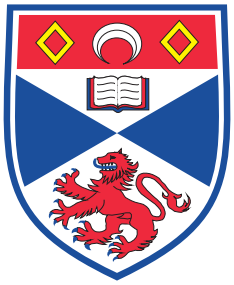
University of St Andrews

CS4099: Major Software Project



A Tactical RPG Engine

Bilal Syed Hussain

Supervisor:
Dr. Ian Miguel

Second Marker:
Dr. Alex Voss

April 4, 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[1].

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] Although there are some common vailent that will be discussed later on

1.2 Project Success

Having completed all the primary objectives as well as nearly all the secondary and tertiary objectives I would say that the project was successful. Notable features that were accomplished include a very customised engine which allows the user to specify most aspects of the 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 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 environment.

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

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 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 handling the game 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 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 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 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 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 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 used 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 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 to 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 similiar 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.

3.4.3 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.

[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.

- ✓ 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 additional dependencies apart from the Java runtime environment.

3.5.1 Security Requirements

Although their 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 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 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.

4 Software Engineering Process

4.1 Methodologies Used

I chose to use a iterative spiral development model for the project. This allowed me to focus on specific parts of the system before moving onto 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 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 the code is written, unit tests to test the functionality is 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 previous created tests.
5. The code is refracted

The major benefits of TDD are that system will be well tested. A 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 stop working as well as to 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 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 becomes apparent when testing the GUI and the editor. To test these aspects of the system I played multiple created game from start to finish with the goal of finding any lingering bugs. In addition 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 objects which simulate only the essential behaviour of the object required. Mock objects abstract the detailed functionality of the implementation away and focuses only on what is required for the test.

[3] JUnit 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 cause of the bug. In particular I chose to use `git`, which is distributed version control system. It differs from traditional client server system 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 human subjects were asked to complete a survey to gain feedback on usability as well 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 considerations for the engine was to make it configureable 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 advance user can specify their own custom classes.

