1 The Scratch Programming Environment
-Messages, Events, and Loops

Introduction: Designing a Brick Breaker Game

The tutorials in the Game Design module will take you from the beginning to the end of building a Brick Breaker game but, as with any project, you must begin with a plan in mind. This section provides an outline of the design for the game you will build. When you create your own game in the project portion of this module, you will need to come up with your own design as well.

Here is our basic design for the Brick Breaker game:

Game Play Our game will consist of a paddle, bricks, and a ball. The paddle will bounce the ball and the ball will hit the bricks. Whenever a brick is hit, it will be destroyed. The player will play the game by controlling the movement of the paddle.

Winning A player can win the game if they break all the bricks in all the game levels.

Losing The player will lose the game if the ball hits the bottom of the screen (instead of the paddle) too many times. In the first level, the player will lose on the third time that the ball hits bottom.

Keeping Score We will award points when the player breaks a brick with the ball.

Levels This game will have two levels. In the first level, we will have one row of bricks across the top of the screen and one ball to hit them. In the second level, we will have two rows of bricks across the top.

Special Effects We will also add some background music and sound effects to make the game more fun to play.

The six tutorials in this module will divide up the project as follows:

Tutorial 1 : Introduction

- Introduction to scratch
- Creating and event Loop
- Making a paddle that moves

Tutorial 2 : Motion and Collision Detection

- Adding a bouncing ball
- Creating and breaking bricks

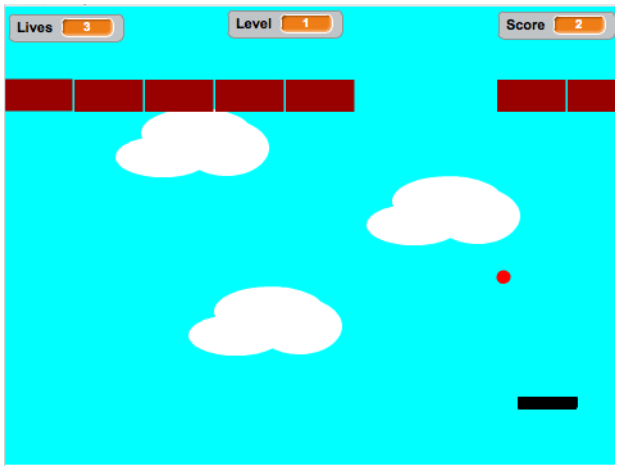
Tutorial 3 : Winning and Losing

Tutorial 4 : Extra Features

- Keeping score
- Using sound

Tutorial 5 : Adding the Second Level

Tutorial 6 : Play Testing



Have fun!

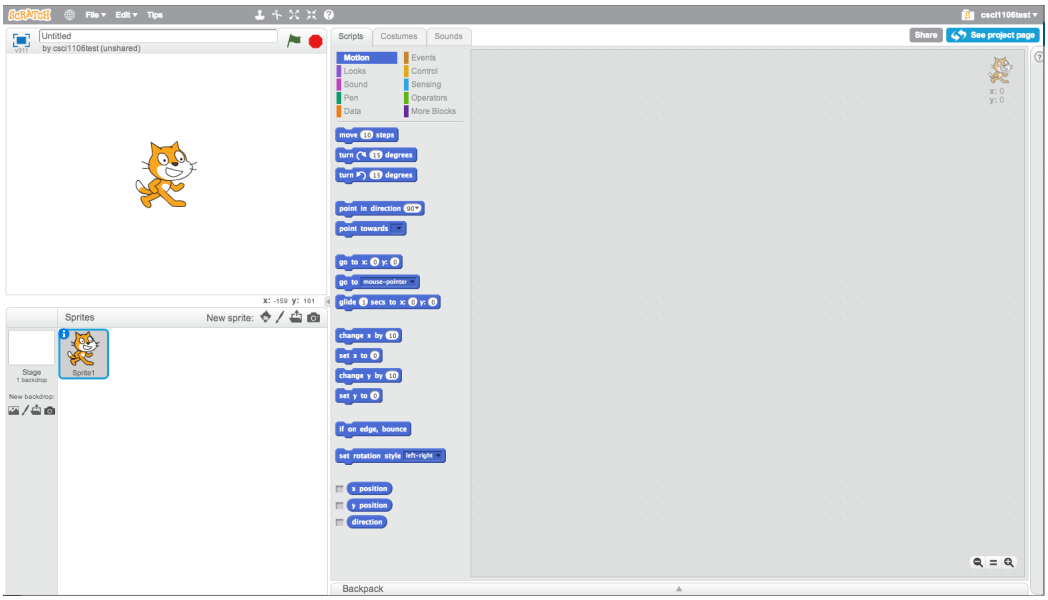
1.1 Exploring the Scratch Environment

In this section, you will create a basic Scratch program that performs some simple animations on the screen. Through this, you'll start exploring the Scratch programming environment and the various components of a Scratch program.

1. Open a web browser and point it to `http://scratch.mit.edu`.

2. Create a Scratch account for your group by clicking on the "`Join Scratch`" button in the top right corner. Your group will use this account to create and manage Scratch programs. Note: only one person should be logged into the account at a time. If you wish, you can also create personal accounts, however, it is not possible to work on the same project from multiple accounts. **Be sure each member of your group knows the account name and password.**

3. Click on "`Create`" after you have finished registering. This is one of the main menu items on the Scratch homepage. The Scratch environment will now be loaded and will look like this:



4. Click on "`Tips`" in the main menu at the top of the environment.

5. Select the "`Getting Started with Scratch`", which will walk you through the steps needed to create a simple animation.

6. Click on "`Tips`" to return to the main help panel.

7. Give your project a name and use the "`File`" menu to "`Save`" the project. The project will be saved to "the cloud", and can be accessed by you from any computer on the Internet. You can also download a local copy of the project by using the "`Download to your computer`" option in the "`File`" menu.

8. Questions for Section 1.1:

   (a) Identify the location and purpose of the following items in the Project Editor: (i) Stage area, (ii) Project title, (iii) Start / Stop buttons, (iv) Sprite list, (v) Block palette, (vi) Mouse coordinates, and (vii) Script area.

   (b) What happens when you click on the "`Costumes`" Tab?

   (c) What is a backdrop?

   (d) What is a block?

   (e) How would you get information on how to use a specific block?

1.2 Creating the Event Loop for the Brick Breaker Game

We are now ready to start setting up our Brick Breaker game. We will give our project a name and we will create its event loop. The purpose of the event loop is generate frame events every 30th of a second. In the movies, the illusion of motion is created by showing 24 pictures per second, which to the human eye looks like a smooth change in the scene. Motion in games work on the same principle, except a little faster. The event loop sends a "FRAME" message to all the sprites in the game every 30th of a second. When a sprite receives the message, it knows that a new frame has started and that it should update itself.

1. Be sure you are logged into your Scratch account and have the Scratch environment running. If you still have the previous project loaded, select "`New`" from the "`File`" menu to start a new project.

2. Set the title of your project to "Brick Breaker" and save the project.

3. Click on the "`Stage`", located to the left of the Sprite List. The Stage icon will become outlined in blue. We will create a script that is associated with the Stage, which generates frame events.

4. Drag the *wait* block from the "`Control`" block palette to the Script Area. Change the value

   in the Wait block to 0.03 seconds. This block will cause the script to wait approximately a 30th of a second before generating a frame event.

5. Drag the *broadcast and wait* block from the "`Events`" block palette to the Script Area and

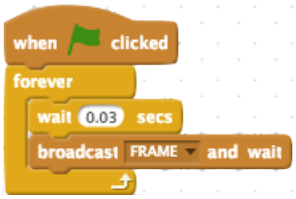
   

   attach it to the bottom of the *wait* block. Select the drop-down menu in the block and click on "`new message...`" Enter "FRAME" as the name of the new message. We now need to make it so that these two blocks are executed repetitively.

6. Drag the *loop forever* block from the "Controls" block palette to the Script Area such that it wraps around the *wait* and *broadcast and wait*. We use this block to repeat the blocks inside of it forever, or until the program is stopped. However, we only want this loop to start, when the game is started.

7. Drag the *when flag clicked* block from the "Events" block palette to the Script Area and

attach it to the top of the *loop forever* block. We use this block because we want the event loop to start when the game is started by pressing the Start (green flag) button. The final script should look like this:



8. Save your project. It is important to save regularly to avoid losing work in the event of a computer or network crash.

