

2D GAME DEVELOPMENT REPORT

KURT ABANO

[STUDENT ID: 24783569]

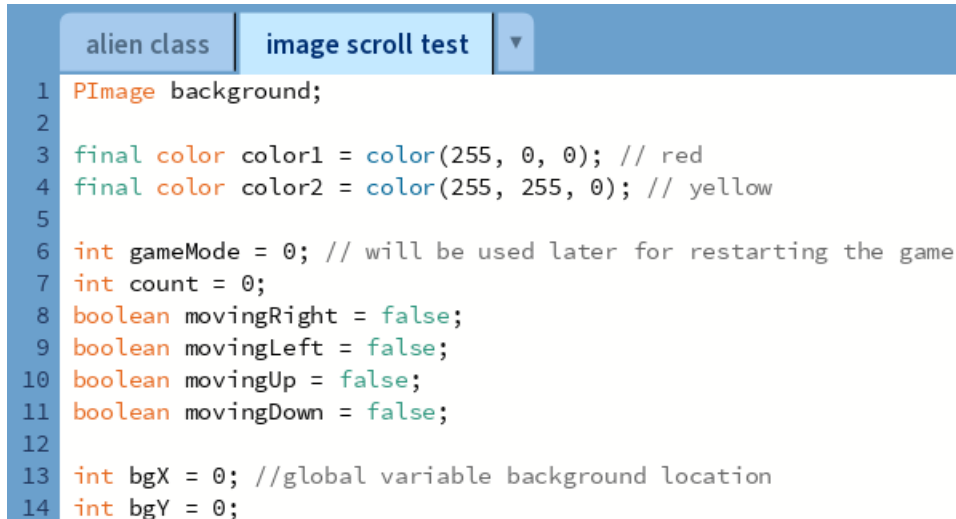
Table of Contents

Version one	3
Scrolling background	3
Version 2	7
Inheritance and Polymorphism.....	7
Implementing an array list.....	12
Game over screen	15
Restarting game	16
Collectable object.....	17
Save and read score (file handling).....	19
Version 3	22
Adding a Player class and player aniamtions	22
Animations for obstacles and collectables.....	27
Death animation.....	29

Version one

Scrolling background

For this game the first thing I wanted to implement was a background that moves depending on the arrow keys pressed.



```
1 PImage background;
2
3 final color color1 = color(255, 0, 0); // red
4 final color color2 = color(255, 255, 0); // yellow
5
6 int gameMode = 0; // will be used later for restarting the game
7 int count = 0;
8 boolean movingRight = false;
9 boolean movingLeft = false;
10 boolean movingUp = false;
11 boolean movingDown = false;
12
13 int bgX = 0; //global variable background location
14 int bgY = 0;
```

The image above shows the global variables I'll be using. The important variables to take note of are the bgX, bgY and the movingRight left up down.

I am going to have it so that when any arrow key is pressed the Boolean values change to true. In my void draw, I'll have an if statement where if the movingRight, left, up or down is true then bgX and bgY will increment by a set amount. This will move the background

```
89 void keyPressed() {
90   if (key == CODED) {
91     if (keyCode == LEFT) {
92       movingLeft = true;
93     } else if (keyCode == RIGHT) {
94       movingRight = true;
95     } else if (keyCode == UP) {
96       movingUp = true;
97     } else if (keyCode == DOWN) {
98       movingDown = true;
99     }
100   }
101 }
102
103 void keyReleased() {
104   if (key == CODED) {
105     if (keyCode == LEFT) {
106       movingLeft = false;
107     } else if (keyCode == RIGHT) {
108       movingRight = false;
109     } else if (keyCode == UP) {
110       movingUp = false;
111     } else if (keyCode == DOWN) {
112       movingDown = false;
113     }
114   }
115 }
116
117
21 void setup() {
22
23   size(800, 400);
24   background = loadImage("seemlessGrass.jpg");
25   background.resize(width, height); //set image to be same size as the canvas
26
27 }
```

using a generic grass jpeg for testing

```

34 void draw ()
35 {
36     //scrolling background image
37     image(background, bgX, bgY); //draw image to fill the canvas
38
39     for (int i = 0; i < 100; i++) {
40         for (int j = 0; j < 100; j++) {
41             image(background, bgX - (i * background.width), bgY + (j * background.height));
42             image(background, bgX + (i * background.width), bgY - (j * background.height));
43             image(background, bgX - (i * background.width), bgY - (j * background.height));
44             image(background, bgX + (i * background.width), bgY + (j * background.height));
45             // repeatedly draws the images around the original image (100 rows and columns)
46         }
47     }
48 }
49
50

```

My draw method pastes an image over 100 rows and columns so that when the character moves it looks as if the background is continuously looping.

```

54 fill(0,0,255);
55 ellipse(width/2, height/2, 20,20);
56
57
58 if (movingRight) {
59     bgX -= 3; // Move smoothly right
60 }
61 if(movingLeft){
62     bgX += 3; // Move smoothly left
63 }
64
65 ////////////////////////////////////////////////// now for y axis
66 if (movingUp) {
67     bgY += 3; // Move smoothly up
68 }
69 if (movingDown) {
70     bgY -= 3; // Move smoothly down
71 }
72

```

Stationary ellipse is a placeholder for the player.

The if statements will determine if the background should move. If any of them are true, then bgX and bgY will increment appropriately which will move the background.



This is the background; the seams are where the other pasted images are. Pressing the arrow keys moves the background.

After this I made my first obstacle that moves.

```

1 | class Obstacle {
2 |     int x =5;
3 |     int y = 30;
4 |     color colour;
5 |     int speed = int(random(4, 8));
6 |
7 |
8 |     Obstacle(int x, int y, color col) {
9 |         this.colour =col;
10 |         this.y = y;
11 |         this.x =x;
12 |     }
13 |

```

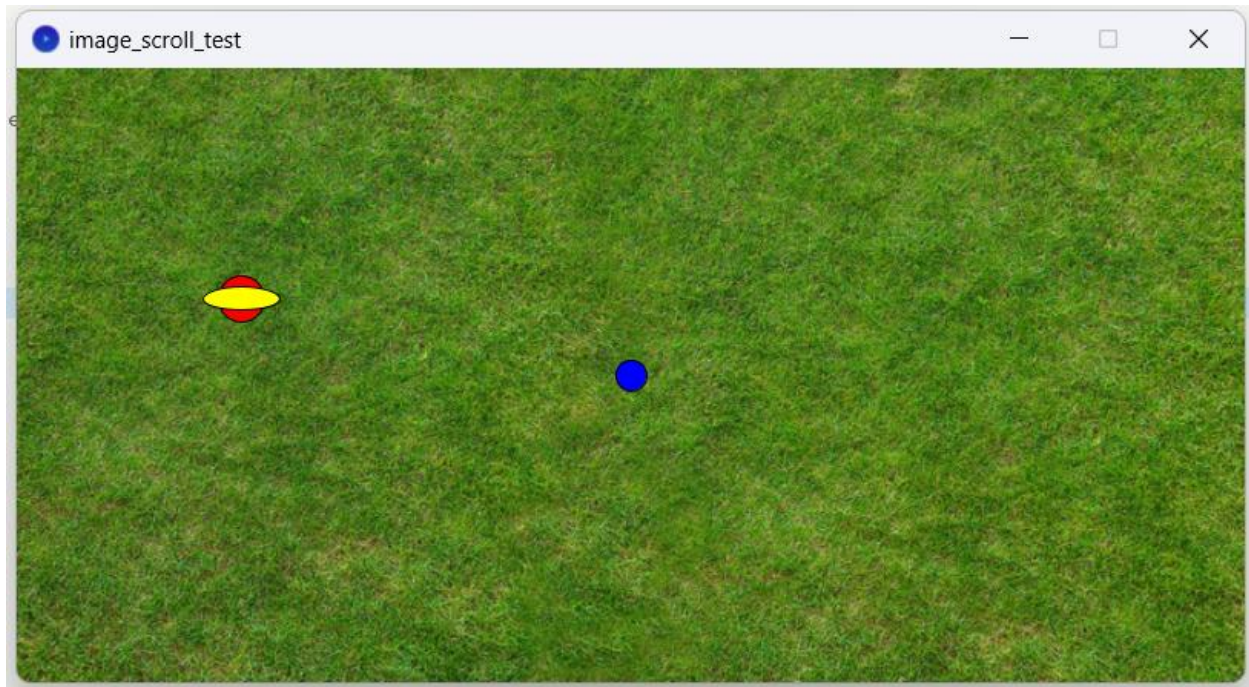
Takes in color as a parameter and has a random speed.

```

22 | void move() {
23 |     x = x-speed;
24 |     if (x< 0) { // if off the screen bring it back to the right edge
25 |         x =800;
26 |         speed =int(random(5, 12));
27 |         y = int(random(height)); // uses a random y value
28 |     }
29 | }

```

Has a method that moves it horizontally. Once it moves off the screen it respawns at a different y value. (movement is temporary, this will be changed).



