Game Maker

Author

Andrew L Ward
ID:785221

Degree

BSc Computer Science with Business Management

Supervisor

Professor Achim Jung

Institution

The University of Birmingham
School of Computer Science

Date

April 2010

Contents

Abstract

The purpose of the project was to create a two-dimensional role playing game maker enabling a user to quickly and easily create, save, distribute and play created games. Emphasis was placed on being able to do this using a small set of powerful tools.

A game is able to be made through the creation and customisation of maps and events that the in-game character (the sprite) can interact with. Once a game has been made, it can also be played using the software. The created game is able to be saved as a single .game file that can be reloaded back into the project for playing or further editing.

Similar already available tools were analysed to identify their key features and to ensure that the game maker had a unique selling point.

The project is ambitious and management of my own time to complete the project specification to deadline was an important focus. The development methodology chosen was used to the reduce risk should the project not be fully completed on time and to be adaptive to unforeseen changes.

Acknowledgements

Professor Achim Jung for supervising the project.

Squaresoft for producing the Final Fantasy 7 game that helped to inspire the project.

Flamingo Ribbon and Utils4J library developers.

PNG, ZIP, XML file formats used.

Developers of the RPG Maker for giving inspiration and ideas.

My friends and family for listening to demonstrations of the project and for testing.

My dad for taking hours out of his schedule to read through the final document and highlight areas of confusion and grammatical errors.

Glossary of Terms

Term	Description
Vector Images	Images represented as a sequence of points in a coordinate system. The shape of the line connecting each point can be defined using mathematical functions.
Game Maker	The name of the software project described in this documentation.
RPG	Role Playing Game. A game where the user experiences events from the perspective of the in game character they are playing.
Sprite	An in game character. This character can often be controlled by the person playing the game.
Re-scaled	A specified bounds of the game map is specified to be viewable. If the area of the bounds is smaller than the viewport size then this area of the image is stretched to fill the viewport.
Re-cropped	The aspect ratio and image scale is kept consistent. The map will display in the viewport as-is on a pixel by pixel basis. No resizing of the image takes place before displaying it onto the screen.
Viewport	A window or screen area that views an interface or graphics.
Servlet	"Server-side programs that give Java technology-enabled servers additional features. Servlets provide web developers with a simple, consistent mechanism for extending the features of a web server and for gaining access to existing business systems." (Sun Microsystems 2001)
Relative Referencing	Referencing the location of a file or folder in reference to the current location of another file/folder/the user. e.g. "..\files\gamesaves\".
Absolute Referencing	Referencing the location of a file or folder as its exact file path. e.g. "C:\Users\Andy\Desktop\Game Maker 1.8 17-03-2010\files\gamesaves\".
Key Listener	A Java interface that allows functions to be executed when a keyboard key is pressed or released. Every time a key is pressed or released the corresponding key pressed and key released functions are executed along with a reference to what key was pressed or released.
Zip Archive	A file format used to contain a compressed state of a directory of files and folders.
JDK	Java Development Kit.
JRE	Java Runtime Environment.
Bitmap Image	The representation of an image as a sequence of bits. The bits represent the colour of every pixel of the image with no compression.
HCI	Human Computer Interaction.
Retro	A term used to describe outdated or old design trends.
SNES	Stands for "Super Nintendo Entertainment System" this is an old games console.
Sega Megadrive	This is an old games console
Anthropometric	The study of the shape of the human body. This can be used to make design decisions about how humans interact with objects.
Layered Pane	A Java class that allows java panels or components to be arranged in layers on top of one and other.
JPanel	A JPanel is a java container for various components such as buttons or labels. The panels background can also be drawn on.
API	Stands for "application programming interface". In the context of this project it is a set of pre defined classes and functions that can be used when programming.

Introduction

The purpose of this project was to design, plan and implement a two-dimensional role playing game maker. A role playing game being a game where the player experiences events from the perspective of the in game character they are playing.

My motivation for the project was to allow the user of the software to easily create a game without the need to have advanced knowledge on how to program. I wanted it to be easy for the user to save, share and play their created games on other computers running the game maker software. From the perspective of the user, the game maker and game player will appear within separate windows, so the game creation tools will not be available when playing a game. The game is then played using keyboard controls.

Background research was undertaken to look into the role playing game makers that already exist as I did not want to just recreate their functionality. The existing tools are large and established programs. It is unrealistic to try and compete with these tools in terms of quantity of functions, therefore it was important that the software had its own unique selling point. The strengths and weaknesses of the existing tools needed to be examined to evaluate what could be improved and what design decisions should be carried forwards into this project.

There are several key design decisions that make this project unique amongst it's alternatives:

- There is the ability to import images to use as the entire background of a map.
- The sprite is able to scale in relation to the perspective of the map.
- The location and movement of the sprite is handled on a pixel by pixel coordinates based system as opposed to the grid based system used in all of the alternatives.

This drastically changes how interaction with the map and events within it occur.

The user is able to place and define customised events that can then be triggered when the game is being played. It is the triggering of events and the transition between maps that gives the user immersive interaction with the game.

Some sections of the report will require that the reader has a low level understanding of basic programming concepts. For example, in some sections pseudo code has been used to illustrate how algorithms have been implemented.

This report will first discuss the background research into the alternative tools available. After this, the system requirements will be defined followed by the design plans for significant parts of the software. The key implementation decisions will then be documented and evaluated. The next section will describe how I was able to make the software stable and robust through testing and good programming practice. Details of how the project was managed and what software development processes were undertaken will then be outlined. Finally I will evaluate and conclude the outcomes of the project.

Background

This section considers what software already exists to create custom RPG games on the PC platform. It will evaluate the benefits and limitations of the alternatives in order to identify the features that should/shouldn't be included in the Game Maker software project.

RPG makers already exist for the PC platform. Here is a list of the main tools available.

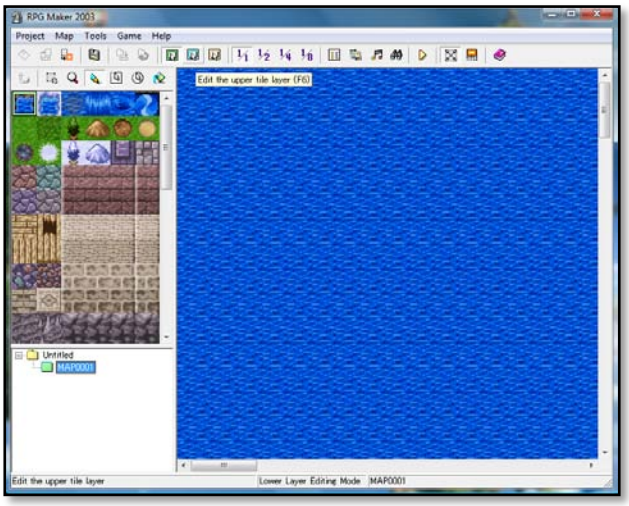
- RPG Maker 95
- RRG Maker 2K
- RPG Maker 2003
- RPG Maker XP
- RPG Maker VX
- RPG Toolkit

The "RPG Maker" series are new versions of each other up to RPG Maker 2003. From RPG Maker 2003 spawned 2 competing RPG Makers: VX and XP and both of these makers are commercial. The interface and usability of XP, VX and 2003 are so similar that they are practically the same and the differences are so subtle that for the purpose of this document I will be treating them as the same thing.

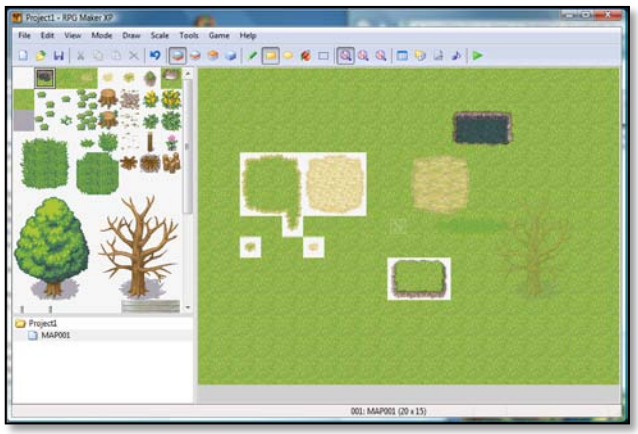
Being the only PC alternative to RPG Maker, it is worth mentioning the RPG Toolkit. It does not significantly compete with RPG Maker as my experience using both tools found that the majority of features within the RPG toolkit exist within RPG Maker.

To illustrate the similarities of the RPG maker series and highlight how the interface is structured, here are some screen shots of the RPG Maker programs:

RPG Maker 2003

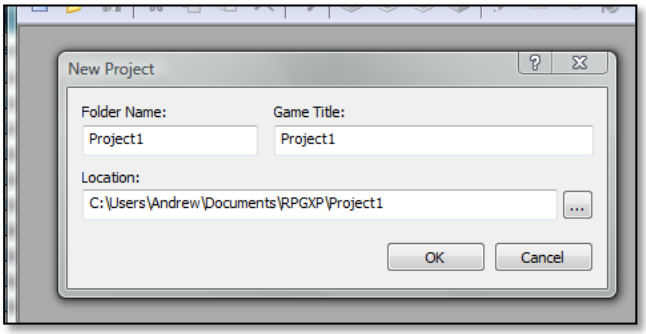


RPG Maker XP



RPG Maker VX

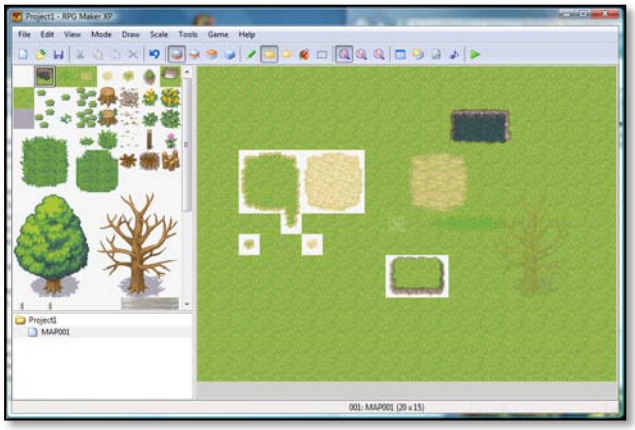
Upon first loading RPG Maker XP an input box is displayed that requests the user to specify a folder that will be used to store the created game files:

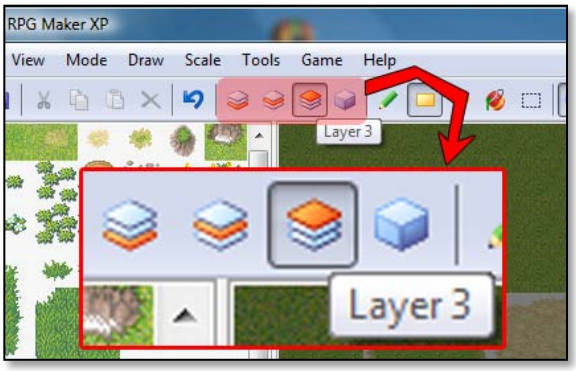


Storing the game creation files as a folder structure is a trend more common with advanced development tools used in programming. For example the net beans java development environment (Sun Microsystems 2010) uses this approach to give the user full control over how their source code is structured. It is much more common for applications to save the current state of what is being edited as a single file. Applications such as the Microsoft Office suite (Microsoft n.d.) save data in this way as the user base their software is not expected to understand the internal working of the saved files.

In my implementation of a development tool I will focus on simplicity and ease of use. It would not make sense to expect the user of the game maker software to understand the internal workings of a saved game file/folder. Making a system that saves the game state as a single file would simplify the process of saving a game and give the project a unique selling point over the RPG Maker program.

After specifying a project name in the RPG Maker, the interface for creating a new map is shown:





The map creation element of this tool is one of its main selling points. It is very easy to use the tools to create a scene that looks and feels a lot like the retro RPG games. There is the ability to make your own map tiles too by importing them from a single image (Known as a tile set):

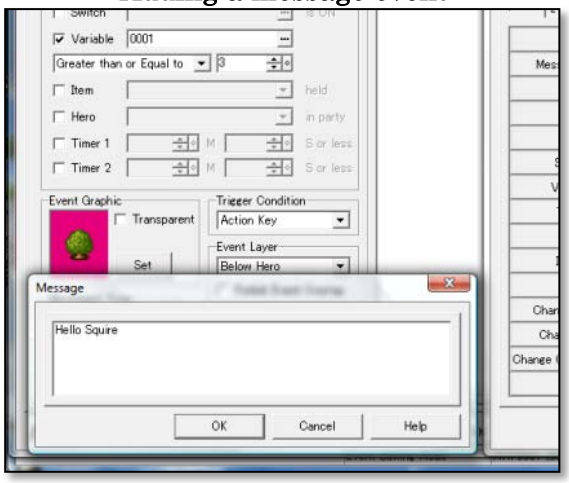
The single image is split into squares that can then be painted onto the map background to create a a scene. Because of the ability to use tile sets to create maps the RPG Maker community have made many that can be downloaded and used. Often they are designed so that a user can create a game to look exactly like the retro RPG games on the SNES and SEGA mega drive games consoles.

Each tile set is of a limited size and a drawback of this is that an image cannot be imported and used to occupy the background as a whole. There is also no facility for a map have a perspective as tiles painted to the back of a map are the same size as those painted at the front.

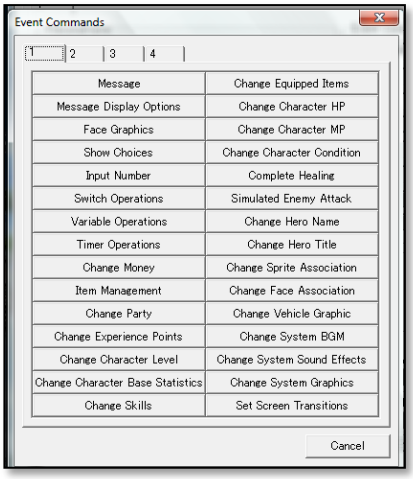
In order to give my project a unique selling point over the grid-based method of map creation I plan to implement an entirely new map creation system that allows images to be imported to use as map backgrounds. This will allow the user to create much richer maps in external applications such as 3D studio Max, Photoshop, Microsoft Paint or through photographs. In doing this I will have to revise how the in-game character (the Sprite) moves and interacts with the map and events on it.

The RPG Maker tool provides a vast array of tools that can be used create a customised game. This makes the tool powerful but difficult to learn. Here is an example of some of the tools that can be used to customise a game using this software:

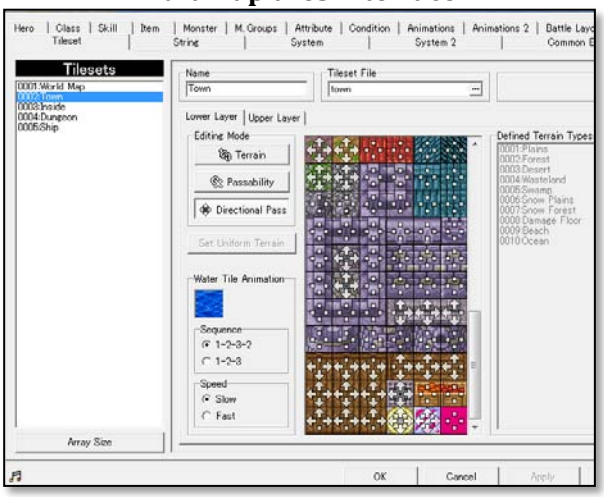
Adding a message event:



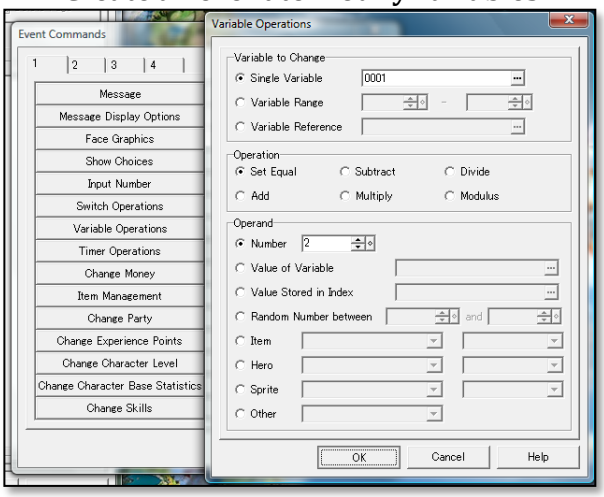
Game event creation interface:



Edit map tiles interface:



Create an event to modify variables:

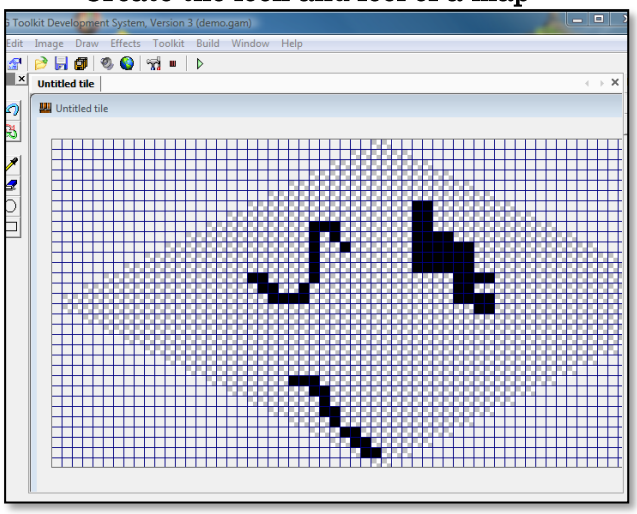


The customisation options are so vast that both learning to use them and actually using them to make a game is very time consuming. I will not be able to compete with the level of customisability that this tool offers. Having less features may seem like a weakness of my software, however having less functionality

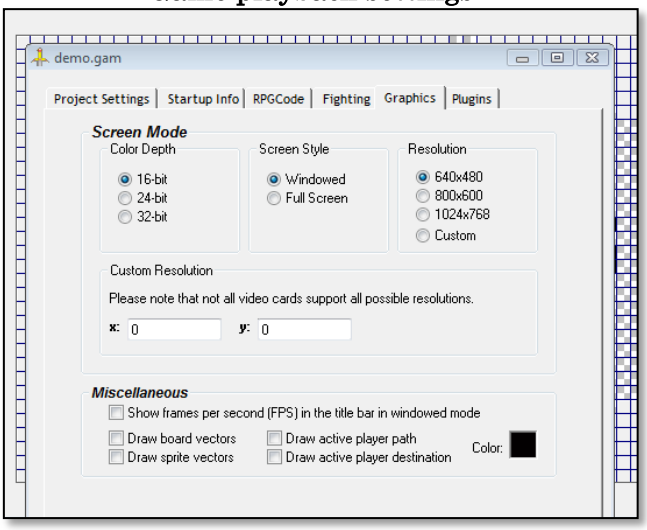
to learn and modify is also a positive outcome. Users will be able to learn the programs features easily create games more quickly.

