CS4303 Video Games: Practical 3
Game: Flagship
Report

170004680

University of St. Andrews

May 2022

Contents

A Introduction

In this practical, an open-world, top-down, single-player, action-adventure pirate/naval-combat game has been developed using `Processing`. The developed game, referred to as '*Flagship*', uses a variety of technical concepts related to video games, which were explored both within the module and through further reading.

The following techniques are of particular note: 2D Perlin noise thresholding for random but persistent map terrain generation, camera view-point panning for infinite open-world exploration, force-based steering mechanics and decision trees for complex AI behaviours (i.e., seek, pursue, wander, and flocking), bin-lattice spatial sub-division for optimised collision detection, and sound effects for player immersion.

A.1 Usage Instructions

The game was developed using the `IntelliJ` IDE. As such, the game cannot be executed from within the `Processing` IDE; the source code is more in line with what the `Processing IDE` converts to when executing the game from its environment. Alternatively, the game is executed using the terminal from within the `src/` directory. The following commands are used to compile and execute the game:

- `javac -cp "../lib/*:." Game.java`
- `java -cp "../lib/*:." Game`

B Design & Implementation

This section details, justifies, and displays the design and implementation of the developed video game, *Flagship*. Where applicable, screenshot evidence is used to show the game in operation and illustrate its features.

B.1 Overall Game Design

This section outlines high-level design decisions which define the developed system as a video game.

B.1.1 Title

The title of the game is, '*Flagship*', named as such for two reasons. First, to highlight the goal of the game, which is to defeat the final boss (i.e., the flagship antagonist) that is ever-present and hunts the player. Secondly, to emphasise the game as a pirate/ship-themed naval-combat game.

B.1.2 Genres

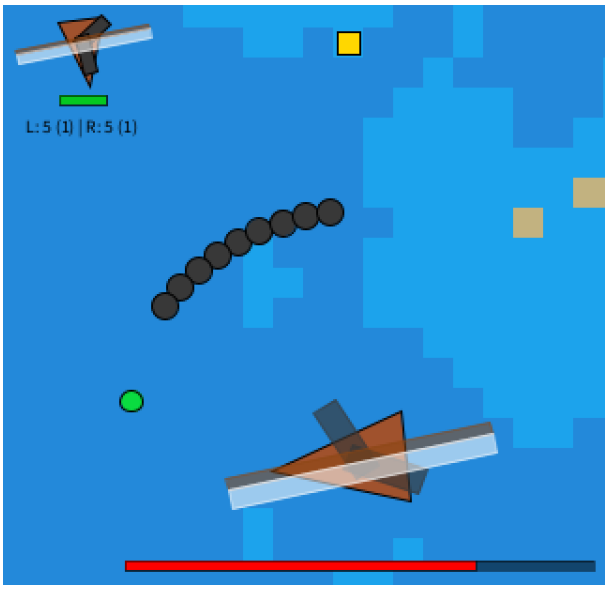
The developed game is associated with several video game genres:

- **2D/Top-Down**: *Flagship* is 2-dimensional with a top-down perspective; the player views the game from above with the x- and y-coordinates of the screen naturally assuming the x- and y-coordinates of an ocean-based map. Games of this type are also often referred to as having a bird's-eye, overhead, or helicopter view.
- **Open-World**: The player can freely explore the virtual world without fixed objectives. This is enabled via a combination of procedural content generation and camera viewpoint panning of the world map.
- **Click-To-Move**: Also referred to as 'point-and-click', this genre encompasses games where the main movement interaction between the player and their avatar is mouse-driven; the player uses their cursor to point and click on space within the play area to indicate movement intention. In *Flagship*, the player clicks on space where their ship avatar should move to, but this movement is influenced by various physics components such as wind strength and sail alignment.
- **Single-Player**: *Flagship* is played by one player interacting with and influencing the virtual world via a keyboard and mouse. Though, the game concept can be extended in many ways for multi-player with the implementation of network connectivity in future work.
- **Action-Adventure**: The player's main objective at any point in the game is either to defeat enemies in naval combat (i.e., action) or flee and explore the open-world (i.e., adventure) to collect loot for ship upgrades and defeat the final boss.

B.1.3 Goals

The overall goal of the game is to defeat the final boss - the flagship antagonist (Figure 1). Thus, the player wins the game when the final boss runs out of health. This final boss is ever-present and constantly hunts the player. Initially, the flagship is over-powered; the player cannot hope to defeat the final boss at the start of the game. However, the final boss is also slow. As such, the player is encouraged to explore the world, defeat enemies, and collect loot to obtain upgrades and progress through the game. It is only through world exploration that the player will be able to stand a chance at defeating the final boss and win.

Figure 1: The following screenshot highlights the goal of the game - to defeat the final boss. The flagship (bottom-right) is a large enemy ship that is difficult to defeat and pursues the player (top-left) throughout the game. In the screenshot, the flagship is demonstrating that it has a large spread-shot (i.e., 9 cannonballs per volley) and high range with which to attack the player.



B.1.4 Player

The player is represented in the developed game by an avatar. This avatar is a ship character the player can control. The player can point-and-click within the map to indicate where the ship avatar should move towards. However, ship movements are subject to various physics aspects, such as wind strength, sail alignment, and water drag. The player can mitigate and even take advantage of such effects by aligning their sail with the wind. The player can also fire volleys from their ship's cannons (Figure 2). Additionally, the player can purchase upgrades with collected gold (using a shop menu displayed when the game is paused) to affect their ship avatar. Upgrades consist of providing additional health, cannon volleys, cannon range, cannonballs per volley, and/or damage per cannonball (Figure 3).

Figure 2: The following screenshots illustrate various aspects of the character discussed thus far. In the left screenshot, there is no point-to-seek assigned for the player character to pursue. Despite this, the player ship is moving (speed is non-zero). This is because the player's sails are aligned with the wind, so the ship is being affected by (i.e., pushed) by a wind force. In the middle screenshot, the player's sails are still aligned with the wind, but a point-to-seek has been set by the mouse (green dot). As a result, the player ship enacts an arrival steering behaviour towards the point-to-seek. In the right screenshot, it is demonstrated that the player is able to fire cannons from their ship towards the aiming reticle. The cannon that is fired is determined implicitly given the location the player is aiming at.

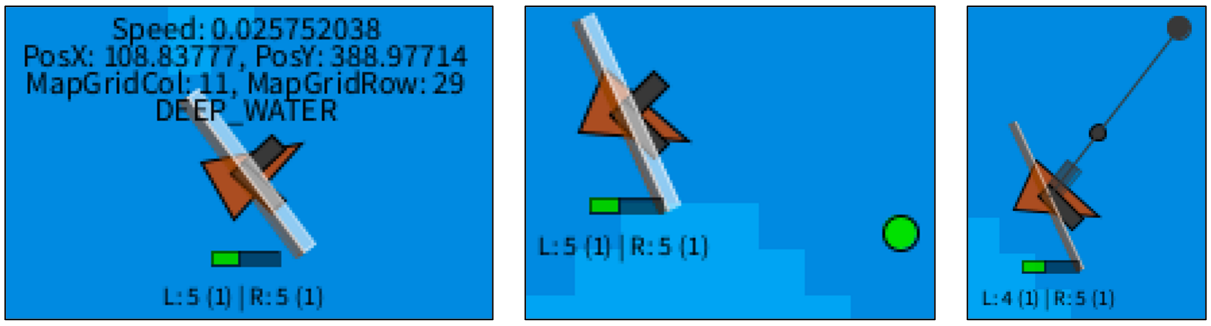
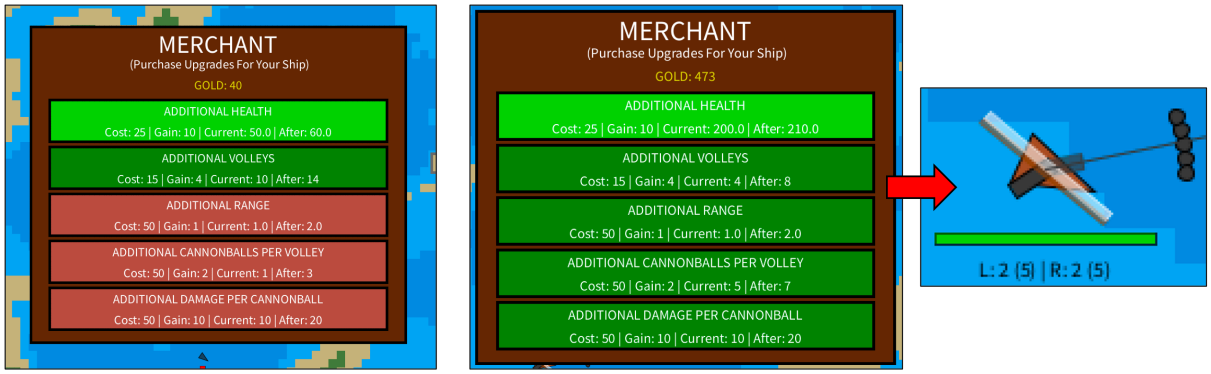


Figure 3: The following screenshots demonstrate the shop menu in action, which the player interacts with to purchase various ship upgrades. In the left screenshot, shop interactivity is highlighted. Upgrades the player can afford given their current amount of gold are green, whilst upgrades they cannot afford are red. The currently hovered upgrade option is brighter as a quality-of-life feature. In the two screenshots on the right, purchased upgrades are being shown in action. In the shop (middle), it's seen that the player has purchased significant health (they now have 200 health points), and they have purchased spread-shot such that their volleys fire 5 cannonballs. This is reflected by their ship health bar, the cannon status indicator below the health bar, and that there are 5 cannonballs being fired (right-most screenshot).



