrieved April 22, 2013 from http://www.corestandards.org/in-the-states

[6] Common Core State Standard Initiatives. website, 2012. Retrieved April 22, 2013 from http://www.corestandards.org/

[7] S. Cooper, W. Dann, D. Lewis, P. Lawhead, S. Rodger, M. Schep, and R. Stalvey. A pre-college professional development program. In The 16th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE 2011), pages 188-192, 2011.

[8] CS Unplugged. website, 2008. Retrieved April 22, 2013 from http://www.csunplugged.org/

[9] CSTA Computer Science Standards. website, 2011. Retrieved April 22, 2013 from http://csta.acm.org/Curriculum/sub/K12Standards.html

[10] Cuny, Jan. "Finding 10,000 Teachers: Transforming High School Computer Science". CSTA Voice: The Voice of K-12 Computer Science Education and its Educators. Vol. 6, Issue 5, pages 1-2, January 2010

[11] Duke University Adventures in Alice Programming. website, 2008. Retrieved April 22, 2013 from http://www.cs.duke.edu/csed/alice/aliceInSchools/

[12] Freudenthal, Eric et al. MPCT - Media Propelled Computational Thinking, In Forty-first SIGCSE Technical Symposium on Computer Science Education, pages 37-41, 2010.

[13] Greenfoot. website, 2013. Retrieved April 22, 2013 from http://greenfoot.org/overview

[14] iMPaCT-Math. website, 2010. Retrieved April 22, 2013 from http://www.impactstem.org/

[15] LEGO MINDSTORM NXT Education. website, 2013. Retrieved April 22, 2013 from http://www.legoeducation.us/eng/categories/products/middle-school/lego-mindstorms-education

[16] Maloney, John et al. “The Scratch Programming Language and Environment”. In ACM Transactions on Computing Education, Vol. 10, No. 4, Article 16, pages 1-15, November 2010.

[17] Pausch, Randy et al. “Alice: A Rapid Prototyping System for Building Virtual Environments”. University of Virginia: IEEE Computer Graphics and Applications, Vol. 15, Issue 3, pages 8-11, May 1995.

[18] Scratch. website, 2009. Retrieved April 22, 2013 from http://scratch.mit.edu

[19] Sontag, Marie. “Critical Thinking with Alice: A Curriculum Design Model for Middle School Teachers”. In Alice '09 Proceedings of the 2009 Alice Symposium, Article No. 2, 2009.

[20] Utting, Ian et al. “Alice, Greenfoot, and Scratch - A Discussion”. ACM Transactions on Computing Education, Vol. 10, No. 4, Article 17, pages 1-11, November 2010.

[21] Wilson, Cameron; Sudol, Leigh Ann; Stephensen, Chris; and Stehlik, Mark. “Running on Empty: The Failure to Teach K-12 Computer Science in the Digital Age”. Association for Computing Machinery and Computer Science Teachers Association, 2010. http://www.acm.org/runningonempty/

[22] Wing, Jeannette. “Computational Thinking”. Communications of the ACM, Vol. 49, Issue 3, pages 33-35, March 2006.

## 7. Appendix

**All Appendix items are on http://www.cs.duke.edu/csed/alice/aliceInSchools and http://www.cs.duke.edu/csed/alice12/brown/thesis/appendix.php**

## Appendix 1: Standard Mappings

Exhibit A - Common Core Mathematics
Exhibit B - Computer Science Teachers Association

## Appendix 2: Tutorials

Exhibit A - Array Tutorial
Exhibit B - Visual List
Exhibit C - Scene Change 2.0
Exhibit D - Astronaut-Humvee Short
Exhibit E - Nonvisual Arrays
Exhibit F - Nonvisual Arrays With Recursion
Exhibit G - Probability World

## Appendix 3: Challenges

Exhibit A - Boat Race Challenge 1
Exhibit B - Boat Race Challenge 2
Exhibit C - Boat Race Challenge 3
Exhibit D - Boat Race Challenge 4
Exhibit E - Calculator Challenge
Exhibit F - Distance Challenge

## Appendix 4: SIGCSE poster

SIGCSE poster

## Appendix 1: Standard Mappings

We have mapped our Alice materials and resources to the mathematics Common Core State Mathematics Standards (grades 5-12) and the Computer Science Teachers Association Computer Science standards for Level 2 (grades 6-9). The Common Core standards are in Exhibit A, while the CSTA standards are in Exhibit B. A majority of the teachers we surveyed acknowledged that they would be more likely to use Alice in their classes if it was compatible with the Common Core curriculum that they are required to teach. Alice will also help apply the CSTA computer science standards, to expose students to certain amounts of computer science and technology throughout their K-12 education.

| Grade | Standard | Alice World | Description |
|---|---|---|---|
| 5th Grade | | | |
| | 5.OA.1 | Order of Operations World | This world tests students knowledge of the order of operations (PEMDAS) and this standard requires that students be able to use parentheses, brackets, or braces in numerical expressions and evaluate them. |
| | | Order of Operations Rap | This world is an animation and song to help students learn and memorize the order of operations in math. |
| | | Distributive Property Tutorial | The Distributive Property world shows how to deal with parentheses in an equation and checks to see if the equations are expanded correctly with an application to the Distributive Property. |
| | 5.OA.2 | Using Pearls to Understand Variables | This standard deals with simple algebraic expressions and interpreting numerical expressions. Even though they do not need to evaluate them this early in the standards, this Alice world shows how to set up and solve algebraic equations using bags of pearls as variables. |
| | 5.OA.3 | Nonvisual arrays | A simpler version of this game that allows students to practice calculating mathematical and algebraic patterns rather than making the list to hold them. |
| | 5.NBT.1 | Rounding Game | This standard requires that students recognize the different places of a multi-digit number (ones, tens, hundreds,...) and they know the corresponding place to the right(/ 10) and left (* 10). The first part of the questions in this game deals with identifying the given place by clicking on the number. |
| | 5.NBT.2 | Scientific Notation | In this standard, students must understand patterns in multiplying numbers by 10 and use exponents to denote powers of 10. This Alice world goes over how to translate numbers into scientific notation form which uses exponents to denote powers of 10 and trailing zeros in a number. |
| | 5.NBT.3A | | Expanded form of numbers- (EX: 347.392 = 3*100 + 4*10 + 7*1 + 3*(1/10) + 9*(1/100) + 2*(1/1000) |
| | 5.NBT.3B | Inequalities | Can extend the inequalities world to include more examples with decimals and fractions in the game. |
| | 5.NBT.4 | Rounding Game | The rest of the rounding game world deals with rounding numbers which is what this standard is, except the world needs to add decimals. |
| | 5.NBT.5 | Basketball Math | In this Alice world, students practice finding the products of numbers in a basketball game. This standard requires students to be able to multiply multi-digit whole numbers, so the maximum values in the game can be increased to practice multiplying larger numbers. |
| | | Multiplication Table | This game allows kids to practice their multiplication skills up to 10 x 10. |
| | 5.NBT.6 | Sign Me Up | This world deals with the division of whole numbers (easier examples) with positive and negative integers. This standard deals with division as well, but goes up to 4 digit dividends and 2 digit divisors. |
| | 5.NBT.7 | Nemo Learns Math | A more advanced version of this game that includes decimals would help students practice this standard of adding, subtracting, multiplying, and dividing decimals to the hundredths place. |
| | 5.NF.1 | Fraction World | This Alice world allows students to add and subtract fractions and go through the method of finding the common denominator, then calculating the numerator and denominator. |
| | 5.NF.2 | | - Word problem to add and subtract fractions. |
| | 5.NF.3 | | - Recognize 3/4 = 3 divided by 4 |
| | 5.NF.4A | Fraction World | This world allows students to practice multiplying and dividing fractions. |
| | 5.NF.4B | | - Area of a rectangle with fractional sides |
| | 5.NF.5A | Reducing Simple Fractions, Fraction World | This Alice world delves into the greatest common factor of numbers with applications in reducing fractions. Fraction World does this with an application to fraction arithmetic. |
| | 5.NF.5B | Simplifying Fractions | This Alice world allows students to practice simplifying fractions and help them learn fraction equivalence: a/b = (na)/(nb) |
| | 5.NF.6 | | - Real world problems and applications of multiplying fractions and mixed numbers. |
| | 5.NF.7A | | - Dividing fractions and whole numbers |
| | 5.NF.7B | | - Dividing whole numbers by fractions |
| | 5.NF.7C | | - Convert different measurement units in a given measurement system (5 cm = .05 m) |
| | 5.MD.1 | | |
| | 5.MD.2 | | - Make a line plot of fractional data. |
| | 5.MD.3A | | - Unit cube |
| | 5.MD.3B | | - A solid figure that can be packed with n unit cubes has a volume of n cubic units. |
| | 5.MD.4 | | - Measure volumes with unit cubes of cubic cm., cubic in., etc. |
| | 5.MD.5A | | - Find volume of rectangular prism using unit cubes. |
| | 5.MD.5B | Volume Formulas | This Alice world deals with learning the formulas for the volumes of different shapes, but this standard only requires students to find the volume of rectangular prisms using V = b*h = l*w*h. It won't help them practice this standard, just memorize formulas. |
| | 5.MD.5C | | - Volume is additive. |
| | 5.G.1 | Lesson on the Coordinate Plane | An introduction to coordinate planes (Axes, coordinates, lines, ordered pairs, etc.) |

| | | | |
|---|---|---|---|
| | 5.G.2 | Plotting Points, Lines, and Scatter Plots | This standard requires students to be able to represent real world data and mathematical problems by graphing points in the first quadrant and interpreting those values. The Plotting Points Alice world takes data created by the student about how far a bicyclist travels and asks them to plot the points and them interpret the data that they came up with. |
| | 5.G.3 | | - Categories of 2D shapes and their properties. |
| | 5.G.4 | | - Be able to classify 2D objects in a hierarchy based on properties. |
| 6th Grade | | | |
| | 6.RP.1 | | - Ratios (2:1) |
| | 6.RP.2 | | - Relationship of ratios to fractions. |
| | 6.RP.3A | | - Tables of equivalent ratios |
| | 6.RP.3B | | - Unit rate problems |
| | 6.RP.3C | | - percentages |
| | 6.RP.3D | | - ratios to convert measurements |
| | 6.NS.1 | Fraction World | This world deals with arithmetic expressions of fractions. |
| | 6.NS.2 | Sign Me Up | Extend this world to include the division of multi-digit numbers. |
| | 6.NS.3 | Basketball Math, Nemo Math, etc. | To accomplish this standard, all we need to do is extend the previous mentioned math Alice worlds to make them harder by adding multi-digit addition, subtraction, multiplication, and division. |
| | 6.NS.4 | Simplifying Fractions | This world allows students to practice finding the greatest common factor between 2 numbers with applications in simplifying fractions. |
| | 6.NS.5 | Walk the Number Line | This world will help students understand the difference between positive and negative numbers. Does not go into real-world applications though. |
| | 6.NS.6A | Walk the Number Line apps | Negatives and positives as opposites, symmetry. (-(-3) = 3) |
| | 6.NS.6B | Kick the Coordinate Plane, Lesson on the Coordinate Plane | In this world (Kick the Coordinate Plane), students click a character to kick a soccer ball to a random position on a graph and must give the coordinates of the point. This goes over points in all 4 quadrants and positive/negative numbers. |
| | 6.NS.6C | Walk the Number Line, Integer Football | Walk the Number Line allows students to move a character around to the correct place on a number line by adding/subtracting positive and negative integers. Integer Football does the same thing, with an application to sports and moving down a football field on given plays. |
| | 6.NS.7A | Inequalities | Students should be able to interpret inequalities with negative numbers. Use this world with more examples with negative numbers. |
| | 6.NS.7B | | - Real world applications for the above standard. |
| | 6.NS.7C | | - Absolute Value |
| | 6.NS.7D | | - Statements of absolute value |
| | 6.NS.8 | Bike Plot | This standard that requires that students be able to solve real-world problems by graphing points, and this world applies that skill to tracking the speed of a bicycle. |
| | 6.EE.1 | Scientific Notation* | The Scientific Notation world uses exponents, but we'll need an Alice world that deals with exponents exclusively. |
| | 6.EE.2A | Using Pearls to Understand Variables | Standard 2a deals with students being able to understand and write expressions using variables and letters to represent numbers. |
| | 6.EE.2B | | - Understand and identify the parts of a mathematical function. (sum, term, product, difference, quotient, factor, coefficient,...) |
| | 6.EE.2C | | - Solving algebraic functions |
| | 6.EE.3 | Distributive Property Tutorial | The distributive property. |
| | 6.EE.4 | | - Identify when two equations are equivalent. [Inequalities] |
| | 6.EE.5 | | - Finding values that make and equation or inequality true. |
| | 6.EE.6 | Using Pearls to Understand Variables | Using variables to represent numbers and write expressions from real life problems. This world is an example but won't help them practice this skill. |
| | 6.EE.7 | | - Writing and solving equations of the form x + p = q and px = q |
| | 6.EE.8 | | - Inequalities with variables and applications. |
| | 6.EE.9 | ModelinXYZ(Kelly) and Mike's world | These worlds allow students to use graphs to represent equations and also go into more advanced functions. Also, these worlds to not deal with tables which are also mentioned in this standard. |
| | 6.G.1 | | - Areas of triangles and special quadrilaterals. |
| | 6.G.2 | | - Find the volume of a rectangular prism |
| | 6.G.3 | | - Draw polygons in a coordinate plane |
| | 6.G.4 | | - Represent 3D figures with rectangles and triangles to find the surface area. |
| | 6.SP.1 | | - Recognize statistical questions. |
| | 6.SP.2 | | - Statistical distributions |
| | 6.SP.3 | Boat Averages | Measures of center (average/median) summarize a group of data with just one value. |
| | 6.SP.4 | Bike Plot | - Display numerical data using dot plots, histograms, and box plots. |
| | 6.SP.5A | | - Reporting the number of observations |
| | 6.SP.5B | | - Describing the nature of observation |
| | 6.SP.5C | Boat Averages | This standard deals with calculating the measures of center (median and mean) of data and the boat averages worlds allow users to practice finding the average speed, distance, and time a boat travels. |
| | 6.SP.5D | | - Relating measures of center to variability |

| | | | |
|---|---|---|---|
| 7th Grade | | | |
| | 7.RP.1 | | - Ratios and averages of measurements |
| | 7.RP.2A | | - Decide whether two quantities are proportional by table or graphing |
| | 7.RP.2B | | - Constant of proportionality |
| | 7.RP.2C | | - Represent proportional relationships with equations |
| | 7.RP.2D | | - Proportional relationship between points on a graph |
| | 7.RP.3 | | - Multistep ratio and percent problems |
| | 7.NS.1A | Walk the Number Line | This standard deals with describing situations where opposite quantities combine to make 0 such as -4 + 4, but this standard gives the example of hydrogen atoms. |
| | 7.NS.1B | Walk the Number Line | This standard wants students to understand that p + q is a distance of the abs(q) from p in either direction. |
| | 7.NS.1C | Walk the Number Line | In this standard, students should understand that subtraction is just adding the inverse: p - q = p + (-q) |
| | 7.NS.1D | | - Properties of operations to add and subtract rational numbers |
| | 7.NS.2A | Basketball Math, etc | Understanding multiplication and distributive property with positive and negative integers (-1)(-1)=1 |
| | 7.NS.2B | Sign Me Up, etc. | Understand that integers can be divided if the divisor is non-zero. With negative values, know that -(p/q) = (-p)/q = (p)/(-q) |
| | 7.NS.2C | | - Use properties of operations as strategies to multiply and divide rational numbers |
| | 7.NS.2D | | - Convert a rational number to a decimal using long division |
| | 7.EE.1 | | - Apply properties of operations as strategies to add, subtract, factor, and expand linear expressions. |
| | 7.EE.2 | | - Rewriting expressions in different forms: a + .05 = 1.05(a) |
| | 7.EE.3 | | - Solve multistep real life problems with positive and negative rational numbers in any form and apply the properties of operations to them... |
| | 7.EE.4A | | - Word problems of the form px + q = r or p(x + q) = r |
| | 7.EE.4B | | - Word problems with inequalities of the form px + q > r or px + q < r |
| | 7.G.1 | | - Solve problems using scale drawings of geometric figures |
| | 7.G.2 | | - Draw geometric shapes with given conditions using rulers, protractors, etc. |
| | 7.G.3 | | - Describe two-dimensional figures by slicing 3D figures. |
| | 7.G.4 | Geometry Game | In this standard, students should know the formulas for the area and circumference of a circle which is practiced in this world along with squares and rectangles. |
| | 7.G.5 | | - Supplementary, complementary, vertical, and adjacent angles |
| | 7.G.6 | | - Solve real world and math problems involving area, volume, and surface area. |
| | 7.SP.1 | 1 Ball, 2 Ball, Red Ball, Blue Ball | This Alice world deals with random sampling from a group of red and blue balls, and in this standard students must learn about gaining information about populations by examining a sample of the population and understand random sampling. |
| | 7.SP.2 | 1 Ball, 2 Ball, Red Ball, Blue Ball | This standard has students use the random sample to draw inferences about the population from the data, and in this world students will predict the number of red and blue balls and see how the samples are simulated. |
| | 7.SP.3 | | - Comparing two different numerical distributions |
| | 7.SP.4 | | - Use measures of center and measures of variability from numerical data from random samples |
| | 7.SP.5 | Probability World | Understanding the definition of probability (the chance an event occurs is between 0 and 1, the likelihood that an event occurs...) |
| | 7.SP.6 | Probability World | Approximating the probability of a chance event by collecting data. |
| | 7.SP.7A | Probability World | Students should develop a uniform probability model and use it to determine the probability of different events. In the game, the user must enter the probability of choosing a random colored ball from a hole. |
| | 7.SP.7B | | - Develop a probability model that may not be uniform. |
| | 7.SP.8A | | - Probability of compound events |
| | 7.SP.8B | | - Represent sample spaces for compound events. |
| | 7.SP.8C | | - Design and use a simulation to generate frequencies of compound events. (simulate Alice?) |
| 8th Grade | | | |
| | 8.NS.1 | | - Irrational Numbers |
| | 8.NS.2 | | - Rational Approximations of irrational numbers |
| | 8.EE.1 | Exponent Laws | This world explains the laws and properties of exponents which students are required to know based on this standard. |
| | 8.EE.2 | | - Square root and cube root |
| | 8.EE.3 | Scientific Notation | Students should be able to know how to use and understand scientific notation. |
| | 8.EE.4 | | - Perform operations with numbers in scientific notation |
| | 8.EE.5 | | - Graph proportional relationships |
| | 8.EE.6 | | - Use similar triangles to calculate why the slope is the same between two points. |
| | 8.EE.7A | | - Linear equations with one variable and one solution |
| | 8.EE.7B | | - Solve linear equations |

| | | | |
|---|---|---|---|
| | 8.EE.8A | | - Students should be able to understand a system of equations and the corresponding point is their intersection. (Graphically) |
| | 8.EE.8B | Systems of Equations | Students should be able to solve systems of 2 linear equations which is what this world helps them practice. |
| | 8.EE.8C | | - Same as the above with real world applications. |
| | 8.F.1 | | - Definition of a function |
| | 8.F.2 | Move in XYZ and Mike's world | Students should be able to compare different functions GRAPHICALLY, also algebraically, numerically in tables, description, etc. |
| | 8.F.3 | | - Linear functions |
| | 8.F.4 | (Slope Quiz) | - Construct a function to create a linear relationship between two points |
| | 8.F.5 | | - Sketch graphs and describe relationship between two functions |
| | 8.G.1A | | - Lines and line segments |
| | 8.G.1B | | - Angles |
| | 8.G.1C | | - Parallel Lines |
| | 8.G.2 | | - Congruency between 2D figures with reflections, translations, and rotations |
| | 8.G.3 | | - Dilations, translations, rotations, and reflections on coordinates |
| | 8.G.4 | | - Similar 2D figures |
| | 8.G.5 | | - Angle sum of triangles |
| | 8.G.6 | | - Prove and explain the Pythagorean Thereom |
| | 8.G.7 | Pythagorean Prom (2D), Pythagorean Theorem in a 3D Problem | "Apply the Pythagorean Theorem to determine the unknown side lengths in right triangles in real-world and mathematical problems in two and three dimensions." |
| | 8.G.8 | Pythagorean Prom | This standard requires students to be able to use Pythagorean's Theorem to calculate the distance between 2 points. |
| | 8.G.9 | Volume Quiz | This world quizzes students on the volume formulas of different shapes including cones, cylinders, and spheres which are specified in this standard. |
| | 8.SP.1 | Bike Plot | Construct and interpret scatter plots. |
| | 8.SP.2 | Bike Plot | Students should know about the line of best fit for a scatter plot data and the end of this Alice world gives an example of finding the line of best fit for the data created by the user. |
| | 8.SP.3 | (Using Pearls to Understand Variables) | - Use linear equations to solve problems |
| | 8.SP.4 | | - Bivariate categorical data |
| High School | | | |
| | N-RN.1 | | - Rational exponents and their properties. |
| | N-RN.2 | | - Rewrite expressions involving radicals and rational exponents. |
| | N-RN.3 | | - Explain why the sum or product of two rational numbers is rational, the sum of a rational number and irrational number is irrational, and the product of a nonzero rational number and an irrational number is irrational. |
| | N-Q.1 | | - Use units as a way to understand problems and to guide the solution for multi-step problems. |
| | N-Q.2 | | - Define appropriate quantities for the purpose of descriptive modeling. |
| | N-Q.3 | | - Choose a level of accuracy appropriate to limitations on measurement when reporting quantities. |
| | N-CN.1 | | - Complex number i such that i^2 = -1. |
| | N-CN.2 | | - Use i^2 and the commutative, associative, and distributive properties to add, subtract, and multiply complex numbers. |
| | N-CN.3 | | - Find the conjugate of a complex number |
| | N-CN.4 | | - Represent complex numbers on the complex plane in rectangular and polar form. |
| | N-CN.5 | | - Represent addition, subtraction, multiplication, and conjugation of complex numbers geometrically. |
| | N-CN.6 | | - Calculate the distance between numbers in the complex plane. |
| | N-CN.7 | | - Solve quadratic equations with real coefficients that have complex solutions. |
| | N-CN.8 | | - Extend polynomial identities to complex numbers. |
| | N-CN.9 | | - The Fundamental Theorem of Algebra is true for quadratic polynomials. |
| | N-VM.1 | | - Recognize vector quantities as having both magnitude and direction. |
| | N-VM.2 | | - Find the components of a vector by subtracting the coordinates of an initial point from a terminal point. |
| | N-VM.3 | | - Solve problems involving velocity and other quantities represented by vectors. |
| | N-VM.4A | | - Add vectors end-to-end, component-wise, and by the parallelogram rule. |
| | N-VM.4B | | - Given 2 vectors in magnitude and direction form, determine the magnitude and direction of their sum. |
| | N-VM.4C | | - Understand vector subtraction. |
| | N-VM.5A | | - Represent scalar multiplication graphically |
| | N-VM.5B | | - Compute the magnitude of a scalar multiple |
| | N-VM.6 | | - Use matrices to represent and manipulate data. |
| | N-VM.7 | | - Multiply matrices by a scalar. |

| | | | |
|---|---|---|---|
| | N-VM.8 | The Matrix | Add, subtract, and multiply* matrices. This standard requires that students be able to multiply matrices of appropriate dimensions. In this Alice world, users are able to practice multiplying 2x2 matrices and learn the method for multiplying matrices. |
| | N-VM.9 | The Matrix | Students should know that matrix multiplication for square matrices is not commutative. In this world, they are able to input the numbers they want into the matrices that will be multiplied and can switch the values to see that they aren't commutative. |
| | N-VM.10 | | - Understand that the zero and identity matrix play a role in matrix addition and multiplication. |
| | N-VM.11 | | - Multiply a vector by a matrix of suitable dimensions. |
| | N-VM.12 | | - Work with 2x2 matrices as transformations in a plane. |
| | A-SSE.1A | | - Interpret parts of an expression. (terms, factors, and coefficients) |
| | A-SSE.1B | | - Interpret complicated expressions by viewing one or more of their parts as a single entity. |
| | A-SSE.2 | | - Use the structure of an expression and identify ways to rewrite it. |
| | A-SSE.3A | | - Factor a quadratic expression to reveal zeros of the function it defines. |
| | A-SSE.3B | | - Complete the square in a quadratic expression. |
| | A-SSE.3C | Exponent Laws | In this standard, students should be able to use the properties of exponents to transform expressions for exponential functions. This Alice world goes over all of the exponent laws with variables, which can be translated into functions and hold the same properties. |
| | A-SSE.4 | | - Derive the formula for the sum of a finite geometric series. |
| | A-APR.1 | System of Equations (2008), System of Equations (2011) | Understand that polynomials form a system analogous to the integers. Polynomials can be added, subtracted, and multiplied, and this Alice world quizzes students on how to add and subtract polynomials using a system of equations. |
| | A-APR.2 | | - Know and apply the Remainder Theorem. |
| | A-APR.3 | | - Identify zeros in polynomials. |
| | A-APR.4 | | - Prove polynomial identities and describe numerical relationships. |
| | A-APR.5 | | - The Binomial Theorem |
| | A-APR.6 | | - Rewrite simple rational expressions |
| | A-APR.7 | | - Understand that rational expressions form a system analogous to the rational numbers |
| | A-CED.1 | *Word problem challenges | Create equations and inequalities in one variable and use them to solve problems. |
| | A-CED.2 | | - Create equations in two or more variables. |
| | A-CED.3 | | - Represent constraints by equations or inequalities. |
| | A-CED.4 | | - Rearrange formulas to highlight a quantity of interest. |
| | A-REI.1 | Using Pearls To Understand Variables | This standard requires students to explain each step in solving a simple equation. The "Using Pearls to Understand Variables" Alice world explains variables using pearls and at the end it provides an example and shows how to solve an equation. |
| | A-REI.2 | | - Solve simple rational and radical equations in one variable. |
| | A-REI.3 | Using Pearls To Understand Variables | Students should be able to solve linear equations and inequalities and this Alice world deals with solving linear equations. |
| | A-REI.4A | | - Use the method of completing the square to transform any quadratic equation. |
| | A-REI.4B | | - Solve quadratic equations by inspection. |
| | A-REI.5 | | - Prove that, given a system of two equations in two variables, replacing one equation by the sum of that equation and a multiple of the other produces a system with the same solutions. |
| | A-REI.6 | System of Equations | This standard deals with solving systems of equations exactly and approximately, and the exact method is practiced in this Alice world. |
| | A-REI.7 | | - Solve a system of linear equations consisting of a linear equation and a quadratic equation. |
| | A-REI.8 | | - Represent a system of linear equations as a single matrix equation. |
| | A-REI.9 | | - Find the inverse of a matrix if it exists and use it to solve systems of equations. |
| | A-REI.10 | | - Understand that the graph of an equation with two variables is the set of all its solutions plotted in the coordinate plane. |
| | A-REI.11 | | - Explain why the x-coordinates of the points where 2 graphs intersect are solutions of the equations. |
| | A-REI.12 | | - Graph the solutions to a linear inequality. |
| | F-IF.1 | | - Understand that a function from one set (domain) connects to another set (range). |
| | F-IF.2 | | - Use function notation, evaluate functions for inputs in their domains, and interpret statements that use function notation. |
| | F-IF.3 | Nonvisual Arrays, Nonvisual Arrays and Recursion in Alice | Students should recognize that sequences are functions, and also defined recursive functions. Both of these Alice worlds use arrays to let students build functions and examine the sequences that they produce, and the second one focuses specifically on recursive functions such as Fibonacci's sequence and factorials. |

| | | | |
|---|---|---|---|
| | F-IF.4 | MoveinXYZ, Bird Graphing | This standard says that for a function that models a relationship between two quantities, interpret key features of graphs and tables (intercepts, intervals of increasing/decreasing, max and min, symmetry, etc.) Both of these Alice worlds deal with graphing functions that the user can examine and compare with other functions. MoveInXYZ uses polynomial functions while Bird Graphing can use all of the math functions built into Alice. |
| | F-IF.5 | MoveinXYZ, Bird Graphing | Students need to be able to relate the domain of a function to its graph and the quantitative relationship it describes. In these Alice world, students can view the graphs of a variety of functions and use the graphs to analyze the domains of the functions. |
| | F-IF.6 | | - Calculate and interpret the average rate of change of a function over a specified interval. |
| | F-IF.7A | MoveinXYZ, Bird Graphing | Graph linear and quadratic functions and show intercepts, maxima, and minima. |
| | F-IF.7B | Bird Graphing | Graph square root, cube root, and piecewise-defined functions including step and absolute value functions. The Bird Graphing Alice world is able to graph the square and cube root functions. |
| | F-IF.7C | MoveInXYZ | Graph polynomial functions, identifying zeros and factorizations when available. This Alice world allows users to create the functions that they want to graph up to the x^4 degree. |
| | F-IF.7D | Bird Graphing | Graph rational functions, identifying zeros and asymptotes. This world allows users to create rational functions if they can create them using the built-in Alice functions. |
| | F-IF.7E | Bird Graphing | Graph exponential and logarithmic functions showing intercepts and end behavior and trigonometric functions. Alice world functions contain these mathematical functions in the advanced math section that can be graphed in this world. |
| | F-IF.8A | | - Use the process of factoring and completing the square in a quadratic function to show zeros, extreme values, and symmetry. |
| | F-IF.8B | | - Use the properties of exponents to interpret expressions for exponential functions. |
| | F-IF.9 | | - Compare properties of two functions each represented in a different way. |
| | F-BF.1A | | - Determine an explicit expression, a recursive process, or steps for calculation from a context. |
| | F-BF.1B | | - Combine standard function types using arithmetic operations |
| | F-BF.1C | | - Compose functions [T(h(y))] |
| | F-BF.2 | | - Write arithmetic and geometric sequences both recursively and with an explicit formula. |
| | F-BF.3 | Bird Graphing | Identify the effect on the graph of replacing f(x) by f(x) + k, kf(x), and f (kx) for specific values of k. In this Alice world, the user can choose a function in Alice and then modify it by making the changes above and choosing a value of k to see how the graph changes for each one. |
| | F-BF.4A | | - Solve an equation of the form f(x) = c and write an expression for the inverse. |
| | F-BF.4B | | - Verify by composition that one function is the inverse of another |
| | F-BF.4C | | - Read values of an inverse function from a graph or table |
| | F-BF.4D | | - Produce an invertible function from a non-invertible function by restricting the domain. |
| | F-BF.5 | | - Understand the inverse relationship between exponents and logarithms. |
| | F-LE.1A | Nonvisual Arrays in Alice | In this standard, students should be able to prove that linear functions grow by equal differences over equal differences. This Alice world shows how functions grow at an equal rate and helps them practice with a quiz to calculate these values. |
| | F-LE.1B | | - Recognize situations in which one quantity changes at a constant rate per unit interval |
| | F-LE.1C | | - Recognize situations in which a quantity grows or decays by a constant percent rate |
| | F-LE.2 | | - Construct linear and exponential functions including arithmetic and geometric sequences |
| | F-LE.3 | Bird Graphing, MoveInXYZ | This standard wants students to observe quantities increasing exponentially, linearly, quadratically, polynomially, etc. in graph and table form. These Alice worlds present these values in graphical form. |
| | F-LE.4 | | - For exponential models, express as a logarithm of the solution. |
| | F-LE.5 | | - Interpret the parameters in a linear or exponential function in terms of a context |
| | F-TF.1 | | - Understand radian measure of an angle |
| | F-TF.2 | | - Explain how the unit circle in the coordinate plane enables the extension of trigonometric functions to real numbers |
| | F-TF.3 | | - Use special triangles to determine geometrically the values of sin, cos, and tan for pi/3, pi/4, and pi/6 |
| | F-TF.4 | | - Use the unit circle to explain symmetry and periodicity of trigonometric functions |
| | F-TF.5 | | - Choose trig functions to model periodic phenomena with specified amplitude, frequency, and midline |

| | | | |
|---|---|---|---|
| | F-TF.6 | | - Understand that restricting a trig function to a domain which is always increasing/decreasing allows its inverse to be constructed. |
| | F-TF.7 | | - Use inverse functions to solve trig equations |
| | F-TF.8 | | - Prove the Pythagorean identity sin^2 + cos^2 = 1 |
| | F-TF.9 | | - Prove the addition and subtraction formulas for sin, cos, and tan |
| | G-CO.1 | | - Know the precise definitions of angle, circle, perpendicular and parallel lines, line segments, point, line, distance, arc, etc. |
| | G-CO.2 | | - Represent transformations in the plane using transparencies and geometry software |
| | G-CO.3 | | - Given a rectangle, parallelogram, trapezoid, or regular polygon, describe the rotations and reflections |
| | G-CO.4 | | - Develop definitions of rotations, reflections, and transformations |
| | G-CO.5 | | - Given a geometric figure, draw the transformed figure |
| | G-CO.6 | | - Use geometric descriptions of rigid motions to transform figures and to predict the effect of a given rigid motion on a given figure |
| | G-CO.7 | | - Use the definition of congruence in terms in terms of rigid motions |
| | G-CO.8 | | - Explain how the criteria for triangle congruence follow from the definition of congruence |
| | G-CO.9 | | - Prove theorems about lines and angles |
| | G-CO.10 | | - Prove theorems about triangles |
| | G-CO.11 | | - Prove theorems about parallelograms |
| | G-CO.12 | | - Make formal geometric constructions with a variety of tools |
| | G-CO.13 | | - Construct an equilateral triangle, a square, and a regular hexagon inscribed in a circle |
| | G-SRT.1A | | - A dilation takes a line not passing through the center of the dilation to a parallel line |
| | G-SRT.1B | | - The dilation of a line segment is longer or shorter in the ratio given by the scale factor |
| | G-SRT.2 | | - Given two figures, use the definition of similarity and decide if they are similar |
| | G-SRT.3 | | - Use properties of similarity transformation to establish the AA criterion for 2 triangles to be similar |
| | G-SRT.4 | | - Prove theorems about triangles. |
| | G-SRT.5 | | - Use congruence and similarity for triangles to solve problems |
| | G-SRT.6 | | - Understand that by similarity, side ratios in right triangles are properties of the angles in the triangle |
| | G-SRT.7 | | - Explain and use the relationship between sin and cos of complementary angles |
| | G-SRT.8 | | - Use trigonometric ratios and Pythagorean Theorem to solve right triangle in applied problems |
| | G-SRT.9 | | - Derive the formula A = 1/2ab sin(c) for the area of a triangle |
| | G-SRT.10 | | - Prove the Law of Sines and Cosines |
| | G-SRT.11 | | - Understand and apply the Law of Sines and the Law of Cosines |
| | G-C.1 | | - Prove that all circles are similar |
| | G-C.2 | | - Identify and describe relationships among inscribed angles, radii, and chords |
| | G-C.3 | | - Construct the inscribed and circumscribed circles of a triangle and prove properties of angles and for a quadrilateral inscribed in a circle. |
| | G-C.4 | | - Construct a tangent line from a point outside a given circle to the circle |
| | G-C.5 | | - Derive using similarity the fact that the length of the arc intercepted by an angle is proportional to the radius |
| | G-GPE.1 | | - Derive the equation of a circle of given center and radius using the Pythagorean Theorem |
| | G-GPE.2 | | - Derive the equation of a parabola given a focus and directrix |
| | G-GPE.3 | | - Derive the equations on ellipses and hyperbolas |
| | G-GPE.4 | | - Use coordinates to prove simple geometric theorems algebraically |
| | G-GPE.5 | | - Prove the slope criteria for perpendicular and parallel lines |
| | G-GPE.6 | | - Find the point on a directed line segment between two given points |
| | G-GPE.7 | | - Use coordinates to computer perimeters and areas of polygons |
| | G-GMD.1 | | - Give an informal limit argument for the formulas for the circumference of a circle, area of a circle, volume of a cylinder, pyramid, and cone. |
| | G-GMD.2 | | - Give an informal argument using Cavalieri's principle |
| | G-GMD.3 | Volume Formulas | Use volume formulas for cylinders, pyramids, cones, and spheres to solve problems. This world will help students learn the formulas of the volumes for different 3D shapes. |
| | G-GMD.4 | | - Identify the shapes of 2D cross sections of 3D shapes |
| | G-MG.1 | | - Use geometric shapes, their measures, and their properties to describe objects |
| | G-MG.2 | | - Apply concepts of density based on area and volume in modeling |
| | G-MG.3 | | - Apply geometric methods to solve design problems |
| | S-ID.1 | Bike Plot | This standard wants students to represent data with plots on a real number line, dot plots, histograms, and box plots. This Alice world has the user create data and then plot the points on a graph. |
| | S-ID.2 | | - Use statistics appropriate to the shape of the data distribution to compare center and spread |

| | | | |
|---|---|---|---|
| | S-ID.3 | | - Interpret differences in shape, center, and spread in the context of data sets |
| | S-ID.4 | | - Use the mean and sd of a data set to fit it to a normal distribution and to estimate the population percentages |
| | S-ID.5 | | - Summarize categorical data for two categories in two-way frequency tables |
| | S-ID.6A | Bike Plot | This standard has students find a function to the data and use functions fitted to data to solve a problem. In this Alice world, after the user plots the points from the data that they create, the best-fit line is drawn and predicts a future value. |
| | S-ID.6B | | - Informally asses the fir of a function by plotting and analyzing residuals |
| | S-ID.6C | | - Fit a linear function for a scatter plot that suggests linear association |
| | S-ID.7 | | - Interpret the slope and the intercept of a linear model. |
| | S-ID.8 | | - Compute and interpret the correlation coefficient of a linear fit. |
| | S-ID.9 | | - Distinguish between correlation and causation |
| | S-IC.1 | | - Understand statistics as a process for making inferences about population parameters |
| | S-IC.2 | | - Decide if a specified model is consistent with results from a given data-generation process |
| | S-IC.3 | | - Recognize the purposes of and differences among sample surveys, experiments, and observational studies |
| | S-IC.4 | | - Use data from a sample survey to estimate a population mean or proportion |
| | S-IC.5 | | - Use data from a randomized experiment to compare two treatments |
| | S-IC.6 | | - Evaluate reports based on data |
| | S-CP.1 | Can I Get Your Number?, 1 Ball, 2 Ball, Red Ball, Blue Ball | This standard wants the students to describe events of subsets of a sample space. Both of these worlds deal with random sampling and creating subsets. The first creates a random set of numbers to form a phone number and the second is randomly sampling from a group of balls. |
| | S-CP.2 | | - Definition of independent events |
| | S-CP.3 | | - Understand the conditional probability of A given B and interpret their independence |
| | S-CP.4 | | - Construct and interpret two-way frequency tables |
| | S-CP.5 | | - Recognize and explain the concept of conditional probability |
| | S-CP.6 | | - Find the conditional probability of A given B |
| | S-CP.7 | | - Apply the Addition Rule of probabilities |
| | S-CP.8 | | - Apply the Multiplication Rule of uniform probabilities |
| | S-CP.9 | Line Up | This standard requires students to use permutations and combinations to compute probabilities, and this Alice world shows the user how to use permutations to find the number of possible ways to order a group of people in a line. |
| | S-MD.1 | | - Define a random variable for a quantity of interest |
| | S-MD.2 | | - Calculate the expected value of a random variable and interpret it as the mean of the probability distribution |
| | S-MD.3 | | - Develop a probability distribution and find the expected value for a random variable defined for a sample space that can be calculated |
| | S-MD.4 | | - Develop a probability distribution for a random variable defined for a sample space assigned empiracally |
| | S-MD.5A | | - Find the expected payoff for a game of chance |
| | S-MD.5B | | - Evaluate and compare strategies of expected values |
| | S-MD.6 | Ready, SET, Go!, War, Choosing Random People From a Class | This standard wants students to use probabilities to make fair decisions. Both of these worlds use probabilities to make decisions within them. The first two use probability in a card game and the last one selects a random student from a class. |
| | S-MD.7 | | - Analyze decisions and strategies using probability concepts |

| | Standard | Alice World/Concept | Description |
|---|---|---|---|
| Level 1 (K-6) | | | |
| Level 2 (6-9) | | | |
| Level 3A (9-10) | | | |
| Level 3B (10-11) | | | |
| Level 3C (11-12/AP) | | | |
| | | | |
| **Level 2** | | | |
| | CT.1 | A lot of our Alice tutorials have problems to solve at the end such as the recursion and nonvisual array tutorial, which shows the user how to build an Alice world that calculates Fibonacci's sequence and asks them to use the same algorithm that creates a world that calculates factorials. We will also have different challenges created for students, where a problem will be given to them and they must come up with the algorithm to solve it in Alice. Trigonometry Prom is an example where the prince needs to find out how far he needs to go to meet the princess under the disco ball. | This standard requires that the students be able to figure out the basic steps in algorithmic problem solving. |
| | CT.2 | In Alice, users are allowed to use the commands "Do Together" and "For all Together" with lists to run multiple instructions at the same time. | Process of parallelization to solve problems |
| | CT.3 | | - Define an algorithm as a sequence of instructions. |
| | CT.4 | Alice allows problems to be solved in different ways. For example, you are able to use lists or arrays to hold a collection of information to use in the program. | In this standard, students should be able to evaluate ways that different algorithms can be used to solve the same problem. |
| | CT.5 | In addition to acting out the searching and sorting algorithms, students could watch Alice animations of different algorithms to sort a group of people by heights or find a specific character in a list, while pausing it and asking questions about what will happen next to help them learn the algorithms. Actually programming these is more for level 3. | Act out searching and sorting algorithms |
| | CT.6 | The tutorials provide detailed instructions on how to complete Alice worlds so students should be able to follow them and create a final project based on the tutorial that they complete. | Describe and analyze a sequence of instructions being followed. |
| | CT.7 | Alice is great for this because data can be represented in graph form (Bike Plot, MoveinXYZ, etc.), as text (Challenges, eventual word problem world), numbers (Fractions, Rounding Game, most math worlds), pictures (billboards), and many other objects (for example, bunnies in Fibonacci sequence and balls in probability world, pearls in Using Pearls to Understand Variables, etc.) Students can generate the data when the world runs, and then store it in lists or arrays to analyze it. Examples of this are the Boat Averages worlds where the world itself collects the times it takes the boat to go through each hoop and the distance or time per hoop, and uses it to calculate the average speed of the boat. | This standard requires that students be able to represent data in various ways (text, sounds, pictures, numbers,...). |
| | CT.8 | These Alice worlds take data or functions input by students and displays them in graphical form- MoveinXYZ, Bike Plot, Mike's graph world, a modified Bar Chart object. Bike plot world physically presents the speed of a bicycle based on when the user clicks and plots the data. | Students must use visual representations to display problem states, structures, and data with this standard. |
| | CT.9 | Most of our Alice project and educational tutorials deal with having students interact with content-specific models. For example, in the Science category, students can interact with a model of the lac operon, a helium molecule, a model of the solar system and planets, and many more. | Students have to interact with content specific models in this standard. |

| | | | |
|---|---|---|---|
| | CT.10 | Alice can be used to simulate problems that need to be modeled or simulated. We will be adding Alice worlds dealing with word problems for students to practice solving and it will help them visualize and model the problem in their mind to help them solve it. | Evaluate what kinds of problems can be solved with modeling and simulation. |
| | CT.11 | | - Analyze the degree to which a computer model represents the real world. |
| | CT.12 | In the Challenges section, there is a problem that the student must solve by filling in smaller functions and methods to achieve the desired results. More advanced challenges will have more sections of code for the student to fill in and find the subproblems to solve. | Decompose a problem into several subproblems |
| | CT.13 | Alice allows for computer science concepts such as hierarchy and abstraction in the use of parameters, local/global variables, inheritance, object methods, etc. | Understand the notion of hierarchy and abstraction. |
| | CT.14 | The Alice materials we have made in Mathematics show connections between math and programming and how they overlap. Alice also has many built-in math based functions such as <, >, =, arithmetic, sin, cos, etc that can be implemented into your programs. Alice can be used to help students practice math concepts such as in Basketball Math, or it can be used to make their own math projects and explore a math subject in Alice such as probability world. | Examine connections between mathematics and computer science |
| | CT.15 | The teacher lesson plans page on the Duke Adventures in Alice site provides many examples of how programming in Alice can relate to other disciplines. Examples of this include using Alice for a book report, a history project, math quizzes, or foreign language quizzes. | Interdisciplinary examples of computational thinking. |
| | CL.1 | Alice itself is a productive multimedia tool that supports learning through a new medium. Students can use Alice for projects, presentations, quizzes, games, etc. | Apply productivity/multimedia tools to support learning through curriculum. |
| | CL.2 | The tutorials on our page have instructions on how to build the worlds that we have. It is possible to have students collaborate on a project to make an Alice world in a group setting by following the instructions given in the tutorials. | Students must collaboratively design, develop, publish, and present products using technology. |
| | CL.3 | | - Collaborate with peers, experts, and others using collaborative practices such as peer programming, team projects, and group active learning. |
| | CL.4 | | - Exhibit dispositions necessary for collaboration. |
| | CPP.1 | | - Select appropriate tools and technology resources to solve problems |
| | CPP.2 | Alice is an example of a multimedia tool that can be used in the classroom to help students engage in their learning. It is also a beginning programming tool that can help students move on to other programming and multimedia tools. | Use a variety of multimedia tools. |
| | CPP.3 | Students can use Alice to design and present products and it is a technology resource. The teacher can have the students be creative and create a story or game using Alice, then present their ideas and final product to the class. | Design, develop, publish, and present products using technology resources. |
| | CPP.4 | | Students will have to demonstrate an understanding of algorithms and their practical application. |
| | CPP.5 | Our Alice tutorials page has many examples of tutorials on how to use these program solutions such as loops, conditional statements, variables, logic, etc. in an Alice world to solve a problem. | Implement problem solutions using a programming language (loops, conditional statements, logic, expressions, variables, and functions) |

| | | | |
|---|---|---|---|
| | CPP.6 | There is an annual competition that students can enter where they must create Alice worlds that teach about computer and internet safety in it's animation. Students can build worlds for that and at the same time learn about good practices in information security. | Demonstrate good practices in personal information security |
| | CPP.7 | Several teachers have come up with Alice worlds to help students learn about different jobs and occupations such as "Career Day", "Business Careers", and "Career Decisions". This type of idea can also be applied to animate how specific jobs use computing and technology. | Identify interdisciplinary careers that are enhanced by computer science |
| | CPP.8 | | - Demonstrate dispositions amenable to open-ended problem solving and programming |
| | CPP.9 | Alice worlds can take data created by the user and implement it into the world for them to analyze. Examples of this are Boat World Averages and Bike Plot, where the user takes data that he creates in the world to calculate the average boat speeds or plot the speed of the bicycle. | In this standard, students should collect and analyze data that is collected from multiple runs of a computer program. |
| | CD.1 | | - Recognize that computers are devices that execute programs |
| | CD.2 | | - Identify electronic devices that contain computational processors |
| | CD.3 | | - Demonstrate an understanding of the relationship between hardware and software |
| | CD.4 | | - Use accurate, appropriate terminology when communicating about technology. |
| | CD.5 | | - Apply strategies for identifying and solving routine hardware problems that occur during everyday computer use. |
| | CD.6 | | - Describe major functions and components of computer systems and networks. |
| | CD.7 | | - Describe what distinguishes humans from machines. |
| | CD.8 | | - Describe ways in which computers use models of intelligent behavior. |
| | CI.1 | | - Exhibit legal and ethical behaviors when using information and technology and discuss consequences of misuse. |
| | CI.2 | | - Demonstrate knowledge of changes in information technologies over time and the effects of those changes |
| | CI.3 | | - Analyze the positive and negative impacts of computing on human culture |
| | CI.4 | | - Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources in real world problems. |
| | CI.5 | | - Describe ethical issues that relate to computers and networks |
| | CI.6 | | - Discuss how the unequal distribution of computing resources in global economy raises issues of equity, access and power. |