9. Questions for Section 1.2:

   (a) What is the difference between the *broadcast and wait* block and the *broadcast* block? Hint: Use the "Tips" panel to find out. Why do you think this is important?

   (b) When does the loop stop? Hint: It does stop.

   (c) What happens when you run your program?

1.3 Creating the Paddle

We are now ready to make use of our event loop to create a mouse-controlled paddle for our game. We will first create a sprite to depict a paddle, and then create a script that will cause it to follow the mouse pointer in the horizontal direction.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded. If not, click on "Go to My Stuff" and select the "Brick Breaker" project.

2. Control-Click on the *Sprite1* in the Sprite Area and select "`delete`". We do not need this sprite and will create our own.

3. Create a new sprite by

   (a) Select the pen icon from the New Sprite menu, located just above the Sprite Area. A new sprite *Sprite1* will be created and Paint Editor will be displayed, enabling you to draw the paddle. Note: If you wish, you can also import pictures from the library by selecting the library icon from the New Sprite Menu and you can import your own pictures by selecting the file folder icon from the New Sprite menu. For now we draw the paddle.

   (b) Draw a tiny dot in the middle of the canvas (this avoids a bug in the Paint Editor).

   (c) Click on the "`Convert to Vector`" button in the bottom right of the screen. In general, it will be easier to work in vector mode than bitmap mode. The drawing tools on the right side of the Paint Editor will be activated.

   (d) Select the rectangle tool and draw a rectangle on the Paint Editor's canvas. You can see how big this sprite is, in relation to the Stage Area, by looking at the Stage Area. Feel free to play around with the Paint Editor. Feel free to change the colour of your paddle, it's size, etc. Note, the paddle should be about 10 to 20 squares wide and 2 to 4 squares high. However, it's up to you to decide how big it should be.

   (e) Click on the "`Change Costume Center`" button in the top right corner of the editor. Drag the cross-hairs to the middle of the rectangle and click. This sets the origin of this sprite, the paddle, to its center. Hence, the position of the paddle is now associated with its center. Hint: the "`Paint Editor Map`" in the "`Tips`" panel, shows the location of all the buttons and their names.

   (f) Click on "`i`" button on the "`Sprite1`" in Sprite List. This will bring up information about the sprite, some of which we will change.

   (g) Rename the sprite from *Sprite1* to *Paddle*, by changing the name in the entry field.

   (h) Drag the *Paddle* sprite in the Stage Area so that its x coordinate is 0 and its y coordinate is around -130.

   (i) Click on the "◁" button in sprite information pane to return to the Sprite List. We are now ready to animate the sprite.

4. Click on the "`Scripts`" tab at the top of the Project Editor. We are now ready to create a script that will follow the mouse pointer each time a frame event occurs, i.e., when the FRAME message is received.

5. Drag the *set y to* block from the "`Motion`" block palette to the Script Area. Change the

   value in the block to -130. This ensures that the Paddle always remains at the same height.

6. Drag the *set x to* block from the "`Motion`" block palette to the Script Area and attach it to

the bottom of the *set y to* block.

7. Drag the *mouse x* block from the "`Sensing`" block palette to the Script Area and insert it

into the field of the *set x to* block. We are nearly done. We want this code to be executed whenever a frame event occurs, i.e., when we receive a FRAME message.

8. Drag the *when I receive* block block from the "`Events`" block palette to the Script Area and

attach it to the top of the *set x to* block. We use this block because we want the code below it to be executed whenever a frame event occurs. The final script should look like this:

9. Save your program.

10. Run your program and try moving your mouse in the Stage Area. The paddle should follow the horizontal position of the mouse pointer.

11. Questions for Section 1.3:

    (a) Suppose you set the center of the *Paddle* to be the bottom left corner of the sprite. What would be the difference in the way your program behaved?

    (b) How often will the position of the *Paddle* be updated?

    (c) Suppose you wanted to make the *Paddle* sprite move vertically rather than horizontally. What changes would you need to make to your program?

1.4 Adding a Background

Lastly, we will create a backdrop for our game. A backdrop is the scenery on the stage, i.e., a picture that is placed behind all the sprites. The default backdrop is a white rectangle. Our backdrop will consist of a blue sky and a few clouds. Each cloud will consist of three overlapping white ovals.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded. If not, click on "`Go to My Stuff`" and select the "Brick Breaker" project.

2. Click on the "`Stage`", located to the left of the Sprite List. The Stage icon will become outlined in blue.

3. Click on the "`Backdrops`" tab at the top of the Project Editor. This will bring up the Paint Editor and show the default backdrop title *backdrop1*. We will modify this backdrop. It is possible to select default backdrops from the library by clicking on the library icon in the top left corner of the Paint Editor. But, in our case, we will draw our own backdrop.

4. Click on the "`Convert to Vector`" button in the bottom right of the screen.

5. Select the rectangle tool, set the rectangle option to filled at the bottom left of the Paint Editor, and select a nice blue colour from the colour palette.

6. Draw a rectangle that covers the entire stage.

7. Select the oval tool, set the filled option as well, and set the colour palette to white. Draw three or more overlapping ovals so they look like a cloud (see below). To draw multiple clouds,

we can group these ovals together and duplicate the group.

8. Group the ovals together by selecting one of the ovals that you drew. Then hold down the shift key and select the remaining ovals. The group button will appear on the right, which you can use to group the shapes into a single cloud graphic.

9. Create duplicate clouds by selecting the cloud graphic and clicking on the duplicate button at the top of the Project Editor to duplicate the cloud. You can then select and drag the duplicate to a new location on the backdrop.

10. Save your program.

11. Run you program.

Congratulations! You have completed Tutorial 1. Save a copy of the game (`BrickBreaker1`) in case you need to return to it:

1. Click on the "`File`" menu and click on "`Save as a copy`".

2. Change the name of the program to "BrickBreaker1"

3. Save your program.

If you ever need to go back to this version of your program, you will find it in "`Go to My Stuff`" from the "`File`" menu. Your working copy should still be "BrickBreaker".

2 Motion and Collision Detection
-Conditionals, Expressions, and Cloning

Motion and Collision Detection

In this tutorial we will look at motion and collision detection. We will first look at how to create a bouncing ball, get it to bounce off walls, get it to bounce off the paddle, and lastly, add some bricks for the ball to break. We will make use of Scratch's built-in mechanism for motion. This mechanism associates a direction with each sprite, which is represented in degrees. That is: up is 0 degrees, right is 90 degrees, down is 180 degrees, and left is 270 degrees—naturally, up is also 360 degrees. One way to move a sprite is to use the *move steps* block, specifying the number of steps (1 step = 1 pixel) you wish the sprite to move. The sprite will then move in its associated direction, the specified number of steps. (This is demonstrated in the short tutorial you went through at the beginning of the previous tutorial.) To change a sprite's direction you can use the *point in direction* block, which requires you to specify a new direction in degree.[1]

As we discussed in class, collisions occur when two sprites come into contact on the stage as a result of motion. Scratch has a couple different mechanisms for detecting collisions and we shall use them in our implementation. You will be exposed to a few new programming constructs along the way, but these will be explained as we go along.

Continue to have fun!

2.1 Creating the Ball

Our first task is to create a ball. The process will be similar to the creation of the paddle. We will first paint the sprite and then animate it.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Create a new sprite:
   (a) Select the pen icon from the New Sprite menu, located just above the Sprite Area. Again, there are various balls found in the sprite library, but we will make our own. Draw a small dot in the center. Switch to vector mode and use the oval tool to draw a ball approximately 3 by 3 squares (over the dot). Note, you can draw a perfect circle by holding down the SHIFT key when drawing the oval. Feel free to give it whatever colours you wish.
   (b) Use the center tool to place the center of the sprite directly in the center of the ball. If the ball is too small, this may be difficult.
   (c) Change the name of the sprite to *Ball* and change the initial direction of the Ball to around 180 degrees by rotating the direction knob located in the information pane of the sprite.
   (d) Save your program.
3. Click on the "`Scripts`" tab at the top of the Project Editor. We are now ready to create a script that will animate the ball each time a frame event occurs, i.e., when the FRAME message is received.

[1] It may be helpful for you to review your primary school geometry.

4. Drag the *move* ☐ *steps* block from the "Motion" block palette to the Script Area. Change

the value in the block to 10 steps—this will make the ball move at a moderate speed.

