CMP302 report

# Summary

## Purpose

This project is a simple framework for an archery game which could be extended to anything from a first-person shooter, MMO, RPG or any other type of real-time action game.

Components:

Bow – The bow gets attached to any character controller class. it contains a charge, reload and release mechanic, spawning an arrow every time one is fired from the quiver.

Arrow – When the bow is reloaded a new arrow is spawned, the arrow stays attached to the bow until the left mouse button is pressed at which point it becomes a projectile. After hitting something the player can pick it up again.

Target – This component is responsible for checking if it was hit by an arrow and if so, it will increment the score appropriately.

UI system – Very simple UI responsible for showing the current arrow count and current score.

# Requirement Specification

## Introduction

The purpose of this project was to create a simple implementation of a bow weapon seen in some action games with a focus on making it easily extendable by both programmers and designers making further development quick.

## Product perspective

The goal of this project was to deliver a working bow and arrow with a target to shoot at to demonstrate the system.

## Product function

1. Allow the user to charge the bow
2. Allow the user to fire an arrow with the velocity being determined by the current charge
3. Allow the user to reload the bow after an arrow is fired
4. Allow the user to pick up the arrow after it has landed
5. Allow the arrows to attach to any solid object it hits
6. Allow the user to practice with the implementation of a target
7. Allow a designer or programmer to derive from the existing bow and arrow and iterate on the current systems allowing them to add things like new arrow types.
8. Allow the designer or programmer to easily edit data relating to the amount of time required to fully charge the bow

## User classes & characteristics

Either a programmer or designer should be able to extend the created functions with relative ease as they are all linked to their corresponding blueprint allowing for simple iteration.

A programmer should not be required to extend the already existing functions however implementing new mechanics and arrow types might require some minor code depending on the complexity.

## Design & Implementation constraints

There is no artist assigned to this project. All models and animations have been made by the developer.

# System Features

## Bow

**Description:**Allows the user to reload, charge and fire the bow. It stores the current ammo count and will only reload if the player has arrows.

**Stimulus / response sequences:**For gameplay, hold right click to charge bow, left click to release the arrow.

For development, the bow can be accessed through blueprint editor and the c++ code will open Visual Studio**.**

**Functional requirements:**REQ1: Reload Arrow  
Creates a new instance of arrow and spawns based on the position of the socket on the bow, decrementing the ammo count.

REQ2: Charge Bow  
Holding the right mouse button charges the bow over 1.5 seconds, letting go will return the bow back to its original state.

REQ3: Fire Arrow  
Pressing the left mouse button will fire the arrow with a velocity based on what the bow’s current charge is.

REQ4: Arrow Count  
The bow should only reload if the player has at least 1 arrow.

## Arrow

**Description:**The projectile that gets fired by the bow.

**Stimulus / response sequences:**  
For development, the arrow can be accessed through blueprint editor and the c++ code will open Visual Studio.

**Functional requirements:**REQ1: Load Into Bow  
Assign to the arrow socket on the bow skeletal mesh.

REQ2: Fire Arrow  
The arrow will have a velocity applied based on how much the bow was charged before it was fired, once fired it also interacts with physics and has gravity applied to it.

REQ3: Detect Collision  
Once the arrow is in the air it will check if it collides with anything and if so, it will attach itself to the actor hit.

REQ4: Pick Up Arrow  
Once the arrow is attached to something it can be picked up by the player.

## Target

**Description:**Detects for collision against other actors and if it finds an arrow it increments the score based on how far away from the centre it is.

**Stimulus / response sequences:**For development, the target can be accessed through blueprint editor and the c++ code will open Visual Studio**.** The multiple target actors can be placed using the Unreal editor.

**Functional requirements:**REQ1: Check for collision  
Check if an arrow has collided with the target.

REQ2: Increment score  
Check the distance from the centre of the target and the point of contact with the arrow and increment score based on that distance.

## Player Character

**Description:**Derived from the example first person character provided by Unreal and extended to suit the needs of the project.

**Stimulus / response sequences:**For gameplay, The controls have been stripped down somewhat allowing for movement, jumping and rotating the camera. Left click and right click are used to call functions inside the bow blueprint.

For development, the bow can be accessed through blueprint editor and the c++ code will open Visual Studio **.**

**Functional requirements:**REQ1: Attach Bow to Player  
Allows the bow to function by attaching it to the character. This can be easily extended to accommodate for more players in online play.

REQ2: Handle Player Input  
Calls the appropriate functions based on input actions (Fire button, Charge button).

## Test Map

**Description:**Test area for the player to test the bow.

**Stimulus / response sequences:**For gameplay, the player can walk around the area but is constrained within a small box.

For development, the scene will be open in the unreal editor.

**Functional requirements:**REQ1: Targets  
Test area must have targets set up for the player to shoot.

## UI

**Description:**Simple UI to display the player’s score and ammo count.

**Stimulus / response sequences:**For gameplay, the ammo count and score will be displayed in the bottom right corner.

For development, the unreal widget editor will open.

**Functional requirements:**REQ1: Display ammo count  
Display the number of arrows left for the player to fire.

REQ2: Display score  
Display the score the player has achieved through testing.

# Method

## Player Character

The player character was derived from the existing sample FirstPersonCharacter blueprint and modified to suit the requirements.

When the player actor is spawned, a bow is attached to the camera component and the UI widget is added to the viewport. Input actions are also added to accommodate for Fire and Charge.