However, several aspects of the player that were presented in the pitch have not been implemented in the developed game for game balancing or implementation feasibility reasons.

First, the player freely upgrades various aspects of their ship avatar independently (amount of range, spread-shot, health, etc.) instead of simply upgrading the ship into pre-defined classes ('brig', then 'galleon', then 'man-o-war', etc.). This allows for a greater flexibility of gameplay, which improves re-playability as the range of possible play styles is increased.

Also, the player avatar is not increased in size as their ship is upgraded, which was initially planned to reflect the player's ship class. It was found that increasing the player's ship size increased game difficulty excessively; ship mobility becomes sluggish and the player gets targeted by undesirable amounts of enemies. Increasing ship size as player health is increased would provide a trade-off when upgrading that makes the game more complex, but this could not be game-balanced in a way that didn't make the increased ship size purely cosmetic - a feature of upgrading already handled by other indicators (e.g., health bar).

Furthermore, the player does not have access to other abilities, such as flame barrels, harpoons or special abilities. The player's only mode of attack is their cannons, though these can be upgraded in several different and meaningful ways. Additional abilities would make the game more interesting and is left for future work. Other game aspects were prioritised for submission, such as improving and adding enemy behaviours to make the open-world more interesting and challenging to explore, which is crucial for an adventure game.

B.1.5 Opponents

Several different types of enemy have been implemented, which introduces fun mechanics to increase the types of variety the player encounters (Table 1). These opponents can be categorised depending on whether they are: weather versus character, mobile versus stationary, boss versus regular, and contact- versus projectile-damage dealers.

Table 1: Table summarising the implemented opponents, and their associated behaviours, within the game.

Opponent	**Type**	**Motion**	**Damage**
Wind	Weather	-	-
Description			
The wind pushes the player's ship in randomly changing directions with differing strengths, which the player must account for. Also, the player has to maintain reasonable sail alignment with the wind for their ship to have mobility (Figure 2 and 9).			

Opponent	Type	Motion	Damage
Enemy Ships	Regular Character	Mobile	Projectile
Description			
If an enemy ship is aware of the player but the player is not in range, then the enemy ship will pursue the player with arrival. This means that once the ship gets sufficiently close to the player, it will start to slow down in order to maintain the player within cannon range and so that shots become increasingly accurate. If the enemy ship is aware of the player and the player is in cannon range, then the enemy ship will fire upon the player. Otherwise, when the enemy ship is unaware of the player, it will wander the map. Note that in all cases, enemy ships are affected by wind and water drag forces; enemy ships will always turn their sail towards the wind though not as immediately as the player is able to do so. Also, enemy ships will always manoeuvre to avoid land (Figure 4).			

Opponent	Type	Motion	Damage
Flagship	Final Boss Character	Mobile	Projectile
Description			
The flagship is always aware of the player and is, therefore, always pursuing the player (with arrival as described with the regular enemy ships). When the player is within cannon range, the flagship will fire upon the player (Figure 1).			

Opponent	Type	Motion	Damage
Forts	Regular Character	Stationary	Projectile
Description			
A given fort can possess between 1 and 4 cannons, which is determined at random. When the player is within the awareness radius of a fort, the fort will aim its cannons towards the player. If the player comes within range of one of the fort's cannons, then said cannon will fire at the player (Figure 5).			

Opponent	Type	Motion	Damage
Mega-Forts	Mini-Boss Character	Stationary	Projectile
Description			
A mega-fort possesses the same behaviour as regular forts except that it has greater health, a greater radius of awareness, and more cannons. Each mega fort possesses 4 large cannons with great range at each of the cardinal directions, and also comprises 4 regular forts at each of the corners of the mega-fort. Note that each corner fort can be individually destroyed to mitigate the danger of the mega-fort, but destroying the overall central fort irregardless will defeat the entire mini-boss (Figure 6).			

Opponent	Type	Motion	Damage
Sirens	Regular Character	Stationary	Contact
Description			
Sirens are stationary enemies within the water. When they are aware of the player they will look towards the player. With a random probability, a siren may also halt the player (i.e., nullify their velocity and remove the player's point-to-seek given by the mouse) and pull the player towards itself. The player must either kill the siren or repeatedly click away from the siren's area of effect to escape its pulling influence. If the player is touched by the siren, then damage is dealt and the siren 'swims away' (i.e., removed from the game) (Figure 7).			

Opponent	Type	Motion	Damage
Sharks	Regular Character	Mobile	Contact
Description			
Sharks are mobile enemies that wander the map area, avoiding land when necessary. When a shark becomes aware of the player, the shark pursues the player but with a lunging capability. Lunging follows the opposite behaviour to arriving; the closer a shark gets to the player (within its area of awareness), the quicker the shark hurls itself toward the player's ship. Sharks deal constant damage while 'biting' (i.e., colliding with) the player's ship, though the inflicted damage per shark is very small. However, sharks will also follow flocking mechanics when they discover each other, travelling as a group separately but cohesively. Larger collections of sharks attacking the player together are more substantial a threat to the player (Figure 8).			

Figure 4: The following screenshots illustrate the behaviour of enemy ships. Enemy ships are indicated by their red health bars. They are of comparable size to the player, indicating that they are regular enemies when compared to the flagship. In the left screenshot, the player is not within the enemy ship's field of awareness (brown circle), so the ship wanders with land avoidance. In the top-right screenshot, the player is within the field of awareness but not within cannon range (the dark-grey circle), so the enemy ship is pursuing the player. In the bottom-right screenshot, the player is within cannon range, so the enemy ship fires at the player.

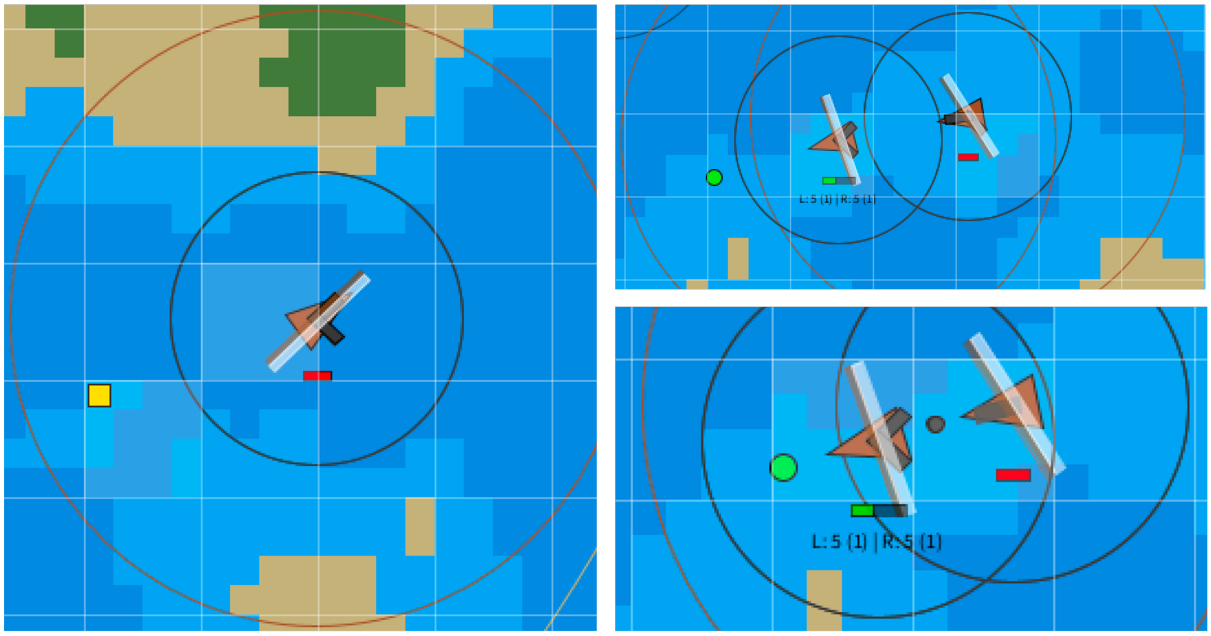


Figure 5: The following screenshots illustrate the behaviour of enemy forts. In the left screenshot, the appearance of forts is shown. Each fort consists of thick, grey walls with between 1 and 4 cannons positioned at the cardinal directions (3 in this case). In the middle screenshot, the player is within the awareness field of a fort but not firing range of its 2 cannons, so the fort simply aims towards the player. In the right screenshot, the player enters range of both of the fort's cannons, so both cannons fire at the player.

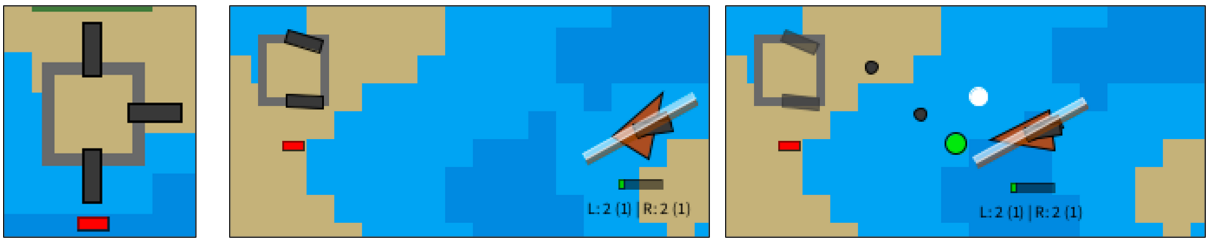


Figure 6: The following screenshots illustrate the behaviour of enemy mega-forts. A mega-fort is much larger than a regular fort, consisting of a large central fort with longer range cannons, and 4 regular forts positioned at its corners. In the left screenshot, the mega-fort's long range cannons are aware of the player and are therefore aiming at the player. The smaller cannons at the corners of the mega-fort are not aware of the player, so they are not pre-aiming at the player. In the right screenshot, the player enters within firing range of the south-most large cannon, so said cannon fires at the player. Also note that the player is now close enough to be pre-aimed by the south-east most corner fort.