5. Drag the *if on edge, bounce* block from the "Motion" block palette to the Script Area and

attach it to the bottom of the *move* ☐ *steps* block. This will cause the ball to bounce when it touches the edge of the Stage Area. Note: This is an example of a *conditional*. The second part of this statement (bounce) is performed if the the statements *condition* (on edge) is true. We will use these kinds of statements often in our programs.

6. Drag the *when I receive* block from the "Events" block palette to the Script Area and

attach it to the top of the *move* ☐ *steps* block, be sure the message to receive is "FRAME". This causes this script to be executed on every frame event.

7. Save your program.
8. Run your program. The ball should start moving and bouncing off the walls.
9. Questions for Section 2.1:
   (a) What would we change to increase the speed of the ball?
   (b) Click on the checkbox beside the *direction* block in the "Motion" block palette.

      i. What happens?
      ii. What happens when the ball bounces?
      iii. What happens if you click on the checkbox again?
      iv. Why is this a useful feature?
   (c) Why does the ball not bounce when it touches the paddle?
   (d) Suppose we wanted to see the current position of the ball. How could we do this?
   (e) Click on the "Stop" button and then on the "Start Flag".
      i. What happens?
      ii. Is this desirable? Why or why not?

2.2 Starting in the Same Place

We want to ensure that each time we start the game, the ball starts in the same place. To do this, we want to *initialize* the position and direction of the ball when the green "`Start Flag`" is pressed.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded. Also, be sure you are in the "`Scripts`" tab at the top of the Project Editor.

2. Drag the *set x to* block from the "`Motion`" block palette to an empty part of the Script Area. Change the value in the block to 0. This will ensure that the horizontal position of the ball is in the middle of Stage Area.

3. Drag the *set y to* block from the "`Motion`" block palette to the Script Area and attach it to the bottom of the *set x to* block. Change the value in the block to 0. This will ensure that the vertical position of the ball is in the middle of Stage Area.

4. Drag the *point in direction* block from the "`Motion`" block palette to the Script Area and

   attach it to the bottom of the *set y to* block. Change the value in the block to 175. This will ensure that the ball's initial direction is down towards the bottom of the stage.

5. Drag the *when flag clicked* block from the "`Events`" block palette to the Script Area and attach it to the top of the *set x to* block. We use this block because we want *initialize* the position and direction of the ball when the game is started. The final script should look like this:

6. Save and run your project.

7. To automatically organize multiple scripts in the same Script Area, control-click (or right-click) on the Script Area and select "`cleanup`" from the pop-up menu. The project Editor will then automatically organize your scripts. Be sure to save your project once more.

8. Questions for Section 2.2:

   (a) Suppose you wanted to initialize the location of the *Paddle* sprite at the start of the game. What would you need to do?

   (b) What happens if the direction of the ball was set to 90 or 270 degrees? Is this a problem?

2.3 Bouncing the Ball off the Paddle

Our next task is to make the ball bounce of the paddle as well as the walls. Unfortunately, it will be a little more complicated than bouncing the ball off the walls. We will need to use a *conditional* statement that performs the bounce of the ball if the ball comes in contact with the paddle. We will also need to compute the bounce of the ball ourselves.[2] What we will do is add to the *Ball* sprite's script that is called when the "FRAME" message is received.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Be sure you are in the "`Scripts`" tab of the *Ball* sprite.
3. Drag the *if then* block from the "`Control`" block palette to the Script Area and attach it

   to the frame event script, right after the *if on edge, bounce* block.
4. Drag the *touching* block from the "`Sensing`" block palette to the Script Area and insert

   into the field of the *if then* block. Select *Paddle* from the drop-down menu of the *touching* block. This block checks whether the current sprite (*Ball*) is touching the specified sprite (*Paddle*). The result, which is either *true* or *false*, is used by the *if then* block to determine whether or not execute the blocks it contains. That is, *if* the *Ball* is touching the *Paddle*, *then* the blocks inside the *if then* block will be executed. In our case, the blocks inside this block need to change the direction of the ball.
5. Drag the *point in direction* block from the "`Motion`" block palette to the Script Area and insert it into the *if then* block. Now we need to figure out the new direction. Recall that when a ball bounces of a flat surface the angle of incidence between the ball and the surface is the angle of the bounce, see Figure 1. Furthermore, the ball travels in the opposite (vertical)

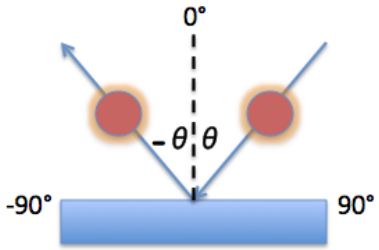


Figure 1: Angle on incidence should equal angle of reflection.

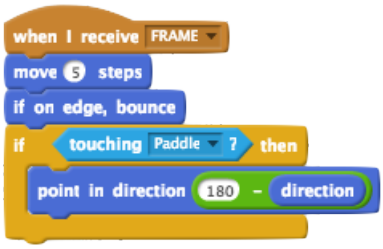
[2] You may need to recall your Grade 6 geometry.

direction after the bounce. Hence, the new direction of the ball, after it bounces is $-\theta$, where θ is the incident angle of the incoming ball, plus 180, i.e. $180 - \theta$.

6. Drag the [ ] - [ ] operator block from the "`Operators`" block palette to the Script Area and

insert it into the [*point in direction*] block. Change the left value of this block to 180.

7. Drag the [*direction*] block from the "`Motion`" block palette to the Script Area and insert it into the right value of the [ ] - [ ] block. The final script should look like this:



8. Save and run your project. Note, you may need to organize the scripts in your Script Area by doing a control-click on the Script Area and selecting "`cleanup`" from the pop-up menu.

9. Questions for Section 2.3:

   (a) What happens if the ball touches the side of the paddle? Is this a problem?

   (b) Is it a problem if the ball bounces off the wall and the paddle at the same time?

2.4 Adding Bricks

Lastly, we are going to add bricks to the game. The bricks are created at the start of the game and are destroyed when a ball hits them. This will be our introduction to *cloning*, that is creating multiple copies of the same sprite.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.

2. Create a new *Brick* sprite, a rectangle of any colour you desire, approximately 12 by 6 squares. You can use any of the available tools in the Paint Editor to decorate your brick.

3. Move the *Brick* sprite to the top left corner of the Stage are by dragging it with the mouse pointer. The left bottom corner of the brick should be at coordinates ($x = -240$, $y = 100$). You can check this by pointing your mouse at the bottom left corner of the *Brick* sprite and looking at the mouse coordinates found below the bottom right-hand corner of the Stage Area.

4. Compute the width of your brick by pointing your mouse at a right corner of the *Brick* sprite and taking the difference. For example if the right corner is located at ($x = -190$), then the width is $-190 - -240 = 50$. Write this number down, you will need it.

5. Click on the "`Scripts`" tab at the top of the Project Editor. We will now write a script to create more bricks. Since we want to create the bricks when the game is started, we will need a script that is executed when the "`Start Flag`" is pressed.

6. Drag the *when flag clicked* block from the "`Events`" block palette to the Script Area. All we want to do is for the brick to make a copy (clone) of itself. The clone will then take care of the rest.

7. Drag the *create clone* block from the "`Control`" block palette and attach it to the bottom

of the *when flag clicked* block. We now let the clone take care of the rest.

8. Drag the *when I start as clone* block from the "`Control`" block palette to an empty part

of the Script Area. A clone is an exact copy of the sprite it was cloned from, including all its properties, such as position, which we can now change.

9. Drag the *move* ☐ *steps* block from the "`Motion`" block palette to the Script Area and attach it to the bottom of the *when I start as clone* block. Set the value in this block to the width of the brick plus 5 to create slight separation between the bricks. Now, we need to create the next clone. But, we should only do so if we are not touching the right wall. If we are touching the right wall, then we do not need to create any more bricks.

10. Drag the *if then* block from the "`Control`" block palette to the Script Area and attach it to the bottom of the *move* ☐ *steps* block. We will use the *touching* block again, but this time we want to execute the blocks inside the *if then* block if we are *not* touching the edge.

11. Drag the *not* operator block from the "`Operators`" block palette to the Script Area and

insert it into the field of the *if then* block.

12. Drag the *touching* block from the "`Sensing`" block palette to the Script Area and insert into the field of the *not* block. Select *edge* from the drop-down menu of the *touching* block. Thus, if the current *Brick* clone is not touching the right edge, we can clone another *Brick*.

