Practical Gaming 2023

# Name of Student: Lucas J. O’Sullivan

# T Number: T00207050

# Name of Project: PG2023-Lucas

# Gameplay

Shoot targets to destroy them and score points; score as many points as possible before time runs out.

* Controls
* Left Mouse Button: Fire
* Left Control: Walk
* Left Shift: Sprint
* Mouse: Look
* W: Move Forward
* A: Move Leftward
* S: Move Rightward
* D: Move Backward

# Coding

* Frame Rate Independence
  + Frame rate independence refers to the writing of code that is executed at a fixed rate, as opposed to as fast as allowed by system resources.
  + Assets such as the game timer and weapon utilise frame rate independence through values that are repeatedly decremented by ‘Time.deltaTime’ to ensure that the weapon’s rate of fire is consistent at all times, and the game timer counts down in real time.
* Interfaces
  + An interface is a fully abstract class that contains methods which can be instantiated by classes that implement the interface. As interfaces are abstract, classes need to contain code within their instances of these methods to make effective use of them.
  + An interface named ‘IShoot‘ contains three methods and is implemented by a class named ‘Weapon\_Control,’ which allows that class to execute code based based on whether the left mouse button is held, or on each click of the left mouse button.
  + An interface named ‘IHealth’ contains two methods and is implemented by a class named ‘TargetHealth’ which allows that class to execute code whenever the object to which it is attached (i.e. the face of a target) is struck by an object containing the ‘FRID\_Bullet\_Script’ class.
* Inheritance
  + Inheritance is where a class assumes the attributes of another class. For example, if class ‘Organism’ contains an attribute named ‘can\_move’ and class ‘Animal’ extends ‘Organism,’ ‘Animal’ acquires the ‘can\_move,’ attribute, and the ‘can\_move’ attribute within the ‘Animal’ class can be initialized without influencing the ‘Organism’ class.
* Case pattern
  + A case pattern involves the programming of a ‘switch’ statement, which is a set of conditions referred to as ‘cases;’ within the same switch, cases can invoke one another, and are each closed with a ‘break’ statement.
  + In programming a ‘recoil’ animation, a switch containing two cases, one for generation of recoil, and the other for recovery from recoil, was created.
  + Whenever the weapon is fired, script ‘RecoilScript’ contains the ‘Recoiling’ case, which is triggered from the ‘Weapon\_Control’ script through ‘Aimpoint\_Script’s’ ‘startRecoiling()’ method, causing the weapon to gradually rotate out of alignment with the player’s point of aim; once a brief period has passed, ‘Recoiling’ invokes the ‘Returning’ case, which causes the weapon to gradually return to alignment with the player’s point of aim.
* Observer Pattern

