Software Requirements Specifications
for Team Qube's Trivia Maze
Version 1.0

Prepared By:
Sam Gronhovd
Daniel Heffley
Kevin Reynolds

December 9, 2014

1.0 Introduction

This section provides an overview of the entire requirement document. This document describes all data, functional and behavioral requirements for software.

- **1.1 Goals and objectives**
  - Ability to navigate a maze in which travelling to the next door requires the answering of a trivia question.
- **1.2 Statement of scope**
  - Command line program in which the player navigates the maze through keyboard inputs and all information is displayed as text.
- **1.3 Software context**
  - This is a toy project with the intention to help us learn to work with each other as a team to accomplish the goal. The Goal of the game is to have the player test their wits in video game trivia.
- **1.4 Major constraints**
  - The major constraints to this project include a lack of experience of certain implementations like setting up Installers and fully utilizing pivotal tracker, and working with GITHub.

2.0 Usage scenario

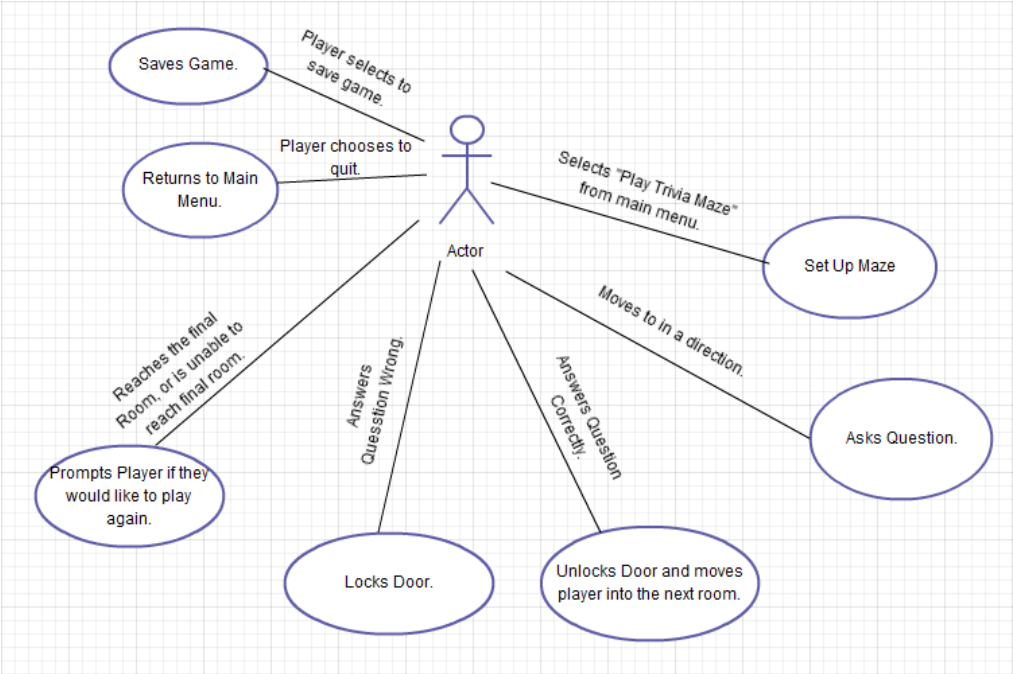
- The Application is used when someone wants to pass the time with some video game trivia.

2.1 User profiles

- The ideal user is somebody with a knowledge of video games, the subject of the trivia questions. As well as someone who is literate in order to be able to read the questions.

