Muhammad Eman Aftab IJE4R1 ije4r1@inf.elte.hu Group 8	**1st assignment/1. Task**	10th October 2024

Task

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.

Plan:

The task is to implement a single-player adventure game called **Yogi Bear**, set in Yellowstone National Park. Below is the detailed observation of the project requirements:

- ☐ **Game Setup:**
  - The game is played on a **2D grid board**, representing the park.
  - The park contains various elements, including:
    - **Mountains** and **trees**, which act as obstacles for Yogi.
    - **Rangers**, who patrol the park and can harm Yogi.
    - **Picnic baskets**, which Yogi must collect to progress.
  - Yogi starts at the **entrance** of the park with **three lives**.
  - The board is populated with these elements at the start of each level.

☐ **Game Rules:**

- **Movement**:
  - Yogi can move in all four directions: UP, DOWN, LEFT, and RIGHT.
  - Rangers can only move horizontally or vertically within the park.

- **Interaction with Rangers**:
  - If Yogi gets within one unit of distance from a ranger, he loses one life.
  - If Yogi has at least one life left, he respawns at the entrance.
  - Losing all lives ends the game.

- **Obstacle Rules**:
  - Mountains and trees block Yogi's movement.
  - Yogi must navigate around these obstacles to collect baskets

    .

☐ **Winning Condition:**

- Yogi wins the level by collecting all the picnic baskets in the park.

- Once all baskets are collected:
  - A new level is loaded (either pre-designed or procedurally generated).

☐ **Game Over:**

- If Yogi loses all three lives:
  - The game ends, displaying a Game Over message.
  - Players can input their name to save their score to the leaderboard.
  - The game offers the option to restart.

Analysis

The **Yogi Bear** game requires implementing a 2D grid-based adventure where Yogi navigates Yellowstone Park to collect picnic baskets while avoiding obstacles like trees, mountains, and patrolling rangers. Yogi starts with three lives, losing one when caught by a ranger, and respawns at the entrance unless all lives are lost, ending the game. The game tracks collected baskets, progressing to the next level upon completion. It features a high score leaderboard for the top 10 players, a dynamic menu for restarting and level selection, and a visually engaging interface. Efficient ranger movements, obstacle detection, and seamless transitions between levels are key to ensuring a fun and challenging gameplay experience.

Solution Plan:

Description about Classes and Objects :

1. Direction (Enum):

- This class is an enumeration representing the four possible directions of movement in the game: UP, DOWN, LEFT, and RIGHT. Each direction is associated with offsets to determine positional changes, ensuring consistency in movement logic.

2. Game :

- The Game class serves as the core controller for the game logic. It tracks Yogi's lives, score, and the current level. This class manages level progression, determines game states like victory or game over, and facilitates Yogi's interactions with the game board.

3.GameUtils:

- GameUtils is responsible for managing the state of a specific game level. It tracks the positions of Yogi, rangers, baskets, and obstacles. It also handles ranger movements, basket collection, and checks for valid moves. This class encapsulates the logic for interactions between game elements at a level-specific scope.

4.Yogi :

- This class represents the main character, Yogi Bear. It stores Yogi's current position on the board and provides functionality to update his position based on player input. Yogi interacts with other elements on the board, such as collecting baskets and encountering rangers.

4. Ranger

- The Ranger class represents a patrolling ranger on the game board. Rangers are dynamic elements that move randomly and serve as obstacles for Yogi. Their positions are updated regularly, and they interact with Yogi by reducing his lives upon close contact.

5. Position

- The Position class represents a coordinate on the game board with x and y values. It provides utilities for calculating new positions based on directions, ensuring consistency in handling movement and spatial relationships on the board.

6. LevelItem

- LevelItem is an enumeration defining all possible items that can appear on the game board, such as trees, mountains, baskets, rangers, and entrances. Each item is associated with a specific symbol, making it easy to represent the board elements programmatically.

7. Levels

- The Levels class provides predefined configurations for the game's levels. Each level is represented as a grid of strings, specifying the layout of obstacles, rangers, baskets, and Yogi's starting position. It also tracks the total number of levels in the game.

8. Board

- The Board class handles the graphical representation of the game. It displays the game board, Yogi, rangers, and obstacles using visual elements. This class ensures the board dynamically updates based on the game state and user interactions.

9. DatabaseManager

- The DatabaseManager class manages the leaderboard by interacting with a database. It saves player scores, retrieves the top 10 scores for display, and allows resetting the leaderboard. This class ensures persistent score tracking across game sessions.

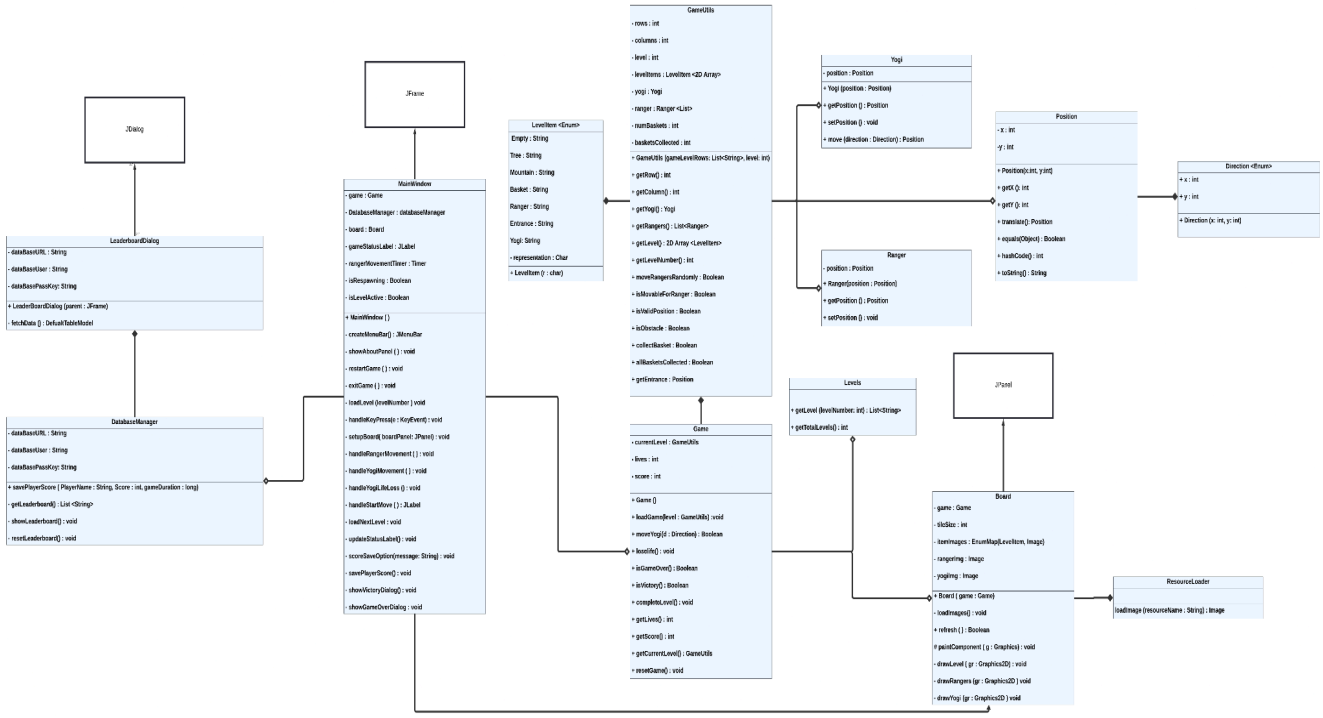
10. LeaderboardDialog

- The LeaderboardDialog class provides a user interface for displaying the leaderboard. It fetches data from the database and presents it in a table format, showing player rankings, scores, and game durations.

