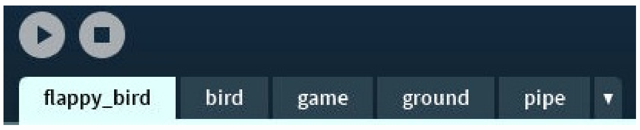
Advanced Topics Lab 1

Lab 1 - Setup

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `advanced`, and double click on `FlappyBird`. Finally, open the `FlappyBird.pde` file. This should open five different files.



Processing IDE with all of the game files open in separate tabs

We are going to create a clone of the game Flappy Bird that makes use of the topics covered in this module. We will be using the Processing (a programming language based on Java) because creating an animation is much easier in Processing than in Java. The tutorial above uses a collection of Processing objects to make the game. We are going to create a composite class, an array of objects, and the lab will use five Processing files.

▼ **Copying & Pasting**

Unfortunately you will not be able to copy code from these pages and paste it into the Processing IDE. Processing is not running in your browser. Instead, it is running on a server somewhere across the internet. We use a virtual network computer (VNC) to show the Processing IDE in your browser. Because of this, you cannot copy/paste code. Using a VNC can also affect performance of the game.

The first thing to do is set up a generic game. **In the `FlappyBird` file**, Declare the `game` variable. In the `setup` method, set the size of the game and instantiate a `Game` object. In the `draw` method, set the background to the `background` attribute of `game`. Then call the `show` and `update` methods.

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}
```

Processing should underline a few lines of code in red, indicating an error. We need to create the `Game` class with a few methods to make these error go away. **In the `Game` file**, declare the `Game` class. For now, give it the attribute `background`. In the constructor, load an image for the background. Then, make a getter for the `background` attribute. Declare the `show` and `update` methods, but do not write any code in the body. The underlined code should go away. The `show` method will be used to place images on the game window. The `update` method will be used to update the positions of the images in the game window.

```
class Game {
  private PImage background;

  public Game() {
    background = loadImage("background.png");
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {

  }

  public void update() {

  }
}
```

info

Try It

Click the triangle button to run you program. You should see a 400 x 719 window appear with a background image. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

Your code should look like this:

`FlappyBird` File

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}
```

`Game` File

```java
class Game {
  private PImage background;

  public Game() {
    background = loadImage("background.png");
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {

  }

  public void update() {

  }
}
```

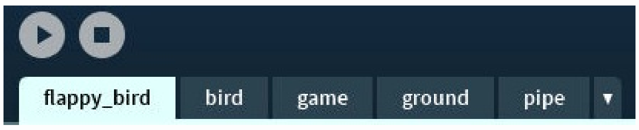
Advanced Topics Lab 2

Lab 2 - The Ground

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `advanced`, and double click on `FlappyBird`. Finally, open the `FlappyBird.pde` file. This should open five different files.



Processing IDE with all of the game files open in separate tabs

In this game, the bird only moves up and down. To give the impression that the bird is flying to the right, the ground is going to be animates. **In the `Game` file**, add the `ground` attribute. This is going to be another component class. Instantiate the `ground` object in the constructor. **Note**, Processing will underline these lines of code as the `Ground` class does not yet exist.

```
class Game {
  private PImage background;
  private Ground ground;

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
  }
```

In the `Ground` file, declare the `Ground` class with attributes for an image file and an x-position. Give the attributes a value in the constructor.

```
class Ground {
  private PImage ground;
  private int x;

  public Ground() {
    ground = loadImage("ground.png");
    x = 0;
  }
}
```

Next, we need the ground image appear in the game. Start by adding a `show` method for the `Ground` class. This should place the image to align with the bottom of the background. Use the `x` attribute to represent the horizontal position of the ground image. The y-coordinate should be 650 so the bottom of the ground image touches the bottom of the window.

```
class Ground {
  private PImage ground;
  private int x;

  public Ground() {
    ground = loadImage("ground.png");
    x = 0;
  }

  public void show() {
    image(ground, x, 650);
  }
}
```

In the `Game` file, update the `show` method to display the ground image. Do this with by calling `show` on the `ground object`.

```
  public void show() {
    ground.show();
  }
```

info

Try It

Click the triangle button to run you program. You should see the background and ground images in the window. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