Version 2

Features to add:

- Movement update
- Array list
- Inheritance and polymorphism
- Collision
- Stationary objects
- Restarting

Inheritance and Polymorphism

Updated my Obstacle class structure made Obstacle a superclass. Created MovingObstacle and StationaryObstacle classes to inherit from it.

```

1  class Obstacle {
2      int x, y;
3      color moverColor;
4
5      Obstacle(color col) {
6          this.moverColor = col;
7          respawn(); // automatically spawns in the object (will be overwritten)
8      }
9
10     void respawn() {
11         // Base respawn if no polymorphism takes place
12         x = int(random(width));
13         y = int(random(height));
14     }
15
16     void display() {
17         // Base display
18         fill(moverColor);
19         ellipse(x, y, 30, 30);
20     }
21 }
22

```

movingObject

```

1  class movingObstacle extends Obstacle {
2      int speedX, speedY; // initialise speed variables, other classes wont have this
3
4      movingObstacle(color col) {
5          super(col); // color from the super class
6          setRandomSpeed(); // Initialize movement speed
7      }
8
9      void setRandomSpeed() {
10         speedX = int(random(-5, 5));
11         speedY = int(random(-5, 5));
12         if (speedX == 0 && speedY == 0) {
13             speedX = 1; // make sure speed is never 0
14         }
15     }
16 }

```

Takes color from the super class. I created a method to set the speed randomly. This function will be called every time the object respawns. This also determines the direction the objects moves in as negative numbers are included


```

17  @Override
18  void respawn() {
19      // Respawn at edges
20      int edge = int(random(4));
21      switch (edge) {
22          case 0: // Top edge
23              x = int(random(width));
24              y = -25;
25              break;
26          case 1: // Bottom edge
27              x = int(random(width));
28              y = height + 25;
29              break;
30          case 2: // Left edge
31              x = -25;
32              y = int(random(height));
33              break;
34          case 3: // Right edge
35              x = width + 25;
36              y = int(random(height));
37              break;
38      }
39      setRandomSpeed(); // new random speed
40  }
41

```

This respawns method overrides the one in the super class. When the object exits the screen, it will respawn randomly at one of the edges of the screen.

```

42  void move() {
43      x += speedX;
44      y += speedY;
45
46      // Check bounds and respawn
47      if (x < -25 || x > width + 25 || y < -25 || y > height + 25) {
48          respawn();
49      }
50  }
51
52  @Override
53  void display() {
54      fill(moverColor);
55      ellipse(x, y, 30, 30); // Circle place holder
56  }
57
58  void update(){
59      display();
60      move();
61  }

```

Our move function does not need to overwrite anything as this method is exclusive to this

class. In our previous move function, the object would only move horizontally but this one can also move vertically. The direction is determined on whether speedX or speedY is positive or negative

```
if (x < -25 || x > width + 25 || y < -25 || y > height + 25) {
    respawn();
}
```

This checks if the object exits the screen, then it recalls the respawn method.

stationaryObject

```
1 class stationaryObstacle extends Obstacle {
2
3     stationaryObstacle(color col) {
4         super(col); // inherit color from superclass
5
6         this.x = int(random(width)); // spawns in a random location
7         this.y = int(random(height));
8     }
9 }
```

inherits color and has a random x and y.

```
10
11 @Override
12 void respawn() {
13     // Respawn at a random fixed position (stationary)
14     int edge = int(random(4));
15     switch (edge) {
16         case 0: // Top edge
17             x = int(random(width));
18             y = -25;
19             break;
20         case 1: // Bottom edge
21             x = int(random(width));
22             y = height + 25;
23             break;
24         case 2: // Left edge
25             x = -25;
26             y = int(random(height));
27             break;
28         case 3: // Right edge
29             x = width + 25;
30             y = int(random(height));
31             break;
32     }
33 }
```

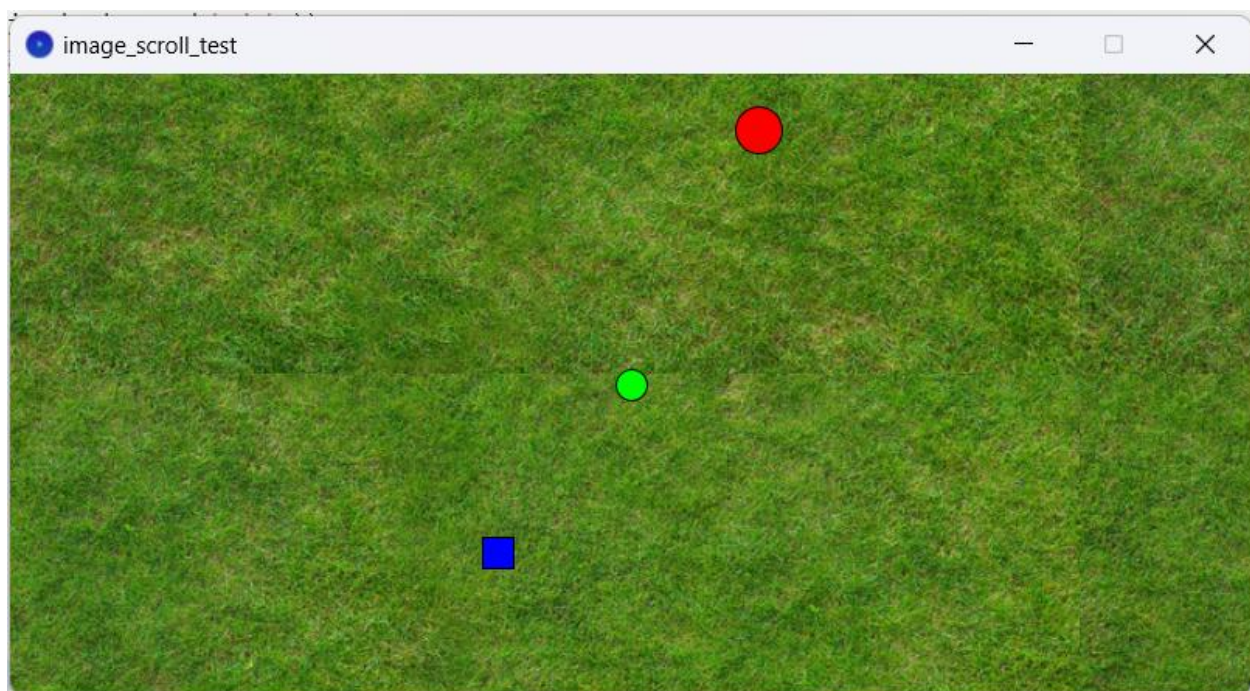
Respawn method is overwritten again (Polymorphism). The difference between this and the one in movingObjects is that speed isn't changed as this class doesn't have a speed attribute

```
35 @Override
36 void display() {
37     fill(moverColor);
38     rect(x - 10, y - 10, 20, 20); // square (temporary placeholder)
39
40
41     if (x < -25 || x > width + 25 || y < -25 || y > height + 25) {
42         respawn(); // if the object exits the screen then we call the respawn method
43     }
44 }
```

As this is a different class the display method is overwritten (Polymorphism) so that we can distinguish between the two objects (moving and stationary classes).

This class also doesn't have a move method

Running my program:



blue represents the stationary obstacles

red represents the moving obstacles

Green is the player character (stays in the middle of the screen)

The respawn methods work correctly once these two objects are off the screen they respawn in a different edge. The moving object also moves in a random direction as speedX and speedY are random and can be negative or positive.

Implementing an array list

Right now, we only have two obstacles on the screen. My game will need more than that, therefore, I'll be using an array list to store any amount of stationary and moving obstacles

Inside main:

```
6 movingObstacle[] movingArray = new movingObstacle[10];
7 stationaryObstacle[] stationaryArray = new stationaryObstacle[10];
```

This initializes my arrays. Here I decided to make an array that can hold 10 items

Inside void setup:

```
32 for (int i = 0; i < movingArray.length; i++) {
33     movingArray[i] = new movingObstacle(color1); // creates an object and stores it inside of the array
34 }
35
36 for (int i = 0; i < stationaryArray.length; i++) {
37     stationaryArray[i] = new stationaryObstacle(color2);
38 }
39 }
40 }
```

These four loops create the objects and store them inside of their arrays. Different colors are passed into them as well.