11. MainWindow

- The MainWindow class is the central graphical interface for the game. It integrates all game components, including the board, menu, and status updates. This class manages player input, transitions between levels, and interactions with the leaderboard. It also handles key gameplay events like ranger movement, level progression, and game over scenarios.

UML Diagram :


JDialog
JFrame
LeaderboardDialog
- dataBaseURL : String
- dataBaseUser : String
- dataBasePassKey: String
+ LeaderBoardDialog (parent : JFrame)
- fetchData () : DefaultTableModel
DatabaseManager
- dataBaseURL : String
- dataBaseUser : String
- dataBasePassKey: String
+ savePlayerScore (PlayerName : String, Score : int, gameDuration : long)
- getLeaderboard() : List <String>
- showLeaderboard() : void
- resetLeaderboard() : void
MainWindow
- game : Game
- DatabaseManager : DatabaseManager
- board : Board
- gameStatusLabel : JLabel
- rangerMovementTimer : Timer
- isRespawning : Boolean
- isLevelActive : Boolean
+ MainWindow ( )
- createMenuBar() : JMenuBar
- showAboutPanel ( ) : void
- restartGame ( ) : void
- exitGame ( ) : void
- loadLevel (levelNumber) void
- handleKeyPress(e : KeyEvent) : void
- setupBoard(boardPanel: JPanel) : void
- handleRangerMovement ( ) : void
- handleYogiMovement ( ) : void
- handleYogiLifeLoss () : void
- handleStartMove ( ) : JLabel
- loadNextLevel : void
- updateStatusLabel() : void
- scoreSaveOption(message: String) : void
- savePlayerScore() : void
- showVictoryDialog() : void
- showGameOverDialog : void
LevelItem <<Enum>>
Empty : String
Tree : String
Mountain : String
Basket : String
Ranger : String
Entrance : String
Yogi: String
- representation : Char
+ LevelItem (r : char)
GameUtils
- rows : int
- columns : int
- level : int
- levelItems : LevelItem <2D Array>
- yogi : Yogi
- ranger : Ranger <List>
- numBaskets : int
- basketsCollected : int
+ GameUtils (gameLevelRows: List<String>, level: int)
+ getRow() : int
+ getColumn() : int
+ getYogi() : Yogi
+ getRangers() : List<Ranger>
+ getLevel() : 2D Array <LevelItem>
+ getLevelNumber() : int
+ moveRangersRandomly : Boolean
+ isMovableForRanger : Boolean
+ isValidPosition : Boolean
+ isObstacle : Boolean
+ collectBasket : Boolean
+ allBasketsCollected : Boolean
+ getEntrance : Position
Game
- currentLevel : GameUtils
- lives : int
- score : int
+ Game ()
+ loadGame(level : GameUtils) :void
+ moveYogi(d : Direction) : Boolean
+ loseLife() : void
+ isGameOver() : Boolean
+ isVictory() : Boolean
+ completeLevel() : void
+ getLives() : int
+ getScore() : int
+ getCurrentLevel() : GameUtils
+ resetGame() : void
Yogi
- position : Position
+ Yogi (position : Position)
+ getPosition () : Position
+ setPosition () : void
+ move (direction : Direction) : Position
Ranger
- position : Position
+ Ranger(position : Position)
+ getPosition () : Position
+ setPosition () : void
Position
- x : int
- y : int
+ Position(x:int, y:int)
+ getX (): int
+ getY (): int
+ translate(): Position
+ equals(Object) : Boolean
+ hashCode() : int
+ toString() : String
Direction <<Enum>>
+ x : int
+ y : int
+ Direction (x: int, y: int)
Levels
+ getLevel (levelNumber: int) : List<String>
+ getTotalLevels() : int
JPanel
Board
- game : Game
- tileSize : int
- itemImages : EnumMap(LevelItem, Image)
- rangerImg : Image
- yogiImg : Image
+ Board (game : Game)
- loadImages() : void
+ refresh ( ) : Boolean
# paintComponent (g : Graphics) : void
- drawLevel (gr : Graphics2D) : void
- drawRangers (gr : Graphics2D) void
- drawYogi (gr : Graphics2D) void
ResourceLoader
loadImage (resourceName : String) : Image

Testing

White-Box:

- **. Invalid Move on Obstacle**
  - **AS A developer**
  - I WANT TO prevent Yogi from moving into obstacles
  - **GIVEN** a move into a tree or mountain is attempted
  - **WHEN** the move is executed
  - **THEN** it is rejected, and Yogi remains in his current position.

- **Valid Move to an Empty Cell**
  - **AS A developer**
  - I WANT TO allow valid moves to empty cells
  - **GIVEN** Yogi selects an adjacent cell
  - **WHEN** the move is executed
  - **THEN** Yogi moves successfully, and the game updates the board.

- **Ranger Collision Detection**
  - **AS A developer**
  - I WANT TO detect when Yogi collides with a ranger
  - **GIVEN** Yogi moves to a cell adjacent to a ranger
  - **WHEN** the collision occurs
  - **THEN** Yogi loses one life, and the game checks for game over or respawn conditions.

- **Basket Collection**
  - **AS A developer**
  - I WANT TO track the collection of picnic baskets
  - **GIVEN** Yogi moves onto a basket
  - **WHEN** the move is executed
  - **THEN** the basket is removed from the board, and the score increases by one.

- **Ranger Random Movement**
    - **AS A developer**
    - I WANT TO validate the random movement of rangers
    - **GIVEN** the rangers' turn
    - **WHEN** rangers attempt to move
    - **THEN** they only move to valid, empty adjacent cells and avoid obstacles.

- **Out-of-Bounds Move Check**
    - **AS A developer**
    - I WANT TO restrict moves outside the board boundaries
    - **GIVEN** a move is attempted beyond the board edges
    - **WHEN** the move is executed
    - **THEN** the move is rejected, and the game state remains unchanged.

- **. Level Completion Detection**
    - **AS A developer**
    - I WANT TO detect when all baskets are collected
    - **GIVEN** Yogi collects the last basket on the board
    - **WHEN** the level is completed
    - **THEN** the game loads the next level or prompts a victory message if it is the final level.

