Programming Technology

Assignment-3

Documentation for Task-1

YOGI BEAR

Written and implemented by

Zayar Htet (DRCVG2)

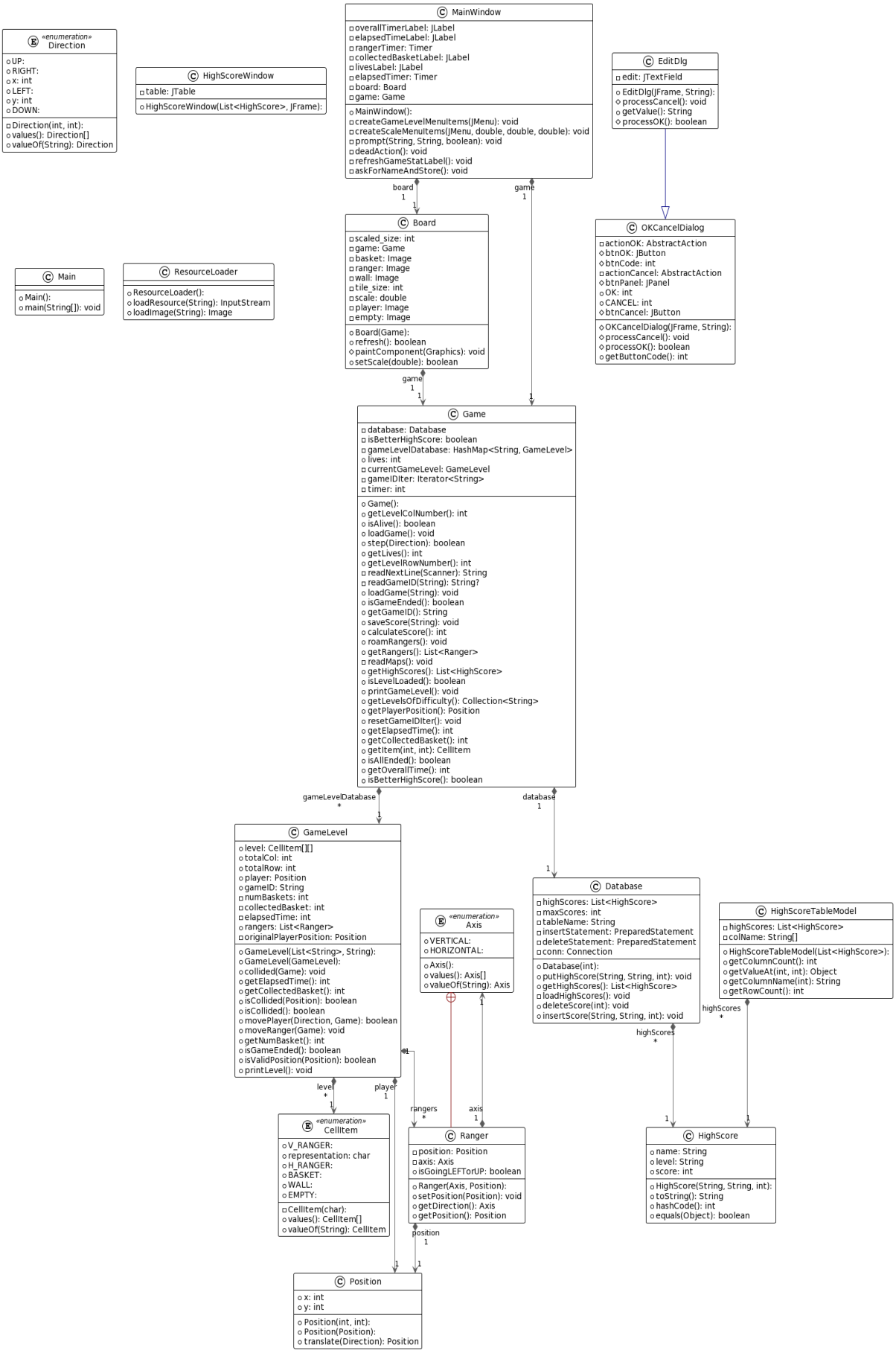
Task Description

Yogi Bear wants to collect all the picnic baskets in the forest of the Yellowstone National Park. This park contains mountains and trees, that are obstacles for Yogi. Besided the obstacles, there are rangers, who make it harder for Yogi to collect the baskets. Rangers can move only horizontally or vertically in the park. If a ranger gets too close (one unit distance) to Yogi, then Yogi loses one life. (It is up to you to define the unit, but it should be at least that wide, as the sprite of Yogi.)

