Project 1: Part 2
Trurene RPG: Development & Evaluation

Djimon Jayasundera

Table of Contents

Development

Test Plan

Unit Testing

Individual functions will be tested using extensive unit testing. There will be a testing file which inputs known values and checks them against their known correct outputs to test each function. This will be included in the project as *UnitTesting.cs*. Only a few functions have return values, only those will be tested because other functions change the value of the global game state and their outcome are essentially impossible to predict. This is because they are based both on randomness and the user input. For this reason, they will be tested during alpha and beta testing (with real users).

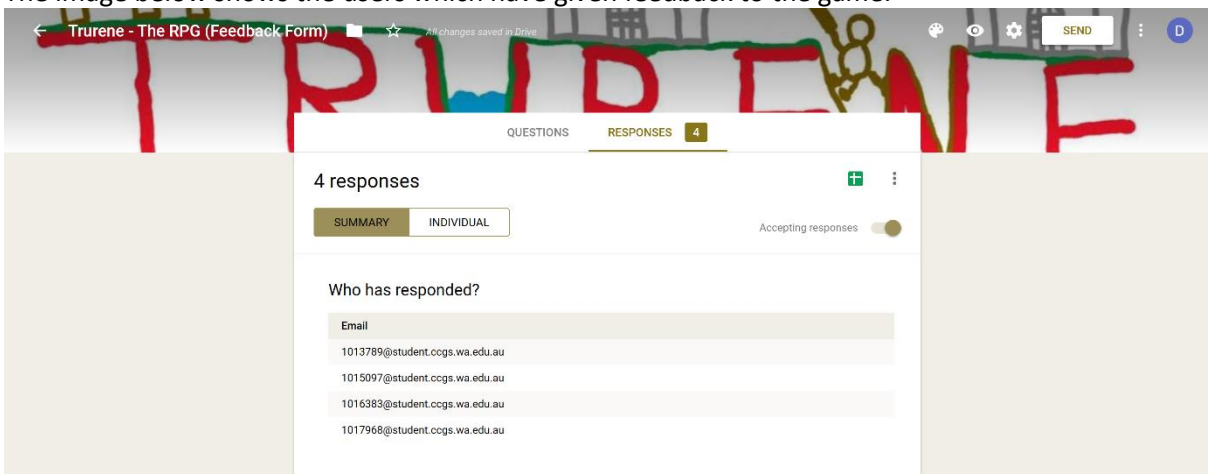
Alpha Testing

This will be done by myself. Normally this is done with a select few users usually the programming team, however in this case it will just be me.

Beta Testing

This will be done by some of the people I know who can run .exe files on their school devices. This includes the year 10 students doing the HS1917 course, and the students doing the year 11 and year 12 Computer Science courses. To help with the beta testing I created a feedback form for them to do.

The image below shows the users which have given feedback to the game:



The following table describes the steps at the beginning of the software before and after the WPF Window is created and destroyed respectively. The input recorded is a simple description of what was typed in. The step name and result are simple descriptions of what the current process is and what happens after the step respectively. This is to make it more general, so that it can encompass a wide range of similar test data. Also, the result is the expected result, since these all work as expected.

Step Name	Input	Result
starting credits	any key	lore step
lore	right arrow	speeds up
new or continue?	"NEW"	world size? step
new or continue?	"CONTINUE"	load step
world size?	number less than 8	world size? step
world size?	number greater than 11	world size? step
world size?	number between 7 and 12	resource pack? step
load?	non-existing filename	load? step
load?	existing filename	resource pack? step
resource pack?	non-existing resource pack	resource pack? step
resource pack?	existing resource pack	start game
ending credits	any key	end

The next table shows test data for some interactions within the game (in a similar style to the one above):

Step Name	Input	Result
	Village	
talk?	[YES] (button)	do NPC interaction
talk?	[NO] (button)	walk away from NPC
	Hunter	
confirmation?	[YES] (button)	do fight with large creature
confirmation?	[NO] (button)	end
	Merchant	
weapon?	"MACE"	gold? step
weapon?	"SWORD"	gold? step
weapon?	"DAGGER"	gold? step
weapon?	invalid entry	weapon? step
gold?	less than current gold	slight penalty, penalty step
gold?	equal to current gold	bonus, penalty step
gold?	more than current gold	penalty, penalty step
gold?	invalid entry	gold? step
penalty	[OK] (button)	penalty message, weapon step
weapon	[OK] (button)	weapon stats, confirm? step
confirm?	[YES] (button)	buy weapon, end
confirm?	[NO] (button)	don't buy weapon, end
	Quester	
currently doing a quest?	yes (user doesn't input this)	end
currently doing a quest?	no (user doesn't input this)	confirm? step
confirm?	[YES] (button)	add quest to map, end
confirm?	[NO] (button)	end