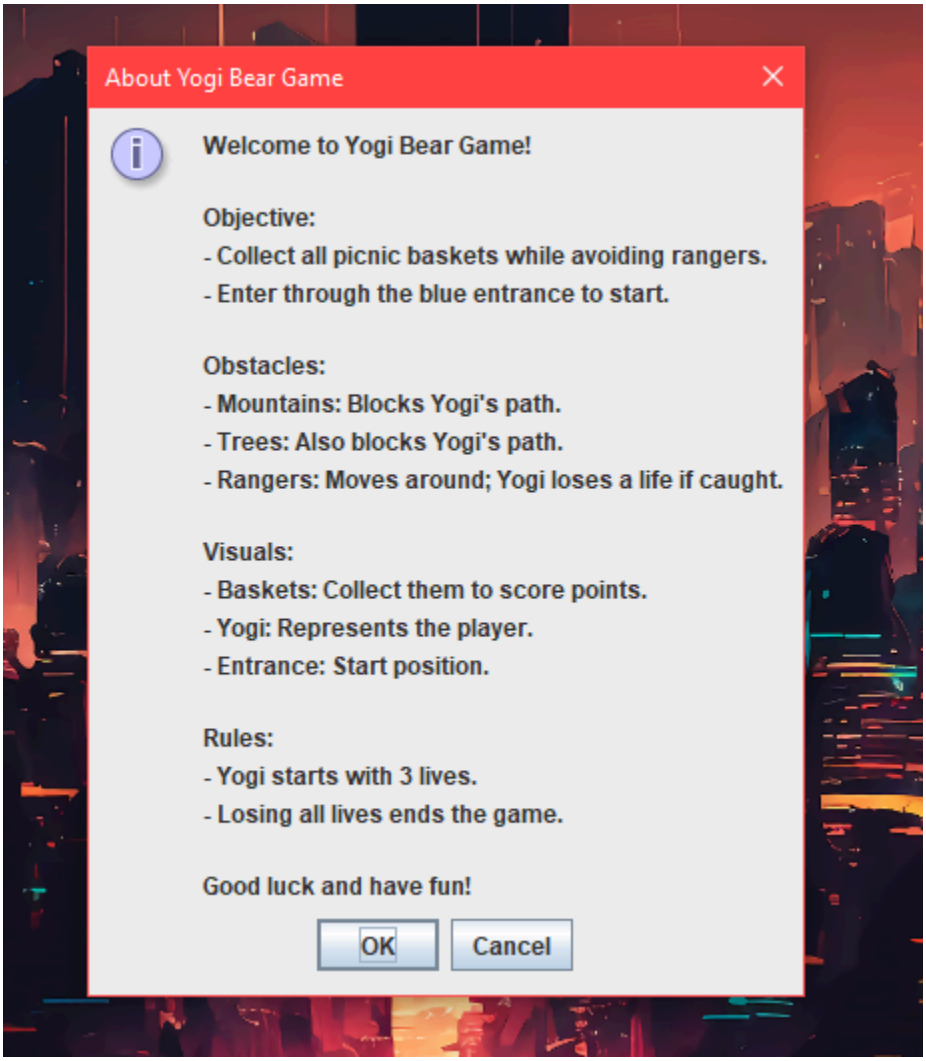
- **Game Over State**
    - **AS A developer**
    - I WANT TO handle game-over conditions
    - **GIVEN** Yogi loses all lives
    - **WHEN** the game checks for remaining lives
    - **THEN** it displays a game-over message and prompts the player to save their score..

- **Respawning Logic**
    - **AS A developer**
    - I WANT TO respawn Yogi after losing a life
    - **GIVEN** Yogi has at least one life remaining
    - **WHEN** a life is lost
    - **THEN** Yogi respawns at the entrance, and the game continues.

- **Leaderboard Save and Display**
    - **AS A developer**
    - I WANT TO save and display player scores
    - **GIVEN** the game ends with a valid score
    - **WHEN** the player opts to save the score
    - **THEN** the leaderboard is updated, and the top 10 scores are displayed.

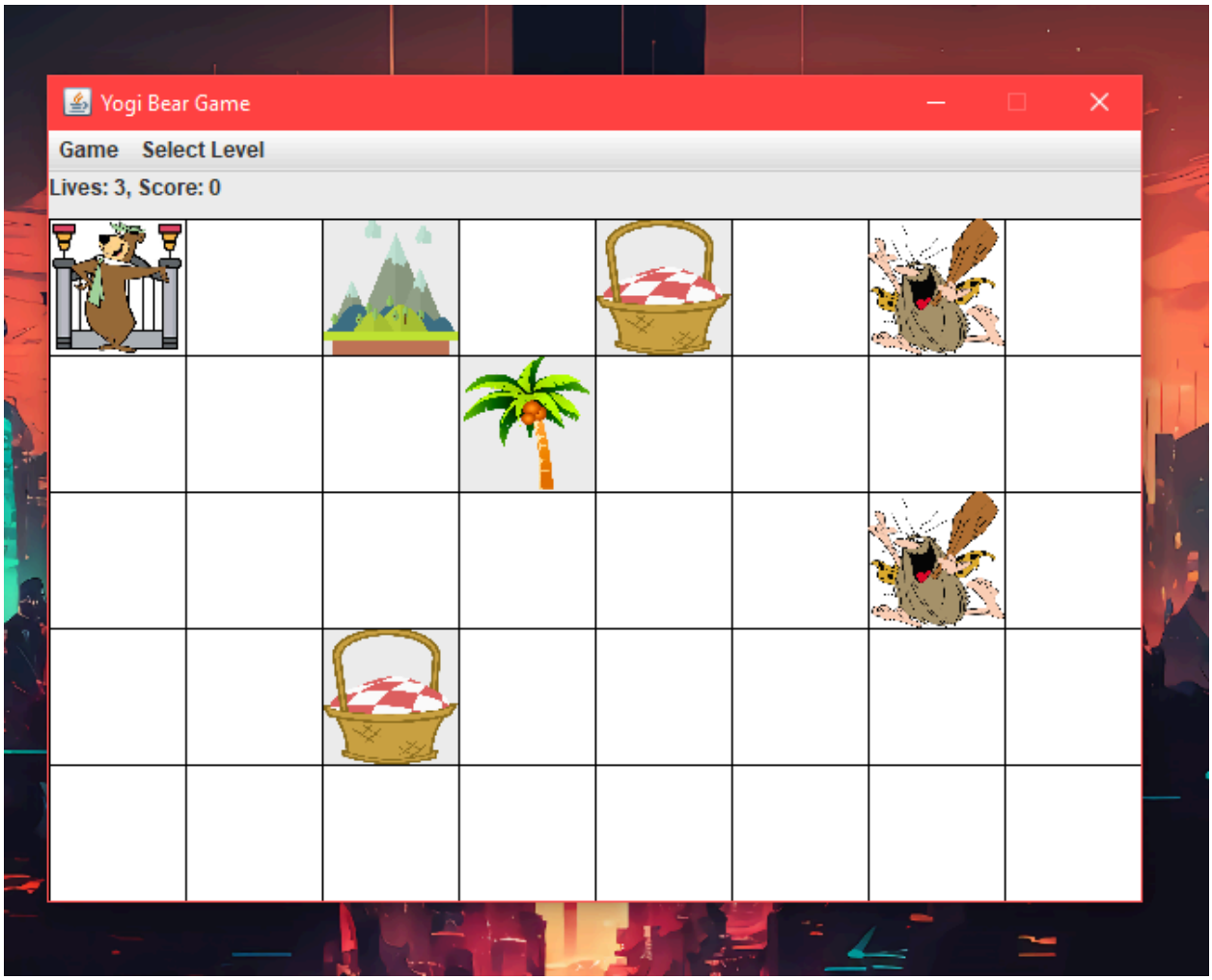
Black-Box:

- **Game Rules Display**
  - **AS A player**
  - I WANT TO understand the game rules
  - **GIVEN** the game starts
  - **WHEN** the rules dialog is displayed
  - **THEN** I can read the rules and proceed to the game.

