2011

Programming with Alice

Madeleine Schep
Columbia College
# Table of Contents

Section 1: Storyboards ................................................................................................................................................... 3  
   Exercises ...................................................................................................................................................................... 5  
Section 2: Setting up the initial scene .......................................................................................................................... 5  
   Exercises ...................................................................................................................................................................... 6  
Section 3: From Storyboards to Code.......................................................................................................................... 6  
   Review ....................................................................................................................................................................... 11  
   Exercises .................................................................................................................................................................... 11  
Section 4: Writing a method ....................................................................................................................................... 11  
   Extra-tips: Styles and copying and pasting ........................................................................................................ 15  
   Exercises .................................................................................................................................................................... 16  
   Extra-tips: dummy objects .................................................................................................................................. 17  
Section 5: Loops, functions and methods revisited ................................................................................................ 17  
   Extra-Tip: Testing code ........................................................................................................................................ 20  
   Saving a new class .................................................................................................................................................... 22  
   Exercises .................................................................................................................................................................... 22  
Section 6: Parameters................................................................................................................................................... 23  
   Exercises .................................................................................................................................................................... 27  
Section 7: Interactive Programs ................................................................................................................................. 28  
   Exercises .................................................................................................................................................................... 32  
Section 8: More events and Conditional statements .............................................................................................. 32  
   Exercises .................................................................................................................................................................... 37
Programming with Alice

Alice is a programming language that allows the user to create 3D-animations. It is designed to teach the student how to program a computer, but in a very different way from most other programming languages. First of all, you don’t have to remember special words or syntax. Secondly, you don’t have to solve computational or text-based problems. Instead of writing lines of code, you will learn to program by creating short animations. Additionally, if we get far enough and have enough time, we could create simple interactive games.

The animations in Alice are not of the highest professional quality, but for many students, it is a good teaching tool that allows full use of their creativity while learning important programming concepts.

You will have some homework assignments in which you need to use the software. If you want to work on these assignments on your own computer you can download Alice from http://alice.org/index.php?page=downloads/download_alice2.2. If you do not want to download Alice on your computer, you can use the Math lab computers to complete your homework.

A few things to know before working with Alice:

- If you want to open Alice file by double-clicking the file in My Computer, you need to associate the files with the program. Instructions will be given in class on how to do it.
- Alice is a great program but it is not perfect. It tends to freeze sometimes. Save often your programs so you will not lose all your work if this happens to you.

Section 1: Storyboards

Programmers write programs to solve problems. A program is a set of instructions for the computer to perform (or execute), written in some special programming language. Writing programs is also known as writing code. The first steps of solving a problem is to understand the problem well and then to design a solution. Once we have the design we can write code.

When programming with Alice, the stages involved in solving a problem will correspond to stages used by professional studios to create an animation. Understanding the problem means developing a clear scenario, i.e. a story to tell. The design will be expressed by a series of frames, called storyboards, which show the major steps in the story. A frame contains an image (for our purpose, it can be rather crude) showing the scene at that point of the story and a brief text describing the action.

For those who feel artistically challenged for the traditional animation storyboard, the design can be a textual storyboard. In this case, the actions are described in brief concise sentences. Rather than being written as the scenario in paragraph style, the text is formatted in such a way that the order and the logic of the actions is made very clear. Any ambiguity or imprecision existing in the scenario should be eliminated in the storyboard. Let’s look at an example that we will follow from scenario to complete animation during the next few lessons.

Scenario: the snowwoman and the penguin

Standing in the middle of a snowfield, a snowwoman greets a penguin approaching on a skateboard. The penguin asks the snowwoman if she wants to try to skate. She agrees. She makes two unsuccessful attempts and each time the penguin laughs, embarrassing the snowwoman. But she does not give up, bids farewell to the penguin, skates away and the penguin runs after her.

3
The first four frames of a possible storyboard for the scenario are shown below.

<table>
<thead>
<tr>
<th>Initial Scene</th>
<th>Scene 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Initial Scene" /></td>
<td><img src="image2" alt="Scene 2" /></td>
</tr>
<tr>
<td>Snowwoman in a field of snow</td>
<td>Snowwoman greets penguin and penguin comes into view on his skateboard.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scene 3</th>
<th>Scene 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Scene 3" /></td>
<td><img src="image4" alt="Scene 4" /></td>
</tr>
<tr>
<td>Conversation between snowwoman and penguin. Snowwoman turns to face penguin after penguin greets her.</td>
<td>Snowwoman tries to skate and falls.</td>
</tr>
</tbody>
</table>

The storyboard needs a few more scenes to reach the conclusion of the story. A textual storyboard for the same scenario follows.

**Textual Storyboard: Snowwoman and Penguin**

1. **Initial Scene:** Snowwoman is alone in the middle of a snowfield.
2. Snowwoman greets penguin.
3. Penguin comes into view on a skateboard.
4. Penguin jumps off skateboard
5. Penguin greets snowwoman.
7. Penguin asks snowwoman if she wants to try to skate.
8. Snowwoman agrees to skate.
9. Repeat twice:
   a. Snowwoman jumps on skateboard and then falls.
   b. Penguin laughs.
   c. Snowwoman stands up red in the face.
   d. Snowwoman’s color returns to normal.
10. Snowwoman turns towards penguin and tells him bye.
11. Snowwoman skates away out of view.
12. Penguin runs after snowwoman and yells for her to wait (at the same time).

Notice that the textual storyboard clarified imprecision in the scenario. For example, in the last sentence of the scenario “snowwoman bids farewell to the penguin, skates away and the penguin runs after her,” we can choose to interpret that snowwoman skates away and immediately (i.e., at the same time) penguin runs after her, or as was done in the storyboard above snowwoman skates away and disappears from view and then penguin runs after her.

**Exercises**
1. Look into Alice gallery and choose three characters from it. Write a short scenario for these three characters.
2. Design the first four frames of the pictorial storyboard of the scenario you developed in exercise 1 above.
3. Design a complete textual storyboard for the scenario you developed in exercise 1.

**Section 2: Setting up the initial scene**

Before we can program an animation we need to set up the initial scene. We will first choose a template, then add and position characters and objects in this template. The PowerPoint SettingUpWorlds shows the tools used to set up the initial scene.

The initial scene you set up should look as depicted in Figure 1. Not seen but added to the world are the skateboard and the penguin. We set up the scene so that the penguin has its right foot on the skateboard and the skateboard and the penguin are facing SnowWoman at the start of the scene. We also set the vehicle property of penguin to skateboard, so that when the skateboard moves, penguin moves with it.

The upper-left pane of the Alice window contains the object tree which displays all the objects added to the world, including those not seen in the initial scene. In our example a skateboard and a penguin were added to the world and appear in the object tree shown at left, but they are out of view in the initial scene. Adding objects in the initial set-up allows interacting with those objects later in the animation. The object tree also contains camera, light, and ground, which by default, are objects that are part of the world when it is first created. You should never delete the camera or the light from the world otherwise the world becomes a very dark place indeed. The root of the tree (which is on top!) is the world object.
Most objects added to the world have subparts. To display the subparts of an object on the object tree, we click on the + next to the name of the object. The object tree expands displaying the subparts below the object to which they belong. The subparts are object themselves and we will be able to interact with them. On our example tree, if we click on the + next to snowWoman, the object tree expands as shown in Figure 2. Note that subparts can have subparts themselves, so that we can repeat the process. We can click on the + next to head and the subparts of the head of snowWomen would be displayed below head.

