Project 1: Part 1
Trurene RPG: Analysis and Design

Djimon Jayasundera

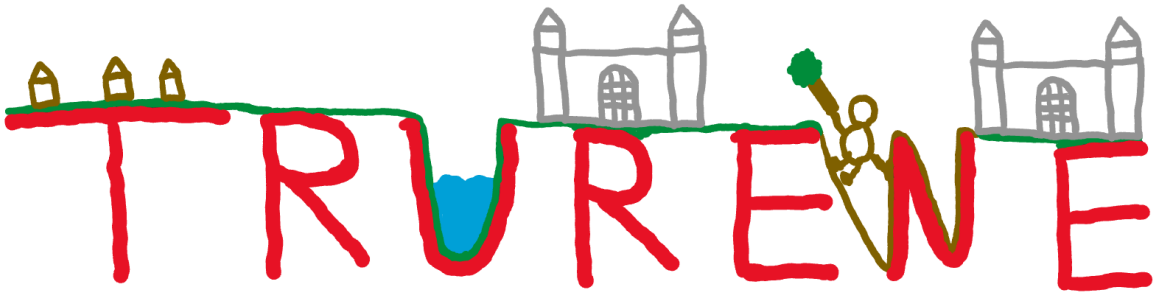
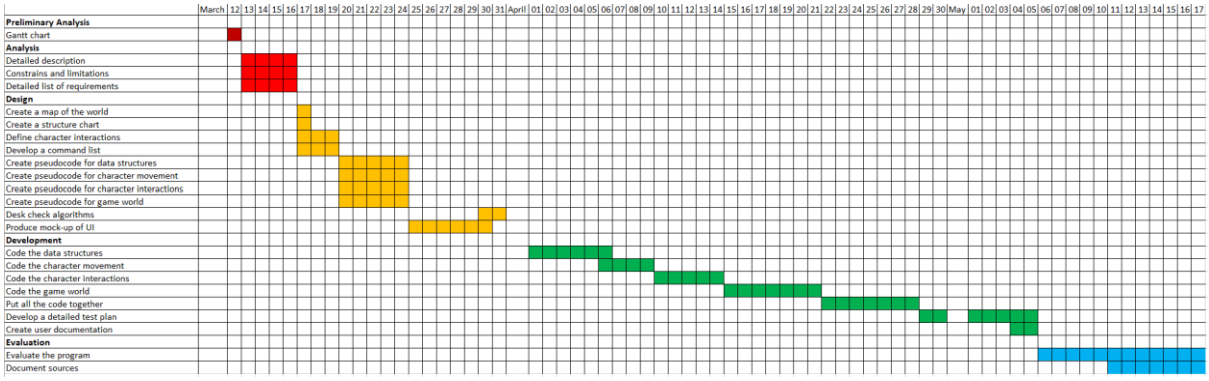


Table of Contents

Preliminary Analysis

Gantt Chart



Analysis

RPG Overview

The User Manual will offer an alternative description designed for a user. It will be found in the User Manual file or on this website: https://elodin77.github.io/Trurene-RPG/
The website will also contain map files and downloads as well as the rules. But the actual GitHub repository will be private (inaccessible) until 17/05/2019.

Synopsis

Lore

The RPG is called Trurene. It is set in a classic medieval fantasy world called Trurene. The background story of the game is as follows:
Aurora was prophesised by powerful elf mages to be the next hero. She was born with a human father and an elvish mother. But after she was born she was taken by the elvish mages and replaced with another human child. The elvish secretly moved her to a remote village in the kingdom. As she grew older she was entrusted into the care of Maeja, who was a powerful elvish mage and taught the art of magic. After she was stronger and faster she was given into the care of an old human warrior, Hawk, who was once an amazing fighter and tracker. He taught her how to use many different weapons. For the past three years, Aurora had been working as a guard for caravans using the dangerous roads far away from the elvish fortress and the human fortress. The areas where Aurora worked had very few guards enforcing the laws and lots of people breaking the laws, even Aurora did not follow any laws. In the recent years tension between the humans and elves had been growing. It started with some racism, then some suspicious deaths, and now they even have separate armies. While the Troll King had been massing an army and raiding farms for food, the humans were blaming the elves for using necromancy on dead trolls and the elves were blaming the humans for forcing their mages to use necromancy to attack the elvish farms with dead trolls. The last time a living troll had been seen was over two hundred years before, but they were defeated with a strong fighter who wielded both metal and magic. The attack was sudden and brutal. The elves and humans may have been able to fight the army of trolls together, but alone they stood no chance. The trolls, being over ten feet, smashed the fortress walls with trees, crushed the trebuchets with their fists, and threw soldiers... at other soldiers. The two great fortresses were destroyed in days. Aurora had to kill the Troll King, she was the only person who could wield metal and magic in the Kingdom of Trurene. If she couldn't do it, nobody could.

Type of Game

Trurene will be a RPG (Role Playing Game) with no multiplayer aspect to it.
The design of Trurene will be a game which will be very easy to learn but very hard to master. This is so that any user can easily learn, play, and understand the mechanics of the game while keeping a genuine competitive element to the game.
It will also be aimed to be a slow-paced strategy game such as a TBS (Turn Based Strategy) rather than a fast-paced skill game such as an RTS (Real Time Strategy) or an RTT (Real Time Tactics). This is to add to the idea behind Trurene being a competitive strategy game, which could realistically be played in tournaments.

Objectives

The user will play as the character Aurora. They will move around the world with the aim of collecting strong enough weapons and spells to defeat the Troll King. The Troll King will destroy the villages in the world one by one in a random order. The aim of the game is to defeat the Troll King in the least number of turns possible while preventing him from destroying every village. If all of the villages are destroyed, the game will be over. If Aurora dies, the game will be over.

Main Character: Aurora

In the RPG the player will be a female elf-human called Aurora. Aurora will have already mastered the sword and bow. Aurora will also be able to learn spells in the game by solving pre-set puzzles in designated locations. Aurora will move around the world by travelling North, South, East or West. Throughout the game, Aurora will be able to obtain different weapons with different statistics as well as spells. Aurora will only be able to have 1 weapon at a time, but unlimited spells.

Main Character: Troll King

The interaction which Aurora has with the Troll King will be hostile, more specifically fighting. The Troll King's location will be hidden until Aurora has learnt every spell. Every two moves that Aurora makes, the Troll King will move one place in one of up to two options towards a designated location. When he arrives at the next designated location he will destroy it. He will also try to avoid Aurora. If she is within two units of him, he will move his army away from her.

Other Characters

The other NPCs are Hawk, Maeja, merchants, questers, and hunters. The interaction which Aurora has with all of these characters will be a non-hostile conversation.
If Aurora meets Hawk, he will tell her the current location of the Troll King. Hawk will always be in a village, except if the village he occupies is destroyed he will move to another random intact village.
Maeja will be a master of magic, who lives alone in a hut. After collecting each spell, Aurora must go to Maeja to learn how to use the spell. Aurora won't be able to acquire a new spell without learning the first one. Aurora also won't be able to use the spell she found until she is taught how to use it by the Maeja.
There could be a merchant, quester or hunter at each village. Two of the three characters will be at each village which will be randomly determined each time Aurora enters a village (so the same village could have different people on different occasions).
Merchants sell weapons with randomly generated statistics for gold. Aurora will say how much gold she is willing to spend on a weapon and the merchant will make an offer of a weapon with an overall strength relative to how much gold Aurora suggested.
Hunters will allow Aurora to move to a square away from the village without taking up a turn. However, it will be guaranteed that Aurora will fight a strong creature. This will mean that Aurora could use the hunter to move away from the village, only to move back onto it the next turn.
Questers (contrary to what their name suggests) will give travellers quests to do for them; which will consist of a location where the traveller must travel to so that they can receive the reward.