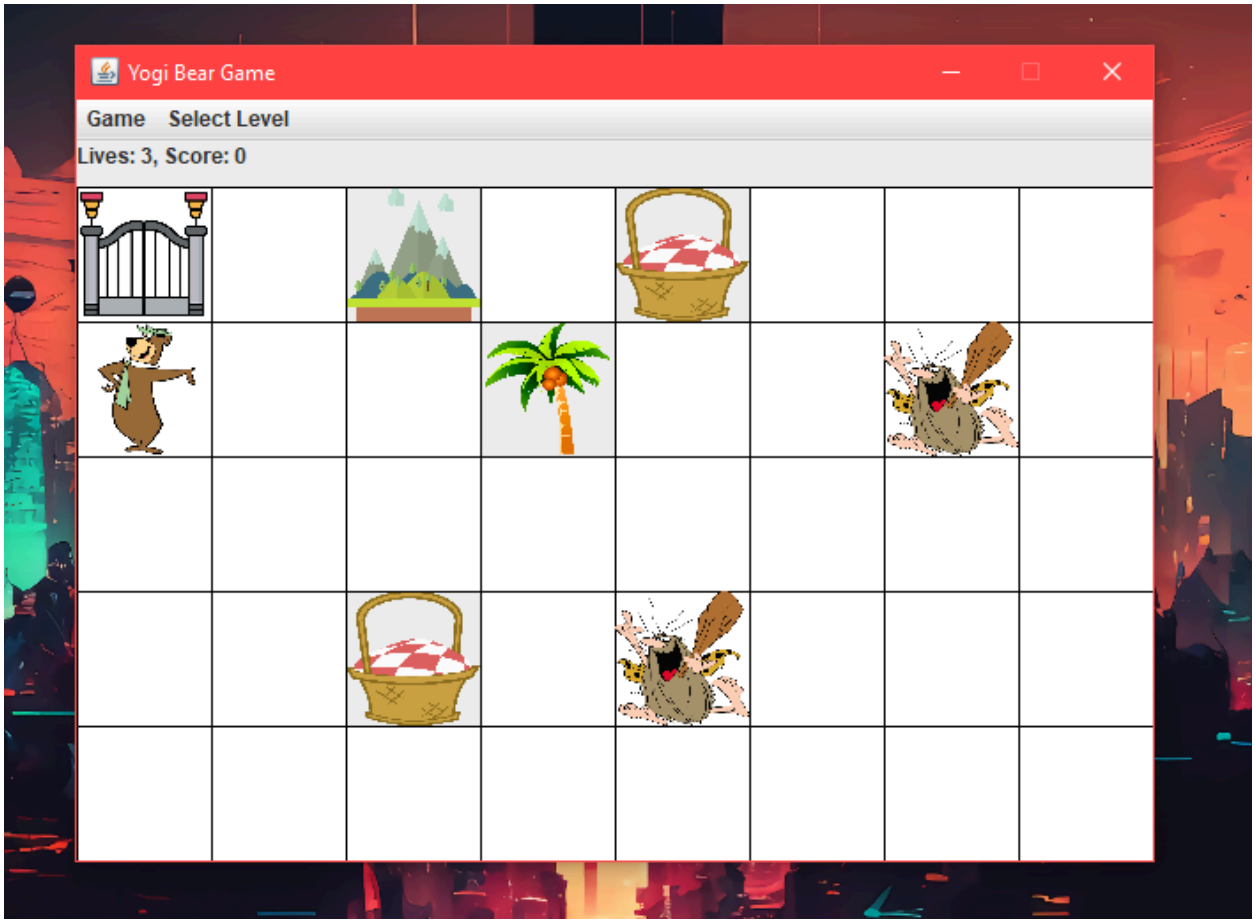


  -

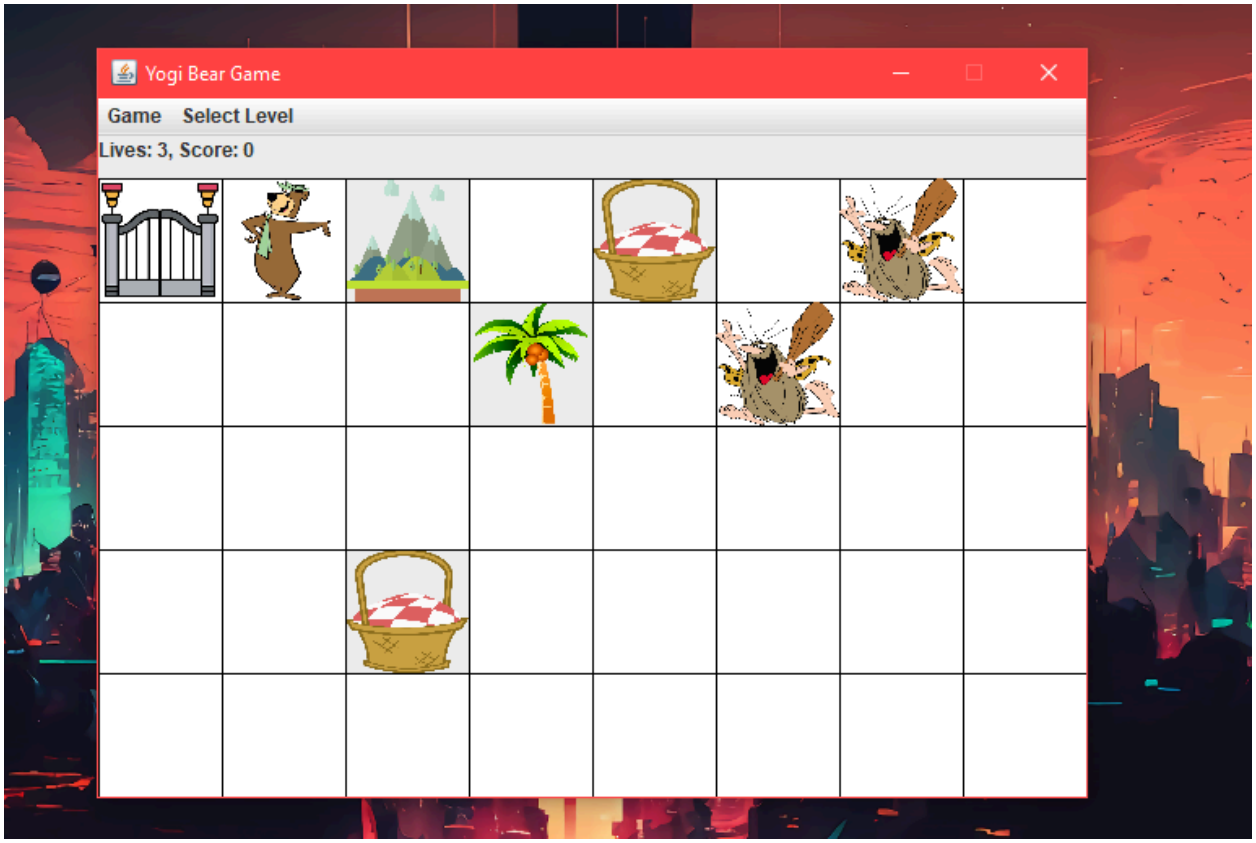
- **Board Initialization**
  - **AS A player**
  - I WANT TO see the initial setup of the board
  - **GIVEN** the game starts or a level is loaded
  - **WHEN** the board is displayed
  - **THEN** all elements (Yogi, rangers, baskets, and obstacles) are placed correctly.



