Implementation

Group 26 - Spice Traders (prior team 22)

Dan Wade
Alice Cui
Robert Murphy
Charlie Crosely
James McNair
Marc Perales Salomo

Implementation

Part a) The working implementation code can be found at: https://github.com/booksaw/spice_traders. The executable JAR file of the game is called "spice_traders.jar" and is found attached to the latest release: https://github.com/booksaw/spice_traders/releases

Part b) Before starting the implementation of the requirements for assessment 2, we reviewed the project as we received it and decided to make a few changes in order to facilitate our future work. These changes include:

- Code organisation: We decided to reorganise the existing code into packages in order to facilitate our future coding tasks and achieve a more logical structure. This is because the previous group left all classes within the same package, making the interaction between classes difficult to understand.

- Movement adjustment: Upon playing the game we found that the player's movements are momentum based without any deceleration, making it hard to control when trying to dodge enemy bullets and obstacles. As the implementation of obstacles is required in assessment 2 we decided to change the movement mechanics so the player slows down when no direction keys are pressed. These adjustments were made within the *GameScreen* class

- Game difficulty rebalancing: We noticed that the game was quite challenging to complete for two reasons, the player's ship only shoots from the sides of the ship making it hard to aim, and that the college's cannons have very high accuracy and bullet speed. Thus we modified the player's shooting mechanism so cannonballs travel to where the player's cursor is on the screen and added random variation to the college cannon shots so they do not all go exactly towards the player. These changes were made within the *GameScreen* and *College* class.

- Addition of a tutorial: Upon playing the game for the first time, many of our group members found the rules of the game ambiguous due to the objective being unclear and there being no indication of how the controls work. This is in conflict with the requirement NFR_OPERATABILITY, so we decided to add a tutorial screen at the start of the game to ensure that the game is easy to understand.

- Resizing issue fixing: When trying the game out on different computer screens we found that the game does not resize appropriately on smaller screens, as this

is against the non functional requirement NFR_RESILIENCE we made sure to resolve this issue.

- Updating the CollegeID system: At the end of the assessment 1 the identity of each college was managed by a control bit where each college had a different ID (for example alcuin had an ID of 1). This made changes between colleges challenging as we were not familiar with which college had which ID. This led us to make the improvement of moving the ID system to an enumeration (called *CollegeMeta)* within the project to control which college an ID was referencing. This made it a lot quicker for us to make changes to the colleges.

To accommodate for the new requirements that were released for assessment 2, we implemented the following features:

- Implementation of five different power ups: To implement the user requirement UR_PLAYER_POWERUP, we created the abstract class *"PowerUp.java"* extended from the Entity class to define the characteristics of power ups in general. Each of the individual power ups then have their own class, extended from the abstract class, containing code written for each of the functionalities. All the implemented powerups can be found within the *gameobjects/entity* package.

- Addition of the save game progress option: We implemented a save game feature to satisfy the user requirement of UR_SAVE, a save button has been added to the menu screen which is accessible by entering the pause menu. The game is saved locally to the player's computer using an xml file. The saving component of the game can be found within the *save* package.

- Game difficulty selection: Another requirement new for assessment 2 was to allow multiple difficulty levels for the game. This was implemented by giving the player the choice to pick a difficulty level at the start of the game and then by scaling the player health and damage received based on the difficulty selected. This satisfies the requirement UR_DIFFICULTY. These changes were made across all enemy classes, but the difficulty selection portion can be found in *DifficultyScreen*.

- Addition of obstacles within the game: For assessment 2, a new requirement was added for a sea monster and bad weather events to be added to the game. To do this we implemented both a Sea Monster in the form of a Kraken (found in

the class *SeaMonster)* and a tornado (found in the class *Tornado*). These additions satisfy the requirement UR_OBSTACLES.

- A* Pathfinding: Though not linked to a specific requirement, this helps with the implementation of UR_OBSTACLES and UR_ENEMY_SHIPS. An implementation of A* Pathfinding was created to make enemy ships, tornados and sea monsters much easier to implement. This module of the system allows an object to specify a location and not worry about the specific pathing details to get to that location.

Two features identified during the requirements elicitation process were not fully implemented. Both are non-functional requirements as are as follows:

1. NFR_TIMING - *Spawned entities (e.g. coins) have a finite lifetime.*
    - Entities like coins and power ups do not despawn, they remain available for the player to collect throughout the game. Special entities like sea monsters and bad weather also have infinite lifetimes, they move around the map so the player must try to avoid them.

2. NFR_PRECISION_CONSTRAINT - *Entities shall not occupy the same position on the map.*
    - Entities of the same nature were made to be in different positions from each other, however power up entities and coin entities will occasionally spawn in the same location in the game. This isn't a major issue because it is rare and can be merely seen as a double gain for the player.