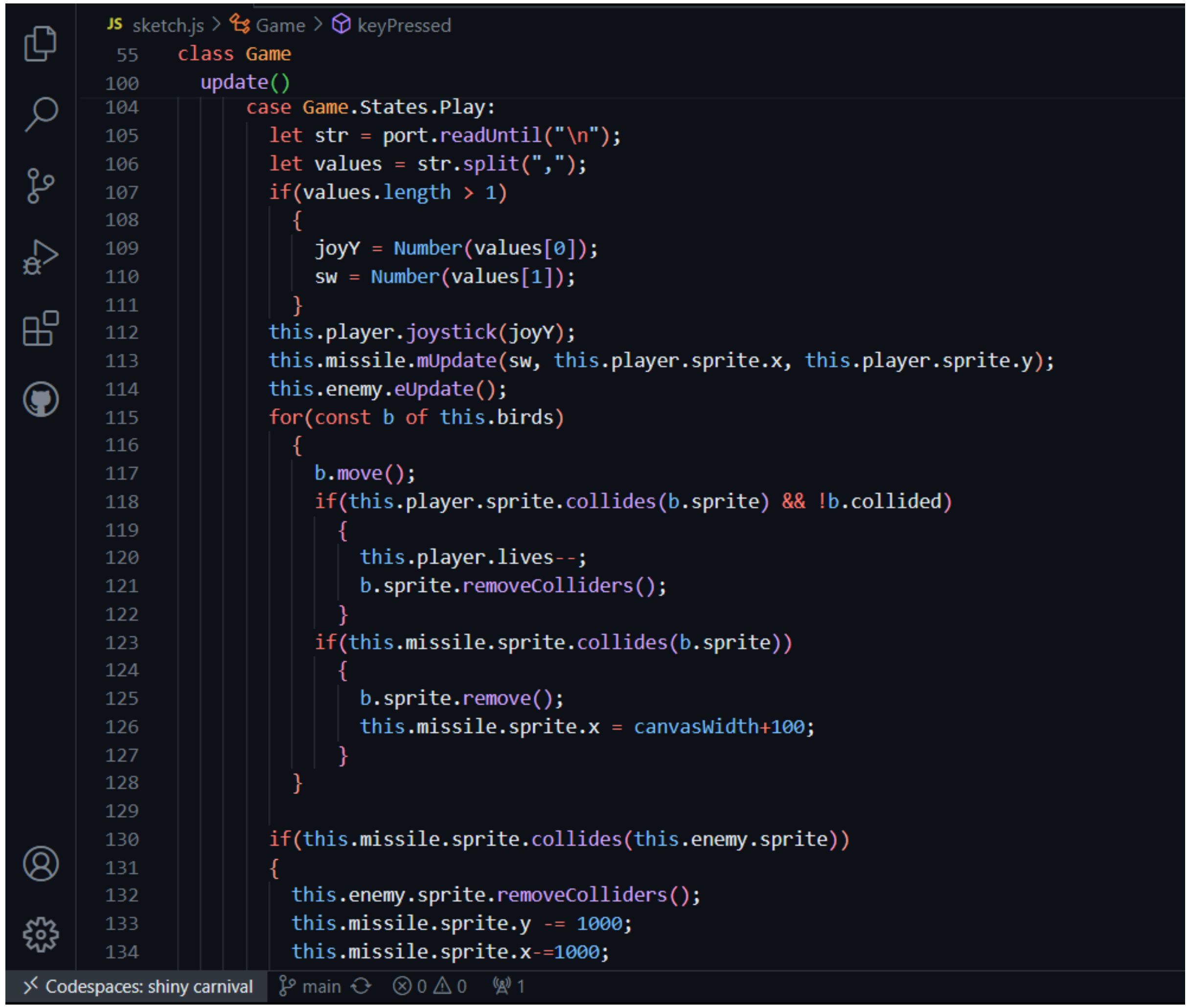
Plane Pursuit

- "Plane Pursuit" is a game where a player engages in a wild pursuit of a combative smuggler plane. The player will have to maneuver through dangerous obstacles like flocks of birds to catch the enemy. Being hit by an obstacle decreases the integrity of the ship. Only once the player successfully negotiates these obstacles will he/she have the perfect shot. If the player can land three shots on the criminal's vehicle, victory is assured.

