



# Thalmic Labs - Myo gesture control armband. Exploring Myo's potential as a game controller

Gesture based Isometric Shooter game using the Myo Thalmic Armband  
[Github repository for this project](#)

David Gallagher, Justin Servis

February 2020

A gesture based UI project utilising Thalmic Labs Myo Armband technology as a game controller. Control a gun wielding 'loose cannon' type guy who's got guns, bullets, a belt that fires bullets.. and a taste for carnage..

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Purpose of the Application</b>	<b>5</b>
2.1	Game Concept . . . . .	5
<b>3</b>	<b>Gestures identified as appropriate for this application</b>	<b>6</b>
3.1	Gestures and Poses . . . . .	6
<b>4</b>	<b>Hardware used in creating the application</b>	<b>7</b>
4.1	Thalmic Labs - Myo Armband . . . . .	7
4.2	Alternatives to the Myo . . . . .	7
4.2.1	Microsoft Kinect . . . . .	7
4.2.2	Leap Motion Controller . . . . .	8
<b>5</b>	<b>Architecture for the solution</b>	<b>9</b>
5.1	Classes . . . . .	9
5.1.1	Input . . . . .	9
5.1.2	Feedback System . . . . .	11
5.1.3	Weapons . . . . .	13
5.1.4	Living Entities . . . . .	14
5.1.5	Other Classes . . . . .	15
5.2	Class Diagrams . . . . .	15
<b>6</b>	<b>Game play and Controls</b>	<b>16</b>
6.1	Movement . . . . .	16
6.2	Actions . . . . .	16
6.3	Using Myo gestures. . . . .	18
6.4	Haptic Feedback . . . . .	20
6.5	Player Engagement . . . . .	20
6.5.1	Sensation . . . . .	21
6.5.2	Challenge . . . . .	21
<b>7</b>	<b>Test Plan</b>	<b>22</b>
<b>8</b>	<b>Conclusions and Recommendations</b>	<b>23</b>
8.1	A Gesture based Myo project . . . . .	23
8.2	Myo as a game controller. . . . .	23
8.3	Future development ideas for this project. . . . .	24
8.4	Final Thoughts . . . . .	24
<b>9</b>	<b>Specification</b>	<b>25</b>
<b>10</b>	<b>References</b>	<b>26</b>
10.1	Reference Images . . . . .	26

# 1 Introduction

The objective of this project is to develop or utilise a series of gestures that provide a user with a natural user interface and enable them to achieve some goal. The gestures should be intuitive and feel appropriate for the context in which they are used. They should be reliable and performant. There should be a clear reasoning for a gesture to be included in the gesture set.

After some initial discussion we decided on a 3D shooter game and set to researching which hardware would suit the project we had in mind. Some examples would include a voice command based interaction where the user will say certain key words or phrases in order to perform a specific action, Leap Motion controller, Microsoft Kinect. (Both of the latter are motion based sensors).

We had a firm idea as to what the end result of this project should be and what game mechanics we wanted to include. We discussed which gestures could be used for performing specific actions and which technologies we might use to implement our game.

We looked at three possible hardware options for this project.

- Motion tracking the player movement using Microsoft Kinect.
- Sensor based gesture recognition using Myo Thalmic Armband.
- Sensor base hand and finger tracking using Leap Motion Controller.

We listed out the pros and cons of each technology as they related to the project we had decided on and chose to develop it with the Myo armband. Further reasoning on the use of the Myo over the Leap or the Kinect can be found in our section on [hardware](#).

Once we had decided on the hardware we set to deciding on what software would fit well with the project. Myo is supported by SDK's in multiple languages. We had a particular interest in working in C Sharp. We looked at [Godot Engine](#) and [Unity Game Engine](#) as candidates for the game implementation but ultimately settled on Unity due to it's overwhelming availability of documentation.

Given that Myo is no longer in production, the [Myo Marketplace](#) does not appear to be very active. This would have been a nice platform on which to share the project for review and feedback. There may be other options in this regard, something we will keep in mind in our [Future Development](#) section.

## 2 Purpose of the Application

The purpose of the application is to assess the Myo's capability as a gaming controller. To test the Myo's existing gesture controls as commands in a game scenario. To form an informed opinion on the Myo as a games controller compared with other existing control schemes. And finally, to provide the user with a fun and engaging experience using the Myo armband as a game controller.

As our game also requires keyboard input, it will be similar to a mouse and keyboard style game-control scheme however the Myo is a challenging replacement for the mouse. Traditionally isometric shooting style games are played using a keyboard and mouse, or with a game controller such as a gamepad, or with a joystick.

Our intention is to examine whether the Myo can really replace something as reliable as any of the traditional gaming controls. As such, we have a few questions to address..

- Does it provide an enjoyable gaming experience..
- Can it keep up with the demands being placed on it in a gaming environment..
- Is it reliable enough to replace other gaming controllers..

### 2.1 Game Concept

Single player game in which the player controls a character. The player will face enemies who are literally programmed to attack on sight and must 'shoot first and ask questions later'. The player will wear the Myo armband, which recognises a number of gestures outlined in [Gestures and Poses](#). The character has access to a series of guns and a radical belt which fires bullets in multiple directions at once, decimating the wave of approaching enemies.

The player will face waves of enemies across a series of maps. Each map progresses with increasing difficulty, as more enemies populate subsequent maps. The player is free to swap weapons as they try to eliminate the waves of enemies, making use of the secondary fire option, and using the terrain to their advantage to attempt to slow the enemies approach. The game serves as a platform to test using 5 of the Myo's gestures, the vibration motor, and tracking the rotation of the players arm in parallel. Navigation around the map will require deft fingers to use the keyboard arrows or WASD keys to move the player up, down, left and right.

## 3 Gestures identified as appropriate for this application

The use of gestures in this project are based on the well designed and developed gestures which are provided with the Myo Thalmic Armband. We knew from the outset that we would want to use two input devices. The gestures and how they are used in relation to controlling the player character are further explained in [Gameplay and Controls](#).

### 3.1 Gestures and Poses

- Fingers Spread - Performed by extending all fingers on your hand as shown in the image in table 3.1
- Fist - Performed by making a fist with your hand as shown in the image in table 3.1
- Wave out - Performed by extending your hand towards the top side of your forearm as shown in the image in table 3.1
- Wave in - Performed by extending your hand towards the under side of your forearm as shown in the image in table 3.1
- Double Tap - Performed by tapping your middle finger off your thumb twice in quick succession as shown in the image in table 3.1






Myo	Pose	Images
		
		

Table 3.1: Myo Pose Table

## 4 Hardware used in creating the application

### 4.1 Thalmic Labs - Myo Armband

In this Project our hardware is the Myo Armband from Thalmic Labs. Myo has in-built gestures as outlined in [Gestures and Poses](#) which we will use to control our player character, replacing the role of the mouse in a generic keyboard and mouse style control scheme. We will leverage the Gyroscope in order to replicate moving the mouse, while the gestures will perform specific actions, each of which will be outlined in the section on [Gameplay and Controls](#).

We referred also to the [Technical Features and Functionalities of Myo Armband](#) which has a detailed breakdown of the components in the armband. In particular the section on 'Myo armband: functionalities and technical features'.

Thalmic Labs' [Myo Gesture Control Armband](#) features an [Inertial Measurement Unit \(IMU\)](#) and 8 surface ElectroMyography sensors (sEMG) in addition to a Windows SDK for development. The Myo armband measures electrical activity from your muscles to detect five [gestures](#) made by your hand.

The 9-axis IMU is comprised of a 3-axes accelerometer, a 3-axes Gyroscope and a 3-axes Magnetometer. This enables the armband to track the motion, orientation and rotation of the user's arm in 3 dimensions.

The inclusion of a vibration motor allows the armband to alert users to events, a feature we are using in our control scheme for haptic feedback.

