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

  + In programming a ‘recoil’ animation, a switch containing two cases, one for generation of recoil, and the other for recovery from recoil, was created.
* Observer Pattern
* Polymorphism
  + Polymorphism is the provision of
* 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; no variables are directly associated with a magic number.
  + While magic numbers are less comprehensible than definite constants and also hamper a script’s flexibility, I felt that
* Model Animation
* Self made models and or animations

  + Targets, geometry, and the weapon were all modelled within Unity.
* Interactions between objects/scripts

  + 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
* 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
  + A sound effect is played
* Feature 3