Fructus Victus

WENTWORTH INSTITUTE OF TECHNOLOGY

COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

SOFTWARE DESIGN AND DEVELOPMENT – COMP 566

Adam Christensen, Steven Fortier, Tony Li, Bryan Quinlan, Zach Thornton

Table of Contents

1. INTRODUCTION

Fructus Victus is a procedurally generated first person exploration game, written in Java using LibGDX. Fructus Victus makes use of existing ray-casting rendering techniques in order to emulate the visuals seen in Wolfenstein 3D and Doom. The goal of Fructus Victus is exploration and survival through a labyrinth. What you discover and accomplish within Fructus Victus is up to the user, and the aim is to make an experience that is unique on each play through.

Upon pressing “New Game,” the user then finds themselves inside of a cavern with one instruction: "Find the source of the mutation!" On the wall in front of the user, instructions for movement and interaction are listed. As the user moves into subsequent rooms, avoiding traps and searching for clues, the game’s story begins to unfold. The world's fruit has begun to mutate, and it’s up to the user to search for the answer at the source.

1.1 Purpose of this Document

The purpose of creating this Software Requirements Specification (SRS) document is to describe key components and functionality of Fructus Victus.

1.2 Scope of the Development Project

Fructus Victus is a procedurally generated game which utilizes ray-casting to render a maze that users explore. The three key components of Fructus Victus are the ray-casting game engine, map generator, and input handler. The ray-casting game engine is the component responsible for calculating and rendering the pixels on the screen. The map generator creates randomized maps based on procedural generation algorithms, and the input handler manages key presses from the user, translating the commands to movements on the screen.

1.3 Acronyms, Abbreviations, and Definitions

1.3.1 Acronyms

2D - An environment that uses the X and Y axis to display in two dimensions.

3D - An environment that uses the X, Y, and Z axis to display in three dimensions.

JRE - Java Runtime Environment

SRS - Software Requirements Specification

1.3.2 Abbreviations

N/A

1.3.3 Definitions

Cellular Automata - An algorithm used to create cave-like maps. It is an algorithm that creates random arrays filled with numbers that represent either walls or floors. It then detects whether there are enough adjacent walls to create a wall, and removes all isolated wall segments. This leaves us with a map that has no disjoint sections.

Failure - A state where the software is not working as it was designed to function.

Game Data - Information that the software saves to allow the user to resume from the same state they last saved it from.

Item - An in-game object such as a trap or collectible which the player can interact with.

LibGDX - A Java game library.

Player - The in-game character that the user is controlling.

Ray-casting - A rendering technique which recomputes the user's visual frame whenever their coordinates or viewing direction changes. For each vertical line displayed on the screen, a ray is cast from the player's position until it collides with a wall. Once the collision has occurred, trigonometry is performed to determine the height of the wall, giving the illusion of a 3D environment.

User - The human controlling the keyboard and computer.

1.4 References

1. Basin, R. (n.d.). Cellular Automata Method for Generating Random Cave-Like Levels. Retrieved February 14, 2015, from http://www.roguebasin.com/index.php?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels

2. Chollak, D. (n.d.). Libgdx/libgdx. Retrieved February 3, 2015, from https://github.com/libgdx/libgdx/wiki/Gradle-and-Eclipse

3. id Software LLC. (1993). DOOM [video game]. GT Interactive.

4. id Software LLC. (1992). Wolfenstein 3D [video game]. Apogee Software.

5. Lode's Computer Graphics Tutorial. (2007, June 17). Retrieved January 20, 2015, from http://lodev.org/cgtutor/

6. Loftis, H. (n.d.). PlayfulJS. Retrieved February 3, 2015, from http://www.playfuljs.com/a-first-person-engine-in-265-lines/

7. Permadi, F. (n.d.). Ray-Casting Tutorial For Game Development And Other Purposes. Retrieved February 3, 2015, from http://www.permadi.com/tutorial/raycast/index.html

8. Procedural Content Generation Wiki. (n.d.). Retrieved February 3, 2015, from http://pcg.wikidot.com/pcg-algorithm:map-generation

1.5 Overview of Document

2. General Description
    - 2.1. User Characteristics
      A description of the average user. Contains basic skills the user is required to have when accessing our software.
    - 2.2. Product Perspective

This section contains information outlining why our product is unique and what it has to offer to any potential user.

