Software Requirements Specification

for

<CSCD350 Trivia Maze>

Version 1.2 approved

Prepared by David Walker, Nick Witmer, and Jenia Rousseva

Team: To Be Continued

12/07/2015

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Team	11/23/2015	First Draft	1.0
Team	12/2/2015	Added more details after working through more of the project	1.1
Nicholas Witmer	12/7/2015	Added JVM and SQLite Versions, Fixed Table of contents, added security info	1.2

1. Introduction

1.1 Purpose

This SRS describes the software functional and nonfunctional requirements of the Trivia Maze as of Iteration 3. This version of the document will be used by members of the team to assess our progress towards the final draft of the game to be turned in at the end of the quarter. This SRS will be revised during the final iteration as we finish our implementation.

1.2 Project Scope and Product Features

This software is a Trivia Maze game where the player traverses the maze and when they come to a door they are asked a true/false, short answer, or multiple choice question to open the door. The questions will be stored in a SQLite database and retrieved when the player attempts to pass a door.

1.3 References

1. Coding Standard Sheet for standards on the code/comments added to the system.

2. Overall Description

2.1 Product Perspective

The Trivia Maze game is a new game which utilizes some algorithms and techniques for maze generation and traversal developed as part of the term project in CSCD 349 Design Patterns.

2.2 User Classes and Characteristics

Player/User:

A Player/User are the ones who have downloaded the game and are trying to play it. They have the ability to use all of the basic functions to save/load/play the game, no access to any admin tools. Technical education is not necessary to play the game.

Admin:

Admin's have access to everything that a Player/User does, in addition, they have access to the admin tools, giving them the ability to add questions, etc (TBD). Some technical education may be necessary depending on how in depth they are intending to get with editing the game.

2.3 Operating Environment

The Trivia Maze game will operate in the JVM, versions 1.7.x and 1.8.x, on Windows.

2.4 Design and Implementation Constraints

CO-1: The system's design and code documentation shall conform to the Coding Standards document developed by the team
CO-2: The system shall use the current standard for SQLite Databases, currently SQLite Version 3.9.2.
CO-3: The system shall use the current standard for JDBC connection to SQLite databases, SQLite JDBC Driver, version 3.8.11.2.

2.5 Assumptions and Dependencies

The system depends on a SQLite Database, SQLite version 3.9.2, and the SQLite JDBC Driver, version 3.8.11.2.

3. System Features

3.1 Save/Load of a Game

3.1.1 Description and Priority

A User who is playing the game may either chose to start a new game or load from a save file, if one exists, additionally, progressed in saved by the game on close, if the user so choses, so that they can pick up where they left off later.

3.1.2 Stimulus/Response Sequences

Stimulus: Player starts up the game and is prompted to load from save or start a new game
Response: Either loads from the save file, or initiates creation of a new game
on close
Stimulus: Player decides to quit the game, is prompted on whether or not they would like to save their game
Response: If the Player decides they would like to save their game, their progress is saved and the game is closed, if not, the game is simply closed without saving.

3.1.3 Functional Requirements

saveGame: The system will write progress of the game to the serialized save file

loadGame: The system, if the save file exists, reads from the file, creating an instance of the game with the saved progress so the user are back where they were.

3.2 Database Connection

3.2.1 Description and Priority

This is implemented in the DatabaseUtility class which allows other aspects of the game to interact with the database without knowing how the database is implemented. The connection to the database will allow the insertion and retrieval of questions for the trivia maze. This is a high priority feature because it is essential for the game to store and display questions for the player to answer.

3.2.2 Stimulus/Response Sequences

Stimulus: the game administrator uses the admin tool to insert a new question into the database
Response: the DatabaseUtility will attempt to insert the question into the database returning true to the caller if the update was successful and false if the update failed

Stimulus: the player attempts to pass through a locked door in the maze
Response: the maze will attempt retrieve a questions from the database, through the DatabaseUtility which will return a Question if the retrieval was successful and currently returns null if it failed. Since returning null is not a good practice this will be changed in our 4th iteration.

3.2.3 Functional Requirements

REQ-1: DatabaseUtility.insertQuestion: Inserts a question into the database, with overloaded methods for each type of question, true/false, short answer, and multiple choice.
REQ-2: DatabaseUtility.retrieveQuestion: Takes a question type and returns a randomly selected question from the database, stored in a Question object.