Animating the Ground

To make the bird appear to fly to the right, the ground needs to move to the left. **In the `Ground` file**, add the `update` method which decreases the `x` attribute by one pixel.

```
public void update() {
  x -= 1;
}
```

In the `Game` file, go to the `update` method and add the code `ground.update()`. This will move the ground image to the left.

```
public void update() {
  ground.update();
}
```

However, you will notice that the ground image will disappear. **Go to the `Ground` file.** We need the ground image to infinitely move to the left. First, update the `show` method so that two ground images next to each other. The first image is at position `x`, and the second is at `x + 470` because the ground image is 470 pixels wide.

```
public void show() {
  image(ground, x, 650);
  image(ground, x + 470, 650);
}
```

The ground images will still disappear however. We want this animation to repeat itself. To do this, reset the value of `x` to 0 when the first image is completely out of the window (when `x` is less than -470). After `x` has decreased by 1, use an if-statement to check for this condition. If true, set x back to 0. This creates a smooth, infinite animation.

```
public void update() {
  x -= 1;
  if (x <= -470) {
    x = 0;
  }
}
```

info

Try It

Click the triangle button to run you program. You should see the background with an animation that moves ground image infinitely to the left. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

Your code should look like this:

`FlappyBird` File

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}
```

`Game` File

```
class Game {
  private PImage background;
  private Ground ground;

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {
    ground.show();
  }

  public void update() {
    ground.update();
  }
}
```

Ground File

```
class Ground {
  private PImage ground;
  private int x;

  public Ground() {
    ground = loadImage("ground.png");
    x = 0;
  }

  public void show() {
    image(ground, x, 650);
    image(ground, x + 470, 650);
  }

  public void update() {
    x -= 1;
    if (x <= -470) {
      x = 0;
    }
  }
}
```

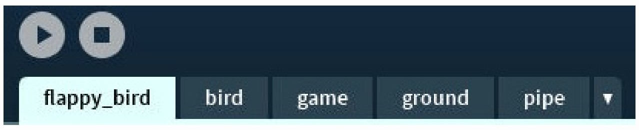
Advanced Topics Lab 3

Lab 3 - The Bird

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `advanced`, and double click on `FlappyBird`. Finally, open the `FlappyBird.pde` file. This should open five different files.



Processing IDE with all of the game files open in separate tabs

You are going to see a pattern emerge. Create a component class. Instantiate an object in the `Game` class. Put the object in the game with the `show` method, then update the object in the `update` method.

In the `Bird` file, create the `Bird` class. It should have attributes for an image, an x-position, and a y-position. In the constructor, load the image and set the position of the bird toward the top-left corner.

```
class Bird {
  private PImage bird;
  private float x;
  private float y;

  public Bird() {
    bird = loadImage("bird.png");
    x = 70;
    y = 80;
  }
}
```

The `x` attribute needs a getter method. The `y` attribute needs a getter and a setter. Finally, create the `show` method which draws the bird image to the window using the `x` an `y` attributes.

```
public float getX() {
  return x;
}

public float getY() {
  return y;
}

public void setY(float newY) {
  y = newY;
}

public void show() {
  image(bird, x, y);
}
```

In the `Game` file, create the `bird` attribute, and instantiate the `Bird` object in the constructor. In the `show` method, add `bird.show();`. This draws the bird image to the game window.

```
class Game {
  private PImage background;
  private Ground ground;
  private Bird bird;

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
    bird = new Bird();
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {
    ground.show();
    bird.show();
  }

  public void update() {
    ground.update();
  }
}
```

info

Try It

Click the triangle button to run you program. You should see the bird image toward the top-left corner of the window. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

Gravity

Be sure you are in the `Bird` file. Gravity should constantly be pulling the bird towards the ground. In addition, the bird should fall faster the further it falls. To do this, we need attributes for `gravity` and `velocity`. Add them to the `Bird` class and the constructor.

```
class Bird {
  private PImage bird;
  private float x;
  private float y;
  private float gravity;
  private float velocity;

  public Bird() {
    bird = loadImage("bird.png");
    x = 70;
    y = 80;
    gravity = 0.1;
    velocity = 0;
  }
```

Add the setter method `seVelocity` to the `Bird` class. This will be used later in the program.