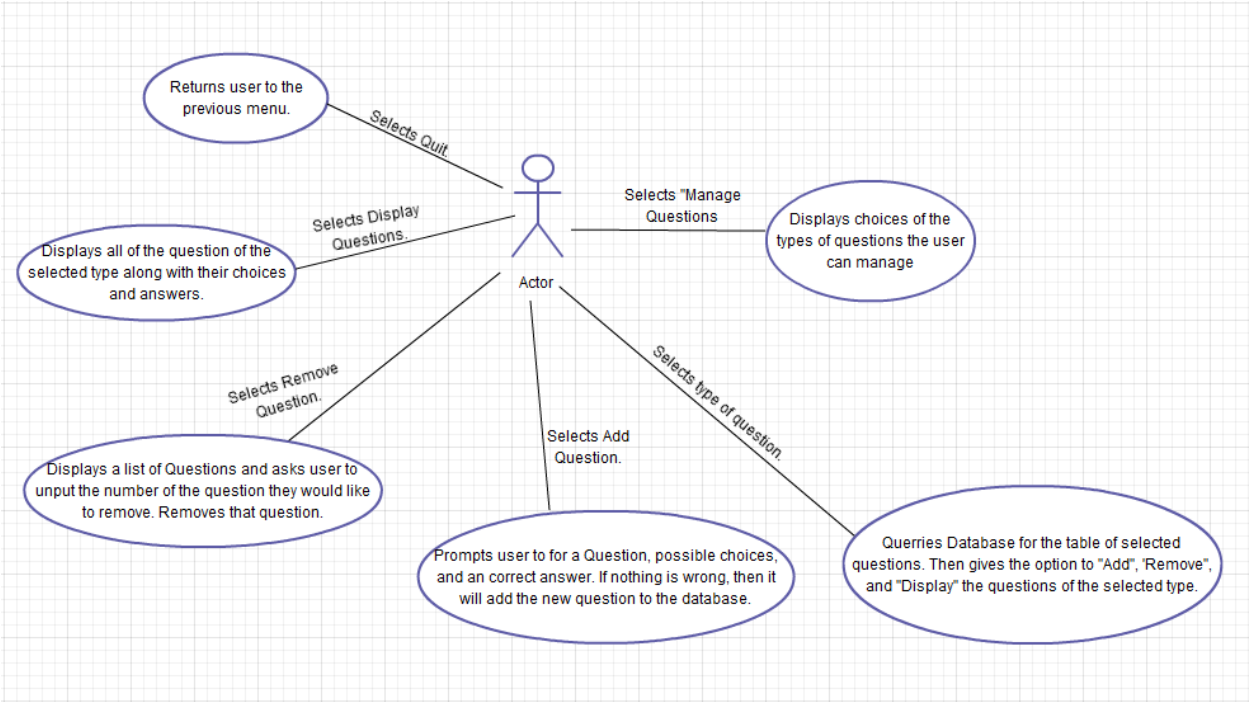
2.2 Use-cases

2.2.1 Play Game



Playing Trivia Maze		
Actors:	User	
Preconditions:	User does not load a saved game.	
Trigger:	User selects "Play Trivia Maze" from the Startup Menu.	
Basic Flow:		
	Actor	Application
	2. User then chooses which direction to travel. Up, Down, Left, Right. 4. User then answers the question presented. 6. User continues to traverse the Maze. 11. User chooses whether or not to play the game again.	1. Application accesses database of questions and sets up the maze for the user to traverse. 3. If the Door in the direction the user chose to travel has not been opened or locked, the application then prompts the user to answer a question. 5. If the question is answered correctly then the door is unlocked and the user is moved to the next room. If the question is answered wrong, then the door is locked and the user cannot pass to the next room through that door, and a check is run to see if it possible to finish the maze. 7. If the door is unlocked the User moves to the next room. 8. If the door is locked, the User will not be able to move to the room through the locked door. 9. If the maze is not solvable, then the user is prompted that they lost and asked if they would like to play again. 10. If the user makes it to the final room of the maze they are congratulated for their superior intellect and asked if they would like to play the game again.
Post Conditions:		
Alternatives:	User could choose to Load a game	

2.2.2 Question Manager



Manage Question		
Actors:	User	
Preconditions:	There are questions in the database to alter	
Trigger:	User selects "Manage Questions" from the Startup Menu.	
Basic Flow:		
	Actor	Application
	2. The user chooses the type of question they would like to manage. 4. The user chooses to Add a question. 6. User enters all of the question information asked by the program. 7. User chooses to Remove a question. 9. User enters the number of the question to be removed. 11. User chooses to Display Questions. 13. User chooses to exit.	1. Application prompts the user with the types of questions that the user can edit. The choices are Multiple choice, True/False, Short Answer. 3. Prompts the user if they want to 'Add', 'Remove', 'Display Questions', or 'Exit'. 5. Prompts user to type in a question, followed by any choices for the answer, and with an answer to the question. 8. Application then displays all of the available questions of that type in a list and prompts the user enter the number of the question to be removed. 10. Removes the selected question, and prompts user to 'Add', 'Remove', 'Display Questions', or 'Exit'. 12. Displays all the questions in that category along with the choices and answers. 14. Exits to the main menu.
Post Conditions:	Database of questions has been changed in some capacity.	
Alternatives:		

2.2.3 Load/Save Game

Load a game of Trivia Maze		
Actors:	User	
Preconditions:	There is a saved game of Triva Maze	
Trigger:	User selects "Load Game" from the Startup Menu.	
Basic Flow:		
	Actor	Application
	2. Plays game as if they started a new game, but with some progress, hopefully.	1. Program loads game from the single save file and builds the maze as it was and loads the user in the room that they saved in.
Post Conditions:		
Alternatives:	User could choose to start a new game	

