Requirements and Analysis Document for the Psycho Hero game project (RAD)

Contents

Version: 1.01

Date: 2013-05-22

Author: Carl Jansson, Erik Tholén, Peter Eliasson & Simon Persson

This version overrides all previous versions.

1 Introduction

This section aims to give a brief overview of the project.

1.1 Purpose of application

The project aims to create a 2D based action adventure game with a top down perspective, fun story, simple controls and a lot of action.

1.2 General characteristics of application

The application will be a desktop, standalone 2D topdown game for Windows/Mac/Linux. The user will be able to control a main character, the “hero”, with both the keyboard and a mouse. The hero can walk around in static rooms. In these static rooms there are obstacles and different kinds of enemies that is constantly opposing the hero and tries to harm him.

When pressing a chosen set of keys on the keyboard, the hero will move in the desired direction. The mouse is used for rotating the hero and aiming with some weapon that the hero can use to kill his enemies. Note that the movement of the hero has nothing to do with the direction he is facing.

The user can walk through doors with the hero to get to another room. In one of the existing rooms there is a boss and when the hero has accomplished the goal of killing this boss, the game is over and the user wins.

1.3 Scope of application

The application will not be a full game but rather a demo for how a full scale game could be. Gameplay and design shall be consistent with how it would be if we would have had more time to make a complete game.

1.4 Objectives and success criteria of the project

Our goal is to make a application that we can proudly present for friends and family as something we have achieved this year.

The application itself should be an entertaining game that has a high replay value. This will be achieved with simple controls, simple graphics and a fun background story that captures the interest of the user. A time based high score system will also capture the more competitive gamers.

1.5 Definitions, acronyms and abbreviations

- 2D based top down perspective: This is different from a 2.5D perspective in that the graphics displays the world from a straight-down perspective instead of slightly from the side.
- GUI: Graphical User Interface.
- IDE: Integrated Development Environment.
- Host: The computer where the game will run.

- The Hero: The fictional protagonist in the game.
- Java: The programming language used. Java is platform independent.
- Jar: The extension of a java program packaged into a runnable file (e.g game.jar).

2 Requirements

2.1 Functional requirements

The player should be able to;

1. Start a new game
2. Choose to play the game in fullscreen
3. Disable sound effects and music
4. Set his own key bindings
5. Control the movement of the hero
6. Interact with enemies
    a. The user will have the option to kill or defeat the enemies with various weapons
    b. Be defeated by enemies
7. Win the game
8. Show a high score of previous games
9. Exit the application

2.2 Non-functional requirements

This section briefly describes requirements that are not related to any specific function of the game.

2.2.1 Usability

The game should be very easy to understand and control, but hard to win. Due to a limited number of maps (or levels) the difficulty might be somewhat easy once the user is familiar with the game. The full game should present a real challenge for the user with enemies that are very hard to kill.
The menus should be self-explanatory and follow the standard formula for games, providing options such as start/resume, options and exit game. There should also be an option to see previous high scores.

2.2.2 Reliability

N/A.

2.2.3 Performance

Like every other game, performance is a critical point where the actual performance may vary across different systems. The goal is however not to keep the player waiting but to provide a smooth and enjoyable experience locked at a maximum of 120 frames per second.

2.2.4 Supportability

Our target platform will be PC where we aim to support the three major operating systems, Windows, Linux and Mac OS. The game is not suitable for mobile platforms, and therefore we can make a GUI that is adjusted for bigger screens. In the future it might be relevant to make console versions of the game.

The architecture used for the code should make it easy to expand on the game with more levels and enemies.

2.2.5 Implementation

The application will use the Java language with Eclipse as the IDE. Git will be used for version control and source code management.

2.2.6 Packaging and installation

The application will be delivered in a zip archive containing a runnable Java .jar file and a resource file containing the graphics and sound. No special scripts will be necessary to run on different platforms since this is included in the jar. In order to run the game the host must have the JRE installed and configured.

2.2.7 Legal

The game is a non-profit project simply intended for learning. All the graphics and audio will be self-made. However, some of the third party libraries we are using are licensed under various open source licenses. They all allow distribution as long as a copyright notice is included, which will be including in a licence.txt file packaged with our software.

2.3 Application models

2.3.1 Use case model

See APPENDIX for UML and use cases.

2.3.2 Use cases priority

1. Move
2. Aim
3. Attack
4. Take Damage
5. Game Over
6. Game Won
7. Enemy Movement
8. Menu
9. Key Bindings
10. Highscore

2.3.3 Analysis model

See APPENDIX.