Spells

There are 4 spells which Aurora can collect. They will be found in the form of artefacts in shrines. To be able to use them, Aurora will have to take the artefact to Maeja to be taught them. Their effects are:

1. The wielder regenerates 10% of the damage they do.
2. The wielder has +20 health.
3. The opponent of the wielder has -20% accuracy.
4. The wielder knows the location of the Troll King, and the wolves.

The spell artefacts will be named: Vampire Artefact, Troll Artefact, Goblin Artefact, and Omniscience Artefact respectively.
The fourth spell won't be received until the first three have been acquired. But not all four will be necessary to kill the Troll King. Strategic players aiming for a low (lower is better) score may attempt to find and defeat the Troll King with only three of the four spells, two of them, or maybe even one! By not unlocking every spell, the user will be able to fight the Troll King on a much earlier turn.

Movement Events

Every turn Aurora will have to move. Every turn the following processes will happen. If Aurora has zero health at the end of her turn, the game will be over.
If she moves onto the Troll King or the wolves she will begin fighting them and nothing else will happen. Also, if she lands on a shrine, village, or Maeja then those events will happen and nothing else. If neither of those happen, then one of the three events below will happen:

1. There will be a large chance that she will begin fighting a creature with random low statistics, if she wins the fight then she will receive a random low amount of gold (currency).
2. There will be a small chance that she will begin fighting a very strong creature with random high statistics, if she wins the fight then she will receive a high amount gold.
3. It will be possible for neither of those above to happen.

As well as those events, at the end of Aurora's turn, she will be told which direction (N/S/E/W) the Troll King will be (approximately). If she lands on a square which the Troll King was on previously she will find evidence of the Troll King's army being there recently. If the turn number is an even number, the Troll King will move.
At the end of every turn Aurora will be told which direction (N/S/E/W) the wolves are (approximately). Every three turns the wolves will move towards Aurora. If Aurora is within three units of them, she will hear them howling. If she is within one unit of them, she will sense imminent death. If Aurora lands on the wolves she will fight them (escaping will usually be the best option since the wolves will be very powerful). If the wolves land on Aurora she will be ambushed and will take damage equal to the power of the wolves before beginning the fight. If the wolves are dead, they will be created back onto the map in a random position.
At the end of every turn, Aurora will regenerate 10% of her health, the Troll King will regenerate 20% of his health, and the wolves will regenerate 5% of their health.

Places

There will be villages, and shrines.
The villages will be marked with a 'V' on the map. They will be targeted by the Troll King in a random order and destroyed one by one. In each village Aurora will have the option to talk to two of the three people: the merchant, the innkeeper, and the quester.
A destroyed village will be marked with a 'X' on the map.
Shrines will be marked with a 'S' on the map. They will each have a riddle. Each attempt will use up a turn. They will each reward Aurora with a random artefact (she doesn't already have). There will be an exception for the Omniscience Artefact which will only be able to be acquired after every other spell has been learnt. There must be four shrines in every map.

Once a shrine has been solved it will not be able to be visited again. If it is solved it will be marked with a 'R'.
Maeja's location will be marked with a 'M' on the map.
Hawk's location will be marked with a 'H' on the map.
The Troll King will be marked with a 'T' but can only be seen after acquiring all of the spells.
The location of Aurora will be marked with an '@' on the map.

Fighting Mechanics

Every weapon will have a power, accuracy and a time value. Every tick both sides will gain one to their preparedness. When their preparedness reaches the time value for the weapon it will be able to strike. The party will then choose either to strike with the weapon and lose the value of time from their preparedness or to continue increasing their preparedness (which may result in them being able to do two strikes in quick succession).
Every time a party is hit, that player will lose 1 preparedness (this means that no weapon can have a time value of 1).
Each time a party strikes there will be a chance of them hitting the enemy based on their accuracy.
If both parties strike at the same time the accuracy will be used to find out which (if either) party strikes the other. If both hit the opposition then the party with the higher power will decrease the other party's attack permanently (for the rest of the battle) by a factor relative to the stronger party's power.
A party will also be able to try to retreat. The success of a retreat will be based on each party's accuracy.
After each tick each party will be able to choose to either prepare, strike (if they can), or retreat.

Constraints and Limitations

Users

The potential users will be friends, family and students at CCGS with the ability to run executable files. The target audience will not be the public as Trurene is a small student-made game for the purpose of an assignment.
The users will play the game most likely by running a shortcut to the executable file. This is because the game will be part of a folder containing the shortcut and other files required for it. The RPG will then be played in the console as a text-based game.

User Interface (Input/Output)

User Input

The only data the user enters will be one of three options for fighting, one of four options for moving, and a word for solving puzzles.
There will be a space for console input throughout the entire game. There will also be buttons for fighting and moving, however only one of the two will show. The console input will also be able to be used for moving and fighting rather than using the buttons.
There will need to be validation for any commands input into the console but not for the buttons since they have fixed inputs. Solving the puzzles will not need validation either, as it will check the input against possible answers. So, in effect, it validates the data entered already.

System Output

The output will be through changing the text on the user interface. This is done through changing what the map will look like, the values for the statistics of characters, and other alerts which will be displayed. An image of what the user interface will look like can be found under "Design>User Interface>Mock-Up" in this document.

User Friendliness

Help will be available for the user in the program, however this won't be built into the program. Instead it will link the user to the manual on internet. This will make the software more user friendly since the user can follow the steps on the help guides while having their program running.
Help will not always need to be used by the user, since the game will be very easy to play. This will be mainly because Trurene is designed to be very similar to many other already existing classic RPGs. In this way, the user will easily understand how to play. As well as this, the buttons and controls will all be named to be self-explanatory. This will make the game easy to learn but hard to master.
When fighting and moving, the user will not only be able to use the buttons available, but there will also be commands which do the same action. Multiple ways of achieving the same outcome will make using the program easier for the user. As well as these two ways, there will also be a menu bar offering a third way of accomplishing most tasks.

Processing Efficiency

Since this game does not only include the movement of the user, but also the movement of AI (Artificial Intelligence) enemies: a slower speed each turn will create a more dramatic effect as they wait for the AI to move. Also, since the game is turn-based, fast execution is not important or necessary.
The reasonable amount of time for each turn after the user would probably be under 3 seconds. Any more than 3 seconds would be an inconvenience and annoyance for the user. But if the time is too short, it will make the user feel rushed and cause the game to feel fast-paced which it won't be.

Hardware

The game is designed to be run on a PC (personal computer). The reason for this is simply because due to it being written in C#, and compiled into an executable file, it is limited to Windows operating systems. Although the memory and algorithm efficiency does not prevent the program from running on a less powerful device such as a phone, making it compatible on a phone creates many more complexities which cannot be dealt with in the time frame allowed. The program uses Primary Memory of the computer including RAM (Random Access Memory) while the user is playing the game and the Secondary Memory of the computer such as a Hard Drive to save the game state. However, the hard disk will be used exclusively when the user saves or loads the game to prevent the processing efficiency from dropping dramatically (if it is used after every action).

Software

Trurene will be programmed in C#. This is so that the program can be executed using a compiler instead of an interpreter (such as Python). Compilers execute programs significantly faster than interpreters. This is because they execute the entire program all at once, but interpreters execute the program line by line. The advantage of being programmed in a language that uses a compiler, is that being able to execute fast means that consumers are far more confident in the program. However, this limits the program to running on a PC, as it is far more complex to convert it from an executable file to an app which can be run on a mobile device.
The software will be programmed and tested using a computer with the following specifications:

- Windows 10 Education.
- 64-bit operating system.
- 8.00GB RAM.
- Intel(R) Core(TM) i5-7300U CPU @ 2.60-2.71GHz.
- x64-based processor.

It will only be appropriate to have the system requirements for this software to be at least those of the device used to create it. This is because the software is only known to work on that device and only known to be stable on that device.

However, it is possible to reduce some of these requirements by comparing Trurene to other similar computer games and their system requirements. The actual system requirements are stated slightly further on.