In my void draw:

```
72 // when an arrow key is pressed background and all obstacles move
73 if (movingRight) {
74     bgX -= 3; // Move smoothly up
75
76     for (movingObstacle obstacle : movingArray) {
77         obstacle.x -= 3; // obstacles x value increments
78     }
79
80     for (stationaryObstacle block : stationaryArray) {
81         block.x -= 3; // stationary obstacles x value increments
82     }
83 }
84 }
```

Since we are using an array list, we need each obstacle to move when the player moves, I have done this by using enhanced for loops

Another loop is used below to update all objects in the array

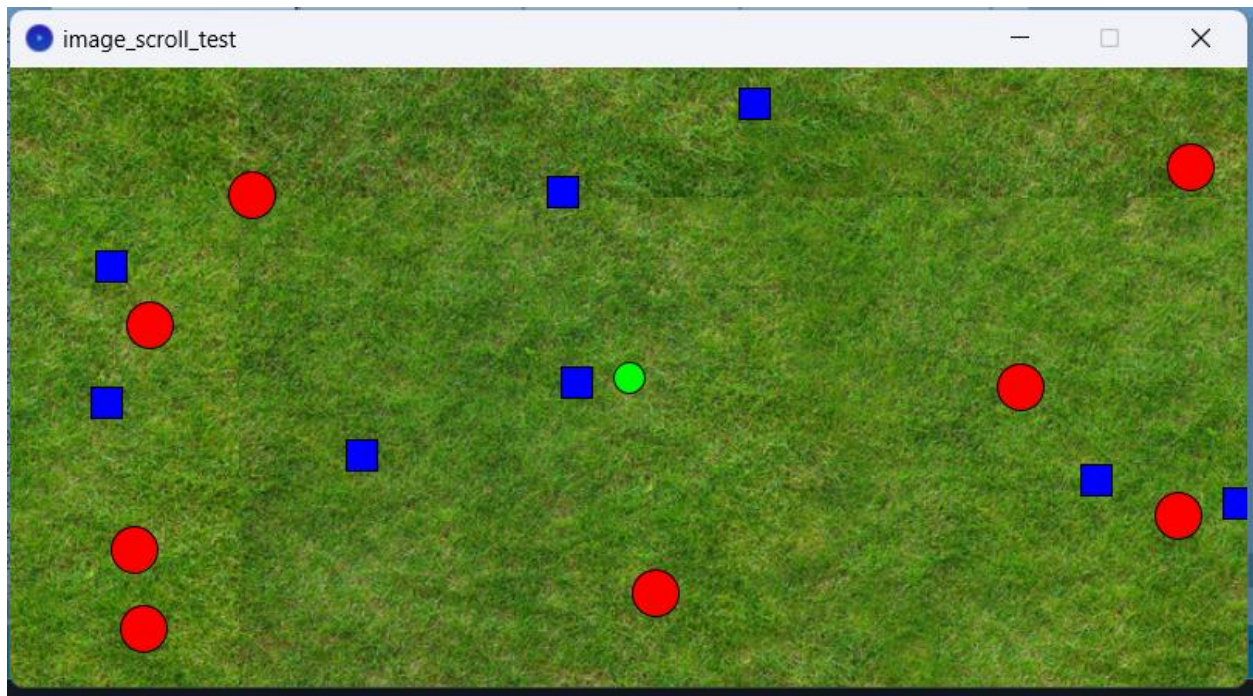
```
127 for (movingObstacle obstacle : movingArray) {
128     obstacle.update();
129 }
```

```

138     for ( stationaryObstacle block : stationaryArray ) {
139         block.display();
140     }

```

Testing my program:



The test was successful. Multiple objects from the array list is correctly displayed.

Collision

Now I am going to add collision so that when any obstacle hits the player then the game stops.

```

22     boolean crash() {
23
24         float targetRadius = 25;
25
26         // Check if the distance between the plane and the center of the screen is less than the radius of the ellipse
27         return dist(this.x, this.y, width / 2, height / 2) < targetRadius;
28     }
29 }

```

This is the method I coded to return the distance between the current object and the player.

Note that as of right now there is no player class, but we do know that the player will always be in the middle of the screen.

This method will be placed in the super class as both stationary and moving obstacles will inherit this method.

Back in main:

In void draw, we will continuously check whether a collision has happened. This check happens for all objects inside my arrays. I have done this by using an enhanced loop. An if statement will change gamemode to 1 if a collision occurs.

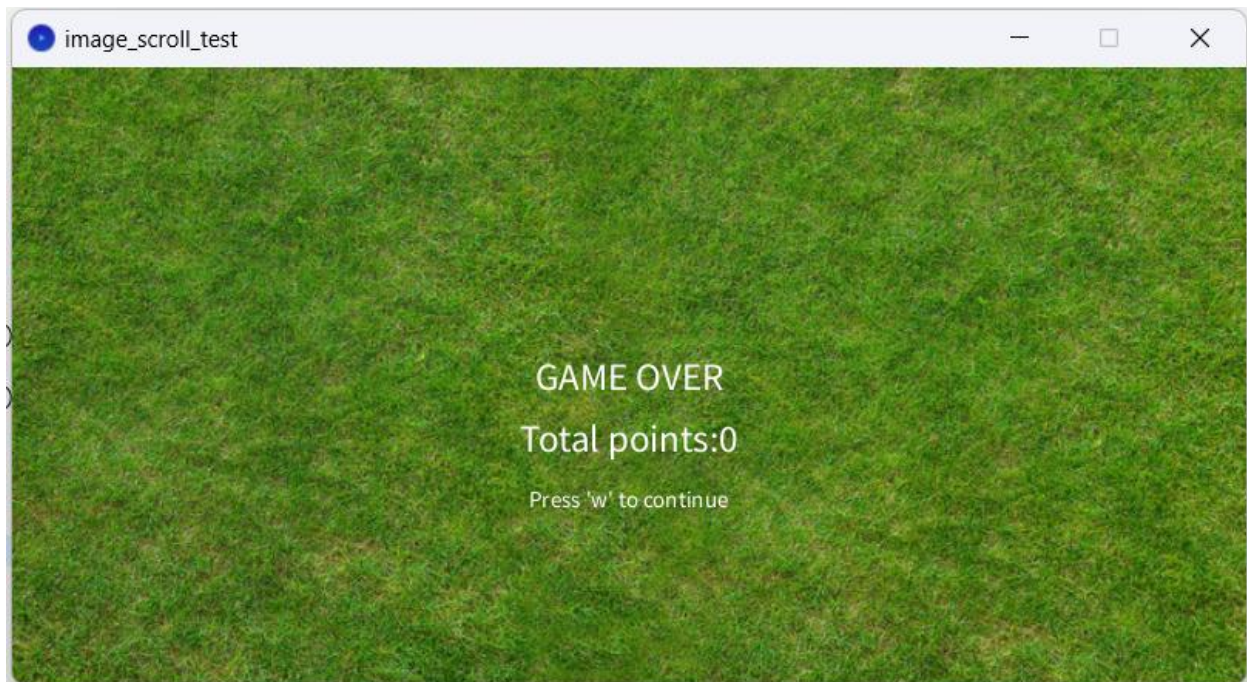
```
126     for (movingObstacle obstacle : movingArray) {
127         obstacle.update();
128
129         // Call the crash() method to check if a crash occurred
130         if (obstacle.crash() == true) {
131             gameMode = 1;
132         }
133     }
134
135
136     for ( stationaryObstacle block : stationaryArray ) {
137         block.display();
138
139         if (block.crash() == true) {
140             gameMode = 1;
141         }
142     }
```

I also changed my program so that void draw only moves the backgrounds and objects if gamemode is 0. This way when a crash occurs the program will stop calling the update/display methods for my objects effectively removing them from the screen

Game over screen

```
192 void displayText()
193 {
194     if (gameMode == 0)
195     {
196         textAlign(CENTER, CENTER);
197         textSize(25);
198         fill(255);
199         text("points:" + count, 80, 40);
200
201     } else if (gameMode == 1) {
202         textAlign(CENTER, CENTER);
203         textSize(25);
204         fill(255);
205         text("GAME OVER", width/2, height/2);
206         text("Total points:" + count, width/2, height/2 + 40);
207         textSize(15);
208         text("Press 'w' to continue", width/2, height/2 + 80);
209     }
210 }
```

This function will be called in void draw. It will display points and when a collision occurs (ie when gamemode becomes 1) new text is displayed



This is what is displayed when a collision occurs

```

191     for (int i = 0; i < movingArray.length; i++ ) {
192         movingArray[i] = null; // removes every obstacle object from the array list
193     }
194
195     for (int i = 0; i < stationaryArray.length; i++ ) {
196         stationaryArray[i] = null; // setting to null removes it form memory
197     }
198

```

When a game over occurs I turn all my objects in my array list to null. Removing them from memory.

Restarting game

Now I am going to implement a feature where when the game ends, we can restart it by pressing 'w'

To do this I added an if statement in the void keyReleased function for when 'w' is pressed and gameMode is 1, we can restart the game.