| Grade | Standard | Alice World | Description |
|---|---|---|---|
| 5th Grade | | | |
| | 5.OA.1 | Order of Operations World | This world tests students knowledge of the order of operations (PEMDAS) and this standard requires that students be able to use parentheses, brackets, or braces in numerical expressions and evaluate them. |
| | | Order of Operations Rap | This world is an animation and song to help students learn and memorize the order of operations in math. |
| | | Distributive Property Tutorial | The Distributive Property world shows how to deal with parentheses in an equation and checks to see if the equations are expanded correctly with an application to the Distributive Property. |
| | 5.OA.2 | Using Pearls to Understand Variables | This standard deals with simple algebraic expressions and interpreting numerical expressions. Even though they do not need to evaluate them this early in the standards, this Alice world shows how to set up and solve algebraic equations using bags of pearls as variables. |
| | 5.OA.3 | Nonvisual arrays | A simpler version of this game that allows students to practice calculating mathematical and algebraic patterns rather than making the list to hold them. |
| | 5.NBT.1 | Rounding Game | This standard requires that students recognize the different places of a multi-digit number (ones, tens, hundreds,...) and they know the corresponding place to the right(/ 10) and left (* 10). The first part of the questions in this game deals with identifying the given place by clicking on the number. |
| | 5.NBT.2 | Scientific Notation | In this standard, students must understand patterns in multiplying numbers by 10 and use exponents to denote powers of 10. This Alice world goes over how to translate numbers into scientific notation form which uses exponents to denote powers of 10 and trailing zeros in a number. |
| | 5.NBT.3A | | Expanded form of numbers- (EX: 347.392 = 3*100 + 4*10 + 7*1 + 3*(1/10) + 9*(1/100) + 2*(1/1000) |
| | 5.NBT.3B | Inequalities | Can extend the inequalities world to include more examples with decimals and fractions in the game. |
| | 5.NBT.4 | Rounding Game | The rest of the rounding game world deals with rounding numbers which is what this standard is, except the world needs to add decimals. |
| | 5.NBT.5 | Basketball Math | In this Alice world, students practice finding the products of numbers in a basketball game. This standard requires students to be able to multiply multi-digit whole numbers, so the maximum values in the game can be increased to practice multiplying larger numbers. |
| | | Multiplication Table | This game allows kids to practice their multiplication skills up to 10 x 10. |
| | 5.NBT.6 | Sign Me Up | This world deals with the division of whole numbers (easier examples) with positive and negative integers. This standard deals with division as well, but goes up to 4 digit dividends and 2 digit divisors. |
| | 5.NBT.7 | Nemo Learns Math | A more advanced version of this game that includes decimals would help students practice this standard of adding, subtracting, multiplying, and dividing decimals to the hundredths place. |
| | 5.NF.1 | Fraction World | This Alice world allows students to add and subtract fractions and go through the method of finding the common denominator, then calculating the numerator and denominator. |
| | 5.NF.2 | | - Word problem to add and subtract fractions. |
| | 5.NF.3 | | - Recognize 3/4 = 3 divided by 4 |
| | 5.NF.4A | Fraction World | This world allows students to practice multiplying and dividing fractions. |
| | 5.NF.4B | | - Area of a rectangle with fractional sides |
| | 5.NF.5A | Reducing Simple Fractions, Fraction World | This Alice world delves into the greatest common factor of numbers with applications in reducing fractions. Fraction World does this with an application to fraction arithmetic. |
| | 5.NF.5B | Simplifying Fractions | This Alice world allows students to practice simplifying fractions and help them learn fraction equivalence: a/b = (na)/(nb) |
| | 5.NF.6 | | - Real world problems and applications of multiplying fractions and mixed numbers. |
| | 5.NF.7A | | - Dividing fractions and whole numbers |
| | 5.NF.7B | | - Dividing whole numbers by fractions |
| | 5.NF.7C | | - Convert different measurement units in a given measurement system (5 cm = .05 m) |
| | 5.MD.1 | | |
| | 5.MD.2 | | - Make a line plot of fractional data. |
| | 5.MD.3A | | - Unit cube |
| | 5.MD.3B | | - A solid figure that can be packed with n unit cubes has a volume of n cubic units. |
| | 5.MD.4 | | - Measure volumes with unit cubes of cubic cm., cubic in., etc. |
| | 5.MD.5A | | - Find volume of rectangular prism using unit cubes. |
| | 5.MD.5B | Volume Formulas | This Alice world deals with learning the formulas for the volumes of different shapes, but this standard only requires students to find the volume of rectangular prisms using V = b*h = l*w*h. It won't help them practice this standard, just memorize formulas. |
| | 5.MD.5C | | - Volume is additive. |
| | 5.G.1 | Lesson on the Coordinate Plane | An introduction to coordinate planes (Axes, coordinates, lines, ordered pairs, etc.) |

| | | | |
|---|---|---|---|
| | 5.G.2 | Plotting Points, Lines, and Scatter Plots | This standard requires students to be able to represent real world data and mathematical problems by graphing points in the first quadrant and interpreting those values. The Plotting Points Alice world takes data created by the student about how far a bicyclist travels and asks them to plot the points and them interpret the data that they came up with. |
| | 5.G.3 | | - Categories of 2D shapes and their properties. |
| | 5.G.4 | | - Be able to classify 2D objects in a hierarchy based on properties. |
| 6th Grade | | | |
| | 6.RP.1 | | - Ratios (2:1) |
| | 6.RP.2 | | - Relationship of ratios to fractions. |
| | 6.RP.3A | | - Tables of equivalent ratios |
| | 6.RP.3B | | - Unit rate problems |
| | 6.RP.3C | | - percentages |
| | 6.RP.3D | | - ratios to convert measurements |
| | 6.NS.1 | Fraction World | This world deals with arithmetic expressions of fractions. |
| | 6.NS.2 | Sign Me Up | Extend this world to include the division of multi-digit numbers. |
| | 6.NS.3 | Basketball Math, Nemo Math, etc. | To accomplish this standard, all we need to do is extend the previous mentioned math Alice worlds to make them harder by adding multi-digit addition, subtraction, multiplication, and division. |
| | 6.NS.4 | Simplifying Fractions | This world allows students to practice finding the greatest common factor between 2 numbers with applications in simplifying fractions. |
| | 6.NS.5 | Walk the Number Line | This world will help students understand the difference between positive and negative numbers. Does not go into real-world applications though. |
| | 6.NS.6A | Walk the Number Line apps | Negatives and positives as opposites, symmetry. (-(-3) = 3) |
| | 6.NS.6B | Kick the Coordinate Plane, Lesson on the Coordinate Plane | In this world (Kick the Coordinate Plane), students click a character to kick a soccer ball to a random position on a graph and must give the coordinates of the point. This goes over points in all 4 quadrants and positive/negative numbers. |
| | 6.NS.6C | Walk the Number Line, Integer Football | Walk the Number Line allows students to move a character around to the correct place on a number line by adding/subtracting positive and negative integers. Integer Football does the same thing, with an application to sports and moving down a football field on given plays. |
| | 6.NS.7A | Inequalities | Students should be able to interpret inequalities with negative numbers. Use this world with more examples with negative numbers. |
| | 6.NS.7B | | - Real world applications for the above standard. |
| | 6.NS.7C | | - Absolute Value |
| | 6.NS.7D | | - Statements of absolute value |
| | 6.NS.8 | Bike Plot | This standard that requires that students be able to solve real-world problems by graphing points, and this world applies that skill to tracking the speed of a bicycle. |
| | 6.EE.1 | Scientific Notation* | The Scientific Notation world uses exponents, but we'll need an Alice world that deals with exponents exclusively. |
| | 6.EE.2A | Using Pearls to Understand Variables | Standard 2a deals with students being able to understand and write expressions using variables and letters to represent numbers. |
| | 6.EE.2B | | - Understand and identify the parts of a mathematical function. (sum, term, product, difference, quotient, factor, coefficient,...) |
| | 6.EE.2C | | - Solving algebraic functions |
| | 6.EE.3 | Distributive Property Tutorial | The distributive property. |
| | 6.EE.4 | | - Identify when two equations are equivalent. [Inequalities] |
| | 6.EE.5 | | - Finding values that make and equation or inequality true. |
| | 6.EE.6 | Using Pearls to Understand Variables | Using variables to represent numbers and write expressions from real life problems. This world is an example but won't help them practice this skill. |
| | 6.EE.7 | | - Writing and solving equations of the form x + p = q and px = q |
| | 6.EE.8 | | - Inequalities with variables and applications. |
| | 6.EE.9 | ModelinXYZ(Kelly) and Mike's world | These worlds allow students to use graphs to represent equations and also go into more advanced functions. Also, these worlds to not deal with tables which are also mentioned in this standard. |
| | 6.G.1 | | - Areas of triangles and special quadrilaterals. |
| | 6.G.2 | | - Find the volume of a rectangular prism |
| | 6.G.3 | | - Draw polygons in a coordinate plane |
| | 6.G.4 | | - Represent 3D figures with rectangles and triangles to find the surface area. |
| | 6.SP.1 | | - Recognize statistical questions. |
| | 6.SP.2 | | - Statistical distributions |
| | 6.SP.3 | Boat Averages | Measures of center (average/median) summarize a group of data with just one value. |
| | 6.SP.4 | Bike Plot | - Display numerical data using dot plots, histograms, and box plots. |
| | 6.SP.5A | | - Reporting the number of observations |
| | 6.SP.5B | | - Describing the nature of observation |
| | 6.SP.5C | Boat Averages | This standard deals with calculating the measures of center (median and mean) of data and the boat averages worlds allow users to practice finding the average speed, distance, and time a boat travels. |
| | 6.SP.5D | | - Relating measures of center to variability |

| | | | |
|---|---|---|---|
| 7th Grade | | | |
| | 7.RP.1 | | - Ratios and averages of measurements |
| | 7.RP.2A | | - Decide whether two quantities are proportional by table or graphing |
| | 7.RP.2B | | - Constant of proportionality |
| | 7.RP.2C | | - Represent proportional relationships with equations |
| | 7.RP.2D | | - Proportional relationship between points on a graph |
| | 7.RP.3 | | - Multistep ratio and percent problems |
| | 7.NS.1A | Walk the Number Line | This standard deals with describing situations where opposite quantities combine to make 0 such as -4 + 4, but this standard gives the example of hydrogen atoms. |
| | 7.NS.1B | Walk the Number Line | This standard wants students to understand that p + q is a distance of the abs(q) from p in either direction. |
| | 7.NS.1C | Walk the Number Line | In this standard, students should understand that subtraction is just adding the inverse: p - q = p + (-q) |
| | 7.NS.1D | | - Properties of operations to add and subtract rational numbers |
| | 7.NS.2A | Basketball Math, etc | Understanding multiplication and distributive property with positive and negative integers (-1)(-1)=1 |
| | 7.NS.2B | Sign Me Up, etc. | Understand that integers can be divided if the divisor is non-zero. With negative values, know that -(p/q) = (-p)/q = (p)/(-q) |
| | 7.NS.2C | | - Use properties of operations as strategies to multiply and divide rational numbers |
| | 7.NS.2D | | - Convert a rational number to a decimal using long division |
| | 7.EE.1 | | - Apply properties of operations as strategies to add, subtract, factor, and expand linear expressions. |
| | 7.EE.2 | | - Rewriting expressions in different forms: a + .05 = 1.05(a) |
| | 7.EE.3 | | - Solve multistep real life problems with positive and negative rational numbers in any form and apply the properties of operations to them... |
| | 7.EE.4A | | - Word problems of the form px + q = r or p(x + q) = r |
| | 7.EE.4B | | - Word problems with inequalities of the form px + q > r or px + q < r |
| | 7.G.1 | | - Solve problems using scale drawings of geometric figures |
| | 7.G.2 | | - Draw geometric shapes with given conditions using rulers, protractors, etc. |
| | 7.G.3 | | - Describe two-dimensional figures by slicing 3D figures. |
| | 7.G.4 | Geometry Game | In this standard, students should know the formulas for the area and circumference of a circle which is practiced in this world along with squares and rectangles. |
| | 7.G.5 | | - Supplementary, complementary, vertical, and adjacent angles |
| | 7.G.6 | | - Solve real world and math problems involving area, volume, and surface area. |
| | 7.SP.1 | 1 Ball, 2 Ball, Red Ball, Blue Ball | This Alice world deals with random sampling from a group of red and blue balls, and in this standard students must learn about gaining information about populations by examining a sample of the population and understand random sampling. |
| | 7.SP.2 | 1 Ball, 2 Ball, Red Ball, Blue Ball | This standard has students use the random sample to draw inferences about the population from the data, and in this world students will predict the number of red and blue balls and see how the samples are simulated. |
| | 7.SP.3 | | - Comparing two different numerical distributions |
| | 7.SP.4 | | - Use measures of center and measures of variability from numerical data from random samples |
| | 7.SP.5 | Probability World | Understanding the definition of probability (the chance an event occurs is between 0 and 1, the likelihood that an event occurs...) |
| | 7.SP.6 | Probability World | Approximating the probability of a chance event by collecting data. |
| | 7.SP.7A | Probability World | Students should develop a uniform probability model and use it to determine the probability of different events. In the game, the user must enter the probability of choosing a random colored ball from a hole. |
| | 7.SP.7B | | - Develop a probability model that may not be uniform. |
| | 7.SP.8A | | - Probability of compound events |
| | 7.SP.8B | | - Represent sample spaces for compound events. |
| | 7.SP.8C | | - Design and use a simulation to generate frequencies of compound events. (simulate Alice?) |
| 8th Grade | | | |
| | 8.NS.1 | | - Irrational Numbers |
| | 8.NS.2 | | - Rational Approximations of irrational numbers |
| | 8.EE.1 | Exponent Laws | This world explains the laws and properties of exponents which students are required to know based on this standard. |
| | 8.EE.2 | | - Square root and cube root |
| | 8.EE.3 | Scientific Notation | Students should be able to know how to use and understand scientific notation. |
| | 8.EE.4 | | - Perform operations with numbers in scientific notation |
| | 8.EE.5 | | - Graph proportional relationships |
| | 8.EE.6 | | - Use similar triangles to calculate why the slope is the same between two points. |
| | 8.EE.7A | | - Linear equations with one variable and one solution |
| | 8.EE.7B | | - Solve linear equations |

| | | | |
|---|---|---|---|
| | 8.EE.8A | | - Students should be able to understand a system of equations and the corresponding point is their intersection. (Graphically) |
| | 8.EE.8B | Systems of Equations | Students should be able to solve systems of 2 linear equations which is what this world helps them practice. |
| | 8.EE.8C | | - Same as the above with real world applications. |
| | 8.F.1 | | - Definition of a function |
| | 8.F.2 | Move in XYZ and Mike's world | Students should be able to compare different functions GRAPHICALLY, also algebraically, numerically in tables, description, etc. |
| | 8.F.3 | | - Linear functions |
| | 8.F.4 | (Slope Quiz) | - Construct a function to create a linear relationship between two points |
| | 8.F.5 | | - Sketch graphs and describe relationship between two functions |
| | 8.G.1A | | - Lines and line segments |
| | 8.G.1B | | - Angles |
| | 8.G.1C | | - Parallel Lines |
| | 8.G.2 | | - Congruency between 2D figures with reflections, translations, and rotations |
| | 8.G.3 | | - Dilations, translations, rotations, and reflections on coordinates |
| | 8.G.4 | | - Similar 2D figures |
| | 8.G.5 | | - Angle sum of triangles |
| | 8.G.6 | | - Prove and explain the Pythagorean Thereom |
| | 8.G.7 | Pythagorean Prom (2D), Pythagorean Theorem in a 3D Problem | "Apply the Pythagorean Theorem to determine the unknown side lengths in right triangles in real-world and mathematical problems in two and three dimensions." |
| | 8.G.8 | Pythagorean Prom | This standard requires students to be able to use Pythagorean's Theorem to calculate the distance between 2 points. |
| | 8.G.9 | Volume Quiz | This world quizzes students on the volume formulas of different shapes including cones, cylinders, and spheres which are specified in this standard. |
| | 8.SP.1 | Bike Plot | Construct and interpret scatter plots. |
| | 8.SP.2 | Bike Plot | Students should know about the line of best fit for a scatter plot data and the end of this Alice world gives an example of finding the line of best fit for the data created by the user. |
| | 8.SP.3 | (Using Pearls to Understand Variables) | - Use linear equations to solve problems |
| | 8.SP.4 | | - Bivariate categorical data |
| High School | | | |
| | N-RN.1 | | - Rational exponents and their properties. |
| | N-RN.2 | | - Rewrite expressions involving radicals and rational exponents. |
| | N-RN.3 | | - Explain why the sum or product of two rational numbers is rational, the sum of a rational number and irrational number is irrational, and the product of a nonzero rational number and an irrational number is irrational. |
| | N-Q.1 | | - Use units as a way to understand problems and to guide the solution for multi-step problems. |
| | N-Q.2 | | - Define appropriate quantities for the purpose of descriptive modeling. |
| | N-Q.3 | | - Choose a level of accuracy appropriate to limitations on measurement when reporting quantities. |
| | N-CN.1 | | - Complex number i such that i^2 = -1. |
| | N-CN.2 | | - Use i^2 and the commutative, associative, and distributive properties to add, subtract, and multiply complex numbers. |
| | N-CN.3 | | - Find the conjugate of a complex number |
| | N-CN.4 | | - Represent complex numbers on the complex plane in rectangular and polar form. |
| | N-CN.5 | | - Represent addition, subtraction, multiplication, and conjugation of complex numbers geometrically. |
| | N-CN.6 | | - Calculate the distance between numbers in the complex plane. |
| | N-CN.7 | | - Solve quadratic equations with real coefficients that have complex solutions. |
| | N-CN.8 | | - Extend polynomial identities to complex numbers. |
| | N-CN.9 | | - The Fundamental Theorem of Algebra is true for quadratic polynomials. |
| | N-VM.1 | | - Recognize vector quantities as having both magnitude and direction. |
| | N-VM.2 | | - Find the components of a vector by subtracting the coordinates of an initial point from a terminal point. |
| | N-VM.3 | | - Solve problems involving velocity and other quantities represented by vectors. |
| | N-VM.4A | | - Add vectors end-to-end, component-wise, and by the parallelogram rule. |
| | N-VM.4B | | - Given 2 vectors in magnitude and direction form, determine the magnitude and direction of their sum. |
| | N-VM.4C | | - Understand vector subtraction. |
| | N-VM.5A | | - Represent scalar multiplication graphically |
| | N-VM.5B | | - Compute the magnitude of a scalar multiple |
| | N-VM.6 | | - Use matrices to represent and manipulate data. |
| | N-VM.7 | | - Multiply matrices by a scalar. |

| | | | |
|---|---|---|---|
| | N-VM.8 | The Matrix | Add, subtract, and multiply* matrices. This standard requires that students be able to multiply matrices of appropriate dimensions. In this Alice world, users are able to practice multiplying 2x2 matrices and learn the method for multiplying matrices. |
| | N-VM.9 | The Matrix | Students should know that matrix multiplication for square matrices is not commutative. In this world, they are able to input the numbers they want into the matrices that will be multiplied and can switch the values to see that they aren't commutative. |
| | N-VM.10 | | - Understand that the zero and identity matrix play a role in matrix addition and multiplication. |
| | N-VM.11 | | - Multiply a vector by a matrix of suitable dimensions. |
| | N-VM.12 | | - Work with 2x2 matrices as transformations in a plane. |
| | A-SSE.1A | | - Interpret parts of an expression. (terms, factors, and coefficients) |
| | A-SSE.1B | | - Interpret complicated expressions by viewing one or more of their parts as a single entity. |
| | A-SSE.2 | | - Use the structure of an expression and identify ways to rewrite it. |
| | A-SSE.3A | | - Factor a quadratic expression to reveal zeros of the function it defines. |
| | A-SSE.3B | | - Complete the square in a quadratic expression. |
| | A-SSE.3C | Exponent Laws | In this standard, students should be able to use the properties of exponents to transform expressions for exponential functions. This Alice world goes over all of the exponent laws with variables, which can be translated into functions and hold the same properties. |
| | A-SSE.4 | | - Derive the formula for the sum of a finite geometric series. |
| | A-APR.1 | System of Equations (2008), System of Equations (2011) | Understand that polynomials form a system analogous to the integers. Polynomials can be added, subtracted, and multiplied, and this Alice world quizzes students on how to add and subtract polynomials using a system of equations. |
| | A-APR.2 | | - Know and apply the Remainder Theorem. |
| | A-APR.3 | | - Identify zeros in polynomials. |
| | A-APR.4 | | - Prove polynomial identities and describe numerical relationships. |
| | A-APR.5 | | - The Binomial Theorem |
| | A-APR.6 | | - Rewrite simple rational expressions |
| | A-APR.7 | | - Understand that rational expressions form a system analogous to the rational numbers |
| | A-CED.1 | *Word problem challenges | Create equations and inequalities in one variable and use them to solve problems. |
| | A-CED.2 | | - Create equations in two or more variables. |
| | A-CED.3 | | - Represent constraints by equations or inequalities. |
| | A-CED.4 | | - Rearrange formulas to highlight a quantity of interest. |
| | A-REI.1 | Using Pearls To Understand Variables | This standard requires students to explain each step in solving a simple equation. The "Using Pearls to Understand Variables" Alice world explains variables using pearls and at the end it provides an example and shows how to solve an equation. |
| | A-REI.2 | | - Solve simple rational and radical equations in one variable. |
| | A-REI.3 | Using Pearls To Understand Variables | Students should be able to solve linear equations and inequalities and this Alice world deals with solving linear equations. |
| | A-REI.4A | | - Use the method of completing the square to transform any quadratic equation. |
| | A-REI.4B | | - Solve quadratic equations by inspection. |
| | A-REI.5 | | - Prove that, given a system of two equations in two variables, replacing one equation by the sum of that equation and a multiple of the other produces a system with the same solutions. |
| | A-REI.6 | System of Equations | This standard deals with solving systems of equations exactly and approximately, and the exact method is practiced in this Alice world. |
| | A-REI.7 | | - Solve a system of linear equations consisting of a linear equation and a quadratic equation. |
| | A-REI.8 | | - Represent a system of linear equations as a single matrix equation. |
| | A-REI.9 | | - Find the inverse of a matrix if it exists and use it to solve systems of equations. |
| | A-REI.10 | | - Understand that the graph of an equation with two variables is the set of all its solutions plotted in the coordinate plane. |
| | A-REI.11 | | - Explain why the x-coordinates of the points where 2 graphs intersect are solutions of the equations. |
| | A-REI.12 | | - Graph the solutions to a linear inequality. |
| | F-IF.1 | | - Understand that a function from one set (domain) connects to another set (range). |
| | F-IF.2 | | - Use function notation, evaluate functions for inputs in their domains, and interpret statements that use function notation. |
| | F-IF.3 | Nonvisual Arrays, Nonvisual Arrays and Recursion in Alice | Students should recognize that sequences are functions, and also defined recursive functions. Both of these Alice worlds use arrays to let students build functions and examine the sequences that they produce, and the second one focuses specifically on recursive functions such as Fibonacci's sequence and factorials. |

| | | | |
|---|---|---|---|
| | F-IF.4 | MoveinXYZ, Bird Graphing | This standard says that for a function that models a relationship between two quantities, interpret key features of graphs and tables (intercepts, intervals of increasing/decreasing, max and min, symmetry, etc.) Both of these Alice worlds deal with graphing functions that the user can examine and compare with other functions. MoveInXYZ uses polynomial functions while Bird Graphing can use all of the math functions built into Alice. |
| | F-IF.5 | MoveinXYZ, Bird Graphing | Students need to be able to relate the domain of a function to its graph and the quantitative relationship it describes. In these Alice world, students can view the graphs of a variety of functions and use the graphs to analyze the domains of the functions. |
| | F-IF.6 | | - Calculate and interpret the average rate of change of a function over a specified interval. |
| | F-IF.7A | MoveinXYZ, Bird Graphing | Graph linear and quadratic functions and show intercepts, maxima, and minima. |
| | F-IF.7B | Bird Graphing | Graph square root, cube root, and piecewise-defined functions including step and absolute value functions. The Bird Graphing Alice world is able to graph the square and cube root functions. |
| | F-IF.7C | MoveInXYZ | Graph polynomial functions, identifying zeros and factorizations when available. This Alice world allows users to create the functions that they want to graph up to the x^4 degree. |
| | F-IF.7D | Bird Graphing | Graph rational functions, identifying zeros and asymptotes. This world allows users to create rational functions if they can create them using the built-in Alice functions. |
| | F-IF.7E | Bird Graphing | Graph exponential and logarithmic functions showing intercepts and end behavior and trigonometric functions. Alice world functions contain these mathematical functions in the advanced math section that can be graphed in this world. |
| | F-IF.8A | | - Use the process of factoring and completing the square in a quadratic function to show zeros, extreme values, and symmetry. |
| | F-IF.8B | | - Use the properties of exponents to interpret expressions for exponential functions. |
| | F-IF.9 | | - Compare properties of two functions each represented in a different way. |
| | F-BF.1A | | - Determine an explicit expression, a recursive process, or steps for calculation from a context. |
| | F-BF.1B | | - Combine standard function types using arithmetic operations |
| | F-BF.1C | | - Compose functions [T(h(y))] |
| | F-BF.2 | | - Write arithmetic and geometric sequences both recursively and with an explicit formula. |
| | F-BF.3 | Bird Graphing | Identify the effect on the graph of replacing f(x) by f(x) + k, kf(x), and f (kx) for specific values of k. In this Alice world, the user can choose a function in Alice and then modify it by making the changes above and choosing a value of k to see how the graph changes for each one. |
| | F-BF.4A | | - Solve an equation of the form f(x) = c and write an expression for the inverse. |
| | F-BF.4B | | - Verify by composition that one function is the inverse of another |
| | F-BF.4C | | - Read values of an inverse function from a graph or table |
| | F-BF.4D | | - Produce an invertible function from a non-invertible function by restricting the domain. |
| | F-BF.5 | | - Understand the inverse relationship between exponents and logarithms. |
| | F-LE.1A | Nonvisual Arrays in Alice | In this standard, students should be able to prove that linear functions grow by equal differences over equal differences. This Alice world shows how functions grow at an equal rate and helps them practice with a quiz to calculate these values. |
| | F-LE.1B | | - Recognize situations in which one quantity changes at a constant rate per unit interval |
| | F-LE.1C | | - Recognize situations in which a quantity grows or decays by a constant percent rate |
| | F-LE.2 | | - Construct linear and exponential functions including arithmetic and geometric sequences |
| | F-LE.3 | Bird Graphing, MoveInXYZ | This standard wants students to observe quantities increasing exponentially, linearly, quadratically, polynomially, etc. in graph and table form. These Alice worlds present these values in graphical form. |
| | F-LE.4 | | - For exponential models, express as a logarithm of the solution. |
| | F-LE.5 | | - Interpret the parameters in a linear or exponential function in terms of a context |
| | F-TF.1 | | - Understand radian measure of an angle |
| | F-TF.2 | | - Explain how the unit circle in the coordinate plane enables the extension of trigonometric functions to real numbers |
| | F-TF.3 | | - Use special triangles to determine geometrically the values of sin, cos, and tan for pi/3, pi/4, and pi/6 |
| | F-TF.4 | | - Use the unit circle to explain symmetry and periodicity of trigonometric functions |
| | F-TF.5 | | - Choose trig functions to model periodic phenomena with specified amplitude, frequency, and midline |

| | | | |
|---|---|---|---|
| | F-TF.6 | | - Understand that restricting a trig function to a domain which is always increasing/decreasing allows its inverse to be constructed. |
| | F-TF.7 | | - Use inverse functions to solve trig equations |
| | F-TF.8 | | - Prove the Pythagorean identity sin^2 + cos^2 = 1 |
| | F-TF.9 | | - Prove the addition and subtraction formulas for sin, cos, and tan |
| | G-CO.1 | | - Know the precise definitions of angle, circle, perpendicular and parallel lines, line segments, point, line, distance, arc, etc. |
| | G-CO.2 | | - Represent transformations in the plane using transparencies and geometry software |
| | G-CO.3 | | - Given a rectangle, parallelogram, trapezoid, or regular polygon, describe the rotations and reflections |
| | G-CO.4 | | - Develop definitions of rotations, reflections, and transformations |
| | G-CO.5 | | - Given a geometric figure, draw the transformed figure |
| | G-CO.6 | | - Use geometric descriptions of rigid motions to transform figures and to predict the effect of a given rigid motion on a given figure |
| | G-CO.7 | | - Use the definition of congruence in terms in terms of rigid motions |
| | G-CO.8 | | - Explain how the criteria for triangle congruence follow from the definition of congruence |
| | G-CO.9 | | - Prove theorems about lines and angles |
| | G-CO.10 | | - Prove theorems about triangles |
| | G-CO.11 | | - Prove theorems about parallelograms |
| | G-CO.12 | | - Make formal geometric constructions with a variety of tools |
| | G-CO.13 | | - Construct an equilateral triangle, a square, and a regular hexagon inscribed in a circle |
| | G-SRT.1A | | - A dilation takes a line not passing through the center of the dilation to a parallel line |
| | G-SRT.1B | | - The dilation of a line segment is longer or shorter in the ratio given by the scale factor |
| | G-SRT.2 | | - Given two figures, use the definition of similarity and decide if they are similar |
| | G-SRT.3 | | - Use properties of similarity transformation to establish the AA criterion for 2 triangles to be similar |
| | G-SRT.4 | | - Prove theorems about triangles. |
| | G-SRT.5 | | - Use congruence and similarity for triangles to solve problems |
| | G-SRT.6 | | - Understand that by similarity, side ratios in right triangles are properties of the angles in the triangle |
| | G-SRT.7 | | - Explain and use the relationship between sin and cos of complementary angles |
| | G-SRT.8 | | - Use trigonometric ratios and Pythagorean Theorem to solve right triangle in applied problems |
| | G-SRT.9 | | - Derive the formula A = 1/2ab sin(c) for the area of a triangle |
| | G-SRT.10 | | - Prove the Law of Sines and Cosines |
| | G-SRT.11 | | - Understand and apply the Law of Sines and the Law of Cosines |
| | G-C.1 | | - Prove that all circles are similar |
| | G-C.2 | | - Identify and describe relationships among inscribed angles, radii, and chords |
| | G-C.3 | | - Construct the inscribed and circumscribed circles of a triangle and prove properties of angles and for a quadrilateral inscribed in a circle. |
| | G-C.4 | | - Construct a tangent line from a point outside a given circle to the circle |
| | G-C.5 | | - Derive using similarity the fact that the length of the arc intercepted by an angle is proportional to the radius |
| | G-GPE.1 | | - Derive the equation of a circle of given center and radius using the Pythagorean Theorem |
| | G-GPE.2 | | - Derive the equation of a parabola given a focus and directrix |
| | G-GPE.3 | | - Derive the equations on ellipses and hyperbolas |
| | G-GPE.4 | | - Use coordinates to prove simple geometric theorems algebraically |
| | G-GPE.5 | | - Prove the slope criteria for perpendicular and parallel lines |
| | G-GPE.6 | | - Find the point on a directed line segment between two given points |
| | G-GPE.7 | | - Use coordinates to computer perimeters and areas of polygons |
| | G-GMD.1 | | - Give an informal limit argument for the formulas for the circumference of a circle, area of a circle, volume of a cylinder, pyramid, and cone. |
| | G-GMD.2 | | - Give an informal argument using Cavalieri's principle |
| | G-GMD.3 | Volume Formulas | Use volume formulas for cylinders, pyramids, cones, and spheres to solve problems. This world will help students learn the formulas of the volumes for different 3D shapes. |
| | G-GMD.4 | | - Identify the shapes of 2D cross sections of 3D shapes |
| | G-MG.1 | | - Use geometric shapes, their measures, and their properties to describe objects |
| | G-MG.2 | | - Apply concepts of density based on area and volume in modeling |
| | G-MG.3 | | - Apply geometric methods to solve design problems |
| | S-ID.1 | Bike Plot | This standard wants students to represent data with plots on a real number line, dot plots, histograms, and box plots. This Alice world has the user create data and then plot the points on a graph. |
| | S-ID.2 | | - Use statistics appropriate to the shape of the data distribution to compare center and spread |

| | | | |
|---|---|---|---|
| | S-ID.3 | | - Interpret differences in shape, center, and spread in the context of data sets |
| | S-ID.4 | | - Use the mean and sd of a data set to fit it to a normal distribution and to estimate the population percentages |
| | S-ID.5 | | - Summarize categorical data for two categories in two-way frequency tables |
| | S-ID.6A | Bike Plot | This standard has students find a function to the data and use functions fitted to data to solve a problem. In this Alice world, after the user plots the points from the data that they create, the best-fit line is drawn and predicts a future value. |
| | S-ID.6B | | - Informally asses the fir of a function by plotting and analyzing residuals |
| | S-ID.6C | | - Fit a linear function for a scatter plot that suggests linear association |
| | S-ID.7 | | - Interpret the slope and the intercept of a linear model. |
| | S-ID.8 | | - Compute and interpret the correlation coefficient of a linear fit. |
| | S-ID.9 | | - Distinguish between correlation and causation |
| | S-IC.1 | | - Understand statistics as a process for making inferences about population parameters |
| | S-IC.2 | | - Decide if a specified model is consistent with results from a given data-generation process |
| | S-IC.3 | | - Recognize the purposes of and differences among sample surveys, experiments, and observational studies |
| | S-IC.4 | | - Use data from a sample survey to estimate a population mean or proportion |
| | S-IC.5 | | - Use data from a randomized experiment to compare two treatments |
| | S-IC.6 | | - Evaluate reports based on data |
| | S-CP.1 | Can I Get Your Number?, 1 Ball, 2 Ball, Red Ball, Blue Ball | This standard wants the students to describe events of subsets of a sample space. Both of these worlds deal with random sampling and creating subsets. The first creates a random set of numbers to form a phone number and the second is randomly sampling from a group of balls. |
| | S-CP.2 | | - Definition of independent events |
| | S-CP.3 | | - Understand the conditional probability of A given B and interpret their independence |
| | S-CP.4 | | - Construct and interpret two-way frequency tables |
| | S-CP.5 | | - Recognize and explain the concept of conditional probability |
| | S-CP.6 | | - Find the conditional probability of A given B |
| | S-CP.7 | | - Apply the Addition Rule of probabilities |
| | S-CP.8 | | - Apply the Multiplication Rule of uniform probabilities |
| | S-CP.9 | Line Up | This standard requires students to use permutations and combinations to compute probabilities, and this Alice world shows the user how to use permutations to find the number of possible ways to order a group of people in a line. |
| | S-MD.1 | | - Define a random variable for a quantity of interest |
| | S-MD.2 | | - Calculate the expected value of a random variable and interpret it as the mean of the probability distribution |
| | S-MD.3 | | - Develop a probability distribution and find the expected value for a random variable defined for a sample space that can be calculated |
| | S-MD.4 | | - Develop a probability distribution for a random variable defined for a sample space assigned empiracally |
| | S-MD.5A | | - Find the expected payoff for a game of chance |
| | S-MD.5B | | - Evaluate and compare strategies of expected values |
| | S-MD.6 | Ready, SET, Go!, War, Choosing Random People From a Class | This standard wants students to use probabilities to make fair decisions. Both of these worlds use probabilities to make decisions within them. The first two use probability in a card game and the last one selects a random student from a class. |
| | S-MD.7 | | - Analyze decisions and strategies using probability concepts |

| | Standard | Alice World/Concept | Description |
|---|---|---|---|
| Level 1 (K-6) | | | |
| Level 2 (6-9) | | | |
| Level 3A (9-10) | | | |
| Level 3B (10-11) | | | |
| Level 3C (11-12/AP) | | | |
| | | | |
| **Level 2** | | | |
| | CT.1 | A lot of our Alice tutorials have problems to solve at the end such as the recursion and nonvisual array tutorial, which shows the user how to build an Alice world that calculates Fibonacci's sequence and asks them to use the same algorithm that creates a world that calculates factorials. We will also have different challenges created for students, where a problem will be given to them and they must come up with the algorithm to solve it in Alice. Trigonometry Prom is an example where the prince needs to find out how far he needs to go to meet the princess under the disco ball. | This standard requires that the students be able to figure out the basic steps in algorithmic problem solving. |
| | CT.2 | In Alice, users are allowed to use the commands "Do Together" and "For all Together" with lists to run multiple instructions at the same time. | Process of parallelization to solve problems |
| | CT.3 | | - Define an algorithm as a sequence of instructions. |
| | CT.4 | Alice allows problems to be solved in different ways. For example, you are able to use lists or arrays to hold a collection of information to use in the program. | In this standard, students should be able to evaluate ways that different algorithms can be used to solve the same problem. |
| | CT.5 | In addition to acting out the searching and sorting algorithms, students could watch Alice animations of different algorithms to sort a group of people by heights or find a specific character in a list, while pausing it and asking questions about what will happen next to help them learn the algorithms. Actually programming these is more for level 3. | Act out searching and sorting algorithms |
| | CT.6 | The tutorials provide detailed instructions on how to complete Alice worlds so students should be able to follow them and create a final project based on the tutorial that they complete. | Describe and analyze a sequence of instructions being followed. |
| | CT.7 | Alice is great for this because data can be represented in graph form (Bike Plot, MoveinXYZ, etc.), as text (Challenges, eventual word problem world), numbers (Fractions, Rounding Game, most math worlds), pictures (billboards), and many other objects (for example, bunnies in Fibonacci sequence and balls in probability world, pearls in Using Pearls to Understand Variables, etc.) Students can generate the data when the world runs, and then store it in lists or arrays to analyze it. Examples of this are the Boat Averages worlds where the world itself collects the times it takes the boat to go through each hoop and the distance or time per hoop, and uses it to calculate the average speed of the boat. | This standard requires that students be able to represent data in various ways (text, sounds, pictures, numbers,...). |
| | CT.8 | These Alice worlds take data or functions input by students and displays them in graphical form- MoveinXYZ, Bike Plot, Mike's graph world, a modified Bar Chart object. Bike plot world physically presents the speed of a bicycle based on when the user clicks and plots the data. | Students must use visual representations to display problem states, structures, and data with this standard. |
| | CT.9 | Most of our Alice project and educational tutorials deal with having students interact with content-specific models. For example, in the Science category, students can interact with a model of the lac operon, a helium molecule, a model of the solar system and planets, and many more. | Students have to interact with content specific models in this standard. |

| | | | |
|---|---|---|---|
| | CT.10 | Alice can be used to simulate problems that need to be modeled or simulated. We will be adding Alice worlds dealing with word problems for students to practice solving and it will help them visualize and model the problem in their mind to help them solve it. | Evaluate what kinds of problems can be solved with modeling and simulation. |
| | CT.11 | | - Analyze the degree to which a computer model represents the real world. |
| | CT.12 | In the Challenges section, there is a problem that the student must solve by filling in smaller functions and methods to achieve the desired results. More advanced challenges will have more sections of code for the student to fill in and find the subproblems to solve. | Decompose a problem into several subproblems |
| | CT.13 | Alice allows for computer science concepts such as hierarchy and abstraction in the use of parameters, local/global variables, inheritance, object methods, etc. | Understand the notion of hierarchy and abstraction. |
| | CT.14 | The Alice materials we have made in Mathematics show connections between math and programming and how they overlap. Alice also has many built-in math based functions such as <, >, =, arithmetic, sin, cos, etc that can be implemented into your programs. Alice can be used to help students practice math concepts such as in Basketball Math, or it can be used to make their own math projects and explore a math subject in Alice such as probability world. | Examine connections between mathematics and computer science |
| | CT.15 | The teacher lesson plans page on the Duke Adventures in Alice site provides many examples of how programming in Alice can relate to other disciplines. Examples of this include using Alice for a book report, a history project, math quizzes, or foreign language quizzes. | Interdisciplinary examples of computational thinking. |
| | CL.1 | Alice itself is a productive multimedia tool that supports learning through a new medium. Students can use Alice for projects, presentations, quizzes, games, etc. | Apply productivity/multimedia tools to support learning through curriculum. |
| | CL.2 | The tutorials on our page have instructions on how to build the worlds that we have. It is possible to have students collaborate on a project to make an Alice world in a group setting by following the instructions given in the tutorials. | Students must collaboratively design, develop, publish, and present products using technology. |
| | CL.3 | | - Collaborate with peers, experts, and others using collaborative practices such as peer programming, team projects, and group active learning. |
| | CL.4 | | - Exhibit dispositions necessary for collaboration. |
| | CPP.1 | | - Select appropriate tools and technology resources to solve problems |
| | CPP.2 | Alice is an example of a multimedia tool that can be used in the classroom to help students engage in their learning. It is also a beginning programming tool that can help students move on to other programming and multimedia tools. | Use a variety of multimedia tools. |
| | CPP.3 | Students can use Alice to design and present products and it is a technology resource. The teacher can have the students be creative and create a story or game using Alice, then present their ideas and final product to the class. | Design, develop, publish, and present products using technology resources. |
| | CPP.4 | | Students will have to demonstrate an understanding of algorithms and their practical application. |
| | CPP.5 | Our Alice tutorials page has many examples of tutorials on how to use these program solutions such as loops, conditional statements, variables, logic, etc. in an Alice world to solve a problem. | Implement problem solutions using a programming language (loops, conditional statements, logic, expressions, variables, and functions) |

| | | | |
|---|---|---|---|
| | CPP.6 | There is an annual competition that students can enter where they must create Alice worlds that teach about computer and internet safety in it's animation. Students can build worlds for that and at the same time learn about good practices in information security. | Demonstrate good practices in personal information security |
| | CPP.7 | Several teachers have come up with Alice worlds to help students learn about different jobs and occupations such as "Career Day", "Business Careers", and "Career Decisions". This type of idea can also be applied to animate how specific jobs use computing and technology. | Identify interdisciplinary careers that are enhanced by computer science |
| | CPP.8 | | - Demonstrate dispositions amenable to open-ended problem solving and programming |
| | CPP.9 | Alice worlds can take data created by the user and implement it into the world for them to analyze. Examples of this are Boat World Averages and Bike Plot, where the user takes data that he creates in the world to calculate the average boat speeds or plot the speed of the bicycle. | In this standard, students should collect and analyze data that is collected from multiple runs of a computer program. |
| | CD.1 | | - Recognize that computers are devices that execute programs |
| | CD.2 | | - Identify electronic devices that contain computational processors |
| | CD.3 | | - Demonstrate an understanding of the relationship between hardware and software |
| | CD.4 | | - Use accurate, appropriate terminology when communicating about technology. |
| | CD.5 | | - Apply strategies for identifying and solving routine hardware problems that occur during everyday computer use. |
| | CD.6 | | - Describe major functions and components of computer systems and networks. |
| | CD.7 | | - Describe what distinguishes humans from machines. |
| | CD.8 | | - Describe ways in which computers use models of intelligent behavior. |
| | CI.1 | | - Exhibit legal and ethical behaviors when using information and technology and discuss consequences of misuse. |
| | CI.2 | | - Demonstrate knowledge of changes in information technologies over time and the effects of those changes |
| | CI.3 | | - Analyze the positive and negative impacts of computing on human culture |
| | CI.4 | | - Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources in real world problems. |
| | CI.5 | | - Describe ethical issues that relate to computers and networks |
| | CI.6 | | - Discuss how the unequal distribution of computing resources in global economy raises issues of equity, access and power. |

## Appendix 2: Tutorials

This appendix contains all of our tutorials for students to build an entire Alice world as a project from start to finish. Some of these examples are math-related, but the focus is mainly on programming in Alice and computer science concepts. These tutorials give the students step-by-step instructions on how to complete a certain Alice world and several of them also have small challenges for the students to try at the end to modify or add new ideas to the Alice world that they just created.

# Arrays

by Chris Brown
under Prof. Susan Rodger
Duke University
June 2012

# Arrays vs. Lists

- Arrays are similar to lists, but elements in arrays are ordered and elements in a list are unordered. When traversing an array, one uses a loop (complicated version) to step through indexing particular items. Not all elements need to be processed. One could access every other element. When traversing a list, the order does not matter and the user wants to handle every element. Use "For all in order" or "For all together" to process the elements in a list.

# Arrays

- The purpose of this tutorial is to demonstrate how to use arrays in Alice worlds. An array is an ordered collection of objects stored by an index. Alice has two types of arrays: visual and nonvisual. Visual arrays put the items on the array index number to show their location in the array. This is useful for objects that need to stand in a line. In nonvisual arrays, however, the items can be put anywhere in the Alice world and still function as an array or they can be used for an array of numbers, strings, objects, etc.

# Standards

**CSTA Standard 5.3.B-** Computer Science Concepts and Practices (CT):

Students will be able to…

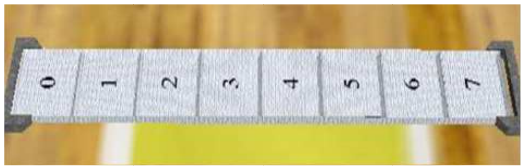
"6. Compare and contrast simple data structures and their uses (e.g., arrays and lists)."

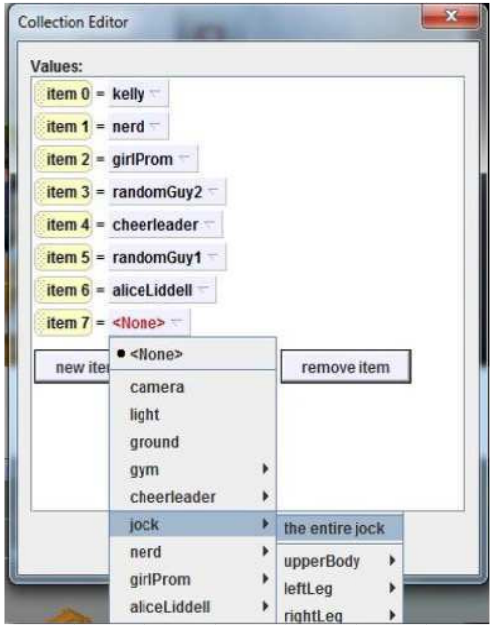
# Getting Started

- You want to add in different people to put into this world. You can find them under the "People" tag or use the students under the "High School" tag. I used 8 people in my world to form the array as well as the coach to describe what's going on (9 people total). Now we're ready to build the array!

# Visual Arrays

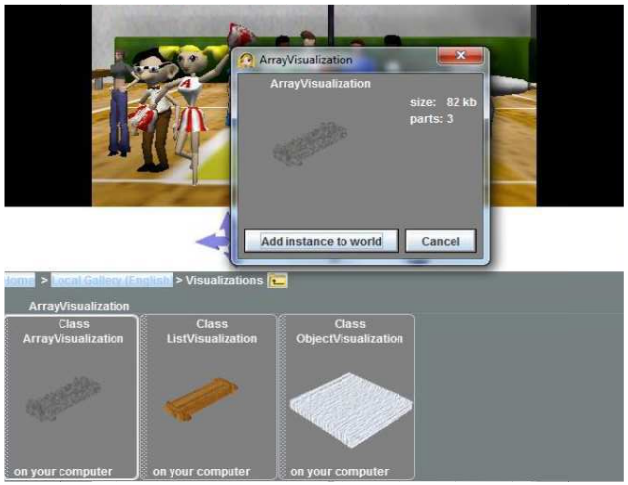
- After adding the ArrayVisualization, a menu will pop up for you to add elements to the array. Select "new item" to add objects into the array and choose the items that you would like to add at the specified index. Make sure to add "the entire ___". You should add 8 people and have one person left over.



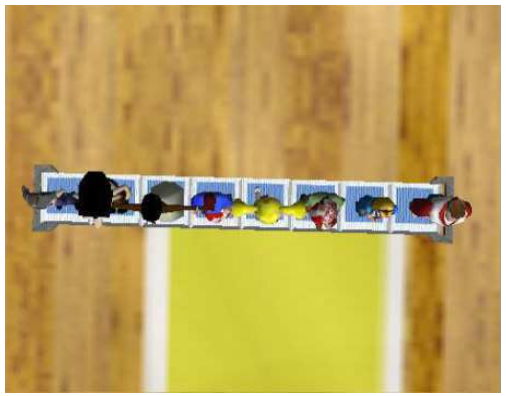

# Visual Arrays

- To use a visual array, go to the "Visualizations" folder in the Alice Local Gallery and add an ArrayVisualization to your world.




# Visual Arrays

- And here is your visual array! Click and drag the ArrayVisualization object to turn it and move it around to get the entire array to fit the screen and facing the camera. Note that the people in the array will move with it. Don't click on the people and move them or this will cause problems accessing the array later on. Click "Undo" if you accidentally move a person.



# Using the Array

- Now we will go over several uses for the visual array including modifying each element of the array, every other element of the array, choosing random elements in the array, accessing specified indices of the array, and swapping elements in the array

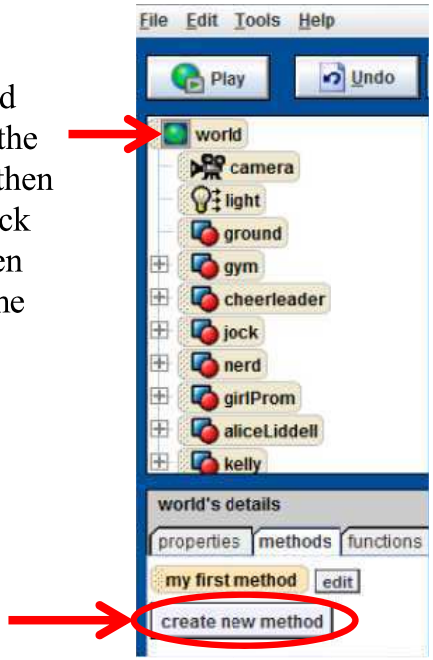
- *(Note that for arrays with n objects, the first element is at index 0 and the last element has the position n-1)!*

# Loops

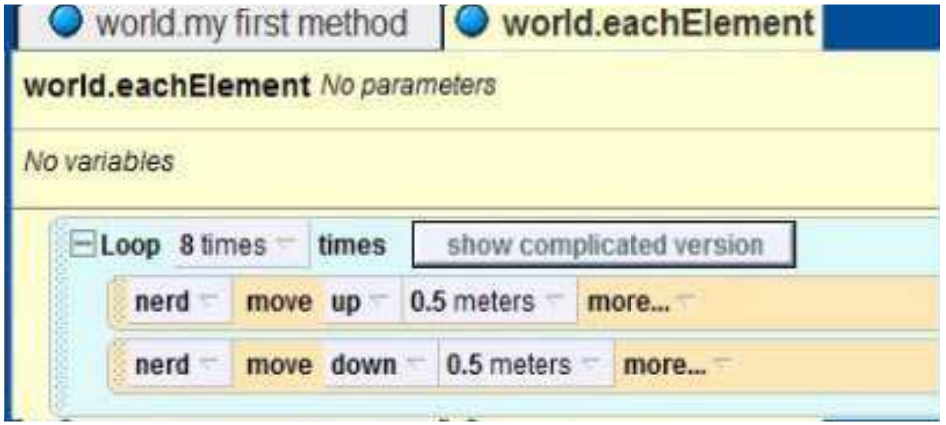
- Loops are very important for iterating through the objects of an array. In this method, drag in a Loop from the bottom of the screen into the "Do Nothing". Have the loop go from 0 to the number of elements in the array, in this case we want to choose 8. When the menu comes up, go down to "other…" and enter 8 into the calculator.


