A Tactical RPG Engine

Bilal Syed Hussain

Supervisor: *Second Marker:*
Dr. Ian Miguel   Dr. Alex Voss

March 28, 2012

Contents

Abstract

In odio velit, semper quis mattis eu, varius et felis. Donec vulputate aliquam purus id feugiat. Fusce vel ante neque, vitae placerat sem. Nam a tortor purus. Aenean laoreet volutpat consectetur. Proin sit amet lorem orci. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Morbi quis tempus lacus.

Donec feugiat ultrices porta. Vivamus laoreet odio sed augue ultrices vitae consequat nibh pharetra. Nam et fringilla est. Sed dolor lorem, luctus aliquet lacinia vitae, mollis vel tortor. Vestibulum aliquam mi eget neque semper aliquam. Duis accumsan sapien tristique tellus fringilla convallis. Nulla odio augue, eleifend sit.

Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is NN,NNN* long, including project specification and plan. words

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the Declaration University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

1 Introduction

An RPG (Role Playing Game) is a game where a player assumes the role of a character. An RPG is usually story driven and the character usually has a quest to complete. In the course of the game the player will go to different environments such as town and dungeons. In these environments the player will have to fight opponents in battles. Combat in RPGs is normally a simple turn based system where players and their opponents take turns to attack each other using various skills.

A Tactical RPG is a sub-genre of an RPG that focuses on the combat side of the genre. A Tactical RPG is series of `battles`, which take place in various environments intertwined with an over-arching story.

Each `battle` is grid based (like chess) where each player has a number of units(pieces). The players take turns to move their units. Each unit has attributes associated with it such as

Figure 1: **Tactics Ogre**[1] a classic Tactical RPG

strength, and hit points that affect all the actions in the game. Like chess there are different kinds of units which affects how the unit moves and what action they can perform. A unit can attack other player's units, the goal of the battle is usually to defeat all the opponents units.

The aim of this project is to create an engine which will take resources such as graphics, sounds and rules of the game to create a runnable Tactical RPG.

1.1 Project Baseline

No previous work was used for this project. All of the project was created during the course of the academic year.

1.2 Project Success

- Finished all the all primary objectives, and nearly all secondary and Tertiary objectives.

- Very useable

- Cross platform

2 Context Survey

2.1 Evolution of Tactical RPGs

Notable TRPGs

- Bokosuka Wars - probably the first TRPG

- Fire Emblem: Ankoku Ryu to Hikari no Tsurugi - First popular TRPG. Characters are unique

- Tactical Ogre:

  - First TRPG with isometric graphics.
  - Character battle order is determined by the character's 'speed' rather, each player moves all their units when its their turn.
  - First to have a branching plot and the player's choice effecting the game.
  - Associated the genre with the word 'Tactics', used by many later games

- Final Fantasy Tactics, widely popular, based on Tactics Ogre.

- Disgaea: Hour of Darkness: Allows the player to play random generated maps. The latest in the series is one of the few TRPGs that contain a map editor.

- Recent game, have mostly mix aspects from other genres, for example Valkyria Chronicles features FPS like shoting when attacking.

2.2 Overview of Game Engines

- `Sim RPG Maker 95`, one of the few tactical RPG's engines

- `RPG Maker` which it is based off.

- Mention engines such `unity` which used to make TRPGs?

3   Requirements Specification

3.1   Project Scope

The aim of the project is to allow the user a highly customisable Tactical RPG. There are three main parts to the project the *engine*, the *GUI* and the *editor*.

The engine will contains all the logic of the game including the progression as well as the battle system. The GUI, will be an isometric view of the game (see Section 6.5.1).

The editor will allows the user to customise the input to engine. This includes visual map making as well as specify all the attributes of the units and weapons. The editor also allows the user to export the game as a standalone application.

3.2   Requirements Overview

A complete listing of requirements is in section 3.6. The main requirements are to create a engine allows a high degree of customisability, a isometric view and exporting the game as a standalone application.

3.3   Overview Description

3.3.1   Product Perspective

3.3.2   Product functions

3.3.3   User characteristics

The system is for anyone would like to create a TRPG. They need not have any experience in creating TRPG before, or each played one before since the a game can be created without `any` programming.