The armband transmits information wirelessly over Bluetooth via a BLE NRF51822 chip to communicate with compatible, connected devices. The range of this connection differs depending on the environment but is generally accepted to be in the range of 10 metres indoors to 50 metres outdoor.

Myo also streams the raw EMG and IMU data. There are a litany of research papers available which leverage the Myo armbands EMG and IMU extensively in the field of [rehabilitation](#) and [recovery](#).

### 4.2 Alternatives to the Myo

Other hardware options to use for a game of this nature are not limited to devices with an IMU, however a gyroscope is very useful for controlling the movement of a character.

#### 4.2.1 Microsoft Kinect

It would be feasible to use a [Kinect](#), using its camera and adding your own custom gestures to control your game character. The Kinect contains three vital pieces that work together to detect your motion and create your physical image on the screen: an RGB color VGA video camera, a depth sensor, and a multi-array microphone. Kinect shows great promise in the area of rehabilitation. The authors of [Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor](#). identify the increasing popularity of health and fitness based games using Kinect and other controllers

capable of motion detection. Their paper explores and assess "an interactive game-based rehabilitation tool for balance training of adults with neurological injury". We believe the Kinect would offer a more active alternative to the Myo, or the Leap, as it's a motion detector using a camera system it can recognize full body gestures such as waving arms, jumping up and down, ducking..

#### 4.2.2 Leap Motion Controller

[Leap Motion Controller](#) is another promising piece of hardware for use in gaming. Leap Motion consists of two cameras and three infrared LEDs. These track infrared light with a wavelength of 850 nanometers, which is outside the visible light spectrum. Due to its wide angle lenses, the device has a large interaction space of about eight cubic feet while it's viewing range is limited to roughly 2 feet (60 cm) above the device. [Gesture-based Interactions in Video Games with the Leap Motion Controller](#) explores the Leap Motion controller and it's capability and limits in gaming at great length.

There are more limitations with the Leap in our opinion, given the limited field of interaction it's suited to stationary interaction. The Myo in comparison has a range of 10 to 50 metres depending on the environment, allowing much more freedom.

As with any motion detecting technology, the Leap, Myo or Kinect to name a few, there are pro's and cons with each. Each finding their respective niche applications for different reasons.



## 5 Architecture for the solution

### 5.1 Classes

This section contains an overview of the class structure used within the application. It is not intended to be exhaustive and encompass every aspect of the game, rather the main systems and how the components within those systems interact with each other. Each section below refers to one of the UML diagrams from section 5.0.2.

#### 5.1.1 Input

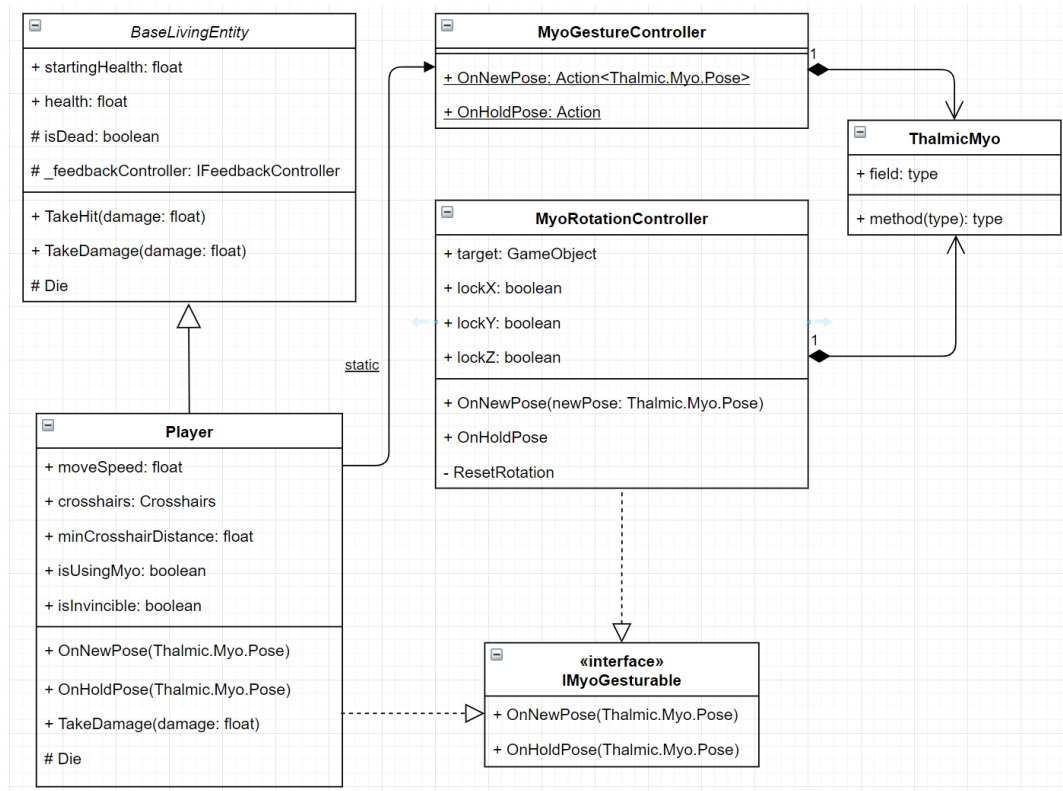


Figure 5.1: UML for Input System

**ThalmicMyo** - provided by the Thalmic Myo SDK, this class is the representation of a Myo device within the Unity application. It maintains an instance of the Myo C Sharp class which is used to interact with the Myo device.

**IMyoGesturable** - an interface which defines behaviours for entities which can be controlled by the Thalmic Myo gesture recognition functionality through the MyoGestureController GameObject. Any implementing class is expected to add the implemented methods to the static events of the MyoGestureController. These methods will then be invoked once the related event has been fired.

**MyoGestureController** - handles the Myo gesture recognition functionality. It maintains a reference to the ThalmicMyo GameObject and can access properties from this

object to determine when a new gesture is performed by the user or when a gesture is being held by the user. The check for the current gesture recognised by the Myo is performed each frame in the Update method.

- When a new gesture is recognised by the Myo, the OnNewPose event is invoked.
- When a gesture is recognised as being held by the Myo, the OnHoldPose event is invoked.

By using the two static events above, any implementations of IMyoGesturable will also have their OnNewPose or OnHoldPose methods called during the appropriate frames.

**MyoRotationController** - handles the reading of the current rotation from the Myo device, calculates the rotation for Unity to use and applies this rotation to the 'target' GameObject (i.e. the Player entity). It maintains a reference to the ThalmicMyo GameObject from which it can read the rotational values.

Through its implementation of IMyoGesturable, when a new Double-Tap gesture is detected, the target's rotation is reset to the 12 o'clock position on screen regardless of the current rotation of the Myo. This allows for a correction to be applied and the user to use the device in a more comfortable position.

### 5.1.2 Feedback System

This system is designed as a set of Plain Old CLR Objects (POCOs) for configurations and a set of interfaces to define the expected functionality. In the scope of this project, the use of these interfaces is not entirely necessary but it would allow for the easy addition of another device in future. It would be a simple task to create new implementations of the required interfaces, e.g. in the case of `IVibrateable`, if a different device with haptic feedback was to be used instead of the `Myo`.