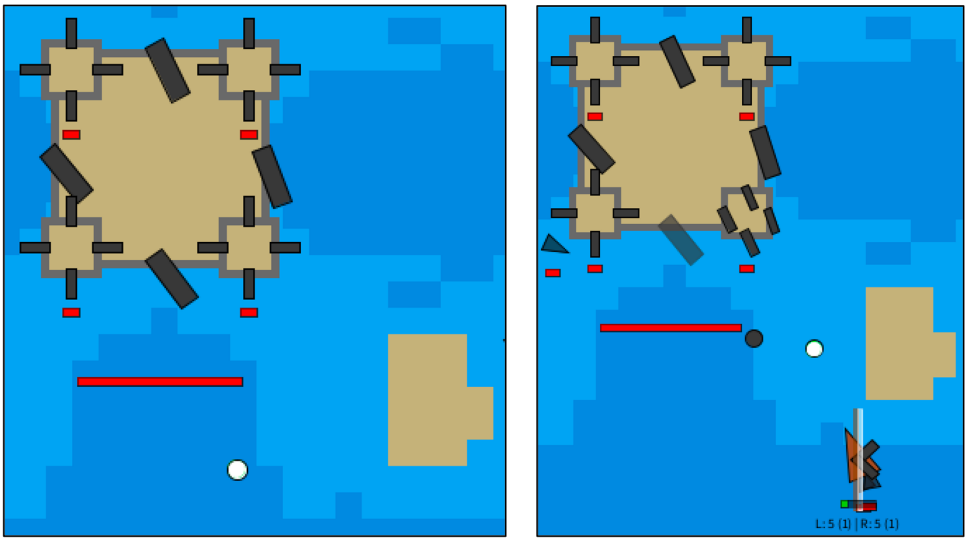


Figure 7: The following screenshots illustrate the behaviour of enemy sirens via a timeline of events. Note that sirens in the game are modelled as small turquoise triangles stagnant within the ocean. In the top-left screenshot, the player moves within the field of awareness of the enemy siren; the siren aims at the player accordingly. Then, the siren starts pulling the player towards itself (top-middle). The player must reset their point-to-seek repeatedly to attempt to escape (top-right). If the player cannot escape (bottom-left), then the player can try to attack and defeat the siren before being dragged too close (bottom-middle). If the player touches the siren, then the siren will deal damage to the player ship and swim away (bottom-right).

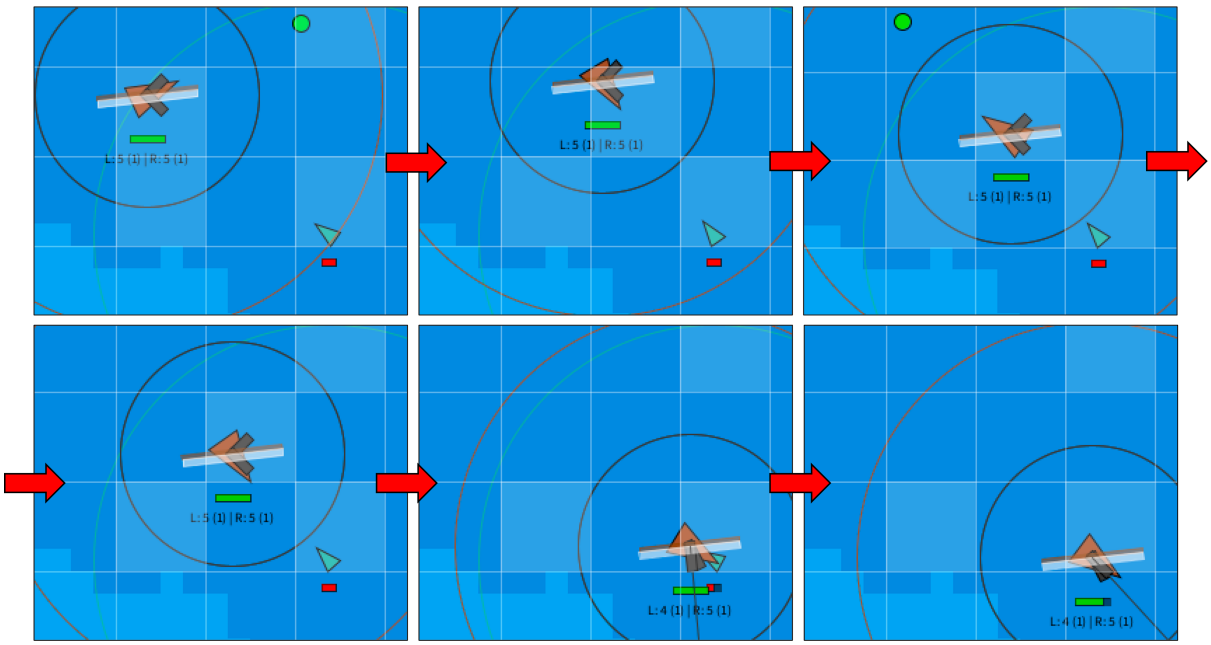
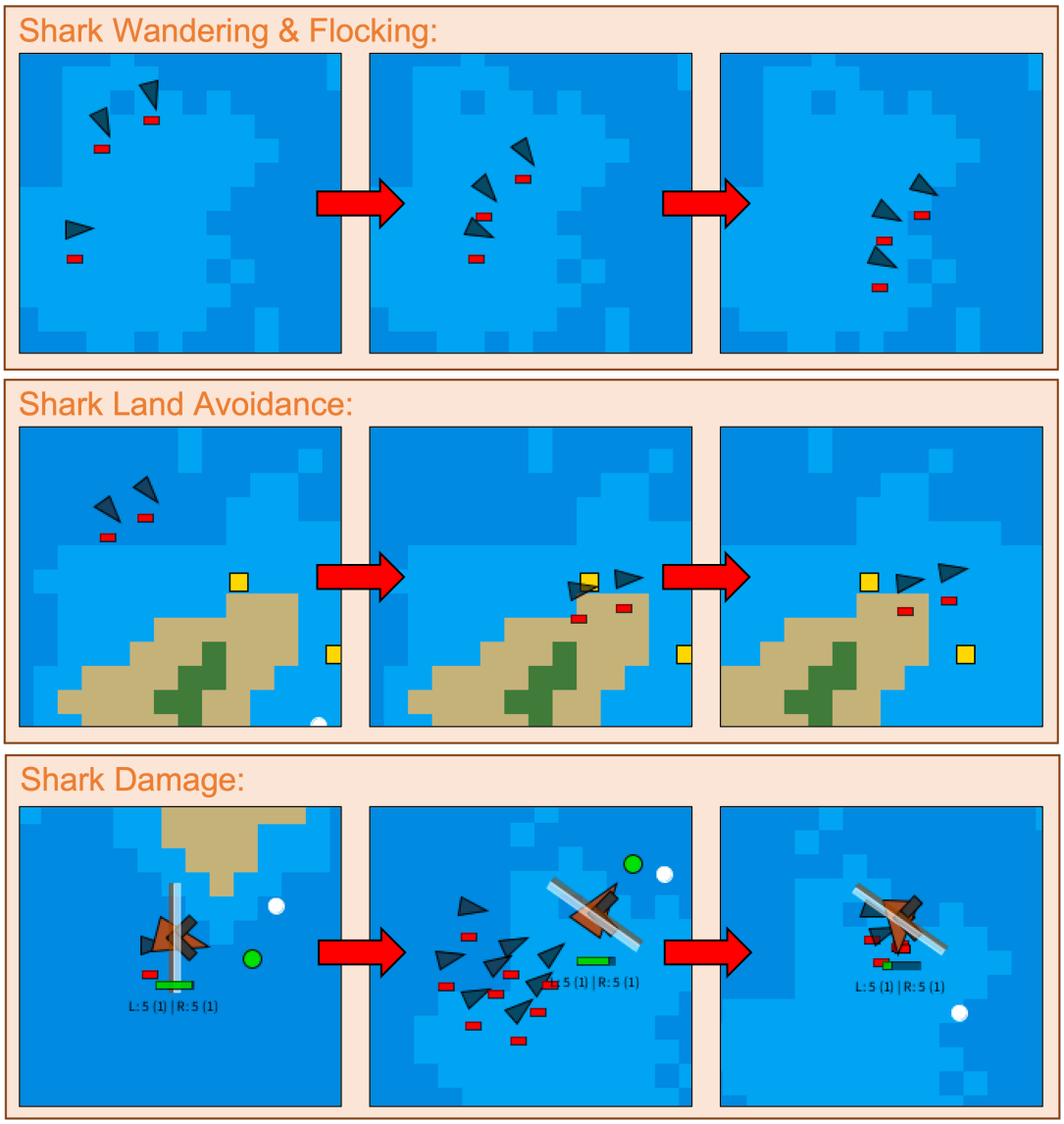