For more advance users, they of course further customise the created game using their own code. This allows the user to completely change most aspects of the game. This could be used for example to make unique abilities or battle system.

3.3.4   Constraints, assumptions and dependencies

The system should be portable i.e it should works on most operating systems. To achieve this I used Java achieve since it would work on any system that has the java virtual machine installed on it.

3.4   Objectives

In the following subsubsections ✓ means that the objective is fully completed, ✗ means incompletion and - signifies partial completion

3.4.1   Primary

The main goal of the primary objectives is to allow the user to create a complex Tactical RPG, with limited customisability.

✓ To develop an engine that takes:

  ✓ The definition of character attributes and a combat system.

  ✓ The definition of a world broken up into the smaller environments.

    ✓ The rules of the game.

    ✓ The kinds of enemies.

  ✓ The definition of a simple story as a wrapper for the whole game, from the start to the conclusion of the game

    ✓ Which is told between the movement between different environments.

and create a playable tactical RPG.

✓ To include in the engine support for the following:

  ✓ `units` with a fixed set of associated attributes such as:

    ✓ Hit-points (which represent the health of the unit).

    ✓ Strength.

    ✓ Defence.

    ✓ Move (The number of tiles the unit can move each turn).

  ✓ `battles` which take place on grid and include:

    ✓ A set number of `units` for each player.

    ✓ A Winning `condition`, which is defeat all of the other player's units.

    ✓ Battles are `turn based` meaning only one unit performs at one time.

    ✓ A combat system.

  ✓ A combat system that includes

    ✓ `combat` between adjacent units.

      ✓ When the unit hit-points are reduced to zero they are `defeated` and are removed from the map

      ✓ A set of rules that govern the combat.

  ✓ A predefined set of behaviours for how the non-player characters should behave.

    ✓ Including pathfinding.

  ✓ An isometric graphical representation of the game.

    ✓ Which shows the grid with all the units.

    ✓ Allows the user to move their units and see the opponents moves.

    ✓ Allows the user to attack the opponent's units.

    ✓ Which allows the user to see a unit status (e.g current hit points).

    ✓ Text will be used to describe the more complex actions such magic.

3.4.2   Secondary

The main goal of the secondary objectives is to allow the user more customisability.

- ✓ Tiles have `height`, where units can only move to tiles of a smilier height.

- • Tiles that are not passable such as sea, lava, etc.

- ✓ Tiles have different movement costs associated with them.

- ✓ A combat system that includes

    - ✓ `combat` between non-adjacent units,

- ✓ Players have items such as weapons that affect the result of combat between units.

    - ✓ Including long distance weapons for the player and AI.

  – Direction and height of the character's tile affects attack. [1]

- ✓ Sound effects.

- ✓ Music.

- ✗ Saving and loading games.

  – Allow the user to specify some of behaviour of non-player characters

    - ✓ An example: always attack a certain kind of unit or always attack the unit with the least Hit Points.

- ✓ A graphical view to allow the user to specify input to the engine.

3.4.3   Tertiary

The goal of the Tertiary objectives are to provide the user with more customisability and to provide a GUI for customising aspects of the engine.

- ✓ A combat system that includes

    - ✓ Support for `skills` which can effect multiple units.
    - ✓ Including weapons that can attack multiple units at the same time.

- ✓ Animations for units and movement.

- ✓ A graphical editor for creating and specifying the input to the engine which allows:

    - ✓ Creating and editing maps.
        - ✓ which also allows placement of enemy units.
    - ✓ specifying the order of the maps.

---

[1]At the moment only the height affects the attack, while the direction is displayed in the GUI and changed based on the unit's movement, it not used in the model.

✓ making animations.

✓ making items such as weapons.

✓ making skills.

✓ making units.

✓ specifying the story, at the start and end of a battle.

✓ specifying the music and sound effects played on each map.

✓ specifying the condition to win a map such as:

  ✓ Defeating all the opponent's units.

  ✓ Defeating a specific unit.

✓ specifying some of the behaviour of the enemy units.

✓ Allows exporting the game as a self contained application.

✗ Custom events

  – Attached to units or titles, could be used for:

    ✗ Making the player win if some enemies unit has less then 50% Hit Points.

    ✗ Damaging a character if step on a specified tile.

    ✗ Showing some part of the story when a player's character reaches a specified tile.