Let's move on to consider the RPG Toolkit. Notable key features of the toolkit that differ from the RPG make series are its abilities to specify the game play resolution, and the ability to edit the maps in an isometric view:

Create the look and feel of a map:



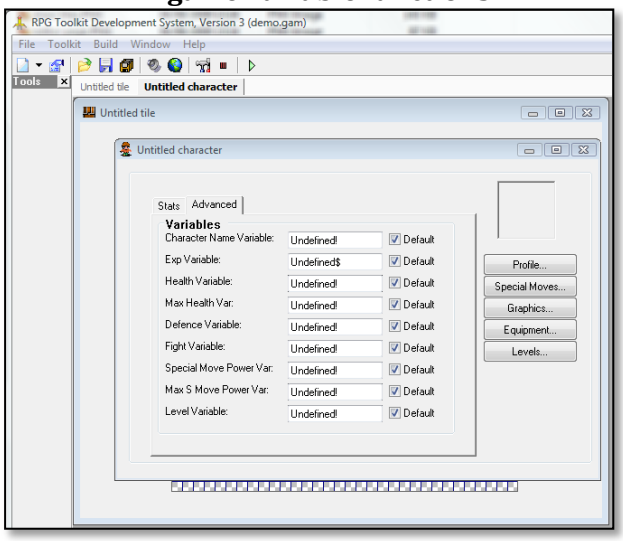
Game playback settings:



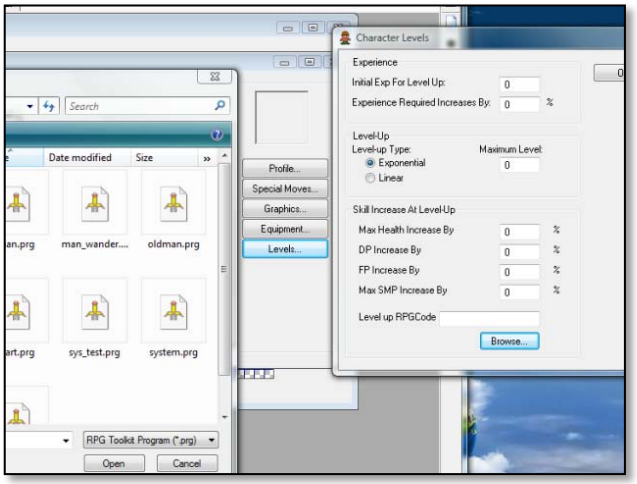
As it has been chosen not to implement a grid-based system of representing maps in the Game Maker, the benefits of creating a game in an isometric grid-view cannot be replicated within it. The ability to specify different playback resolutions gives the RPG Toolkit an advantage over the RPG Maker. I will need to consider the best way of handling the playback resolution in the design section of this report.

Overall the RPG Toolkit is primitive, unintuitive and confusing to use. Its functionality and customisability is also very limited. Here is an example of its confusing options and interface:

In-game variable functions



Define character attributes:



Learning how to create a game using the RPG toolkit is not straightforward, mainly due to the interface not being intuitive to learn to use. It is not clear what should be put into the variable fields, for example there is no indication of what the "RPGCode" field requires.

These criticisms of the RPG toolkit support the assertion that the Game Maker project must be easy to use. A user should be able to develop a game without the need to consult tutorials or help documentation. To do this effectively I will consider aspects of well known HCI (Human Computer Interaction) research, most notably the concept of improving interface learning though consistency of interface creation (See HCI heuristics Appendix). Interfaces must be created to operate in a way that meets the users expectations: this can be done by mimicking the interface to be very similar to tools that the user is likely to be familiar with. This philosophy is supported in Nielsen design heuristics "Use both knowledge in the world and knowledge in the head" (Nielsen 2005).

Analysis

I came to the conclusion to make a game development program whilst also exploring what tools currently exist to create basic RPG Games. As you can see from the background section, there are very few tools available that are aimed at a user with little technical expertise. From this I identified that there was a need for an RPG game maker to compete with RPG Maker 2003 and RPG Maker VX.

As it would be unfeasible to compete with RPG Maker in terms of functionality, ease of use can be the area of competitive advantage with the proposed Game Maker program. My analysis found that the process of creating a game using RPG maker was not straight forward and the process could be improved to be much simpler. There are lots of menu items, sub menu items and tabs which make the HCI element of the maker confusing and could make it difficult for a novice user to simply pick up the tool and use it without reading the help documentation.

The key way that 'RPG maker' works is to use a grid-based system to create, edit and view the map, characters and events. With my project I have the ability to change this convention in order provide a solution that creates the game in a completely new way. To have a game where the maps are based on images as opposed to square tiles painted on a grid. This method would allow a user to take photos or draw pictures that they could then use as maps in the game. A coordinate based system of representing the sprites position on the map could be used in conjunction with this to aid in the replacement of the grid-based system.

Specification

I have split the specification into two subcategories: the core features and the additional functionality. This is because the main aim of the project is to implement the core first and the additional features aren't absolutely necessary to have a completed project. I aim to implement the additional features once the core is complete.

Core Features

1. User can create a map from an image.
2. User can define what areas of the map can be walked on.
3. User can select a sprite that will walk around the map.
4. User can define the depth of the map:
   4.1. The sprite must change size depending on where they are positioned on the map to give the illusion of walking into the distance.
5. The user must be able to define which areas of the map that the sprite can walk underneath.
6. Images must be able to be placed on top of the map.

7. User can create events that happen after certain triggers:
   7.1. Teleport to a coordinate on any map.
   7.2. Display a message.
8. Create actions that trigger events:
   8.1. Sprite walks onto a certain coordinate.
   8.2. Button is pressed whilst the sprite is within a certain coordinate.
   8.3. Some events must have a boundary.
9. Development tools are displayed on a ribbon and/or a side pane.
10. The game development must be able to save and load their game development progress.
11. The game should be able to be saved in a way that it can then be played via the tool.
12. When the game is played by a player, they should not be able to modify anything unless it is via events. The experience of playing the game should be abstract to the experience of making it.

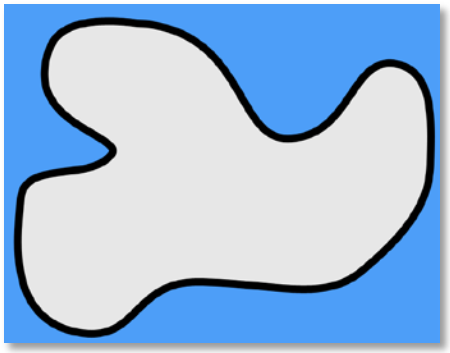
Additional Features

1. User can add music to a level from a repository of midi files or from an mp3 file.
2. Music must play in the background.
3. Items can exist within the game:
   3.1. User can create a list of items that can be obtained by the character.
   3.2. The character could pick up the items from a chest.
   3.3. Or buy them from a shop using in-game money.
   3.4. Events set to be triggered only if an item has/has not been collected.
4. Event triggers could be expanded to give new ways in which an event is triggered:
   4.1. Choose yes or no when having a dialogue with an in game character (NPC).
   4.2. Collecting an item.
   4.3. Using an item.
   4.4. Sprite interacts with an object(s) a set amount of times.
5. Events could be expanded to give the user more options when creating a game:
   5.1. Change the map background music.
   5.2. Change the map image.
   5.3. Have a dialogue with an NPC where you chose what to say from a list of options.
   5.4. Change the walk-able area on a map.
   5.5. Add, remove or change an image overlaying part of the map.
6. The Model – Control – view framework could be expanded to work within a servlet. This would allow the game maker to be used via the internet:
   6.1. A repository of games could be set up on a server so that friends can share games between one and other.
7. Game maker and game player tools could be separate. This would allow you to be able to play the game without needing to have the tools to make one.
8. Functionality could be added that allows a character to have battles in a similar way to the battles on the old final fantasy games:
   8.1. Experience could be gained from battles which lead to a character gaining levels and becoming more powerful.
   8.2. Monsters/enemies would require to have stats as well as the character that is fighting them
   8.3. Items could be purchased that increase the characters stats.
   8.4. Items could be equipped to increase the characters stats.

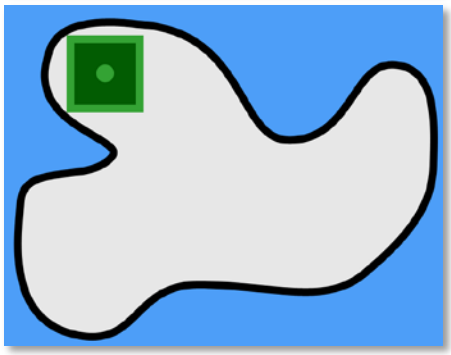
Design

Collision Detection

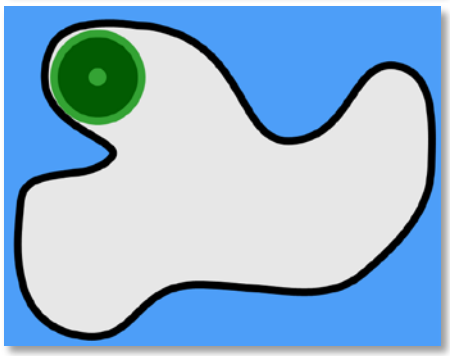
The sprite must be able to walk around the map when the game is being played. When making a game the user must be able to specify what areas of the map can be walked on. A mechanism must be implemented that prevents the sprite from walking onto an area that they are not allowed to.



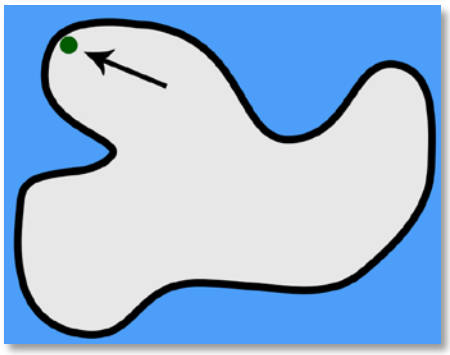
Consider:
To visualise how this would work, consider that the picture (To the left) represents an area on a map that can be walked within. The darker blue area cannot be walked within and the light grey area can.



Rectangular collision detection
The solution is fairly simple as it only requires a comparison between the corners of the square and the non-walk-able area.

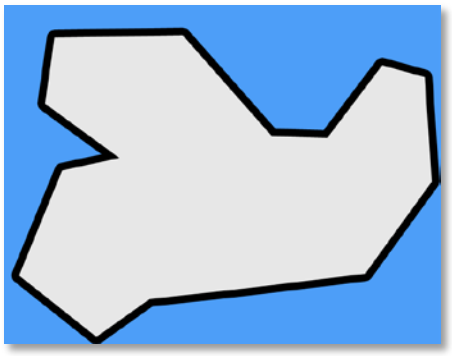


Circular collision detection
This method uses a circle as the area to use to detect collisions. The algorithm checks that the distance between the centre of the circle and the nearest non walk-able boundary is less than the circles radius.



Single point collision detection
This method is the most simple. It checks that a single point is / is not within the non-walk-able are.

Before it can be properly decided which collision detection method is best, it is also necessary to considered how to store the non walk-able area of the map. Here is a brief description of the main methods available that are worth consideration:

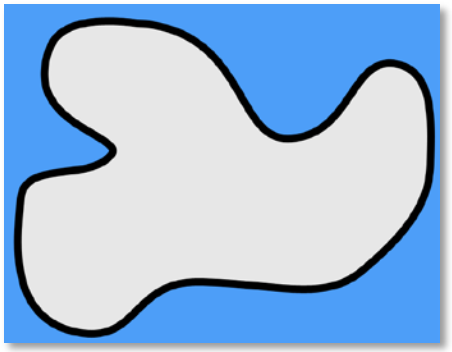


Straight-line connected points

Fixed points are stored in sequence. The area is considered to be the lines drawn between these points in sequence with the start and end points joined up.

This method is lightweight to store. It is easy to calculate if another shape is within the area by comparing the map coordinates against the shape coordinates. It would also be easy for a user to draw the walk-able area by pointing and clicking on the screen with the mouse.

The disadvantage with this method is that the shape produced doesn't look as aesthetically pleasing as the calculus-curve method.

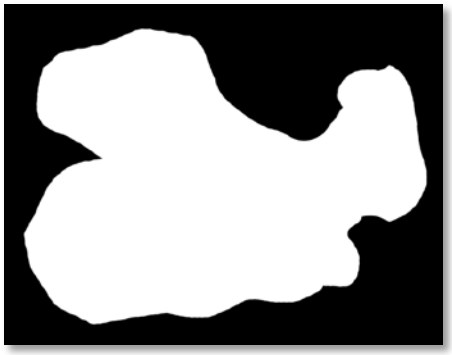


Calculus-curve connected points

This method also stores points in a logical sequence. Additional information is stored with each point indicating the curvature of the line to the next point. Calculus is used to represent this curve.

This method creates aesthetically pleasing shapes. Also like the straight-line method it is easy to store, even considering the calculus information.

The issue with this method is that it is more difficult to perform collision detection, more processing is required which could affect playing performance. Also it would be more difficult for a user to define the area as they would not only have to plot the points, but they would also have to define the curvature of each line.



Pixel-by-pixel

The area is stored as a black and white mask of pixels. White pixels can be walked on, dark ones can't.

This method would be easy for the user to draw as they could just paint on where the non walk-able area is, and erase it too where necessary.

The disadvantage with this method is that it requires much more information to store the area. Each pixel in the entire area would need to be defined. Also collision detection would be a more complicated process as each pixel in the collision-area would need to be checked individually.

With simplicity and optimisation in mind the 'single point collision detection' method will be used to represent the sprite and the 'straight line connected points' method to represent the walk-able area on the map. This method will allow for the most efficient algorithm to be used to ensure the sprite does not walk within the non-walk-able areas of the map. The sprite's coordinate position can be compared against the points that make up the walk-able area. If a coordinate is within the areas points then the

sprite will be permitted to navigate to that position. Efficiency is important because it is likely that this algorithm will be executed many times a second as the sprite moves.

As the outline of the walk-able area is not visible when the game is being played, there is no need to have it looking aesthetically pleasing, it just needs to be functional and fast. For that reason I exclude the calculus and pixel based methods of storing the walk-able area.

Storing the Game

There is a requirement to save a created game and it is important that the most appropriate method is chosen. The main options for storing the data that will be considered are:

- SQL Database. (NTC Hosting n.d.)
  - Advantages:
    - Efficient to retrieve and store data using SQL statements.
    - Well documented and understood.
    - Good design would be easy to maintain.
    - A normalised approach would allow for easy addition of new data.
  - Disadvantages:
    - Fairly difficult to transport data between databases by the end user (The Game Player).
    - Machine developing or playing the game would need to have a SQL database installed.
    - Data has to be loaded into objects and methods must be written to handle the loading and saving of each piece of data.
- CSV text files. Text files containing values separated by commas. (Wikipedia 2010)
  - Advantages:
    - Simple to read and write simple sequential data.
    - Lightweight files.
    - Quick and dirty to implement.
  - Disadvantages:
    - Not suitable for storing relationships between data
    - Structure can lack definition and be hard to modify once a standard has been set without breaking things.
    - Data has to be loaded into objects and methods must be written to handle the loading and saving of each piece of data.
- Binary Object Dump. Output the state of the objects in memory as binary stored in a file. (Greanier 2000)
  - Advantages:
    - Extremely simple to store. Just set objects as serializable and save them to a file.
    - Efficient to load data as it is already in the form of objects, the objects just need to be de-serialized and stored back into memory.
  - Disadvantages:
    - It is difficult to ensure backwards compatibility between program versions. Changes to code could mean that objects cannot be loaded correctly.
    - Difficult to interpret outside of the program that created the object dump. A lot of information about the objects must be known to load the data.
- XML. This is a mark-up language that stores data within XML tags. (W3C 2010)

        - Advantages:
            - Widely supported and acknowledged
            - Very portable. The XML files can easily be transferred and read.
            - Easy to add new data without changing the way older program versions read the XML. Can ensure backwards/forwards compatibility of games created.
        - Disadvantages:
            - Data has to be loaded into objects and methods must be written to handle the loading and saving of each piece of data.
            - Not optimised to search through large amounts of data (like SQL is)
    - Folder Structure.
        - Advantages:
            - Widely acknowledged and understood.
            - Great for storing multiple file types such as csv, XML, jpeg, png etc.
            - Easy to add new levels to the folder structure.
        - Disadvantages:
            - Has to be used in conjunction with other files to actually store data
            - Not as fast to search for information as with SQL
    - Zipped Folder Structure. (Wikipedia 2010)
        - Advantages:
            - Multiple files can easily be stored within a single zip archive.
            - Could be used to easily transport an entire game within a single file.
        - Disadvantages:
            - Reading compressed data is less efficient to reading uncompressed data.
            - Not as fast to search for information as with SQL.
    - PNG image container. PNG is a compression method for storing images. (Roelofs 2008)
        - Advantages:
            - Optimised for storing 2D graphics information of a uniform nature. It is good at storing vector images and images that contain block colours.
        - Disadvantages:
            - Only suitable for images.
            - Images with a lot of non-uniform information will have a large file size
    - JPEG image container. JPEG is a compression method for storing images. (W3C 1996)
        - Advantages:
            - Optimised for storing 2D photographs or images that are of non-uniform in nature. For example photographs.
            - Small file size of images with lots of non-uniform data.
            - Supports transparency with the addition of an alpha channel.
        - Disadvantages:
            - It is not lossless compression so image information is lost when stored as this format.
    - AVI, MPEG4, MPEG video container. These are compression methods and containers for videos, some are optimised for storing different types of video data.
        - Advantages:
            - Highly optimised for storing video information.
        - Disadvantages:
            - Only suitable for storing video information.

There are a lot of different options available so the requirements of my game storage mechanism must be considered when making the decision on which to use:

- Needs to be portable (games can be shared).
- Needs to be backwards compatible.
- Needs to be diverse and support the storage of both files and data.
- Needs to be able to be used to save and load the created game for editing.
- Needs to be able to be used to save and load the created game for playing.

I have chosen to apply zip compression to the folders that contain the XML file and the images so that they are contained within a single file. This is important as a single file is easier for the end user to manage and share, they do not need to know anything about the contents of the file.

I have chosen to use XML to store the games object variables and references. XML is parsed, therefore backwards compatibility of older game saves can be handled easily within the program at the parsing stage. New tags and attributes can be added that do not disrupt the data structure making XML a much better option than using CSV files. The representation of information within XML tags is a logical and widely adopted methodology and this makes it much easier to read and understand as a programmer (Horstmann 2008). Its portability also makes it a much more suitable solution than SQL, it is easy to send an XML document without doing any additional pre-processing.

Even though the user is likely to use photographs to store images, and JPEG is theoretically the best compression method for this format I have chosen to use PNG. This is because a significant amount of information is lost when a jpeg is resaved many times (See JPEG recompression. Appendix). I do not want the amount of times a user saves their made game to effect the quality of their background images. PNG is more suitable because images stored in this format lose no information. It is also beneficial because it supports the ability to add items to the foreground that have transparency.

After evaluating all of the alternatives I plan to use a zip file structure to store the games files. Within the zip will be a folder structure to keep the file organised. The zip file will contain XML documents to represent data and references, and will contain PNG images that represent the visual look and feel part of the map.