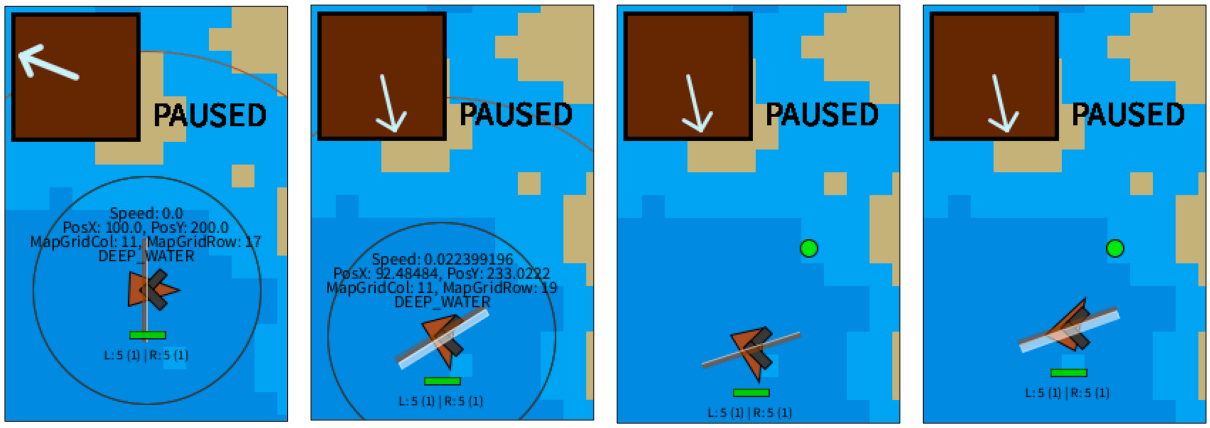
Figure 8: The following screenshots illustrate the behaviour of enemy sharks. In the top set of screenshots, it is shown that sharks wander around the map when they are unaware of the player and flock together when they meet with other sharks. In the middle set of screenshots, it is shown that sharks will steer to avoid land (though to a lesser extent than enemy ships so sharks come closer to the land). In the bottom set of screenshots, it is shown that a single shark will deal very limited damage to the player and is more of a nuisance. However, allowing a large shiver of sharks to build up and attack the player will cause substantial damage to be taken.



Weather Versus Character Opponents:

Weather opponents refers to the simulated weather conditions within the game that the player must overcome to progress. The weather is not a conventional enemy the player must defeat in combat, whilst the character enemies are. For example, wind has been implemented in the game with a randomly changing direction and strength. The player must align their ship's sail with the wind to be able to move; a higher degree of alignment provides a greater degree of mobility (Figure 9).

Figure 9: The following screenshots show the effect of wind within the game, as demonstrated on the player ship. Wind direction and strength is shown by an indicator in the top-left of the screen. An arrow points in the direction of wind, and the thickness of the arrow indicates wind strength. In the left-most screenshot, the player's sail is furled (i.e., opposite direction to the wind), so the player is not affected by the wind. In the left-middle screenshot, the player's sail is aligned with the wind, so the player's ship is moving even though a point-to-seek has not been set. In the right-middle screenshot, the player has set a point-to-seek (green dot) but the ship is not moving because the player has no alignment with the wind. In the right-most screenshot, the player's ship only begins to move once the player has also aligned their sail with the wind. Note that aligning a ship's sail with the wind is required for ship movement, but ships are not limited to move in the direction of wind (as would be the case in reality). A greater alignment of the ships' sail with the wind gives greater mobility. The degree of sail alignment with the wind is indicated by the size of the ship's white sail; a greater alignment means a bigger sail rectangle, as this is how a sail catching more wind would look from the top-down perspective.



Note that several weather opponents presented in the pitch have not been implemented in the game as other game aspects were prioritised for submission. This includes: whirlpools (which would differ to the sirens only in appearance), tidal waves (which would require a more rectangular instead of circular underlying game object geometry), and storms (which would require cloud generation in the map). Also note that ocean current simulation has not been implemented, though this would not present a significant challenge to implement in the future. A flow field can be generated using the 2D Perlin noise of the map grid. Each tile in the map grid can be assigned an ocean current heading and strength that ships would be affected by. This flow field would naturally flow around and away from generated land terrain, as desired.

Stationary Versus Mobile Opponents:

Stationary enemies are spawned to a fixed location within the procedurally generated map (i.e., forts and sirens), whilst mobile enemies use decision trees to enact several steering behaviours (i.e., ships and sharks). All mobile enemies wander the open-world or pursue the player based on their awareness of the player. A character's awareness is implemented by determining if the player's proximity is within a radius of awareness, which differs according to the type of character. In general, larger opponents have a greater field of awareness (note that the final boss is always aware of the player). The implementation of awareness within the game as being fully circular as opposed to a sector (i.e., 360° view instead of restricted file-of-view) is intentional as this is more natural within the game context. For example, the crow's nest of a ship allows viewing in all directions.

More complex combinations of steering behaviours can be enacted in certain conditions. Such combinations of behaviours are easily performed as the movement system is force-based, so simply adding the forces given by different steering behaviours enacts several objectives at once (via their resultant force).

For example, sharks wander but also follow a flocking behaviour, travelling in groups (known as a 'shiver') as they discover each other. Flocking is a steering behaviour which combines character separation, alignment, and cohesion such that characters operate as individual agents but appear to act as intelligent groups.

Additionally, all mobile opponents (which are water-based) will also enact land avoidance steering when they get close to any land. This is achieved via a ray-casting and terrain-sampling approach. Each character casts three rays to the edge of its area of awareness; one straight ahead of its orientation, and one one either side of this straight ray (clockwise and anti-clockwise) with a sufficient angle of rotation. By sampling along said rays, it can be determined if a mobile character is heading towards land. If this is the case, then the character steers towards the direction of the clockwise or anti-clockwise ray depending on whose samples result in fewer land terrain hits. Several samples are required along these rays to detect irregular shapes of land that would otherwise be missed by sampling just once or even a couple of times.

Boss Versus Regular Opponents:

Boss enemies present an especially difficult challenge for the player. Two different types of boss enemy have been implemented: mega-forts (mini-boss) and the flagship (final boss). These bosses serve as progress way-points within the game. Initially, bosses will be difficult to defeat, but as the player upgrades their ship, the bosses become increasingly easier to defeat. As such, bosses are spawned with a fixed difficulty (i.e., fixed damage, health, etc.). On the other hand, regular enemies are spawned frequently and scale with the player's ability in order to remain relevant throughout the game and present a constant challenge to the player. However, these regular enemies do not scale equally with the player, so the player will discern that they are getting noticeably stronger with successive upgrades.

Some forms of enemy presented in the pitch have not been implemented simply due to time constraints, but their implementation would not be difficult as future work. For example, whales and squids were outlined, and more difficult mini-bosses based on these enemies could be added (e.g., 'white-whale' or 'Cthulhu'). Also, armada patrols with differing affiliations (e.g., Spanish versus English) could be enacted by assigning ships to a country and allowing ships to follow a flocking behaviour similar to the sharks. Their decision trees could then be extended to allow them to attack ships of a different affiliation and not just the player.

Combat- Versus Projectile-Damage Opponents:

Finally, opponents will deal damage via contact with the player's ship or by launching projectiles (i.e., cannonballs) towards the player.

Sirens and sharks deal damage via contact but using different mechanics. Sirens randomly halt and draw in the player, dealing a set amount of damage if the player allows themself to be touched by the siren. This mimics the lore of sirens as possessing enchanting voices that draw in sailors, which would naturally include those operating the player's ship. Sharks deal constant damage to the player when colliding (i.e., biting at the player's ship) such that the damage of one shark is barely noticeable (i.e., a nuisance) but a shiver of sharks will present a significant draw on the player's health.

Alternatively, projectile-damage enemies (i.e., forts and ships) launch cannonballs at the player to deal damage. The difficulty of enemies that employ cannons is determined by the damage dealt per cannonball, the cannon range, and the number of cannonballs in a volley (i.e., how many cannonballs are in the spread-shot when a cannon fires). For example, forts and enemy ships have cannons that initially fire a single cannonball when fired but this scales with the player. The flagship fires many cannonballs per volley at the player in a spread which is difficult to dodge at game start.

To game balance the cannons, they possess a cool-down period between firings. A cannon which is on cool-down is given transparency to show it is not ready to fire yet. This mimics the re-packing, re-loading, and fuse-burning time of actual cannons. Also, cannons impart their velocity on the cannonballs they fire, which allows the game to produce more realistic cannon fires on the ships; cannonballs appear to travel down the barrel of the cannon that fires them. Interestingly, this means that ships are less accurate at hitting targets the greater their velocity. This affords the player the ability to dodge enemy projectiles. However, the player will also be required to develop a skill at pre-firing targets depending on their speed or they will otherwise have to sacrifice movement speed to hit targets, which will make the player more vulnerable.