3.3 Generate a Random Maze

3.3.1 Description and Priority

At the start of each new game a random maze will be generated. The size of the maze will vary based on the difficulty level chosen by the player. The maze generation will be done using Eller's algorithm for generating a perfect-maze (there is only one path from start to finish). However, since we desire a non-perfect maze for this game, i.e. multiple paths from the start position to the end position, because paths may become blocked off along the way, we convert this maze into a non-perfect one by opening up positions along the shortest path. For every open position, we randomly decide if there should be a question there and also randomly determine the question type: true/false, multiple choice, or short answer.

3.3.2 Stimulus/Response Sequences

Stimulus: The player decides to start a new game.

Response: A new maze is randomly generated and a Maze object is returned.

3.3.3 Functional Requirements

REQ-1: Ellers class: Generates a perfect random maze via Eller's algorithm.
REQ-2: Maze.findEntrance: Finds and marks the start position (S) in the maze.
REQ-3: Maze.findExit: Finds and marks the goal position (G) in the maze.
REQ-4: Maze.findPath: Finds the shortest path from the start position to the end position. Can also convert a perfect maze into a non-perfect one if the Boolean flag is set to true.
REQ-5:Maze.setFinalMaze: Converts a non-perfect maze with only walls and open cells into a maze with walls, open cells, and questions cells.

3.4 Traverse the Maze

3.4.1 Description and Priority

This functionality is implemented in the Maze class. The player will be able to see their current location in the maze, and their path of visited spots in the maze. A player will have the option of moving either north, east, south, or west. As long as the position that the player wants to enter is not a wall (blocked position), the player is allowed to move to that cell. Then, depending on whether the cell contains a question or is an open cell, the player will either be presented with a question or allowed to continue traversing the maze. User input is scrubbed to make sure the player enters a valid direction.

3.4.2 Stimulus/Response Sequences

Stimulus: Player chooses a valid direction: north, east, south, or west relative to his/her current position to move to.

Response: The player successfully moves to that spot if it is an open space, a question space, or a visited space, or the player is asked to choose another direction if that position is walled off.

Stimulus: The player successfully moves to a new cell.

Response: If the cell is an open cell or a visited cell, the player gets to continue traversing the maze. Otherwise, if the cell contains one of the three question types, a random question of that type is obtained from the database and the player must answer the question correctly to open up that cell; of he/she answers the question incorrectly then that position becomes a wall.

3.4.3 Functional Requirements

REQ-1: Maze.print: Prints the maze to be displayed to the user. Only cells adjacent to the player current position are displayed.
REQ-2: Maze.canMove: Checks if the cell in the Maze is a wall.
REQ-3: Maze.getDirection: Gets the direction to move from the player (north, east, south, or west) and verifies that this is a valid direction. Loops until the user quits or enters a valid direction.
REQ-4: verifyDirection: Checks whether the direction in which to move entered by the user is indeed a valid direction, i.e. not a wall. If valid, marks the cell as visited and updates the current row and column.

3.5 Questions Hierarchy

3.5.1 Description and Priority

There will be an abstract class called Question that provides the basic functionality of a question in within the Trivia Maze game. For each of the three types of questions supported, there will be a class which inherits from Question and provides the specific functionality for that type of question (true/false, multiple choice, or short answer). A QuestionFactory will also be used to create a generic Question type whose underlying structure is one of the three types of questions (via polymorphism).

3.5.2 Stimulus/Response Sequences

Stimulus: The player moves to a cell in the maze containing a question of one of the three types.

Response: The database is queried for a question of the appropriate type and a Question object is created a returned using the QuestionFactory.

3.5.3 Functional Requirements

REQ-1: TrueFalseQuestion class: Inherits from Question and provides the functionality for a true/false question.
REQ-2: MultipleChoiceQuestion class: Inherits from Question and provides the functionality for a multiple/choice question.
REQ-3: Short Answer class: Inherits from Question and provides the functionality for a short answer question.
REQ-4: ErrorQuestion class: Inherits from the Question class and is used to make sure a valid question is returned from the database.
REQ-5: Question.getValidInput: Checks if the user enters valid input for the answer for a given question type.
REQ-6: Question.specialFunction: A hook o be overridden in the subclasses for any special functionality for that particular type of question.
REQ-7: Question.printQuestion: Prints the question and its answer choices if applicable.
REQ-8: Question.checkCorrectAnswer: Check if the answer passed in is the correct answer, ignoring case. This method may be overridden in the subclasses for more specific functionality.