Exercises

4. Create the initial scene for the storyboard you created in exercise 3 page 5.
5. Set up a scene with three penguins on top of each other. Use the Quad view to make sure that the penguins are really on top of each other.

Section 3: From Storyboards to Code

We have completed the design phase (i.e., we’ve designed a storyboard) and we have created the initial scene. We are now ready to write code to animate the characters. We need to follow the storyboard and translate each scene or each item of the textual storyboard into code. Before we launch into this task let us explore a little more the Alice interface shown in Figure 3.
At the top of the interface, the middle pane shows the initial scene with the camera tools below it, the left pane shows the object tree and the right pane is the event pane that we will discuss later on. On the bottom of the interface, the right pane is the editing pane in which we will write code. At the bottom of that pane are a number of tiles such as “Do in order” and “Do together” that are words with special meaning, called keywords, which we will use in our code. Finally, the bottom left pane contains the details of the objects and includes three tabs: properties, methods and functions. To access the details of a given object, e.g., snowWoman, the object must be selected in the object tree by clicking on it (single left-click; the object tile turns purple when it is selected).

The properties of an object describe the object: its color, opacity, vehicle, etc. The methods of an object give the object its behavior: it can move, turn, talk… Objects come with many built-in methods. We will also learn how to create new methods giving new behaviors to objects. Functions are used to answer questions about the object. We will study functions later on.

We are now ready to implement our storyboard. The code we write will be written in a world method called my first method. It is the method that is played when the animation is run. The first item after the initial scene is “snowWoman greets penguin.” We’ll make snowWoman say “Hello!” Figure 4 shows some of the methods for the snowWoman object. Notice that since all these methods are for snowWoman they read snowWoman move, snowWoman turn, etc. If we look at the penguin methods, we would get penguin move, penguin turn, etc.

There is a method snowWoman say, this is of course the method we will use to make her say “Hello!”

But first, since the action described in the storyboard will be carried out one after the other or “In order”, we will drag the tile Do in order from the bottom of the editing window and drop it at the top of the editing window (where Do Nothing lies). This creates a Do in order block in which we can add commands that will be executed sequentially (one after the other) when the animation is played. We then drag the tile “snowWoman say” to the right in the editing pane within the In Order tile (we replace the “Do nothing” by the tile “snowWoman say”). When we drop the tile, we can choose what to say. We can choose “Hello”, one of the pre-defined choices, or click on other and type “Hello!” ourselves. The code looks now as follows.

Click on Play (button above the object tree) and observe your first program run. You might find that the text bubble disappears a little too fast. By default all methods run for one second but we can change the duration by specifying another one. The line of code snowWoman say Hello! is followed by more followed by

---

1 To set it apart from the rest of the text, Alice code will be written in the font new courier.
a down arrow \( \nabla \). A \( \nabla \) signals that a change can be made. For example if you click on the \( \nabla \) following Hello! You can change the text that snowWoman will say; if you click on the \( \nabla \) following snowWoman you can change the object (character) who will say Hello! If you click on the \( \nabla \) following more, one option is duration. Select it by pointing the mouse to the word duration, then several options for the duration appears. You can select a pre-defined option by clicking on it or click on other and type the appropriate number in the dialog box. Let's select 2 seconds.

We are going now to implement step two of the storyboard: Penguin comes into view on a skateboard.

We will first implement an easy version of this step. Let us assume that the penguin has given himself a big push and he is simply gliding toward snowWoman. Because, as set up in the initial scene, the skateboard is the vehicle of penguin, penguin will follow all movements of the skateboard. We can therefore concentrate on making the skateboard move towards snowWoman. Since we set up the initial scene so that the skateboard and penguin are facing snowWoman and no characters have moved yet, when the skateboard moves forward it will move in the direction of snowWoman, and so will penguin. Right now we will just be satisfied to experiment with numbers to see by how many meters the skateboard should move. We will soon see a better way to solve this problem. So we get the code:

Code 2

Let's discuss those few lines of code. Do in order indicates that the actions included in the block are to be executed one after the other or sequentially. By default, commands in a program are executed sequentially (i.e, if you omit the Do in order block the methods will be executed in order; but using the Do in order block has advantages). The two lines that follow are method calls; first a call to the method say for the object snowWoman, then a call to the method move for the object skateboard.

Caution: if you try to use the method move toward instead of move, this method moves an object center toward the center of the target object, which can result in unexpected effects. For example if we used it to bring the skateboard toward snowWoman, and since the center of the skateboard is between its wheels and the center of the snowWoman is at the center of its body, the skateboard would start flying! When you want to make an object A move toward an object B, first turn object A to face object B using the turn to face method. Then use the move method and choose forward for the direction. You can make those actions simultaneous (Do together) or sequential (Do in order).

Center of an object
To see where the center of an object lies, select the object on the object tree. If the object shows on the display window (initial scene) it is in a yellow bounding box and directional axes for the object go through it. The directional axes meet at the center of the object. For example, the mummy shown at right has its center just behind its feet.
We now need to make Penguin jump off its skateboard (step 4). This is a case where the storyboard is not yet precise enough and needs refinement before being implemented. To jump off its skateboard, penguin will need to move left or right (or forward but that might be tricky) and downward to reach the ground. Those actions will be executed at the same time: they are concurrent.

The refinement of the storyboard will be:

4. Do together
   a. Penguin moves left 1 meter
   b. Penguin moves down to the ground

How far down should penguin move? As before we could experiment until we are happy with the amount we choose, but we could have penguin floating on the snow without noticing it. But of course there is a precise answer: the amount penguin should move down is its distance from the ground. We can find this information under the function tab. Functions answer questions about objects in the World. Some functions for the object penguin are shown in Figure 6. The question to which we need an answer is: How far is penguin above the ground? So we use the function penguin distance above ground. When we use this function we are asked to provide additional information through a drop-down list: distance above what? We choose ground.

Let’s implement this step. First we need to add a Do together block within the Do in order block. We get to the code shown on Code 3. Since we want to call penguin’s method, we must select penguin in the object tree. Then inside the Do together block we add two calls to the method move. For the first one we choose the direction left and an amount of 1 meter. For the second one we choose the direction down and any amount; we will change that amount shortly. We get the code shown on Code 4.

Now we have to change the amount the penguin moves down to penguin distance above ground. Penguin should still be selected in the object tree, click on the functions tab. Drag
distance above until it hovers over 0.5 meters (or any amount you had put there) and the box surrounding it becomes green. When the box is green, let go of the mouse button. The menu that allows you to select distance above what opens; select ground. The code is shown in Code 5.

Make sure you run the program to test it!
We need to add one more step here that is easy to overlook. Once penguin is off his skateboard the vehicle of penguin should not be the skateboard anymore. Otherwise when the skateboard moves penguin will move along with it which would be rather strange. A property can be changed in the code. To do that we select penguin in the object tree since we change one of penguin’s property and select the properties tab in the object details pane. We drag [vehicle = skateboard] right below the last of the existing lines of code in the Do in order block but outside the Do together block and release the mouse when a green line appears. In the list that appears, we select The entire world. The line of code that appears is penguin set vehicle to world. This is not a visible action and it should not take any time to be completed as it might create a pause in the animation, which might look unnatural. Therefore we set the duration to 0 second. After this line of code is executed the vehicle property of penguin will be the entire world until it is changed again.

