

UNIVERSITY OF ST ANDREWS

CS4099: MAJOR SOFTWARE PROJECT



A Tactical RPG Engine

Bilal Syed HUSSAIN

Supervisor:

Dr. Ian MIGUEL

Second Marker:

Dr. Alex VOSS

April 10, 2012

Contents

Abstract	6
Declaration	6
1 Introduction	7
1.1 Project Baseline	7
1.2 Project Success	8
2 Context Survey	9
2.1 Evolution of Tactical RPGs	9
2.2 Overview of Game Engines	9
3 Requirements Specification	10
3.1 Project Scope	10
3.2 Requirements Overview	10
3.3 Overview Description	10
3.3.1 Product Perspective	10
3.3.2 Product functions	10
3.3.3 User characteristics	10
3.3.4 Constraints, assumptions and dependencies	10
3.4 Objectives	10
3.4.1 Primary	10
3.4.2 Secondary	12
3.4.3 Tertiary	12
3.5 Specific Requirements	13
3.5.1 Security Requirements	13
3.5.2 User Interface Requirements	13
4 Software Engineering Process	14
4.1 Methodologies Used	14
4.1.1 Test Driven Development	14
4.2 Mock Objects	14
4.3 Version Control	15
5 Ethical Considerations	16

6	Design	17
6.1	Methodologies	17
6.2	Engine	17
6.2.1	Data Format	17
6.2.2	XML schema	17
6.3	Game Progression	17
6.4	Game Mechanics	18
6.5	Map Rendering	18
6.5.1	Introduction	18
6.5.2	Tilemap	18
6.6	Units	20
6.7	Weapons & Skills	21
6.8	Editor	22
6.8.1	Visual Map Editing	23
7	Implementation	24
7.1	Overview	24
7.2	Data Format	25
7.2.1	Resources	26
7.2.2	Sprite Sheets	26
7.3	Engine Development	27
7.3.1	Map	27
7.3.2	Units	27
7.3.3	Movement and Path Finding	27
7.3.4	AI Behaviour	28
7.3.5	Turn Ordering	28
7.3.6	Battle System	28
7.3.7	Conditions	29
7.3.8	Dialog	29
7.4	Saving and Loading	29
7.5	Inter-compatibility	30
7.6	Map Rendering	30
7.6.1	Drawing Algorithm	31
7.6.2	Images and Texture Mapping	32
7.6.3	Units	32
7.6.4	Reusability	32

7.7	Map Rotation	32
7.8	User Interface	33
7.8.1	Controls	33
7.8.2	Menus	33
7.8.3	Other Features	33
7.9	Editor Development	33
7.9.1	Overview	33
7.9.2	Weapons & Skills	35
7.9.3	Unit Editor	35
7.9.4	Sprite Sheet Editor	35
7.9.5	Music Editor	35
7.9.6	Map Editor	37
7.9.7	Visual Map Creation	38
7.9.8	Dialog Editing	40
7.9.9	Project Selection	41
7.9.10	Exporting	42
7.10	Ant Build File	42
8	Evaluation and Critical Appraisal	43
8.1	Objectives	43
8.1.1	Primary Objectives	43
8.1.2	Secondary Objectives	43
8.1.3	Tertiary Objectives	44
8.2	Results of User Testing	45
8.3	Analysis of Responses	45
8.3.1	System usability scale	46
8.3.2	Results	46
8.3.3	Analysis	46
9	Conclusions	47
9.1	Future Work	47
10	Testing	48
A	User Manual	49

B	Project Structure & Specification	50
B.1	Assets	50
B.2	images	51
B.3	Maps	51
B.4	Music	51
B.5	Sound Effects	51
B.6	Editor	52
B.6.1	Dialog Data Format	52
B.7	Data Format Extension Mechanism	53
C	Questionnaire	54
C.1	Editor Usability Scale	56
C.2	Playing a pre-created game	56
C.3	Questions	57
D	Scripting	58
D.1	Language Choice	58
D.2	Data Exposed	58
D.3	Action	59
D.4	Winning Conditions	59
D.5	Unit Events	59
D.6	Tiles Events	60
D.7	AI Events	60
E	BugFixes notes	61

List of Figures

1	Tactics Ogre[?] a classic Tactical RPG	7
2	Activity diagram which a high level of overview of the engine	18
3	The two main types of isometric tilemaps	19
4	The State diagram of a single turn of a player's unit	20
5	The above figure shows the attack range of various weapons and skill	21
6	Unit panel	22
7	Visual Map Editor	23
8	Overview of the implementation.	24
9	A 128×128 sprite sheet containing a tileset for a map	26

10	A example of a Skill. The red squares are the <i>attack range</i> , the green the <i>attack area</i>	28
11	A example of a dialog	29
12	A isometric map which shows the features of the renderers including height, empty tiles, wall textures and unit placement	30
13	The above figure shows normal and slanted tiles, The red X is the start point for drawing	31
14	Unit Editor	34
15	A sprite sheet editor for the tiles	36
16	The Music Editor	36
17	The Map Editor	37
18	The map editor show a map created by Stasyk's terrain generator.	39
19	The dialog editing panel	40
20	The Project Selection Window which is shown at startup	41
21	Project Structure	50
22	Editor Project Structure	52
23	The map to create	55

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 words NN,NNN* long, including project specification and plan.

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 skills¹.

A Tactical RPG is a sub-genre of an RPG that focuses on the combat side of the genre. A Tactical RPG is a 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**[?] 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 have different abilities which affects how the unit moves and what actions they can perform. A unit can attack another player's units. The goal of the battle is usually to defeat all the opponent's units although other goals are possible.

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.

¹Although there are some common variants that will be discussed later on

1.2 Project Success

Having completed all the primary and secondary objectives as well as nearly all the tertiary objectives, I would say that the project was successful. Notable features that were accomplished include a very customisable engine which allows the user to specify most aspects of their created game and an editor which allows the user to make a complex game without any programming(although programming can be used to further enhance their game).

The ability to export the user's created game as a standalone application without any external dependencies is a significant feature of the project. Since Java was used, the created game is cross-platform and can run on any system with a complete Java runtime environment².

Results from the usability studies were very positive, with most users saying they enjoyed using the system and found it usable.

²i.e The GUI and Editor won't run on Android since it does not have the Swing toolkit.

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 enable the user to create a highly customisable Tactical RPG. There are three main parts to the project: the *engine*, the *GUI* and the *editor*.

The engine will contain all the logic of the game including handling the game progression as well as the battle system. The GUI will be an isometric view of the game (see Section 6.5.2).

The editor will allow the user to specify the input to the engine. This includes visual map making as well as specifying all the attributes of the units and weapons in addition to the winning conditions. The editor will also allow the user to export the game as a standalone application.

3.2 Requirements Overview

A complete listing of requirements is in section 3.4. The main requirements are to create an engine which allows a high degree of customisability, an isometric view and the ability to export 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 that would like to create a TRPG. They need not have any experience in creating TRPG before, or even played one before since the game can be created without any programming.

For more advanced users, they can 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 systems.

3.3.4 Constraints, assumptions and dependencies

The system should be portable i.e it should work on most operating systems. To achieve this I chose to use Java since it would work on any system that has the Java runtime environment installed on it.

3.4 Objectives

In the following subsubsections ✓ means that the objective is fully completed, ✗ means incompleteness 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 to defeat all of the other player's units.
 - ✓ Battles are `turn based` meaning only one unit performs at any 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 similar height.
- ✓ Tiles that are not passable such as sea, lava, etc ³.
- ✓ 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 the unit's actions. ⁴
- ✓ 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.

³The engine as well as the GUI full supports impassable tiles. The editor has partial support only allowing 'empty' tiles

⁴At the moment only the height affects the attack, while the direction is displayed in the GUI and changes based on the unit's movement, it not used in the model.

- ✓ 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.⁵

3.5 Specific Requirements

The system should allow the user to export the created game with no additional dependencies apart from the Java runtime environment.

3.5.1 Security Requirements

Although there are no security requirements in the objectives, they could 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 and there are many tools to manipulate the files. The disadvantage of this is that the end users of the created game can also access the data, hence they 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 visually display to the user which action the opponent performs. The GUI should give visual feedback for any actions the user makes.

⁵The engine and GUI support displaying dialog at any time, but the editor only support creation of dialog for the start and end of a battle.

4 Software Engineering Process

4.1 Methodologies Used

I chose to use an iterative spiral development model for the project. This allowed me to focus on specific parts of the system before moving on to the next component.

Prototypes were extensively used, especially 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 assess whether the code matched the minimum requirements. The JUnit library⁶ was used to write the unit tests.

The main stages in TDD are[?],[?]:

1. Before the code is written, unit tests to test the functionality are created. 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 previously created tests.
5. The code is refracted.

The major benefits of TDD are that the system will be well tested. An additional benefit is that it prevents new features from introducing bugs. Combined with version control, as discussed in the next section, it makes it very easy to find bugs since the unit tests can be used to find out *when* the code stopped working as well as find out *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 known beforehand. Since all components of the model are programmed to an interface, it allows the use of mock objects^{4.2}.

However TDD has few drawbacks such as the difficulty of realising all possible test scenarios, which becomes apparent when testing the GUI and the editor. To test these aspects of the system, I played multiple created games from start to finish with the goal of finding any lingering bugs. In addition, I carried out 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 testing. The aim is to create mock objects which simulate only the essential behaviour of the object required for testing. Mock objects abstract the detailed functionality of the implementation away and focus only on what is required for the test.

⁶JUnit 4.8.2, see www.junit.org/ for details

4.3 Version Control

Version control keeps track of all changes to a project. It keeps the history of changes and helps to detect when a bug was first introduced, hence the cause of the bug. In particular I chose to use `git`, which is a distributed version control system. It differs from traditional client server systems such as Subversion in that each user has a complete copy of the repository.

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

5 Ethical Considerations

Although participants were asked to complete a survey to gain feedback on usability as well 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 there 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 pattern has many benefits, as discussed below.

6.2 Engine

The main considerations for the engine was to make it as configurable as possible. To achieve this everything was designed in terms of interfaces, which allowed the particular implementation to be changed. The objects themselves were loaded from their serialised form on disk.

6.2.1 Data Format

XML was chosen as the data format for this project. The main reason is that it is human readable, this allowed me to create and test the data format before the editor was created.

The data was designed to be extendable, as well as provide implementations of the various assets. An advanced user can specify their own custom classes.

Listing 1: Example of Custom weapon

```
<weapon class="custom.customWeapon" uuid="0f0dee33">
  <name>Longicollis</name>
</weapon>
```

Listing 1 shows how a custom weapon can be used. The `class` attribute is the fully qualified name of the class. The only thing the user has to do is to add a `jar` with their classes in it to the `classpath` of the game.

6.2.2 XML schema

XML schema is a way of validating an xml document. A schema was produced for each serialised object⁷. The main use of this was for testing the hand written xml I used before the editor was created. It also helped ensure that the xml produced by the editor was correct.

6.3 Game Progression

Figure 2 shows the overview of how the created game progresses. Each game has a number of maps where a battle takes place. After the map is loaded, any relevant dialogue is displayed, along with the winning conditions. The player's units are then placed on the map, and the battle starts⁸. If the player loses the battle, a gameover screen is shown and the game ends. In contrast if the player wins, he/she advances to the next map, if there is one.

⁷in the `schemas` directory

⁸While the engine allows the user to choose where their units are placed, the GUI does not due to time constraints. The editor supports specifying the starting location for the player's units.