Listing 1: Example of Custom weapon

```
<weapon class="custom.customWeapon" uuid="0f0dee33">
    <name>Longicolnis</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 a xml document. A schema was produced for each serialised object[4]. 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 winning conditions. The player's units are then placed on the map, and the battle starts[5]. If the player's 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.

[4] in the `schemas` directory

[5] While the engine supports the user's to choose where 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.

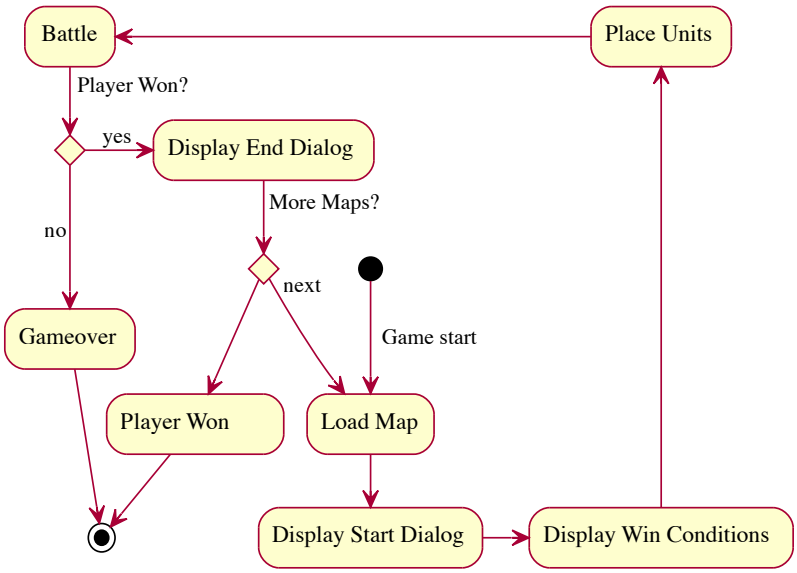


Figure 2: Activity diagram which a high level of overview of the engine

6.4 Game Mechanics

As apparent from the name of the project, a TRPG is turn based. There were two main choices; 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 stat 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 the 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 experience where I think it works better.

6.5 Gui

6.5.1 Tilemap

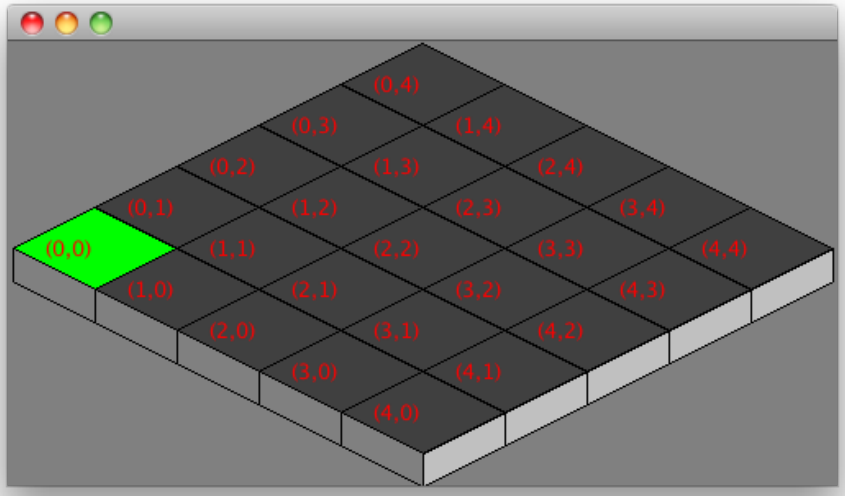
There were two main choices for the isometric tilemap, a "Diamond" map or a "Staggered" map [4], examples of both are shown below in figure 3. Prototypes of each map type were created and the following was found.

