Integrating Computing into K-12 Education using Alice

by

Elizabeth Liang

under the supervision of Dr. Susan Rodger,
Department of Computer Science, Duke University

May 2011

__________________________  
Susan Rodger, Advisor

__________________________  
Jeffrey Forbes

__________________________  
Robert Duvall

Honors Thesis submitted in partial fulfillment of the requirements for graduation with Distinction in Computer Science in Trinity College of Duke University.
Abstract

In an effort to integrate computing concepts into K-12 education, a practical solution is to integrate computing concepts into teachers' existing lessons. Using Alice, teachers can add interactive animation components to their lessons and students, instead of doing traditional projects such as reports and posters, can learn computing while creating an Alice world as their project. This research focuses on creating resources for students and teachers to learn Alice in a context that makes it easy to apply Alice to subjects already being taught at school, particularly science. In middle school science, Alice can be used as a modeling tool to teach concepts that are traditionally hard to demonstrate through physical experiments for practical reasons. Alice can also be used to generate data visualizations, which are an integral part of the science curriculum in North Carolina. While developing these resources, we solicited the input from teachers across the state and then outlined possible steps for further research with the ultimate goal of using Alice to integrate computing into the science classroom.
Table of Contents

Abstract ................................................................................................................................. 1

1. Introduction & Background ............................................................................................ 4
   1.1. Previous Work ........................................................................................................... 5
   1.2. Alice Programming Adventures .............................................................................. 6

2. Methods & Results .......................................................................................................... 7
   2.1. Developing Worlds and Tutorials .......................................................................... 7
       2.1.1 Beginning and Topical Tutorials ........................................................................ 7
       2.1.2 Projects ............................................................................................................. 8
       2.1.3 Example Worlds ............................................................................................... 9
       2.1.4 Tutorial Website ............................................................................................... 9
   2.2. Alice in Middle School Science ............................................................................. 9
       2.2.1 Lac Operon Model ........................................................................................... 10
       2.2.2 Surveying Science Teachers ........................................................................... 11
       2.2.3 Bar Chart Object ............................................................................................ 12
       2.2.4 Bar Chart Application ..................................................................................... 13

3. Discussion ...................................................................................................................... 14
   3.1 Conclusion ................................................................................................................ 14
   3.2 Future Work .............................................................................................................. 14

Bibliography ..................................................................................................................... 17
## Appendices

### Appendix A: Alice Tutorials & Projects

<table>
<thead>
<tr>
<th>Exhibit</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Princess and Dragon</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Light Tutorial</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>Alice Gallery Folders</td>
<td>61</td>
</tr>
<tr>
<td>4</td>
<td>Sharing Alice</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>Alice Pong</td>
<td>67</td>
</tr>
<tr>
<td>6</td>
<td>Treasure Hunt</td>
<td>80</td>
</tr>
</tbody>
</table>

### Appendix B: Example Science Worlds

<table>
<thead>
<tr>
<th>Exhibit</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hot Spot Volcano &amp; Helium Molecule</td>
<td>94</td>
</tr>
<tr>
<td>2</td>
<td>lac operon</td>
<td>95</td>
</tr>
<tr>
<td>3</td>
<td>Bar Chart World &amp; Tutorials</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>Atmosphere World</td>
<td>101</td>
</tr>
</tbody>
</table>
1. Introduction & Background

A report recently conducted by the Association for Computing Machinery (ACM) and the Computer Science Teachers Association (CSTA) revealed that computer science education in our K-12 classrooms is at a crisis [1]. Policy makers at all levels are seeking “to expand the capacity and quality of science, technology, engineering and mathematics education in the U.S.” because they recognize that computer science is changing the world and workforce and more importantly the daily lives of our students. Yet, most of the country has few or no computer science education standards and few consider computer science a ‘core subject’. Until the policy framework and precedent for quality computer science education is in place nationwide, educators interested in preparing students for computer science must turn to the core subjects.

There is an increasing need to combine teaching of traditional knowledge in the traditional core subjects (math, science, etc.) while at the same time imparting “technology literacy” [1] for the new technological world they live in. Many curriculum requirements in schools are being updated to require students to not only master a concept but also be proficient in areas of computing and lay a foundation for further studies in computer science [2]. A potential solution that has been shown to fill both these needs are Alice lesson plans [3]. Alice [4] is a 3D virtual worlds programming environment that is especially built to introduce computer science to those with little or no experience, and its storytelling aspect is particularly attractive to middle and high school girls. With Alice, students can easily create interactive stories and games (called worlds). Teachers take the content from their traditional lesson plans in the core subjects and update them to involve not only instruction with but the student’s own creation of an Alice world.
While there are alternative tools to expose students to computing such as Scratch [5] and using Robotics [6] to motivate children, our research focuses on Alice. There are various obstacles to using Alice including the need for resources in the school to catch up technologically to the demands of the teachers and students [7], Alice has been shown to be an effective teaching tool [8]. My project centers around developing resources to facilitate adoption of Alice in the K-12 classroom with a particular emphasis on middle school science.

1.1. Previous Work

Alice has been taught in middle and high school summer camps as well as in introductory computer science courses in an attempt to excite a future generation of computer scientists. Previous work in integrating computing into the K-12 classroom using Alice has focused on training teachers in Alice and encouraging them to create lesson plans using Alice, which are then implemented in the classroom. Various summer workshops have been held with the focus on giving the teachers resources to teach and learn Alice and to then build Alice worlds and lesson plans. The workshops may also offer the teachers an opportunity to test and review lesson plans on students who are also attending ‘Alice Camps’ in the area.

One such study called the SPIRIT program offered summer workshops to K-12 teachers of various grades and subjects with the goal for each teacher to create three Alice-based lesson plans to use in the up coming school year. The SPIRIT program showed not only that Alice-based lessons implemented by teachers have a positive impact on children’s learning but also that this approach is especially effective in the subject of science [8]. The positive impact is shown to occur for all children regardless of their learning ability and enforces Alice’s ability to equalize and improve learning in the K-12 classroom [8].
1.2. Alice Programming Adventures

The Adventures in Alice Programming [9] project aims to integrate Alice into K-12 schools with workshops that are run at various sites across the US including Duke. The workshops run with the goal of creating lesson plans over the summers with teachers who would use them in the fall. The audience is K-12 teachers, some of which are familiar with Alice and already use it in the classroom or in afterschool outreach clubs. The first workshop in 2008 spanned three weeks and a number of lessons, some of them science focused, were developed [10]. The following year, some of these teachers returned for a follow-up workshop with more Alice instruction and gave them the opportunity to share their use of Alice during the school year as well as give some direction to the future development of materials. 2009 was also the year of an Alice symposium, a chance for Alice developers, researchers, and users to gather and share developments on using Alice [10].

The second round of workshops in 2009 focused mainly on revising old resources and developing tutorials with a specific focus. The idea is that teachers can use the general beginning tutorials to give students the background they need to use Alice and then if the students have a particular need, topical lessons can be picked up individually. Beginner workshops were scaled down to a week instead of 3 weeks and focused on giving teachers the basic skills they needed to develop their own world and showing them how to access and go through the repository of topical tutorials on their own. The most recent set of workshops in 2010 followed the same format although follow-up workshops indicated that tutorials that encompassed Alice “projects” such as book reports for English or game development should be a direction to take [10]. By creating contained Alice projects that students could apply, Alice would be more readily translated to the classroom.
2. Methods & Results

2.1. Developing Worlds and Tutorials

An important component of the developed resources are Alice worlds and tutorials. Tutorials are resources that guide either a student or instructor through the building of an Alice world, usually introducing a new concept in the process. They are designed to be able to be followed by students alone but also as individual Alice lessons that teachers can go through with an entire class. The development process consisted of creating the target world first then recreating the world, taking screen shots and writing up an instructional PowerPoint and handout along the way. After this the materials go through a revision process by the other researchers. Ideally the tutorial can also be tested on groups of teachers during the summer workshops and resulting feedback can be used to continually improve the tutorial.

2.1.1 Beginning and Topical Tutorials

The first portion of my summer work focused on developing shorter topical tutorials and then a longer four part getting started tutorial called “Princess and Dragon” [Appendix A, Exhibit 1]. The 4-part getting started tutorial was used during the workshop to introduce Alice to the teachers. Once the teachers completed the tutorial, they could use the resource page to prepare the same tutorial for their classroom. Each section took about 45 minutes to an hour, which makes it perfect to do over four separate days in a middle or high school classroom. Fundamental concepts such as object positioning, methods, functions, and camera movement are taught so that after completing the set of 4 tutorials, students have a foundation to build on. Because of this, many of the more complicated tutorials on the resource site have the 4-part introductory tutorial as a suggested prerequisite.
The shorter topical tutorials ranged in topic. There are two broad categories of topical tutorials—those that focus on computer science concepts and those that teach animation and special effect concepts. I created three short tutorials during the first summer of research. The “Light Tutorial” [Appendix A, Exhibit 2] was did not specifically focus on any computing concepts but allowed the world creator to extend the storytelling and creative artistic aspect of Alice. The other two are support tutorials for teachers who are using Alice in a school setting. The “Alice Gallery Folders” [Appendix A, Exhibit 3] tutorial shows them how to create their own collection of objects to enhance their teaching experience with Alice by potentially grouping their own objects into the library for students to access or putting all elements of an assignment in one place to minimize searching through the gallery. Finally the “Sharing Alice” [Appendix A, Exhibit 4] teaches how to easily export Alice worlds into videos to be uploaded onto sites such as YouTube so students can share their work.

2.1.2 Projects

The second portion of the summer was spent developing Alice projects. Specifically, many of the follow-up attendees requested tutorials on how to make Alice games since that’s what their students had been asking about during the school year. The two games I developed tutorials for were “Alice Pong” [Appendix A, Exhibit 5], a very basic game that outlines important things such as boundaries and collisions, and “Treasure Hunt” [Appendix A, Exhibit 6], a model for a point-and-click game. Although the games themselves did not directly relate to any particular subject, the goal was to teach some general game building concepts so that teachers and students could create games as projects for individual subjects. Game development in Alice also aids in teaching pre-programming analysis, which is important in computer science development [11].
2.1.3 Example Worlds

Example worlds aim to showcase finished applications of Alice for a particular subject. Because the workshop audience includes teachers from almost every subject found in middle and high school, example worlds that are created include core subjects like science and math as well as electives such as foreign language and music.

2.1.4 Tutorial Website

All the resources developed by the summer researchers are then collected onto a tutorial website [12]. Materials are divided into three large categories: Tutorials, Example Worlds, and Videos. Tutorials are divided into three categories: Getting Started, Topical, and Projects. The idea is that students begin with one or some of the Getting Started tutorials, which are arranged depending on much time they have. Once students get the hang of Alice they can either do projects or create their own Alice worlds and use topical tutorials when they want to learn how to do something specific. Some of the teachers created the examples and videos, researchers created others. Example Worlds are divided into subjects to allow teachers to gather ideas and resources as to how Alice can be used in their particular subject. Videos are for visitors who do not have Alice on their computer or do not want to download a world and open Alice. These visitors can to watch a short segment of the world and still have the opportunity to get an idea of what Alice can do. These videos are offered in multiple formats and able to be linked to PowerPoint and other instructional media and allow easy sharing of Alice worlds although videos understandably cannot reproduce the interactive component in many of the worlds.

2.2. Alice in Middle School Science

Alice can play a huge and helpful role in the middle school science classroom. The link between the physical and life sciences and computing is already acknowledged in the curriculum
objectives for middle school science. Units in the 8th grade North Carolina Standard Course of Study (NCSCS) for Science include “Using Technology to Study our World” and “Technology for a better tomorrow” [2]. A goal common to both units is to learn how to use software to view and process data for communication. Many of the other units have sections where Alice can fill in as a model.

An interactive model can also be used to collect data. Many times the concept being presented has no experiment that can easily be done in a classroom; Alice can provide an interactive alternative that can be set up to collect data. Beyond teaching concrete facts, science curriculums aim to develop skills in students that will help them in understanding the scientific data. In fact, the NCSCS in science states that the achievement of “scientific literacy” is the primary goal of the science curriculum. Being able to create and read the visualizations of data used in scientific articles is a core part of their mission [2]. Being able to analyze data using visualizations is an important skill that Alice allows students to learn easily. An example of this approach can be seen in the “Bike Plot” world, created by a fellow researcher [13]. As a boy rides his bike, students can collect information about how far he travels as time elapses. Later on, this data is used to introduce how points are plotted on a 2D grid. Although the application for the bike world is a math classroom, collecting and visualizing data is equally important in a science classroom. The goal of this portion of the research was to create similar sorts of teaching resources for the middle school science classroom.

