Name: Aichurok Kanatbekova
Neptun ID: RIMR52
Date: November 14, 2021
Task number: 2

Task description:

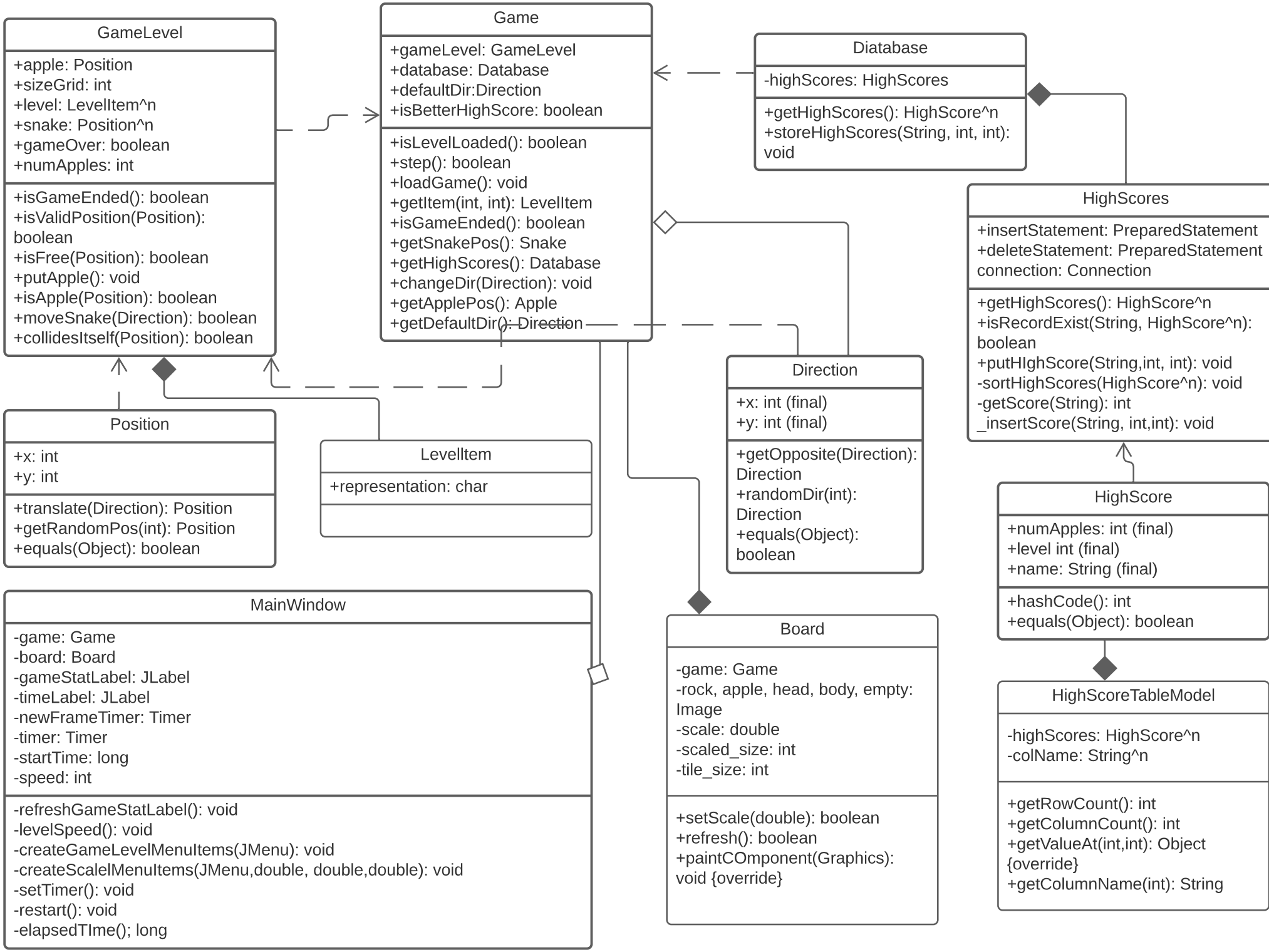
We have a rattlesnake in a desert, and our snake is initially two units long (head and rattler). We have to collect with our snake the foods on the level, that appears randomly. Only one food piece is placed randomly at a time on the level (on a field, where there is no snake).
The snake starts off from the center of the level in a random direction. The player can control the movement of the snake's head with keyboard buttons. If the snake eats a food piece, then its length grow by one unit.
It makes the game harder that there are rocks in the desert. If the snake collides with a rock, then the game ends. We also lose the game, if the snake goes into itself, or into the boundary of the game level.
In these situations show a popup messagebox, where the player can type his name and save it together with the amount of food eaten 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.