3.5 Specific Requirements

The system should allow the user to export the created game with no addicanl dependencies apart from the java runtime evnvment.

3.5.1 Security Requirements

Although their are no security requirements in the objective, they should nevertheless be considered in the future. Xml is used as the data format for the created games. This aids maintainability since xml is human readable. The disadvantage of this is that user of the created game can also access the data, hence could edit it or steal the resources of the game. A solution to this problem would be to encrypt the data files so they are not editable by the end users.

3.5.2 User Interface Requirements

The GUI should provide a isometric view of the game as shown in 1. The GUI should the user to visually see which action the opperent performs. The GUI should give visual feedback for any actions the user makes

3.6   Objectives

In the following subsubsections ✓means that the objective is fully completed, ✗means incompletion and- signifies partial completion

3.6.1   Primary

The main goal of the primary objectives is to allow the user to create a complex Tactical RPG, with limited customisability.

✓ To develop an engine that takes:

   ✓ The definition of character attributes and a combat system.

   ✓ The definition of a world broken up into the smaller environments.

       ✓ The rules of the game.

       ✓ The kinds of enemies.

   ✓ The definition of a simple story as a wrapper for the whole game, from the start to the conclusion of the game

       ✓ Which is told between the movement between different environments.

   and create a playable tactical RPG.

✓ To include in the engine support for the following:

   ✓ `units` with a fixed set of associated attributes such as:

       ✓ Hit-points (which represent the health of the unit).

       ✓ Strength.

       ✓ Defence.

       ✓ Move (The number of tiles the unit can move each turn).

   ✓ `battles` which take place on grid and include:

       ✓ A set number of `units` for each player.

       ✓ A Winning `condition`, which is defeat all of the other player's units.

       ✓ Battles are `turn based` meaning only one unit performs at one time.

       ✓ A combat system.

   ✓ A combat system that includes

       ✓ `combat` between adjacent units.

           ✓ When the unit hit-points are reduced to zero they are `defeated` and are removed from the map

           ✓ A set of rules that govern the combat.

   ✓ A predefined set of behaviours for how the non-player characters should behave.

       ✓ Including pathfinding.

   ✓ An isometric graphical representation of the game.

✓ Which shows the grid with all the units.

✓ Allows the user to move their units and see the opponents moves.

✓ Allows the user to attack the opponent's units.

✓ Which allows the user to see a unit status (e.g current hit points).

✓ Text will be used to describe the more complex actions such magic.

3.6.2   Secondary

The main goal of the secondary objectives is to allow the user more customisability.

✓ Tiles have `height`, where units can only move to tiles of a smilier height.

● Tiles that are not passable such as sea, lava, etc.

✓ Tiles have different movement costs associated with them.

✓ A combat system that includes

　✓ `combat` between non-adjacent units,

✓ Players have items such as weapons that affect the result of combat between units.

　✓ Including long distance weapons for the player and AI.

　– Direction and height of the character's tile affects attack. [2]

✓ Sound effects.

✓ Music.

✗ Saving and loading games.

　– Allow the user to specify some of behaviour of non-player characters

　　✓ An example: always attack a certain kind of unit or always attack the unit with the least Hit Points.

✓ A graphical view to allow the user to specify input to the engine.

3.6.3   Tertiary

The goal of the Tertiary objectives are to provide the user with more customisability and to provide a GUI for customising aspects of the engine.

✓ A combat system that includes

　✓ Support for `skills` which can effect multiple units.

　✓ Including weapons that can attack multiple units at the same time.

---

[2]At the moment only the height affects the attack, while the direction is displayed in the GUI and changed based on the unit's movement, it not used in the model.

✓ Animations for units and movement.

✓ A graphical editor for creating and specifying the input to the engine which allows:

   ✓ Creating and editing maps.

      ✓ which also allows placement of enemy units.

   ✓ specifying the order of the maps.

   ✓ making animations.

   ✓ making items such as weapons.

   ✓ making skills.

   ✓ making units.

   ✓ specifying the story, at the start and end of a battle.

   ✓ specifying the music and sound effects played on each map.

   ✓ specifying the condition to win a map such as:

      ✓ Defeating all the opponent's units.

      ✓ Defeating a specific unit.