2.2.1 \textit{Lae} Operon Model

There is a need for interactive models in the science classroom to help conceptualize objects that are either too small or too large to be practically demonstrated in the classroom. Alice’s ability to easily produce interactive animations means that with the right mix of objects from Alice’s library and creativity, these models can be created using Alice. Examples where
Alice is used as an instructive interactive model include the “Helium Molecule” [Appendix B, Exhibit 1] developed by a previous researcher, and the “Hot Spot Volcano” [Appendix B, Exhibit 1] world created by a science teacher at one of the summer workshops.

Like the helium molecule, I created a lac operon model that uses Alice to help visualize concepts that are too small to see physically in a classroom [Appendix B, Exhibit 2]. Many middle school and high school biology textbooks classically use the lac operon as an example for gene regulation [14]. The model used simple differently colored geometric shapes to represent enzymes and molecules inside the cell. Two sliders marked “Lactose” and “Glucose” allowed students to play with the lactose and glucose levels independently and observe how that changed the interaction of molecules in the cell and ultimately affect the rate the lactase gene is transcribed.

One challenge to producing models, especially scientific, in Alice is the need for accurate objects. There are a lack of objects in the Alice gallery that are suited for teaching science. Ordinary geometric shapes were repurposed to represent enzymes, compounds, and DNA in the lac operon model. Current examples on the resource website showcase earth science and ecosystems because these types of objects are well represented in the Alice library. Other subjects such as genetics need objects made before teachers can easily create a model.

2.2.2 Surveying Science Teachers

The first steps we took involved reaching out to teachers and compiling a list of interested teachers who either currently taught or were certified to teach middle school science. The final list of teachers consisted of ten teachers from North Carolina and one from Virginia. Each received an electronic survey with the following questions:

1. Name and School / School District that you teach in.
2. What grade and subject do you currently teach?
3. Do you currently use Alice in the classroom? If so, how?

4. Do you have any ideas for how Alice can be used to teach science in middle and high school? What are those ideas?

Of these teachers, nine responded to the survey directly. Two of the teachers taught high school science, four taught 7th grade science, one taught 6th grade science, one teacher currently taught art and photography but had used Alice to teach 6th grade science in the past. One teacher focuses on special needs students and thus teaches multiple grades and multiple disciplines to a smaller classroom.

All of the teachers have had some experience with Alice, most of the experience coming from our workshops and tutorials. Four of the teachers had not used Alice in the classroom but were very enthusiastic about the prospect of doing so. Many of reasons for this included the school computer lab not having Alice set up yet or not enough time to develop the lessons for this year. All of them had plans for using Alice at some point in the future using the flexible time they have at the end of the year and the smart boards located in the classrooms to help. One of the 7th grade science teachers had used Alice in the past but this year did not have time this year in the curriculum—she mentions that she plans to return to using Alice next year. The remaining four teachers currently used Alice in the classroom. Many of them had students use Alice in projects. The most specific response was from Gary, a high school earth science teacher who said: “I teach Earth Science from a project based format which allows me to use Alice a great deal. I have let students do projects on it to turn in. I have used it to introduce a unit. I have used it to teach a unit. I plan on using it in all these ways this next semester.”

All of the 7th grade teachers suggested topics that paired Alice with earth science to illustrate concepts such as the rock cycle, ecosystems, and weather. Human Body was also a popular topic that teachers wanted to use Alice in. These topics were narrowed and I focused on
a couple to create example worlds that would showcase Alice’s ability to visualize data to aid in the instruction of science. Once these example worlds were finished, I sent them to the teachers for feedback. In general, the feedback from the five teachers that responded was positive, many of the teachers also expressed interest in the follow-up workshop being held this summer. One of the teachers suggested expanding the atmosphere world to include the different layers of the earth so that it could be used for two different units.

The lessons that I have developed have not yet been used in the classroom as both units that I picked to do are taught early in the school year. The teachers express that they hope to use them next year and give feedback and direction as a result.

2.2.3 Bar Chart Object

I created a bar chart world in Alice to aid in visualizing a collection of data [Appendix B, Exhibit 3]. The visualization method I focused on uses rectangles that resize height to indicate value. The bars can be put together in a list to form a bar chart that animates changing values by increasing or decreasing the height of the rectangles. Alice allows the list to be easily modified from the end so the list can be easily created if the user already has the complete data in a spreadsheet or if the data collection is taking place incrementally. However, the data is not easily modified if one wants to insert a point at the beginning or the middle.

The motivation for the world came from a high school biology course of study document from one of the teachers at the workshop [15]. The unit that I focused on was population change because it seemed to naturally lend itself to data visualization. One of the suggested activities for the unit is an activity where students simulate and record the changing population of fish in the pond originally using goldfish crackers. Alice provides a way to simulate the population of fish but also will allow the data to be collected and visualized in the same program. By changing the numbers of fish that were taken from the pond, students could see
how populations changed and instead of recording the numbers by hand, the numbers could be inputted into a growing list in Alice. As each round saw the students taking more and more fish, eventually the population stopped increasing and began to quickly crash—a model for unsustainable fishing—that they can quickly see both visually but also in the bar chart that is automatically created in the end.

This bar chart became the model for a more general bar chart object. I created a tutorial [Appendix B, Exhibit 3] to show instructors how to create a bar chart object in any of their worlds as well as a template world that they can start off with that already contains a chart. Included in the tutorial are instructions for changing elements in the chart such as the title and colors.

2.2.4 Bar Chart Application

To showcase the bar chart visualization in another application, I created an atmosphere world that uses two animating bars, one for temperature and one for pressure. I chose the atmosphere primarily because it was a topic that multiple teachers in the survey mentioned that they wanted to see in an Alice world. It is also a topic that would be greatly enhanced by data visualization. Since the layers are differentiated by how temperature behaves in relation to altitude, it is important to differentiate how pressure and temperature behave differently. As the scientist advances through the different layers of earth’s atmosphere, students are able to learn about each layer as well as visually appreciate how the two variables of interest change [Appendix B, Exhibit 4].

The data for the world is stored in three different lists in the world—temperature, pressure, and altitude. I obtained the data for the world by combining NASA [16], UK’s Discovering Antarctic Organization [17], and Portland State University [18] models. Originally a single list of 3-tuples, they had to be separated because the types of information that can be
stored in Alice are limited. The lists of data and bars are tied together with methods written in the Alice world. The final goal is for special charting objects to be built into the Alice library and take advantage of the list and array capabilities that are already built into Alice and tie the list directly to the object. Having a built-in object would mean that instructors and students could easily take any list in any world and drop in a chart visualization object to visualize data in the world. We have contacted Dr. Wanda Dann [19] and her team at CMU and they plan to create this object.

3. Discussion

3.1 Conclusion

In order for computer science to be viably introduced to middle school students, it is important that computing concepts be integrated into the classes that these students take. Teachers need to have access to both the computer lab equipment as well as Alice resources if Alice is going to be a tool in their lesson plans. These resources include general Alice tutorials but each subject presents different challenges and opportunities for Alice to be used. In particular, science can be paired successfully with Alice if Alice is used to create engaging and interactive models to motivate students. Many concepts in science are hard to demonstrate and visualize because they either take place on a cosmic or molecular scale. Alice can address both issues of scale because it presents a virtual world that can be manipulated into a model given the right objects. In particular, educational models that emphasize the collection and visualization of data can be created in Alice to help improve the scientific literacy of our students.
3.2 Future Work

The next steps are to have the tutorials and projects used in the classroom by science teachers who have previously attended Alice Programming Adventures and showed interest in using Alice more extensively. The tutorials and projects should be revised based on their feedback and new projects and tutorials created tailored to their vision and traditional lesson plan. Hopefully these same teachers will attend follow-up workshops where researchers can work with them one-on-one to gather feedback and develop new applications. Many of the science teachers from the workshops have lessons in mind that they would like to use Alice in mentioned in their responses to the survey. Further research into applications for charting in Alice will help guide what sort of lesson plans Alice best serves as a tool for. Because many of these teachers are certified for and teach classes besides science, it should also be considered what other subjects could make use of this approach. For example a math class could create a probability model in Alice as well as use it to visualize data in certain lessons.

While rectangles may be the simplest shape to use, other common visualization structures that students will encounter in science literature should be considered. Pie charts and Punnet squares are potential ways of visualizing that Alice could be used for. While a circular pie chart would be difficult to do in Alice, a square pie chart would be a viable alternative. Unlike in the bar chart, squares would resize in height and width so that the area of each square makes the correct percentage of the whole. Tricky elements to this approach would be the need to figure out how to best partition the area and to have enough square elements present so that many different breakdowns were possible. The list of data would have to be sorted and the larger squares done first and then the smaller squares would fill in the rest of the space. The calculations for doing this would be complicated and perhaps a better approach to doing this type of visualization can be found.
The CMU team will develop general charting objects and, once they are finished, existing tutorials should be revised. In addition to general charting objects, the continual creation of new objects into the Alice library, especially those that can be used in science lessons such as DNA or the human body would help science teachers tremendously. Other applications for Alice in a science classroom besides in modeling and data visualization should also be further researched to help determine what sorts of objects should be developed and added into the Alice library.
Bibliography


Appendices

Appendix A: Alice Tutorials & Projects

Exhibit 1: Princess and Dragon

Princess and Dragon is a 4-part introductory tutorial. It was developed for the 2010 workshop and is also available for teachers to use with students. The following four tutorials build on one another. For example part two starts off with the finished world from part one. Finished worlds for each part are also located on the website. The storyline was chosen to appeal to both male and female middle school students. The first tutorial, introducing a dragon and princess, focuses on object placement and shows how to create a simple animation in Alice. The second tutorial has the dragon capturing the princess and focuses on writing object level methods and putting methods together. The third tutorial contains a knight and his horse rescuing the princess and introduces basic events, camera control, and functions. The fourth tutorial is optional and includes topics that are less computer science oriented and more animation oriented such as using 3D text, billboards, and sound to make the final scene more polished.
Princess & Dragon
Part 1: Objects in an Alice World

Introduction

Welcome to Alice! Alice is an innovative 3D programming environment that makes it easy to create an animation. These animations can be used to:
- tell a story
- make an interactive game
- create a video that you can share online
- and much more!

Alice uses 3D graphics and a drag-and-drop interface so its fun and easy to learn.

In this four part tutorial, you’ll be learning some of the basics of Alice so that you can program your own stories and games.

Overview

In this tutorial, we’ll be creating a short animated story about a princess who is captured by a dragon and rescued by a knight that comes riding in. The tutorial is broken up into four parts.

- Part 1: Objects
- Part 2: Methods & Properties
- Part 3: Cameras & Events
- Part 4: Billboards, Sound, & 3D-Text

Step 1: A New Alice World

In Part 1 we’ll set up the distressed princess calling for help on top of a tower. This will cover how to set up a world, add objects, position them, and create a simple animation.

Start Alice and after it loads (it may take awhile) you will see the “Welcome to Alice!” screen.

From this screen you can start a new world with a template or open a saved Alice world from earlier.

Click on the Templates tab to start a new Alice world and you will see 6 basic templates. Select grass and then click open.
Step 1: Main Screen

The main Alice screen is broken up into sections.

Step 1: Saving an Alice World

Click on File in menu at the top of the screen, select Save World and give a name to your world. Click Save when you are done.

Alice worlds are saved as .a2w files. Remember where you save the world so that you can find it later!

Step 2: Adding Object Screen

Click on the Add Objects button in the World Preview Pane and You will see the screen change into the Adding Objects screen.

Step 2: Dummy Objects

A Dummy Object allows us to save the position of the camera and return to it. Before you add or position any objects, it’s a good idea to save the original camera position.

Click on the more controls button under the positioning tools, select drop dummy at camera.
**Step 2: Renaming Objects**

You will notice in the object tree that a new folder called Dummy Objects has appeared. Right click it and rename it cameraViews. Open the folder by clicking on the + next to it. Right click on the dummy object inside and rename it originalView.

**Step 2: The Gallery**

The Gallery contains different types of objects that you can add to a world in Alice. It is organized into folders which are listed in alphabetical order.

Scroll to the right until you find the People folder. Click on it. Scroll to the right until you find Cinderella. She will be our princess. Click on her and you will see a screen pop up.

Click on Add instance to world and you will see Cinderella appear in the world!

Congratulations you’ve added your first Alice object!

**Step 2: More Objects**

Click either the folder icon next to People or click Local Gallery to exit the people folder.

Now add a dragon and a tower. Find the Medieval folder and add a Dragon and a Tower.

**Step 3: Positioning Tools**

The panel to the right contain a variety of positioning tools that will help you move the objects on the screen to set up the scene.

To use the tools, you click on the tool that you want and then click & hold on the object you want to position. Move the mouse to watch the tool work!

DO NOT use the last tool on the right, it does not work.

If you mess up, use the Undo button in the toolbar!
**Step 3: Positioning Tools Continued**

Use the **mouse** tool to move the **tower** to the left and move **Cinderella** back so that she is almost at the base of the tower.

Use the **Up and Down** tool to move the **tower** down a little and **Cinderella** up so that she is in the doorway of the tower. Move the **dragon** up off the ground.