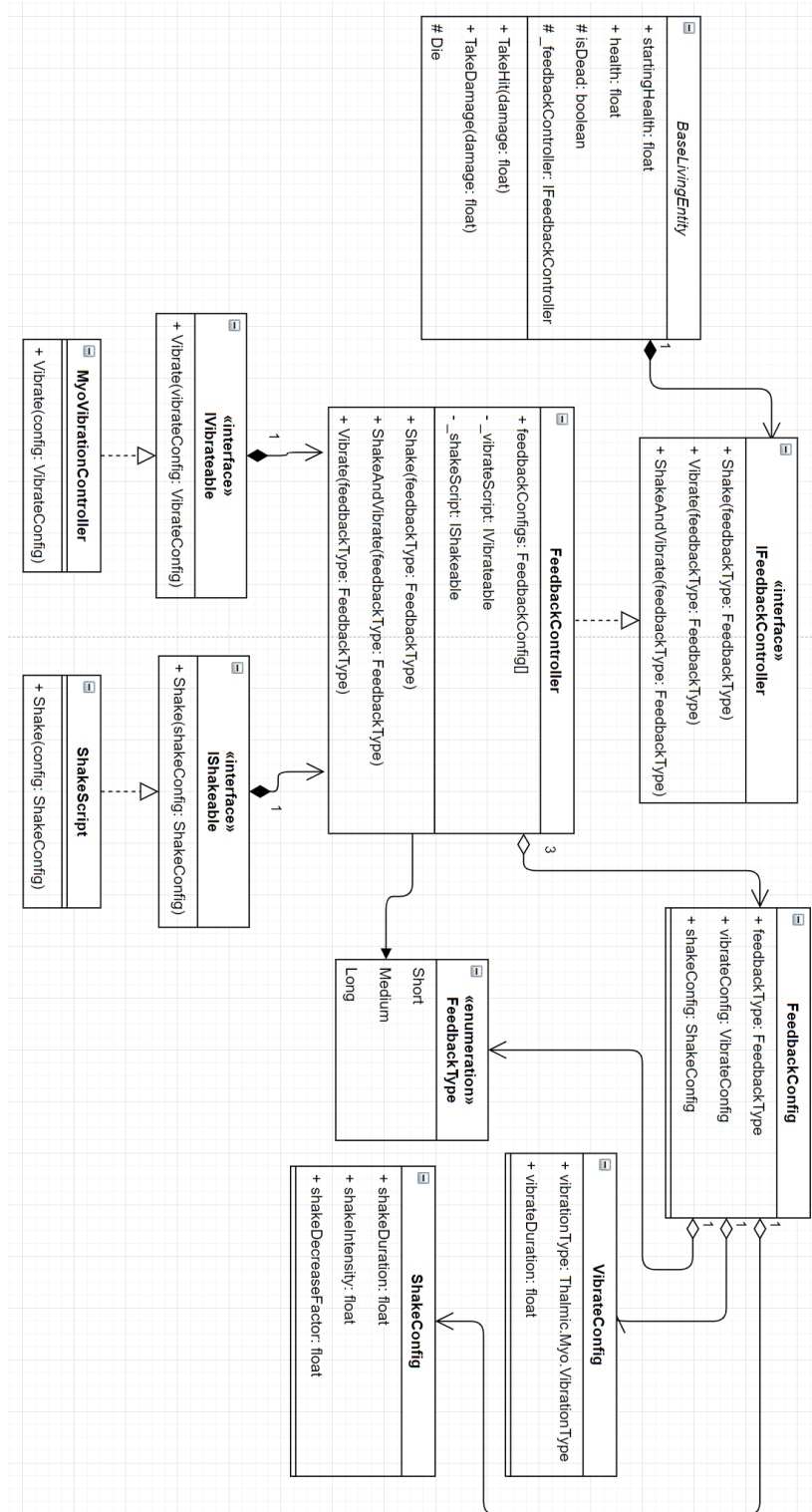


Figure 5.2: UML for Feedback System

**VibrateConfig** - Plain Old CLR Object (POCO) class which describes a vibration configuration. A vibration configuration is composed of a type and duration, e.g. (Short, 300ms)

**ShakeConfig** - similar to a VibrateConfig this is also just a POCO class which describes a shake configuration. A shake configuration is composed of a duration, intensity and a decrease factor.

**FeedbackConfig** - another POCO class which is composed of a FeedbackType enum, a VibrateConfig and ShakeConfig.

**IShakeable** - interface which contains a single Shake method which takes a ShakeConfig object as its argument. This should be implemented by any class which needs the functionality to shake the screen as a form of user feedback.

**IVibrateable** - interface which contains a single Vibrate method which takes a VibrateConfig object as its argument. This should be implemented by any class which needs the functionality to perform vibration as a form of user feedback.

**MyoVibrationController** - an implementation of IVibrateable for the Myo device.

**ShakeScript** - an implementation of IShakeable to shake the screen.

**IFeedbackController** - interface which defines the feedback operations which can be performed: Shake, Vibrate and ShakeAndVibrate. Each method takes a parameter of enumerated type FeedbackType of which there are three values: Short, Medium and Long. This functionality has been abstracted to an interface to ease the support of multiple different devices with haptic feedback as desired.

**FeedbackController** - implementation of IFeedbackController for Myo haptic and screen shaking feedback. The typical usage involves a single instance of this in the game scene, which other objects can obtain a reference to and invoke the IFeedbackController methods as needed.

### 5.1.3 Weapons

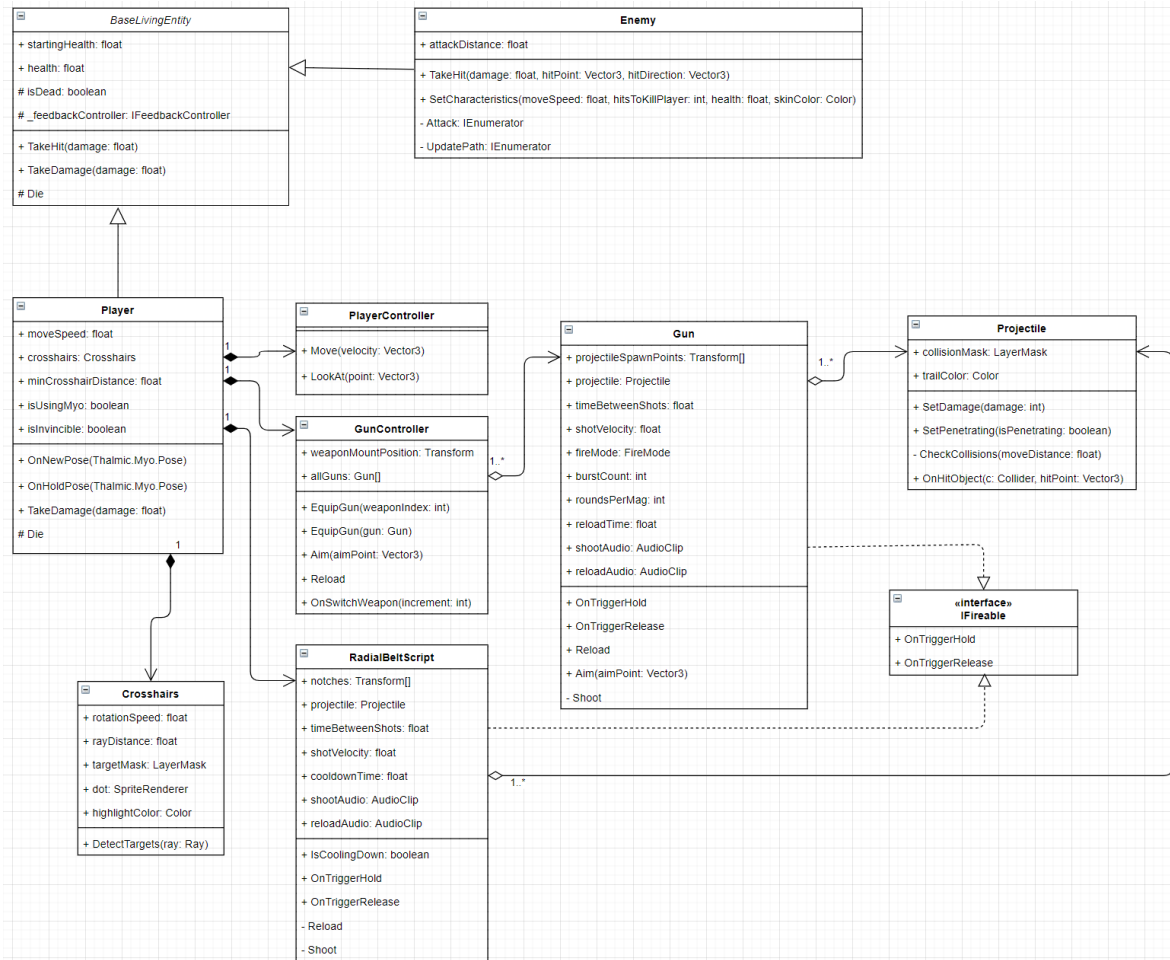


Figure 5.3: UML for Living Entities and Weapons

**Projectile** - represents any projectile fired from any weapon, be it a Gun weapon or the RadialBelt weapon. When a projectile collides with an IDamageable entity, it will call that entity's implementation of the IDamageable.TakeHit method. Once time has passed in the game equal to the projectile's lifetime variable, the projectile is destroyed so as to not waste memory. Three public methods are present which allow the projectile's damage, speed and penetrating variables to be set on instantiation.