# Using the Array

- Create a new world method by clicking on "world" in the object tree on the left and then under "world's details" click on the methods tab and then "create new method". Name this method *eachElement*.



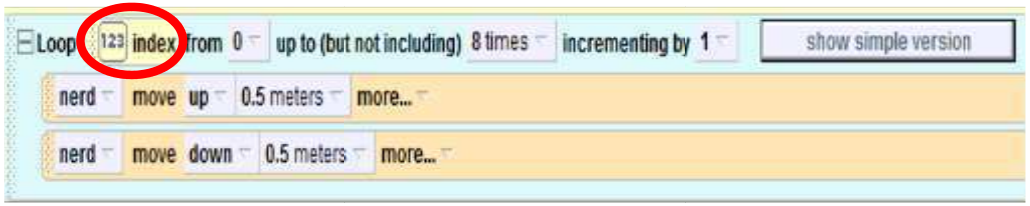

# Each Element




- To get each element to do something in order, click on any element of the array in the object tree and go to the methods tab. The will be your "default person" that you will use for all of the characters' methods that you want the elements of the array to perform. Have your object move up ½ meter then move back down.

# Loops

- If you click on "show complicated version", you will see more information about how loops work. The *index* variable is used to traverse the array and its value will increase from 0 to 7 each time you go through the loop.

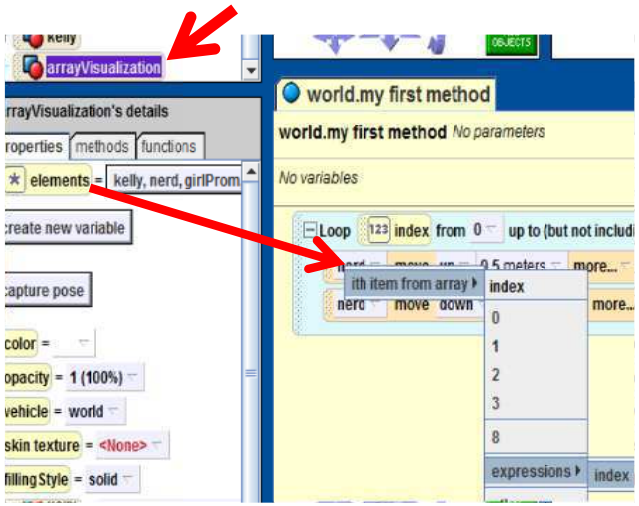



# Each Element

- And that concludes how to modify each element in an array in Alice! Make sure to test this method by changing the Events panel at top right corner select "When the world starts, do → world.eachElement" to run this when the world starts.


# Each Element

- Next, click on arrayVisualization in the object tree and go to the properties panel. Under properties, drag the *elements* property over your "default person" that you used before to get the instructions and select "ith item from array" → "expressions" →"index" to access the object at position *index* in the array.



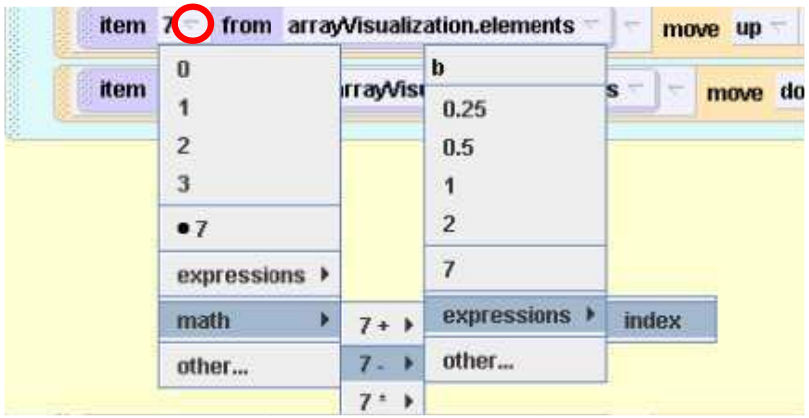

# Each Element (Reverse)

- Now, we want to go through the elements of the array backwards. Unfortunately, the complicated loop will not let you increment by negative numbers, so we will have to use the formula 7 – *index* to process the correct position in the array. Note that as index increases, 7 – *index* will decrease.

| index | 7 – index (current position of the array) |
|---|---|
| 0 | 7 |
| 1 | 6 |
| 2 | 5 |
| 3 | 4 |
| 4 | 3 |
| 5 | 2 |
| 6 | 1 |
| 7 | 0 |

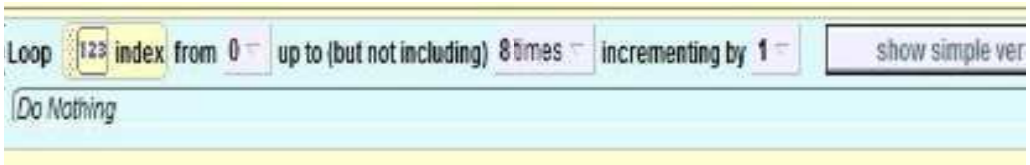
# Each Element (Reverse)

- To go through the array in reverse, you only need to change one thing. When you select the ith item from array, select the highest index of the array, in this case, 7. Then, click on the down-arrow next to the number and select "math" → "7 –" → "expressions" → "index".



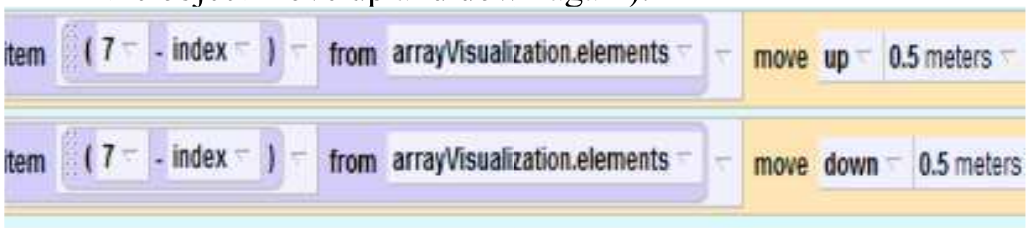

# Every Other Element

- Next, we will discover how to change every other element in an array. Start by creating a new world method called *everyOtherElement*, and adding a loop into the method from 0 to 8. Make sure to hit the "show complicated version" button of the loop, and your loop should be a longer structure like this:




# Each Element (Reverse)

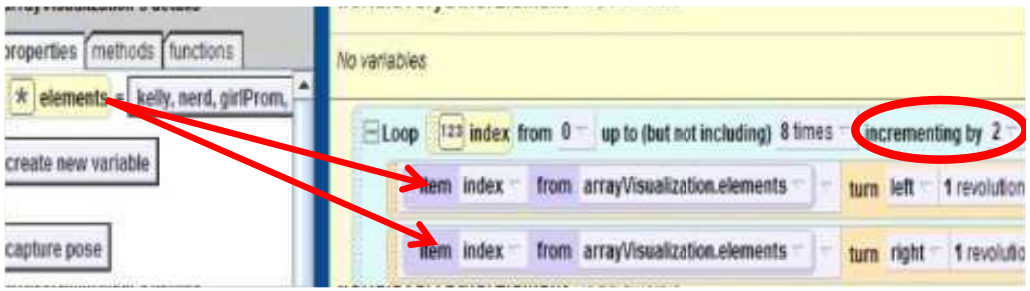
- If you create a method called *reverse* to try this out, the two lines inside of the loop should look like this (Have the object move up and down again):




- Now, add a line at the beginning of the method to have the coach say, "Now in reverse", under the coach's methods before the loop starts and you are done with the *reverse* method. Test this method out by changing "When the world starts, do → world.reverse" in the Events editor.

# Every Other Element

- There are a couple of ways to modify the elements of an array at a particular interval. The simplest way is to change the "incrementing by" portion of the loop to 2 or whatever interval you want. Add the methods that you would like done in the loop and only every other element will act out the method. We chose to have them turn left and then right 1 revolution each direction. Remember, when asked for the ith item from array after dragging the elements property into the method, go to "expressions" → "index".

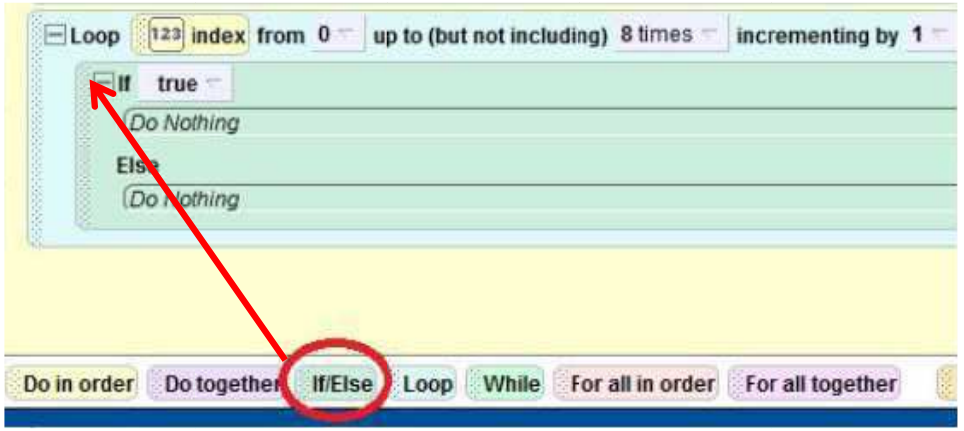


# Every Other Element

- On your own, try creating a method that has every third element in the array do the action of your choice. Name this method *everyThirdElement*. Don't forget to test your new method to see if it works by changing the event "When the world starts, do –"…

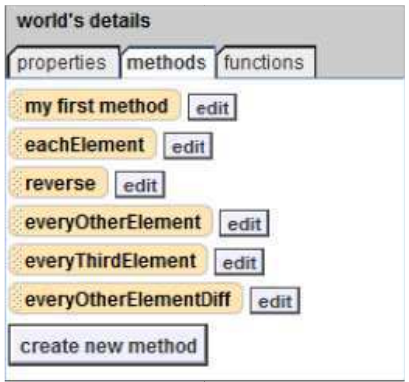
# Every Other Element Different

- To do this, you first need to drag an If/Else block inside of your loop from the bottom of the window and select true for now.

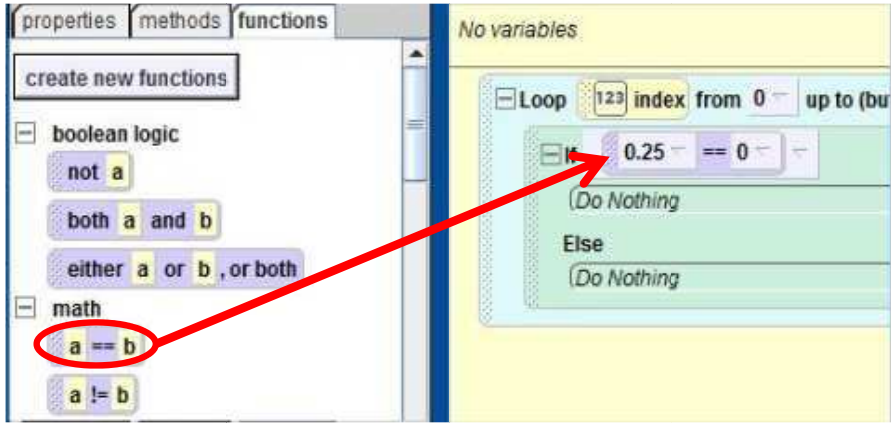



# Every Other Element Different

- Another way to modify every other element in an array is to check if the index is even or odd, or is divisible by some number. This is better if you want every object to do an action, then every other object to do a different action. Create a new world method called *everyOtherElementDiff*.

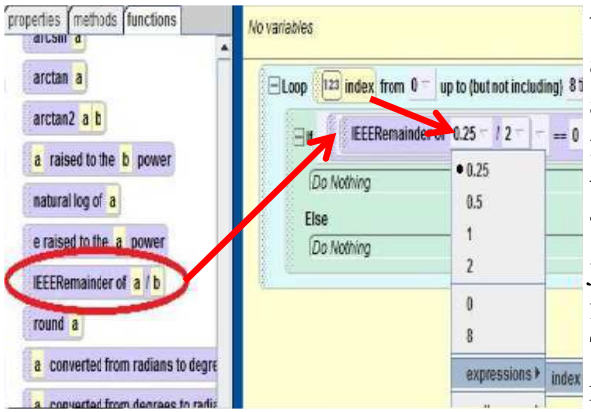



# Every Other Element Different




- Next, go to the world's functions tab and select "a==b". Let a be any arbitrary number (I chose 0.25 but this value will be replaced) and let b = 0. Remember that you may need to select "other…" and type in 0.

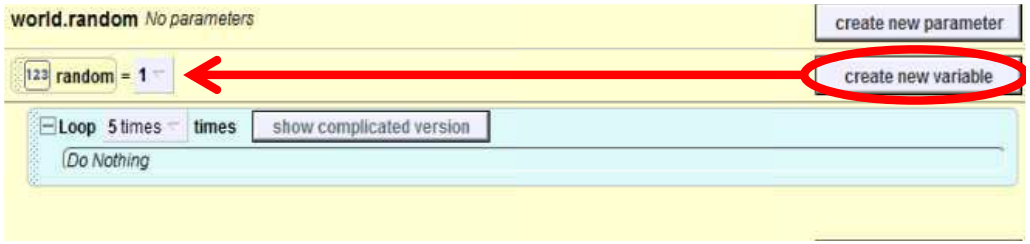
# Every Other Element Different



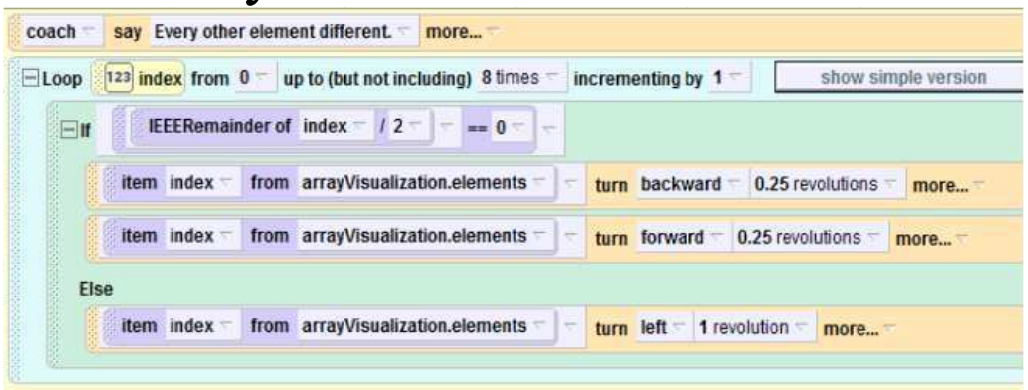

- Scroll down in the functions tab, select "IEEERemainder of a/b" and drag it over the arbitrary value that you used for a before. When asked for the value of a, go to "expressions" → "index" **OR** just drag the index variable from the loop, and let b = 2. This will check to see if the remainder of the index divided by 2 returns a remainder of zero or not, determining if the index is odd or even.

# Random Elements

- Now, we'll go over how to choose random elements from an array. First, create a new world method called *randomElements* and add a loop from 0 to the number of random positions that you want in this method (I chose 5). You will also want to create a new number variable and call it *random* that will be used to calculate the random indices to be modified. You should see your variable show up to the left.



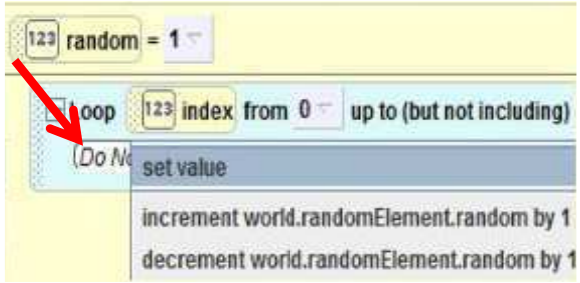

# Every Other Element Different

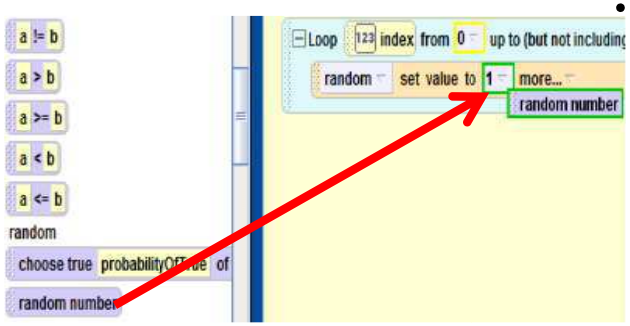



- Now, have the objects at the even indices turn backward then forward ¼ revolution, and the objects at odd indices just turn left one revolution. (Remember to use the "default person", then replace them with the elements property of the array). And that concludes modifying every other element in an array! Have your extra person say "Every other element different." and test this method by changing the event when the world starts.

# Random Elements

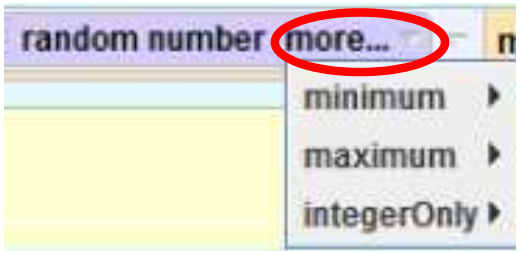
- Now drag the *random* variable into the loop and when prompted, set the value of *random* to any value for now. In the example below I set the value to 1.




- Then you want to go to the world's functions tab and find the "random number" function under the random heading. Drag that over the number that you originally set as the value for *random*.

# Random Elements

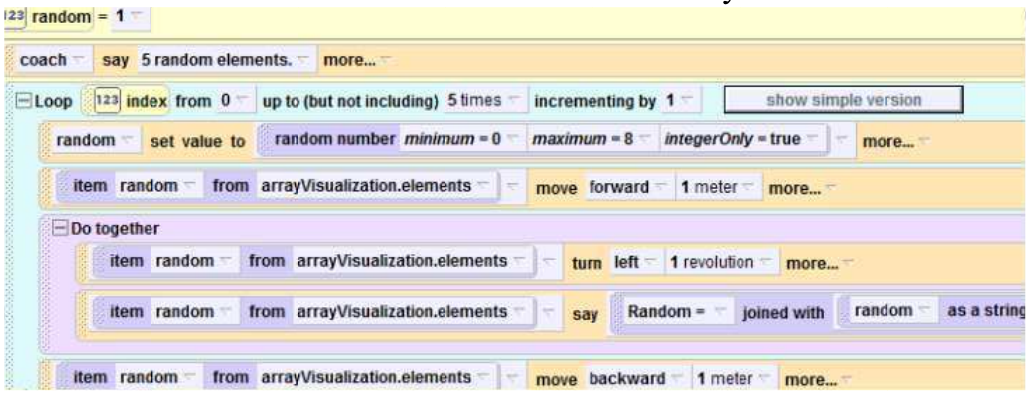



- Click on "more…" next to the random item that you just dragged into the method. We only need integer values from 0 to 7 to get the positions in the array, so set the minimum value to 0, the maximum value to 8, and integerOnly to be true.


- Note: This statement generates random numbers up to but not including the maximum value (8), making 7 the largest possible integer produced.

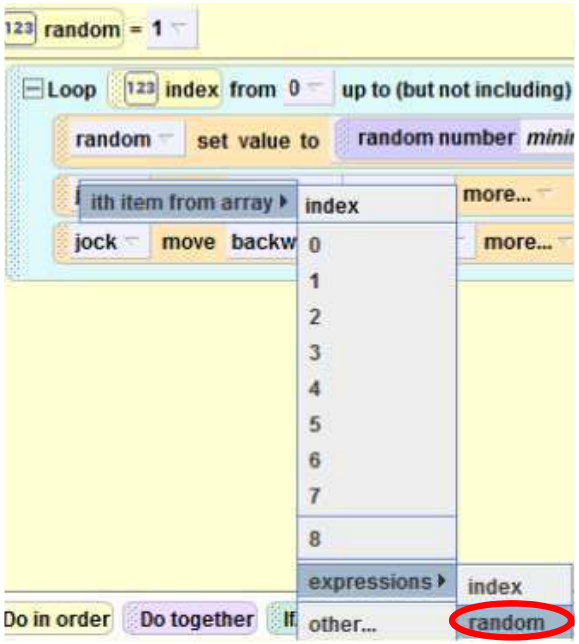
# Random Elements

- And that concludes the portion on random elements in an array. The code below has the coach say "5 random elements", then selects 5 random objects from the array that move forward, turn and say their position number, then move back. Make sure you don't forget to change the event "When the world starts do" → *randomElements* and run your world.




# Random Elements

- This finishes creating the random number generator. Now, fill in the code with the instructions that you want, using your "default person" and then replacing him with the elements of the array from arrayVisualization's properties. But this time, instead of choosing "index" as the ith element from array, go to "expressions" → **"random".** Our random person moves forward 1 meter, turns and says the value of random, and then moves back to their position.

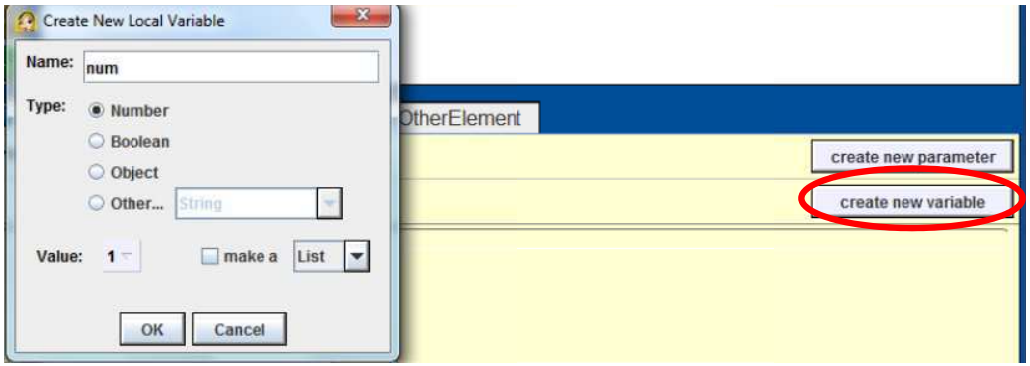



# Specific Elements

- Now we will see how to modify certain elements of an array. For example, if you only want the certain positions in the array to do something. We want the user to choose 2 objects and then have them switch places. To accomplish this, you will first need to create a new world method we will call *specificElements* and then create a second method to swap the two elements chosen.

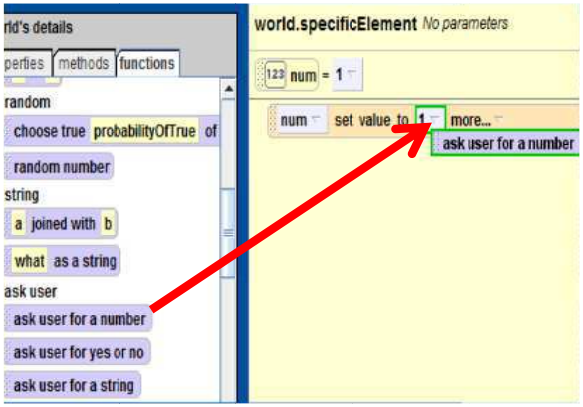
# Specific Elements

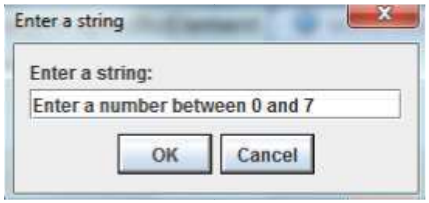
- Now we need to create a variable to obtain the user input. Click on "create new variable" in the top right corner of the method. Make this a number variable and call it *num*.




# Specific Elements

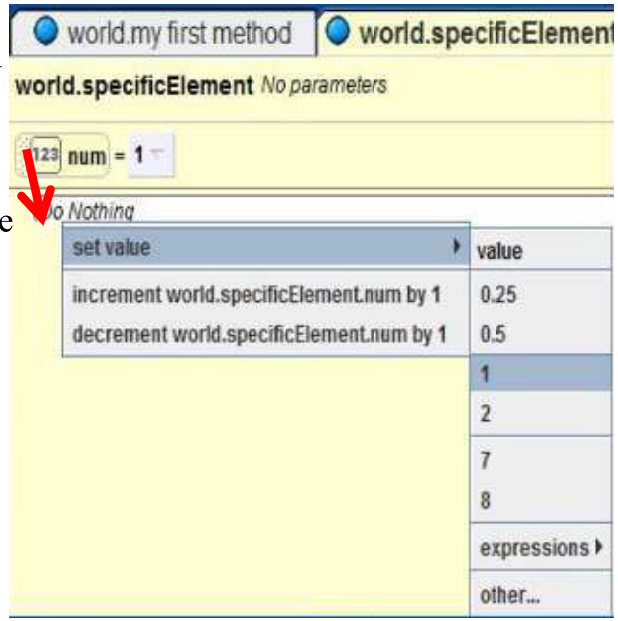
- Then, go to the world functions tab and drag the "ask user for a number" function over the "1" that we previously put for the value of *num*. This will create a pop-up box to ask the user for a number while the world is running. When prompted for a question to enter, type in a string to ask the user for a number between 0 and 7.



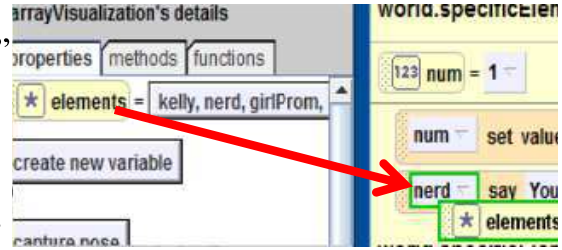

# Specific Elements

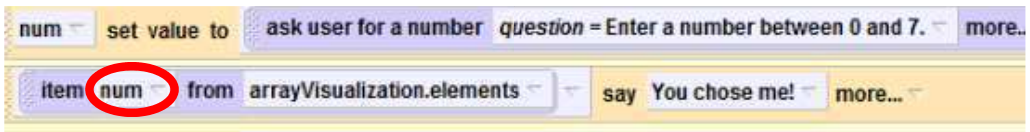
- The new variable should appear at the top of the method editor. Drag the *num* variable into the method, and set the value to 1 for now.




# Specific Elements

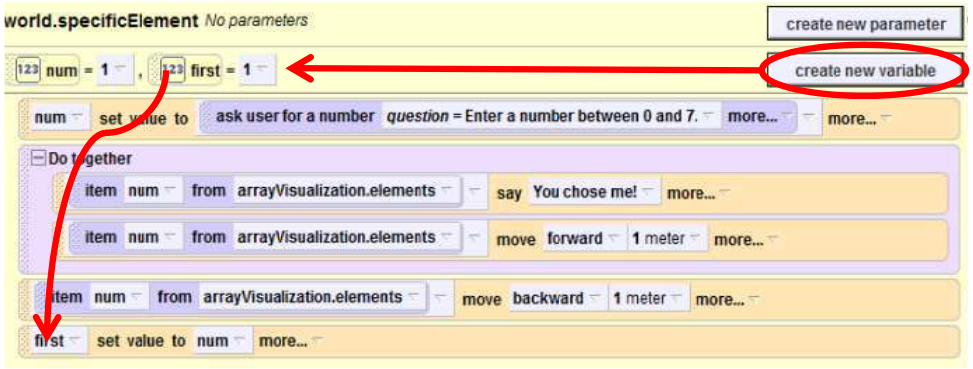
- Now, add whatever instructions that you want into the method by using the methods of the "default object" in the array. When dragging the elements of the array from arrayVisualization's properties over the object name this time, select *num* under "ith element from array" → "expressions".

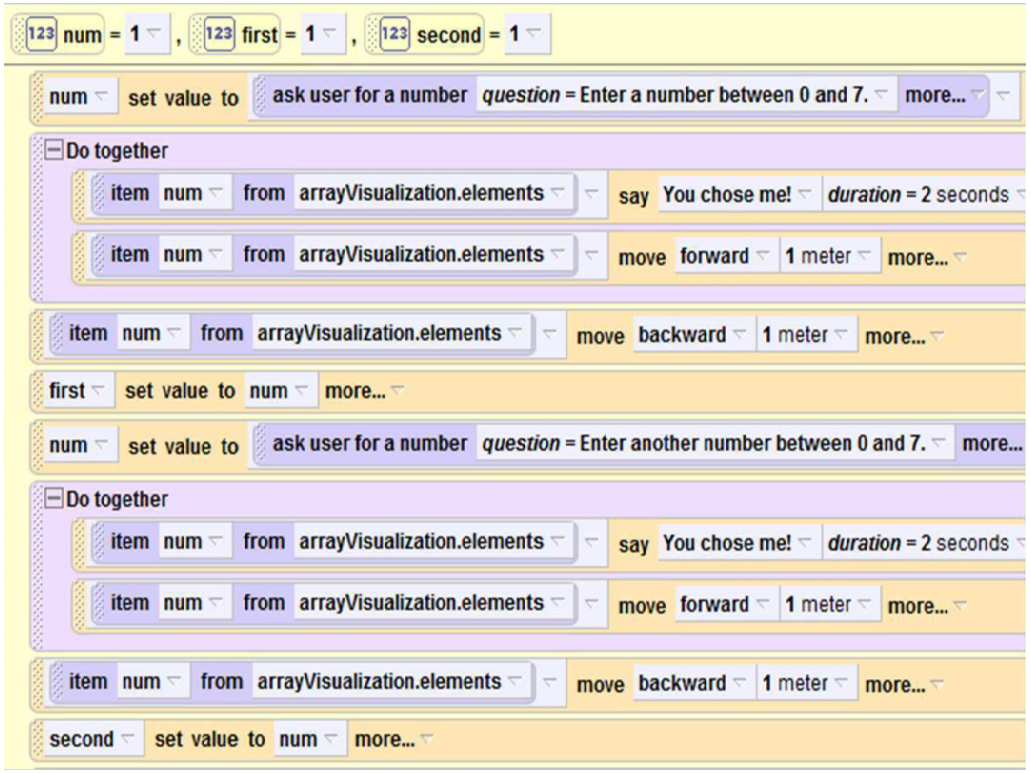


# Specific Elements

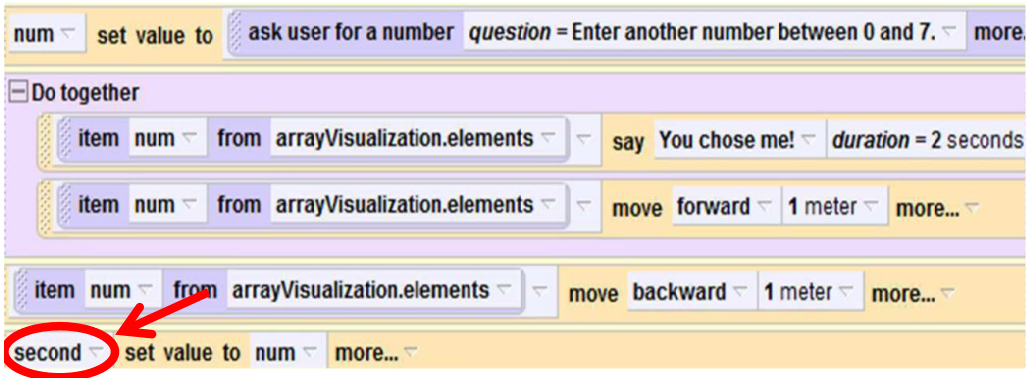
- Have the character move forward and say "You chose me" at the same time, then have them move back to their position. Now, create a new number variable called *first*. This variable will save the number that the user enters, because we will need it later on. Drag *first* into the method and set this variable to "expressions" → *num*.




# Specific Element

- Next, you are going to want to copy all of the instructions that we just added and move them all to the bottom and replace all of the <None> subjects with *num* as they are in the first half. Create a new number variable called *second* to store the second value the user will enter and put it where *first* was in the top part to save the second value. The entire method code can be seen on the next slide.



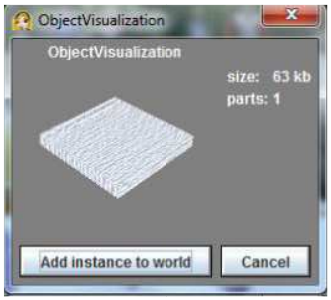

# Specific Elements

- That concludes how to access a specific element of an array. Make sure to test this method by changing the Event. Below is a picture of the pop-up box to ask the user for an index at runtime.


- *Note: Entering a number that does not exist in the array (anything less that 0 or greater than 7) will cause the program to crash and you will see an "index out of bounds exception" error.*

# Swapping Elements

- Now, we want two elements in the array to swap places. To do this, we'll need to add another visualization object. Click on Add Objects and go to Visualizations in the Local Gallery. Import an ObjectVisualization.

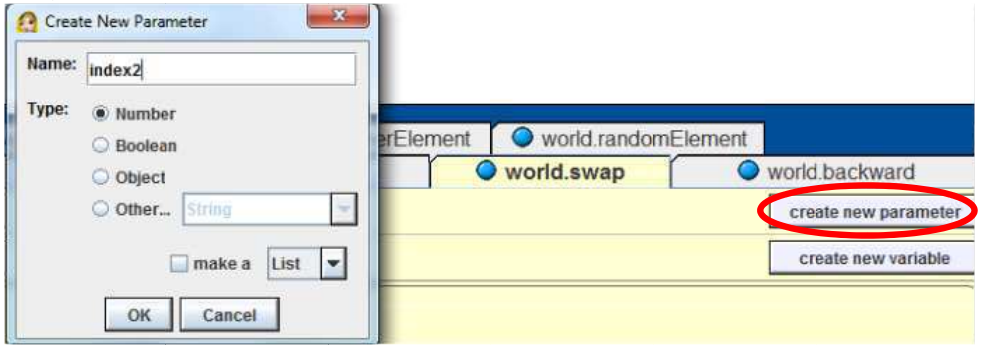



# Swapping Elements

- In this example, the positions that will be swapping places are 0 and 7.
- The algorithm for swapping elements in an array has 3 steps: The first is to move the object from the first index to the ObjectVisualization.


# Swapping Elements

- Back in the method editor, create a new world method called *swap*. In this method create two number parameters that will pass in the indices that will switch places from *specificElements*. To create a new parameter, click on the "create new parameter" button to the right of the editor. Name the first parameter *index1* and the second one *index2*. The next two slides provide an example of how we will swap two elements in the array using these parameters and the objectVisualization.

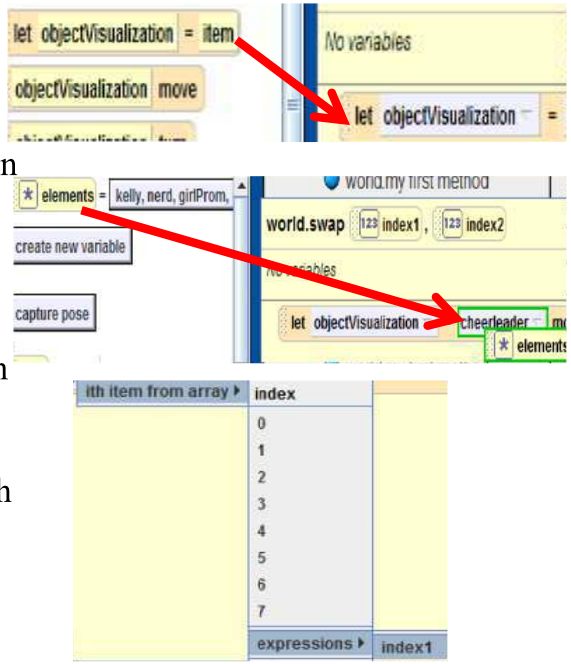



# Swapping Elements

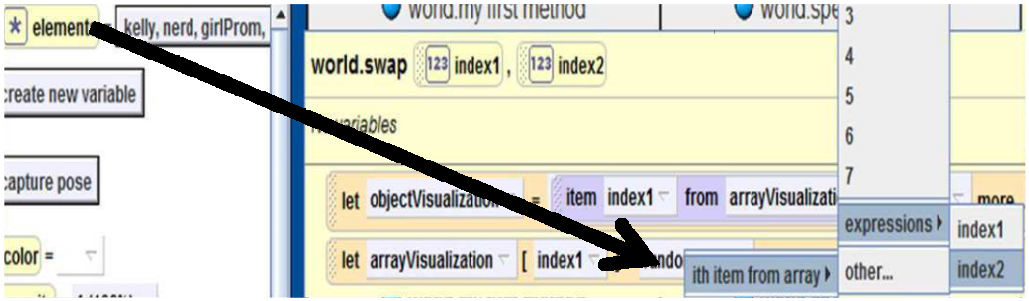
- Next, move the object at the second index of the array to the space at the first index.
- Then, move the object on the objectVisualization to the second index in the array.

# Swapping Elements

- To move the object from the first index of the array to the ObjectVisualization, go to the methods of objectVisualization and drag "let objectVisualization = item" into the editor. For now choose your "default person". Then, go to arrayVisualization properties and drag the elements of the array over the object as the item. Choose "ith item from array" → "expressions" → *index1*.




# Swapping Elements



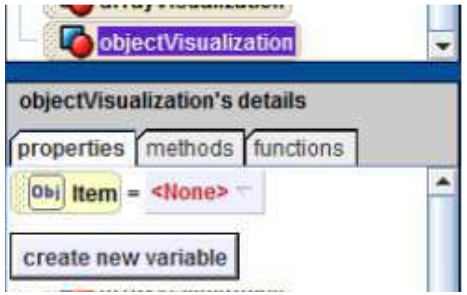

- Resulting Code:

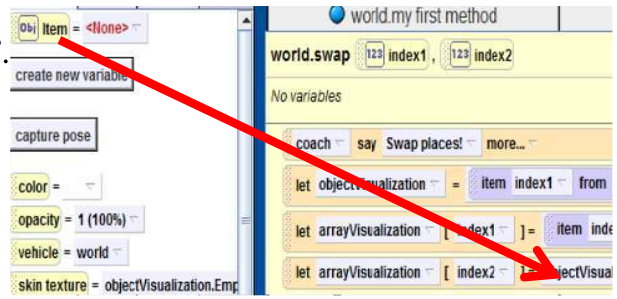

# Swapping Elements

- Next, in arrayVisualization methods, drag "let arrayVisualization [index] = item" into the editor. Choose *index1* as the index and select your "default person" again. Go to the elements of the array under and drag it over the object you used for item, selecting *index2* as the index. This can be seen on the next slide.


# Swapping Elements

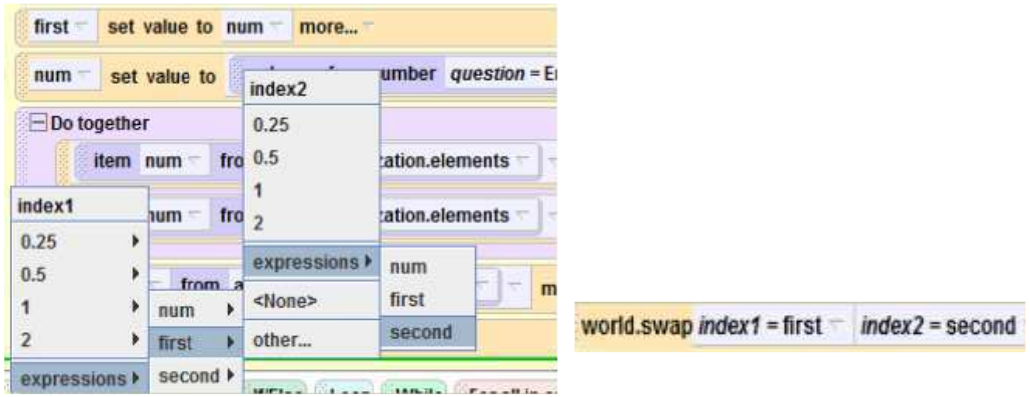
- For the last step, you will need to drag "let arrayVisualization [index] = item" into the editor. Select *index2* as the index, and the item will be replaced by the element on objectVisualization under the properties tab **OR** go to "expressions" → "objectVisualization.item".



# Swapping Elements

- Drag the *swap* method at the bottom of the *specificElements* method where the user chooses 2 elements, and have those two inputs be the indices of the objects that are switched in the array. Choose the parameters to be "expressions" → *index1*→ "expressions" → *index2*.




# Conclusion

- Arrays are very useful in programming and iterating through a group of objects. This tutorial explains how to go through the objects in an array in order, in reverse order, every other element, elements at different intervals, random elements, selected elements, and how to swap elements.

# world.my first method

- The very last thing we need to do is add all of these new methods that we created into world.my first method. They should all be listed under world's methods and you want to drag all of them in except for *swap*, which is called in *specificElements*. Change the event to run world.my first method when the world starts and this world is complete.

# Visual Lists

By Chris Brown
under Prof. Susan Rodger
Duke University
July 2012

# Set Up

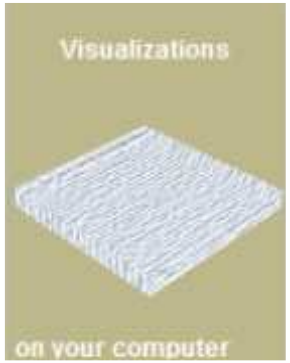
- Open any environment template and then click on the "Add Objects" button. Go into the "Animals" Folder and import 7 different animals into your world (I added the Chicken, Cow, Bunny, Penguin, Monkey, Turtle, and Husky).

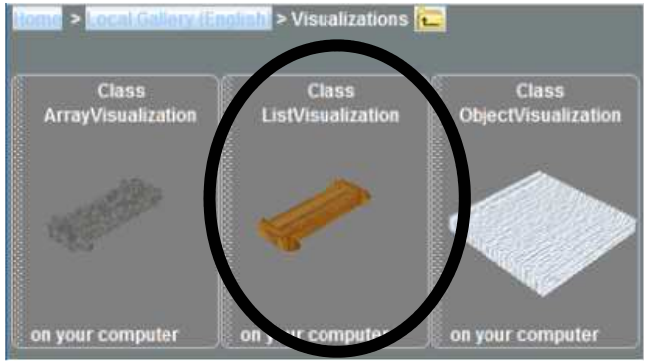

# Visual Lists

- In this tutorial, you will learn how to use visual lists in Alice with the ListVisualization object. We will make a group of characters perform actions in order and then together at the same time, and then cycle through the objects by removing and inserting characters in the list. Visual lists are the same as nonvisual lists, except that there is an actual list object that the characters stand on so that you can see their position. This makes the characters stand in a line. The list object can also be made invisible.

# Set Up

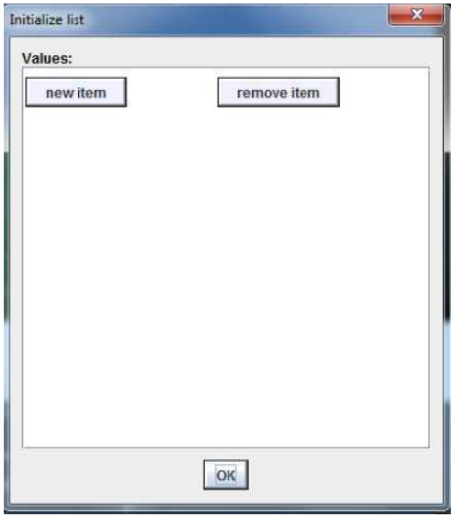
- Go back to the Local Gallery after adding your animals and find the "Visualizations" folder near the end. We want to add a ListVisualization object.



# Set Up

- After you add a ListVisualization, a window that looks like this should pop up:

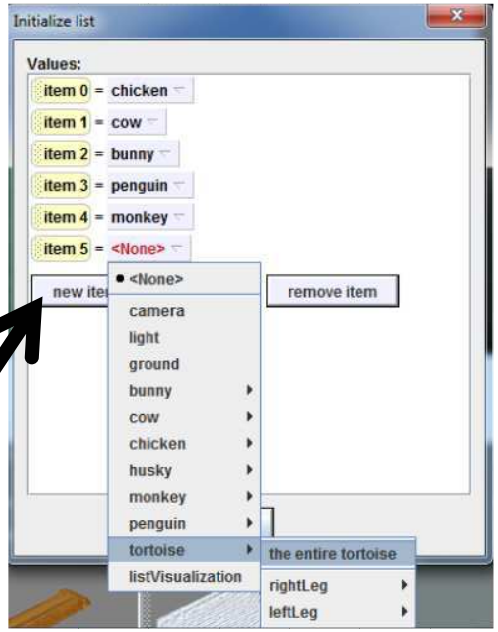



# Set Up

- Click and drag the ListVisualization object to turn it and move it around to get the entire list to fit the screen and facing the camera. Notice that the animals in the list move with it. Don't click on the animals and move them or this will cause problems accessing the objects of the list later on. Click "Undo" if you accidentally move an animal. When you finish, your world should look something like this:

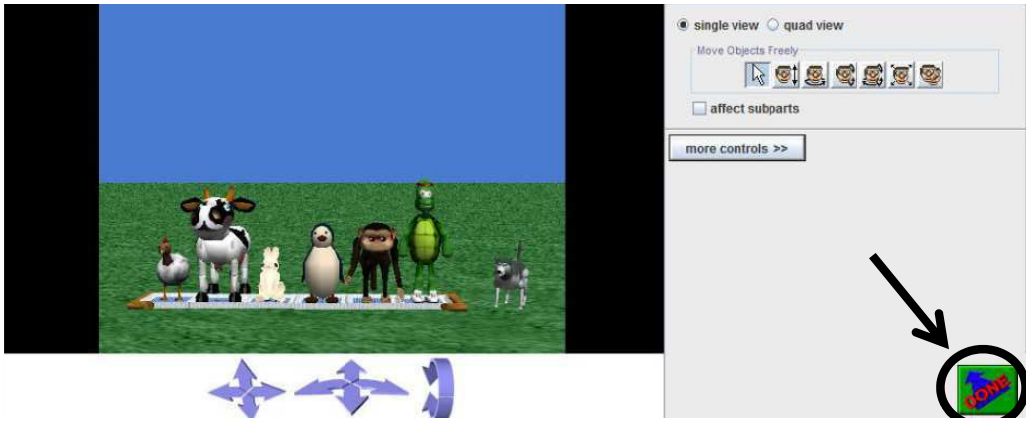
# Set Up

- Click on the "new item" button to add 6 of the 7 objects into the list (item 0 – item 5). Make sure when you add the object, you choose "the entire ____".



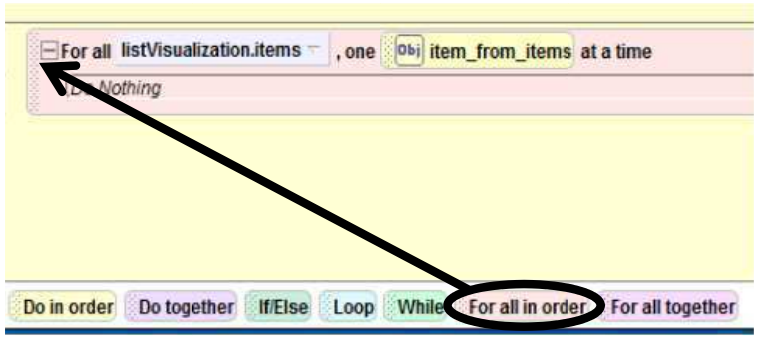

# world.my first method

- Now we are done adding the visual list and we're ready to use the visual list we created. Click the Done button to open up world.my first method.



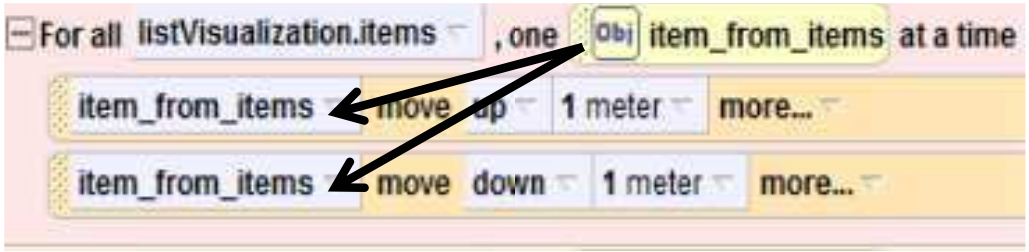
# For all in order

- First, we want each object in our list to move up 1 meter and then move back down in order. To do this, drag in a "For all in order" instruction into world.my first method and select "expressions" → "listVisualization.items".