   ✓ specifying some of the behaviour of the enemy units.

   ✓ Allows exporting the game as a self contained application.

✗ Custom events

   – Attached to units or titles, could be used for:

      ✗ Making the player win if some enemies unit has less then 50% Hit Points.

      ✗ Damaging a character if step on a specified tile.

      ✗ Showing some part of the story when a player's character reaches a specified tile.

4   Software Engineering Process

4.1   Methodologies Used

I choose to use a iterative spiral development model for the project. This allowed me to focus on specify parts of the projects of the system before moving onto the next component.

Prototypes were extsaiely esscialy in the GUI when choosing how to render the map.

4.1.1   Test Driven Development

Test-driven development (TDD) was utilised in the project. This also helped with verification of requirements as tests assert whether the code matched the minimum requirements. The `JUnit` library[3] was used to write the unit tests.

The main stages in TDD are[2],[3]:

1. Before any code was written, unit tests to test the functionality. These will initially fail.
2. Code is written to pass the test and no more.
3. If more functionality is required, first the test is written and then the code to pass it.
4. Changes to the previously written code must pass all previous created tests.
5. The code is Refracted

The major benefits of TDD are that system will be well tested as the added benefit of prevents new features from introducing bugs. Combined with version control as discussed in the next section, it make it very easy to find bug since the unit tests can be used to find out *when* the code broke as well as *which* piece of code was the root cause.

This method of development was perfectly suited to implementing the algorithms in the engine (such as unit movement) because the expected output was know beforehand. Since all components of the model were programmed to an interface, it allows the use of mock objects4.2.

However TDD has few drawbacks such as the difficulty of realising all possible test scenario, which because apparent when testing the GUI and the editor. To this these aspects of the system I played multiple created game from start to finish with the goal of finding any lingering bugs. In addiction I did user surveys as well as usability studies to find any unexpected defects in the user interface. (results in)

4.2   Mock Objects

Mock objects are used predominantly in very large software development projects to aid in testing. The objective is to create object which simulate only the essential behaviour of the object required. Mock objects abstracts the detailed functionality of the implementation away and focuses only on what is required for the test

4.3   Version Control

Version control keeps track of all changes to a project. Keeps the history of changes helps finding when a bug was first intodusced and hence what caused the bug. In preticual I choose to used

---

[3]JUnit 4.8.2, see www.junit.org/ for details

`git`, which is distributed version control system. In differs from trainally client server system such as Subversion in that each user has a complete copy of the repotidry.

Distributed version control system have the advantages of allowing change to committed locally, even without an internet connection. This particularly useful for this project since it allows experimenting with various features before choosing the features to integrate into the system.

5   Ethical Considerations

Algnough human subjects were asked to complete a suvery to gain feedback on usability as testing, no personal information was stored. Following the ethics requirements of the School of Computer Science I submitted a 'Preliminary Ethics Self Assessment Form'. The result of the assessment was that were no ethical issues raised by this project.

6 Design

6.1 Methodologies

The Model, View, Controller (MVC) design paradigm was chosen to be used to structure the project. The MVC design patten has many benefits as discussed below.

6.2 Engine

The main considerisons for the engine was to it made it configureable as possible. To achieve this everything was designed in term of interfaces, which allowed the particle implementation to be changed. The objects themselves were loaded from serizeled rep of the objects.

6.2.1 Assets

All the assets used in the model including weapons, skills and the units themself are configured via the xml data format.

6.3 Game Progression
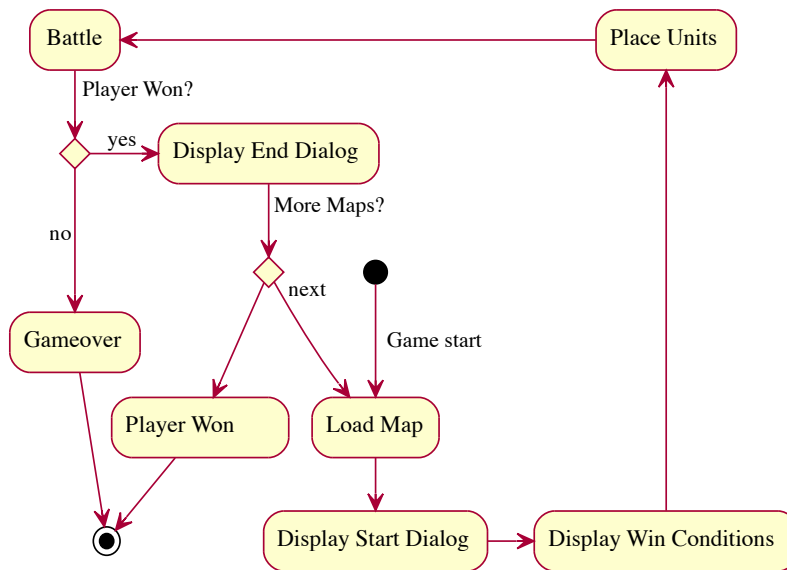
Figure 2: Activity diagram which a high level of overview of the engine