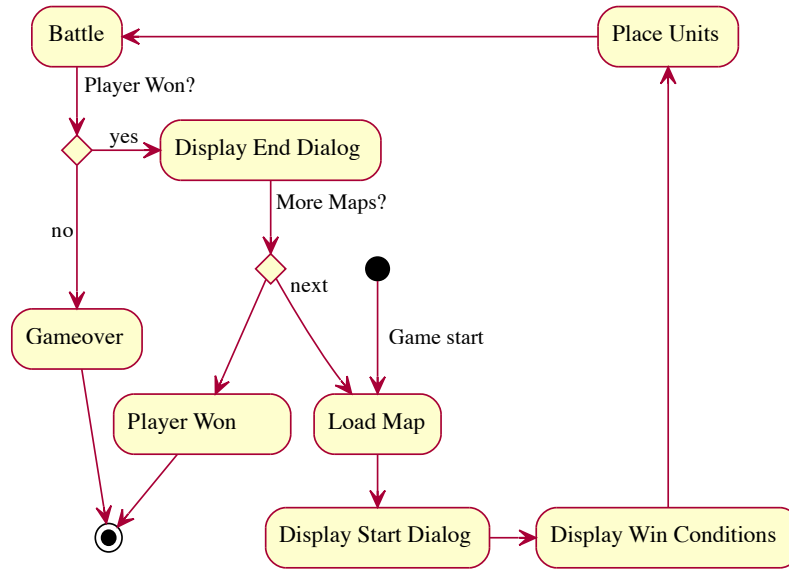


Figure 2: Activity diagram which a high level of overview of the engine

6.4 Game Mechanics

As apparent from the name of the project, a TRPG is turn-based. There were two main choices for turns works; the first is where the players take turns to move all the units, the other method is where the next unit to move is based on some statistic such as the unit's agility. Both methods are widely used, the first more often in games that focus on micro-management such as Civilization but there are exceptions to this such as Disgaea.

The second method has the advantage of allowing more dynamic unit ordering. An example of how this is commonly used: A unit can get their turn quicker if they did not perform any action on their last turn. It also has the advantage of making the game flow much quicker, since the player does not have to wait too long to move their next unit.

I chose to use the second method because it allows the user more customisability since they can choose how unit ordering works. The other reason is based on my own experience, where I think it works better.

6.5 Map Rendering

6.5.1 Introduction

An isometric viewpoint is a way of display 3d objects in 2d while giving the impression that it s 3d. Two different variations are shown in figure 3.

The main benefit of an isometric map compared to a topdown view is that height can be easily displayed.

6.5.2 Tilemap

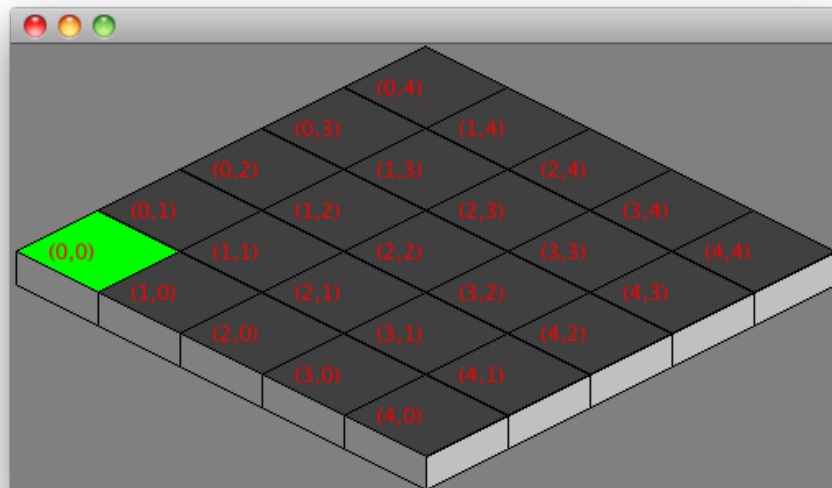
There were two main choices for the isometric tilemap, a "Diamond" map or a "Staggered" map [?], examples of both are shown below in figure 3. Prototypes of each map type were created and the following was found.

The “Staggered” was the first to be considered and it had the following advantages:

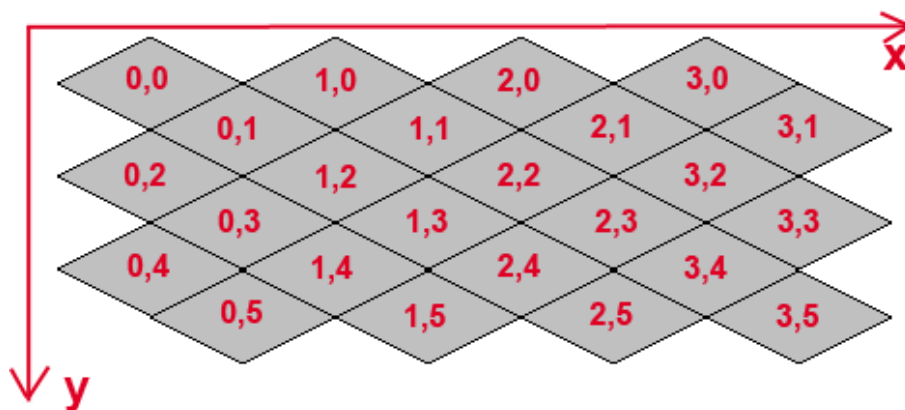
- The map filled up the screen with very little wasted space, so the user can see more of what is happening on the map.

The “Diamond” map was chosen for the following reasons:

- “Diamond” map look more aesthetically pleasing than “Staggered” maps because it has no ragged edges.
- If the map is large enough, it will fill the whole screen negating the “Staggered” map advantage.
- Simpler to think about, since a “Diamond” map is just a rectangular map rotated.



(a) Diamond Map



(b) Staggered Map

Figure 3: The two main types of isometric tilemaps

6.6 Units

I designed the actions a unit can take as a finite state machine as show in figure 4. The main actions the unit can take are:

Move This allows the unit to move to any tile in it's movement range.

Attack Attack any opponent in their weapons range.

Skill Use one of their skills

Wait Finish their turn without taking any action.

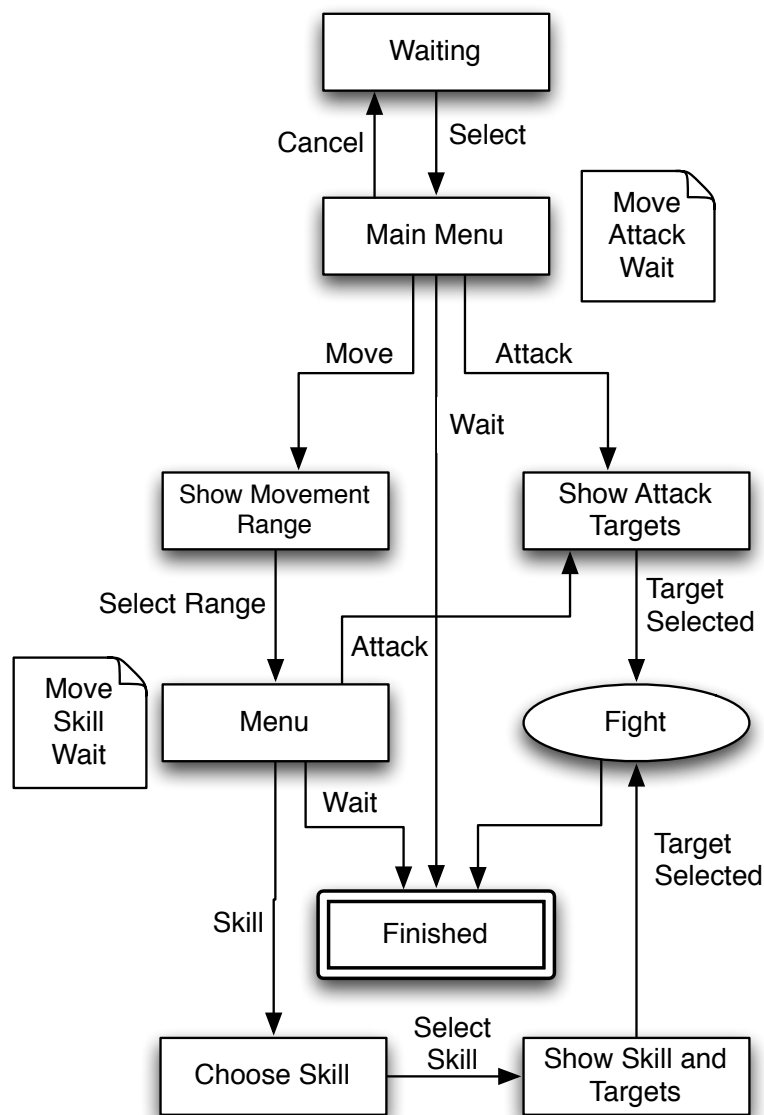


Figure 4: The State diagram of a single turn of a player's unit

6.7 Weapons & Skills

As is common in TRPG, I included weapons as well as skills in the design. A Weapon has a certain strength, a specified range and it may have unique effects. As discussed in section 6.2.1, the weapons and skills are completely configurable, but to enable the user to create common weapons, I designed the following weapons to be included as standard in the engine.

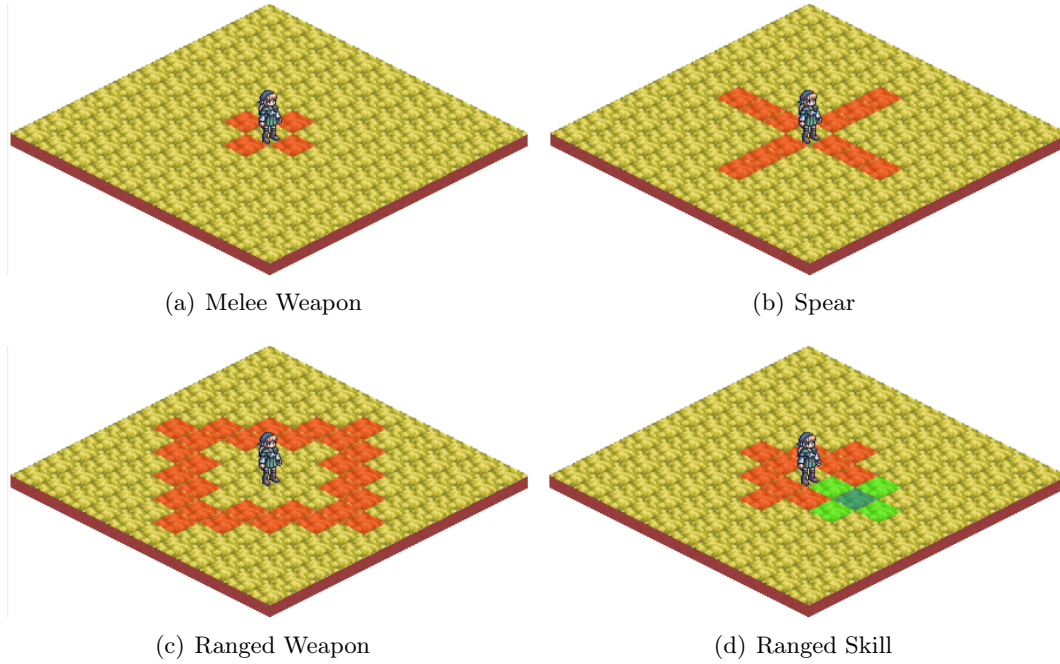


Figure 5: The above figure shows the attack range of various weapons and skill

Melee Weapon this is the simplest kind of weapon, where the unit can only attack adjacent enemies.

