DOCUMENTATION

Jose Alejandro Garcia Marcos

2020

Functional requirements

FR1: Reading inputs from console. The program can read inputs from console and use them for its different functions.

FR2: Displaying a menu. The program can display a menu at the start and return to it once the option selected has been completed. It uses number inputs from console to operate.

FR3: Start a game with the given parameters. The program creates a matrix with the given dimensions (26 columns maximum) and creates the number of mirrors given as well (at random positions and inclinations). The number of mirrors provided must not be greater than the number of spaces in the matrix (referred to as cells from this point on).

FR4: Displaying Cell matrix. When the user starts playing, the program displays a matrix representing the cells and mirrors within them (not appearing at first). As the user makes moves and/or finds mirrors, the matrix will show this.

FR5: Finding a Specific Cell. The program is able to get to a Cell given its coordinates. This is used in the laser-shooting and mirror-locating functions.

FR6: Keeping track of a user's username. The program keeps track of the username provided in the game parameters. When the user ends a game, the score is associated with the username and saved.

FR7: Keeping track of score. The program has a simple game point system. For every move (shooting or tracking) the score increases by one. If the user returns to menu before finding all mirrors, a penalization of a 100 point per mirror not found is used.

FR8: Saving scores. The program is able to keep track of the scores of users who finished games (by finding all mirrors or typing menu). This information is persistent through serialization (binary files kept in data folder).

FR9: Loading scores. The program is able to load previously saved scores through serialization. If there is a file with scores saved, it will load it, if not, it starts from 0.

FR10: Shooting lasers. The program is able to read the location of the cell from which the user tries to shoot. The cell must exist within the matrix. The user must input the coordinates of the cell they want to shoot from. That cell's row must be input first (as a number) followed by its column (as a letter). If the cell is a corner cell, it must be followed by either "*h*" (horizontal) or "*v*" (vertical) indicating the direction of the shot. If the input is correct, the program will display a matrix with "*S*" (start) where the laser started and "E" (exit) where it exited. If the laser starts and ends in the same cell, the matrix displays "B" (both).

RF11: Locating mirrors. The program is able to tell a user if their guess at a mirror's position and orientation is right. While playing, to locate a mirror, the user must start their move input with an "*L*" (locate) followed by the cell's coordinates (as described in FR7) and ending with either "*L*" (left) or "*R*" (right) indicating the orientation of the mirror. If the user's guess is right the mirror will be displayed (as either “\” or “/” depending on its orientation); if the guess Is not right, an “*X*” will be displayed in the cell the user guessed.

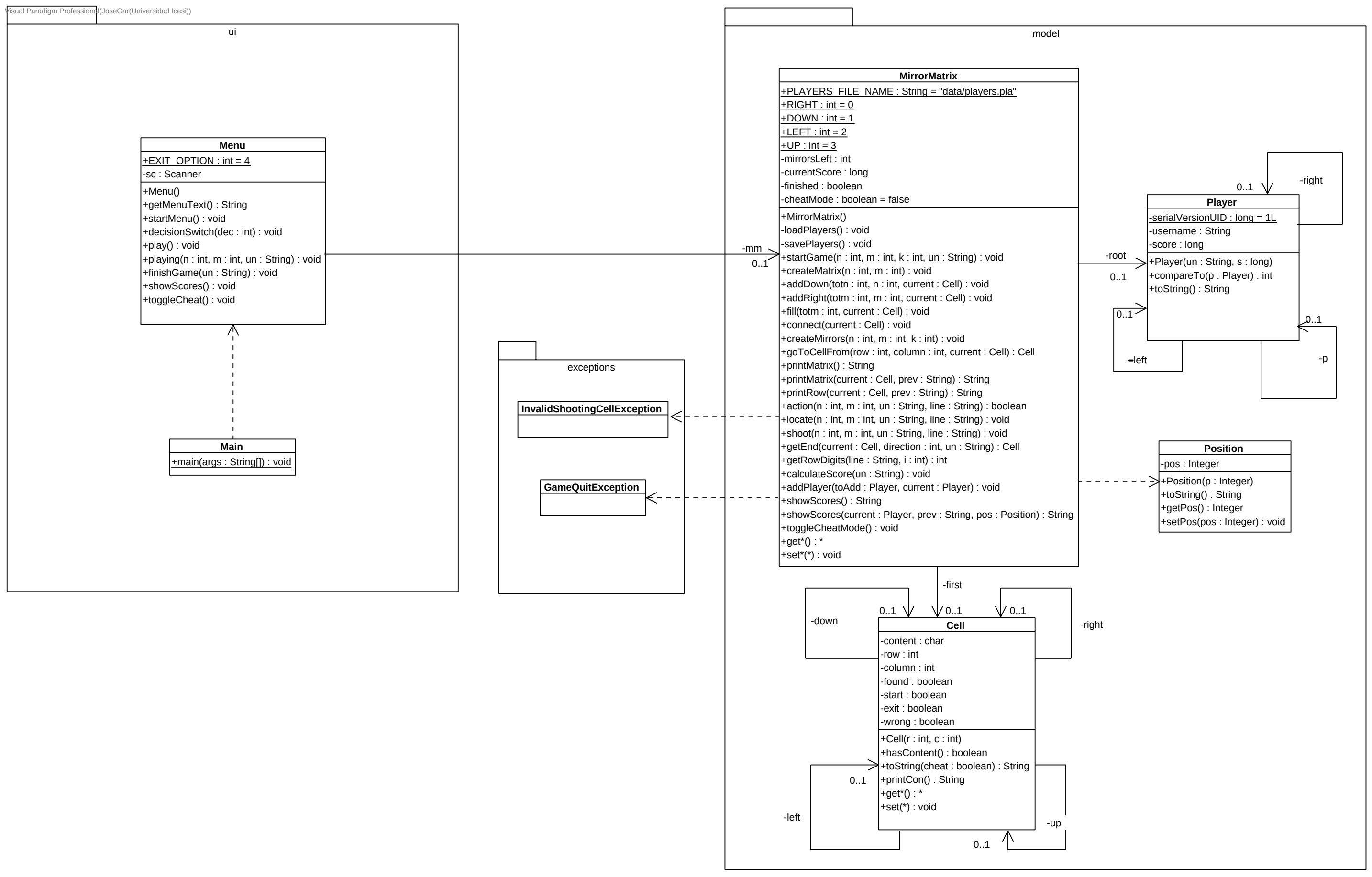
RF12: Keeping track of mirror remaining. The program can keep track of the mirrors left unfound in the matrix. This number starts at the number of mirrors provided by the user in the game’s parameters and it decreases as the user finds mirrors.