2.3. Overview of Functional Requirements
Outline of all intended functions of the game. There will be at least 1 method dedicated to each of these. These intended functions are everything from basic movement to map generation.

2.4. Overview of Data Requirements
This section details what data will be created and stored by the software. This includes game data, maps, and player location.

2.5. General Constraints, Assumptions, Dependencies, Guidelines

2.1. General Constraints
These are the basic hardware requirements such as minimum RAM and supported graphics cards.

2.2. Assumptions
These are the assumptions made about the user of Fructus Victus. These are simple but necessary to play the game.

2.3. Dependencies
What Fructus Victus is dependent upon in order to run.

2.4. Guidelines
This section states what operating systems Victus runs on, as well as information on multiplayer and gameplay.

2.6. User view of Product Use
This section outlines what the user's initial experience with the game will be. This does not delve into gameplay, but rather sticks to what the user sees, such as what menu options are available to them.

3. Specific Requirements

3.1. External Interface Requirements
This section provides an overview of expected input and output from the system.

3.2. Detailed Description of Functional Requirements
3.1. Template for describing functional requirements
3.2. Through 3.2.x - The description of each functional requirement.

3.3. Performance Requirements
Contains all necessary specifications required of the user's system in order to run Frucus Victus, as well as the game's standard resolution.

3.4. Quality Attributes
How Fructus Victus delivers usability, reliability, performance, and supportability.

4. Other Requirements

Includes any other requirements not specified. This includes use case diagrams.

5. Charts/Diagrams

This is where you will find various diagrams and charts explaining the relationships between entities of Fructus Victus.

1. Hierarchy Diagram
2. Class Diagram
3. User Sequence Diagram
4. Game Sequence Diagram

2. GENERAL DESCRIPTION

Fructus Victus is an exploration game which utilizes ray-casting for rendering, and procedural generation algorithms for map creation.

2.1 User Characteristics

Fructus Victus contains no violence and is targeted to players age 5+. The game requires that the user to know how to use a keyboard, and has hardware meeting the minimum requirements specified in section 2.5.1.

2.2 Product Perspective

Fructus Victus is a standalone product, with no competitors. Since ray-casting is an outdated technique, there are few games being developed that still use it. Fructus Victus will be the first game which is both ray-casted and utilizes procedural world generation.

2.3 Overview of Functional Requirements

The functional requirements expand upon how the renderer works, how the map generator functions, and how the input control manager handles input from the user. These all require an input which later leads to an output rendered to the screen.

2.4 Overview of Data Requirements

The game will log the user's progress using states, and can be paused and exited at any time without losing data. If the game is exited due to a crash or other unexpected failure, the user will be able to reload the game and lose at most the last 5 minutes of play-time. Maps and save files will be saved to the disk. Input from the keyboard is required to play the game.

2.5 General Constraints, Assumptions, Dependencies, Guidelines

2.5.1 General Constraints

We require a computer with a minimum of a Intel GMA 4500MHD, 4GB RAM, and a 2.0GHz Core 2 Duo processor. The computer must also have a minimum of 200MB of free storage space to store the game.

2.5.2 Assumptions

It is assumed that the user has a functional keyboard, computer, and monitor. It is assumed that our user has a means to operate their keyboard. The user is required to have a display which can output resolutions greater than or equal to 1024x768.

2.5.3 Dependencies

Fructus Victus depends upon a computer with a working version of the JRE.

2.5.4 Guidelines

Fructus Victus must be playable on Linux, OSX, and Windows. It will contain no multiplayer components and must contain both procedurally generated levels as well as a progressable story-line. The renderer will be created using Ray-casting.

2.6 User View of Product Use

The user's first interaction with the game after launching is a view of the menu screen. This menu will contain options to start a new game, continue the previous game, change the game settings, or quit the game. If the user decides to start a new game their player is placed in a random, procedurally generated world in which the user is free to move the player around using the keyboard to interact with items which unravel parts of the story. If the user pauses the game, a menu will appear with an option to save their current position in the game, or to quit the game and return to their desktop. If the user decides to continue the previous game from the main menu, the player will continue from the most recent game save. The settings menu allows the user to change system settings such as key bindings and screen resolution. Quitting the game brings the user back to their desktop.