You are now ready to translate into code steps 5 to 8 of the textual storyboard on your own.
5. Penguin greets snowWoman.
7. Penguin asks snowWoman if she wants to try to skate.
8. SnowWoman agrees to skate.
Since all these actions are executed one after the other, the corresponding method calls will be added inside the Do in order block below the line penguin set vehicle to world duration = 0 seconds shown on Code 6.
Review
- We have used two constructs called control structures to control the order of execution of the instructions written in the animations:
  - Do in order: the instructions are executed sequentially, i.e., one after the other in the order they are written.
  - Do together: the instructions are executed concurrently, i.e., at the same time.
- By default each instruction takes one second to execute. This duration can be modified.
- The items and characters that are placed in an Alice world, as well as the camera and the ground, are objects that have properties, methods and functions.
  - Methods give behaviors to objects (e.g., move, turn, say).
  - Functions answer questions about the object.
- If the vehicle property of object A is set to object B then when object B moves, object A moves along with it. The vehicle property can be modified in code or while setting up the initial scene.

Exercises
6. Open in Alice the world named AliceExercise6 and implement in my first method the following textual story board.

   Story board for exercise 6
   a. Initial scene: a shark is in the water in front of an island. Not far from the shark, on the side of the island, just at the surface of the water is a green fish.
   b. The shark turns to the fish and moves close to the fish. (Hint: use shark distance behind fish for the amount the shark should move.)
   c. The shark says: “I’ll eat you!”
   d. The fish says: “You won’t catch me!”
   e. The fish moves behind the island.

Section 4: Writing a method
Before continuing with our scenario, let us refine our animation. Since the ground is so flat, instead of gliding into view, the penguin will push himself along with his left foot (his right foot was set on the skateboard). We could add the necessary code to my first method, but it will soon become quite long and difficult to follow. So instead we will write another method.

A method is a piece of code that is given a name and that can be reused over and over again by calling that name in other pieces of code. You have used methods already that have been written by other programmers: the methods called move, say, turn to face, etc. Those methods are class-level methods because they act on just one kind of object or class. A class is, loosely speaking, the type of an object. For example snowWoman is of class SnowWoman and penguin is of class Penguin. We have also written my first method which is a world-level method because it involves at least two types of objects (or classes). Any method that involves two different classes must be written as world-level method because they can only be used in the world in which they have been created.

Let us go back to the penguin skating. We want to refine step 3 in such a way that that we see the penguin pushing his skateboard with his left foot.
The skateboard will still move forward but at the same time penguin’s left foot will turn forward and then backward several times to simulate the impulsion with the left foot. Hence the refined storyboard
3. Repeat 3 times
   a. Do together
      i. Skateboard moves forward 2 meters
      ii. Do in order
      penguin left foot turns \( \frac{1}{4} \) rev. forward
      penguin left foot turns \( \frac{1}{4} \) rev. backward

We will write step a as a world-level method. We first assign a name to the method. We always try to choose a name that appropriately represents the action coded in the method. Although Alice allows blank spaces in name of methods, I will follow the rules of most high-level programming languages that permit only letters, digits and the underscore as characters in the name of a method. The underscore is usually used to replace blank spaces. For example if you look at the methods of penguin, you will see such methods as wing_flap or turn_head_right. So let us create the method skate_left_foot. Since we want to create a world-level method, we need to make sure that world is selected in the object tree and the methods tab is selected. Currently only one method (my first method) is listed under the method tab. Underneath is the button create new method. We click on this button. In the dialog box that appears (see Figure 7), we type skate_left_foot and click OK.

A new tab titled world.skate_left_foot is added to the editing window. It is open so that we can add code for this method. We now proceed in the same way as when we wrote code for my first method. We need to implement

   a. Do together
      i. Skateboard moves forward 2 meters
      ii. Do in order
      penguin left foot turns \( \frac{1}{4} \) rev. forward
      penguin left foot turns \( \frac{1}{4} \) rev. backward

There is one new feature here: the use of subparts. It is not penguin that turns but only his left foot. As noted on page 6, we can access the subparts of an object on the object tree by clicking on the + next to the object name. If a subpart has subparts itself then we can click in the + next to it again to access those subparts, and continue until we get to the part that we want to reach. Here if we click on the + next to penguin, we see that penguin has a head, right leg, left leg, right wing and left wing. No left foot, but the left leg has a + next to it indicating the presence of subparts. So we click on it and see the subpart foot appear (see Figure 8)

**Warning:** sometimes a 3D-model is missing a part that you would expect. Then you have to adapt the code you plan to write. For example, if the penguin did not have a separate foot, we might have tried to move the whole leg.

To write the code for penguin left foot turns \( \frac{1}{4} \) rev. forward we just need to make sure that penguins’foot is selected (as shown in Figure 8) and use the turn method from this object.

The final code for the method skate_left_foot is shown in Code 7.

When we run the program now, we notice that nothing has changed in the animation. Writing a method is not enough to get it to execute. Writing a method is like writing a recipe. We tell the computer what it needs
to do whenever it is called. But, in the same way that a recipe will not feed us as long as it stays just words on a page and nobody tells the cook to “execute” the recipe, the code written in a method will not do anything as long as it is not called.

When we hit the play button in Alice, my first method is called, and that is how the code we wrote in my first method is executed. So let us go back to our refined storyboard.

1. Initial Scene: SnowWoman is alone in the middle of snowfield.
2. SnowWoman greets penguin.
3. Repeat 3 times
   a. Do together
      i. Skateboard moves forward 2 meters
   ii. Do in order
      penguin left foot turns ¼ rev. forward
      penguin left foot turns ¼ rev. backward
4. Penguin jumps off skateboard

The current code of my first method is shown in Code 8. Because the code is still simple enough it is not very difficult to find which line(s) of code to substitute with new code. But it would be even easier if we had followed a good programming practice: including comments. Comments are annotations that the programmer adds to the code that the computer simply ignores. Comments are used to make the code easier to read and correct later on. Good comments make the code more understandable to people who did not write the code and are useful when code has to be shared. Comments are an essential part of professional programs. In my first method, comments should introduce the steps of the storyboard. In other methods, an introductory comment indicates the purpose of the method and other comments are added to detail intricate steps. Comments are added in Alice by dragging the tile (found below the editing pane) at the appropriate place within the code. After dropping it, we click on the next to no comment and select other. We type the comment we want to add.
Code 9 shows my first method that has been appropriately commented.

---

Code 9

We now see clearly that the line of code after the comment “// Penguin slides into view” must be deleted. Instead the actions coded into the method skate_left_foot must be repeated three times, i.e., the method skate_left_foot must be called three times. To call that method, since it is a world-level method, we select world in the object tree and we select the methods tab. After deleting^2 the line of code skateboard move forward 5 meters, we can drag three times the skate_left_foot to replace this line. Partial code is shown at right.  (Code 10)

Well, it almost works except that penguin’s foot and skateboard do not seem to be synchronized. Let’s go back to skate_left_foot. In the Do together block, the skateboard moves forward for one second (default duration) and at the same time penguins left foot turns forward for one second and then turns backward for one second for a total duration of 2 seconds. So while penguin’s foot moves backward the skateboard does not move forward. The timing needs adjusting so that the total lengths of the actions that are done together are equal. This is illustrated below.