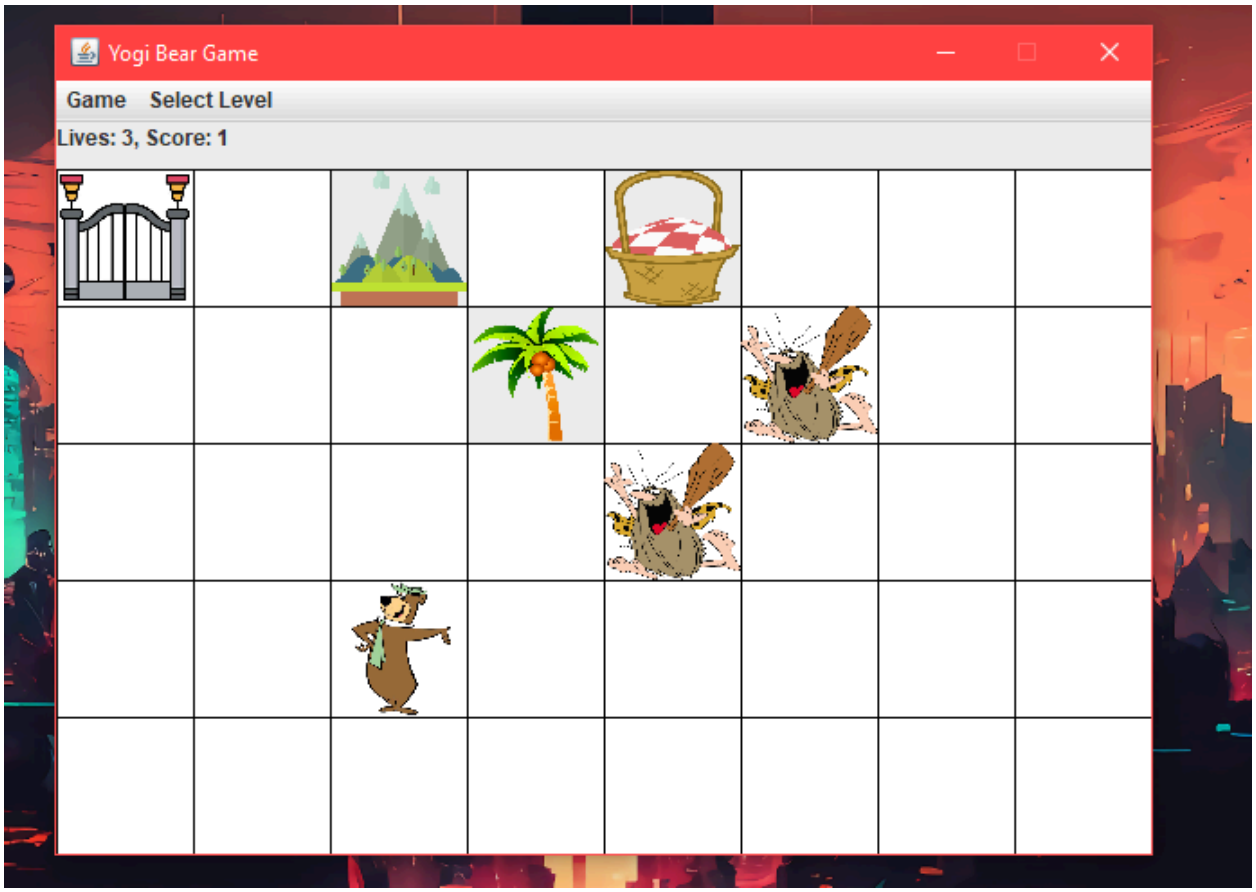
- **Valid Movement**
  - **AS A player**
  - I WANT TO move Yogi to valid adjacent cells
  - **GIVEN** Moving yogi to adjacent cell
  - **WHEN** I press an arrow key for a valid move
  - **THEN** Yogi moves to the target cell, and the board updates.



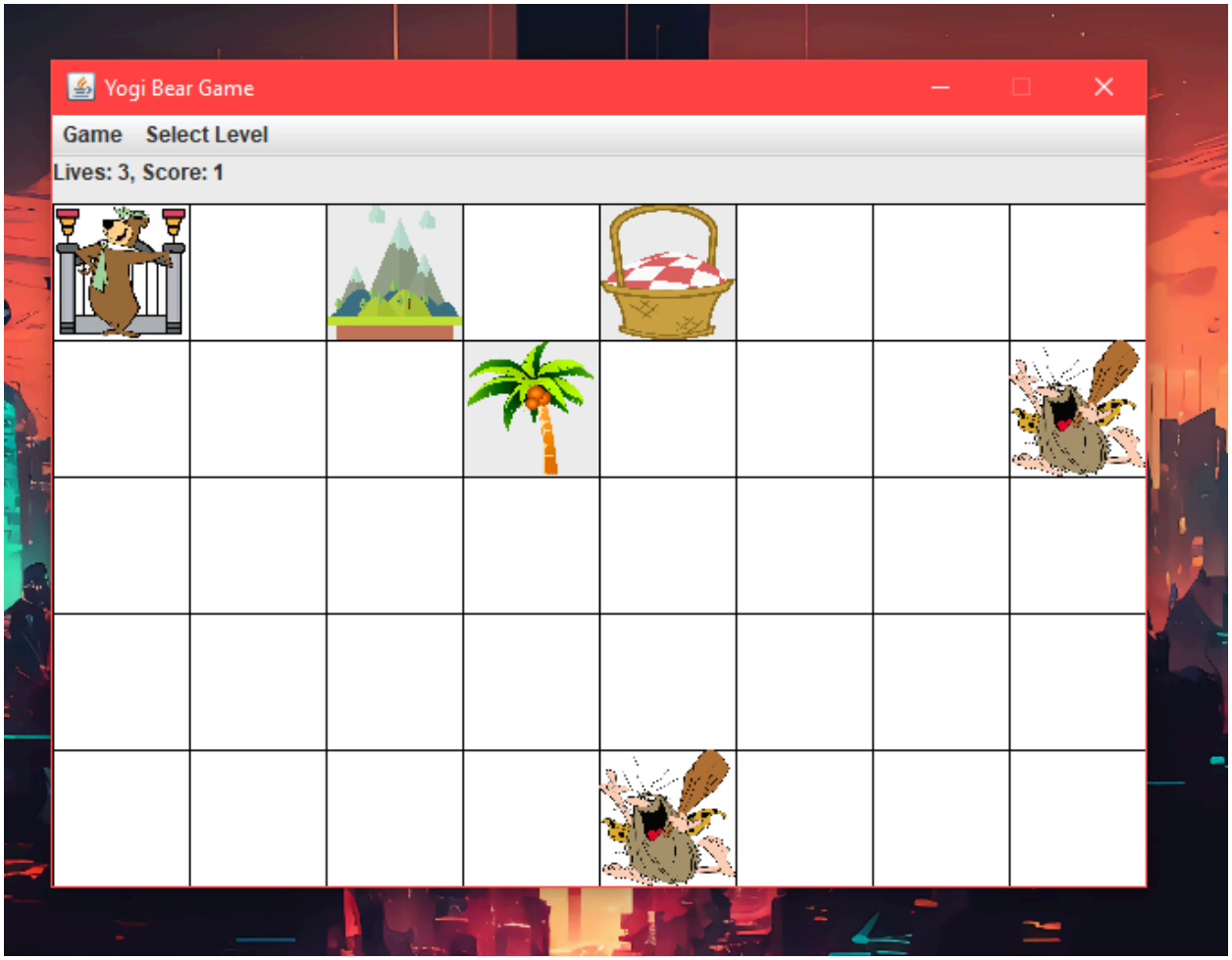
- **Invalid Movement**
  - **AS A player**
  - I WANT TO be prevented from making invalid moves
  - **GIVEN** an obstacle, or boundary is in the way
  - **WHEN** I attempt to move Yogi
  - **THEN** the game doesn't update the board, and Yogi remains in place.