3.6 Question Handler

3.6.1 Description and Priority

A QuestionHandler class will be used to manage question retrieval from the database when the player lands on a question cell in maze.

3.6.2 Stimulus/Response Sequences

Stimulus: The player moves to a cell in the maze containing a question of one of the three types.

Response: The database is queried for a question of the appropriate type and a Question object is created a returned using the QuestionFactory. The question flag is set to 1 for “used” in the database. The question is displayed to the player and he/she answers it. The maze is updated accordingly to the player getting the question correct or not.

3.6.3 Functional Requirements

REQ-1: getQuetionFromDB: Retrieves a question from the database depending on the type requested
REQ-2: checkFlagsReset: If the question obtained from the database is an invalid question, i.e. ErrorQuestion, then resets the flags for all of the questions of that type in the database and queries the database again for a question. Returns a Question type object.

REQ-3: handleQuestion: Prompts the player to answer the question, checks if the correctness of the player's answer to the question and then updates the maze accordingly based on the player's response. If the player answered the question correctly, the cell is marked as visited and the player may cell in that cell. Otherwise, if the player answered the question incorrectly, then the cell is walled-off and the player is moved back to the most recently visited position.

3.7 Check if a Path Exists to the End of the Maze

3.7.1 Description and Priority

Before the maze is displayed to the player and the player is asked which direction he/she wants to move in, we must check if at least one path exists from the player's current position to the goal position in the maze. If such a path does not exist then the game is discontinued.

3.7.2 Stimulus/Response Sequences

Stimulus: A player has just finished answering a question and the maze has been updated or he/she has just moved into a new room successfully.

Response: Check if at least one path exists from the player's current position to the target position in the maze. If not, terminate the current game and notify the player. If such a path exists, allow the player to keep playing.

3.7.3 Functional Requirements

REQ-1: findPath: In the Maze class there will be a method that checks if a path exists from the position that is passed in as a parameter to the goal position in the maze.

4. External Interface Requirements

4.1 User Interfaces

Main Menu:

The highest level of menu the user can encounter in the game, gives them the choices of traversing the maze, Saving their game or Quitting. The choices are accepted by the user with the number: 1, 2 or 3.

Direction Menu:

This the menu the user encounters when they choose to traverse that maze. The options are N, S, E, W or Q, the first four representing directions the move in the maze and Q being the option to return back the main menu, not moving in any direction.

Admin Tool:

Upon launching the Admin Tool the user will be presented with the menu consisting of 4 options, True False, Short Answer, Multiple Choice, and Exit. When a user has selected a question type they will be prompted for a question, the answer to the question, and in the case of a short answer they will be prompted for a list of keywords or in the case of multiple choice question they will be prompted the 4 options for answers to that question. When they have completed the input for

the question they will be given a message indicating the success or failure of inserting the question and the main menu will be displayed until they choose to exit.

4.2 Software Interfaces

We will use the SQLite JDBC Driver, version 3.8.11.2, to connect to our SQLite database from our Java programs.

4.3 Communications Interfaces

No communication functions required by this product.

