3. Implementation

Group 11 - 11 Musketeers
Osama Azaz
Adam Dawtry
Tom Jackson
Brendan Liew
Holly Reed
Harry Ryan

Implementation

Game download (.jar):
https://github.com/UOY-ENG1-Team-11/yorkpirates/releases/tag/v2.0.0
GitHub repository: https://github.com/UOY-ENG1-Team-11/yorkpirates

The base architecture of the program (as established by the previous team) remains largely unchanged. However, a number of extensions, including several new classes, have been added in order to implement the new requirements. All requirements have been met to some degree.

New Requirements

UR.POWER_UPS

For this requirement, it was necessary to add power-up items that the player could collect somehow in order to gain a boost to their statistics (e.g. speed, damage.) This was implemented by placing consumable power-ups with unique icons around the game world, which the player could pick up by sailing into in order to gain that specific power-up. The following classes were added for this purpose:

Consumables.java – Represents an object in the game world that can be picked up and "consumed" by the player. Largely implemented for the benefit of the ***UR.POWER_UPS*** requirement, but allowed room in the project for other consumable items (e.g. floating treasure chests in the ocean that would grant the player additional plunder.)

PowerUps.java (extends Consumables) – Handles instances of collectable power-ups in the game world. The creation method can be invoked to place one of five power-ups (attack speed, damage, health, invincibility and speed) in the game world, which the player can pick up to gain their benefits for a period of time.

PowerUpsTimer.java – An instance of this class represents a new timer for a power-up the player has picked up, so power-up timers can be independent of one another.

UR.ATK_SHIP

This requirement stated that there must be combat between boats in the game. The player already had the capability to shoot damaging cannonballs, so all that needed to be added was the enemy ships and their AI. The following class was added:

Boat.java (extends GameObject) – Instances of this class represent ships which can sail around in the game world and shoot enemy ships. Extending the player class, which already represents a boat with most of the necessary faculties, was not practical, as the player class has a number of methods which were not useful for the AI ships, such as those used for camera control and handling keyboard input. The AI uses simple pathfinding from its current point towards the player, and will hover a certain range away from them while shooting cannonballs.

UR.OBSTALCES

For this requirement there had to be obstacles which were dynamic in how you interact with them (i.e. not just stationary rocks that get in the way.) The obstacle that was added is a wave, represented by the following class:

Enemy_Wave.java (extends GameObject) – Will occasionally spawn in the game just beyond the player's field of view and move towards the position the player was at when the wave was created. If it collides with the player, it deals damage and disappears, and if not, it will travel a fixed distance before disappearing on its own.

UR.DIFFICULTY

This requirement states that several difficulties must be present in the game, selectable from the start. For this, some changes were made to the code which instantiated **College** so that their statistics could be increased for higher difficulty levels. The following class was also added:

NewGameScreen.java (extends ScreenAdapter) – An additional screen added for difficulty selection. It appears when the player chooses to start a new game on the title screen, and there are 3 difficulties: Easy, Medium and Hard which progressively make enemies more deadly.

UR.WEATHER

Bad weather events must occur in the game and have an effect on it. Besides the initial "clear" weather, a "storm" weather type has been added, which increases the frequency and speed of the waves which were implemented for ***UR.OBSTALCES***. A new class was created to handle weather events:

WeatherManager.java – Handles the occurrence of weather events in the game and the random spawning of damaging waves in all weathers. Around every 30 seconds the WeatherManager enables a storm. The storm will end after a period of time and the weather will be set back to normal for another 30 seconds.

UR.SPEND_LOOT

The player must be able to spend the plunder they earn in-game for some benefit. This was done by adding a shop feature with a new class:

ShopScreen.java (extends ScreenAdapter) – Provides necessary function calls and display of UI elements to facilitate a shop screen. A player may open the shop menu to spend their gold on one of three permanent upgrades: attack speed, damage, or movement speed.

UR.SAVE_LOAD

This requirement asked that the player be able to save and load games to and from a file while keeping the gamestate the same between saving and loading. This was implemented by adding new methods to convert elements to and from a .json format to existing classes such as **Player** and **College**, as well as new classes whose data needed

saving such as **Boat**. Things that are stored in the .json include any actors in the game, their health, coordinates, the player's current gold; anything which would otherwise change the gamestate substantially were it not saved.

Updates to Old Requirements

FR.GAME_SOUND.2

Music was implemented originally, but sound effects were missing. To complete this requirement fully, sound effects have been added and are on the same toggle as the music, which are both muted by default per request. All sound effects have been generated using Chiptone [1] which is made free to use for all purposes under the CC0 licence [2]. A new class was created to easily load and play sound effects as needed:

SoundManager.java – The SoundManger loads in all the game sound effects from the start and has methods to play each one. Where a sound effect should be played in other parts of the game code, a corresponding method call has been added. Adding a new sound effect is as simple as creating a new variable to hold that sound, and a method to play it.

NFR.STABLE

The game must be stable and not crash, and the original did meet this requirement as far as was tested. However, an issue was noted with how the game loaded its textures. Initially, textures were loaded each time a new class instance was created, but they were not properly disposed when the instance was no longer needed. Over several new games, if the player or any colleges were to fire enough projectiles there was risk of a memory leak occurring, causing the program to crash. As well, instead of loading re-used textures once and simply toggling their visibility as needed, they were loaded and unloaded every time they were used (notably the hurt flash effect when a college is damaged and flashes red.) This was fixed by creating a new class and changing the way the game loaded textures:

TextureHandler.java – Stores all the textures that will be used frequently right at the start, and centralises all the methods necessary to deal with textures in the game, namely loading and disposing. Dispose and other **TextureHandler** method calls have been added to appropriate places in the program, namely classes where sprites are being dealt with.

Features Not Fully Implemented

All the requirements are satisfied to some extent within the final version of the project, however there are some features which were developed that did not make it in.

A* Pathfinding Algorithm

There was an initial attempt to have the non-player ships pathfind using the A* algorithm. However, converting the representation of the game world that was stored in the program into a format that was traversable by A* was a difficult task that was ultimately decided to be not worth the time invested when a simpler pathfinding method could be used instead in order to meet the project deadline while still meeting the requirements.

Additional Consumables

Only the power-ups made it into the final version due to time constraints.

Bibliography

[1] T. Vian, *Chiptone,* self-published to itch.io, Jan. 2021, Accessed on: Feb. 17 2022. [Online] Available: https://sfbgames.itch.io/chiptone

[2] Creative Commons Corporation, *CC0 1.0 Universal*, Nov. 2013, Accessed on: Feb 17 2022. [Online] Available: https://creativecommons.org/publicdomain/zero/1.0/legalcode