# Software Requirements Specification

## for

# Trivia Maze

**Team: Fast Fire File Flyers(4F)**

**Members: Brandon Fowler, James White, Zach Lontz, Daniil Borisov**

**May 13, 2014**

# 1. <u>Introduction</u>

## 1.1 Purpose

This SRS describes the software requirements of the Trivia Maze project for CSCD350. This document is to be used by the team Fast Fire File Flyers that will be building this project.

## 1.2 Scope

This will be a single player, millionaire themed, trivia maze game; that tests a players knowledge and problem solving ability.

## 1.3 Software Context

This software is intended to function as entertainment for a single user; as well as teach the user new information.

# 2. <u>Overall Description</u>

## 2.1 Functional Overview

In full functionality, this software will allow a user to traverse a two dimensional grid of rooms that make up a maze. Each room has four doors, with the exception of rooms on the edge of the grid. When the user tries to move between rooms, they will be asked a multiple choice question at each door. If the user answers correctly, they will be able to move into that room, otherwise the door is locked and the user must choose a different path.  The user will have the option to use millionaire styled life lines in order to pass difficult questions. The user will either reach the end of the maze and win or exhaust all possible paths and lose the game.

## 2.2 Generic Use-Case

The player is given a starting point in the maze. The player chooses to move towards the exit. The player is asked a question. The player answers correctly and is allow to move towards the exit. The player tries to move another step towards the exit. The player is asked a question. The player answers incorrectly and is blocked from moving towards the exit. The player will continue in a similar manner until the player either wins or loses the game.

## 2.3 Operation environment

This software will be used in a Windows environment that supports Java JRE 7.

## 2.4 Design Constraints

All code will be written in and must conform to Java 7.

## 2.5 User Data

No user data or state will be saved after the game has ended.

## 2.6 Dependencies

This software will depend on a SQLite database, in order to retrieve questions and answers for the user.

## 2.7 Assumptions

The user is able to read and use a mouse/keyboard to interact with the software.

# 3. <u>Software Model Description</u>

## 3.1 Classes

This software model is comprised of five Java classes.

### 3.1.1 Room Class

The Room class has two to four door values that can be set to represent open or closed. It also has values for if a particular room is occupied by the player or if it is the maze exit.

### 3.1.2 Maze Class

The Maze class stores a 2D representation of a grid of Room objects. The Maze class also contains functionality for manipulating, printing, and error checking itself.

### 3.1.3 Player Class

The Player class stores values associated with the player such as location, items, and available life lines; and also has functionality for manipulating these data values.

### 3.1.4 Database Class

The Database class is responsible for connecting to a SQLite database and preparing a trivia question.

### 3.1.5 Game Class

The Game class implements the main program behavior by initializing other class objects, and manipulating them, while interacting with the users decisions.

# 4.  Interface Requirements

## 4.1  User Interfaces

The user is expected to view and interact with the Windows command line, in order to play the Trivia Maze game.

## 4.2  Hardware Interfaces

The user is expected to use a mouse and keyboard, in order to supply decision data that is required from the user.

# 5.  Testing Requirements

## 5.1 Maze Traversal

A test will be implemented to check if the maze is still possible to complete, after doors are locked.

## 5.2 J-Unit

J-Unit tests will be implemented in order to test maze bounds, valid user movements, maze completion, and various core functionality concerns.

## 5.3 User Testing

Extensive user testing will be done by the project development team to test presentation, functionality, and possible programming errors.