# SEPR 2019/20 Assessment 3

## Team CheatCodez

Jonathan Grout, George Lesbirel, Cheuk Wang Wu, Lauren Quarshie, Muaz Atif, Lillian Coultas

# Implementation Report

**Key**
Throughout this report, any requirements referenced will be in capitals and highlighted in yellow e.g. NO_HEAL, risks referenced will be in green e.g. R-01, and references to the change register will be in blue e.g. REQ_1.

As a group, we voted on which other group's project to pick for this assessment to make sure every team member had a chance to express their opinion (also helping to avoid R-04, as Salt N Sepr (SNS) had the majority vote, but every team member was given the chance to express their opinion, and we were all happy with the project we picked). We decided to choose SNS's game for this assessment because it was impressive visually and the code was well commented and laid out, making it easy to understand and add to. The documentation provided was well written and clearly formatted, and also showed that the majority of their tests for assessment 2 passed, giving us less extra work for this assessment in terms of making the game playable. The functionality we needed to implement for this assessment, (namely the minigame and patrols), had clear associated requirements, but also gave us some freedom in the specific implementation as the minigame hadn't been designed, allowing us to create one ourselves.

**Implementation of the requirements in our code**
The focus of this assessment was on adding the minigame and patrols into the game. The list of requirements that were already fully implemented are:

- User Requirements: CONTROL_TRUCK, CONTROL_SPRAY, RETURN_HOME, VARIED_TRUCKS, CREATE_MAP, MENU.
- Functional Requirements: CONTROL_TRUCK_FUNC, CONTROL_SPRAY_FUNC, RETURN_HOME_FUNC, VARIED_TRUCKS_FUNC, CREATE_MAP_FUNC, FORTRESS_HEAL.
- Non functional Requirements: RESILIENCE, GAME_ACCESSIBILITY.

Some requirements were also partially implemented - DESTROY_ENTITIES and DESTROY_ENTITIES_FUNC require the player to be able to destroy both aliens and fortresses, however there were no alien patrols implemented yet, so these requirements had only been satisfied for fortresses thus far. This also applies to the GAIN_INCOME user requirement, as gaining points for destroying fortresses has already been implemented, just the patrol part of this requirement is left. As well as this, the NO_VIOLENCE user requirement and AUDIENCE_ACCESSIBILITY functional requirement can only be satisfied if there is no violence in the completed game, and since the game was only partially complete, they were only partially satisfied. WIN_GAME was partially implemented, as, although there was a function which checked if the player had destroyed all 6 fortresses (and therefore won the game), it didn't inform the user they had won, and just returned them to the main menu. This was confusing, as it might have been mistaken for the game crashing, or the player losing the game, and also meant that the associated WIN_GAME_FUNC functional requirement wasn't satisfied, so even though the WIN_GAME user requirement was technically satisfied, we felt the implementation could be improved upon.

The minigame, as mentioned in the added MINIGAME requirement (documented as REQ_9), needed designing, planning, and implementing in this assessment, as the previous documentation for this group didn't include any plans for what the minigame should be. This

gave us a lot of freedom in designing it, and we decided on a game wherein the player has to rotate pipes to create a valid path from the top of the screen to the bottom, aiming to create a path for water to flow through and refuel the fire truck (giving it a bonus when the minigame is complete). This minigame has a different style to the main game, but is still aligned to the same theme, as it's about refilling the fire truck the player controls in the game. In accordance with GAME_ACCESSIBILITY, the minigame is available from the main menu.

To satisfy the CREATE_ENTITIES_FUNC requirement, and the associated CREATE_ENTITIES user requirement, we added ET patrols to the game. The patrols range about the map in a set route, and the player could encounter them when they drive the firetruck around. They act the same in every game, as the requirement says, and they are spawned as soon as the game begins. When an alien is destroyed the player gains a small number of points, which completes the implementation of the GAIN_INCOME user requirement (as points are also gained by destroying alien bases. This part of the requirement was implemented by the previous group). Aliens are destroyed by spraying water at them, as per the DESTROY_ENTITIES_FUNC requirement - once an alien runs out of health, it is removed from the screen. These patrols also complete the implementation of the DESTROY_ENTITIES user requirement (and all associated functional requirements), as they had already been partially satisfied by the previous group's implementation of alien fortresses. Patrols also satisfy the NO_HEAL requirement, as if they are damaged there is no way for them to regain health, and continue the NO_VIOLENCE requirement, as they just disappear when they run out of health, with no violence shown. The points gained from destroying them are used for the LEADERBOARD and LEADERBOARD_FUNC requirements, as they count towards the player's score.

We also added a leaderboard, satisfying the LEADERBOARD and LEADERBOARD_FUNC requirements. The leaderboard is available from the main menu, as GAME_ACCESSIBILITY requires. The name associated with the score is set to the current date and time, and this cannot be changed (in accordance with the SECURITY non-functional requirement, as no personal details need to be entered into the database, and the user cannot enter any details). We also changed the WIN_GAME implementation, so that when the player has won the game, they are told if they've won or lost the game before being taken back to the main menu. This makes it clearer for the player, and completes the implementation of both the WIN_GAME and WIN_GAME_FUNC requirements.