Network Communication

Trurene will use network communication. This is for the purpose of uploading any maps which users create themselves to its web server as well as their scores for playing that map. This will be to keep the game competitive. However, this will not influence the processing efficiency since it will only send information at the end of the game. As well as not influencing the processing efficiency of the RPG itself, this method also prevents any issues that could occur with large numbers of users playing Trurene simultaneously.

GIGO

The concept of GIGO (Garbage In, Garbage Out) is the idea that inaccurate or poor-quality input will result in inaccurate or faulty output. GIGO will not influence the processing efficiency. This will be because Trurene will only have limited user inputs, which are validated to prevent the program from outputting any "garbage". This validation will also prevent the program from crashing unexpectantly which would lower the consumer confidence.

Development Requirements

The code for Trurene will first be planned with pseudocode and structure charts. Next it will be written as pseudocode so that it can be easily understand and converted into real code when it is ready. Finally, it will be written in the language of C#. The software must have internal documentation which include comments within the source code and external documentation such as Rulebooks and Troubleshooting Guides.

The Pre-analysis, Analysis and Design parts of the project must be finished by the 11th of April 2019. The finished product including the Development and Evaluation stages must be created by the 17th of May 2019.

Due to the small amount of time available to create the project, the method used to create the game will be RAD (Rapid Application Development). This technique is a form of iterative development (as opposed to linear development). This technique is unique in that a fully functional product is created very quickly, it is then made better over time while users have the option to give feedback. This method is becoming very popular recently, because it can often result in much better-quality products than other methods since the users can give input on the product while it is still in its development stage. However, by using the method the developer must be aware of "feature creep" and make sure that it does not happen. This happens when the developer adds more and more features which may not necessarily be important to the game and instead make it more complicated.

This method also allows for the RPG to be submitted in a fully functional state at any time after its first development even if all of its planned features have not been implemented.

The Gantt chart outlines the time allocated to each area of the project. However, while some areas of the project are being developed, the program will be simultaneously worked on with simple functionality. There are three areas which need to be developed: the fighting area, the map area, and the user input area. As well as these, four main areas of programming need to be completed: the user interactions, the user movement, the main game, and the game world. A consideration of the time that it will take for each section has been used to help create a Gantt Chart. Following the RAD method, these will all be programmed simultaneously while adding function to each one in steps.

Technical Specifications

The game will be programmed in C#, so the skill of programming in C# is necessary. The software being used to program the software will be Visual Studio 2017, so experience in programming using Visual Studio is also necessary.
Potential issues may be with multiple entities (characters) at the same location or at the same location as designated locations. In this case some icons will have to be omitted and only the necessary ones will need to be displayed on the map.
The requirements listed (mainly related to processing efficiency) can be accomplished by not using linear searches. For example when updating the map, rather than using a linear search, it is more efficient to use algorithms which run in O(1) (constant) time. This can be done in this example by storing the locations of village positions separately, so to find the intact ones only the list of their positions needs to be iterated through rather than the entire map.
Trurene must be able to be played on a PC will no problems with processing efficiency. It must have three sections for the UI, one for fighting, one for the map, and one for the user input. The console (in the user input area) will offer alternative methods of playing the game. More information of the processing efficiency requirements can be found in "Analysis>Constraints and Limitations>Processing Efficiency" in this document.
There will be some useful sources for the development of the game in the website shown at the start of this document. This document contains instructions on how the game will be played (rules of it) in "Analysis>RPG Overview" in this document. There is pseudocode which the actual code can be based on in "Design>Data Structures" and "Design>Algorithms" in this document. There is also a structure chart which can be used in "Design>Structure Chart" in this document. Also, a plan of what the user interface will look like can be found in the "Design>User Interface" section.

System Requirements

After a consideration of the Technical Specifications and the Development Requirements, a list of system requirements has been calculated. For this software there are only two operating systems, because only those two can be tested (and confirmed to be stable) with the software in the time constraints.
A comparison between this RPG and other pre-existing games including *StarCraft (1998)*, *Doom 3 (2004)*, *Star Wars: The Force Unleashed (2009)* and their respective system requirements had been used to help create the following system requirements.

Minimum Requirements	
OS (Operating System)	Ubuntu 16 or Windows 10
CPU (Central Processing Unit)	Intel Core 2 Duo at 1.5GHz
Memory	500MB RAM
Hard Drive	500MB free hard disk space
Graphics Hardware	Integrated graphics chip

Data Storage and Security Issues

Game states will be stored in separate files (save file or starting map file). Because the progress is stored in a separate file, it will be able to be manually changed by a user to "cheat" in the RPG. The maps are also stored in a separate file and can also be changed manually.
Despite this, the game will have no security issues. This will be because Trurene was never intended to be a competitive or commercial. The ability for the user to manually change the game state and starting map will allow them to change it to challenge themselves or make it easier if they are struggling. These will both help the user feel more enjoyment from playing the game, which is one of the key aims of the game.

Legal and Ethical Issues

The initial plan for some interactions were changed after some research showed that the original plan could have had potential issues with Copyright and Trademarks. These changes have been marked.

Potential Issues

The software contained the phrase "Winter is coming" within the text in the program. This phrase is trademarked by HBO (Home Box Office). This could have potentially resulted in HBO filing a DMCA (Digital Millennium Copyright Act) against the software. In the past, a service provider called Redbubble had a legal obligation to pull down the work of a 13-year-old-girl's artwork because it had that phrase in it after HBO sent a takedown notice to the website (warning them that there could be a lawsuit as it breached copyright).
However, HBO is unable to claim the rights to the phrase "Winter is coming" in all contexts. The phrase is also so simple that it does not receive very broad protection. If the person making the infringement was trying to purposely evoke a connection, affiliation, or an association with Home Box Office or any of their content.
This software will be in a completely unrelated context to HBO. But if I had other text in the software similar to key dialogue in HBO's content such as "a Lannister always pays his debts", there would have been evidence of the developer trying to create a connection with HBO. After researching this, the initial planned interactions for many of the NPCs in the game have been changed.

Professional Ethics

These are related to anything which is morally and/or ethically wrong but not necessarily illegal. These ensure that the software will not be harmful to the user, even if the software is completely legal. Although some processes which the software might undertake may not be covered by the law (but still be harmful), morally wrong processes will not happen by the software as it would not be ethical. Since the program is open source, it is highly unlikely that any unethical processes can happen to any users running the original code.

Legal Responsibilities

Trurene will not require the user give any of their personal details at all, even down to their name. This will prevent the software from violating the Privacy Act of Australia.
Similarly, the software will not prompt or message the user for any of their contact details such as their email. This will prevent the software from violating the Spam Act of Australia.
Although the software may need the user to enter a "name" for their high score to be recorded. This will not be intended to be their real name.
The software must behave as intended. It will be an open source game, which means that the source code will be available to the public. In this case it will be available through the platform GitHub. This will mean that nothing can be hidden or misrepresented about the software. Through making the code open source, users will be able to decide themselves about the safety of the software and will have no excuse for not knowing what the software is going to do (they can ask a programmer if they do not understand it).

Copyright

This software will have a MIT (Massachusetts Institute of Technology) license. This license will place very little restriction on the copyright of the program and will work well with the program being an open source software. Copies of this type of license can easily be found and the exact license for this software can be found here:
https://github.com/Elodin77/Trurene-RPG/blob/master/LICENSE.txt

This license will prevent the developer or copyright holders being liable for any issues experienced by users. This will protect the developer and copyright holders from being liable for malicious copies of the software being distributed (since users will be able to easily find, copy, alter, and redistribute the source code and executables of the software).
The license also allows users to use the source code with almost complete freedom. Trurene isn't for commercial uses, so a MIT license helps other programmers with their own projects.

User Responsibilities

It is the responsibility of the user to follow the EULA (End User License Agreement) including but not limited to the use of the program. This outlines what the software can and cannot be used for and other rules regarding the way in which a user may use the software.
It is also the responsibility of the user to follow the Copyright License supplied by the developer of the software. This is referring to the rights granted to the developer including, but not limited to the guidelines related to duplication and distribution of the software.
Due to the code being open source, users will be able to copy and alter the code. It will therefore be their responsibility, to not alter the software to be malicious and/or harmful to potential users and proceed by distributing the software possibly as if it was an original. This will not only be unethical in portraying the software and developer to be what they aren't but will also be illegal due to its malicious activity and harmful effects that it may have on other potential users.