**Turn the tower** so that the camera sees it straight on.

**Use the Tumble tool** to move the **dragon** in the air so that he looks like he is flying.

**Step 3: Positioning Tools Continued**

Positioning objects in Alice is tricky. Take your time and don’t be afraid to play with a tool until you get the hang of it. If you accidentally move the ground or the camera, just use **Undo**.

Move the **tower** forward (using the **mouse** button) and use the **Resize** tool to make it smaller so that you can still see it in the screen.

Click on **quad view** and you will see a new view of the world with four windows. We will use this view to move **Cinderella** onto the **tower**.
Use the Scroll and Zoom tool on the from the top view in order to get the top of the tower into view. First zoom out until you find the tower, then zoom in on it and scroll until you get the right view. Use the mouse tool to move Cinderella in that view onto the tower. (If she goes into the tower instead of on top of it, move her up more in single view)

Practice using zoom and scroll until you can get a good view of the tower in all four screens.

When you are done, click on single view to go back to one screen.

If you click on an object, either in the viewing pane or in the object tree, you will notice that lines appear around and through the object.

Click on the dragon. The center of the object is where the green, blue, and red lines meet. When you tumble the dragon for example, he moves around his center. The dragon’s center is located in the middle of his body but each object has its own center.

Click on Cinderella. You will notice that her center is around her knees.

You will also notice the yellow bounding box of the object. This can be useful when you’re trying to line up the object. There are three orientation lines to help you. The green line indicates the object’s sense of up. The red line indicates the object’s sense of right. In Alice, everything happens relative to the object so even if it is your left, keep it mind that it is Cinderella’s right. The blue line indicates the object’s sense of forward and backward.

Between the viewing pane and the gallery you will see three sets of arrows. These are the camera controls. We will be using them next to create a new camera view. To use them, click on the arrow you want and hold release when you are done.
Step 4: Camera Positioning

- Move the camera so that the tower is in the center of the view.
- Drive the camera so that you can see Cinderella.
- Tilt the camera slightly.

We have created a new camera view! Drop a dummy to save it and name it towerView.

Step 5: Methods

Click the green DONE button to go back to the main view.

Now that we have all our objects in place, it’s time to learn how to animate them in Alice. We will make Cinderella jump up and say she is trapped.

The object tree shows all the objects that are in the world. Click on Cinderella since she is the object we want to animate.

Click on methods in the details pane.

When you click on an object, its details show up in the details pane. There you can see the object’s properties, methods, and functions. A method is a sequence of instructions that will be carried out when instructed. All the methods you see in details show the basic behavior that every object knows how to do. Built in methods are combined to create more complicated animations.

Step 5: First Instruction

A list of options will appear, asking you what Cinderella should say. Click other... and enter oh no, I’m trapped! Then press OK.

You will see the new instruction appear in the method editor.
**Step 5: Testing**

Click **Play** in the toolbar to test the new instruction. The words disappear too quickly so we would like to change that.

The viewing screen has many tools to help you. You can adjust the speed the animation plays through with the slider as well as pause, resume, restart, and stop the animation anytime. Always remember to **Close** the viewing screen to return to the main view.

*Test frequently when you are building an Alice world. You can speed through parts that are working and pause to get a better look at parts you want to fine tune. Don’t wait until you have many lines written to test!*

**Step 5: Duration**

Click on **more...** in the **say** instruction to bring up a list of options. Select **duration** and then 2 seconds.

*All instructions are set to 1 second by default so doubling the time will make the words stay on the screen for longer.*

Click **Play** in the toolbar to test. The words now stay up long enough for everyone to read it.

*There are many other options in the more menu such as changing the font and color of the words. Try them out!*

**Step 5: Move Method**

In order to make her jump, we will move her up and then down.

Drag **Cinderella move** from the details pane into the method editor above the **say** method that is already there. Release when you see the green line where you want the instruction. Select **up ½ meter**.

**Step 5: Copy a Method**

Right click on the Cinderella move method and select **make copy**. You will see a copy appear below the original. This is useful when you want to copy a single line of code.

Since we want her to move down in exactly the same way she moved up we made a copy. Click on the **up** and select **down**. Press **Play** to test this.
Step 5: Do Together

Better but not quite what we want, we want her to jump up and say together. We can do this by using a Do together block. There are two main types of blocks, Do in order and Do together. The default is a Do in order.

![Do together block](image)

Drag in a Do together block from the bottom of the method editor in.

![Do together block](image)

Drag each of the move and say methods in by clicking on the bumpy left side and dragging it into the Do together block.

Step 5: Do in Order Continued

![Do in order block](image)

Drag in another Do in order inside the Do together.

Step 5: Do In Order

When you press Play, nothing happens! This is because the move up and move down happen at the same time, cancelling each other out!

![Do in order block](image)

Drag in a Do in order into the Do together block. Drag the move up and move down commands inside.

![Do in order block](image)

Press Play, why does it work now?

Step 6: Subparts

At this point we are basically done with part 1. But Cinderella’s arms don’t look quite right. Let’s review the animation skills we’ve learned and make them look more natural.

![Object tree](image)

This time we only want to animate Cinderella’s arms. To do that we must select her arm.

Go to the object tree and click on the + next to Cinderella then next to hips, torso, upperBody until we see the leftUpperArm. Click on that.

Objects in Alice are made of subparts that you can animate. You will notice that the arm has its own details pane and bounding box, just like the bigger Cinderella object.
Step 6: Subparts Continued

From the leftUpperArm’s detail pane, drag in a roll instruction into the new Do in order. Select right and then other… Using the calculator, punch in .2 and then click Okay.

The roll command moves an object around it’s center. Since the center of the arm is near the shoulder, the roll command will cause the arm to look like it is swinging. We choose right because remember it is relative to the object that instructions are animated and we want the left arm to swing toward Cinderella’s right.

Step 6: Another Copy

Another way to copy code is to use the clipboard at the right of the toolbar.

Drag the entire Do in order with the two roll methods onto the clipboard.

Drag the clipboard into the Do together.

This method is useful for copying chunks of code.

Step 6: Subparts Continued

To roll the arm back downwards, drag a roll command below the previous one and this time select left and then other… and punch in 0.4.

Play the world to test it. It looks good! Now let’s do it again for the other arm.

Step 6: Another Copy Continued

We want the new block of code to refer to the right arm. Click on the arrow next to leftUpperArm in the first roll method in the block and select Cinderella, hips, torso, upperBody, rightUpperArm, the entire rightUpperArm.

Do the same for the other roll method then change the right to left and the left to right.
This is the final code for the tutorial. **Play** your world one last time to make sure everything is right.

Congratulations on finishing part 1! In part 2 you will learn more about methods and how to do more animating in Alice.
Princess & Dragon
Part 2: Teaching a Dragon to Fly—Methods & Properties

Welcome to Part 2 of the Princess & Dragon tutorial. In Part 1 we covered how to set up a world, add and position objects, and create a simple animation.

In Part 2 we’ll add more animations so that the dragon will kidnap the princess.
This will cover how to change camera views, create and edit methods, and change properties.

Step 1: Dummy Object Review

In Part 1 you learned how to drop a dummy object to save the view of the Camera. We dropped one named originalView (the location of the camera when Alice starts) and another dummy object called towerView (when we moved the camera to get a close up view of the tower).

You can find these in the folder that we named CameraViews. We will now use these saved Camera views.

Step 1: Setting Camera View

There are two ways to change the camera view. This first way changes the camera view before the animation.
Select the camera from the object tree and drag it into the world preview pane.

You will see a menu pop up, go to camera set point of view to and select CameraViews, originalView. This will set the scene back to the original camera view.
In order to change the camera view during animation, we must drag in a method.

Find camera set point of view to from the camera’s list of methods (you will need to scroll down). Drag this in right above but outside of the Do together.

Select CameraViews, towerView.

We create new methods that put together the simple methods Alice gives us for each object in order to create short animations that we can use over and over. We’ll use this to teach the dragon to fly.

Click on dragon in the object tree and in the methods pane, click create new method. Name it fly and select OK.

You will see a new tab pop up in the method editor labeled dragon.fly and also a new method called fly in the dragon’s methods.

Drag in the same instruction at the end of the method, outside of the Do together.

This time set it to the originalView.

Play the world and you will see that the first instruction we added makes the camera zoom in on the tower and that the second instruction returns the camera to the first view.

There are now two tabs in the method editor. Each tab represents a different method and the code for that method.

Click on world.my first method and you will see the code that we wrote before.

Click on the dragon.fly tab and you will see there is no code. The code that we will put in here will teach the dragon to fly.

CAUTION: When you have multiple tabs, always make sure the correct tab is up before dragging and dropping in code.
Step 2: DragonFly Object Method

There are two types of methods you can create, object level methods and world level methods. We have created an object level method because the method is inside the object dragon. In an object level method, all the animations must stay within the object and not use any other objects in the world.

First drag in a Do together then a Do in order inside and another Do together. These blocks set up the method that we will fill in.

Step 2: Animation

Drag in a roll command from the rightWing’s list of methods into the second Do together.

Select right, other, and punch in .15.

Step 2: Animation Continued

Click on the + next to the dragon’s rightWing in the object tree and then click on flap.

Drag in the flap roll method into the Do together, select left, other. Punch in 0.1.

Drag in the flap turn method into the Do together, select right and then ¼ revolution.

Most of the time when you are creating your own animation, you will need to play around with the numbers and methods until you get it just right. In this tutorial we give you the exact amounts but it took a lot of trial and error to find the right number.

Note: Your dragon may have a rightWingClose instead of rightWing depending on your version of the Alice object. These are the same but with different names. Use rightWingClose instead of rightWing and instead of flap use rightWingFar.

Since this is an object method in the object dragon, we will only use objects that are part of the dragon. The dragon has a lot of parts but we will focus on his wings which are further divided into flaps.

Click on the + next to the dragon in the object tree and scroll down until you find the rightWing which we will animate.
Drag the block of code from the clipboard underneath the Do together inside the Do in order block. Reverse the directions from right to left and left to right on each line.

Press Play. When you try to test the method we’ve been writing a warning box will pop up.

In Alice there has to be a starting point for running your code. In the event editor, you can see When the world starts, do world.my first method. The default is to run the code in the first method you wrote, my first method. For testing purposes, we can change the method that is run. In this case we would like to test out the dragon.fly method to make sure it works before we integrate it into the program.

In the event editor, change the method that plays When the world starts to dragon.fly.

Copy the entire Do in order block onto the clipboard.

Drag from the clipboard into the big Do together, underneath the previous Do in order.

Now press play and you will see the dragon’s right wing flap once. Let’s repeat for the left wing!
Change all the rightWing references to leftWing by clicking each of the lines next to dragon.rightWing. Select dragon, leftWing, the entire leftWing. Do the same for the flap references.

Flip the order of the two Do together blocks by dragging the second one above the first one.

This is what it should look like when you’re done. Go to the next slide to see the final code for the method.

Now we need to create a method that uses dragon.fly.

To create a world level method, click on the world in the object tree and in the methods pane, click on create new method. Name it capturePrincess since that is what it will do. You will see a new tab pop up in the method editor and a new method in the world’s methods list.

This is the final code for dragon.fly. Press play to see the dragon flap his wings!
The *dragon.fly* method is an object method because all of the code refers to the *dragon* and no other object. My first method is a world method because multiple objects are referred to: the *camera*, the *towerView* object, and *Cinderella*.

Drag a loop into the *world.capturePrincess* method. Select 2 times.

Into the loop, drag in a *Do together*. Then click on *dragon* in the *object tree* and under *methods*, drag in the *fly* method we just finished writing. Put it inside the *Do together*.

Drag the *dragon turn* method into the *Do together*, select right 1 revolution.

Click on *more...* and select *duration 2 seconds*.

*This is to make sure the dragon finishes one turn after he flaps his wings once.*
Step 3: Testing CapturePrincess

In the Events editor, change the When the world starts event from dragon.fly to capturePrincess.

Press Play to test capturePrincess.

It looks like the dragon is just turning in place here, but we want to dragon to go around the tower. That can be done using asSeenBy.

AsSeenBy can be used to make an object go around another object.

Step 3: As Seen By

In order to make him go around the tower we will use asSeenBy. Click on more and select asSeenBy tower.

Press Play to test capturePrincess once more.

You will see that the dragon goes around the tower!

Step 3: Animation

To move the dragon to Cinderella first drag in a Do together and then two move commands. One should be up 5 meters and the other right 10 meters.

If the dragon’s feet isn’t in view at the end of the animation when you test this, decrease the amount he moves up from 5 meters to 4 meters.

Step 3: Animation Continued

Drag in a dragon move toward command into the Do together. Select 2 meters, Cinderella, the entire Cinderella.

We want to change the 2 meters to 3 meters so click on amount and select other... punch in 3 into the calculator.
### Step 3: Animation Continued

The next step is to have Cinderella “picked up” by the dragon.

Drag in a Do together and then click on Cinderella in the object tree. Drag in a move to command and select the dragon’s frontRightLeg, lowerLeg, foot, the entire foot.