Spear A spear can attack enemies in it's range, in a single direction. The unique feature is that it damages all enemies in that direction at the same time.

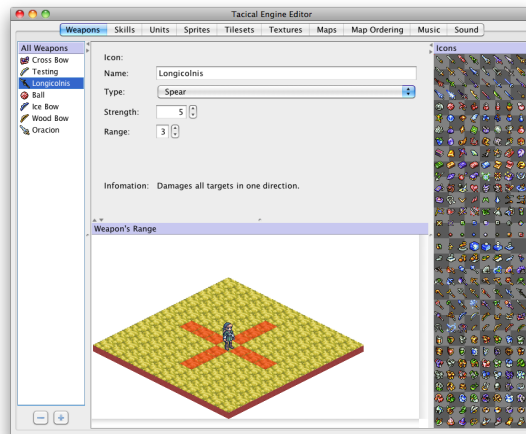
Ranged Weapon A ranged weapon attacks a single faraway target. It has the limitation of not been able to attack enemies which are too close (as shown in the diagram).

Ranged Skill A skill can attack enemies in it's range. The main difference from a weapon is that the skill has an attack area (shown in green) which are the units affected. Unlike most weapons, skills affect *all* units in the attack area.

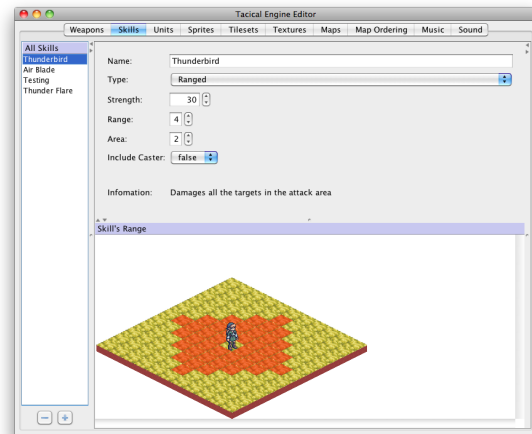
6.8 Editor

The editor was designed to be easy to use. I designed a tabbed interface which allows the user to choose which parts to edit. One of the main features of the editor is the updating of related panels. For example, if the user creates a new weapon, it will automatically be added to the list of available weapons. The main components of the editor are shown below.

Weapons & Skills the editor is designed to allow the user to visually see how changing the stats affect it. This includes showing the user the attack range of the weapon/skill.



(a) Weapons Panel



(b) Skill Panel

Units All the attributes of the units are user editable. This includes their weapon, skill as well as the unit's sprites.

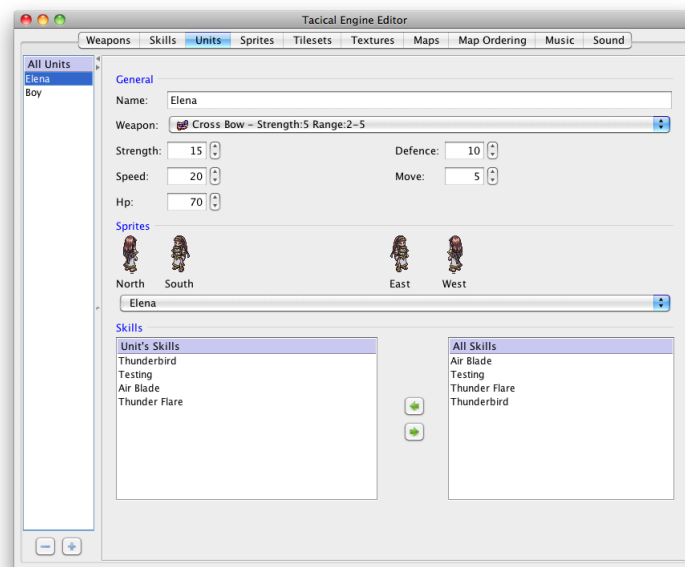


Figure 6: Unit panel

6.8.1 Visual Map Editing

The editor supports visual map editing as shown in figure 7. This allows the user to design their map without writing lots of xml. One of the notable features is that the user can specify the enemy's locations, which gives the user more freedom, compared with older TRPG where the enemies always start on the same corner on every map.

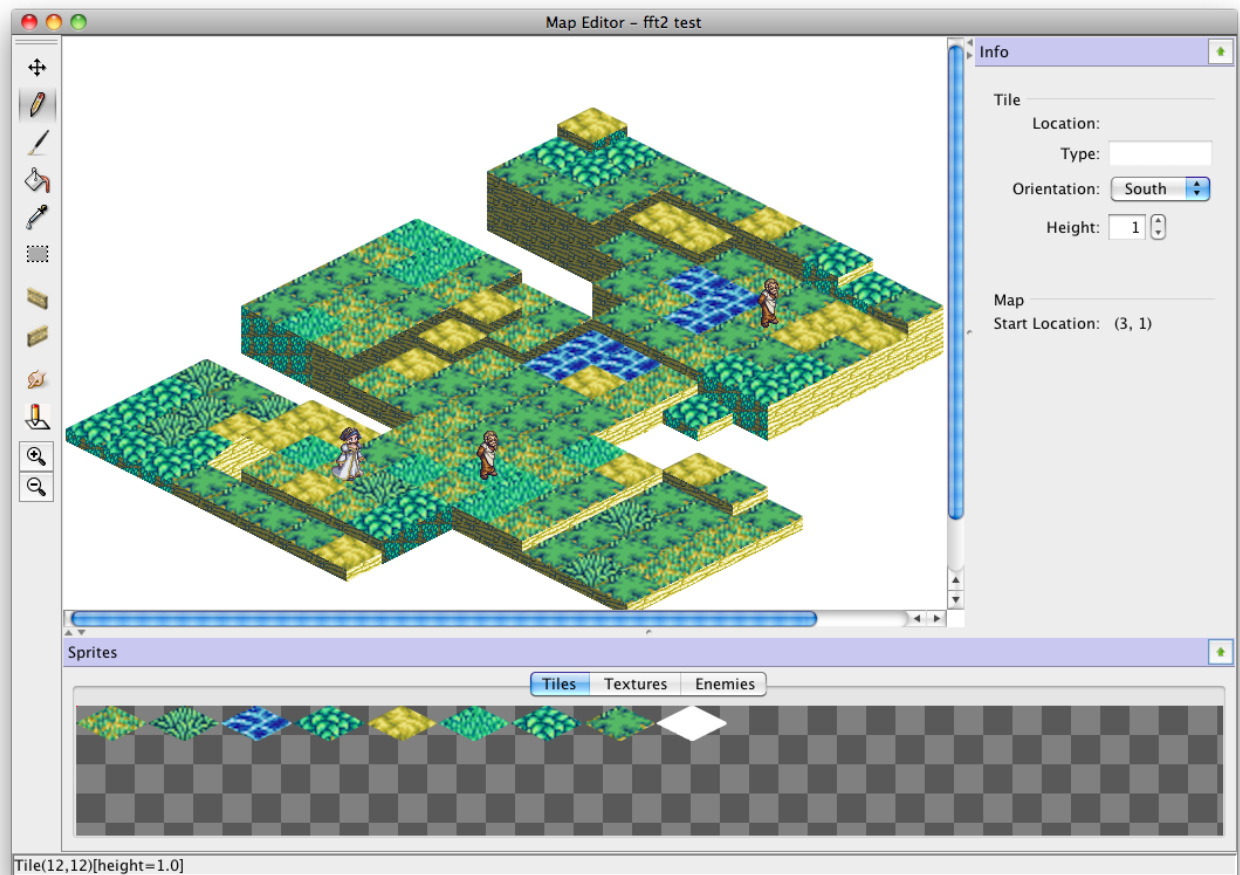


Figure 7: Visual Map Editor

Each tile has many attributes associated with it, these include

- The main image.
- The left and right wall images.
- A specified height.
- An *orientation*, which affects which way the slanted tiles are drawn.

The tools are based on their equivalents in a painting program and include a pencil for drawing the currently selected tile image onto the selected tile. Other tools include enemy unit placement using an icon of a pointing finger.

7 Implementation

7.1 Overview

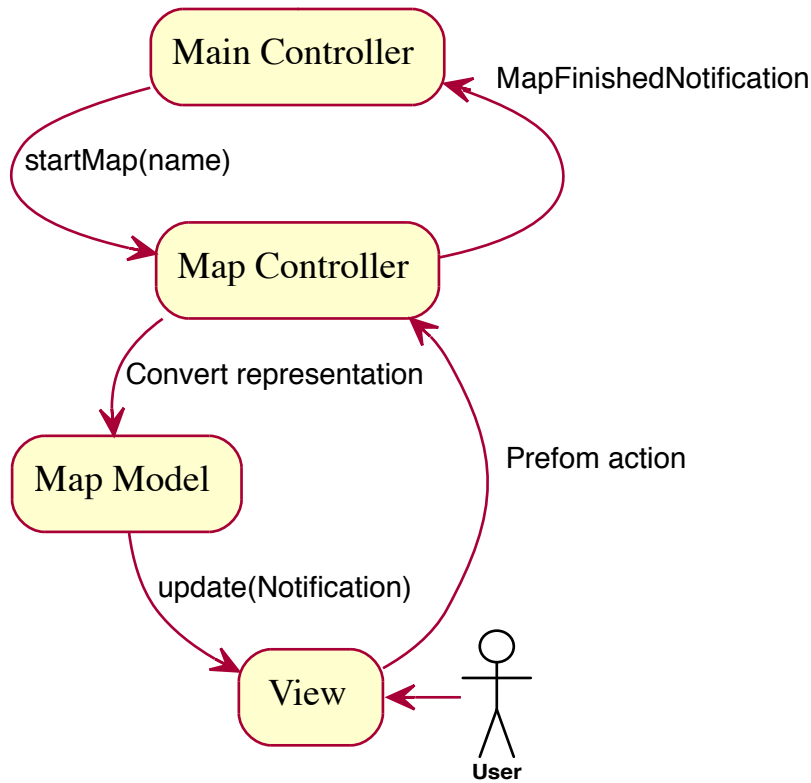


Figure 8: Overview of the implementation.

The system is structured using MVC for the overall architecture as well as using the Observer Patten as shown above.

The `MainController` handles the overall logic, including the game progression. Although the objectives only require a isometric map view, the implementation was designed to be more general hence a *separate* controller for each stage of the game is used. When the stage is finished (e.g. when a map has been completed) the controller notifies the `MainController`, which decides what to do next.

The observable components (i.e. the view, and the map controller) communicate using notification objects which encapsulate any relevant information. For example the model sends a `UnitMovedNotification` when the computer controlled opponent moves one of their units. This notification includes a reference to the unit moved as well as the path it took to get there. The information is used by the GUI to display an animation of the unit moving to the user.

7.2 Data Format

The schema for the data format was only slightly changed for the reason stated in section 7.5. To parse and serialise the xml the Xstream Java library was used.

XStream is an open source library used to serialise Java objects to and from XML. One of its major benefits is that it abstracts over the parsing and serialisation and allows the user to focus on what the data should be used for.

XStream achieves this through the use of Java annotations⁹, as shown in the below example¹⁰.

Listing 2: Example of class that is serialisable with XStream

```
@XStreamAlias("tile")
public class SavedTile {
    protected final String type;
    protected final int height;
    protected int startingHeight;

    @XStreamAsAttribute
    protected final int x;
    @XStreamAsAttribute
    protected final int y;

    protected Orientation orientation;
    protected String leftWall, rightWall;

    private Object readResolve() {
        if (orientation == null)
            orientation = Orientation.TO_EAST;
        if (startingHeight == 0 && height != 0) startingHeight
            = height;
        return this;
    }
}
```

As shown above no extra logic apart from the annotations is needed for serialisation. Another benefit of XStream is that it allows setting default values of the fields. This allows the user to omit redundant tags, as shown in the xml where most of the tags have been omitted.