3. SPECIFIC REQUIREMENTS

3.1 External Interface Requirements

3.1.1 User Interfaces

- The Main Menu should contain menu items that are selectable by using only keyboard controls.
- When the user is in game they should only be able to access other menus from the pause screen.

3.2 Functional Requirements

3.2.1 Raycast Map Renderer renders image on screen.

3.2.2 Raycast Map Renderer handles x and y position changes, x and y direction changes, and x and y plane changes, from input control manager.

3.2.3 Map Generator generates map chunk.
3.2.4 Map Generator returns map height, map width, and the map to renderer.
3.2.5 Input Control Manager processes keypresses.
3.2.6 Input Control Manager updates user position.
3.2.7 Raycast Map Renderer renders items.
3.2.8 User interacts with item.

3.3 Detailed Description of Functional Requirements

3.3.1 The renderer class will take the map, map height, and map width from the map generator class and wall textures from the assets folder to render the image on the screen.

3.3.2 The renderer class will update its x and y position, x and y direction, and x and y plane data from the control class.

3.3.3 The map generator class will use the Cellular Automata method to generate a 2D array representing the map for the game.

3.3.4 The map generator class will pass the map, map height, and map width to the renderer class.

3.3.5 The control class will loop to read inputs from the user until the game state changes.

3.3.6 The control class will update the renderer class' x and y position, x and y direction, and x and y plane data.

3.3.7 The renderer class will take the item textures from the assets folder to render the item on the specified place in the map data.

3.3.8 The control class will put change the game state to a "Interacting" game state with a description of the item.

3.*4* Performance Requirements

The game runs at 60 frames per second assuming a computer with a minimum of an Intel GMA 4500MHD, 4GB RAM, and a 2.0GHz Core 2 Duo processor or equivalent. The JRE also needs to be installed.

3.*5* Quality Attributes

3.5.1 Usability: The game can be launched by the user without any prior knowledge. The user can change the settings menu by using text descriptions of each button. The game is intuitive to the point where the only instructions the user requires is the controls, the goal of the game, and player death triggers.

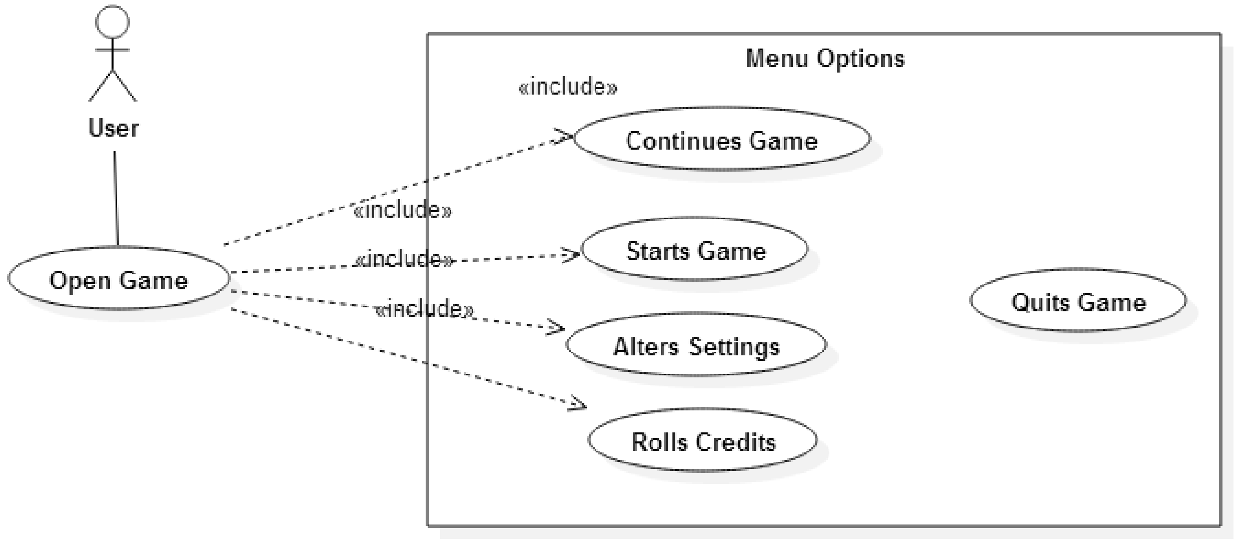
3.5.2 Reliability: The game is robust due to the game's priority of saving game data. In the event of a game crash, user has to restart the game and continue off the automatic saved game data. There may be a difference in game progress since the automatic save occurs every 5 minutes. If user quits the game halfway, the game will automatically save the data. In an event of a hardware failure, the user will have to fix their hardware before being able to continue with the game. Any key that is not defined in the control settings menu will be ignored by the game to prevent any errors. Since this game requires minimal ram, memory, and cpu usage, it will operate well under intense computer workloads.