Drag in a Cinderella move command into the Do together. Have her move down 1 meter.

We want this to happen immediately so set the duration of both commands inside the Do together to 0 seconds (click on more and enter in 0 into the calculator).

Watch the animation and you will see that Cinderella is instantaneously picked up by the dragon.

### Step 3: Vehicle Property

In order for the dragon to fly off with Cinderella we need to ‘glue’ Cinderella to the leg of the dragon. To do this we will use the vehicle property. Properties are information about an object. You can change them in a method just like you can animate the parts but they reflect the current state of the object.

The vehicle property is set to world by default. When you change the vehicle property it means that whenever the vehicle moves, the object also moves with it. So when the dragon moves, Cinderella will move with it.

### Step 3: Gluing the Princess

To change the vehicle of an object during an animation, click on the properties tab and drag vehicle into the code. Put it inside the Do together and select dragon, the entire dragon.
Step 3: Animation Continued

To finish up the code drag in another Do together. We will move the dragon away from the tower. Drag in two dragon move commands, one for moving up 5 meters and another for left 10 meters. Put these inside the Do together.

Outside of the Do together, drag in another move command for down 5 meters. The method is now finished! See the next slide for a copy of the final code.

Step 4: Calling Methods

Now let’s put all the code that we’ve written together.

Click on the world.my first method tab in the method editor
Click on world in the object tree and find the capturePrincess method under the methods tab.
Scroll down to the end of my first method and drag the capture Princess method into the very end of the method.

Step 4: Comments

Comments are notes for people and are not code, so when your code is executed, comments are ignored.

Drag in two comment lines into the very top of world.my first method. Double click on them and enter in your name and today’s date.
Step 4: Color Property

You can also change the properties of subparts of an object. Properties can be changed either before animation to set up a world or during animation. One property that can be fun to change is the color property. Not every object can be colored but do try it out!

Click on Cinderella's skirt in the object tree and click on the box next to the color property. Select magenta and watch her dress change color!

Step 4: Color Property Continued

To change the color of an object back, click on the box of color and select no color. This will return the original color back to the object.

Another way to change color is during an animation. In order to change properties during in animation we need to turn it into a line of code.

Click on color and drag it into my first method. Release it right before the Do together. Select Magenta.

Congratulations!

To play the whole animation, change the method called by the When the world starts event back to world.my first method.

Congratulations on finishing Part 2! Part 3 will teach you more about events and different uses for the camera.
Welcome to Part 3 of the Princess & Dragon tutorial. In Part 2 we covered how to change camera views in an animation, create and edit object and world level methods, and how to change properties.

In Part 3 we’ll add more animation so that a knight will come in on a horse. This will cover more on camera views as well as introduce you to new events and functions.

**Step 1: Adding Objects**

First a knight and horse must be added to the scene. Click on the Add Objects button to go to the Adding Object Screen. Click on the Medieval gallery folder and add a horse and a knight.

**Step 1: Positioning Objects**

Use the positioning tools to turn the horse and knight toward the camera.

Then move the knight so that he is in the middle of the horse with the mouse tool.

Finally move the knight up until he looks like he is sitting on the horse.
Step 1: Vehicle Reviewed

To glue the knight to horse, click on knight in the object tree. Under properties, set the vehicle of the knight to the horse, the entire horse.

Remember that changing the vehicle property this way is an instant change to the property and part of the setup of the world. There is no dragging in a line of code so it is not part of the animation.

Step 1: Positioning Objects Continued

Right Click on the knight’s right leg in the object tree select roll, left 0.1 revolutions.

Now Roll the left leg right 0.1 revolutions. This will move the legs of the knight so he looks more like he is riding.

Step 1: Vehicle Review Continued

Right click on the horse in the object tree and under properties, change the vehicle of the horse from the world to the camera.

The trick here is to glue the horse to the camera so that when the camera view changes in the scene editor, the horse (and knight) moves with it. When we are done creating a new camera view we can simply ‘unglue’ the horse.

Step 1: Camera Positioning

Drive the camera to the right until you can no longer see any of the other objects. Now you can ‘unglue’ the horse from the camera by setting the horse’s vehicle back to the entire world.
Step 1: Dummy Object Review

Remember that a Dummy Object allows us to save the position of the camera and return to it. It’s like bookmarking a camera view. We did this with the originalView and towerView. Now we want to add a bookmarked camera view of the knight and horse into the CameraViews folder.

Click on the more controls button under the positioning tools, select drop dummy at camera.

The dummy object will appear in the CameraViews folder in the object tree. Rename it knightView.

Right click on the camera in the object tree and select camera set point of view to CameraViews, originalView.

Click on DONE. Now it’s time to animate!

Step 2: Change Camera View

First we want the camera to swing over to the knight during the animation. Drag a camera set point of view to method into the very end of my first method.

Select CameraViews and then knightView.

Step 2: knightRescue World Method

Now let’s create a new world level method. In this method the knight will ride over to the princess and yell at the dragon to drop her. Then the knight will catch the princess.

Click on world in the object tree and under methods, click on create new method.

Name it knightRescue.
Step 2: Animation

Click on knight in the object tree. Drag in a knight say method into the new knightRescue method. He will call out a damsel in distress!

You will notice the AnimateBreathing method in the knight’s methods. We won’t use it in this tutorial but some objects come with more complicated methods unique to that object built in.

world.knightRescue No parameters

No variables

- knight say a damsel in distress!
- more...

Step 2: Animation Continued

Click on the horse in the object tree.

Drag in a turn to face method and select cinderella, the entire cinderella.

Then have the horse move forward 15 meters.

world.knightRescue No parameters

No variables

- knight say a damsel in distress!
- more...
- horse turn to face cinderella
- more...
- horse move forward 15 meters
- more...

Step 2: Mouse Click Event

To test the method, we will create a new type of event.

Click on create new event in the events editor and select When the mouse is clicked on something. Select the entire knight and knightRescue for the blanks.

Events create new event

- When the world starts
- When a key is typed
- When the mouse is clicked on something

Step 2: Animation Continued

The animation looks fine so far but we want to make sure that the knight makes it to the tower. Let’s change the camera view and check.

Click on camera in the object tree and drag in a set point of view to method.

camera’s details properties methods functions

- camera get a good look at
- camera move to
- camera move toward
- camera move away from
- camera orient to
- camera turn to face
- camera point at

camera set point of view to originalView

Select CameraViews, originalView. Now press Play.
Step 2: Testing

Depending on where you moved the camera when creating the new view, the horse and knight may be in the final shot of the animation.

If not, change the amount that the horse moves forward from 15 to 20 and test again, the knight and horse should be in view when the animation ends.

Step 3: New Camera View

We can also create moving camera scenes by tying the camera to an object. Drag in a camera set point of view to method between the two horse methods and select the knight, upperBody, neck, head, helmet, faceGaurd.

Once again the vehicle property will be used to tie the camera to the knight as he is moving.

Drag the camera vehicle property into the code after the set point of view to knight's facegaurd command and select the entire knight.

Drag in another one after set point of view to originalView and set the vehicle to world.

Press play to see what the knight sees!

Step 3: Animation

Click on the knight in the object tree and scroll down until you find the knight say method.

Drag it into the method and have him say "let her go!"
Step 3: Animation

Now let’s finish the animation and rescue the princess.

Click on dragon in the object tree and find the list of methods.

Drag in the turn to face method and select the entire knight.

Drag in a move forward command and for now select 1 meter.

Step 3: Functions

Since we do not know how far the dragon is from the knight we will use a function. A function in Alice does not change the world like a method but it gives us information about the world like the distances between two objects.

Drag it into the code over the 1 meter until there is a green box. Drop and select knight, the entire knight.

Step 3: Animation Continued

Click on horse in the object tree and drag in a turn to face command. Select the entire Cinderella.

Following that, drag in a horse move forward command and select 1 meter.

Step 3: Function Continued

Click on knight in the object tree and then find his list of functions. Drag the knight distance in front of function over the 1 meter and select the entire Cinderella.

Set the duration of both commands to 0.5 seconds by clicking on more at the end of each line.

We want the horse to bring the knight to Cinderella, so we will use a function since we do not know the exact distance.
Drag in Cinderella’s vehicle property into the method and set it to the entire world. This will 'unglue' her so that when the dragon moves, she will no longer move with it.

Now drag in a Do together. We will now drag in commands to make Cinderella fall into the arms of the knight and the dragon fly away.

Drag in another Do together and into that, drag a Cinderella move backward command (remember to click Cinderella in the object tree to get a list of her methods). Set it right now for 1 meter. We will use another function to determine how far she will fall.

Play the world at this point. You will see that Cinderella is left on her back in the air. Because animation happens relative to the object we need to move her backward to make her fall.
Drag in a Cinderella roll left command set it for \( \frac{3}{4} \) revolution and duration 0.25 seconds.

Drag in another Cinderella move command for down \( \frac{3}{2} \) meter.

The next step of the animation is to raise the knight's arms to catch her.

First find the arms of the knight under his upperBody in the object tree.

For each arm, drag in a turn forward command, select \( \frac{3}{4} \) revolution.

Press Play to see the final animation!

This slide and the next slide contain the final code for world.knightRescue.

Congratulations on finishing Part 3! Part 4 will teach you how to embellish your animations so that they are more engaging. We will use billboards, sound, and 3D text to make this world come to life!
Princess & Dragon
Part 4: Breathing Fire—Adding Effects to Alice

Welcome to Part 4 of the Princess & Dragon tutorial. In Part 3 we covered how to create an event to change camera views as well as introduced you to functions.

In Part 4 we’ll add more events, tying them to ‘special effects’ in Alice to make the animation more realistic.

This will cover billboards, sound, and 3D-text.

Step 1: Fire Object

The first thing we want to do is have the dragon roar and breathe fire. We’ll do this by reviewing some of the techniques from parts one through three as well as introduce a new type of event and show you how to integrate sound into your Alice worlds.

Click on Add Objects to access the gallery. Find the Special Effects folder in the Local Gallery.

Find the Fire object and add an instance of it to the world.

Step 1: Positioning the Fire

Use the positioning tools to move the fire to the mouth of the dragon.

You will need to use quad view to turn and rotate the fire so that the base is at the mouth of the dragon.

Click on done to exit add objects mode.
**Step 1: isShowing Property**

Under the **properties** of the fire, set the **vehicle** property to the **dragon**. Remember, this will glue the fire to the dragon.

*Another property that is useful is the isShowing property. Like the name suggests, by setting isShowing to false, you can make objects disappear. To make them reappear, set isShowing to true.*

Set the isShowing property of the fire to false.

---

**Step 1: New Method**

We will create a new method called dragonFire to put in our code.

Click on world in the object tree.

Click on create new method under the methods tab and name the method dragonFire.

**dragonFire** is a world method because it will use the fire and also sound from the world. Since these objects are different in the object tree, it must be a worldMethod.

---

**Step 1: isShowing & Wait**

Click on fire in the object tree. The first thing we will animate is making the fire appear and then disappear.

Drag the isShowing property into the code and set it to true. Put in a Wait command and select 1 second. This will simply make the code do nothing for a second.

Drag in another isShowing property and set it to false. Set the duration of the isShowing lines to 0 seconds.

**Step 1: When Key is Pressed Event**

To make this code run we will create a new event that runs code when a key is typed.

Select create new event in the event editor. Select When a key is pressed.

In the first blank, select Letter and then F. In the second blank select dragonFire.

You will notice that there is a new event in the events editor under fire: When the world starts, do fire.spin like crazy. Some objects come with built in events; in this case this method helps the fire look natural.
Step 1: Testing dragonFire

Now play your world. When the dragon begins to fly around the tower, press F.

You will notice that every time you press F, the fire will appear. This happens whether you can see the dragon in the frame or not.

Step 2: Adding Sound

If you have not downloaded the creature_roar2.wav file from the tutorial site, do so now and save it in a place you can find it.

There are many short clips of sounds for free online. Check the terms of use before you use music however. For this particular animation we are looking for a noise that sounds like a dragon and lasts for about a second.

Click on world in the object tree and then on the Properties tab. Scroll down until you see Sounds and click on the + next to it to reveal the sound controls.

Click on import sound and find the creature_roar2.Wav. Click on import when you have found it. You will see that it appears under Sounds.

You can preview the sound by pressing the green play arrow.

You will see that you can also record a sound. This feature currently only works on the Alice 2.2 PC version and not the Mac version.

An alternative for Mac users is to use another program to record and save the sound and then import it into the world.

Step 2: Animation with Sound

We want the fire and sound to appear and play at the same time.

Drag in a Do in order and put all three lines of code inside. Drag in a Do together and put the Do in order inside.

Drag in the creature_roar2 clip into the Do together outside of the Do in order.
**Step 2: Testing dragonFire Again**

Play the world and press F when the dragon begins flying. You should see the fire and hear the dragon roar.