Figure 2 shows the overview of how the created game progress. Each game has a number of maps where a battle take place. After the map is loaded any relanent dialog is display, along with winning conditions. The player's units are then placed are the map[4]. And the battle (which will be discussed below takes). If the player's loses the battle, a gameover screen is shown and the game end. In contrast if the player wins he/she advances to the next map, if there one.

---

[4]While the engine support allowing the user's to choose where the their units are placed, the GUI does not due to time constraints. The editor does support specify the starting location for the player's units

6.4   Gui

The
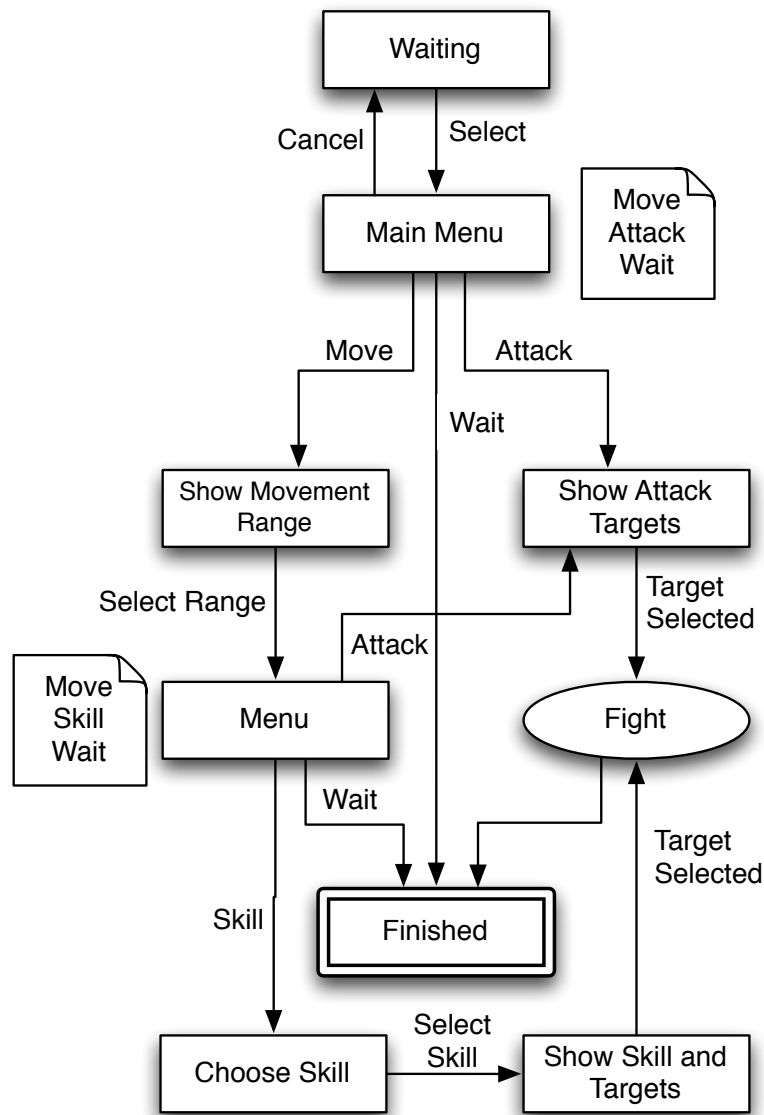
Figure 3: The State diagram of a single turn of a player's unit