The code below shows the reinitialization of my obstacle objects. This will spawn new objects in different locations.

```

238     if ((key == 'w') && (gameMode == 1)) { // && gameMode== 1 ensures we can only restart the game after a collision
239         gameMode = 0; // meaning objects will be displayed again
240         score = 0;
241         scoreSaved = false; // Reset flag for the next game session
242
243         deathFinished = false;
244
245
246         for (int i = 0; i < movingArray.length; i++ ) {
247             movingArray[i] = new movingObstacle(color1); // reinitialise my enemy objects so they spawn in different spots and have different speeds
248         }
249
250         for (int i = 0; i < stationaryArray.length; i++ ) {
251             stationaryArray[i] = new stationaryObstacle(color2);
252         }
253     } // to restart the game i reinitialise / overwrite all the objects in my array list
254     // this will give new x and y coordinates to my objects when gamemode becomes 0 again
255

```


Collectable object

I decided to add a collectable class that will spawn collectable objects in my game. Collecting them will increment your score / points.

```

1 class Collectable {
2
3     int x, y;
4     color col;
5
6     Collectable(color col) {
7         this.col = col;
8
9         this.x = int(random(width)); // spawns in a random location
10        this.y = int(random(height));
11    }
12
13    ...

```

This class has a respawn method identical to the obstacle class.

```

44    boolean collect() {
45
46        float targetRadius = 25;
47
48        // Check if the distance between the plane and the center of the screen is less than the radius of the ellipse
49        return dist(this.x, this.y, width / 2, height / 2) < targetRadius;
50    }
51 }

```

similar to the crash Boolean method, this method determines whether the player is touching the collectable.

Back in main inside void draw:

```

155    coin.display(); // display coin
156    if (coin.collect() == true){
157        score += 1; // increment score if player touches collectable
158        coin.respawn(); // respawn coin in a different location
159    }

```

note that once the player collects the coin, the respawn method is called, placing the coin in a different location

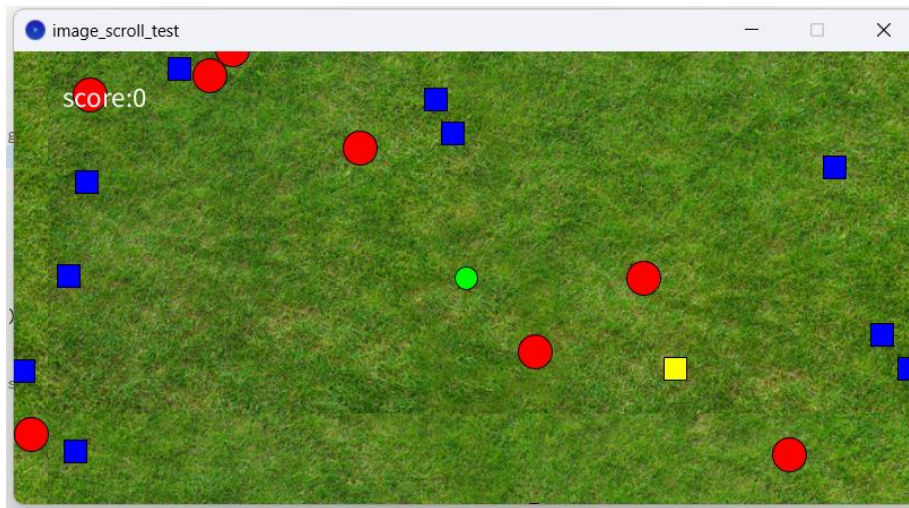
I also made the coins position change when the player moves (similar to the background and obstacles)

```

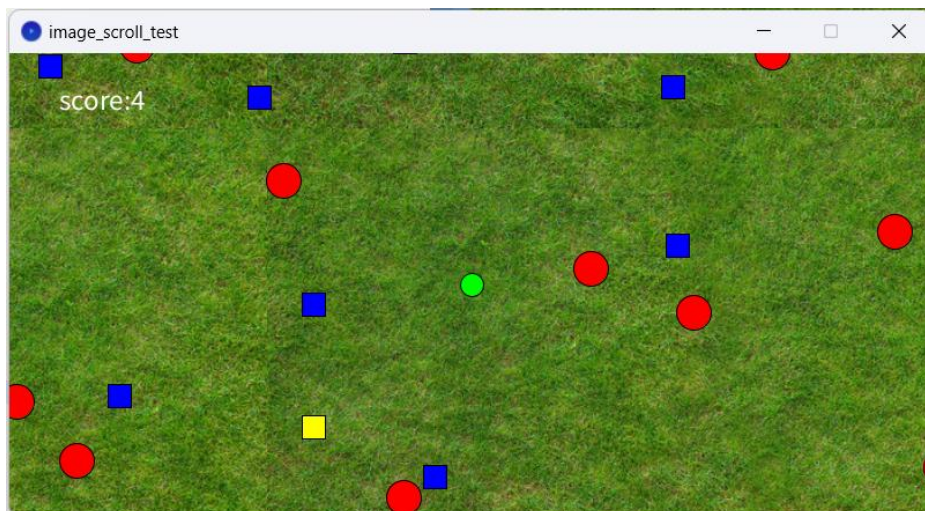
108    if (movingUp) {
109        bgY += 3; // Move smoothly up
110        coin.y += 3;
111
112        for (movingObstacle obstacle : movingArray) {
113            obstacle.y += 3;
114        }
115
116        for (stationaryObstacle block : stationaryArray) {
117            block.y += 3;
118        }
119    }
120

```

Running my program:



We can see the coin represented as a yellow square



Score increments when I collect the yellow box

Save and read score (file handling)

```

258 void readScore()
259 {
260     // creates an array consisting of the contents already inside of the score.txt
261     String[] existingScores = loadStrings("scores.txt");
262
263
264     textAlign(LEFT, TOP);
265     textSize(20);
266     text("Scores:", 20, 20);
267
268     for (int i = 0; i < existingScores.length; i++) {
269         text(existingScores[i], 20, 40 + i * 30); // Display each score with spacing
270     }
271 }

```

Creates an array consisting of the contents inside my score.txt file. Appends the new score to the array. Then saves this new array into the score.txt adding the new value.

```

216 boolean scoreSaved = false;
217

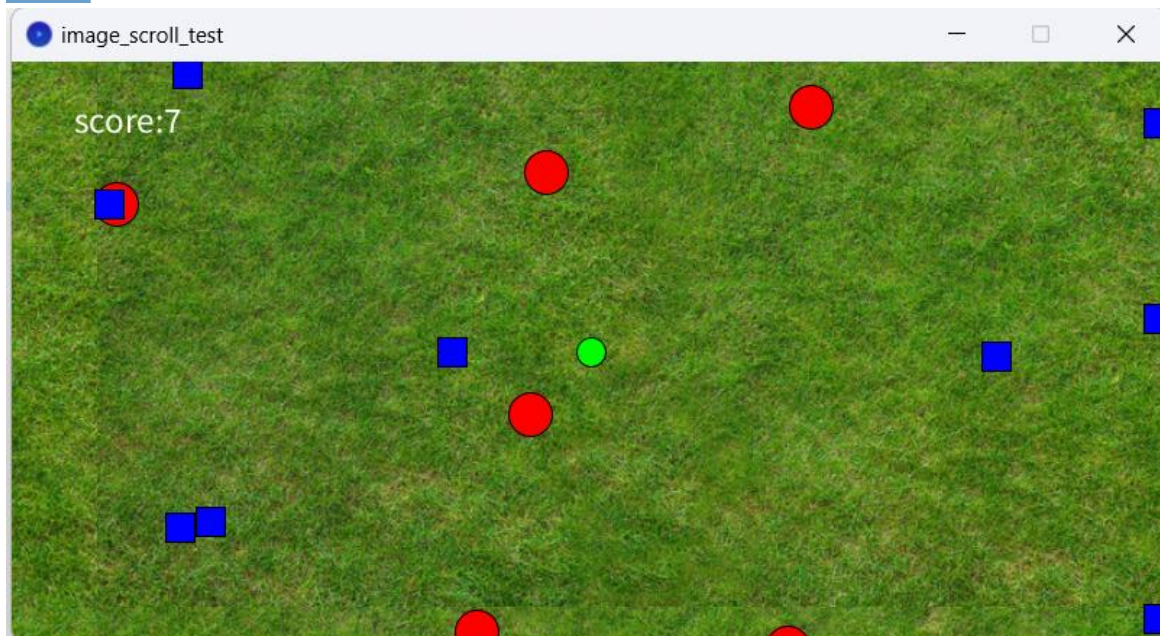
```