Game update method

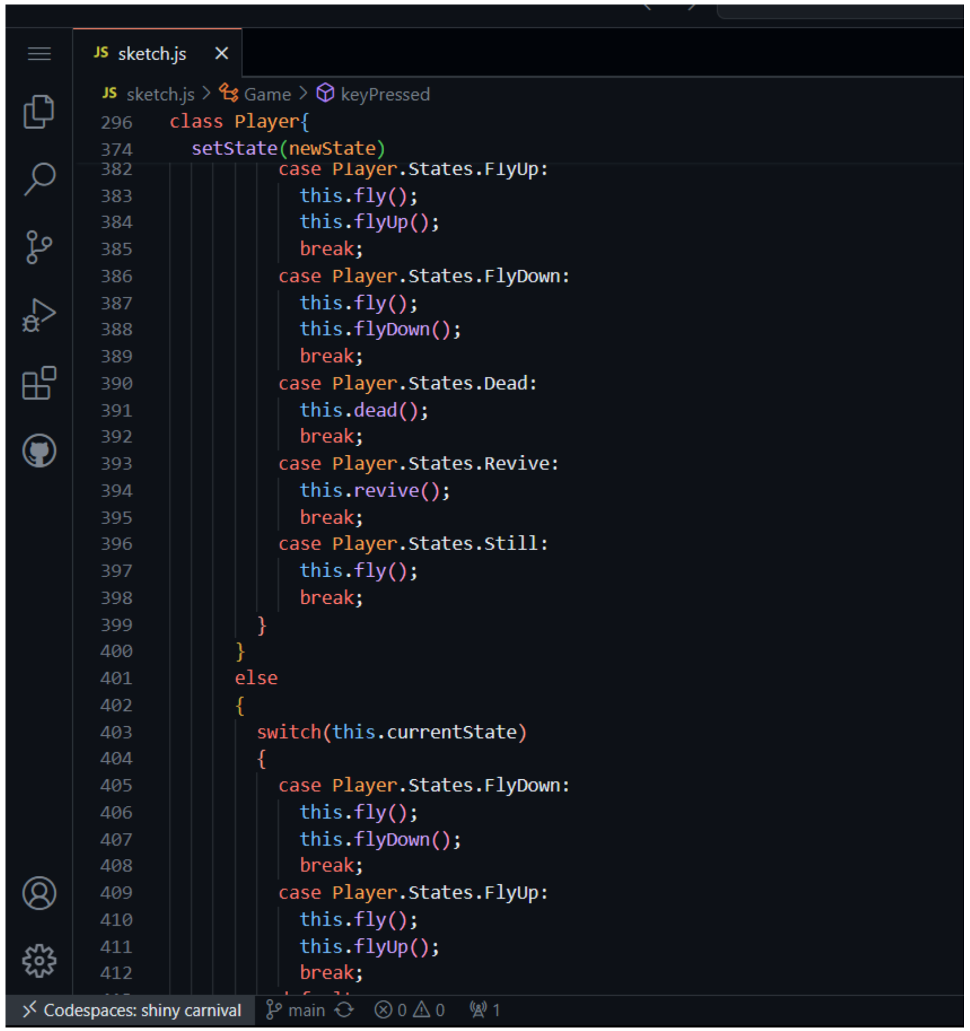
This method is what allows the game to change during play and is where the most important logic lies. Given are expressions that determine whether objects have collided, spawn obstacles, and change individual objects' states.



```
class Game
 update()
   case Game.States.Play:
    let str = port.readUntil("\n");
    let values = str.split(",");
    if(values.length > 1)
     {
      joyY = Number(values[0]);
      sw = Number(values[1]);
     }
    this.player.joystick(joyY);
    this.missile.mUpdate(sw, this.player.sprite.x, this.player.sprite.y);
    this.enemy.eUpdate();
    for(const b of this.birds)
     {
      b.move();
      if(this.player.sprite.collides(b.sprite) && !b.collided)
       {
        this.player.lives--;
        b.sprite.removeColliders();
       }
      if(this.missile.sprite.collides(b.sprite))
       {
        b.sprite.remove();
        this.missile.sprite.x = canvasWidth+100;
       }
     }

    if(this.missile.sprite.collides(this.enemy.sprite))
    {
     this.enemy.sprite.removeColliders();
     this.missile.sprite.y -= 1000;
     this.missile.sprite.x-=1000;
```