2.3.4 User Interface

The application should be able to run on all somewhat new computers. Everything from ultrabooks with a 1366x768 resolution to full hd 1920x1080 screens. No special effort will be made for the game to work on higher resolution screens.

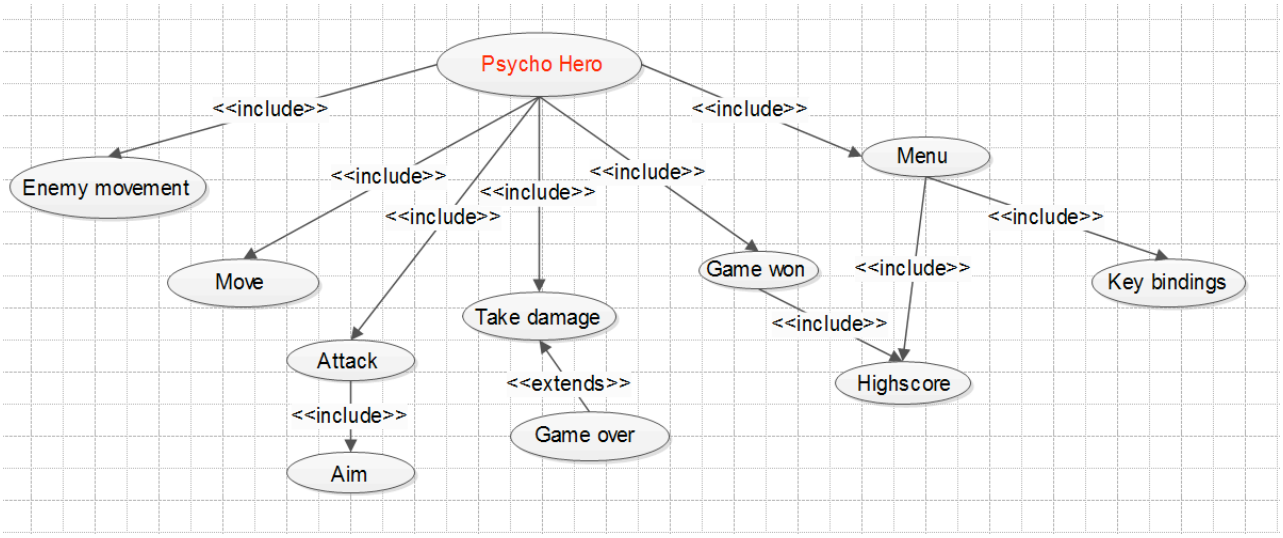
2.4 References

Top down: http://en.wikipedia.org/wiki/Top-down_perspective#Top-down_perspective

Appendix

Use cases

Overview.



Use case texts

Use case: Move

Summary: The user can walk around in a two dimensional space with user defined keys on the keyboard.

Priority: High

Extends: -

Includes: -

Participators: The actual player

Normal flow of events

	Actor	System
1	Presses assigned button for desired direction	
2		Moves the hero towards the desired direction with an animation that makes the hero seem to walk “natural”
3	Releases the button	
4		Stops the movement and walking animation of the hero

Alternate flows

Flow 2.1 - Player walks into an obstacle

	Actor	System
2.1.1		Stops the movement of the hero, and the hero gets “stuck” next to the obstacle. The walking animation keeps playing

Flow 2.2 - Player walks into a door

	Actor	System
2.2.1		Switches to another room with new obstacles and enemies. The hero appears next to the door in the new room.

Use case: Aim

Summary: The player will be using the mouse to change the position of the crosshair which represents the point at where the hero is aiming.

Priority: High

Extends: -

Includes: -

Participators: The actual player

Normal flow of events

	Actor	System
1	Moves the crosshair using the mouse.	
2		The position of the crosshair is changed according to the movement of the mouse. The head of the hero is turned so that he faces the crosshair.

Use case: Attack

Summary: The player presses the attack key. The hero attacks with currently equipped weapon.

Priority: High

Extends: -

Includes: Aim

Participators: The actual player

Normal flow of events
Attack with a melee weapon in the direction the hero is aimed, hitting an enemy.

	Actor	System
1	Presses assigned button for attacking	
2		The hero attacks in the direction he is aiming
3		If there is an enemy in range, the enemy is damaged based on weapon equipped by hero

Use case: Take Damage

Summary: The hero takes damage from some source.