13. Drag the *create clone* block from the "`Control`" block palette and attach inside the *if then* block. The two scripts are shown in Figure 2.

14. Save and run the program. You should see the bricks and the ball passing through them.

Figure 2: Scripts for creating bricks.

15. Questions for Section 2.4:

    (a) What would happen if you did not first check whether the *Brick* is not touching the right edge before cloning it?

    (b) Why does the ball not bounce off the bricks?

2.5 Destroying Bricks

Lastly, we will add scripts to detect ball bouncing and destroy the bricks. As part of this process we need to do two things: (i) reverse the direction of the ball, when it hits a brick, and (ii) hide the brick when it is hit by the ball. We will hide the brick instead of destroying it because we can reuse it when we create the second level of the game.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.

2. Select the *Ball* sprite in the Sprite Area and select the "`Scripts`" tab at the top of the Project Editor. We will now add to the script that controls the movement of the *Ball* sprite. The script in question is executed when a "FRAME" message is received. A ball bouncing off a brick has the same behaviour as a ball bouncing off a paddle. Hence, the code for bouncing a ball off the brick will be nearly identical to that of the code for bouncing the *Ball* off a *Paddle*. Thus, we can use the classic copy-and-modify technique to add the necessary code.

3. Control-click on the *if then* block that checks whether the *Ball* is touching the *Paddle*. (There should only be one *if then* block in the script.) Select "`duplicate`" from the pop-up menu. This will create a duplicate of the *if them* block and its contents. Attach the duplicate after the original.

4. Select *Brick* from the drop-down menu in the *touching* block used by the duplicated *if then* block. The scripts associated with the *Ball* sprite are shown in Figure 3.

5. Save and run your program. The ball will now bounce off the bricks, but the bricks do not disappear. To make the bricks disappear we will need to modify the scripts associated with the *Brick* sprite.

6. Select the *Brick* sprite in the Sprite Area and select the "`Scripts`" tab at the top of the Project Editor. We will now add a script that is executed on each frame, i.e., when a "FRAME" message is received.

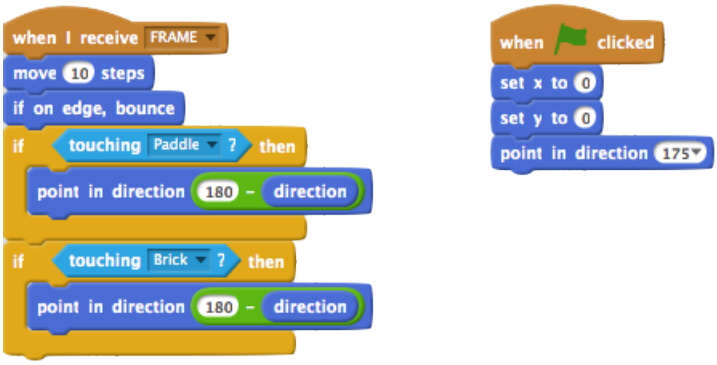


Figure 3: Scripts for bouncing the ball.

7. Drag the *when I receive* block from the "`Events`" block palette to the Script Area and place it in an empty area. Be sure the message to receive is "FRAME".

8. Drag the *if then* block from the "`Control`" block palette to the Script Area and attach it to the bottom of the *when I received* block.

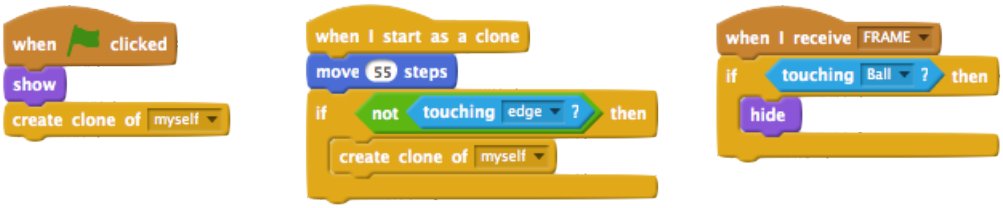
9. Drag the *touching* block from the "`Sensing`" block palette to the Script Area and insert into the field of the *if then* block. Select *Ball* from the drop-down menu of the *touching* block.

10. Drag the *hide* block from the "`Looks`" block palette to the Script Area and insert it into

the *if then* block. This will hide the *Brick* sprite if the *Ball* touches it.

11. Drag the *show* block from the "`Looks`" block palette to the Script Area and insert it between

the *when flag clicked* and *create clone* blocks. The scripts should look like this:



12. Save and run your program. The ball will now bounce off the bricks and the bricks disappear.

13. Questions for Section 2.5:

(a) Why is the *show* block necessary? Hint: Try removing it and see what happens.

(b) Give three reasons why this game, in its present form would be a little boring.

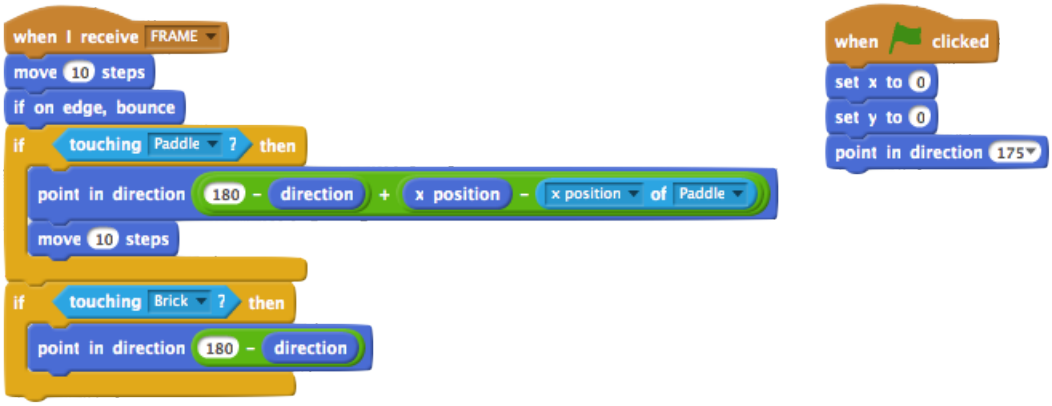
2.6 Minor Adjustments

Lastly, we will make two minor adjustments to the way the ball is deflected by the paddle.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the *Ball* sprite in the Sprite Area and select the "`Scripts`" tab at the top of the Project Editor. We will now modify the *Ball* behaviour when it touches the paddle.
3. Using the blocks from the "`Operators`", "`Motion`" and "`Sense`" block palettes change the expression in the *point in direction* block from 180 − *direction* to

$$180 - \boxed{direction} + \boxed{x\ position} - \boxed{\boxed{x\ position}\ of\ \boxed{Paddle}}$$

4. Insert a *move* *10* *steps* block Immediately after the *point in direction* block that was just modified. The resulting script should look like this:



5. Save and run your program.
6. Questions for Section 2.6:

   (a) How does the first modification (the change of direction) affect how the ball moves. Hint: Try it out by bouncing the ball off of various parts of the paddle.

   (b) What is the purpose of this change?

   (c) What is the purpose of the second change (the additional *move* block? Hint: What can happen if that *move* is not there?

Congratulations! You have completed Tutorial 2. Save a copy of the game (`BrickBreaker2`) in case you need to return to it.

3 Game State and Progress
-Variables, Conditionals, and Keyboard Inputs

Game State and Variables

In this tutorial you will use variables to keep track of the game state and to determine if the player is winning or losing the game. Game state can be something as simple as the number of remaining lives that a player has and the number of bricks she has broken, to more complex things such as the current game level, the player's power-ups, etc. In fact, our game already has state, which includes the position of the bricks, ball, and paddle, the direction of the ball, whether the bricks are hidden or not, and whether they are a clone or not. This is called *intrinsic* state because it is innate to the sprites. i.e., a sprite has to have a position, a velocity, and a direction. Program state such as a score or the number of lives remaining is not a necessary part of a sprite, and in fact is not even needed in the game, unless we wish to make the game more compelling.

To store program state we use *variables*. A variable is simply a named location in the program where information can be stored. The name of the variable is used to access it, just like a street address tells you where your home is. The name of a variable indicates its purpose. For example, if you have a variable called *RemainingLives*, the name indicates that this variable stores the number of remaining lives that a player has. You can name a variable any way you want, but meaningful names will make your programs easier to understand. Variables can store either numbers, such as 42, or strings such as "*Hello world!*". However, it's up to you, the programmer, to ascribe meaning to the values stored in the variable. This meaning will typically be reflected in the name of the variable.