# For all in order

- Now, drag the "item_from_items" object from the "For all in order" loop and drag it over the animal that you chose. This will make it so that each object will now move up and down in order.




# For all in order

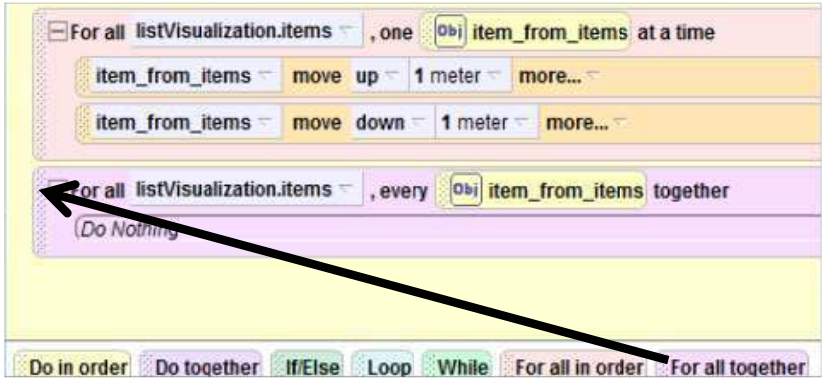
- In the object tree, select one of the animals in your list as a "default object" (I chose the chicken) to use their methods. Have the chicken move up and then move down 1 meter in the "For all in order".


# Play Your World

- If you play your world now, you will see that each animal in the list will move up and then move down. Next we will make all of the animals in the list turn at the same time.

# For all together

- To get all of the elements in a list to do an action at the same time, drag in a "For all together" from the bottom of the method editor and go to "expressions" → listVisualization.items as the list.

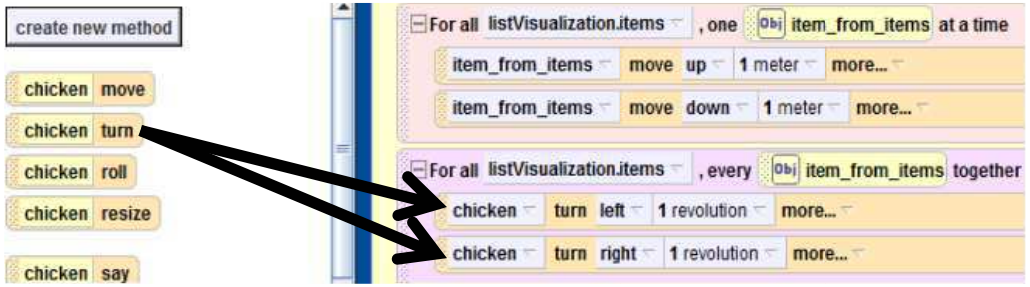



# For all together

- Replace your "default object" with the "item_from_items" variable in the "For all together" instruction.


# For all together

- Now, go to your "default object" from earlier, and have your object turn left 1 revolution and then turn right 1 revolution.

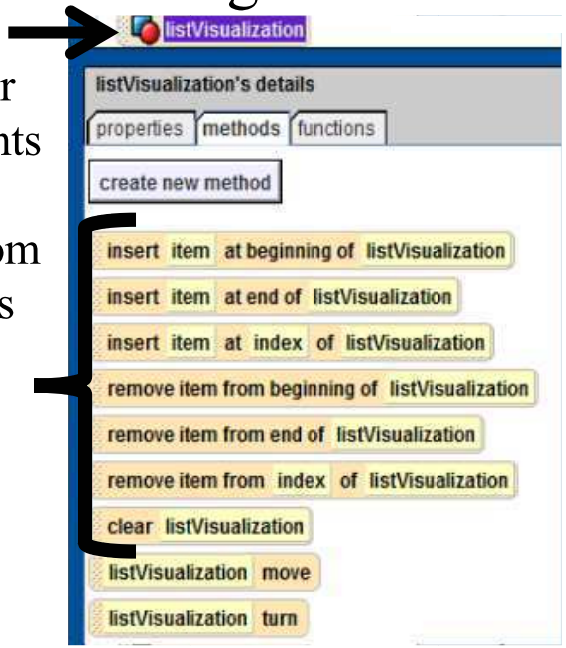



# Play Your World

- Try playing your world and you will see, after each object moves up and down in order, that they will all turn left and then turn right together.

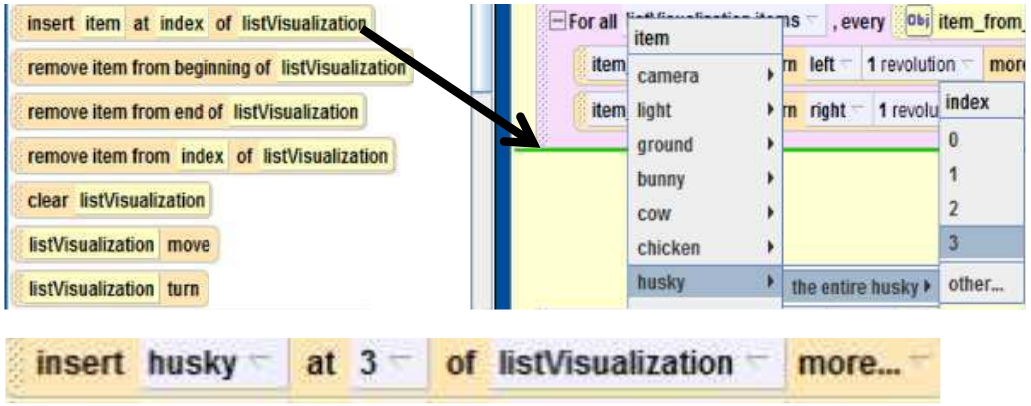
# Inserting/Removing Items

- Now we will go over how to insert elements into a list and then remove elements from a list. These methods can be found under listVisualization's methods.




# Inserting/Removing Items

- First, we want to add our last animal into the list. Drag the method "insert <item> at <index> of listVisualization" and choose your last animal as the item (Mine is the husky) and 3 as the index. This will put the husky into the middle of the list.

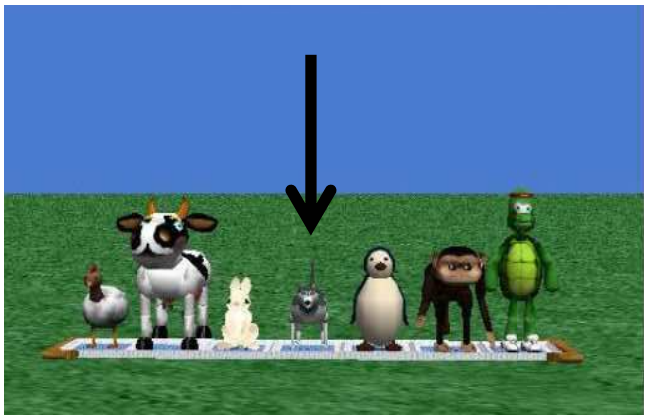



# Inserting/Removing Items

- When using these methods to insert and remove items of a list, make sure that you be careful where you remove and insert items because this can cause problems in your list. (For example, removing an object from the middle of the list will cause two items to be positioned on top of each other, even though list will be changed correctly. We reported this bug in the summer of 2012.)

# Inserting/Removing Objects

- If you run the world now, you will see that the husky moves to the middle of the list at position 3.



# Inserting/Removing Items

- Now go to world methods and create a new method called *cycle*. For all of the elements in the list, this method will remove the first element of the list, shift all of elements in the list over one space, and then add the first object at the end of the list.


# world.cycle

- Now, before we remove the first item, we want to make it invisible so that two animals will not be placed on top of each other. Go to your default object's properties and drag the "isShowing" property into the "For all in order" and set its value to false. Then, drag the "item_from_items" variable over the object's name.


# world.cycle

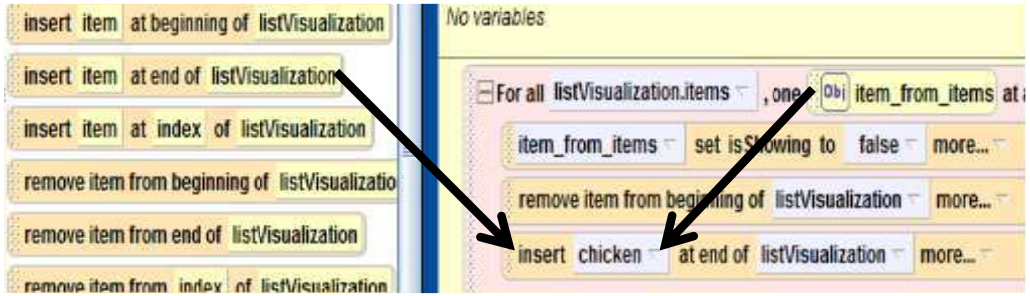
- The first thing we need to do is drag a "For all in order" into world.cycle and select listVisualization.items as the list.


# world.cycle

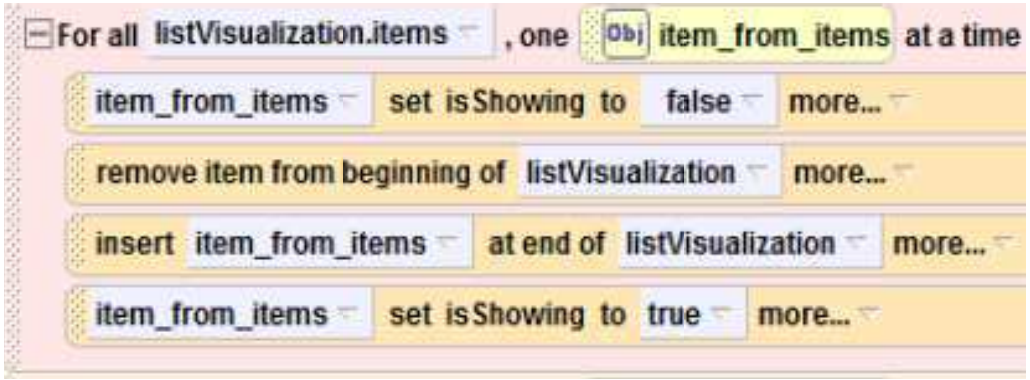
- Go to listVisualization's methods and drag in the "remove item from beginning of listVisualization" method.

# world.cycle

- Drag the method “insert <item> at end of listVisualization” into the “For all in order” and select your default object as the item. Then, replace this object with “item_from_items”.

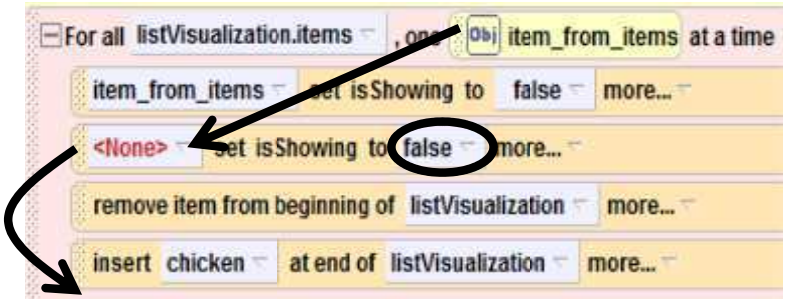



# world.cycle



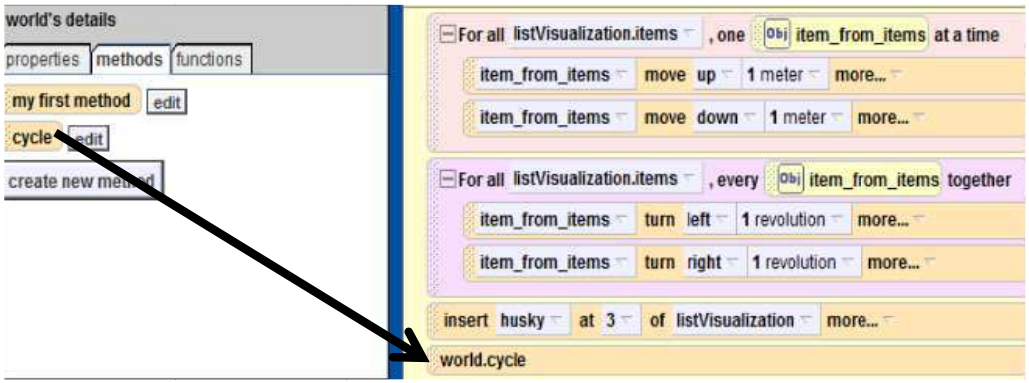

# world.cycle

- Finally, we need to make the object visible again at the end of the list. A short cut to do this is to right-click on the first method and make a copy of it. Make sure you change the “false” value to “true”, drag “item_from_items” over the subject <None>, and move this copy to the bottom of the “For all in order”. The final code can be seen on the next slide.




# world.my first method

- The last thing that we need to do is add world.cycle method into world.my first method. Click on the world.my first method tab and find the *cycle* method under world’s methods. Drag this method into the bottom of world.my first method and play your world.

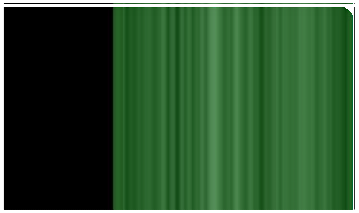


# Challenges

- That concludes the tutorial on visual lists in Alice. Try creating a new method that has all of the animals do a back-flip in order and then a front-flip all together.
- Make another method that cycles through the items in the list backward, moving the object at the end of the list to the front.

# Collections of Objects

- Visual lists are only one way to group objects in Alice. You can also use nonvisual lists, visual arrays, and nonvisual arrays. Tutorials for these can be seen on the Duke Alice Tutorials website.



# Scene Changes 2.0

This is a modification of the Scene Change tutorial
written by Deborah Nelson in June 2009
By Chris Brown
Duke University
Under the direction of
Professor Susan Rodger
July 2012



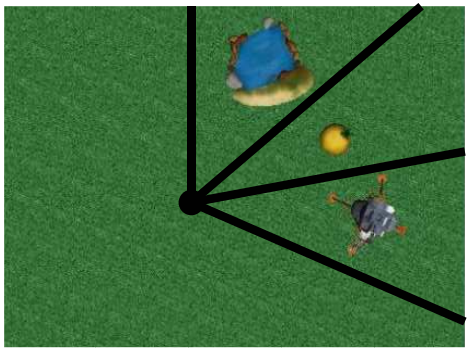
# Scene Changes

- This tutorial will show you how to create different scenes in Alice. You will make the Alice world dark, then light again into a different scene with a different ground texture and new objects. You will also learn how to move a character between scenes and do different actions.



# Scene Changes

- In the picture below, you can see how we will set up the 3 scenes. We will put in the three different environments, then rotate the camera from a fixed spot to move to each one while fading out and fading in the light. You can have anywhere up to 8 scenes in a world using this technique.





# Scene Change Note

- This tutorial is for users who have a version of Alice 2.2 later than March 2012. If you have an earlier edition, use the previous scene change tutorial on the Duke Alice tutorials website. The old tutorial tells you how to drop in an Alice class that contains the six ground textures. As of March 2012, that is no longer needed as you can easily import other ground textures. This tutorial also shows you how to use the “move to” and “orient To” methods to have a character move between scenes.

# Standards

**CSTA Standard 5.3.B-** Computer Science Concepts and Practices (CPP):

Students will be able to…

"1. Use advanced tools to create digital artifacts (e.g., web design, **animation**, video, multimedia."



# Load world

- Open a new world, with any template.
- Save it in a directory that you can find again.
- After you have opened the file go into the "Layout" mode by clicking on the green button Add Objects (toward the middle of screen).
- Overview: creating scene changes
  - Add objects.
  - Drop dummy objects at camera positions.
  - Write two methods for transition.

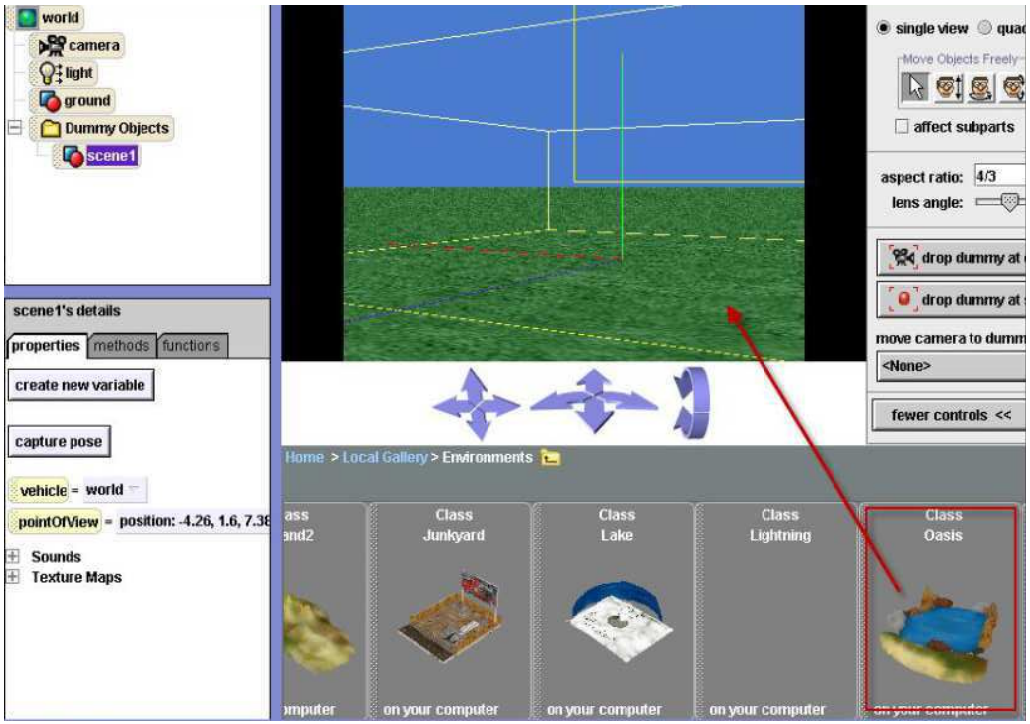


# Part One: Set up

- Click more controls. Click drop a dummy at the camera.
- In the object tree, expand the Dummy Objects folder. Rename the dummy 'scene1' (by right clicking on it and selecting rename).
- Go to the Environments folder.
- Scroll over to Oasis. Drag Oasis into the scene.

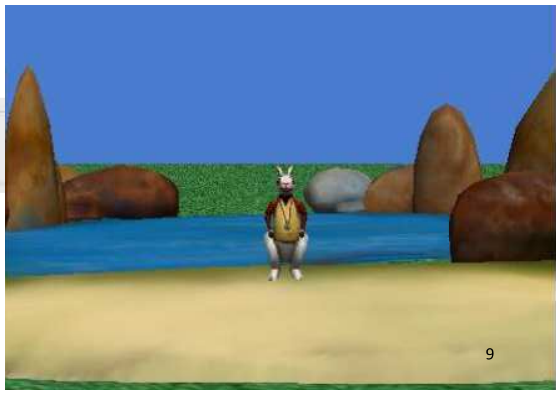
➢See the screenshot on the next slide for an illustration





# Add a Character

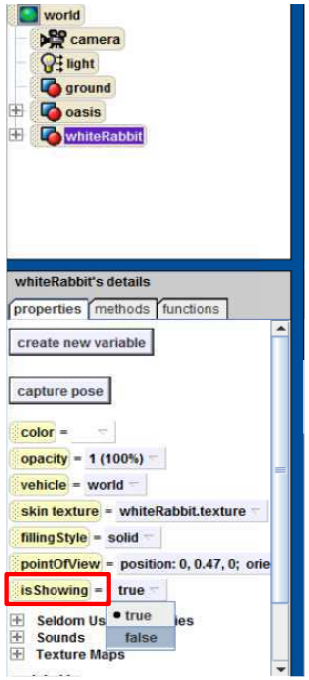
- While you are in scene 1, add a WhiteRabbit object from the Animals folder in the Local Gallery. Use the buttons in the top right corner to move the WhiteRabbit so that it is standing on the oasis and facing the camera as seen below.



# Add a Character

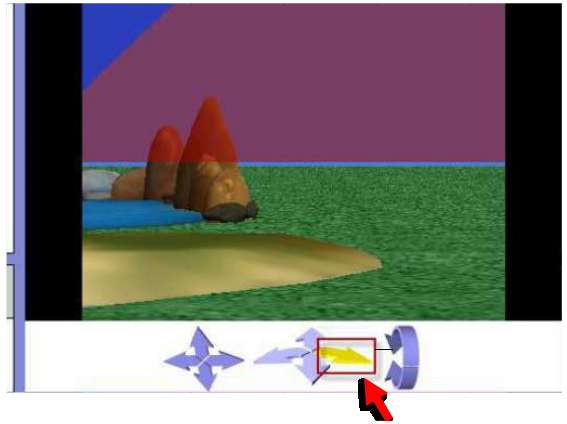
- Now, we want to set this object to be invisible. Click on whiteRabbit in the Object Tree and click on the properties tab. Find the isShowing property and set the value to false. This should make the WhiteRabbit disappear from the scene. The best way to move characters from scene to scene is to use invisible placeholder objects.






# Set up Scene 2- move camera over

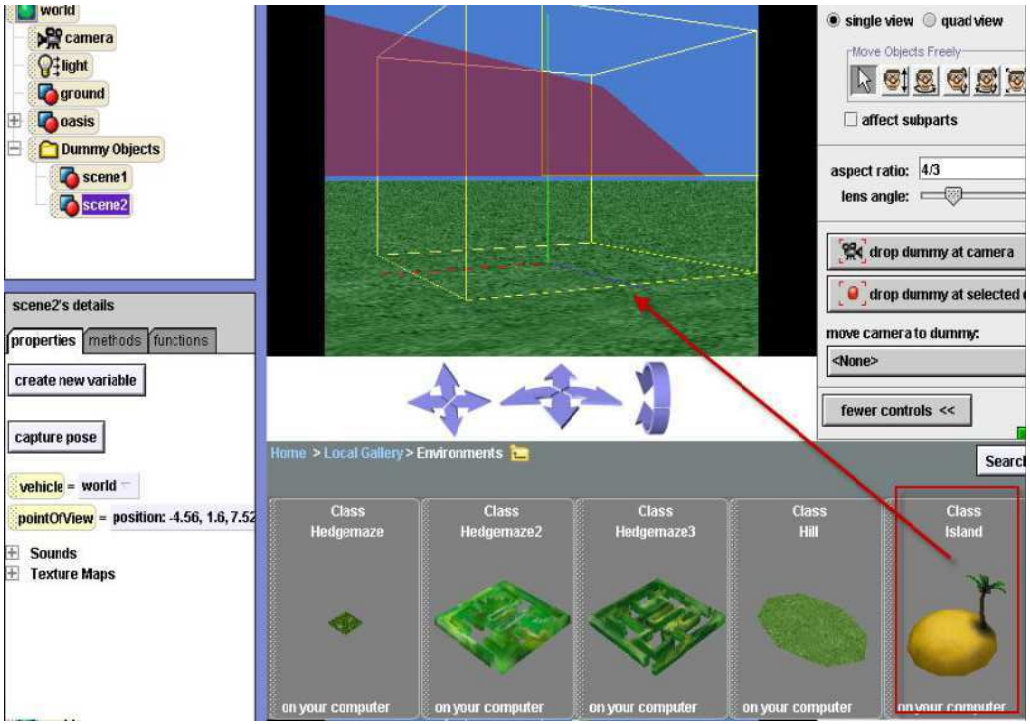
- Once the oasis is in your scene, use the camera position arrow to move the camera view, until you can no longer see the oasis.





# Drop dummy at camera

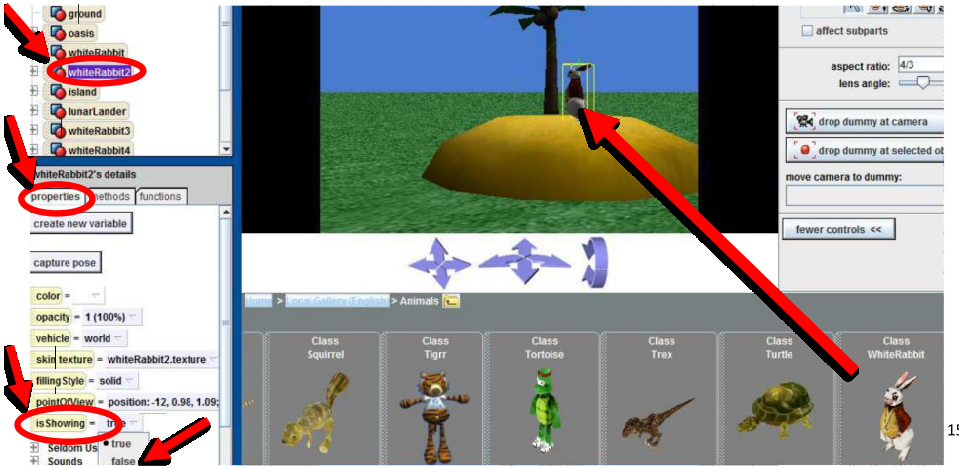
- When you can no longer see the oasis or the dummy object, drop a new dummy at the camera.
- In the object tree, rename this dummy ‘scene2’.
- In the Environments folder, scroll over to Island Drag Island into the scene.
- See the screenshot on the next slide for an illustration.






# Add a Character

- After the island is in view, import another WhiteRabbit object and position so that it is facing the palm tree.
- Click on whiteRabbit2 in the Object Tree and set its isShowing property to false. The "orient to" method will turn the character to face the same direction as the invisible whiteRabbit2.






# What to do if you can't find your object

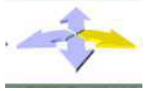
- Sometimes, when you drag an object into your world, it appears at the origin of the world, which is not in your new scene.
- Scroll down to that object - in this case island –in object tree.
- *Right click* on island and select methods, move to, camera.
- Now, drag your island into your scene. If that doesn't work, *right click* on it in the object tree again and select methods, orient to, camera.
- *Right click* on it in the object tree again and select methods, move, forward, 5 meters.

Once you have one object in your scene, you can use that as the reference to move all of the other objects into your scene.



# Set up scene 3

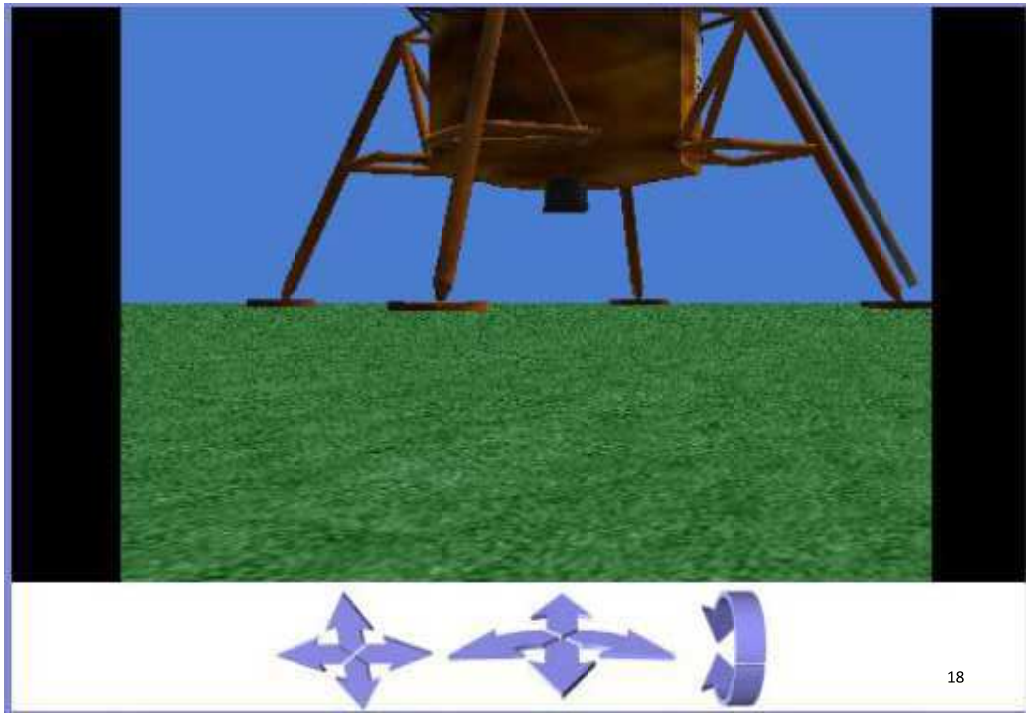
- Move the camera over (to the right) until you can no longer see the island.



- Drop a dummy at the camera.
- In the object tree, rename it "scene3".
- Go to the Space Folder.

- Drag lunarLander into the scene.
- Before you release the mouse to drop it into the scene, hold down the shift key on your keyboard. Continue to drag the object.
- If you're on a PC, you will see the yellow bound box move up because shift makes your object move up as you drag it in.
- See the screenshot in the next slide for a illustration of where my lunarLander is positioned.







# Add a Character

- Import another WhiteRabbit into scene 3 in front of the Lunar Lander and set its isShowing property to false. You should have a total of 3 whiteRabbits, one in each scene. Now go back to scene 1 and add in another WhiteRabbit character, but let this one stay visible.

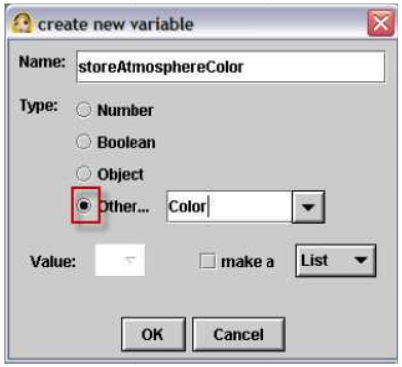


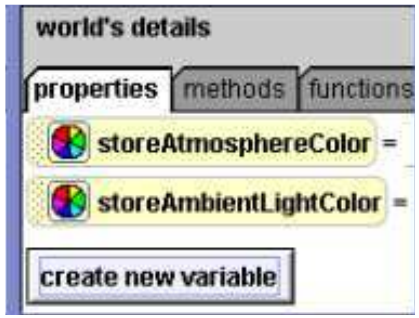
# Part Two: Writing methods

- Click on the Done button to go back to the method editor.
- In the world detail pane, click the properties create new variables.
- Name it 'storeAtmosphereColor'.
- Select type Other and select Color.
- Make another color variable and name it 'storeAmbientLightColor'.

➢ See the screenshot on the next slide for an illustration

# The color property variables



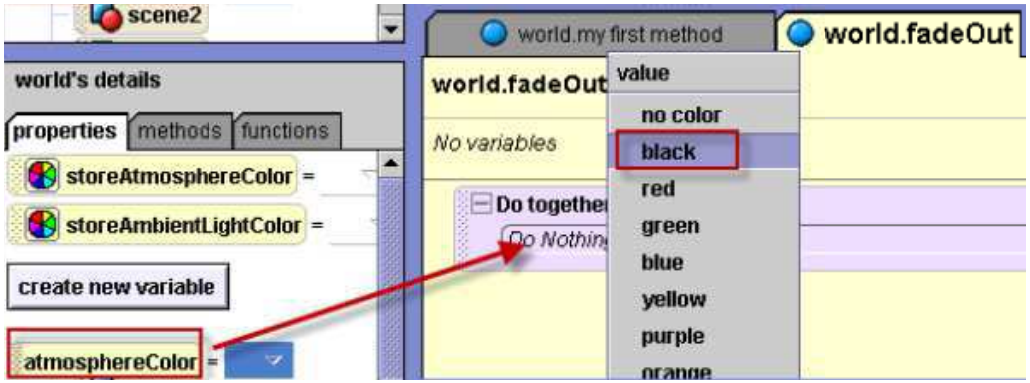



# Write fadeOut method

- Create a new method named ‘fadeOut’.
- Drag in a Do together from the bottom of the window.
- Click on the properties tab in the world details pane.
- Drag atmosphereColor into the fadeOut method. Set value to black. Note, this is not the variable we created.
- See the screenshot on the next slide for an illustration.



# world.fadeOut

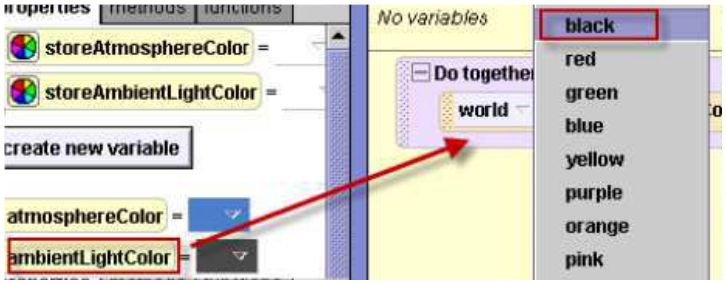



- Resulting code:




# world.fadeOut

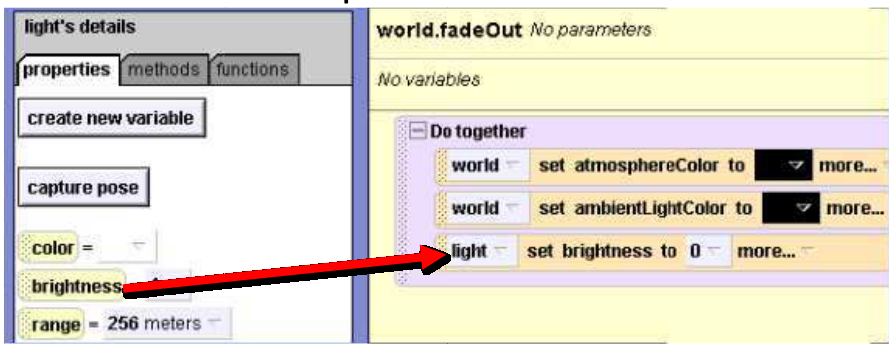
- Drag ambientLightColor into the do together. Set value to black.




- Resulting Code:

# world.fadeOut

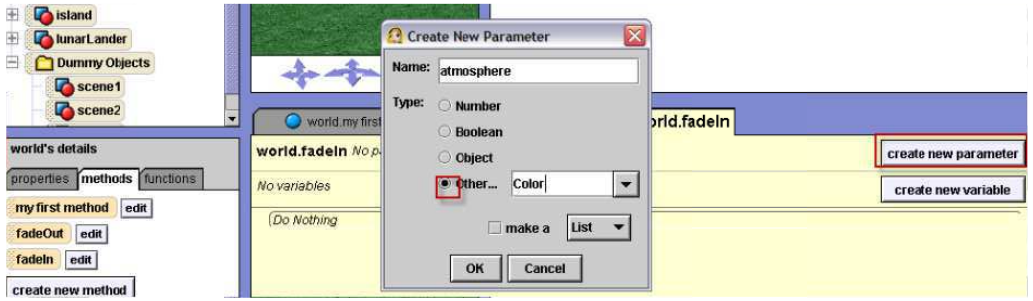
- Click on the light in the object tree. From the properties tab, drag brightness into the fadeOut method. Select Other and set value to 0.
- Here is the complete method:






# Write fadeIn method

- Click on the methods tab in the world details pane. Create new method. Name it 'fadeIn'.
- Click create new parameter in the method. Name it 'atmosphere'. Select type color.






# world.fadeOut

- To test the fadeOut method, in the events panel, change the 'myfirstmethod' to fadeOut.
- Play your world.


- The screen should fade to be completely black.




# world.fadeIn

- Drag in a Do together.
- Then, click on the properties tab in the world details pane.
- Drag "atmosphereColor" into the fadeIn method. Set value to expressions, select the parameter, atmosphere.
- Resulting code:

# world.fadeIn

- Drag “ambientLightColor” into the fadeIn method. Set value to expressions, select “storeAmbientLightColor”
- Click on light in the object tree
- In the properties tab, drag “brightness” into the method. Set value to 1.
- Resulting code:




# Write scene one method

- Click on world in the object tree and click on the methods tab. Create a new method: name it ‘scene1’.
- In the beginning of this method, drag in a Do together. Then, go to whiteRabbit4’s methods and drag “whiteRabbit4 move to- whiteRabbit” and “whiteRabbit orient to- whiteRabbit” into the Do together. This puts the visible white rabbit into the same position as the invisible placeholder and then turns it to face the same direction.



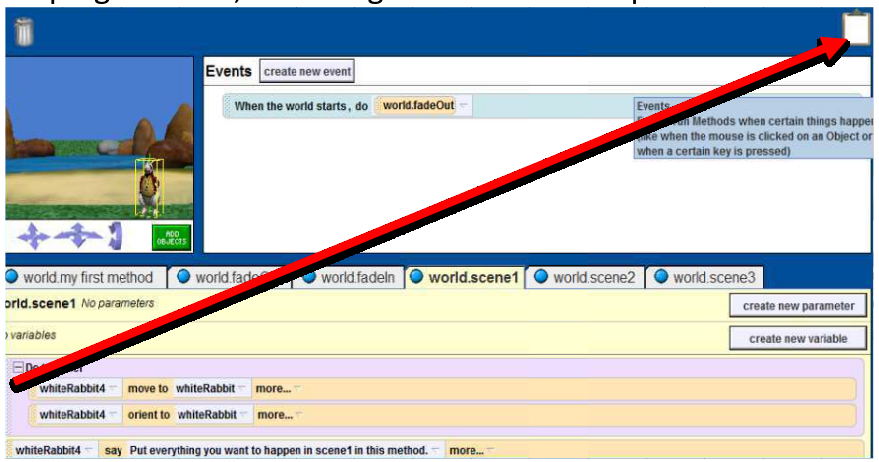

# Write scene one method

- Outside of the Do together statement, drag in a “whiteRabbit4 say- ” and enter the string “Put everything you want to happen in scene 1 in this method.” Below is the entire code for scene1.



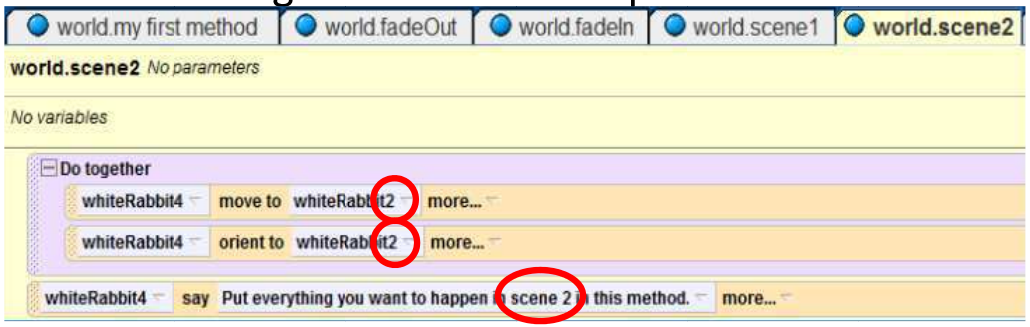

# Write scene two method

- Create a new world method: name it ‘scene2’.
- We want to do the same thing in scene 2, except we want whiteRabbit4 to move to whiteRabbit2. You can copy the instructions from scene1 by dragging them to the clipboard in the top right corner, then drag them from the clipboard into scene2.



# Write scene two method

- After pasting the code into the scene2 method, you will need to change all of the occurrences of whiteRabbit to whiteRabbit2 and change whiteRabbit4's quote.






# In world.my first method: Store the initial properties

- Click on the world.my first method tab.
- Drag in a Do together.
- Drag the color property variable we created, "storeAtmosphereColor" into the do together
- Set value to no color. Drag atmosphereColor from the pane on to the blank.




# Write scene three method

- Create new world method: name it 'scene3'.
- Repeat the steps for creating scene2 except we want to refer to whiteRabbit3 this time.




# Store the initial properties

- Drag the color variable we created, "storeAmbientLightColor" into the Do together and set value to no color for now.
- Drag "ambientLightColor" from the pane over the no color value.
- Resulting code:

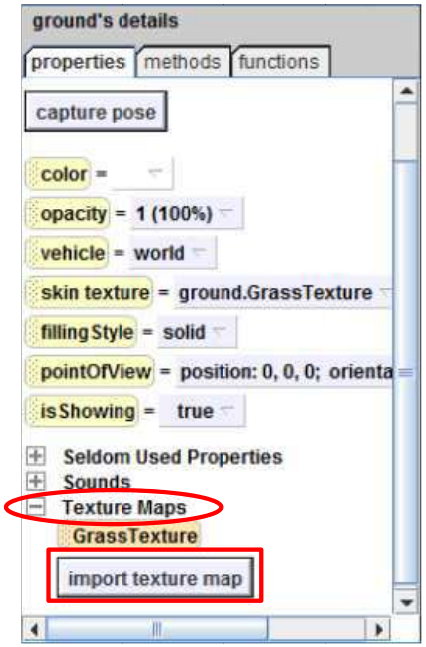
# Scene change: Camera

- Click on the camera in the object tree.
- Drag the camera "set point of view to" method into the Do together. Select Dummy Objects, select scene1.




# Changing the Ground

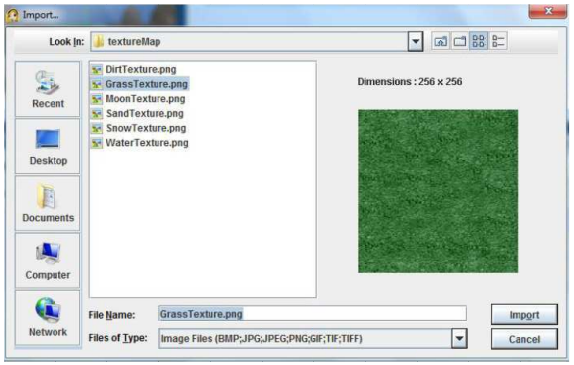
- Click on ground in the object tree.
- Go to ground's properties.
- Under properties, go to the last property and select Texture Maps. Click on the "import texture map" button.






# Changing the Ground

- After clicking on the button to import texture maps, a window should pop up with all of the different ground textures in Alice.

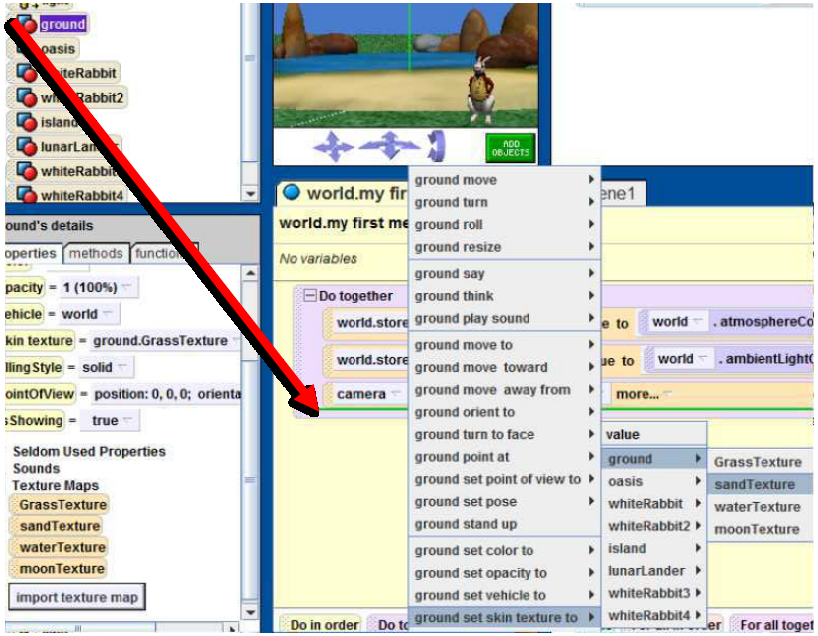



- For this world, you will need to import SandTexture, WaterTexture, and MoonTexture.



# Changing the Ground

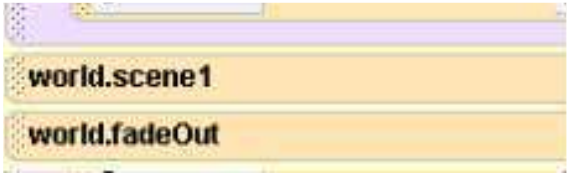
- Drag the ground item from the Object Tree and drop it into the Do together after changing the point of view of the camera. Select "ground set skin texture to" → "ground" → "sandTexture". This is a method that will change the ground texture from grass to sand at this point in the code. See the next slide for a screenshot of these instructions.




- Resulting code:




- Click on the methods tab in the world details pane
- Drag the scene1 method into world.my first method, *underneath* the do together
- Then drag in the fadeOut method

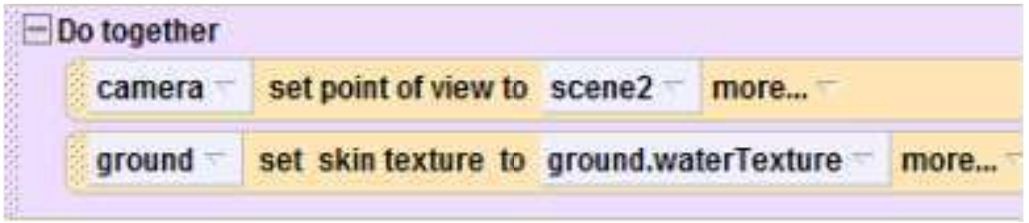



- To play you world, remember to change the event back to world.my first method. You should see your scene 1 method and then camera will fade out.




# Animation for Scene 2

- Drag in a new do together from the bottom of the window.
- In the do together:
  - Set the camera point of view to (dummy object) scene2.
  - Set the ground skin texture to waterTexture.






# Animation for Scene 2

- Underneath the do together, drag in the fadeIn method. For the parameter, select expressions, select the variable “storeAtmosphereColor”.
- Then drag in the scene2 method.


- Play your world and you should see your scene 1 code, the camera fade to black, and then the camera fade back in to your scene 2 method.

# Animation for Scene 3

- Drag in the fadeOut method underneath the Do together.
- Drag in a new Do together from the bottom of the window.
- In the do together:
  - Set the camera point of view to (dummy object) scene3.
  - Set the ground skin texture to MoonTexture.




# Animation for Scene 3

- Underneath the Do together, drag in the fadeIn method.
- For the parameter, select black (because the atmosphere is black in space).
- Drag in the scene3 method.


- Play your world and you will see all three scenes with the camera fading out and in for each one. The entire code for world.my first method can be seen on the next slide.






# Recap

- A fadeOut and fadeIn method are used for transitions.
- The camera position and ground texture are set for each scene.
- To simplify world.myFirstMethod, a separate method is written for each scene.

# Challenges

Try adding the following modifications to your world:

- Have the White Rabbit perform different actions in each scene (e.g.- spin 3 times on the oasis, turn around the palm tree on the island, and do a back-flip on the moon.
- Visit scene 1 again in between visiting scenes 2 and 3.
- Add a 4$^{th}$ scene with a different skin texture and have the white rabbit move there after scene 3 and do something.

# An Introduction to Alice (Short Version)

Edited By Chris Brown
under the direction of Professor Susan Rodger
Duke University, November 2012
This is a shorter version of the full tutorial by Jenna Hayes in June 2009

# Topics

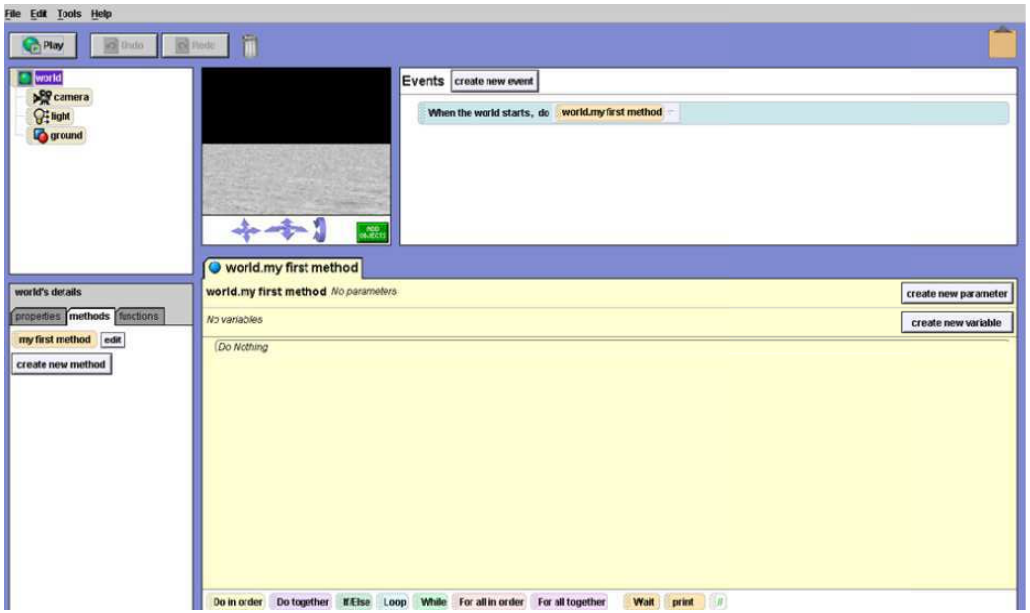
- This Alice world will tell a story about an astronaut on the moon. In this tutorial, you will learn how to add objects, move and adjust objects, use methods, create new methods, create events, and use the "vehicle" property.

- This tutorial takes 25-30 minutes. Topics that are not covered in this version but are in the complete version are creating dummy camera views, moving the camera, using Do Together in a method, and turning "as seen by" another object.