Player setState method

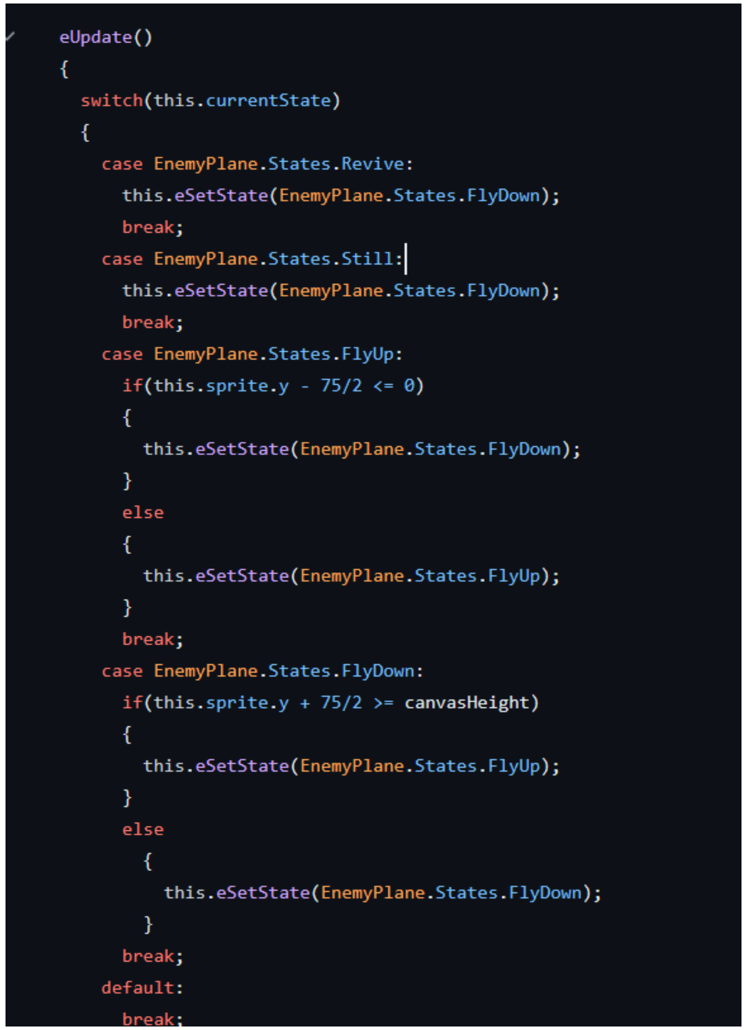
This method is the main meat that allows a finite state machine to work. It changes the state of the player object depending on certain conditions satisfied during gameplay and adjusts the player's sprite and attributes accordingly.



```
class Player{
 setState(newState)
     case Player.States.FlyUp:
      this.fly();
      this.flyUp();
      break;
     case Player.States.FlyDown:
      this.fly();
      this.flyDown();
      break;
     case Player.States.Dead:
      this.dead();
      break;
     case Player.States.Revive:
      this.revive();
      break;
     case Player.States.Still:
      this.fly();
      break;
    }
   }
   else
   {
    switch(this.currentState)
    {
     case Player.States.FlyDown:
      this.fly();
      this.flyDown();
      break;
     case Player.States.FlyUp:
      this.fly();
      this.flyUp();
      break;
```