  + An observer pattern was created to ensure that each target would take damage only whenever the face of the target was struck by a bullet.
  + Using the ‘OnCollisionEnter()’ method, each bullet checks for the presence of the ‘IHealth’ interface in the object it struck using “collision.transform.GetComponent<>,” and if ‘IHealth’ is present, calls the object’s ‘takeDamage()’ method.
  + The “Target Face” asset contains ‘IHealth’ and the class ‘TargetHealth’ which inherits “takeDamage(float incomingDamage)” from ‘IHealth.’
* Polymorphism
  + Polymorphism is variation in the behaviour of an inherited method on a per-class basis.
  + An interface or abstract class can store a method and be implemented or extended by multiple classes; this saves time and improves consistency among scripts by having them call a common method.
* Communication between scripts/game objects
  + Through Unity, C# scripts can communicate with each other through the ‘FindObjectOfType<>()’ method and by extending each other; an example of how this appears is this: public class ScriptA{ScriptB receiver; void Start(){receiver = FindObjectOfType<ScriptB>(); receiver.getMessage();}}
  + Heavy usage of communication between scripts has been made to enable several functions.
  + When the ‘space’ key is first pressed, methods are called to: enable player control, hide introductory text, display the ‘timer’ and ‘score’ objects, and begin the game timer, which counts downward.
  + Once the game timer has counted down to zero, methods are called to: disable player control, display closing text, hide the ‘timer’ and ‘score’ objects, and stop the game timer.
* Instantiation and Prefabs
  + Within the Unity editor, instantiation is the process of creating an object from within a running game, and prefabs are objects, or sets thereof, that are saved to files, which allows for quick and easy reuse of assets.
  + Prefabs were created for easy instantiation of bullets, targets, target spawners, and sound emitters.
  + Whenever the weapon fires, a bullet is instantiated. Once the ‘health’ value of a target has been lowered to at least zero, a method is called to destroy the target, then instantiate an empty object that plays a sound effect (i.e. a ‘sound emitter’) and destroys itself after a short period, and to instantiate an empty object that idles for a short period, then instantiates another target and destroys itself; the cycle repeats until the player stops destroying a target.
* Magic Numbers
  + In programming, ‘magic numbers’ are effectively numbers created out of thin air; a magic number is hard-coded, and no variables are initialized with a magic number.
  + While magic numbers are less comprehensible than named variables and also hamper a script’s flexibility, owing to the fact that several magic numbers cannot all be changed at once, I felt that in one case, a magic number of one was justified; in the ‘Weapon\_Control’ script, there is a variable named ‘roundsPerSecond’ and a variable named ‘refire\_delay,’ which both determine how fast the weapon can fire in succession; whenever the weapon is fired, ‘refire\_delay’ equals one divided by ‘roundsPerSecond.’ The magic number appears only once in the script, and I feel that a variable named ‘refire\_delay’ can be presumed to have a relationship with one named ‘roundsPerSecond.’
* Model Animation
  + Animation in Unity can be performed through the ‘Animator’ component.
* Self made models and or animations
  + Targets, geometry, and the weapon were all modelled by me within the Unity editor.
  + Within the Unity editor, objects can be moved, scaled, and rotated using the toolset to the left of the Scene view, or by adjusting the ‘Transform’ values of selected objects. Objects may also be mounted to one another as part of a parent-child relationship.
  + The Unity editor allows for the creation of primitive shapes, such as cubes and spheres.
* Interactions between objects/scripts
  + A way objects can interact with each other is through the ‘OnCollisionEnter()’ method.
  + The face of a target contains a ‘health’ value, and a bullet contains a ‘damage’ value. Whenever a bullet collides with a target’s face, the latter’s ‘health’ value is subtracted by the former’s ‘damage’ value.
* Proper code placement

  + It seems to be generally preferred that lines take up no more than sixty characters in length;
  + Nested statements should be indented within each other to reduce readers’ confusion.
* Code repetition
  + Repetition is the writing of code that is executed repeatedly; this is typically done using loops, such as ‘while’ and ‘for’ loops, though Unity provides an ‘Update’ method to repeatedly execute code with each frame update, and a ‘FixedUpdate()’ method to repeatedly execute code at a fixed rate.
  + Repetition is used to enable the weapon to fire successively for as long as the player holds down the left mouse button.
  + Using ‘Time.deltaTime’ within the ‘Update()’ method, a float named ‘refire\_delay’ is constantly decremented, and whenever the weapon fires, ‘refire\_delay’ is set to one and then divided by a float named ‘roundsPerSecond;’ while ‘refire\_delay’ is greater than zero, the weapon cannot fire a follow-up shot.
* Feature 1
  + Programmed ‘recoil’ animation
* Feature 2: Sound effects
  + Sound effects can be played in Unity by attaching an audio source component to a game object; a script defining an ‘AudioSource’ object, an ‘AudioClip’ object, and a float can be used to play a specific audio clip through a specific audio source, at a given volume (I recommend that the value of the float does not exceed 1.0f).
  + A sound effect is played using the ‘PlayOneShot()’ method; the difference between ‘PlayOneShot()’ and the method ‘Play()’ is that calling ‘Play()’ in a sound object causes any sound clip already playing from that sound object to stop playing, whereas ‘PlayOneShot();’ permits repeated playing of a sound clip without intermittent stopping.
  + I used free assets and made edited copies. I do not take credit for the audio recordings used in this project.
* Feature 3