Listing 3: Serialised form of the above class.

```
<tile x="0" y="0">
  <type>grass</type>
  <height>1</height>
</tile>
```

⁹A special form of syntactic metadata that can be added to source of a Java file, with the notable feature of being retained in the compiled class files.

¹⁰Getter, Setters and trivial constructor omitted.

7.2.1 Resources

All resources that loaded are `Identifiable`, that is they have a unique id. There are two main advantages to using this scheme. The first is that it allows caching of resources which means that there is a single instance for each resource (such as weapons and images). This is especially important for the images, to reduce the memory requirements as well as the load times.

The other advantage is that I could use the same framework for loading and saving all the resources, hence saving development time as well as reducing code duplication.

A detail description of the structure and required resource of a project is in the appendix [B](#).

7.2.2 Sprite Sheets

A sprite sheet is a collection of images combined together. The advantage of this is that a single image is loaded, which reduces loading times. It also makes it easier to cache images, since a *subimage* can be efficiently created¹¹. A subimage shares the same image data as original so it is an ideal way for each tile to have access to its images[?].

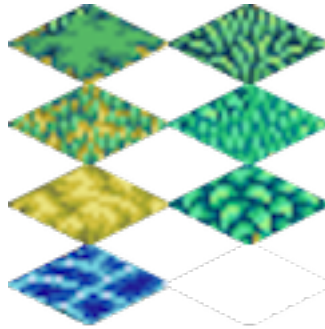


Figure 9: A 128×128 sprite sheet containing a tileset for a map

Sprite sheets make maps more reusable, since the tileset can be changed without any consequences¹².

I created a sprite sheet editor to allow the user to easily edit the tilesets. Using sprite sheets allowed abstracting over the file system, hence increasing the usability of system. The editor was reused for editing other images such as the character images and can even be used independently.

¹¹using the Java method `BufferedImage.getSubimage`

¹²As long as the new tileset has the same number of image or the `tilemapping` has entries for missing images.

7.3 Engine Development

7.3.1 Map

The Map class handles the overall logic and game flow. The main components are:

- The tiles for the maps. Each tile includes height information as well as the tile's location, as shown in listing 2.
- The enemy units.
- The `TileMapping` which specify what images to use in addition to *how* the tile is drawn.
- The conditions of the maps. These include a winning conditions (such as “Defeat All Enemy Units”, the placement of player's units and a `turnComparator` which decides how the turn ordering works.
- The events. These include the dialog which displays parts of the story at the start and end of each battle.

The other major responsibly of the map is to send notifications to the view.

7.3.2 Units

A `Unit` has set of attributes whose values can be specified by the user. A unit is the abstract representation and is used for serialisation. A `MapUnit` extends the unit's sets of attributes with extra information such the location and the current hit points.

A unit has a specified weapon (as discussed in section 6.7). A weapon has a *attack range* which specified how the unit attacks. As an example consider a spear which attacks all units in a specified direction and a bow which attack a single faraway target. A unit with same attributes would play very differently with either of examples since a spear can be only used in melee combat, whereas a bow can only be used from a distance.

A unit has a set of skill (as descried in section 6.7). The main difference apart from the possibility of having *multiple* skills is the concept of an attack area. The *attack area* is tile that are effect by the skill, which includes *both* friendly and enemy units. A skill can either include the caster in the area or explicitly exclude the caster, which gives the user more flexibility.

While default implementation of skills and weapons are provided, the advance user can create their user using the extension mechanism of the data format (see section B.7).

Units can “level up” which means the attributes of the unit increase. Levelling up can also effect how quicker the unit can get their turn. How level up is defined in the battle system as discussed in section 7.3.6.

7.3.3 Movement and Path Finding

To find the movement range movement range of unit Dijkstra's algorithm was used. Dijkstra's algorithm finds the shortest path from a tile to each other tile. This is of course expensive as the map is large so as a improvement I only searched *locally*. Since a unit can move at most n squares in any direction (n being the square the unit is allowed to move) there is no point

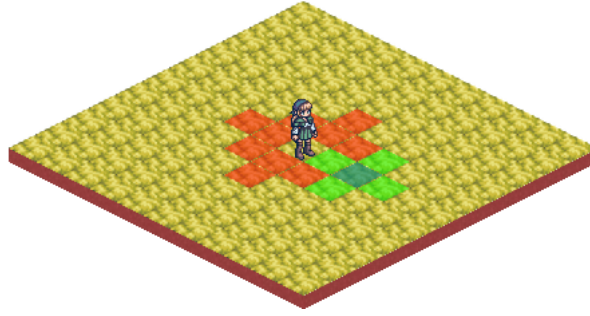


Figure 10: A example of a Skill. The red squares are the *attack range*, the green the *attack area*

checking any tile which is more then n squares away. Searching *locally* means not finding the paths to any `tile` which is more then n squares are away.

The path are then used to display *animated* unit movement to the user. This required keeping track of the *direction* in the movement algorithm.

Other algorithm such as A^* (which is generally an improvement for Dijkstra) was considered for finding the shortest path between two tiles. A^* was rejected since it does not provide much improvement since the paths to *every* tile in range are needed. Since the paths were cached the difference would negligible and hence was not worth the time implementing and testing it.

7.3.4 AI Behaviour

Each has AI unit has a *Behaviour* which specifies how the unit acts and which of the player's unit to target. The engine provides a default implementations which try to get as close as possible to target while attacking any of the player's units that happen to be in range to be on the way. The targets the engine provides by default include attacking the player's unit that has the lowest hp and attacking the unit with highest strength. Like other parts of the engine a user can user can provide their own behaviour to further customised by linking their classes.

7.3.5 Turn Ordering

The engine provides dynamic unit ordering. This allows the ordering to change based on units attributes or actions. The default implementation allows the unit with highest speed to move first. To make the ordering more interesting it also takes account of the unit's action when deciding which is the next unit to move. For example if the unit does not move (i.e. uses `wait`) they get their next turn quicker. As with other parts of the engine the user can uses their classes to further customise the unit ordering.

7.3.6 Battle System

The battle system controls how units interact with each other. It specifies how damage is calculated as any affects results from an action. The engine provides a default implementation where

- The damage taken by a weapon is calculated by adding the weapons power to unit's strength and subtracting the opponent's defence. The lowest a weapon can do is zero, i.e.

negative damage is not allowed.

- The damage done by a skill is independent of the unit's attributes.

The damage done by a weapon is also affected by difference in height of the attack tile and target's tile. Attacking from a higher height gives a bonus whereas attacking from below reduces the attack.

The battle system also defines how units “level up”. The default implementation levels up a unit when they gain 100 experience points. The unit gain experience points by perform actions. Each action gives a different amount of experience point ranging from none from using `wait` to 60 (before any modifiers are applied).

The amount of experienced gained is dependant on level on the attacker and the target. Attacking a higher level unit gives a bonus whereas attacking a lower level unit reduces the amount of experienced points gained.

7.3.7 Conditions

A map has a win condition which can be specified by the user. The engine provides two default implementation, defeat all enemy units and defeat a specific unit. The user can of course provide their own conditions which can be basically anything e.g. move to a specific tile.

7.3.8 Dialog

The engine supports displaying dialog to the user at the start and end of a battle.

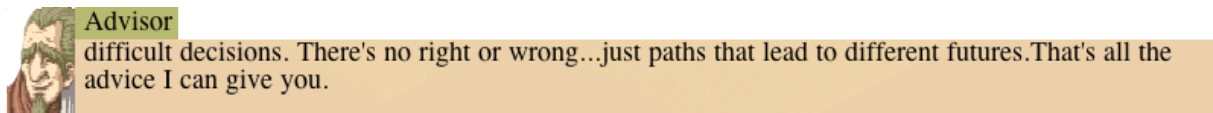


Figure 11: A example of a dialog

One of the major features is that is that the engine handles line wrapping and pagination of the text which is in contrast to many earlier games. This allows the user to focus on writing the plot of game rather than on fitting the text into the dialog box.

Optionally a speaker can be specified for the dialog, this can be used to have a conversation between characters on a map, to make the dialog more interactive.

The editor as discussed in section 7.9.8 supports editing of the dialog. It has the notable features of allowing the user to visually reorder parts of the dialog. It also supports importing and exporting the dialog as a text file. This allows the user to write the script of the game independently in whatever application they prefer.

7.4 Saving and Loading

The engine supports Saving and Loading at any point during the game. The only possible downside for the user is that when the saved data is loaded, game continues from the *beginning* of the map which was lasted played.

The reason for this limitation is to keep the save format small and easy to load. Using this methods means the only details that need to be saved are the maps left to be played and units attributes (which can change since units can “level up”).

The other limitation is that there is only one save file. This not as big limitation as since the save file is stored in users home directory, meaning each user would have their own save.

7.5 Inter-compatibility

As discussed previously the maps use xml as their data format, one the advantages of this was that it required very little changes to the data format to have incompatibility with Oleksandr Stasyk’s Terrain Generator’s output format. The Terrain generator allows uses various algorithms to produce a sensible looking map. Users can use these maps as a starting point for their own creation, hence lowers the time and effort to design a map.

The terrain generator is called by the editor on behalf on the user with appropriate settings¹³ saving the user from having to configure the many options available in the terrain generator (since the terrain generator is a command line application).

7.6 Map Rendering

The GUI uses isometric viewpoint to display the map.



Figure 12: A isometric map which shows the features of the renderers including height, empty tiles, wall textures and unit placement

The map is composed of isometric tiles which have the following features:

- The height is *exactly* half the tile’s width.
- To create the tile, a square tile is rotated by 45° then height is halved.

One the major features of the map render is how the height is specified. Instead of using a bunch of carefully constructed tiles to simulate height, each tile has a specified height which the

¹³In the form of a YAML configuration file `bundle/config.yaml`

renderer uses to draw the tile with appropriate height.

7.6.1 Drawing Algorithm

The algorithm for drawing tiles, first the top of the tile is drawn, then the walls are drawn. It is quite general since it supports tiles which can slant in any direction as shown in figure 13(b) where the tile is slanting to the north as well as zooming and arbitrary height ¹⁴.