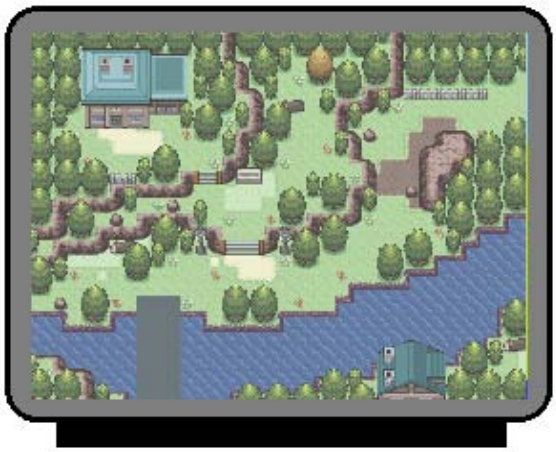
Game Resolution

Another key decision to make is how to handle the viewport of the game when the game is in play mode. The following options are worth consideration, with combinations of methods being a possibility:

Windowed Mode – Map Re-Cropped

Windowed Mode – Map Re-Scaled

Full Screen Mode – Map Re-Cropped



Full Screen Mode – Map Re-Scaled

Re-cropped: The aspect ratio and image scale is kept consistent. The map will display in the viewport at its original scale. No resizing of the image takes place before displaying it onto the screen.

Re-scaled: The viewport is set to be a fixed size (for example 800x600 pixels). If the window/screen is larger than the viewport then the area that is being displayed in the viewport is stretched to fill the window/screen.

The re-cropped resolution options would always keep the same image aspect ratio and scale. If the viewport size is increased you simply see more of the map to fill this increased area.

- Advantages:
  - Map components that have an area (such as map image, walk-able area, sprite, event boundaries) do not need to be rescaled depending on viewport size.
  - Simple to implement.
  - Background image looks smoother and shaper as it is not being distorted in any way.
- Disadvantages:
  - Maps smaller than a large viewport will have to be centred on the screen with a large empty area around them.
  - The user has to import big images to cater for larger screen sizes.
  - Smaller maps will always fit within the viewport. This could make the game less immersive as the character would not expose new areas of the map as they move around.

The Re-Scaled option always displays the same amount and area of the map. If the viewport is expanded, the viewed area of the map is increased to fill the new viewport size.

- Advantages:
  - The same sized area of the map always fills the viewport regardless of image size. The person making the game can make design decisions based on this.
- Disadvantages:
  - Events boundaries walk-able area, sprite boundaries etc have to be scaled depending on the viewport size.
  - More complicated to implement.
  - Distorts the background image.

Despite having a large list of advantages and disadvantages favouring the re-cropped option of displaying the map, the re-scaled option is what will be used. This is so that the person making the game

does not have to worry about different users having different experiences based on their map and viewport size.

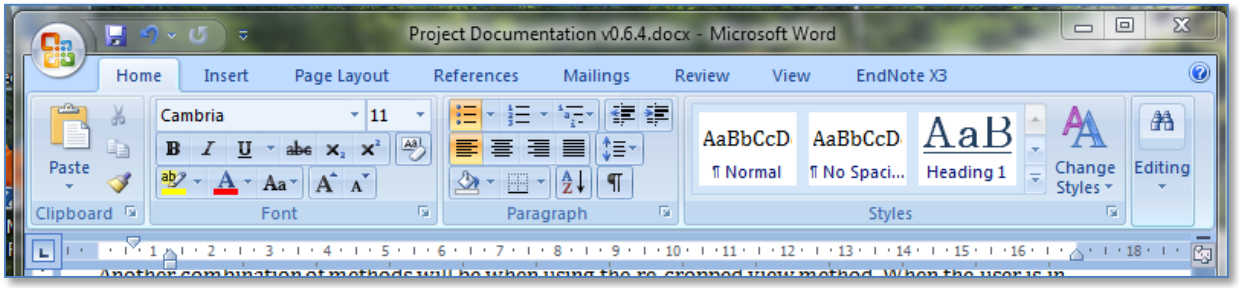
So that the benefits of the re-cropped mechanism aren't lost, the user playing the game will have the option to play in full screen or in windowed mode. Full screen mode will be rescaled, so the viewport will be set to 1024x768 and will fill the screen. In windowed mode the map will retain its native scale, so if the person playing the game wants to increase the viewable area of the map then they can do so by increasing the size of the window.

1024x768 full screen resolution was chosen because the W3C statistics show that 93% of web users are using computers with monitors of this resolution or higher (see W3C Screen Resolution Statistics (w3schools 2009)(w3schools 2009)appendix). If I increased the resolution to be higher than this then a maximum of 50% of users would have screens capable of viewing at that resolution.

User Interface

A ribbon and a toolbar are common methods of structuring a user interface. The advantages and disadvantages of both methods will be considered in this section.

Example of the ribbon interface within Microsoft Word 2007:



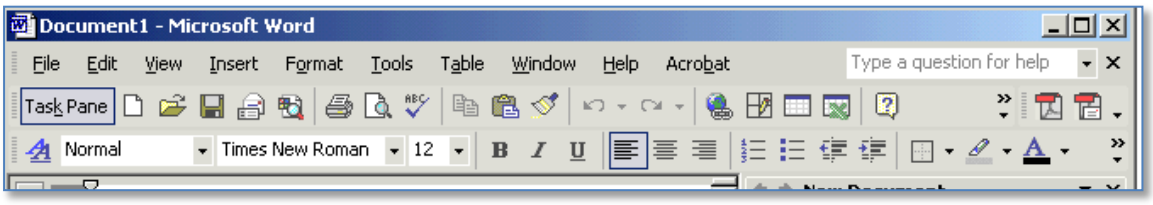
Key benefits of the Ribbon:

- Easy to find command buttons.
- Commands put into context headings. This makes its new commands easy to understand.
- Ribbon tools currently in view can easily be changed by clicking on tabs. This means the current most relevant tools are quickly available.
- Modal view of the ribbon means that commands aren't lost under many levels of tool-bar hierarchy
- It looks nice which makes the program inviting to use.

Key limitations of the Ribbon:

- It consumes more vertical pixels that it's toolbar equivalent.
- Only beneficial if the extra space that it uses increases usability.
- Command icons do not require labels, or labels may be hidden if the ribbon is resized. This means that icons must be chosen that are self explanatory.

The alternative to using the ribbon is to use toolbars. Example toolbar from Microsoft Word 2000:



Key benefits:

- If a small number of tools are used then the toolbars do not take up much vertical space.
- The file menu bar allows for a hierarchical structure of commands to be defined.

Key Limitations:

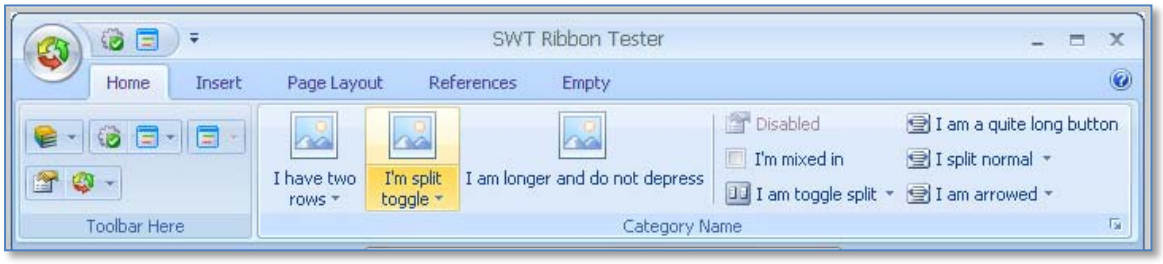
- Toolbars are static so if there are lots of tools that the user needs to use then the toolbar quickly grows in height.
- Height of icons is constrained to be the same height as a toolbar row.
- Icons can only be groups within a single row.

After analysing the benefits of the ribbon interface outlined by Microsoft (Microsoft n.d.) it was concluded that a ribbon interface is a better solution than the toolbar interface. The largest factor influencing this decision is that the ribbon allows for the quick movement between contextual working tools. The ribbon will be used to display tools available to the user to use when making a game.

Here are screen shots generated from two open source ribbon API's. These were chosen for analysis because they are the two most expansive java ribbon API's available in terms of functionality:

Flamingo Ribbon (java.net 2008)

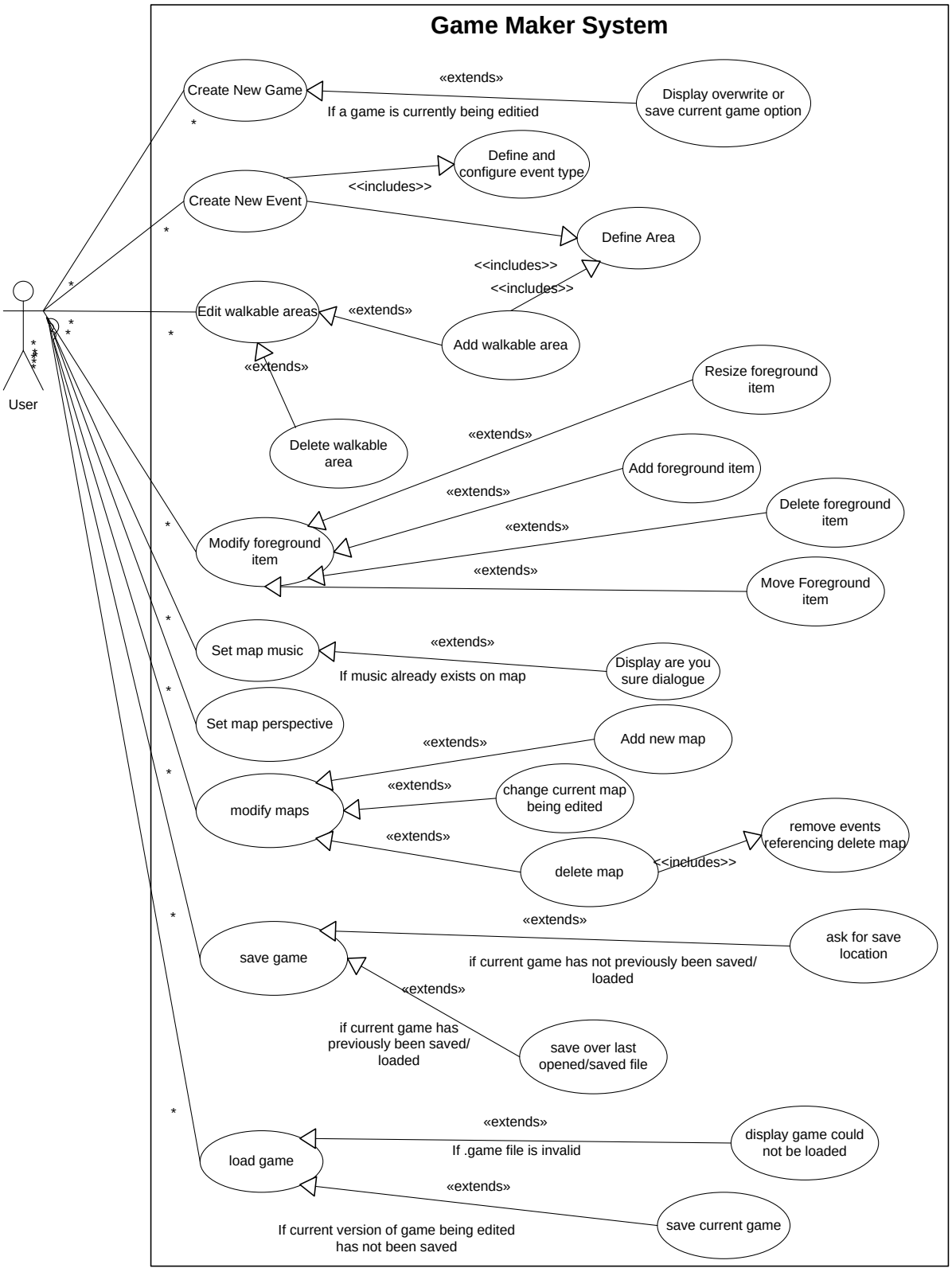
SWT Ribbon (crumhorn 2009)



I have chosen to use the Flamingo Ribbon over the SWT ribbon. The Flamingo Ribbon has more customisability options than SWT and is a lot more mature in its level development. The Flamingo Ribbon is licensed under the Berkeley Software Distribution (BSD) License (See Flamingo license - Berkeley Software Distribution (BSD) License (Flamingo 2008)(Flamingo 2008) in Appendices). This means that as long as the terms of the licence are included in the project source code and documentation (In this case, this document) then the code can be used and modified without limitation. The SWT ribbon did not specify what licence it used which is another reason not to use it. Pre-implementation mock-ups can be found in the User Interface Mock-up appendix.

Use Case Diagram

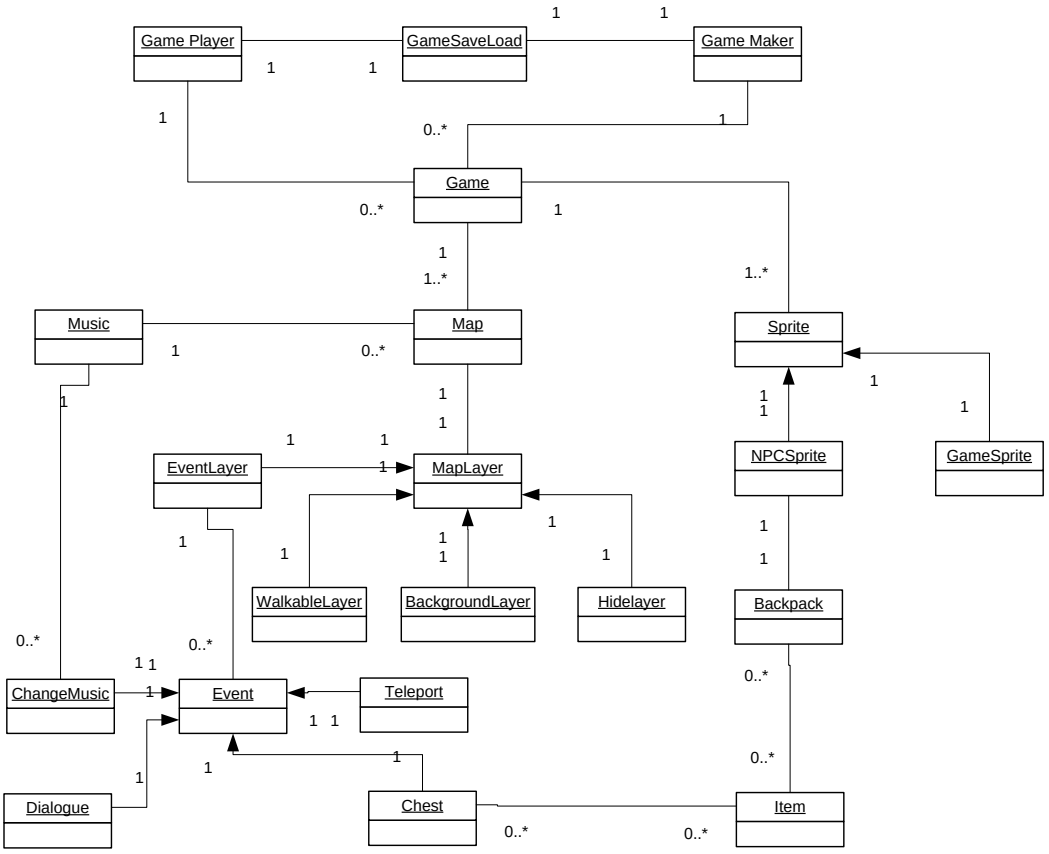
This is the use case diagram that represents how a user will be able to interface with the game creation elements of the project:



Outlining the use cases for game maker helps to identify what interface controls are going to be required to make a game (see User Interface Mock-up appendix). It can also be aid the thought process when designing the programs class structure.

First-Cut Class Diagram

This first cut class diagram is a preliminary design of the final class structure of the project. I anticipate that small changes will be made to this throughout the implementation phase, however the final structure will be very similar.



The main motivation for designing the class structure is to simplify the way in which extensions can be added to the project. The MapLayer contains further independent layer objects that extend the functionality of the map. More sub layers can be added to extend the capabilities of the map layer if required. This is important because it means that the design focus can be put into completing the core of the project first and any extensions to the core as defined in the specification as "Additional Features" can be added later without significant rework to the existing code being required.

Events will be able to be added to the system by implementing the event class and adding a new reference to the event in the event layer class. Noncore components such as the backpack object can be removed without impacting the core backbone of the program code.

Implementation

Overview of Completed Project

The completed system allows a user to create game. They are able to do this by creating new maps and setting the background of those maps to look like a level. The start position of the sprite can be defined. A play button can be pressed at any point to play the created game and the keyboard is used to control the sprite when playing the game. The game is able to be played in windowed or full-screen modes.

Tools are available to allow the user to define where the sprite can walk on the map when the game is being played, this was an important design consideration as a map would feel less interactive if the sprite was able to walk anywhere without restriction. Images can be added onto a foreground layer that is above the sprite when the game is played. This can be used to give the impression that the sprite is in a three dimensional environment because it is able to walk behind objects.

Illusion of 3D

Overlaying foreground image

Bad walk-able area design

The user must be careful if designing a map in this way that they position their walk-able areas so that the sprite will not intersect the foreground object at a location when it is supposed to be in front of it.

The perspective can be defined on a map so that the sprite changes size and it walks towards / away from the front of the map. This gives the impression of distance. Every map created is able to have its own perspective defined.

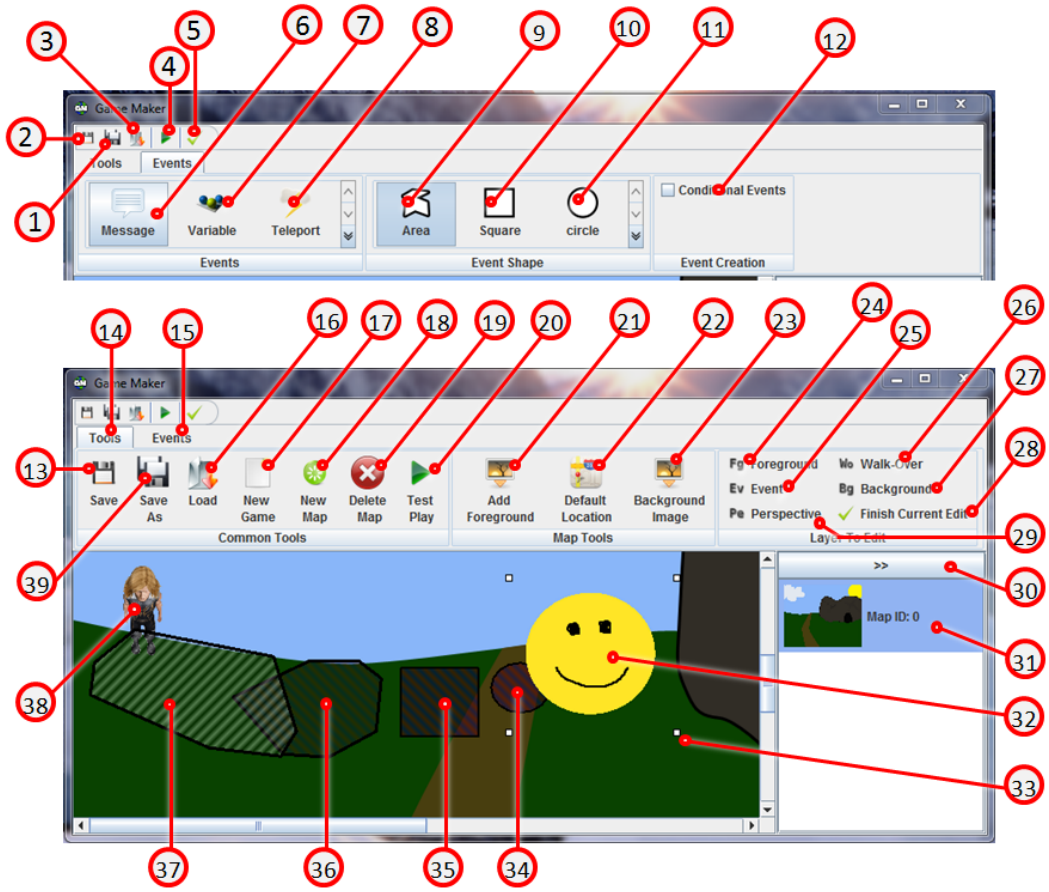
Events can be created that occupy an area of the map that can then be triggered by pressing the action keyboard key when in the play mode. There is an in game variable system and events can be created to be conditional on these events meeting specified rules (e.g. if variable 1 is greater than value 0). Events can be placed that: Change the in-game variables; display a message; teleport the sprite to specific coordinates on a new game map.