# Starting Off

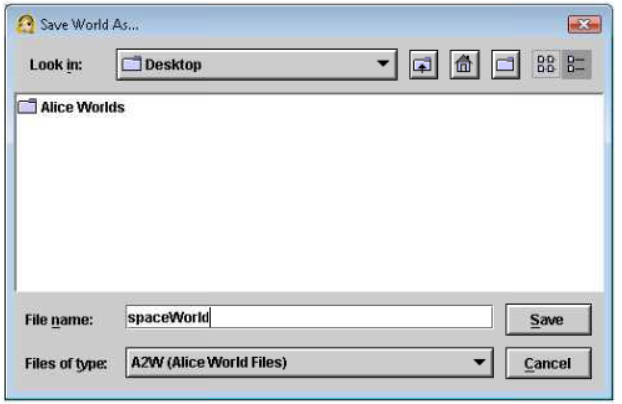
•Our first step is to choose a background.
•When you open Alice, a box will pop up that has six different choices of background. It looks like the box to the right.
•Select the space background, because our world will be in space.
•Click on space and then click Open.

After you click Open, your screen will look like this:




# Saving your world

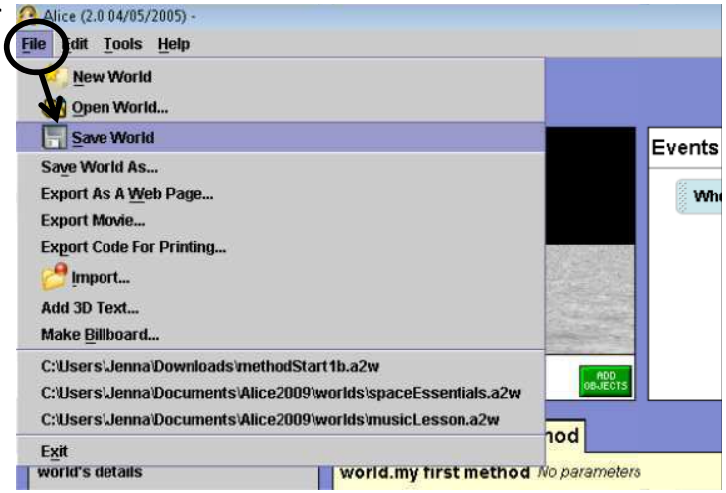
•In the box that pops up, name your world spaceWorld, and save it in a place that you will be able to find again, such as in a folder on your Desktop.




# Saving your world

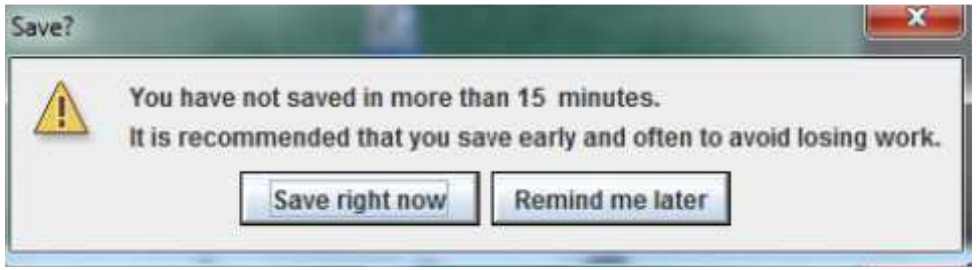
•Before we do anything else, let's save our world. You should also always do this before you close out of Alice.

•Click on File at the top left-hand corner of your screen, and then click on Save World.




# Saving your world

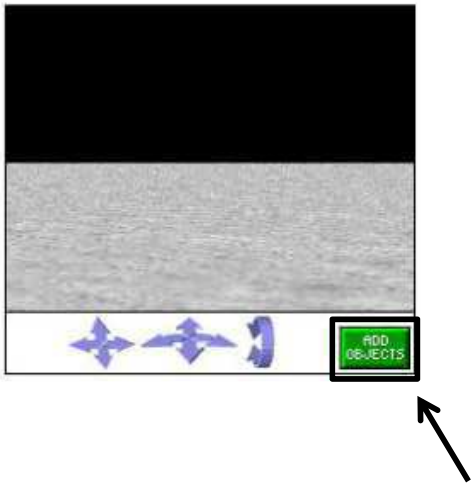
•Also, while you're working on your Alice world, this box will pop up about every 15 minutes.




•You should always click Save right now. This way, if Alice crashes, or if your computer crashes, you will have backups of your world and will not lose all of your work!

# Adding objects to your world

•Now, we will add some objects to the world.
•Just below the picture of your empty space world, there is a small green button that says Add Objects.
•Click on this button.




# Adding objects to your world

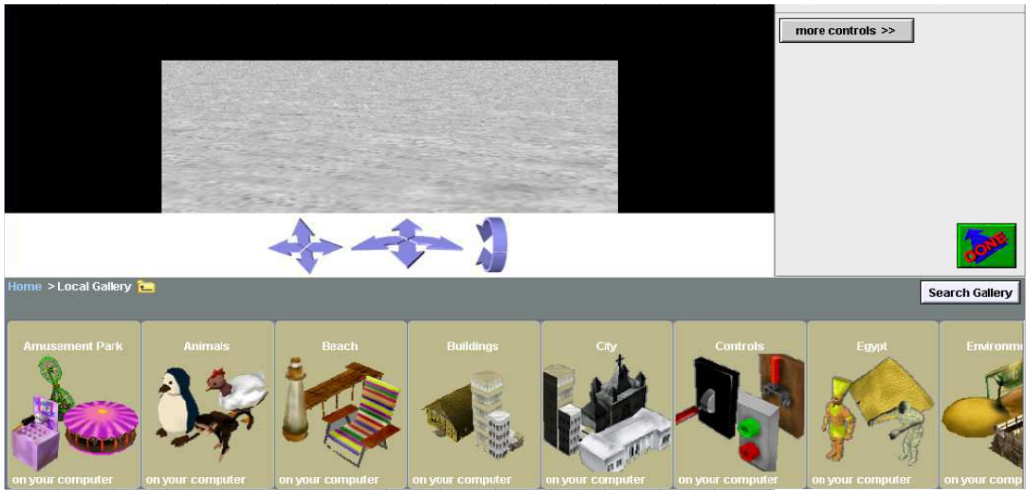
•Scroll to the right until you see the Vehicles folder, and click on it.



•Scroll to the right again until you see the Humvee.
•Click on the Humvee.
•On the box that pops up, click Add instance to world.
•The humvee will appear in the center of the space screen.

# Adding objects to your world

A new screen will appear, on which there is a large selection of objects below the space screen that you can add into your world. This is called the Local Gallery. Each folder of objects in the gallery has a different theme.




# Adding objects to your world

The humvee takes up most of your screen, but we will re-size it later. First, let's add another object to your world.


•Click on Local Gallery above the pictures of objects to go back to the gallery starting screen.

# Adding objects to your world

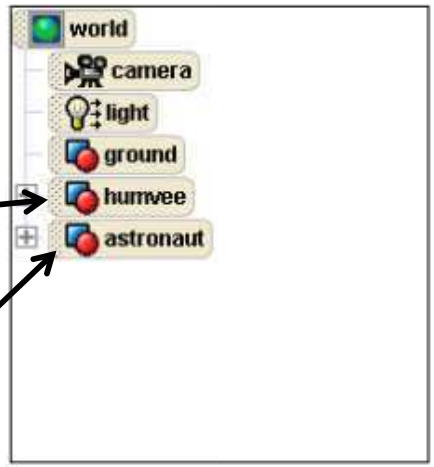
•Next, scroll to the right until you see the Space folder.
•Click on this folder.
•Click on the Astronaut.



•Click Add instance to world on the box that pops up.
•The astronaut will be added to your world, but you won't be able to see him/her yet.

# The Object Tree

•When you add objects to your world, they will appear in a list on the left of your screen, called the Object Tree.
•The humvee that you added will be on the object tree.
•Even though you can't see the astronaut yet, his/her name will also appear in the object tree. That way you know that he/she is actually there.




# Adding objects to your world

Your space world will look the same after adding the astronaut. This is because he/she is being hidden by the humvee!

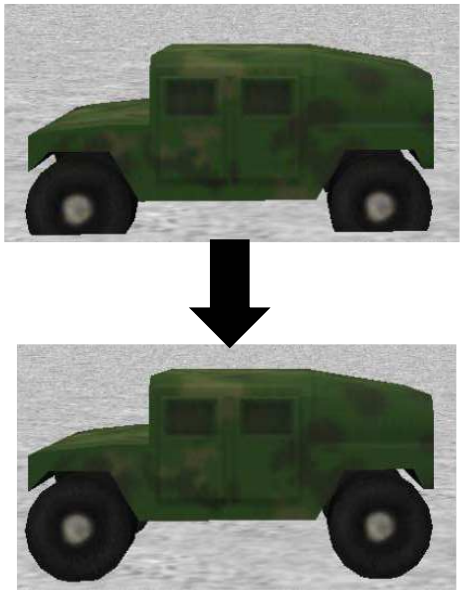
# Positioning the objects

•Look at the right side of your screen.
•There is a group of buttons with faces on them.
•These buttons are used to position objects.
•The first thing we will do is make the humvee smaller. Click on the resize button, which is the one with the four arrows coming out of the face.


•Click on the humvee, and hold down your mouse. Move your mouse around, and the humvee will get bigger and smaller!
•Downsize the humvee until you can see the astronaut's feet.

# Positioning the objects

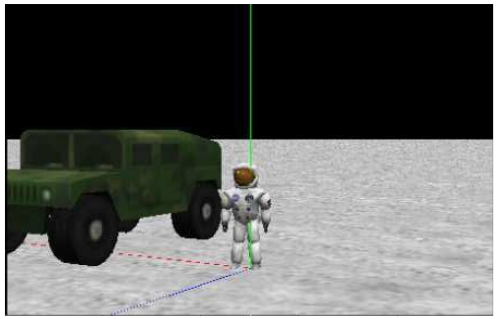
•This button will move your objects up and down.
•Click on this button, and then move the humvee up and down. Position it so that its wheels are directly on the ground.
•Here's a hint: Move it down so that its wheels disappear into the ground, and then slowly move it back up. You may have to use the white arrow button again to move the humvee back if it starts to disappear off of the screen. The second you see all of its wheels appear out of the ground, you know it is directly on the ground.

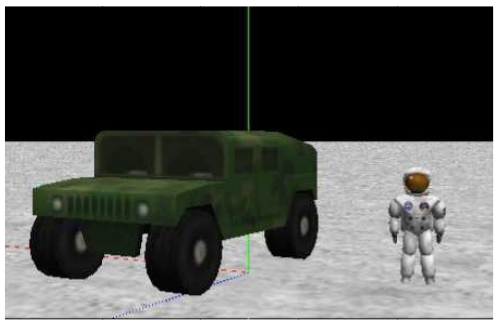


Try doing the same thing with the astronaut!

# Positioning the objects

•Now, click on the button with the white arrow on it, as pictured above. Click on the humvee and move it to the left of the astronaut.
•Then, click on the astronaut and move him/her to the right.
•Move the humvee to the right so that it is completely on the screen. Your screen should look something like this:





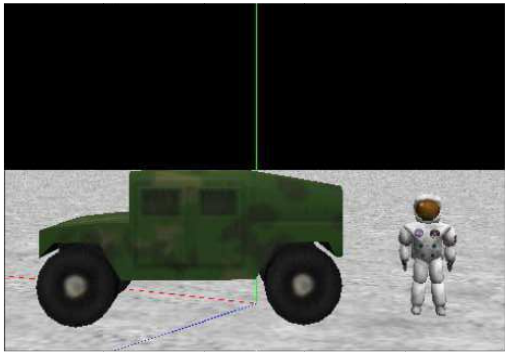
# The Undo button is your friend!

•What if you make a mistake, like accidentally clicking on the ground and moving it?
•You can click on the Undo button above the object tree to undo the last thing you did.
•Use this button whenever you mess up, or want to get rid of something you just did.

# Positioning the objects

•This button is used to spin your objects around. →

•Try spinning your humvee so that it is parallel to the screen.
•Your screen should look something like this: →

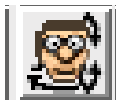


Now that we are done positioning the objects, we can start to animate the characters in the world!

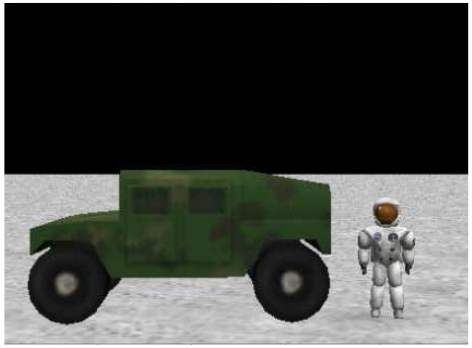
# Positioning the objects

•This button will turn your character backwards or forwards. →

•This button will turn and rotate your object in pretty much any direction. →

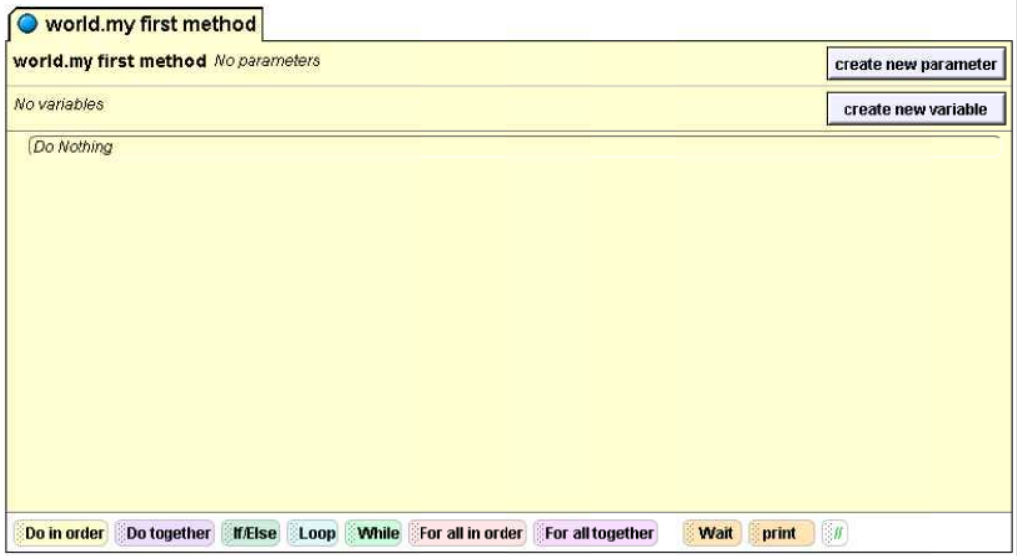


•If you want to, try these buttons out on your objects. When you're finished, click Undo until your screen looks like this again. →



# Methods

•The large tan rectangle in the center of your screen is called the Method Editor. Right now, it is blank.



# Methods

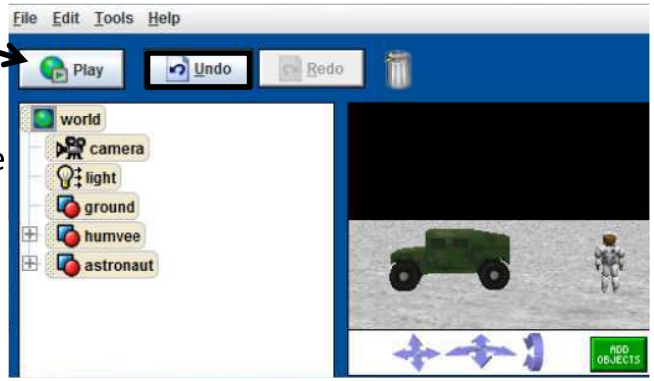
•The method editor is where you can make your characters do things.

•Your characters already know how to do certain things.

•These are some of the things that your astronaut already knows how to do. To find this list, click on astronaut in the object tree. Then look below the object tree at the box that says astronaut's details, and click on the methods tab. This list will appear.


# Methods

•Now press the Play button in the upper left-hand corner of the screen to see what these methods will look like in your world.

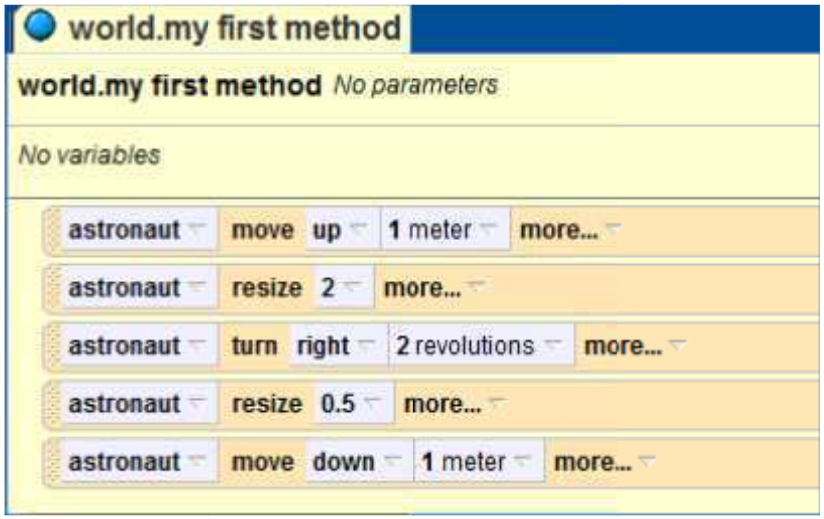



•Challenge: What code should we add to return the astronaut to his original size and position?

# Methods

•To tell your astronaut to do something, click on one of these methods, hold down your mouse, and drag and drop it into your method editor. Try dragging a few of them to see what they look like. For most of them, such as move, you will have to select a direction or a distance when you drop it. These are called *parameters* for the methods.

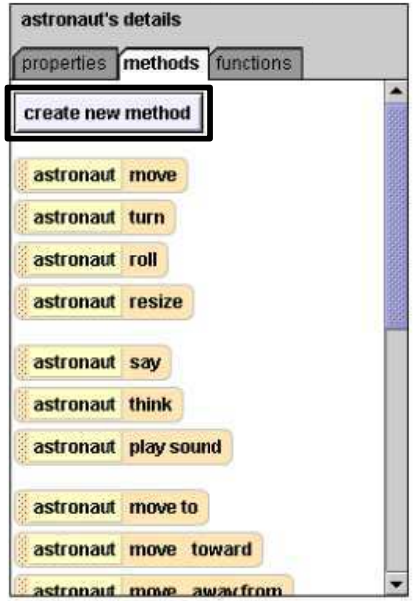

# world.my first method



# Methods

•To teach your astronaut new things, you can combine these methods that he/she already knows into new methods.

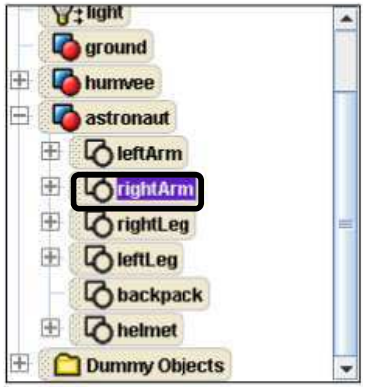
•Let's try creating a new method. We will create a method that makes the astronaut wave. Make sure you have clicked on astronaut in the object tree. Then, go to the methods for the astronaut and click create new method.




# astronaut.wave

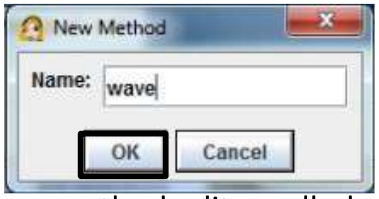
•In your object tree, click on the + sign next to astronaut. This will show you the astronaut's parts.

•Click on rightArm in the object tree so that you can get a list of the rightArm's methods. We will use these methods to teach the arm to wave.

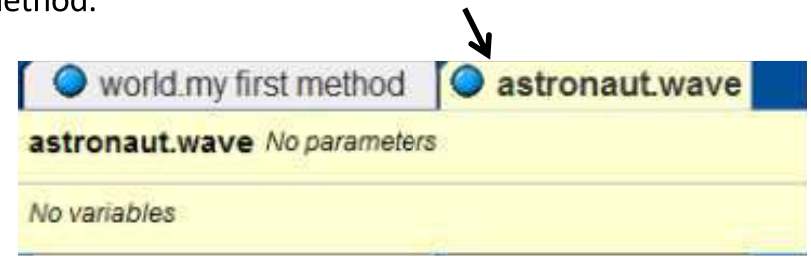



# astronaut.wave

•In the box that pops up, type wave, then click OK.



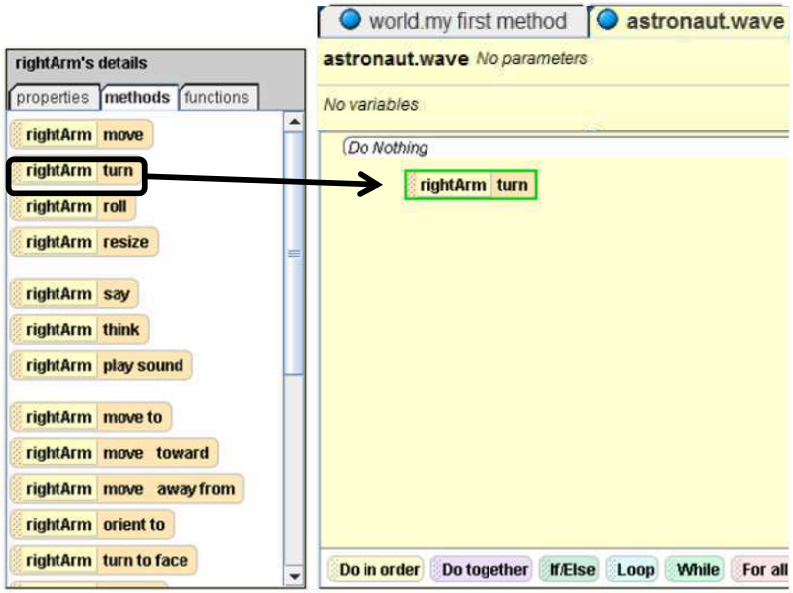

•You should see a new tab appear in your method editor called astronaut.wave. This is the space where you will create the Wave method.




# astronaut.wave

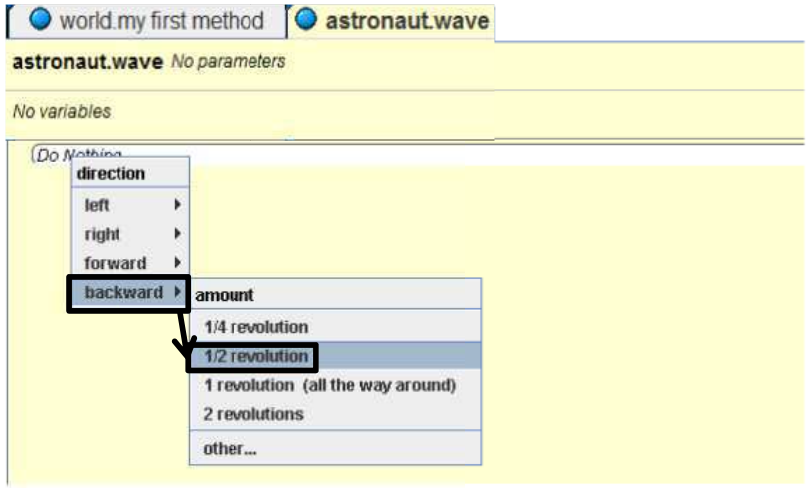
•Look back at the rightArm's list of methods and find rightArm turn.

•Click on this method and hold your mouse down, and drag it over to the method editor. Then release your mouse to drop it there.



# astronaut.wave

•A small gray menu of directions will appear. In this menu, select backward. Another menu will appear, this time of how many revolutions you want the arm to turn. Select ½ revolution.



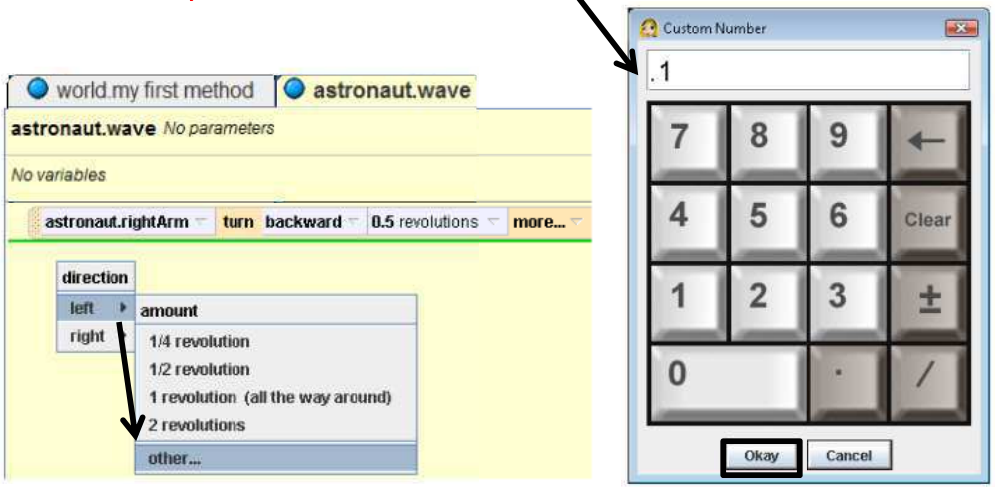

# Events

•Now that we have written part of a method, we want to figure out how to see it in action. When you press the Play button in the upper left-hand corner of your screen, your world will use an event that can show you your methods. An event is a way to call methods when your world is played.


•The event editor is found in the top right-hand corner of your screen.

# astronaut.wave

•Now, click on the rightArm roll method. Drag and drop it into astronaut.wave. For the direction, select left, and for amount, select other…. A calculator will appear. Type .1 into this calculator, and then click Okay.

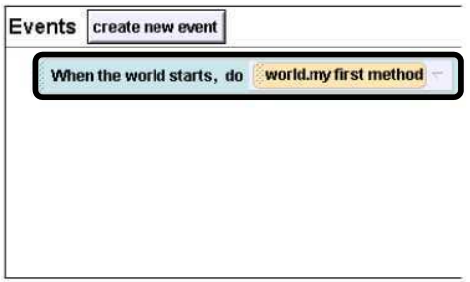


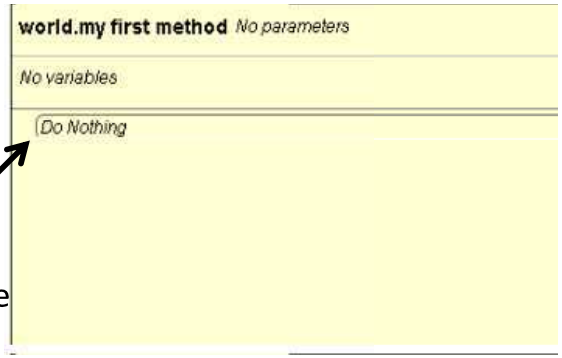

# Events

•There is one event in your event editor already. It says When the world starts do world.my first method. This tells your world what to do when you press Play.

•This means that when you press Play and your world starts, whatever methods you have in the world.my first method tab are carried out in your world.

•But if you click on your world.my first method tab in your method editor, you will see that it is empty!



# Events

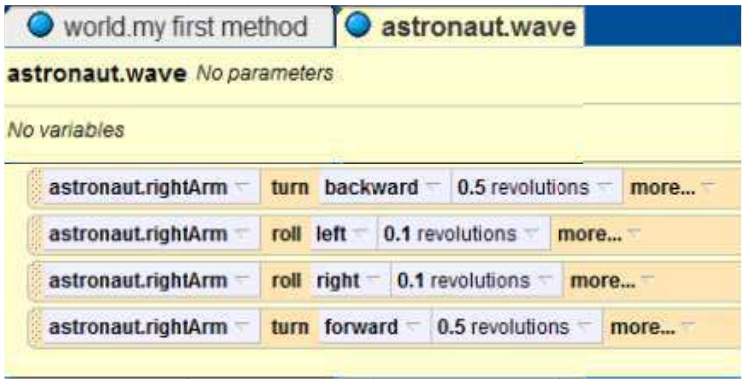
•This means that when you press Play, nothing will happen in your world. Try pressing Play to see that this is true.


•So how do we make astronaut.wave happen in our world?

•We could try changing the event that is already there to astronaut.wave. To do this, click on the down arrow next to world.my first method in the event editor, and then choose astronaut, and then Wave.


# astronaut.wave

•Now we need one more line of code, that tells the astronaut to put his arm down. Drag and drop a rightArm turn method. Select forward, and ½ revolution. This will be the final code for your method.




•Play your world one more time to test out the complete astronaut.wave.

# astronaut.wave

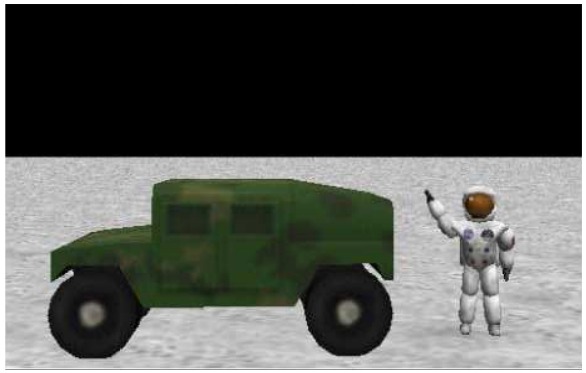
•Now press Play to see what astronaut.wave looks like so far.

•Let's add more to the method. Drag and drop another rightArm roll, and select right, and then other…. Type in .1.


•Play your world again to test astronaut.wave.

# Events

•We can make spaceWorld even more interesting by creating a new event that adds interaction in our world.

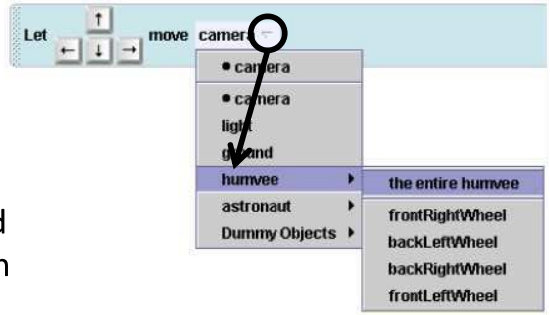


•First, let's change the event in the event editor back to my first method.

# Events

•Now we are going to make an event. The event will allow you to control the humvee with the arrow keys when you play your world.


•Click on create new event in the event editor. Then click on Let the arrow keys move <subject>.



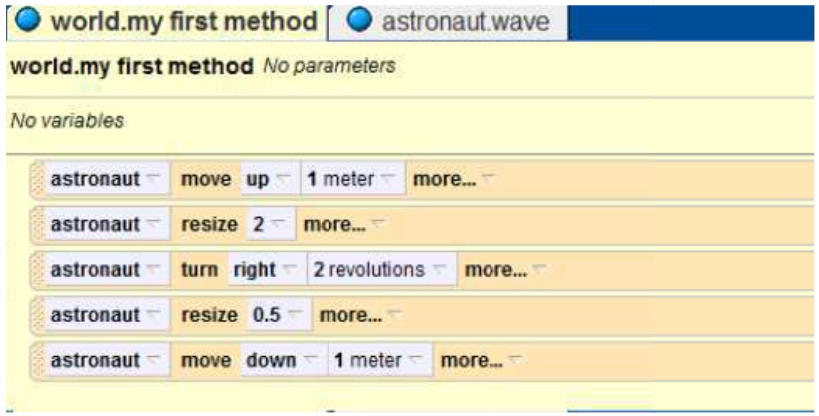

•Change the event from camera to humvee by clicking on the down arrow next to camera, and then selecting humvee, and then the entire humvee.

# Pulling it all together

•Now it's time to pull this all together!

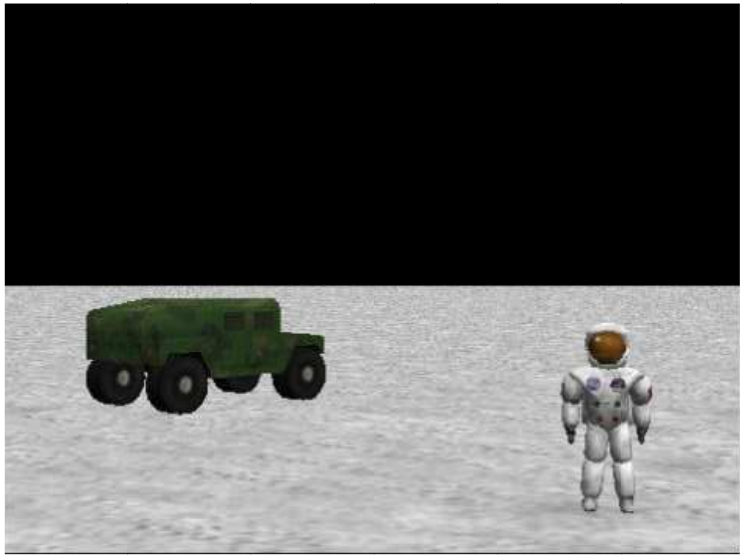
•In your method editor, click on the tab at the top that says world.my first method.

•This tab should have the very first instructions we added at the beginning, but now we are going to use it as a place where we bring all of the methods that we have written so far together.




# Events

•Play your world, and test out this new event by pressing the arrow keys and seeing what happens to the humvee.



# Putting it all together

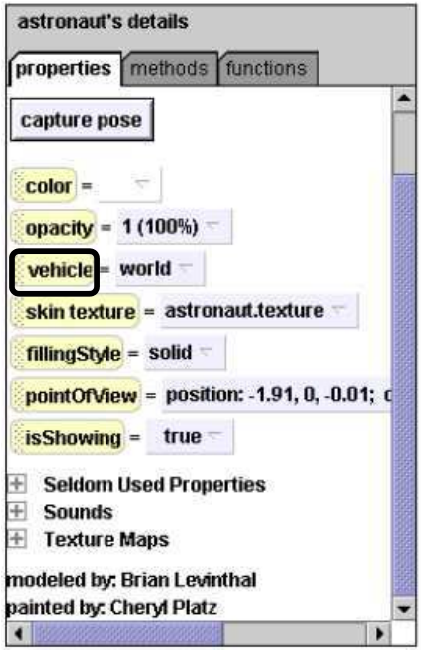
- First, add astronaut.wave into world.my first method by dragging it into the method. Now, make the humvee roll left 1 revolution. Then, have the astronaut move to the the entire humvee so that it will eventually be able to ride it when you drive. Your code should look like this so far:

# Drive Humvee

•Now we need a way to glue the astronaut to the humvee so that when the humvee moves, the astronaut will move with it.

•We can do this using the vehicle property.

•To find vehicle, click on the astronaut's properties tab, and find the button that says vehicle.

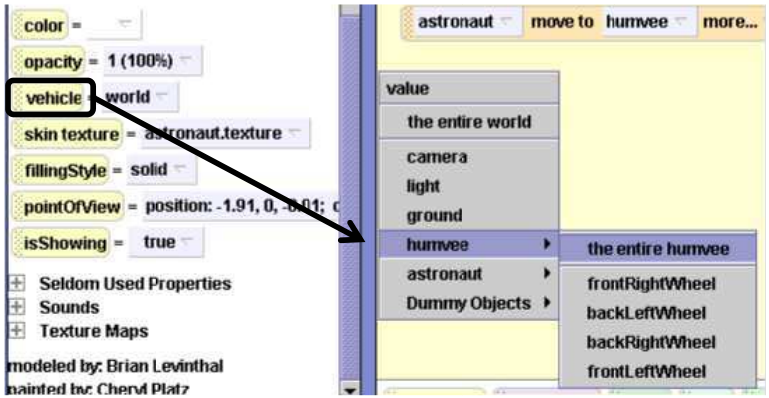



# Pulling it all together

•Now, drag an astronaut say method at the bottom of your code, and click on other…. Type in, "Use the arrow keys to drive me around!". Change the duration on the command to make the speech stay on the screen longer. To do this, click on more… on the astronaut say line of code, then choose duration, other…, and then type in 4. Your final code will look like this:


# Drive Humvee

•Click on the vehicle button and drag and drop it into your method.
•On the gray menu that drops down, select humvee, and then the entire humvee.
•This will set the humvee as a vehicle to your astronaut. When the humvee moves, the astronaut will go with it.

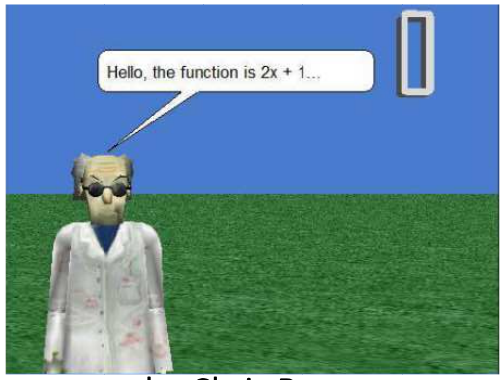



# Pulling it all together

•Now test your world by pressing play. When your methods are done playing out, try steering the humvee around with the arrow keys.


(http://www.cs.duke.edu/csed/alice/aliceInSchools/)

# Nonvisual Arrays




by Chris Brown
under Prof. Susan Rodger
Duke University
June 2012

# Standards

**CSTA Standard 5.3.B-** Computer Science Concepts and Practices (CT): Students will be able to…"6. Compare and contrast simple data structures and their uses (e.g., arrays and lists)."

**NC Standard Course of Study Mathematics Grade 6-**

**Goal 5:** The learner will demonstrate an understanding of simple algebraic expressions.

**Objective 5.0.2:** Use and evaluate algebraic expressions.

# Nonvisual Arrays

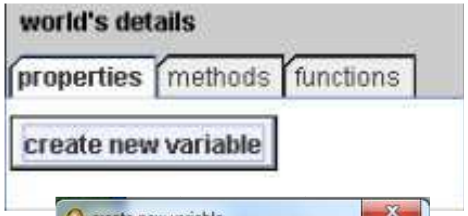
- This tutorial will display how to create and use nonvisual arrays in Alice. Nonvisual arrays can be an array of any object or data type that don't necessarily have to be in order in the world. In this world, we will fill the array with solutions to the equation 2x + 1 and have the user calculate the answers for random integer values of x.

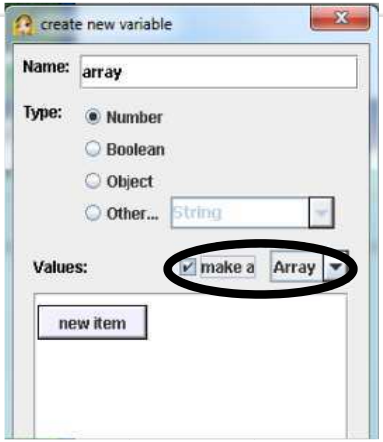
# Set Up

- Click on the "Add Objects" button. For this world, all you will need to add is the MadScientist character under the "People Heading" in the Local Gallery and a 3D text object to keep score. Right now, set the text to be 0.

# Nonvisual Array

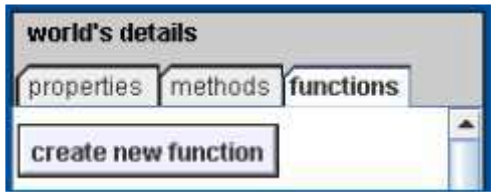
- Now we're ready to start building the array. Make sure that world is selected in the object tree, and then go to properties under world's details. Click the "create a new variable" button and call this variable *array*. Make sure that it is a number variable and that you check the box for "make a" and select Array. Leave the array empty for now.

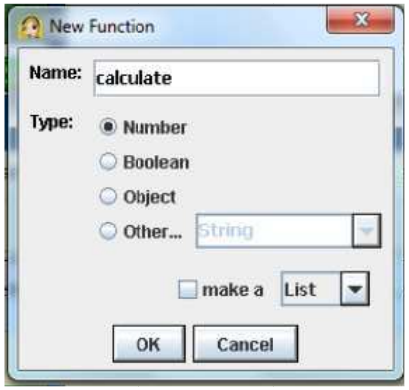



# Function

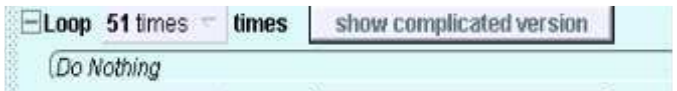
- In the world's details pane, go under "functions" and create a new function that returns a number. Name this function *calculate*. In here, we will calculate the values to be put in the array.



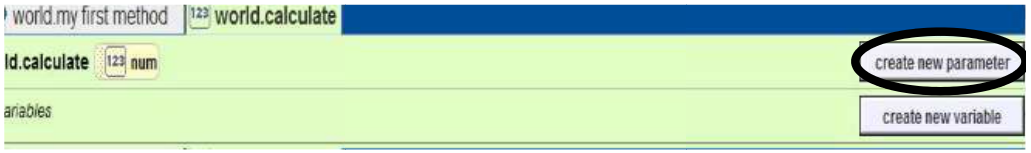




# Loop


- Next, we'll see how to fill in the array. Suppose we want an array with all of the solutions to the equation "2x + 1" from 0 to 50. Add a loop from the bottom of the editor into the method and enter 51 as the end value [0, 50] by selecting "other…" in the menu and typing 51 into the calculator that pops up.




# Function

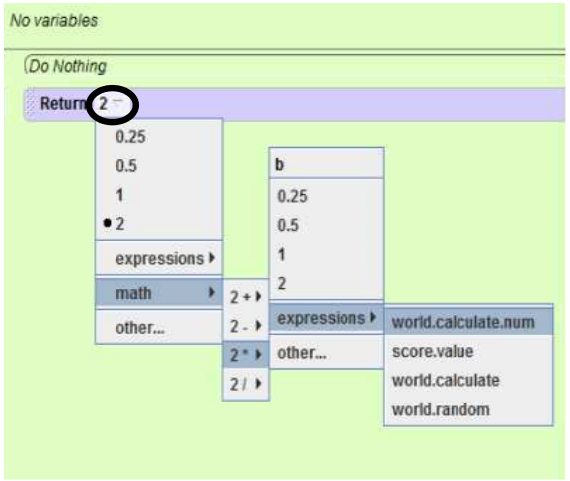




- This function will need a parameter to pass in different values to calculate. Click on "create a new parameter" on the right and name this number parameter *num*.

# Function

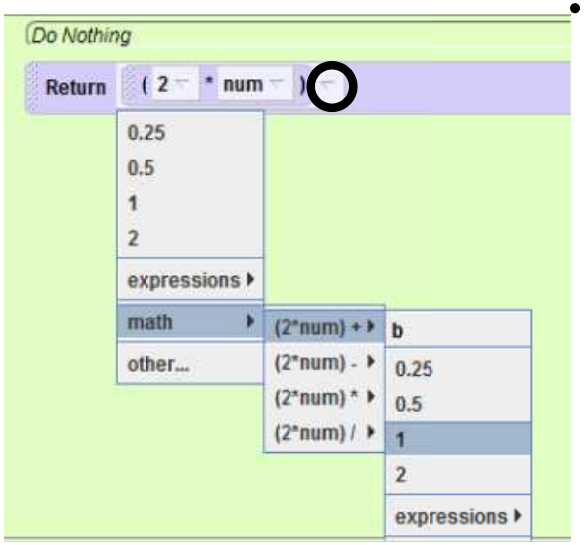
- Where the function returns 1, click on the drop down arrow, choose “other…”, and select 2. Then go back to the same menu next to the 2 and select “math”→ “2 *”→ “expressions”→ *world.calculate.num*.




# Filling in the Array

- Go back to world.my first method. Under world properties, drag the array variable into the loop and select “set item <index> to <item> in world.array”. Select *index* to be the index and *world.calculate* to be the item, both under “expressions”.

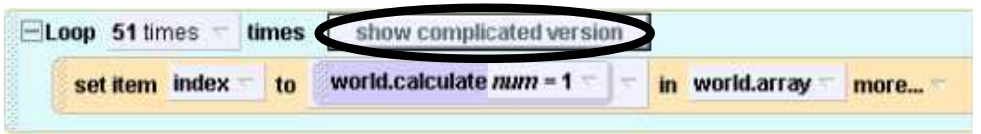

# Function




- Now, click on the arrow next to “Return (2 * num)” and go to “math” → “(2*num) +” → “1” to create a function that calculates 2x + 1.

# Filling in the Array

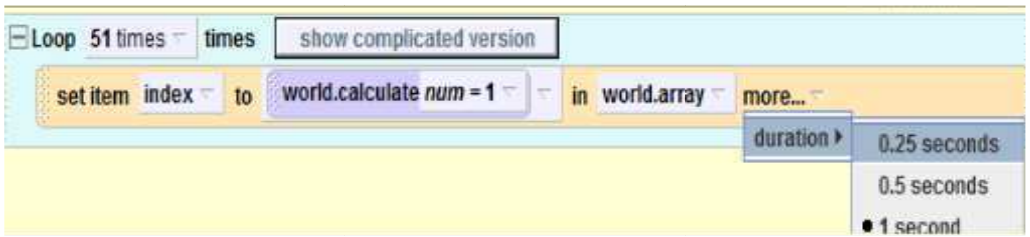
- Now, click on “show complicated version” on the loop in the method.

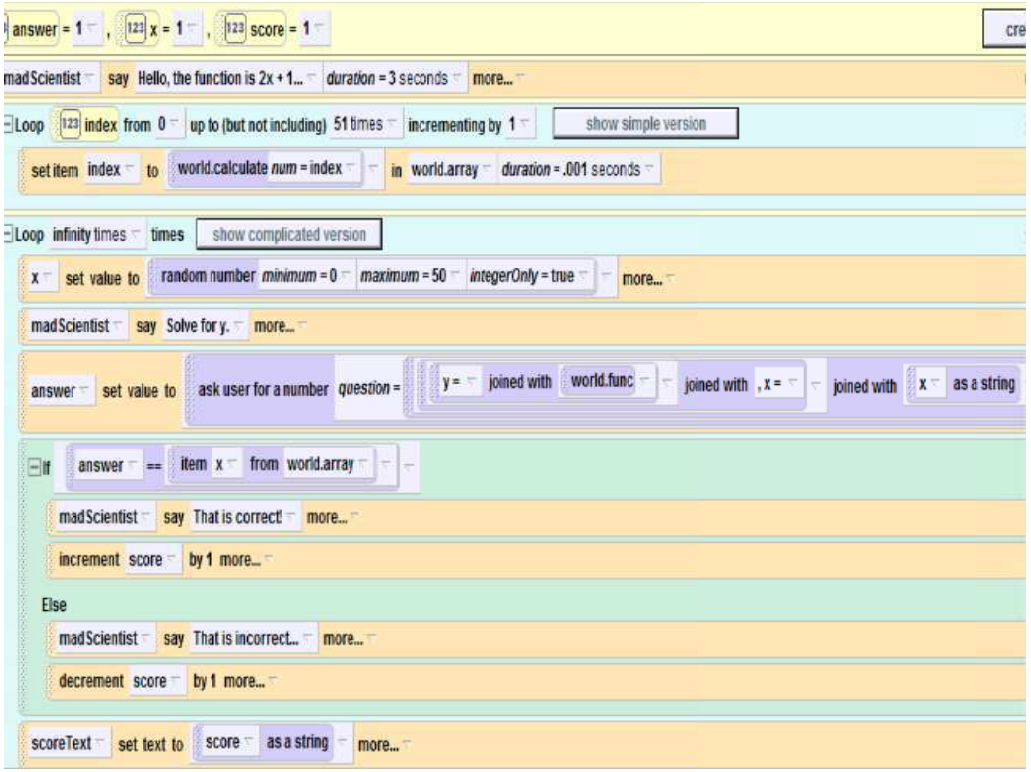



- Where the command in the loop says “world.calculate num=1”, drag the “index” object in the loop over it.

# Filling in the Array

- You are going to want to speed up this process, otherwise filling in 50 values in the array will take a while. Click on “more…” at the end and go to duration and set the value to something very small like .25 seconds.



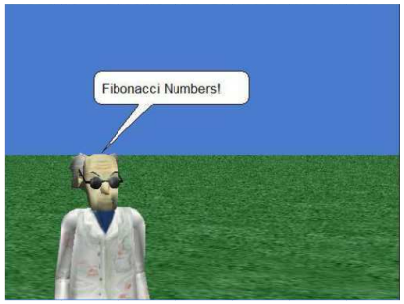

# Complete the World

- Now that the array is filled, we want to choose random values of x and then ask the user to input the solution of the equation at the given value of x, keeping score with how many they get right. Try this on your own, a basic solution is on the next slide.

# Challenges

- Make a similar world and change the function to a different equation such as $y = 3x + 5$ or $y = x^2 + 2x - 1$.

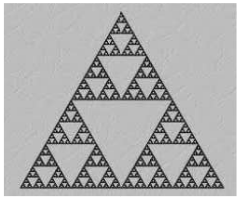
# Nonvisual Arrays and Recursion

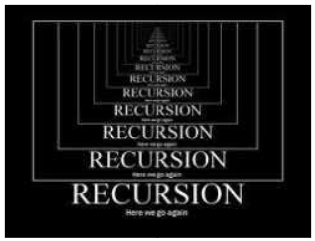


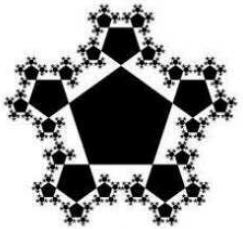

by Chris Brown
under Prof. Susan Rodger
Duke University
June 2012

# Recursion

- This presentation will also show how to use recursion, which is an advanced Computer Science programming concept. Recursion is when a function must call itself with a smaller problem in order to solve a larger one. It's similar when a word is used in the definition of the word, but using code. Here are some images showing examples of recursion below:





# Nonvisual Arrays

- This tutorial will display how to create and use nonvisual arrays in Alice. Nonvisual arrays are collections of any object or data type that don't necessarily have to be in order in the world as opposed to visual arrays, but they are still ordered in the array structure. We sill use this to store the values of our recursive function so that we don't have to calculate it each time we want to ask the user to solve for a specific value.

