Module	SEPR
Year	2019/20
Assessment	3
Team	Dalai Java
Members	Jack Kershaw, Max Lloyd, James Hau, Yuqing Gong, William Marr, Peter Clark.
Deliverable	Implementation

Implementation Report

Changes to Architecture

We made a few changes to the Architecture outlined by NP Studios in their second assessment as the new features we were required to implement inherently required new classes and relationships between classes.

As we were required to implement the Minigame, we added a MinigameState to the class diagram. Similar to the other states, this class extends the State class and is accessed by the GameStateManager class via the PlayState class. The MinigameState class makes use of another class, Pipe, an extension of the Entity class which was previously mentioned in the abstract architecture of NP Studios' work. (https://npstudios.github.io/files/SEPR_UML_CLass_Diagram_1.pdf). We have added this class to our concrete architecture, adding the 'inPipeRange()' method to check whether the pipe is in range of the map. We designed this functionality in these classes in order to continue the object-oriented structure of the code, and to prevent the existence of a 'god class'.

We have also added another class named 'Explosion' to our architecture diagram - this does not inherit attributes from other classes, however it is called exclusively from the Play state in order to add animation when an Entity is destroyed. This class has been created as part of our attempt to separate the animation away from the core functionality of the game.

We have also modified the Alien class so that it includes a new method, 'moveTowardsFireStation(), which takes as input the vector position of the fire station and uses it as a waypoint to move towards, and we have modified the Firetruck class so that it contains new methods truckMovement() and truckDirection() in order to increase the functionality of our class.

Changes to Requirements

There were some necessary changes to be made to the statement of requirements declared by NP Studios in their second assessment due to the new features that needed to be implemented in our submission for the third assessment.

There were three key features that needed to be implemented for the third assessment; a total of six fortresses across any number of maps, aliens patrolling around the map and the addition of a minigame of a different style to the original game.

To determine how these features should be implemented and what requirements were necessary, during our team meeting we discussed the attributes of each of these features and what these should achieve on an abstract level. We were then able to begin to formulate how this would be translated into code and then implemented into the program itself.

In regards to the implementation of the minigame, the concept was outlined in the abstract architecture report provided to us by NP Studios. As stated above, the game that they had intended to implement was a game in which the player rotates various pipes to create a connection from start to end. This is reflected in the alteration to the user requirement UR_minigame. To help define the properties of the minigame, our requirements document saw the addition of a functional requirement FR_minigame_play which defines that the user must click the pipes to rotate them. Furthermore it saw the addition of

FR_minigame_outcome and FR_minigame_complete which defines the condition that must be achieved for the minigame to reach the ‘complete’ stage and the effect that occurs when the minigame reaches the ‘complete’ state.

For assessment 3 we realised that we needed to add two more user requirements, being UR_enemy and UR_station_refill. The addition of UR_enemy was to create an abstract class for the necessary addition of alien patrols. To add alien patrols into the game we defined various new functional requirements, including FR_alien_move and FR_move_towards_station. These outlined that the aliens would be able to move between positions on the screen and that after a set amount of time, alien patrols should all converge onto the fire station after a defined time. The user requirement UR_station_refill is linked to the new functional requirement FR_no_refill which disallows the user to refill the water supply of the fire truck once the fire engine has been destroyed. We added two new functional requirements linked to UR_instruct_engines - FR_engine_fire and FR_fortress_recover. These allow the user to shoot water at enemy patrols and fortresses and for the fortresses to passively recover health when not being attacked respectively.

We have also re-added UR_save_load_quit, the requirement that the user should be able to save and quit the game, which was removed by NP Studios in Assessment 2. This was partially down to the fact that a number of functional requirements that had not been deleted still linked to the requirement and partially also because we believe it is important to have this as a requirement as it is an important feature of the game. However, we re-defined the requirement such that the saving was automatic and between levels instead of via a dedicated ‘save’ option, and hence we also removed NFR_save which related directly to the prior definition.

Changes to Risks

After viewing the risks defined and updated by NP Studios, we believed them to be a comprehensive cover of all the risks we might encounter. We also believed that the updated likelihood and severity values were correct and that the mitigations given were suitable. Therefore, we made the decision not to edit the risk register.

Changes to Code

To complete the product, we were required to make a number of changes to the existing product. This would result in changes to code, which we were aware could have side effects. In order to mitigate this, we ensured that the most significant changes were made early on in the development process to leave time for rectifying any major errors that these might produce (a mitigation for risk R14_Length), and carried out regression testing (as documented in our Testing report) to ensure that functionality was not adversely affected.

One of the changes we made was the addition of the minigame. The minigame was defined as its own class and as an extension of the State class, as defined in our architecture, and made use of a class called Pipe which we also implemented from the architecture. We have implemented the requirement that the minigame is to take place when the user reaches the fire engine (FR_open_minigame) by launching the minigame (UR_minigame) as soon as a truck re-enters the range of the fire station, and the requirement that the minigame should take less than 5 minutes (NFR_main_focus) by adding a time limit of 60 seconds to our minigame. We purposefully made the minigame

easy and imposed a time limit on it as to not take away from the experience of the main game.