<table>
<thead>
<tr>
<th>Skateboard move forward</th>
<th>Penguin’s left foot turns forward</th>
<th>Penguin’s left foot turns backward</th>
</tr>
</thead>
</table>

^2 To delete any block appearing in the code (a method call, a Do in order block, a comment, etc.), you can either right-click on the block (any of the appropriately colored part) and click on delete or left-click on the block and drag it to the trash. When the trash can and the block are surrounded by green boxes release the mouse.
So that penguin does not seem to move forward too slowly, we do not change the duration of the forward motion (1 second), hence each turn of the penguin’s foot has to last only 0.5 second. Code 11 shows the corrected code for `skate_left_foot`.

![Code 11](image)

The code shown in Code 10 is repetitive, which we try to avoid. Most programming languages include special programming constructs, called **control structures** that allow repetition. The simplest repetition control structure in Alice is called Loop. (Repetitions in programming are usually called loops). We just have to tell Alice how many times we want to “loop” through the actions to repeat. The tile for the loop block is available below the editing pane. We drag it below the comment // Penguin slides into view, select other for the number of times and type 3. We drag `world.skate_left_foot` within the loop block (replacing Do Nothing). We say that this method call is in the **body** of the loop. Delete the other calls to `world.skate_left_foot`. The (partial) code should now look as follows

![Code 12](image)

The animation runs similarly as with the previous code.

**Extra-tips: Styles and copying and pasting**

**Styles**

As written, the motion of the penguin is not very smooth. If we click the ▼ appearing after more for many built-in methods, one of the options is **style**. The possible styles are gently, begin gently, end gently, and abruptly. The style “gently” means that the action coded by the method starts slowly, has a steady pace in the middle then decrease slowly. Begin gently and end slowly mean that only the beginning or the end have the slow pace. The style “abruptly” means that the speed of the action is steady throughout the execution. By default, the style “gently” is applied. When the same action is repeated, the style “gently” results, counter intuitively, in an action that is not smooth. In this case it is better to choose the style “abruptly”. So if we want penguin to glide more smoothly to snowWoman, we can edit `skate_left_foot`. We click on the ▼ after more in the method call `skateboard move forward 2 meters` and choose the style abruptly.
Copying and pasting

Quick copying and pasting can save time. If we need to duplicate almost the same piece of code (e.g., when we made the penguin’s foot turn forward then backward), we can right-click on the block we want to duplicate and select make copy. A copy of the block appears just beneath the block we are copying. We click on the appropriate \( \nabla \) to make the needed changes.

Another way to copy that is convenient when we want to copy and paste between different methods is to use the clipboard. The clipboard is located on the upper-right corner of the Alice interface. When it is empty it appears as 💻, while, it turns white 🟢 when it contains some code. To place a block of code in the clipboard, we drag it to the clipboard. When the block that we drag and the clipboard become surrounded by green boxes, we release the mouse. (Note that this replaces any old clipboard content.) To paste the content of the clipboard onto the editing pane, we drag the clipboard to the place where we want to add the code. When a green line appears at the place where we want to add the code, we release the mouse.

Exercises

For the exercises below open the world named WizardHockeyGame. The initial scene (see Figure 9) shows a wizard standing on a frozen lake. Also in the scene are a hockey puck, a hockey stick and a hockey goal. There are also two dummy objects that have been renamed originalPuckPosition and originalStickPosition. Dummy object are not seen but are used to keep track of locations in the world. More information about the use of dummy objects is given in the Extra tips section following this exercise set.

7. Partial code has already been written for the world. We have created two world-level methods and added a few line of code in my first method. Answer the following questions about the existing code.
   a. What are the names of the two world methods that have been added?
   b. Using the name of the method and reading the code in the methods (click on edit next to the method name so that the code is displayed in the editing pane), guess what each method does and add an appropriate introductory comment at the beginning of each method.

8. In this exercise you will add code to my first method. Currently my first method ends with the comment “// Hockey stick shoots goal.” Add the appropriate method call so that this action is implemented. Then test your animation.

9. After the code you added for exercise 8, implement the following storyboard (do not forget to include comments):
   a. The wizard lifts and then lowers his left arm in front of him.
   b. The puck goes back to its original position.

10. We will now have the wizard lift his right arm on the side and the hockey stick will shoot three times.
    a. Create a new world-level method called shoot_three_times. This method should make the hockey stick shoot the puck three times into the goal, i.e.:
       repeat 3 times
       shoot goal
       puck returns to original position
    (Hint: use a loop and use the method that were already written.)
b. Add code to my first method (either at the beginning or at the end of code of exercise 9) that make the wizard lift his right arm on the side and then he hockey stick shoots three times the puck into the goal (hint: call the method you wrote in question a).

Extra-tips: dummy objects
Dummy objects can be added to the world in the initial scene to mark special locations. To add a dummy object when setting up the initial scene, click on the button \[ \text{more controls >>} \] that appears below the positioning icons in the initial set-up window (see Figure 11). Controls shown in Figure 10 appear. To place a dummy object at an existing object in the initial scene, we select the object in the object tree and click on \[ \text{drop dummy at selected object} \]. A dummy object is added to the dummy object folder in the object tree. The dummy objects are called dummy, dummy2, dummy3, etc. which is not very helpful to remember what they represent. Therefore as soon as we add a dummy object, we open the dummy object folder and rename it with a meaningful name: we right-click on the dummy tile, select rename, and type the new name.
We can also place a dummy object at the camera. This is useful to remember the original position of the camera if we want to change it and then get back to it; or if we want to choose in advance different shooting angles. We move the camera in through the initial scene at those different locations and as we go along drop camera dummies. We also take care of giving meaningful name to the camera dummies, such as originalCameraPosition.
A demo about creating object dummies and camera dummies is including in your CD in the Alice folder.

Section 5: Loops, functions and methods revisited
Our storyboard has been implemented up to step 8, let us continue with step 9.
Step 9 starts with the word repeat. This indicates the use of a repetition control structure and since it says clearly repeat twice (i.e. two times) the simple Alice loop seems perfectly appropriate. We therefore need only focus on the actions that need repeating:

a. Snowwoman jumps on skateboard and then falls.
b. Penguin laughs.
c. Snowwoman stands up red in the face.
d. Snowwoman color returns to normal.

Step a is made up of two different actions. In order to keep my first method easier to read and understand, to use good programming practice and to practice writing methods we will write methods for those actions. The first method will be snowWoman_jumps_on_skateboard, or snowWoman_jumps_on_board to shorten the name. It involves two objects so it is a world-level method. We create the method making sure that world is selected in the object tree. The tab \[ \text{world.snowWoman_jumps_on_board} \] should appear in the editing pane. Before we forget we can immediately write the introductory comment for the method:
// The snowwoman jumps on the skateboard. But now we need to think about how to make this happen. To jump on the skateboard snowWoman must first face the skateboard, then she must move forward the distance that separate her from the skateboard and she must move up the height of the skateboard. Those last two motions should be concurrent. So the storyboard for the method is

snowWoman_jumps_on_board
  1. Snowwoman turns to face skateboard
  2. Do together
     a. snowWoman moves forward distance between snowWoman and skateboard
     b. snowWoman moves up height of skateboard

In both items a. and b. we use a function. In item a. we need a function of the snowWoman object to get her distance from the skateboard; in item b. we need a function of the skateboard object to get the height of the skateboard. In each case the function provides information about objects in the world. In these cases, the functions provide a number, but some functions can provide other kind of values such as true/false values, called boolean values. Translating the method storyboard into code is straightforward. You must only remember to first use dummy values for the amounts snowWoman must move forward and up, and then replace those dummy values with the functions. The initial code will look as in Code 13.

