ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit

Documentation for Yogi Game

Exercise Description:

Yogi Bear wants to collect all the picnic baskets in the forest of Yellowstone National Park. The park contains obstacles, such as mountains and trees, which impede Yogi's movement. Additionally, rangers patrol the park, making it harder for Yogi to collect the baskets. Rangers move horizontally or vertically and can harm Yogi if they get too close (one unit distance). If Yogi loses all three lives, the game ends, and the player's name and score can be saved in a high-score database. If Yogi collects all the baskets, the game progresses to a new level.

Analysis:

The game requires a system to simulate Yogi's movement, ranger patrols, collision detection, and game level progression. The game must recognize win or loss conditions and provide a high-score mechanism. The challenges lie in ensuring smooth interaction between game components, proper validation of player moves, and accurate tracking of game state.

Key Classes of the Game

1. Player

Description: Represents Yogi Bear, the protagonist, who moves in the park to collect baskets while avoiding rangers.

Responsibilities:

- Moving across the game board while avoiding obstacles and rangers.
- Collecting baskets to progress to the next level.
- Respawning at the entrance after losing a life.

2. Ranger

Description: Represents the game's enemies, which move horizontally or vertically to impede Yogi's progress.

Responsibilities:

- **Moving**: Patrolling the park based on predefined movement patterns.
- Reducing Yogi's lives if they get within a one-unit distance.

3. GameController

Description: Manages the interaction between the game model and the view, ensuring game rules are enforced.

Responsibilities:

- Processing player inputs to move Yogi and trigger game actions.
- Managing win and loss conditions and initiating new levels.
- Handling the high-score system and menu interactions.

4. GameFrame

Description:

Represents the main game window that houses the GamePanel and provides additional menu options.

Responsibilities:

- Initializing and displaying the main game window.
- Hosting the GamePanel where the game is rendered.
- Providing menu options for restarting the game and displaying the high-score table.
- Managing the overall frame layout and integrating the game controller with the user interface.

5. GamePanel

Description:

Handles the graphical rendering of the game board and its components, as well as capturing player input for movement and interaction.

Responsibilities:

- **Displaying the Game Board:** Renders the grid, Yogi, rangers, obstacles, and baskets.
- **Capturing User Input:** Processes keyboard inputs to control Yogi's movements and actions.
- **Updating the Display:** Refreshes the game board and visual elements based on changes in the game state.

- **Displaying Game Status:** Shows the current game status (e.g., lives remaining, baskets collected) in a status bar at the bottom.
- **Error Feedback:** Highlights invalid moves or conditions through visual or text feedback.

Solution plan:

Design Patterns

The game employs the **Model-View-Controller (MVC)** pattern:

