Newton's Bounty - Coursework Report

Grant Anderson, Paulius Bieksa
40218994@live.napier.ac.uk, 40216180@live.napier.ac.uk
Edinburgh Napier University - Games Engineering (SET09121)

Keywords – Newton's Bounty, Napier, space, duel

1 Introduction

The aim of this coursework was to design, implement and evaluate a simple prototype game using C++ and SFML. The game in question is a top down spaceship battle game for 1 or 2 players. Players will control a single ship, momentum will carry the ship as it would in outer space (for example, to come to a halt the player would need to fire thrusters in the opposite direction of movement). The objective is to destroy the other ship. The ships will have broadside cannons as a main means of doing that. Ships are able to counter incoming missiles with a dedicated turret also controlled by the player. This results in a hectic scenario where the player must consider their ship's position, velocity, defence and offence all at once.

Inspiration One of the main inspirations for this game is Battlefleet: Gothic Armada. The spaceships in that game have a feeling of weightiness that our game aims to replicate. Battlefleet: Gothic Armada also inspired the thruster system in Newton's Bounty with it's special maneuvers.

The repository for the code can be found at: https://github.com/Granderson89/games_eng_cw

The promo webpage can be found at: https://grantanderson.bitbucket.io/portfolio-newtonsbounty.html

2 Changes from original game design document

The scope of the game has been reduced slightly. The warp jump ability is not in the current feature list, as well as procedurally generated environment topology. The movement system has been streamlined - instead of having simple movement controls and the thruster system, Newton's Bounty only uses the thrusters to propel the ship. This reduces the number of buttons/keys required for full controls and also does away with the energy mechanic. The default control scheme has also been updated.

3 Software design

AI State Machine In order for an AI to be competitive it needs to be able to respond differently according to the situation. Our AI needs to be able to attack, chase and flee from the player. The state model displayed in figure 1 shows these states and the conditions for the transitions between them. When the AI is initialised it moves into the Face and Shoot state which turns the ship so that the cannons are facing the player and fires weapons constantly. If the player gets too close or the AI's health is too low, it will transition to the Flee state to avoid taking too much damage. This moves the ship away from the player and turns so that the presented cross section is as small as possible making it more difficult to hit. While in this state it will continually fire missiles. When the optimum distance is reached it will transition back to Face and Shoot. If the opponent becomes too distant, it will transition to Seek, moving the ship closer to the player, again, presenting the smallest cross section. When optimum distance is reached it will transition back to Face and Shoot. Seek can also transition to Flee if the AI's health drops too low.

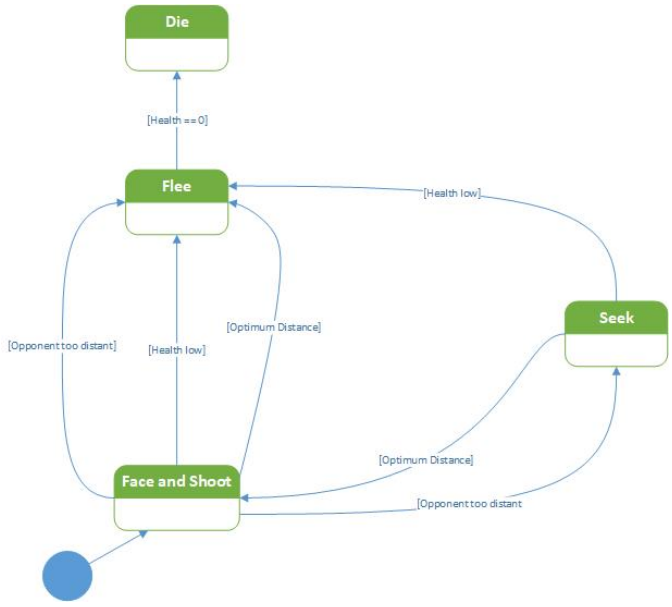

Figure 1: **AI State Machine Model** - Modelled behaviour of AI

Weapon Firing Sequence One of the most important interactions in the game is the firing of projectiles and the collision of that projectile with the enemy ship. The se-

quence diagram in figure 2 shows the events that lead up to a projectile either making contact and damaging the enemy or reaching the end of it's lifetime. The Weapon Components are created at the start of the scene. When the player fires, it calculates which direction it must fire in (port or starboard), then creates the appropriate projectile, informing it of the direction and plays it's weapon launch sound. The projectile applies an impulse to itself based on it's speed and direction then continually checks for collision with the enemy ship. It will either collide with, and therefore damage, the enemy ship and delete itself, or run out of lifetime and delete itself.