Priority: Medium

Extends: -

Includes: -

Participators: The application

Normal flow of events
Damage is inflicted on the hero.

	Actor	System
1	Enemy visibly hits hero with melee or ranged attack	
2		Certain amount of damage is inflicted on the hero. Amount of damage is reduced from hero life
3	Player sees visible indication of losing life points on screen	
4		If not enough HP to survive, Game Over

Use case: Game Over

Summary: The hero's health points reach zero.

Priority: Medium

Extends: Take Damage

Includes: -

Participators: The actual player

Normal flow of events
Score is shown and display option to retry or quit.

	Actor	System
1	Player loses too much life and "dies"	
2		System switch to a game over menu
3	Player sees a taunting window	
4		Waits for player to return to main menu

Use case: Game Won

Summary: The game is won and player should enter his/her name for potential high score

Priority: Medium

Extends: -

Includes: Highscore

Participators: The actual player

Normal flow of events
Enter a name and shift to high score.

	Actor	System
1	Wins the game	
2		Changes display to congratulate the player and give option to enter name
3	Enters name or leaves textfield blank, then presses the enter key	
4		Shows Highscore

Use case: Enemy Movement

Summary: Enemies move around on the screen

Priority: Medium

Extends: -

Includes: -

Participators: Application

Normal flow of events

	Actor	System
1	NPC waits short random time	

2		Calculate a new random position based on a predefined box that the enemy can walk in
3		Make a path based on new position
4	NPC starts walking along path	

Alternate flows

Flow 2.1 - NPC walks into another dynamic object

	Actor	System
2.1.1	NPC collides with another dynamic object	
2.1.2		Current path is set to null and a new one will be calculated

Flow 2.2 - NPC is in line of sight with hero

	Actor	System
2.2.1	No static object is between NPC and hero	
2.2.1		Makes a new path based on hero position
2.2.1	Starts walking new path against hero	

Use case: Menu

Summary: When the game is started you should be taken to main menu

Priority: Low

Extends: None

Includes: Key Bindings, Highscore

Participators: The actual player

Normal flow of events

	Actor	System
1	User double click on the jar file in which the game is packaged.	
2		Main method runs and program initiates
3	User sees main menu.	
4		Waits for further commands

Alternate flows

Flow 2.1 - Open the menu while playing

	Actor	System
2.1.1	User presses the esc button on keyboard	
2.1.2		Game gets paused
2.1.3		Main menu loads, with resume option included
2.1.4	User sees main menu.	
2.1.5		Waits for further commands

Use case: Key Bindings

Summary: The user should be able to alter which keys on the keyboard that are assigned to different actions in the game (like walking with the hero).

Priority: Low

Extends: -

Includes: -

Participators: One user

Normal flow of events

	Actor	System
1	Clicks on “Options” in the game menu	
2		Options menu is displayed
3	Clicks on “Key Bindings”	
4		Displays all key bindings
5	Clicks on the character next to the game action that he or she wants to change the key binding for	
6		Replaces the clicked character with “---” and starts listening for a new key to assign
7	Presses new key to assign to choosen action	
8		Binds new key to the chosen action

Alternate flows

Flow 2.1 - Go back to main menu from options

2.1.1	Clicks on “Options” in the game menu	
2.1.2		Options is displayed
2.1.3	Clicks on the backbutton or presses esc key on keyboard	
2.1.4		Displays the game menu again

Use case: Highscore

Summary: A list of highscores is displayed.

Priority: Low

Extends: -

Includes: -

Participators: The application

Normal flow of events

A list of the highest scores is displayed after user wins the game.

	Actor	System
1	Use Case: GameWon	
2		A list of high scores is displayed, and if good enough, the new score is included in the list