Saving a Game of Trivia Maze		
Actors:	User	
Preconditions:	The player has started a Game of Trivia Maze	
Trigger:	User selects "Save" from the Gameplay Menu.	
Basic Flow:		
	Actor	Application
	2. User continues to Play Trivia Maze.	1. Applications converts all objects to be saved into binary using a serializer and stores them in output.txt
Post Conditions:		
Alternatives:	User could choose to Load a game	

Quiting a Game of Trivia Maze		
Actors:	User	
Preconditions:	The User has started a Game of Trivia Maze	
Trigger:	User selects "Quit" from the Gameplay Menu.	
Basic Flow:		
	Actor	Application
	2. User chooses what to do in the main menu.	1. Application returns to main menu of the application.
Post Conditions:		
Alternatives:		

2.3 Special usage considerations

- This program will only be able to work on a Windows Operating System, and will require a keyboard.

3.0 Data Model and Description

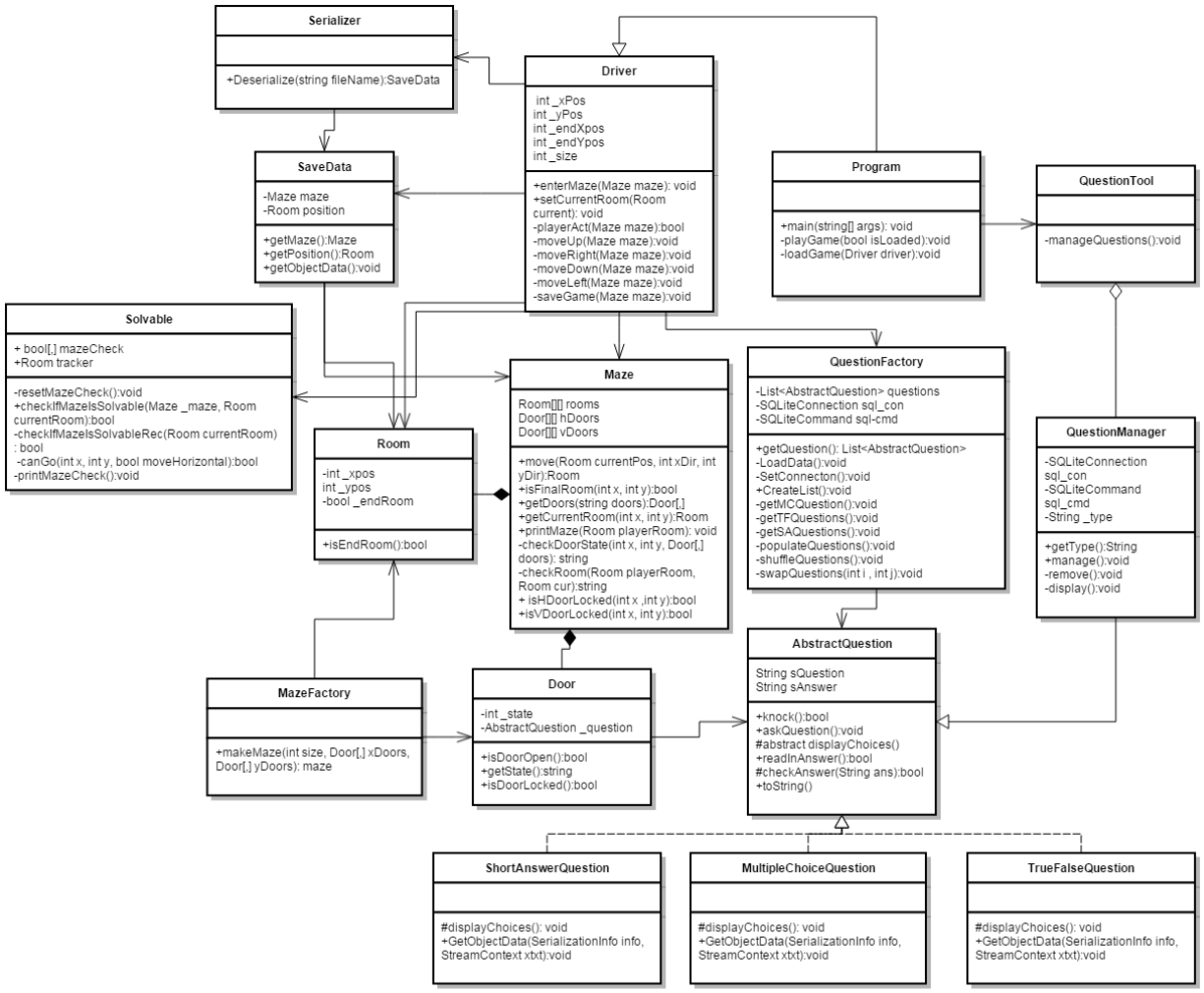
This section describes information domain for the software

- **3.1 Data Description**
  - Data objects that will be managed/manipulated by the software are described in this section.
  - Each trivia question will be loaded into a Question object which will handle the displaying of the question and the receiving of the player's answers.