# Recursion

- In this tutorial, our recursive function will be Fibonacci's sequence of numbers. In Fibonacci's sequence, each successive number is calculated by adding the preceding Fibonacci numbers. Initially, the $0^{th}$ Fibonacci number is 0 and the $1^{st}$ Fibonacci number is 1 (0, 1, 1, 2, 3, 5, 8,…). Note here that fib($x$) = the $x$th Fibonacci number.
- Ex: fib(2) = fib(1) + fib(0), fib(3) = fib(2) + fib(1), …fib(n) = fib(n-1) + fib(n-2)…

# Standards

**CSTA Standard 5.3.B-** Computer Science Concepts and Practices (CT):

Students will be able to… "3. Explain how sequence, selection, iteration,

and recursion are building blocks of

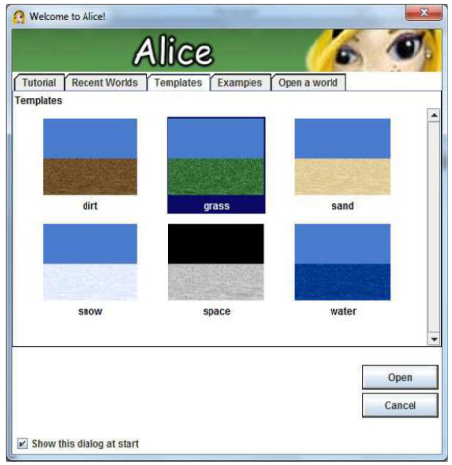
algorithms."

**CSTA Standard 5.3.B-** (CT):

"6. Compare and contrast simple data structures and their uses (e.g., arrays and lists)."

# Getting Started

- After opening Alice, choose any environment template and open it in Alice.




# Nonvisual Arrays and Recursion

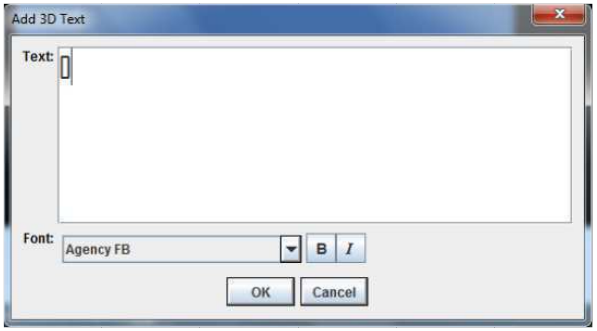
- In this world, we will use Alice to create a world where the user will have to enter the nth number of the Fibonacci sequence as prompted by the world. We will use recursion to calculate the values of the Fibonacci series and store those values in a nonvisual array.

# Getting Started

- The only thing that we will need to add to the world is a person to ask the questions and a 3D text object to keep score. Click on Add Objects and import the MadScientist in the Local Gallery in the People section.

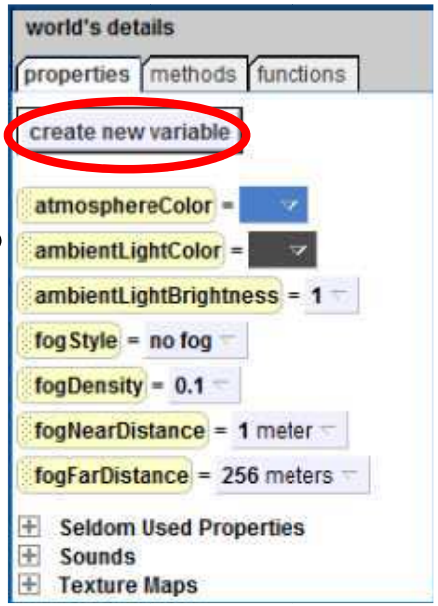
# Getting Started

- Then, scroll all the way to the right of the Local Gallery and select "Create 3D Text". Set the string to be "0".


# Nonvisual Array

- Now we're ready to start coding. Click "Done" on the right of your screen to go back to the method editor. To create the array, go to the "properties" tab under world's details and create a new variable.

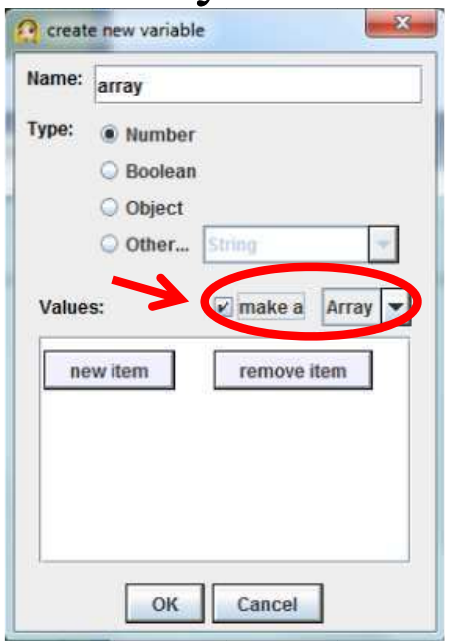



# Getting Started

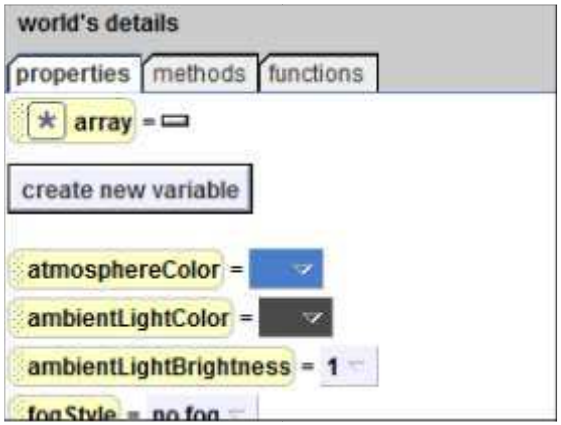
- Arrange the objects in the world using the move objects in the top right corner. Your world should look something like this:


# Nonvisual Array

- Name the variable *array* and make sure that it is of type Number. Then, at the bottom of the pop-up box, check "make a" and select Array from the menu. Don't add any items to the array yet.



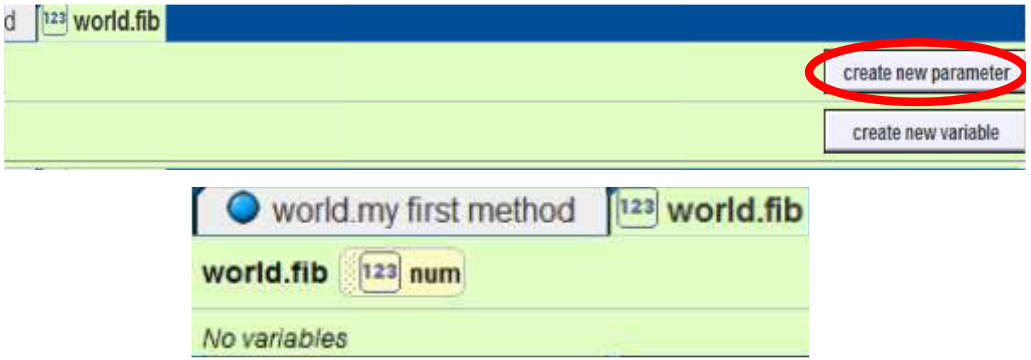
# Nonvisual Array




- You should see that the *array* variable was created in your world under the world's properties. Now we are ready to fill in the array with the Fibonacci numbers with a recursive function.

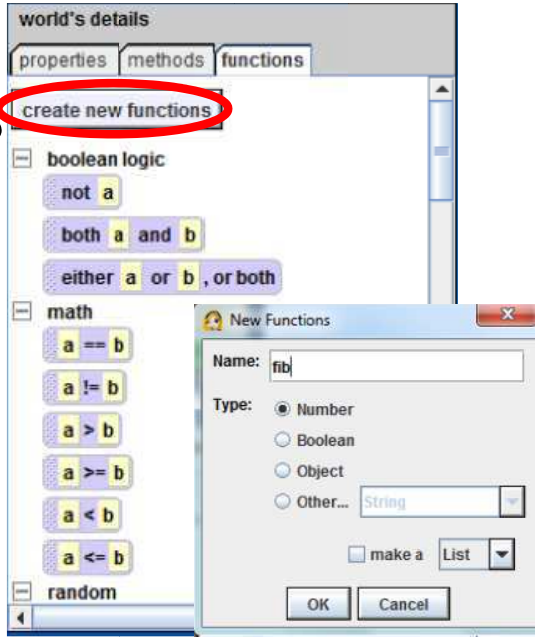
# Function

- Now in the *fib* function, we want to create a parameter to pass into the function when we call it. Call this parameter *num* and make it a number parameter.




# Function

- To create the function, go to the "functions" tab under world's details. Click on the button to create a new function and name this function *fib* and make sure it's a Number type function.




# Recursion

- When using recursion, the first thing you need to do is make sure that there is a "base case", or a way out so that you do not get trapped in infinite recursion. In this case, the base cases are: when *num* is 0, *fib*(0) = 0; and when *num* is 1, *fib*(1) = 1.

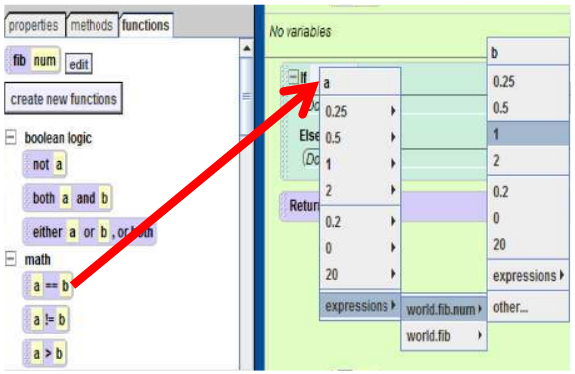
# Recursion

- At the beginning of this function, drag in an If/Else statement from the bottom of the screen into the "Do Nothing" and set the value to true for now.


# Recursion

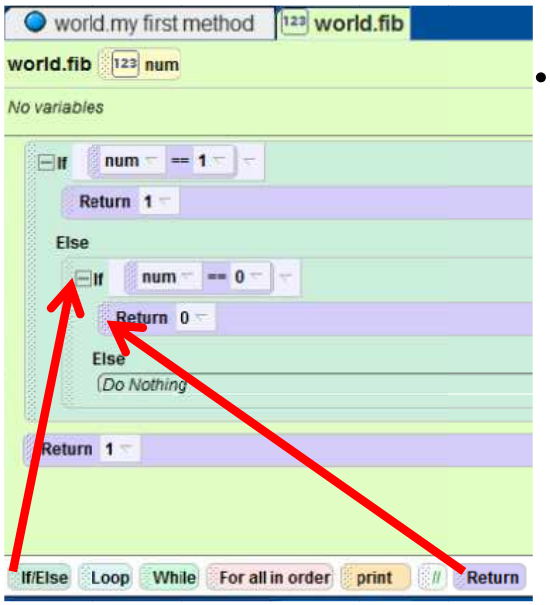
- Now, drag in a Return statement from the bottom of the screen to below the If. We want the function to return 1 if *num* is 1, so Return 1 if a == b is true.


# Recursion




- Now, under the world's "functions" tab, find "a==b" under the math heading. Drag this over the true value in the If/Else and set the value of a to world.fib.num and b to 1.


# Recursion



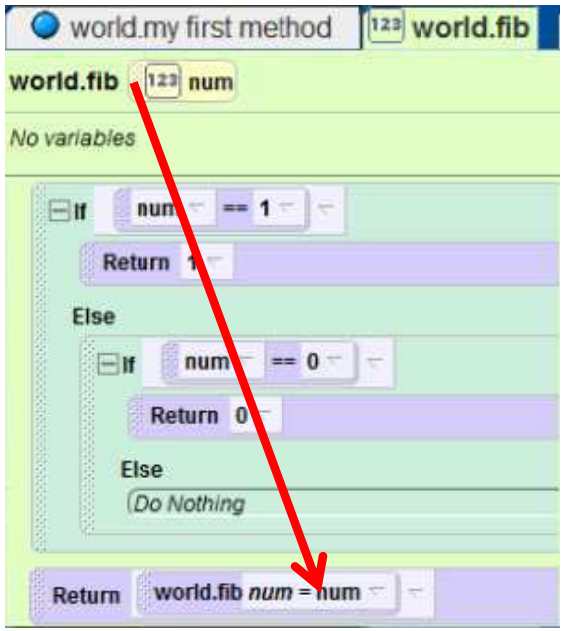

- In the Else section, we will need to check if the value of *num* is 0. Drag another If/Else statement inside the Else and check if *num* is 0. If *num* is 0, then we want to return 0. (In the menu for Return, you may need to select "other…" and type in 0.)

# Recursion

- At this point, we have finished checking for the base cases and are ready to use recursion. Recursion will be used at the bottom of the function, where it says Return 1.

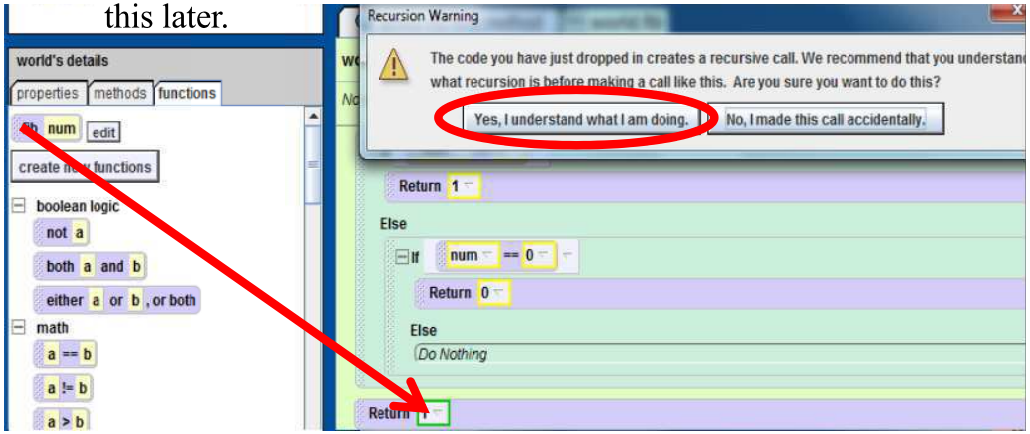
# Recursion

- Notice that inside the world.fib function we are calling world.fib. That is recursion.
- Now drag the *num* parameter over the value of *num* = 1 in the final Return value.

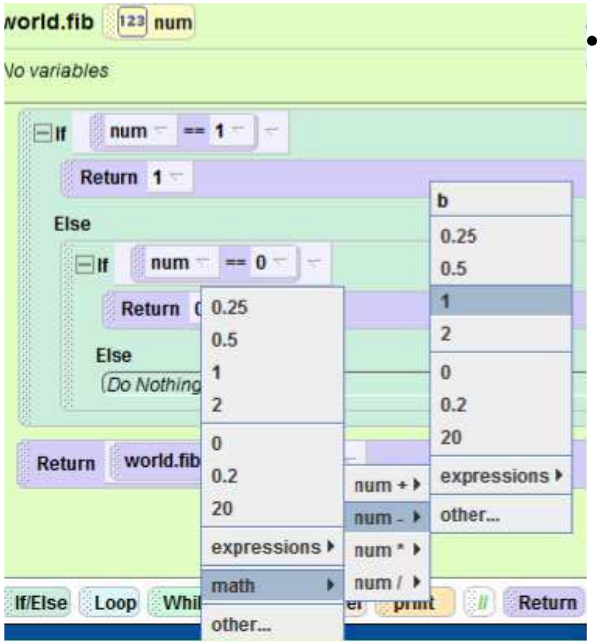



# Recursion

- In the world's functions tab, drag *fib* that we created over the 1 as the Return value. Alice will display a warning dealing with recursion to make sure you know what you are doing. Make sure to click Yes. Choose any value to be *num*, we will change this later.



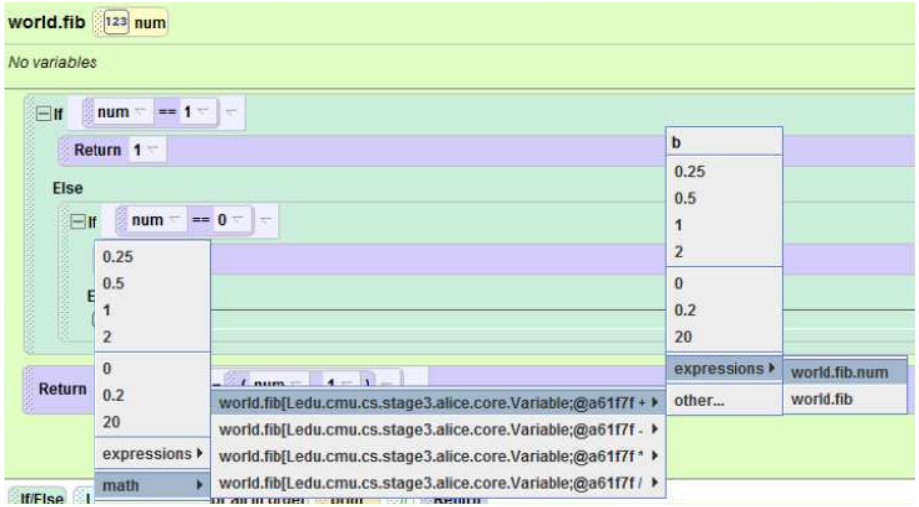

# Recursion




- Select the down arrow next to *num*, and go to "math" → "num -" → "1".

# Recursion

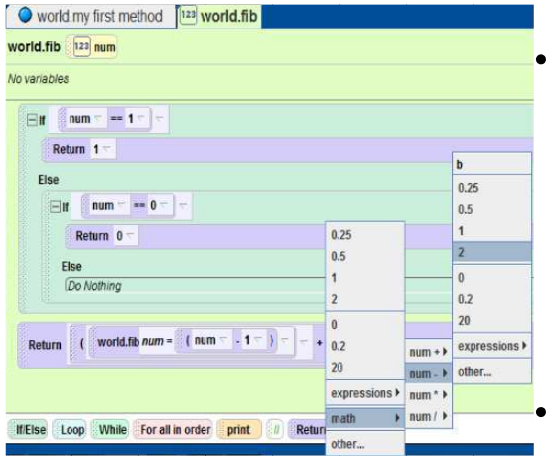
- Now select the down arrow at the end of the return statement and select “math” → “world.fib[… +” → “expressions”→ “world.fib”




# Recursion

- That concludes our recursive function. *fib* must call itself in order to find the sum of the previous 2 values of num. Note that there are two recursive calls to world.fib in the return statement and both values, (*num* – 1) and (*num* – 2), are smaller than *num*, which is the value world.fib is called with originally. Now, in world.my first method, we will fill in the array with the Fibonacci values.

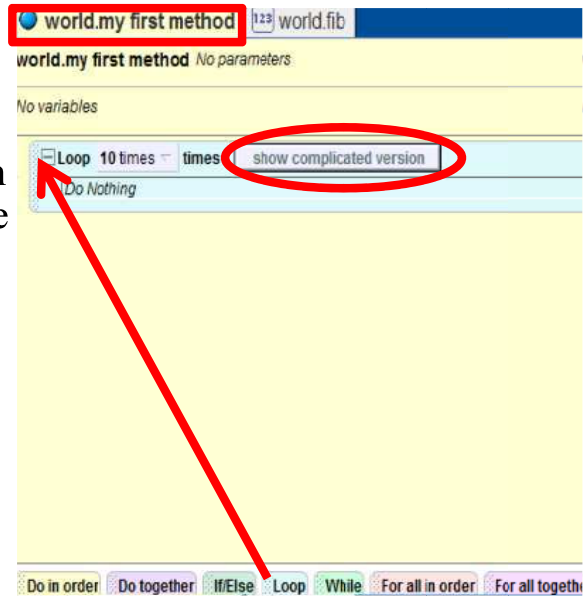
# Recursion




- Finally, drag the *num* parameter over *num* = 1, and click the down arrow next to *num* and go to “math” → “num - ” → “2”.
- The final Return statement should look like this:

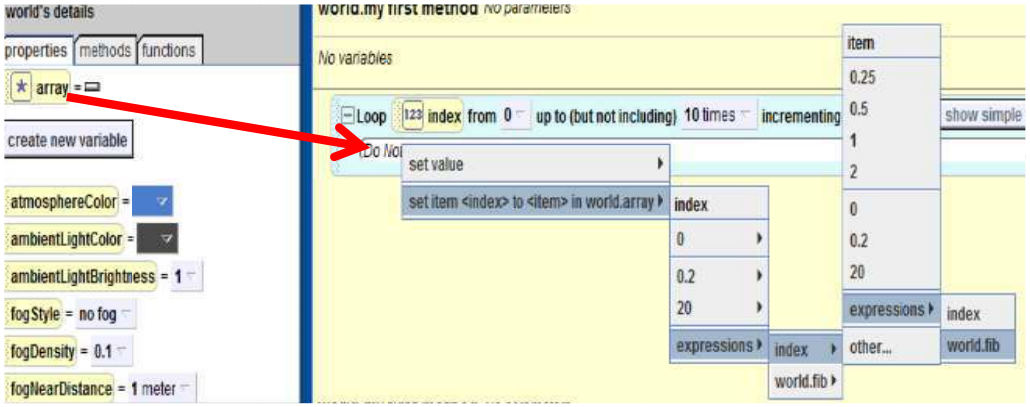

# world.my first method

- Click on the world.my first method tab.
- Drag a Loop into the method from the bottom of the screen and choose 10 times. This will eventually calculate the first 10 numbers of the Fibonacci series. Also click on “show complicated version”.



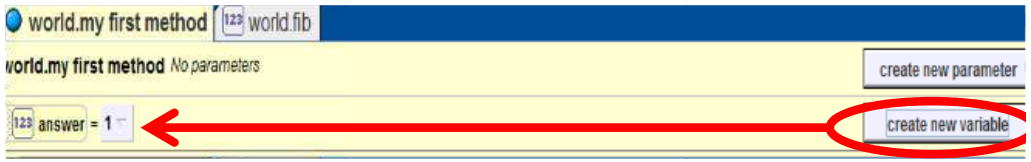
# world.my first method

- Now, drag the array into the loop and select "set <index> to <item> in world.array". Set the index to be *index* and item to *world.fib*, both under "expressions".



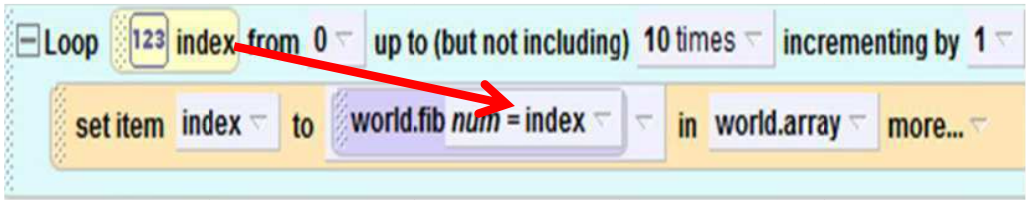

# world.my first method

- We are now finished filling in the array, and are ready to quiz the user. Create a new Number variable called *answer*. This variable will take the user's input.



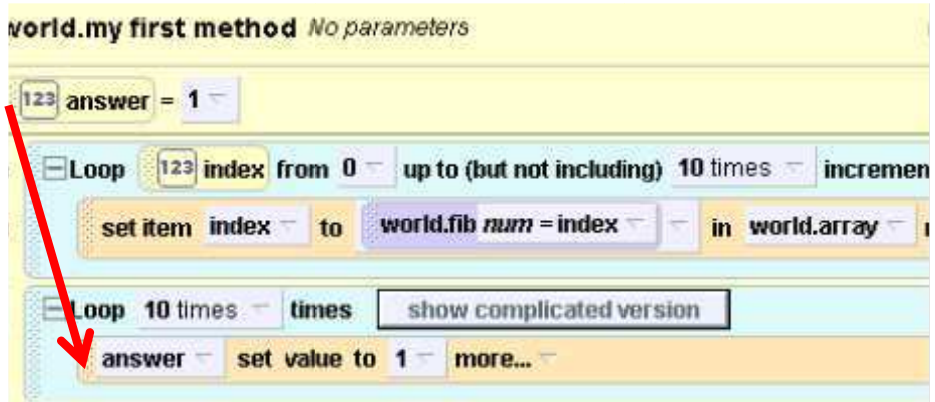

# world.my first method

- Once you have done that, then drag the index element from the loop to pass in as the parameter for *fib*.



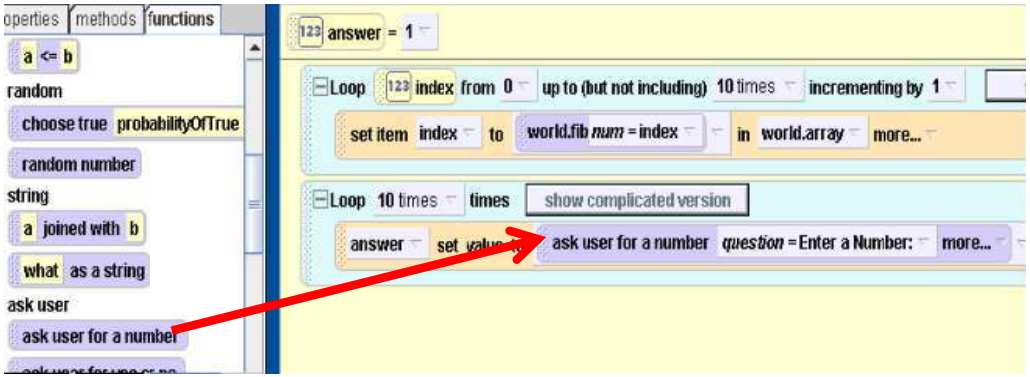

# Quiz

- Drag another loop into the method and set it to be the same length as the previous loop (in this case, 10). Then move the *answer* variable into this new loop and just set the value to 1 for now.



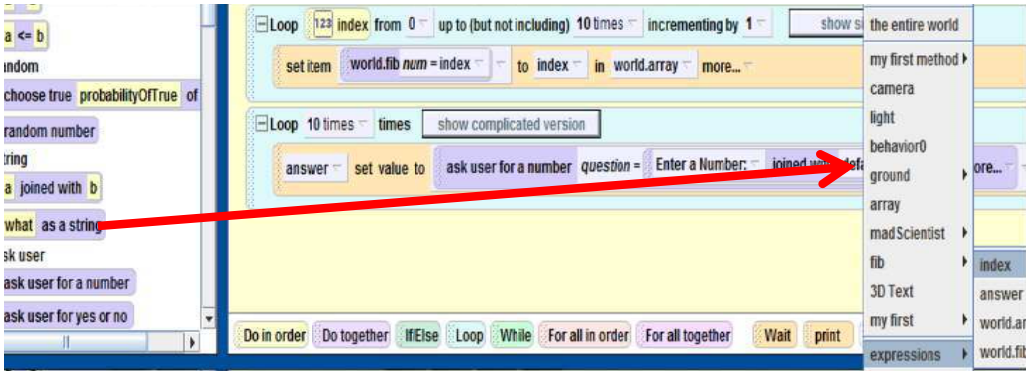
# Quiz

- Now, go to the world's functions tab and under the "ask user" heading, drag "ask user for a number" over the 1 that we set as *answer*'s value. This function will set the value of *answer* to be whatever number the user inputs when prompted while the world is running.




# Quiz

- Then drag "what as a string" over the default string for b and select "expressions"→ "index".

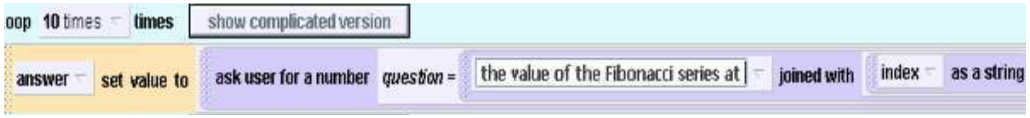



# Quiz

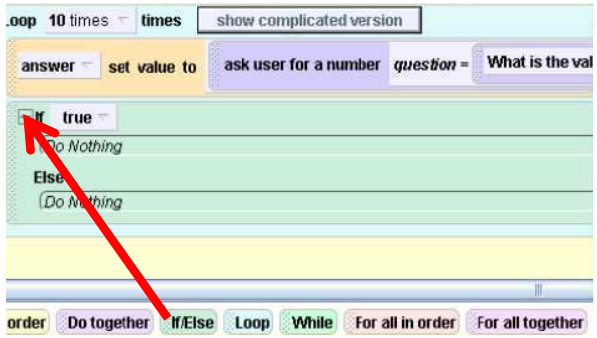
- Now, under the world's functions "string" heading, drag "a joined with b" over the question asked for the value of *answer*. Let b be the default string for now.


# Quiz

- For the value of *a* in the question, click on it and ask something like, "What is the value of the Fibonacci series at ".

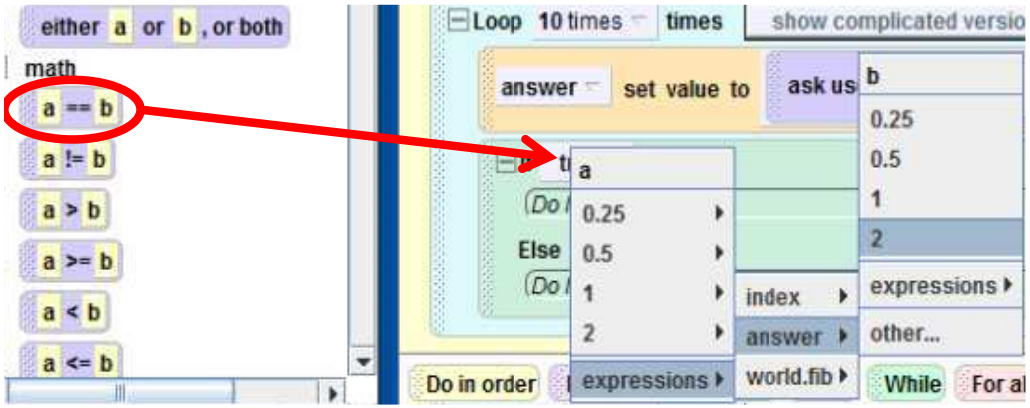



- Now, drag an If/Else into the second loop and select true.



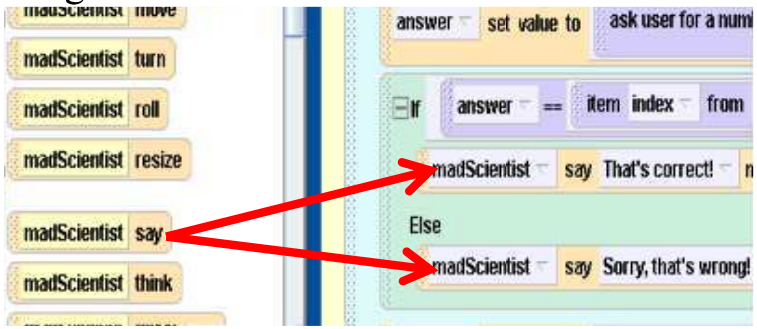
# Quiz

- Under the world's functions, drag "a == b" over the "true" in the If/Else statement. Set a to be *answer* under "expressions" and b to be any value for now.



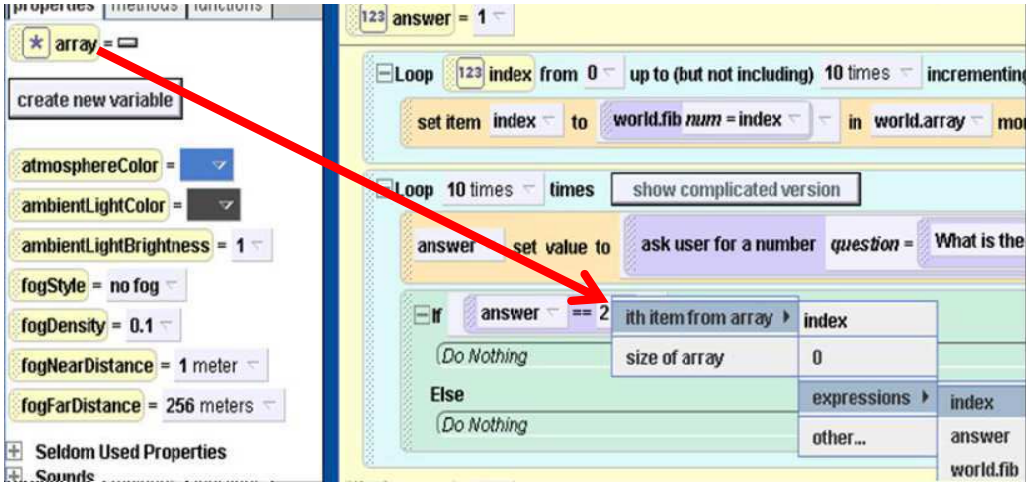

# Quiz

- If the user gets the answer right, we want to update the score and have the MadScientist say "That's right!". If they get it wrong then we want to have him say, "Sorry, that's wrong." The *say* method is under the MadScientist methods. Click on MadScientist in the object tree and drag "madScientist say" into the If/Else and enter the string after clicking "other…".

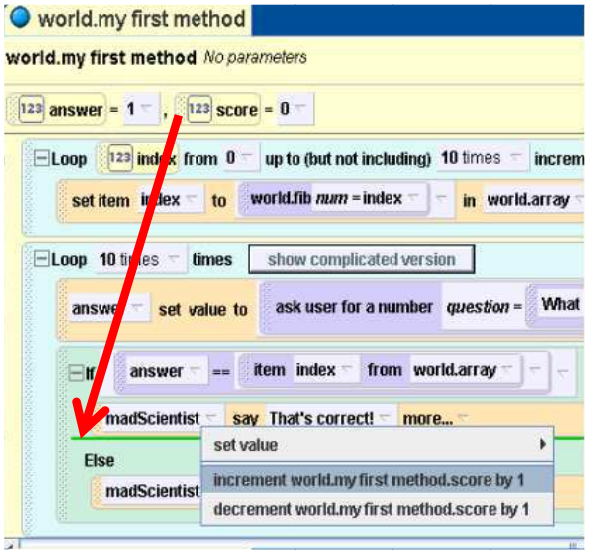



# Quiz

- In world's properties, drag *array* over the value of b and choose *index* as the "ith item from array".



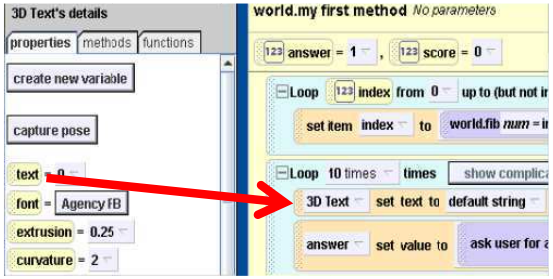

# Quiz

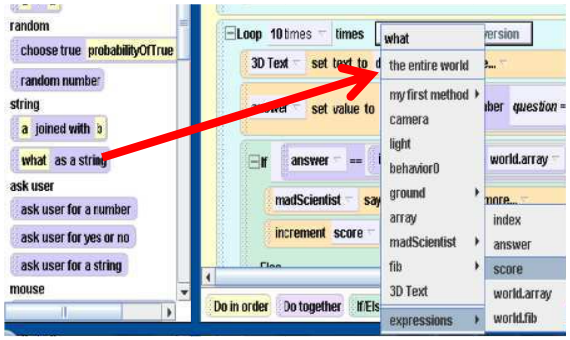



- For keeping score, create a new number variable called *score* and set it to 0. Drag *score* into the If part of the If/Else and choose "increment score by 1".

# Quiz

- Now to update the score on the screen, go to the properties of 3D Text and drag "text " to the top of the second loop.

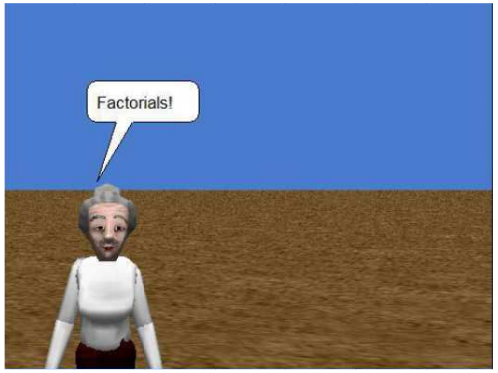



- Over the default string, go to the world functions and drag "what as a string" over it, and then select *score* under expressions.

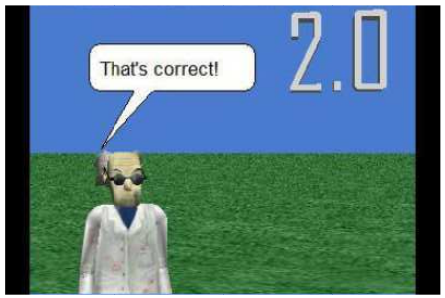
# Challenge

- Try creating a world similar to this one that asks users to calculate factorials, another recursive mathematical function. What would be the base case?

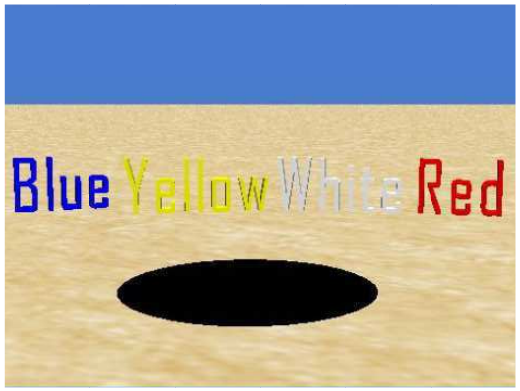



# Conclusion

- That concludes our world! Run the world to try it out and take the quiz to calculate the first 10 Fibonacci numbers. This world used nonvisual arrays to store all of the values and recursion to calculate the values of the Fibonacci series.



# Probability




by Chris Brown
under Prof. Susan Rodger
Duke University
July 2012

# Standards

**NC Standard Course of Study Mathematics Grade 6-**

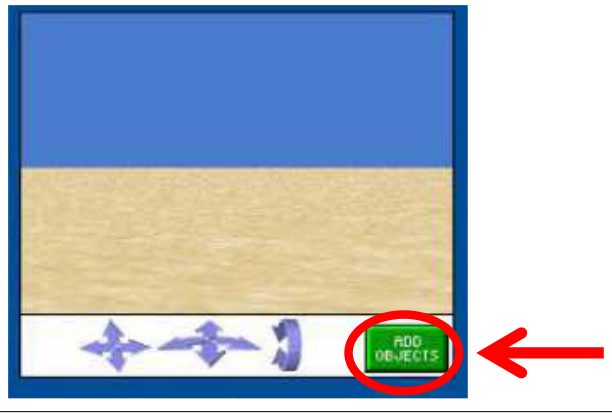
**Goal 4:** The learner will understand and determine probabilities.

# Probability

- In this world, we will create a probability game where students will have to give the probability of choosing a certain colored ball from a hole in the ground. They are provided with the total number of balls and the number of balls for each color and will be asked to calculate the probabilities of choosing random balls out of the group. This world will help students learn about probabilities and fractions.

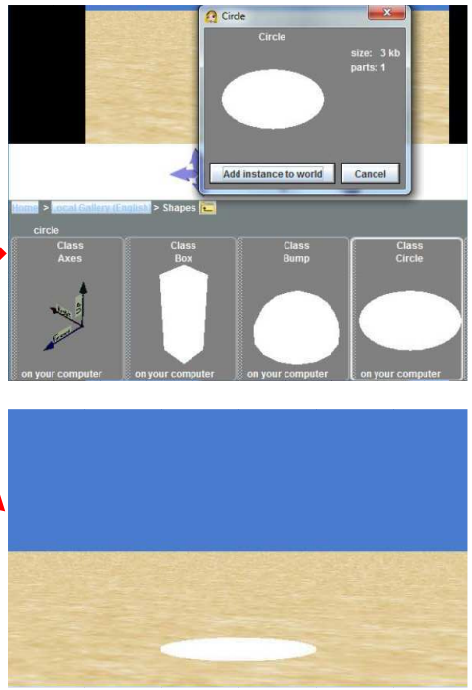
# Set Up

- For this world, I chose the Sand template after opening Alice. Once you have done that, click on the "Add Objects" button to add all of the things we will need into this world.



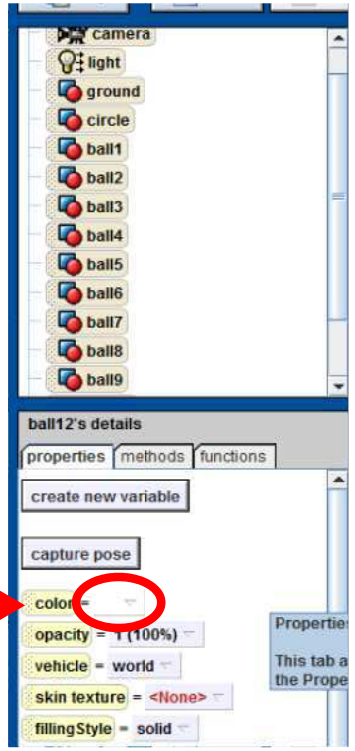
# Set Up

- The first object we want to add is a circle, to represent a hole in the ground where we will place our balls. Scroll over to the "Shapes" folder in your local gallery and import the "Circle" class.

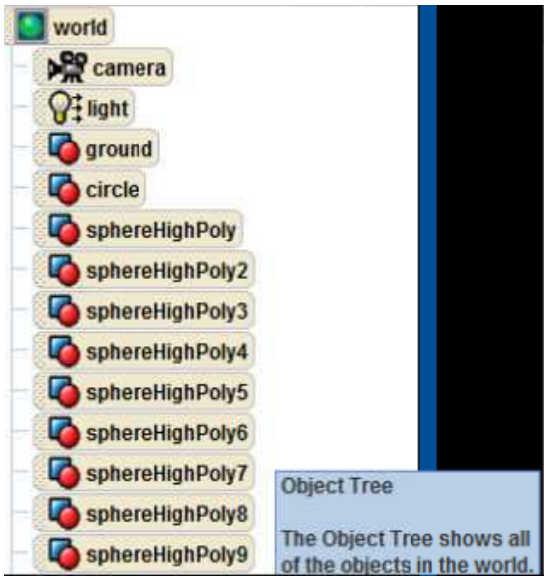



# Set Up

- To change the color of a ball, make sure its name is highlighted in the object tree and select the properties tab under the ball's details. Click next to the color property to change the color of the ball and you should see the ball change colors.




# Set Up

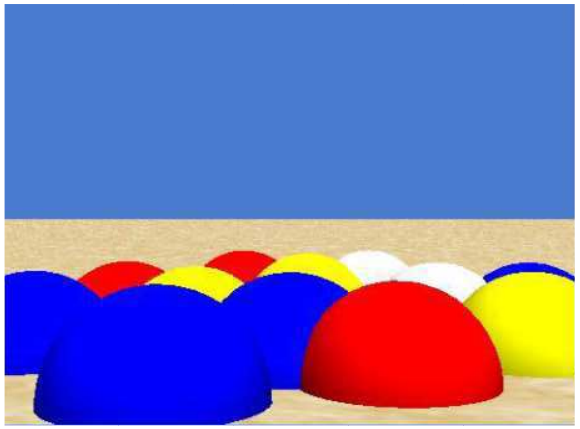



- Now, while we are in the Shapes folder, we want to add 12 spheres to represent the colored balls that the program will choose randomly. SphereHighPoly makes the spheres look rounder, so I used those. You will see 12 sphereHighPoly objects added to the object tree in the top right corner. I changed their names to ball1 to ball12 just for clarity by right clicking on the name and selecting rename.

- Note: After adding each one, you will need to move it or Alice will place all 12 spheres on top of each other.

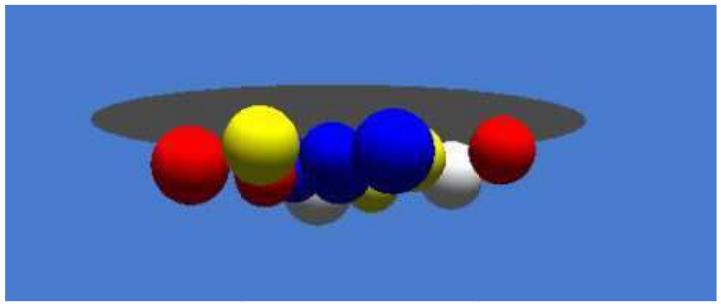
# Set Up

- Change the colors of the balls so that there are 4 blue balls, 3 red balls, 3 yellow balls, and 2 white balls.



# Set Up

- Now, resize the balls using the icons on the right side of the screen so that they are smaller and move them so that they are all “inside” of the hole, or just below the circle in this case. Here’s a picture from the bottom, but your world should look like the picture on the next slide.

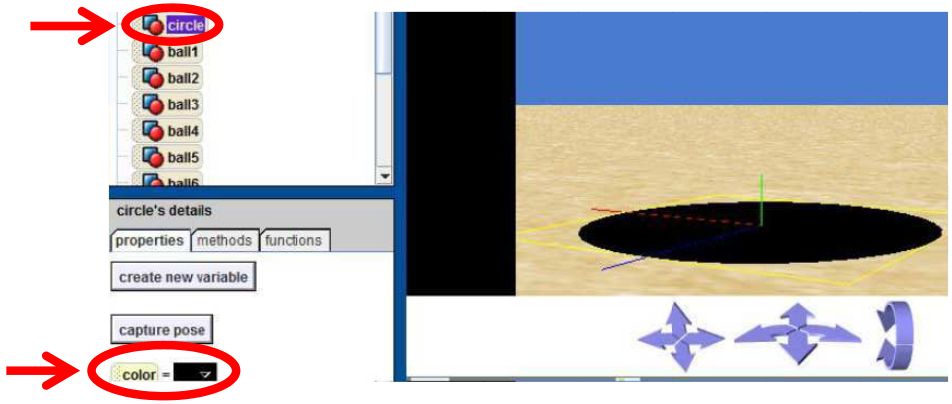


# Set Up

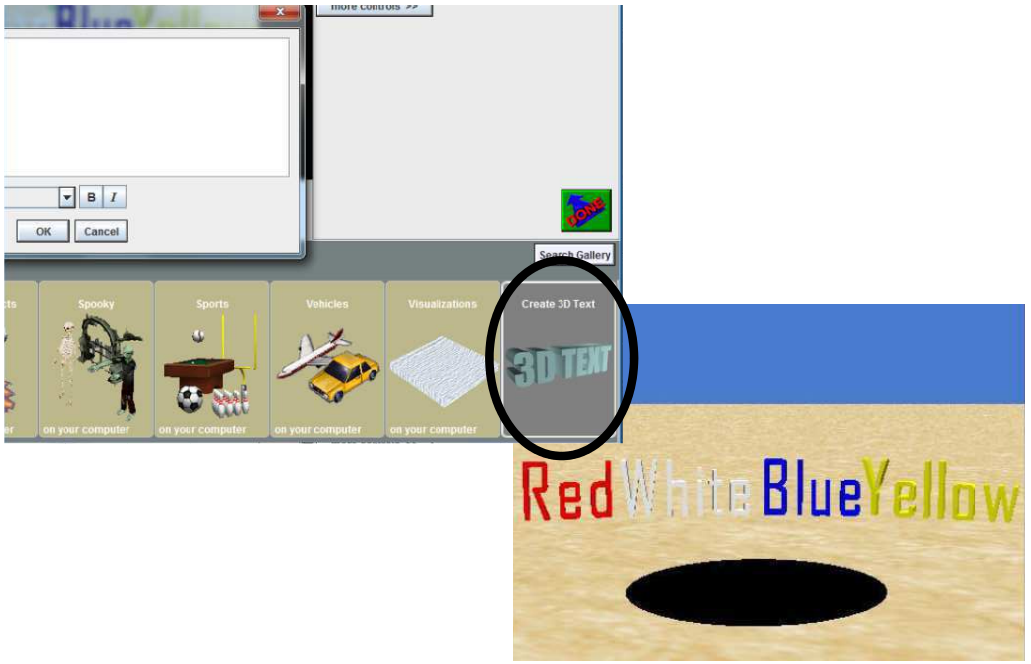
- Next, we want to add 3D text objects, one for each possible color of the balls, to update while the game is being played with the number of balls of that color left. Scroll to the end of the Local Gallery and import four 3D text objects, setting the text of each one to “Red”, “White”, “Blue”, and “Yellow”. Also, go into the properties of each of these new text objects and change their color to their respective string values. I renamed the 3D text objects red, white, blue, and yellow to make a distinction between them in the Object Tree.
- See the next page for the final setup of the 3D text objects.

# Set Up

- Change the color of the circle to black so that it actually looks like a hole by going to circle’s properties and changing the color to black.