**Shell** - the script component of a GameObject which is instantiated each time a Gun weapon is fired. This is designed to appear as a bullet casing being ejected from the weapon and once its lifetime has expired, it is destroyed within the game.

**Gun** - represents one of the Player's main weapons in the game. It is an implementation of IFireable and handles firing, reloading and weapon sounds. Public variables are present so that guns have different rates of fire, reloading times, magazine sizes, sounds, etc. Its methods are typically called from the GunController component of the Player entity. A gun can have multiple Projectile spawn points, e.g. a shotgun weapon has multiple pellets which are each spawned from a projectile spawn point.

**IFireable** - interface which defines behaviour similar to a weapon's trigger through two methods - OnTriggerHold and OnTriggerRelease.

**RadialBelt** - represents the Player's secondary weapon which can be fired only once before initiating a cooldown timer. Radial Belt notches act the same as projectile

spawn points, with each one spawning a separate projectile. Similar to Gun, it is also an implementation of `IFireable`. Projectiles fired from this weapon are penetrating, i.e. they may travel through an enemy after applying damage and continue on to damage any more enemies which they collide with.

#### 5.1.4 Living Entities

The UML for this section can be seen in Fig. 5.3

**IDamageable** - an interface which defines behaviours for entities which can be damaged. Currently, Enemy entities can take damage from projectiles fired by any of the Player entity's weapons and the Player entity can take damage from a direct hit from an Enemy or by falling off the playing map.

**BaseLivingEntity** - abstract class which is the base class for living entities within the application, i.e. Player and Enemy. Keeps track of the entity's health and whether the entity is currently 'alive'.

This class is an implementation of `IDamageable` and since `BaseLivingEntity` is abstract, both methods are implemented but marked as virtual to allow overriding where required.

An implementation of `IFeedbackController` is referenced which allows for any subtype to initiate user feedback (see Feedback Section)

**Player** - The Player class adds a lot of functionality to its supertype, most notably the ability for the Player entity to receive input from the user which allows it to be controlled on screen. This class holds a number of references to script components which facilitate certain behaviours:

- **PlayerController** - controls the Player entity's movement and rotation through the invocation of its two public methods: `Move` and `LookAt`.
- **GunController** - controls the firing, reloading and switching of the Player's main weapons. After performing checks, this functionality is generally delegated to the equipped Gun instance.
- **RadialBeltScript** - controls the firing and cooldown timer of the Player's secondary weapon. Similar to the relationship between Gun and GunController, this script delegates firing to the RadialBelt GameObject.

**Enemy** - In contrast to the Player class, each Enemy entity's movement is controlled by Unity's `NavMeshAgent` class which is an AI implementation. Each enemy entity has a target variable of type `BaseLivingEntity` (which is set to the Player) and will track and home in on this target when instantiated. Once an enemy is within a certain distance of the target, it will attack and call the target's `IDamageable.TakeDamage` method.

### 5.1.5 Other Classes

#### Sound and Effects

- **AudioManager** - In game audio controller for managing audio sources and volumes.
- **MusicManager** - Manages the in-game menu and gameplay music.
- **SoundLibrary** - Central store for all AudioClip used in the game, e.g. sounds for firing projectiles, dying, bullet impact. Allows for retrieval of sound effects by name.
- **MuzzleFlash** - Muzzle flash effect for weapons. A container for sprites to display MuzzleFlash upon firing a gun.

#### Map Generator and Enemy Spawning

- **EnemyWave** - POCO class which represents a wave of enemies in the game. Each level of the game is defined by an instance of this class.
- **MapGenerator** - Utility to generate a map of size and populates it with obstacles of size and with percentage of occurrence.
- **Spawner** - Manage spawning mechanics and locations for enemies in each wave.

#### UI and Menus

- **Crosshairs** - Controls the display and rotation of the crosshairs GameObject on the UI. The position of the crosshairs is handled by the Player class.
- **UIController** - The main game UI manager. Handles scene transition, displaying or hiding various UI elements such as the Game Over screen, updating the score in the UI and animating the New Wave popup.
- **Menu** - UI Menu for starting and quitting the game, options for screen resolution and volume controls.
- **ScoreKeeper** - Keeps track of the score in the game.

#### Utility

- **Enums** - Definitions of states for enemy behaviours (Idle, Chasing, Attacking), gun firing modes (Single, Burst, Auto), audio channels (Master, Music, Sfx) and durations of feedback (Short, Medium, Long).
- **Utility** - Generates a seed to randomize the appearance of generated maps from MapGenerator.
- **RotationMaths** - Contains static methods for performing rotational manipulations of vectors. Used by the MyoRotationController class to apply the rotational data from the Myo device to the Player GameObject.

## 5.2 Class Diagrams

Full size class diagrams available [here](#) in the github repository.

## 6 Game play and Controls

Character movement is a mixture of keyboard input and Myo armband. The Myo armband is the perfect candidate hardware to use for a shooting game. It's worn on the users arm and interacting with it requires the user to perform hand gestures. We have put careful consideration into which hand gestures are used for which in-game action and further outline these decisions in the section on [Actions](#).

The fact you would hold a gun in your hand and that you wear the Myo on your arm is not lost on us, this factored into us choosing the Myo. We feel this adds a layer of realism to the game, which is important in providing the experience we were aiming for.

### 6.1 Movement

Arrow keys and WASD keys for character navigation in a game environment are ubiquitous in the vast majority of pc games, similar to using thumb-sticks on a game controller has become more ubiquitous since the release of the first playstation/original xbox. Though thumbsticks existed before the release of Sony and Microsoft's consoles, their popularity really fastened the concept into the game controller control scheme. As the Myo has a gyroscope to capture rotation in real time we decided to use the armband to manage the character rotation. In a game of this nature, the rotation would often be controlled by a thumb-stick or a mouse.

- Up Arrow / W key.
  - Move character in forward direction.
- Down arrow / S key.
  - Move character in backward direction.
- Left arrow / A key.
  - Move (Strafe) character in left direction.
- Right arrow / D key.
  - Move (Strafe) character in right direction.
- Myo Armband with player arm rotation.
  - Rotate character in the y-axis (horizontal to the ground), to turn the character to face enemies.