Apart from the minigame and patrols, the product brief also specifies the implementation of two other features; the ET patrols will destroy the fire station a fixed period of time after the first fortress has been destroyed, and the fortresses become harder to flood over time. To satisfy the latter, we implemented additional fortress upgrades - the game should only last for 5 minutes maximum, as per the TIME_ACCESSIBILITY requirement, so when two thirds of this time has elapsed, the maximum health and current health of all remaining alien fortresses double, making them harder to destroy. This satisfies the FORTRESS_IMPROVEMENT requirement, REQ_5 on the change register. We also implemented the ET patrols destroying the fire stations. When there is only 20% of the game time left, the patrols will destroy the fire stations, in order to satisfy both the

TIME_ACCESSIBILITY non functional requirement, as this will stop the player repairing their trucks, making the game end soon after this point, and the DESTROY_STATION requirement added by our team (documented as REQ_7 in the change register), as the firetrucks being destroyed is also explicitly mentioned in the brief.

**Implementation of the architecture in our code**

We haven't made any significant changes to the previous software, except those required to implement the minigame, patrols, and leaderboard. This means that we can be sure that the code still implements the architecture from the previous assessment, as we have only updated it, and made minor modifications to existing files, and have not changed anything significant. The changes we have made are also mostly as a consequence of features that were not planned out in previous assessments, e.g. minigame and leaderboard, and so consist mainly of entirely new classes, with the existing classes/relationships remaining mostly unchanged. We've included an updated UML class diagram, to show relations between classes only (see **Appendix 2.1**).

| Feature : New | Minigame |
|---|---|
| Files added/modified | DB.java, Pipe.java, MinigameScreen.java, MinigameSprite.java, MainMenuScreen.java, Constants.java |
| Requirement(s) reference | MENU, MINIGAME, GAME_ACCESSIBILITY |
| Explanation | "Pipe" class was added, screens "MinigameScreen" and "MinigameSprite" were also added. "MainMenuScreen" class was updated to allow the minigame to be accessed from the main menu. "Constants" was updated to include minigame constants. To implement the minigame itself, we used a **2D array** to generate a route from the bottom of the screen to the point at the top of the map where the hose is placed. There are **8 different states** the pipe can be in, so we use **4 bits** to show which type of pipe should be rendered. The ending pipe is only given 1 state, regardless of its rotation. A checking function has been implemented which checks if there is a valid path of pipes. Each pipe has 2 sides, the input, and the output. The checking function starts from the first pipe, and checks that the output side of the pipe has an input side of a different pipe on the correct adjacent square. It continues checking like this until it either reaches the end pipe (a valid path), or reaches an output pipe with no matching input on the next square (an invalid path). If there is a valid path, the player is told they succeeded, and the minigame ends, returning the player to the main menu. |

| Feature : New | Patrols |
|---|---|
| Files modified | ET.java, Traveller.java, Constants.java |
| Requirement(s) reference | CREATE_ENTITIES, DESTROY ENTITIES, DESTROY_ENTITIES_FUNC, NO_HEAL |

| | |
|---|---|
| Explanation | ET class was added with non-primitive data types **“Circle”, ”MapGraph”, and “Texture”**. Pathfinding classes were added (Heuristic, MapGraph, Node, Path, Traveller classes), in order for the **libGDX built in A* pathfinding algorithm** to correctly work with the game. **AI dependency** was added to the build.gradle file and the **pathfinding function** was called. Constants.java file was updated to include new constants. Patrols were implemented by placing nodes at road corners and drawing connections between them to make the path for the aliens to traverse. Using **libGDX's built in A* pathfinding algorithm**, the patrols go along these paths in a preset order, which doesn’t change between games. Each alien has a range of detection and when a fire truck enters this range the alien shoots at it. This will stop when the firetruck leaves the range. |

| | |
|---|---|
| Feature : New | Leaderboard |
| Files modified | DB.java, HighScoreRecord.java, MainMenuScreen.java, LeaderboardScreen.java, Constants.java |
| Requirement(s) reference | LEADERBOARD, LEADERBOARD_FUNC, SECURITY |
| Explanation | “DB”, “HighScoreRecord”, and “LeaderboardScreen” classes were added. The Database class was used to handle **I/O** with the score database. “MainMenuScreen” class updated to allow the leaderboard to be accessed from the main menu, Constants.java updated to include leaderboard constants. When the player wants to see the leaderboard, the top three records (ordered by highest score) are pulled and displayed to the player. At the end of every game, a new record is added to the database, which only stores the player’s name, and score. The name is set to the current date and time, and the score is the player score added to the time remaining on the game. |

**Features not implemented**

In SNS’s requirements specification, multiple requirements, such as OPEN_SHOP and BUY_ITEM and their corresponding functional requirements OPEN_SHOP_FUNC and BUY_ITEM_FUNC, describe the implementation of a shop in which fire trucks with different specs can be bought for points or some type of currency. We have chosen not to implement these requirements as a mitigation strategy for risk R-01, which describes an underestimation of the time taken to develop the software. This ensures that adequate time is assigned to the implementation of essential requirements/architectures i.e. user requirements with a "SHALL” or “SHOULD” priority, and requirements extracted from the product brief itself. We contacted the SNS team via email to ask whether or not the shop functionality and requirements came directly from the stakeholder or customer, and they confirmed that they did not. This confirmation, combined with the associated requirements being of the lowest priority (“MAY”), was our justification for deciding not to implement the shop functionality.