6.5   View

6.5.1   Tilemap

There were two main choices for the isometric tilemap, a 'Diamond' map or a 'Staggered' map [?], examples of both are shown below.
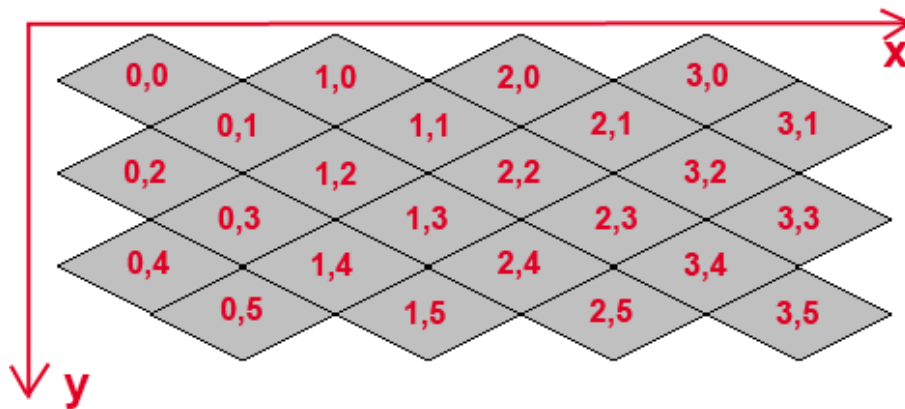
The 'Staggered' Map has following advantages:

- The map fill up the screen with very little wasted space, so the user can more of what happing on the map.

The 'Diamond' map was chosen for the following reasons:
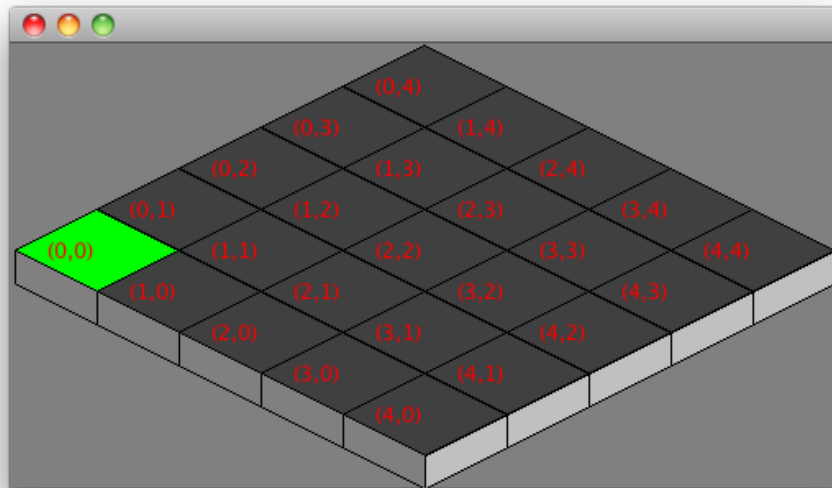
(a) Diamond Map

(b) Staggered Map

Figure 4:   The two main types of isometric tilemaps

- 'Diamond' map look nicer then 'Staggered' maps because it has no ragged edges.

- Since that maps are large (at lest 15×15) the space wasted at the edges of the map does not matter as much.

- Simpler to think about, since a 'Diamond' map is just a rectangular map rotated.

<u>Maths about isometric tilemaps?</u>

6.6   Editor

6.6.1   Exporting

The editor can export a project as a complete package, either as a Mac OS X application or as jar. These application don't requires any external resources, apart from a recent version of java [5].

A notable feature of the editor is that jar will work on any java enabled platform, since the jar contains all required libraries for each platform. The OS X application can even be export on other platforms.

While most of the testing was done on OS X [6], it also works well on Linux [7]. It even has limited compatibly with Windows[8] (apart from some minor graphics issues).

---

[5]specifically Java 1.6+
[6]Mac OS X 10.6 Snow leopard
[7]Science Linux x.y
[8]Tested on Windows 7 32 bit

7   Implementation

7.1   Engine Development and Testing

7.1.1   Maps

7.1.2   Units

7.1.3   Events

7.1.4   Algorithms

7.2   View Development and Testing

7.2.1   Map Rendering

7.3   User Interface

7.4   Data Formats

7.4.1   XML

7.4.2   Custom Classes

7.5   Editor Development and Testing

7.5.1   Overview

7.5.2   Map Editor

7.5.3   Unit Editor

7.5.4   Event Editing

7.5.5   Exporting

8 Scripting

Scripting allows the user to customise aspects of the game. This includes customising the opponent's AI, custom winning conditions and user defined events.