To insert the distance between snowWoman and the skateboard, we select snowWoman in the object tree and select the functions tab. We drag the function snowwoman distance to over 1 meter (next to forward), when both 1 meter and snowwoman distance to are surrounded by green boxes, we release the mouse and select skateboard> entire skateboard in the dropdown menu. We proceed similarly to add the skateboard function height (except that there will not be any drop down menu).

Code 14 shows the final code for the method.

We add a call to this method at the end of the existing code in my first method so that we can test it. (Remember if we do not call a method, it does not execute!)

3 See page 14 if you forgot how comments are added.
We now have to write a method for the fall of poor snowWoman. We will have snowWoman falling on her nose and the skateboard sliding in the other direction. Let us first write a method just for the fall of snowWoman. Since it involves only one object, we will write it as a class-level method. Let us review classes and objects.

The snowWoman object we see on our world is an instance of the class SnowWoman. Figure 12 shows the typical dialog box that opens when we add an object to a world. Here we add a snowWoman object. It will be an object of the class SnowWoman (note the capital S), and the prompt on the dialog box says “add an instance to the world.” If we add two objects of that class to a world as shown in Figure 13, then the object tree shows that the two objects are given different names, snowWoman and snowWoman2.

We can change the properties of one of the snowWomen without affecting the other one, as shown in Figure 14. But all the objects of a given class have the same properties, with possibly different values (e.g., different colors, different size) and they have access to the same methods.

We will now learn to add methods to a class.

Here is the storyboard for the method fall

Fall
1. Do together
   a. Turn forward ¼ revolution
   b. Move forward 0.5 meter

If snowwoman only rotates forward she would look suspended in the air because her center is in her middle and she rotates around her center, hence the need to make her move forward too.

Creating a class-level method is no more difficult than creating a world-level method; it only requires selecting the proper object in the object tree before creating the method. We want to create a method for snowWoman, so we select snowWoman in the object tree and select the methods tab in the object’s details pane. Then we click on the button create new method. We type the name of the method, fall and click ok.
We write the method as we did for world-level methods without forgetting the introductory comments. The code for the method is shown below (Code 15).

We can now write a method that makes snowWoman standup looking embarrassed. Should it be a world-level or class-level method? To answer this question, we ask ourselves whether it involves only one object or several objects. Since the method involves only snowWoman, it will be a class-level method for the class snowWoman. To make snowWoman stand-up we reverse the motions that made her fall down. We will only add a step to make sure that her bottom moves down to ground level. The storyboard for the method is

Stand_up_embarrassed
1. snowWoman head turns red – duration 0 seconds
2. Do together
   a. Turn backward ¼ revolution
   b. Move backward 0.5 meter
3. snowWoman moves down to ground – duration 0.5 seconds (we should not notice this motion too much)

As earlier, we create the method stand_up-embarrassed for snowWoman by clicking on create new method making sure that snowWoman is selected in the object tree and typing the name of the method in the dialog box. We are now ready to type the code for this new method.

The first action requires changing a property of snowWoman’s head. We proceed in the same way as when we modified the penguin’s vehicle property. We select the properties tab of the snowWoman’s head. Then drag color = ⬇️ to the editing pane. When we drop it in the editing pane, a dialog box allows you to choose the color. We click on red. The property is transformed into the method call: snowWoman.head set color to red.

Extra-Tip: Testing code
It is often useful to test pieces of code as soon as they are written before completing the whole method and without calling the method in my first method. We can do this by using the event pane on the upper right of the Alice interface (see Figure 15). The only event we have in the pane is “when the world starts.” When this event occurs, world.my first method is executed. As shown by the arrow ▼ next to world.my first method in the event pane, we see that the name of the method can be changed. If you click on the ▼, we can select another method by either clicking directly on the name of a world-level method or by clicking first on an object name and then the method name of a class-level method for that object. For example, we can click on ▼>snowWoman>stands_up_embarrassed. The event becomes
Now when we play the world, the method `stands_up_embarrassed` will be executed.

Remember that we only added the code to make the head of `snowWoman` become red. Try it. Not only her head becomes red, her hat gets red too! And it might be just a little bit too red (Figure 16). We can correct the mistake by clicking the ∇ next to more after the color red and select How much? object only (object and parts was selected by default). To change the color to a paler red, click on the ∇ next to the color red and select other. A color palette appears. You can select the appropriate shade of red. You can also select the RGB (Red-Green-Blue) tab and make up the color you wish by selecting amounts of red, green, and blue. Try it to review what we learned earlier about graphics!

The completed code is shown in Code 16 below.

We are now ready to code the action that must be repeated twice:

- a. Snowwoman jumps on skateboard and then falls.
- b. Penguin laughs.
- c. Snowwoman stands up red in the face.
- d. Snowwoman color returns to normal.

We need to add one more detail: the skateboard moves away from snowWoman when she falls. The easiest way to make sure that the skateboard moves in the opposite direction from the direction snowWoman is facing is to give the skateboard the same orientation as snowWoman with the method `orient to`. Then the directions forward, backward, left right, etc. will be the same for the skateboard and snowWoman.

Here is the piece of code added to my first method.

Figure 16
Comments:

- All the code added to the body of the loop (i.e. the code within the green loop tile) is executed in its entirety twice.
- A `wait` command was added between `snowWoman.stands_up_embarrassed` and `snowWoman.head.set color to ___` so that the action is paused for the indicated time between the two methods. The `wait` command is found with the keywords below the editing pane.

Rules to remember:

1. If an action of the storyboard involves only one character (or object) in the World, create a class-level method for this action.
2. If an action involves at least two objects in the world (including camera, ground, etc.) it must be a world-level method.
3. A method will not execute (“play”) if it is not called.

Saving a new class

Once we have added methods for an object, we can save the object as a new class that can be reused in other worlds. We need to change the name of the object, for example we change `snowWoman` to `fallingSnowWoman` and save the class. To change the name of the object `snowWoman`, right-click on `snowWoman` in the object tree and select rename. Modify the name to `fallingSnowWoman`. Do not include any blank space in the name. Then right-click on this “new” object `fallingSnowWoman` and select save object. Browse the folder where you saved your Alice files and click OK. Notice that the name of the file is already given: `FallingSnowWoman.a2c`. `FallingSnowWoman` is the name of the object but now the first letter is capitalized. The file extension is `.a2c` while files that contain Alice worlds have the extension `.a2w`. A demo on how to save a class and how to import the object in a new world is included in your CD.

Exercises

11. Add code to the end of `my first method` to implement step 10 in such a way that the two actions (turns towards penguin and say bye) are concurrent. Before implementing the code in Alice answer questions a and b below.
   10. Snowwoman turns towards penguin and tells him bye.
       a. Do you use an `In order` block or `Do together` block?
       b. There is no method called turn towards. How is it called?
12. Create a world with two different kinds of animal. For each animal create a class-level method to make it move. The kind of move will depend on the animal. For example, a rabbit would hop but a bird would
fly. The richer the details in the movement the better. Before choosing the animals and the “moves” read exercise # 13.

13. Continuation of number 12. Using the methods you wrote in exercise #12, write some code in my first method to implement a brief story. For example: “the rabbit is chasing the turtle, but the mighty turtle escapes.”