*Make sure your volume is on!*

The next step is to add the words “The End” to the animation and have them show up at the end after the knight catches the princess.

**Step 3: 3D Text**

Click on Create 3D Text at the end of the local gallery in order to add a 3D Text object to the world. Type in the “The End” and select a font that you think looks good. We chose *Lucida Blackletter* but every computer has their own list of fonts. Click OK when you are done.

*Rotate* and *move* the text up with the positioning tools.

**Step 3: 3D Text Properties**

You will notice that the 3D Text object appears in the object tree. 3D Text is an object just like all the other objects in Alice, but they do have some special properties. Click on the properties and you will see that you can change what the text says, and what font it is from the properties panel.

Change the *color* of the text to *grey*.

Set the *isShowing* to *false*.

We will now animate the text to appear at the end.

**Step 3: 3D Text Animation**

We will add this code to the end of *knightRescue*. If you do not have the *knightRescue* tab open, go to world methods and then click the *edit* button next to *knightRescue*.

Scroll down to the end of *knightRescue* and drag in the 3D Text *isShowing* property into the very end of the method, outside of the last *Do together*. 

```plaintext
world.knightRescue No parameters

No variables

knight.upperBody.leftArm turn forward 0

dragonFire add

The End text

3D Text set isShowing to true more...

vehicle = world

isShowing = false

vehicle = world

isShowing = true
```
Step 4: Making a Billboard

A billboard in Alice is a flat rectangle with a picture on it. You can either find the images for billboards online or you can make your own in an image editing program (like Paint or Photoshop) or Powerpoint.

Either download the forest-1.jpg file from the tutorial website or use another image of a forest that you find online. Save it on your computer so you can find it later.

To put the billboard into the world, click on File and select Make Billboard. Find the image and click Import.

Step 4: Positioning the Billboard

Click on add objects in order to move the billboard into place. Position the billboard behind the tower and the dragon. By resizing it and moving it backwards.

Billboards can also be animated like any other object. We will add a curtain billboard that will rise at the beginning of the animation and drop back down at the end.

Step 5: Another Billboard

The curtain we will be using is another billboard that you can download from the tutorial site. You can also find similar images online.

Make a Billboard with the stage-curtain1 image or another similar image. Turn and Move it forward and resize it so that it covers the entire view.

Step 5: Animating Billboard

To raise the curtain at the beginning we will add a line of code to world.my first method.

Click on the world.my first method tab in the method editor. Click on the stage-curtain1 billboard in the object tree.

Drag the stage-curtain1 move method into the code directly below the two comment lines. Have the curtain move up 10 meters. Set the duration to 5 seconds.
To lower the curtain at the end we will add a line of code to world.knightRescue.

Click on the world.knightRescue tab in the method editor.

Drag the stage-curtain1 move method into the code into the end directly below the 3D text isShowing command. Have the curtain move down 10 meters. Set the duration to 5 seconds.

Congratulations on finishing Part 4! Use the basic skills that you have learned to create your own Alice animation. Continue learning about the features of Alice in other tutorials on the site. Welcome to the Alice world!
Exhibit 2: Light Tutorial

The light tutorial is part two of a three part topical series called “Lights, Camera, Action!”. The focus of this tutorial is how to use light to enhance the story that is being told in the Alice virtual world. The tutorial encourages the creator to think of the Alice world as a stage and the scene to animate is a dance between a ballerina and a nutcracker prince. The storyline was chosen to specifically appeal to girls who tend to use Alice more to tell stories in their worlds rather than to build games. It goes through different light objects that can be found in the Alice library as well as introduces some tricks and special effects such as fog.
Lights Camera Action!
Part 2: Controlling Light in Alice

By Elizabeth Liang
under the direction of
Professor Susan Rodger
Duke University
June 2009

Introduction

Download the Alice World that goes along with this tutorial.
You will be learning about how to manipulate light in Alice by changing the properties of the light in
the world as well as adding your own light objects.

The Alice world begins with our ballerina and her nutcracker prince
in the middle of a winter wonderland. The stage is set for
them to dance around each other
before the play ends and the
dreamer awakes. We will first
darken the stage then add spotlights
so that the scene really looks like a
ballet being put on. The characters
will slowly disappear as the
background changes to dawn.

Step 1: Darkening the World

Changing the atmosphereColor will darken the sky and the
ambientLightColor will set a shade over the whole world. We
want to blur the background between the ground and sky so
that it looks like the curtain at the back of the stage. We can do
this by editing the fogStyle.

Go to the properties pane of the
world.
Change the atmosphereColor
and ambientLightColor to black.
Change the fogStyle to distance.

Step 2: Adding Spotlights

Now we are going to add spotlights to the scene to highlight the players of our ballet as well
as the stage.

Click on add objects and go to the
folder in the gallery called Lights.
Find the class StageSpotLight and
add an instance of it to your
world.

Use the object moving tools to
rotate your light and move it up so
that it illuminates the stage from above.
Add two more spotlights. Move them closer and down and then tumble them so that they point at the two objects on the stage.

You should now have three spotlights in your world, one pointing at the stage, one at the blueBallerina, and one at the toySoldier.

Now we want to hide these spotlights (but not their light) from the scene.

Go to the properties tab for each of the spotlight objects and set the isShowing property on each of them to false.

Once the lights are set up we can start the dance!

Click done to go back to the main window.

Go to the methods for the blueBallerina. Drag in and drop the blueBallerina turn method into the method editor for world.myFirstMethod. Select left 1 revolution.

Click on more asSeenBy toySoldier (the entire toySoldier) to make her circle around her partner. Do the same for the toySoldier but make him go around the blueBallerina.

Click Play to test your world.

You will notice that the lights do not follow the dancers, let’s fix this.

To make the lights follow the dancers, we need to change the vehicle property of the two spotlights in the front.

Go to the properties pane of the spotlight on the left and set the vehicle to toySoldier (the entire toySoldier).

Do the same with the spotlight on the right except setting it to blueBallerina.

Play your world and see if you can see the lights follow them as they spin around each other.

We can now add another object from the light folder, a lightbulb, which will give us a way to change the color of the light in the scene.

Click on add objects and in the Light folder, add an instance of the class Lightbulb.

Click on the newly added Lightbulb in the object tree and in the properties pane, set the isShowing to false in order to hide the physical bulb but not the light.
**Step 6: Changing Color**

Click on the lightBulb in the object tree and go to the properties pane. Drag and drop the color property into the method editor at the end. Select blue.

Click on more... and select duration, 2 seconds. Play the world to see the color of the light around the stage change to blue.

Repeat this three more times for a total of four color change calls. Select any color you choose. Now we need to make the light changing and dancing happen together.

**Step 7: Loop**

Since we want to repeat the dancing motion until the lights stop changing, we are going to use a loop.

Drag the Loop from the bottom of the method editor into the top of the method. Select 5 times. Drag the two turn methods into the loop.

**Step 8: Do in order & Do together**

Drag the Do in order right after the loop and put each of the lightBulb color change methods into it.

Add one more color change into the Do in order but this time set it to no color so that we can change the scene back to normal.

**Step 9: Dawn**

In the Nutcracker story, Clara the ballerina wakes up to find that it was all a dream underneath the Christmas tree. We want to create a new world level method that controls the light so that it looks like dawn is coming and the characters are fading away as if it were a dream.

Go to the methods pane of the world and click on create new method. Name this method dawn. You should see a new method called dawn appear in the method list of the world methods pane as well as in a new tab in the method editor.
**Step 9: Dawn**

In order to mimic the colors of sunrise, we are going to change the atmosphereColor.

Go to the world properties pane and drag in the&color property into world.dawn.

Select a color from the menu. If you don’t see the color you want, click other and a panel of colors will come up. Select the color you want and press OK.

Repeat this with a few more colors.

**Events**

Create new event

When the world starts, do **world.dawn**

Change the starting event in the events editor to **world.dawn** and press play to see the colors change.

Right now it doesn’t look very natural so let’s change the duration to 2 seconds the the style to abruptly. You can do this by clicking on “more...”.

**Step 10: Opacity**

Drag the dawn method from the world myfirstmethod pane into the end of the Do together block.

In order to sync the fading out with the coming of the sun, set the duration for each of the opacity calls. You may need to adjust the length of the duration in order to coordinate it with the dawn.

Do the same for the toySoldier. Change the events editor so that when the world starts, myfirstmethod plays. Play the world to see the dancers fade out at the end.
Now we can use the fog and atmosphere to create a curtain effect. Go to the world properties pane to set the fogFarDistance to 1 meter. You will see the screen go black. Drag the fogFarDistance into the beginning of the method and set it to 256 with a duration of 6 seconds.

At the very end of the method. Set the atmosphereColor back to black and the fogFarDistance back to 1 meter with a duration of 2 seconds.

Congratulations! You have now finished the tutorial. Light manipulation is a powerful tool in Alice that can be used to change the mood and add special effects to your scenes. In the same way that colored light and spotlights are used in theaters to highlight certain objects, you can do the same in Alice with the tricks that we have taught you here. Feel free to explore other ways of changing light on your own!
Exhibit 3: Alice Gallery Folders

The Alice Gallery Folders tutorial was developed after a teacher at one of the workshops asked about how to create her own folders in the Alice library and fill them with objects so students would be able to easily find either objects that she modified or wanted to group together for a particular assignment or project. The tutorial is a support tutorial for teachers and most of the tutorial involves following instructions to modify the Alice file system rather than doing anything in particular within Alice. While this is not an intended way for Alice to be modified and used by its creators, the flexibility of editing the library of objects allows teachers to save time in the classroom.
Making Folders
Adding/Editing Local Gallery Folders

By Elizabeth Liang
under the direction of
Professor Susan Rodger
Duke University
June 2009

Step 1: Finding the Folder

This tutorial will show you how to create and edit folders within the local gallery. This skill is useful when you want to limit what objects the creator can work with within Alice or help the creator find certain objects more easily.

The first step is to access the contents of your Alice folder. You should look wherever Alice is installed or running on the computer. The folder should be named Alice 2.2. Open it.

For Macs, Alice is an application after you install it. If Alice is not in your Applications folder, find where you downloaded and installed Alice. You should see an Alice 2.2 icon.

Right click and select Show Package Contents.

Step 2: The Gallery Folder

Find the Required folder and open it. Find the gallery folder and open that to go to the list of local gallery folders.

For Macs, the Required folder is inside the Contents folder.

Each folder is a sub folder in the local gallery, some with more sub folders inside.

Each folder contains .a2c files which are Alice object files and a directoryThumbnail.png file which is the image that appears on the folders in the local gallery.

Step 2: The Make Up of a Gallery Folder

If you open Alice and compare the folders you can see how Alice reads in each of the individual components of a gallery folder. Bugs and Dinosaurs do not contain directoryThumbnail.png files so they appear as blank folders.
Step 3: Creating a Folder

Create a new folder inside the gallery folder. Rename it Assignment 1.

Now if you restart Alice and go into the local gallery you can see the newly created folder. Because there is no directoryThumbnail.png or .a2c files, the folder is empty and has a blank folder icon.

The next step is to fill the folder with the objects we want. You can repeat this process to create new sub folders within Assignment 1 (for example Part 1 and Part 2).

Step 4: Filling a Folder

In the finder window, find and copy an object in the local gallery you want in the new folder. In this example we will make a copy of the Penguin.a2c file from Animals.

Once the .a2c file is copied to the clipboard, go to the Assignment 1 folder and paste it inside. Repeat this process to put the RandomGuy2 from the People folder and SeeingStars from Special Effects.

MAKE SURE YOU COPY AND DO NOT SIMPLY DRAG THE ITEM OVER

Step 5: Directory Thumbnail

If you do not like the blank folder default you can create your own thumbnail. To create a directory thumbnail for the new folder open up an image editing program such as photoshop or paint.

Create a new document that is 128 pixels by 128 pixels and fill it with the icon that you want. Ideally the background is either transparent or a tan color. If your editing program doesn’t support this that’s ok.

Save it as directoryThumbnail.png and move it into your Assignment 1 folder.

Step 6: Testing in Alice

Restart Alice and check the local gallery to see your folder with the image and the three Alice objects you added into it.

Congratulations! You have finished this tutorial. While you cannot add new objects to the local gallery, you can add new folders and subfolders with copies of the objects that already exist. Use this trick to help form new groups and sub groups of objects to fit the worlds you want to create.
Exhibit 4: Sharing Alice

The Sharing Alice tutorial primarily teaches students and teachers how to export their Alice world as a video so that it can be more easily shared with others. The primary barrier to sharing Alice worlds is that the recipient needs a copy of Alice in order to open and play a .a2w file. By exporting the world as an animation, the recipient will not be able to interact with the world fully but at least they will get a sense of what the world is like. Although Alice is free, many computers, especially at the homes of the students, will not have Alice and the students want to be able to share their creation with family and friends.
Sharing Alice
Exporting Movies, Code, and Web Pages

Hug_World
Created by Lewis Carroll

By Elizabeth Liang
under the direction of
Professor Susan Rodger
Duke University
July 2009

Part 1: Exporting a Web Page

To export your world as a web page click on File in the upper left corner of Alice and select Export As A Web Page...