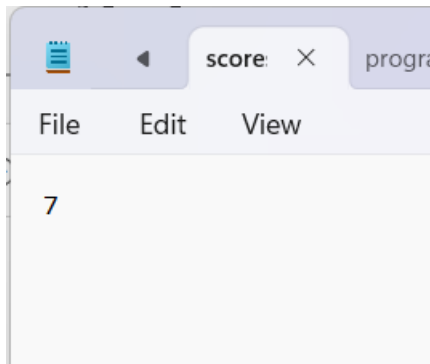
This boolean function will allow it so that the score is only saved once

```

237 if (!scoreSaved) // avoids the score repeatedly saving
238 {
239     saveScore(); // calls save score function
240     scoreSaved = true;
241     println("score saved");
242 }

```





score correctly saves in my score.txt file

When the player gets a game over i coded it so that the current scores are displayed.

```

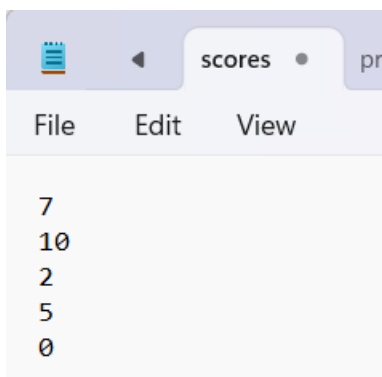
258 void readScore()
259 {
260     // creates an array consisting of the contents already inside of the score.txt
261     String[] existingScores = loadStrings("scores.txt");
262
263     textAlign(LEFT, TOP);
264     textSize(20);
265     text("Scores:", 20, 20);
266
267     for (int i = 0; i < existingScores.length; i++) {
268         text(existingScores[i], 20, 40 + i * 30); // Display each score from the array
269     }
270 }
271

```

This function converts the contents in score.txt into an array then it iterates through each value placing each of them underneath each other.

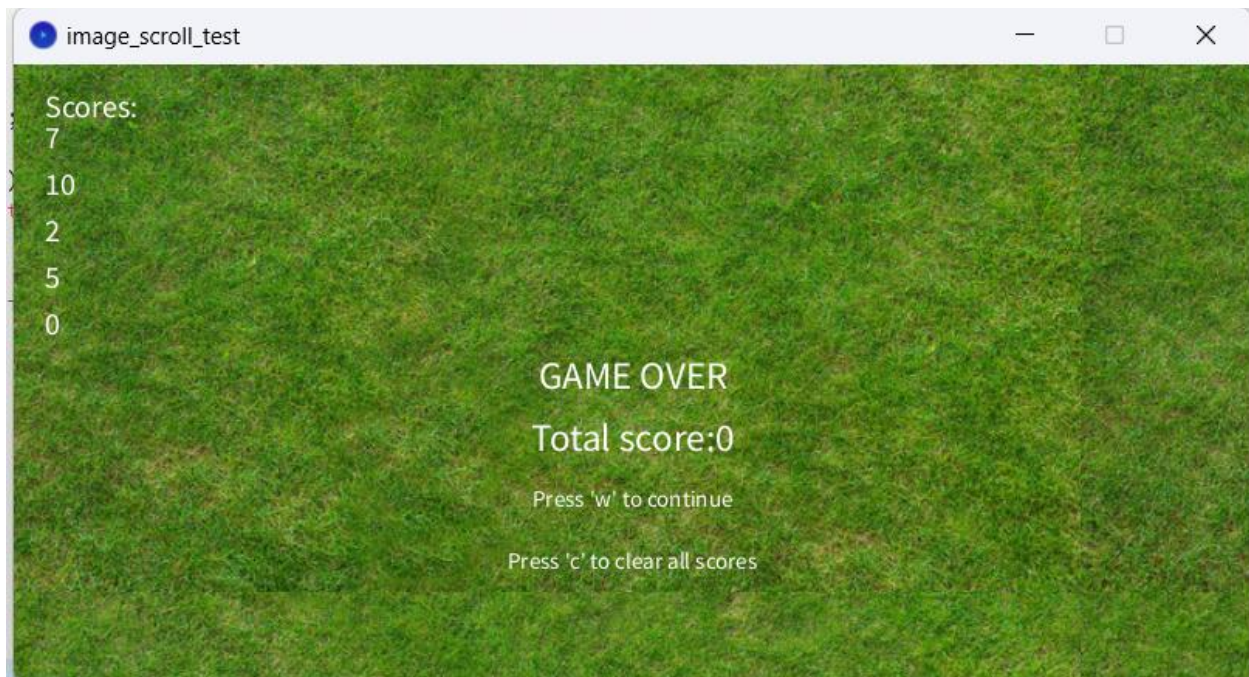
This function is called when the player gets a game over

Testing this:



This is what's in my score.txt file

This is what appears when the player gets a game over



I also added a key bind to clear memory

```
209     if (key == 'c') {  
210         saveStrings("scores.txt", new String[] {}); // Clear the file by overwriting it with an empty array  
211         println("Score file cleared.");  
212     }
```

Version 3

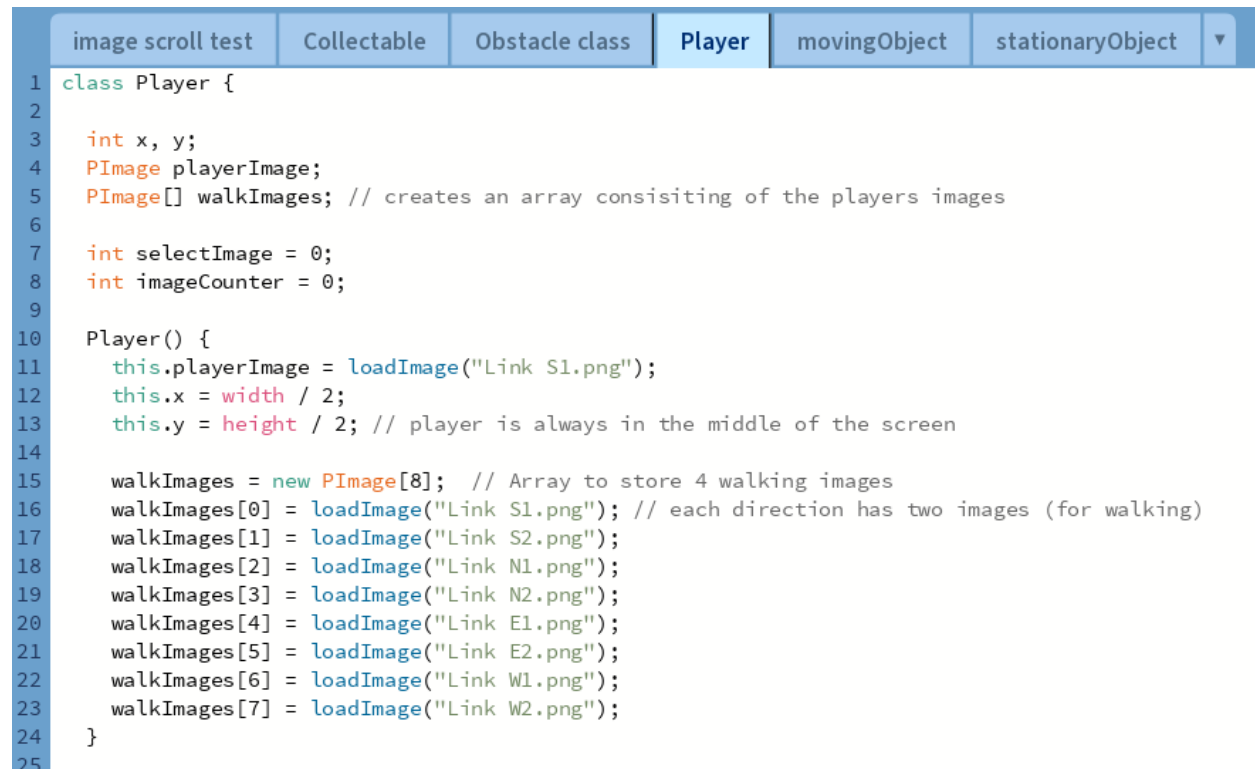
Features to add:

- Images for player
- Animations
- Add player class to put animations inside it

Adding a Player class and player animations

As I am going to be giving my player animations, I found it appropriate to make a player class.