In this tutorial you will use variables to keep track of the number of bricks that a player has destroyed and number of times the player has missed the ball. If a player manages to break all the bricks, she wins the game. If a player misses the ball too many times, he loses the game.

Continue to have fun!

3.1 Creating the Win and Lose Backdrops

Our first task is to create the "You win!" and "You lose." backdrops, which will be displayed when a player wins or loses the game, respectively.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Click on the "`Stage`", located to the left of the Sprite List.
3. Click on the "`Backdrops`" pane, as you did when you created your first backdrop in Tutorial 1.
4. Rename this backdrop from "backdrop1" to "Level1".
5. Click on the pen tool to the left of the Paint Editor to create two new backdrops and draw what ever kind of "You win!" (or "You lose.") backdrops that you wish. Rename the backdrops *Win* and *Lose*, respectively.
6. Save your program.

3.2 Creating the Variables

Our next task is to create the variables. We will create three variables: a *Bricks* variable, denoting the number of bricks remaining, a *Lives* variable, denoting the number of remaining lives that a player has, and a *speed* variable, which will be associated with the *Ball* sprite and will denote the speed of the ball. In the past, the speed of the ball was always 10 steps per frame, i.e., the ball moved 10 steps each frame. However, once the game is won or lost, the ball should stop moving, i.e., it should have a speed of 0. Since the speed can change, we need a variable to keep track of this. This is a good rule of thumb: *If a property or value in your program will change during the execution of your program, you will likely need a variable to keep track of it.*

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the *Ball* sprite and click on the "`Scripts`" tab.
3. Select the "`Data`" block palette and click on the "`Make a Variable`" button. A dialog will appear asking you to name the variable and whether the variable should be "`For all sprites`" or "`For this sprite only`".
4. Name the variable "Lives" and select "`For all sprites`". Then, click "`OK`". **Do not make this variable a Cloud variable.** The variables you create will appear in this block palette. Notice that the variable name and value is also displayed in the top left corner of the stage. You can control this behaviour by checking or unchecking the checkbox beside the corresponding variable block in the block palette. Try clicking on the check box a couple times to witness this behaviour. Leave the *Lives* variable checked so that the player knows how many lives she has left.
5. Create a second variable called *Bricks* that is also "For all sprites". It is up to you to decide if you wish to leave it displayed.
6. Create a third variable called *speed* (note the lower case) that is "For this sprite only", i.e., for the *Ball* sprite. This is because only the *Ball* sprite needs to keep track of how fast it is traveling. Your "`Data`" block palette should look as shown in Figure 4. By convention variables that are for all sprites are called *global* variables, while variables that are for only this sprite are called *local* variables. To determine whether or not a variable should be global or local, you need to ask which parts of your program will be using it. If different sprites will be using it, then the variable should be global. Otherwise, the variable should be made local. By convention, global variables are capitalized, while local variables are not.
7. Save your program.

We are now ready to start making use of the backdrops and variables.

3.3 Counting Bricks

We use the *Bricks* variable to track the number of bricks that remain. Each time a brick is created, the variable is incremented and each time a brick is broken (struck by the ball), the *Brick* variable is decremented. If there are no more bricks, i.e., the *Bricks* variable is 0, then the game is won.

Figure 4: The Data block palette.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.

2. Select the *Brick* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane.

3. Drag the *set* ☐ *to* ☐ block (see previous figure) from the "`Data`" block palette to the Script Area and insert it into the script that is executed when the green flag is clicked, right after the *show* block. Set the first field of the block to *Bricks* and the second field to 1. This *assigns* the value 1 to the variable *Bricks*. Or, to put it another way, *Bricks* now stores the value 1. We do this because there is exactly one brick on the stage when the game starts up. Next, every time a brick is cloned, we need to change the *Bricks* variable by 1.

4. Drag the *Change* ☐ *by* ☐ block (see previous figure) from the "`Data`" block palette to the Script Area and insert it into the script that is executed when a brick starts as a clone, right after the *when I start as clone* block. Set the first field of the block *Bricks* and the second field to 1. This changes (*increments*) the value of the *Bricks* variable by 1. Lastly, we need to decrement the variable, when a brick is broken.

5. Insert another *Change* ☐ *by* ☐ block into the script that is executed when a "FRAME" message is received, immediately after the *hide* block. Set it to change the value of the *Bricks* variable by -1. That is, when a brick is broken, the respective *Brick* sprite is hidden, and the *Bricks* variable is decremented. If there are no more bricks, then the game is won. Consequently, we need to check if the number of bricks is less than 1, and if so, stop the game and change the backdrop.

6. Insert an *if then* block after the decrement block, and use the operator block ☐ $<$ ☐ to implement the condition *Bricks* < 1. If so, a "WIN" message should be sent informing all sprites that the game is over.

7. Insert a *broadcast and wait* block into the *if then* block, and create a new message to be sent: "WIN". See Figure 5 for how the scripts should look.

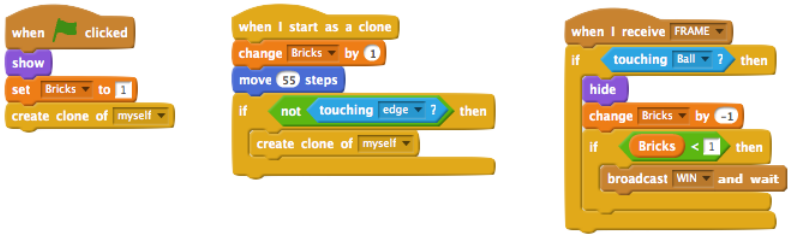
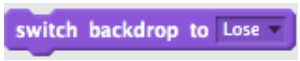


Figure 5: Scripts for creating and destroying bricks.

8. Select the "`Stage`" and be sure you are in the "`Scripts`" pane. We will now add a script that switches to the "You win!" backdrop when it receives the "WIN" message.
9. Drag a *when I receive* block in to the Script Area and set the expected message to "WIN".
10. Drag a *switch backdrop* block from the "`Looks`" palette and attach it right after the



when I receive block. Set the field of *switch backdrop* to the "Win" backdrop.

11. Drag another *switch backdrop* block from the "`Looks`" palette and attach it right after the *when flag clicked* block. Set the field of *switch backdrop* to the "Level1" backdrop. This will ensure that the right backdrop is in place when the game starts.
12. Save and run your program. If the checkbox beside the *Bricks* variable is checked, you can watch it being decremented every time the ball bounces off a brick. The "You win!" backdrop should also appear once all the bricks are gone.
13. Questions for Section 3.3:
    (a) Why is it necessary to send a "WIN" message to change the backdrop? I.e., why can't the *Brick* sprite do it directly? Is there an alternative way of accomplishing this?
    (b) Why does the ball continue to bounce even though the game is over?

3.4 Counting Lives

Next, we will use the *Lives* variable to track the number of lives that remain. Each time the ball misses the paddle, one life is lost and the *Lives* variable is decremented. Once there are no more lives, i.e., the *Lives* variable is less than 1, the game is lost.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the *Ball* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane.
3. Drag the *speed* variable block from the "`Data`" block palette to the Script Area and insert it into the field of the *move* ☐ *steps* blocks—there should be two such blocks in the scripts. This will allow us to stop, start, or change the speed of the ball as necessary.

4. Drag the [*set* ☐ *to* ☐] block from the "`Data`" block palette to the Script Area and insert it into the script that is executed when the green flag is clicked, at the end of the script. Set the first field of the block *Lives* and the second field to 3. Thus, the player will be allowed to miss the ball three times before he loses the game.

5. Drag another [*set* ☐ *to* ☐] block from the "`Data`" block palette to the Script Area and insert it after the first [*set* ☐ *to* ☐] block. Set the first field of the block *speed* and the second field to 10. Thus, the initial speed of the ball will be 10 steps per frame. We now need to add code to check when the player has missed the ball.

6. Attach a [*show*] block to the [*set* ☐ *to* ☐] block.

7. Insert an [*if then*] block at the end of the script that is executed when a "FRAME" message is received. Set the condition for the statement to be *y position* < *y position of paddle*. If the condition is true, the ball needs to be reset and the *Lives* variable decremented.

8. Inside the [*if then*] block, set the x and y position of the ball to 0 and set its direction to 175. Hint: this is done in another script so you can duplicate those blocks.