14. The 3 most often used categories of functions are: proximity, size, and spatial relation. For each of these three categories give the name of three functions and indicate for each whether it returns a number or a boolean value (true or false). (Important: to answer the question make sure that an object is selected, not the world, when you look at the functions tab).

Section 6: Parameters
We will now conclude the implementation of the storyboard. The last steps to implement are

11. Snowwoman skates away out of view.
12. Penguin runs after snowwoman and yells for her to wait(at the same time).

We will write a method snowWoman_skate_forward to implement step 11. This method involves both snowWoman and the skateboard so it should be a world-level method. We need of course refining step 11 before implementation. SnowWoman need to jump on the skateboard and then skate forward a certain distance. How to make snowWoman jump on the skateboard? Well, we have already a method for that (snowWoman_jumps_on_board), no need to rewrite special code for it. This is one of the important reasons for writing methods: it allows us to reuse code by just calling the methods. It saves time and it makes code much more understandable.

How do we make snowWoman skate forward? We have to remember how penguin skated: the vehicle property of penguin was set to the skateboard (in the initial set up) and then the skateboard moved forward. Before we do that we have to make sure that the skateboard has the same orientation as snowWoman.

Here the storyboard for the method.

snowWoman_skate_forward
1. snowWoman jumps on skateboard
2. Set orientation of skateboard to orientation of snowWoman (duration 0 second)
3. Set vehicle property of snowWoman to skateboard (duration 0 second)
4. Skateboard move forward a certain distance.

Here we have remained vague about the distance to move forward because we do not want to set it yet. We will use a parameter. Many built-in methods we have used so far had parameters. For example move has parameters direction and amount. When we drag the method move to the editing pane and drop it the lists shown in Figure 17 appear. They allow us to choose values for the parameter direction and the parameter amount. The values we choose are called arguments. Parameters are used in methods and functions so that the user of the methods or functions can pass some information, such as in what direction to move and by how much. We will write the method

snowWoman_skate_forward so that the user of the method can choose how far to skate forward.
Creating a method with a parameter

1. The method is a world-level method so select world in the object tree then select the methods tab.
2. Click on create new method. Type the name of the method, snowWoman_skate_forward, in the dialog box and click OK.
3. On the right-hand side of the editing pane, click on the button create new parameter. In the dialog box type the name for the parameter. We will call it how_far. The parameter should not include any blank space or special characters. Make sure that the type Number is selected. Then click OK.
4. The icon \[ \mathbf{[l_2]} \text{ how}_\text{} \_\text{far} \] appears next to name of the method.
5. We implement the storyboard for the method up to step 3. We get the code shown in Code 18.

```
world.snowWoman_skate_forward how_far

1: SnowWoman skates forward a given distance.

Do in order

// world.snowWoman_jumps_on_board

skateboard ⏸️ orient to fallingSnowWoman ⏸️ duration - 0 seconds ⏸️ more...

// world.snowWoman_falling

fallingSnowWoman ⏸️ set vehicle to skateboard ⏸️ duration - 2 seconds ⏸️ more...
```

Code 18

6. To implement step 4, we select skateboard on the object tree and drag the method move to the editing pane below the last line of code shown in Code 18. In the drop down list, we choose forward for the direction, then for amount we see an option called expression. We point our mouse to it and click on how_far, the parameter. The drop down list and the completed code are shown below.

```
world.snowWoman_skate_forward how_far

1: SnowWoman skates forward a given distance.

Do in order

// world.snowWoman_jumps_on_board

skateboard ⏸️ orient to fallingSnowWoman ⏸️ duration - 0 seconds ⏸️ more...

// world.snowWoman_falling

fallingSnowWoman ⏸️ set vehicle to skateboard ⏸️ duration - 2 seconds ⏸️ more...
```

Let us go back to what we could observe while we created this method with a parameter. First the parameter must have a name. It is used as place-holder for some information and we need to be able to refer to this information by a name. Second the parameter has a type. This is what is usually called data type (we discussed data types earlier in the course). If the information needed is a distance, duration, or a height the data type
would be a number. If the information takes just the values true or false then the type is boolean. If the information is words then the data type is string. The type of the parameter will dictate where it can be used when we write the code of the method. For example the parameter how_far, which is a number, can be used at any place where a number is expected. So it can be the amount skateboard move forward, but it could also be used in skateboard turn how_far revolutions, since the number of revolutions is ... a number. Alice will not allow using the parameter where the data type does not match. Note also the parameter is available only in the method in which it is defined.

We have now to use the method snowWoman_skate_forward in my first method to pursue our story. After adding the appropriate comment to introduce step 11 at the end of the existing code, we only have to make a call to snowWoman_skate_forward to implement step 11. When we release the mouse, because the method has a parameter, a drop down list appears to request input for the value of the argument. Since the type of the parameter is number the proposed arguments are numeric. We choose a value randomly and test the animation. If snowWoman does not disappear from the screen at the end of the animation, change the value of the argument in the call to snowWoman_skate_forward.

We are now ready to conclude the animation with step 12:
12. Penguin runs after snowwoman and yells for her to wait (at the same time).

We will just add code to my first method. Penguin has already some methods available in addition to the basic ones. Unfortunately the programmer who wrote them did not follow good programming practices: the code is not commented, some variable names are not meaningful (e.g., x) and the method names themselves are quite confusing (what is the difference between walk and walking?). By experimenting with a penguin object we can see that the walk method makes a penguin walk in place. The parameter move_time indicate the number of steps and wing flaps: there are move_time wing flaps and move_time “pairs” of steps (one with the right leg and one the left leg) each time walk is called. The walking method combines the action of the walk method and a move forward to have the penguin actually walk forward. We will use the method walking to make the penguin pursue snowWoman. You'll modify it in an exercise so that penguin speeds up to a run. The code presents no new difficulty and the complete code of my first method is shown on page 26.
// SnowWoman greets penguin

fallingSnowWoman say Hello! duration = 2 seconds more...

// Penguin slides into view

// Penguin jumps off skate board

Do together

given world,skate left foot

distance above ground = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

distance in front = more...

penguin set vehicle to world more...

// Conversation between penguin and snowWoman

penguin say Hello there! more...

fallingSnowWoman turn to face penguin more...

penguin say Do you want to Skate? duration = 2 seconds more...

fallingSnowWoman say Sure! Thanks! duration = 2 seconds more...

// SnowWoman has 2 unsuccessful tries and penguin makes fun of her

Do together

skateboard orient to fallingSnowWoman duration = 0 seconds more...

Do together

fallingSnowWoman falls

skateboard move backward 1.5 meters more...

penguin say HAHAHAH more...

fallingSnowWoman stands up embarrassed

wait 2 seconds

fallingSnowWoman head set color to more...

style = end gently more...

// SnowWoman says bye to penguin

fallingSnowWoman turn to face penguin more...

fallingSnowWoman say BYE! IT WAS FUN more...

// SnowWoman skates away

world,snowWoman,skate_forward how_far = 6

// Penguin runs after snowWoman

penguin turn to face fallingSnowWoman more...

Do together

penguin walking x = 6

penguin say WAIT! duration = 6 seconds more...
Exercises
Use the code of the method walking for the penguin class shown in Code 20 to complete exercises 15 and 16.