To replicate animation I will be using images.



```

1  class Player {
2
3      int x, y;
4      PImage playerImage;
5      PImage[] walkImages; // creates an array consisting of the players images
6
7      int selectImage = 0;
8      int imageCounter = 0;
9
10     Player() {
11         this.playerImage = loadImage("Link S1.png");
12         this.x = width / 2;
13         this.y = height / 2; // player is always in the middle of the screen
14
15         walkImages = new PImage[8]; // Array to store 4 walking images
16         walkImages[0] = loadImage("Link S1.png"); // each direction has two images (for walking)
17         walkImages[1] = loadImage("Link S2.png");
18         walkImages[2] = loadImage("Link N1.png");
19         walkImages[3] = loadImage("Link N2.png");
20         walkImages[4] = loadImage("Link E1.png");
21         walkImages[5] = loadImage("Link E2.png");
22         walkImages[6] = loadImage("Link W1.png");
23         walkImages[7] = loadImage("Link W2.png");
24     }
25

```

walkImages is an array consisting of all the images ill use for the player. These are the images.



in my void display method, the image displayed depends on a variable called select Image. Using match case and different index of the walkImages array we can display the correct image when a key arrow is pressed

```

26 void display() {
27     switch(selectImage) { // selectImage is determined on the current arrow key pressed
28     case 0:
29         playerImage = walkImages[0]; // different images are displayed for NSEW
30         playerImage.resize(55, 55);
31         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
32         break;
33
34     case 1:
35         playerImage = walkImages[2];
36         playerImage.resize(55, 55);
37         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
38         break;
39

```

Back in main:

```

131     if (movingDown) {
132         bgY -= 3; // Move smoothly down
133         coin.y -=3;
134         player1.selectImage = 0;

```

If movingDown is true (which occurs when the down arrow is pressed) the selectImage attribute becomes 0 which means the program will follow case 0 and display “Link S1”.

This code is repeated for the rest of the directions.

When the player moves, I coded it so that two images alternate over a period of time to emulate a walking animation (hence why I have 8 images imported, 2 images per direction).

To do this I made a walkAnimation method in my player class

```

54 void walkAnimation()
55 {
56     switch(selectImage) { // selectImage is determined on the current arrow key pressed
57     case 0:
58         if (imageCounter<20) // lower half means display first images, upper half means display second image (simulates animation)
59         {
60             playerImage = walkImages[0];
61             playerImage.resize(55, 55);
62             image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
63         } else {
64             playerImage = walkImages[1];
65             playerImage.resize(55, 55);
66             image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
67         }
68         break;

```

This follows a similar structure to the display method as it determines direction based on selectImage. However, this time we have an imageCounter that increments everytime the function is called. Based on whether the imageCounter is above or below 20 a different image is displayed. This essentially alternates between the two images.

```

110     imageCounter++;
111     if (imageCounter>40) {
112         imageCounter =0;
113     }

```

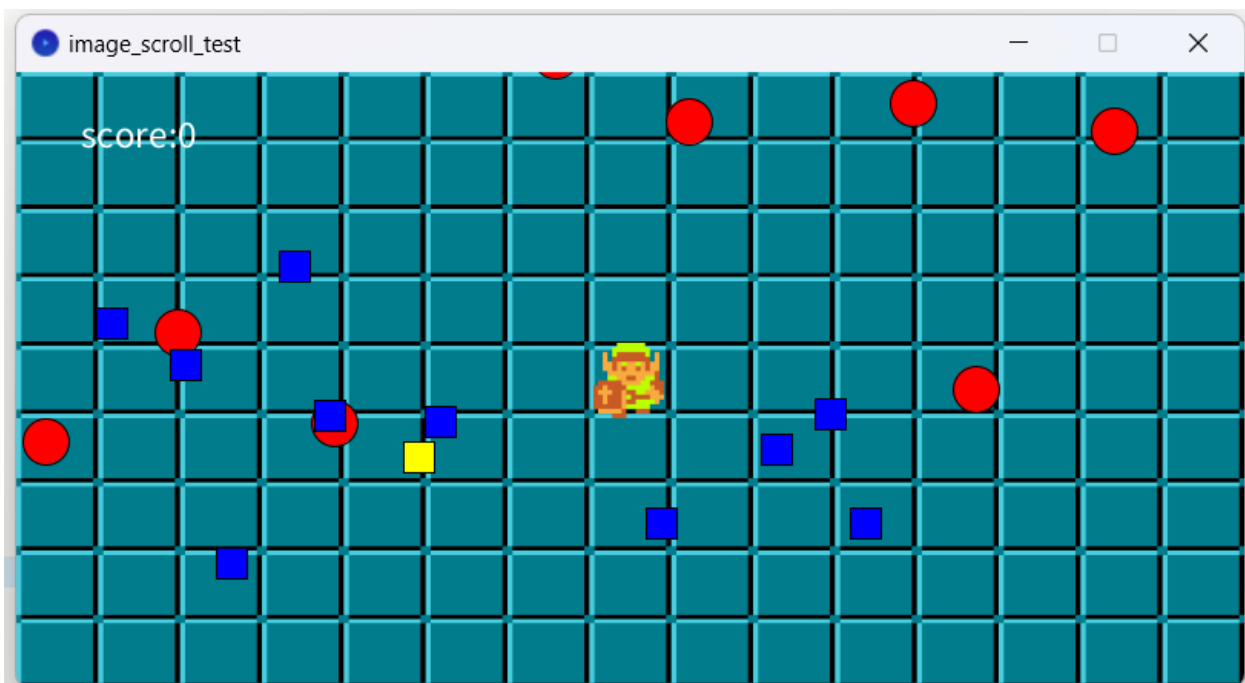
Another difference between this method and the display method is that this method is only called when an arrow key is held down, as it wouldn't make sense to animate the character if they are not moving.


```
131 .....  
132 if (movingDown) {  
133     bgY -= 3; // Move smoothly down  
134     coin.y -=3;  
135     player1.selectImage = 0;  
136     player1.walkAnimation(); // walk animation only plays while movingDown is true (or while down arrow is held)
```

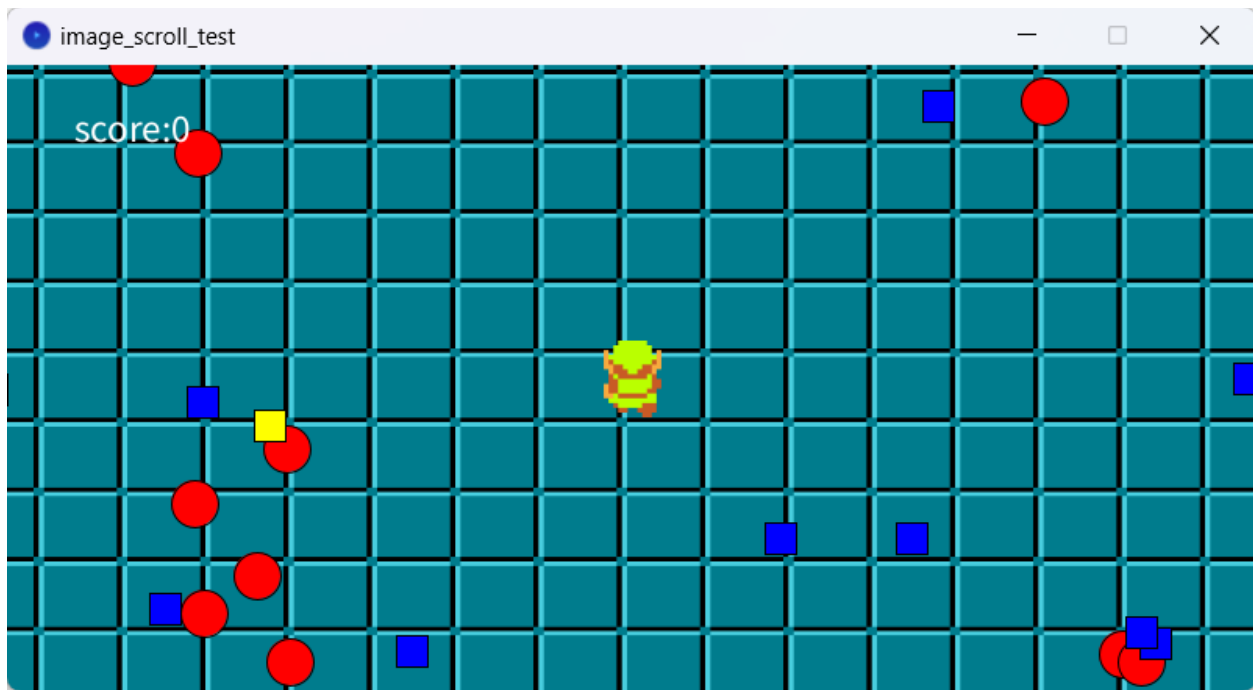
This ensures that the animation only plays when the character is moving

This code is also pasted for the other directions.