3.5.3 Performance: The game starts quickly with minimal loading time. Scalability isn't an issue for this game since it is an offline game. This game is available 100% of the time running at 60 frames per second for the user as long as their hardware meets the minimum requirements stated in 2.5.1 to run this game. The game will run with a minimum video resolution of 1024x768. The game is available after installation.

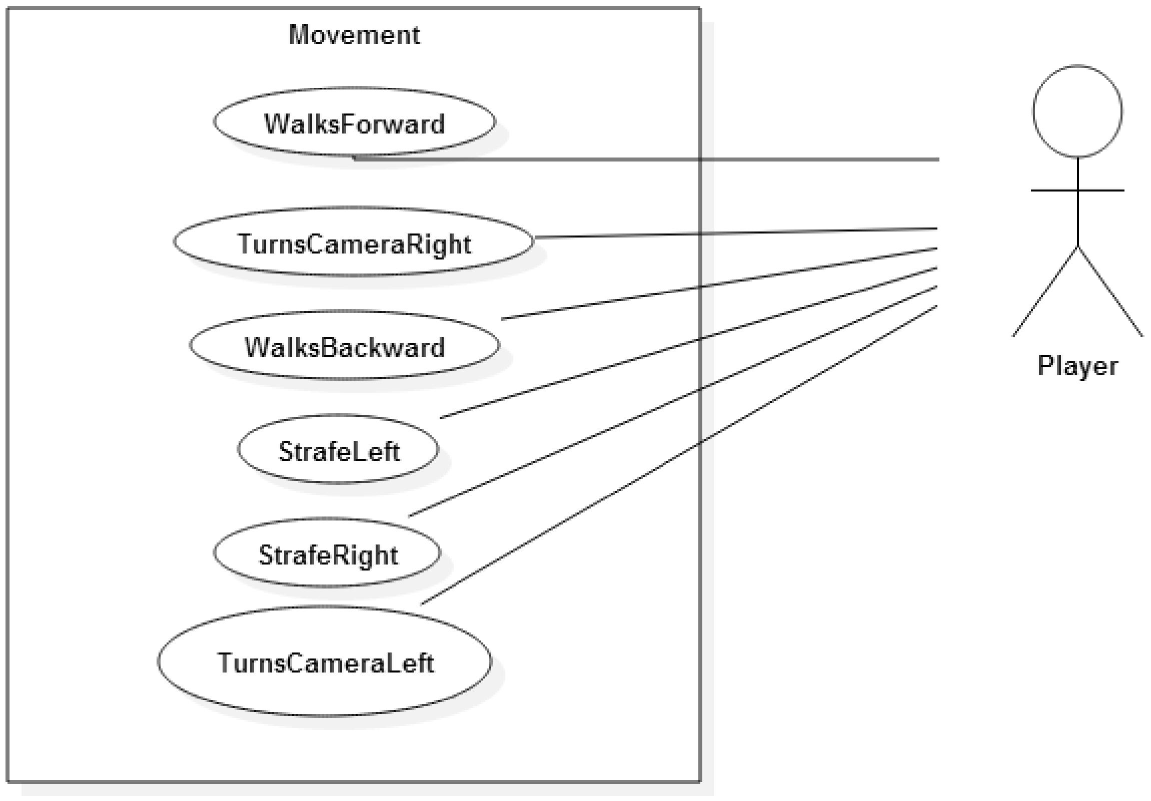
3.5.4 Supportability: The game will work on all versions of Windows, Linux, and OSX that support the JRE. The game will be open source, so community fixes will be accepted. Bugs may be reported to the Fructus Victus GitHub page under issues for the community to see.

