OUR EC327 PROJECT DOCUMENTATION TITLE HERE

Title of Game: . . .

How to Play: . . .

# 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 the potential graders of this project, who are the EC327 faculty members.

* Since our target audience 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 rip-off?” we could only realistically tell the people who are forced to run our project to assign a grade to it.
  + One could say maybe gamers would; however, a typical modern video game is leagues more advanced than what we have. It will simply not meet the expectations of the modern gamer.
  + Maybe very young children would enjoy our game. This could very well be true, but toddlers could literally be amused by a set of dangling keys

# Framework

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. We had to use this instead due to the file size limitations of GitHub.

# Game Development Itself

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 games assets:

* Script
  + Contains the C# files that gives
* Prefabs
* Scenes
* Sprites

We used Unity’s built-in Collider class to determine if any game objects are colliding with one another.

# Classes

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
* Bunker - represents the bunker
* Projectile - represents the invaders and players’ projectile
* SplineController - controls the spline

## 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 = 5.0f
  + Speed of the player

#### 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 *destroyed* is invoked from the Projectile class, call LaserDestroyed()
* void LaserDestroyed()
  + Sets \_laserActive to false
* void OnTriggerEnter2D(Collider2D other)
  + If an Invader or a missile collides with the player sprite, restart the scene

## Invader & Invaders Class

*The Invader sprite’s behavior is contained in two classes: Invader and Invaders. 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

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
* float animationTime = 1.0f
* System.Action killed

#### Private Members

* SpriteRenderer \_spriteRenderer
* int \_animationFrame
* void Awake()
  + - first function unity calls
    - looks for the component specified under the same gameobject that Invader is attached to
    - in this case, it is looking for SpriteRenderer
* void Start()
* void AnimateSprite()
* void OnTriggerEnter2D(Collider2D other)

### Invaders

*The plural Invaders class contains the functions for the Invader sprite that would not be affected by an individual sprite. ========1-2-=12-21xdxdxd*

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
  + - prefabs for invaders
* 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
  + - missile prefab
* float missileAttackRate
  + - speed of missile launch
* int amountKilled
  + - amount of killed invaders
* int amountAlive
  + - amount of alive invaders
* int totalInvaders
  + - total amount of invaders
* float percentKilled
  + - percentage amount of killed invaders

#### Private Members

* Vector3 \_direction
  + - direction of which the invader is moving
* void Awake()
  + - first function unity calls
    - looks for the component specified under the same gameobject that Invader is attached to
* void Start()
  + - once program is awake, this function is called
* void AnimateSprite()
* void Update()
  + - update the status of the invader
* void AdvanceRow()
  + - change the direction of the invaders
* void InvaderKilled()
  + - the invader gets killed

## 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.

## Bunker Class

*The Bunker class contains the script for the bunkers.*

#### Private Member

* void OnTriggerEnter2D(Collider2D other)
  + If the projectile enters the hitbox of another object, it invokes the destroyed action. It then deletes itself.

## SplineController Class

*The SplineController class contains the script for the spline.*

This class adds the following behaviors to a spline:

* ???

#### Public Members

* Invader[] prefab
  + The direction of the projectile
* 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

#### Private Members

* float xPosB
* float yPosB
* float xPosC
* float yPosC
* void Awake()
* void Update()
* Vector3 GetEdgePoint()
* Vector3 QuadraticLerp(Vector3 a, Vector3 b, Vector3 c, float t)
* void InvaderKilled()