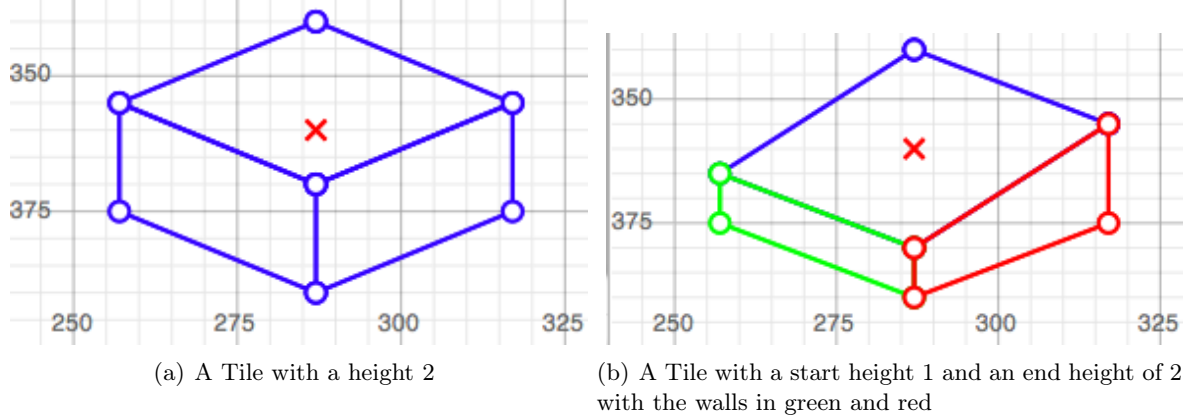


Figure 13: The above figure shows normal and slanted tiles, The red X is the start point for drawing

Mathematically the set of points used to draw tile are as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \left\{ \begin{bmatrix} tx \\ ty - h2 \end{bmatrix}, \begin{bmatrix} tx + \frac{hoz}{2} \\ ty - h2 + \frac{vet}{2} \end{bmatrix}, \begin{bmatrix} tx \\ ty - h1 + vet \end{bmatrix}, \begin{bmatrix} tx - \frac{hoz}{2} \\ ty - h1 + \frac{vet}{2} \end{bmatrix} \right\} \quad (1)$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \left\{ \begin{bmatrix} tx \\ ty - h2 \end{bmatrix}, \begin{bmatrix} tx + \frac{hoz}{2} \\ ty - h2 + \frac{vet}{2} \end{bmatrix}, \begin{bmatrix} tx \\ ty - h1 + vet \end{bmatrix}, \begin{bmatrix} tx - \frac{hoz}{2} \\ ty - h1 + \frac{vet}{2} \end{bmatrix} \right\} \quad (2)$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \left\{ \begin{bmatrix} tx \\ ty - h1 + vet \end{bmatrix}, \begin{bmatrix} tx - \frac{hoz}{2} \\ ty - h1 + \frac{vet}{2} \end{bmatrix}, \begin{bmatrix} tx - \frac{hoz}{2} \\ ty + \frac{vet}{2} \end{bmatrix}, \begin{bmatrix} tx \\ ty + vet \end{bmatrix} \right\} \quad (3)$$

tx, ty are the start point (shows an red X in the above figure)

vet, hoz is the height and width of the tile adjusted for the zoom and pitch

$h1, h2$ are difference in the start height and end height and via verse

where equation 1, is for the top of the tile, equation 2 for the right wall and equation 3 for the right wall.

The equation converted to Java easily since the `Graphic` class has the methods `drawPolygon` which draw a polygon using set of points conveniently in same form as the equations.

¹⁴A simulate of the equations was made first and is in `IsomerticRendering.gcx` (Mac Only).

7.6.2 Images and Texture Mapping

The top of tile can be texture mapped or be drawn directly from an image. Texture mapping repeats a usually small image across the tile. This is the simpler method for the user since they only have to provide a small texture image, but they loose control over where exactly the image is drawn. Texture mapping is also required for slanted tiles since too many variations to provide image for,

The user also has the option of using an image which is drawn directly. To do this the would have to provide their own isometric tile¹⁵. This method allows more control on what is drawn.

Walls like slanted tiles are always texture mapped since they to many variations to provide images for.

7.6.3 Units

Each image has at minimum *four* images associated with it, for each direction (east, west, north, south). These are used to show which way the unit is facing. The unit can also has an animation associated with each direction, to making unit movement more interactive. Th unit can optionally have a portrait which if available would be used for their dialog

7.6.4 Reusability

Since the map renderer is not dependant on Main GUI, it could be reused in many different places in the editor.

7.7 Map Rotation

The map renderer supports for different rotations which the user can choose from. This is implemented *not* by actually rotating the map (since this would be Expensive) but by changing the draw order. By default (0,0) is on the top left of the map. Rotation mealy changes the location of (0,0). This means that none of the tile have changed so all the cached calculations can be kept.

Rotating the map can be used by the user to see units that are hidden or obstructed by tiles.

¹⁵There are script to help the user though

7.8 User Interface

7.8.1 Controls

The main control scheme for the GUI, but the user can perform nearly every action using the mouse. All the elements were designed to work equally well with the mouse and keyboard. This means for example that the tiles are clickable, (which result in tile being selected when clicked).

7.8.2 Menus

The GUI uses a menu system to handle the unit's actions. The menu is quite general since it will resize to the longest item in the menu. It also supports nested menus which are used when user chooses to use a skill. The menu is controllable using the keyboard using the arrows keys, the user can also click a item directory to select it.

7.8.3 Other Features

The GUI supports of other features to improve the user experience these including

- Zoom in/out which lets the user see more detail.
- Changing the pitch of the map (increasing the pitch flattens the map which allows the user to easily see obstructed units).
- A log which can be used to see which actions have taken place. Useful to find out what the enemy has done if the user miss it the first time.
- The key mapping is displayed to user on startup.
- The GUI automatically scroll to enemy unit's location if they are out of range when it is there turn.
- The unit information shows the unit's attributes include the current hit points. They are are displayed in different colour for opposing units to easily distinguished them.

7.9 Editor Development

7.9.1 Overview

The editor allows the user to customise nearly all aspects of the engine. Notable features include visual map making as well as exporting the created game as a completely self contained application with *no* external dependencies apart from a recent version of Java¹⁶.

The editor uses a tabbed interface, the benefit is that it shows the users whats available in the editor. Each tab shares the same overall structure, for two reason, namely to keep the all aspect of the editor consistent hence making it easier it to use. The second reason is that, most of the code can be reused for each tab saving development time.

¹⁶specify Java 6+

In each tab the of current resources created is listed on the left as shown in figure 14. The user can click on any of the resources to customise them. At the bottom of the list are two buttons the left button (–) removes the selected resource whereas the right button (+) adds new resource. Since it is likely the user would create simpler resources (e.g weapon of the same type differing only in Strength) any new created resource takes on the characteristics of last selected item.

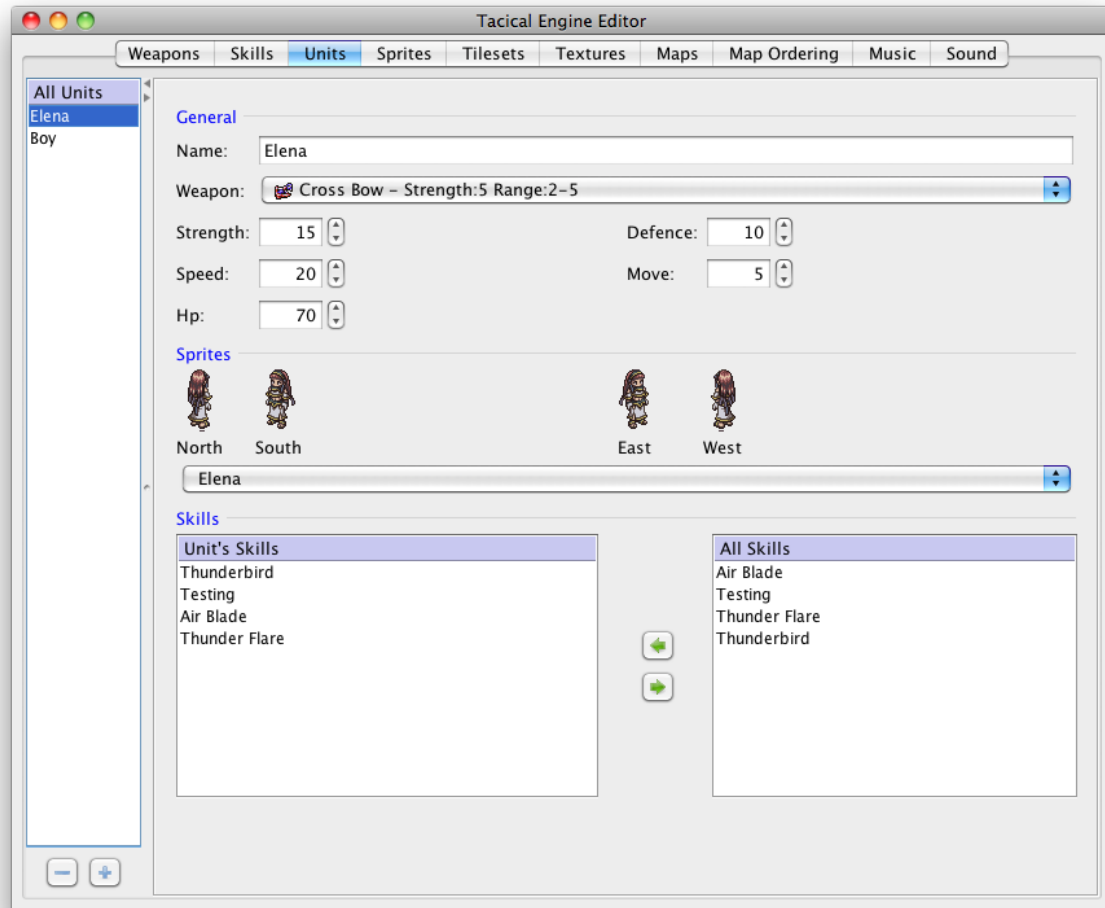


Figure 14: Unit Editor

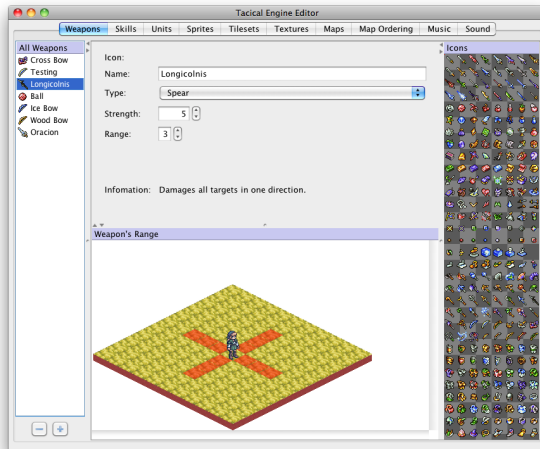
To prevent error and increase usability, `JSpinner` were used wherever numeric input was needed. `JSpinner` allows a minimum and maximum range to be specified, the component will then reject any input which is not in range.

Each of the tabs can only edit their data but can *read* data from other tabs, as shown above where the weapons and skills are received. One of the advantages is the data is always consistent. This was made a lot simpler since the user can only edit one set of resources at a time.

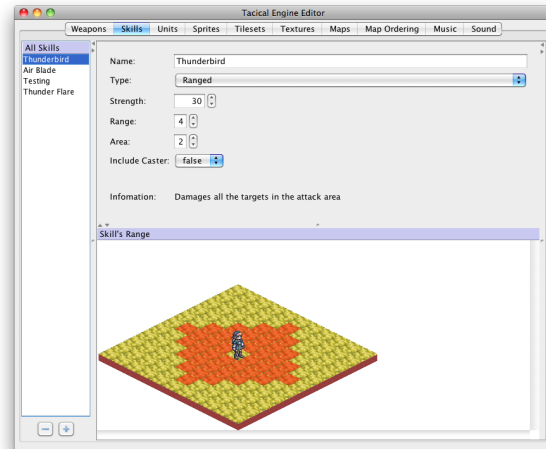
Since the tabs only expose their resources (instead of handle I/O as well), this means that the tabs can be reused e.g. (Sprite, Tilesets and textures share nearly all their code).

7.9.2 Weapons & Skills

The editor support customising the weapons and skills as shown.



(a) Weapons Panel



(b) Skill Panel

The editor shows visually what the attack range looks like. This is updated whenever the user makes any change to the resource. This required very little extra code since the map renderer was general enough to be easily embedded.

