Finterly Hu
R0688968

Basic Programming Maze Project 2017-2018

Game Explanation

A Beagle puppy (player), chased a car (start) and is now far way from home (end).

Player navigates maze using N, S, E, W keys. X to exit. Player begins with points equal to 2 times the size of the maze and loses 1 point for each failed move or retraced step. Game ends when player has 0 or fewer points, reaches end, or chooses to exit.

Walls and fake walls appear as --- or | , doors as d or –d-, and breakable walls as b or –b-. Fake walls are revealed and removed after player crosses them.

Hidden items are key, hammer , and trophy and added to player possession when found. Keys can open any doors and hammers can be reused on breakable walls. A trophy doubles the player's points.

Class Description

Color Key:	**Class**	***args***	method	methodsFromOtherClass	variable	*enum*

1. **Main**
   - Launches program taking player name as ***arg[0]*** and maze text file path as ***arg[1]***.
   - Constructs new player, game, and maze.
   - initializeMaze by loadMaze, validateMaze, and printMaze 2D in Console.
   - Interacts with user to play game.
   - If player reaches *end*, status = 0, processScore and writeScoretoFile high_score.txt.

2. **Game**
   Interact with user with Scanner to choosePlay. 'X' exits status 2 in **Main**.
   If user enters 'P', intializePlay setPosition of player to *start* getStartPosition; setPoints of player; and getCell currentCell.

   Game continues while player has validPoints. 'X' exits with status 2 in **Main**. incrMove plus 1 when player successfully move. incrFail plus 1 when player fails to move (e.g. walks into *wall)*. Player points minus 1 for each failed move and each time player retraces his/her steps.

   Method play interacts with user. N, S, E, W input determines **Wall** type player must cross and coordinate of nextCell. passWall checks if wall of cell is passable, and allowPass checks if the nextCell exists. If true, player move and updateCellContent of nextCell. Hidden items found in updateCellContent are added to possession. If nextCell content is *trophy*, player points double. If content is *end*, exits play status 0 in **Main.** Otherwise, continue nextCell becomes currentCell.

   To pass door or breakable wall, player must hasPossession key or hammer. When player successfully crosses a *door, breakable,* or *fake* wall, setWall sets crossed wall to **Wall** *no wall*.

Finterly Hu
R0688968

3. **Player**
   Player begins at *start* with points equal to 2 times getMazeSize*.* Player has attributes name, moves, fails, points, **Intpair** position, and List<**Content**> possession.

4. **IntPair**
   Holds position coordinates xpos, ypos.

5. **Maze**
   loadMaze reads, trims, and parses input text file lines, values are set to **Cell** in List<List<**Cell**>> cellList. Order of rows is reverse in cellList as intended by maze input file.

   isValidXY checks XYcoordinates are non-negative integers between 0 and 40, mazes larger than 40 x 40 are not suitable for console printing.

   After cellList is loaded, validateMaze calls: checksForNull, maze isRegularRectangle, validateWall of cell matches neighboring cell wall, and validateStartEnd (maze must have exactly one start and end). HashMap<Cell, String> lineInfoMap stores line information of original input file which getLineInfo retrieves to generate helpful error messages.

   printMaze prints a 2D representation of cellList.

6. **Cell**
   Each **Cell** represents a line in maze input text file. **Wall** type attributes northwall, southwall, eastwall, and westwall; **Content** type attribute content.

   Method incrVisit increases cell visit by 1 and getVisit returns number of past visits in **Game** when player retraces step.

7. **Wall**
   Enumeration of wall types **Cell** can have. fromString throws exception if input file contains unrecognized value.

8. **Content**
   Enumeration of contents **Cell** can contain. fromString throws exception if input file contains unrecognized value.

9. **Score**
   Score class has attributes name, mazeName, moves, fails, and points.