8.1 Language Choice

There were three main choices using Javascript, using JRuby[9], or building a 'domain specific language'.

Creating a 'domain specific language' was considered initially, this would have the following advantages:

- Provides more abstraction, and allow the complex details to be hidden.

- Easier to validate since the languages contains a very few constructs.

but was rejected because:

- of the time to create and test the new language.

- of the cost of creating tools for the new language, there are already source code highlighters and debuggers for Javascript and Ruby.

- of the loss of efficiency, the Javascript parser in the JDK as well as JRuby is very efficient and provides advance features such as 'just in time compilation' [10] which would not be possible to implement for the new language within the time constraint of the project.

JRuby has the following advantage:

- Easier syntax for interacting with Java then javascript.

- Easy to use with the embedding API in the JDK.

Javascript was chosen over Ruby as a scripting language for the following reasons:

- Javascript embedding is build into the JDK, so the user does not have install anything extra. It also has the advantage of being cross platform.

- Javascript is easy to learn, and average user is more likely to have used it before as compared to Ruby.

8.2 Data Exposed

Events can be attached to `units`, `tiles` in a battle, globally in a battle and to the AI. All events are passed a `mapinfo` object which contains the following as read only data:

- A hashtable of the players unit and a hashtable of the enemies units. For each unit this includes

---

[9]A Ruby implementation written in java

[10]A method to improve the runtime performance, by translating the interpreted code into lower level form, while the code is be run

- all the unit's attributes such as the location, and `hit points`.
- if the unit has been defeated.

- The `leader` unit of each side if there is one.

- The number of turns taken.

The `mapinfo` object contains the following methods:

`win` The player wins the battle.

`lose` The player loses the battle.

`dialog` The player is shown the specified dialog (to show the user some the plot). Can be directed from a specify unit, or a global message.

`action` Executes the specified action.

This allows the user to make complex events without them changing the model to much.

8.3 Action

A `action` is a set of unit defined actions. For example a poison action could reduces the a units 'hit points' by 10%

8.4 Winning Conditions

The user can specify the winning conditions based on what occurring in the battle, examples include

- If opponent's leader's hp $< 50\%$ then `win()`.

- If <character> dies then lose().

- If number of turns > 20 then lose()

8.5 Unit Events

Unit events get passed the specified unit as well as the `mapinfo`. the event can be specified to execute when:

1. The unit finishes its turn.

2. The unit is affected by magic.

3. The unit is attacked.

4. The unit attacks.

Example: When <unit> attacked counter attack.

8.6   Tiles Events

Tiles get passes the specified tile as well as the Unit. The event can be specified to execute when

- A unit moves to a tile.

- A unit moves though a tile.

Example: On unit moving though `action(posion)`

8.7   AI Events

The behaviour of AI can be customised, with `commands` such as:

- Attack the player's unit with highest/lowest hp.

- Attack the player's leader unit.

- If player's leader's hp $< 20\%$ `heal(leader)`.

- Attack player's characters of class $<$class$>$.

The AI events `mapinfo` has addition methods including:

attack Attack the specified unit.

follow Move as close as possible to the specified unit.

heal Heal the specified unit.

move Move to the specified location.

wait Do nothing this turn

The `commands` themselves can be conditional, as example

```
Listing 1: Conditional AI Event

If opponent's leader's hp < 20% then
    heal(leader).
else If player has a leader unit then
    If player's leader's hp < 20% then
        Attack the player's leader unit
    else
        Attack the player's closet unit with the lowest hp.
    end
else
    wait
end
```

9   Evaluation and Critical Appraisal

9.1   Results of User Testing

System usability scale (SUS) was used [4]. This works by giving even numbered questions a score of (5 - *value*) and odd numbered questions a score of of (*value*-1). Questions that contributed a high score show that the system is usable.

10 Conclusions

11   Testing

11.1   Feedback

- It was hard to see which unit was selected. This was fixed by displaying 'Current' in selected unit's info. The info window of the selected unit was also lightened to to make it more obvious.