# Set Up



# Set Up

- Now we want to save the start camera position because we will be moving the camera. To do this, you will click on "more controls >>" on the right and then click on 

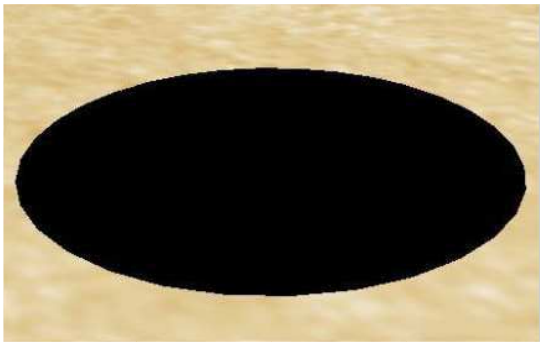
. In the object tree, you will see a new folder called Dummy Objects and you are going to want to rename the dummy object in that folder to *start*.

# Set Up

- Now we have all of the objects that we will need set up in this world and can start programming! Click on the Done button to continue to world.my first method.

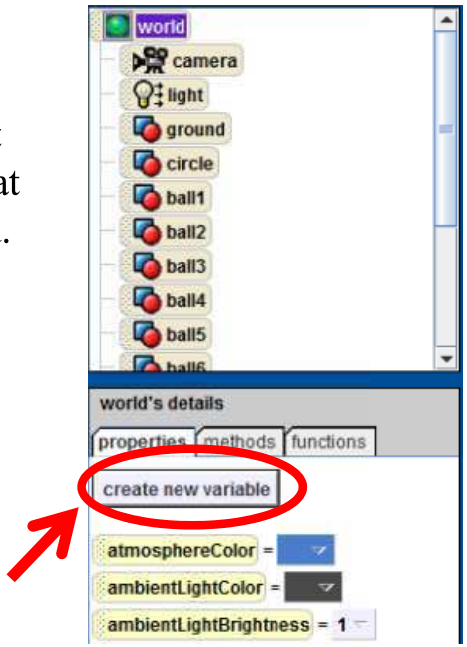
# Set Up

- Now we want to add a new camera view to animate when the balls are taken out of the hole. Use the arrows below the camera view to move the camera to get a close up of the hole by itself, and drop another dummy object at the camera. In the same folder, rename this camera view object *choose*.

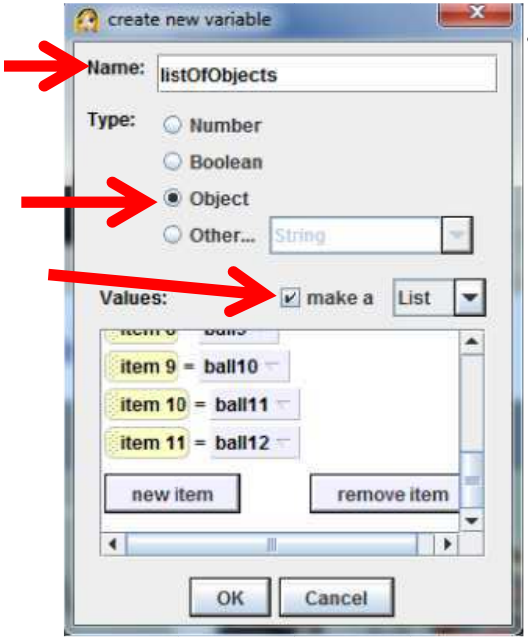


# Lists

- The first thing that we need to do is make a list of all the ball objects that we added into the world. Make sure world is selected in the Object Tree, and select "create new variable" under world's properties.

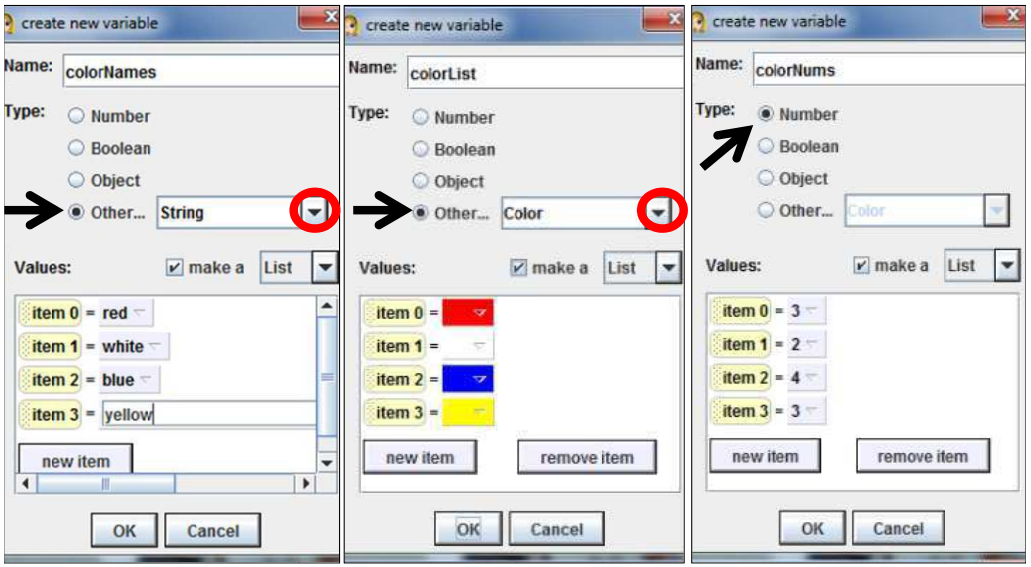


# Lists




- When the menu pops up, name the variable *listOfObjects*. Make sure that you select Object as the variable type and check the box next to "make a List". Add 12 new items (item 0 – 11) and choose each ball to be an item in the list. Click OK when you are finished and you should see this list under world's properties.

# Lists

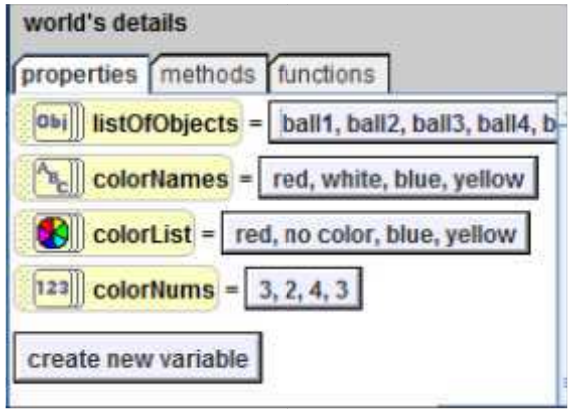



# Lists

- Now we want to create three more lists: one that contains the string of color names (*colorNames*), one that contains the actual colors (*colorList*), and finally one that contains the number of balls at each color left (*colorNums*). Note that the indices of all three of these lists must correspond to the same color (0 = red, 1 = white, 2 = blue, 3 = yellow).

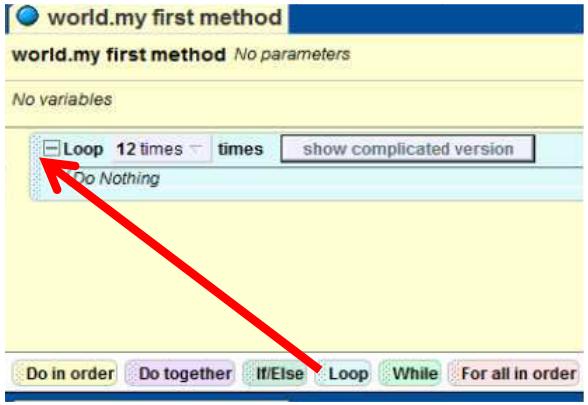
# Lists

- When you're done, you should see four list variables under world's properties.



# world.my first method

- Now we're ready to start adding code to world.my first method. The first thing that you want to do is drag a Loop into the "Do Nothing" section of the method and set the value to however many times you want to quiz the user to calculate the probability.

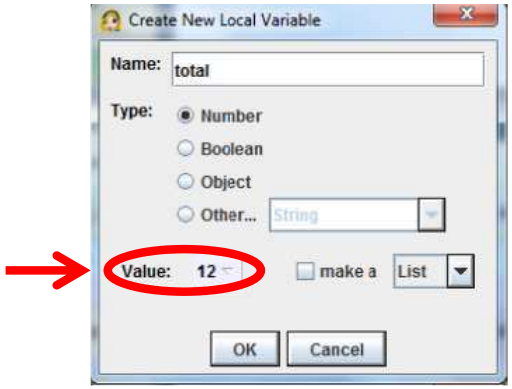



# Changing 3D Text

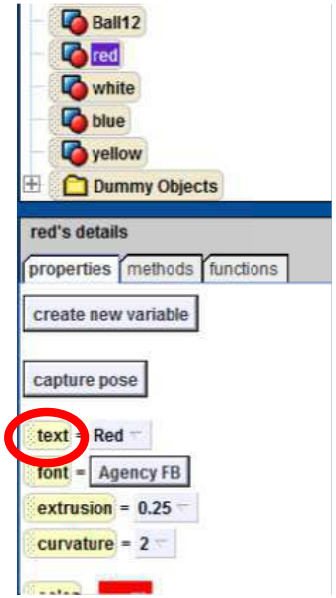
- The first thing that we want to do is change our 3D text so that it displays the number of balls left at each color. Drag a Do together into the loop we just added to change all of the 3D text values at the same time.


# world.my first methd

- Create a new number variable called *total* that will keep track of the total number of balls left in the hole. Each time a ball is taken the total will decrease by 1. Initially set the value of *total* to 12.



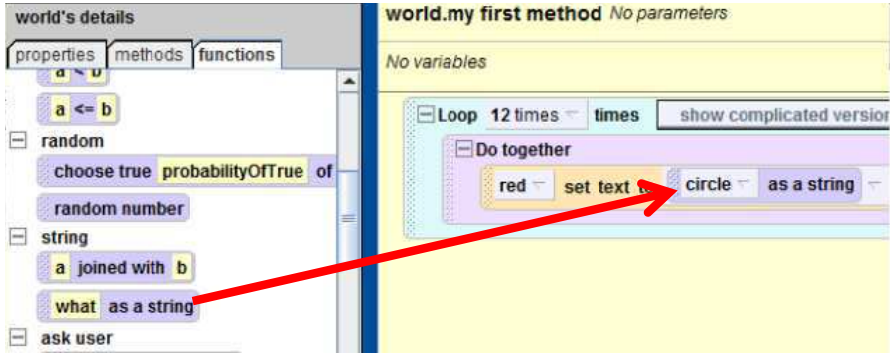

# Changing 3D Text




- Now, to change the text of a 3D text object, you will go to its properties and drag the "text" variable into the Do together. Set the value to default string for now, because we will change this next.

# Changing 3D Text

- Click on world in the object tree and go to world's functions. Under the string tab, you should see a "what as a string" function. Drag that over the value of the default string that you set the 3D text red to. You can also choose any object for now because this will be replaced. I chose "circle" in this example.

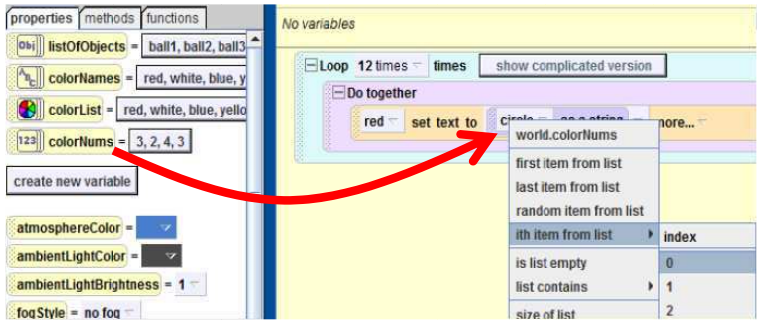



# Changing 3D Text


- Then, you are going to want to set the other 3D text values to their corresponding value in the *colorNums* list inside of the Do together as shown above.

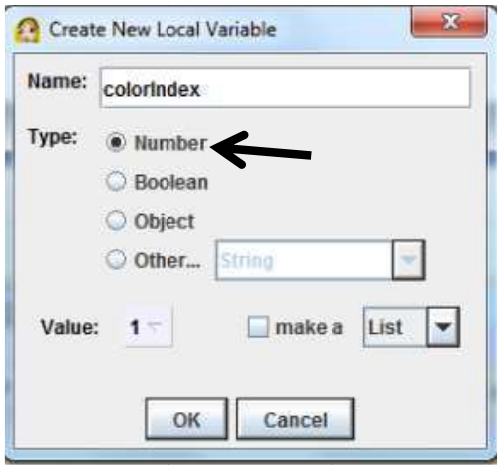
# Changing 3D Text

- Now go to world's properties to the lists that we created earlier. Drag the list *colorNums* over the arbitrary object that you picked earlier. When asked for the index, go to "ith item from list" → "0", to choose the number at position 0 in the list which represents the number of red balls.

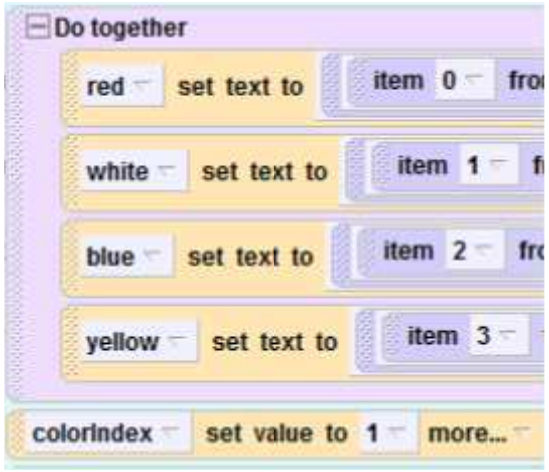



# Random Number

- Now we want to create a random number variable to ask the user the probability of selecting random balls from the hole. Click on "create new variable" in the right corner of the method editor and call this new number variable *colorIndex*. Make sure the Type is "Number".



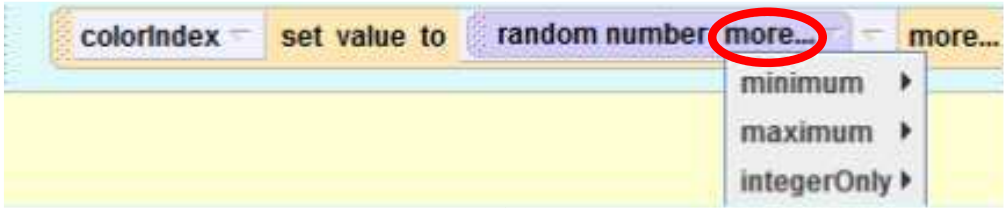
# Random Number




- Drag the *colorIndex* variable from the top of the method into the loop but below the Do together, and set the value to any integer for now.

# Random Number

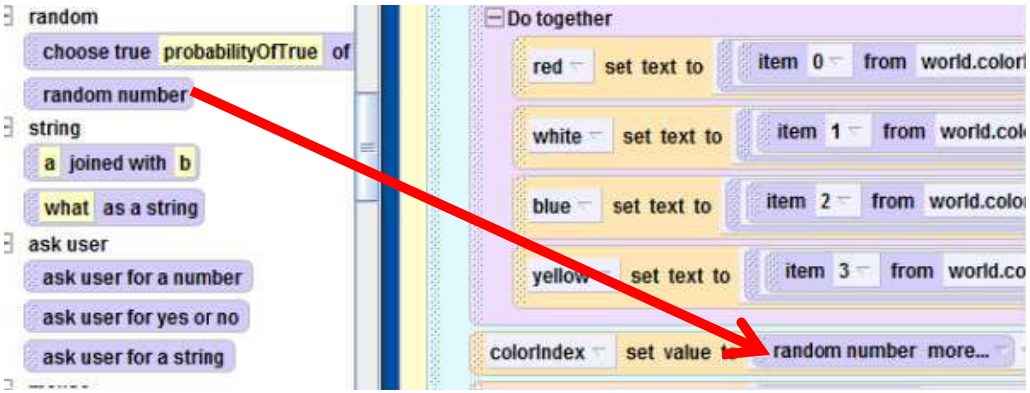
- Click on the purple "more…" in the random number function and set the minimum value to 0, the maximum value to 4, and integerOnly to be true.



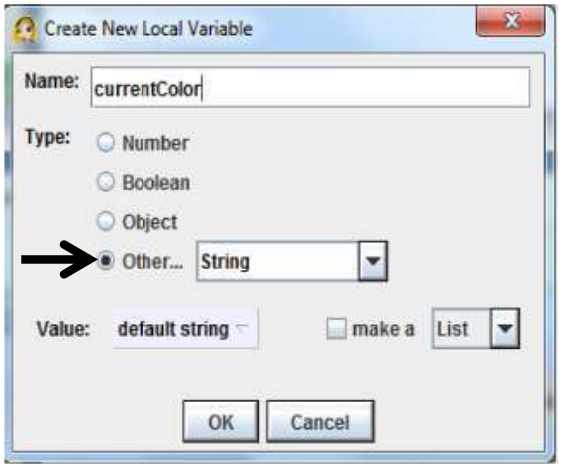


# Random Number

- Under world's functions random tab, drag random number over the value that you just chose for *colorIndex*.




# world.my first method



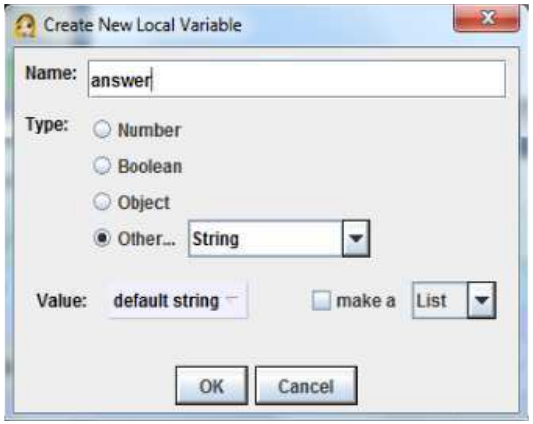

- Now, in world.my first method, create a new String variable called *currentColor*. This variable will keep track of the string of the current random color so we can ask the user to calculate the probability.

# world.my first method

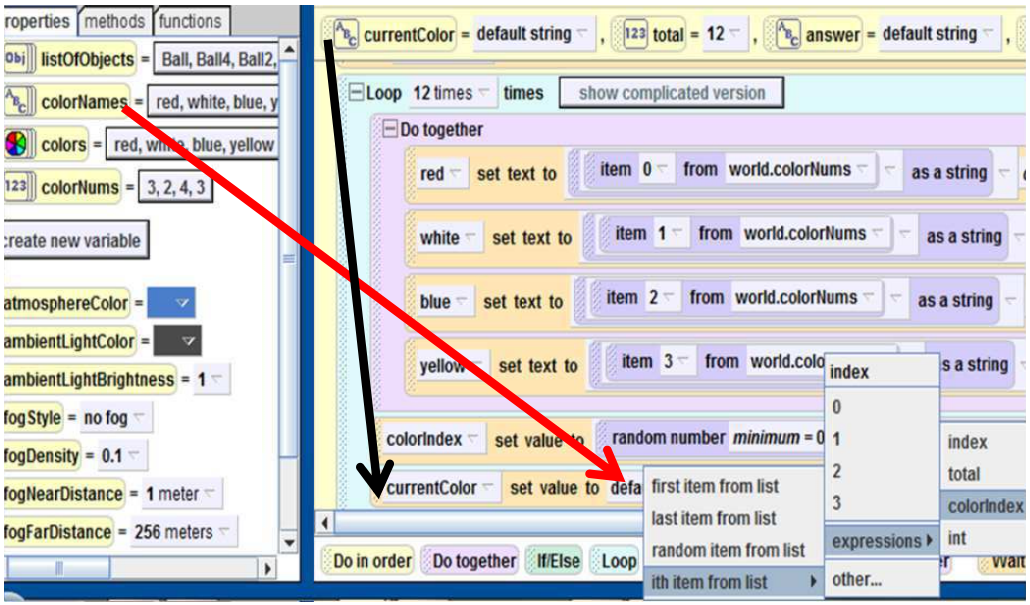
- We want to set the value of *currentColor* to the value of the list *colorNames* at our random *colorIndex*. First, drag the *currentColor* variable to the bottom of the loop and set the value to default string for now. Then drag the *colorNames* list over the default string and go to "ith item from list" → "expressions" → "*colorIndex*" for the index. You can see a picture on the next page.

# Quiz

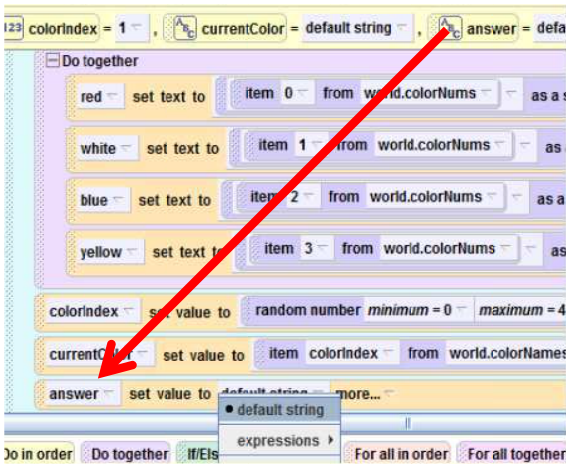
- At this time, we are ready to quiz the user on the probability of choosing a certain ball. The first thing we want to do is create a new string variable called *answer* in world.my first method, to save the answer that the user will give.




# world.my first method



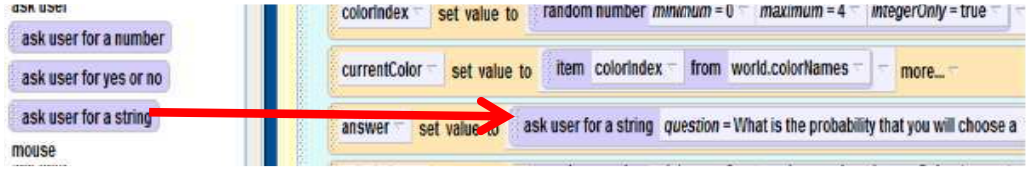

# Quiz




- Next, we want to set the value of *answer* so that it will save the answer that the user types in. Drag *answer* to the bottom of the loop and set the value to the default string for now.

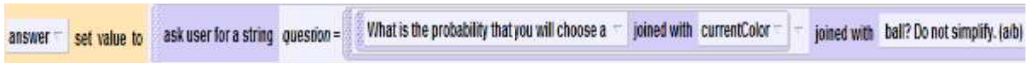
# Quiz

- Then, go to world functions under the ask user tab and drag “ask user for a string” over the default string that we set *answer* to. When prompted for a question, select “other” and type in, “What is the probability that you will choose a ” and we’ll complete this question later.



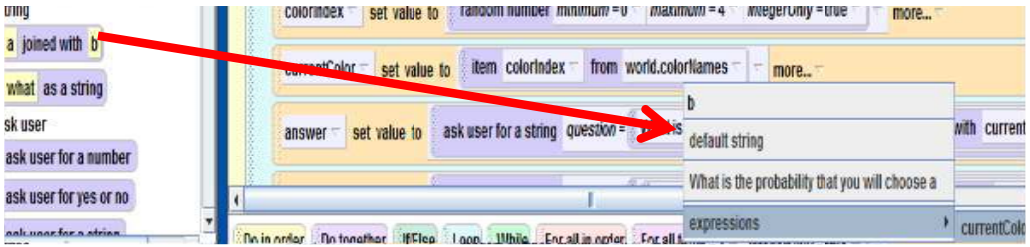

# Quiz

- Repeat the last step and drag “a joined with b” over the entire question string. This time for b, choose other and type in “ ball? Do not simplify. (a/b)”, to give the user more information about the input of their answer.



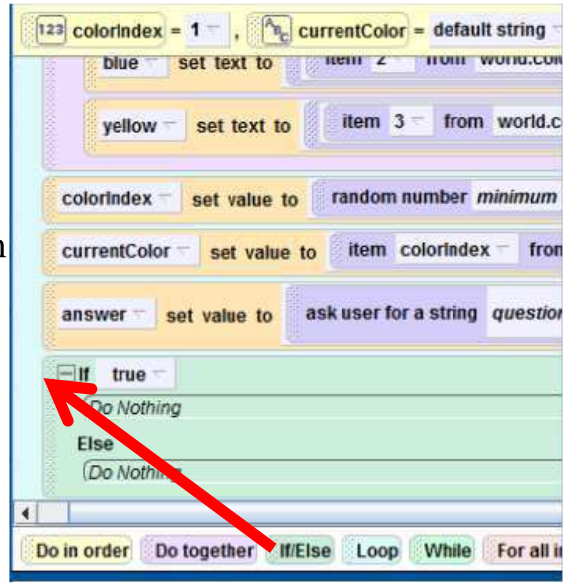

# Quiz

- In world functions under the string tab, there is an “a joined with b” function. Drag that over the string we just typed earlier in for the question and for b, go to “expressions” → *currentColor*.



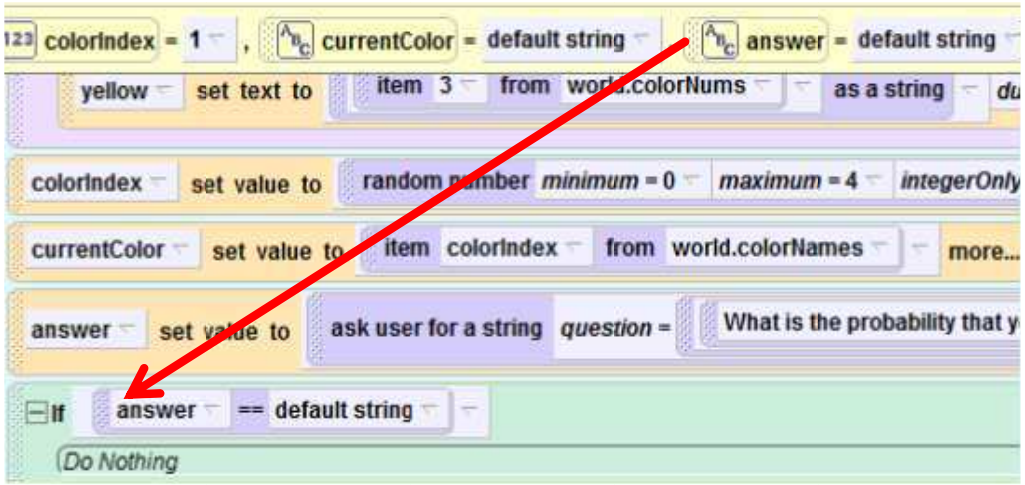

# Quiz

- Next, we want to check if the answer that the user gave is actually correct. We can do this by dragging up an If/Else statement from the bottom of the method editor and dropping it in the bottom of the loop. Set the condition to be true for now.



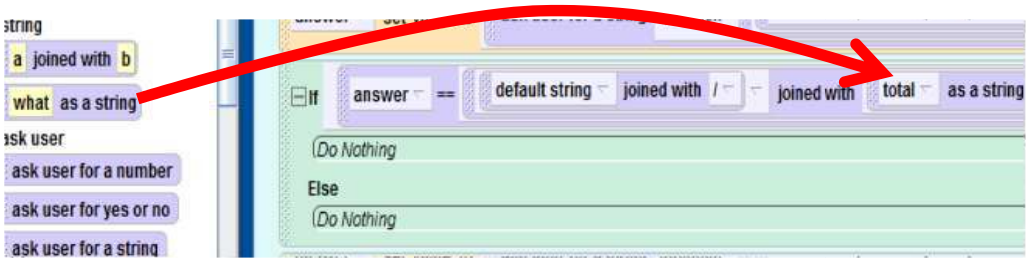
# Quiz

- Drag the *answer* variable over the true in the If/Else statement and select "answer ==" → default string.



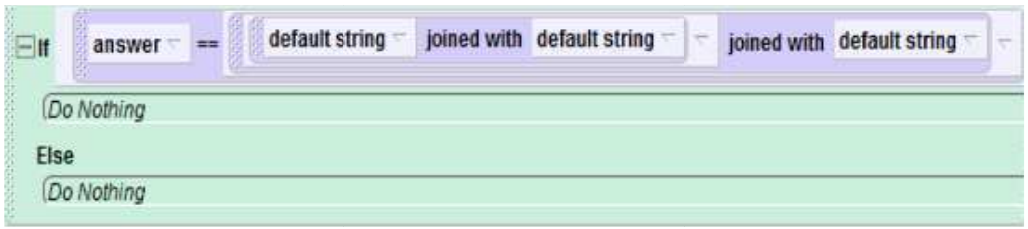

# Quiz

- For the center default string, simply change that to "/" to represent the bar of the fraction. The denominator will be the last default string, so go to world functions and drag the "what as a string" function over the last default string and choose "expressions" →*total*.




# Quiz

- Drag in an "a joined with b" function from world's functions over the default string, and then drag another "a joined with b" function over the previous two.

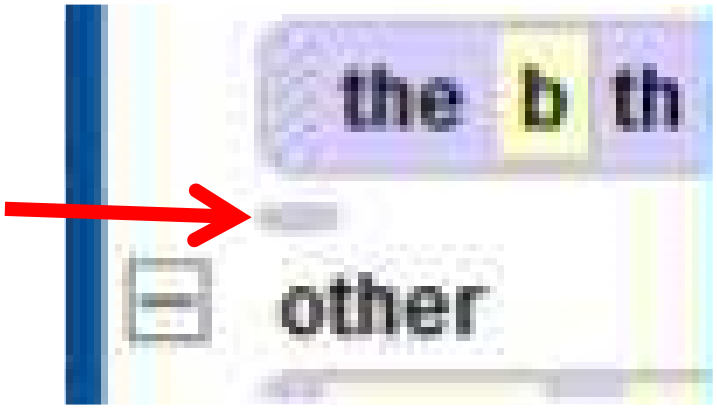



# Quiz

- One thing to note is that in Alice, integers are represented as i.0. Instead of having the user type in a.0/b.0, we just want their answer to be in the form a/b. To do this, we will need to use the world's "int as a" function.

- Note: As of March 2012, there is a bug in Alice that hides this function for some reason, but it is a very tiny purple speck at the bottom of the advanced math section of world functions. Drag that over total and select "more…" → "a" → "expressions" →*total*. The next few slides will show this clearly. This bug should be fixed in the next version.

# Quiz




# Quiz

- In the default string representing the numerator, you will also need to drag the world functions "what as a string" over the default string, and then the "int as a" function over that. This time, however, choose any value for a and we will change this later. I chose 1 below.

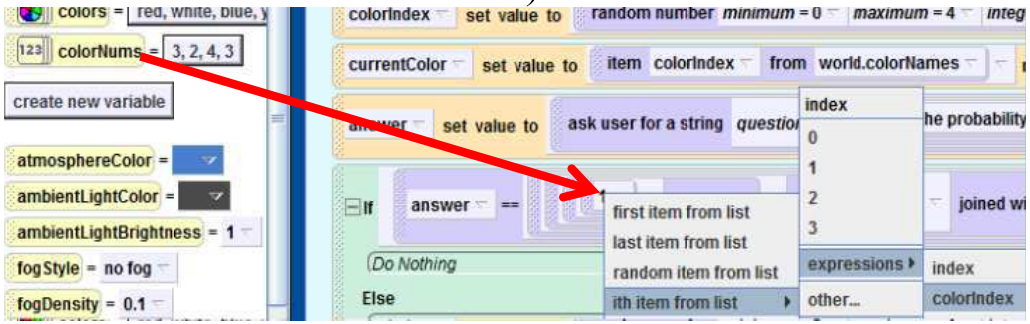

- Note that there are 4 tabs before the "1" for 3 functions, that means your statement is correct with the "int as a" function bug.

# Quiz

- Drag this function over *total* that we placed in the "what as a string" function earlier. To set the value, choose "more" → "a" →"expressions" → *total.*

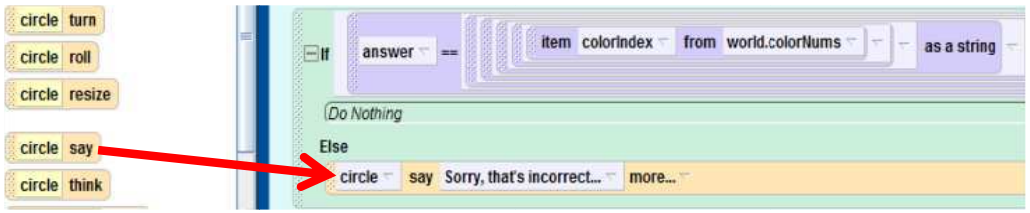

# Quiz

- Under world's properties, drag the *colorNums* list over the value that you chose earlier and select "ith item from list" → "expressions" →*colorIndex* to get the number of balls of the specific color left. The correct answer should be (# of balls for the current color/total number of balls).

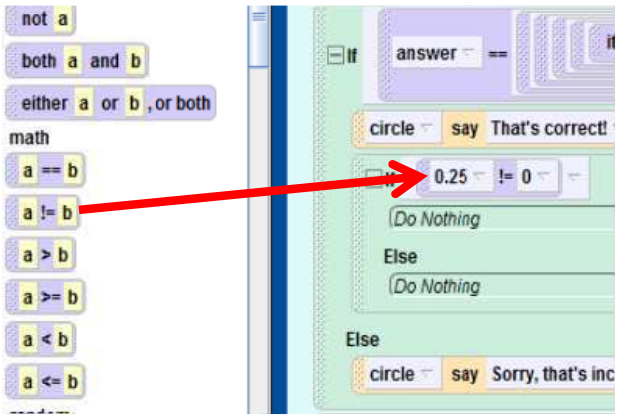


# Quiz

- Now we are done checking to see if the answer is correct, but we need to decide what to do if the answer is right or wrong. If the answer is wrong, we want the circle to say, "Sorry, that's incorrect…". Click on circle in the object tree and go to circle's methods. Drag "circle say" into the Do Nothing of the Else portion of the If/Else statement at the bottom and choose "other" to type in a string. If the answer is right, have the circle say "That's correct!".



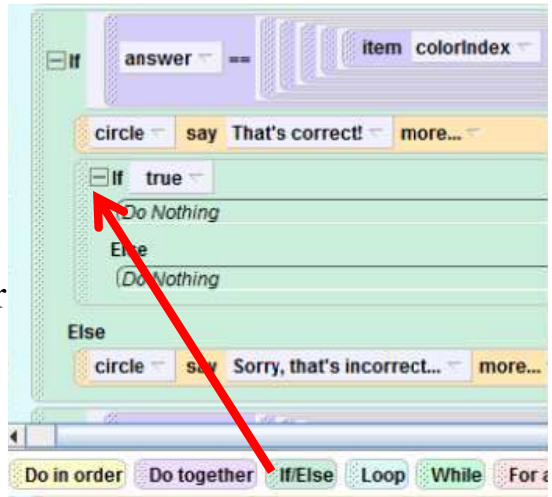

# Quiz




- Go to world's functions and drag the function "a != b" over the condition that you set the If/Else to. Right now choose any value for a (a = .25 in this picture) and let b equal 0.

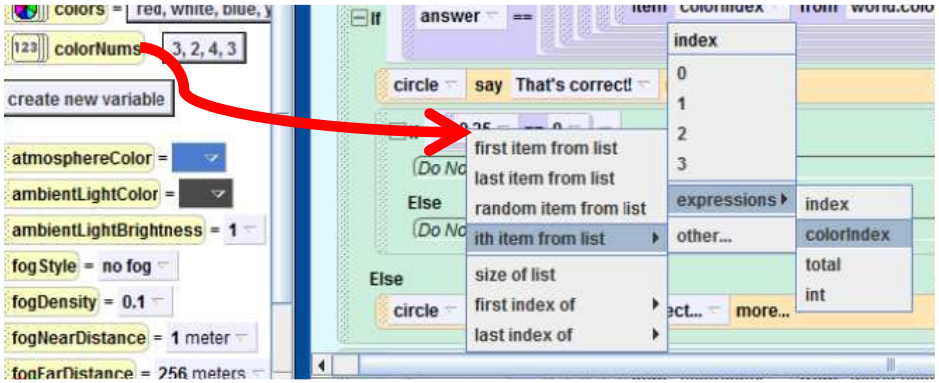
# Quiz

- If the user gets the answer right, the first thing we want to do is make sure that the number of balls of that color isn't 0. In the If portion, drag in another If/Else statement and select true for now.




# Quiz

- Click on the world's properties tab, and drag the *colorNums* list over the value that you chose for a and select "expressions" → *colorIndex* as the ith item from the list.

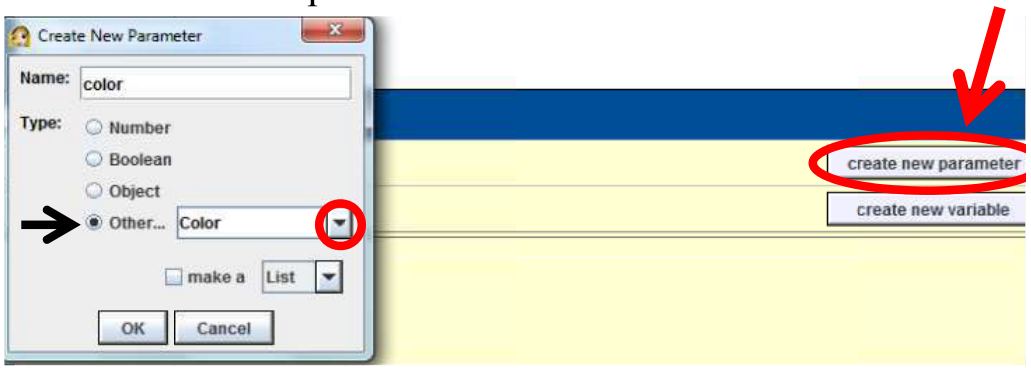


# Quiz

- If the number of balls of the current color is not 0, then we will want to animate the process of picking the ball out (which we will do in another method), decrement the value of the number of balls for that color in *colorNum*, and decrement the total number of balls by one. Otherwise, we do not really want to do any of those things.

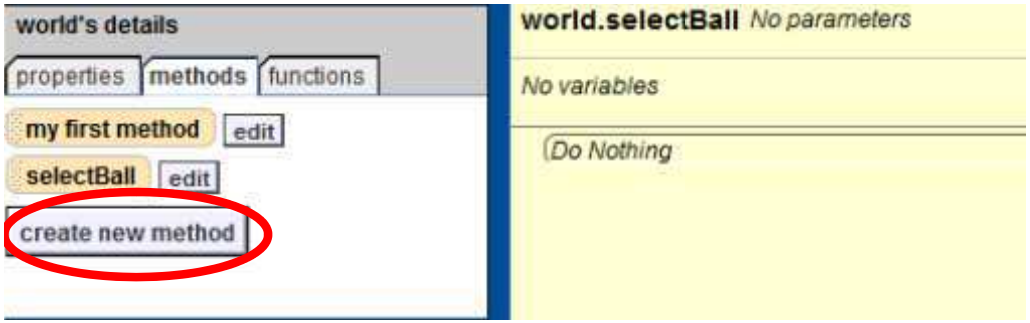
# world.selectBall

- For this method, we will need to pass in a parameter, or information from the world, so that it will know which color ball to take out. Click on "create new parameter" on the right side of the method and make a new Color parameter and name it *color*.




# world.selectBall

- Go to world's methods and create a new method called *selectBall*.

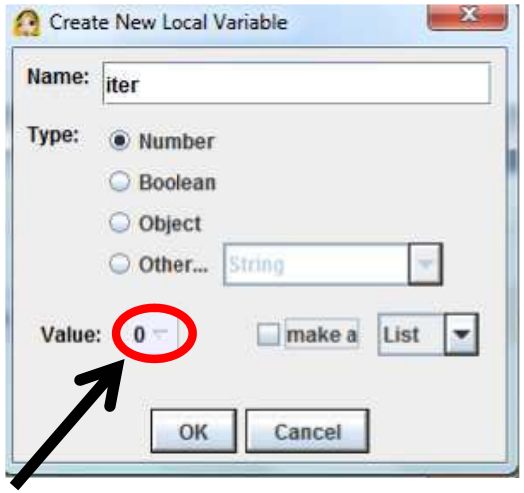



# world.selectBall

- The first thing that we want to do in this method is to change the camera view to the position that we saved earlier. Go to the camera's methods and drag in the method "camera set point of view to" and then select "Dummy Objects" → "choose" as the object.

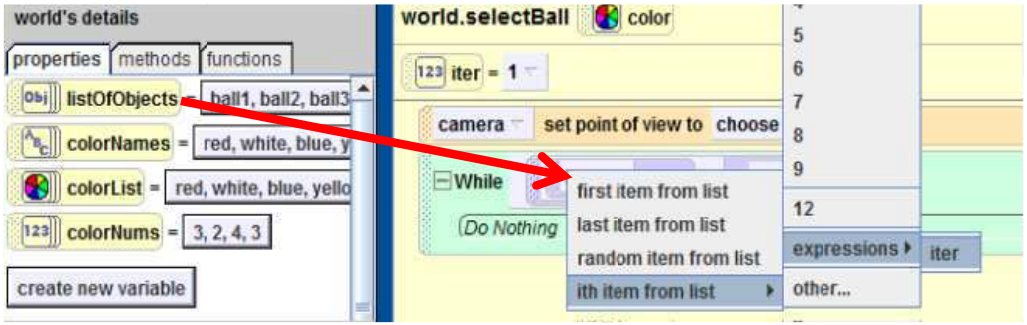
# world.selectBall

- Now, we want to go through each object in the *listOfObjects* until we find a ball that is the right color. To do this, we will need a temporary number variable to iterate through the list. Create a new variable named *iter* and initialize its value at 0.

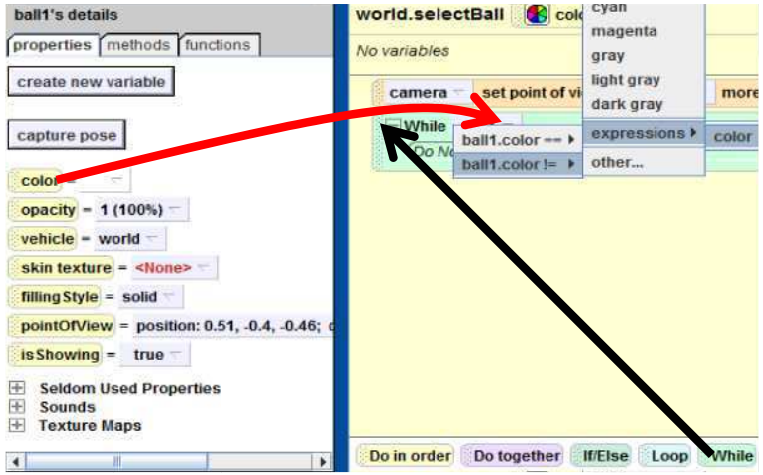



# world.selectBall

- Now, go to world's properties and drag our *listOfObjects* over ball1 and choose "ith item from list" → "expressions" → *iter*.



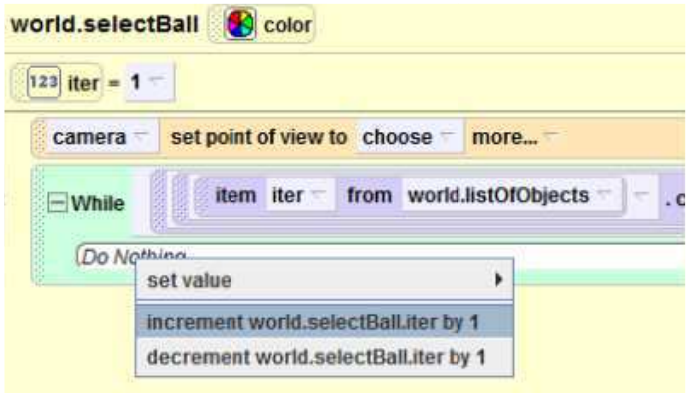

# world.selectBall




- Drag a While loop from the bottom of the method editor into the method and set the condition to true for now. Then, go to any of the ball1's properties and drag their color property over the true and select "ball1.color !=" → "expressions" → *color*.

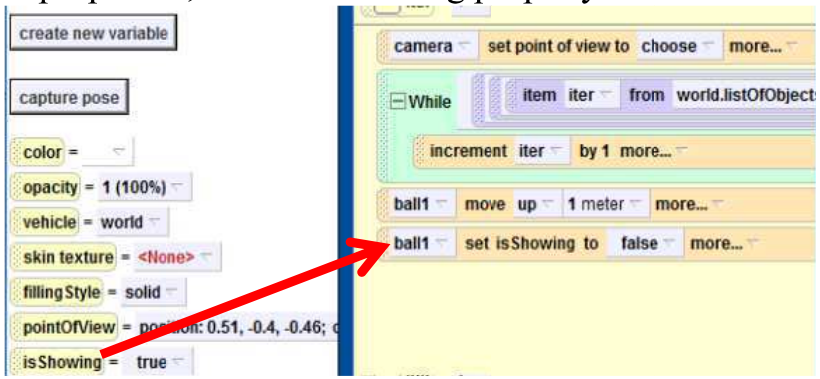
# world.selectBall

- While the ball at position *iter* is not the right color, we want to increment *iter* by 1 to check the next position. To do this, just drag *iter* into the While loop and select "increment world.selectBall.iter by 1".



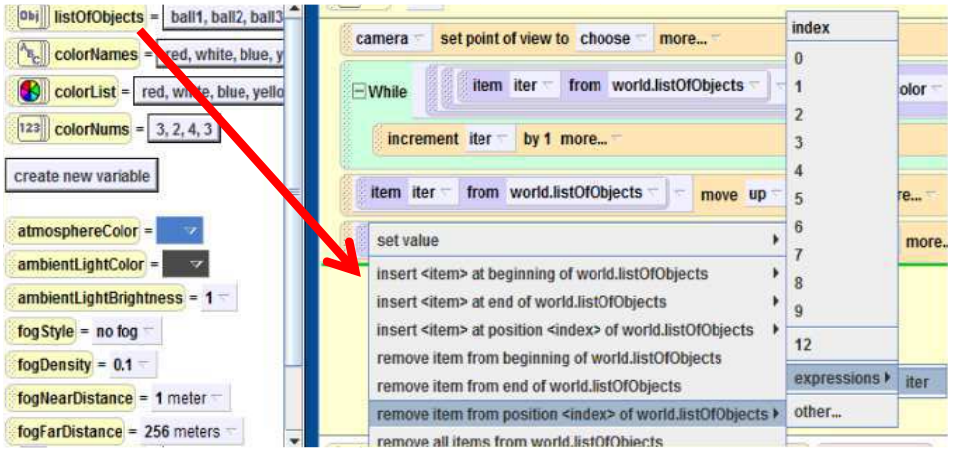
# world.selectBall

- As soon as we find a ball that is the right color, we want that ball to move out of the hole in the ground, become invisible, and then delete that ball from our list. To do this, first go to ball1's methods and have ball1 move up 1 meter outside of the While loop. Then, under ball1's properties, set the isShowing property to false below that.



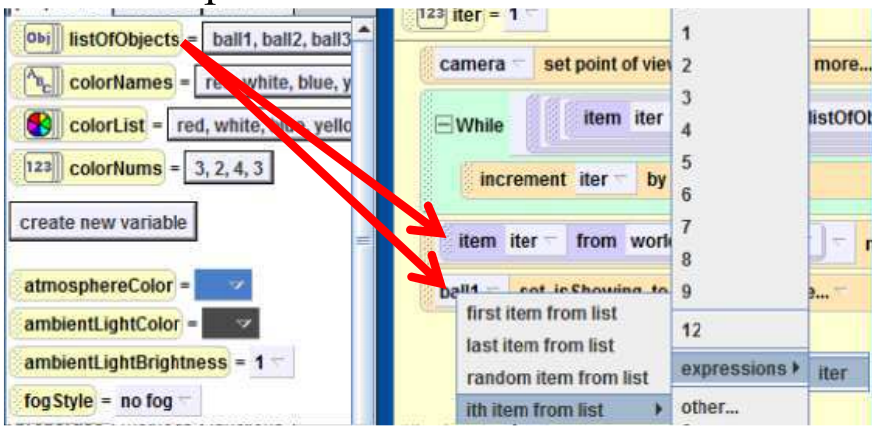

# world.selectBall

- To remove an object from a list, drag the *listOfObjects* to the bottom of the method and choose "remove item from position <index> of world.listOfObjects" and select *iter* as the index.



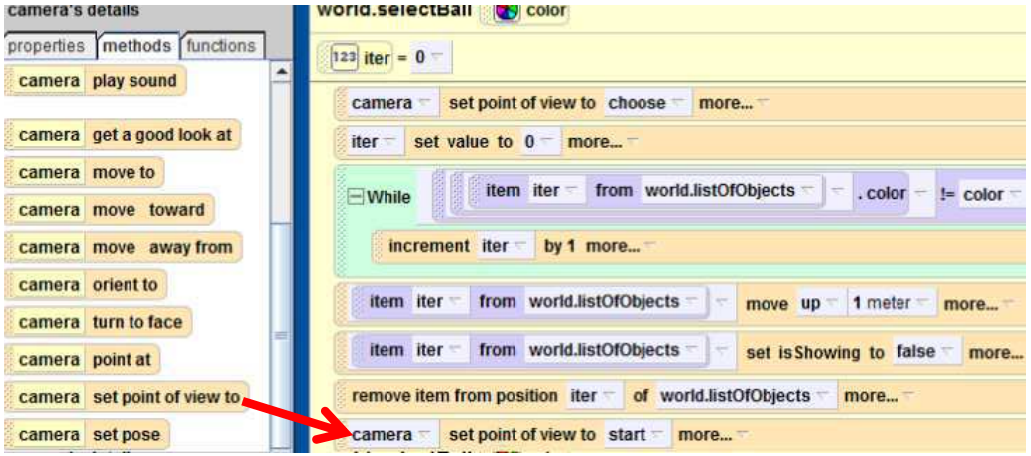

# Quiz

- Go to the world's properties and drag *listOfObjects* over both instances of ball1 that we just added and choose *iter* as the index under "expressions".