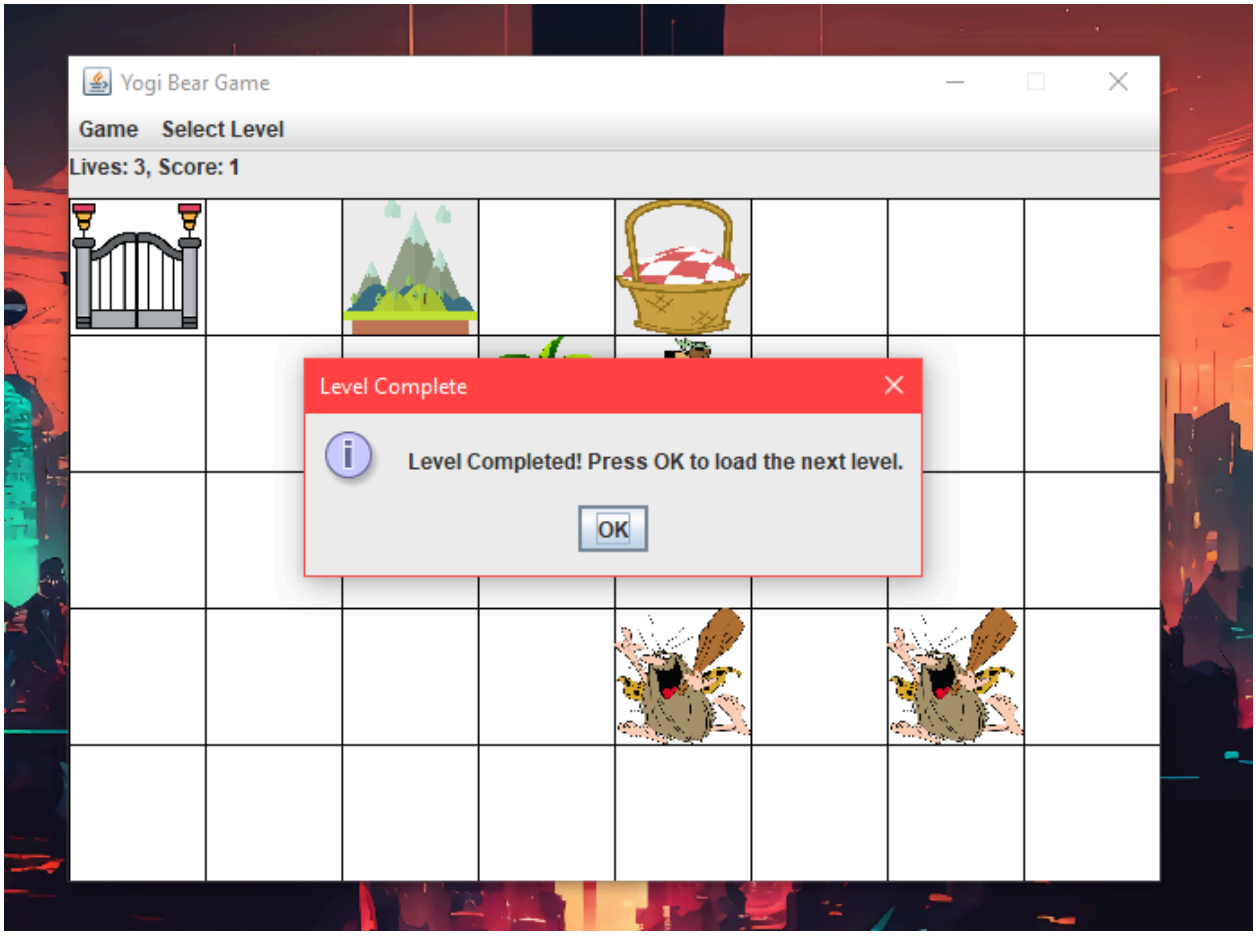
- **. Basket Collection Feedback**
  - **AS A player**
  - I WANT TO see feedback for collecting baskets
  - **GIVEN** I move Yogi onto a basket
  - **WHEN** the basket is collected
  - **THEN** the game updates the score and removes the basket from the board.