9. Add a [*change* ☐ *by* ☐] block to the [*if then*] statement. Set it to decrement the *Lives* variable. We now need to check whether the *Lives* variable is less than 1, and if so, stop the ball and send a "LOSE" message to stop the game.

10. Insert another [*if then*] block right after the decrement block and set its condition to *Lives* < 1. If this is true, then we need to (i) hide the ball, (ii) set the ball's speed to 0, and (iii) broadcast a "LOSE" message to the game.

11. Add a [*hide*] block and a [*set* ☐ *to* ☐] block inside the [*if then*] block we just added. Set the latter to change the value of the *speed* variable to 0.

12. Insert an [*broadcast and wait*] block after the [*set* ☐ *to* ☐] block, and create a new message to be sent: "LOSE". Similarly, we need to hide and stop the ball when the game is won.

13. Drag a [*when I receive*] block in to the Script Area and set the expected message to "WIN".

14. Attach a [*hide*] block and a [*set* ☐ *to* ☐] block to the [*when I received*] block, and set the latter to change the *speed* variable to 0, see Figure 6. Next, we need to ensure that the "You lose." backdrop is displayed when the "LOSE" message is sent.

15. Select the "`Stage`" and be sure you are in the "`Scripts`" pane.

16. Drag a [*when I receive*] block in to the Script Area and set the expected message to "LOSE".

17. Drag a [*switch backdrop*] block from the "`Looks`" palette and attach it immediately after the [*when I receive*] block. Set the field of [*switch backdrop*] to the "Lose" backdrop. We also need to ensure that all bricks are hidden once the game is lost.

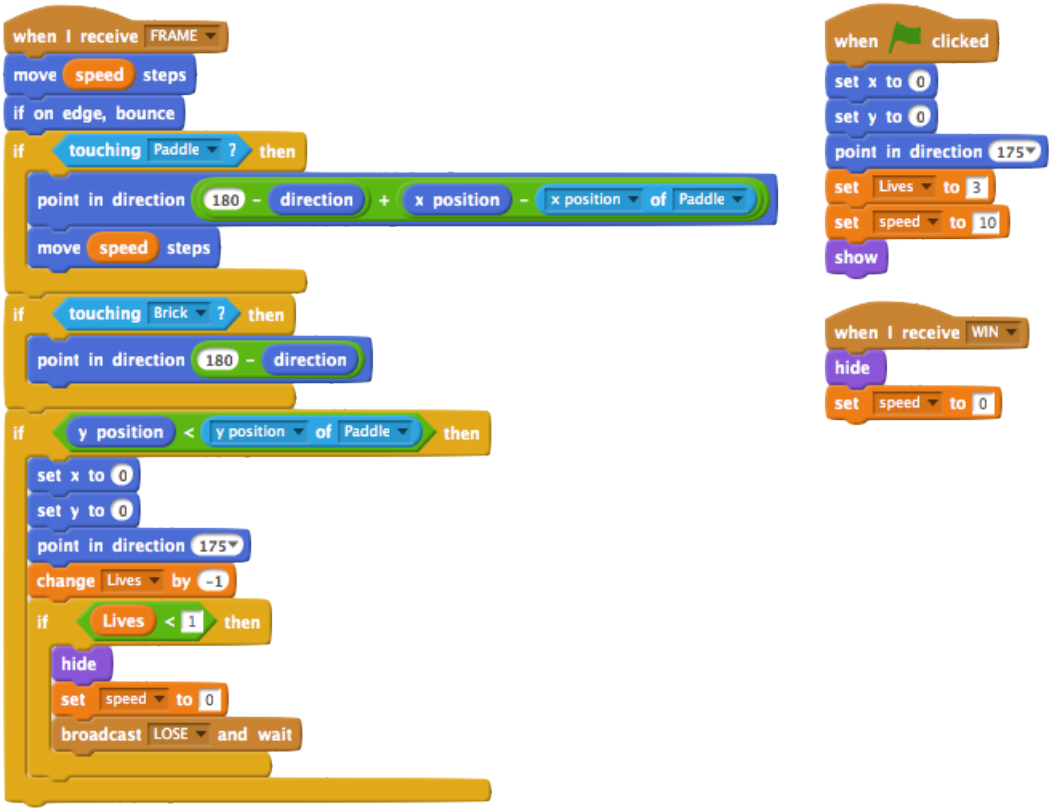


Figure 6: Scripts controlling the ball.

18. Select the *Bricks* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane.
19. Drag a *when I receive* block in to the Script Area and set the expected message to "LOSE".
20. Attach a *hide* block to the *when I receive* block. Lastly, regardless of whether the game is won or lost, the paddle must be hidden.
21. Select the *Paddle* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane.
22. Drag a *when I receive* block in to the Script Area and set the expected message to "WIN".
23. Attach a *hide* block to the *when I receive* block.
24. Drag a *when I receive* block in to the Script Area and set the expected message to "LOSE".
25. Attach a *hide* block to the *when I receive* block.
26. Drag a *when flag clicked* block in to the Script Area.
27. Attach a *show* block to the *when flag clicked* block.
28. Save and run your program. Test your program and make sure that it has the correct behaviour for both winning and losing.

29. Questions for Section 3.4:

    (a) Suppose you wanted to increase the speed of the ball when it bounced of a brick. What changes would you need to make?

    (b) Why is the condition "y position of ball < y position of paddle" a reasonable way to check whether the player has missed the ball?

    (c) When the game is lost or won, why must the speed of the ball be set to 0?

    (d) Suppose that you wanted to play the game again. How would you restart the game (in its current form)?

3.5 Play Again?

Lastly, we want to allow the player to play the game multiple times simply by hitting the space bar at the "You win!" or "You lose." screens. This will introduce us to keyboard events and will demonstrate how keyboard input can be integrated into a game.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the "`Stage`" and be sure you are in the "`Scripts`" pane.
3. Drag a *when* ☐ *key is pressed* block into the Script Area and set the expected key to "space".

   Whenever the space key is pressed this script will be executed. But, we only want the game to restart if we have lost or won the game.
4. Attach an *if then* block to the *when* ☐ *key pressed* and set the condition in the block to

   $$Bricks < 1 \textbf{ or } Lives < 1$$

   using the operator and variable blocks to construct the above expression. This means, the condition is true when either *Lives* or *Bricks* is less than 1, i.e., the game is lost or won, respectively. If this condition holds when the space bar is pressed, we restart the game.
5. Insert a *set* ☐ *to* ☐ block and a *switch backdrop* block into the *if then* block and use them to set the *Bricks* variable to 0 and switch to the "Level1" backdrop.
6. Insert a *broadcast and wait* block after the *switch backdrop* block, and create a new message to be sent: "START". When sprites receive this message they will reset themselves to play another game.
7. Select the *Paddle* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane.
8. Drag a *when I receive* block in to the Script Area and set the expected message to "START".
9. Attach a *show* block to the *when I receive* block.

10. Select the *Ball* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane.
11. Drag a [*when I receive*] block in to the Script Area and set the expected message to "START".
12. Duplicate the code attached to the [*when flag clicked*] and attach it to the [*when I receive*] block.
13. Select the *Brick* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane.
14. Drag a [*when I receive*] block in to the Script Area and set the expected message to "START".
15. Attach a [*show*] block to the [*when I receive*] block.
16. Attach a [*change* ☐ *by* ☐] block to the [*show*] block and use it to increment the *Bricks* variable by 1.
17. Save and run your program. Test your program and make sure that it has the correct behaviour for both winning, losing, and restarting.
18. Questions for Section 3.5:
    (a) When the game is restarted, why do we explicitly reset the *Bricks* variable to 0?
    (b) How does the *Bricks* variable eventually get its original value back, i.e., the number of bricks on the stage?
    (c) Suppose you wanted to control the paddle with a keyboard (left/right arrow keys) rather than the mouse. What changes would you need to make to your code?

Congratulations! You have completed Tutorial 3. Save a copy of the game (`BrickBreeaker3`) in case you need to return to it.

4 Extra Features
-Playing Sounds and Fixing Bugs

Score, Sound, and Feature Fixes

In this tutorial you will add a score keeper to your game and learn how to add a sound track and sound effects. You will also fix a feature (bug) in the game that you may have noticed earlier.

Bugs are program flaws that make your program behave in an unexpected and incorrect manner. Bugs are a natural part of software development and happen on a regular and frequent basis. The bug in question occurs when a ball bounces off a brick but the brick does not disappear. We will look at why this bug occurs and how to fix it, as this is a common kind of bug that you may encounter in the future.