There was an addition of 3 new levels, all of them offering a unique design to provide a different gameplay experience from each other. The design for each level was specifically based around allowing the user the opportunity to implement personalised strategies for beating the level, thus fulfilling UR_strategy. Each of the levels contains one fortress, with a current total of 6 levels which now satisfies FR_6_levels. This fulfils the updated requirement for the current assessment that the game contains a total of 6 alien fortresses.

In terms of modifying the existing code, we have renamed the 'Timer' class to the 'Stopwatch' class. This is due to the fact that java has a Timer class', therefore making it difficult to use both in the same class as they have the same name, making a clear distinction between the two.

We have added some code to the PlayState class which will draw 'RefillWarning.png' onto the screen. This only occurs when one of the playable Fire Engines' water levels goes below 20. The picture is a simple warning to display that the Fire Engine needs to return to the Fire Station to refill its water.. This fulfils the requirement UR_refill_warning as the user can now easily be made aware when they need to refill. Furthermore it was made in a non-intrusive fashion so it does not disrupt gameplay by hindering visibility.

We have extended the Projectile class so that it implements a different way to attack by adding a second constructor. This extra constructor is used to create multiple streams of water randomly. The intention with this additional code was to allow for a bigger depth to gameplay, giving the user more control over any given situation, allowing them to craft a personal strategy for the current state of the game. This new shooting mechanic is invoked by pressing "q". Furthermore we added a small random variance to the bullet direction, marginally increasing the difficulty without affecting the core gameplay. The addition of this class was intended to implement the user requirement, UR_strategy.

We have added some additional functions to the classes FireTruck and PlayState, as laid out in the concrete architecture. The code inside the FireTruck class is the addition of two attributes - maxHealth and currentHealth. There was also the addition of the methods getMaxHealth(), getCurrentHealth(), setCurrentHealth() and updateCurrentHealth(). The intention with the addition of these methods and attributes was to allow the fire engine to 'repair' itself once in range of the fire station which will raise its health up from the current health to the maximum health for each Fire Engine. The maxHealth attribute is necessary as it ensures that the fire trucks health has an upper bound that it can't exceed while still allowing for a unique value for each instance (FR_unique_engines). The code was added into the main PlayState class implements such functionality. (UR_station_refill). We have also moved some extra functionality into the FireTruck class, simplifying the process of controlling each Fire Truck.

Functionality has been added to the Aliens to allow them to patrol the map. Prior to this change, the Aliens would simply hover in place moving back and forth slightly and we felt this did not fit the requirements of the patrols. Now they move around on predefined patrol routes for each level. Halfway through the level timer they all converge on the fire station and destroy it, this was also implemented to better suit the game brief. The method built to implement this is called updateToFireStation() which allows the aliens to converge on the fire station. It takes the position of the fire station as a parameter and using the method, moveAlongGrid(made by NP Studios) begins to path towards the fire station.

There has been an addition of a class called Explosion, implementing the Explosion class we had created within our architecture. This will display an animation over the location of an alien/fire engine when their health is reduced to zero. The purpose of this was to give a visual indication as to the state of the alien/fire engine (alive/destroyed). This now satisfies the functional requirement FR_enemies_die, FR_engine_destroyed.

Whilst testing the game, we also noticed that there was a 'memory leak' on the level select screen - the system was using considerably more memory than expected, an incredibly undesirable feature that made the game unplayable on less powerful machines. We were able to fix this issue by initialising all textures at the start of the LevelSelectState class as opposed to calling them on every 'update()' call, and disposing of them in the dispose method().

All the code we have added has been labelled with a comment, beginning with either 'Dalai Java', 'DJ' or 'Assessment 3' in order to indicate what was added as part of this assessment.

Non-implemented Features

Our game satisfies the vast majority of the user requirements to a high standard. One requirement that is not fully satisfied is FR_engine_destroyed - whilst it is quite clear that a fire engine is destroyed (it displays the 'explosion' animation and is then removed from the screen), there is no specific notification that displays this on the screen.

The non-functional requirement NFR_redability is also not fully implemented as there are a few notices (in particular the 'Need to Refill!' warning) which have relatively small text that cannot be read from 5 metres away, however the decision to make the text small was to avoid obscuring a larger part of the map, thus satisfying NFR_artwork.

There were also a small number of bugs with respect to the minigame. As the screen is filled with random pipes once the route has been found, there is a small chance that two sequences of pipes form a route, one of which will not lead to a complete minigame. Also, when the minigame is won once, the Fire Engine will refill the second time regardless of whether the game is won or lost, thus not completely satisfying FR_minigame_outcome.

Links

Concrete Architecture: https://dalaijava.github.io/files/Architecture.png
Requirements: https://dalaijava.github.io/files/Updated%20Statement%20of%20Requirements.pdf