- The maze will be a rectangular shape, and consist of a 2-dimensional array of Room objects, and 2 2-dimensional arrays of doors, one for the horizontal doors, and another for the vertical doors.
- To travel between rooms in the maze, we will use Door objects that will contain the state of that door, whether it is locked, open, or still just closed.
- The Driver object will keep track of the player's coordinates, and move the player throughout the maze.

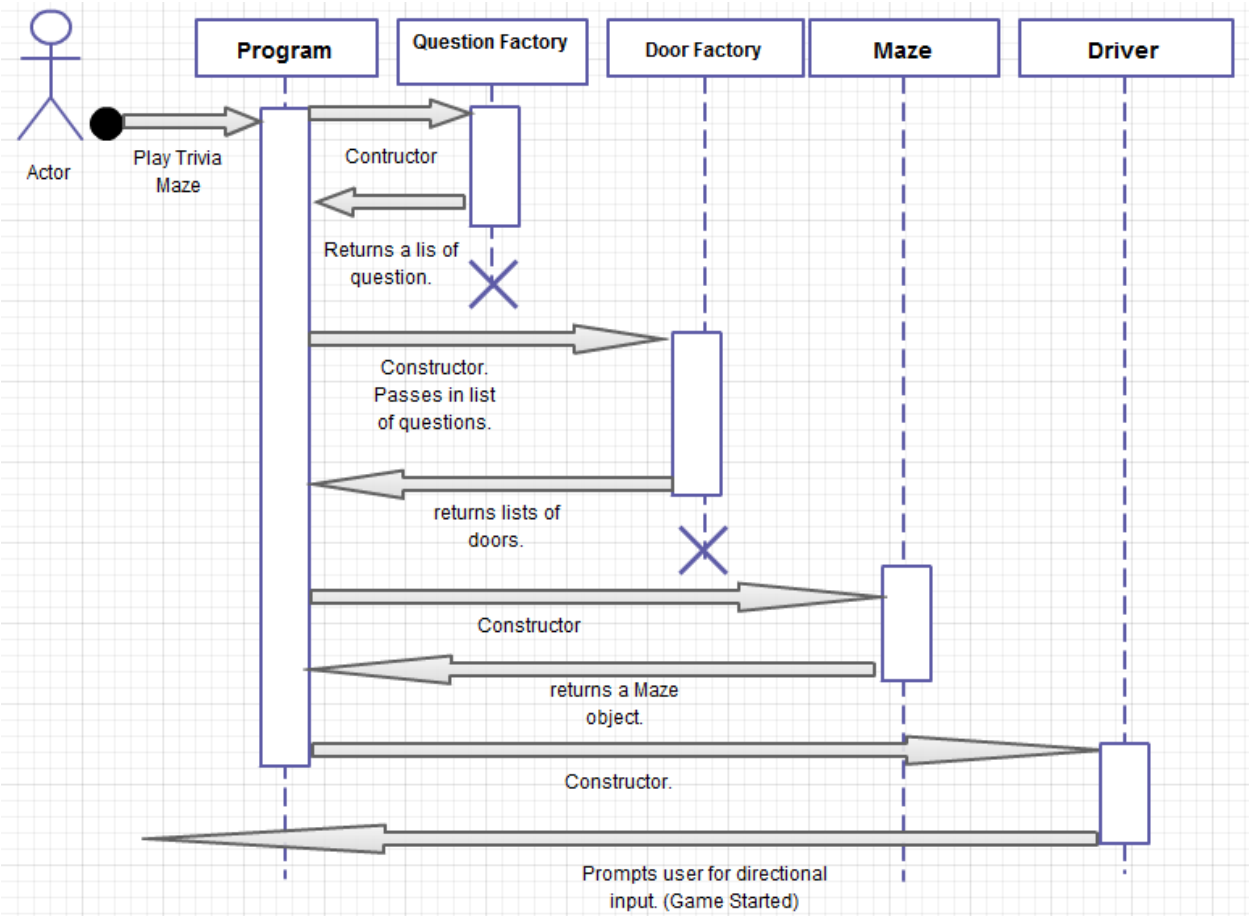
- **3.1.1 Data objects**
  - Data objects and their major attributes are described.
  - **3.1.1.0 Question Factory**
    - SQLite Connection - holds file connection data.
    - SQLite Command - a string that stores a SQL query command.
    - Questions - a list of questions that will be created.
  - **3.1.1.1 Question**
    - question - a string that holds the question text.
    - choices[] - an array that holds the strings of choices for the question.
    - answer - a string that contains the correct answer from the choices.
  - **3.1.1.2 Door**
    - Question - a Question Object that the door will call when knocked on.
  - **3.1.1.3 Room**
    - xPosition - the x position of the door in the Maze's 2D array of door objects.
    - yPosition - the y position of the door in the Maze's 2D array of door objects.
    - isFinalRoom - a boolean that is true if the room is the end of the maze.
  - **3.1.1.4 Maze**
    - Rooms[][] - a two dimensional array that holds all of the Room Objects in the maze.
    - DoorsX[][] - a 2D array of doors that connect rooms horizontally.
    - DoorsY[][] - a 2D array of doors that connect rooms vertically.
  - **3.1.1.5 Driver**
    - CurrentRoom - holds a room object that is used to hold the player's current position.
  - **3.1.1.6 Maze Factory**
    - Rooms[][] - This is where the 2D array is made to be passed to the Maze class.
  - **3.1.1.7 Door Factory**
    -
  - **3.1.1.8 Question Tool**
    -
  - **3.1.1.9 Question Manager**
    -
- **3.1.2 Relationships**
  - Relationships among data objects are described using an ERD- like form. No attempt is made to provide detail at this stage.