4.1 Keeping Score

Our first task is to add a score feature to the game, which will track and display the player's score. The simplest kind of scoring is to add 1 every time a brick is destroyed. Feel free to make the scoring in your game more complex. As you may have guessed, we will use a *Score* variable to store the score and to display it.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the *Brick* sprite and be sure you are in the "`Scripts`" pane.
3. Create a new variable called *Score* that is global, i.e., "for all sprites". Notice that the value of the variable is displayed on the Stage Area. Drag the *Score* display to the right corner of the stage. This is how the score will be displayed to the player.
4. Insert the [*set* ☐ *to* ☐] block into the scripts that are executed when the green flag is clicked or when the "START" message is received. Set the block to initialize the *Score* variable to 0, i.e., initially a player's score is 0.
5. Insert a [*change* ☐ *by* ☐] immediately after the [*change* [*Bricks*] *by* [*-1*]] in the script that is executed when a "FRAME" message is received. Set it to increase the *Score* variable. That is, whenever the ball touches the brick, and the brick is hidden, the *Bricks* variable is decremented and the *Score* variable is increased.
6. Save and test your program.
7. Questions for Section 4.1:
   (a) Suppose different bricks were worth different amounts of points.
      i. How could each brick keep track of how many points it is worth?
      ii. How would the scoring code change in this situation?
   (b) Suppose every time that a player missed the ball, his or her score would decrease and if the score dropped below 0, the game would end. What would you need to do to implement this?

4.2 Adding Sound Effects

To add a sound effect, we simply need to play a selected sound when something happens in our game. For example, if the ball bounces off a brick or a paddle, we want to play a bouncing sound.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the *Ball* sprite and click on the "`Scripts`" tab.
3. Click on the "`Sounds`" pane in the Project Editor. This allows us to create, upload, or select sounds from a library. For our purposes, we will select a sound from the library.
4. Click on the "`Sound Library Icon`" in top left corner of the Sound Editor. A list of sounds should appear.
5. Select the "pop" sound from the list. It will appear in the Sound List in the left column of the Sound Editor.
6. Click on the "`Scripts`" pane to add blocks to play the sound.
7. Drag the *play sound* block from the "`Sound`" block palette and insert immediately before

   the *point in direction* block in the *if then* block that tests whether the ball is touching a paddle (or brick).
8. Save and test your program.

4.3 Adding a Sound Track

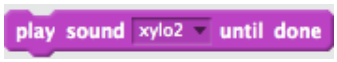
We can also use the same sound blocks to add a sound track to the game. A sound track is a piece of music that is repeatedly played while the game is in progress and ceases once the game ends. This is the approach we take to implement a sound track in our game.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the "`Stage`" and click on the "`Sounds`" pane.
3. Click on the "`Sound Library Icon`" and select the "xylo2" sound. Note, you can upload your own sound track if you wish, but for now this will do. As before, the sound will appear in the left column of the Sound Editor. We are now ready to add more code.
4. Click on the "`Scripts`" pane.
5. Drag another *when I receive* block in to the Script Area set it to receive the message "SOUNDTRACK" by creating the new message.
6. Drag a *repeat until* block from the "`Control`" block palette. Attach it to the *when I receive* block. Set the condition to

   $$Bricks < 1 \textbf{ or } Lives < 1$$

   That is, keep repeating until there are no more bricks or no more lives.

7. Insert a *play until done* block from the "Sound" block palette into the *repeat until* block.



8. Attach a *stop all sounds* block from the "Sound" block palette to the end of scripts that

   execute when a "WIN" or "LOSE" message is received. This will stop all sounds once the game is won or lost. Lastly, we need to start the soundtrack when the game begins.

9. Insert a *broadcast* block after the *switch backdrop* blocks that switch the backdrop to "Level1" (there are two scripts where this occurs). Set the *broadcast* block to send the "SOUNDTRACK" message, which will then start the soundtrack.

10. Save and test your program.

11. Questions for Section 4.3:

    (a) What would happen if the condition for the *repeat until* block never became true?

    (b) Supposed you wanted to add a voice saying "You win!" or "You lose." when the game is won or lost. Describe how this can be implemented in your game.

4.4 Fixing a Bug

As mentioned previously, there is a bug in our game. Occasionally, when a ball hits a brick, the ball bounces but the brick does not disappear. This is due to a "race condition" that occurs between the scripts that receive the "FRAME" message. Specifically, the scripts associated with the *Ball* and *Brick* sprites. The latter script checks if the ball is in contact with the brick, and if so hides the brick. The former script first moves the ball forward, checks whether the ball is in contact with the sides, paddle, or brick, and bounces the ball if any of these conditions are met. The scripts run at the same time, when a "FRAME" message is broadcasted, but some scripts may run faster or ahead of others. So, if the *Brick*'s script performed its check before the ball moves, and then the ball moves and comes in contact with a brick, the ball will bounce but the brick will not hide because that check was done before the ball came in contact with the brick. An example of an analogous situation is when you check the fridge and realize you are out of milk. You go to the store to buy milk. Meanwhile, your roommate returns and puts milk that she had purchased into the fridge. You return to find that there is actually milk in the fridge, even though the last time you checked there was not.

To fix this problem, we will make it so that the *Brick* sprite will check if it is in contact with a brick only when the *Ball* has already determined that contact has been made. That way, the check by a *Brick* sprite will only occur after the ball has been moved, thus eliminating the race condition.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the *Brick* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane. Consider the script that is executed when the "FRAME" message is received.
3. Change the message that this script waits for from "FRAME" to "CHECK" by creating a new message called "CHECK".
4. Select the *Ball* sprite from the Sprite Area and be sure you are in the "`Scripts`" pane. Consider the script that is executed when the "FRAME" message is received.
5. Insert a *broadcast and wait* block into the *if then* block that tests whether the *Ball* sprite is touching a *Brick* sprite, right before the *point in direction block*. Set the *broadcast and wait* block to broadcast the "CHECK" message. That is, if the ball is in contact with a brick, the "CHECK message is sent. Each brick receives it, and checks if it is in contact with the ball. If it is, it hides itself.
6. Save and test your program.
7. Questions for Section 4.4:
   (a) Why does the brick still need to check whether it is in contact with the ball before hiding itself? I.e., Why is a similar check performed by the ball not sufficient on its own?
   (b) Why do we use the *broadcast and wait* block rather than a *broadcast* block?
   (c) Why does the *broadcast and wait* block have to be inserted immediately before the *point in direction* block. Hint: If the origin of the *Ball* sprite was not perfectly centered. what could happen when the *Ball* is turned?

Congratulations! You have completed Tutorial 4. Save a copy of the game (`BrickBreaker4`) in case you need to return to it.

4.5 Bonus Features

Try adding additional effects or features to your game. For example:

1. Multihit bricks (Bricks that require multiple hits to be broken.)
2. Multicoloured bricks
3. Multiple balls
4. Keyboard controlled paddle
5. Additional sound effects
6. Powerups that increase the size of the paddle
7. Etc...

5 Multiple Levels
-Increasing Complexity

Adding a Second Level

In this tutorial you will add a second level to the game. We will be making a number of modifications throughout our game and adding sprites and messages. You are encouraged to improve the game and levels as much as possible. Feel free to change things up and be creative.

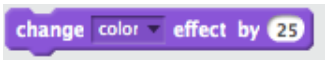
5.1 Add a Second Level Backdrop

The first thing to do is to add a second level backdrop to the stage.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Add a new backdrop to the *Stage* and call it "Level2". This is what will be displayed when the second level starts.
3. Add a script that is executed when the second level commences:
   (a) The script should begin with a *when I receive* block that expects to receive the message "LEVEL2". (You will need to create the new message.)
   (b) The script should switch backdrops to "Level2" using the *switch backdrop* block.
4. Save your program.
5. Questions for Section 5.1:
   (a) Describe what you would need to do if you wanted to change the soundtrack for the new level.

5.2 Changing the Appearance of Ball and Paddle

You may wish to change the appearance of the ball and paddle for the second level. To do this we will use the *change effect* block from the "`Looks`" block palette.



1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.
2. Select the *Paddle* sprite and add a script that is executed when the second level commences:
   (a) The script should begin with a *when I receive* block that expects to receive the message "LEVEL2".
   (b) Use the *change effect* block to change the brightness of of the paddle. Try entering different values into the field and see what the resulting paddle looks like.
3. Append a *clear graphic effects* block to the script that is executed when the "START" message is received.

