***SOFTWARE DESIGN SPECIFICATION (SDS)***

***LABYRINTH***

*CREATED BY:*

**DANIEL MATHIEU**

**JOHN FOUAD**

**PATRICK BARRON**

**GABRIEL WOODSUM**

**CHRISTOPHER WONG**

**October 24, 2016, REV A**

**1. Introduction**

The goal of this project is to develop a multi-player, online maze game, Labyrinth. There will be a maximum of 4 players per game, each attempting to reach the same final destination before the other players. There will be obstacles in the way, spread out randomly throughout the maze that will attempt to prevent the users from reaching the destination. User will have the ability to run away from obstacles. User controls will be limited to moving around using WASD/arrow keys.

Different conditions have to be satisfied by the final schedule and which have been specified in the SRS.

This document specifies the final system design for the system. It also gives some explanations on how the design evolved and why some design decisions were taken.

**2. Design Considerations**

There are several issues that will affect the design of this project.  They are extensibility, modularity, fault-tolerance, reliability, reusability, robustness, security, usability, portability, performance, and scalability.  Extensibility is the ability to add new functionality to the project, such as a health bar.  Modularity is the ability of the code to be broken up into different sections, which allows for better debugging and optimization of the different parts.  This is shown in the architecture where the different parts of the user interface and server have to interact.  Fault-tolerance is the resistance to and ability to recover from a failure, like an unexpected user disconnection or server failure.  Reliability is the ability of the project to perform for the given specifications of internal and environmental conditions over time periods.  Reusability is the ability to use preexisting functions from other resources with limited to no modification, such as using the UNITY libraries to create the user interface.  The robustness of the project is the system’s ability to perform despite unpredictable or invalid input.  The security is the ability to deter attacks against the designer’s will, such as gaining access to the entire map layout.  Usability is the ability for the users to navigate the project with ease.  The performance is the project’s ability to complete tasks within acceptable memory and time constraints.  Portability is the ability to be used in multiple environments with varying conditions.  Scalability is the ability of the project to accommodate a growing number of users or data.

**2. 2.1 Assumptions**

Software

* The user interface is dependent on the UNITY libraries.
  + This depends on WebGL.

Usage

* It is dependent and assumed that the user will use the “wasd” keys to move their player.
* It is assumed that the maze generated will be large enough to contain all of the players.

Operational Environment

* It is assumed to be running in a modern web browser.
* It is assumed that the hardware has a minimum of 1Gb of RAM and a 1.5GHz processor.
* It is assumed that the system latency is equal to or less than 100ms.

**2.2 Constraints**

* The data transmit rate between the server and user will increase the performance time.
* The memory limitation of the server and user will impact the scale of the data that can be stored on it, which impacts the functionality and number of users.
* The speed limitation of the server and user will impact the performance time of the system.

**2.3 System Environment**

* The system must interact with the user’s hardware.
  + i.e. processor, memory, networking chip, and graphics circuit
* The system must interact with the user’s web browser software.

**3. Architectural Design**

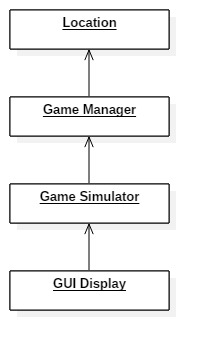
****

Figure 1: Aggregation Structure

    The core of the architectural design is the use of a client that an interfaces with a back end server over an XML-RPC connection to get the locations of the different players at any given time. The backend server will use a Sqlite3 database to store the location of all of the players and bots. The client will then interface with the GUI by following a similar protocol of sending the players and bots locations. The GUI will take in this data and convert it into the movement of the different players and bots. The GUI will also send the client the updated location of the player to send to the backend server for storage.

**3.1 Overview**

Backend Server: Manages the location data for the different players and bots. The servers main role is to send the clients the updated location data for the others players in the game, while storing the updated location of the player in a Sqlite3 database.(C++)

GUI: Gives users a simplistic approach to setting up and playing the game. Consists of code to make a simple user interface that can be easily navigated. This GUI will interface with the client simulator by sending updated location data and updating the image based on location data from the client.(Unity IDE, C#)

Client: Poles the server over an XML-RPC connection for updated location data of the other players and sends the updated location data of the player to be stored. The client then interfaces with the GUI by sending updated location data received from the server to be converted into a graphical image.(C++/PHP)

**3.2 Rationale**

We are using the architecture that we have chosen because it is feasible and will allow for a simplistic user interface from the user’s perspective, as in it will be easy for them to navigate through the menu and setting up games. Also we must start off with having the backend server be the backbone of our architecture because it is critical in getting our game to function on a multiplayer level. Lastly, Labyrinth is very much a user interface game which is why GUI is our second most important function behind the server.