7.9.3 Unit Editor

The unit allows the user to edit all that attributes of the units. These include the skills, weapon as well as the images unit as shown in figure 14.

7.9.4 Sprite Sheet Editor

Sprite sheet editor are used for many of the tabs including the units's image, textures and tileset.

7.9.5 Music Editor

The music editor supports choosing which music are available to use in the game. When a track is selected in the editor, the metadata (include the 'artist' and 'album') of the track is displayed. As a further enhancement, the track can be previewed this especially useful when used in combination with the sound effect editor (which has exactly the same interface) to see what the combination of sounds and music sound like.

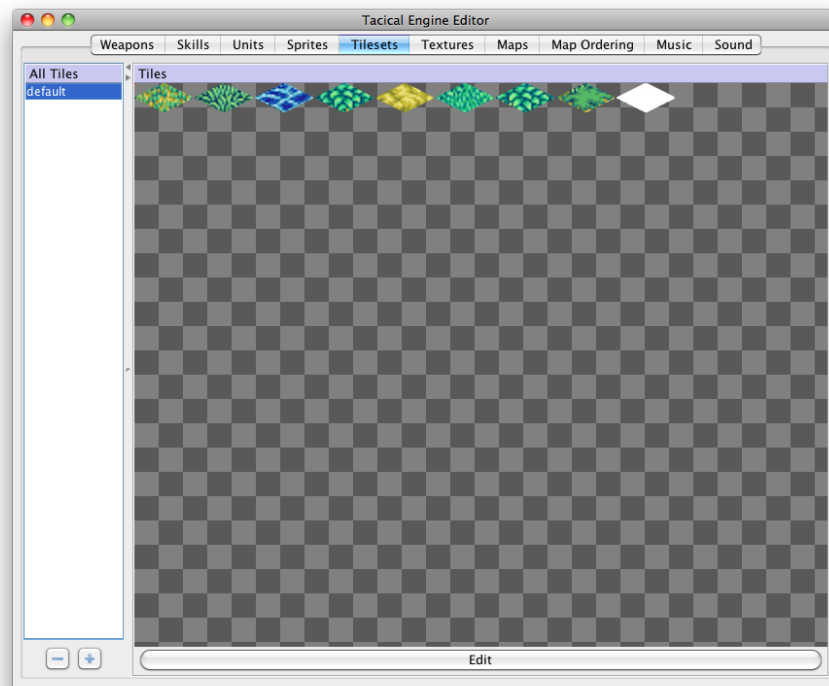


Figure 15: A sprite sheet editor for the tiles

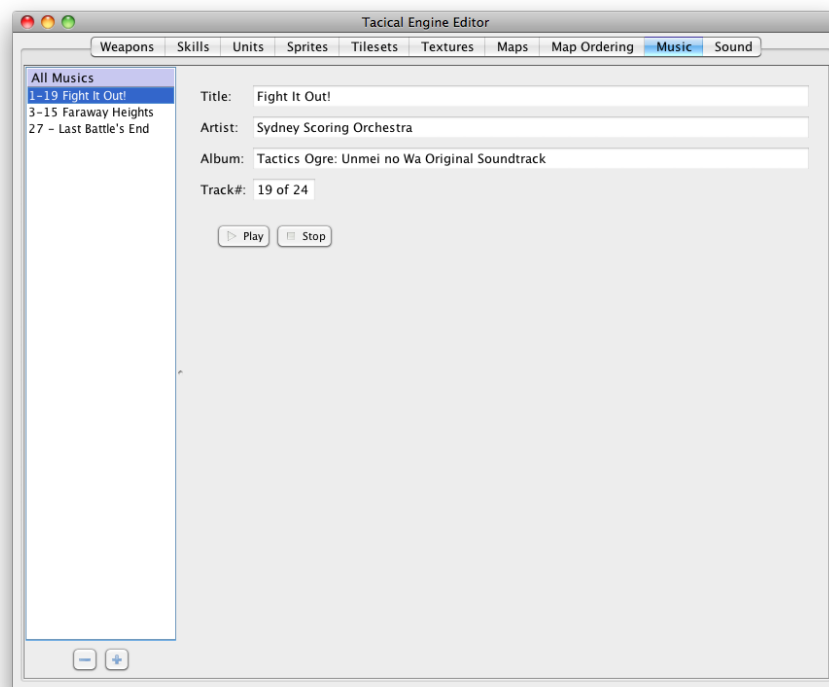


Figure 16: The Music Editor

7.9.6 Map Editor

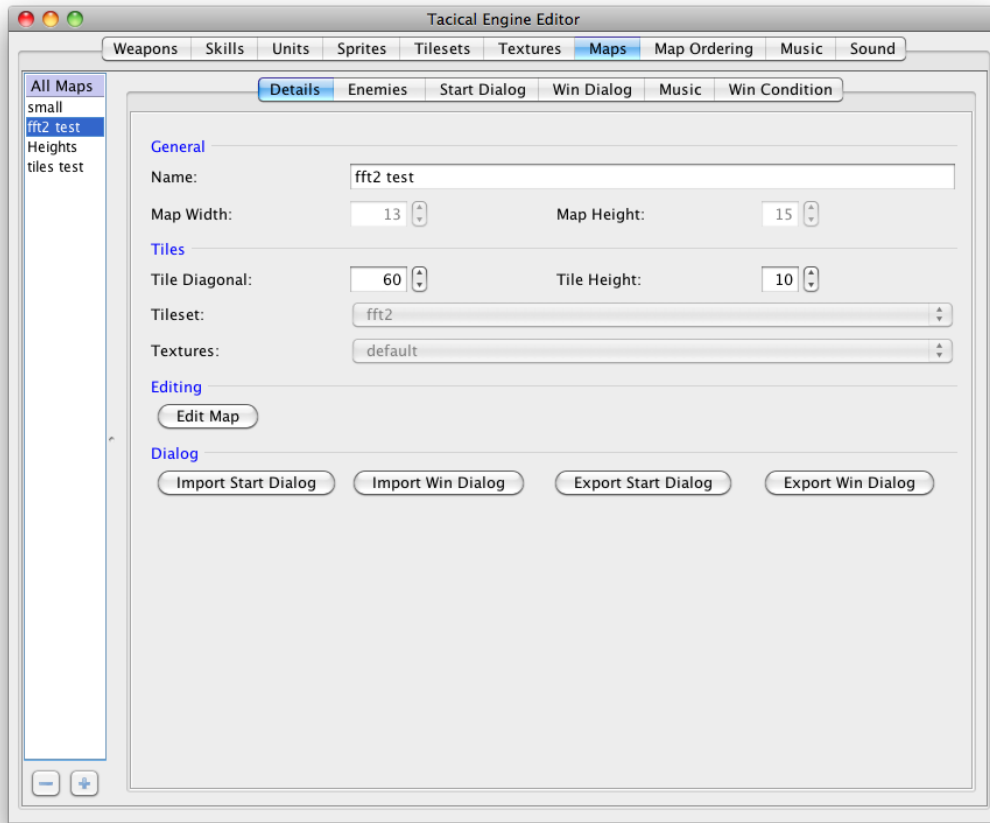


Figure 17: The Map Editor

The map allows the user to customise all the aspects of a created map. The editor is composed using a series of tabs so that the user does not get overwhelmed with all the options available. More importantly it allowed reuse of the unit editor (section 7.9.3) for enemies units (with the addition of extra fields to allow customisation of the enemy behaviour).

Usually all the resources of an editor are loaded when the tab is selected and then cached. While this did work for the maps, it caused, unacceptable startup time hence I found an alternative method.

Lazy Loading only loads a resource when it is actually used instead of just been referenced. In the case of the map editor a map is only loaded if the user tries to *edit* map. This significantly reduces loading time of the editor especially if the user has created a large number of maps.

Details Tab

The details tabs allows the user to change a maps name, as well the size of the tiles. The size of the map can not be changed after creation to keep the implementation of the simple¹⁷. Another reason for not allowing the user to change the map size is that, some of data such as the enemy unit placement may become invalid, which goes against the goal of always keeping

¹⁷Although the engine does support changing the map size.

the data consistent.

For the same reasons the editor does not let the user change the associated tileset or texture set after creating the map although the engine does support it. Note that the user can still add tiles to the tileset, so it is not a large restriction.

The details tab also supports Dialog importing as well as exporting, which will be discussed in section 7.9.8.

Music Tab

The music tab allows the user to select from any music they imported in the music editor (section 7.9.5).

Win Condition Tab

This tab allows the user to choose the condition the player has to satisfy to win the map. Although the engine supports arbitrary conditions, the editor provide the two most common conditions used in TRPG, namely default all enemy units, and default a specific unit, which allows them reasonable customisability.

Map Creation

When creating the map the user can specify the size of map and tileset to associate. An interesting addition to the editor is as previously mentioned, that is to can use Stasyk's terrain generator which can be used as a starting point for the user's creations as shown in figure 18.

7.9.7 Visual Map Creation

The visual map editor allows the user to design a map with writing any xml (the format the engine take as input). The editor shows the available tiles in the associated tileset in bottom spilt. The user can use the tools on in the tool palette on the left to place the tiles. The tools include:

Pencil Draws the selected tile in `Tiles` onto any tile that is click on the map.

Brush Same as Pencil but sets the height and orientation as well (from the info panel).

Paint Set the image of *all* tiles in the selected area.

Eyedropper Set the selected tile in `Tiles` to the image of the clicked tile.

Area Tool Used to select a number of tiles on the map.

Left Wall Set the texture of the left wall.

Right Wall Sets the texture of the right wall.

Pointing Finger Places enemy units on the map.

Pointer to Tile Set the player's start location.