There are save and load functions that allow the created game to be saved and loaded from a zip archive that has been renamed with the extension .game. This archive contains all of the images imported into the game as well as an XML file representation of the created game.

The rest of this implementation section of this report will expand on the decisions posed in the creation of this functionality.

Interface

The ribbon interface has proved to be a worthwhile choice of interface layouts. Buttons are large and clear and it has made the software easier to use. Here is a screenshot from the game making interface along with labels that describe each interface item:



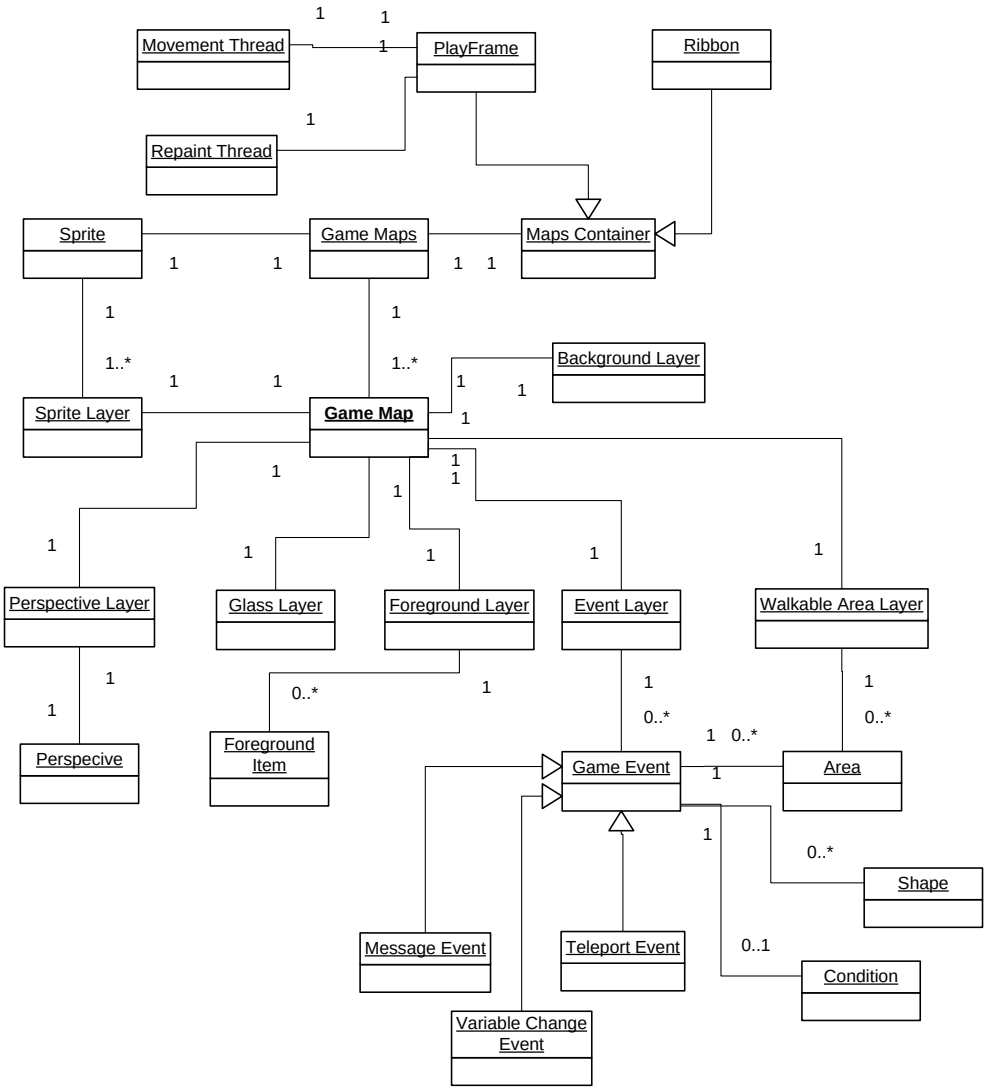
Ref	Item	Function
1, 39	Save As	This button allows the user to save their created game to a specified location.
2, 13	Save	Saves the game to the last saved or last loaded file. If the file has not already been saved / loaded then it will present the user with the option of where to save.
3, 16	Load	Prompts the user to load a .game into the program.
4, 20	Play	Plays the game that is currently being edited.
5, 28	Finish Edit	Completes any current drawing or event creation process. Middle click on the mouse also has the same effect.
6	Message Event	Click to create message events. Once pressed the cursor changes to a crosshair. Now the map can be clicked on in order to specify the event area. When in play mode, message events display a message on the screen when triggered.
7	Variable Event	Click to create variable change events. Once pressed the cursor changes to a crosshair. Now the map can be clicked on in order to specify the event area. When in play mode, Variable events can display a message on the screen when triggered; they then change the value of an in game variable.
8	Teleport Event	Click to create variable change events. Once pressed the cursor changes to a crosshair. Now the map can be clicked on in order to specify the event area. When in play mode, Teleport events teleport the sprite to the specified map at the specified location.

9, 10, 11		Set Event shape. Set whether the event that you create on the map is in the shape of a square, circle or custom defined area.
12	Conditional Events Toggle	If checked, all events created will prompt the user to specify what condition must be met before the event will be triggered in play mode.
17	New Game	Creates a new game. The user will be prompted to save or discard their current game.
18	New Map	Adds a new map to the game that can then be selected and edited from the right panel (See Item 31)
19	Delete Map	Deletes the current selected map and any teleport events that teleport to the selected map
21	Add Foreground	Once selected, click on a location on the map to add a foreground image at that position. Once the map has been clicked, the user will be prompted to select the image that they wish to display at that location
22	Default location	Once selected the user can click on a location on the map. This location will then be used as the default starting position when playing the game.
23, 27	Background Image	User will be prompted to locate an image that they want to use as the map background. Once selected the background will change to be this image.
24	Edit Foreground	Once selected, the user is able to move and resize foreground items by clicking on them and interacting with them using the mouse.
25	Edit Event	Add the last selected event to the map. User will need to specify where.
26	Walk over	Once selected, the user can click on the map to specify where the sprite can walk. Double click or press the finish edit button to complete the area.
29	Edit Perspective	This function is used for setting how big the sprite is depending on its position on the map. Once pressed the map will be shaded blue, and green. Drag horizontally in the blue area to set the height at the maps horizon point or in the green area to set the sprite size at the front of the map. Right click to set the horizon at the click position.
30	Minimise Panel	Minimise the right panel.
31	Maps List	A list of all of the maps created so far. Click on a map to view it.
32	Foreground Item	An example foreground item. In foreground edit mode click on the item, it will then be selected and brought to the front.
33	Foreground Re-sizer	Click on it and drag to move it. Click and drag on the white boxes in its corners in order to resize the object.
34,35,36	Event Areas	These represent the location of a created event. Events are represented by an area filled with red and blue lines drawn at a 45 degree angle from top left to bottom right.
37	Walk-able Areas	A representation of the area that the sprite can walk within. Walk-able areas are filled with white and black lines drawn at a 45 degree angle from bottom left to top right.
38	Start location	The sprite is displayed on the location on the map that it will start at when the game is played.

Shortly into my implementation of the Flamingo Ribbon API I had to make significant changes to how the API stored buttons. When adding a button to the ribbon, it did not set an action command for the button and there weren't any predefined functions that allowed me to fix this easily through a simple additional function call. This was fundamental as it did not allow for a single action listener class to handle interaction under the Model-View-Controller structure. To fix this the code was modified to automatically make the name of a button the action command value, functions were also added that allowed for a custom definition of the action command variable. Fortunately, Flamingo Ribbon licence permitted these changed to me made legally.

Icons used in the interface design are free from iconspedia.com. Some icons require referencing of the source and those references can be found in the Icon References (IconsPedia n.d.)(IconsPedia n.d.)appendix.

Final First-Cut Class Diagram



Above is a simplified class structure of my completed project. It is simplified because it does not contain less significant classes to handle operations such as input dialogues or file access. This class structure can be analysed to see how I used the model-view-controller (See Model-View-Controller Design Structure Appendix) design principal within my program.

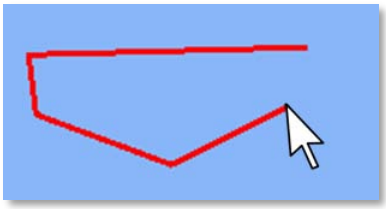
The classes that implement the Maps Container class act as view section of the model-view-controller. The interface items, action listeners etc are managed here. The Game Maps and Game Map classes are the controller elements, these handle the processing of commands initiated through this view section. Functions within Game Map are most commonly initiated via the Game Maps controller. The layers that are contained within the "Game Map" layer are the model and these store the state of each map and layer of each map.

The design and play both interact with the same controller class to simplify creating the play mode of the game. A lot of the functionality of playing the game was then already available as the same Game Map class of object can be used by both Make (Ribbon.java) and Play (PlayFrame.java) classes.

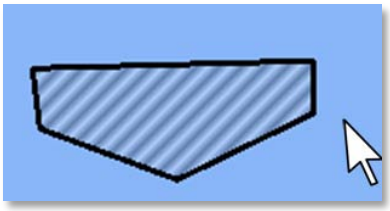
Representing Areas

Event areas and the walk-able areas are represented as a sequence of points. The shape object has its own drawing function that can be called by the object that references it to make it simple to draw onto a panel. Each area has a Boolean state to indicate whether the area has been completed or not. This state is used when drawing to identify if a complete shape is drawn, or only the boundaries of the shape.

Unfinished Area



Finished Area



Having an independent drawing function saved development time as any aspect of the game maker that required an area to be implemented or drawn could simply reference a new area object. This was useful as both the walk-able area and events make use of area representation.

Storing of an area as points also made it simple to save the created game into an XML format because this vector representation reduced the amount of information needed to be stored for each area to only its points. If a different area representation method had been used then an alternative representation of the saved area would also require to have been implemented.

Foreground Items

Foreground items can be added to the foreground items layer of the map. In the foreground edit mode, the user can select a foreground item and resize it using one of the four white resize boxes in the corners of the image.

Add and selected single foreground item

Add and resize second item

When in the foreground edit mode the game map listens for when a mouse has been pressed. When pressed and released the coordinates are passed to the foreground layer where it checks the coordinate against the bounds of all of the images that it references. If an image is within those bounds then the four resize boxes are displayed on the corners of the image and the image if this image is behind another then it is brought to the front.

The management of the resizing of foreground items had to be manually coded and was a great deal more complicated than originally anticipated. The four resize boxes are drawn on a layer above everything else called the 'Glass Panel'. The glass panel was required to prevent the resize boxes appearing behind another foreground image. If a foreground item is selected then the foreground panel

listens for the coordinates of the mouse every time that it is moved. If the mouse is over a resize box then the cursor is changed to indicate that the box can be dragged to resize the image.

If the mouse is pressed when it is within the bounds of one of these boxes then the pressed coordinates are stored along with a reference to which box had been pressed. The foreground layer then listens to the mouse position and updates a second coordinate that indicates where the mouse has been dragged to. These from and to coordinates in reference to the index of the resize box that was dragged can be used in order to move and resize the image.

Conditions had to exist to handle what happens when each corner is selected. If the top left hand box is selected and dragged then the image must be both moved and resized as the images are drawn in reference to the top left hand corner. However if the bottom right hand box was dragged, the image only has to be resized. Another issue is if the bottom right hand box is dragged to the top left of the top left hand box. In this case the box must again be moved and resized. (The algorithm to do this can be found in the Resize foreground items algorithm appendix).

This design decision was inspired by commonly used design heuristics (see HCI heuristics appendix) as resizing images through interaction with their bounding resize boxes is a commonly used method. For instance, it is used in most of the Microsoft Office and Open Office tools as a method of resizing objects. A user is more likely to be familiar with this way of viewing it and be able to instinctively use the function as intended.

Sprite Perspective

One of the original goals of the project playback was to be able to give the impression of navigating around a three dimensional environment without actually having to process any 3D graphics. One way that this was achieved was by having a folder of images representing the sprite facing in eight different directions. The images within these folders would then be played in sequence in alphabetical order to give the impression of the character walking across the screen.

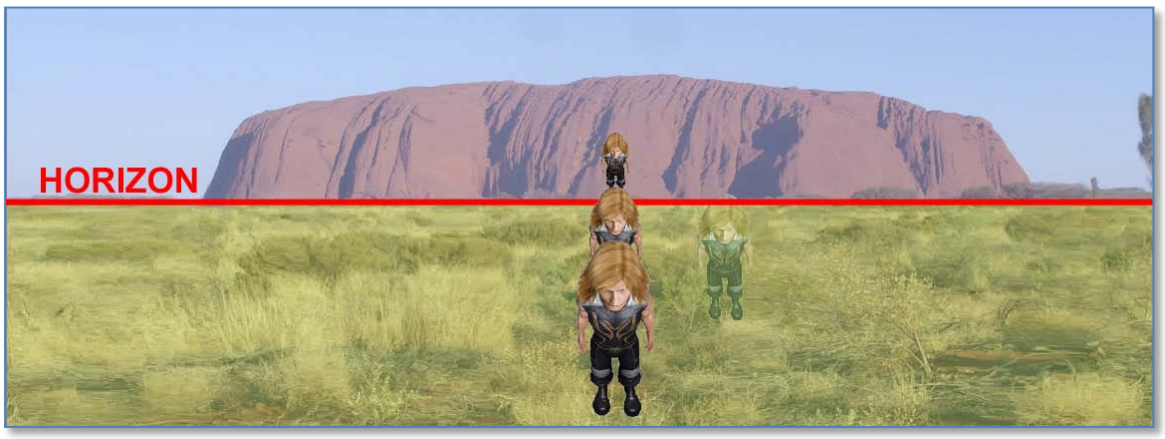
The above sprite was the one used in my program. It was made using a trial version of iClone 3D animation software (Reallusion Inc 2010). The images are .PNG images with a transparent background.

Considering that simplicity of game creation is a key goal of the project, it was the most suitable solution for the problem as using a sequence of images simplifies the process of customised sprite creation. If a user knows how to use a 3D modelling package then they can just export the frames as a sequence of images like I have. Alternatively it gives the option for a user to draw their own sprite frame-by-frame, there is then no requirement to understand 3D modelling tools. The RPG Maker package evaluated in the background section has a thriving community that create and share customised resources such as sprites and maps, this easy customisation and sharing of characters could also be applied to the Game Maker.

Another way of creating a pseudo-3D character is to implement a system of creating perspective. If a user uploaded a photograph to use as a background it is important for interaction to look realistic that the sprite changes size in relation to the perspective of the image:

To achieve this the sprite was given a default size of 70 pixels in width and then assigned a percentage value of this width for the front and the back of the map. As the sprite moved around the map it was resized in relation to its current position and the perspective percentages. For example if the top percentage was 0 and the front percentage was 100: the sprite would be 70pixels wide if at the front of the map; 35 pixels wide if directly in the middle and 0 pixels wide if at the back.

This presented an issue because if the horizon of the image was in the centre the smallest the sprite could be at this position is 50% of the size of the sprite at the front of the image. For example if the sprite was 100 pixels tall at the front of the image and 0 pixels at the back and there is a building at the horizon of the image that is 30 pixels tall. If the horizon was in the middle of the image then the sprite would be 50 pixels tall at this position, which is larger than the building. To fix this a method of representing the position of the horizon was implemented:



The back percentage was changed to relate to the relative size of the sprite at the horizon instead of the back of the image. It was then made so that the user can right click anywhere on the map to set the horizon of the map to that point, this is illustrated in the above picture. Once the horizon is set, the user can left click and drag their mouse horizontally in order to change the size of the sprite at the horizon, they can also do the same underneath the horizon to alter the sprites size at the front of the map.

This is a pseudo code representation of the updated perspective algorithm:

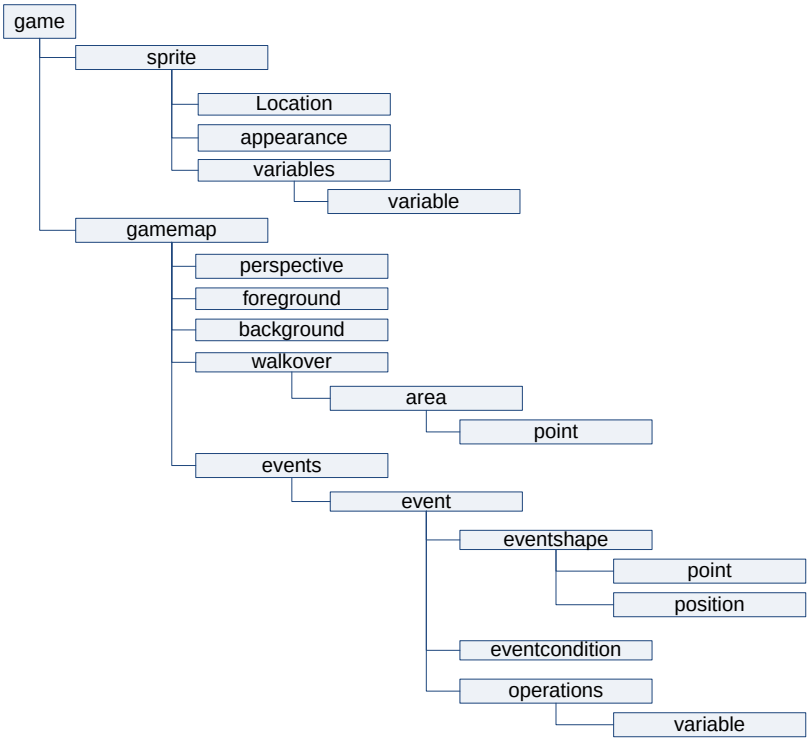
```
//For calculating the sprite size in reference to the horizon
Logical height = height of the front of the map to the horizon

// Works out the sprites position between the front and the horizon as a percentage
```

```
Position as Percentage =  (Sprite Y Position - Horizon Y Coordinate)  /
                          (( Logical height - Horizon Y Coordinate)/100)

// Difference = the sprites perspective percentage in reference to its position
Difference = (Front Percentage - Back Percentage ) * Position as Percentage / 100

// Compensates for if the front perspective percentage is larger than the horizon percentage
// and vice versa.
If Front Percentage > Back Percentage
    The Perspective = Back Percentage + Difference
else
    The Perspective = Back Percentage - Difference

//Sets the sprite width based on its perspective at its current position
Sprite Width = Default Sprite Size * The Perspective
```

This algorithm is calculated and the image of the sprite is resized every time the state of the game is refreshed by threads in the game play mode. The relative perspective of the sprite is also applied to the speed at which it walks so that the sprite moves across the screen in relation to its size. There were concerns that resizing the sprite image so frequently would have a significant impact on performance, however because the images of the sprite are relatively small (200 pixels x 300 pixels) this was not a problem on any of the computers that the program was tested on. The original 200x300 image is always kept in memory and when it is resized a copy is made. This is to prevent permanent image information loss if the sprite were to be reduced and then increased in size. The copy is overridden every time the sprite is resized in order to make efficient use of memory.