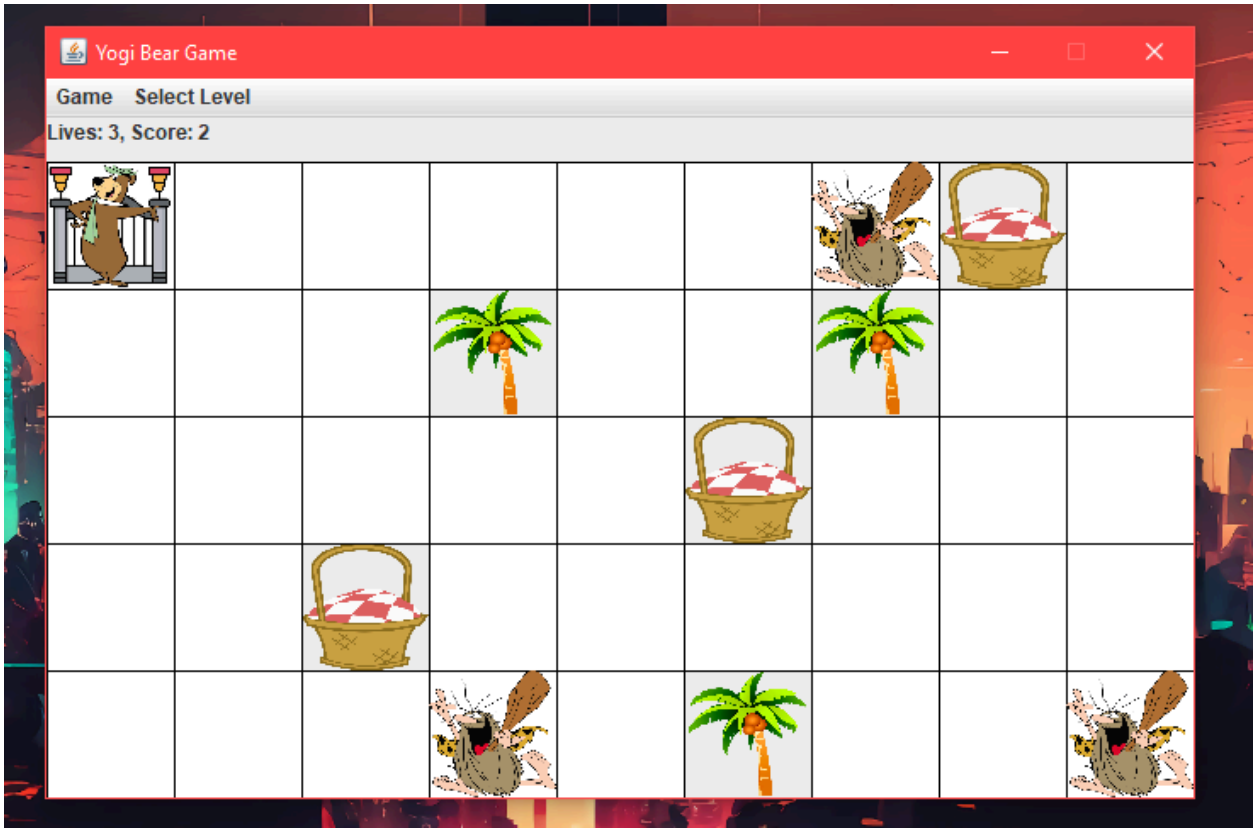


- **Random Ranger Movement**
  - **AS A player**
  - I WANT TO see rangers move randomly
  - **GIVEN** I do nothing
  - **WHEN** the game transitions to the ranger's move randomly
  - **THEN** the rangers move randomly, updating the board.

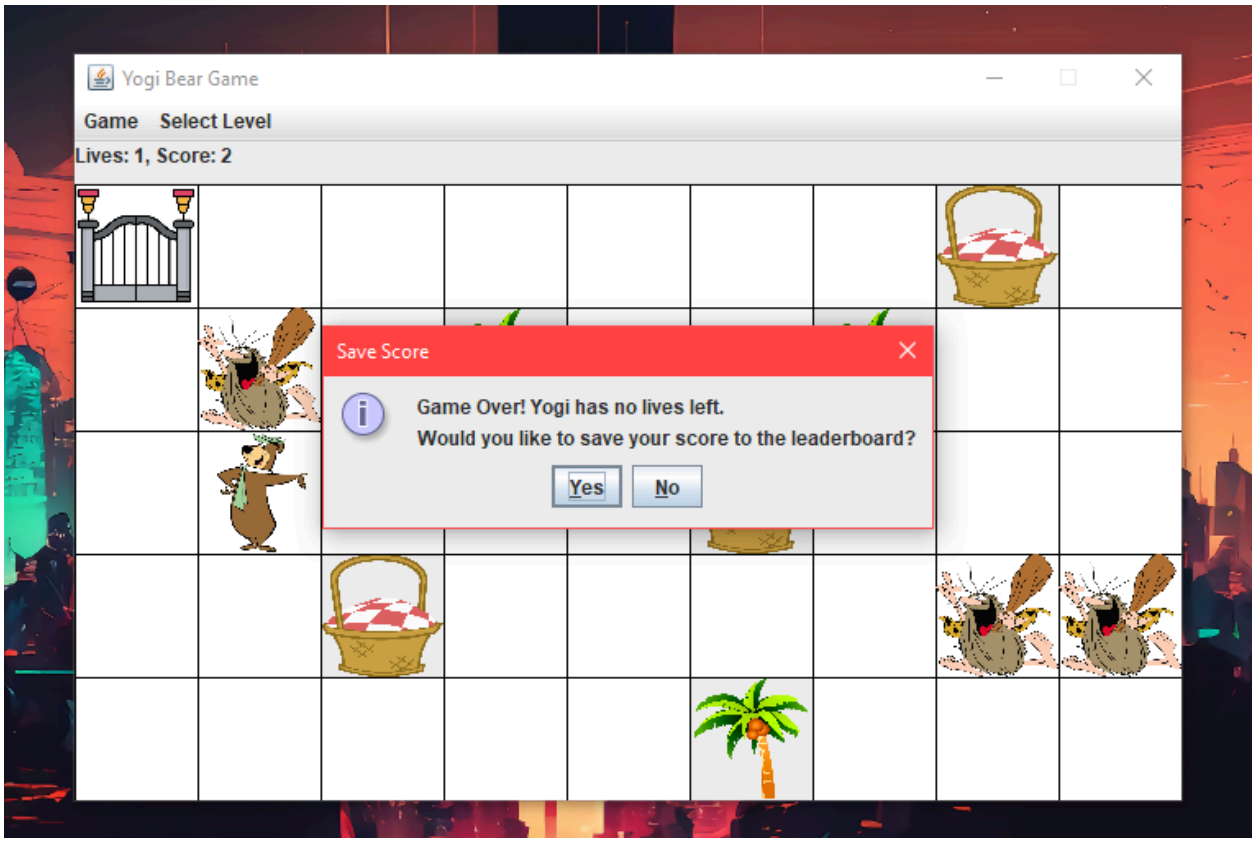


- **Level Progression**
  - **AS A player**
  - I WANT TO advance to the next level after collecting all baskets
  - **GIVEN** I collect the last basket on the board
  - **WHEN** the game detects level completion
  - **THEN** the next level is loaded, or a victory message is displayed.