The "Staggered" was the first to be considered and the following advantages:

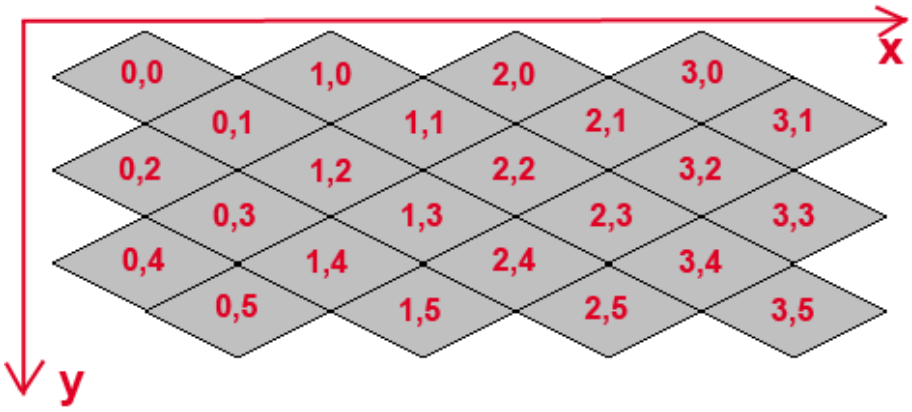
- The map fill up the screen with very little wasted space, so the user can more of what happing on the map.

The "Diamond" map was chosen for the following reasons:

- "Diamond" map look more ascetically pleasing then "Staggered" maps because it has no ragged edges.
- If the map is large enough it will fill the whole screen negating "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 allow the unit to move to any tile in it movement range.

Attack Attack any opponent in their weapons range.

Skill Use one of the skills

Wait Finish their turn without taking any action.

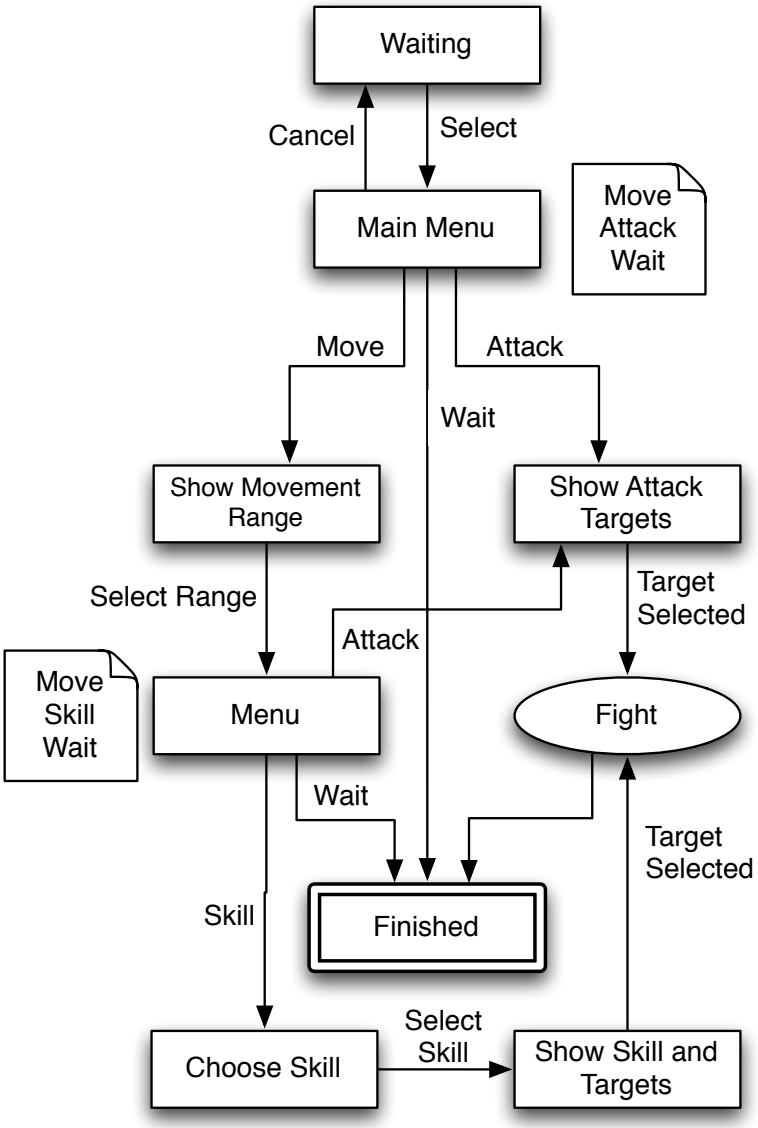
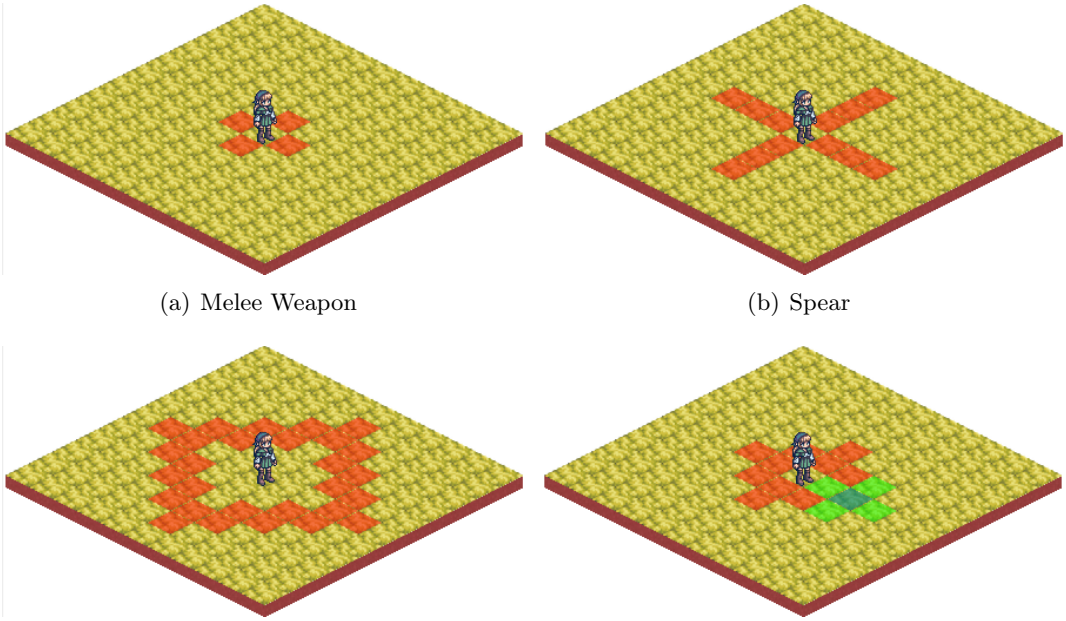


Figure 4: The State diagram of a single turn of a player's unit

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 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 include as standard in the engine.



(a) Melee Weapon (b) Spear

(c) Ranged Weapon (d) Ranged Skill

