Programming Technology
Assignment 3

Name: Aliia Bazarkulova
Neptun code: HZ4BV8
Task: 3 (Labyrinth)

Description:

Create the Labyrinth game, where objective of the player is to escape from this labyrinth. The player starts at the bottom left corner of the labyrinth. He has to get to the top right corner of the labyrinth as fast he can, avoiding a meeting with the evil dragon. The player can move only in four directions: left, right, up or down. There are several escape paths in all labyrinths. The dragon starts off from a randomly chosen position, and moves randomly in the labyrinth so that it choose a direction and goes in that direction until it reaches a wall. Then it chooses randomly a different direction. If the dragon gets to a neighboring field of the player, then the player dies. Because it is dark in the labyrinth, the player can see only the neighboring fields at a distance of 3 units. Record the number of how many labyrinths did the player solve, and if he loses his life, then save this number together with his name into the database. Create a menu item, which displays a highscore table of the players for the 10 best scores. Also, create a menu item which restarts the game. Take care that the player and the dragon cannot start off on walls.

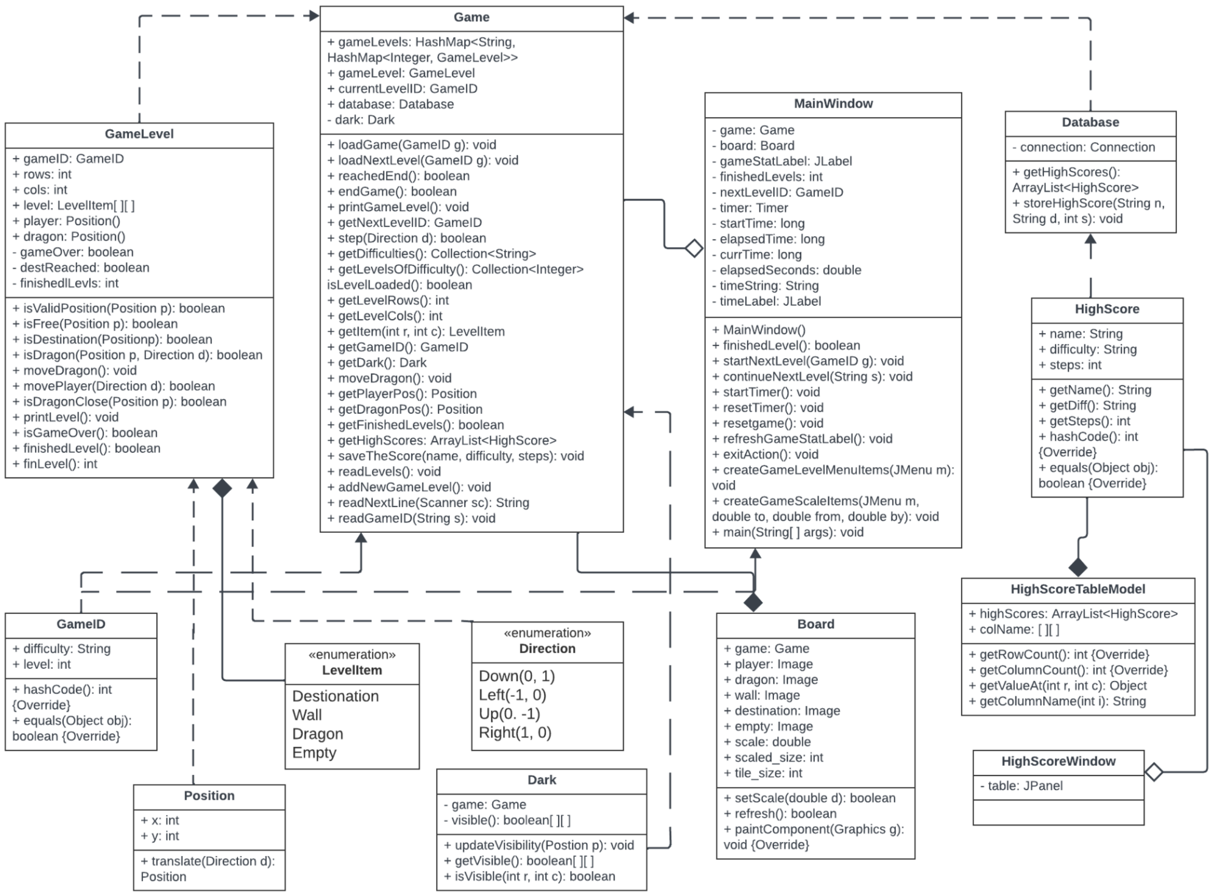
Description of classes and methods:

- MainWindow: the main class where the game is instantiated and called
    - MainWindow(): the constructor creates all the needed menu items, starts the game and keeps playing until ends
    - finishedLevel(): helps to find if the level was finished to jump to next
    - startNextLevel(): starts the next level depending what was the finished current
    - continueNextLevel(): starts the next difficulty, if the player played the last level of previous and wants to continue
    - startTimer(): starts the timer
    - resetTimer(): sets the time to 0
    - resetgame(): starts the game again
    - refreshGameStatLabel(): keeps refreshing the game statistics

    - createGameLevelMenuItems(): creates the possible menu items
    - createGameScaleMenuItems(): creates the possible scales
- Board: this board initializes the board of the game
    - setScale(): sets the scale of the game that player has chosen
    - refresh(): refreshes the board
    - painComponent(): paints the board depending on the map and player position
- Dark: this class lets us see only the 3 neighboring fields of the player
    - updateVisibilty(): the visibility of the player will change depending on his movement
    - getVisible(): getter for the visible 2D array
    - isVisible(): getter of the specific cell's visibility
- HighScoreWindow: this class creates the window where the database of scores is displayed
- HighScoreTableModel: this class creates the table for the database and determines how they will be displayed
- HighScore: this class creates objects of HighScore class
    - getName(), getDiff(), getSteps(): getters of HighScore attributes
- Database: this class connects the database and the game
    - getHighScores(): returns the list of scores that are in database
    - storeHighScore(): stores the result of the last played game
- Position: this class determines the positions of player and dragon
- Direction: enum type that has 4 different moves - Up, Down, Right, Left
- LevelItem: enum type to represent different items of the game, such as: player, wall, dragon, empty, destination
- GameID: this class determines the type of games can be depending on their difficulty and level
- GameLevel: this class creates the logic of the game, where we can start the game, play it, jump between levels, win or lose
    - isValidPosition(): lets us know if the move is within the bounds
    - isFree(): helps us to determine if the cell is empty to move there
    - isDestination(): helps to determine if the cell is the destination
    - isDragon(): helps to determine if dragon has reached the player
    - moveDragon(): will move the dragon randomly if the move is valid
    - movePlayer(): will move the player, hopefully towards the end
    - isDragonClose(): helps to determine if the dragon is close, because then the player dies
    - printLevel(): helps to print the level we're currently playing
    - isGameOver(): to determine if we have lost
    - finishedLevel(): to determine if we have reached the destination

    - finLevels(): keeps track of the levels player has finished without dying
- Game: this class creates the game, and brings all of the classes together
    - loadGame(): starts the initial game
    - loadNextLevel(): loads the next level of the game
    - reachedEnd(): helps to find if the player reached the end of the labyrinth
    - endGame(): helps to find if the player has been eaten by the dragon
    - printGameLevel(): prints the current level
    - getNextGameID(): gets what should be the next level to move there
    - step(): helps to determine if the next step is good
    - getDifficulties(), getLevelsOfDifficulty(): getters for the difficulty of the game
    - getLevelRows(), getColRows(): getters for the number of rows and cols in the game
    - getItem(), getGameID(), getDark(): getters for different attributes of the game
    - getPlayerPos(), getDragonPos(): getters for the current positions of the player and the dragon
    - getFinishedLevel(), getHighScores(): getters for the numbers of labyrinths solved and the scores in the database
    - saveTheScore(): saves the score to the database
    - addNewGameLevel(): adds the game levels that were read from the levels
    - readNextLine(): to make sure that we added all of the levels
    - readGameId(): to read all of the possible GameIDs
    - readLevels(): start reading the levels