EnemyPlane Class

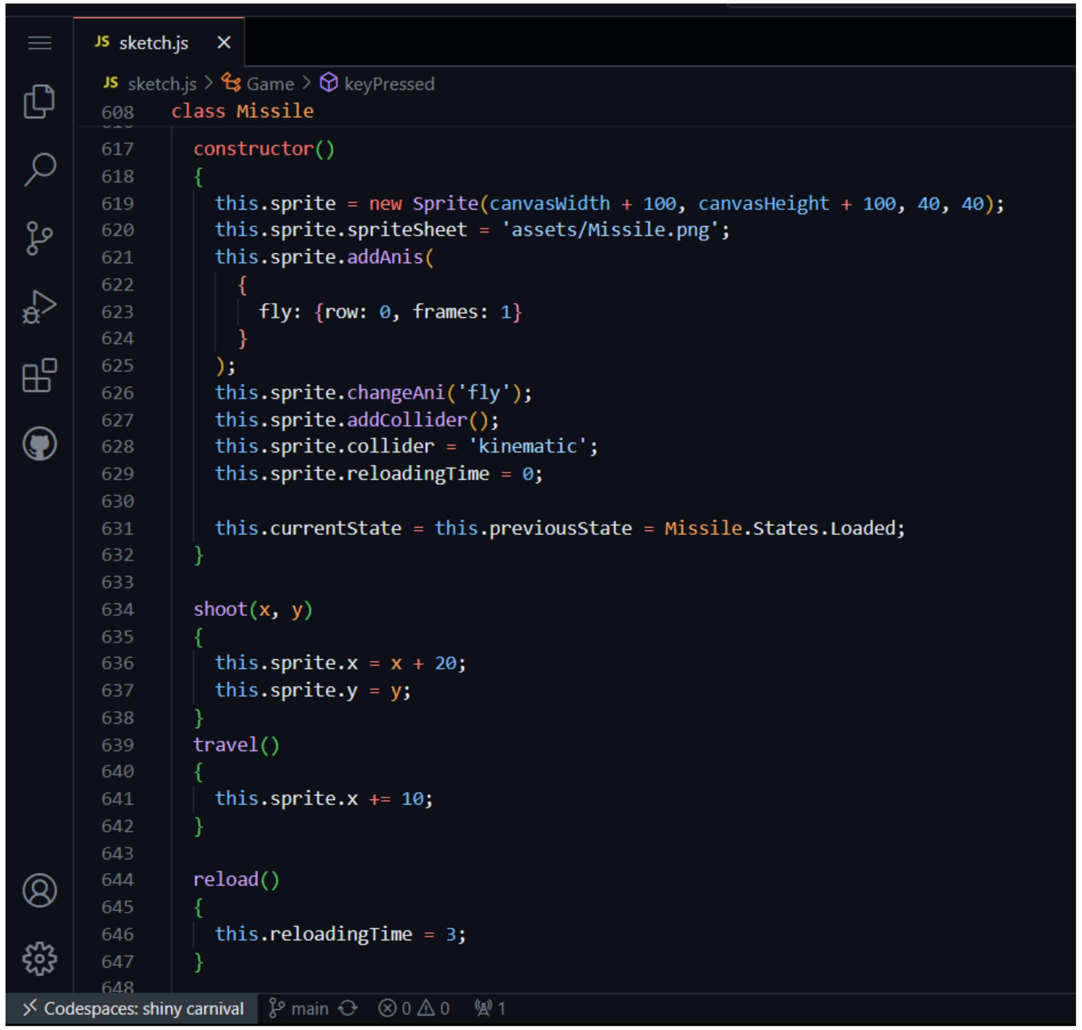
Most of the methods and attributes of the EnemyPlane class are derived from that of the Player. However, they allow for automatic movement.



```
eUpdate()
{
  switch(this.currentState)
  {
    case EnemyPlane.States.Revive:
      this.eSetState(EnemyPlane.States.FlyDown);
      break;
    case EnemyPlane.States.Still:
      this.eSetState(EnemyPlane.States.FlyDown);
      break;
    case EnemyPlane.States.FlyUp:
      if(this.sprite.y - 75/2 <= 0)
      {
        this.eSetState(EnemyPlane.States.FlyDown);
      }
      else
      {
        this.eSetState(EnemyPlane.States.FlyUp);
      }
      break;
    case EnemyPlane.States.FlyDown:
      if(this.sprite.y + 75/2 >= canvasHeight)
      {
        this.eSetState(EnemyPlane.States.FlyUp);
      }
      else
        {
          this.eSetState(EnemyPlane.States.FlyDown);
        }
      break;
    default:
      break;
```