XML Implementation

Here is a diagram illustrating the structure of the XML document produced when saving a game:



Each box represents an entity and each link represents inheritance within that entity. Each entity corresponds to an object within the game maker and contains attributes that represent the state of

these objects at the point at which they were saved (See XML entity attributes reference: appendix for a detailed description of entity and attribute definitions).

Implementing the saving and loading of XML was achieved through a tree like execution path through objects in memory. You should notice that the XML structure above has a similar structure to the class structure, this was a deliberate design solution to simplify the save/load process. All classes that required representation as XML implemented a custom interface 'XMLable'. Consequently they were required to have 'toXML' and 'fromXML' functions that handled the creation and parsing of XML. These functions are coded to call the 'toXML' and 'fromXML' functions of any of their child components that also require to be represented as XML.

Another key benefit of using this method of implementation is apparent when a new class is created that will contain information that needs to be saved. The amount of code rework required as a result of the change is limited to the "toXML" and "fromXML" functions of the new class and its parent. This means that new classes and functionality can be added to the program and only a small number of changes to other classes need to be made for them to be fully integrated with the save/load mechanism.

Because the 'toXML' and 'fromXML' functions can be explicitly defined, the programmer has full control over what information is stored when the game is saved. Images used as map backgrounds and foreground items are exported as independent image files using PNG image compression (Roelofs 2008) and references to these images are defined in the XML document. If object serialisation had been used to store the game state then this would not have been an option. Consequently these images would have been stored uncompressed as a binary dump of the objects themselves resulting in a much larger final file size. Considering that an objective of saving the game as a single file was for it to be portable, a large file size would have been a conflicting outcome.

The JDOM API's (JDOM n.d.) were used to process the saving and loading of the XML file. JDOM was used to simplify the process of creating a well-formed XML document and is a widely used and understood set of libraries. JDOM was used over other alternatives such as FINK (FINK 2009) as I already had experience using it and knew it would be fit for purpose. When saving, a completed string created by calling the 'toXML' methods is passed to the JDOM object. It then ensures that the string is well formed XML and that useful exceptions are thrown when necessary, it also formats and exports the code with correctly positioned tabs and indents to make it easier to read. When loading the XML its elements are passed to the 'fromXML' functions where they are deconstructed appropriately to create/populate objects.

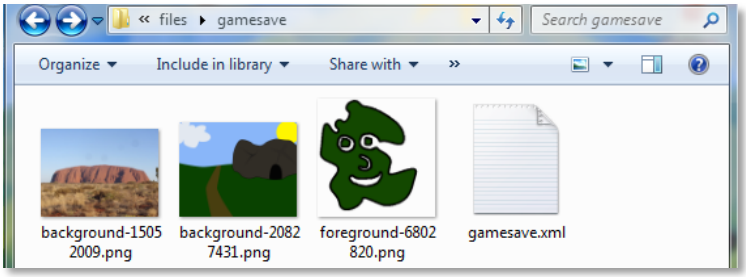
Other benefits of using XML to save the game will be discussed further in the testing section.

Saved Game File

When a game is saved, the images and XML file that represent the game are stored within a 'gamesave' folder that is located within a parent folder 'files' alongside the project executable .jar file. relative referencing ensured that as long as the structure of the distributed folder remained the same, the software would run from any location. This design decision made it much easier to run the program on different machines in comparison to using absolute referencing.

This folder is then compressed into a .zip archive that is renamed to be .game and then placed in a location specified by the user. The Utils4J API (Future Invent Informationsmanagement 2009)was used to simplify the process of creating a .zip directory. The utils4J APIs are licence under the LGPL open source licence (GNU 2009).

Example 'gamesave' folder structure



When the .game file is loaded, the 'gamesave' folder is replaced with a new one containing the contents of the file. This made implementation easier because standardised java file operations can be made using local references to items within this folder.

As you can see from the screenshot above, each image has a prefix indicated that it is a background or a foreground image. The original implementation was to have a 'gamesave' folder to contain the subfolders 'background' and 'foreground' in order to store background and foreground images separately however this had to be changed.

Windows identified folder structures using a backslash "\" character in the address string whereas Linux and Unix based operating systems use a forward slash"/". Throughout the project code it was ensured that the 'File.separator' reference was always used when defining a URL or file reference so that file references are always handled correctly across different platforms.

The Utils4J API must not have made the same provisions as if a archive was made on a windows machine and then extracted on a Linux one then the folder structure would not be preserved. For example: instead of having the folder 'foreground' contain an image '1234.png' there would be no folder structure and the file would be called 'foreground\1234.png" within the root folder. A simple solution to this issue was to remove having a subfolder system completely and replace it with file name suffixes instead.

Play Mode Controls

The keyboard is the input method used for in game controls. Here are the button functions:

Key	Function
Arrow (Direction) Keys	Controls the movement of the sprite around the map.
Space Bar & Return	These are the action keys. If the sprite is above an event that can be triggered, pressing one of these keys will trigger that event.
F11	Toggles into and out of full screen mode.

Shift	If held, the sprite moves around the map faster when the arrow keys are pressed.
ESC	If the game is being played in full screen mode then the ESC key returns the view back to windowed mode. If the game is in windowed mode when ESC is pressed then the game play mode is exited and the game maker screen reappears.

The decision regarding what keys to use for which functions were a combination of applying Nielsen's usability heuristics (see HCI heuristics appendix) and through anthropometric considerations.

It is very common to use the return key as a method of confirming entry. For example in the case of using website forms, you can always press enter to complete the form rather than having to press the submit button. However when playing the game it was found that the return key was too close to the arrow control keys to be able to comfortably use both. For this reason the space key can also be used as the action button. Anthropometrically it is a large key that is far enough away from the arrow keys that the player is able to keep their arms at a shoulder width distance apart and using this as the main action key makes the game more comfortable to play.

The shift key was chosen for the button that makes the sprite run due to its distance from both the arrow and space keys. The run key is required to be kept pressed whilst using the arrow keys so it is much easier to do both of these functions with separate hands. Also the positioning of the shift key allows the little finger of the left hand to keep it pressed while the thumb still has full access to the space key. This makes the games most commonly used functions easy to access without discomfort.

The HCI heuristics indicate that you should make design decisions based on what the user is likely to already be familiar with. F11 is commonly used to toggle full screen mode in a lot of existing applications (for example Mozilla Firefox (Mozilla 2009)) which is why it was chosen to also use it for the this function. Similarly the ESC key is often used to exit applications and display modes.

Walking

The sprite has been designed to be able to walk in eight different directions. The directional arrows on the keyboard are used to control the sprite however this could not simply be implemented using a key listener. This is because a key listener triggers the pressed and released functions when any key is pressed, there is no standard function for recognising when multiple keys have been pressed together.

To resolve this an array of four Boolean values was created, each index referencing one of the four arrow buttons. When a button is pressed, the corresponding Boolean value is set to true, when release it is set to false. Rather than update the sprite position when a button is pressed, the thread that updates the game state checks the value of these Boolean values upon every iteration and updates the sprite position and direction accordingly.

Because threads had been implemented to handle the sprites movement this solution could be easily implemented without significant modification of the program code.

Efficient Playback

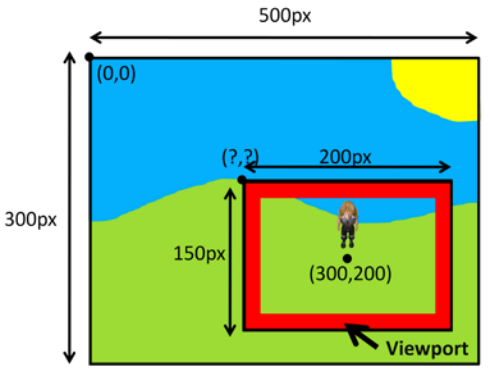
The initial code that was written to handle the playing of the game worked, however playback was not smooth and the screen would flash. The original implementation had games layers within a layered pane and this layered pane was contained within a scroll pane. To keep the sprite in the middle of the screen viewport a 'scroll a rectangle into view' function was called, followed by the Layered-panes predefined method 'repaint'. This issue was not anticipated in the design phase of the project.

There were two solutions to this problem that had to be implemented to improve the playback performance: Double buffering and removing the usage of the scroll pane by implementing my own method of painting the map in a way that still scrolls the sprite into view.

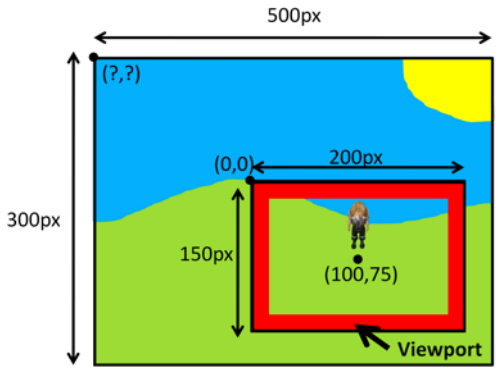
Double buffering (Sun Microsystems 2010) is the painting of components to a buffer before painting it to the viewport. The advantage is perceived performance as the screen is painted onto the viewport as a completed render as opposed to displaying a partially drawn image. This fixed the screen flashing issue but did not solve the problem of playback being smooth.

The playback performance was only slow if the image was being scrolled, so I came to the conclusion that it was the JScrollPane that was causing the issue. To fix this the image had to be drawn on a panel in the correct position in reference to the viewport *(Note: Java draws images onto a panel in reference to the top left hand corner of the panel, so the top left is point 0,0.).*Consider these images:

Coordinates in reference to image



Coordinates in reference to viewport



In order to draw the image so that the sprite is within the centre of the viewport the image has to be drawn at a position that allows this. Here is the pseudo code to achieve this:

```
If Background Image Height > Viewport Height
        // if  here then the image must be drawn outside of the viewport coordinates.
        // in other words drawing coordinates must be negative.
        Y =  0  –  ( (Viewport Height - Background Image Height) / 2 )
Else
        If (Spite Y position in  reference to image) >= Image Height
                //If the image is smaller than the size of the viewport then draw in the centre
                Y = Image Height – Viewport Height

//Sign is reversed so that image draws from top left of viewport
Y = 0 - Y
Note: This code can also be applied to x if you replace height for width.
```

Known information is: the size of the map; the size of the viewport; the location of the sprite; and that the sprite must to be drawn in the centre of the viewport. If drawing the sprite in the centre of the viewport results in the image edges being drawn on the screen then the edges of the image should clip

to the edge of the viewport instead. In the "coordinates in reference to the viewport" example the top left position of the image must be drawn at coordinates (-150,-75).

Threads in play mode

When the game is being played the graphics on the screen needed to be updated to display the current state of the game. Initially the appropriate panels were set to repaint every time the sprite moved, however this resulted in jerky playback because the rate at which the scene was being updated and repainted was not constant.

To fix this two threads were implemented, one that would update the state of the game without repainting it onto the panel and a second to repaint the panel. Both threads execute their functions at a set frame rate in order for the game to play back smoothly. I was able to achieve this by putting the main thread function within an endless loop and then having a wait function that waited for a specified number of milliseconds. For example a wait time of 1000/20 would result in a refresh rate of 20 times a second. An advantage with using threads was that the 'Thread.sleep(...)' function could be used instead of busy waiting to make the program wait efficiently.

Initially the thread execution rates were both set to a constant 20 frames per second (fps). The playback was smoother than before however it was still not smooth enough to appear seamless. PAL and NTSC are video format standards used across Europe and the USA. The PAL format plays video at a frame rate of 25, NTSC plays at a frame rate near 30 (29.98fps) (Dozal n.d.). I chose to apply the higher of the two rates as the thread refresh rate. Being widely adopted video formats they have proved to be of an acceptable standard. A much higher rate would result in poor playback performance on lower power computers so the decision was made not to increase the repaint rate to above 30.

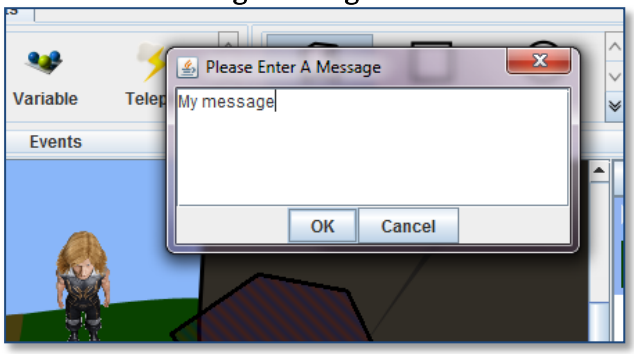
At a refresh rate of 30 times-per-second playback was much smoother however occasionally a frame would skip as the sprite was walking. With both threads refreshing at the same rate, there were occasions where the painting thread would paint before the updating thread had updated the state of the game since the last repaint. The frame rate of the updating thread was increased to 40 times per seconds to compensate for this. The CPU overhead of an additional 10 refreshes per second is an acceptable compromise as the updating thread does not repaint.

Event Creation

Games created using the Game Maker are event driven. Once the core of the game making and playing elements of the program were completed it is the diversity of the events that make the software powerful. It was important that events could be easily programmed into the system.

To create a new event you must create a new java class that extends the abstract class GameEvent. The notable functions within GameEvent to be overridden are 'configureEvent' and 'eventAction'. The 'configureEvent' method is called when an event is created so that the user making the game can set what the event does. The event action function is called when an event is triggered when playing the game. In the example of message events, a dialogue box is displayed asking for input when 'configureEvent' is called and a message is displayed on the screen when the 'eventAction' function is triggered:

Message Configure Event



Message Event Action

The restrictions with this design decision is that the programmer doesn't have flexability over what parameters can be passed to and from the 'eventAction' and 'configureEvent' functions. However overall this is a good way of handling this because once the new event class has been created a link between an interface button and the creation of the new event. This then be made to allow this event to be created within the game. Drawing of the event onto the map will automatically work because the GameEvent class has been extended.

All of the events currently available to be created were designed within this process.

Event Triggering

Every game map has a layer that stores all the events on that map. When the game is being played, the program listens for an action key to be pressed. If the sprites location is within an events area when the action key is pressed then the events 'eventAction' method will be called to handle the event. If the event has been defined as conditional then the contents of the 'eventAction' function will not be called unless the condition has been met.

In the specification it was planned that events could also be set to be triggered when the sprite walks over them. This was not implemented as an issue arose that could not be resolved by the project deadline: Events that were set to be triggered upon walkover would be triggered every time the sprite was moved regardless of how much by. To cater for this a mechanism would need to be implemented that only triggered events within a certain time threshold, or that only triggered an upon entering the event area (to retrigger the event the sprite would need to exit and then renter the area). Also there was a risk of an endless loop being created if a teleport event was created that teleported the sprite into the bounds of another teleport event that in turn teleported the sprite back.

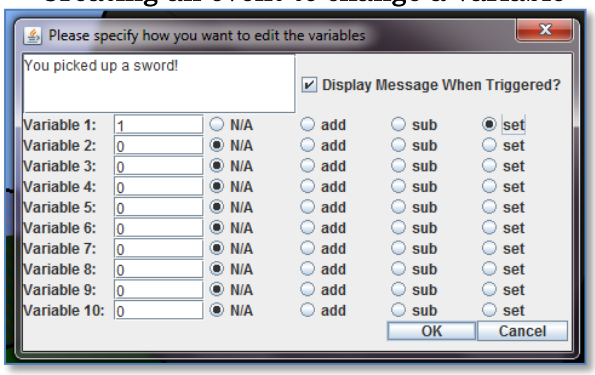
Only allowing events to be triggered by pressing an action button was a simple and effective way to resolve these issues which allowed the project to be completed by the deadline, however it did create another issue. When a person makes a game, they have to make it clear that an event is contained within a certain area for the person playing the game to know that they can press the action button to trigger that event. In some cases the player may easily recognise that it is likely that an event will be triggered for example like placing teleport events on a door but this will not always be the case.

In-Game Variables

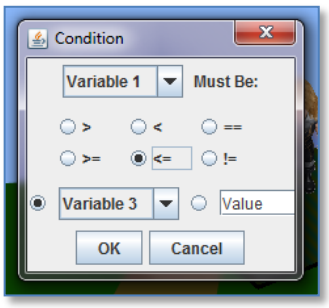
It was originally designed for the sprite to have a backpack of items and some events could only be triggered if an item was held in the backpack. The decision was made not to implement the backpack system but to instead have an in-game variable system. Consequently every game that is created has ten

in-game variables that are initialised to 0. These can be modified in game play by triggering a variable change event.

Creating an event to change a variable



Setting the condition of a new event



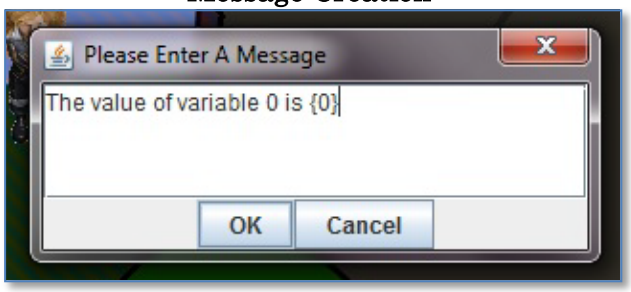
If a backpack system had been used then it would have been a list of Boolean values with a corresponding item name, if the item was held then the value would be set to equal true. A variable system is a lot more powerful as each variable stores an integer, this allows for more complicated event conditions to be created. Also the person making the game is still able to use the in game variable system to represent the collection of items and triggering of events as a result.

Message Variable Parsing

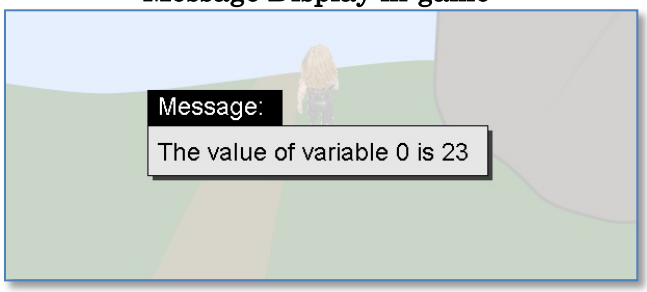
For my project demonstration I wanted to create a small quiz using the game maker to illustrate its broadness of application. In preparing this it was identified that there was a need to reference the in game variables in order to give the person playing the game feedback on their performance on the quiz. For example, every time a question is answered correctly variable 1 could be incremented, at the end of the quiz the player could be told the value of variable 1 so that they know how they performed on the quiz.

A new event could have been created to display a message when triggered that displayed a message alongside the value of a variable, however this solution was not adopted because it would complicate the user interface with more controls. Instead the system of doing this was inspired by the BBCode system used on most internet forums (See BBCode Appendix).

Message Creation



Message Display in-game



A parsing system was implemented that allowed users to put the index of a variable within curly brackets of a message event. There is no requirement to have other tags so it is acceptable use the curly bracket as a tag even though it has no name to identify it.