If Yogi still has at least one life from the original three, then he spawns at the entrance of the park. During the adventures of Yogi, the game counts the number of picnic baskets, that Yogi collected. If all the baskets are collected, then load a new game level, or generate one. If Yogi loses all his lives, then show a popup messagebox, where the player can type his name and save it to 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.

Solution

UML Diagram


«enumeration»
Direction
UP:
RIGHT:
x: int
LEFT:
y: int
DOWN:
Direction(int, int):
values(): Direction[]
valueOf(String): Direction
HighScoreWindow
table: JTable
HighScoreWindow(List<HighScore>, JFrame):
MainWindow
overallTimerLabel: JLabel
elapsedTimeLabel: JLabel
rangerTimer: Timer
collectedBasketLabel: JLabel
livesLabel: JLabel
elapsedTimer: Timer
board: Board
game: Game
MainWindow():
createGameLevelMenuItems(JMenu): void
createScaleMenuItems(JMenu, double, double, double): void
prompt(String, String, boolean): void
deadAction(): void
refreshGameStatLabel(): void
askForNameAndStore(): void
EditDlg
edit: JTextField
EditDlg(JFrame, String):
processCancel(): void
getValue(): String
processOK(): boolean
board
1
1
game
1
Board
scaled_size: int
game: Game
basket: Image
ranger: Image
wall: Image
tile_size: int
scale: double
player: Image
empty: Image
Board(Game):
refresh(): boolean
paintComponent(Graphics): void
setScale(double): boolean
OKCancelDialog
actionOK: AbstractAction
btnOK: JButton
btnCode: int
actionCancel: AbstractAction
btnPanel: JPanel
OK: int
CANCEL: int
btnCancel: JButton
OKCancelDialog(JFrame, String):
processCancel(): void
processOK(): boolean
getButtonCode(): int
Main
Main():
main(String[]): void
ResourceLoader
ResourceLoader():
loadResource(String): InputStream
loadImage(String): Image
game
1
1
1
Game
database: Database
isBetterHighScore: boolean
gameLevelDatabase: HashMap<String, GameLevel>
lives: int
currentGameLevel: GameLevel
gameIDIter: Iterator<String>
timer: int
Game():
getLevelColNumber(): int
isAlive(): boolean
loadGame(): void
step(Direction): boolean
getLives(): int
getLevelRowNumber(): int
readNextLine(Scanner): String
readGameID(String): String?
loadGame(String): void
isGameEnded(): boolean
getGameID(): String
saveScore(String): void
calculateScore(): int
roamRangers(): void
getRangers(): List<Ranger>
readMaps(): void
getHighScores(): List<HighScore>
isLevelLoaded(): boolean
printGameLevel(): void
getLevelsOfDifficulty(): Collection<String>
getPlayerPosition(): Position
resetGameIDIter(): void
getElapsedTime(): int
getCollectedBasket(): int
getItem(int, int): CellItem
isAllEnded(): boolean
getOverallTime(): int
isBetterHighScore(): boolean
gameLevelDatabase
*
1
database
1
1
GameLevel
level: CellItem[][]
totalCol: int
totalRow: int
player: Position
gameID: String
numBaskets: int
collectedBasket: int
elapsedTime: int
rangers: List<Ranger>
originalPlayerPosition: Position
GameLevel(List<String>, String):
GameLevel(GameLevel):
collided(Game): void
getElapsedTime(): int
getCollectedBasket(): int
isCollided(Position): boolean
isCollided(): boolean
movePlayer(Direction, Game): boolean
moveRanger(Game): void
getNumBasket(): int
isGameEnded(): boolean
isValidPosition(Position): boolean
printLevel(): void
«enumeration»
Axis
VERTICAL:
HORIZONTAL:
Axis():
values(): Axis[]
valueOf(String): Axis
1
1
Database
highScores: List<HighScore>
maxScores: int
tableName: String
insertStatement: PreparedStatement
deleteStatement: PreparedStatement
conn: Connection
Database(int):
putHighScore(String, String, int): void
getHighScores(): List<HighScore>
loadHighScores(): void
deleteScore(int): void
insertScore(String, String, int): void
HighScoreTableModel
highScores: List<HighScore>
colName: String[]
HighScoreTableModel(List<HighScore>):
getColumnCount(): int
getValueAt(int, int): Object
getColumnName(int): String
getRowCount(): int
highScores
*
highScores
*
level
*
1
player
1
rangers
*
axis
1
«enumeration»
CellItem
V_RANGER:
representation: char
H_RANGER:
BASKET:
WALL:
EMPTY:
CellItem(char):
values(): CellItem[]
valueOf(String): CellItem
Ranger
position: Position
axis: Axis
isGoingLEFTorUP: boolean
Ranger(Axis, Position):
setPosition(Position): void
getDirection(): Axis
getPosition(): Position
HighScore
name: String
level: String
score: int
HighScore(String, String, int):
toString(): String
hashCode(): int
equals(Object): boolean
position
1
1
1
Position
x: int
y: int
Position(int, int):
Position(Position):
translate(Direction): Position