- **3.1.3 Complete data model**



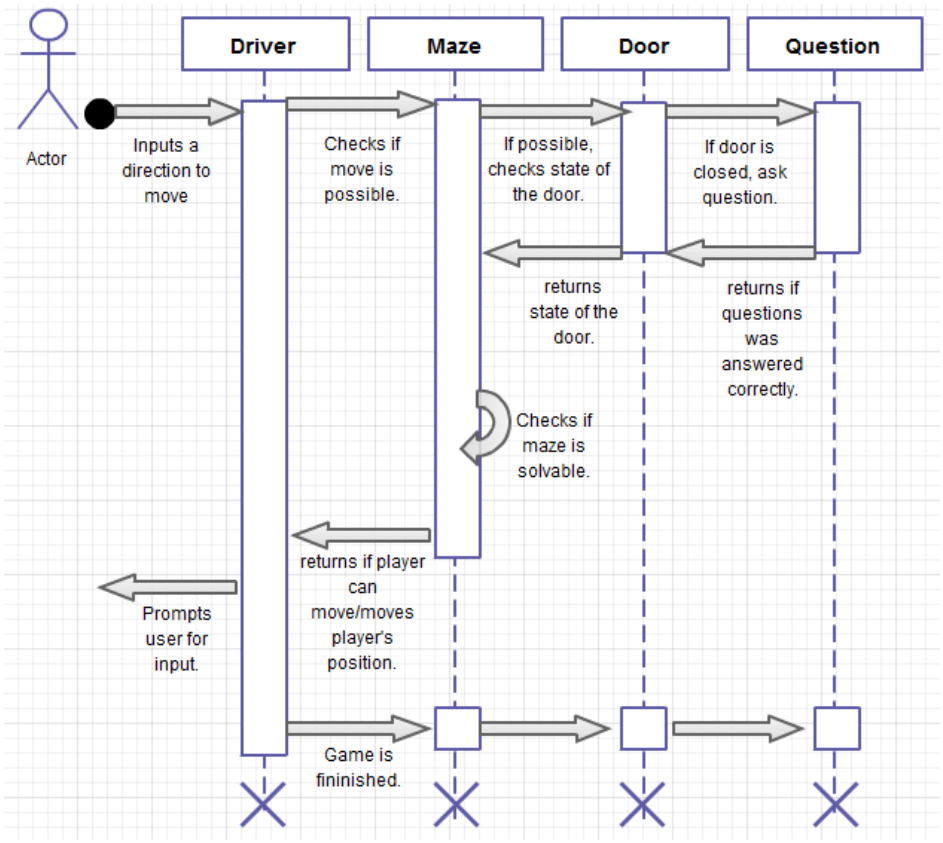
4.0 Functional Model and Description

- **4.1 Initialization**
  - **4.1.1 Processing Narrative**
    - When the program is initialized it will create a Question Factory, a Door Factory, and a Question Factory.
    - The Question Factory will open a connection with a database and retrieve the information and create Question Objects with that information. Once all the Question Objects are created the connection to the database will be closed.
    - The Door Factory will take the questions, the Question Factory made and assign a Question Object to each Door.
    - The Maze Factory will retrieve the doors from the Door Factory and assign each room between 2 and 4 doors depending on the room's position in the maze.
    - The Maze Driver is then initialized at the [0][0] element of the Room 2D array.
  - **4.1.2 Flow diagram**



