Implementation Report

Team 12

February 2020

1 Implementation of Architecture and Requirements

Requirements specified in assessment 2 were modified slightly and then strictly implemented. These requirements were changed as they did not encapsulate the full range of requirements needed to fulfil the deliverable. Requirements specified in assessment 1 concatenated with requirements in assessment 2 provided the full range of requirements needed for this deliverable, the full list can be found here. Additionally, some requirements were removed fully following a customer consultation. These requirements were decided to no longer be fitting for the project or the needs of the customer following the design decision to implement a mini-game which would be triggered if a patrol entity were to run into the user during the game (full details of this change can be found in section 3). The transcript of the customer approval for striking specific requirements is linked on our website here.

2 Significant Changes

As part of the third assessment there were two key features that needed to be implemented: ET patrols and the minigame. The sections below show how we added them and the justifications for our decisions.

2.1 ET Patrols

In the second stage of our change management we found that their architectural model was similar to ours. This similarity, going forward in the "Unfreeze" section of the change management allowed us to implement the ET Patrols to match other entities such as the firetrucks or fortresses. Calling them within the game screen without compromising the current architecture or the code.
Following with the style of our predecessors we created two classes for the patrols: Patrol and PatrolType. The latter acts as a kind of constructor class that has different variations of patrols and the former with the key functions and code that they all share.
The requirement FR_PATROL_ATTACK dictated that the patrols should be able to attack fire trucks, we decided that they did this differently than how the the requirement FR_PATROL_DAMAGE specified by entering a minigame with the player whenever they meet a fire truck and having them attack within it. We ensured that we contacted the customer to confirm us removing this requirement.
We decided also to remove the requirement FR_PATROL_SIGHT as we thought it would it would make them too difficult to avoid when controlling four trucks and was therefore hindering the requirement UR_ENJOYABILITY. Again we ensured to contact the customer to confirm the changes since it was a requirement produced from an interview.

2.2 Minigame

In the second stage of the change management we found that the mini-game could be implemented in many different ways. Initially we had two main ideas, we could start a mini-game on fortress encounter or on a patrol encounter. Due to the nature of their architecture and keeping consistent with the requirements FR_AI, the ET patrols and ET fortresses are controlled by the computer AI and FR_FORTRESS_ATTACK, ET fortresses attack trucks. We chose the latter of those ideas which avoided changing the Fortress class and affecting the architecture in an adverse way. After making our decision, before the "Unfreeze" stage, we

contacted our customer to confirm this choice. These fulfill FR_ACCESS_MINIGAME and UR_MINI_GAME since the mini-game can be accessed from the main game.
After deciding how were going to access the mini-game we brain stormed multiple mini game ideas. Ultimately we decided on a rhythm game as we believed it would contribute most to the requirement UR_ENJOYABILITY and wouldn't take too long for the player to complete.

Our mini-game is a rhythm game, we created six classes for the mini-game: DanceScreen, Dancer, DanceManager, DanceChoreographer, DanceMove and DanceResult as well as creating an interface BeatListener. DanceScreen renders the mini-game, changes screen on exit condition and gets inputs from the player. Dancer handles the entities involved in the minigame. DanceManager handles the functionality of the mini-game implementing BeatLister as well. DanceChoreographer handles the moves the player has to press. The requirement UR_MINI_GAME_THEME states that The mini-game should be different in style, but aligned to the theme of the main game. We decided to take the entities and the same colour pallet to use in our mini-game, so we could align it to the main game but at the same time make it a fresh style of gameplay.

2.3 Pause Screen

After establishing most of our key changes in stage three of our change management "Unfreeze", we thought the pause screen was redundant since it didn't fulfill any requirements or a key purpose in the game. So we added functionality that allowed players to change and add routes of fire trucks while paused. We did this by changing existing methods so this didn't have an effect on the architecture. Our reasoning was to make it easier to plan out strategies, especially with four fire trucks being in use at once. This made the game more enjoyable thereby contributed to the UR_ENJOYABILTY requirement.
After introducing the patrols we found that the player ran into them too often. So introducing a more strategical element e.g. a way to avoid them more easily, helped make the game more fun.

2.4 FireTruckType Restructure

FireTruckType and FireTruck to some extent have been slightly restructured for two different purposes. Firstly, the textures used for rendering the fire trucks were initialised within the FireTruck class to be the default sprites which was a problem as it meant they couldn't vary between the different fire truck types and that was a key way of distinguishing between the fire trucks. To solve this we moved the variable to be within FireTruckType so it could vary.
The second change we made was to change the variable speed within FireTruckType from an int to a float so we could vary the speed of the trucks more flexibly.

2.5 Path finding

When testing the game early on in the "freeze" section of our change management we discovered that the path finding system for drawing routes was very inflexible and not able to connect corners that aren't drawn well by the player. This proved to be frustrating when trying to play the game as you would have to keep backtracking to redraw a path that you didn't quite complete properly.
To fix this problem we added a path finding algorithm which would extend the path to add a route between where the mouse left the road and where it reentered. This generally made the game easier to control and contributed to requirements NF_CONTROLS and UR_ENJOYABILITY.
We extended the "FireTruck" class which already handled the movement to implement a breadth-first path finding algorithm. We did this within the "FireTruck" class so we could conserve the previous architecture.