5. Other Nonfunctional Requirements

5.1 Security Requirements

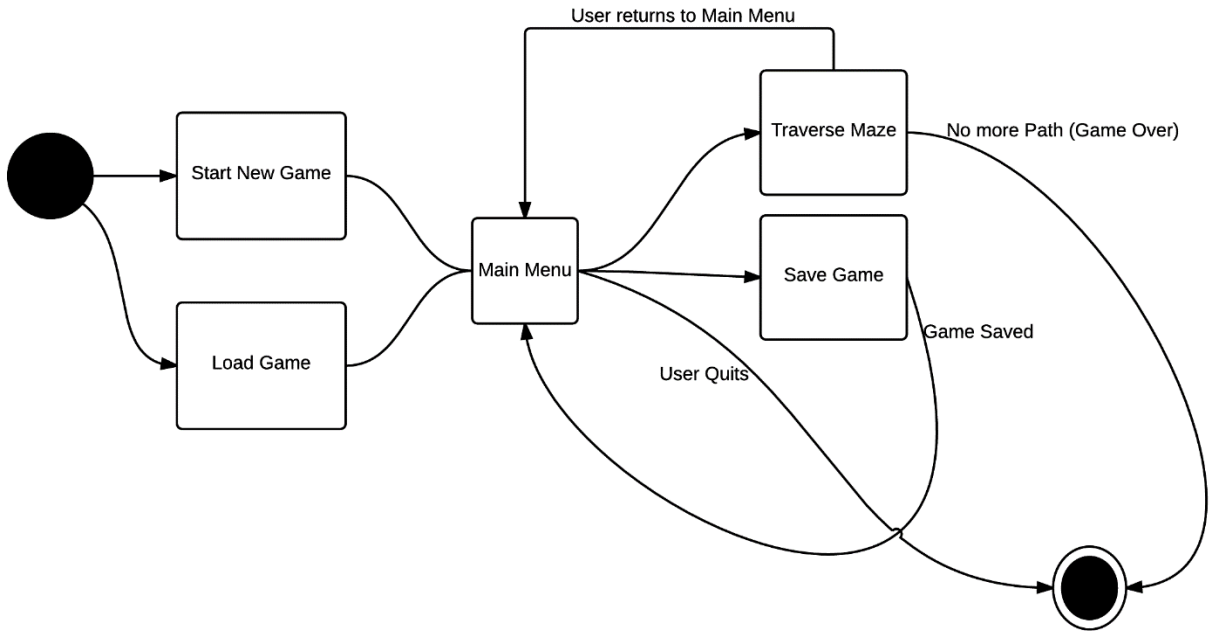
All input is scrubbed at the time it is accepted from the user to assure that it matches the structure and type of input expected for a given input request. PreparedStatements in Java are used to prevent SQL injection when interacting with the database using input from the user.

5.2 Software Quality Attributes

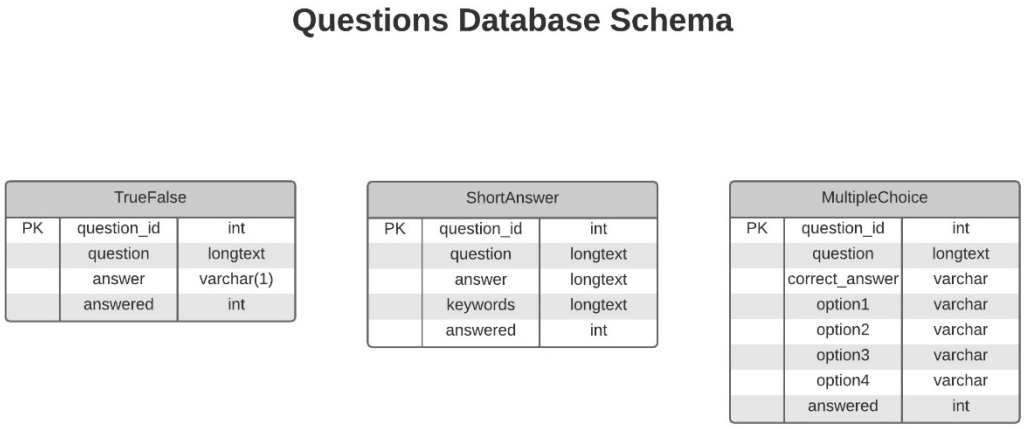
All components of the system will be reasonably tested by unit tests, primarily using JUnit 4 in Eclipse.

