Module Code
COM00008I

UNIVERSITY of York

BSc, BEng and MEng Degree Examinations 2019–20

DEPARTMENT OF COMPUTER SCIENCE

Software Engineering Project (SEPR)

Open Group Assessment

Module	Software Engineering Project (SEPR)
Year	2019/20
Assessment	4
Team	The Dicy Cat
Members	Michele Imbriani Daniel Yates Luke Taylor Isaac Albiston Martha Cartwright Riju De Sean Corrigan
Deliverable	Implementation Report

2. IMPLEMENTATION

As discussed in the Project Review Report, the project we decided to pick for this final Assessment was NPStudios', whose project was the one we developed in Assessment 2. This meant that from the earliest stages of this assessment's iteration, we were all well aware of all the architecture's strengths and flaws, as well as having a thorough, low-level understanding of it.

In order to accommodate the **Powerup feature** requested from the client, our team decided to implement it adhering to a reward-prone system: powerups are dropped by the ET patrols after being defeated. Opposed to randomly spawning powerups throughout the map and leaving to the player the task of roaming around in order to arbitrarily find them (which, to some extent, aided or penalised the player according to RNG), this feature was slightly more challenging to implement, but we believe that it succeeds in encouraging the player in adopting a more action-oriented gameplay style, which, considering the mixed and, perhaps, transitory target audience, is what we believe to be the most attention-catching choice of gameplay.

To accomplish this, we created a powerup class which inherits from the "*GameObject*" in order to display the icon which the player could drive over to obtain that powerup. This class is initialised any time 3 aliens are destroyed and the type of powerup which spawns is decided randomly.

To implement the functionality of the powerups, we created a new enum called *"PowerUp"* in the "*FireTruck*" class which has three states: "*OBTAINED*", "*ACTIVE*" and "*NULL*". These "*PowerUp*" enums are stored in a list, with each item in the list reflecting a different possible powerup that could be active. This list is controlled by a variable which acts as an indicator for which power up is currently selected by the user. This indicator is displayed in the HUD and when the player presses the Tab key the next power up is selected. Each instance of *FireTruck* has its own list of *PowerUp* enums allowing for the player to consider which truck should have the power up whenever they appear.

The **powerups** implementation was obviously the one which altered the game's **GUI** the most. We believe that in a fast-paced game such as ours, visual feedback was a top priority in order to make each powerup visible, clear and straightforward for the player to understand. Since our team felt that this implementation was a major turning point in the overall gameplay of our game, we decided to take this a step further, making sure that the best possible visual experience was provided to the player. Therefore, every powerup sprite was hand drawn by our team -which facilitated the task of creating powerup icons that were unambiguous, as well as avoid any copyright-related issues- with an emphasis on colour selection: the icon must be visible at all time on the map (hence, no colours that resemble the map tiles' were used) and the images inside the icons must be clearly distinguishable from their background (high contrast of colours). Our team is highly satisfied with the final result obtained in the gameplay, and we believe that this feature was implemented in the most optimal way possible.

Another feature that was to be implemented is different **Difficulty levels**. For this feature, several ideas were considered, but we quickly came to realisation that they were starting to become increasingly more challenging to implement as ideas were put out, and that they quickly got out of the scope of this project. Our final decision for this feature was therefore to make the fire trucks increasingly more fragile and vulnerable to projectile by decreasing the health point of each fire truck by a multiplier of 0.5. This made for a straightforward, basic, yet very efficient implementation which also made its relative testing uncomplicated.

On the **GUI** side, this change did not impact noticeably the visual feedback for the player: after playing around with how this implementation could be reflected visually in the gameplay, we came to the conclusion that for consistency's sake the most optimal option would be to keep the health bar's visual constant. Out of the different visual possibilities that our team tested, we agreed that this choice was the wisest.

Finally, the last feature to be implemented was the **saving option**. Arguably the least impactful when it comes to GUI, this feature was tricky to implement due to the many elements inside the game that need to be considered. We decided to implement it in a way that would be reset the saved statistics every time the game is executed: similarly to all the features implemented in this Assessment's iteration, this choice was made keeping the target audience's experience as a priority. We decided that the game savings would get reset whenever the game is closed and executed: a user during an open days is extremely unlikely to close the game and restart it – a more probable scenario would however see the player wishing to pause the gameplay with the intention of resuming it without closing the game. Hence, needing to keep savings stored internally while the program is closed is not something that would ultimately be of best use.

This decision eliminated some hindrances, but it by no means made the task of implementing the feature any easier. As a matter of fact, it necessitated two new classes- GameSave and SaveScreen. GameSave is where the actual "saving" takes place, an instance of the class represents a save. When an instance of GameSave is instantiated in GameScreen, an empty save is created. Then, when the method "saveGame" is called, the instance of GameSave updates certain values to match those of the game that is running, effectively replacing the empty save with a save of the current game. The values saved are all the values needed to create a game, including the time left, the number of fortresses left to be destroyed and the difficulty.
To keep track of all of the saves (we implemented three separate save files), we created a static list within GameSave containing instances of GameSave. When the game is saved, the save slot must be referenced in order for GameSave to update the correct element of the list.
SaveScreen is a screen that shows the user the save slots and allows them to save/load a game. SaveScreen references the static list in GameSave, and for each of the three buttons either displays "Save Slot x" or "Empty Slot" depending on whether the corresponding element of the list has been saved to or not. This also allowed us to stop users from loading an empty slot and saving to a slot that already has a save.

Besides from the main implementations requested from the client, the game underwent some minor changes to fulfil different purposes: some were made in order to make GUI more visually pleasing for the user, as well as one which is more focused on the gameplay in order to grant a more interactive gaming experience in toto.

When it comes to gameplay, our team believed that the previous team's minigame mechanics (pipes game triggered every time a truck enters fire station's range) obstructed the game flow: every time a fire truck needed refilling, anywhere between 1 to 3 minutes would be spent on the minigame. Therefore, we added a feature that allows the player to decide when to fill all their trucks at once using a hotkey when they are within fire station range and which, upon mini game completion, would fill all trucks within range. This removed the annoyance of constant uncontrollable interruptions of the gameplay loop by the mini game.

When it comes to GUI, a logical feature to implement is an element being added to the main HUD that appears when the current truck is within range of the fire station, prompting the player to refill the truck.

An extra instruction on the mini game page has also been added, informing players that not all pipes may be needed, as well as that there must be no open connections. This was added as the full rules of the mini game inherited from the previous team who programmed the minigame were not clear enough.

Finally, in order to increase the visual feedback that the player receives from attacking the ET fortresses, we decided to give a fresh look to the fortresses' destruction animation aesthetic. Each Fortress now has six (0 - 5) different .png Texture images associated with it, starting at the stage_0 Texture and end with the stage_5 Texture. There are six increments of HP (100%-80%, 80%-60%, etc.) which upon reached change the Texture accordingly.

References:

[1] “Color psychology: a critical review.”, Whitfield TW1, Wiltshire TJ., Department of Design, Teesside Polytechnic, England, 1990 Nov;116(4):385-411.

[] Schloss, K.B., Palmer, S.E. Aesthetic response to color combinations: preference, harmony, and similarity. Atten Percept Psychophys 73, 551–571 (2011). https://doi.org/10.3758/s13414-010-0027-0