```
  public void setVelocity(float newVelocity) {
    velocity = newVelocity;
  }
```

Create the `update` method for the `Bird` class. The `velocity` should increase by the amount of `gravity` for every frame of the game. Then add `velocity` to the y-position of the bird. Finally, the bird should not be allowed to fly through the top of the window or fall through the bottom of the window. Use the `constrain` method for the `y` attribute. The `0` represents the top of the window; `612` represents when the bottom of the bird image touches the top of the ground image.

```
  public void update() {
    velocity += gravity;
    y += velocity;
    y = constrain(y, 0, 612);
  }
```

In the `Game` file, modify the `update` method so that the `bird` object calls its `update` method.

```
  public void update() {
    ground.update();
    bird.update();
  }
```

info

Try It

Click the triangle button to run you program. You should see the bird fall to the ground and stay there. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

Flapping the Bird’s Wings

Click the mouse will cause the bird to “flap” its wings. **Go to `FlappyBird` file**. Processing has a built-in method called `mouseClicked`. This is called whenever the mouse button is pressed and then released. When this happens, have the bird flap its wings. However, we are not in the `Game` file, so you have to say `game.bird.flap` to call the `flap` method on the `bird` object.

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}

void mouseClicked() {
  game.bird.flap();
}
```

Go to the `Bird` file. Add the `flap` method to the `Bird` class. Set `velocity` to 0 so that the bird stops its downward descent. Then subtract 2.5 from the `velocity` attribute. This will cause the bird to fly up a bit.

```
public void flap() {
  velocity = 0;
  velocity -= 2.5;
}
```

info

Try It

Click the triangle button to run you program. You should see the bird rise when the mouse is clicked. The bird should not be able to fly out of the window. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

Your code should look like this:

`FlappyBird` File

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}

void mouseClicked() {
  game.bird.flap();
}
```

`Game` File

```java
class Game {
  private PImage background;
  private Ground ground;
  private Bird bird;

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
    bird = new Bird();
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {
    ground.show();
    bird.show();
  }

  public void update() {
    ground.update();
    bird.update();
  }
}
```

`Ground` File

```
class Ground {
  private PImage ground;
  private int x;

  public Ground() {
    ground = loadImage("ground.png");
    x = 0;
  }

  public void show() {
    image(ground, x, 650);
    image(ground, x + 470, 650);
  }

  public void update() {
    x -= 1;
    if (x <= -470) {
      x = 0;
    }
  }
}
```

`Bird` File

```
class Bird {
  private PImage bird;
  private float x;
  private float y;
  private float gravity;
  private float velocity;

  public Bird() {
    bird = loadImage("bird.png");
    x = 70;
    y = 80;
    gravity = 0.1;
    velocity = 0;
  }

  public float getX() {
    return x;
  }

  public float getY() {
    return y;
  }

  public void setY(float newY) {
    y = newY;
  }

  public void setVelocity(float newVelocity) {
    velocity = newVelocity;
  }

  public void show() {
    image(bird, x, y);
  }

  public void update() {
    velocity += gravity;
    y += velocity;
    y = constrain(y, 0, 612);
  }

  public void flap() {
    velocity = 0;
    velocity -= 2.5;
  }
}
```

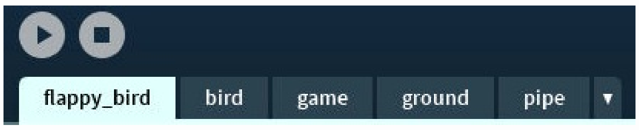
Advanced Topics Lab 4

Lab 4 - Pipes

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `advanced`, and double click on `FlappyBird`. Finally, open the `FlappyBird.pde` file. This should open five different files.



Processing IDE with all of the game files open in separate tabs

Go to the `Pipe` file. Create the `Pipe` class. It has `x` and `y` coordinates. A `Pipe` object has both a top pipe and a bottom pipe, so the class needs an image for each of them. The `speed` attribute controls how fast the pipes move. Finally, the `heights` attribute is an array of 3 different integers. These numbers determine if the gap between the pipes is towards the bottom, in the middle, or towards the top of the window.

```
class Pipe {
  private int x;
  private int y;
  private PImage top;
  private PImage bottom;
  private float speed;
  private int[] heights = new int[3];
}
```

The `Pipe` constructor takes an argument for the `x` attribute. The first thing to do is give the `heights` array its values. The `y` attribute is randomly selected from the `heights` array. Set the speed attribute, and load images for the two pipes.

