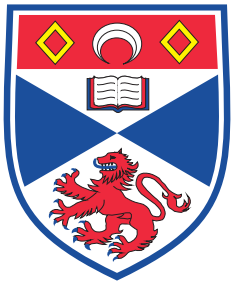
University of St Andrews

CS4099: Major Software Project



A Tactical RPG Engine

Bilal Syed Hussain

Supervisor:
Dr. Ian Miguel

Second Marker:
Dr. Alex Voss

February 24, 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

1 Introduction

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

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

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

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

such as 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 Baseline

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

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 Objectives

Key

✓ Finished
✗ Not Started
– • ⋆ In progress

3.1 Primary

The main goal of the primary objectives is 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 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.
  - ✗ The set of selected character attributes.

  and create a playable tactic 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` such as defeat all of the other players units.
    - ✓ Battles are `turn based` meaning only one unit performs at one time.
    - A combat system. **Nearly**
  - 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.
    - ✓ A isometric graphical representation of the game.
        - ✓ Which is show the grid with all the units.
        - ✓ Allow the user to move their units and see the opponents moves.
        - ✓ Allows the user to attack the opponents units.
        - ✓ Which allows the user to see a unit status (e.g current hit points).
        - ✓ Text will be to describe the more complex actions such magic.

3.2 Secondary

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

- ✓ Tiles have `height`, where units can only move to tiles of a smilier height.
- ✗ Tiles that are not passable such as sea, lava, etc.
- ✓ Tiles have different movement costs associated with them.
- ✓ A combat system that includes
    - ✓ `combat` between non-adjacent units,
    - Players have items such as weapons that affect the result of combat between units.
        - ✓ Including long distance weapons for the player and AI.
        - ✓ Including weapons that can attack multiple units at the same time.
- ✗ Direction and height of the character's tile affects attack.
- ✓ Sound effects.
- ✓ Music.
- ✗ Saving and loading games.
    - Allow the user to specify some of behaviour of non-player characters
        - ✗ Through the use of scripting.
        - ✗ An example: always attack a certain kind of unit or always attack the unit with the least Hit Points.
    - A graphical view to allow user specify the input to the engine.

3.3 Tertiary

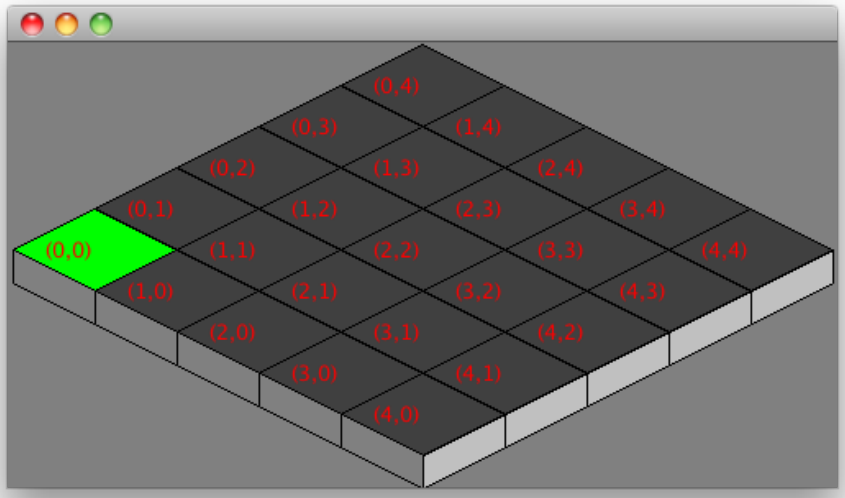
The goal of the Tertiary objectives are provide the user with more customisability and to provide a GUI for simple scripts.

- ✓ A combat system that includes
  - ✓ Support for `skills` which can effect multiple units.
- ✗ 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.
    - Showing some part of the story when a player's character reach a specified tile.
- A graphical editor for:
  - ✓ making custom maps,
  - ✓ making animations.
  - ✗ specifying the input to the engine.
  - ✗ making events
  - ✗ Scripts for simple event such as 'Defeat the leader' as a winning condition.
- ✓ Animations for units and movement.