6. Appendix

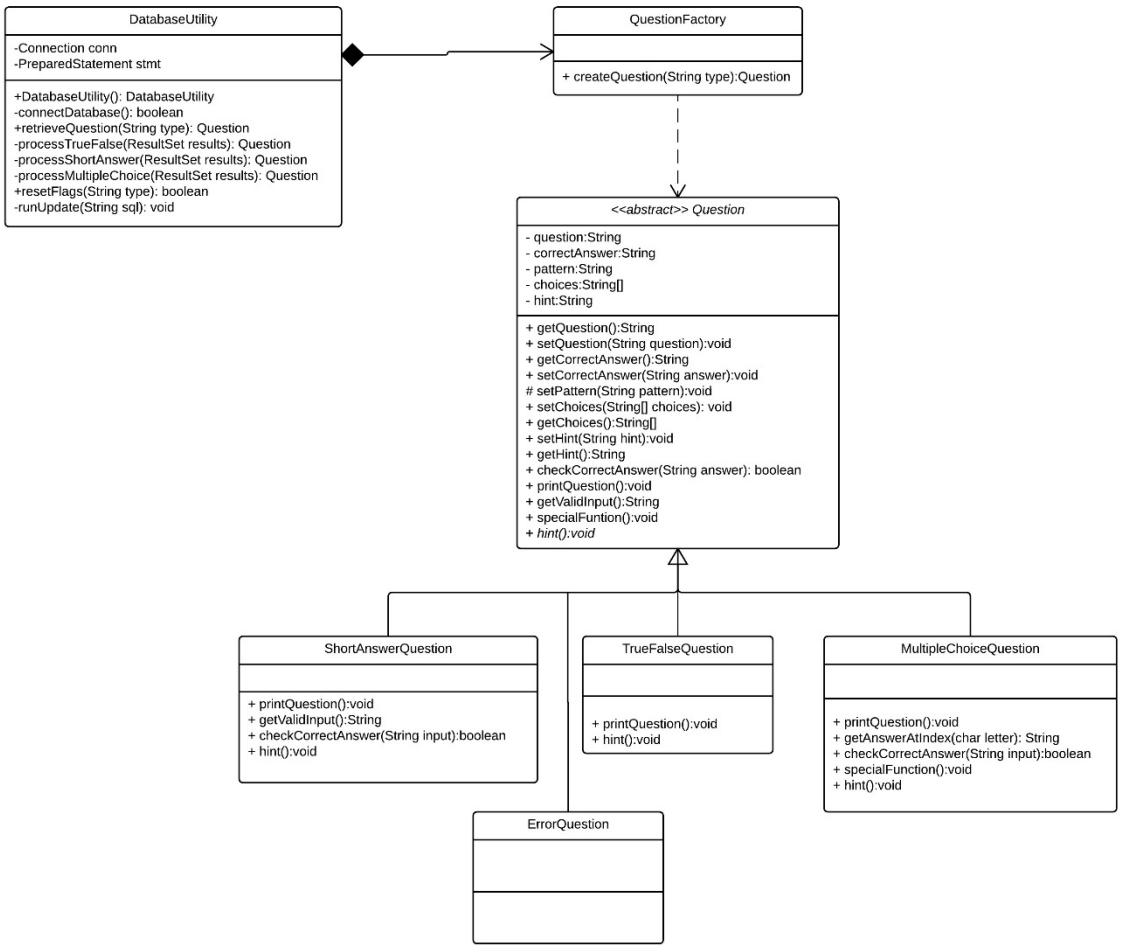
6.1 State Diagram for the Flow of the Game



