# IMPLEMENTATION REPORT

NP Studios – Team 8

Team Members
Lucy Ivatt, Jordan Spooner, Alasdair Pilmore-Bedford, Matthew Gilmore, Bruno Davies, Cassandra Lillystone

**Refactor:** After starting implementation for this assessment we ran into a few problems. The first being the use of 'Kroy.mainGameScreen. ...' which caused major issues when testing classes that used this type of access. To overcome this, we refactored code to ensure there was as little dependency as possible. More information on how the testing was completed and how it has been changed is in our Testing and Evaluation Report.

**Powerups:** We were required to implement at least 5 power ups, therefore we created two new requirements **UR_POWER_UPS** and **SFR_POWER_UPS**, two new classes *PowerupBox* and *StatusIcon*, and a partial UML diagram to show changes to architecture (https://npstudios.github.io/assessments/UpdatedUMLA4.png). To gain a power up the player must locate the *PowerupBox* and play the minigame. We decided on this method as we believed it better met the requirement **UR_MINIGAME**; before, the minigame was only accessible from the main menu, but now it is embedded in the main game. The players minigame score is then used to determine which power up they have earned, and the correct *StatusIcon* appears on the Firetruck/HUD. Table 1 in the appendices shows the powerups we decided to implement. The required score for each powerup is based on how much of an advantage that power up provides: the better you perform in the minigame, the better the advantage.

**Difficulty:** We were required to implement three difficulty levels; easy, medium and hard. Therefore, we created two new requirements **SFR_DIFFICULTY** and **UR_DIFFICULTY_LEVEL** and added buttons to select difficulty on the Firetruck selection screen. As difficulty increases, there are more aliens per patrol and fire trucks have less health and damage. More aliens are harder to dodge, and the fire truck specs largely affect how easy/hard it is to defeat the fortresses to win the game.

**Saving:** To enable saving we created the following: new requirements **UR_SAVING** and **SNFR_SAVING**, a save function which saves all necessary current position / health information etc. to a file and a load function which can access this file and put the game into its previous state. We also added 3 buttons to the pause menu, for the three different save slots available, which calls the save function as well as a new class *'LoadWindow'* so the user can access a page from the menu and load from any of the three slots. This addition meant a slight change to the architecture (https://npstudios.github.io/assessments/UpdatedUMLA4.png). Due to the fast pace of our minigame we believed it was unlikely that the user would want to save during minigame so we decided to not implement a save state of this specifically, however any power ups gained from the completion of the minigame will be saved.

**Fortress Stats:** To fulfil **UR_ET_UNIQUE_SPEC** fortresses needed unique health and damage values and we noticed that the previous group did not implement this. We created a new list which contained these attribute values for each fortress. When a fortress is instantiated, it accesses the values in this list and passes it to the constructor. This method is similar to how the varying Firetruck stats are implemented and therefore we chose it to keep the code consistent. Additionally, **UR_ET_IMPROVEMENT** (fortresses becoming stronger overtime) was unimplemented so we doubled the fortress damaged halfway through the game as well as increasing each fortresses health every time a truck is destroyed. These both encourage the user to be as fast as possible while taking as little damage, encouraging more strategic gameplay. We also noticed that one of the fortress' bullets patterns did not aim towards the fire truck, so we changed the variable \`aim\` from to true on that pattern, improving the implementation of **SFR_FORTRESS_ATTACK.**

**Win Condition Fix:** We found by playtesting that the game was won before all 6 fortresses were destroyed. This was due to the fortress count still being set to 3 from assessment 2. This violated **UR_WIN_CONDITION** therefore to fix this we set the fortress count to the correct value by checking how many fortresses were currently alive and calling the end game condition when it reaches 0.

**Map Edits:** We added the York City walls around the perimeter, stopping the player from running into an invisible barrier made more confusing . Thereby removing ambiguity and making the game easier to understand (SNFR_SIMPLE). The warping looked quite unpleasant and could confuse players, also violating SNFR_SIMPLE. Furthermore, when the fortresses were destroyed their 'dead' texture did not match the building. We improved this by designing and replacing their textures, to improve the consistency in the game to fulfil SFR_FORTRESS_DESTROY and SFR_SIMPLE. We also changed the texture for the destroyed fire station, previously consisting of just a cross through the 'alive' texture. This initial design was very underwhelming, possibly violating UR_FUN.

High Score Fix: We noticed that the high score did not save between instances of the game, even if Kroy had not been exited. We fixed this by reimplementing the existing methods of *setHighScore(int)* and *getHighScore()* so that they used libGDX preferences which saved the score between game instances. This better fits the requirements UR_HIGHSCORE, SNFR_HIGHSCORES which specifies there should be a 'local record' of the high scores saved. We wanted the user to be able to keep their record of their score as it both improves its replay factor as well as being intriguing factor for open days; encouraging prospective students to try and beat the current high score.

Instructions Screen: During our initial observations, we saw that UR_INSTRUCTIONS was unimplemented. Therefore, to implement it added a 'Controls' button to the Main menu. To create the controls screen, we created a 'ControlsWindow' class to keep consistent with the current code base i.e. we made sure to implement it in the same way that the *'OptionsWindow'* is already implemented. Furthermore, we edited the switch statement in the *'MenuScreen'* class to include the 'CONTROLS' case which would draw the instructions and controls to the screen.

Window Size: When playing we found it was quite difficult with such a small game window, such as avoiding patrols, so we increased the size of the game window in the Kroy class. We believed this helped to better meet the requirement UR_INTUITIVE. This meant we also had to change the menu image as the previous file was too small.

Firetruck Attack: We slightly edited the implementation of SFR_FORTRESS_DESTROY. Previously the user would attack a fortress when in range automatically, however, we wanted to give the user more choice about when and where to use their water which we believe created the opportunity for more strategic gameplay and makes the game more engaging. To do this, we added a conditional to the statement previously used for the Firetruck attack that ensured the space key must be held down.

Mini map: The map of the game is quite large and navigating it can be difficult. Previously this was fixed by a zooming feature, however we found the animation of the game whilst zoomed out to be distractingly choppy. Thus, the zoom has been replaced with a mini map, which draws all the game objects to the bottom left corner. This allows the user to navigate to objectives without having to be constantly zoomed far out or wandering around lost, in order to meet SNFR_SIMPLE. This is especially useful for the power ups which are small and would otherwise be difficult to locate. The mini map is also on a toggle so the user can hide it when needed, reducing screen clutter.

Maximum Patrols: As there was no maximum number of UFOs that could spawn, we found it made the game overly difficult as time progressed, and it also caused a memory leak after a prolonged amount of time. Therefore, we linked the maximum spawn of UFOs (to a fortress) to the difficulty chosen by the player.

Minigame Redesign: After embedding the minigame as a way to achieve power ups, we redesigned the character, making it a flying goose carrying a present. This was to suggest the goose was bringing the powerup to the player, making the minigame link more to the main game and better fulfilling UR_MINIGAME. We also redesigned the obstacles to keep the art style consistent. Ensuring the game is as aesthetically pleasing as possible due to the fact it will draw students in at Open Days, which is a key purpose of the game.

## Appendix

### Table 1

| Powerup | Score Required |
|---|---|
| Infinite Water on Selected Truck | 2-5 Points |
| Infinite Health on Selected Truck | 6-10 Points |
| Destroy all UFOs | 11-20 Points |
| Additional Time | 21-25 Points |
| Fire Truck Resurrection | 26-30 Points |
| Freeze all Enemies | 30+ Points |