Three functions exist inside the player character:

* Add Arrow: called when the player picks up an arrow off the ground and adds it to the ammo count.
* Reload Bow: called after a 0.3 second delay following an arrow being fired. It checks if there are any arrows left and decrements the ammo count and calls the ‘reload’ function inside the bow blueprint.
* Increment Score: called from the ‘Target’ blueprint, incrementing the score based on how close to the centre the arrow hit.

## Bow

One of the functional requirements for this project was to have a functioning bow. The bow class is responsible for ‘listening’ for events being called.

**Begin Charge:**A timeline will start incrementing the ‘chargeAmount’ variable which is used to determine the velocity of the arrow when it is fired.

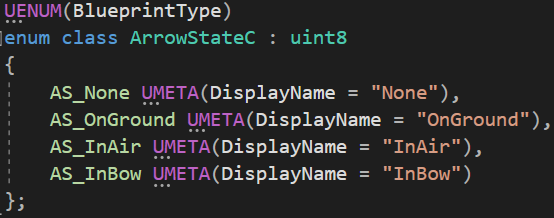
**StopCharge:**Stops the timeline and resets it back to the default value. Called when the arrow is fired, not when right click is let go.

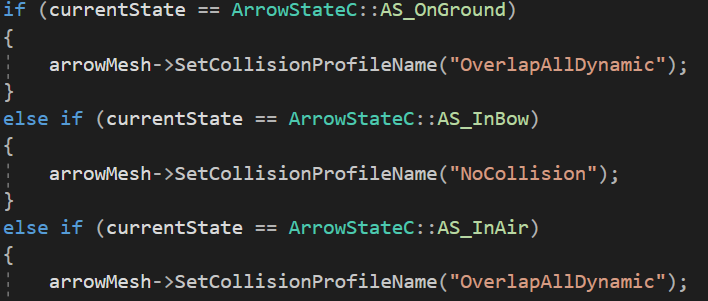
**EndCharge:**When right click is released without the arrow being fired it will reverse the timeline giving the illusion of ‘unwinding’ the bow.

**Fire:**Called when the arrow is fired, it detaches the arrow from the player actor and gives it a projectile component after which the ‘Fire Arrow’ function inside the arrow blueprint and set the current arrow attached to the bow to be NULL followed by calling the ‘Stop Charge’ event.

**Reload:**Calls the ‘Reload’ function which spawns a new arrow at the socket position on the bow skeletal mesh which allows it to move with the bow being charged.

## Arrow

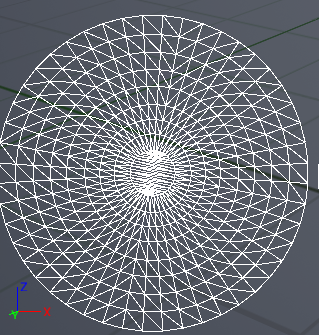
Stores the current state as an enum which is responsible for determining what the arrow should be doing. Switching states also applies different collision behaviours.

* AS\_None is the default state when the arrow is initialised
* AS\_OnGround for when the arrow hits something and gets attached
* AS\_InAir means the arrow is currently flying
* AS\_InBow means the arrow is ready loaded into the bow and will be the next one to be fired

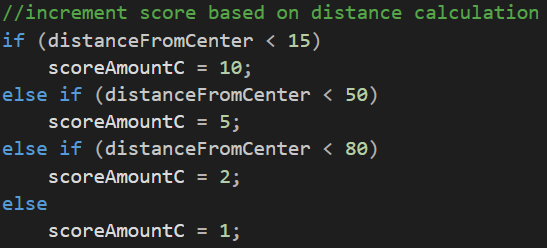
The arrow has a projectile component which has the gravity and velocity set to 0 until it enters the AS\_InAir state and is fired from the bow.

When fired the arrow will switch to the corresponding state and set the velocity to its’ forward vector multiplied by linear interpolation (Lerp) from 100 to 4000 based on the current bow charge.

## Target

The target actor is responsible for listening for collisions with and arrow actor and determining how far away it landed from the centre. This is done by setting a variable for the centre position of the target followed by a simple equation which returns the radius from the point of collision to the centre point. Based on that radius the score is incremented.

The collision point is calculated based on the Y and Z axes; this means that the target face must be pointing along the X axis. A line trace implementation was attempted however this didn’t work with the model used therefore this implementation, while not ideal, was chosen as it demonstrates the core functionality of the application.



# UML Diagram

# Conclusion

To conclude, I felt the project went well, the main issues were caused by unreal’s hot reloading which lead to many crashes and bugs which forced me to compile from visual studio most of the time. The project itself was challenging but it gave me a much better understanding on how game engines work and the workflow involved in creating an application using one.

## Problems

The first big setback was that I made everything using blueprints first and only started coding after. This exposed some flaws with my design, having to go back and overhaul a few functions to suit the requirements set.

The second issue was that there are some functions that didn’t get converted from blueprint to code as there were some dependencies which required even more changes.

The third issue was the line trace I wanted to implement but had to ultimately scrap as for whatever reason my UV channel did not work properly on the target model.

## Positives

I went into this project with a clear plan in mind, having laid out everything in a simple design document before I started helped a lot with code structure.

I also learned a lot throughout the process, now being able to use unreal for other projects which will speed up development drastically.