### 6.2 Actions

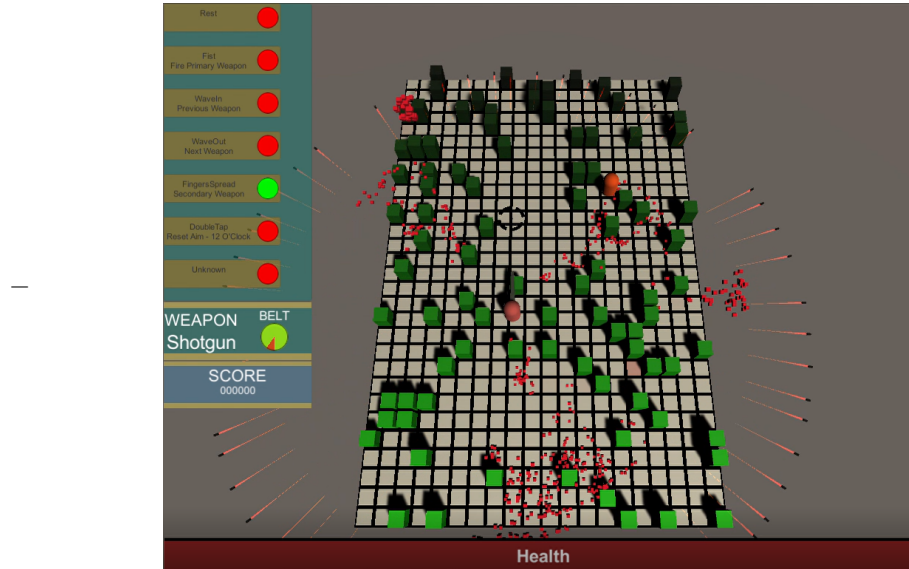
Assigning intuitive gestures to appropriate actions is important for the player to quickly absorb the control system and immerse themselves in the game play. We are replacing some well understood controls for character movement and actions, with the use of arm tracking and Myo gestures.

Where a player might normally expect to use a key such as 'E', or 'mouse button 1', or

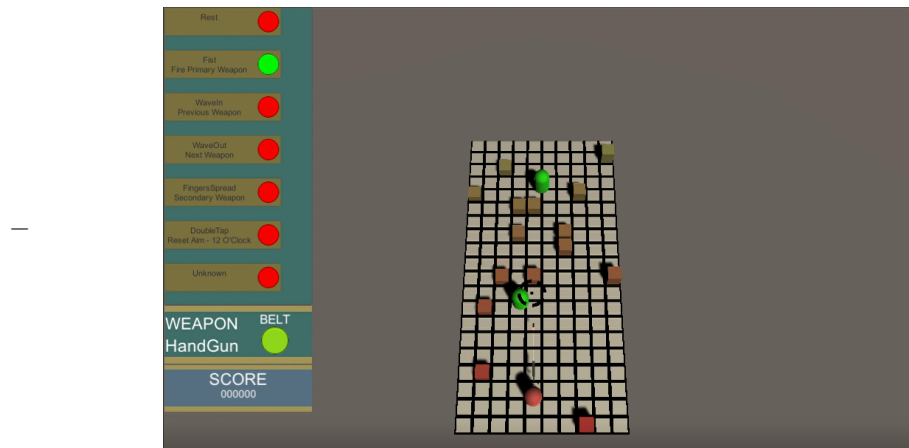


a game pad button such as 'X' or 'Y' to perform an in game action - here they will be replaced with gestures recognised by the Myo.

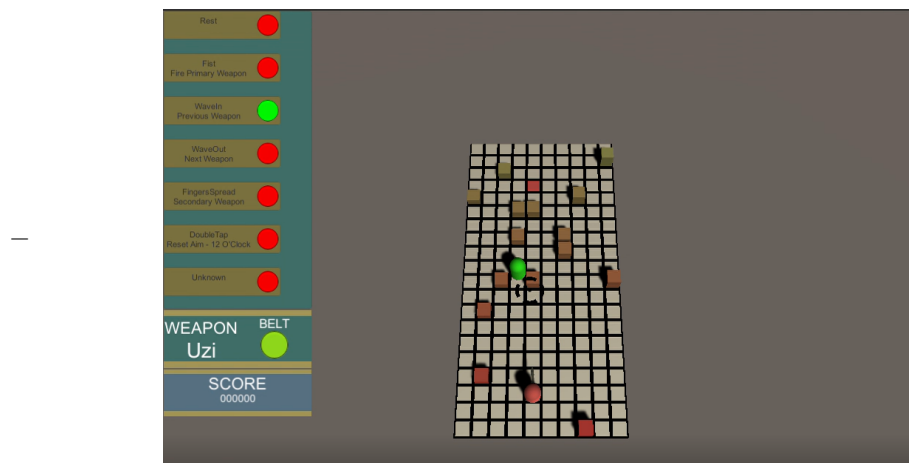
- Fingers Spread
  - Fire secondary weapon.



- Fist
  - Fire primary weapon.

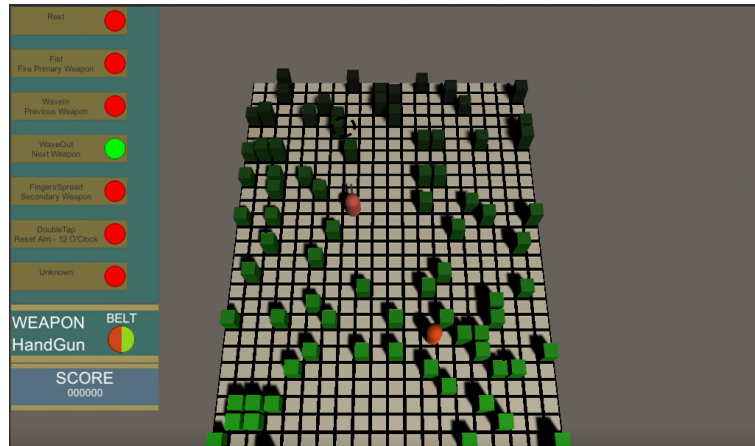


- Wave In
  - Switch to previous weapon.



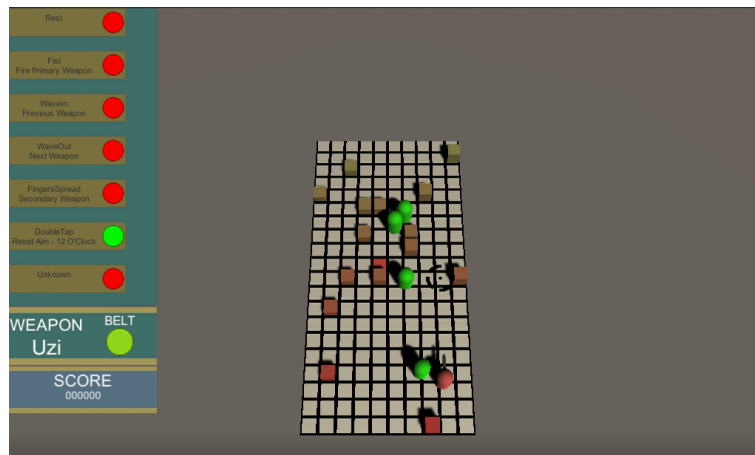
- Wave Out
  - [Switch to next weapon.](#)

—



- Double Tap
  - [Reset Player aim direction.](#)

—



## 6.3 Using Myo gestures.

Careful consideration went into planning which gesture would be assigned to which in-game action.

In our [Actions](#) section we outline the gestures and actions relationship in our control scheme. The most intuitive use of the gestures are those that the player only needs to think about once and can then perform as though they had always known that 'this gesture performs this action'.

The following is a list of Myo gestures and examples of an intuitive behaviour to attach to that gesture. This should be viewed in the context of a game, but is also appropriate for other software or hardware applications. In theory, anyone could argue a wide range of actions could apply to this list of gestures, but we've kept it concise and are only referring to well understood intuitive gestures.

- Double Tap Index finger to thumb.
  - Select an item.
  - Pick up an item.
  - Press a button.
  - Configure a position - this is a bit specific to using the Myo however.