A window will pop up, fill in the fields and then click Save.

The webpage will contain a simple video of the Alice world. The Size in browser will set how large you want the video to be.

Click on Browse to set where you want to create the directory for the webpage.

You can also show the code for the world on the webpage if you like. More about creating a webpage of the code later.

Part 1: Exporting a Web Page

Open the newly created folder in the directory you saved it in. You will see three files inside: the .a2w Alice World, the .html file with the same name as the directory, and aliceapplet.jar.

Open the .html page in a browser to see the webpage Alice created for you. The applet will convert the Alice world into a video with a simple player. Try this out, you may need to download additional files for it to work.

When you upload the page, upload the entire folder and link to the .html file.

Note: The video on created on the webpage is not an accurate translation of your Alice world. You will find that sometimes nothing will happen or only certain methods and calls get played. To get the full animation we need to export the world as a video.
Part 2: Exporting a Video

Once again click on File and this time select Export Video...

Alice will prompt you to save again before opening a new player. Name the movie and then select Record. The world will start playing.

Part 3: Exporting the Code

To get a document with the code, go to File and Export Code for Printing...

You will see a list of all the methods and events and functions used in the world. Select the ones you want to see (events and all open tabs get added automatically) and use browse to create an html document to save the code as. Click Export Code when you are finished.

Part 2: Exporting a Video

When the animation is done, click Stop Recording next to the Record button. Then click Export Video.

Now you should see a .mov file appear in the directory you saved your world in.

Note: Not all computers will be able to watch a .mov file. You can use another program to convert the file into other video formats such as .avi and .mp4. For Macs the movie will appear sped up. You can edit the duration of methods in Alice to fix this.

Part 3: Exporting the Code

Open the newly created .html file and you can see that all the code is laid out on the page just as it is normally in Alice.

You can print this document if you want to make a hardcopy of the code or you can upload it as a webpage.

That’s it for this tutorial! Alice world movies can be uploaded onto sites such as YouTube. The .html files you created can be edited with any .html editor to customize just like any other webpage. Go and share your Alice creations!
Exhibit 5: Alice Pong

Many of the teachers at the follow up workshops said that their students expressed interest in learning how to make games using Alice. While individual elements are taught in the topical tutorials such as a timer, score, and collisions, there was no game tutorial that showed students how to build a game from beginning to end. Alice Pong is a simple game to play but in building it, students learn a variety of techniques common to many types of games such as how to check if things collide using functions in Alice, how to use BDE events to control the game loop and player interaction, and when to use variables to keep track of state in the game.
Alice Pong
Recreating Pong in Alice

By Elizabeth Liang
under the direction of Professor Susan Rodger
Duke University
June 2010

Introduction
The goal of this tutorial is to create a two player pong game in Alice.

Two different people will be able to move paddles up and down (each player gets his/her own paddle) in order to try and keep the ball in bounds. A player wins when the other player is unable to return the ball.

After the game ends, the game will reset and they can play each other again.

In the process, we will show you different ways of doing collision detection in Alice as well as two different ways of using keys to interact with a world. Variables will be used to keep track of state in the game. A BDE event will be used to put together all of the methods.

Step 1: Setting Up the Game
Start with a space template. In order to make the screen all black we will move the ground out of the picture.

Move the ground down until you cannot see the blue and red lines. Then move the ground using the turn objects forward and backward tool and turn the ground until you can not see it. Set the ground isShowing to false.

Step 1: Adding Objects
Now from the shapes folder, add in two boxes and a sphere.

Use the positioning tools to move the boxes to the sides of the screen, turn them to face the screen straight on. You will need to resize the sphere and boxes to be smaller. Use quad view to make sure the sphere is in between the two boxes. Once it is in the center, move the sphere up so that it is in the middle of the main screen.
Now from the shapes folder, add in a halfCylinder. This will make the invisible top border of our game.

Resize and move it so that it forms a ceiling for the game. Make sure that it is over the sphere and near the two boxes.

Set the isShowing property for the halfCylinder to false to hide it.

Add in another sphere object. We will use this as an invisible object that holds the reset point of the ball.

Right click on sphere2 in the object tree and under methods, have it move to the sphere. Set the isShowing to false.

In order to make the code less confusing, we need to rename the objects in the tree. To rename an object, right click on it and select rename.

Rename sphere2 to resetPoint and sphere to ball. Rename the halfCylinder to topBorder. Click on each box to figure out if it is the leftPaddle or rightPaddle. Name each accordingly.

In order to intuitively move the ball and paddles we will orient them to the camera. By doing this, our right becomes the ball's right and our left becomes the ball's left etc.

Right click on the ball in the object tree, select methods, ball orient to, camera.

Do the same for the rightPaddle and leftPaddle.

Changing the orientation is helpful for objects where right and left is not obvious (like the ball and the paddles).
Step 2: BDE Review

Events

- [create new event]
  - When the world starts, do
  - world.my first method

- While <None> is true
  - Begin: Nothing
  - During: Nothing
  - End: Nothing

- While something is true
  - Let the mouse move <objects> - mouse move the camera
  - Let the arrow keys move <subject>
  - Let the mouse orient the camera

The reason BDE Events are useful for games is that we can have the code run as many times as we need to until the game ends. We do not know how many times the code will need to run until the game is won or lost and a BDE takes care of that for us.

We can check if the game is still going in the conditional and then as long as the game is going (no one has lost) the during code will keep playing.

The bulk of our game will be run through a BDE event. To create a BDE event, click on create new event and select While something is true.

A BDE is like a while loop for events. It will kick in every time the condition becomes true with whatever is in the Begin portion of the event. It will then do the during portion and continue until the condition is no longer true. Then it will do the code in End. For a more in-depth review see the “Lights Camera Action Part 3 BDE tutorial”.

Step 2: World isRunning

- [create new variable]
  - world.isRunning = false

Under world properties, click on create new variable. Name it isRunning and select Type Boolean and Value false.

This variable will keep track of the state of the game (whether people are playing or not). It starts off at false because we want the players to start when they are ready. We will set it back to false after a player has lost.

Step 2: World isRunning Continued

Drag the new isRunning variable into the condition of the BDE next to While.

Now create a new world method called gameLoop.

Step 2: The Game Loop

Drag gameLoop into the During portion of the BDE.

The game loop is what we want to keep repeating while the game is running. The game loop will be unique for each game. In this game we want to do three things in our game loop, each in its own method.

These are the methods we will be creating next:

World.CheckBounce will check if the ball needs to bounce off either the walls or the paddle.

Ball.update will move the ball to its new place based on the velocity of the ball.

World.checkEnd will check if the game ends. Basically this checks if the ball is past either of the paddles and off the screen.
Step 2: Check Bounce

Create a new world method called checkBounce. This is where we will put code that checks if the ball should bounce and simulate a bounce.

In checkBounce, drag in a Do together and then drag in four separate If statements (one for each bounce situation). Adding comments like the ones shown at right will help us keep track of what we are doing.

Step 2: Check Left Paddle

The first thing we will check is if the ball is colliding with the left Paddle and therefore should bounce off. Under world’s functions, drag in a < b into the condition of the first If statement. Select 1 twice.

Find the ball’s functions. Drag in a ball distance to function over the first 1 and select the leftPaddle.

Step 2: X & Y Velocity

When the ball bounces, the direction of the ball changes. We need to keep track of the ball’s directional speed (velocity) in the game. Because this state is specific to the ball, we will create two variables in the ball object (for a review on variables, see the timer and score tutorials).

Click on the ball in the object tree and click on create new variable under properties.

Name the variable xVelocity, make sure it is of type Number and set the Value to 0.6.

Do this again to create a yVelocity variable.

A positive xVelocity means the ball is moving to the right. A positive yVelocity means the ball is moving upwards. Negative velocity means the reverse direction (left, down).

Later, by changing the 0.6 value you can control how fast the ball moves in the game. Increasing the number moves the ball faster and lowering makes the ball move slower.

Step 2: Check Left Paddle Continued

Drag the ball’s width function over the second 1.

Click on the arrow next to it and add 0.1

The 0.1 here gives a little buffer to our collision detection. It helps the bounce look more natural.
Drag the `xVelocity` found in the ball's properties into the first part of the if statement. Select set value, expressions, ball.xVelocity.

Do the same thing again in the second if statement, the one for the right paddle. This time instead of ball distance to leftPaddle use ball distance to rightPaddle. The finished code for the two statements are shown.

Click on the arrow next to the second ball.xVelocity. Select math and xVelocity * -1 in order to reverse the direction of the ball.

This is the final code for the first if statement in checkBounce.

To check if the ball should bounce off the topBorder of the game, we will use the ball distance below topBorder function. We will check that that is less than the ball's width. Instead of changing the xVelocity, change the yVelocity.

This is the final code for the if statement that checks for bouncing off the top.
### Step 2: Check Bottom

The check for the bottom is similar except we use the **ball distance above ground** function. The final if statement in the method is shown below. You are now done with checkBounce!

After we finish the game loop and you get a chance to test the game, you can clean up the bounces by modifying checkBounce. Just like we added 0.1 to ball’s width in the left and right checks, you can add (if the bounce seems to be bouncing too late) or subtract (if the bounce is coming too early) small amounts to the appropriate conditional check to make the bounces more realistic.

// check bottom

if (ball:ball.distance above > ground) more...

ball.vVelocity = set value to (ball.vVelocity = -1) more...

else (Do Nothing)

### Step 2: Calling Check Bounce

Click on the world.gameLoop tab. Click on the world in the object tree. Drag checkBounce from the world methods into world.gameLoop.

### Step 2: Creating ball.update

Create a new object method under **ball** and name it **update**. This is where we will move the ball around the screen.

Note: ball.update is an object method because it only deals with the ball.

### Step 2: Update the Ball

Drag in a **Do together** into the method.

Drag in two **ball move** commands into the do together. One that moves the ball right 0.25 meters and one that moves the ball up 0.25 meters.

Set the **duration** for both commands to 0.1 seconds.

Each time the ball updates the ball will move a small amount, by looping this in the game many times, the ball will look like it is moving continuously.
Step 2: Update the Ball Continued

Do together
- ball = move right = 0.25 meters
- ball = move up = 0.25 meters
- duration = 0.1 seconds

Click on more and set the style to abruptly.

Click on the arrows right after 0.25 meters and select math and 0.25 * and expressions, ball.xVelocity for the move right and ball.yVelocity for the move up. This will allow the velocity to influence movement. Below is the final code for ball.update.

By changing the style, we will make the movement of the ball smoother.

Step 2: Calling Ball Update

world's details
- properties
  - ball
  - ball.xVelocity
  - ball.yVelocity

- methods
  - move
  - turn
  - roll
  - resize

world.gameLoop
- No parameters
- No variables
- update
  - ball.update

Call ball.update in the gameLoop by clicking on the world.gameLoop tab. Drag ball.update into the code.

Step 2: Creating checkEnd

world's details
- properties
  - ball
  - leftPaddle
  - rightPaddle

- methods
  - myFirstMethod
  - gameLoop

- functions
  - checkBounce
  - checkEnd

Create a new World Method called checkEnd. This is where we will check if the ball has gone out of bounds and if so, stop the game.

Step 2: Check End

Ball's details
- properties
  - ball
  - leftPaddle
  - rightPaddle

- methods
  - myFirstMethod
  - gameLoop

- functions
  - checkBounce
  - checkEnd

- spatial relation
  - ball is to the left of
  - ball is to the right of

Drag in two if statements into world.checkEnd. One to check if the ball is to the left of the leftPaddle and one to check if the ball is to the right of the rightPaddle. You will find these functions in the ball's functions under spatial relation.

Unlike the distance functions that we used earlier in checkBounce, these functions check if the ball has passed by an object without colliding with it. We can use this to test if the ball has passed the paddle without bouncing, therefore assuming the player missed it.
Step 2: Check End Continue

If the player has missed the ball, we want the game to end. Drag the isRunning variable from the world into the two If statements. Set the value to false to end the game. The method checkEnd is now finished.

Step 2: Calling Check End

Call the checkEnd method in the gameLoop by dragging it in from the world's methods.

If checkEnd turns the game off, we must have a way of turning it on. Create a new When a key is typed event. Set the key to Y and drag in isRunning set the value to true. The game will turn on when the player presses Y.

Step 3: Move Paddles

Create a new world level method called movePaddle. This is where we will put the code that moves the paddle when the event plays.

Create two new parameters for this method. One that is of type object named paddle and the other of type Direction named direction.

We use parameters so that we can avoid creating repeating code where only a couple things change. In this case the only differences are whether it is a left paddle or right paddle moving up or down. This method is a world method because it can be either paddle object being moved.

