Space Invaders

Developed by: Space Crusaders

Team Members: Phuong Tran, Noah Markowitz, Yiran Yin, Arthur Savage, Benjamin Hsu

Mentors: Pranet Sharma, Ashley Rabalais

ENG EC327 Fall 2022

# How to Play:

### Controls:

* ‘WASD’ / ‘ARROW KEYS’ - Move Player Sprite
* ‘SPACE KEY’ / LEFT MOUSE BUTTON’ - Shoot Laser

### Win Condition:

* Annihilate all Densmore heads to win the game
* The game will restart if the player sprite collides with missiles or the staff

# Intention

Our intention with this project is to create a very slightly humorous game inspired by the classic arcade game Space Invaders, with its target audience being EC327 students and the potential graders of this project, who are the EC327 faculty members.

Since one of our target audiences is the EC327 faculty members, we have directly asked them for input on art direction for certain objects in our game.

When we asked ourselves, “Who would play our Space Invaders?,” we believe that the students in class ENG EC327 Fall 2022 will most likely enjoy the game. Outside of our class, we believe that very young children would play our game. Toddlers could also be amused by a set of dangling keys. In addition, we believe that since our Space Invaders game is simple and classic, we think that the same audience who enjoys Google hidden games or Google games on search engines can find our game interesting.

# Background

For our project, we decided on the Unity game engine to construct our Space Invaders-like video game.

* Unity is a cross-platform game engine that supports 2D and 3D graphics and scripting through the **C#** programming language. It provides many useful built-in features and is popular among beginner developers.
* More precisely, we used an older **Unity Editor** release version: **2021.3.14f1**. This is because it is the latest version supported by Plastic SCM, the version control system we have chosen to use.
* Plastic SCM is a version control system for Unity that allows us to work on the same program more easily. It is like GitHub but more specialized for Unity projects.

# Challenges

Our biggest challenge was learning how to use Unity and its intricacies. Our group was not familiar with game engines and learning it cost us a lot of time towards development.

Another challenge we encountered is that we initially could not push the source code to GitHub because GitHub cannot do commits on the entirety of a Unity project as the files are too big and too many. Hence, we used Unity **Plastic SCM**, which is a version control and source code management tool, to collaborate.

Unity Plastic SCM only allows 3 people within an organization to collaborate. Alternatives were either outdated or had a similar monetization method. If we want to add more people, we each have to pay for a subscription. We were limited to three developers and, some members did not have access to the source code and must wait for the main developers to upload the script or game content to Google Drive. Because of this, the documentation took a longer time to complete than expected.

However, we were eventually able to resolve how to upload the source code to Github. We did this by using Github’s “.gitignore” template for a Unity project. However, to allow this to work we had to place the source code in a separate github repository.

# Game Development

We started by creating a new project on **Unity Editor** using its built-in 2D template. This template is packaged with useful features for 2D games and uses Unity’s built-in 2D rendering engine.

There are four folders that hold the game's assets:

* Scenes
  + Contains the files that set up all game objects onto a level in the game. Also used to load a menu for navigating the game.
* Prefabs
  + Contains configured game objects that are reused in the game
* Sprites
  + Contains all the .png files that we used to visually represent the game objects that are active in the scene.
* Script
  + Contains the C# files that give all game objects their behavior

## Scenes

The scenes folder contains two scenes:

* Menu - This is the main menu that is displayed which includes 2 buttons:
  + Start: click to start the game
  + Credits: display credits of people who are involved in the game development
* Game - This is where the player plays the game

## Prefabs

The prefabs folder contains all the game objects with specific configurations such as its associated sprite, or its hitbox size. For them to appear, they are loaded with the scene or instantiated by the script.

In here we have:

* 6 SplineObject Prefabs
  + Each one represents a TA ship and is associated with their sprite
  + Has its own hitbox size that is automatically set by Unity depending on the .png size
  + Instantiated by SplineController script
* Player Prefab
  + This is associated with the player sprite.
  + Loaded with the scene
* GroupInvader Prefab
  + This is supposed to represent the Densmore heads
  + Instantiated by Invaders script
* Laser Prefab
  + This represents the laser that the player shoots
  + Its direction is set to +y so it shoots upwards
  + Instantiated by the Projectile script
* Missile Prefab
  + This represents the missile that the Densmore heads shoot
  + Its direction is set to -y
  + Instantiated by Projectile script

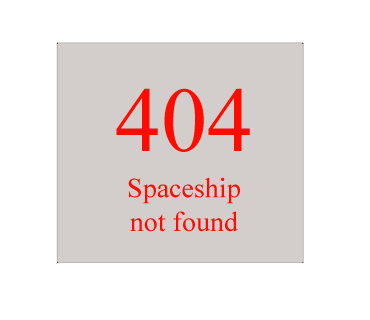
## Sprites

These are the artwork that is used to represent our objects in-game. We emailed Professor Densmore and our Teaching Assistants to ask for permission to create sprites customized for each person. We also asked each person how they want their spaceships to look. Here are the spaceships that we created based on their preferences:

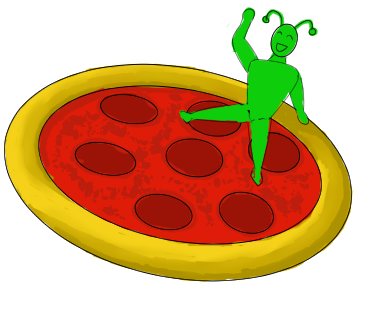
Densmore head



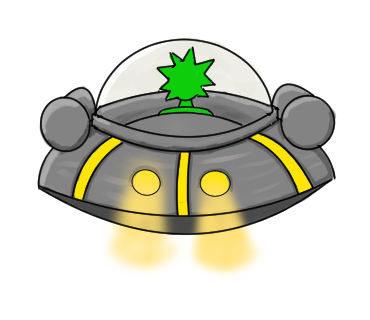
Efe Sencan’s sprite



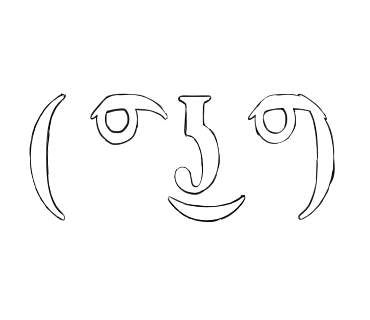
Farbin Fayza’s sprite



Manuel Segimon’s sprite



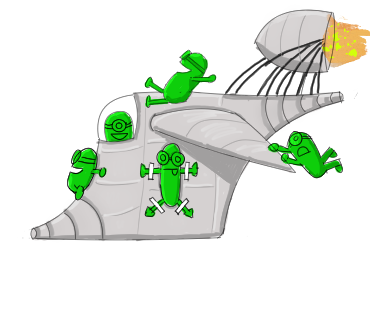
Shamir Legaspi’s sprite



Pranet Sharma’s sprite



Rishav De’s sprite



Player’s sprite



## Classes

Our game is entirely made from class objects. Within the script folder, the game consists of five classes with their own methods and attributes:

* Player - represents the player sprite
* Invader - represents an individual Invader
* Invaders - represents all Invaders
* SplineController - controls the spline movement of Invaders
* Projectile - represents the invader's missiles and player’s laser

### Player Class

*The Player class contains the script for the commands and functions of the player sprite.*

The Player class adds the following behaviors to the player sprite:

* The user can move the player sprite left and right by pressing the ‘A’ and ‘D’ keys, respectively
* The user can have the player sprite shoot a laser by pressing ‘SPACE’ or ‘Left Mouse Button’

#### Public Members

* Projectile laserPrefab
  + A laser sprite prefab with the projectile script attached to it
* float speed
  + Speed of the playerInitial value is 5.0f

#### Private Members

* bool \_laserActive
  + Boolean that determines if there is a laser active in the game
* void Update()
  + Checks for key presses from a user that would control the player's sprite. Pressing ‘A’ or ‘D’ moves the sprite left or right at the set speed. Pressing ‘SPACE’ or ‘LEFT MOUSE BUTTON’ instantiates a laser prefab.
* void Shoot()
  + If a laser is not active, !\_laserActive, allows the player to instantiate a laser prefab. When destroyedis invoked from the Projectile class, call LaserDestroyed()
* void LaserDestroyed()
  + Destroys the laserSets \_laserActive to false
* void OnTriggerEnter2D(Collider2D other)
  + If an Invader or a missile collides with the player sprite, restart the scene

## Invader, Invaders, & SplineController Classes

*The Invader sprite’s behavior is contained in three classes: Invader, Invaders, and SplineController. The Invader class pertains to the behavior of an individual sprite, such as its animations, while the Invaders class controls the behavior of the sprites as a group.*

### Invader Class

The Invader class gives the Invader sprites the following behaviors:

* Gives each unique sprite its own basic animation that changes every second
* If a laser enters its hitbox, the Invader deletes itself

#### Public Members

* Sprite[] animationSprites
  + - an array of type Sprite containing all Sprite elements
* float animationTime
  + - how often it cycles between different Sprites
    - initial value is 1.0f
* System.Action killed
  + - delegate to keep track of the killing

#### Private Members

* SpriteRenderer \_spriteRenderer
  + - a Unity array contains the Sprite we assign to it
* int \_animationFrame
  + - the indexer to cycle the elements of array animationSprites
* void Awake()
  + - first function unity calls
    - looks for the component specified under the same game object that Invader is attached to
    - in this case, it is looking for SpriteRenderer
* void Start()
  + - once the program is awake, this function is called
* void AnimateSprite()
  + - changing which sprite is being shown for each object
* void OnTriggerEnter2D(Collider2D other)
  + - If the projectile enters the hitbox of another object, it invokes the destroyed action. It then deletes itself.