```
  public Pipe(int xPos) {
    heights = new int[]{295, 425, 562};
    x = xPos;
    y = heights[(int)random(heights.length)];
    speed = 2.0;
    top = loadImage("topPipe.png");
    bottom = loadImage("bottomPipe.png");
  }
```

Go to the `Game` file. We want four different pipes, so we need the attribute `pipes` which is an array of type `Pipe` with a length of 4. Since `pipes` is an array, we need a loop that runs four times in the constructor. Each time the loop runs, a new `Pipe` object is created. The `250` represents the number of pixels between each set of pipes.

```
class Game {
  private PImage background;
  private Ground ground;
  private Bird bird;
  private Pipe[] pipes = new Pipe[4];

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
    bird = new Bird();

    for (int i = 0; i < pipes.length; i++) {
      pipes[i] = new Pipe(width + i * 250);
    }
  }
```

Drawing and Moving the Pipes

In the `show` method for the `Game` class, loop through the `pipes` array and call the `show` method for each element. Processing will show an error because the `show` method has not yet been created. This is okay for now. We want the ground image to appear on top of the pipes, so put the loop before the `ground` object.

```
public void show() {
  for (Pipe p : pipes) {
    p.show();
  }
  ground.show();
  bird.show();
}
```

We are going to do something similar in the `update` method. Instead of calling `show`, call the `update` method. Again, Processing shows an error, but this is okay. It does not matter if the loop is before the `ground` object.

```
public void update() {
  ground.update();
  bird.update();

  for (Pipe p : pipes) {
    p.update();
  }
```

Go to the `Pipe` file. Create the `show` method which will display the top and bottom pipes. We want to create a gap of 224 pixels between the pipes. That is why the y-position of the top pipe subtracts 635 (the height of the pipe image plus 224). In the `update` method, subtract the `speed` attribute from the `x` attribute.

```
public void show() {
  image(top, x, y - 635);
  image(bottom, x, y);
}

public void update() {
  x -= speed;
}
```

info

Try It

Click the triangle button to run you program. You should see four sets of pipes (top and bottom) move to the left. Eventually they will all disappear. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

Your code should look like this:

`FlappyBird` File

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}

void mouseClicked() {
  game.bird.flap();
}
```

`Game` File

```
class Game {
  private PImage background;
  private Ground ground;
  private Bird bird;
  private Pipe[] pipes = new Pipe[4];

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
    bird = new Bird();

    for (int i = 0; i < pipes.length; i++) {
      pipes[i] = new Pipe(width + i * 250);
    }
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {
    for (Pipe p : pipes) {
      p.show();
    }
    ground.show();
    bird.show();
  }

  public void update() {
    ground.update();
    bird.update();

    for (Pipe p : pipes) {
      p.update();
    }
  }
}
```

Ground File

```
class Ground {
  private PImage ground;
  private int x;

  public Ground() {
    ground = loadImage("ground.png");
    x = 0;
  }

  public void show() {
    image(ground, x, 650);
    image(ground, x + 470, 650);
  }

  public void update() {
    x -= 1;
    if (x <= -470) {
      x = 0;
    }
  }
}
```

`Bird` File

```
class Bird {
  private PImage bird;
  private float x;
  private float y;
  private float gravity;
  private float velocity;

  public Bird() {
    bird = loadImage("bird.png");
    x = 70;
    y = 80;
    gravity = 0.1;
    velocity = 0;
  }

  public float getX() {
    return x;
  }

  public float getY() {
    return y;
  }

  public void setY(float newY) {
    y = newY;
  }

  public void setVelocity(float newVelocity) {
    velocity = newVelocity;
  }

  public void show() {
    image(bird, x, y);
  }

  public void update() {
    velocity += gravity;
    y += velocity;
    y = constrain(y, 0, 612);
  }

  public void flap() {
    velocity = 0;
    velocity -= 2.5;
  }
}
```

`Pipe` File

```
class Pipe {
  private int x;
  private int y;
  private PImage top;
  private PImage bottom;
  private float speed;
  private int[] heights = new int[3];

  public Pipe(int xPos) {
    heights = new int[]{295, 425, 562};
    x = xPos;
    y = heights[(int)random(heights.length)];
    speed = 2.0;
    top = loadImage("topPipe.png");
    bottom = loadImage("bottomPipe.png");
  }

  public void show() {
    image(top, x, y - 635);
    image(bottom, x, y);
  }

  public void update() {
    x -= speed;
  }
}
```