The advantage of this implementation is that is avoids the need to create an extra event to handle the display of the in-game variables, instead the existing message event can be used. Also multiple variable references can easily be displayed by using multiple tags within a single message. The disadvantage is that the user is required to know that they must use the tag in order to reference a variable.

To parse the tags the regular expression "[{|}]" was used to split the string in to an array by every occurance of a curly bracket. Every odd array position is then checked for a number. If a number is found then it is the variable at the index of that number that is inserted into its position. If a non integer value is found between the brackets then the message parsing fails and is displayed with the original tags and no variable substitution.

The array is then printed in sequence. This method was used because it is simple and fast. There are ten in-game variables so the modulus of the variable reference is used to ensure only a valid index is returned. For example if {11} was used as a reference, the value of variable index 1 would be used because 11%10 = 1.

This validation method was used to simplify the implementation of this feature however it has limitations as the user needs to be aware that this is the way invalid indexes will be handled. If more time was available to complete the project then the program could be engineered to throw an error if an invalid variable index is used.

Creating a Teleport Event

When a teleport event is created, the following dialogue is displayed to the user allowing them to specify where the teleport event will take the sprite when triggered.

To define a teleport location, the map is selected from the right hand maps list, the user can then click on the right hand picture of the map to define the location on the map. A marker is placed in that location to give feedback to the user that their input has been accepted. Once the user is happy with their decision, the OK button can be pressed to confirm it. If cancel is pressed then the event is not added to the map.

The dialogue was designed to have a similar layout to the main game making interface so that the user would be familiar with it. It also facilitated for code reuse as the same interface objects that had been used in the main interface (such as the right hand panel) were used in the dialogue. The ok and cancel buttons of dialogue boxes frequently placed in the bottom right hand corner of a dialogue in that order. This was replicated in all of the dialogue boxes used in the program to make sure that a user would be familiar with the design principals used in the software.

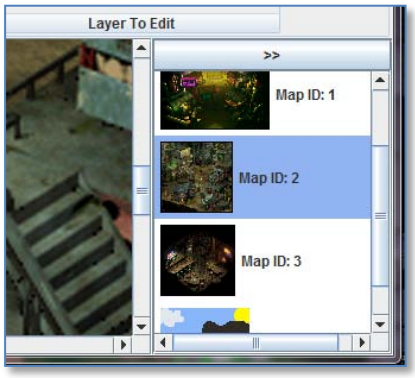
Having a visual interface to define the teleport location is more difficult to implement, however it is more straight-forward to use. An alternative would have been to present the user with a dialogue box

that requests the ID and coordinates of the map to be written and then confirmed. This would have been much easier to implement as less components would have needed to be integrated, but it would have been a much more difficult system to use. It would have required the user to remember and calculate the map ID and corresponding coordinates. The improvement in usability was worth the development time that was required to create the custom dialog box.

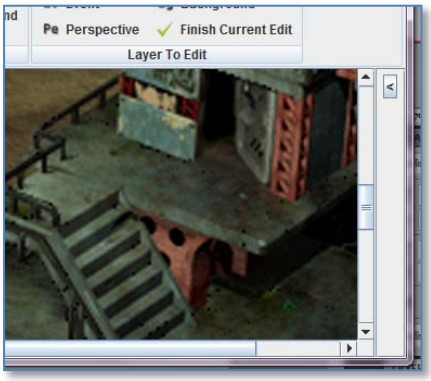
Right Maps Panel

It was important for a user to be able to quickly browse through the maps that they had created. It was decided to implement a dedicated side panel to handle this functionality.

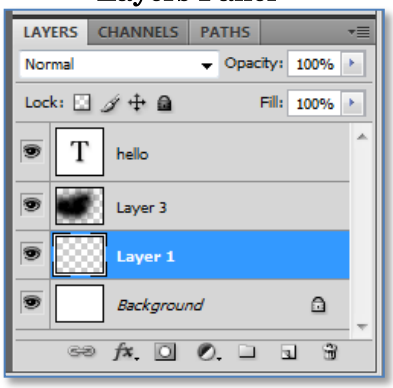
Expanded Right Panel



Collapsed Right Panel



Adobe Photoshop CS4 Layers Panel



To change the current map being edited, the user can just click on the map that they want to edit. If the user wants more space to work within then the panel can be collapsed via pressing the button labelled ">>". Once collapsed it can be expanded again by pressing the "<" button.

This management of maps within a right hand panel was another HCI consideration. Similar panels to the right of the interface can be seen within popular and established software tools. A good example of this is the popular photo and image manipulation tool Adobe Photoshop (Adobe n.d.).

The Java 'JList' library exist that allow for lists of text however there are not any standard libraries for creating a list of images. To create the map list, the java list API had to be extended to handle the painting and selection of panels as list items. An important HCI consideration (see HCI heuristics appendix) is to reduce the amount of information that the user has to remember. Using a text only list would have required them to select a list item in order to see the map that it represents and the user would have to remember which map id corresponds to each image. This would have conflicted with that HCI principal.

Testing

Development Hardware

The main development machine used to create and test the project was a desktop PC with the following specification: Window 7 professional operating system; Java 1.6 JDK Intel Core 2 Duo 1.6Ghz; 2Gb DDR2 Ram; integrated graphics; 1280x1024 monitor resolution. This is an acceptable standard to set as the minimum system requirements to run the software as it is a low power computer compared to today's standards, especially considering the absence of a dedicated graphics card. This is with the exception of Windows 7 operating system as the program testing has shown that it works on Windows XP and Windows Vista. These operating systems were tested within a virtual machine running on a high powered desktop computer of the following specification: Windows 7 operating system; i7 3Ghz quad core processor; 12Gb DDR3 RAM; dedicated 1Gb Graphics Card.

The lowest performance test machine used was a HP Net book with the following specification: Windows XP Professional Operating system; Java 1.6 JRE; 1.7gb SDRAM; single core VIA 1.2Ghz processor; integrated graphics; 1280x768 monitor resolution. This is a very low power computer, it is not even capable of playing back standard definition YouTube videos. The program worked, however game playback was not as smooth as on the higher powered test computers. This testing showed that the program was able to run on low-power computers but the slow performance confirms that higher minimum hardware requirements should be advised.

Java Out of Memory Exceptions

When testing on various computers it was identified that the software would throw a java "out of memory exception" when entering play mode. It was found that some java environments are assigned a much lower maximum memory size than others. This error can be eliminated by increasing the size of the heap allocated java, executing the program with the following command achieves this:

java -Xms256m -Xmx512m -jar Game_Maker.jar (Spell 1999)

This command increased the maximum heap size that java can use to 512Mb which has proved through testing to be large enough.

Exception Handling

To simplify the process of handling exceptions, all exceptions are caught within the action listener of the ribbon class, this at the control level of the Model-View-Controller design (See Model-View-Controller Design Structure Appendix). Exceptions thrown print their stack trace so that the source of the bug can be easily identified by the developer. Exceptions also always display a dialogue box to the user to indicate that an error has occurred.

For example, if the user tried to load in a file into the program that is not a valid .game file. An exception will be thrown down to the controller, this will result in an error message being displayed to the user telling them that the file could not be opened.

It was decided that catching exceptions at the controller section of the software was a better alternative than within the model section. If an exception was thrown and caught deep within the model section then the location that is thrown at would have to be purposefully located and checked for appropriate

exception handling. In the current implementation all of the catch blocks that handle the programs exceptions can be found in the one place. If dialogue boxes and error messages are placed within each one, it makes it less likely that the program can throw an error without the user being notified because unhandled exceptions can be easily identified by the programmer at this level.

Custom exceptions were created to ensure that meaningful messages and errors are thrown. For example if a parameter passed to a function does not conform to what the function requires then a custom exception could be thrown. When this exception is caught, more meaningful dialogue messages can be displayed to the user and more useful errors are printed for debugging.

XML

Using XML to represent data is very intuitive. It is simple to read by the programmer which makes it much more convenient to debug. It was easy to see whether bugs were being caused through a saving or a loading error by evaluating whether the XML document was correctly formed and that attribute values are valid.

If the XML document looks to be correct, then it is fair to assume that the loading functions are erroneous if the error has occurred after loading. If the XML document does not look correct then it can only be the save functions at fault.

Saving as XML provides an extra mechanism where incorrectly formed objects could throw an exception. This has resulted in errors being identified that may have otherwise gone unnoticed.

White Box Testing

White box testing is testing that the inner workings of classes and functions are working as expected. The incremental model of software development used in this project resulted in new classes and objects being created that would then be integrated into the main backbone of the program code. White box testing was applied to each new class and function within it to ensure that their internal processes were working correctly. A main method within the class is written to test it in isolation from the main backbone of the code.

For example the following algorithm exists within the software to check that a point is contained within the bounds of a list of shapes:

```
01) Take coordinates of point from parameter
02) Loop through every shape within the object
03)         Check that coordinates are within each shape
04) Return true if a coordinate is found to be within any shape
05) Return false if otherwise.
```

A set of test shape lists were created in order to test the bounds of the validation. One list may contain no shapes whereas another may contain every possible shape object to make sure that every eventuality is tested to work correctly.

Each line within the function can then be tested against its expected function. A breakpoint would be set on line 02. The state of the variables could then be checked to ensure they are correct. This test would then be applied line by line against every validation criteria and inputs and results would be plotted in a table:

Line	Test	Expected result	Actual Result	Solution
02	Ensure that correct number of shapes are returned	Variable "numshapes" will be equal to 5 as 5 is the size of the test set	Numshapes == 4	Numshapes is using an exclusive counting system. Make sure that the loop terminates on **"i <= numshapes"** as opposed to **"i < numshapes".**
...	...	...	...	...

White box testing is useful because it allows the individual elements of a function to be tested in isolation. If the function as a whole isn't behaving as expected then white box testing can be used to isolate the individual line causing the problem. A weakness of white box testing is that it is very time consuming to test for a large amount of test cases.

JUnit testing (Oracle Corporation n.d.) is an automated method of running and validating test cases in bulk and this can be used to reduce how long it takes to run these test plans. It's still time consuming to write the test cases that will be executed with white box testing as it is the internal workings of a function that are being tested. Junit testing is simpler when applied to black box testing due to only needing to measure if the output of a function is as expected given the input parameters.

Black Box Testing

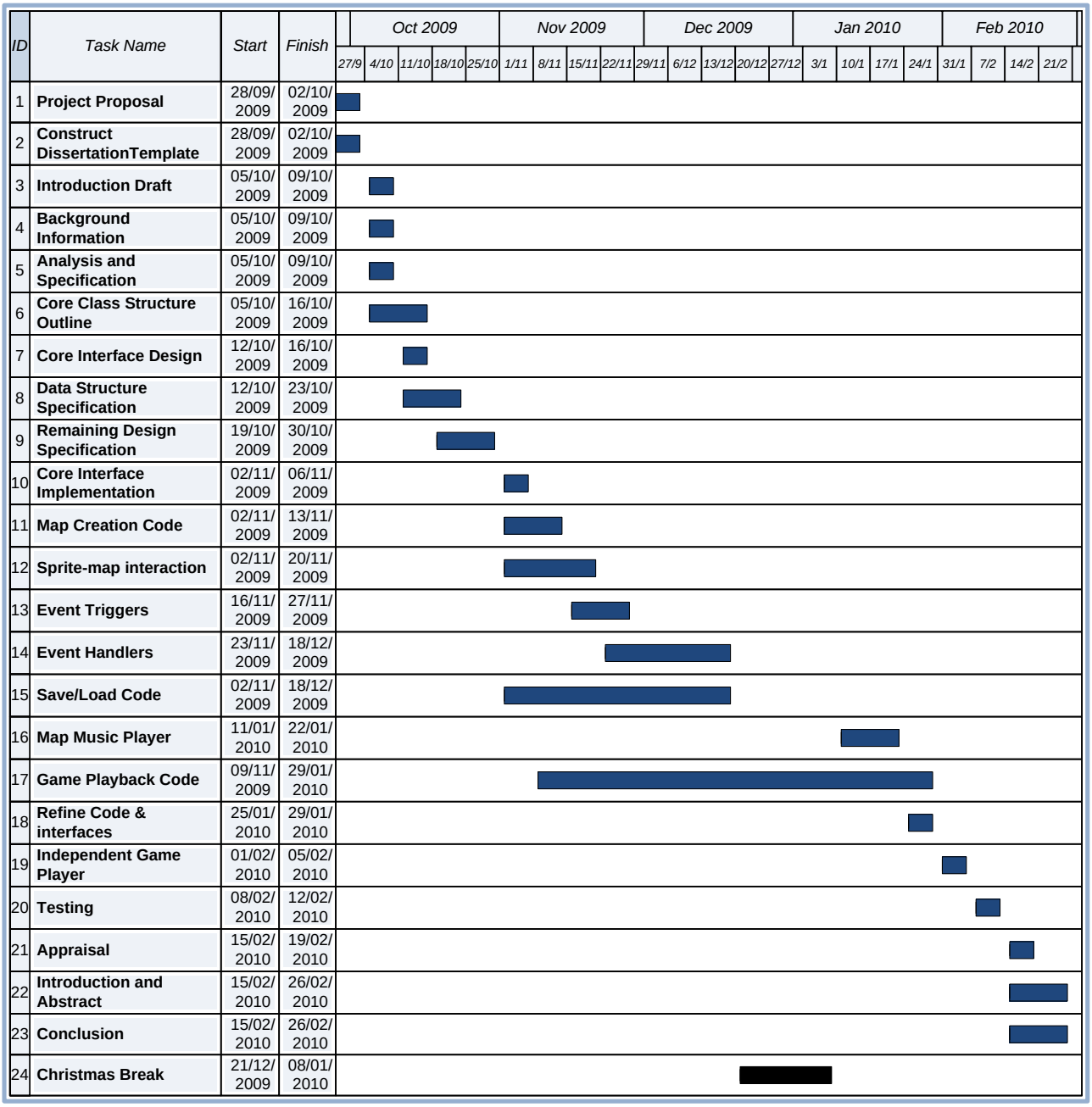
A drawback of white box testing is that it cannot be applied in all situations. With graphics the analysis of the actual result may be to visually check that the sprite is walking in the correct direction, or that the viewport is refreshing smoothly as the sprite moves. White box testing cannot be used to test this as effectively as black box testing.

With black box testing the internal workings of classes and functions are not dissected and tested individually. Instead a set of inputs are given to an object or function and the test is to see if the output is as expected. This method of testing was used when testing a lot of the graphical elements of the project. Most commonly a game environment would be created by the program and the test would be to see if the image displays correctly on the screen. If it doesn't display correctly then the function that controls that aspect of the graphics would be identified as needing to be modified. If possible, white box testing will be applied to this function to correct it. If not then personal initiative must be used when bug fixing to identify the possible cause and solution.

Project Management

Planning

The project was managed by dedicating time within my schedule (See Appendix Personal Timetable)specifically for my project work. A preliminary Gantt chart was created at the beginning of the project in the project proposal:



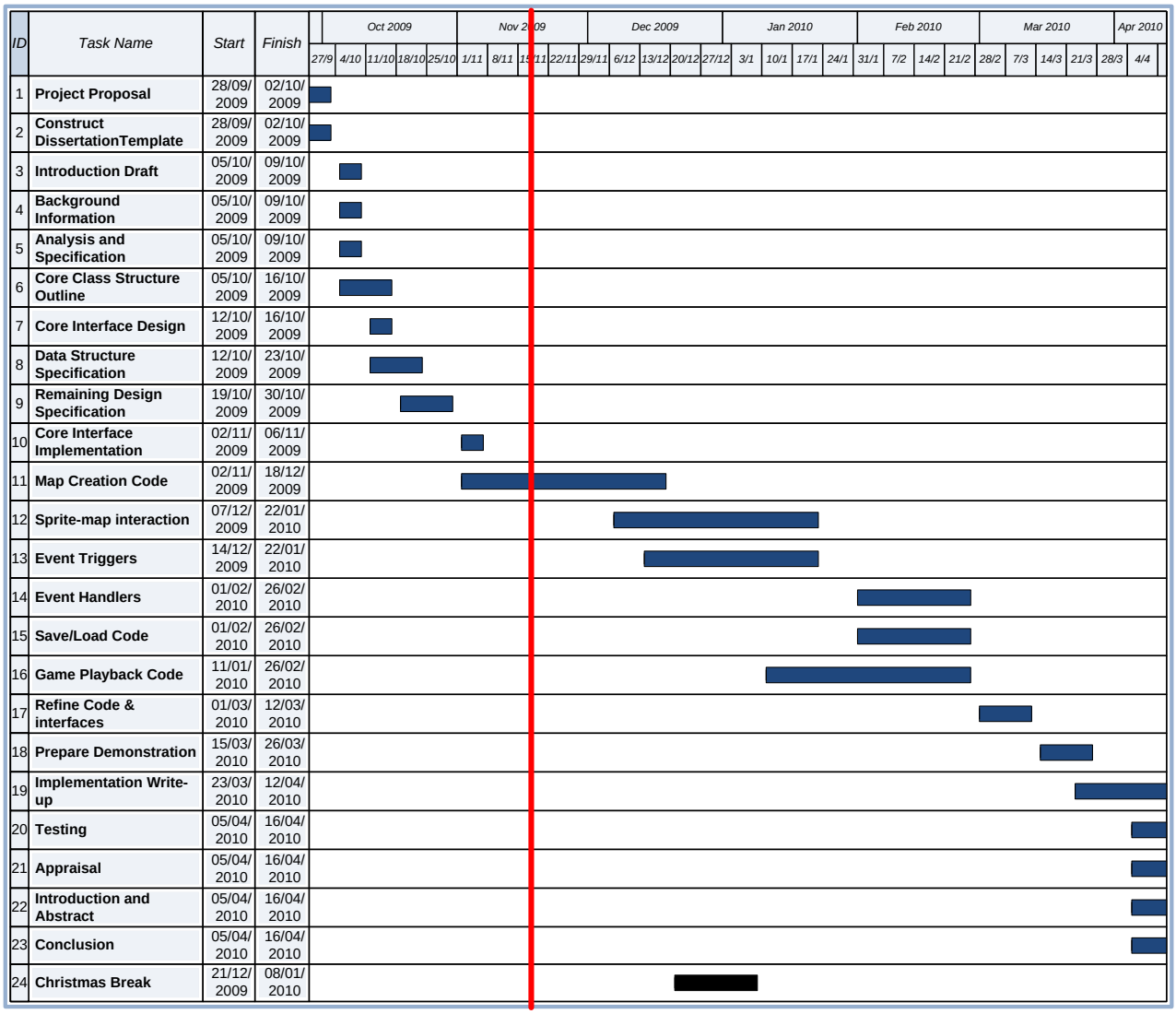
ID	Task Name	Start	Finish
1	Project Proposal	28/09/2009	02/10/2009
2	Construct DissertationTemplate	28/09/2009	02/10/2009
3	Introduction Draft	05/10/2009	09/10/2009
4	Background Information	05/10/2009	09/10/2009
5	Analysis and Specification	05/10/2009	09/10/2009
6	Core Class Structure Outline	05/10/2009	16/10/2009
7	Core Interface Design	12/10/2009	16/10/2009
8	Data Structure Specification	12/10/2009	23/10/2009
9	Remaining Design Specification	19/10/2009	30/10/2009
10	Core Interface Implementation	02/11/2009	06/11/2009
11	Map Creation Code	02/11/2009	13/11/2009
12	Sprite-map interaction	02/11/2009	20/11/2009
13	Event Triggers	16/11/2009	27/11/2009
14	Event Handlers	23/11/2009	18/12/2009
15	Save/Load Code	02/11/2009	18/12/2009
16	Map Music Player	11/01/2010	22/01/2010
17	Game Playback Code	09/11/2009	29/01/2010
18	Refine Code & interfaces	25/01/2010	29/01/2010
19	Independent Game Player	01/02/2010	05/02/2010
20	Testing	08/02/2010	12/02/2010
21	Appraisal	15/02/2010	19/02/2010
22	Introduction and Abstract	15/02/2010	26/02/2010
23	Conclusion	15/02/2010	26/02/2010
24	Christmas Break	21/12/2009	08/01/2010

Careful consideration of the projects key features in the design proposal stage of the project lead to the identification that the original Gantt chart outlined in the proposal (the above chart) was not an accurate representation of how the project would actually be completed.