User Documentation

This will be found in the files included with the project and also through the GitHub webpage of this project. This will include a Quick Start Guide, and a Troubleshooting guide. There will also be User Documentation in the program (i.e. tutorial) to help the user play even if they have never read any of the guides. This is not including the documentation of labelled buttons, and notification text in the game.
There is also some extra documentation that can be found on the GitHub repository for the project at https://github.com/Elodin77/Trurene-RPG, this is mentioned multiple times throughout the other user documentation.

Evaluation

System Requirements

Below shows the original system requirements and which had been accomplished at some point during the development of the program.

Basic Functionality

Component	Description
UI in console.	User Interface is implemented through the console as text with only ASCII graphics.
Fixed interactions.	The merchant, quester, and hunter interactions are basic and do not change.
Fixed enemy statistics.	The statistics (e.g. health, attack...) of enemies (e.g. creatures, wolves, and Troll King) are fixed and do not change.

Better Functionality

Component	Description
Save and Load.	The user can save and load the game state through the UI.
Random interactions.	The merchant, quester, and hunter interactions are complex and have an element of randomisation.
Random enemy statistics.	The statistics (e.g. health, attack...) of enemies (e.g. creatures, wolves, and Troll King) have an element of randomisation.
NPC movement algorithm (not randomised).	The Troll King and wolves have an optimal movement algorithm which has no element of randomisation.

Advanced Functionality

Component	Description
NPC movement algorithm (randomised).	The Troll King, and wolves have an optimal movement algorithm which has an element of randomisation.
Dramatic delays.	The software contains intentional delays for dramatic or realistic purpose.
Generated worlds.	The user has the option to start on a new randomly generated world.
Quest messages (random choice).	The message given to the user when they reach a quest location is chosen out of a preset list of messages.

Extended Functionality

Component	Description
Network communication.	The software can upload high scores and maps to a server.
UI simultaneously shows all sections.	The UI simultaneously shows the map view, fighting view, and a console view.
UI has progress bars.	The UI has progress bars for aspects which will benefit from them (e.g. health bars).
UI has colours.	The UI has colours to represent different aspects of the game (e.g. colours for different symbols on the map).
Modular spells.	The number and effects of spells are defined in the game state. This allows for custom amounts and effects of spells.
Quest messages (randomised)	The message shown to the user when they reach a quest location is completely random.

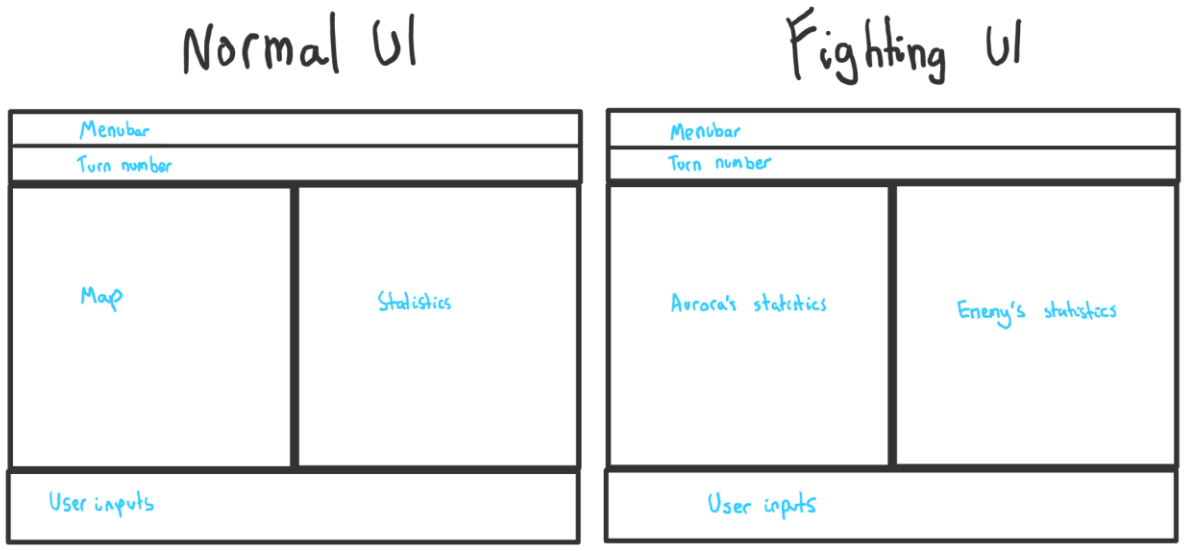
The program meets all of the requirements expected as well as more. It also meets the requirements for the other areas in the original Design section including *Technical requirements*.
Overall, the project successfully meets all of the original requirements, but there is evidence of some feature creep. However, most of these features could be easily justified for making the user experience better when playing Trurene.