Code 20

15. Suppose the that the following call is made: . Fill in the blanks with the appropriate numbers (no letters!).
   a. Penguin will move forward _____ meters.
   b. Penguin will move forward during _____ seconds.
   c. Penguin flaps his wings _____ times and takes ____ steps per meter. (For our purpose a step is a right and left step.)
   d. Penguin walks at a speed of ____ meters per second (use answer from a and b to answer this question)

16. Add a method run with one parameter for the object penguin. You should imitate the walking method but penguin should moves at a greater speed and takes fewer steps per meter.
Use the method run instead of walking in my first method of penguin and snowwoman.

17. In this exercise use the world named JoeyFenceAndTree. The initial scene, which contains joey, a fence and a birchTree is shown in Figure 21. For each method you have to write in the exercise, choose appropriately the type (i.e., class-level or world-level).
   a. Write a method hop for joey that makes him hop one meter forward.
   b. Write a method hop_to_fence, that makes the kangaroo hops to the fence. You must use the method hop from question a., a loop, and a built-in function (the built-in function is used to “count” the number of hops needed). Make sure you test your method by calling it in my first method or by using the event pane.
   c. Write a method called hop_to_object that makes the kangaroo hop to any object chosen when the method is called. You need to use a parameter for this method as place-holder for the object. (Hint: the method is similar to hop_to_fence but fence is replaced by the parameter.)
   d. Call the method in my first method to make the kangaroo hop first to the fence and from there hop to the tree.
Section 7: Interactive Programs

Until now our programs have been animations. The sequence of actions has been completely controlled by the programmer. This sequence of actions is what is called the control of flow: it controls what happens first, what happens second, what happens next, etc.

In interactive programs, actions depend on user input. For example when you play a video game, what happens on the screen depends on what you do with the joy stick (or mouse or keys depending of the type of input device you use). The flow of control is now determined by the user input. The program will react to events.

An event is generated when the user provides an input or when something else notable in the system happens. In an interactive program, events will usually produce a response. The programmer writes methods whose purpose is to respond to particular events: they are called event-handling methods.

In Alice the process to create an event and to link an event to a method is simple. You need of course to do some planning first. In an interactive program, your storyboard should list the events you want to program and the corresponding event-handling methods. Let us look at an example.

On purpose the scenario was vague about the events. We will be more precise as we develop the storyboard. The initial scene for this scenario is shown in Figure 22. As can be seen by the shaded axes shown at each object location, dummy objects have been placed at the location of every instrument and the girl for use later in code. Let’s start refining the scenario. Each time we click on one of the instrument, that instrument plays. The corresponding storyboard is shown in Figure 23.
We have four events: clicking on one of the four instruments. To each event we associate a method. We start first by writing the methods. In this case the methods are all very similar. Because each method use a sound file that need to be imported into the world, the methods must be world-level methods rather than class-level methods. Let us write the method for the bongos; the other ones are similar. We first create the method `bongos_play` as a world-level method as usual. There is only one action so no need to add a `Do in order` block. Select bongos in the object tree and drag `bongos play sound` in the editing pane. A dropdown list is displayed (Figure 24). Click on `Import sound file…` (second choice from bottom). A dialog box (Figure 25) opens up that allows browsing and finding the sound file we want to import. When we have located\(^4\) and selected the file bongos10.mp3, we click `import`.

\(^4\) The sound files for Alice world are located in a folder named Alice Sound Files within the Alice folder on your disk. D:\Alice\Alice Sound Files
Now, we need to associate the event with the event-handling method. This is a very simple process in Alice. At the top of the event pane, we click on the button `create new event`, and select “when the mouse is clicked on something.” A new event is added to the event pane as shown in Figure 26. It reads: \textit{when mouse is clicked on anything, Do nothing}. The \(\nabla\) next to anything and Nothing indicate as usual that these two arguments can be changed. We want to implement: \textit{when mouse is clicked on Bongos, play bongos}. So we click on the next to anything, and select bongos in the drop down list. Then we click on the next to Nothing and click on \textit{play_bongos} in the drop down list. We have linked the event “click on bongos” to the event-handling method “play_bongos” (Figure 27).

To test the world we click on play. Nothing happens since no code was added to \textit{my first method}. But if we click on the bongos, a bongo sound track should play for 10 seconds.

To implement the rest of the scenario, we just have to create the three event-handler methods \textit{play_guitar}, \textit{play_horn}, and \textit{play_sax}. We create a mouse click event for each of the three instruments and link each method to the appropriate method. The event pane then contains the events shown in Figure 28.

These four events seem a little repetitious and crowd the events pane. We will see later how to modify that. But first let’s add some other events. Consider the new storyboard below.
**Storyboard**

**Event: Type b**

*Event Handling Method:*

- Girl goes to bongos
- Girl’s arm move to bongos
- Do together
  - Girl’s arms play the bongos
  - Bongos plays bongos10.mp3
- Girl’s arms move back to sides
- Girl moves back to original position

**Event: Type g**

*Event Handling Method:*

- Do together
  - Girl turns to face guitar
    - Guitar turns around (in position to be played by girl)
- Guitar moves in front of girl
- Girl moves arms to hold guitar
- Guitar plays guitar10.mp3
- Do together
  - Guitar moves back to original position
  - Girl moves back to original position

**Event: Type h**

*Event Handling Method:*

- Girl turns to face horn
- Horn orients to girl
- Horn moves in front of girl
- Girl moves arms to hold horn
- Horn plays horn10.mp3
- Do together
  - Horn moves back to original position
  - Girl moves back to original position

**Event: Type s**

*Event Handling Method:*

- Girl turns to face sax
- Sax orients to girl
- Sax moves in front of girl
- Girl moves arms to hold sax
- Sax plays sax10.mp3
- Do together
  - Sax moves back to original position
  - Girl moves back to original position
There are four events, which are typing the keys “b”, “g”, “h”, or “s”. For this type of events, Alice is case-insensitive, which means that typing a lowercase letter such as b or an uppercase letter such as B yield the same result. The event-handling method associated to “typing b” will have the girl move to the bongos, play the bongo and move back to her original position, while the event-handling methods associated to the three other events, “typing g, h, or s” will result in the girl turning to the instrument designated by the letter typed (guitar for g, horn for h, sax for s), then the instrument moves to the girl’s arms, plays its music and finally both instrument and girl move back to their original positions. Three of the four event-handling methods have already been written. They are called girl_plays_bongos, girl_plays_guitar, and girl_plays_horn. We will first associate these existing methods with the corresponding events. We hit the button and select the option *When a key is typed*. The event is added to the event pane. We click on the \(\nabla\) next to *any key* and select *Letters >B* (Figure 29). Then we click on the \(\nabla\) next to *Nothing* and select girl_plays_bongos. The event is now . We add the event for typing “g” and “h” similarly.

**Exercises**

19. Modify the method play_guitar, so that the guitar moves while the sound track is played (e.g., it goes up and down several times). Modify also play_horn and play_sax. Make the movement of each instrument distinct. Check your program by playing the world and clicking on the instruments.

20. Add the events “*when the key g is typed*” and “*when the key h is typed*” and associate the events with the event-handling methods girl_plays_guitar and girl_plays_horn respectively.

21. Write the method girl_plays_sax. It should be very similar to girl_plays_horn. Add the event “*when the key s is typed*” and associate it with the event-handling method girl_plays_sax.