Step 3: Move Paddles Continued

In move paddle we have to do three checks. We need to first check if the game is running and then we need to check that the paddles are in bounds. If we didn't do these checks then the paddles would be able to go off the screen and players would be able to move the paddles before the game started.

Drag in an if statement. Into that, drag in another two If statements nested inside the first.

Into the first condition drag in the isRunning world variable.

Drag the direction parameter in, setting it == up in the second If condition and == to down in the final If condition.
Step 3: Move Paddles Continued

Under the world’s functions, find the both a and b function under boolean logic.

Drag them into both conditions and select true. We now need to check if the paddle is in bounds.

Step 3: Move Paddles Continued

Click on the rightPaddle in the object tree and find the list of spatial relation functions.

Drag in rightPaddle is above ground into the first condition and rightPaddle is below topBorder into the second condition.

Drag the paddle object parameter from the top of the method over both instances of rightPaddle so that it no longer specifies which paddle is being checked in the method.

Step 3: Move Paddles Continued

The next part of this game is to create events to control the interaction. There are four controls total, Up + Down for each player.

Create four new When a key is typed events.

The right player will use the up and down arrow keys and the left player will use the letters E and D.

In order to make it so that the paddles will move when the player holds down the key, change the events to While a key is pressed by right clicking on the four events.

You will see that this changes the When event to a When event that loops and has a BDE.
Drag in the `movePaddle` method into the During portion of each While a key is pressed event.

Select `rightPaddle` for the first two, up then down. Select `leftPaddle` for the last two, up then down.

Now our controls are done!

Finally we can test the game! Press Play and then Y to start the game.

In addition to trying to play and move the paddles around, make sure the paddles do not go off the screen if you continue to press up and down.

Make sure the game stops when someone looses, to continue testing, just press restart.

You may notice that your game, while working, isn’t polished. If the paddles are stopping too early or too late, try moving the ground and top border up and down to decrease or increase the size of the screen. (Remember to set them to isShowing before you move them.)

If the ball seems to be bouncing too early or too late off the borders or the paddle, add or subtract buffer from the checkBounce method. You will need to continue playing with and testing your game to make it better.

We know that pressing Y will start the game but for those that do not know, we need an instruction that tells them to. We will do this with 3D text.

Create a new 3D Text object that says “Press Y to begin”. Position the object in the center between the two paddles so it is easily readable.

Create a new world method called `reset`.

Click on 3D Text in the object tree and drag the isShowing property into the new `world.reset` method. Set isShowing to false.

To make replaying the game easier we will write two methods so that the players can restart the game after the game ends.
**Step 4: Reset**

In addition to removing the Press Y instruction, we want to move the ball to the resetPoint and make it visible again.

Drag a `ball.moveTo resetPoint` method into the code. Drag the `ball.isShowing` property into the code and set it to `true`.

**Step 4: Finishing the BDE**

Now we will call the two methods reset and end in the BDE loop. In the event editor, drag the `reset` method into the Begin portion of the While event and the `end` method into the End portion of the While event.

**Step 4: Instructions Continued**

The last thing to do is to make a billboard with the instructions for playing the game on it and have it appear when the world starts and then disappear.

Make a billboard with the instructions (or download it from the tutorial site) and import it into the world. Resize and move it forward so that it covers the screen.
Click on the world.my first method tab in the method editor. Here we will put the code to have the instruction billboard we put in show and then disappear.

Click on the billboard in the object tree. Drag the isShowing property into the code and set it to true. Drag in a Wait and set it to 5 seconds to give the players enough time to read the board. Then drag in another isShowing property and set it to false to make it disappear.

Press Play to play Pong!

Congratulations! You have now finished the tutorial. Games can be difficult to do in Alice because they put together many different concepts that you have learned. You can use the techniques that you have learned from this tutorial (such as collision detection and game loops) and embellish them to create more complicated games.

There are many games out there that follow a similar pattern. See if you can add a score (see the score tutorial) to keep track of how many times a player has won or lost in the session. (Hint: you will need two variables to keep track of state—one for each player—and update them in the checkEnd method). Can you make a four player pong game?

For a bigger challenge, see if you can use the idea of collision detection to create a break-out game where you try to hit blocks and clear the screen. Use your imagination!
Exhibit 6: Treasure Hunt

A teacher attending the follow up workshop mentioned that she wanted to learn how to create a click-adventure game in Alice. Click-Adventure games are a combination of game and story. The user clicks on objects in the screen to figure out how to put together the right combination of elements to complete the puzzle. In this case the man is trying to find the treasure on the island but must first get the map. The map is behind the ship so the ship must be moved but in order for the ship to be moved, the light house needs to be turned on. Many of these games lack the dynamic real-time elements that other games have but they can become complicated with the many different outcomes that need to be accounted for. State variables are used to help keep track of what events can be triggered and a hidden list is used to allow users to “pick up” objects.
Treasure Hunt
A Point & Click Adventure Game in Alice

By Elizabeth Liang
under the direction of
Professor Susan Rodger
Duke University
July 2010

Introduction
This tutorial shows you how to create a point and click adventure game in Alice. If you are not familiar with point and click adventure games, I suggest that you download the finished world for this tutorial and play it to get a feel for what it will be like.

Point and click adventure games will rely mostly on click events to direct the story.

As we build the game, we will review scene change, parameters, functions and if statements. If any of these concepts are new to you, consider going through the topical tutorials on those topics before beginning this tutorial.

Now, let's get started in building this world!

Playing a Click Adventure Game

For this simple game the solution involves the following:

Click on the light bulb in the lantern to pick it up. Click on the red arrow to go to the dock.

Click on the lighthouse to give it the light bulb and turn it on. Now the ship can move. Click on the sailboat and Click on the map it leaves behind.

Click on the red arrow to go back to the island. Click on the Jack to give him the map.

Click on the X that appears to reveal the treasure!

The fun in adventure games is the puzzle that the user must figure out in order to unlock the ending. In this case the goal is to find the treasure.

Step 1: Setting Up the World

The first step is to set up the world. This means positioning all the objects that we will need in our story.

There are two scenes in this game, an island and a dock. We will set up the island scene first then the dock scene. If you want to have more than two scenes in your game, review the scene change tutorial to learn how to add more scenes.

Create a new world with a water template.

This game does not have the superGround.a2c object imported unlike the scene change tutorial because we will only be using the water background. If your game requires different ground textures, you will want to see the scene change tutorial to do this.
Step 1: Island Scene Objects

Add an island from the environments folder, a sailboat from the vehicles folder, and jock from the high school, students and teachers folder to the world.

Position the objects roughly as shown in the image.

If you cannot find an object in the gallery, use the Search Gallery tool.

Step 1: Lantern & Light bulb

Add a light bulb (from the light folder) and a lantern (from the objects folder). You’ll notice that the light bulb is in the island.

Right click on the light bulb in the object tree, select methods, move to lantern, flame. This will move the light bulb to the lantern so that we can see it.

Step 1: Lantern & Light bulb Continued

Resize the light bulb and move it so that it is in the center of the lantern.

Glue the light bulb to the lantern by clicking on the light bulb in the object tree and going to its properties. Set the vehicle property of the light bulb to the entire lantern.

Move the lantern so that is is above the jock’s head and looks like it is being hung in the coconut tree on the island as shown.

Step 1: 3D Text

Create two 3D Text objects, one that says Congratulations! and another that is just an X.

Rename them in the object tree and position as shown.

Add a ToyBox2 object from the furniture folder to the island, resizing it and positioning it on top of the X.

If an object is not visible when you add it, click on the object in the object tree and move it up 1 meter to help locate it in the scene.
**Step 1: Island Arrow**

In the Web Gallery, find the Triangle object in the Shapes folder. If you do not have access to the web, you can use a triangular prism colored red.

Right click on the triangle in the object tree and roll it left \( \frac{1}{3} \) of a revolution. Position it in the upper left hand corner of the screen.

Rename the triangle in the object tree to islandArrow.

**Step 1: Hiding the Treasure**

Set the isShowing property of the sailboat, jock, toyBox2, and the two 3D Text objects to false. This is because we only want those objects to show up later.

Click on the X 3D Text object in the object tree and find the properties panel. Set the color of the X to black. We are finished setting up the island scene!

**Step 1: Dock Scene**

From the object tree, turn the camera left \( \frac{1}{2} \) revolution to find a space to set up the new scene.

Drop a dummy at the camera to save the view. Expand the cameraViews folder and rename dummy to dockView.

To switch between views, simply right click on the camera and set the point of view to the desired view.

**Step 1: Beach Terrain**

Add a Beach Terrain object from the Beach folder into the world. Switch to quad view to position it.

You want to turn/move it from the top view so that the beach looks as shown. The beach should cover the horizon line between the water and the sky of the dock single view. The island scene view should still have open water behind it. Change the camera view back and forth to adjust until it looks correct.
**Step 1: Dock Scene Objects**

Add a **Pier** and **Lighthouse** from the Beach folder to the scene. Use the from the top view to move them into the dock scene and then use single view to position them.

Put the lighthouse on the right hand side of the beach and turn the pier so the wide end is facing the camera. Move the pier to the left hand side of the screen.

Also add a **rock** to the scene. The rock will initially be small. Move the rock to the pier and then resize it and move it to the right, in front of the lighthouse in the water.

---

**Step 1: Another Boat**

Add another instance of the **sailboat** and **jock** into your world. Move the new sailboat (sailboat2) in front of the map. Resize the boat so that the sail covers the map entirely, hiding it from your view.

Move the new jock (jock2) to the boat (sailboat2) in the dock scene.

---

**Step 1: Dock Arrow**

Click on jock2 in the object tree and set the vehicle of jock2 to sailboat2.

Add another **Triangle** and position it at the upper right hand corner of the dock scene.

Rename it dockArrow.

---

**Step 1: A Pirate Map**

The next thing to add is a treasure map. Since there is no treasure map object, we will use a billboard.

Either download the pirate-map.gif from the website for this tutorial or find your own image of a treasure map.

Make a Billboard of this image and set it on the pier as shown in the figure to the right. Hint: Try moving the billboard to the pier and then moving it up and forward.

Now we will hide this map behind a sailboat.
Step 1: Light Beam

To create a beam of light coming from the lighthouse, we will use the flashlight. Add a flashlight from the objects folder.

When moving the flashlight to the lighthouse you will need to resize and check in quad view so that the flashlight is inside the uppermost part of the lighthouse. The body of the flashlight should be covered as much as possible by the lighthouse with only the beam shining outward.

Set the isShowing property of the flashlight to false. You are now done with setting up the world!

Step 2: Change Scene

The first thing to do is to create a changeScene method that we can call when the arrows are clicked on. Create a new world level method called changeScene.

In this method, create a new object parameter named cameraView.

Create a new color variable called skyColor.

Step 2: Change Scene Continued

The first thing to do in the changeScene method is to save the current color before we fade out.

Drag the skyColor variable into the method. Select set value to black. Set the duration to 0 seconds.

Click on world in the object tree and select the atmosphereColor property and drag it over the black box in the code.

Step 2: Using Fog

The next step is to fade out of the scene. Drag the world atmosphereColor property into the code and set it to black. Change the duration to 0 seconds. Drag the world’s fogStyle property into the code and set it to density. Change the duration to 0 seconds.

This activates the fog in the Alice world which we will use to cover our scene so that it looks like it is fading out. Unlike the scene change tutorial, we use fog instead of changing the world’s light to fade out. This is because we have a light object (the light bulb) in the scene and simply darkening the world’s light would not hide everything in the scene. Fog will black everything out.

Finally, drag in the fogDensity property into the code and set it to 1. Set the duration for 2 seconds.
Step 2: Change Scene Continued

Click on camera in the object tree and find the camera set point of view to method. Drag this into the code. Select expressions, cameraView.

Change the duration to 0 seconds.

By using the cameraView object parameter, we will be able to easily switch between scenes without rewriting code.

Step 2: Using Fog Continued

To finish up the method and have it fade in to the new scene, reverse the commands we dragged in earlier. Drag in a set fogDensity to 0 command with a duration of 2 seconds.

Set the atmosphereColor back to the skyColor we saved earlier and set the fogStyle to no fog; both commands have a duration of 0 seconds.

Above is the final code for world.sceneChange.

Step 2: Change Scene Events

Create two new ‘When the mouse is clicked on something’ events.

Set the events to happen when you click on each arrow.

Step 3: List Visualization

The next thing we will do is allow the player to pick up certain objects. To do this we will use a ListVisualization that will store those objects.

In the Visualizations folder, add a ListVisualization object into your world. To initialize the list, click new item once. Then click OK to add it to the world.

Test this by playing your world and clicking on the arrows.
Step 3: List Visualization Continued

Move the listVisualization into the dock scene and bring it forward until the wooden border just touches the bottom of the screen.

Move it off screen so that you can only see half of it as shown.

Set the vehicle of the listVisualization to the camera and set the isShowing property to false.