- **4.1.3 Interface description**
  - There is no input or output during initialization
- **4.1.4 Transforms**
  - N/A
- **4.1.5 Performance Issues**
  - Special performance required for the subsystem is specified.
- **4.1.6 Design Constraints**
  - No changing questions on the fly.

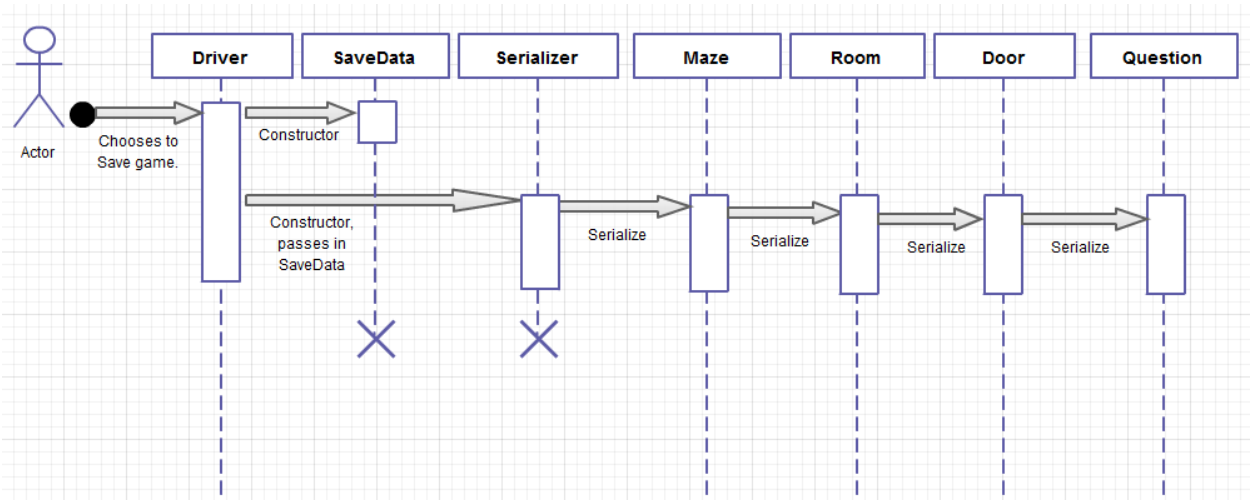
- **4.2 Navigating the Maze**
  - **4.2.1 Processing Narrative**
    - The user is given a the options to move "up", "down", "left", or "right" permitting if that direction is not blocked by a locked door. If the door is closed, then the program prompts the user with the question stored in the door object. If the user answers the question correctly the door will open and the player will be moved to next room. If they answer the question wrong the door will lock, preventing movement through that door. If the player reaches the end of the maze they will be prompted that they won and if they would like to play again. If there are no paths available to the finish, then they will be shown that they lost and asked if they would like to play again.
  - **4.2.2 Flow diagram**



- **4.2.3 Interface description**
  - While playing, each turn there will be a menu displayed which directions to travel. The user then inputs those directions.
- **4.2.4 Transforms**
  - N/A
- **4.2.5 Performance Issues**
  - N/A
- **4.2.6 Design Constraints**
  - N/A