2.6 Automatic Fire

Another feature we decided to change was to make the fire trucks automatically fire instead of firing on a button hold. We made this decision as having to hold down a button for a long period of time can distract players from controlling other trucks and can be a challenge for players with motor disabilities. The

only downside to this decision is that you cannot choose not to fire at an ET fortress but we thought that there were few situations where a player would want that. This change was not specifically affected by any requirements but it does contribute to UR_ENJOYABILITY. This change didn't affect the architecture as we only refactored and reduced on the current code base.

3 New Features

New features were added to support the changes designed to fulfill the requirements and architecture described in the previous section. The tables below describe the key new features that have been implemented in the project and the relation they have to requirements and architecture. Many requirements and architecture links reference back to assessment 1 as these had been removed for the updated version of requirements in assessment 2 due to them not being part of the deliverables at the time. A new amended version of the requirements can be found here here.

3.1 Non-Primitive Data Types

	Files Modified	Requirement	Architecture	Explanation
Patrol	Patrol.java GameScreen.java GameState.java	UR_Patrols (Assessment 1)	Patrols were modified from assessment 1 architecture to an architecture more in line with the changes in assessment 2. The arch-texture mimics closely that of a similar types such as fortresses and firetrucks.	Patrols are designed to move around between 4 preset points and trigger the mini game if they collide with a firetruck. This was added to fulfil the UR_Patrol requirements, further described in the previous section.
PatrolType	PatrolType.java GameScreen.java	UR_Patrols (Assessment 1)	This class was designed to match the architecture of similar classes such as fortresses and firetruck. This type holds the basic blueprint of all patrols.	PatrolType was used to define all the unique stats for each patrol.
BlasterParticle	BlasterParticle.java	N.A	BlasterParticles were not included in the original architecture because the way in which patrols attack was modified with respect to customer requirements in this implementation. Patrols only enter dance attack when they directly run in-to a firetruck and only a specific instance of the patrol may attack the fire station using BlasterParticle.	WaterParticle type was modified to be compatible with alien entities. This particle type will only fire at a station entity and will only originate from the patrol type Station, a special patrol with the unique purpose of dest-rowing the station.
Dancer	Dancer.java DanceScreen.java	UR_Minigame (Assessment 1	Dancer is a data type used by the DancerScreen, our minigame. This was not included in the original architecture as it is an extension of assessment 2.	Dancer is the entities that take part in the dance off minigame. Their health and movement is controlled by this data type making them easy to control from the minigame screen.
DanceMove & Dance Result Enums	DanceMove.java DanceResult.java	N.A	These enum types were implemented within the new dancer architecture not defined in previous assessments.	These enum types were used to define the constants relating to dancing moves, eg. UP, DOWN, LEFT ... and dancing result eg. GREAT, GOOD, LATE ...

3.2 Data Structures

	Files Modified	Requirements	Architecture	Description
CircularLinkedList	Patrol.java CircularLinkedList.java	UR_Patrol (Assessment 1)	This data structure was not described in the original architecture as it was implemented as a way to control patrol movement which was only added in this deliverable.	The implementation of a circular linked list was adapted from Baeldung . This was used in order to allow patrols to continuously rotate through a list of points on the screen without ever reaching the tail of the list. This approach was used as it was based on the already implemented path finding algorithm used by the prevpious developers which implemented the same algorithm with a queue.
DanceChoreographer	DanceChoreographer.java	N.A	N.A	Dance Choreographer is an implementation of a list which returns the next move in the list that the user must perform, these are generated at random.

3.3 Significant Algorithms

	Files Modified	Requirement	Architecture	Description	Complexity
Path Finding	Firetruck.java	UR_Eenjoyability	No architectural changes were made	The path following algorithm was modified to be more user friendly. This was modified from am algorithm which was adding tiles as they were being selected on the map, to an algorithm which would extrapolate points between two tiles using a breadth-first search method allowing the user to perceive a more seamless drawing of tiles on the map.	$O(\|\|V\|\| + \|\|E\|\|)$

4 Unimplemented features

FR_ANIMATION dictated that there should be different stages of damage represented by sprites for the ET fortresses and fire trucks. We did not implement different sprites for the fire trucks as we thought it would make too little of a difference, as the trucks are very small, to be worthwhile implementing over other features. There were some features originally needed for requirements that we have amended, such as FR_VIEW_TIMER which stated there should be a timer counting down to when the fire station is destroyed. We thought that an interesting feature would be for the player to be able to delay the destruction of the fire station by fighting the patrol as you would normally do however this would mean that the timer would have to be set back every time it was delayed so agreed with the customer that it was unnecessary and removed it. The player does still have an idea of how long it will take though as the patrol will be visible on the screen at all times.