4. Select the *Ball* sprite and add a script that is executed when the second level commences:

   (a) The script should begin with a *when I receive* block that expects to receive the message "LEVEL2".

   (b) Use the *change effect* block to change the colour of of the ball. Try entering different values into the field and see what the resulting ball looks like.

   (c) Use a *change* ☐ *by* ☐ block to change the speed of the ball by 5.

5. Append a *clear graphic effects* block to the script that is executed when the "START" message is received.

6. Save your program.

7. Questions for Section 5.2:

   (a) Suppose the paddle was initially black, and you wanted to change it to a light grey. What value of brightness would you use in the *change effect* block?

   (b) Suppose the ball was initially red, and you wanted to change it to a yellow. What value of colour would you use in the *change effect* block?

   (c) Why do we need to add the *clear graphic effects* block to the scripts that receive the "START" message?

5.3 Adding Bricks

In addition to increasing the speed of the ball, we will add a second row of bricks to the second level. We will do this by duplicating the *Brick* sprite and modifying both the original and duplicate.

1. Be sure you are logged into your Scratch account and the "Brick Breaker" project is loaded.

2. Click on the "`duplicate`" tool at the top of the project editor and the click on the *Brick* sprite in the Sprite List Area. A duplicate sprite, called *Brick2* will appear.

3. Change the colour of the duplicated brick sprite by editing its costume.

4. Be sure both brick sprites are visible on the stage. A quick way to do this is to first select each of the sprites and then click on the *show* block in the "`Looks`" block palette.

5. Position the *Brick2* sprite above the *Brick* sprite on the stage. That is, they should both be near the top left corner of the stage. We are now ready to modify the code for each of the brick sprites.

6. Select the *Brick* sprite from Sprite List Area.

7. Add a script that is executed when the second level commences:

   (a) The script should begin with a *when I receive* block that expects to receive the message "LEVEL2".

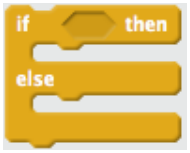
   (b) The script should first *show* the brick and then increment the *Bricks* variable by 1.

8. Add a *Level* variable that is "for all sprites." This will be used to distinguish between the first and second levels. Drag the display of the variable to the top center of the stage, between the *Lives* and *Score* variable displays.

9. Initialize the *Level* variable to 1 by inserting a [*set* ☐ *to* ☐] blocks into the scripts that are executed when the flag is clicked or when the "START" message is received.

10. Modify the script that is executed when the "CHECK" message is received to load the second level once the first level is completed. That is, if the number of bricks drops to 0, we check whether the current level is 1 or 2. In the former case, we start the second level and in the latter case, we win the game:

    (a) Insert a [*if then else*] block into the [*if then*] that tests whether *Bricks* < 1.

    (b) Set the condition of the [*if then else*] block to test whether *Level* $= 1$.

    (c) Move the existing [*broadcast* [*WIN*]] block into the *else* part of the [*if then else*] block.

    (d) In the *if then* part of the [*if then else*] block, increment the *Level* variable by 1 and broadcast the "LEVEL2" message.

11. Select the *Brick2* sprite from Sprite List Area. The modifications in this case are much simpler.

12. Modify the script that is executed when the green flag is clicked by replacing the [*show*] and variable initialization blocks with a single [*hide*] block.

13. Add a script that is executed when the "LEVEL2" message is received. The script should [*show*] the brick and increment the *Bricks* variable.

14. Remove the variable increment block from the script that is executed when the sprite is started as a clone.

15. Delete the script that is executed when the "START" message is received.

16. Select the *Ball* sprite from Sprite List Area.

17. Modify the [*if then*] block that checks whether the ball is touching the *Brick* sprite to check whether it is touching the *Brick* **or** *Brick2* sprites.

18. Save and test your program.

19. Questions for Section 5.3:

    (a) In the *Brick2* sprite:

        i. Why do we replace the [*show*] block with [*hide*] block?

        ii. Why do we remove the variable initialization and incrementation blocks from many of the scripts?

        iii. Why is the "START" script not required?

    (b) Describe what you would need to do if you wanted to require two ball bounces to break the second level bricks. That is, the ball has to strike the brick twice before the brick breaks.

    (c) Describe what you would need to do to add a third level.

Congratulations! You have completed Tutorial 5. Save a copy of the game (`BrickBreaker5`) in case you need to return to it.

5.4 Bonus

Try adding a third level with additional effects or features to your game. For example:

1. Multihit bricks (Bricks that require multiple hits to be broken.)
2. Multicoloured bricks
3. Multiple balls
4. Multiple soundtracks
5. Keyboard controlled paddle
6. Additional sound effects
7. Powerups that increase the size of the paddle
8. Etc...

6 Playtesting
-Improving your game

Making your Game Even Better

Congratulations! You've just finished the basic implementation of your brick breaker game. Now, the question is: Is it good? Can you make it better? How could you modify your game without adding features to make more people interested in playing? Maybe you should ask them? If you did so in a careful and systematic way, you would be playtesting your game. In this tutorial you will ask for the feedback of your peers to decide on what changes you should make.

First, make sure that you have worked out most or all of the bugs (program failures) in your game. Your testers may give up or be unable to provide you with helpful feedback if it is not made easy enough for them to try.

6.1 Determining What to Evaluate

We first need to consider what we can change about the game.

1. Create a list a list of things about your game that you could change to suit your potential gamers. For example,

   Speed : Paddle speed and ball speed

   Colour Scheme : Background colour, ball colour, brick colour,

   Sound : Background music, paddle bounce sound, and brick break sounds

   Feel free to add to this list.

2. Choose one aspect of your game that you hypothesize could be improved. Then select three different settings for your peers to try. You will need to prepare a version of your game for each setting (i.e. 3 different versions of your game) to make it easy for them to try your game. This is done by making copies of your game, loading each one into a different tab of the browser, and making appropriate modifications to each.

3. Questions for Section 6.1:

   (a) Which element of your game do you want to evaluate?

   (b) What are the settings you will propose?

6.2 Setting Up

Next, we need to create the documentation for the playtest.

1. Create instructions for your testers. These instructions will briefly tell them about the purpose of the study, let them know that they can stop at anytime without penalty, and identify the specifics of what they will evaluate. It will also tell them how to actually run your games, including brief instructions about how to play (i.e. how to win, moving the paddle, etc.).

2. Create the questionnaire. This is how you will request and record the tester's findings. In the form of questions, request ratings from your testers. You could ask testers to rate each setting from 1 to 5 (the Likert scale), or instead rank the settings from most preferred to least. Finally, you may wish to ask additional, easy-to-answer-questions that will help you decide on which setting is best. These ratings are subjective (opinion-based) measures. This is useful because people often choose what things they do (or games they play) for subjective reasons. However, you may also want to have objective measures (such as score for each game played) to evaluate the effectiveness of an option. Finally, leave your contact information and don't forget to thank them for their participation. Request that testers to save their responses in an electronic file on the desktop (i.e. "Save As" with a random 5-digit number as the filename. This helps maintain confidentiality or anonymity. Leave the instructions and questionnaire in a text document on the desktop for easy access.

3. Pilot your playtest. Take time to run through the instructions and procedure as a group to see that what you are asking isn't too difficult or takes too long, including the steps of recording answers in the questionnaire and saving the file, etc. You want to make it as realistic as possible. Make adjustments as necessary.

4. Questions for Section 6.2:

   (a) How many rounds should testers play to make a good assessment? Consider that you will have limited time (8 min) per tester.

6.3 Playtesting

Take approximately 1 hour as a group to be the testers for other groups' playtests. Take turns playing the games and recording findings, but everyone in the group should feel free to offer their opinions so that you can provide feedback as a group. Do your best to follow their instructions.

6.4 Evaluation and Followup

1. Evaluate the results of the playtest. Compute the average (or mean) rating for each setting, as well as any other numerical results (e.g., score) that you requested in your playtests.

2. Questions for Section 6.4:

   (a) What were the average ratings or rankings for each setting?

   (b) Which setting is best and why? Remember to cite data from your study, including comments from your testers, if any, that would help make your case. What is one reason why your choice may, in fact, not be the best one?

3. Make changes to incorporate the best setting, as determined from your playtests.

Congratulations! You've completed your Brick Breaker game!