Changes to Original Design

Observable Changes

The largest change from the original design would be the fact that the entire UI is a WPF (Windows Processing Form) instead of a Console based UI.
The other main change is the notifications section and the checkboxes which are both on the far right of the UI. This means that the final UI has three columns of information rather than two.
The original design had separate windows for fighting and the map, but the final design shows all of them simultaneously.

The original UI plan was:



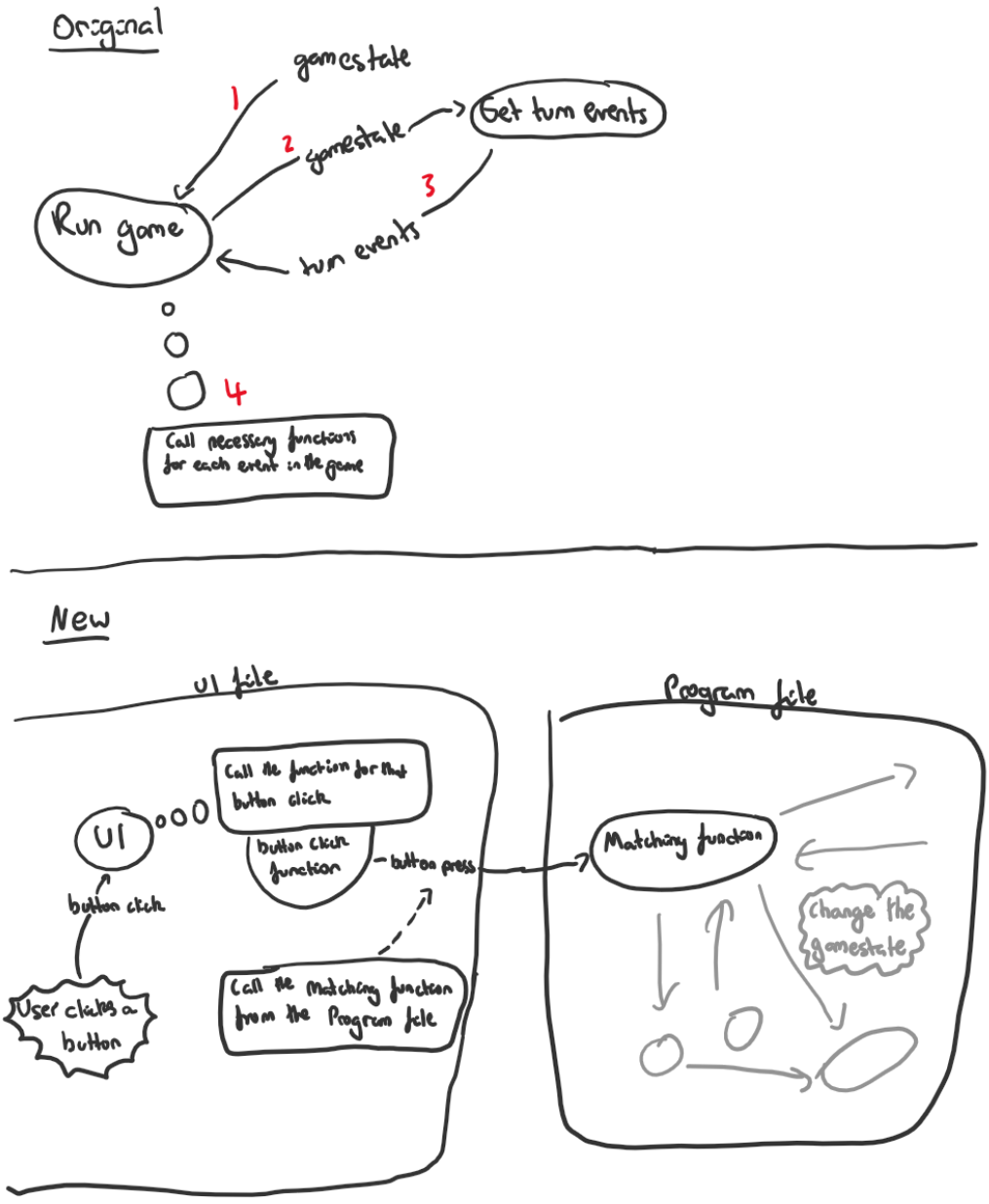
The final UI is:

Algorithm Changes

There is a major change in the way that the program coordinates with the UI which drastically changes the structure chart. The original design had certain functions which executed each in-game turn and called the necessary functions for what happens, such as *Game* (from the original design). The function *DoFight* originally executed the entire fight. However, the use of a WPF prevented the use of this sort of system. This is because once the WPF is started in the main *Program.cs* thread, not a single other line of code in that main function will be executed until after the entire WPF UI closes. Instead of a single main function (i.e. *Game*) to run the game, the WPF needed to do this instead. However, the WPF also has a "run" function which starts it and similarly, nothing else in that thread can be executed until after the UI closes.

All of this means that the entire game has to be coordinated by the clicking of buttons by the user. Two diagrams are shown below to try to illustrate and clarify this idea.

Note: the diagrams are not an accurate representation of the code and do not use any conventions, they were just diagrams used for planning.



The final design has significantly more functions and data structures than the original pseudocode. However, these functions only enhance the gameplay and are not part of the base algorithms. For example; there are many functions associated with the UI, but these are not part of the actual game algorithms. The reason for why they exist is just to enhance the gameplay (or add features).

Many of the functions took the game-state variable as a parameter (*World world*) which is shown both in the structure chart and in the pseudocode. But in the final design this and other variables are global variables instead, meaning that they are not being passed as parameters but able to be accessed at any place in the program. The reason why I did this, was because when implementing the WPF (Windows Processing Form) User Interface, some of the entities such as the progress bars could not access the variable for the game-state (since they are executed in a separate thread to the rest of the program). The only solution for this was to create a global variable for the game-state which was directly accessed and changed by the functions. Since it is a global variable, there was no need to pass it by value or by reference (which would also have worked) to any of the functions.

The data structures originally planned all exist in the final design. However, there are a few additions which were not included in the original design.
The two data structures *TextProperty* and *FormattedText* are both only used to create formatted text (e.g. colour and font) to be displayed in the notifications box on the UI.
The data structure *Creature* was only created to simplify the process of fighting and generating enemies. For example, enemies randomly generated for Aurora to fight will not need a *position*, but they will need a *reward* value. So rather than creating complex ways of storing and altering these based on what Aurora is fighting, the *Creature* data structure is used to simplify the process of fighting.

Potential Improvements

User Friendliness

A potential improvement is more clarity in general (and more obvious directions of what to type). Many of the users in the beta testing had trouble figuring out some aspects of the gameplay especially the fighting mechanics. Another area was at the start in the console, the exact words to type in was sometimes unclear (e.g. choosing the resource packs requires the user to type in "classic" or "pixel art mod" character for character).
Another potential improvement is making the input more user friendly. For example, currently, the user must type certain keywords specifically in CAPITALS rather than in any form. Although this is not too much of a problem, since there are key bindings to allow the user to not have to type in any words.

Aesthetics

The aesthetics of the game can always be improved indefinitely. This includes backgrounds, more icons (e.g. hearts for health) or more sound effects.

Algorithm

The algorithm cannot be upgraded indefinitely without feature creep happening. But it can be improved in some ways, especially the method which the enemies use to fight. Rather than simply striking every time they can, they could use the *wait* option to their advantage and maybe even *retreat* if they are very low on health.

Features

There are very little additional features which can be added without the software having feature creep. This is because the software already has all of the expected requirements as well as some extra requirements. This is already technically a level of feature creep.
Despite this there are still some features which could be added such as a leader board.

Known Bugs

Currently, there are no known bugs.

Bugs which existed in previous versions can be seen by looking through the history of the commits to the GitHub repository to find when they were fixed.

Sources Used

Stack Exchange Inc. (n.d.). *Stack Overflow*. Retrieved from https://stackoverflow.com/.
Atlassian. (n.d.). *Atlassian Support*. Retrieved from https://confluence.atlassian.com/.
C# Corner. (2019). Retrieved from https://www.c-sharpcorner.com/.
Microsoft. (n.d.). *Microsoft Docs*. Retrieved from https://docs.microsoft.com/.
WPF Tutorial. (2007). *The complete WPF Tutorial*. Retrieved from https://www.wpf-tutorial.com/.
Microsoft. (n.d.). *Microsoft | Power Platform*. Retrieved from https://powerusers.microsoft.com/.
Reddit. (n.d.). *Reddit*. Retrieved from https://www.reddit.com/.