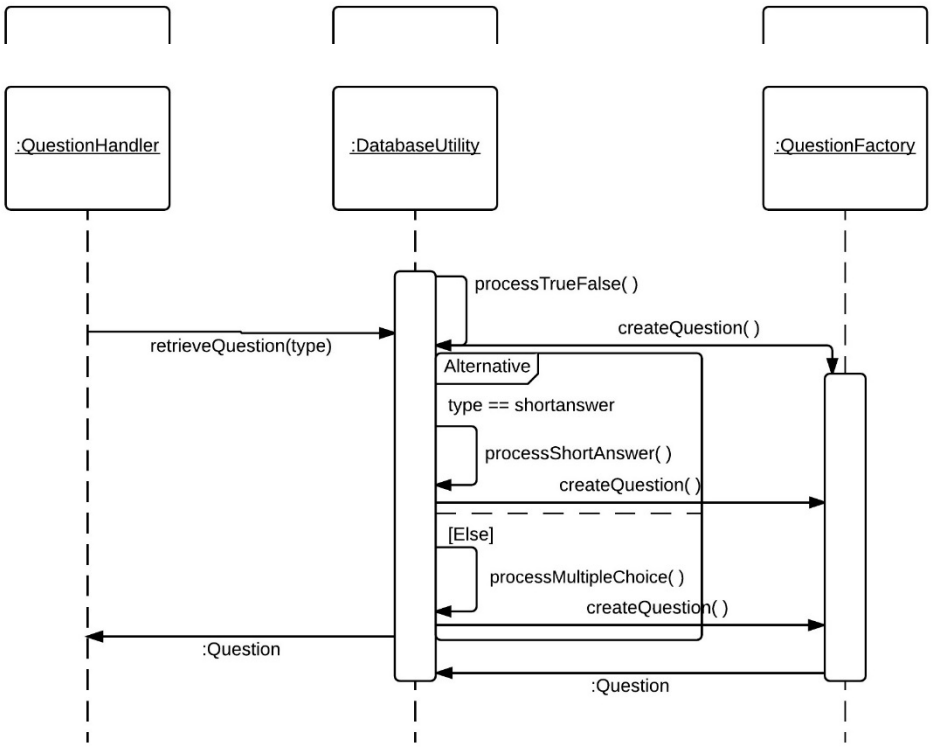
6.2 Database Schema



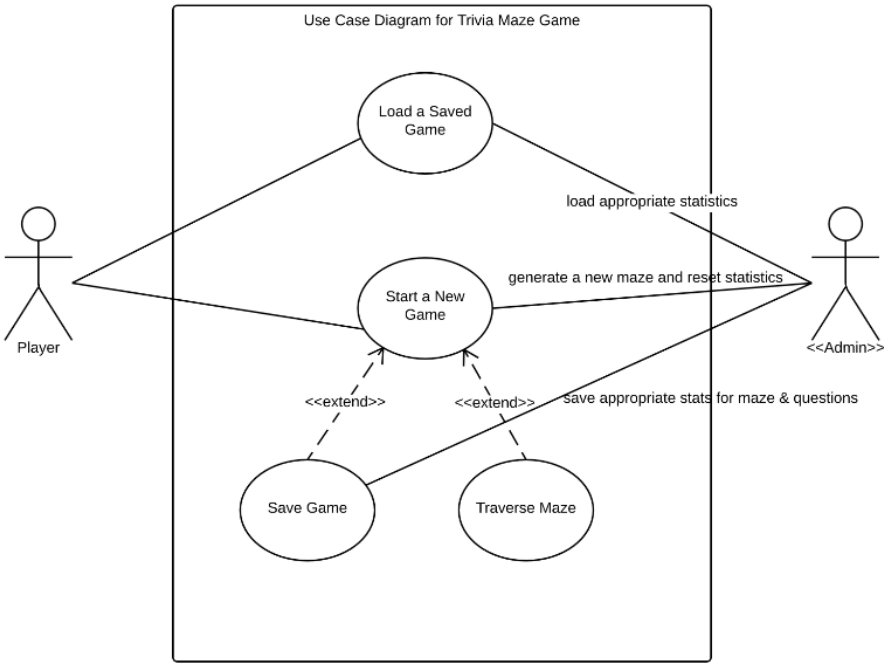
6.3 Class Diagram for the Question Hierarchy and Database Utility



6.4 Sequence Diagram for Question Retrieval



6.5 Use Case Diagram



6.6 Use Case Description

* Name: Answer a question
* Description: Player (user) attempts to move into a room in the maze containing a question and is prompted to answer one of three types of question. If the player answers the question correctly, he/she may enter the room; otherwise, the room is walled off and the player moves back to his/her most recently visited position.
* Level: Primary task
* Primary Actor: Player
* Preconditions:
    1. The player is active in the game.
    2. The maze is populated with open cells, question cells, and wall cells.
    3. Maze traversal is implemented, i.e. the player can choose to move north, east, south, or west into open cells and cells containing questions, but not cells that are walls.
    4. A database with valid questions of each type exists.
* Success End Condition: Player successfully enters the room, i.e. the room is marked as 'visited' within the maze.
* Failure End Condition: The room is walled off and the player is moved back to the most recently visited position.
* Trigger: Player lands on a cell (enters a room) in the maze that is coded as one of the three types of questions: true/false, multiple choice, or short answer.
* Main Success Scenario:
    1. Player tries to enter a room in the maze that contains a question.
    2. Player is presented with a trivia question either true/false, multiple choice, or short answer.
    3. Player is prompted to answer the question.
    4. Player's answer is verified with the correct/acceptable answer for that question.
    5. Player successfully moves into the room and the cell is marked as visited so that the player may move in and out of that room freely in the future.
* Extensions:

3a. Player provides an unacceptable answer choice for that particular type question.
    3a1. An error message is printed indicating an invalid response and the player is re-prompted (go back to step 2) until the player provides an acceptable response.

5a. Player answers the question incorrectly.
    5a1. The room is walled off and the player is moved back to the most recently visited position.
    5a2. Check if there still exists a path to the target position of the maze.
    5a3. If an open path exists, go to step 1, else the current game is ended and the player is notified.