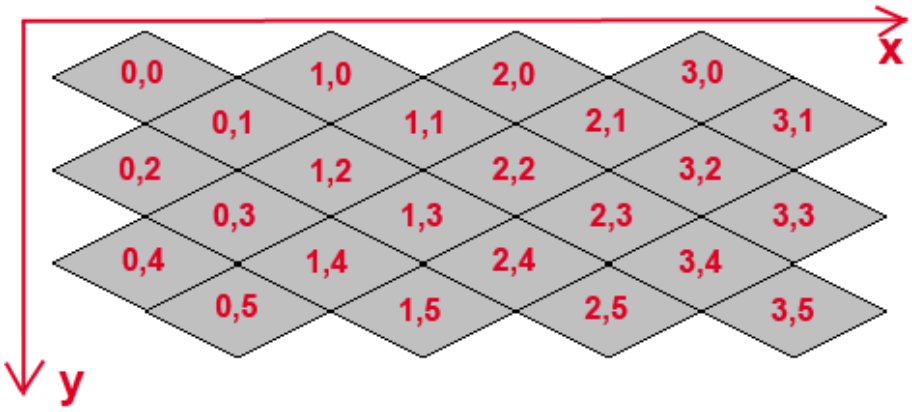
4 Design

4.1 Tilemap

There were two main choices for the isometric tilemap, a 'Diamond' map or a 'Staggered' map [2], examples of both are shown below.



(a) Diamond Map



(b) Staggered Map

Figure 2: The two main types of isometric tilemaps

The 'Staggered' Map has following advantages:

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

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

- 'Diamond' map look nicer then 'Staggered' maps because it has no ragged edges.
- Since that maps are large (at lest 15 × 15) the space wasted at the edges of the map does not matter as much.

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

Maths about isometric tilemaps?

5 Ethical Considerations

- Collection of data from questionnaire.
  - Just result of questionnaire, no personal data.
- Asking users to create a game.
- Asking users to play the created game.

6 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.

6.1 Language Choice

There were three main choices using Javascript, using JRuby[1], 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' [2] 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.

6.2 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:

[1] A Ruby implementation written in java

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

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

6.3 Action

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

6.4 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()

6.5 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.

6.6 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)`

6.7 AI Events

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

- Attack the player's unit with highest/lowest hp.
- Attack the player's leader unit.
- If player's leader's hp $< 20\%$ `heal(leader)`.
- Attack player's characters of class <class>.

The AI events `mapinfo` has addition methods including:

`attack` Attack the specified unit.

`follow` Move as close as possible to the specified unit.

`heal` Heal the specified unit.

`move` Move to the specified location.

`wait` Do nothing this turn

The `commands` themselves can be conditional, as example

Listing 1: Conditional AI Event

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

A Questionnaire

1. Have you played a Tactical RPG before?
   (a) If yes, did Engine have features you like to create in a game?
2. How easy to use was the Engine?
3. What particular aspects of the Engine did you like?
4. What particular aspect of the Engine did you dislike?
5. Any comments?

A.1 Using the engine to create a game.

	← strongly disagree			agree completely →
1. I think that I would like to use this system frequently.	☐	☐	☐	☐	☐
2. I think that I would help from an person experienced with the system in able to use the system.	☐	☐	☐	☐	☐
3. I thought that the system is easy to use.	☐	☐	☐	☐	☐
4. The game I created was fun.	☐	☐	☐	☐	☐
5. I think the system would be easy for most people to learn.	☐	☐	☐	☐	☐
6. I needed lots of extra knowledge to use the system.	☐	☐	☐	☐	☐

A.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 other comments:

_____

_____

References

[1] Quest, "Tactics Ogre: Let Us Cling Together," 1995. 3

[2] E. Pazera, *Isometric game programming with DirectX 7.0*, ser. Game Development Series. Prima Tech, 2001. 8