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 range, in a single direction. The unique feature is that it damages `all` enemies in that direction at the same time.

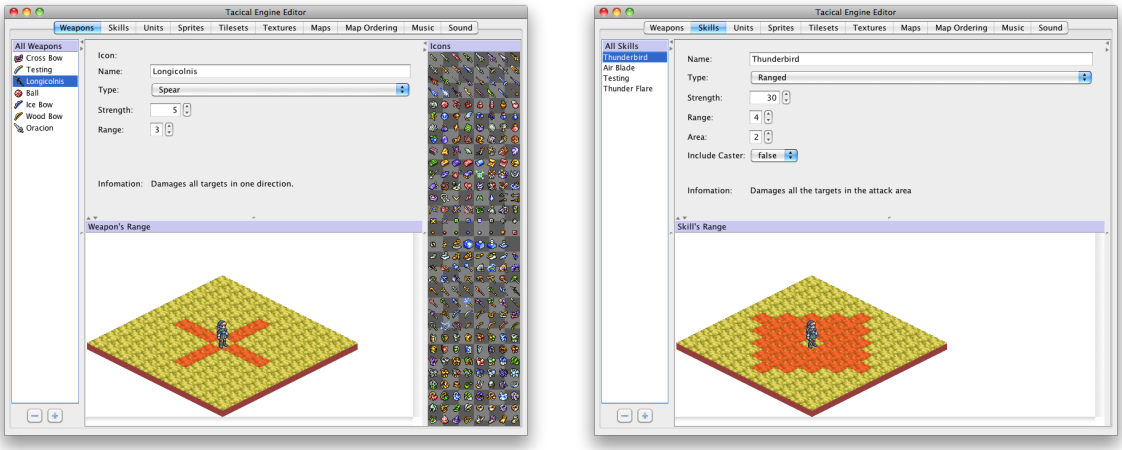
Ranged Weapon A ranged weapon attacks a single faraway target. It has the limitation of not been able to attack enemies which are too close (as shown in the diagram).

Ranged Skill A skill can attack enemies in 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 allow the user 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 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 how changing the stats affect the 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.

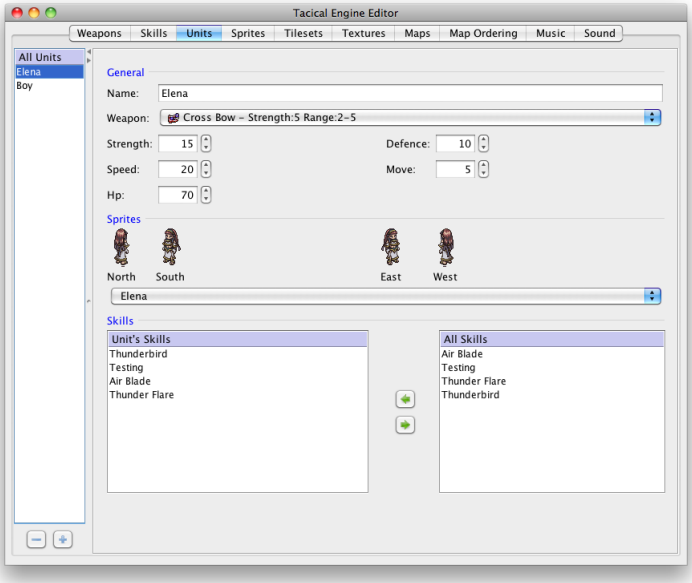


Figure 6: Unit panel

6.8.1 Visual Map Editing

The editor support 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 enemies locations, which give the user more freedom compare with older TRPG where the enemies always start on the same conner on every map.

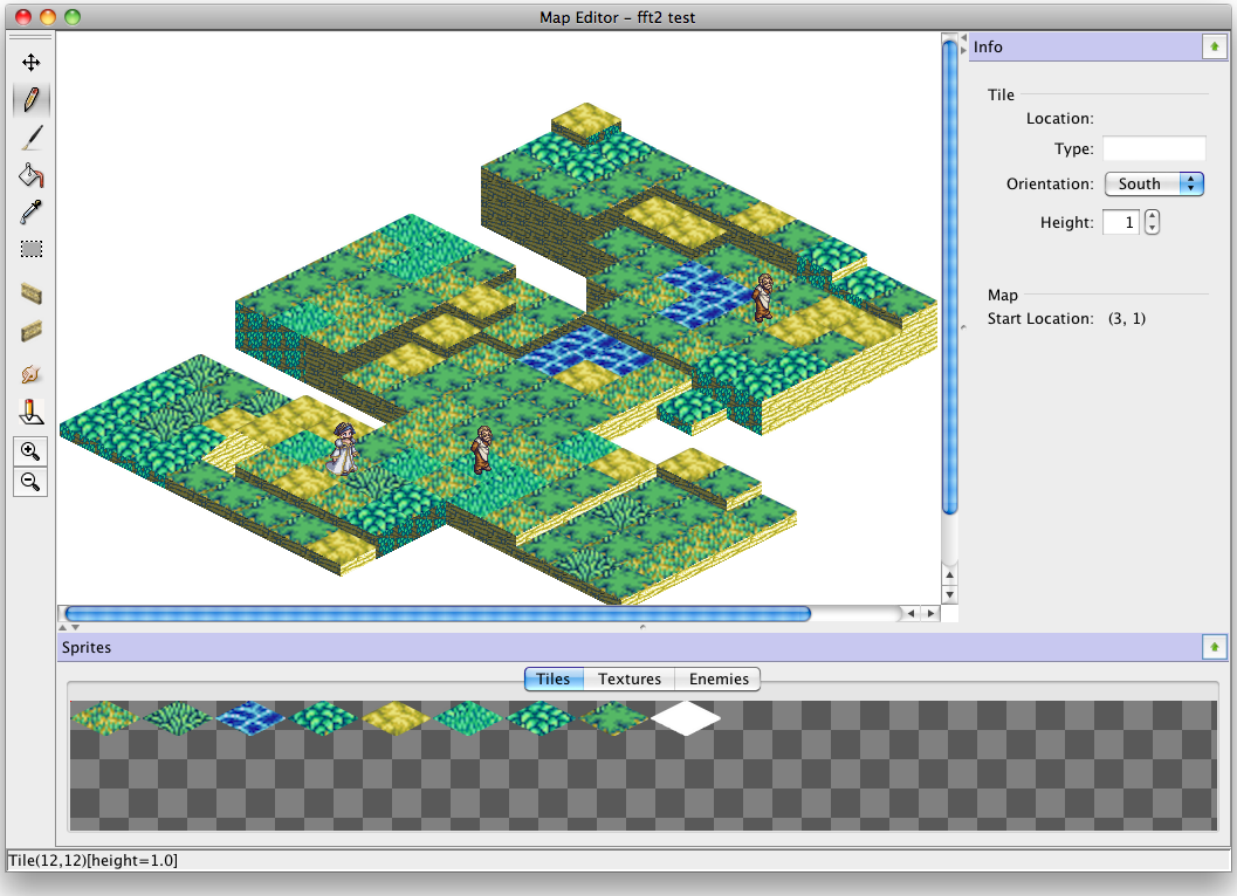


Figure 7: Visual Map editor

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 slanted tile are drawn.

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

7 Implementation

7.1 Overview

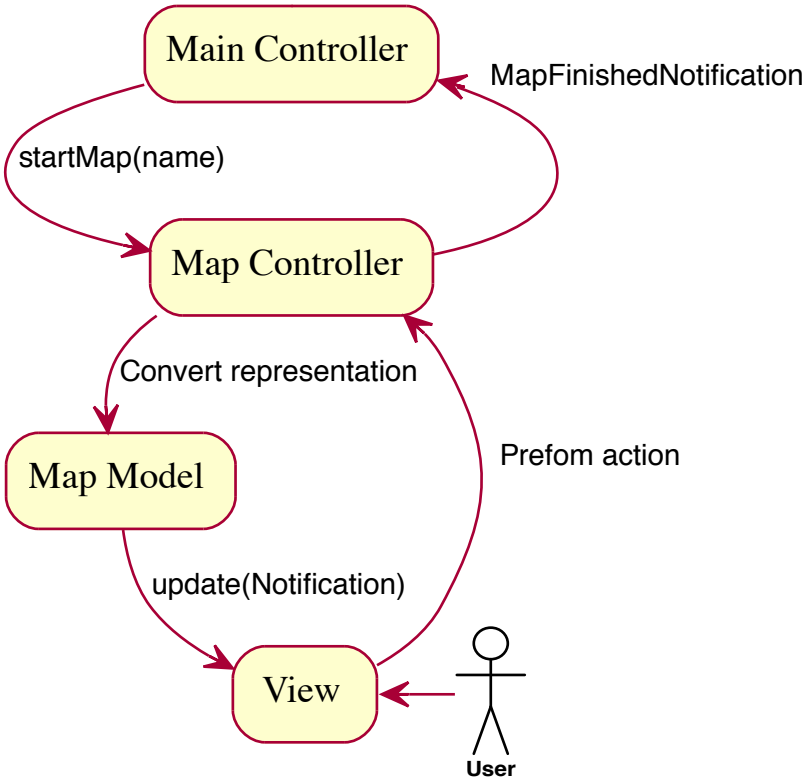


Figure 8: Overview of the implementation.

The system is structures 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 required 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 (e.g. when a map has been completed) the controller notifies the `MainController`, which decides what to do next.

The obsverable components (i.e the view, or the map controller) communicate using `notification` objects which encapsulates any relevant information. For example the `model` sends a `UnitMovedNotification` when the computer controlled opponent move one of their units. The notification includes a reference to the unit moved and the path it took to to get there. This information is used 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 7.3.6. To parse and serialisation 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 the it major benefits it 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 though the use of Java annotations[6], as shown in the below example[7].

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 need for serialisation. Another benefit of XStream is that it allows setting default values. This allows the user to omit redonent 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>
```