Classes and Methods

MainWindow

This class extends JFrame and the parent class of the windows. This will initiate the window dimension and on click exit button feature. The MainWindow has 2 parts which are Game Statistics Label part and Game Board part. Both parts are represented with JPanel and stored as member fields.

The constructor will initialize the all the necessary field and define the KeyListener and Timer.

1. **prompt() : void** – PRIVATE
   This will stop all the timer and prompt the message with JOptionPane and JDialog Text Editor.
2. **deadAction() : void** - PRIVATE
   Will check if the player is still alive and if not, it will end the game and ask the player name.
3. **askForNameAndStore() : void** - PRIVATE
   Will Prompt the JDialog with text editor.
4. **createGameLevelMenuItems(p: Point) : void** - PRIVATE
   Create the menu item for the MainWindow.
5. **refreshGameStatLabel() : String** – PUBLIC ABTRACT
   Will refresh the game statistics label.

Board

This class extends the JPanel and will create a game board. Parsing the model and transforming into the User Interface will be done in this class by overriding paintComponent() method of the JPanel.

Game

This class will read the file and store all the data in the list and will iterate the stored levels once the user has finished. All the features of the game are inside the GameLevel.java.

1. **readMaps() : void - PRIVATE**

   will parse the text file into the proper model.

2. **Proper getters and setters**

GameLevel

This class performs the main functionality of the game. All the incoming data are parsed and stored properly in the matrix and array respectively. Rangers are stored in the array and player is stored as a member variable.

1. **moveRanger(Game gl) : void** – PUBLIC
   This will move the ranger according to horizontally or vertically by 1 unit. If the roaming ranger collided with the player, then the lives of the player will be reduced, which is inside the Game gl.
2. **isCollided(Game gl) : boolean** – PRIVATE
   Will iterate the stored ranger and check if the player (x,y) values is equal to the position of the each ranger.
3. **movePlayer(Direction d) : Cell** – PUBLIC
   This will move the player to the proper direction if the direction is valid.

CellItem – Enum of value to represent the each cell.

Direction – Enum of value to represent the movement of the player and ranger.

Events and Event Handler

There are 3 event handler, which are **KeyPressed**, **Timer** and **ActionEvent** for menuitems. If the user clicked A,W,S,D or UP, DOWN, LEFT, RIGHT, then the position of the player in the model will be changed and the whole UI will be rendered again. Timer is for elapsed time and ranger movement. The ranger will be moved according to the model and the delay is 600 ms. The statistics label will be rendered during the delay for elapsed time is 1000 ms, ActionEvent will perform the on click event like showing scoreboard.

After every movement of player and rangers, the program will check if they are collided each other or not and perform necessary action like, reducing the lives. After that, the user interface will be rendered again.