Requirements

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.

Design

Game State File

Below is the template:

[rows] [cols] [turnNum] [numVillages] [numShrines]
[questPosRows] [questPosCols]
[hawkPosRows] [hawkPosCols]
[maejaPosRows] [maejaPosCols]
[auroraPosRows] [auroraPosCols]
[auroraHealth]
[auroraMaxHealth]
[auroraAccuracy] [auroraPower] [auroraTime]
[auroraSpell1?] [auroraSpell2?] [auroraSpell3?] [auroraSpell4?]
[auroraGold]
[trollKingPosRows] [trollKingPosCols]
[trollKingHealth]
[trollKingMaxHealth]
[trollKingAccuracy] [trollKingPower] [trollKingTime]
[wolvesPosRows] [wolvesPosCols]
[wolvesHealth]
[wolvesMaxHealth]
[wolvesAccuracy] [wolvesPower] [wolvesTime]
[villagePositions]
[destroyedVillagePositions]
[ShrinePositions]
[ShrineSolved]

An example game state file is shown below:

13 13 0 6 4
-1 -1
-1 -1
11 11
9 9
100
100
50 10 4
0 0 0 0
0
6 6
140
70
80 35 4
1 1
50
50
90 40 2
2 6,6 2,5 9,5 11,9 5,11 5
4 8,5 7,7 5,8 4
0 0 0 0

World Map

The map will be a grid of characters representing entities. The design of this game will allow for there to be different world maps. The world map will be defined in an input file. This will allow for users to create their own maps, share them, and challenge other people with them.
Below shows the map for the example game state file shown earlier:

```
  1234567890123
1 OOOOOOOOOOOOO
2 OWOOOOOOVOOOO
3 OOOOOOVOOOOOO
4 OOOOOOOOOOOOO
5 OOOOOOOOSOOOO
6 OOOOOOOSOVOVO
7 OOVOOOTOOOOOO
8 OOOOOSOOOOOOO
9 OOOOSOOOOOOOO
0 OOOOOVOOO@OOO
1 OOOOOOOOOOOOO
2 OOOOOVOOOOOMO
3 OOOOOOOOOOOOO
```

Notice that the map above shows the Troll King and the wolves, these will be unable to be seen unless Aurora has the Omniscience Artefact. The map also has its column and row numbers loop back around to 0. Although when these are referred to in the game, they will be referred to with their full coordinates. The map will be shown like that to allow for an infinite sized map while keeping the map easy to read.

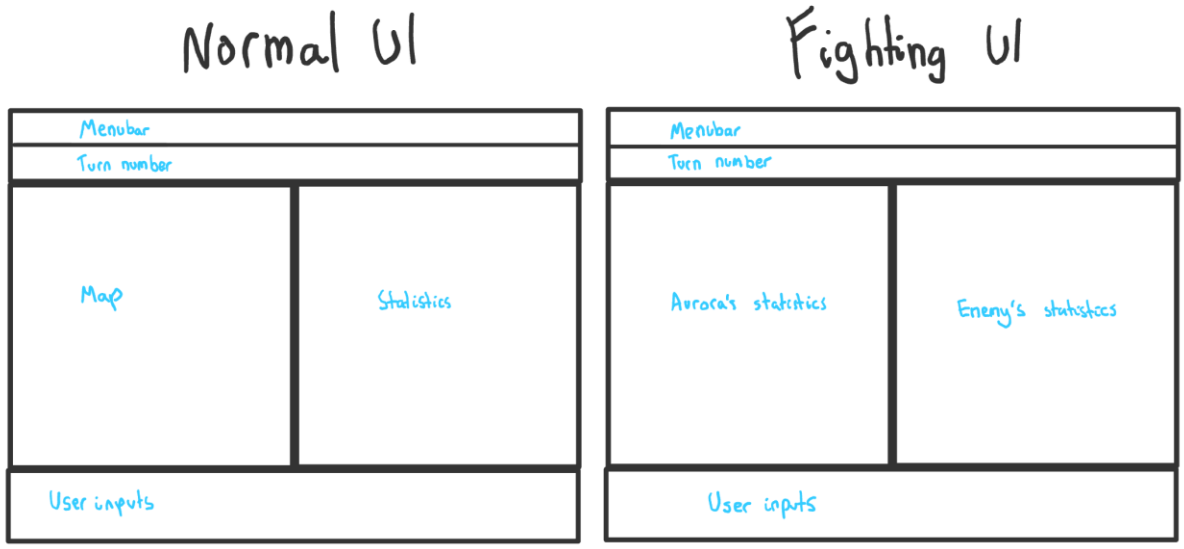
The symbols on the map are as follows:

Symbol	**Description**
W	Wolves
T	Troll King
@	Aurora
M	Maeja
H	Hawk
V	Village
X	Destroyed Village
S	Unsolved Shrine
R	Solved Shrine
Q	Quest Location

On the map, the player will travel back and forth from Maeja to an Unsolved Shrine a few times before travelling towards the Troll King. The player will also travel past villages on their way to collect weapons. Although, the actual places where the player travels to will change.