Decription of classes and methods:

Direction class - it is actually an enum which represents four directions to which snake can turn to.
 1)getOpposite(Direction) method - given a direction, returns its opposite direction. For example: if left, then right, if up, then down
 2)randomDir() method - returns random direction which will be needed for a snake to start moving in random direction in the beginning of the game
Game class - this class does the basis of the game.
 1)loadGame() method - it makes sure that snake starts the game in random direction.
 2)step() method - checks whether the game ended and if not, then it makes step namely, it moves the snake
 3)changeDir() method - checks if the given direction is not opposite of the default direction, if yes it modifies the default direction to given direction
GameLevel class - class which holds the essence of the game. Its constructor takes default direction which is indeed randomly chosen and this class makes sure that empty places, rocks, snake, and apple all appear in proper places on the board.
 1)isGameEnded() method - checks whether the snake is in three deadly conditions: it collided with rock, boundary, or itself
 2)copy constructor - which makes it easy to restart the game
 3)isValidPosition(Position) method - makes sure that given position is not out of the boundary
 4)isFree() method - makes sure given position is in valid position and that position is not apple, snake, or rock, but empty
 5)putApple() method - makes sure apple is put at random places where there are no rock, snake, pos is valid and free
 6)moveSnake(Direction) method - moves the snake's positions
 Namely, it checks whether the next position has apple in it or not
 And if yes, it increases the snake's size by adding new position to

linkedlist which represents a snake. If not, then it shifts snake's
position
7)isApple(Position) method - checks if in the given position we have an apple
8)collidesItself(Position) method - checks if snake has collided with its body.
LevelItem class - enum representing sprites of the game such as ROCK, APPLE, etc
Position class - helper class which makes it easy to move snake along the board
Database class - needed for storing scores
HighScore class - needed for storing scores
HighScores class - needed for storing scores
Board class - UI essence happends here, responsible for painting sprites, refreshing, repainting
loading the images, setting scales
HighScoreTableModel class - needed for viewing stored scores
HighScoreWindow class - needed for viewing stored scores
MainWindows class - makes the base UI components such as Menu, game board, menu items.
Also actionlisteners are implemented here. Updates the labels of the game such as current
score, elapsed time etc
1)levelSpeed() method - according to the level of the game, it sets the speed in which snake
should move, i.e. speed of repainting the frame
2)createLevelMenuItems(JMenu) - based on the user chosen level, it sets the speed
and restarts the game according to the set parameters
3)createScaleItems() emthod - changes the scale of the game board and menu. Zooming
4)setTimer() - calculates the elapsed time
5)restart() - restarts the game

//Further improvements - pause can be implemented to stop snake's movement so that it does not die


GameLevel
+apple: Position
+sizeGrid: int
+level: LevelItem^n
+snake: Position^n
+gameOver: boolean
+numApples: int
+isGameEnded(): boolean
+isValidPosition(Position): boolean
+isFree(Position): boolean
+putApple(): void
+isApple(Position): boolean
+moveSnake(Direction): boolean
+collidesItself(Position): boolean
Game
+gameLevel: GameLevel
+database: Database
+defaultDir:Direction
+isBetterHighScore: boolean
+isLevelLoaded(): boolean
+step(): boolean
+loadGame(): void
+getItem(int, int): LevelItem
+isGameEnded(): boolean
+getSnakePos(): Snake
+getHighScores(): Database
+changeDir(Direction): void
+getApplePos(): Apple
+getDefaultDir(): Direction
Diatabase
-highScores: HighScores
+getHighScores(): HighScore^n
+storeHighScores(String, int, int): void
HighScores
+insertStatement: PreparedStatement
+deleteStatement: PreparedStatement
connection: Connection
+getHighScores(): HighScore^n
+isRecordExist(String, HighScore^n): boolean
+putHIghScore(String,int, int): void
-sortHighScores(HighScore^n): void
-getScore(String): int
_insertScore(String, int,int): void
Position
+x: int
+y: int
+translate(Direction): Position
+getRandomPos(int): Position
+equals(Object): boolean
LevelItem
+representation: char
Direction
+x: int (final)
+y: int (final)
+getOpposite(Direction): Direction
+randomDir(int): Direction
+equals(Object): boolean
HighScore
+numApples: int (final)
+level int (final)
+name: String (final)
+hashCode(): int
+equals(Object): boolean
MainWindow
-game: Game
-board: Board
-gameStatLabel: JLabel
-timeLabel: JLabel
-newFrameTimer: Timer
-timer: Timer
-startTime: long
-speed: int
-refreshGameStatLabel(): void
-levelSpeed(): void
-createGameLevelMenuItems(JMenu): void
-createScalelMenuItems(JMenu,double, double,double): void
-setTimer(): void
-restart(): void
-elapsedTIme(); long
Board
-game: Game
-rock, apple, head, body, empty: Image
-scale: double
-scaled_size: int
-tile_size: int
+setScale(double): boolean
+refresh(): boolean
+paintCOmponent(Graphics): void {override}
HighScoreTableModel
-highScores: HighScore^n
-colName: String^n
+getRowCount(): int
+getColumnCount(): int
+getValueAt(int,int): Object {override}
+getColumnName(int): String