Missile class

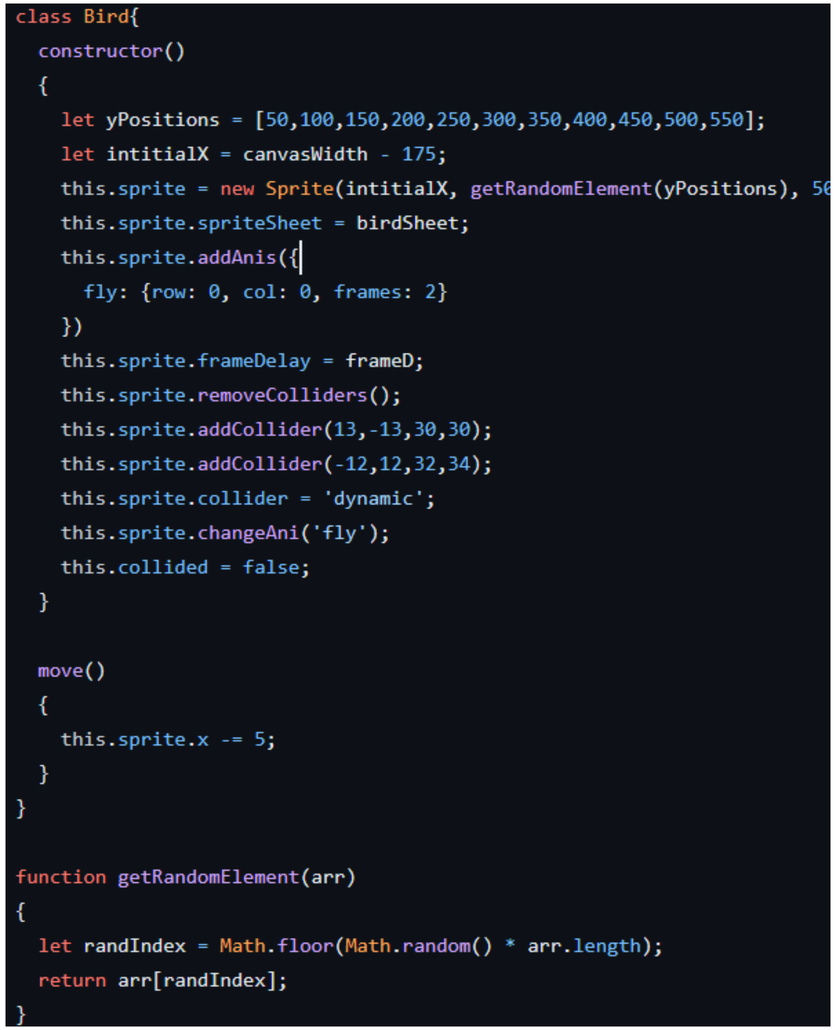
This is a custom class that allows the player to “shoot” what looks like a missile toward the enemy. The class includes methods and attributes that allow a player to reload their weapon. The missile sprite has colliders that detect whether it touches another sprite with colliders.



```
class Missile
  constructor()
  {
    this.sprite = new Sprite(canvasWidth + 100, canvasHeight + 100, 40, 40);
    this.sprite.spriteSheet = 'assets/Missile.png';
    this.sprite.addAnis(
      {
        fly: {row: 0, frames: 1}
      }
    );
    this.sprite.changeAni('fly');
    this.sprite.addCollider();
    this.sprite.collider = 'kinematic';
    this.sprite.reloadingTime = 0;

    this.currentState = this.previousState = Missile.States.Loaded;
  }

  shoot(x, y)
  {
    this.sprite.x = x + 20;
    this.sprite.y = y;
  }
  travel()
  {
    this.sprite.x += 10;
  }

  reload()
  {
    this.reloadingTime = 3;
  }
```