- Fist.
  - Grab an object.
  - Crush an object.
  - Stop (in a Military command type scenario).
  - Hold an object.
  - Hand Fist.
  - Pull a trigger.
- Fingers Spread.
  - Drop an object.
  - Stop gesture - could have a range of meanings depending on the context. Such as to stop a video playing, stop an incoming projectile in a game scenario.
  - Hand Open.
- Rotate/Move Myo arm.
  - Rotate selected object clockwise, counterclockwise.
  - Rotate selected object in 3D space x, y and z.
  - Move an object.
  - Hand Rotate.
- Wave In.
  - Slide a slideshow panel left/right.
  - Open/close a door.
  - Wrist flexion (where the wrist is flexed towards the inside forearm).
- Wave Out.
  - Slide a slideshow panel.
  - Open/close a door.
  - Wrist extension (where the wrist is flexed towards the outside forearm).

With regards to Hand Open, Fist, Wrist extension/flexion - in a game sense, these gestures could animate between different character hand states in a literal sense such as open hand, closed hand or fist etc.

We determined suitable actions to perform with the Myo gestures where the action performed was closely related to one of the intuitive actions we've listed in our table.

One example of which is the 'Fist' pose to fire a weapon. As you would normally hold a gun with your hand clasped around a handle, this gesture feels intuitive and fit for purpose. Alternatively, the 'Fingers Spread' pose fires the secondary weapon which releases a spray of projectiles in every direction. There are not many hand gestures associated with a hand having all fingers spread and extended. However, similar to how a person might fling their hand to spray off excess water after rinsing them under a tap, this gesture is flinging projectiles away from the player character. We think there is a strong link between the two actions and decided it was fit for purpose.

The 'Wave In' and 'Wave Out' poses are used to cycle forwards and backwards through the weapons available to the character. We assigned these actions knowing that the same poses are used in a slide show application we looked at during our initial research, to cycle

forwards and backwards through a series of photos or slides. This application of the poses makes perfect sense as there's a direct mapping of the action performed by the user to the outcome within the game.

Finally the 'Double Tap Middle finger to Thumb' pose. This is used in game to reset the aim direction of the player. We had to assign one of the poses to this action as we felt the game really needed this feature. For reasons outlined in the [conclusions and recommendations](#) section regarding some unexpected behaviour from the Myo, there was a need to be able to make the player character aim to 12 O'Clock. In some instances when beginning the game the player rotation would be pointing the wrong direction, or upon changing the map we also observed this behaviour.

We assigned the poses on order of what we thought would be the most fitting application and were left with the double tap pose, so it became the reset aim direction action. Not the most intuitive, but still a useful gesture in terms of gameplay.

## 6.4 Haptic Feedback

Myo provides haptic feedback in the form of vibration with three vibrationTypes. We will be leveraging these to provide feedback to the player about in game events, like taking damage from the enemies, offloading projectiles from a weapon, and player death, adding to player engagement.

Haptic feedback is common in many games, and many games controllers have a vibration motor in them. It would be a shame not to put the Myo's vibration to work! The following is a breakdown of our approach to using the vibration as a form of player feedback, we coupled it with screen shake to further enhance the effect of the feedback to the player in certain situations.

### *General Outline of Haptic Feedback and Screen Shake*

#### *Player Takes Damage*

- Short - It's just a flesh wound..
- Medium - I see bones sticking out..
- Long - I feel dead..

#### *Player Fires Weapon*

- Short - I shot something!
- Medium - Did my belt just fire bullets!?!..
- Long - N/A.

## 6.5 Player Engagement

Considering fun in games is important to keep the player interested, invested and immersed in your game.

[8 Types of Fun in Games](#) identifies key points in this regard and explores different types of fun a game can deliver to a player. While Myo Isometric Shooter will incorporate most of the points made to some degree, the following are the key focus points:

### **6.5.1 Sensation**

Game as sense-pleasure.

The player will get a sense of fulfillment and enjoyment from learning the game mechanics during the early stages and then being put through their paces as the game progresses. Haptic feedback may also play a role in this regard as the Myo has vibration available, and will alert the user to having been attacked.

### **6.5.2 Challenge**

Think outside the box to complete the level.

Challenge will be core to the game feel and design. The player has a gun.. but it doesn't shoot bullets. To defeat the enemies the player will need to throw objects at them.

## 7 Test Plan

Throughout the development process, visible errors will be fixed as they arise. However, slight errors in obstacle (or other entity) behavior can be overlooked while testing for overall functionality, and then after “playing a round” of the game, those behavioral errors can be tweaked until they are appropriate. Big errors that offer no immediate solution to the programmers will be documented for future solving and can be discussed during development meetings for potential solutions or workarounds.

Optional beta-testing period: Play the game and attempt to ‘break’ the game essentially, trying to do things that should not be possible in the game. EG. Attempt to run through obstacles or move outside the ‘track area’.. Tap or click within game area to ensure no false tap/click behaviours exist.

The software will be deemed good enough to deliver after it is noted that the character behavior is consistently appropriate each time the game is run for at least 5 test-runs of the game/rounds as well as having the behavior of all visual components (Panels, text boxes, etc.) involved with the Graphical User Interface (GUI) be consistently appropriate for all tests of the game.

## 8 Conclusions and Recommendations

### 8.1 A Gesture based Myo project

We had numerous ideas in terms of which direction to take this project.

- As remote control.
  - Arduino Robot.
  - Fly a drone.
  - Robotic Arm.
- Slideshow controller with 'Slideshow' application.
- Design a game
  - Plane game. Control a plane as it flies through a procedurally generated environment. Dodge obstacles.
  - Crane game. Manipulate objects, pick up, winch up/down, rotate, drop, place into appropriate spaces (square block in square hole etc).
  - Shooter game. Control a 'man with a gun' character in a procedurally generated, complete by defeating waves of enemies through a series of randomized levels.

Given that we settled on building a game and using the Myo as a controller, we are confident that we produced an ideal test-bed for implementing gestures in an informed and intuitive manner. We tested the Myo's viability as a games controller. We made appropriate use of gestures and reasoned that the gestures we used were fit for purpose in our game. Throughout our project we researched our decisions as they related to both user experience and game play controls.

As we have shown in our research, there are many papers written about the use of appropriate gestures in games development, and we sought to leverage this and implement it in a meaningful way. Additionally there are papers written about other uses for the Myo armband, which were a valuable source of insight for us during this project.

### 8.2 Myo as a game controller.

The Myo has a lot of promise as a game controller, the scope for developing interesting game mechanics using one or multiple armbands has a high ceiling, and is a lot of fun to think about. As with all hardware it's a demanding development process and presents challenges with configuration and implementation. Myo's gyroscope allows angular tracking in 3D space, while it's EMG sensors recognize 5 unique hand gestures.

We discovered limitations to using the Myo as a controller during development.

Overloading the player with controls for the game is a concern, and this can be demonstrated by removing the keyboard input and relying solely on the Myo as a controller. During early testing and talks about our control scheme we realized that the player would need more than just the Myo in order to navigate around the in game terrain and also react to enemy behaviour. We decided on having keyboard input for moving the player left/right/forward/backward as this can easily be achieved with the players free arm.

The Myo will occasionally recognize gestures while no gesture has been performed by the user. During gameplay this becomes an issue as player character actions are unexpectedly fired, perhaps causing distraction, breaking immersion, and being a general nuisance for the player. Note that on occasion the firing of an unexpected action can be helpful if it's destroys your enemies.. But the broader concern is when using the Myo in a more serious manner, where unexpected gesture recognition might cause injury or obscure test results. A minor limitation of the Myo, though this applies to anytime the Myo is being used, is that it need to warm up so the sensors can accurately detect your hand gestures. This is not the most ideal scenario when it comes to playing games. Generally games are more suited to 'pick up and play' type controllers, with no down time while the player waits for the controller to warm up or sync or configure. This is more of an observation however, and doesn't impact on the Myo performance as a controller. Once it has warmed up, it requests a re-sync and is good to go.