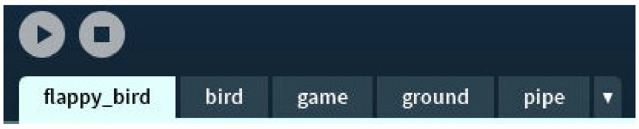
Advanced Topics Lab 5

Lab 5 - Repeating the Pipes

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `advanced`, and double click on `FlappyBird`. Finally, open the `FlappyBird.pde` file. This should open five different files.



Processing IDE with all of the game files open in separate tabs

Go to the `Pipe` file. Each time a pipe leaves the game window to the left, it needs to reappear. This pipe should also choose a different height to add a bit of randomness to the game. When the `x` attribute is less than -80, then the pipe has passed the left side of the game window. Modify the `update` method to check for this condition. If true, call the `startOver` method.

```
public void update() {
  x -= speed;
  if (x < -80) {
    startOver();
  }
}
```

We now need to declare the `startOver` method. Set the `x` attribute to 910. This means the “first” pipe becomes the “fourth” pipe. The also maintains the same distance between pipes. The `y` attribute should change as well. Select at random an element from the `pipes` array.

```
private void startOver() {
  x = 910;
  y = heights[(int)random(heights.length)];
}
```

info

Try It

Click the triangle button to run you program. You should see four sets of pipes (top and bottom) move to the left. This should be an infinite animation. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

Stopping the Game

When the bird hits a pipe, the game should be over. Create the method `touching` and pass it a `Bird` object. This method returns a boolean. Since most of the time the bird is not touching a pipe, the default return value is `false`. Determining if the bird is touching a pipe is a two-step problem. First, ask if the bird (defined by the `x` value plus 51) overlaps the `x` value of the pipe. That means the bird is greater than `x` (the front of the pipe) and less than `x` plus 80 (the end of the pipe).

```
public boolean touching(Bird bird) {
  if (bird.getX() + 51 > x && bird.getX() < x + 80) {

  }
  return false;
}
```

The next step in solving this problem is to ask if the bird is **not** flying through the vertical gap between the pipes. The bottom pipe is at location `y` and the gap is 224 pixels. So the top pipe is `y` plus 224. Add another conditional that asks if the bottom of the bird (`y` plus 38) is less than the bottom pipe and if the top of the bird is greater than the top pipe (`y` minus 224). Put the not operator (`!`) in front of this compound conditional. If a bird is overlapping the x-values of the pipe and it’s y-value is not in the gap, then the bird is touching a pipe. If all of this is true, return `true`.

```
public boolean touching(Bird bird) {
  if (bird.getX() + 51 > x && bird.getX() < x + 80) {
    if (!(bird.getY() + 38 < y && bird.getY() > y - 224)) {
      return true;
    }
  }
  return false;
}
```

Go to the `Game` file. Once we have determined that the bird is touching a pipe, we are going to stop the game. In the `update` method, find the for loop that moves each pipe. Use a conditional that asks if that pipe is touching the bird. If this is true, call the `gameOver` method.

```
  for (Pipe p : pipes) {
    p.update();
    if (p.touching(bird)) {
      gameOver();
    }
  }
```

For now, the `gameOver` method will completely stop the game. Use the `noLoop` command. This tells Processing to stop the `draw` method in the `FlappyBird` file. This stops the entire program.

```
public void gameOver() {
  noLoop();
}
```

info

Try It

Click the triangle button to run you program. When the bird hits a pipe, the game stops. You have to close the window before restarting the game. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

Your code should look like this:

`FlappyBird` File

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}

void mouseClicked() {
  game.bird.flap();
}
```

`Game` File

```
class Game {
  private PImage background;
  private Ground ground;
  private Bird bird;
  private Pipe[] pipes = new Pipe[4];