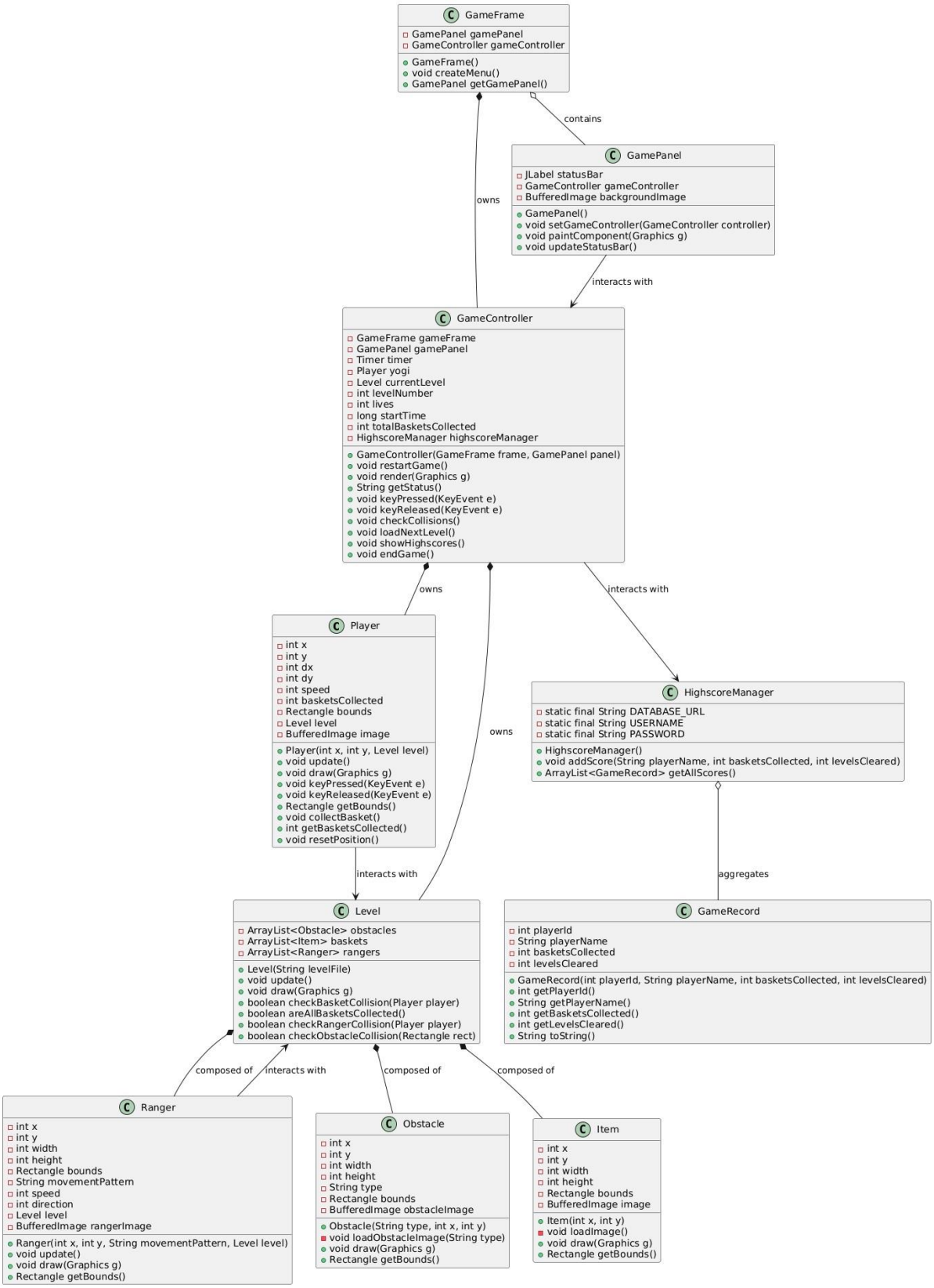
- **Model:** Manages game logic and state (e.g., player, ranger, obstacles, baskets).
- **View:** Provides graphical representation and captures user inputs.
- **Controller:** Processes inputs, updates the model, and refreshes the view.

Class Diagram:

ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit



ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit

Description of Important Methods:

GameFrame

1. **createMenu()** ○ **Purpose:** Initializes and sets up the game menu.
    - **Event Handlers:**
        - **Restart Menu Item:** Calls gameController.restartGame() to restart the game.
        - **Highscores Menu Item:** Calls gameController.showHighscores() to display the highscore table.
    - **Functionality:** Adds the menu bar and its items to the GameFrame.
2. getGamePanel() ○ **Purpose:** Provides access to the GamePanel.
    - **Functionality:** Returns the gamePanel instance for interaction with other components.

GamePanel

1. **setGameController(GameController controller)** ○ **Purpose:** Links the GameController to the GamePanel.
    - **Functionality:** Adds the GameController as a KeyListener for handling player inputs.
2. **paintComponent(Graphics g)** ○ **Purpose:** Renders the game elements on the panel.
    - **Functionality:** Draws the background, invokes gameController.render() to draw game objects, and updates the status bar.
3. **updateStatusBar()** ○ **Purpose:** Updates the status bar with the current game state.
    - **Functionality:** Calls gameController.getStatus() to fetch the game status (lives, baskets, time).

GameController

1. **restartGame()** ○ **Purpose:** Restarts the game.
    - **Functionality:** Resets the level, lives, and basket count; reloads the first level.
2. **showHighscores()** ○ **Purpose:** Displays the high-score table.
    - **Functionality:** Fetches and shows the top scores in a dialog box.
3. **render(Graphics g)** ○ **Purpose:** Renders the game objects.
    - **Functionality:** Draws the current level and Yogi.
4. **keyPressed(KeyEvent e)** ○ **Purpose:** Handles player movement input.
    - **Event Handler:** Updates Yogi's movement direction based on arrow or WASD keys.

5. **keyReleased(KeyEvent e)** ○ **Purpose:** Stops player movement when keys are released.
    - **Event Handler:** Resets Yogi's movement direction.

6. **checkCollisions()** ○ **Purpose:** Detects and handles collisions between Yogi, rangers, and baskets.
    - **Functionality:** Updates lives, basket count, or triggers a new level/game over based on collisions.

7. **loadNextLevel()** ○ **Purpose:** Loads the next level when all baskets are collected.
    - **Functionality:** Advances the level and resets Yogi's position.

8. **endGame()**
    - **Purpose:** Ends the game when Yogi loses all lives.
    - **Functionality:** Displays a name input prompt and saves the score.