Another minor concern which partly falls into the limitations is that if the user moves the Myo armband on their arm, the armband must re-sync to the users machine. Generally this might not be an issue, however as we discovered during development, wearing the armband for extended periods of time can become a little uncomfortable, require moving the Myo a little bit to relieve pressure on the arm, and hence a re-sync. This interrupts the flow of gameplay and along with the aforementioned factors could discourage engagement with the Myo as a controller when there are other means available.

### **8.3 Future development ideas for this project.**

Two Myos!

With the limited development time available to us for this project, implementing two Myos as controllers was a bit outside of what we thought achievable. However it would make sense that the next stage of development for this project might include a secondary control scheme for the second Myo.

With two Myo's, the player could control the directional movement of the player using one arm. For example:

Wave In/Out for left right movement, Fingers extended and Fist for forwards/backwards, or actually moving the movement Myo arm forwards/backwards. There are many more options available with the second Myo armband.

### **8.4 Final Thoughts**

We are confident that, while there are some limitations to using the Myo as a game controller, it is still feasible. Tweaking the configuration of the controller takes a little more time but the results are encouraging once some time and consideration are spent here. It's normal to spend time balancing inputs for games so this is nothing serious. It makes for a fun and engaging experience, provides immersion, and the possibilities of coupling Myos with VR units would make further immersion achievable.

Aside from games development, during research we discovered many other avenues for development with the Myo. Arguably the most appropriate use of the Myo armband is in the medical field where it is used in numerous rehabilitative applications, from tracking arm movement to assisting people with inoperable limbs to operate remote robotic arms. The applications in this area are profound and life changing for people.



## 9 Specification

Unity Game engine to run development build.

Windows/MacOS/Linux - The Myo sdk is available for each of the above Operating systems.

Myo Armband

## 10 References

[Thalmic Labs' Myo Gesture Control Armband](#)  
[Myo SDK and resources](#)  
[Myo Marketplace](#)  
[Technical Features and Functionalities of Myo Armband](#)  
[A Wearable Rehabilitation System to Assist Partially Hand Paralyzed Patients in Repetitive Exercises](#)  
[MYO Armband for physiotherapy healthcare: A case study using gesture recognition application](#)  
[Microsoft Kinect](#)  
[Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor](#)  
[Leap Motion Controller](#)  
[Gesture-based Interactions in Video Games with the Leap Motion Controller](#)  
[Unity Engine](#)  
[Godot Engine](#)  
[Github repository for this project](#)