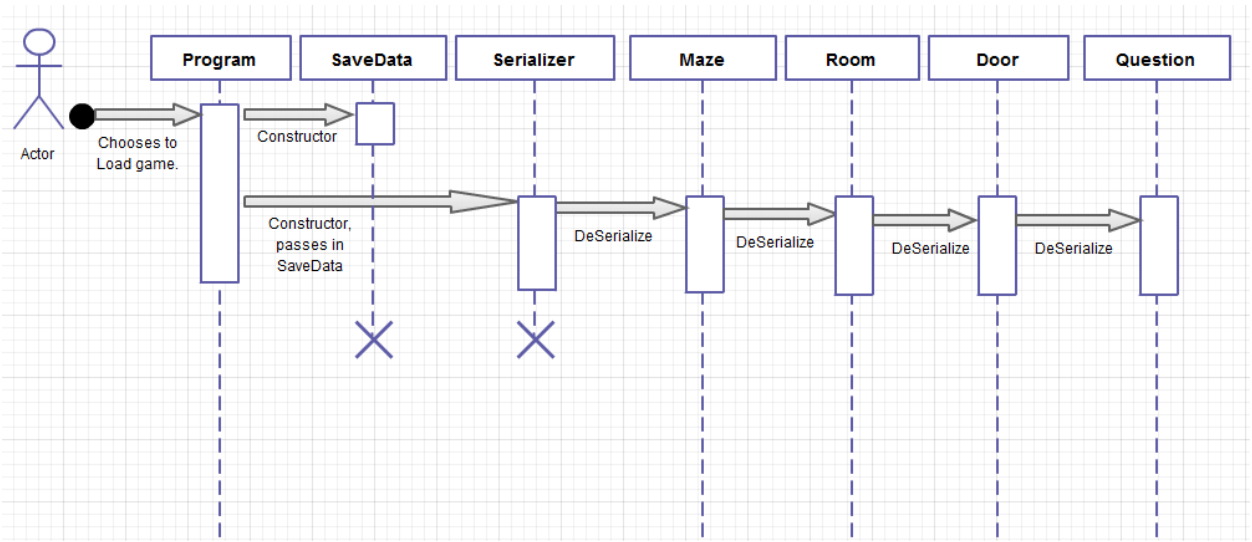
- **4.3 Checking if Maze is Solvable**
  - **4.3.1 Processing Narrative**
  - **4.3.2 Flow diagram**
  - **4.3.3 Interface description**
  - **4.3.4 Transforms**
  - **4.3.5 Performance Issues**
  - **4.3.6 Design Constraints**
- **4.4 Saving Game**
  - **4.4.1 Processing Narrative**
    - User chooses the menu option to save the game. The application then serializes the Maze class and all of the classes held in that class along with the position of the player in the maze.

- **4.4.2 Flow diagram**



- **4.4.3 Interface description**
  - User selects the save option in the game.
- **4.4.4 Transforms**
  - N/A
- **4.4.5 Performance Issues**
  - Hardly any performance hit while saving due to the relatively small amount of data being serialized.
- **4.4.6 Design Constraints**
  - The player must be playing the game in order to save.

- **4.5 Loading Game**
  - **4.5.1 Processing Narrative**
    - From the main menu, when the user selects "Load Game" it will see if a save file exists. If a file exists it will run the data through a deserializer and create the Maze object as it was when the game was save, as well as positioning the player in the room in which they saved the game.
  - **4.5.2 Flow diagram**



  - **4.5.3 Interface description**
    - User chooses the load game option from the main menu.