**3.3 Conceptual (or Logical) View**

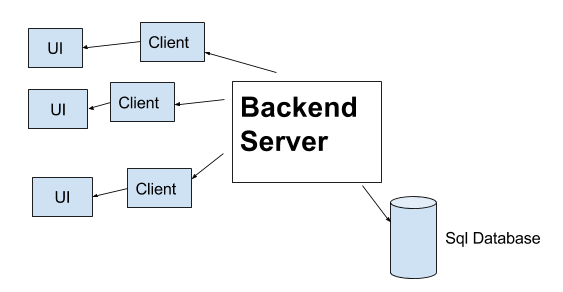


Figure 2: Conceptual Architure

The backend server manages the location data for all of the players in the game, along with the automated bots. The server interfaces with a database to store the locations of all of the players at any given time. The server is routinely polled by the client for the location of the other players and the bots. The client also sends the updated location of the player it is managing to the server in order to be stored and used by other users. The client finally interfaces with the GUI by sending the updated location of all of the other players, while receiving the updated location of the player. The GUI converts these new data points into the movement of the other players and bots.

**4. Low Level Design**

**4.1 Class Diagram**

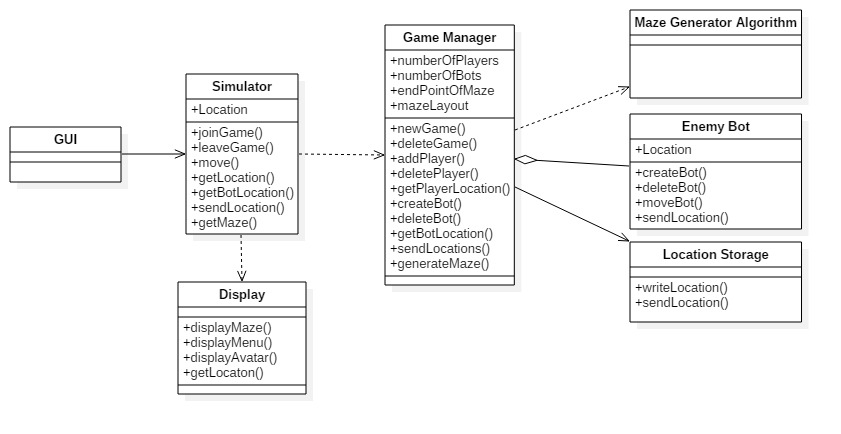


Figure 3: Class Diagram showing all classes and association in the system

**4.2 Sequence Diagrams**

**4.2.1 Principle Action: Join/Leave Game**

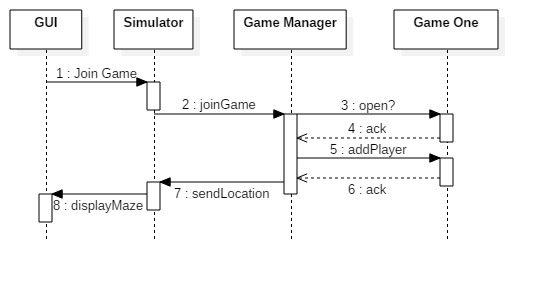
****

Figure 4: Sequence Diagram for Principle Action Join Game

**4.2.2 Principle Action: Create/ Delete Game**

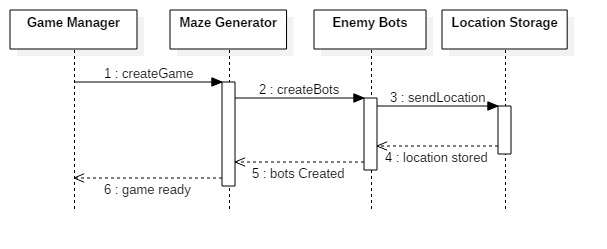
****

Figure 5: Sequence Diagram for Principle Action Create Game

**4.2.3 Principle Action: Display Maze**

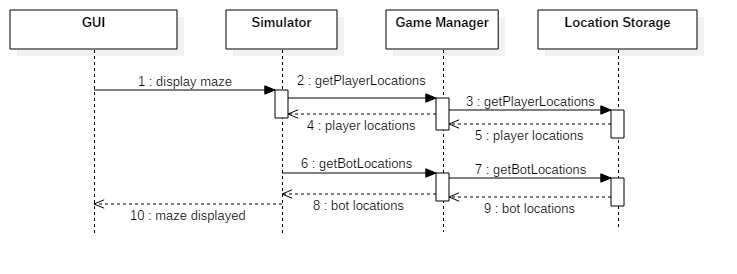
****

Figure 6: Sequence Diagram for Principle Action Display Maze

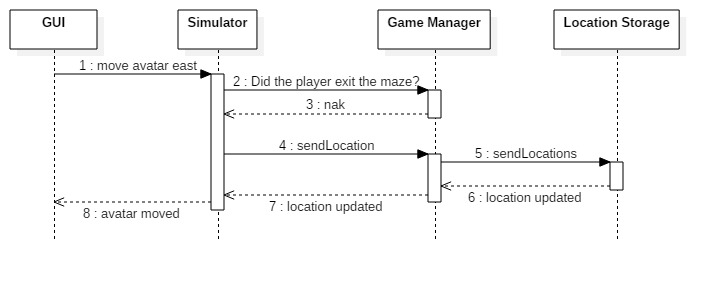
**4.2.4 Principle Action: Move Player**

Figure 7: Sequence Diagram for Principle Action Move Player

**4.3 State Diagram**

**4.3.1 Game Manager**

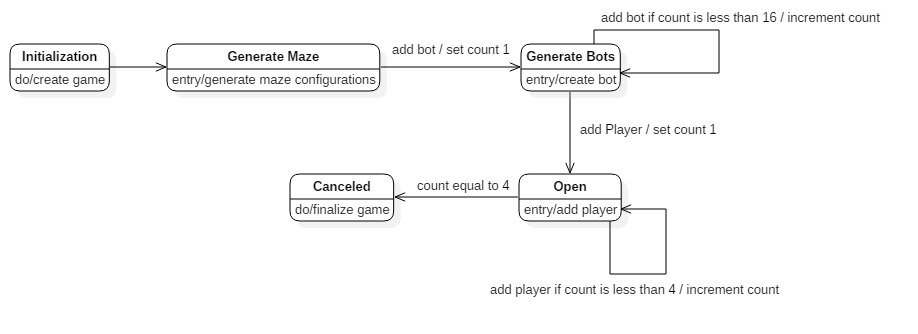
****

Figure 8: State Diagram for Game Manager

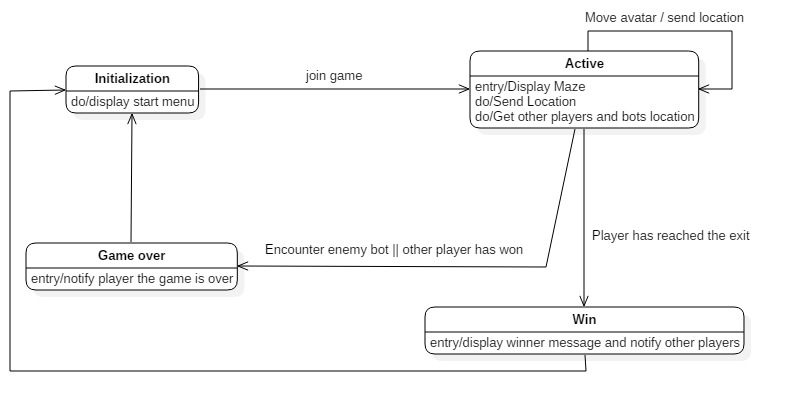
**    4.3.1 Game Simulator**

Figure 9: State Diagram for Game Simulator

**5. User Interface Design**

Upon entering www.labyrinthgame.com, one will see a menu with the following options:

* Join Game
  + Prompted to enter a username
  + User assigned one of four avatars
  + Servers will enter the user into the next available game; games will start once 4 people join a game, next game will begin when 4 others are slated into a game
* How to Play
  + A description of the game objective and controls for the game will be shown
  + Users shall be given control of the avatar's motion through the use of the “w”, “a”, “s”, “d”, or arrow keys
* Settings
  + Volume control

Upon successfully joining a game:

* User will see the back of the player as the game is played in third person
* User will see the immediate area in front and around them which can include walls, fellow players, and enemy bots
* There will be a beam of light descending upon the final destination that the user will be able to see
* Walls of the maze will be high enough that you cannot see over them
* To exit the game while it is still running, click the “X” in the top right corner of the screen.
* All of this will be created with Unity Personal, free software available for download that helps create 3D games
  + Coding will be done in C# or JavaScript
  + Unity contains hundreds of possible libraries you can import and use; we will decide with our discretion which are suitable for use

Upon the end of a game when one user reaches the final destination.

* Game over screen
  + States which user won the game
  + Click on “New Game” to enter into the next available game

*2D maze (backup plan)*

* User will be represented as a dot on the screen
* Walls of maze will be represented by lines
* Obstacles will be represented by x’s
* The final destination will be represented by a red circle
* Shown with a bird’s eye view, so each user will see the whole maze at once
* Overly-simplified version of the 3D counterpart, only will be used as a backup plan if all else fails