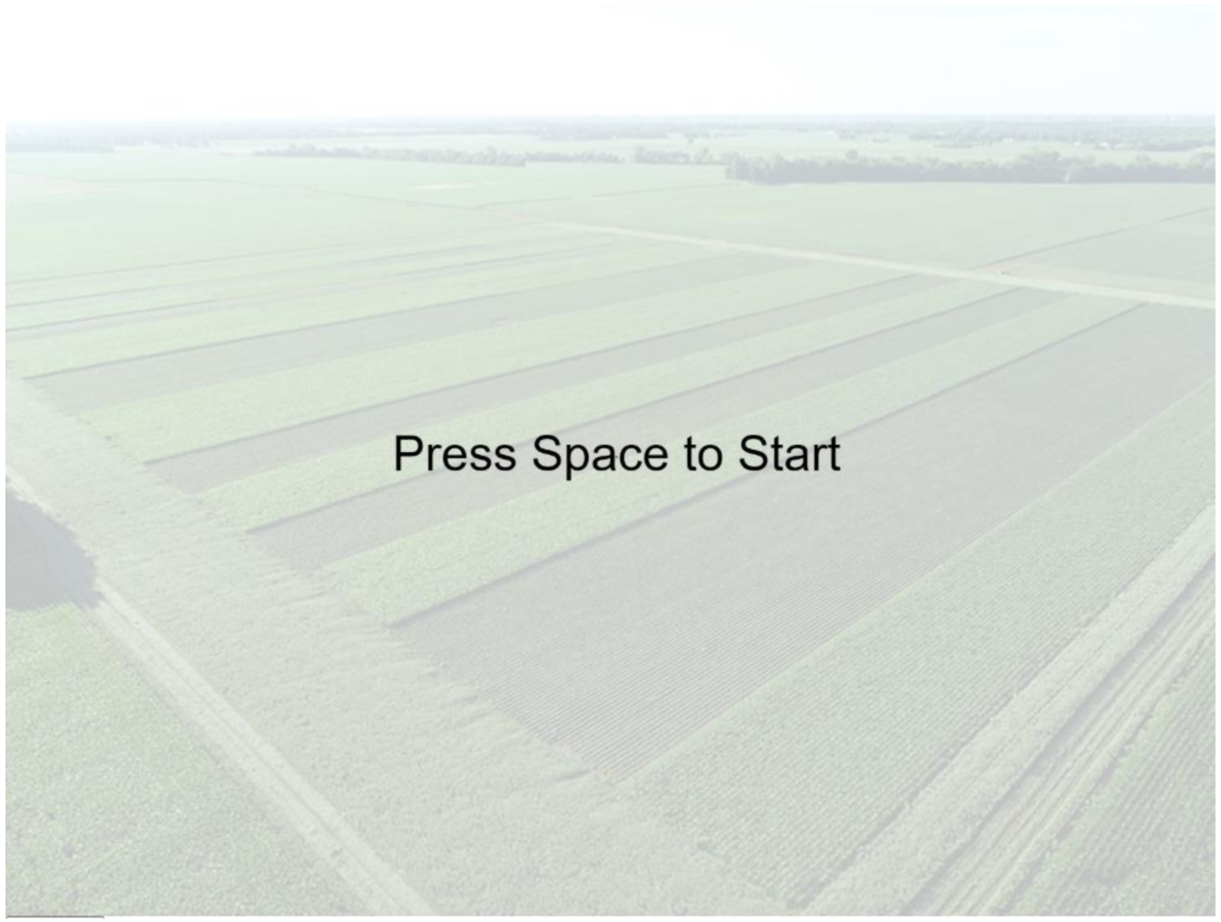
Bird Class

This is a simple class used to create bird obstacles during gameplay.