### 10.1 Reference Images

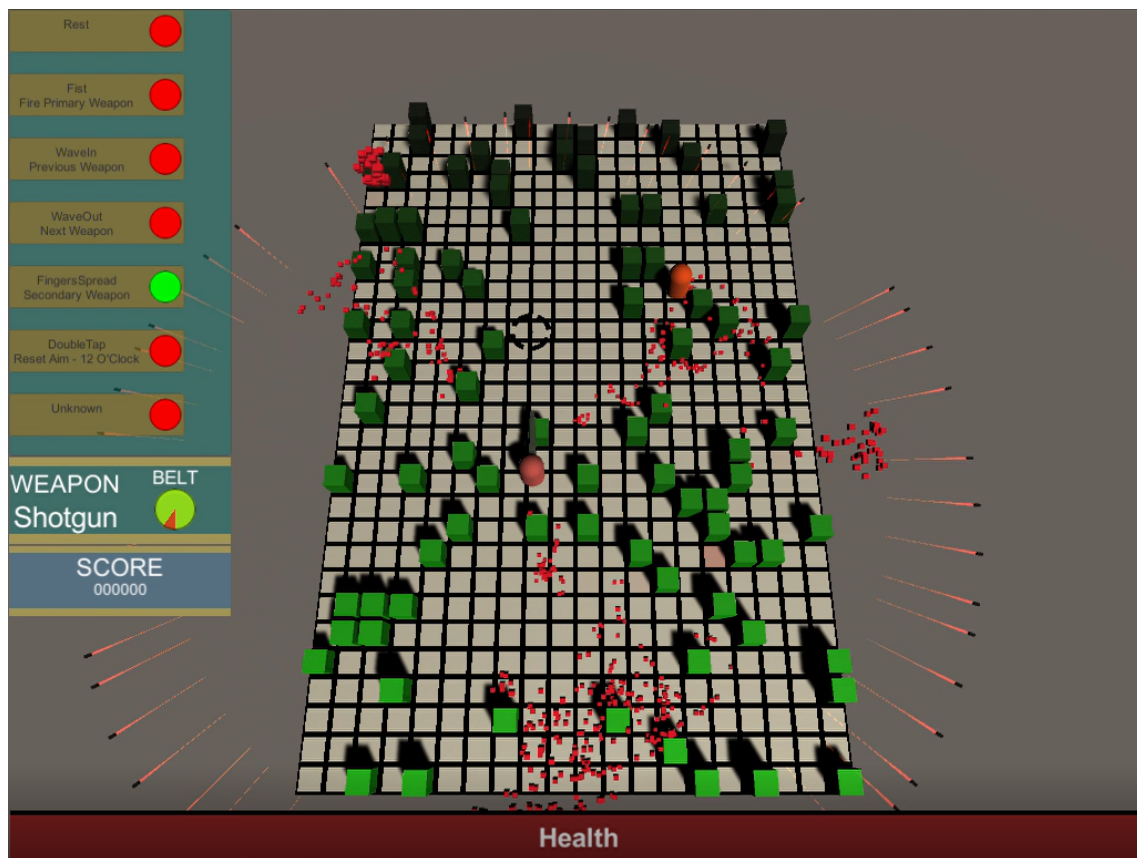


Figure 10.1: Firing the Secondary Weapon

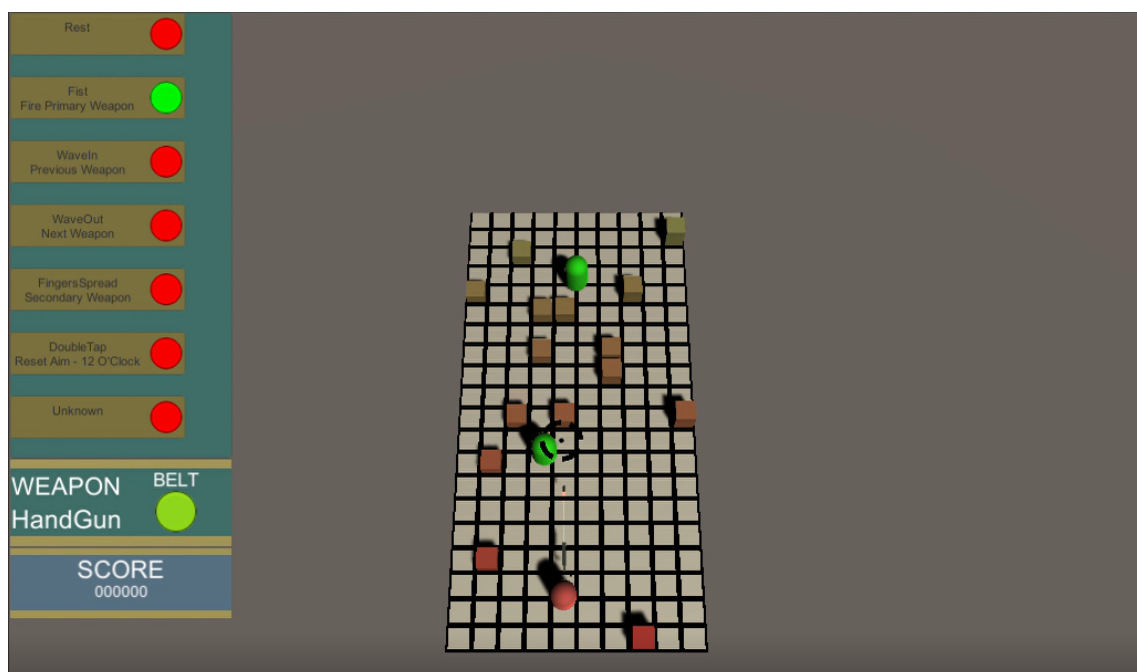


Figure 10.2: Firing the Primary Weapon

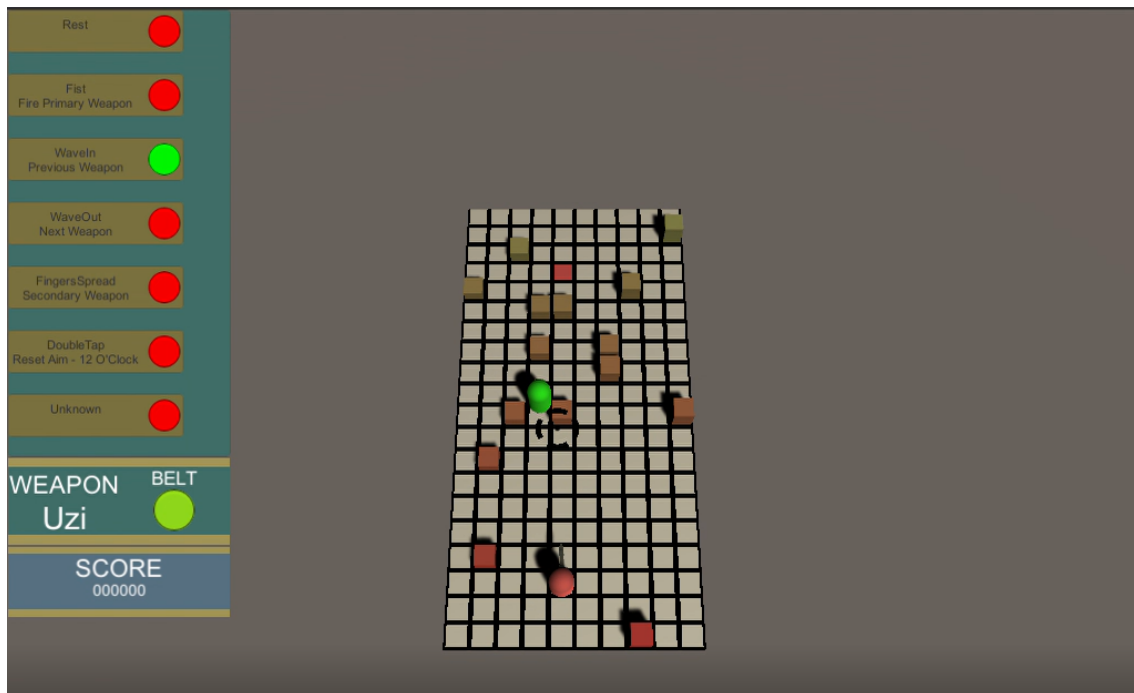


Figure 10.3: Switch to previous Weapon

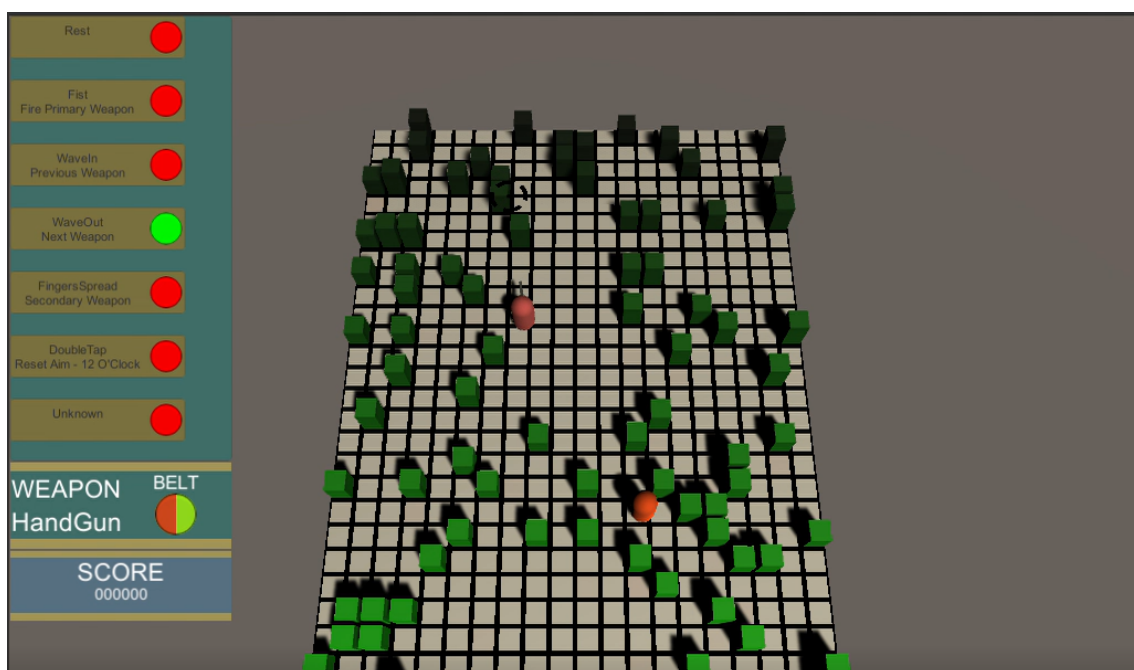


Figure 10.4: Switch to next Weapon

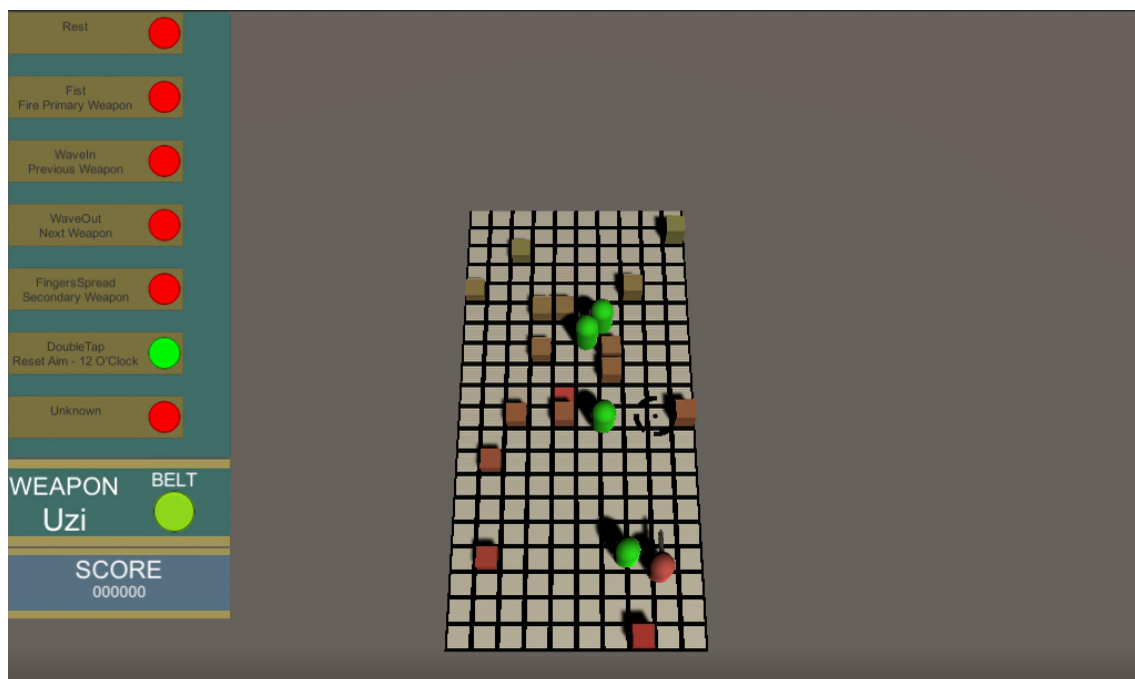


Figure 10.5: Reset player Aim direction