B.1.6 Rules/Mechanics

There are many rules and mechanics present in the game, which are covered in the following list:

- The player is able to explore the open-world map by panning the camera, which is enacted by moving the cursor towards the edges of the screen. The camera view will pan in the relevant direction according to which side of the screen the cursor approaches.
- All characters have health, and are defeated once their health is depleted. Characters do not regenerate health within the game. When the player is defeated, they are re-spawned nearby on the screen but suffer a gold penalty (i.e., a portion of their gold is taken away as a consequence of dying).
- All characters are confined to their relevant terrain types. Thus, water enemies cannot go on land, and land enemies cannot enter the water. However, water enemies that hit the land do not suffer a health penalty; the land as an obstacle is viewed as providing sufficient challenge to the game.
- All regular (i.e., non-boss) opponents are procedurally spawned into the world as the camera view-point is panned. Regular opponents are despawned after a period of inactivity outside the bounds of the map at the camera view-point. Opponents are permitted to ignore physics, such as avoiding land, when outside of the map area so as to quickly become relevant to the player, which limits the amount of game resources being wasted on irrelevant game objects.
- All character opponents strictly seek combat with the player and do not fight each other; though, their projectiles can harm one another if the player is savvy enough to use this to their advantage. This was enforced as to limit the amount of characters that would be wastefully generated and killed by more powerful AI, providing little to no interaction with the player. However, allowing the AI to fight one another would create a more dynamic open-world (this was discussed prior with enemy ships possibly being assigned to countries that can combat each other).
- Ships are subject to wind and water drag when moving. The degree of alignment between a ship's sail and the wind direction determines how mobile the ship is at moving in a desired direction (e.g., in pursuit of the player for enemy ships, or towards the player's point-to-seek for the player ship).
- Ships simply need to align their sail with the wind to be mobile, but this does not deter the direction they can travel. It may be expected that the ships must travel in the direction of the wind, but this mechanic proved too difficult overall and needlessly restricted player choice at exploring the open-world.
- Ships possess a port-side and starboard-side cannon, which are implicitly fired based on the side of the ship a target is (relatively). Thus, the player does not need to specify which cannon to fire from, which would needlessly complicate the game interaction. Instead, the player simply aims at a target with the cursor.
- The player's cannons have a limited amount of volleys, whilst enemy cannons have infinite ammunition. All cannons have a range, a number of cannonballs per volley (i.e., spread-shot), and an amount of damage dealt per cannonball. Also, cannons have a cool-down period between firings.
- The player is able to upgrade attributes of their ship: amount of health, volleys, cannonballs per volley, damage per cannonball, and cannon range. However, such upgrades require gold to purchase.
- The player can collect loot by guiding their player ship over loot crates to attain gold. A random amount of gold (between minimum and maximum bounds) is assigned per loot crate collected. Also, a random amount of volleys will be assigned to the player's cannons.
- Cannons in motion (i.e., on ships) impart said motion onto the cannonballs they fire. Thus, cannonballs move as expected and down the barrels of the cannons that fire them, even when the cannon is moving. This replicates real-world physics and introduces a degree of inaccuracy and/or pre-firing that is required for ships at speed. Enemies do not pre-fire the player, making it easier for the player to dodge enemy projectiles (which is especially needed when spread-shot is involved). However, enemy ships will slow down as they approach the player, which causes them to become more accurate with their cannons.

- Cannonballs move as projectiles that are affected by drag. Once drag has slowed a cannonball sufficiently, the cannonball is removed from the game (as though the cannonball sank into the water/ground from the top-down perspective). Note that the implementation of gravity is not relevant given the game's bird's eye view; gravity is simply encapsulated with the drag force. When a cannonball collides with a character, the amount of damage the cannonball is assigned is inflicted on the character and the cannonball is removed from the game.

B.2 Game Interaction

This section details and justifies the suitability of interaction between the player and the game via the mouse and keyboard controls.

- **Ship Movement:** The game is of the 'point-and-click' genre. The player controls the location of an aiming cursor on the screen by moving the mouse. At any point, the player can left-click with the mouse (whilst no keys are engaged, so ship movement is the default action) to specify a desired location within the map that the player ship should move towards.
  The 'point-and-click' movement mechanic was used instead of the typical 'WASD' set-up as it was desired for the ship to be affected by forces that would decouple control from such keys. For example, wind may make the ship move forwards and left when the player is holding just the 'W' key, which makes the keyboard controls less meaningful.

- **Sail Alignment:** The player must align their ship's sail with the wind to gain mobility. The player can hold the '1' key to enter a sail alignment mode. Whilst this key is engaged, the mouse cursor (relative to the player ship) is used to specify a direction the sail is facing. When the player is satisfied with their sail alignment, they can simply release the key as confirmation.

- **Cannon Firing:** The player can fire their ship's cannons as a means of attack against enemies. Similarly to sail alignment, the player can hold the '2' key to enter a cannon firing mode. Whilst this key is engaged, the mouse cursor (relative to the player ship) is used to specify a direction to aim the cannons. Which cannon (port or starboard) is being aimed is implicitly handled based on the relative position of the cursor to the ship (i.e., if the aim is port-side of the ship, then naturally the port-side cannons should be fired). A left-click whilst aiming will confirm the current trajectory and fire a volley from the appropriate cannon. When the player wishes to stop firing their cannons, they simply release the '2' key.
  Note that the cannon firing mode requires a left-click to launch a cannon volley, as opposed to releasing the key (as is the case with sail alignment), so that the player can change their mind and not fire any cannons should they wish.

- **Shop Interaction:** The 'tab' key is used to toggle pausing of the game. Whilst the game is paused, a shop menu will be displayed. The player can use their mouse to hover over upgrade options within the shop. Left-clicking an option will purchase the corresponding upgrade, if the player has sufficient gold.

Development and debugging controls have also been enabled so that the features of the developed game can be verified easier. Typically, these options would be disabled to a standard player.

- Pressing the 'M' key will skip to the game over screen.

- Pressing the 'N' key will toggle a developmental view of the game. In such a view, the following elements are displayed: the bin-lattice spatial sub-division of game objects, the awareness radius of characters, the range of cannons, the size of the various game object data structures, and individual character attributes (e.g., speed, screen position, map grid location, and terrain under character).

- Pressing 'B' toggles pausing of the game but without display of the shop menu so that the state of the game can be better viewed at a particular time.

B.3 Implementation Techniques

This section seeks to detail key techniques that have been used in the game implementation which are of particular note due to their importance and/or technical complexity.

B.3.1 Procedural Map Generation & Camera Panning

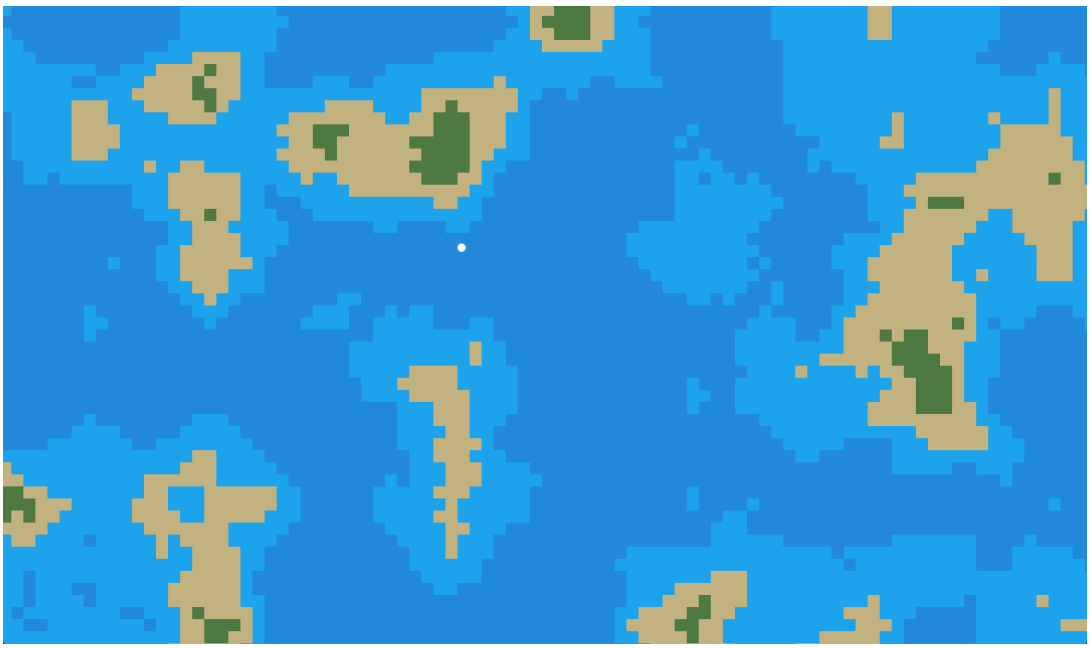
A crucial element of the game is the ability for the player to explore an open world. To achieve this, procedural terrain generation has been coupled with camera view-point panning.

Procedural Terrain Generation: The map is generated using 2D Perlin noise (Figure 10). Perlin noise has been specifically used in the implementation because it possesses several highly desired properties for the game. Firstly, Perlin noise has a more organic appearance compared to other noise generators because it produces a naturally 'smooth' sequence of pseudo-random numbers. This allows for transitions between terrains to appear natural. Also, noise values can be generated infinitely in the map plain. Though, more crucially, the noise value given at any particular coordinate is deterministic. This allows for terrain persistence within the game; if the player revisits a coordinate in the game, then they can expect to see the same landmark terrains as on previous visits.

The map is tiled as a grid, meaning that Perlin noise is not generated for every pixel in the screen, but instead for tiles of a pre-determined size (tiles in the game are 16x16 pixels). The size of the tiles was selected to balance terrain fidelity so the map would be interesting and not too 'blocky' in appearance, whilst also limiting the overhead required to generate and re-generate the map grid.