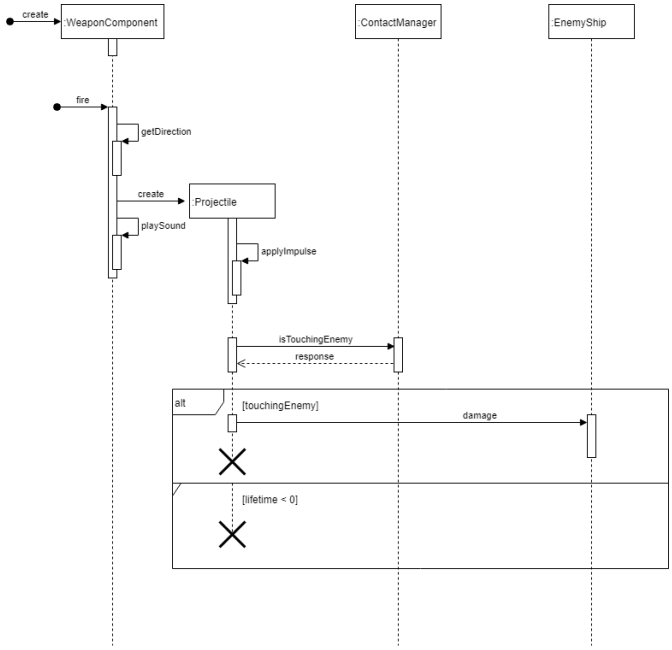

Figure 2: **Weapon Firing Sequence Diagram**

Input manager To prevent player controls from being scattered throughout different components an input manager class was created for this game. The class stores a list of player input values and a list of possible player actions. This separation ensures that player controls can be changed without changing any code. The input manager class also makes sure all of the player controls are in the same place. The class is static, which means it does not have to be initialised, while having it's action list easily accessible by any class or component that requires it.

Resource manager In a similar fashion to the input manager, the resource manager provides easy access to game assets to various components and classes. The assets are loaded once when the game starts up, which is only required before the resources can be used.

4 Implementation

The player ships are entities with multiple components attached. Each component manages a different aspect of the ships functions. The weapon components deal with firing the correct type of projectiles when necessary and updating the weapon sprites on the ship when a weapon change is required.

Collisions Box2d collision category bits are used in order to ensure that a player's own projectiles do not damage or collide with them. These inform the collision system of the types that each object can collide with (players collide with each other and their opponents projectiles, projectiles collide with their target and the defensive turret projectiles, defensive turret projectiles only collide with their opponent's projectiles).

Two player camera SFML does not have an explicit camera class, so a component was created to facilitate the needs of the game. This camera works by calculating the center-point and distance between the players then setting the SFML view position and dimensions accordingly. The algorithm for this is fairly simple - it finds the horizontal distance between the players and compares it with the vertical distance modified by the aspect ratio of the camera, then a view size is calculated from the larger of the two values.

5 Evaluation

5.1 Comparison against original concept

The game is very close to the original concept. The major differences are the few features we were unable to implement such as the warp jump ability and the random asteroids and debris. The core objective was to create a hectic environment which requires skill to play well. This has certainly been achieved as the environment can quickly become filled with projectiles that the player must deal with in order to stay alive.

5.2 Comparison against other games in the genre

The implementation of the game captures that heavy movement feel of a massive ship similar to the games that inspired it (Battlefleet: Gothic Armada and Assassin's Creed IV: Black Flag). This game has more of an arcade style and so it seems more appropriate to compare it to similar games such as Towerfall. While on the surface Towerfall seems like a very different game, it creates a similar hectic playing experience, overwhelming players with tactical options every second.

5.3 Quality discussion

Due to the nature and time constraints of the project we were unable to source and fully polish the graphics. Therefore the look of the game is of a fairly low and mismatched quality. However, we concentrated on having polished game mechanics and these are of a higher quality. After some user testing we found that the game could benefit from further game balancing (for example, torpedos seem far superior to other weapons). There

are several preferences options that the user can customise. Some of the controls are not customisable as initially this did not seem to be sensible (for example changing movement on the controller). However, by the end of the project lifecycle, changes to mechanics were made which could open up the possibility of full customisation in the future.

5.4 Possible improvements

- Addition of warp jump feature
- Addition of asteroids and space debris
- The boundary looks quite out of place and is not always effective at keeping the players close. It would be better implemented with forces which push the ships closer together
- The torpedos and missiles could have an area of effect when they explode
- Interesting obstacles could appear in the game such as black holes, planets or wormholes
- Addition of shields
- Animations of projectile explosions

6 Summary of resources

The ships and thruster burns were designed and created by Arianna Law. All other assets were sourced from OpenGameArt.org.

- UI Menu Button - https://opengameart.org/content/sci-fi-ui-panel
- Plasma Cannon - https://opengameart.org/content/2d-spaceship-construction-kit
- Torpedo Pod - https://opengameart.org/content/2d-spaceship-construction-kit
- Missile Mount - https://opengameart.org/content/2d-spaceship-construction-kit
- Defensive Turret - https://opengameart.org/content/2d-spaceship-construction-kit
- Plasma Shot - https://opengameart.org/content/2d-spaceship-construction-kit
- Torpedo - https://opengameart.org/content/2d-spaceship-construction-kit
- Missile - https://opengameart.org/content/2d-spaceship-construction-kit
- Background - https://archive.org/details/Stars_2D
- Menu Music - https://opengameart.org/content/through-space
- Gameplay Music - https://opengameart.org/content/last-stand-in-space
- Plasma Shot Sound - https://opengameart.org/content/4-projectile-launches
- Torpedo Launch Sound - https://opengameart.org/content/4-projectile-launches
- Missile Launch Sound - https://opengameart.org/content/4-projectile-launches
- Thruster Sound - https://opengameart.org/content/space-ship-engine-sounds
- Explosion Sound - https://opengameart.org/content/bomb explosion8bit