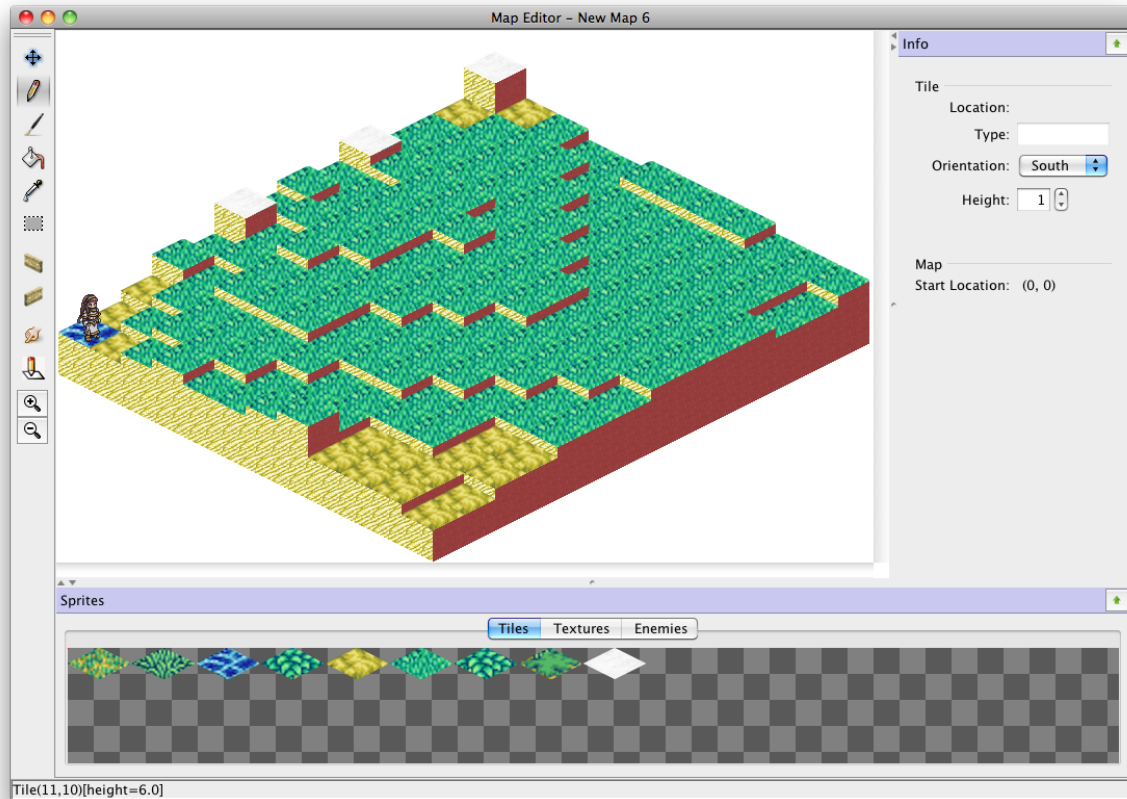
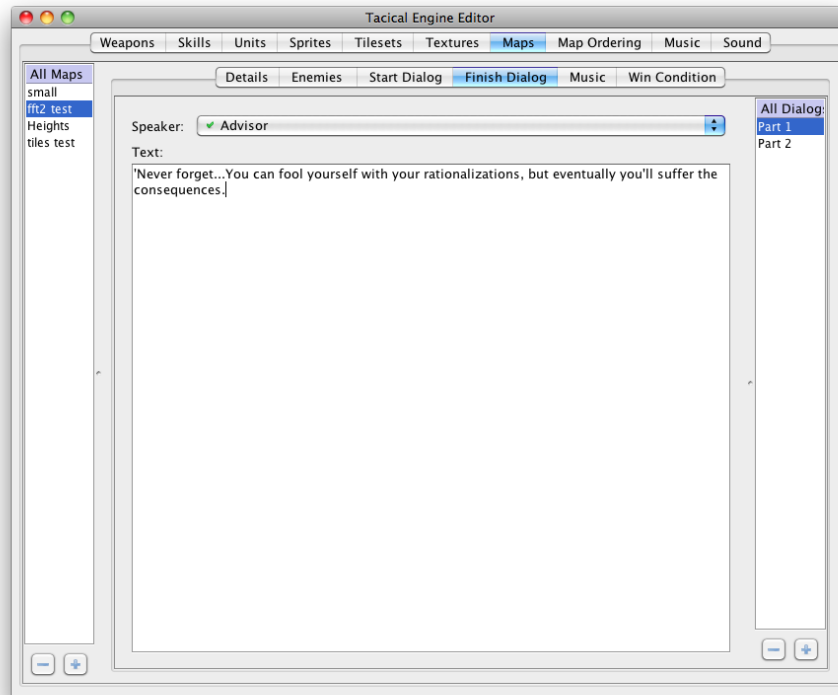


Figure 18: The map editor show a map created by Stasyk’s terrain generator.

The panel on the right displays data on the selected tile on the map. It also allows the user to edit the orientation and height of the tile. As shown in figure 7 Tiles can be hidden. This allows more customisability, e.g. it could be used to make “islands” (a set of tiles with empty tiles around them), and set of bridges between them.

The visual map editor support editing of both textured and non-textured tiles. As discussed before non-textured tiles allow the user more control over what is drawn so they apply more detail. The advantage of textured tiles is that slanted tiles can be used, which easily allow the creation of slopes and stairs for example.

7.9.8 Dialog Editing



1

Figure 19: The dialog editing panel

The editor supports the creation of dialog. A dialog is a sequence of parts. Each part has the text associated with it as well optionally having a speaker. The speaker can be selected from any unit that has already been placed on the map. This is a vast improvement on the original design where the user types the name of the speaker since there is no less chance of entering invalid data.

As mentioned before, the engine takes care of pagination as well line wrapping when displaying the dialog in the game. This frees the user from having to fit the text into a specified area.

Import/Export

The editor also supports importing as well as exporting the dialog in format shown below.

Listing 4: Shows the format used for the dialog

```
- speaker1: Some text
- speaker2: Some more text
- none:      This part has no speaker
```

This allows the dialog to be easily written in the user's preferred application. This could be useful if separate person writes the dialog of the game since the person does not need a copy of the editor. The format is described in more detail in the appendix [B.6.1](#)

Although the dialog xml format could be edited by the determined user, this would require significantly more effort than the above format since the user would have to find the unique id

of each unit each they want use as a speaker. It would also require creating unique ids for dialog part which would be likely be error prone, hence the above format is used in import/export, mainly for the user's conveniences.

7.9.9 Project Selection

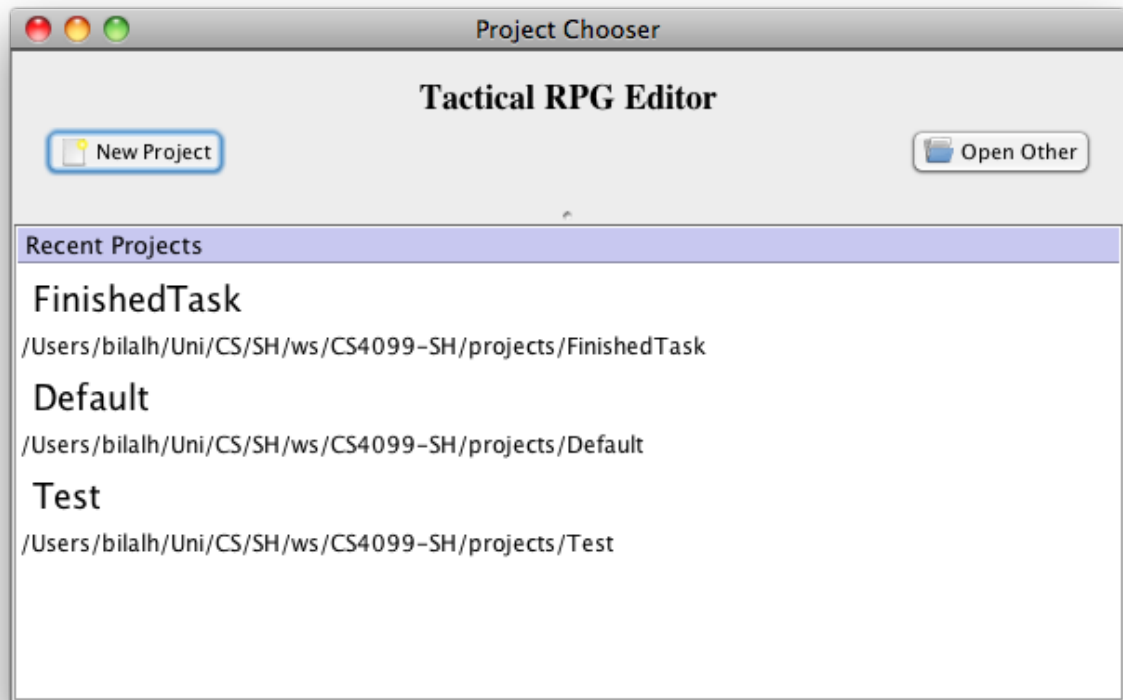


Figure 20: The Project Selection Window which is shown at startup

When the editor starts It displays the above startup screen which allows the user to create a new project or open an existing project. Since a user is likely to work on the same project for a significant period of time, the editor keeps tracks of recent projects.

7.9.10 Exporting

The editor can export the game as a complete package, either as a Mac OS X application or as jar. These application don't require any external resources, apart from a recent version of Java¹⁸.

A prominent feature of the editor is that the 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 exported on other platforms.

While most of the testing was done on OS X ¹⁹, it also works well on Linux ²⁰. It even has limited compatibility with Windows ²¹ (apart from some minor graphics issues).

7.10 Ant Build File

The task of creating the Editor as a self contained application with no external dependencies apart from obviously Java is non-trivial and error prone. To automate this task an Ant build file was created, which has a added benefit of allowing the creation of distribution from the command line.

The main steps to create the self contain editor are:

- Compiling the code.
- Creating the jar of the engine.
- Creating the jar of the editor
- Creating the directory structure for the editor Mac OS X application.
 - This includes creating the plist[?] with the applications.
 - Setup the native libraries and the Java include path.
 - Adds the icons.
 - Linking the Java wrapper so that icons appears in the Dock.
- Create the cross platform Jar for Editor.
 - Combines the Jars into a single Jar.
 - Set up the native libraries path.
 - Set up the Java libraries path.
- Create the cross platform Jar for exporting the game.
- Create Mac OS X application for exporting the game.
- Copy the resources for the starting project.

The ant build files takes care of all the above steps and can be invoked by doing `ant dist`. The build files has many other features such as running all tests and creating a web page. These features are described are the top of the build file (`build.xml`).

¹⁸specifically Java 1.6+

¹⁹Mac OS X 10.6 Snow leopard

²⁰Science Linux x.y

²¹Tested on Windows 7 32 bit

8 Evaluation and Critical Appraisal

8.1 Objectives

The objectives as listed in section ?? are extensive, ranging from primary objectives that required a basic engine capable of creating a Tactical RPG to tertiary objectives which required the creation of an editor which customisable most aspects of the engine.

Since all the primary and secondary objectives were completed as well as nearly all the tertiary objectives I consider the project a success, I will now discuss the objectives in details.

8.1.1 Primary Objectives

The primary objectives focus on basic functionality of the engine, I have marked these objectives as completed. Notable improvements from the objectives include the engine handling the pagination and line wrapping of the dialog automatically which frees the user from fiddling with the dialog to make it fit on the screen.

Combat between units was as primary requirement, which was completed with many additional features, which are discussed later.

The primary requirements were to make an isometric graphical representation of the game, this was very successful. The view had useful features such as allows the game to be played with either the mouse or the keyboard.

8.1.2 Secondary Objectives

The main goal of the secondary objectives is to allow the user more customisability. This associating attributes such as height and movement cost to tiles was a requirement that was completed. In addition, the engine supports *empty* tiles which are impassable and not displayed to the user. This allows the creation of more complex maps as shown in figure 7. Another enhancement is that *slanted* tiles can be used, they are tiles that have a different start and end height and could be used for making stairs or cliffs.

The secondary objectives required a combat system that supports the use of weapons and non-adjacent targets for both the player and the AI opponent. Weapons in particular deserve mention since they are completely configurable. The engine of course provides default implementations of common weapons such as ranged weapons and spears (which shows the weapons can have extra effects such as attacking multiple targets).

The default implementation of the battle (User can use their own battle system, but it's slightly complex) takes many attributes such as the units' level and difference in height between the attacker and the target to determine the outcome of a battle.

I completed the objective of allowing music and sound effects, the format used is Ogg Vorbis (Appendix B.4). In addition, wave files can be used for sound effects, this was implemented since the format seems to be commonly used for this purpose.

Save and loading the game were implemented as required, but with the restriction that when a save file is loaded the player begins at the *beginning* of the last played map.

Allowing the user to customising the AI behaviour, is a completed objective. In addition the engine provides default behaviour including the example of attack the player's unit with lowest HP. The engine also provides default actions such when moving to a faraway target, if the target is not reachable, most as close as possible and attack any player's unit that is in ranch.

8.1.3 Tertiary Objectives

The main objectives of these requirements is to produce a editor that allowed customising most aspects of the engine as well as combat improvements and units animations, all them having being completed.

Combat requirements improvements include `skills` that can affect multiple units, and weapons which can attack multiple units. Further enhancements to this system include complete customisable range (e.g could be used to attack all enemy units around the attack).