List visualizations are one of the tools in Alice to help understand how lists work. In this case we will be using it to line up the objects that the user picks up in the game. By setting the vehicle to the camera, we ensure that the objects follow the player around. This is a useful trick for games where you want the user to collect ‘items’.

Step 3: Pick Up Method

Create a new world level method named pickUp.

In this method, create a new object parameter named object. This will represent the object that we want to pick up.

When we click on the object that we want to be picked up, this method will kick in to pick it up.

Step 3: Pick Up Continued

Drag the object parameter into the code after the if statement and select object say.

Have the object say ‘just a regular’ and set the duration to 2 seconds.

Create another object say command after the Else and have it say ‘you have picked up a’. Again, set the duration to 2 seconds.

Make sure that there is an extra space after the last word in each string.
Go to the methods of the listVisualization object and find the insert item at end of listVisualization command. This will both add the object to the list as well as move it onto the visualization.

Drag it into the Else portion of the code, above the object say command. Select expressions, object.

Create two new When the mouse is clicked on something events.

When the lightBulb and pirate-map are clicked on, the world.pickUp method should be called on those objects.
Step 4: My First Method

Create a new world variable called `storeAtmosphere`.

The type should be `color`.

Drag the new variable into world.my first method and set the value to `black`.

Drag the world `atmosphereColor` property over the black box and set the `duration` to 0 seconds.

Step 4: My First Method Continued

Drag the world `atmosphereColor` property into the code and select `black`. Set the `duration` to 0 seconds.

Add in `island say` commands to give the user instructions on how to interact with the world. Remember to set the `duration` so that the text is displayed long enough.

This is the final code for world.my first method. Your directions may vary.

Step 4: Testing My First Method

Before we test we need to change the camera view back to the island. Right click on the camera and have it set point of view to the islandView.

Watch the animation and change the `duration` of the instructions if needed.

Step 5: Lighthouse On

Create a new world method called `lighthouseOn`.

This is the method we will call when we click on the lighthouse. We want the lighthouse to turn on if the user has picked up the lightBulb, otherwise we want to give a hint about what to do.

Drag in an `if/Else` statement and once again drag the `listVisualization contains item` function into the conditional. Select the `lightBulb`.

Click on the objects in this world to find the treasure!
We want the lightBulb to disappear so drag the lightBulb isShowing property into the IF portion and set it to false.

We also want the flashlight to appear. Drag the flashlight’s isShowing property into the IF portion and set it to true.

Now to lighten the sky, drag the world atmosphereColor property into the code beneath the other commands and select expressions, storeAtmosphere.

To make it more interesting, we will have the light beam turn round and round in the lighthouse.

Drag the flashlight into the Events Editor and select When the world starts. This will create an event under the heading “flashlight”.

An object event, like an object method, should only involve the object. If you save the object that the event is tied to, the object’s events will go with it. Since this event only involves the flashlight, it makes sense to have it as an object event.

Right click on the new When the world starts event to change it to a While the world is running BDE event.

Create a new “When the mouse is clicked on something” event. When the lighthouse is clicked, lighthouseOn should happen.

First try clicking on the lighthouse without the light bulb. The jock in the dock scene should give the hint. Then pick up the light bulb and click on the light house. The sky should lighten and the beam of light should appear from the lighthouse.
Drag a **flashlight turn** command into the **During** part of the BDE. Have it turn **left 1 revolution**.

Set the **duration** to **2 seconds** and the **style** to **abruptly** so that there is no pause between each rotation.

Play the world again and get the lighthouse to light up. You should see the beam of light should be spinning as long as the world is running.

---

**Step 6: Sailboat Move**

Once the lighthouse is on, we want the sailboat to be able to move to the island when it is clicked on. If the lighthouse is not on, then we want the jock to give another hint.

Create a new world level method called sailboatMove. Drag an **If/Else Statement** into the new method.

Drag the **flashlight isShowing** property into the condition of the **If** statement.

Drag a **sailboat2 move forward 10 meters** command into the **If** portion of the code, this will move the sailboat off screen.

Drag in a **Do together** and set make the sailboat2 and jock2 disappear by setting the isShowing property for both to false. Make the sailboat and jock on the island scene appear by setting their isShowing property to true. Change the duration for all four commands to 0 seconds.

In the **Else** portion, have the jock2 say that ‘It’s too dark to navigate’. Set the duration to 2 seconds.

We’re now done with this method!

Create a new ‘**When the mouse is clicked on something**’ event to run the sailboatMove method when sailboat2 is clicked on.
Step 6: Testing Sailboat Move

Play the world. Try clicking on the sailboat without the lighthouse on. The jock should give the hint.

Turn the lighthouse on and move the boat away. It should allow you to click and pick up the map. If the map is too big you may need to resize it.

Go back to the island scene and make sure you can now see the jock and sailboat in that scene.

Step 7: Find X

Once the jock is on the island, we want to be able to 'give him' the map so the X will appear and the treasure can be found. If the user hasn't picked up the map, we want the jock to give a hint.

Create a new world level method called findX.

Drag an If/Else Statement into the new method.

Under the listVisualization's functions, drag the contains item function into the conditional. Select the pirate-map.

Set the isShowing of the pirate-map to false and the x to true in the if portion.

In the Else section, have the jock say 'I wish I had a map' as a hint. Set the duration to 2 seconds.

Create a new 'When the mouse is clicked on something' event and run the findX method when the jock is clicked on.

Test the world by clicking on the jock on the island with and without the pirate map.
Step 8: Show Treasure

Once the X is revealed, the user will click on it to show the treasure and end the game. We have one more method to write to do this.

Create a new world level method called showTreasure.

Drag in the toyBox2 isShowing property and set it to true. Then drag in a jock say command and have the jock say ‘There’s the treasure! Thanks for your help!’ Set the duration to 2 seconds. Finally drag in the isShowing property of congratulations! and set it to true. We’re almost done!

Congratulations!

Play your world one last time and test out the story to the end, making sure that the treasure chest appears at the end.

Congratulations on finishing this tutorial and creating a click-adventure game!

Click adventure games become more fun with the more items that you add into the world for the user to interact with. We did just the bare minimum but you can always add other click events to give hints (for example, clicking on island could say that the treasure is here somewhere...).

Create your own adventure game with a new story line. Don’t forget to add click events to interact with the world and if statements to check if the story should be advanced!

Step 8: Show Treasure Event

Create a new ‘When the mouse is clicked on something’ event and run the showTreasure method when the x is clicked on.

These are all the events in this world. You will see that nearly all of them are click events. This is after all a click-adventure game!
Appendix B: Example Science Worlds

Exhibit 1: Hot Spot Volcano & Helium Molecule

The completed Helium Molecule tutorial uses a slightly transparent sphere to mimic the electron cloud and color-coded spheres to represent the different parts of the molecule. The electrons orbit the nucleus when the animation is played.

The Hot Spot Volcano world is narrated by a ‘mad scientist’. He explains the forces and process that goes into creating a Hot Spot Volcano. The animation is advanced when the user clicks the mouse. A counter in the world keeps track of how many clicks have passed and advances the portions accordingly.
Exhibit 2: lac operon

Each colored geometric shape in the world represents a different player in the model. The legend that appears when the user presses ‘I’ describes how the different parts interact. Each player moves randomly and their interactions are set by probaility.

States in the world are represented as variables. The lactose variable is set to true when the slider is pressed and because of this, the repressor protein (blue) leaves. This leaves a space for RNA polymerase to bind to and the transcription method will be called.

When the glucose variable is set to true, the promoter is now free and the RNA polymerase has a lowered probability of binding but even if it did try to bind, the repressor protein is blocking access to the DNA because lactose is low.
The world begins at a pond with six fish. The player is then instructed to take fish from the pond. At the end of every turn new fish appear in proportion to the number of fish in the pond. Depending on how many fish the user takes, the population will change at the end of each round. This data is stored in a list.

After 5 days, the data will automatically be graphed in a bar chart.
Animated Charting
Using the Alice Bar Chart Template World

By Elizabeth Liang
under the direction of
Professor Susan Rodger
Duke University
December 2010

Introduction
The Bar Chart World template world allows you to easily display numerical data using a bar chart in Alice. This tutorial will show you how to use the template world. In addition, extra instructions at the end will show you how to expand on the world.

The first step is to download the barChartTemplate.a2w file from the website. Go ahead and open it up in Alice.

You will see that the world is set up with a bar chart in the middle of the screen that has 5 differently colored bars on top of a grid. Around the grid are labels that display the title, and the meanings and numbers for each axis.

Play the world to get a feel for what happens, notice that the labels change and the bars move along the grid to show 5 different values.

Part 1: The Data
So now that you’ve seen how it works, let’s begin to customize it to show your data. Click on world in the object tree and then click on the properties pane. You will see that there are three list variables in the world’s properties. Click on the grey box that represents the data list.

This will open up the Collection Editor. Edit the items in the list so that it matches your data. Click OK when you are finished.

What if you have more than 5 points of data? Later on in this tutorial we will show you how to add more bars to the chart but for now just work with your first 5 points.

Part 2: The Labels
Our next step is to edit the labels on the chart so people will know that the chart represents. Click on the world methods tab. Click the edit button next to the setUp method. A new tab will pop up in the method editor labeled world.setUp.

The first Do together in the method deals with the titles of the graph and the axes. You will edit the text here to change what labels appear when the world is played. You will also need to set the maxValue for the bars in the chart.
Part 2: The Labels

The second Do together in the method deals with the labels underneath each bar. You can either leave these as numbers or you can edit them to give them meaning.

Play your world to see the finished chart!

For example here, the numerical labels have been changed to text labels that indicate that each bar is tied to a particular day of the week.

As you are changing the text, sometimes your labels will be too big or too small. Feel free to go to the “Add Objects” view and move the 3D text around so that things fit better.

Extra 2: Adding Numbers to the Y Axis

To add a new number label to the Y axis, we will need to create a new 3D Text object. Click on the green ADD OBJECTS button.

Scroll to the end of the gallery and click Create 3D text. Type in “min” for the text and click OK. We will use this to label the minimum value on the axis.

The new object will appear as 3DText in your object tree, rename it to yMin. Expand the labels folder and drag the yMin object into the folder.

Extra: Changing a Bar’s Colors

To change the color of a bar in the bar graph, go to the object tree and expand the folder of the bar whose color you want to change. In this case, we will make the first bar orange.

Each bar is made up of four squares labeled a-d in the object tree. For each of those squares, set the color property to orange.

Changing a bar’s colors can be useful if you want all the bars to have the same color or if you want to color code certain bars in your chart.

Extra 2 Continued

Click on yMin in the object tree and click on the properties tab. Create a new variable called minValue. minValue’s type should be Number and starting value should be 1. Click OK when you are done.

Set the color of the object to black.

Turn and resize the word and move it to the bottom of the y axis as shown.
Extra 2 Continued

Scroll down in `world.setUp` and drag in the `minValue` variable into the code; set the value to 0. Drag in the `text` property directly underneath it and select default string. Replace the default string with the `what as a string` function found in `world functions`. Select `yMin.minValue` from the labels folder. Play the world.

Extra 3: Editing the Grid

The grid in this world is simply a billboard with an image of a bunch of lines that form squares. In order to make the squares of the grid seem larger or smaller, you can resize the billboard, or create a new one with a fewer or greater number of squares.

Download or create an image of a grid (you can find one along with this tutorial called `graph.jpg`).

Edit it in an image editing program such as paint to the correct dimensions.

Import this into the world as a billboard and rename it as "grid".

Position it behind the bars on the chart.

Extra 4: Adding a New Bar

An individual bar in the chart is made from four squares stacked on top of each other.

Find the Shapes folder in the online gallery and add four Squares. Position them to match the other bars.

Group these squares together by right clicking on the object tree and selecting "create new group". Name the group with the bar number you are creating.

Move each square into the new folder and name them `a-d` with `d` being the top square and `a` being the bottom square.

Set the `vehicle` property of `a-c` to `d`. This will glue the squares together.

Extra 4: Adding a New Bar Continued

Each bar also contains a reset point which is simply a Bump object made invisible.

From the same shapes folder add a Bump and rename it `barResetPoint`. Move it into the `resetPoints` folder. Right click on it in the object tree and move it to the "d" square of the bar you are creating.

Set the `isShowing` Property of the reset point to `false`. 
Extra 4: Adding a New Bar Continued

In order to connect your new bar to the rest of the world, you will need to update some lists.

In the World Properties details pane, there are three list variables associated with the world. Edit a list by clicking on the values after the "=" sign.

To the barChart list, add a new item, the "x" square of your new bar.

To the resetPointList, add the new object from the resetPoints folder that you created.
In the atmosphere world, world list variables store values for pressure, temperature, and altitude at each point. These are translated into numbers and bars in the animation with temperature on the left and pressure on the right. Altitude is presented at the bottom. In order to animate the bars at each point in the animation, the code updates the current value for each list item and then resizes the new value compared to the old value. 100 must be added to the temperature since it doesn’t make sense to graph negative values using a bar chart.