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
    bird = new Bird();

    for (int i = 0; i < pipes.length; i++) {
      pipes[i] = new Pipe(width + i * 250);
    }
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {
    for (Pipe p : pipes) {
      p.show();
    }
    ground.show();
    bird.show();
  }

  public void update() {
    ground.update();
    bird.update();

    for (Pipe p : pipes) {
      p.update();
      if (p.touching(bird)) {
        gameOver();
      }
    }
  }

  public void gameOver() {
    noLoop();
  }
}
```

Ground File

```
class Ground {
  private PImage ground;
  private int x;

  public Ground() {
    ground = loadImage("ground.png");
    x = 0;
  }

  public void show() {
    image(ground, x, 650);
    image(ground, x + 470, 650);
  }

  public void update() {
    x -= 1;
    if (x <= -470) {
      x = 0;
    }
  }
}
```

`Bird` File

```
class Bird {
  private PImage bird;
  private float x;
  private float y;
  private float gravity;
  private float velocity;

  public Bird() {
    bird = loadImage("bird.png");
    x = 70;
    y = 80;
    gravity = 0.1;
    velocity = 0;
  }

  public float getX() {
    return x;
  }

  public float getY() {
    return y;
  }

  public void setY(float newY) {
    y = newY;
  }

  public void setVelocity(float newVelocity) {
    velocity = newVelocity;
  }

  public void show() {
    image(bird, x, y);
  }

  public void update() {
    velocity += gravity;
    y += velocity;
    y = constrain(y, 0, 612);
  }

  public void flap() {
    velocity = 0;
    velocity -= 2.5;
  }
}
```

`Pipe` File

```
class Pipe {
  private int x;
  private int y;
  private PImage top;
  private PImage bottom;
  private float speed;
  private int[] heights = new int[3];

  public Pipe(int xPos) {
    heights = new int[]{295, 425, 562};
    x = xPos;
    y = heights[(int)random(heights.length)];
    speed = 2.0;
    top = loadImage("topPipe.png");
    bottom = loadImage("bottomPipe.png");
  }

  public void show() {
    image(top, x, y - 635);
    image(bottom, x, y);
  }

  public void update() {
    x -= speed;
    if (x < -80) {
      startOver();
    }
  }

  private void startOver() {
    x = 910;
    y = heights[(int)random(heights.length)];
  }

  public boolean touching(Bird bird) {
    if (bird.getX() + 51 > x && bird.getX() < x + 80) {
      if (!(bird.getY() + 38 < y && bird.getY() > y - 224)) {
        return true;
      }
    }
    return false;
  }
}
```

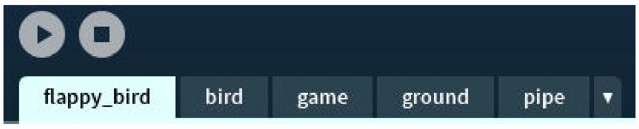
Advanced Topics Lab 6

Lab 6 - Showing the Score

info

Open the File

You have to tell Processing the file you want to open. In the Processing window, click `File` then `Open...`. On the left towards the bottom of the list, click `workspace`. Double click on `code`, double click on `advanced`, and double click on `FlappyBird`. Finally, open the `FlappyBird.pde` file. This should open five different files.



Processing IDE with all of the game files open in separate tabs

Go to the `Game` file. We need to determine when the game is being played and when it is over. Add the `active` attribute to the `Game` class. When this attribute is `true`, the game is being played. If `false`, then the game is over. Set the value to `true` in the constructor. Also add the `score` attribute of type float. Set its value to 0 in the constructor.

```
class Game {
  private PImage background;
  private Ground ground;
  private Bird bird;
  private Pipe[] pipes = new Pipe[4];
  private boolean active;
  private float score;
  private PFont scoreFont;

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
    bird = new Bird();
    active = true;
    score = 0;
    scoreFont = createFont("Dejavu Sans Bold", 32);

    for (int i = 0; i < pipes.length; i++) {
      pipes[i] = new Pipe(width + i * 250);
    }
  }
```

Find the `update` method in the `Game` class. We are only going to update the `Bird` object only if `active` is true. We are also going to update the score with the `updateScore` method.

```
    if (active) {
      bird.update();
      updateScore();
    }
```

Define the `updateScore` method so that it adds 0.01 to the `score` attribute every frame of the game. The score slowly increases as the game does not end.

```
  public void updateScore() {
    score += 0.01;
  }
```

Find the `show` method in the `Game` class. Add a conditional so that the bird only appears in the game if `active` is true. Then call the `showScore` method.