The Gantt chart required rework to ensure that the goals that had been set were actually achievable. To do this some non-core components were removed from the development plan such as removing the requirement of having a music player that plays music dependent upon which map the sprite is on.

It was also identified that some areas of the project could not be completed in isolation of other areas. For example the play section of the game would require a saved game file to be loaded into the game playback functions. This meant that any new events that were created would also have to be integrated into the save/load mechanism in order for them to be tested and played.

Revised Gantt chart as of 01/11/2009 (note: the red line represents the revision date):



ID	Task Name	Start	Finish
1	Project Proposal	28/09/2009	02/10/2009
2	Construct DissertationTemplate	28/09/2009	02/10/2009
3	Introduction Draft	05/10/2009	09/10/2009
4	Background Information	05/10/2009	09/10/2009
5	Analysis and Specification	05/10/2009	09/10/2009
6	Core Class Structure Outline	05/10/2009	16/10/2009
7	Core Interface Design	12/10/2009	16/10/2009
8	Data Structure Specification	12/10/2009	23/10/2009
9	Remaining Design Specification	19/10/2009	30/10/2009
10	Core Interface Implementation	02/11/2009	06/11/2009
11	Map Creation Code	02/11/2009	18/12/2009
12	Sprite-map interaction	07/12/2009	22/01/2010
13	Event Triggers	14/12/2009	22/01/2010
14	Event Handlers	01/02/2010	26/02/2010
15	Save/Load Code	01/02/2010	26/02/2010
16	Game Playback Code	11/01/2010	26/02/2010
17	Refine Code & interfaces	01/03/2010	12/03/2010
18	Prepare Demonstration	15/03/2010	26/03/2010
19	Implementation Write-up	23/03/2010	12/04/2010
20	Testing	05/04/2010	16/04/2010
21	Appraisal	05/04/2010	16/04/2010
22	Introduction and Abstract	05/04/2010	16/04/2010
23	Conclusion	05/04/2010	16/04/2010
24	Christmas Break	21/12/2009	08/01/2010

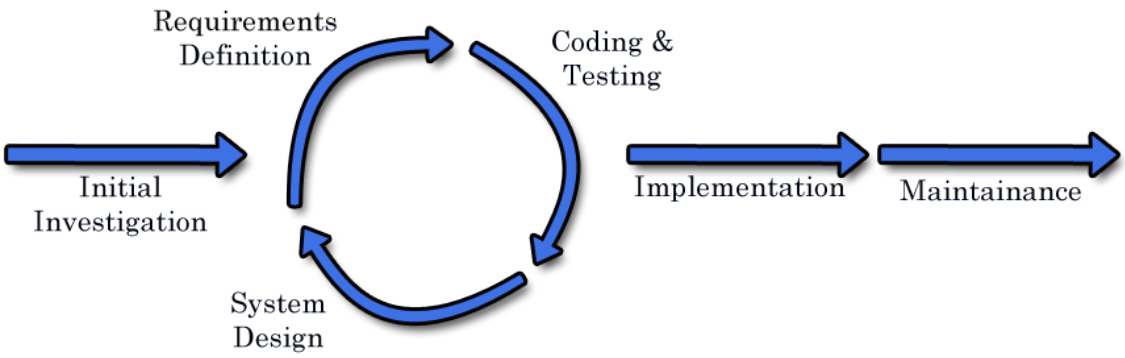
As you can see the 'event handlers', 'save/load code' and 'game playback code' sections have been revised to be completed in conjunction with each other. This revision was a much more accurate representation of how the project was actually implemented.

Design Methodology

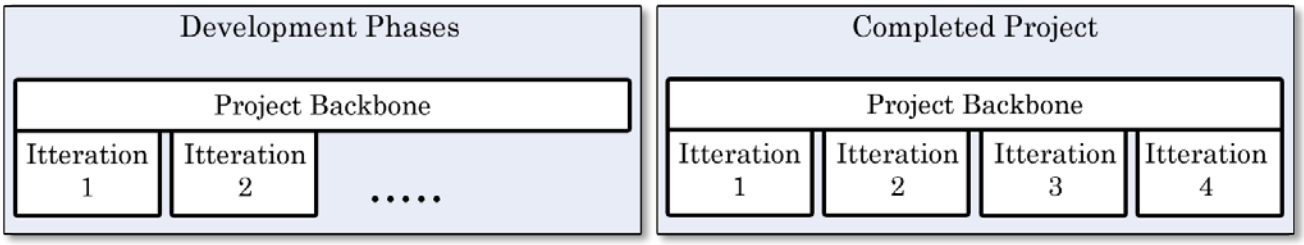
The software design process throughout the project followed an agile design pattern. Agile design encompasses a range of possible iterative development processes, the process that I chose to apply was the prototyping development process (CMS 2008). The main "backbone" requirements of the project were outlined early on, after this each project component was more precisely defined and changed at

the beginning of each design increment. These elements of the project were created in sections which could then join onto the main backbone of the project once completed. Each time an iteration was completed the software could be used and tested as a partially complete system.

Prototyping Design Methodology

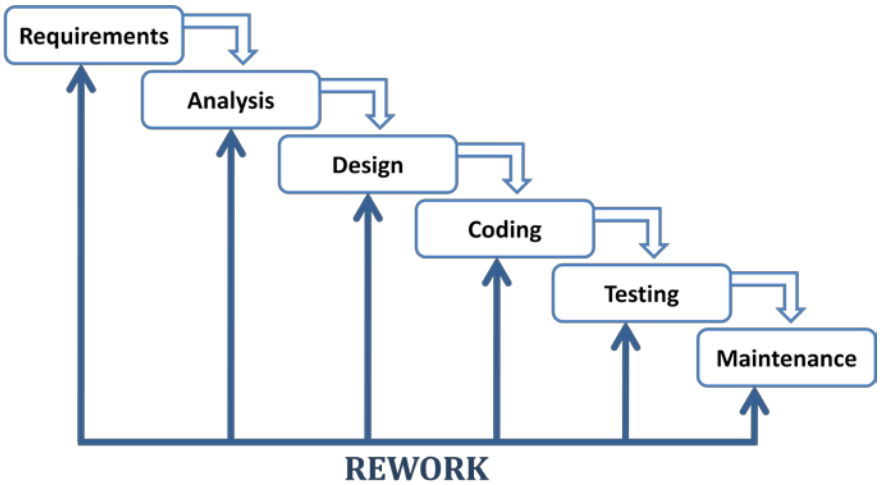


Interpretation of the CMS prototype diagram (CMS 2008).



This methodology was most suitable for my project because it facilitated for change. There were stages in the implementation where a prototype iteration had highlighted that part of the system needed unexpected modification. For example, the issues discussed earlier regarding improving playback smoothness through double buffering and the replacement of an in-game items system with in-game variables.

An alternate design methodology that I could have used is the waterfall model (Jackson, Introduction to Project Management - Project Genesis 2001):



If a ridged and non-iterative development process such as the waterfall model had been applied then the need for such changes may not have been identified until the overall project testing phase. These changes could be expensive in terms of time to implement due to their late identification. This test

phase is at the end of the project and other key features may have already been integrated with the piece of code that now needed to be changed.

A limitation of the prototyping process is that for large systems the process of documenting each design iteration can be lengthy. It can be difficult to manage because there is the risk that members of the design team will complete lots of poorly designed iterations with limited documentation. This was not a problem in this project because I was the only person involved in its development. I was able to effectively manage this through setting very clear goals of what the project should achieve and when as illustrated in the previous gantt chart.

The relatively small scale of the project as a whole(In comparison to larger corporate projects)also meant that each iteration was small. Consequently the requirements and design stages of each iteration could happen quickly without the need to keep track of progress by writing expansive documentation each time.

I wanted to use the prototyping model because my project was ambitious. The prototyping model facilitates for frequent iterations of usable software to be created. By using this model if I was unable to complete the project due to unforeseen circumstances then I would at least be able to submit and demonstrate an iteration of the software. This was a risk reduction strategy (Jackson, An Introduction to Project Management - Managing Risk 2009) that fortunately did not need to be utilised.

Appraisal

I am pleased that the outcome of the project has been successful. With the exception of events being triggered automatically when the sprite walks on them, I have met all of the core requirements that were outlined in the specification. I was also able to develop the extension requirement to enable events to be conditional on the state of an in-game variable system. The software is cohesive and its structure allows for the remaining extension requirements to be integrated into the system if more development time was available.

One area for concern regarding the project is its relatively heavy use of memory. It is recommended that the java environment is allocated 512Mb of heap space. This is due to the images that are used as maps being stored in memory. The PNG file format is compressed, however the java image format represents images in memory as a bitmap. To reduce the memory footprint of the software, a memory management mechanism could be implemented to remove any map images from memory that are not currently being edited or played. This would reduce memory consumption, however it could have an impact on performance as the program would need to load the images of a map into memory from the hard drive every time a new map is viewed.

When demonstrating the software to people, they seem to be immediately impressed by the professional look and feel of the projects user interface. The interface is aesthetically pleasing and the interface is intuitive as a result of taking into consideration various human computer interaction heuristics. I am happy that the complex underlying code can be interacted with quickly using a concise set of easy to use tools.

The project works on Windows, Mac and Linux systems with the java 1.6 runtime environment installed however I have chosen to only officially support Windows systems. This is because the majority of the testing has been executed on this operating system. Even though the testing has proved the system to be robust on Windows, I do not want to make the statement that the software is completely robust on the other operating systems without having done expansive testing on these systems. I would have liked to have done this testing if more time was available.

Playback performance was a major issue in the development of the project. It was an unforeseen problem that I was unsure how to fix when it arose. Removing the use of the JScrollPane and implementing my own map scrolling components in combination with double buffering was an effective solution to this problem. Play back of a created game is now a much more enjoyable experience as a consequence of this smoother image rendering process. I am happy that I effectively managed my time. I was able to ensure the project did not slip behind schedule as a result of this issue even though it took several days of unexpected research and development time to resolve.

Integrating third party libraries for the interface and zip file creation saved time overall however development decisions had to be made to cater for limitations within these libraries. This was a positive learning experience that would not have encountered if I had chosen to recreate the functionality used in these libraries myself.

The final software is robust, performs well and is easy to use. It is a complete system with an advanced set of simple to use functions. The addition of an in game variable system permits a much broader genre of games to be created using the software than just the originally anticipated role playing games. I illustrated this by using the software to make a quiz for the project demonstration.

Conclusion

The completed software is an easy to use game making program that meets the original core specification. The decisions outlined in the design section of the project were implemented in the final software and only one element of the specification needed to be excluded at the implementation phase.

The project has surpassed expectation as the range of game genres that can be created in it are more than just role playing games. The depth of the software could further be improved by increasing the amount of event types available to created, for example the ability to show and hide foreground items. The software has been designed to facilitate for this extension in the future.

The ability could have been added to enable each map to play music. This would help to make games created feel more immersive. A new map layer could have been added below the sprite layer that worked in the same way to the foreground layer but at a lower level. This layer would store images that could then be shown/hidden via a new event to make it appear like items had been picked up.

The memory management of the project is the largest issue with the final software and has resulted in the final program using a large amount of memory. A function to unload images within maps from memory that aren't currently being viewed or edited would solve the issue. If future development were to happen then this is the first issue that should be resolved.

Currently when a created game is played, it is first saved as a .game file and then a new game object is created that is used by the game playback mechanism. This function is very slow as it requires all of the games images to be saved and then reloaded into memory from the hard drive. If the system system had been implemented to only load the images into memory as they were needed, this save/load process would also be faster.

The creation and capture of exceptions has made the software easier to debug and ensures that any errors thrown are appropriately handled. Unexpected errors in the save and load code were identified early as in order for a game to be played, the .game file and it's corresponding XML document had to be exported and loaded into the playing module. The software is more robust as a result of this design decision.

Choosing to use an iterative project development lifecycle proved to be a good decision as problems and required changes were identified throughout the development of the project. The implementation decisions that were not outlined in the design were able to be made at the beginning of each development iteration. This meant that any core changes to the program (such as replacing the JScrollPane) were able to be implemented without major rework to the system designs being required.

The project was ambitious. It is the largest software project that I have ever had to create and it has taught me a lot about project management. I have had to structure my time and set strict objectives. Learning how to research new knowledge that is directly applicable to the task at hand is a valuable skill that I have improved on throughout this project. When the issue with graphics playback performance presented itself I had to do a lot of on the spot research regarding how to fix the problem. Learning how to manage and structure such a amount of code sections was a valuable outcome of the project as the project is made up of over fifty java classes.

I am pleased with the final outcome of both the software project and the report. I have achieved a great deal in a relatively small amount of time.

References and Bibliography

Adobe. *Adobe Photoshop.* http://www.adobe.com/products/photoshop/photoshop/ (accessed 04 08, 2010).

BBvode.org. *BBCode tags reference.* http://www.bbcode.org/reference.php (accessed 04 03, 2010).

CMS. *SELECTING A DEVELOPMENT APPROACH.* 27 03 2008. http://www.cms.hhs.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf (accessed 04 01, 2010).

crumhorn, emil. *Ribbon Widget.* 2009. http://hexapixel.com/projects/ribbon (accessed 10 16, 2009).

Dozal, Philip. *What Is NTSC & PAL?* http://www.ehow.com/about_5116964_ntsc-pal.html (accessed 03 29, 2010).

FINK. *Package jdom-1:1.0-2.* 18 09 2009. http://pdb.finkproject.org/pdb/package.php/jdom (accessed 03 29, 2010).

Flamingo. *Flamingo license.* 2008. https://flamingo.dev.java.net/license.html (accessed 03 23, 2010).

Future Invent Informationsmanagement. *Small Open Source Java Tools and Libraries .* 2009. http://www.fuin.org/utils4j/index.html (accessed 03 29, 2010).

GNU. *GNU LESSER GENERAL PUBLIC LICENSE.* 06 17 2009. http://www.gnu.org/licenses/lgpl.html (accessed 03 29, 2010).

Greanier, Todd. *Discover the secrets of the Java Serialization API.* 07 2000 . http://java.sun.com/developer/technicalArticles/Programming/serialization/ (accessed 03 22, 2010).

Horstmann, Cay. "XML Tags and Documents." In *Big Java*, by Cay Horstmann, 962-963. Stempel Garamond, 2008.

IconsPedia. *Your Source of Free Icons.* http://www.iconspedia.com (accessed 04 05, 2010).

Jackson, Joan. *An Introduction to Project Management - Managing Risk.* Birmingham: The University of Birmingham, 2009.

—. "Introduction to Project Management - Project Genesis." *University of Birmingham School of Computer Science.* 29 09 2001. http://www.cs.bham.ac.uk/~jxj/2009_2010/Lectures/Projectgenesis.pdf (accessed 04 11, 2010).

java.net. *Flamingo Swing component suite.* 2008. https://flamingo.dev.java.net/ (accessed 10 18, 2009).

JDOM. *JDOM Binaries.* http://www.jdom.org/downloads/index.html (accessed 03 29, 2010).

Microsoft. *Ribbons.* http://msdn.microsoft.com/en-us/library/cc872782.aspx (accessed 03 23, 2010).

—. *Save a file.* http://office.microsoft.com/en-us/word/HP012330331033.aspx (accessed 03 20, 2010).

Mozilla. *Keyboard shortcuts .* 15 12 2009. http://support.mozilla.com/en-US/kb/Keyboard+shortcuts (accessed 03 30, 2010).

Nielsen, Jakob. *Ten Usability Heuristics.* 2005. http://www.useit.com/papers/heuristic/heuristic_list.html (accessed 03 22, 2010).

Norman, Donald A. *The Design of Everyday Things.* Basic Books, 2002.

NTC Hosting. *SQL.* http://www.ntchosting.com/databases/structured-query-language.html (accessed 03 22, 2010).

Oracle Corporation. *Writing JUnit Tests in NetBeans IDE.* http://netbeans.org/kb/docs/java/junit-intro.html (accessed 04 2010, 05).

Reallusion Inc. *iClone4 EX Download - Experience iClone4 FREE with No Expiration!* 18 01 2010. http://www.reallusion.com/iclone/iclone_trial.asp (accessed 03 2010, 30).

Roelofs, Greg. *Portable Network Graphics (PNG) Specification, W3C/ISO/IEC version.* 03 02 2008. http://www.libpng.org/pub/png/spec/iso/ (accessed 03 22, 2010).

Shneiderman, Ben. *Designing the user interface: Strategies for effective human-computer interaction.* Vol. 3. Addison-Wesley, 1998.

Spell, Brett. *Setting Heap Sizes.* 29 03 1999. http://www.devx.com/tips/Tip/5578 (accessed 04 05, 2010).

Sun Microsystems. *Double Buffering and Page Flipping.* 12 01 2010. http://java.sun.com/docs/books/tutorial/extra/fullscreen/doublebuf.html (accessed 03 25, 2010).

—. *Glossary.* 2001. http://java.sun.com/products/jlf/ed2/book/HIG.Glossary.html (accessed 03 23, 2010).

—. *Model-View-Controller Architecture .* 2002. http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html (accessed 03 24, 2010).

—. *Source Editor.* 2010. http://netbeans.org/features/ide/editor.html (accessed 03 20, 2010).

W3C. *Extensible Markup Language (XML).* 14 03 2010. http://www.w3.org/XML/ (accessed 03 22, 2010).

—. *JPEG JFIF.* 13 02 1996 . http://www.w3.org/Graphics/JPEG/ (accessed 03 22, 2010).

w3schools. *Browser Display Statistics.* 2009. http://www.w3schools.com/browsers/browsers_display.asp (accessed 10 12, 2009).

Wikipedia. *Comma-separated values.* 19 03 2010. http://en.wikipedia.org/wiki/Comma-separated_values (accessed 03 22, 2010).

—. *ZIP (file format).* 18 03 2010. http://en.wikipedia.org/wiki/ZIP_%28file_format%29 (accessed 03 2010, 22).

Appendices

HCI heuristics

Shneiderman's Principles of Human-Computer Interface Design (Shneiderman 1998):

1. Strive for consistency
2. Enable frequent users to use shortcuts
3. Offer informative feedback
4. Design dialogs to yield closure
5. Offer error prevention and simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load

Norman's 7 Principles (Norman 2002):

1. Use both knowledge in the world and knowledge in the head.
2. Simplify the structure of tasks.
3. Make things visible: bridge the gulfs of Execution and Evaluation.
4. Get the mappings right.
5. Exploit the power of constraints, both natural and artificial.
6. Design for error.
7. When all else fails, standardize.

Nielsen Usability Heuristics (Nielsen 2005):

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

JPEG recompression.