UML diagram:


Game
+ gameLevels: HashMap<String, HashMap<Integer, GameLevel>>
+ gameLevel: GameLevel
+ currentLevelID: GameID
+ database: Database
- dark: Dark
+ loadGame(GameID g): void
+ loadNextLevel(GameID g): void
+ reachedEnd(): boolean
+ endGame(): boolean
+ printGameLevel(): void
+ getNextLevelID: GameID
+ step(Direction d): boolean
+ getDifficulties(): Collection<String>
+ getLevelsOfDifficulty(): Collection<Integer>
isLevelLoaded(): boolean
+ getLevelRows(): int
+ getLevelCols(): int
+ getItem(int r, int c): LevelItem
+ getGameID(): GameID
+ getDark(): Dark
+ moveDragon(): void
+ getPlayerPos(): Position
+ getDragonPos(): Position
+ getFinishedLevels(): boolean
+ getHighScores: ArrayList<HighScore>
+ saveTheScore(name, difficulty, steps): void
+ readLevels(): void
+ addNewGameLevel(): void
+ readNextLine(Scanner sc): String
+ readGameID(String s): void
GameLevel
+ gameID: GameID
+ rows: int
+ cols: int
+ level: LevelItem[ ][ ]
+ player: Position()
+ dragon: Position()
- gameOver: boolean
- destReached: boolean
- finishedlLevls: int
+ isValidPosition(Position p): boolean
+ isFree(Position p): boolean
+ isDestination(Positionp): boolean
+ isDragon(Position p, Direction d): boolean
+ moveDragon(): void
+ movePlayer(Direction d): boolean
+ isDragonClose(Position p): boolean
+ printLevel(): void
+ isGameOver(): boolean
+ finishedLevel(): boolean
+ finLevel(): int
MainWindow
- game: Game
- board: Board
- gameStatLabel: JLabel
- finishedLevels: int
- nextLevelID: GameID
- timer: Timer
- startTime: long
- elapsedTime: long
- currTime: long
- elapsedSeconds: double
- timeString: String
- timeLabel: JLabel
+ MainWindow()
+ finishedLevel(): boolean
+ startNextLevel(GameID g): void
+ continueNextLevel(String s): void
+ startTimer(): void
+ resetTimer(): void
+ resetgame(): void
+ refreshGameStatLabel(): void
+ exitAction(): void
+ createGameLevelMenuItems(JMenu m): void
+ createGameScaleItems(JMenu m, double to, double from, double by): void
+ main(String[ ] args): void
Database
- connection: Connection
+ getHighScores(): ArrayList<HighScore>
+ storeHighScore(String n, String d, int s): void
HighScore
+ name: String
+ difficulty: String
+ steps: int
+ getName(): String
+ getDiff(): String
+ getSteps(): int
+ hashCode(): int {Override}
+ equals(Object obj): boolean {Override}
HighScoreTableModel
+ highScores: ArrayList<HighScore>
+ colName: [ ][ ]
+ getRowCount(): int {Override}
+ getColumnCount(): int {Override}
+ getValueAt(int r, int c): Object
+ getColumnName(int i): String
HighScoreWindow
- table: JPanel
GameID
+ difficulty: String
+ level: int
+ hashCode(): int {Override}
+ equals(Object obj): boolean {Override}
«enumeration»
LevelItem
Destionation
Wall
Dragon
Empty
«enumeration»
Direction
Down(0, 1)
Left(-1, 0)
Up(0. -1)
Right(1, 0)
Board
+ game: Game
+ player: Image
+ dragon: Image
+ wall: Image
+ destination: Image
+ empty: Image
+ scale: double
+ scaled_size: int
+ tile_size: int
+ setScale(double d): boolean
+ refresh(): boolean
+ paintComponent(Graphics g): void {Override}
Position
+ x: int
+ y: int
+ translate(Direction d): Position
Dark
- game: Game
- visible(): boolean[ ][ ]
+ updateVisibility(Postion p): void
+ getVisible(): boolean[ ][ ]
+ isVisible(int r, int c): boolean