```
    if (active) {
      bird.show();
    }

    showScore();
```

Start by aligning the text to be centered left and right as well as up and down. If `active` is true, the set the font to `scoreFont` and make the font white. Remember, the `score` attribute is a float, but we only want to display the score as a whole number. Convert `score` to an integer and place it in the top-left corner.

```
  public void showScore() {
    textAlign(CENTER, CENTER);
    if (active) {
      textFont(scoreFont);
      fill(255);
      text((int)score, 25, 25);
    }
  }
```

Finally, change the `gameOver` method so that it changes `active` to false. Remove the line about `noLoop`. The bird and score should disappear while pipes and ground are still animated.

```
  public void gameOver() {
    active = false;
  }
```

info

Try It

Click the triangle button to run you program. As you play the game should increase. When you hit a pipe, the bird should disappear from the window. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

High Score

The next step is to keep track of the high score. Add the attributes `highScore` and `msgFont` to the `Game` class.

```
private float highScore;
private PFont msgFont;
```

In the constructor, set `highScore` to 0 and set the font family and size for `msgFont`.

```
  highScore = 0;
  msgFont = createFont("Dejavu Sans Bold", 40);
```

We want to show a message when the game is over. Modify the `showScore` method with and `else` statement. Create the variable `msg` that has the message to appear when the game is over. This is a long message, so use the newline character (`\n`) to break up the message over three lines. Concatenate the `highScore` attribute as well. Set the color to red, and place the message in the middle of the window.

```
public void showScore() {
  textAlign(CENTER, CENTER);
  if (active) {
    textFont(scoreFont);
    fill(255);
    text((int)score, 25, 25);
  } else {
    String msg = "Press Space Bar\nto Play Again\nHigh Score:
      " + (int)highScore;
    textFont(msgFont);
    fill(255, 0, 0);
    text(msg, width/2, height/2);
  }
}
```

Right now, `highScore` is 0. In the `gameOver` method, ask if `score` is greater than `highScore`. If so, set `highScore` to the value of `score`. This should allow the high score to appear in the message when the game is over.

```
public void gameOver() {
  active = false;
  if (score > highScore) {
    highScore = score;
  }
}
```

info

Try It

Click the triangle button to run you program. When you hit a pipe, a red message appears on the screen with the high score. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

Restarting the game

The final part of the game is to restart it when the space bar is pressed. In the `update` method of the `Game` class, add an `else if` statement for when `active` is false. `keypressed` is a built-in method that returns true if any key on the keyboard is pressed. You then have to ask if `key` (a built-in variable that stores the key that was pressed) if the space character. If this is true, call the `restart` method.

```
    if (active) {
      bird.update();
      updateScore();
    } else if (keyPressed) {
      if (key  == ' ') {
        restart();
      }
    }
```

The `restart` method should put the game back to its starting state. Set `active` to true, set `score` to 0, put the `bird` back to 80 pixels from the top of the window, set the `velocity` of bird back to 0, and set the pipes back to their original starting positions.

```
public void restart() {
  active = true;
  score = 0;
  bird.setY(80);
  bird.setVelocity(0);

  for (int i = 0; i < pipes.length; i++) {
    pipes[i] = new Pipe(width + i * 250);
  }
}
```

info

Try It

Click the triangle button to run you program. After hitting a pipe, you should be able to restart the game by pressing the space bar. Close the window or click the square button to stop the program. **Note**, you need to manually save your work in Processing. Click `File` and then `Save`, or press `Ctrl + S` on the keyboard to save your work.

▼ **Code**

Your code should look like this:

`FlappyBird` File

```
Game game;

void setup() {
  size(400, 719);
  game = new Game();
}

void draw() {
  background(game.getBackground());
  game.show();
  game.update();
}

void mouseReleased() {
  game.bird.flap();
}
```

Game File

```
class Game {
  private PImage background;
  private Ground ground;
  private Bird bird;
  private Pipe[] pipes = new Pipe[4];
  private boolean active;
  private float score;
  private PFont scoreFont;
  private float highScore;
  private PFont msgFont;

  public Game() {
    background = loadImage("background.png");
    ground = new Ground();
    bird = new Bird();
    active = true;
    score = 0;
    scoreFont = createFont("Dejavu Sans Bold", 32);
    highScore = 0;
    msgFont = createFont("Dejavu Sans Bold", 40);

    for (int i = 0; i < pipes.length; i++) {
      pipes[i] = new Pipe(width + i * 250);
    }
  }

  public PImage getBackground() {
    return background;
  }

  public void show() {
    for (Pipe p : pipes) {
      p.show();
    }
    ground.show();
    if (active) {
      bird.show();
    }
    showScore();
  }

  public void update() {
    ground.update();
    if (active) {
      bird.update();
```