User Interface

Mock-Up

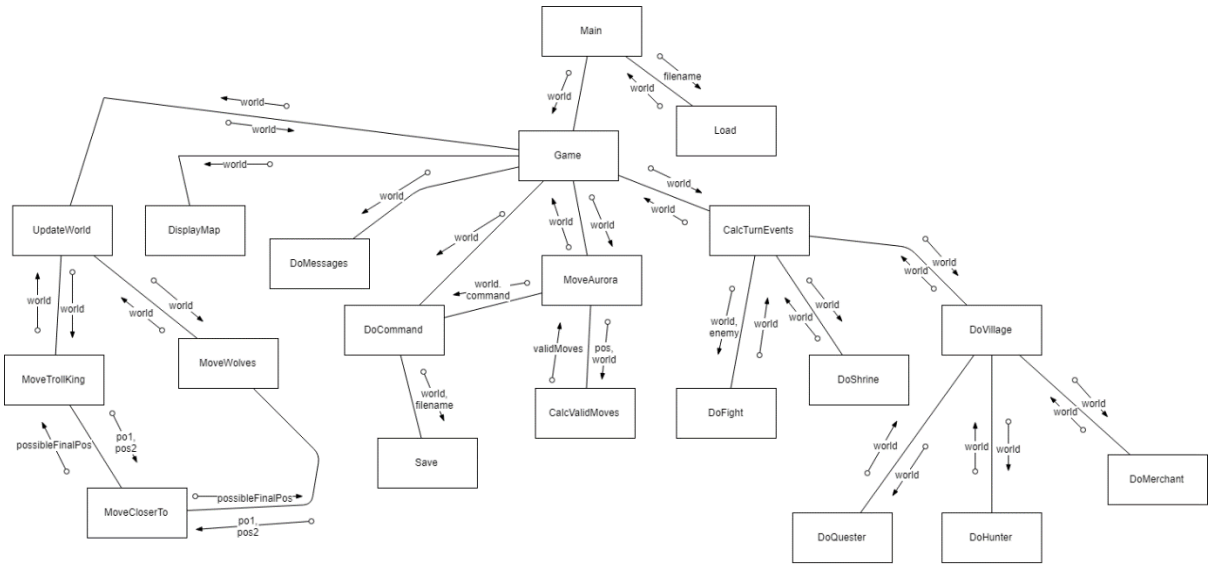


Discussion

The UI (User Interface) must consider the map of the world, the current statistics of characters and creatures, and the interactions (including fighting, dialogue, and commands).
The UI will have two views. One of the views will be for fighting and the other one is the normal UI. The reason for this, is because during a fight the user must focus on the fight and has no reason to want to see the map of the world. For this reason, the fighting view has no map and instead shows the statistics of the enemy. The statistics of both Aurora and the enemy is very user friendly and allows the user to easily compare their statistics with the enemy and decide what option to do during the fight. However, when not fighting the Normal UI will show instead.
Both views of the UI have a menu bar. This allows for multiple ways of accomplishing the same task making it more user friendly and giving them tools which they would be familiar with from other software.
The UI also has a section for user inputs. This section will contain a text box for commands, a section for saving and loading, a section for moving, and a section for fighting. The sections for moving and fighting will only contain buttons. They will also be above the others. This is because they will be referred to more often than the other sections. The command section allows the user a third way of accomplishing most tasks.
The use of buttons will prevent any occurrences of GIGO, but the data entered into the text box still must be validated.
The menu bar on the UI will make it significantly easier for the user. It will offer help guides for the user if there is anything that they do not understand and a troubleshooting guide as well for fixing common issues. However, these will not be within the program. Instead there will be a copy of them within the software package as well as on the internet available for them to see. The reason for this is to make it more user friendly, by allowing the user to simultaneously refer to the User Manual or Troubleshooting Guide while simultaneously having the software executing.
The turn number of the game is shown directly under the menu bar. The reason for this is because the turn number is arguably the most important aspect of the game. Since it determines how well you did at completing the game.

Structure Chart

The chart of data being passed between functions is shown below:



Interactions (Examples)

Numbered lists indicate that one of the messages will be shown.
<> indicates a variable.
[] shows an instruction.
Words with a ~~strikethrough~~ indicate that they have been changed from how they were initially planned to <u>underlined words</u>. These happened after research on legal issues was done.
These interactions do not have pseudocode because they are very likely to change, and not complicated in comparison to the other more important functions.

Fighting

Tick: <tick>
<Enemy name>
    Max Health: <Enemy max health>
    Health: <Enemy health>
    Accuracy: <Enemy accuracy>
    Power: <Enemy power>
    Time: <Enemy time>
Aurora
    Max Health: <Aurora max health>
    Health: <Aurora health>
    Accuracy: <Aurora accuracy>
    Power: <Aurora power>
    Time: <Aurora time>
Action ('STRIKE'/'PREPARE'/'RETREAT'): <user input>

1) Invalid entry. [back]
2) The enemy tries to: <enemy action>

...
<winner> has won the battle!

1) You have acquired <reward> gold!
2) You are dead!

Maeja

1) You have now harnessed the power of the <spell artefact>!
2) There is nothing for me to teach you!

Hawk

My ~~little birds~~ spies have told me that the Troll King is hiding at <location>.

Quester

1) I don't trust you. You haven't completed your last quest, so how can I be sure that you will complete this one
2) Here, I marked on your map where I want you to go. My friend will find you when you arrive. You will be rewarded~~, a quester always pays his debts~~.

Merchant

What weapon do you want ('MACE'/'SWORD'/'DAGGER'): <user input>

1) Sorry, but unfortunately, I don't sell that. [back]
2) Yes, I can get you one of them.

But first show me your gold: <user input>

1) Sorry, but you can't see the weapon until you tell me how much gold you'll spend. [back]
2) Hey! You trying to cheat me? I see <gold> gold, the rest isn't gold! I hate liars.
3) I'm sure you got more gold than that. Think you're better than me? I don't like you.
4) I love honest people. I'll go get a nice weapon for you.

I found a nice weapon for you.

I think you could hit <accuracy>% shots, with <power> strength. Although it would probably take you <time> seconds to prepare.

Do you want to buy this weapon ('YES'/'NO'): <user input>

1) Here it is. A better weapon than you deserve.
2) Here is your weapon. One of my best.
3) Well that is a shame. You seemed like a nice fella. I have other customers so please excuse me.
4) Well go on and get out of here. I have better customers who don't just waste my time.

Hunter

Are you sure that you want me to find you some nice loot ('YES'/'NO'): <user input>

1) I hope you find some good loot... and survive. [Begin fighting]
2) I wouldn't go out there either. ~~Winter is coming, and the deep snow makes it hard to fight.~~

Commands

Following are commands which can be used most of the time.

Command	**Description**
/hawk	Gives the location of Hawk.
/maeja	Says if Aurora must go to Maeja before getting another spell, and which spells she has learnt.
/aurora	Displays information on Aurora.
/wolves	Displays information on wolves (if Aurora has Omniscience artefact).
/trollking	Displays information on the Troll King (if Aurora has Omniscience artefact).
/save <filename>	Saves the game to file.
/quit	Quits the game.

Data Structures (Pseudocode)

```
RECORD World
        // Misc variables
        rows <- 0
        cols <- 0
        turnNum <- 0
        numVillages <- 0
        numShrines <- 0
        // Locations
        questPos <- NEW Pos // (-1, -1) if no quest
        maejaPos <- NEW Pos
        hawkPos <- NEW Pos // (-1, -1) if not created
        villagePositions <- [] // Contains items of record Pos (shows intact villages)
        destroyedVillages <- [] // Contains items of record Pos (shows destroyed villages)
        shrines <- [] // Contains items of the record Shrine
        targetVillagePos <- NEW Pos
        // Characters
        aurora <- NEW Character
        trollKing <- NEW Character
        wolves <- NEW Character
        // Aurora-specific variables
        gold <- 0
        spells <- [0,0,0,0] // 0=not acquired, 1=not learnt, 2=learnt

RECORD Character
        pos <- NEW Pos
        health <- 0
        maxHealth <- 100
        attack <- [0,0,0] // accuracy, power, time

RECORD Creature
        health <- 0
        maxHealth <- 50
        attack <- [0,0,0] // accuracy, power, time
        reward <- 10

RECORD Shrine
        pos <- NEW Pos
        solved <- 0 // 0 is unsolved, 1 is solved

RECORD Pos
        row <- 0
        col <- 0
```

Algorithms (Pseudocode)

Only the important functions are completed with pseudocode. The rest of the functions have comments showing the steps which they will undertake. Some functions are also not shown.
Also, the pseudocode contains many "magic numbers" that may or may not be explained. They are constants which will all be defined at the start of the program, this allows them to be fine-tuned to balance the game.
Also, the data type is only specified before the variable name if it is unclear to prevent the line from being too crowded, long, and hard to read.
The pseudocode for fighting and other dialogue interactions do not have pseudocode because they are simple math (for fighting) and very likely to change (for dialogue).

Main

```
MODULE Main()
	// Show background info, and tell user that user manual can be found in folder
	// Load the game
	INPUT(filename)
	world <- Load(filename)
	// Start the game
	Game(world)
	OUTPUT(world.turnNum) // Output score

END Main
MODULE Game(world)
	// The world in the parameter must be a generated/read one which is valid
	BOOL trollKingAlive = world.trollKing.health>0
	BOOL auroraAlive = world.aurora.health>0
	WHILE auroraAlive AND trollKingAlive AND world.villagePositions.LENGTH>0
		world <- UpdateWorld(world) // Update the world
		DisplayMap(world) // Calculates and outputs the world
		DoMessages(world)
		INPUT(command)
		WHILE command != "MOVE"
			DoCommand(world,command)
			INPUT(command)
		END WHILE
		world <- MoveAurora(world)
		world <- CalcTurnEvents(world)
		trollKingAlive <- world.trollKing.health>0 // Boolean
		auroraAlive <- world.aurora.health>0 // Boolean

	END WHILE
	// Doesn't return anything
END Main

MODULE DisplayMap(world)
	map <- [["O"*world.cols]*world.rows]
	FOREACH (pos IN world.villagePositions)
		map[pos.row][pos.col] <- "V"
	END FOREACH
	FOREACH (pos IN world.destroyVillages)
		map[pos.row][pos.col] <- "X"
	END FOREACH

	OUTPUT("   ",) // Outputs without newline at end
	FOR (col=0 TO world.cols)
		OUTPUT(col%10,) // Outputs without newline at end
	ENDFOR
	FOR (row=0 TO world.rows)
		OUTPUT(row+1,) // Outputs without newline at end
		OUTPUT(map[row])
	ENDFOR
	// Returns nothing
END DisplayMap

MODULE Save(world, filename)
	//Open file
	// Write values from world into file
	// Returns nothing
END Save

MODULE Load(filename)
	world <- NEW World
	// Open file
	// Read in values from file and assign them to world.
```

```
        // Validate the world by making sure:
        //      shrines, villages, and Maeja are all on different squares.
        //      making sure data types are valid (by having an exception case if there is an error)

        RETURN world

END Load
```