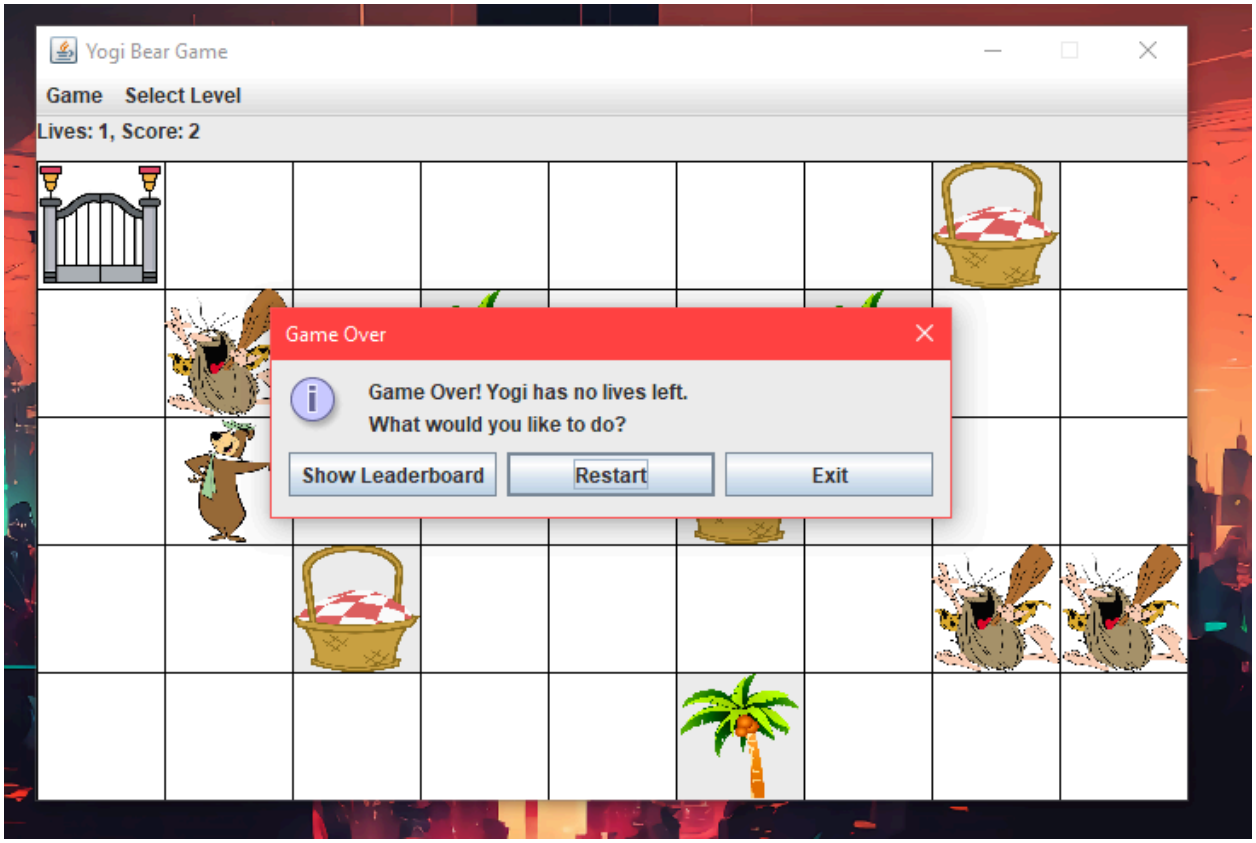


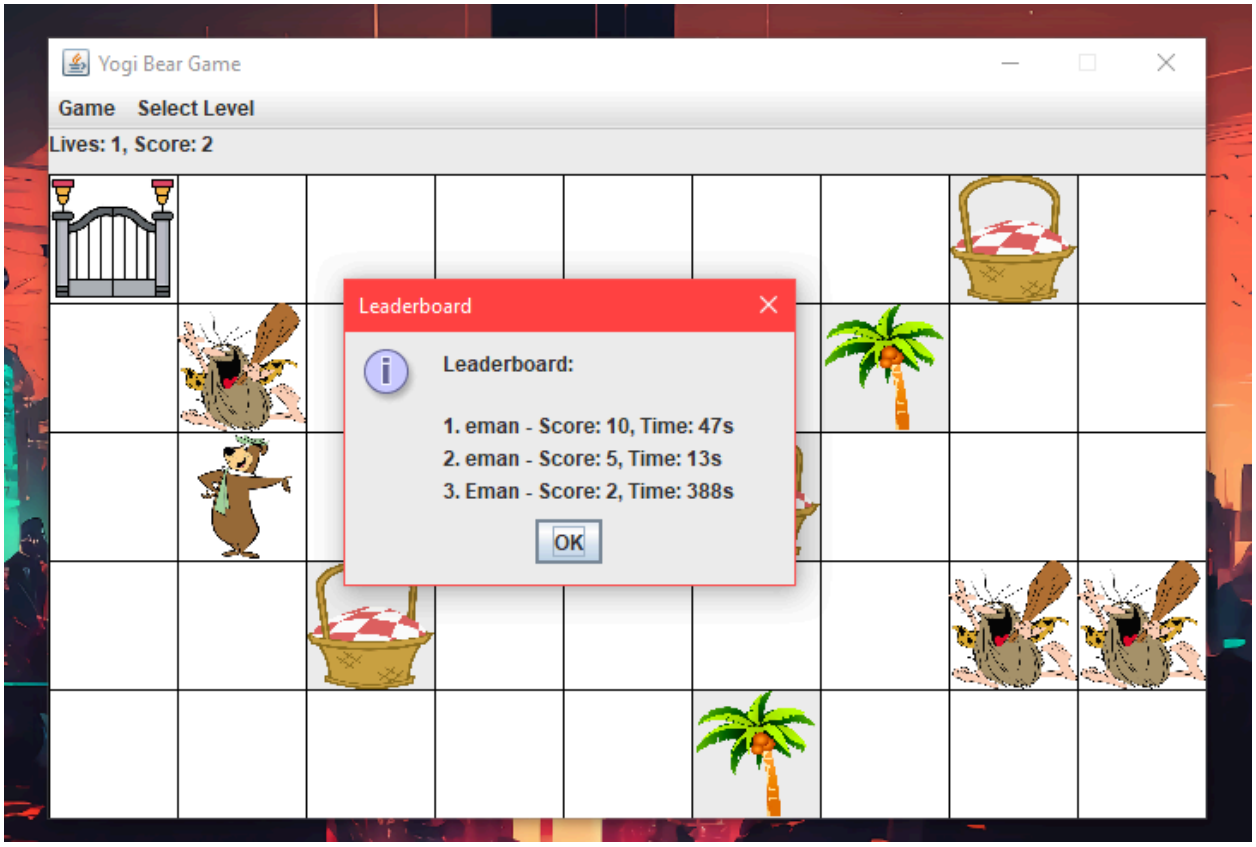

Yogi Bear Game
Game
Select Level
Lives: 3, Score: 2

- **Game Over Feedback**
  - **AS A player**
  - I WANT TO be notified when the game is over
  - **GIVEN** I lose all lives
  - **WHEN** the game detects no remaining lives
  - **THEN** it displays a game-over message and prompts me to save my score.



- **Leaderboard Interaction**
    - **AS A player**
    - I WANT TO view and update the leaderboard
    - **GIVEN** I finish a game or select the leaderboard menu
    - **WHEN** I interact with the leaderboard
    - **THEN** I see the top 10 scores and can update or Override the data if they have values more than 10.





Button Handlers:

1. Level Selection Menu
Event: ActionEvent triggered when a level menu item is selected.
Handler: Loads the selected level and initializes the board.

2. Restart Button
Event: ActionEvent triggered when the restart button is clicked.
Handler: Resets the game state and reloads the first level.

3. Leaderboard Button
Event: ActionEvent triggered when the leaderboard button is clicked.
Handler: Displays the leaderboard dialog.

4. Exit Button
Event: ActionEvent triggered when the exit button is clicked.
Handler: Closes the game application.