```
        updateScore();
      } else if (keyPressed) {
        if (key  == ' ') {
          restart();
        }
      }

      for (Pipe p : pipes) {
        p.update();
        if (p.touching(bird)) {
          gameOver();
        }
      }
    }

    public void gameOver() {
      active = false;
      if (score > highScore) {
        highScore = score;
      }
    }

    public void updateScore() {
      score += 0.01;
    }

    public void showScore() {
      textAlign(CENTER, CENTER);
      if (active) {
        textFont(scoreFont);
        fill(255);
        text((int)score, 25, 25);
      } else {
        String msg = "Press Space Bar\nto Play Again\nHigh
        Score: " + (int)highScore;
        textFont(msgFont);
        fill(255, 0, 0);
        text(msg, width/2, height/2);
      }
    }

    public void restart() {
      active = true;
      score = 0;
      bird.setY(80);
      bird.setVelocity(0);

      for (int i = 0; i < pipes.length; i++) {
```

```
      pipes[i] = new Pipe(width + i * 250);
    }
  }
}
```

Ground File

```
class Ground {
  private PImage ground;
  private int x;

  public Ground() {
    ground = loadImage("ground.png");
    x = 0;
  }

  public void show() {
    image(ground, x, 650);
    image(ground, x + 470, 650);
  }

  public void update() {
    x -= 1;
    if (x <= -470) {
      x = 0;
    }
  }
}
```

Bird File

```
class Bird {
  private PImage bird;
  private float x;
  private float y;
  private float gravity;
  private float velocity;

  public Bird() {
    bird = loadImage("bird.png");
    x = 70;
    y = 80;
    gravity = 0.1;
    velocity = 0;
  }

  public float getX() {
    return x;
  }

  public float getY() {
    return y;
  }

  public void setY(float newY) {
    y = newY;
  }

  public void setVelocity(float newVelocity) {
    velocity = newVelocity;
  }

  public void show() {
    image(bird, x, y);
  }

  public void update() {
    velocity += gravity;
    y += velocity;
    y = constrain(y, 0, 612);
  }

  public void flap() {
    velocity = 0;
    velocity -= 2.5;
  }
}
```

Pipe File

```
class Pipe {
  private int x;
  private int y;
  private PImage top;
  private PImage bottom;
  private float speed;
  private int[] heights = new int[3];

  public Pipe(int xPos) {
    heights = new int[]{295, 425, 562};
    x = xPos;
    y = heights[(int)random(heights.length)];
    speed = 2.0;
    top = loadImage("topPipe.png");
    bottom = loadImage("bottomPipe.png");
  }

  public void show() {
    image(top, x, y - 635);
    image(bottom, x, y);
  }

  public void update() {
    x -= speed;
    if (x < -80) {
      startOver();
    }
  }

  private void startOver() {
    x = 910;
    y = heights[(int)random(heights.length)];
  }

  public boolean touching(Bird bird) {
    if (bird.getX() + 51 > x && bird.getX() < x + 80) {
      if (!(bird.getY() + 38 < y && bird.getY() > y - 224)) {
        return true;
      }
    }
    return false;
  }
}
```

Advanced Topics Lab Challenge

Problem

The `BankAccount` class is defined in the IDE to the left. Create the `toString` method that returns a string representation of a `BankAccount` object.

Expected Output

Create two different `BankAccount` objects.

```
    //add code below this line

    BankAccount account1 = new BankAccount(2432, 89.52);
    BankAccount account2 = new BankAccount(1998, 239.43);

    //add code above this line
```

If you print each object, you should see the following output.

Method Call	Output
`System.out.println(account1);`	BankAccount[checking=2432.0, savings=89.52]
`System.out.println(account2);`	BankAccount[checking=1998.0, savings=239.43]