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

[7] Getter, Setters and trivial constructor omitted.

7.2.1 Resources

All resources that loaded are `Identifiable`, that they have a unique id. There are two main advantage to using this scheme. The first is that it allows caching of resources which means that there's 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 it meant 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 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 make it easier to cache images, since a *subimage* can be efficient created[8]. A `subimage` shares the same image data as original so is ideal way for each tile to have access to it's images[5].

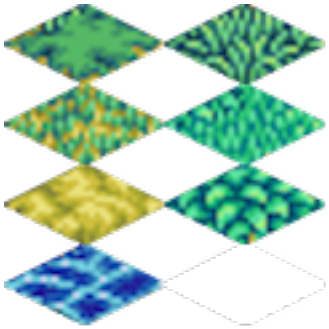


Figure 9: A 128×128 sprite sheet containing a tileset for a map

Sprite sheets make maps more reusable, since the tileset can be changed without any constiance[9].

I created a sprite sheet editor to allow the user to easily edit the tilesets. Using sprite sheets allows 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 interpedently.

[8] using the Java method `BufferedImage.getSubimage`

[9] As long as the new tileset has the same number of image or the `tilemapping` has entires for missing images.

7.3 Engine Development and Testing

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 unit.
- The `TileMapping` which specify what images to use when draw the images 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 dialog.

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

7.3.2 Units

- Has a set number of attributes inuldes weapons, images
- behaviour for the ai.

7.3.3 Conditions

- win conditions.

7.3.4 Dialog

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

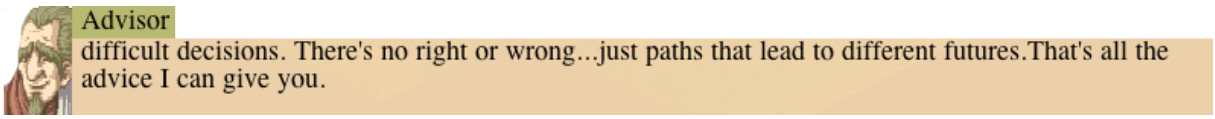


Figure 10: A example of a dialog

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 writes the plot of game rather then 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 intertavage.

7.3.5 Algorithms

- Unit movement
- path finding
- AI behaviour

7.3.6 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 senabient looking map. Users can use these as a starting point, to make it for them to design their maps.

7.4 Gui Development and Testing

7.4.1 Map Rendering

- Isometric maths
- how reusable it is
- efficient

7.5 User Interface

- unit animations
- menus

7.5.1 Custom Classes

7.6 Editor Development and Testing

7.6.1 Overview

7.6.2 Map Editor

7.6.3 Unit Editor

7.6.4 Event Editing

7.6.5 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 java[10].

[10] specifically Java 1.6+

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 [11], it also works well on Linux [12]. It even has limited compatibly with Windows[13] (apart from some minor graphics issues).

[11] Mac OS X 10.6 Snow leopard
[12] Science Linux x.y
[13] Tested on Windows 7 32 bit

8 Evaluation and Critical Appraisal

8.1 Results of User Testing

8.1.1 System usability scale

System usability scale (SUS) was used [6]. 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.