The requirement for animated unit movement was completed. This further improved by taking account of the direction the unit is traveling in, which was done by have a image for each of the four directions.

Editor

A editor that can customise nearly all aspects of the engine was created. The most prominent features are the visual map editor (section ref), unit creation(section ref), dialog editing(section ref) and exporting as a self contained application. The editor met all the tertiary requirements also had addition improvement. These include a easy to edit dialog format for importing/exporting and inter-compatibility with Stasyk's Terrain Generator to give a user a start point for their map designs.

Scripting & Custom Events

A tertiary requirements was to implement custom events though the use of scripting. This would allow the user to specify events such as making the player win if some enemies unit has less then 50% Hit Points.

To implement this I had planed to use the Java Scripting API[?]. The would allow embedding a dynamic scripting language. The main benefit of this, is that the code does not have to be compiled which means it does have the same hurdles with class linking as my Extension Mechanism(Appendix B.7). The language chosen for embedding was Javascript as it is build into the JDK, so the user does not have install anything extra and of all the scripting language Javascript is more likely to have been used before by the average user, these design are described in more detail in section D.

Some people such as ref advocate forgoing scripting completing. Pointing out that scripting is a lazy out of let user customisability since it can be quite for user to create events. Using their advice of creating visual editors for creating events instead I allows the user to specify common events from the editor, with the option of link they own classes as a last resort.

Although the scripting was not implementation due to time constraints, note that it was tertiary objectives and all the example can be still be specified by editor. In particular win conditions can be specified (section ref) and common winning condition can even can be selected

from the editor. The engine support shows dialog at arbitrary point in the game which could be used to support the requirement “Showing some part of the story when a player’s character reaches a specified tile”.

The engine supports the user using their own classes though the extension mechanism in Appendix B.7, which allows a slightly more limited form of scripting but with the benefit of being type safe but with the drawback of being more complex.

8.2 Results of User Testing

In total six participants took part in the user testing, more people were interested but could not due to time constraints. See Appendix C for the Questionnaire.

8.3 Analysis of Responses

The participants can be grouped into three groups, those who have played a TRPG before, those who have not and those who don’t play games at all.

Feedback from all groups

Both groups thought the editor supported all the features they would like if they created a TRPG. Most of the participants answered that they would like to use the editor again. Interestingly, all users who were familiar with TRPG used the keyboard exclusively, this I attribute to participants being used to the input mechanism since most TRPG are on consoles²². Those who had no experience playing TRPG unexpectedly used both the keyboard and mouse at the same time (generally selecting the units with the mouse then using the keyboard for performing any actions).

Feedback from those who had played a TRPG before

Participants that had experience playing a TRPG before found the engine very intuitive. They especially liked how everything was customisable. The visual map editor also gained significant praise for it’s easy of use and it’s range of features. Other features that participants liked were how a project could be exported as a standalone Mac OS X application with virtually no effort.

When playing the pre-created game, all of the participants were able to play, and even win, the game!. Some of the participants were able to figure out all of the out the controls with out even reading the help screen.

Feedback from those who had not played a TRPG before

In contrast, participants who never played a TRPG before felt overwhelmed with the available options, although they figured how to use it quite quickly. They particularly liked how robust the editor was. The main features that were requested were an overview of each feature in the editor and how they interacted. The participants commented on the pleasing visual appearance of the editor.

When playing the pre-created game, some of the participants were initially confused by the interface. All participant were able to finish the game even if they could not win. The participant

²²such as Playstation 2

unexpectedly kept on using the enter key to perform actions on the unit even though the help panel stated otherwise. In response to this feedback, I added the enter key as an alias of the action key (default 'x').

8.3.1 System usability scale

A System Usability Scale (SUS) was used [?]. This works by giving each even numbered questions a score of (5 - *value*) and odd numbered questions a score of (*value*-1). Questions that contributed to a high score showed that the system is usable.

Based on this schema, the maximum positive contribution is four. To get the overall usability score, the sum of the questions contributions is multiplied by 2.5.

8.3.2 Results

The usability result was 61. This is a very good result since a score of 50 means that the system is useable.

8.3.3 Analysis

Since no individual question scored below two, the system has most of the features the users wanted.

9 Conclusions

9.1 Future Work

- Improvement to levelling up. Usually a unit does not have access to all of its skills at the beginning of the game, but it gains access to them when levelling up. This would make the produced game more balanced, since only skills appropriate to the unit stats could be used.
- Implementation of an overworld map with a battle occurring 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

10 Testing

- 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 Project Structure & Specification

This appendix contains all the technical information of the ‘project’ and the data format used.

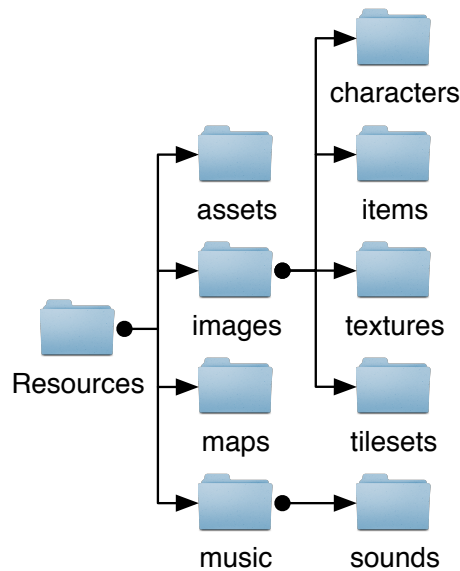


Figure 21: Project Structure

A game’s resources are organised as shown above. One of the main restrictions, is that all external links have to be relative to the `resources` directory. All xml files **must** conform to the schemas in the `schemas` directory.

B.1 Assets

Assets are stored in the following format:

Listing 5: Assets format

```

<name>
  <entry>
    <uuid>128bit</uuid>
    <resource uuid="128 bit">
    </resource>
  </entry>
</name>

```

The `assets` directory **must** contains the following files conforming to schemas in `schemas/assets`.

Listing 6: Required Assets

```

maps.xml
music.xml
ordering.xml
skills.xml
sounds.xml

```

```
textures.xml
tilesets.xml
units.xml
unitsImages.xml
weapons.xml
```

B.2 images

All images are stored as sprite sheets(see section 7.2.2). There are *three* required files for each sprite sheet.

```
name.png
name.xml
name-animations.xml
```

The sprite sheet itself is stored as a png(Portable Network Graphics) in `name.png`. `name.xml` contains the coordinates of each image in the sheet as well as the dimensions of the sheet. `name-animations.xml` contains the unique id of the sprite sheet and can optionally specify animations.

B.3 Maps

Each map needs the following *five* files:

name.xml which contains the tile data as well as references to the other files.

name-conditions.xml which includes among other information, the winning conditions.

default-enemies.xml which contains the enemies data along with their positions on the map.

default-events.xml which optionally contains the dialog which is shown at the start and end of the battle.

default-music.xml which contains the background music and sound effects.

Maps also need a `tilemapping` which maps the tile's type to their image. A default `tilemapping` is created by the editor when a tileset is saved with the name `tileset-mapping.xml`.

B.4 Music

The engine supports only Ogg Vorbis which is “a completely open, patent-free, professional audio encoding and streaming technology”[?]. Compared with other formats such as MP3, Ogg Vorbis has no licensing issues.

B.5 Sound Effects

Sound effects can either be Ogg Vorbis, or wave(.wav) format. Sound effects should be less than 7 seconds, otherwise the sound effect may be truncated.

B.6 Editor

For the editor the follow structure is used.

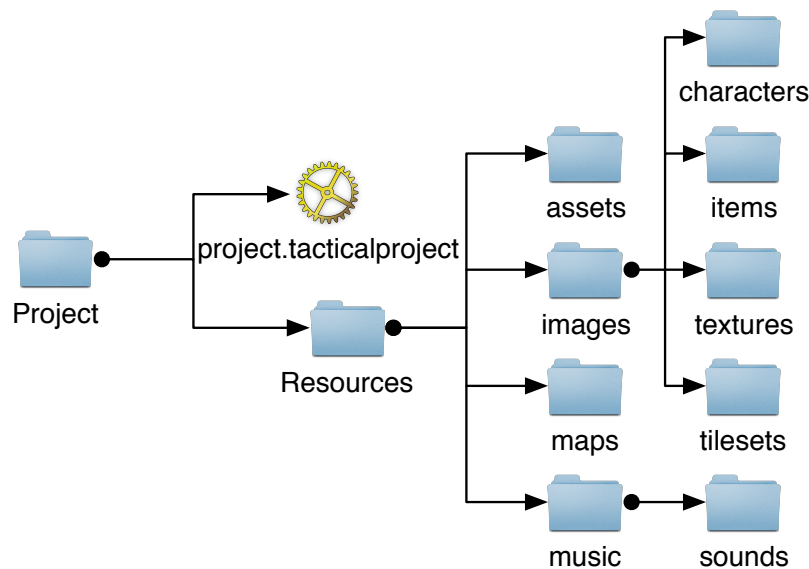


Figure 22: Editor Project Structure

The main change is the addition of the `project.tacticalproject` file which contains settings specific to the editor. It also has the added benefit of allowing the user to click on the `project.tacticalproject` to open the editor if using the Mac version.

B.6.1 Dialog Data Format

The dialog format is actually YAML[?]. This allows more formatting options as well the simple format discussed previously. The advantage of using YAML is that there are parsers available for java[?]. This saved development time since I did not have to build a parser. The main advantage of YAML is that there are very few characters that need to be escaped namely `:` if it appears at the start of the dialog's text or speaker's name. `#` also has to be escaped since it is the comment character in YAML. Since these characters are fairly uncommon in dialog, it should not pose too much inconvenience to the user. In any case the use of the form `- Speaker: |-`, then writing the text on an intended line escapes all characters in the block, solving this problem.

Listing 7: A example show the features of the dialog's data format

```

- Speaker:    Some text.
- none:       no speaker.
- New Enemy 2: |-
    "All the text in this block (until two newlines),
    are part of New Enemy 2's dialog".
    No characters need to be escaped in this block e.g #    : \
- Other speaker: Even more text.
  
```

Specifically the dialog must satisfy the following:

- At least one portion of text. A empty file is invalid
- Each portion of text must have a speaker identified by their name or none for no speaker.
- No other elements are allowed.

When the dialog is imported, if there is a unit with the specified name, it is associated with the portion of text otherwise the text is treated as having no speaker.

B.7 Data Format Extension Mechanism

- can specify classes

C 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 23: 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

C.1 Editor Usability Scale

© Digital Equipment Corporation, 1986.

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

← strongly disagree agree completely →

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

2. I found the system unnecessarily complex.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

3. I thought the system was easy to use.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

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

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

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

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

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

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

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

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

8. I found the system very cumbersome to use

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

9. I felt very confident using the system

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

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

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

C.2 Playing a pre-created game

1. I found the game intuitive

← strongly disagree agree completely →

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

2. The game had a appropriate level of difficulty.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

3. I enjoyed playing the game.

<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
--------------------------	--------------------------	--------------------------	--------------------------	--------------------------

4. Please share any comments about the game :

C.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:

D 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.

D.1 Language Choice

There were three main choices using Javascript, using JRuby²³, 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' ²⁴ 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.

D.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

²³A Ruby implementation written in java

²⁴A 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.

D.3 Action

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

D.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()`

D.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.

D.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)`

D.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 8: 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
```

E BugFixes notes

- Not adding ext to files
- Allows not ogg files for music
- Dialog import/export
- Terrain gen
-