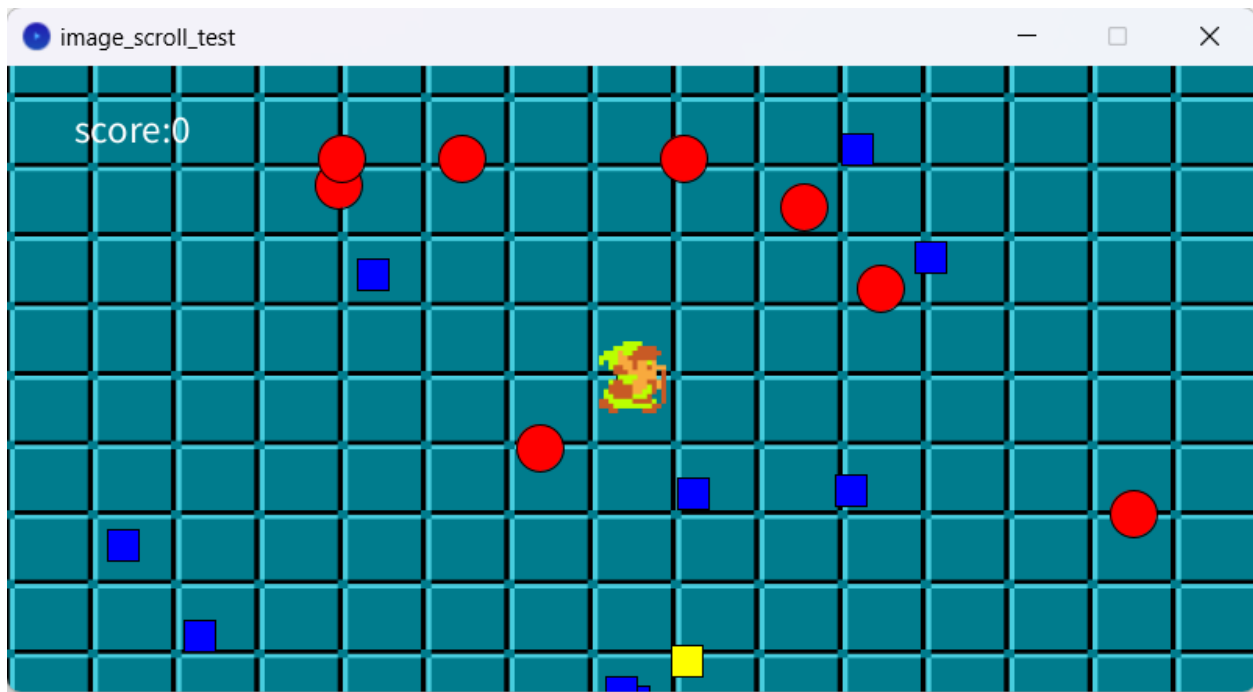
Testing this: *note I changed the background as well
holding down



Holding up



Holding right



Holding Left



Animations for obstacles and collectables

I changed my collectable objects display method

```

1 class Collectable {
2
3     int x, y;
4     color col;
5     PImage collectableImage1;
6     PImage collectableImage2;
7
8     int imageCounter;
9
10    Collectable(color col) {
11        this.col = col;
12
13        this.x = int(random(width)); // spawns in a random location
14        this.y = int(random(height));
15
16        collectableImage1 = loadImage("Rupee 1.png");
17        collectableImage2 = loadImage("Rupee 2.png");
18    }
19

```

```

42 void display() {
43     if (imageCounter < 10) // alternates between the two images
44     {
45         collectableImage1.resize(50, 50);
46         image(collectableImage1, x-collectableImage1.width/2, y-collectableImage1.height/2);
47     } else {
48         collectableImage2.resize(50, 50);
49         image(collectableImage2, x-collectableImage2.width/2, y-collectableImage2.height/2);
50     }
51     imageCounter++; //increment
52     if (imageCounter > 20) { // reset when it exceeds 20
53         imageCounter = 0;
54     }
55
56     if (x < -25 || x > width + 25 || y < -25 || y > height + 25) {
57         respawn();
58     }
59 }
60

```

This display method alternates between two images. When image counter is less than 10 is displays image1, when image counter is more than 10 then it displays image 2.

This simulates a short animation for our collectable object. The images ill be using is this:



✓ Rupee 1



✓ Rupee 2

Both my enemy objects will follow the exact same logic and code just with a different image.

stationaryObject images:



✓ Enemy 1



✓ Enemy 2

movingObject images:



Testing this:



Death animation

The only animation left to make is a death animation.

I chose to add a deathAnimation method in my player class. I wanted my death animation to involve the screen turning gray, the player character will spin and disappear. Graying out the screen and getting rid of the enemies was essential for effectively communicating a game over.

```

16  walkImages = new PImage[10]; // Array to store 8 walking images + 2 death images
17  walkImages[0] = loadImage("Link S1.png"); // each direction has two images (for walking)
18  walkImages[1] = loadImage("Link S2.png");
19  walkImages[2] = loadImage("Link N1.png");
20  walkImages[3] = loadImage("Link N2.png");
21  walkImages[4] = loadImage("Link E1.png");
22  walkImages[5] = loadImage("Link E2 (1).png");
23  walkImages[6] = loadImage("Link W1.png");
24  walkImages[7] = loadImage("Link W2 (1).png");
25
26  walkImages[8] = loadImage("Death 1.png"); // these are not used for walking
27  walkImages[9] = loadImage("Death 2.png"); // these will be used for the death animation
28  }
29

```

I first added two new images for the characters death animation.



✓ Death 1



✓ Death 2

These two images will play consecutively to replicate the character poofing / disappearing

```

119 void deathAnimation()
120 {
121     if (imageCounter < 15) // lower half means display first images, upper half means display second image (simulates animation)
122     {
123         // character spins
124         playerImage = walkImages[0];
125         playerImage.resize(55, 55);
126         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
127         filter(GRAY);
128     } else if (imageCounter < 30) {
129         playerImage = walkImages[6];
130         playerImage.resize(55, 55);
131         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
132         filter(GRAY);
133     } else if (imageCounter < 45) {
134         playerImage = walkImages[2];
135         playerImage.resize(55, 55);
136         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
137         filter(GRAY);
138     } else if (imageCounter < 60) {
139         playerImage = walkImages[4];
140         playerImage.resize(55, 55);
141         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
142         filter(GRAY);
143     } else if (imageCounter < 75) {
144         playerImage = walkImages[0];
145         playerImage.resize(55, 55);
146         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
147         filter(GRAY);
148     }
149 }

```

The animation uses the same imageCounter logic, where different images are displayed when image counter is above or below a certain number.

The code above makes use of our characters NSEW images. Notice how they are filtered gray.

```

149     // character fades away
150     } else if (imageCounter < 90) {
151         playerImage = walkImages[8];
152         playerImage.resize(55, 55);
153         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
154         filter(GRAY);
155     } else if (imageCounter < 105) {
156         playerImage = walkImages[9];
157         playerImage.resize(55, 55);
158         image(playerImage, x - playerImage.width / 2, y - playerImage.height / 2);
159         filter(GRAY);
160     }
161
162
163     imageCounter++;
164     if (imageCounter > 150) {
165         imageCounter = 0;
166
167         deathFinished = true;
168     }
169

```

These two else ifs display the disappearing images.

Back in main the game over screen only displays once my deathFinished Boolean becomes true. Once the death animation finishes playing this variable becomes true.

```

16     boolean deathFinished = false;
17

```

By default, this value is false

Back in main:

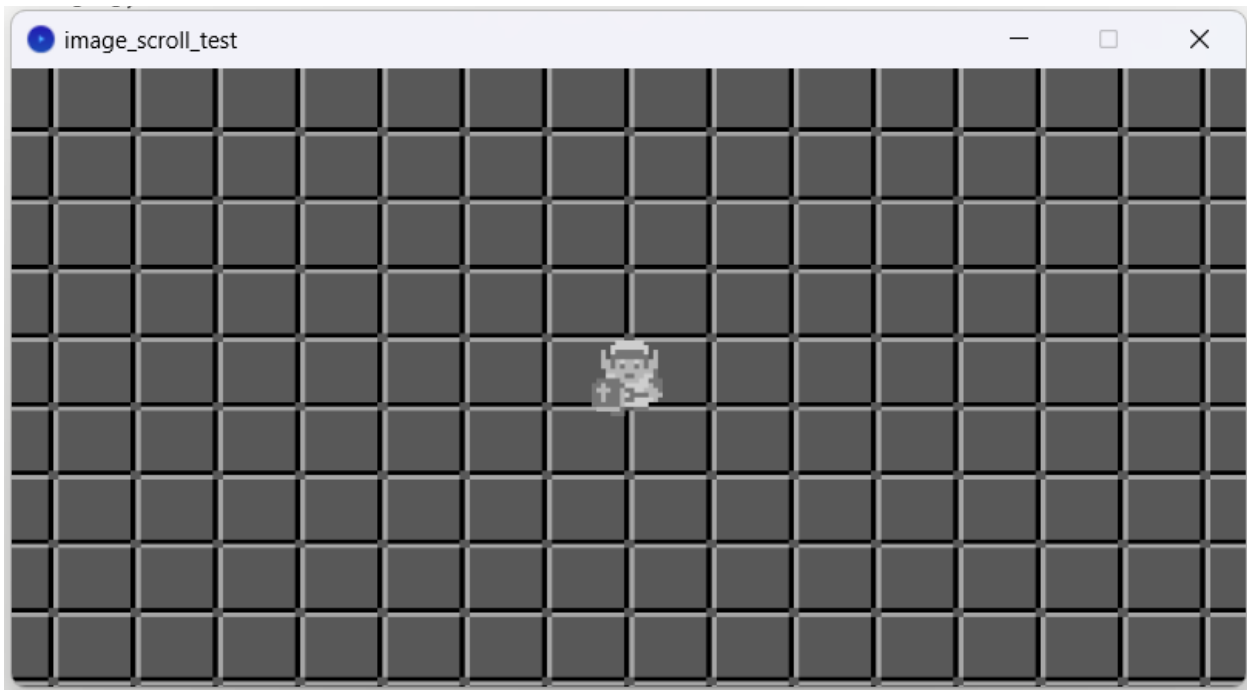
This if statement only occurs if gameMode is not 0 (i.e. when the player collides with an obstacles)

```

191     if (!deathFinished) {
192         player1.deathAnimation(); // Play death animation
193     } else {
194         displayGameOver(); // Show game over screen once death animation finishes
195     }
196 }

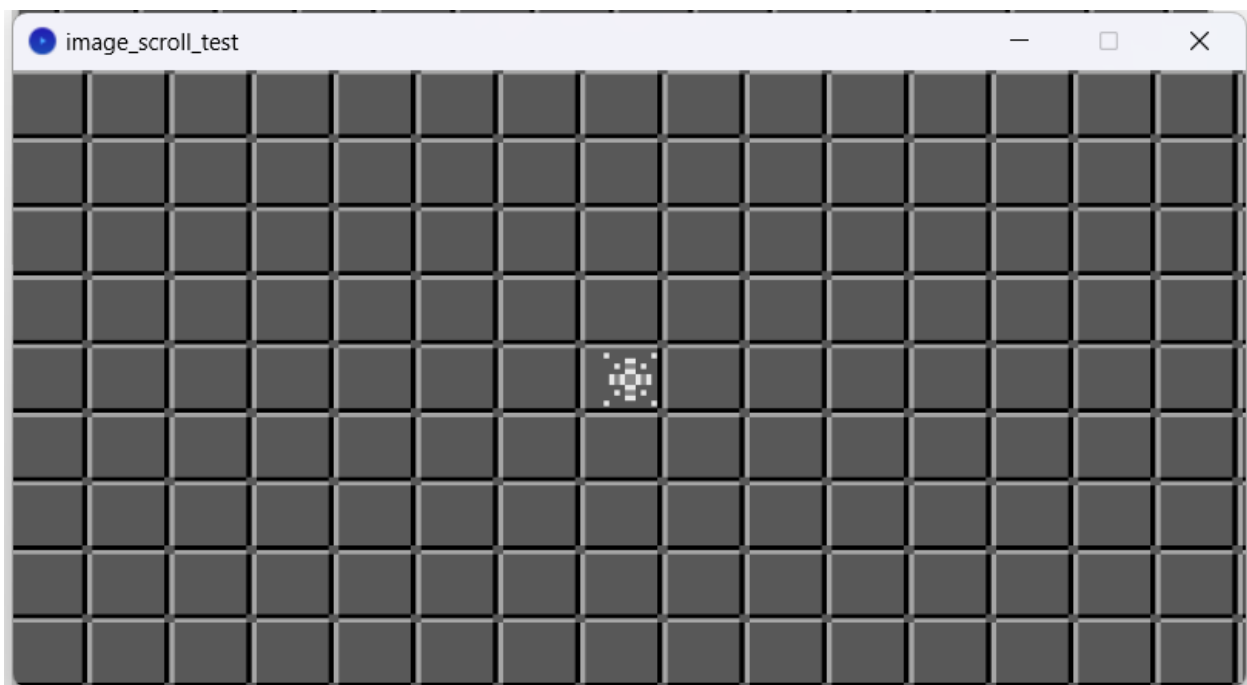
```

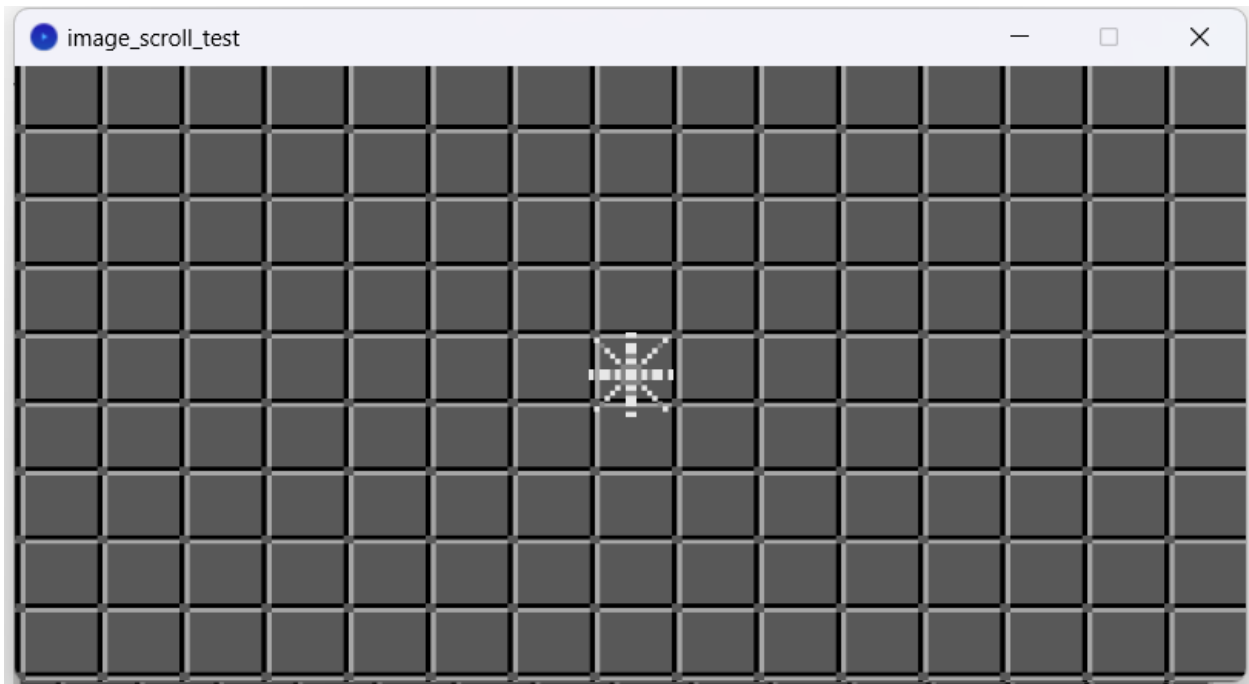
Testing this:



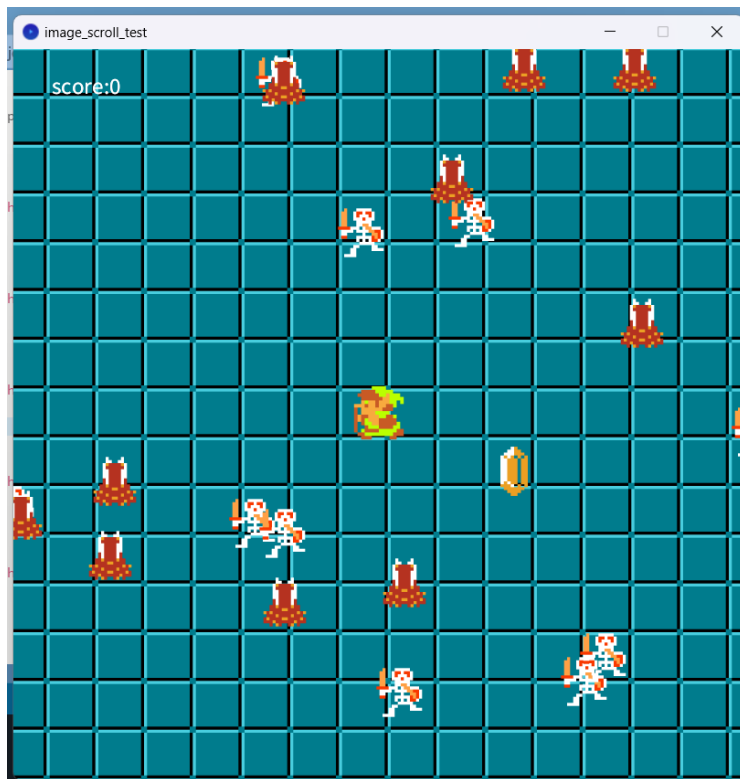
screen graying out and all objects being destroyed after the player is hit.

Character disappearing animation:





My final change is to make the window just a bit bigger making the game a bit easier to play:



This concludes my 2d game development report. Thank you for reading.