User Movement

```
MODULE MoveAurora(world)
        INPUT(command)
        WHILE (command NOT IN CalcValidMoves(world.aurora.pos,world) AND command NOT IN commands)
                OUTPUT("That is an invalid entry!")
                INPUT(command)
        END WHILE

        IF (command IN commands)
                DoCommand(world,command)
        ELSE
                world.aurora.pos.row <- command[0]
                world.aurora.pos.col <- command[1]
        ENDIF
        RETURN world

END MoveAurora

MODULE CalcValidMoves(pos,world)
        validMoves <- []
        IF (pos.row < world.rows-1)
                APPEND [pos.row+1,pos.col] TO validMoves
        ENDIF
        IF (pos.row > 0)
                APPEND [pos.row-1,pos.col] TO validMoves
        ENDIF
        IF (pos.cols < world.cols-1)
                APPEND [pos.row,pos.col+1] TO validMoves
        ENDIF
        IF (pos.cols > 0)
                APPEND [pos.row,pos.col-1] TO validMoves
        ENDIF
        RETURN validMoves
END CalcValidMoves
```

User Interactions

```
MODULE CalcTurnEvents(world)
        somethingHappened <- FALSE
        // Check if Aurora fights the Troll King
        IF (world.aurora.pos = world.trollKing.pos)
                world <- DoFight(world,world.trollKing.attack)
                somethingHappened <- TRUE
        // Otherwise, check if she enters a village
        // This way, she cannot enter a village which has just been destroyed the same turn
        ELSE IF (world.aurora.pos IN world.villagePositions)
                world <- DoVillage(world)
                somethingHappened <- TRUE
        ENDIF
        // Check if Aurora fights the wolves
        IF (world.aurora.pos = world.wolves.pos)
                world <- DoFight(world,world.wolves.attack)
                somethingHappened <- TRUE
        ENDIF
        // Check if Aurora enters a shrine
```

```
FOREACH shrine in world.shrines
        IF world.aurora.pos = shrine.pos
                world <- DoShrine(world)
                somethingHappened <- TRUE
                BREAK
        ENDIF
END FOREACH
// Check if Aurora fights a small creature
IF NOT somethingHappened
        // 0.2 represents the probability, the function returns a value TRUE or FALSE (0 or 1)
        // Constants like this must be tuned so that the game is balanced
        IF RANDOM(0.2)
                fightCreature <- 1
        ELSE
                fightCreature <- 0
        ENDIF
ENDIF
// Check if Aurora fights a large creature
IF NOT somethingHappened and NOT fightCreature
        IF RANDOM(0.05)
                fightCreature <- 2
        ENDIF
ENDIF
IF fightCreature = 1
        Creature smallCreature <- NEW Creature
        smallCreature.maxHealth <- RANDOM(15 TO 25)
        smallCreature.health <- smallCreature.maxHealth
        smallCreature.attack[0] <- RANDOM(40 TO 100)
        smallCreature.attack[1] <- RANDOM(5 TO 15)
        smallCreature.attack[2] <- RANDOM(3 TO 5)
        // 0.5 is the ratio of creature power to gold reward
        smallCreature.reward <- MATH.FLOOR(smallCreature.attack[1] * 0.5)
        world <- DoFight(world, smallCreature)

ENDIF
IF fightCreature = 2
        Creature largeCreature <- NEW Creature
        largeCreature.maxHealth <- RANDOM(25 TO 75)
        largeCreature.health <- largeCreature.maxHealth
        largeCreature.attack[0] <- RANDOM(60 TO 100)
        largeCreature.attack[1] <- RANDOM(15 TO 25)
        largeCreature.attack[2] <- RANDOM(3 TO 5)
        largeCreature.reward <- largeCreature.attack[1] * 0.5
        world <- DoFight(world,largeCreature)
ENDIF
// Regenerate Aurora's health
IF world.aurora.health > 0 AND world.aurora.health < world.aurora.maxHealth
        world.aurora.health <- MATH.FLOOR(world.aurora.maxHealth * 1.1)
        IF world.aurora.health > world.aurora.maxHealth
                world.aurora.health <- world.aurora.maxHealth
        ENDIF
ENDIF
// Regenerate the Troll King's health
IF world.trollKing.health > 0 AND world.trollKing.health < world.trollKing.maxHealth
        world.trollKing.health <- MATH.FLOOR(world.trollKing.maxHealth * 1.2)
        IF world.trollKing.health > world.trollKing.maxHealth
                world.trollKing.health <- world.trollKing.maxHealth
        ENDIF
ENDIF
// Regenerate the wolves' health
IF world.wolves.health > 0 AND world.wolves.health < world.wolves.maxHealth
        world.wolves.health <- MATH.FLOOR(world.wolves.maxHealth * 1.05)
        IF world.wolves.health > world.wolves.maxHealth
```

```
                        world.wolves.health <- world.wolves.maxHealth
                ENDIF
        ELSE
                // Respawn the wolves because they were killed by Aurora
                world.wolves.health <- world.wolves.maxHealth
                world.wolves.pos.row <- RANDOM(0 TO world.rows-1)
                world.wolves.pos.col <- RANDOM(0 TO world.cols-1)
        ENDIF
        // Check if Aurora is on a quest location
        IF world.aurora.pos = world.questPos
                world.questPos <- [-1,-1]
                world.gold <- world.gold + RANDOM(30 TO 40)
                OUTPUT("The friend walked up to you and asked you to do a few jobs for them.")
                OUTPUT("Then they gave you a small pouch of gold. Wow.")
        ENDIF
        RETURN world
END CalcTurnEvents

MODULE DoFight(world,enemy)
        // This function will have two declarations with the same name but different parameters and content.
        //      One of them will be for if enemy is of the class Character (for Troll King and wolves)
        //      The other one will be for if enemy is of the class Creature (for fighting small or large creatures)
        // Do the fight
        RETURN world
END DoFight

MODULE DoShrine(world)
        // Check that the shrine at the location of Aurora is unsolved
        // Check that the user has no spells which they haven't learnt yet
        // Generate a random spell out of the remaining ones
        RETURN world
END DoShrine

MODULE DoVillage(world)
        // Check if Hawk is there
        // If Hawk is there, tell Aurora the location of the Troll King
        // Generate random 2 people out of quester, hunter and merchant
        // Ask which one Aurora wants to speak to
        // Run the function for the person Aurora wants to speak to
        RETURN world
END DoVillage

MODULE DoQuester(world)
        // Check if there is already a quest
        // If there is one ask Aurora if she wants to forget about her current quest
        // Find valid position with nothing on the square.
        // Tell a story about the quest and give keyword.
        // Add new quest to map with a "Q"
        RETURN world
END DoQuester

MODULE DoHunter(world)
        // Run the hunter interaction
        RETURN world
END DoHunter

MODULE DoMerchant(world)
        // Run the merchant interaction
        RETURN world
END DoMerchant

MODULE DoMessages(world)
        // Print the direction of the Troll King
```

```
        // If the proximity is in the given range give a warning for Troll King
        // If the proximity with the wolves is in the given range give a warning
        // Doesn't return anything
END DoMessages
```