Alternate flows

Flow 2.1 - Highscore accessed by using menu

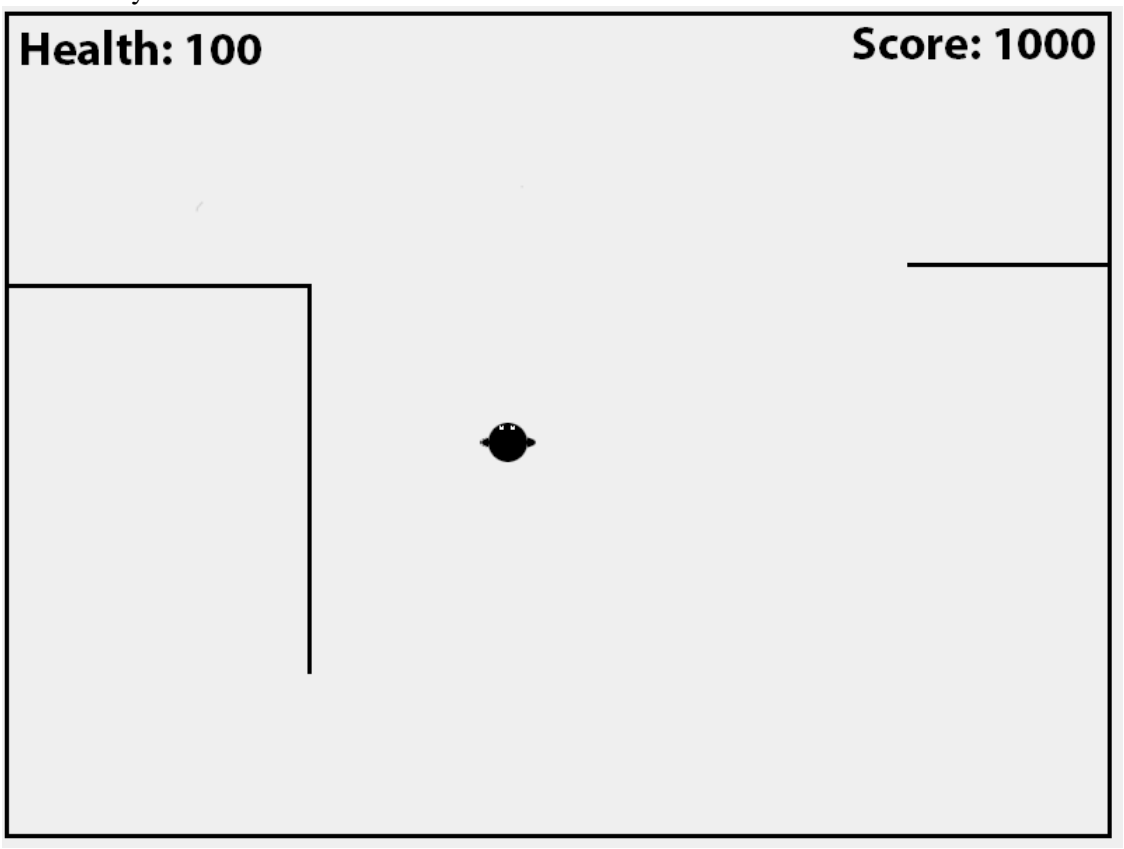
	Actor	System
2.1.1	Clicks on “High Score” in the game menu	
2.1.2		The list of high scores is displayed