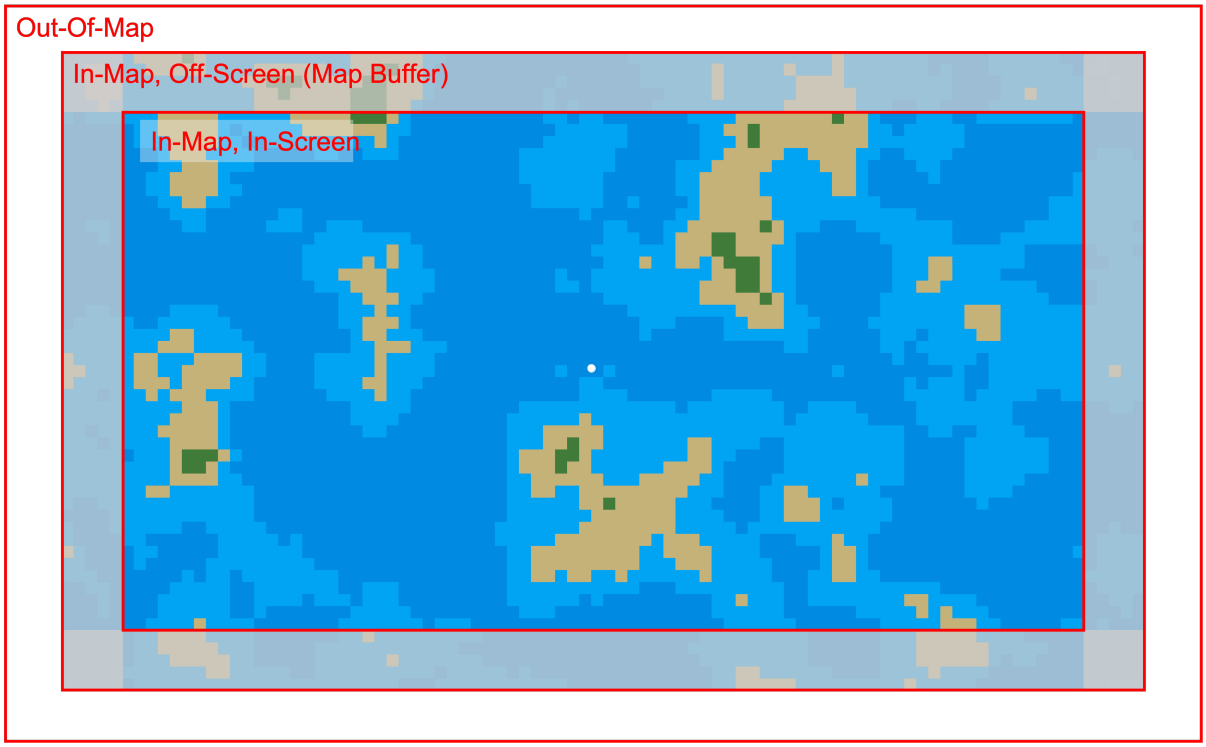
A terrain type is generated for every tile in the map grid using the Perlin noise value calculated from the coordinate of each tile. This noise value is essentially considered as a depth value, such that a value closer to 0 results in deeper water, whilst a value closer to 1 results in higher ground. In the game, there are 4 terrain types capturing this concept: deep water, shallow water, sand, and grass. These different terrain types have different threshold values that were balanced to produce aesthetically pleasing maps that are mostly ocean dominated since the game is naval-based.

Figure 10: The following screenshot illustrates map terrain generation within the game. The map is displayed as a grid of tiles. There are 4 different types of terrain assigned per tile: deep water (dark blue tiles), shallow water (light blue tiles), sand (brown tiles), and grass (green tiles). The terrain generation appears natural, with shallow water encompassing the land masses before receding to deep water.



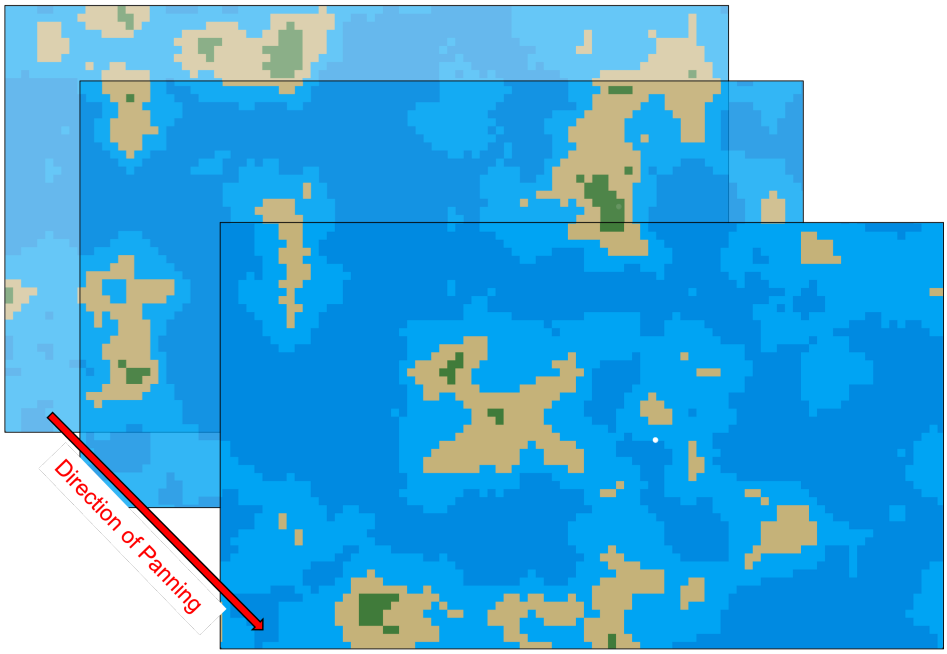
Furthermore, the map grid that is generated is not limited to the screen; there are buffer tiles around all edges of the screen. This creates three distinct regions concerning the game map: in-screen (i.e., on-screen, so clearly within the map grid), in-map (i.e., off-screen but within the map grid), and out-of-map (i.e., off-screen and out of the map grid). By including buffer tiles around the edges of the screen, artefacts caused by panning the camera view-point and generating new map tiles are non-visible, and game objects can spawn seamlessly and interact with the player's camera view more naturally (Figure 11).

Figure 11: The following illustrates map generation with the game as having a buffer area around the screen.



Camera Panning: To allow for open world exploration, the player is able to pan their camera viewpoint to discover new areas of the world (Figure 12). The player's viewpoint is centred within the map grid, so tiles at the edges of the screen are always surrounded by a buffer space of generated map. Thus, panning consists of generating new tiles along the map edge in the direction being panned and shifting all other tiles in the map grid accordingly. For example, if the player pans upwards, then a new first row of tiles is generated in the map grid, and all other rows of tiles need to be shifted down a row (the last row is subsequently shifted out of the map grid). In the implementation, this process is carried out more simply by re-generating the map grid for new coordinates resulting from a pan, with the slight optimisation that map re-generation need only occur when the pan is greater than one tile in distance.

Figure 12: The following illustrates map panning within the game, which is enacted by moving the cursor towards the edges of the screen. In this case, panning is being used to explore the map in the south-eastern direction.



Furthermore, game objects need to be shifted in accordance with a pan to maintain their map position. For example, a stationary fort should be seen to move with the land it is stationed on when the camera pans. Initially, this was implemented by simply translating the co-ordinate space of the screen when a pan occurred. However, this resulted in several issues, namely by over-complicating the mapping of game object coordinates to map grid cells and by complicating the display of elements indiscriminate to panning, such as the heads-up-display and player aiming reticle. Thus, the positions of all game objects are simply updated whenever a pan occurs, which is reasonable given the limited amount of game objects that can be present, as determined via the spawning and despawning mechanics.

B.3.2 Spawning & Despawning Mechanisms

In the face of open-world exploration via procedural terrain generation with camera panning, spawning and despawning techniques were implemented. These techniques allow for the open-world to be appropriately populated with game objects for the player to interact with without adversely affecting performance due to overzealous additions of game objects.

Spawning: When the player pans the map a distance greater than the size of a map tile, new terrain tiles are generated in the direction of panning. For example, panning left generates a new column of map tiles on the left of the map grid, and 'pushes' the right-most column of tiles out of the map grid. This terrain is generated in the buffer zone of the map; tiles that are generated and maintained off-screen. When such tiles are generated, opponents are spawned with a probability, which depends on the possible enemy type considered for spawning, for each new tile. For example, sharks are spawned in the ocean more frequently compared to other opponents so their flocking behaviour can be sufficiently enacted. As game objects are spawned in the buffer zone, which is off-screen and out-of-sight, their addition to the game seamlessly integrates with player experience as though such game objects have always been present. Game objects are also spawned depending on the type of terrain assigned to the new map tiles: forts spawn on sand, enemy ships and sharks spawn in deep water, sirens spawn in deep or shallow water, and loot spawns in shallow water.

Despawning: Spawning game objects to populate the world as the player explores maintains player interest and interactivity so the player is incentivised to explore. However, freely populating the world with game objects that are not removed unless the player collects/defeats them can result in very many game objects. This can create a significant resource drain that hinders frames-per-second, which can degrade player experience. Exasperating this issue is that most of such game objects will be out of the map bounds because they act autonomously of the player. Thus, resource usage for maintaining such objects is wasteful; they are not benefiting the experience of the player. Therefore, a despawning mechanic was added to the game to limit the amount of redundant game objects and ensure that the majority of active game objects are involved in game-play.