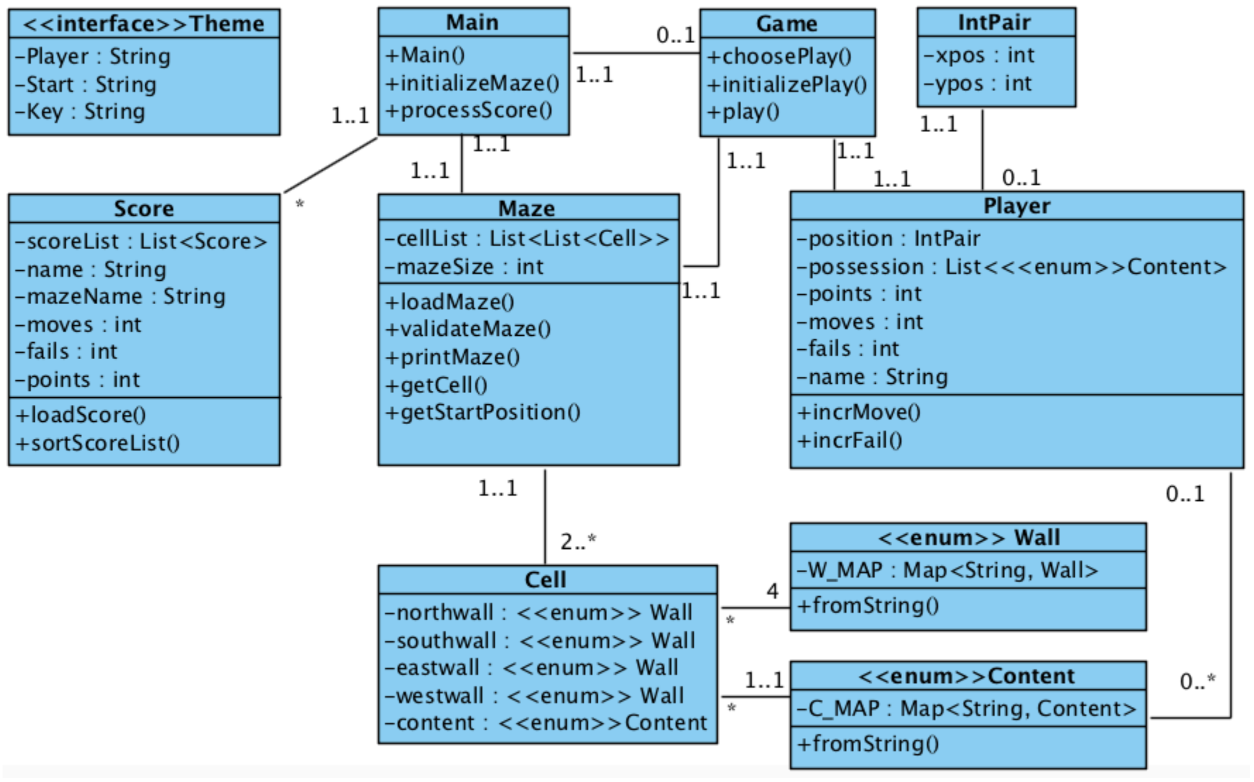
   If high_score.txt exists, loadScore reads, trims, and parses lines, values are set to **Score** in List<List<**Score**>> scoreList.

   sortScoreList sorts scores in scoreList by descending order of points. scoreList is called by writeScoretoFile in **Main**.

10. **Theme**
    Interface holds the unicode strings for characters printed in the console. Implemented in other classes as needed.

Finterly Hu
R0688968

Class Relationship



Strength/Weakness/Comments

Strengths: Code is (hopefully) clean and readable, without duplicate or very long methods.

Project thoroughly validates maze and prints detailed input file line information when exceptions occur. Exceptions are handled as gracefully.

Theme, Wall, and Contents class are easily modified to include new features such as a different theme or additional objects.

Weaknesses: Program only works with regular rectangle mazes. Irregular shapes such as diamonds will not work.

Project does not implement GUI and is therefore limited. Project could also be extended to allow user to choose from several mazes or reset a maze. The possibilities are endless.

Project is not perfect and therefore likely has bugs and requires maintenance.

Comments: In the beginning, the project was challenging because of my unfamiliarity with Java. Later, I was able to return to and improve earlier code. It was a rewarding experience and the desire to improve the project made coding addictive.