Flow 2.2 - There are no previous scores

	Actor	System
2.2.1		A placeholder list of empty-scores is displayed

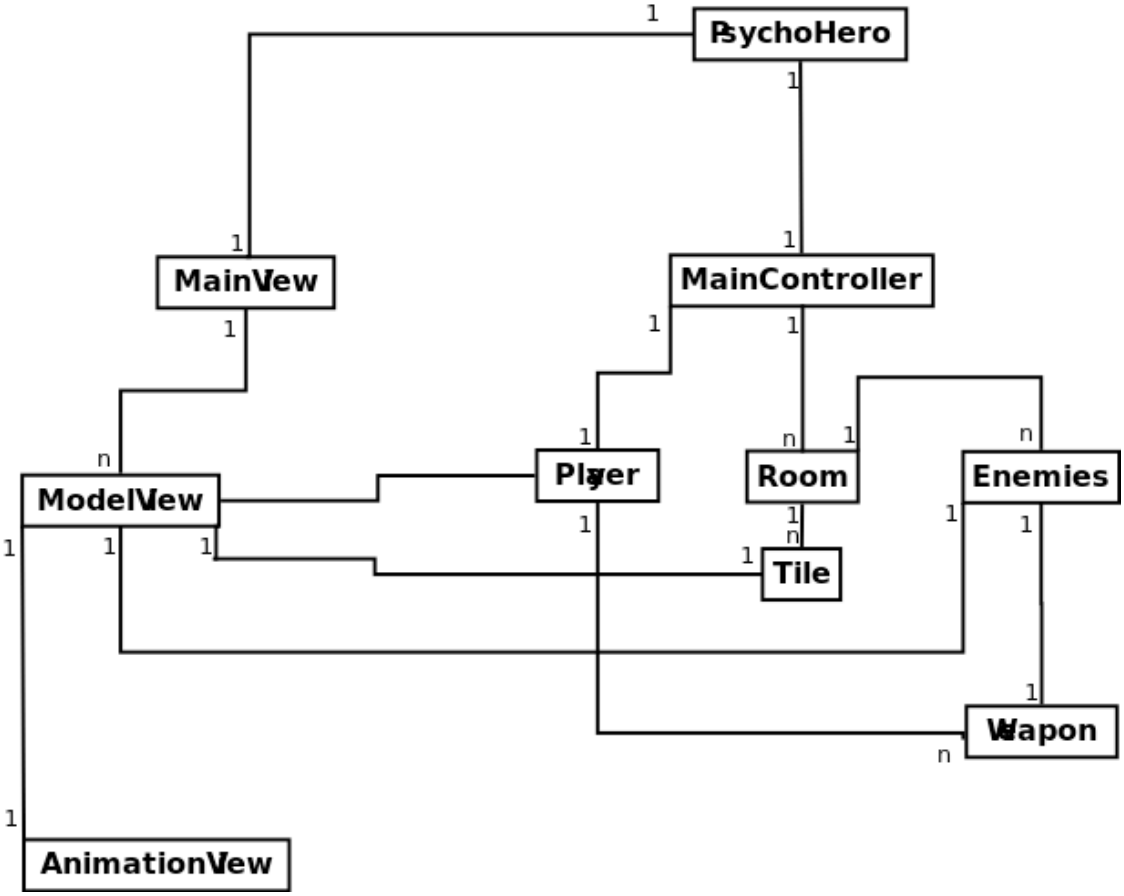
GUI

Preliminary GUI.



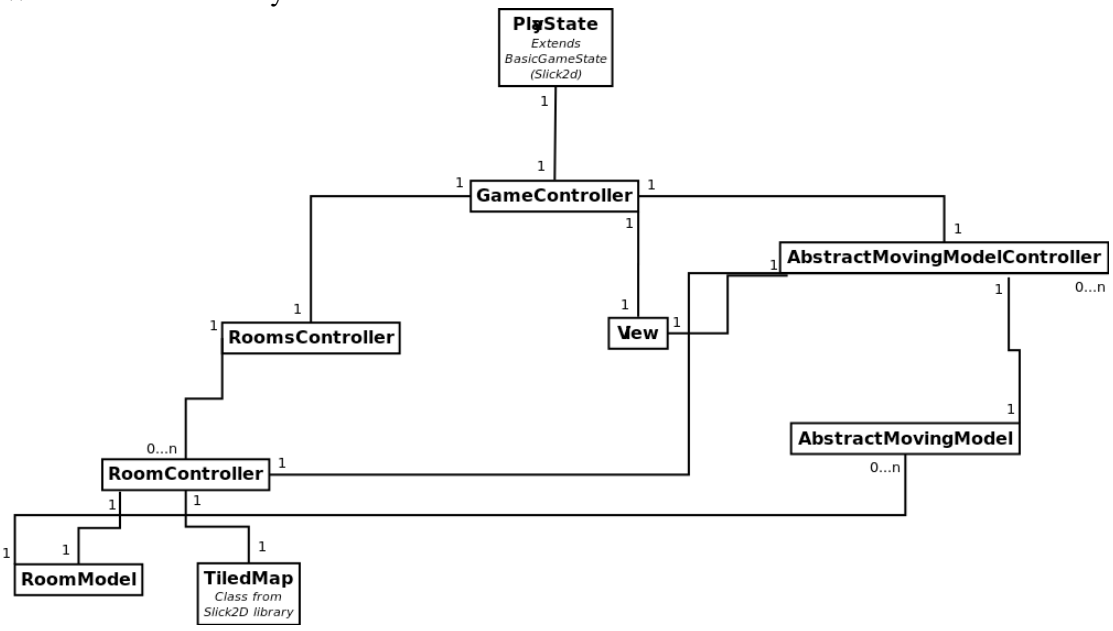
Analysis model

The first preliminary analysis model.



Final result

What the model finally looked like:



The final model looks a lot like the first preliminary one. However, a few things have changed:

- Many of the interfaces have become abstract classes instead, to avoid duplicated code as all moving models in the game have similar logic.

- Most models and rooms now has it's own controller. This is for dividing the game logic into smaller, more managable and understandable pieces. Thanks to this, the game is a lot easier to extend with new functionality, new rooms and new models.

- There is no MainView taking care of all other views in the game. Instead, every model has it's own corresponding view and both the model and the view is managed by a model specific controller.