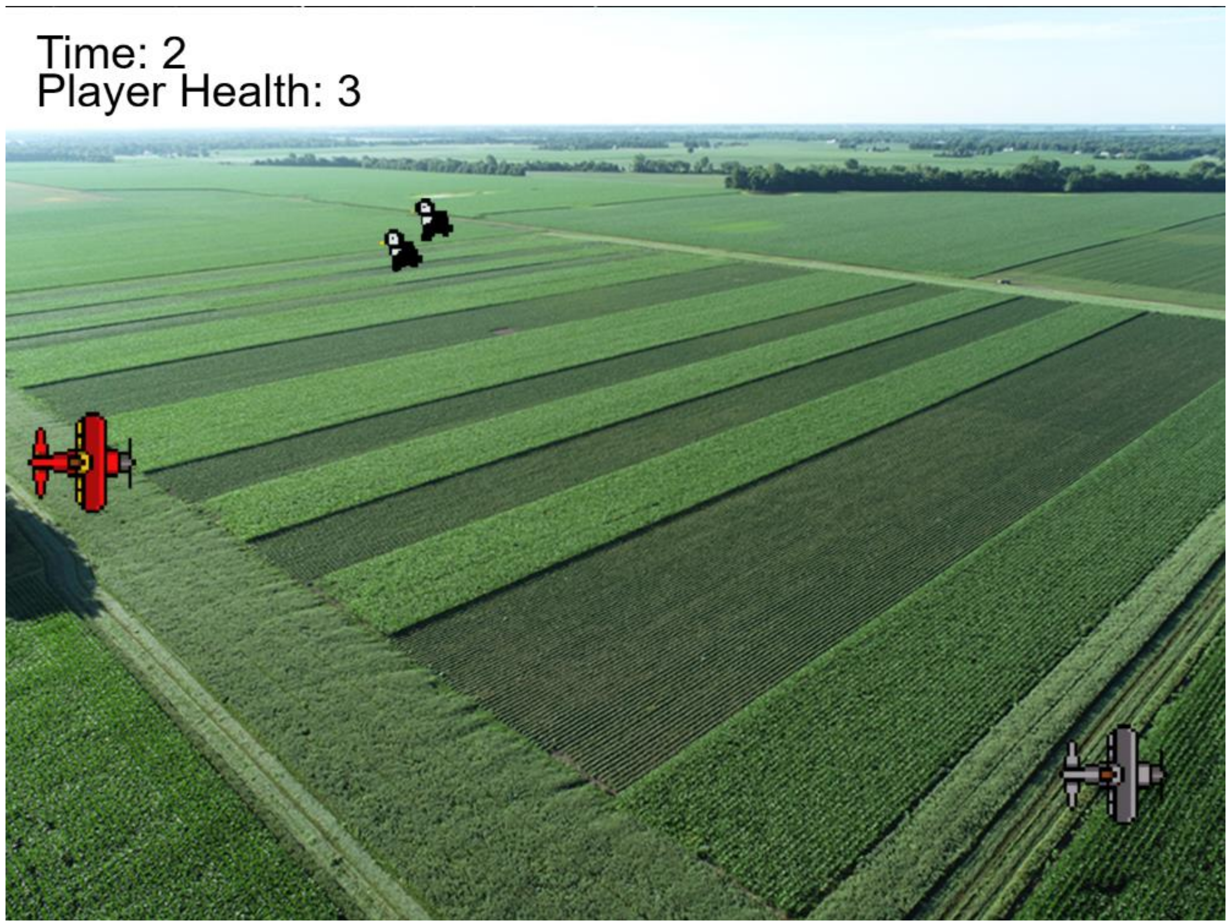


```
class Bird{
  constructor()
  {
    let yPositions = [50,100,150,200,250,300,350,400,450,500,550];
    let intitialX = canvasWidth - 175;
    this.sprite = new Sprite(intitialX, getRandomElement(yPositions), 50
    this.sprite.spriteSheet = birdSheet;
    this.sprite.addAnis({
      fly: {row: 0, col: 0, frames: 2}
    })
    this.sprite.frameDelay = frameD;
    this.sprite.removeColliders();
    this.sprite.addCollider(13,-13,30,30);
    this.sprite.addCollider(-12,12,32,34);
    this.sprite.collider = 'dynamic';
    this.sprite.changeAni('fly');
    this.collided = false;
  }

  move()
  {
    this.sprite.x -= 5;
  }
}

function getRandomElement(arr)
{
  let randIndex = Math.floor(Math.random() * arr.length);
  return arr[randIndex];
}
```