**Section 8: More events and Conditional statements.**

The event pane has become very crowded: there are four key typed events and four mouse click events. We will regroup all mouse click events in a single event. The story board for that event is shown in Figure 30. Notice first a new control structure: the “if” statement also called conditional statement. In the statement

\[
\text{If clickedObject is bongos, play_bongos}
\]

the word “if” is followed by a condition “clickedObject is bongos.” If this condition is true, then play_bongos is executed. If the condition is false, then play_bongos is ignored and the program would move to the next line. A condition is an expression that is either true or false. It is called a Boolean expression.
Let us write the method `play_music`. It is of course a world-level method. So we select the world in the object tree and click on create new method. We then create the parameter by clicking on create new parameter (button at top right of editing pane). The name we used in the storyboard, `clickedObject`, was helpful to understand the current storyboard but we should be able to use this method in different situation. So let us use another less specific name, for example `instrument`, since it is what the parameter will represent.

In the parameter dialog box, we type the name of the parameter, select the data type “object” (Figure 31) and click OK. The “if” keyword can be found as the other Alice keywords below the editing pane. We first drag a `Do in order` block and then an “if” tile within the `Do in order` block. When we drop the “if” tile, a drop-down list with two condition values appear (Figure 32). We can select either ones. Then we drag the parameter tile in the place of the condition (true or false). When both the condition and the parameter tile are framed in green (Figure 33), we release the mouse. Then we select `instrument == bongos` (Figure 34).

![Figure 31](image1)

![Figure 32](image2)

![Figure 33](image3)
Alice does not have a simple if statement. It only has an “if…else” statement. So there are always two alternatives: if the condition is true, the set of statements below “if” is executed, if the condition is false, then the set of statements below “else” is executed. Since in our storyboard nothing happens when the condition is false, we leave the statement “Do nothing” below else. But we drop the method play_bongos below “if instrument == bongos.” The resulting code is shown in Code 21.

Following the storyboard for the method play_music, we add three more “if … then” statements in the Do in order block to complete the method. The completed code is shown in Code 22.

We now have to associate the method to the event. Before doing that we will delete the four events “When the mouse is clicked on bongos, guitar, horn, and sax” that were in the event pane. Next, we click on create new event and select “when mouse is clicked on something.” This time we will not select any particular object to replace “anything” in “When ▼ is clicked on anything ▼, do Nothing ▼.” We click on the ▼ next to “do Nothing” and select the method play_music. Since the method has a parameter, a drop-down list appears to select an argument. We select expression > object under the mouse cursor5 (see Figure 35).

We should of course test the program: we play the world and click on each instrument to check that the correct music is played. You will notice the sound

---

5 Alice sometimes produces an error when you place immediately the parameter value (object under the mouse cursor) in this step. If this occurs, choose any object for instrument (e.g. guitar) then replace guitar by object under mouse cursor.
track of one instrument has to be played completely before the next instrument can be played. We also need to click on other objects, such as the girl or the walls, and check that nothing happens.

We will experiment with one last event while learning more about the “if” or conditional statement. In a regular window environment, the mouse can be used to drag objects. We use this capability constantly when writing code in Alice. “Dragging” or “moving” an object with the mouse is an event. When we click on, we have the option “let the mouse move <objects>.” The difference with other events we have created so far is that we do not associate an event-handling method with this event. The association is done in the “background”: it is the method that allows the object to actually be moved. What we are going to do is add some code in my first method so that the animation can react to the object being moved. We will implement the following scenario.

**Scenario**

As long as the world runs, when the girl is within one meter of an instrument, the instrument plays its sound track. Only the girl can be moved with the mouse. To return the girl to her original position, click on the girl.

The scenario includes two events: “mouse click on the girl” and “mouse moves the girl.” An event-handling method for the first event should return the girl to its original position. As mentioned earlier the latter event does not have an event-handling method. However, my first method should implement the first sentence of the scenario. This analysis leads us to the design represented in the storyboard shown below.

**Storyboard**

**My first method**

While the world runs

  If girl is within 1 meter of bongos
  play_bongos

  If girl is within 1 meter of guitar
  play_guitar

  If girl is within 1 meter of horn
  play_horn

  If girl is within 1 meter of sax
  play_sax

**Event:** When mouse moves girl
No event-handling method

**Event:** When mouse is clicked on girl
**Event-handling method:** girl_back_to_original
  Set girl’s point of view to original girl position (dummy)

We will first implement my first method. The first line of the scenario reads “while the world runs.” While is a keyword for a repetition control structure. But instead of indicating how many times statements must be repeated as when using the keyword loop, we provide a condition after the keyword while. As long as the condition is verified, i.e. evaluates to true, the statements are repeated. When the condition becomes false the while statement terminates. The condition, as the condition that appears after an if statement, is a Boolean expression.
When we write “while the world runs”, we mean that the condition should remain true as long as the animation window is open. Therefore we will write “while true.” The condition (true) will never become false. This would often be a problem in other programming contexts since it creates an infinite loop that can “freeze” your program. Here it is not a problem since we will manually close the animation window and stop my first method and therefore the loop.

We make sure that my first method tab is open and drag from the bottom of the editing pane a while tile. When we drop the tile, a drop-down list with the option true and false is displayed. We select true and get the block . We now have to add the four if statements within the while loop. The condition of the first if statement reads “girls is within 1 meter of bongos,” therefore it requires us to seek information about the girl. This usually can be done by using a function. The girl’s functions include one that says: randomGirl3 is within threshold of object, where threshold and object are two parameters. Let’s use this function. We drag it to replace true that is after if (not after while!) and in the drop-down lists we select 1 meter and bongos (Figure 36). Below the if clause we insert the method call play_bongos. The other three if statements are completed in a similar fashion and the code of my first method is shown in Code 23.

To create the event “when mouse moves girl”, we click on the button and select “let mouse move <objects>.” Then we click on the ▼ next to <objects>. We get the option: create new list. We click on that option and a dialog box open. The event allows us to move several objects in the world if we wish. So we have to put the objects that can be moved by the mouse into a list. Let call this list movableObjects and enter that name in the dialog box. The items of the list will be objects, so we select object among the data type options. The dialog box looks as in Figure 37. We need to add to the list the objects that can be moved by the mouse; for our storyboard it is only the girl. We click on the ▼ to replace <None> by randomGirl3. When the dialog box looks as the in Figure 38 we click OK.
The event displayed in the event pane is "move world.movableObjects". We notice that it does not refer to randomGirl3 directly but only to the list movableObjects. We can now play the world and move the girl close to each instrument to test the program.

The last event and associated event-handling method do not present any special challenge and are left as an exercise.

Exercises
22. Complete the implementation of the storyboard page 35
   a. Write the method girl_back_to_original (the code of this method is just one line long.)
   b. Create the event “When mouse is clicked on randomGirl3” and associate the method created in a. to it.
   c. Test your program: move the girl around then click on her. She should come back to her original position.

23. Create an event to allow the mouse to move the instruments. Hint: you should put the instruments in a list.

24. In this exercise, we will use the world InteractiveWizardHockeyGame. It is similar to the world we had in section 4, but we will now make the world interactive. The world has two methods, shoot_goal and puck_returns_to_original.
a. Add an event so that if the key S is clicked, then the stick shoots a goal.
b. Add an event so that if the key B is typed the puck returns at its original position.

25. Challenge exercise: Add an event to the interactiveWizardHockeyGame that allows the mouse to move the hockey stick. Add code to my first method so that when the hockey stick gets very close to the hockey puck, the hockey puck follows the stick. Hint: you'll need to add code to my first method.