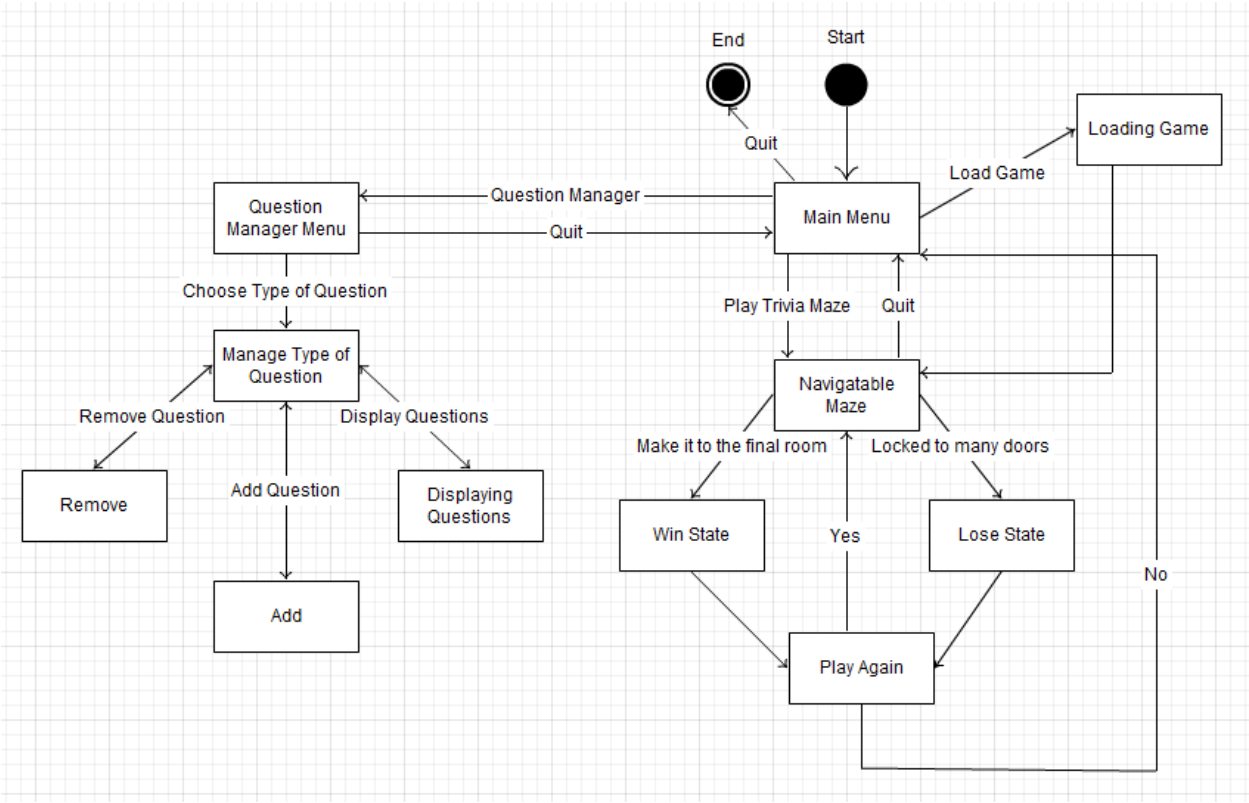
        - **4.5.4 Transforms**
            - N/A
        - **4.5.5 Performance Issues**
            - Hardly any performance hit while saving due to the relatively small amount of data being deserialized.
        - **4.5.6 Design Constraints**
            - There must exist a save file in order for a game to be loaded.
- **4.6 Exiting Game**
    - **4.6.1 Processing Narrative**
        - While playing the game, the user chooses the "Quit" option in the menu and the application will leave the for gameplay loop and return the user to the main menu.
    - **4.6.2 Flow diagram**
        - N/A
    - **4.6.3 Interface description**
        - There is an option in the gameplay menu for the user to exit the game.
    - **4.6.4 Transforms**
        - N/A
    - **4.6.5 Performance Issues**
        - N/A
    - **4.6.6 Design Constraints**
        - N/A

- A detailed description of each software function is presented. Section 4.1 is repeated for each of n functions.

5.0 Behavioral Model and Description

- **5.1 Description for software behavior**
    - **5.1.1 Events**
        - Answering of a question will cause the door to change state from closed to either locked or unlocked .
        - The game as a whole will remain in the same state until the player reaches the final room, which will trigger a change to the game-win state, or until there is no possible route to the final room, this will trigger a switch to a game-lose state.
    - **5.1.2 States**
        - The game will have 3 main states, playing, won, or lost.
        - The doors will have 3 states, closed, locked, and unlocked.

- **5.2 State Transition Diagrams**



6.0 Restrictions, Limitations, and Constraints

This program is limited by only being able to operate on a Windows system. It is also constrained in only being able to convey information through text through the console.

7.0 Validation Criteria

The approach to software validation is described.

- **7.1 Classes of tests**
  - Maze Testing-
    - 2D array of rooms is not null;
    - Rooms are not outside the bounds of the array.
    - Has 2 2D arrays of Door objects that are not null.
    - Doors are not outside the bounds of the array.
  - MazeFactory Test-
    - Make sure that the 2D array returned is the correct size.
  - Room Test-
    - xPosition, yPosition are not null.
    - xPosition, yPosition are within the bounds of the maze.
    - Only one finalroom exists within the maze.
  - Door Test-
    - The door contains a question.
    - The door has the proper x and y coordinates.
    - State changes correctly.
  - DoorFactory Test-
    - Make sure that the array returned is the correct size.
  - Question Test-
    - Question does not contain any null members.

        - Questions print out data correctly.
        - Parses input and is able to compare it to the answer.
    - QuestionFactory Test-
        - Handle not finding a database file.
        - Doesn't return a null Question List.
    - Driver Test-
        - Handle null arguments
    - Maze Test-
        - Handles null arguments
    - QuestionManager -
        - Handle no database file exception
        - Handle non existing Table exception
- **7.2 Expected software response**
    - It will handle exceptions with grace and let the user know something went wrong if it crashes.
    - We expect this program to not crash unexpectedly.
- **7.3 Performance bounds**
    - It will run. There maybe a performance hit when querying the database.