- It was hard to see which are my units. This was fixed by displaying the player's unit's info in green and the enemy's unit's info in red.

- Some users could not figure out the key bindings of the game. This was fixed by displaying a list of all key binding at the start of the game.

A   User Manual

B   Questionnaire

Task

The task involves creating a single level of a Tactical RPG (Each level is grid based (like chess) where each player takes turns to move and/or attack the opposing player).

Weapons

Name	Weapon Type	Strength	Icon
Long Bow	Ranged	30	
Black Spear	Spear	20	
Ice Sword	Melee	10	

Skills

Name	Type	Range	Area	Strength
Air Blade	Ranged	2	0	25
Thunder Flare	Ranged	4	1	15

Units

Agrias	
Weapon	Long Bow
Strength	20
Move	3
Skills	
Air Blade	

Elena	
Weapon	Black Spear
Strength	30
Move	5
Skills	
Thunder Flare	

Map Enemies

Mustadio	
Weapon	Long Bow
Strength	20
Move	3
Skills	

Druksmald	
Weapon	Ice Sword
Strength	30
Move	5
Skills	

Zalbaag	
Weapon	Ice Sword
Strength	25
Move	5
Skills	

Ajora	
Weapon	Ice Sword
Strength	20
Move	5
Skills	

Map

Figure 5: The map to create

Win Condition

Defeat Specific Unit – Elena.

Start Dialog:

Text You can not Win!

Speaker Kyou

End Dialog:

Text How did I lose?

Speaker Elena

Music:

Background Music 3-15 Faraway Heights

B.1   Editor Usability Scale

1. I think that I would like to use this system frequently.

← strongly disagree   agree completely →

☐ ☐ ☐ ☐ ☐

2. I found the system unnecessarily complex.

☐ ☐ ☐ ☐ ☐

3. I thought the system was easy to use.

☐ ☐ ☐ ☐ ☐

4. I think that I would need the support of a technical person to be able to use this system.

☐ ☐ ☐ ☐ ☐

5. I found the various functions in this system were well integrated.

☐ ☐ ☐ ☐ ☐

6. I thought there was too much inconsistency in this system

☐ ☐ ☐ ☐ ☐

7. I would imagine that most people would learn to use this system very quickly

☐ ☐ ☐ ☐ ☐

8. I found the system very cumbersome to use

☐ ☐ ☐ ☐ ☐

9. I felt very confident using the system

☐ ☐ ☐ ☐ ☐

10. I needed to learn a lot of things before I could get going with this system

☐ ☐ ☐ ☐ ☐

B.2   Playing a pre-created game

← strongly disagree   agree completely →

1. I found the game intuitive

☐ ☐ ☐ ☐ ☐

2. The game had a appropriate level of difficulty.

☐ ☐ ☐ ☐ ☐

3. I enjoyed playing the game.

☐ ☐ ☐ ☐ ☐

4. Please share any comments about the game :

B.3   Questions

5. Have you played a Tactical RPG before?

_____

6. Did Engine have features you like to create in a game?

_____

7. How easy to use was the Engine?

_____

8. What particular aspects of the Engine did you like?

_____

9. What particular aspect of the Engine did you dislike?

_____

10. What features would you like to see added to the Engine in the future?

_____

11. Please share any other comments:

_____

C   Future Work

- Improvement to levelling up. Usually a unit does not have access to all of its skill at begining, but gains access to them when levelling up. This would make the produced game more balanced, since only skill appropriate to the unit stats could be used.

- Implementation of an overworld map with a battle happening at each location. This would allow the user to choose which map to play. A good use of this would be a branching storyline where the plot is changed depending on which maps the player plays.

- Better Ai

- Scripted Events

References

[1] Quest, "Tactics Ogre: Let Us Cling Together," 1995. 5

[2] E. Murphy-Hill, "Test-Driven Development." 15

[3] C. Desai, D. Janzen, and K. Savage, "A survey of evidence for test-driven development in academia," *ACM SIGCSE Bulletin*, vol. 40, no. 2, pp. 97–101, 2008. 15

[4] J. Brooke, "SUS: A quick and dirty usability scale," in *Usability evaluation in industry*, P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, Eds. London: Taylor and Francis, 1996. 26