Example of how image quality is lost.

Original Image

Image saved 100 times using JPEG compression

W3C Screen Resolution Statistics (w3schools 2009)

The current trend is that most computers are using a screen size of 1024x768 pixels or more:

Date	Higher	1024x768	800x600	640x480	Unknown
January 2009	57%	36%	4%	0%	3%
January 2008	38%	48%	8%	0%	6%
January 2007	26%	54%	14%	0%	6%
January 2006	17%	57%	20%	0%	6%
January 2005	12%	53%	30%	0%	5%
January 2004	10%	47%	37%	1%	5%
January 2003	6%	40%	47%	2%	5%
January 2002	6%	34%	52%	3%	5%
January 2001	5%	29%	55%	6%	5%
January 2000	4%	25%	56%	11%	4%

Flamingo license - Berkeley Software Distribution (BSD) License (Flamingo 2008)

Copyright (c) 2005-2008 Flamingo, Kirill Grouchnikov. All Rights Reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

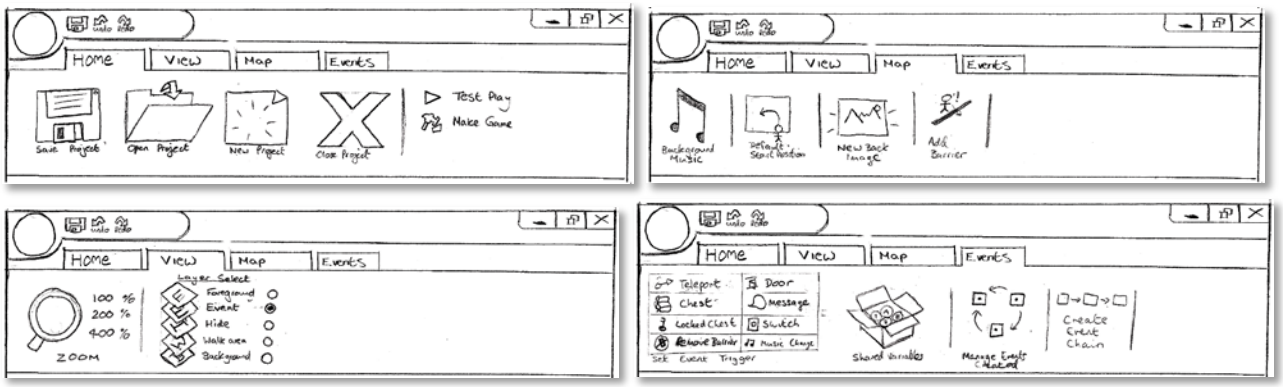
- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of Kirill Grouchnikov nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

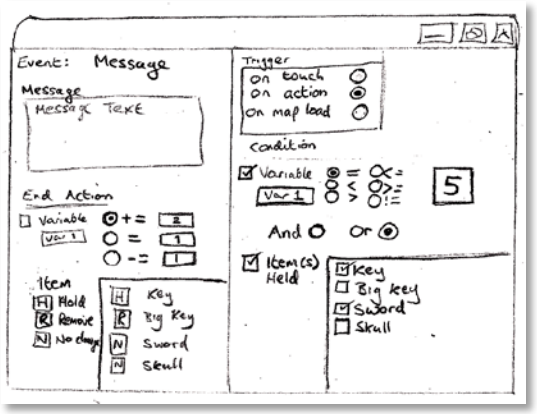
User Interface Mock-up

Below is a provisional mock-up of how the interface used by my program should look:

Ribbon States:



Creation of conditional events:



Icon References (IconsPedia n.d.)

Icon	Required source or licence reference	Icon	Required source or licence reference
	www.pixel-mixer.com		Creative Commons Attribution-Non-commercial
	GNU Lesser General Public License		GNU Lesser General Public License
	GNOME icon artists GNU General Public License, version 2		GNOME icon artists GNU General Public License, version 2
	IconsPedia.com		Paul Gucolav Free For Personal Use

Any icons not referenced here are either free for non-commercial use from iconspedia.com or were custom made by me as part of the project.

BBCode

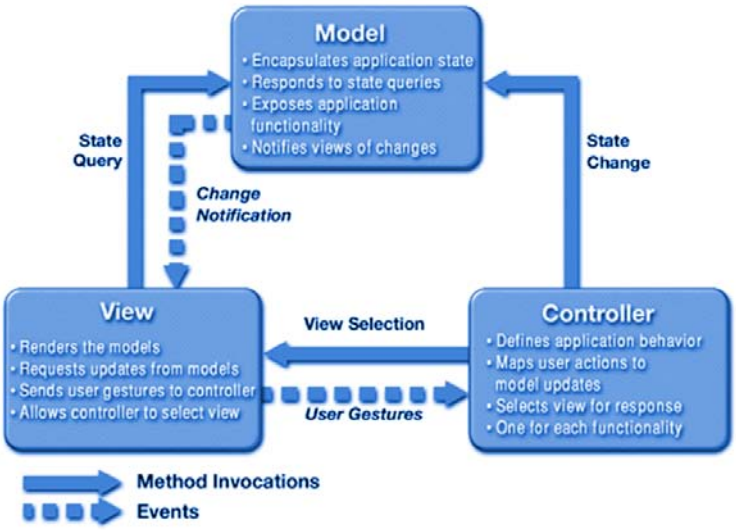
BBcode is a system for replacing tags with html code in forums. It is beneficial because it gives the forum programmer the ability to allow users to format their posts. Validation can be applied to the BBCode tags to prevent malicious code being injected into the forum. Also BB code tags are simple to use and easy to remember. For example the bbcode '[url]google.com[/url]' would be replaced with 'google.com' when parsed (BBvode.org n.d.).

XML entity attributes reference:

Entity	Attribute	Variable	Representation
Location	-	-	This entity represents the start location of the sprite for when the game is played.
Sprite	-	-	XML representation of the in-game sprite.
	mapid	Integer	The id of the map that the spite will start on.
	xaxis	Integer	The x coordinate start position of the sprite.
	yaxis	Integer	The y coordinate start position of the sprite.
appearance	-		The appearance entity is intended to reference the folder of images that will be used to represent how the sprite looks. In my application I

			didn't use this as only one sprite was made, however it could be used if the project was extended.
	ref	String	The name folder of sprite images to use. This Folder must contain the sub folders: 'N', 'NE', 'E', 'SE', 'S', SW', 'W', 'NW' and each folder must contain the same amount of images.
variables	-		Stores the starting state of the in game variables.
variable	index	Integer	The index of the variable.
	value	Integer	The starting value of the variable.
gamemap	-		This entity represents the state of an in game map.
perspective	front	Double	The relative perspective of the sprite at the front of the map. This number is a percentage.
	back	Double	The relative perspective of the sprite at the back of the map. This number is a percentage.
foregrounditem	img	String	The name including file extension of the image used as the foreground item.
	xaxis	Integer	The x coordinate of the top left hand corner of the image.
	yaxis	Integer	The y coordinate of the top left hand corner of the image.
Background	img	String	The name including file extension of the image used as the background for the map.
walkover	-		This element contains all of the areas that make up the walkover area of the map.
area	-		This element contains points that make up the area. All stored areas are assumed to be completed therefore the last point connects to the first point.
point	index	Integer	The index of the point within the area. e.g. index 0 is the first point.
	xaxis	Integer	The x coordinate of the point.
	yaxis	Integer	The y coordinate of the point.
events	-		This entity contains all of the events on the map.
Event	-	String	The character data contained within this entity tag is the message to be displayed. This is only the case if the event type is 'message' or 'variable'
event	type	String	The type of event. e.g. 'message', 'teleport', 'variablechange'.
	trigger	String	How the event will be triggered. e.g. 'walkover', 'action'.
eventcondition	v1	Integer	The index of the in game variable to be compared against.
	v2	Integer	This can either be used as a reference to a second variable index to be compared against or as a value. the attribute 'varorval' dictates this.
	varorval	String	Can only contain the string 'variable' or 'value'. Dictates whether the v2 attribute references a variable index or an absolute value.
	operation	Integer	The index of the variable operation to be used. 0 = '>', 1 = '<', 2 = '==', 3 = '>=', 4 = '<=', 5 = '!=', -1= unset
	enables	String	'true' if the event condition is enabled and therefore must be satisfied. 'false' if the event condition is not enabled and is not required to be satisfied.
eventshape	shape	String	'area', 'square' or 'circle'.
position	-	-	This entity is only present if the attribute 'shape' parent entity 'eventshape' is not 'area'.
	width	Double	The width of the shape.
	height	Double	The height of the shape.
	xaxis	Double	The x coordinate position of the top left hand corner of the shape.
	yaxis	Double	The y coordinate position of the top left hand corner of the shape.
operations	-	-	This entity is only present if the 'type' attribute of the parent element 'event' is 'variablechange'.
variable	index	Integer	The index of the variable to modify.
	value	Integer	The value to use as an operator.
	operation	String	Must contain one of the strings: "add", "subtract" or "set". This attribute dictates how the attribute 'value' will be applied to the variable defined by the attribute 'index'. e.g. if index = 0, value = '2' and operation = 'add' then every time the event is triggered 2 will be added to the variable at index 0.

Model-View-Controller Design Structure



(Sun Microsystems 2002)

Resize foreground items algorithm

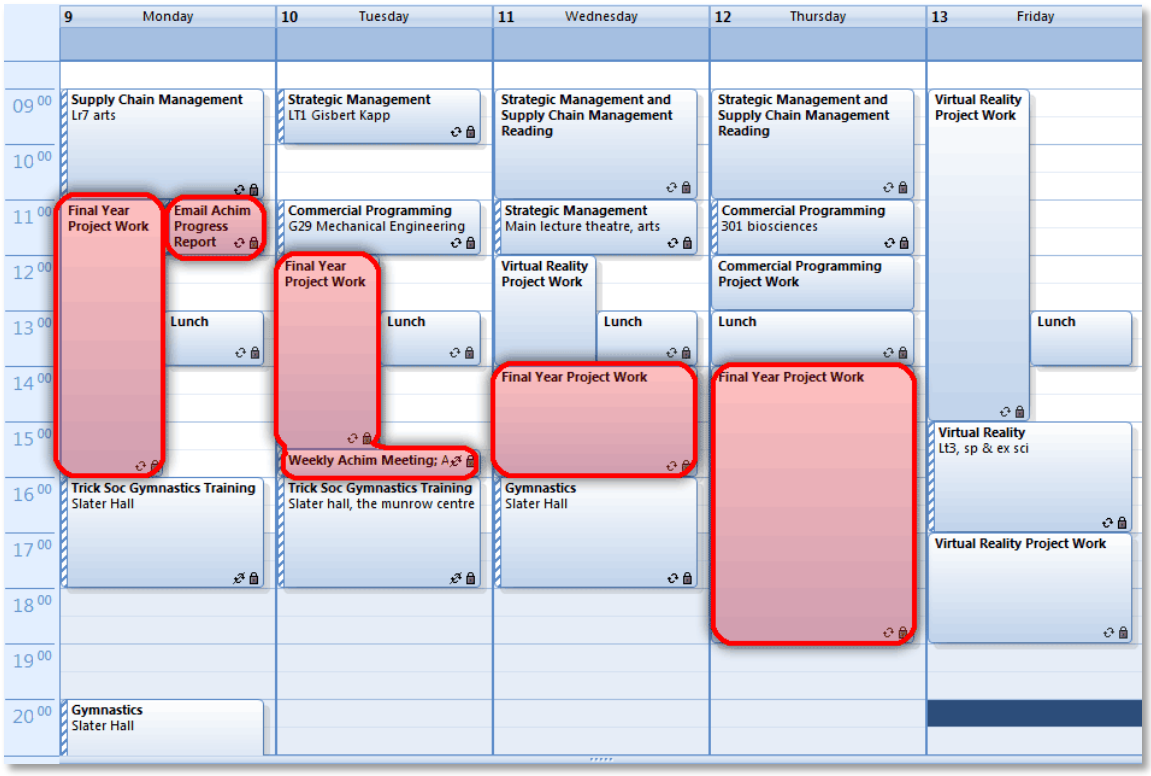
```
//the distance that the mouse has been dragged
Width Change = to X coordinate - from X coordinate
Height Change = to Y coordinate - from Y coordinate

//a possible  new width of the image based on the mouse drag
Total Width = Starting width of image + Width Change;
Total Height = Starting height of image + Height Change;

//logical top left hand coordinate of the new image
minX = top left hand X coordinate of the image
minY = top left hand Y coordinate of the image

//by default the new width is the distance that the mouse has
//been dragged
width = Width Change;
height = Height Change;


//Resizers 0 and 1 are both on the same y axis
if Resizer Index == 0 || Resizer Index == 1{

        if top left hand Y coordinate of image
          + Height Change
          < bottom right hand Y coordinate of image{

                //top left hand corner of the image is set to
                the where the mouse was dragged to
                minY = to Y coordinate;

        }else{

                //top left hand corner of the image is set to
                the bottom right hand corner of the original
                image
                minY = bottom right hand corner Y
                        coordinate of the original image;

        }

        height = height of the original image - Height Change

}else{

        If Total Height < 0 {
                minY = top left hand y coordinate of image
                        - Total Height;
        }
                height = Total Height;
        }
}
```

```
//Resizers 0 and 2 are both on the same X axis
if Resizer Index == 0 || Resizer Index == 2{

        if top left hand X coordinate of image + Width Change
           < bottom right hand X coordinate of image{

                //top left hand corner of the image is set to
                //the where the mouse was dragged to
                minX = to X coordinate;

        }else{

                //top left hand corner of the image is set to
                // the bottom right hand corner of the
                // original image
                minX = bottom right hand corner X
                        coordinate of the original image

        }

        width = Width of the original image - Width Change;

}else{

        If Total Height < 0 {
                minY = top left hand X coordinate of image
                        -  Total Width;
        }
         width = Total Width;


}


Now set the new image to be drawn from the coordinate
(minX , minY)
Also set the new image dimensions to the newly calculated
width and height.
```

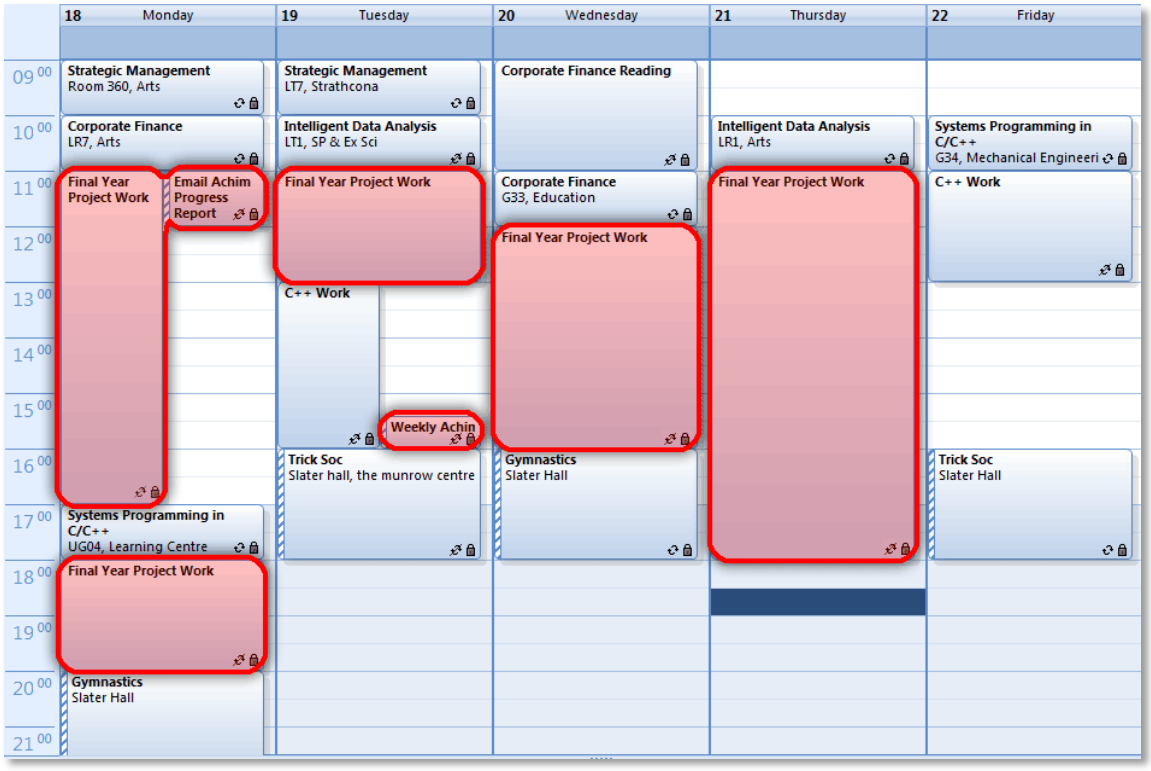
Personal Timetable

These are screen shots from my outlook timetables to illustrate how I planned my final year project work alongside my other responsibilities. Mondays and Thursdays were the days where I would get the most significant amount of project work completed.

Semester 1 Timetable



Semester 2 Timetable



Data CD Structure

This section describes the structure of the data CD provided alongside this project. The CD contains 4 folders labelled: 'Diagrams and Images', 'Documents', 'Runnable' and 'Source Code'.

Diagrams and Images Folder

This folder contains all the images, diagrams and production files for all of the images and diagrams contained within the project dissertation.

Documents Folder

The project proposal (**Project Proposal v1.1.docx**), project demonstration presentation (**Demonstration.docx**) and final project dissertation (**Project Documentation v1.4.docx**) can be found in the 'Documents' folder. The proposal and project dissertation are in both word 2007 (.docx) and adobe PDF (.pdf) formats. The demonstration document is in both PowerPoint 2007 (.pptx) and adobe PDF (.pdf) formats.

Runnable Folder

This contains all of the files needed to play the game. See the 'Running The Project' appendix for further details.

Source Code Folder

This folder contains all of the source code, files and libraries used when making the game. The Game maker folder within the source code folder is in a format that can be loaded directly into Netbeans as a project.

The text file 'Contact Details.txt' contains my contact details. 'Using the Program.pdf' details how to use the game making interface and how to control the game when in play mode.

Running The Program

Insert the CD accompanying this project and Copy the contents of the 'Runnable' to the hard drive of a computer running Windows XP, Windows Vista or Windows 7. The hard drive must have both read and write permissions. The minimum system requirements to run the software are as follows:

- Java 1.6 JRE or higher must be installed.
- Intel Core 2 Duo 1.6Ghz processor.
- 2Gb DDR2 RAM.
- Integrated graphics.
- 1280x1024 monitor resolution.

The project may run on lower power computers however it might not perform optimally.

To run the project: Open the 'Game Maker 1.8' folder that was copied to the hard drive and double click on 'Game_Maker.jar'.

Alternatively you can run the project via the command prompt. To do this navigate to the 'Game Maker 1.8' folder and run this command: java -Xms256m -Xmx512m -jar Game_Maker.jar
To ensure the right amount of memory has been allocated for the program, it is advised that the command prompt method of running the software is used. The program will crash if not allocated enough memory.

Pre-made game examples can be found in the folder: 'Game Maker 1.8/files/gamesaves'. An installer for the Java 1.6 JRE can be found in the root directory of the CD and is named ' jxpiinstall.exe', an internet connection is required to install. Details of the user interface can be found in the 'Interface' section of the report.