4. OTHER REQUIREMENTS

4.1 User Game Menu Use Case Diagram

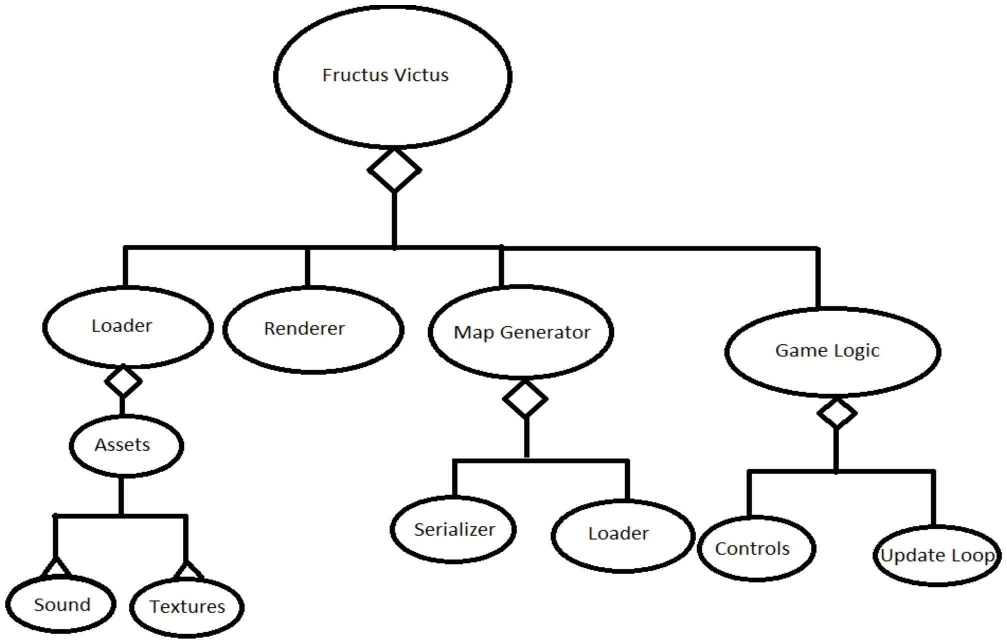


4.2 Player Control Use Case Diagram

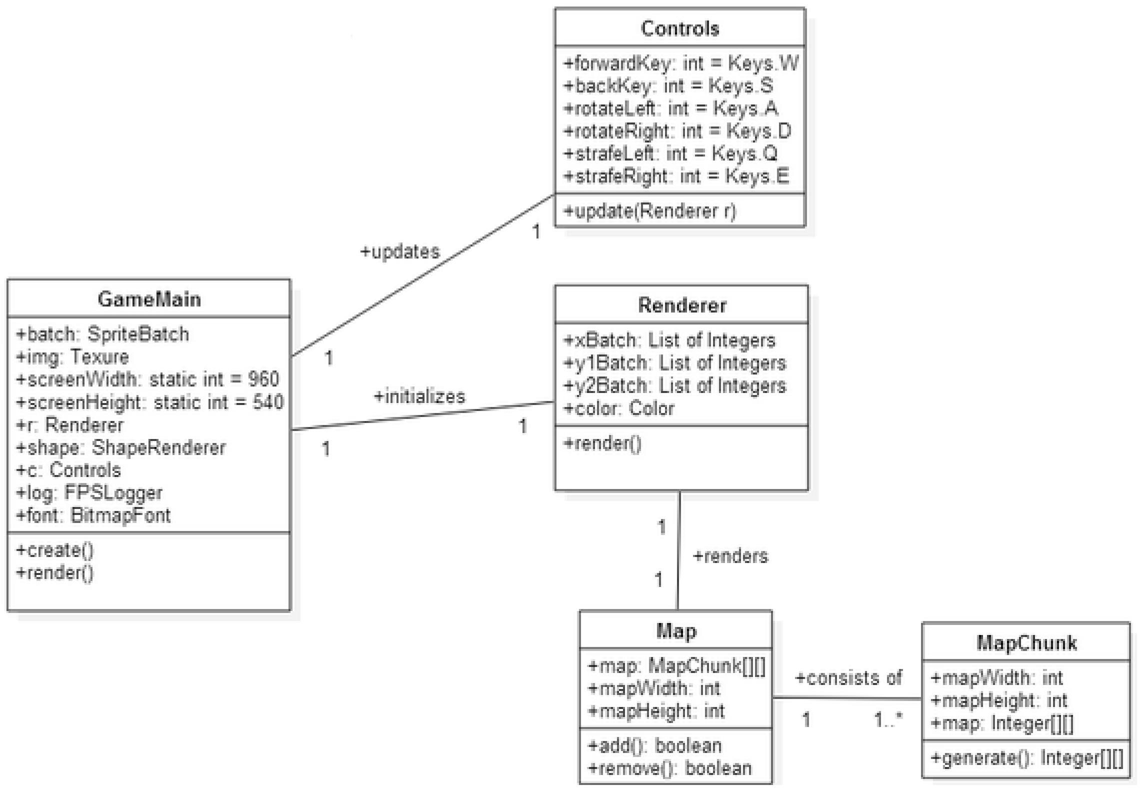