RF13: Ending a game. The program ends the game when the user has found all mirrors or if the user types “*menu*” as a move input while playing (issuing a 100-point penalty per mirror not found).

RF14: Displaying scores. The program is able to display the usernames of the users that have played along with their scores and position (position being first). This is a menu option.

RF15: Cheat mode. The program has a menu option to toggle cheat mode. With this, the user is able to see the mirrors at all times, as well as their coordinates.

Class diagram


ui
Menu
+EXIT_OPTION : int = 4
-sc : Scanner
+Menu()
+getMenuText() : String
+startMenu() : void
+decisionSwitch(dec : int) : void
+play() : void
+playing(n : int, m : int, un : String) : void
+finishGame(un : String) : void
+showScores() : void
+toggleCheat() : void
Main
+main(args : String[]) : void
exceptions
InvalidShootingCellException
GameQuitException
model
-mm
0..1
MirrorMatrix
+PLAYERS_FILE_NAME : String = "data/players.pla"
+RIGHT : int = 0
+DOWN : int = 1
+LEFT : int = 2
+UP : int = 3
-mirrorsLeft : int
-currentScore : long
-finished : boolean
-cheatMode : boolean = false
+MirrorMatrix()
-loadPlayers() : void
-savePlayers() : void
+startGame(n : int, m : int, k : int, un : String) : void
+createMatrix(n : int, m : int) : void
+addDown(totn : int, n : int, current : Cell) : void
+addRight(totm : int, m : int, current : Cell) : void
+fill(totm : int, current : Cell) : void
+connect(current : Cell) : void
+createMirrors(n : int, m : int, k : int) : void
+goToCellFrom(row : int, column : int, current : Cell) : Cell
+printMatrix() : String
+printMatrix(current : Cell, prev : String) : String
+printRow(current : Cell, prev : String) : String
+action(n : int, m : int, un : String, line : String) : boolean
+locate(n : int, m : int, un : String, line : String) : void
+shoot(n : int, m : int, un : String, line : String) : void
+getEnd(current : Cell, direction : int, un : String) : Cell
+getRowDigits(line : String, i : int) : int
+calculateScore(un : String) : void
+addPlayer(toAdd : Player, current : Player) : void
+showScores() : String
+showScores(current : Player, prev : String, pos : Position) : String
+toggleCheatMode() : void
+get*() : *
+set*(*) : void
-root
0..1
Player
-serialVersionUID : long = 1L
-username : String
-score : long
+Player(un : String, s : long)
+compareTo(p : Player) : int
+toString() : String
-right
0..1
0..1
-left
0..1
-p
Position
-pos : Integer
+Position(p : Integer)
+toString() : String
+getPos() : Integer
+setPos(pos : Integer) : void
-first
-down
0..1
0..1
0..1
-right
Cell
-content : char
-row : int
-column : int
-found : boolean
-start : boolean
-exit : boolean
-wrong : boolean
+Cell(r : int, c : int)
+hasContent() : boolean
+toString(cheat : boolean) : String
+printCon() : String
+get*() : *
+set(*) : void
0..1
-left
-up
0..1