### Invaders Class

The Invaders class gives the Invader sprites the following behaviors:

* Instantiates individual Invaders in an 8x5 grid formation
* Invaders repeatedly float from one side of the screen to the other, shifting down a position upon reaching each end of the screen
* A random Invader launches a projectile toward the player sprite at regular intervals depending on the number of Invaders left in the game

Public Members

* Invader[] prefabs
  + - A prefab with the invader script attached to it
* int rows
  + - Initial value = 5
* int columns
  + - Initial value = 11
* float spacing
  + - Initial value = 2.0f
* AnimationCurve speed
  + - AnimationCurve: x y graph, x is the percentage and y is the speed
* Projectile missilePrefab
  + - A Projectile prefab with the missile script attached to it
* float missileAttackRate
  + - speed of missile launch
* int amountKilled
  + - amount of killed invaders
* int amountAlive
  + - amount of alive invaders
* int totalInvaders
  + - the total amount of invaders
* float percentKilled
  + - the percentage amount of killed invaders

#### Private Members

* Vector3 \_direction
  + - the vector direction the invader is moving
* void Awake()
  + - first function unity calls
    - looks for the component specified under the same game object that Invader is attached to
* void Start()
  + - once the program is awake, this function is called
* void AnimateSprite()
  + - changing which sprite is being shown for each object
* void Update()
  + - update the status of the invader
* void AdvanceRow()
  + - change the direction of the invaders
* void MissileAttack()
  + - allows invaders to launch missile attacks
    - chances of invaders launching a missile increase as the number of invaders decrease
* void InvaderKilled()
  + - the invader gets killed

### SplineController Class

The SplineController class gives certain Invaders these movement behaviors:

* Invaders can move in quadratic-like curves in the game scene instead of only moving horizontally left and right

#### Public Members

* Invader[] prefab
  + A prefab with the invader script attached to it
* int totalInvaders
  + The total number of invaders
  + Initial value is 4
* float speed
  + speed of the spline
  + Initial value is 1.0f
* Vector3 PointA
* Vector3 PointB
* Vector3 PointC
* Vector3 PointD
  + These 4 Vector3 member variables represent the vector direction of the Invader along the curve path. Acts as anchor points along a spline.

#### Private Members

* float xPosB
* float yPosB
* float xPosC
* float yPosC
  + These 4 members represent the x and y positions of the Invader within the curve path
* float timer = 0.0f
  + Tracks our time
* float timeToMove = 2.5f
  + How long it takes for Invader to travel along its path
* float timerSpeed = 0.2f
  + Speed of timer
* float interpolateAmount
  + This is the amount that is interpolated along a spline
* void Awake()
  + Randomly instantiates an Invader prefab along the edge of the screen and calculates the path it will take
* void Update()
  + Continuously change the position of the Invaders using the
* Vector3 GetEdgePoint()
  + Sets the vector direction the Invader should have when it is instantiated on the edge of the game scene
  + The direction is determined by which screen edge is the closest to the random coordinate that is generated
* Vector3 QuadraticLerp(Vector3 a, Vector3 b, Vector3 c, float t)
  + Interpolates a value along a quadratic path given a path of Vector3 anchor points
  + Returns the current vector along the path of a, b, and c interpolating a value of t
* Vector3 CubicLerp(Vector3 a, Vector3 b, Vector3 c, Vector3 d, float t)
  + Uses QuadraticLerp to get two Vector3 values
  + Interpolates the cubic line given these two values.
  + Randomizes the Invader’s spline movements a little more than perfect quadratic curves.
* void OnTriggerEnter2D(Collider2D other)
  + When the Spline Invaders collides with a boundary, it will change PointA and PointD
  + This spawns them somewhere else and gives them a brand new path to take

## Projectile Class

*The Projectile class contains the script for both the Player and Invader projectiles.*

This class adds the following behaviors to a projectile:

* Move the projectile at a constant speed in a straight line from where it is instantiated
* Deletes itself when it collides with another hitbox

#### Public Members

* Vector3 direction
  + The direction of the projectile
* float speed
  + The speed of the projectile
* System.Action destroyed
  + Action that is invoked when the projectile is destroyed

#### Private Members

* void Update()
  + Projectile moves in a set direction and speed from the position it is invoked
* void OnTriggerEnter2D(Collider2D other)
  + If the projectile enters the hitbox of another object, it invokes the destroyed action. It then deletes itself.

## MainMenu Class

*This class controls what the play button does*

void PlayGame()

* This loads the SpaceInvader scene once the play button it is clicked