All game objects that are outwith the map bounds for more than 10 seconds are removed from the game. This time period is large enough to allow for several beneficial outcomes, such as game objects having time to make their way back into the map area, or being persistent enough that a player 'heading back on themself' will expect to see enemies that were recently present on-screen. Furthering this mechanic is that all mobile characters that fall outside of the map area due to player exploration will ignore their usual decision tree behaviours and will instead seek the player to re-enter the map and begin contributing to the player experience again. During such time, some game rules are ignored to assist this process, such as enemy ships needing to avoid land. In fact, characters are not confined to their terrain when they are outside of the map, but must follow the rules again as soon as they re-enter. These additions aim to limit the amount of necessary despawns by affording characters the ability to regain relevance to the game.

B.3.3 Character Movement & Decision Making

Several steering algorithms have been implemented as behaviours available for the characters to utilise: seek, flee, pursue, arrive, wander, and flock (i.e., cohesion, alignment, and separation). Fleeing and evading are not currently utilised by any characters. All steering behaviours have been implemented using a force-based approach; thus, several desired objectives can be applied at once to a character simply by aggregating the forces acquired by enacting separate behaviours. For example, sharks may

simultaneously flock with nearby sharks, whilst wandering, and whilst also avoiding land.

Decision making about which steering algorithms each character should use is controlled via simple decision trees. As such, each character requires an awareness of the other characters in the game, which has been implemented using fields (i.e., circles) of awareness. The decision trees and resulting behaviours of each character within the game was provided in previous sections (B.1.4 and B.1.5).

B.3.4 Collision Detection & Optimisation

There are two forms of collision detection and resolution in the developed game that are considered distinct: collisions between game objects, and collisions between a game object and play area boundaries.

The former case concerns collision detection/resolution between different game object instances (i.e., cannonballs hitting characters, sharks biting the player ship, and sirens touching the player ship). All game objects are modelled as circles, so collision detection involves simply checking whether the Pythagorean distance between the (central) positions of two objects is less than or equal to the sum of their radii. If this is the case, then the circles are intersecting (i.e., objects are hitting each other) and collision resolution is required. Resolution takes the form of enacting the expected behaviour when the two concerned game objects interact, which has been covered in previous sections (B.1.6).

Note that characters have a circular collision geometry but are modelled as isosceles triangles so their orientation is easily discernible. Thus, 'pixel clipping' can occur where characters may appear to hit game objects on the display but this is not detected by the game. However, this is minimally observed; the difference between a character's display model and hit detection model is just several pixels. The circle representing each character is slightly larger than the in-circle enclosed by their triangular display.

With regards to detecting and resolving collisions between all the game objects, the game uses the bin-lattice spatial sub-division technique to optimise collision detection querying. In bin-lattice spatial subdivision, the map grid is divided into a collection of bins, each bin consisting of several terrain tiles in a square area (Figure 13). At each update, all game objects are distributed amongst and registered into bins based on their position. Detecting whether a given object collides with any other object does not, therefore, naively require checking against all other objects. Instead, a game object need only compare against objects also in their bin (and neighbouring bins to account for edge cases). This is particularly useful given the procedural spawning of enemies within the game. Due to random spawning, a very large amount of enemies may be present within the map, so minimising overhead per enemy is essential. Additionally, this bin-lattice spatial sub-division is necessary for the flocking behaviour of sharks; sharks should not globally flock with one another but instead only with spatially nearby sharks, which are provided by the bin-lattice. Whilst a valuable optimisation, the necessity of bin-lattice spatial sub-division is mitigated by the fact that most characters simply check for collisions with the player. Use of the technique would be leveraged best in a more dynamic version of the game where characters interact with each other and not just the player.

Figure 13: The following illustrates use of the bin-lattice spatial sub-division technique within the game. An overlay has been added showing how the map grid is sub-divided into bins containing 4x4 tiles. Every game object within the map is registered to one of such bins, which is shown by the relevant bin having been filled mostly-transparent white.

The latter collision detection case concerns keeping characters confined to their play area, which is the type of terrain they are permitted to traverse across. Ships and sharks must travel within the ocean and should avoid the land. Generally, impulses are used for non-player characters in the event that they do make contact with terrain their steering behaviour will attempt to avoid. When a moving character hits terrain they should avoid, the relevant directional component of its velocity is negated, appearing as a bounce, and the object's orientation is updated. When the player ship attempts to leave the ocean, their position is simply updated to the land boundary position they are trying to traverse across; they cannot cross the land. However, note that characters are permitted to ignore their terrain avoidance and boundary rules when they are outside of the map area so they can make their way into the map quicker and avoid being despawned. This aims to maximise the relevance of spawned entities to the player experience.

B.3.5 Sound Effects

Sound effects have also been added to the game using the `minim` library. Fundamental sound design concepts have been followed, such as sounds necessarily relating to the objects/actions they portray. Thus, the core events of the game have been given audio output to communicate themselves to the player. This creates a richer and more immersive experience for the player. References for the sound files are given in the appendix (see F.1). Sound effects present within the game are as follows:

- Game music plays during the title screen.
- A game success chime plays when the game is completed.
- Ambient ship noises are played during gameplay.
- Cannon fire triggers a corresponding gunpowder-exploding/cannonball-launching sound effect.
- Loot collection triggers a corresponding gold jingle-ling sound effect.
- Cannonball impacts/collisions trigger a corresponding impact/thud sound effect.
- Character deaths trigger a corresponding character death/disabled audio chime.

- Shark attacking the player ship triggers vigorous water sloshing, signifying that the sharks are attacking the underside of the player ship.
- Sirens sing an enchanting song when the player is caught within their area of effect.
- A wind gust sound effect is triggered whenever the wind direction changes.

C Context

As a pirate/naval-combat game, *Flagship* is accompanied by many similar games capitalising on this exciting theme; the theme naturally loans itself towards the highly popular action, adventure, and role-playing game genres.

Many video games can be attributed to having a likeness with the developed game: 'Assassins Creed (AC): Black Flag', 'Sea of Thieves', 'Blackwake', etc. Though these games are of a different format as first-person 3D games, they share similar gameplay elements to challenge the player. Other games also exist which have a similar format: 'Sid Meier's Pirates!' and 'Overboard! (1997)'. These games have a similar top-down perspective, though they maintain a fixed-size 3D world and different control mechanics (movement is keyboard-based).

Overall, *Flagship* is no mere clone of any of these games. Instead, *Flagship* draws inspiration from these titles by aggregating interesting gameplay features and re-interpreting the experience format to create a simple but novel game building upon the genre.

The following discusses specific similarities and differences of note between *Flagship* and similar games:

- Similar games possess retail quality and are effectively finished products; their gameplay elements are more advanced and polished compared to *Flagship*. *Flagship*, in its current state, is a casual game suited in the group of browser games. The primary goal in developing *Flagship* was to demonstrate learnt video game concepts related to physics, AI, and procedural content generation.
- All discussed games similarly have ocean-based, open-world maps, though they have either a fixed map size (e.g., 'AC: Black Flag' and 'Sea of Thieves') or present the world in stages ('Overboard!'). In contrast, *Flagship* has a truly open-world with infinite procedural map generation. This is simply a difference to similar games and is not necessarily an improvement; fixed size worlds allow for greater experience tailoring with denser and hand-crafted content (a lesson observed with the continuing development of 'No Man's Sky').
- None of the discussed games have point-to-click controls; ship movement is controlled through a keyboard or controller, and often via proxy through a person avatar (i.e., the player's avatar is a pirate that can interact with a ship, as opposed to just the ship itself). In this latter case, many games (e.g., 'AC: Black Flag' and 'Sea of Thieves') have a richer experience, adding upon the naval experience by allowing the player to operate as a pirate; the player is not limited to their ship avatar and can enjoy island exploring, treasure hunting, ship boarding, hand-to-hand combat, etc.
- Many of the similar games have weather components that affect the player's ship, introducing a more dynamic gameplay experience. The most popular forms of weather mechanics are storms, which were not implemented in *Flagship* for the submission, and wind, which affects the player's ship movement. In *Flagship*, wind acts much the same as is experienced in 'Sea of Thieves'; the player must keep the wind in their sails to maintain mobility, though the player does not necessarily have to travel in the direction of wind.
- All of the discussed games have AI enemy opponents to challenge the player, and in some cases other human opponents via multi-player, which *Flagship* does not support.
- All have some form of enemy ship, and most have similar fort mechanics to challenge the player from the land. The mega-fort mini-boss that has been developed draws heavy inspiration from the forts in 'AC: Black Flag'; a fort is comprised of many components that can be tackled individually to mitigate the fort's ability over time.
- As is expected from the genre, most similar games possess shark enemies. Though some of these games have shark grouping mechanics (e.g., 'Sea of Thieves'), none implement a BOID-like behaviour which develops larger groups of sharks into a greater threat to the player ship. To the best as can be discerned, this aspect is unique within *Flagship*.
- Many bosses are present in the discussed games. For example, 'Sea of Thieves' has a host of different enemy boss types, and 'AC: Black Flag' has different enemy ship bosses at the four corners of its world. These bosses have different specialities and lores, which enriches the gameplay experience.

However, in no such cases are bosses tied so closely to the goals of the game as in *Flagship*; bosses in 'Sea of Thieves' and 'AC: Black Flag' are supplementary to the game's story, whilst the overall goal of *Flagship* is to defeat the ever-present game boss.