Game World and NPC Movement

```
MODULE UpdateWorld(world)
        // Destroy Villages
        IF world.trollKing.pos = world.targetVillagePos
                APPEND world.trollking.pos TO world.destroyedVillages
                world.targetVillagePos <- RANDOM CHOICE FROM world.villagePositions
        ENDIF
        // Move Hawk if he is on a destroyed village
        IF world.hawkPos IN world.destroyedVillages
                hawkPos <- RANDOM CHOICE FROM world.villagePositions
        ENDIF
        // Move NPCs
        IF world.turnNum % 2 = 0
                world <- MoveTrollKing(world)
        ENDIF
        IF world.turnNum % 3 = 0
                world <- MoveWolves(world)
        ENDIF
        RETURN world
END UpdateWorld

MODULE MoveTrollKing(world)
        // Move the Troll King away from Aurora if he is within the proximity range
        // Otherwise move him towards the target village
        // Pick a random item from the possible moves towards the target village
        RETURN world

MODULE MoveWolves(world)
        // Move the wolves towards Aurora
        RETURN world
END MoveWolves

MODULE MoveCloserTo(pos1,pos2)
        // Works out entity1's final positions after moving towards entity2
        possibleFinalPos <- [] // Will contain possible movements for the Troll King
        IF pos1.row < pos2.row
                finalPos <- [pos1.row+1,pos1.col]
                APPEND finalPos TO possibleFinalPos
        ELSE IF pos1.row > pos2.row
                finalPos <- [pos1.row-1,pos1.col]
                APPEND finalPos TO possibleFinalPos
        ENDIF
        IF pos1.col < pos2.col
                finalPos <- [pos1.row,pos1.col+1]
                APPEND finalPos TO possibleFinalPos
        ELSE IF pos1.col > pos2.col
                finalPos <- [pos1.row,pos1.col-1]
                APPEND finalPos TO possibleFinalPos
        ENDIF
        RETURN possibleFinalPos
END MoveCloserTo

MODULE DoCommand(world,command)
        // Runs the command, command is a string
        // print stuff and do commands
        // Doesn't return anything
END DoCommand
```

Desk Checking

Only the functions which have the most complicated logic (and affect User Movement and User Interactions) are being checked with trace tables. CalcValidMoves and MoveCloserTo is important for the movement of every character. CalcTurnEvents is the largest and most complicated function in the program.

A number of changes have been made to make this process feasible and possible:

1. The functions which are called by CalcTurnEvents which have no pseudocode (and just have stubs instead) are simply given arbitrary returned values. This is because the "stub functions" are simple, and not important to the program.
2. Because some of the parameters being passed into the functions are large, similar values for world will be used in every test with minor changes. The value for world was also customised and not necessarily possible to exist so that as many lines as possible can be tested with each desk check.
3. Whenever a test is made with different values for world, rather than showing it all over again, the individual parts that have changed will be outlined.
4. Often "..." is shown for simplicity to represent the text previous (usually the line above).
5. The normal format for a trace table cannot be used (due to obvious reasons). I came up with an alternative way to desk check which changes the result of previous statements to the new statements. This is shown more clearly when looking at the actual "trace tables".

The input file with the template is shown below for reference (although note that the names in square brackets are not the actual names of the variables):

Values for World world	
[rows] [cols] [turnNum] [numVillages] [numShrines]	13 13 0 6 4
[questPosRows] [questPosCols]	0 0
[hawkPosRows] [hawkPosCols]	2 6
[maejaPosRows] [maejaPosCols]	11 11
[auroraPosRows] [auroraPosCols]	9 9
[auroraHealth]	100
[auroraMaxHealth]	100
[auroraAccuracy] [auroraPower] [auroraTime]	50 10 4
[auroraSpell1?] [auroraSpell2?] [auroraSpell3?] [auroraSpell4?]	0 0 0 0
[auroraGold]	0
[trollKingPosRows] [trollKingPosCols]	6 6
[trollKingHealth]	140
[trollKingMaxHealth]	70
[trollKingAccuracy] [trollKingPower] [trollKingTime]	80 35 4
[wolvesPosRows] [wolvesPosCols]	1 1
[wolvesHealth]	50
[wolvesMaxHealth]	50
[wolvesAccuracy] [wolvesPower] [wolvesTime]	90 40 2
[villagePositions]	2 6,6 2,5 9,5 11,9 5,11 5
[destroyedVillagePositions]	
[ShrinePositions]	4 8,5 7,7 5,8 4
[ShrineSolved]	0 0 0 0

The code for the functions being tested (with line numbers) are shown below:

```
1. MODULE CalcTurnEvents(world)
2.     somethingHappened <- FALSE
3.     // Check if Aurora fights the Troll King
4.     IF (world.aurora.pos = world.trollKing.pos)
5.         world <- DoFight(world,world.trollKing.attack)
6.         somethingHappened <- TRUE
7.     // Otherwise, check if she enters a village
8.     // This way, she cannot enter a village which has just been destroyed the same turn
9.     ELSE IF (world.aurora.pos IN world.villagePositions)
10.        world <- DoVillage(world)
11.        somethingHappened <- TRUE
12.    ENDIF
13.    // Check if Aurora fights the wolves
14.    IF (world.aurora.pos = world.wolves.pos)
15.        world <- DoFight(world,world.wolves.attack)
16.        somethingHappened <- TRUE
17.    ENDIF
18.    // Check if Aurora enters a shrine
19.    FOREACH shrine in world.shrines
20.        IF world.aurora.pos = shrine.pos
21.            world <- DoShrine(world)
22.            somethingHappened <- TRUE
23.            BREAK
24.        ENDIF
25.    END FOREACH
26.    // Check if Aurora fights a small creature
27.    IF NOT somethingHappened
28.        // 0.2 represents the probability, the function returns a value TRUE or FALSE (0 or 1)
29.        // Constants like this must be tuned so that the game is balanced
30.        IF RANDOM(0.2)
31.            fightCreature <- 1
32.        ELSE
33.            fightCreature <- 0
34.        ENDIF
35.    ENDIF
36.    // Check if Aurora fights a large creature
37.    IF NOT somethingHappened and NOT fightCreature
38.        IF RANDOM(0.05)
39.            fightCreature <- 2
40.        ENDIF
41.    ENDIF
42.    IF fightCreature = 1
43.        Creature smallCreature <- NEW Creature
44.        smallCreature.maxHealth <- RANDOM(15 TO 25)
45.        smallCreature.health <- smallCreature.maxHealth
46.        smallCreature.attack[0] <- RANDOM(40 TO 100)
47.        smallCreature.attack[1] <- RANDOM(5 TO 15)
48.        smallCreature.attack[2] <- RANDOM(3 TO 5)
49.        // 0.5 is the ratio of creature power to gold reward
50.        smallCreature.reward <- MATH.FLOOR(smallCreature.attack[1] * 0.5)
51.        world <- DoFight(world, smallCreature)
52.
53.    ENDIF
54.    IF fightCreature = 2
55.        Creature largeCreature <- NEW Creature
56.        largeCreature.maxHealth <- RANDOM(25 TO 75)
57.        largeCreature.health <- largeCreature.maxHealth
58.        largeCreature.attack[0] <- RANDOM(60 TO 100)
59.        largeCreature.attack[1] <- RANDOM(15 TO 25)
60.        largeCreature.attack[2] <- RANDOM(3 TO 5)
61.        largeCreature.reward <- largeCreature.attack[1] * 0.5
62.        world <- DoFight(world,largeCreature)
63.    ENDIF
64.    // Regenerate Aurora's health
65.    IF world.aurora.health > 0 AND world.aurora.health < world.aurora.maxHealth
66.        world.aurora.health <- MATH.FLOOR(world.aurora.maxHealth * 1.1)
67.        IF world.aurora.health > world.aurora.maxHealth
68.            world.aurora.health <- world.aurora.maxHealth
69.        ENDIF
70.    ENDIF
```

```
71.         // Regenerate the Troll King's health
72.         IF world.trollKing.health > 0 AND world.trollKing.health < world.trollKing.maxHealth
73.                 world.trollKing.health <- MATH.FLOOR(world.trollKing.maxHealth * 1.2)
74.                 IF world.trollKing.health > world.trollKing.maxHealth
75.                         world.trollKing.health <- world.trollKing.maxHealth
76.                 ENDIF
77.         ENDIF
78.         // Regenerate the wolves' health
79.         IF world.wolves.health > 0 AND world.wolves.health < world.wolves.maxHealth
80.                 world.wolves.health <- MATH.FLOOR(world.wolves.maxHealth * 1.05)
81.                 IF world.wolves.health > world.wolves.maxHealth
82.                         world.wolves.health <- world.wolves.maxHealth
83.                 ENDIF
84.         ELSE
85.                 // Respawn the wolves because they were killed by Aurora
86.                 world.wolves.health <- world.wolves.maxHealth
87.                 world.wolves.pos.row <- RANDOM(0 TO world.rows-1)
88.                 world.wolves.pos.col <- RANDOM(0 TO world.cols-1)
89.         ENDIF
90.         // Check if Aurora is on a quest location
91.         IF world.aurora.pos = world.questPos
92.                 world.questPos <- [-1,-1]
93.                 world.gold <- world.gold + RANDOM(30 TO 40)
94.                 OUTPUT("The friend walked up to you and asked you to do a few jobs for them.")
95.                 OUTPUT("Then they gave you a small pouch of gold. Wow.")
96.         ENDIF
97.         RETURN world
98. END CalcTurnEvents

1. MODULE CalcValidMoves(pos,world)
2.         validMoves <- []
3.         IF (pos.row < world.rows-1)
4.                 APPEND [pos.row+1,pos.col] TO validMoves
5.         ENDIF
6.         IF (pos.row > 0)
7.                 APPEND [pos.row-1,pos.col] TO validMoves
8.         ENDIF
9.         IF (pos.col < world.cols-1)
10.                 APPEND [pos.row,pos.col+1] TO validMoves
11.         ENDIF
12.         IF (pos.col > 0)
13.                 APPEND [pos.row,pos.col-1] TO validMoves
14.         ENDIF
15.         RETURN validMoves
16. END CalcValidMoves

1. MODULE MoveCloserTo(pos1,pos2)
2.         // Works out entity1's final positions after moving towards entity2
3.         possibleFinalPos <- [] // Will contain possible movements for the Troll King
4.         IF pos1.row < pos2.row
5.                 finalPos <- [pos1.row+1,pos1.col]
6.                 APPEND finalPos TO possibleFinalPos
7.         ELSE IF pos1.row > pos2.row
8.                 finalPos <- [pos1.row-1,pos1.col]
9.                 APPEND finalPos TO possibleFinalPos
10.         ENDIF
11.         IF pos1.col < pos2.col
12.                 finalPos <- [pos1.row,pos1.col+1]
13.                 APPEND finalPos TO possibleFinalPos
14.         ELSE IF pos1.col > pos2.col
15.                 finalPos <- [pos1.row,pos1.col-1]
16.                 APPEND finalPos TO possibleFinalPos
17.         ENDIF
18.         RETURN possibleFinalPos
19. END MoveCloserTo
```