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

8.1.2 Results

In total six people predicated in the usability survey. The usability result was 61. This is very good result since a score of 50 mean that the system is useable.

8.1.3 Analysis

Since no score was below two the system had 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 skill at beginning, 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

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

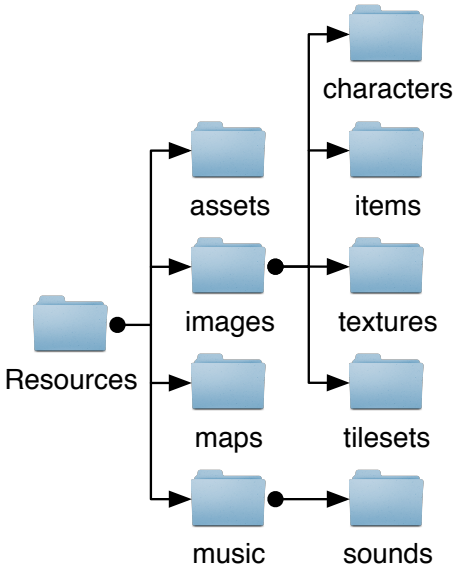


Figure 11: Project Structure

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

B.1 Assets

Assets are store in following format:

Listing 4: Assets format

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

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

Listing 5: 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 *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 images in the sheet as well as the dimensions of the sheet. `name-animations.xml` contains the unique id of the sprite sheet and can optional specify animations.

B.3 Maps

Each map needs the following *five* files:

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

name-conditions.xml which include 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 shon the start and end of battle

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

Maps also need a `tilemaping` which maps the tile's type to their images. A default `tilemaping` 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"[7]. Compared with other format such as MP3, Ogg Vorbis has no lieancely issues.

B.5 Sound Effects

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

B.6 Editor

For the editor the follow structure is used.

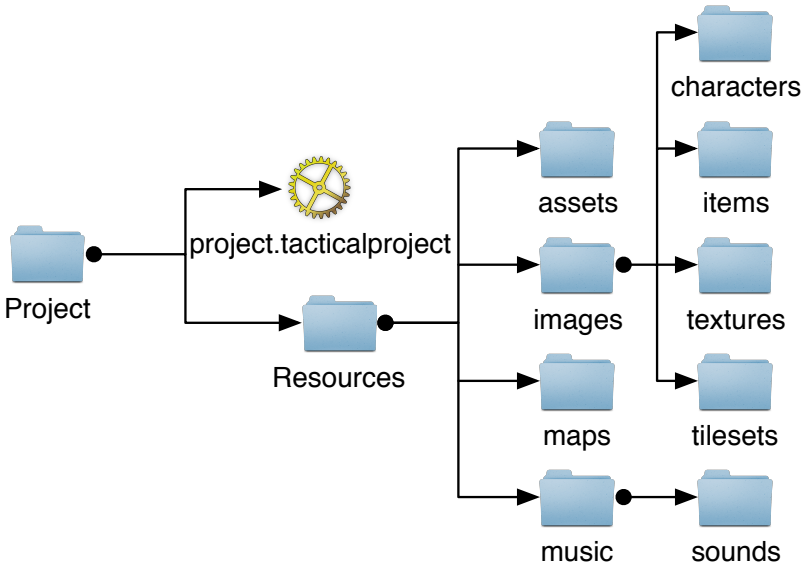


Figure 12: Editor Project Structure

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

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 13: 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

	← strongly disagree				agree completely →
1. I think that I would like to use this system frequently.	☐	☐	☐	☐	☐
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	☐	☐	☐	☐	☐

C.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 :

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 JRuby[14], 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' [15] 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

[14] A Ruby implementation written in java

[15] 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.

D.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%

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 6: 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
```

References

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

[2] E. Murphy-Hill, "Test-Driven Development." 12

[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. 12

[4] E. Pazera, *Isometric game programming with DirectX 7.0*, ser. Game Development Series. Prima Tech, 2001. 16

[5] Oracle, "BufferedImage Javadoc," http://docs.oracle.com/javase/6/docs/api/java/awt/image/BufferedImage.html, [Online; accessed 03-April-2012]. 24

[6] 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. 28

[7] Xiph.Org, "Ogg Vorbis," http://www.vorbis.com/, 2012, [Online; accessed 03-April-2012]. 33