- Some similar games use mermaid lore (e.g., mermaids are spawn markers in 'Sea of Thieves'), which is popularly associated with the pirate genre. However, the use of sirens to draw in sailors (and their ships as a result) via their ethereal voices appears to be an original gameplay mechanic in *Flagship*. This is surprising given the profound use of this lore in popular culture surrounding the pirate theme.
- The use of a merchant to allow the player to upgrade their ship is a crucial component of *Flagship* which is popularly used in similar games to different degrees. In 'Sea of Thieves', upgrades to the ship are mostly cosmetic. 'AC: Black Flag' is most similar in its upgrade mechanics to *Flagship*, where real properties of the ship are upgraded, allowing for greater health, cannon damage, range, and cannonballs per volley. However, 'AC: Black Flag' develops this further with the use of rams, harpoons, etc.

D Evaluation

This section describes the process of testing and evaluating the game, noting in particular aspects of the game that were changed based on observations during play-testing. An overall evaluation of the game is also undertaken. Note that many evaluation analyses have already been discussed as appropriate within the design and implementation section (B).

D.1 Play-Testing And Game-Balancing

Typically, video game testing occurs in phases during development. Given the nature of this practical, all game testing is characterised as alpha testing. In alpha testing, parallel testing is performed during development to ensure that the game is developed without glitches or crashes, and game balancing of key features can be performed. The following summarises aspects of the game that were altered based on observations during game-play.

A notable change differing from the pitch is that player ship upgrading is more modular (discussed in B.1.4). In summary, instead of having the player upgrade their ship into pre-defined classes ('brig', 'man-o-war', etc.), they are instead afforded the ability to upgrade individual properties of their ship (e.g., amount of health, range, damage, and spread-shot). This change is beneficial for player experience; the player can build their ship to match certain play-styles or player difficulties, and this adds to the re-playability of the game. This change was made after performing multiple playthroughs of the game. In some playthroughs, due to the random spawning of enemies, sharks are more of an issue, and prioritising more spread-shot early in the game helps to deal with groups of enemies. On the other hand, sometimes the final boss's pursuit is particularly effective, so adding player health earlier is desired to improve life longevity and suffer gold penalties less often.

Furthermore, ships within the game were initially implemented to be fully driven by the wind. However, this mechanic was frustratingly difficult. First, the random application of wind strength meant that if little wind was present then the player had to remain effectively stationary and could not explore the open-world. Second, given sufficient wind strength, the player would be forced to explore roughly in the direction that the wind was blowing. Overall, this reduced game-play to simply altering the sail alignment with the point-to-seek having insufficient effect on ship mobility. Thus, the mechanic was changed to be less realistic but more beneficial for game-play. As mentioned, the interaction between ship mobility and wind mimics that of 'Sea of Thieves': heavier wind pushes the player's ship around more, the player's degree of sail alignment with the wind determines the extent of their mobility, and the player is free to move in any direction they desire (though they will travel less quickly against the wind than if travelling with it). Overall, this altered mechanic is much more beneficial for the game, though it is difficult to game balance and perhaps still requires attention to make the effect of wind more prominent a feature within the game.

Outwith the significant changes just discussed, changes to the game based on observations largely conformed to micro-adjustments to the game configuration to balance the game. For example, by altering the relative properties of enemies to challenge the player, such as by ensuring that sharks aren't too quick given their lunging capability, by ensuring that the pulling force experienced by sirens is challenging but evadable, or by ensuring that the scaling of enemy difficulties keeps the different enemy types as a challenge throughout the game. Other adjustments were also based on aesthetics, such as by adjusting how close ships and sharks would come to the land before steering away from it, or by altering the spacing of flocks of sharks so that they would be close enough to be identified as a group but not too close as to ensure the number of sharks in a flock would be discernible.

D.2 Overall Evaluation

Overall, the game has been well-developed and game-balanced to create an enjoyable but challenging gameplay experience. Features of the game presented in the pitch were prioritised based on importance, and developed incrementally, allowing for each implemented feature to a have high quality of integration. This is opposed to a submission that perhaps would have covered more of the features presented in the pitch, which was extensive, but therefore failed to integrate everything sufficiently well within the game. As a result, the developed game pools together favourable features from similar games but adds aspects of

originality in terms of mechanics and format. The developed game also presents effective use of physics, AI, and procedural content generation concepts presented within the module. Where appropriate, further video game concepts from the wider literature have been implemented, such as through use of bin-lattice spatial sub-division for collision detection and flocking optimisation.

Despite this, several aspects of the game can be further developed in future work to improve the overall quality of the game in terms of user experience.

The player only has cannons as a form of attack. Whilst this mechanic has been well-developed, the player experience can be made more interactive by providing further modes of attack. During development, creating an interesting open-world with challenging enemies was prioritised over pitched ideas about giving the player harpoons, fire barrels, and special abilities. However, given an interesting world for the player to explore, the player should also be supplied with interesting means to interact with the world. Currently, the cannons and sail alignment mechanics have been well-developed and game-balanced to mitigate this issue of player interaction being somewhat limited or one-dimensional. Future work can be carried out to add the discussed player abilities.

Furthermore, the enemies within the game act as autonomous agents and appear intelligent in their objectives. However, these objective are limited to interacting with the player; all enemies will wander (if able) until they are aware of the player, and will then pursue and combat the player. A more dynamic open-world can be attained by expanding on the decision trees of the opponents to include other AI members. This would greatly increase the immersion experienced by the player as being just a component within an active and changing open-world. One such suggestion has been discussed, that enemy ships and forts may form parts of alliances and battle each other. Such an idea is present within 'AC: Black Flag' with great effect. The addition of such features would not be difficult in the present implementation, with difficulty likely arising from having the AI fight each other somewhat stupidly such that the player may witness their interaction and so that game characters are not needlessly spawned to then be removed quickly in most cases simply due to them facing stronger AI.

Also, ship cannons currently do not operate fully realistically. Cannons use forces to fire their projectiles, which are affected by the ship's velocity, and fired cannons have cool-down periods to mimic the real-world and balance the game. However, whilst ships have port-side and starboard-side cannons, the operational field of such cannons is not limited; they can aim within the full 180° on each side of the ship. As a result, ships can shoot straight ahead when in reality they should have to turn so that their cannons face an opponent they wish to fire upon. Initially, it was thought that having static cannons that fire straight ahead perpendicular to the heading of the ship would be less fun for the player. Does the player want to have a ship rudder mechanic to turn their ship to be able to fire cannons at an opponent, or do they want to have an aiming and firing mechanic for their cannons instead? The latter mechanic was chosen as being more enjoyable for the player, but also because it made more sense within the game. If there was a rudder, would this rudder also affect movement of the ship? If so, then is the click-to-move interaction appropriate? Would use of a rudder make the already difficult game too difficult to enjoy? A compromise was planned, which was not implemented and is left for future work, by limiting a cannons' field of operation on a ship to be lower than the full 180°. Thus, ship cannons could not be fired forwards or aft of the ship but would still maintain a degree of swivel allowing them to be aimed.

Another small improvement would be in re-configuring the upgrade options available to the player. Currently, each upgrade always costs the same and improves the ship by the same amount. However, successive upgrades should become more costly and upgrade the player ship less and less each time. This is a popular mechanic in video games which encourages the player to upgrade their avatar to be more well-rounded and, more crucially, enforces upgrade maximum boundaries to prevent the player from becoming too over-powered or breaking the game.

E Conclusion

In conclusion, an open-world, top-down, single-player, action-adventure, pirate/naval-combat game has been developed using `Processing`. The developed game, referred to as '*Flagship*', uses a variety of technical concepts related to video games. A context survey was conducted, outlining how the features of the developed game relate to and differ from similar video games. The developed game is simple and novel, which includes many effective and interesting features from similar games but also introduces original gameplay elements. An evaluation of the developed game was also performed, discussing how the testing of the game resulted in required changes to improve the game-play experience. Areas of improvement that remain were also discussed, along with their possible implementation as a part of future work.

F Appendix

F.1 References

F.1.1 Implementation Resources

- AI Steering Behaviours: https://natureofcode.com/book/chapter-6-autonomous-agents/ [Accessed April 2022]
- Bin-Lattice Spatial Sub-Division: Reynolds, Craig. (2000). Interaction with Groups of Autonomous Characters. Game Developers Conference. 21. http://www.red3d.com/cwr/papers/2000/pip.pdf [Accessed April 2022]

F.1.2 Sound Effects Links

- Game Music: https://www.youtube.com/watch?v=M0DLY9bgXi0ab_channel=FreeRoyaltyBackgroundMusic
- Game Over Chime: https://www.zapsplat.com/music/cartoon-success-fanfare/
- Ambient Ship Noises: https://www.zapsplat.com/music/internal-old-wooden-ship-movements-wood-creaking-rumbling-1/
- Cannon Fire Sound Effect: https://www.zapsplat.com/music/8bit-medium -explosion-bomb-boom-or-blast-cannon-retro-old-school-classic-cartoon/
- Impact Thud Sound Effect: https://www.zapsplat.com/music/clothes-fast-movement-into-a-punch-thud-1/
- Character Disabled Sound Effect: https://www.zapsplat.com/music/anime-hit-burst-mini-explosion/
- Shark Attack Sound Effect: https://www.zapsplat.com/music/shark-feeding-at-surface-of-water-with-splashes-and-movements-as-shark-shakes-head-to-tear-or-rip-apart-food/
- Wind Gust Sound Effect: https://mixkit.co/free-sound-effects/wind/
- Siren Song: Original piece of audio created by myself.