# world.selectBall

- The last thing that we need to do in this method is set the camera view back to the "start" position under "Dummy Objects". Here's the final code for this method.

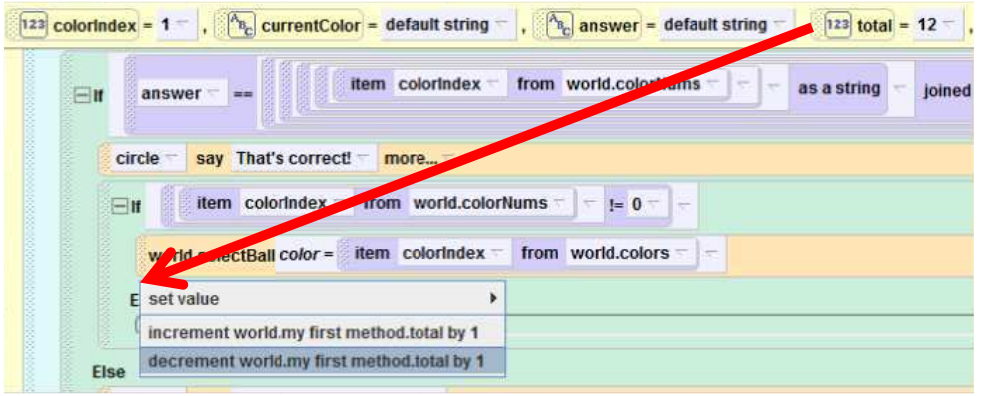


# world.my first method

- Now, we should add world.selectBall into our main method. Under world's methods, you should see world.selectBall. Drag the method into the second If statement and pass in any color for now.

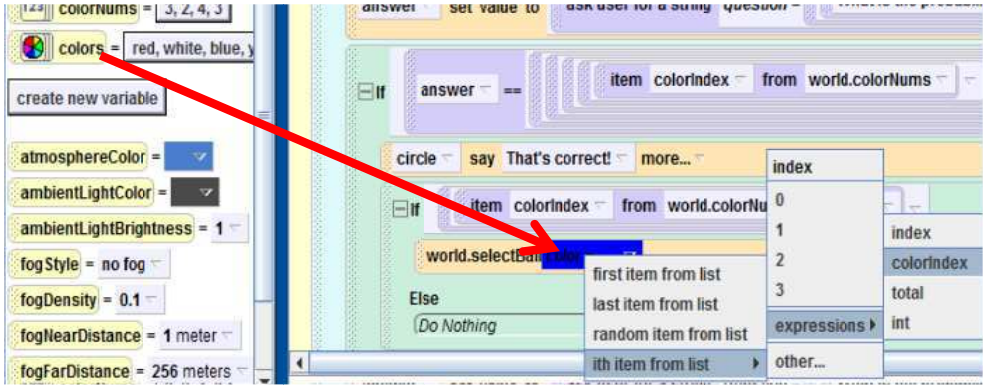

# world.my first method

- Next, drag the *total* variable just below the method call for world.selectBall and choose "decrement world.my first method.total by 1" to subtract one from the total number of balls.



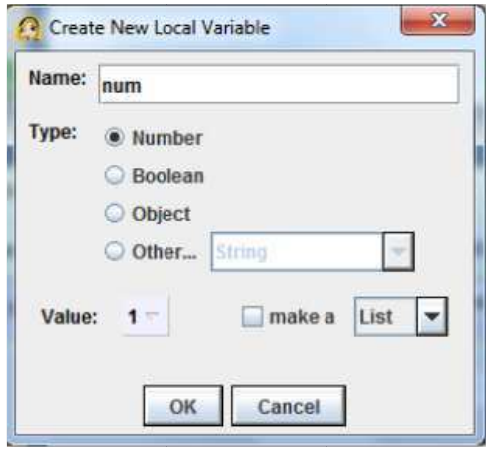

# world.my first method

- To get the color we want, go to the *colors* list under world's properties and drag it over the random color that you chose and choose *colorIndex* as the ith item from list.



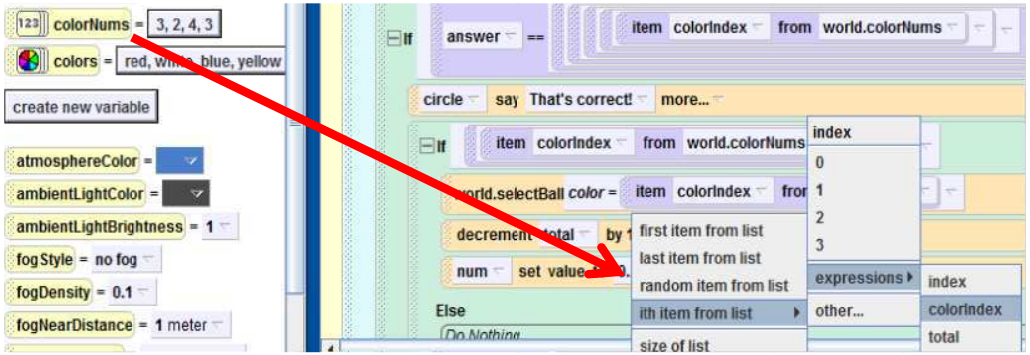

# world.my first method

- Now in world.my first method we want to decrement the number of balls for the specific color that was chosen. To do this, we will need to create a placeholder number variable and call it *num*.

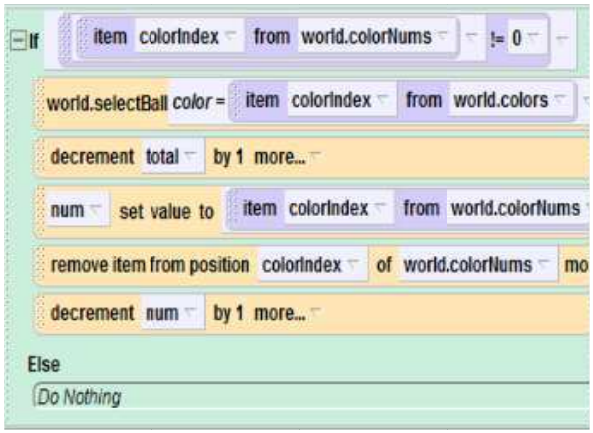


# world.my first method

- Drag the *num* variable right below where we decremented *total* and set the value to any number. Then, under world properties, drag *colorNums* over the number and choose *colorIndex* as the ith item from list.



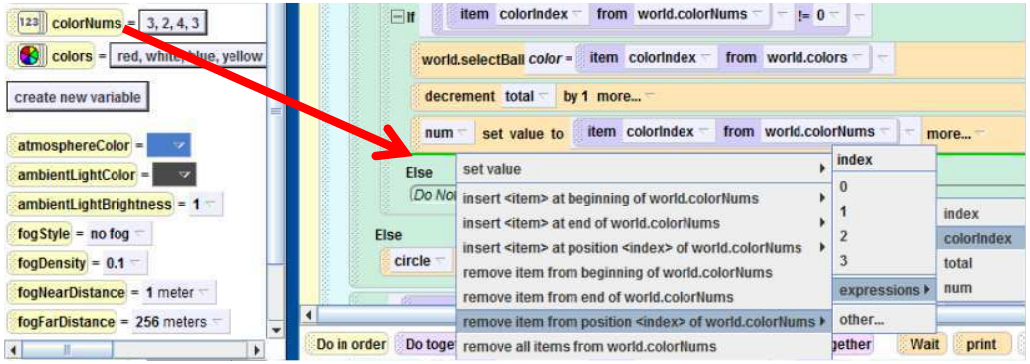

# world.my first method




- To decrement the value, just drag the *num* variable below the last instruction and select "decrement world.my first method.num by 1".

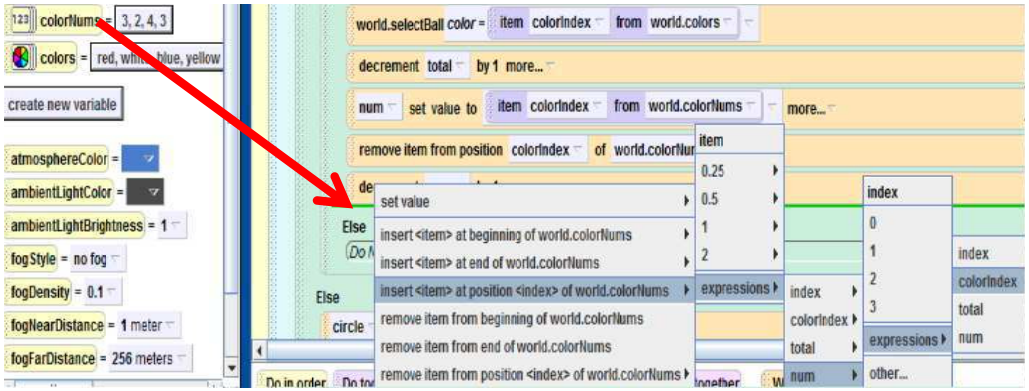
# world.my first method

- Next we want to remove the number of balls of *currentColor* from the list, decrement the value, and then put it back into the list at the same position. To remove the number, drag in the *colorNums* list and select "remove item from position <index> of world.colorNums"→ "expressions" →*colorIndex*




# world.my first method

- Now drag *colorNums* into the method and this time select "insert <item> at position <index> of world.colorNums" → "expressions"→*num* → "expressions" →*colorIndex* to put the new value in the *colorIndex* position.

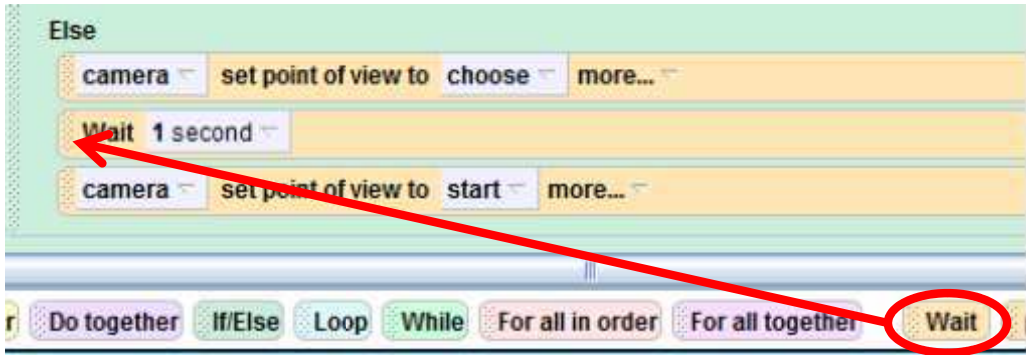


# world.my first method

- That concludes the actions that we need to do if the number of balls of a certain color is more that 0, but what about if there are no balls of that color left? We just want the camera to move to the choose position, wait 1 second, and then move back.

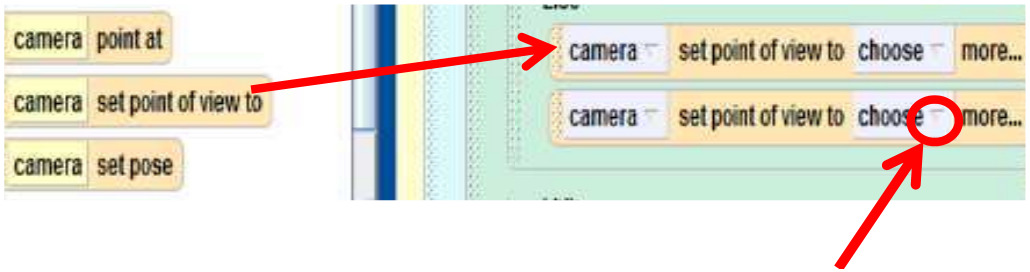
# world.my first method

- Finally, in between the two "camera set point of view to" methods, drag in a Wait instruction from the bottom of the method editor and set the duration to 1 second. The entire method code can be seen on the next 2 slides.



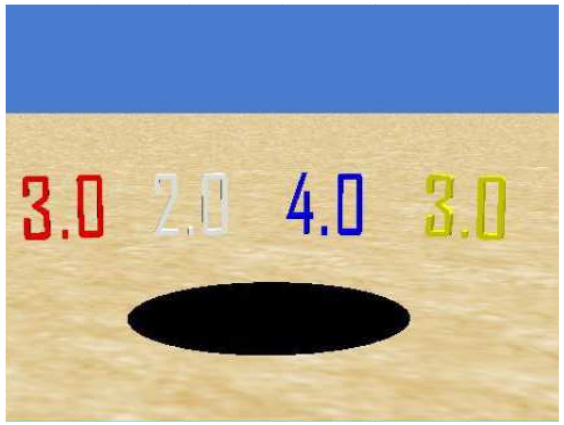

# world.my first method

- Go to camera's methods drag "camera set point of view to" in the empty Else section. Select the choose view from Dummy Objects. Right-click on this line and make a copy of it. In the copy, go back to the object to set the point of view to, go to Dummy Objects and this time choose the start view.



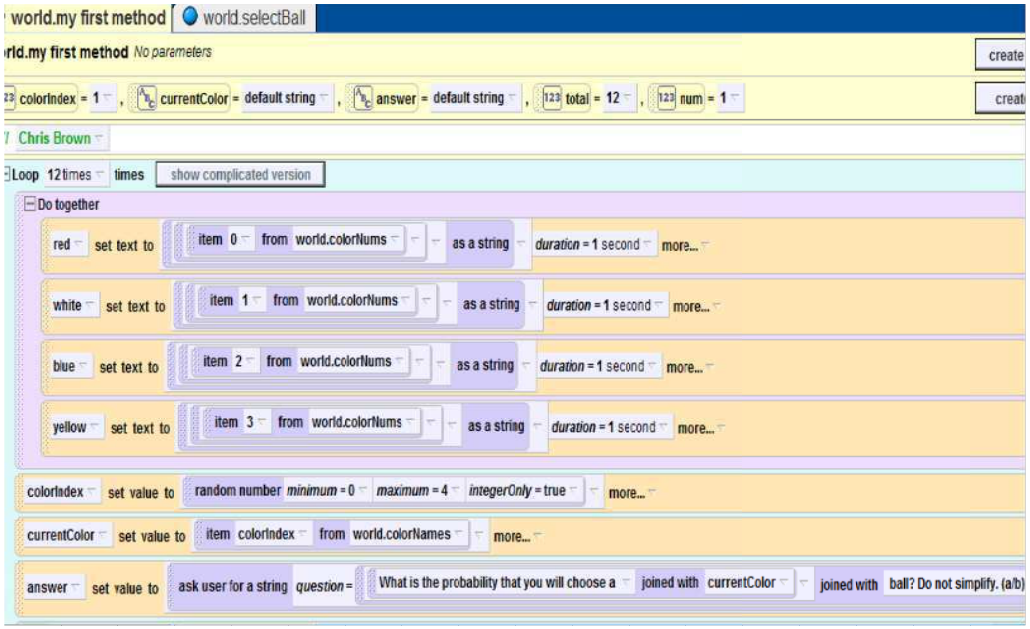

# Play your World

- Play your world to test it.




- The world should work, but notice that the 3D text objects display the decimals rather than the number itself. Use the "int as a" function from earlier to fix this.

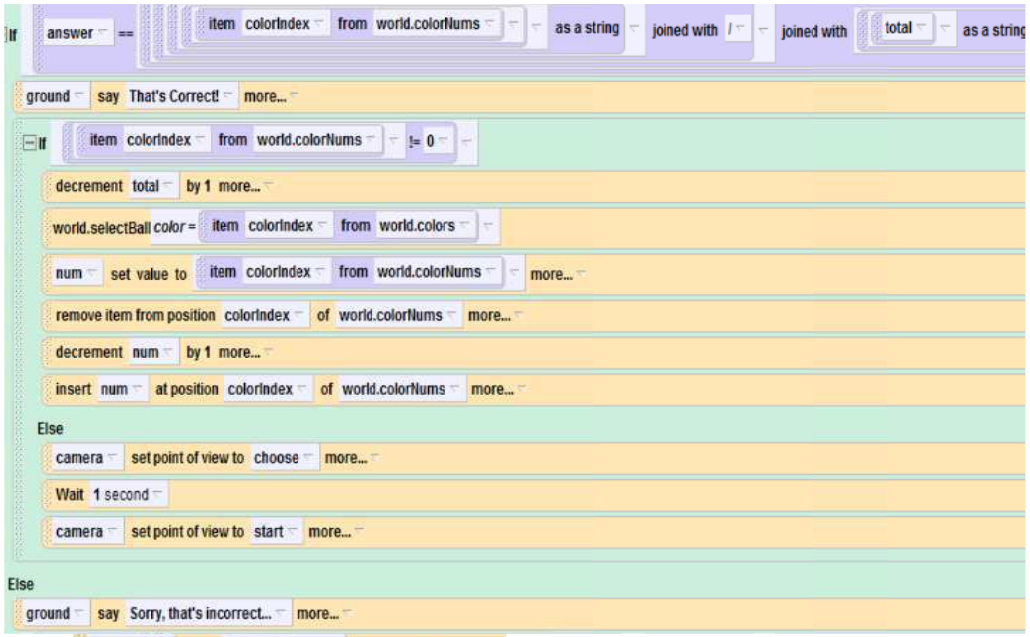
# world.my first method




# Probability Game

- And that concludes the probability game! Try playing your world to see how it turned out. You can also try different things such as changing the amount of balls, the colors, the number of balls for each color, the objects to be used, etc. to help your students get a better understanding of probability.

# world.my first method




- Outside of the loop.

# Challenge

- Add a billboard with instructions for the game.
- Try modifying this probability game world so that there are different colored balls and a different number of total balls in the hole for the user to calculate. Don't forget to also change the number of times the user is asked in the loop!

## Appendix 3: Challenges

This appendix contains all of our Math Challenges. These are Alice worlds where students must fill in a function or a specific part of an Alice world in order to complete it so that it works properly. For the challenges, students will not need to be introduced to everything about Alice programming, but they will only need to know specific topics relevant to the challenge.

# Boat Racing Game Challenge #1


By Chris Brown

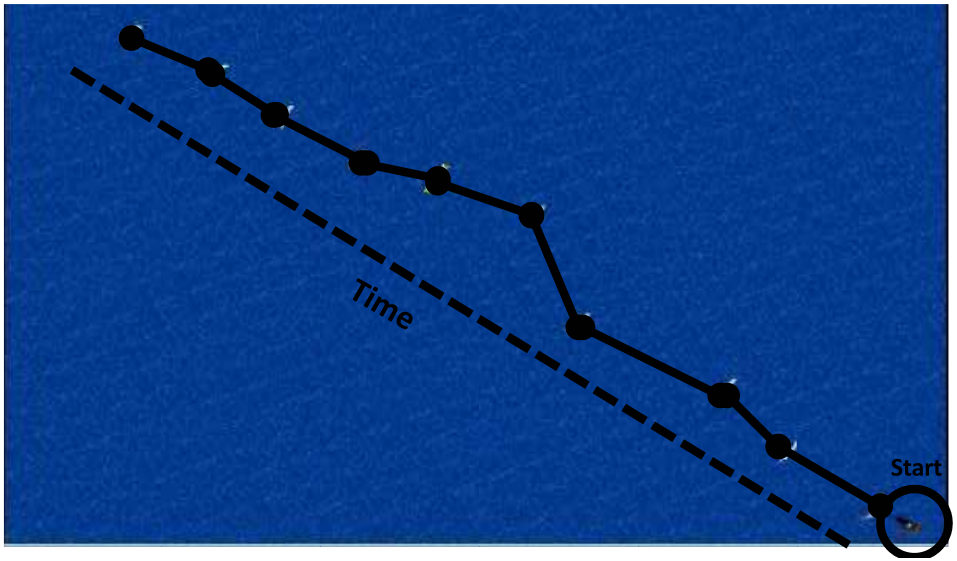
Under the direction of Professor Susan Rodger

Duke University, January 2013

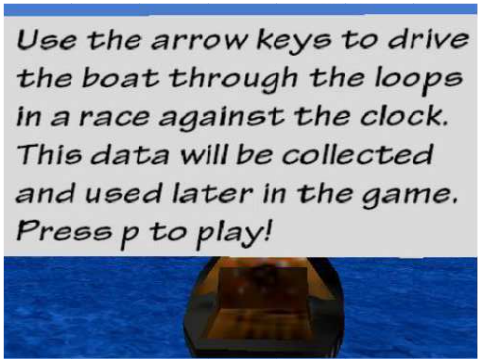
Based off of the Boat Racing Game by Jenna Hayes

# Average Time

- We want to find the average time for each arch in this game.




# Boat Race




- In this world, you must control a boat to travel through the 10 arches in the race course in order to win the game. The faster your time, the better you will do! We want you to modify this game so that you will know the average time your boat travelled between each pair of arches when the race ends.

# Challenge

- For this challenge in the boat race world, you will need to complete the “average” function to calculate the average time of the boat between a pair of arches that it had to travel through and modify the “win” method to display the average speed between pairs of arches to the user once they have completed the game.

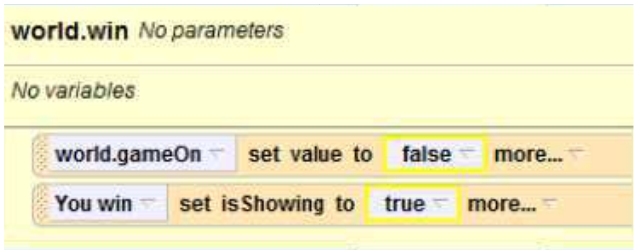
# world.average

- You are given the total time that the game took and you need to calculate the average amount of time/arch.


- Hint: There are 10 total arches in the game.

# world.win

- Now, modify this method so that the text that once displayed the timer displays the average time of the boat. Add other methods and explore Alice to see what you want to add to this, such as having the boat explain ("say") what the value that you computed represents.



# Boat Racing Game Challenge #2


By Chris Brown

Under the direction of Professor Susan Rodger

Duke University, January 2013

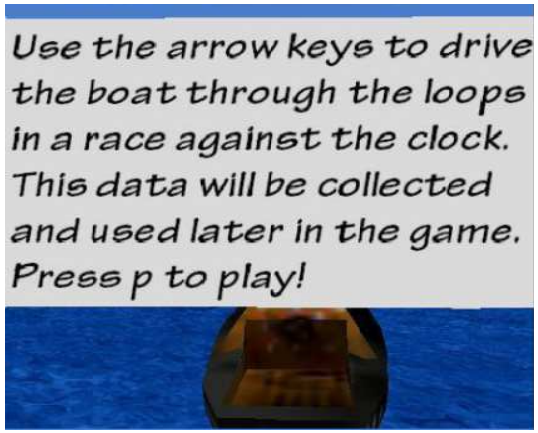
Based off of the Boat Racing Game by Jenna Hayes

# Challenge

- For this version of the boat race world challenge, you will need to complete the "average" function to calculate the average distance that the boat travels to go through each hoop in the game, and then modify the "win" method to display the average speed to the user once they have completed the game. The program is already set up to collect the distances between each arch, you will use those to calculate the total distance between each of the arches that you must pass through.

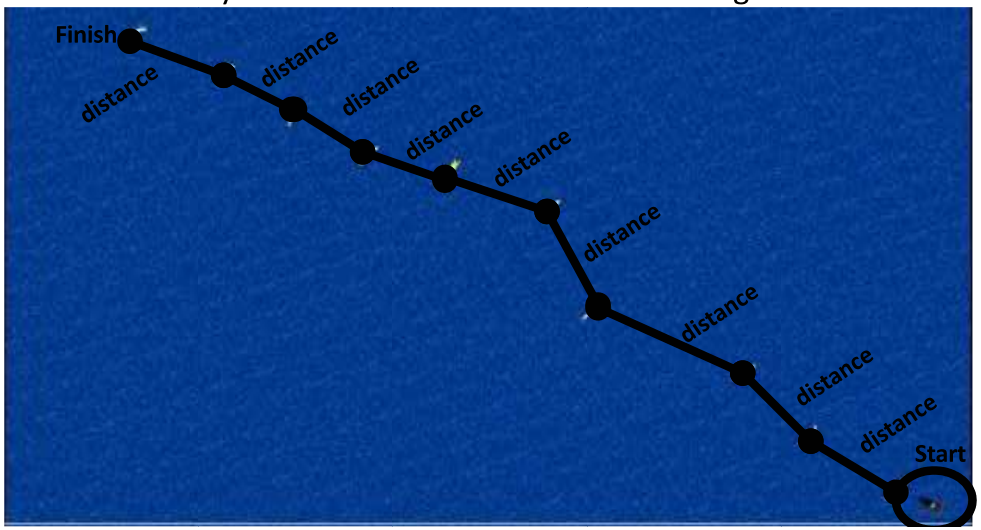
# Boat Race

- In this challenge, you must drive the boat to travel through 10 arches in a race course in order to win the game. The faster your time, the better you will do! To make things more difficult, the arches are placed in a random position each time you run this game, so no two courses will be alike. We want you to add to this game, so that in the end you will know the average distance between arches.




# Average Distance

- We want to calculate the average distance between pairs of arches. The code provides a list of this information called *distances* for you to use to calculate the final average.



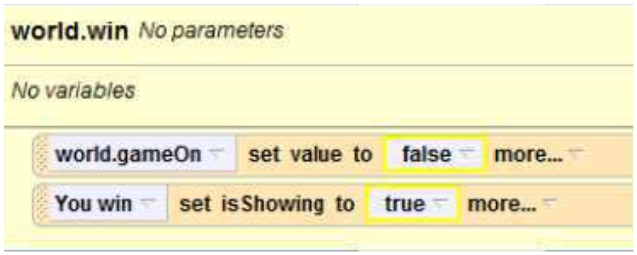
# world.average

- You are given a list of the distances between pairs of arches and you will need to calculate the average distance between two arches (meters/arch) it took the boat to finish the game.


- Hint: There are 10 total arches in the game and the boat starts a short distance before the first arch.

# world.win

- Now, modify this method so that the text that once displayed the timer displays the average distance the boat travelled between arches. Once again, feel free to explore Alice and add other aspects to the winning screen of the game. For example, have a fish come up out of the water to congratulate you on your victory!



# Boat Racing Game
# Challenge #3


By Chris Brown

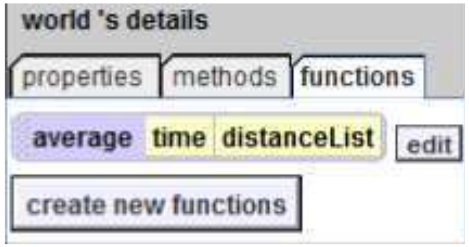
Under the direction of Professor Susan Rodger

Duke University, January 2013

Based off of the Boat Racing Game by Jenna Hayes

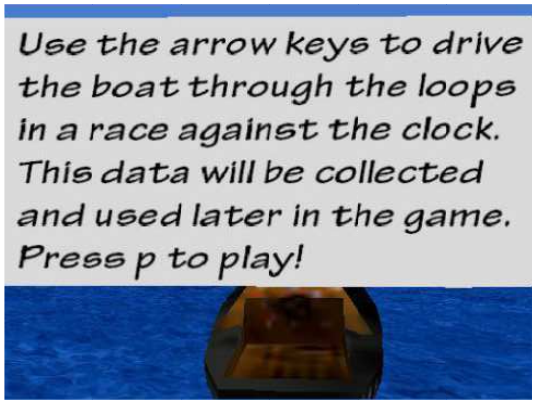
# Challenge

- In this challenge, you will need to complete the "average" function to calculate the speed of the boat over time in meters per second, and then modify the "win" method to display the speed to the user once they have completed the game.



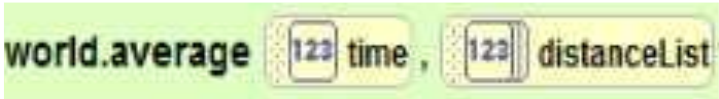

# Boat Race

- In this world, you must control a boat to travel through the 10 arches in the race course in order to win the game. The faster your time, the better you will do! We want you to add to this game, so that in the end you will know the speed your boat travelled throughout the race.



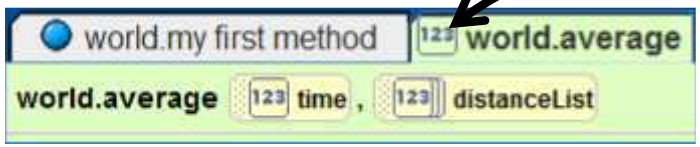

# world.average

- When you play the game, we've already written the code to complete the total time and collected the distances between each pair of arches in a list. You will need to fill in the *average* function to compute the speed. In this function, you are given the total time that the game took and a list of the distances in meters between each hoop as parameters.



# world.average

- First you will need to sum up the values in the list to calculate the total distance*, then divide that by the time it took to complete the run through the arches.

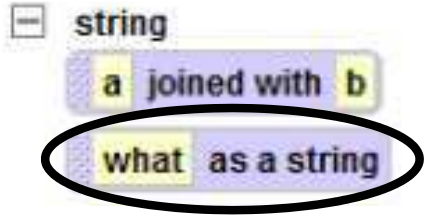



This means that your function will return a number!

*Hint: Use a loop and create a new variable to find the sum of the values in a list!

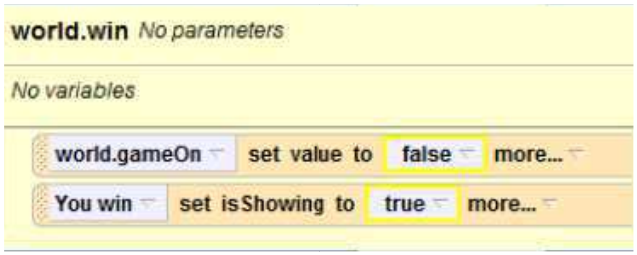
# world.win

- world.gameOn is just a variable that is true whenever the game is running and false when the game is over. To display a number as text, you will need to use the “what as a string” function under the string section of the Alice world functions.




# world.win

- Now, modify this method so that at the end of the game, the text that once displayed the timer displays the speed (meters/second) the boat travelled throughout the game. Feel free to use other Alice methods to creatively show the data after the game has been won, for example, making the boat do a backflip!



# Boat Racing Game Challenge #4


By Chris Brown

Under the direction of Professor Susan Rodger

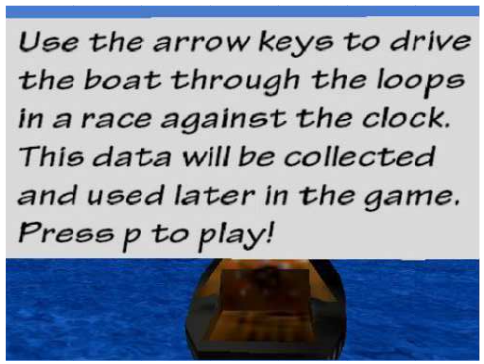
Duke University, January 2013

Based off of the Boat Racing Game by Jenna Hayes

# Average Time

- We want to find the average time it takes you to complete one game out of as many games you decide to play. Every time you finish a race, Alice will ask you if you want to play again. If you select “Yes”, then the values from the race you just completed will be saved. If you click “No”, then we want Alice to display the average time it took you to finish the number of games you played.

# Boat Race




- In this world, you must control a boat to travel through the 10 arches in the race course in order to win the game. Try to complete the race as fast as possible! We want you to modify this game so that you will know the average time it takes you to complete one game over multiple games.

# Average Time

# Challenge

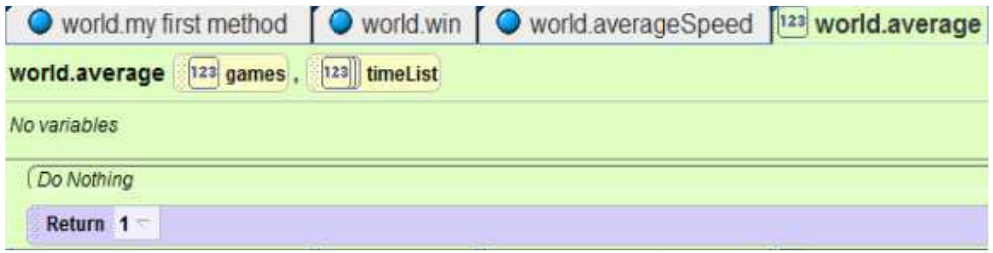
- This challenge is a more advanced version of Challenge #1. In the boat race world, you will need to complete the "average" function to calculate the average time it takes you to complete a series of games. You will also need to modify the "win" method to prompt the user and ask if they want to play again, then display the average time it took them to completed the game when they are done playing.

# Complete world.win

- First, we will need to change the world variable playAgain to see if the player wants to try the game again to get a better score. To do this, go under world functions and set playAgain to the function "*ask user for yes or no*".


# Complete world.average

- You will be given timeList, a list of the total times that each game took along with games, a variable to keep track of the number of games. You will need to calculate the average amount of time per game.




# Complete world.win (Part 2)

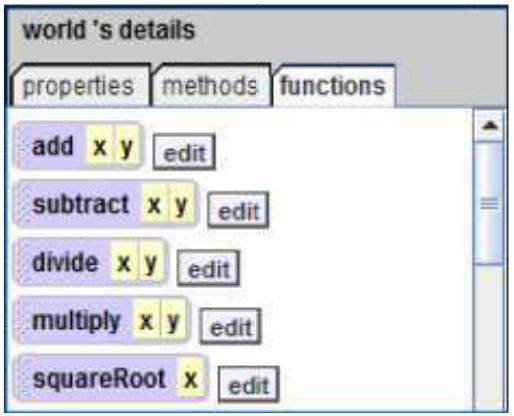
- Now, add to this method so that if playAgain is *true*, the world will add the current time to the timeList and increment the number of games played, then reset everything to start the game over. Make sure to use the reset method already created for you. If playAgain is *false*, then have the boat say what the average time was for all of the games out of how many games you played (i.e. "You completed the game in an average of _____ seconds for _____ games."), and add your own animation at the end.

# Calculator Challenge

By: Chris Brown
Under the direction of Professor Susan Rodger
Duke University, January 2013

# 1. Starting Functions Challenge

- Functions are chunks of code that return a value when they are called. All of the functions that you need have already been built, but right now they don't do anything except return the value 1. Finish the program by filling in these functions using Alice built-in functions so that they will return the correct values and make the calculator run correctly!

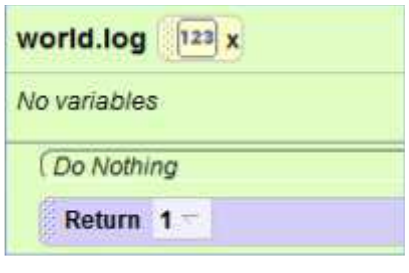



# Introduction

- In this world, you will be able to use Alice as a calculator to evaluate expressions. However, it is not complete. It is your job to create the functions that you want your calculator to be able to accomplish so that when you enter the values and click a button on the calculator, the program will know what to do. You will be able to enter the numbers that you want to use, and click on "CE" to clear the screen and "ENT" to compute the final answer when you are finished.

# 2. Log Challenge

- If you look through the Alice world advanced math functions, you will see that there is not a function to calculate the logarithm of a number in base 10 (natural log is base *e*). You will need to find a formula to calculate the log base 10 to put into this function.



# 3. Exponents Challenge

- Right now, our calculator does not do exponents. You will need to add a button, so that when it is clicked the calculator will know to raise the value to a certain power. The button can be a new billboard object, and you will have to create a new function with parameters as well as an event to run the function when the button is clicked.

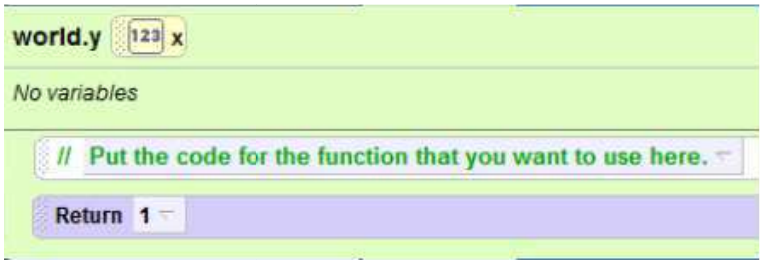
$a^b$

# Other Challenge Ideas

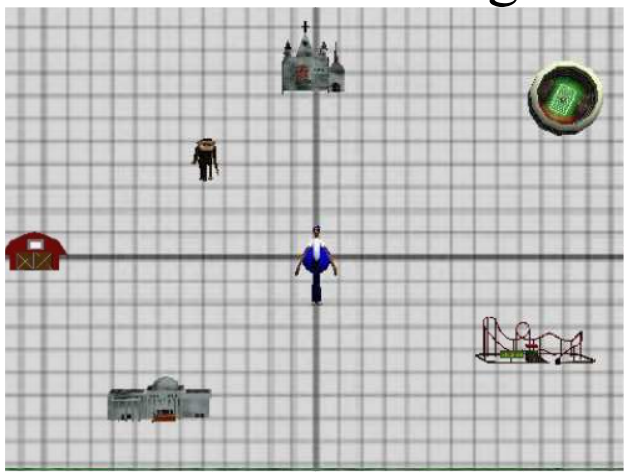
- Use loops to implement the exponent challenge, rather than the "a raised to the b power" function.
- Pretend that multiplication does not exist. Use loops and addition to create the same affects of multiplying two numbers together.
- Make a function to calculate the factorial of a number. (6 factorial = 6! = 6*5*4*3*2*1)
- Build a +/- button to easily switch between positive and negative numbers.
- Create other buttons to go with all of the Alice's advanced math functions (cos, sin, tan,…) and add to your calculator.

# 4. General Function Challenge

- Create a special button, *y*, that computes the value of a function you specify and you can plug in any value for x to solve for y. For example, if you specify in world.y the function $y = x^2 - 3x + 10$ and set the value of x to be 5, this function computes 20, which means y = 20.



# Distance Challenge

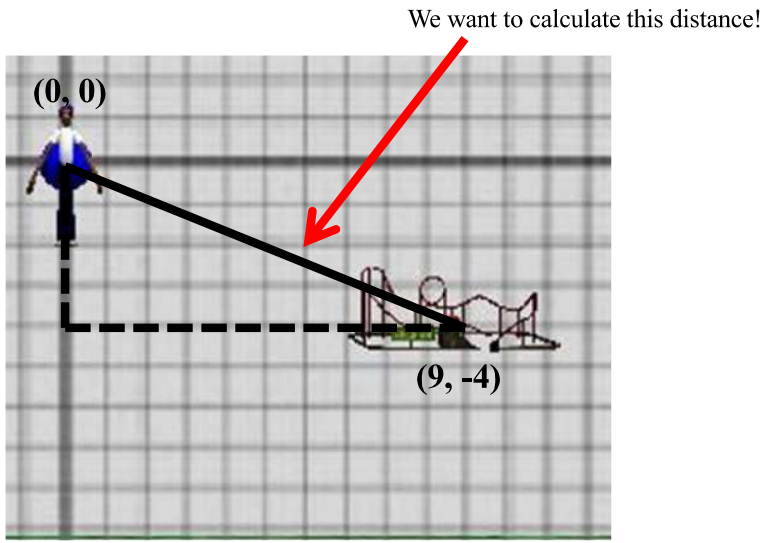


By: Chris Brown

Under the direction of Prof. Susan Rodger

Duke University, January 2013

# Problem

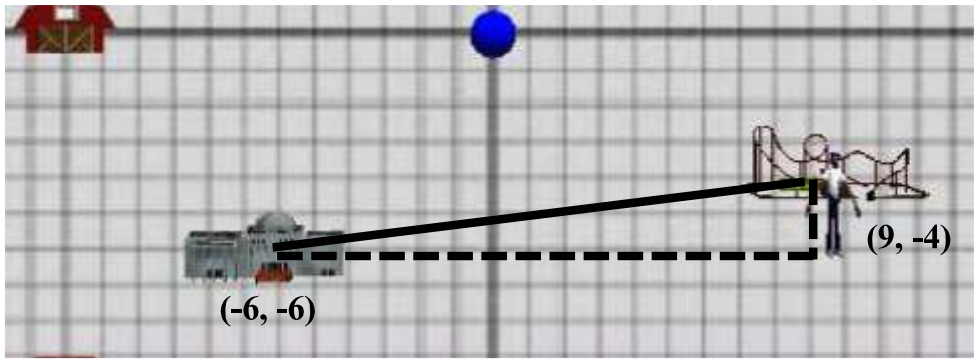



# Problem

- It's Jimmy's first time in a new city, and we need to help him find his way around! He knows where he is located in the center at (0, 0) to start, and he also knows the coordinates of each place he wants to visit. Your job will be to fill in the distance function, to calculate the distance from Jimmy to his destination. You can move around the city by clicking on the different places of interest that you want to visit.
- 1 unit = 1 meter

# Problem

- Now, Jimmy wants to go from the theme park to the museum…



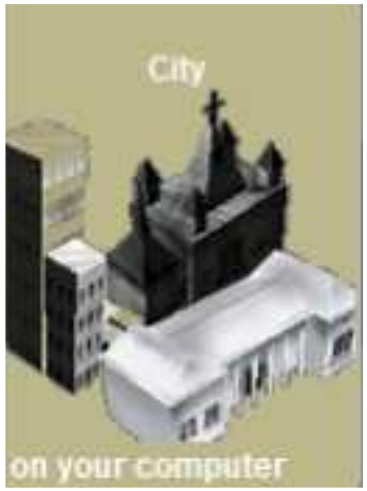
# world.distance

- Each time the distance function is called, four parameters are passed in with information you will need. It provides you with Jimmy's x value, Jimmy's y value, the destination's x value, and the destination's y value. It is your job to correctly return the distance from Jimmy to his destination in this function.

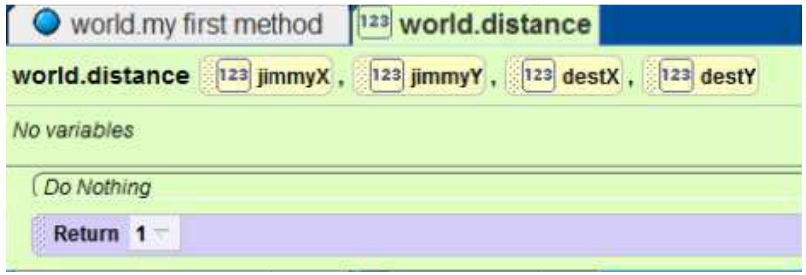
- Hint: What is the distance formula? It can be derived from Pythagorean's Theorem.

# Bonus Challenge!

- Go into the Alice Object Gallery to add new places, with an original animation every time Jimmy visits that place. Make sure to create variables for your location's x and y position to be passed into your distance function later on.




# world.distance




Fill this in!

## Appendix 4: SIGCSE poster

This appendix contains the "Integrating Computer Science into Middle School Mathematics" poster that we presented at the ACM's Special Interest Group on Computer Science Education (SIGCSE) conference on March 8, 2013 in Denver, CO. It contains all of the work we completed up to that point and displayed the math materials and resources we have created for students and teachers to use.

# Integrating Computer Science into Middle School Mathematics

SIGCSE 2013
Denver, CO
March 8, 2013

Supported by NSF Grants DRL-1031351, CRA, and IBM Faculty Awards

Susan H. Rodger, Dwayne C. Brown, Jr., Michael Hoyle, and Michael Marion  Duke University, Durham , NC  USA

Our project is part of the Adventures in Alice Programming Project at Duke University. In particular, our project is integrating computer science into middle school math using Alice. We show several ways for students to improve their math skills while engaging their interest in programming.

## Goal

- Integrate computing into middle school mathematics
  - Students improve math skills
  - Students learn about computing

## Implementation

- Teach programming with Alice
  - Developed Alice tutorials on programming concepts and animation concepts
  - Tutorials for sample projects
- Integrate math and programming
  - Alice worlds to practice math concepts
  - Tutorials to build such worlds
  - Math challenge worlds

## Adventures in Alice Programming

- 2-week Teacher workshops
  - Over 200 teachers, middle school, high school, some elementary
  - All disciplines
  - Teach Alice, Develop Lesson Plans
  - One-week follow-up workshop
  - Summers 2008-2015, funding for lodging
- Main Sites:
  - Duke University, Durham, NC
  - Charleston/Columbia, SC
  - San Jose, California

## Adventures in Alice Programming web site

www.cs.duke.edu/csed/alice/aliceInSchools


## Free Curriculum Materials/Lesson plans

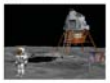
- Over 60 free Alice Tutorials (from getting started to specific topics, sample projects)
- Teacher lesson plans available
- Most use Alice for projects – instead of poster, report
- Subject teachers using Alice
  - Language Arts
  - Mathematics
  - Science
  - History
  - Foreign Language
  - Music, Art
  - Media, Technology
  - Business
- middle school and high school, some elementary

## Example: Getting Started Tutorial teaches:

- Placing objects
- Moving objects
- Setting up Camera tripods and moving between views
- Using built in methods and writing your own
- Gluing objects together
- Adding sound, 2D pictures to enhance world

## Create Animations, Interactive Stories and Games with Alice

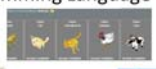
- Alice is a 3D virtual worlds programming environment
  - Hands-on!
  - Interactive!
  - Visual!
  - Drag and Drop!
  - Less Error prone!
  - Exciting Results right away!



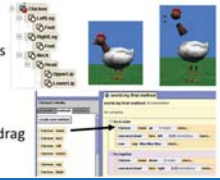
- Alice has the potential to excite kids about computer science in the same way that experiments excite kids about chemistry, physics and biology!
- Students learn problem solving, logic and critical thinking skills
- Developed by Randy Pausch , CMU – Alice is Free! alice.org

## Alice Programming Language

- Has libraries of 3D objects



- Objects have multiple parts that are moveable



- Select code, drag and drop

## Operations on Fractions

Press 'A' for addition
Press 'S' for subtraction
Press 'M' for multiplication
Press 'D' for division


## Order of Operations

Click on the operation that happens next.
Then compute the value of that operation.



## Asteroids Game


## Tutorial for Astroids Game

- We're going to re-create the 1979 Atari classic video game "Asteroids" in Alice.
- In the game, you pilot a ship around in any direction and use the spacebar to shoot a laser at incoming asteroids.
- The incoming asteroids vary in size and speed and fly in from off the screen.
- If you hit all of the asteroids, you win! If you crash into one, you lose.

## Asteroid Game Challenges

- Can you create a score object that counts the number of asteroids that have been hit?
- Can you create a billboard with instructions for the game? Can you make it disappear when the game starts?
- Can you create 3D text that appears when you win or lose that tells you whether you've won or lost?

## Calculator

In this world, you will be able to use Alice as a calculator to evaluate expressions. However, it is not complete. It is your job to create the functions that you want your calculator to be able to accomplish so that when you enter the values and click a button on the calculator, the program will know what to do. You will be able to enter the numbers that you want to use, and click on "CE" to clear the screen and "ENT" to compute the final answer when you are finished.

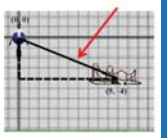

## Exponents Challenge

- Right now, our calculator does not do exponents. You will need to add a button, so that when it is clicked the calculator will know to raise the value to a certain power. The button can be a new billboard object, and you will have to create a new function with parameters as well as an event to run the function when the button is clicked.

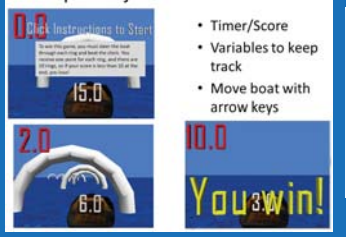
$a^b$

## Math Challenge – Calculate Distances

- Compute the distance between the boy and the places to visit



## Sample Project - Boat Race Game

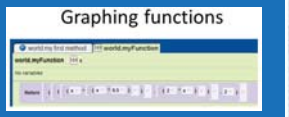


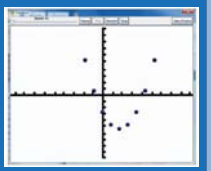

- Timer/Score
- Variables to keep track
- Move boat with arrow keys

## Math Challenges with Boat Race

- Add on after completing the boat race
  - Modify the race to compute the average speed of the boat as it travels through the hoops.
  - Modify the game to calculate the average distance between hoops
  - Modify the race to run several times and calculate the average race time

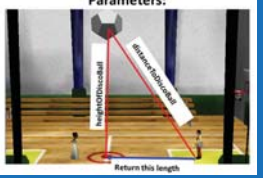
## Graphing functions





## Trig Prom Challenge

**Parameters:**




## Trig Prom Challenge

**Story:**

Fred told his prom date he would meet her under the disco ball at midnight for a dance. Now the time has come but he needs your help to find her!

Use what you know about trigonometry to get him to his date!