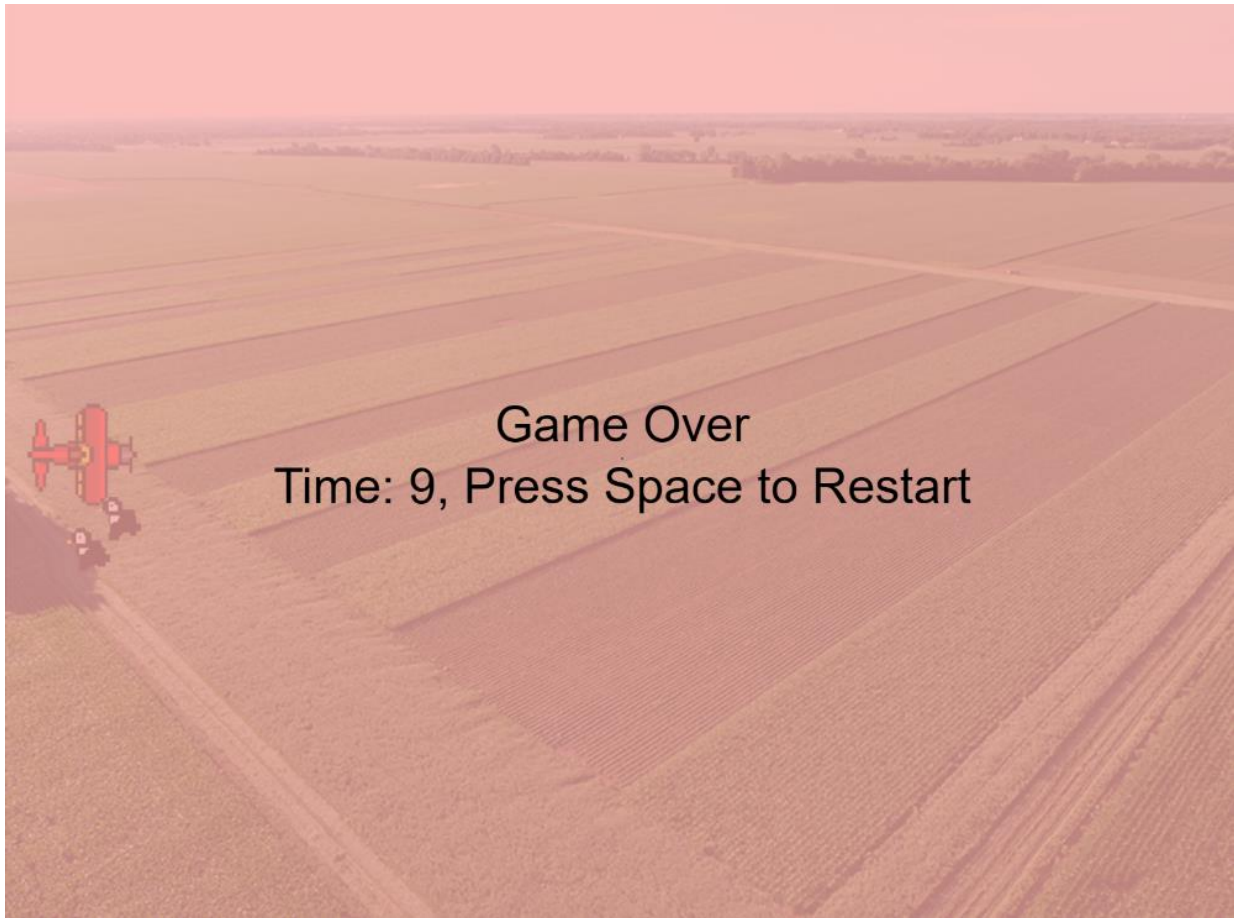
Game Start Screen


Press Space to Start

Game play



Game over



Game Over
Time: 9, Press Space to Restart

Problems

- I could never get that Tone.js library to work. I did everything as I did in previous assignments, but Tone.js always gave me problems and would never work all the time. I scoured over the internet to find solutions to my problems; however, I could find no solutions. My personal opinions on the library itself may have also dissuaded me from doing what I could have.

Future Development

- There is definitely room for improvement in this game. Sound is a must-have in any bestseller. Perhaps I can shorten the code by adding inheritance and polymorphism. Maybe I can add a method that allows the enemy to shoot back.