CalcTurnEvents

world is the value shown.

line	somethingHappened	world.aurora.pos=world.trollKing.pos	world.aurora.pos IN world.villagePositions	world
2->12	FALSE	FALSE	FALSE	world
	somethingHappened	**world.aurora.pos=world.wolves.pos**	**---**	**world**
14->17	FALSE	FALSE	---	world
	somethingHappened	**shrine.pos**	**world.aurora.pos=shrine.pos**	**world**
19->24	FALSE	4 8	FALSE	world
19->24	FALSE	5 7	FALSE	world
19->24	FALSE	7 5	FALSE	world
19->25	FALSE	8 4	FALSE	world
	somethingHappened	**RANDOM(0.2)**	**fightCreature**	**world**
27->35	FALSE	TRUE	1	world
	NOT somethingHappened and NOT fightCreature	**fightCreature=1 fightCreature=2**	**creature**	**world**
37->63	FALSE	TRUE FALSE	smallCreature	DoFight(world, smallCreature)
	world.aurora.health>0 ...<world.aurora.maxHealth	**world.aurora.health**	**world.aurora.maxHealth**	**world**
65->70	TRUE TRUE	50	100	world
66/68	...	60	...	...
	world.trollKing.health>0 ...<world.trollKing,maxHealth	**world.trollKing.health**	**world.trollKing.maxHealth**	**world**
72->77	TRUE FALSE	140	70	...
73/75	...	140	70	...
`	**world.wolves.health>0 ...<world.wolves.maxHealth**	**world.wolves.health**	**world.wolves.maxHealth**	**world**
79->89	TRUE FALSE	50	50	...
80/82	...	...	...	...
	...aurora.pos=world.questPos	**world.questPos**	**world.gold**	**output**
91->96	FALSE	---	---	---
	---	**---**	**---**	**return**
97				world

world.questPos = world.aurora.pos = world.trollKing.pos = world.wolves.pos = 9 9

line	somethingHappened	world.aurora.pos=world.trollKing.pos	world.aurora.pos IN world.villagePositions	world
2->12	TRUE	TRUE	FALSE	DoFight(world,world.trollKing.attack)
	somethingHappened	**world.aurora.pos=world.wolves.pos**	**---**	**world**
14->17	TRUE	TRUE	---	DoFight(world,world.wolves.attack)
	somethingHappened	**shrine.pos**	**world.aurora.pos=shrine.pos**	**world**
19->24	TRUE	4 8	FALSE	world
19->24	TRUE	5 7	FALSE	world
19->24	TRUE	7 5	FALSE	world
19->25	TRUE	8 4	FALSE	world
	somethingHappened	**RANDOM(0.2)**	**fightCreature**	**world**
27->35	TRUE	---	0	world
	NOT somethingHappened AND NOT fightCreature	**fightCreature=1 fightCreature=2**	**creature**	**world**
37->63	FALSE	FALSE FALSE	---	world
	world.aurora.health>0 ...<world.aurora.maxHealth	**world.aurora.health**	**world.aurora.maxHealth**	**world**
65->70	TRUE TRUE	30	100	world
66/68	...	60	...	...
	world.trollKing.health>0 ...<world.trollKing,maxHealth	**world.trollKing.health**	**world.trollKing.maxHealth**	**world**
72->77	TRUE FALSE	90	70	...
73/75	...	...	70	...
`	**world.wolves.health>0 ...<world.wolves.maxHealth**	**world.wolves.health**	**world.wolves.maxHealth**	**world**
79->89	TRUE TRUE	40	50	...
80/82	...	42.5	50	...
	...aurora.pos=world.questPos	**world.questPos**	**world.gold**	**output**
91->96	TRUE	9 9	0+10=10	The friend walked up to you... Then they gave you a small...
	---	**---**	**---**	**return**
97	---	---	---	world

CalcValidMoves

world is the value shown.

Values	
Pos pos.row	9, 0, 12
Pos pos.col	9, 0, 12

line	pos	pos.row<world.rows-1	pos.row>0	pos.col<world.cols-1	pos.col>0	validMoves
2->15	9 9	TRUE	TRUE	TRUE	TRUE	10 9 8 9 9 10 9 8
2->15	0 0	TRUE	FALSE	TRUE	FALSE	1 0 0 1
2->15	12 12	FALSE	TRUE	FALSE	TRUE	11 12 12 11

MoveCloserTo

world is the value shown.

Values	
Pos pos1.row	0,1,2
Pos pos1.col	0,1,2
Pos pos2.row	1,0,2
Pos po2.col	1,0,2

line	pos1 pos2	pos1.row<pos2.row	pos1.row>pos2.row	pos1.col<pos2.col	pos1.col>pos2.col	possibleFinalPos
3->18	0 0 1 1	TRUE	FALSE	TRUE	FALSE	1 0 0 1
3->18	1 1 0 0	FALSE	TRUE	FALSE	TRUE	0 1 1 0
3->18	2 2 2 2	FALSE	FALSE	FALSE	FALSE	