5. APPENDIX

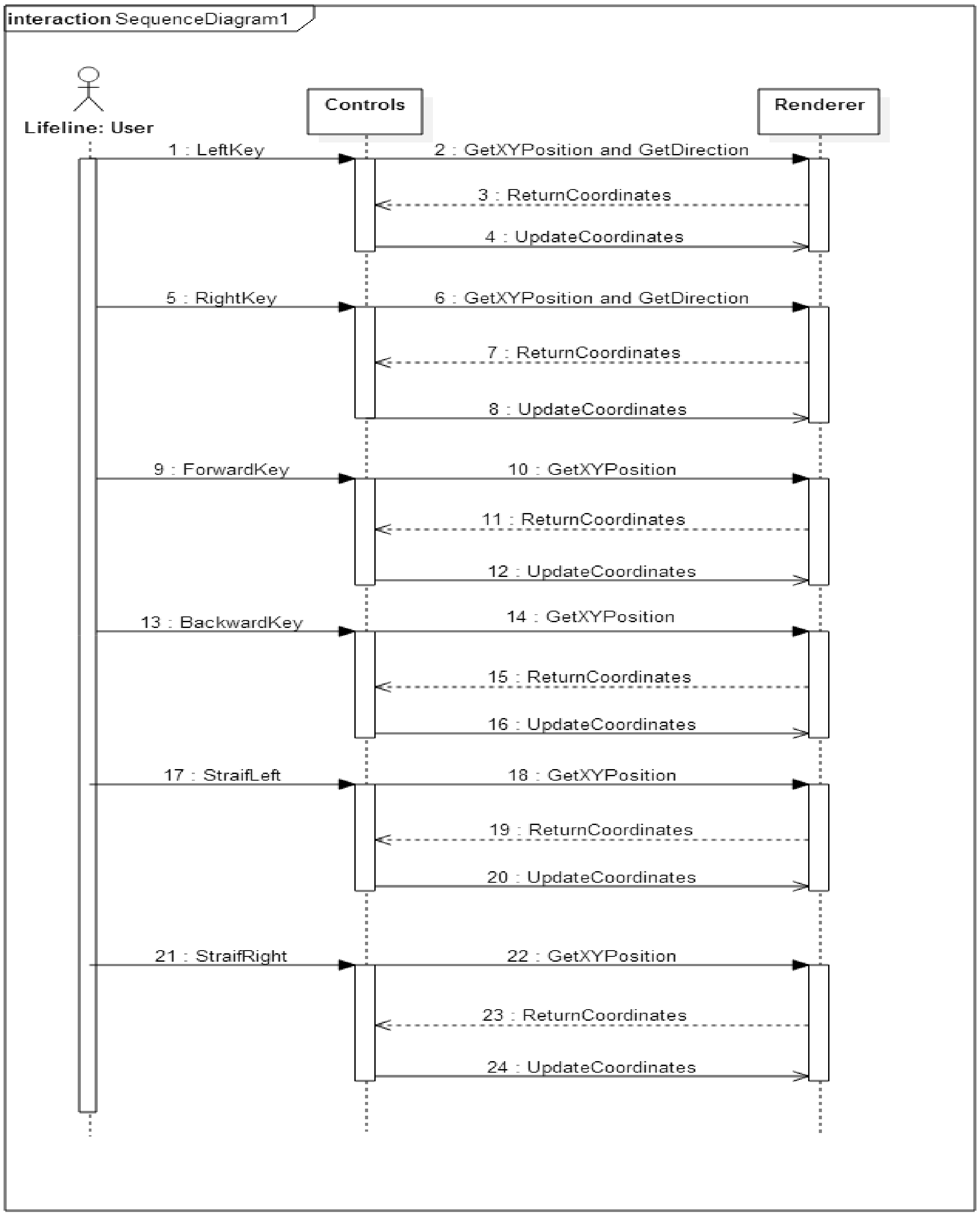
5.1 Hierarchy Diagram



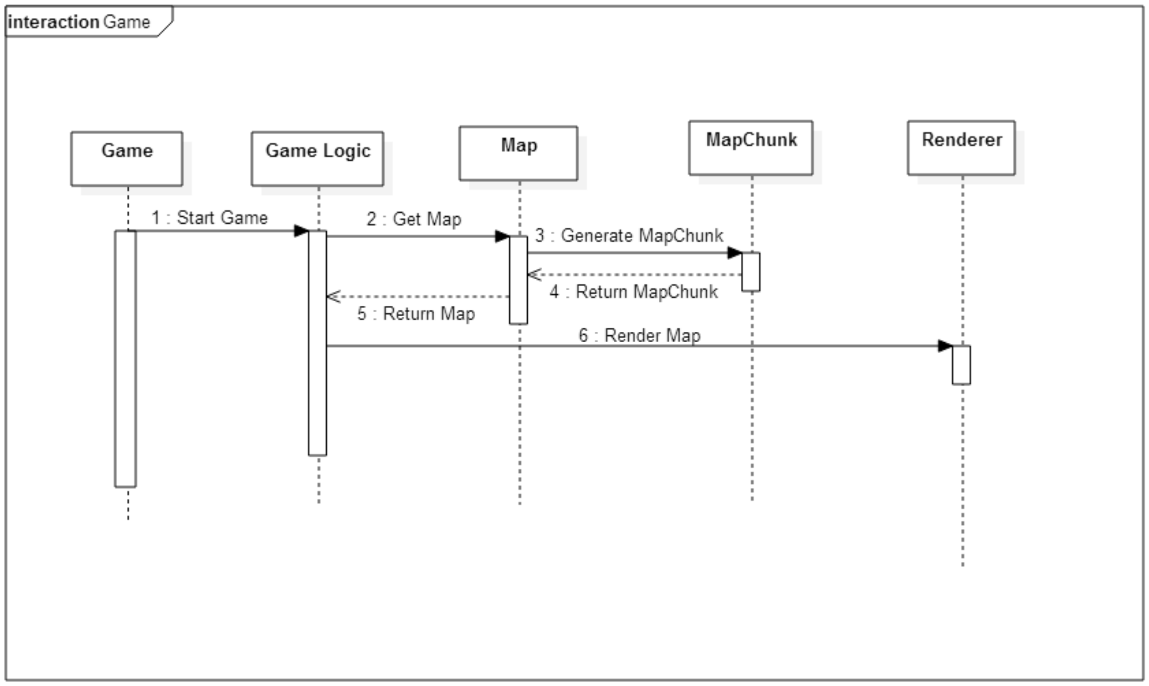
5.2 Class Diagram:



5.3 User Sequence Diagram



5.4 Game Sequence Diagram


interaction Game
Game
Game Logic
Map
MapChunk
Renderer
1 : Start Game
2 : Get Map
3 : Generate MapChunk
4 : Return MapChunk
5 : Return Map
6 : Render Map