Level

1. **loadLevelFromFile(String levelFile)** ○ **Purpose:** Initializes the level by reading its configuration from a file.
    - **Functionality:** Populates obstacles, baskets, and rangers based on the file's data.

2. **checkBasketCollision(Player player)** ○ **Purpose:** Checks if Yogi has collected a basket.
    - **Functionality:** Removes the basket and returns true if a collision occurs.

3. **checkRangerCollision(Player player)** ○ **Purpose:** Checks if Yogi is too close to a ranger.
    - **Functionality:** Returns true if a ranger is within one unit of Yogi.

4. **checkObstacleCollision(Rectangle rect)** ○ **Purpose:** Validates if a rectangle (Yogi or ranger) collides with any obstacle.
    - **Functionality:** Returns true if a collision is detected.

HighscoreManager

1. **addScore(String playerName, int basketsCollected, int levelsCleared)** ○ **Purpose:** Saves a new high score to the database.
    - **Functionality:** Inserts the player's name, baskets collected, and levels cleared into the database.

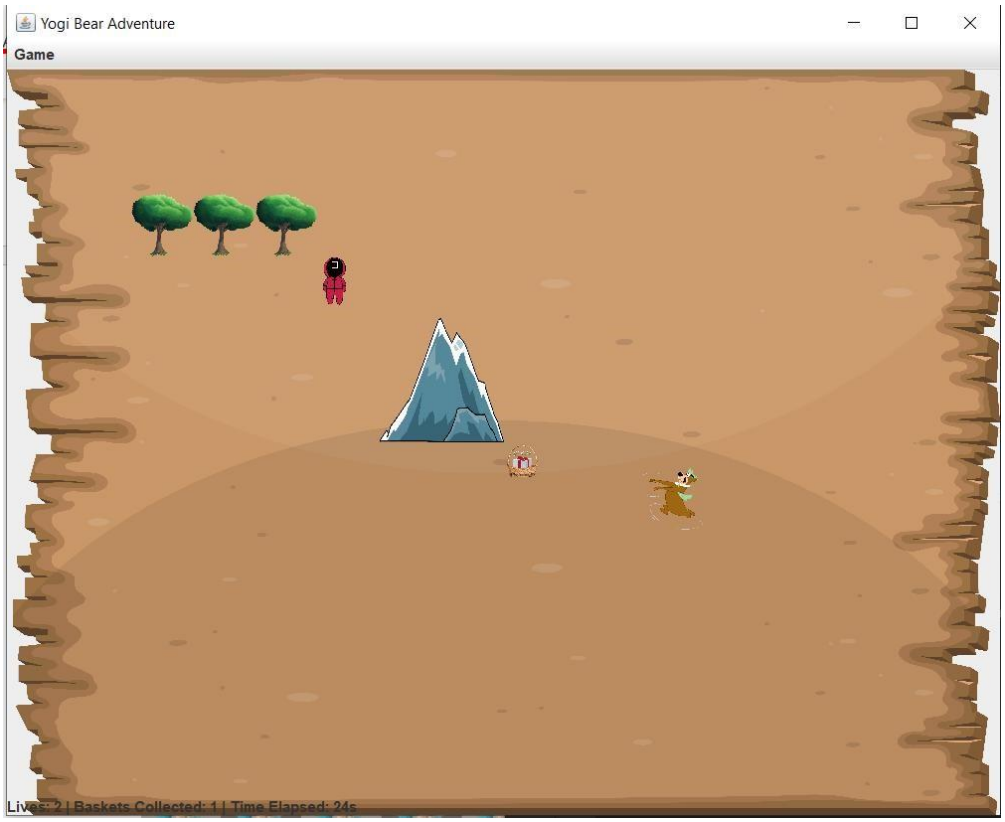
ID: EE8LMT

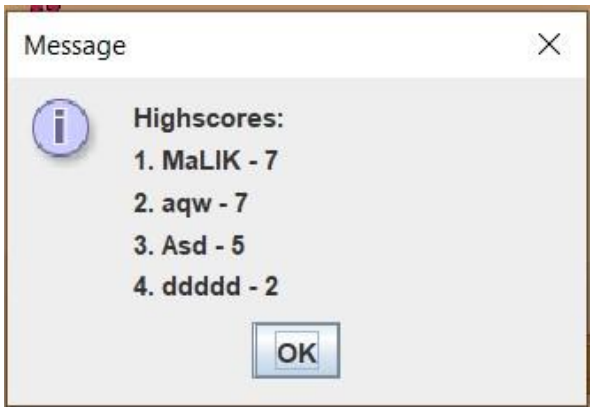
Assignment 2 P07

Name: Abdul Basit

2. **getAllScores()** ○ **Purpose:** Retrieves the top scores from the database.
    - **Functionality:** Fetches and returns a sorted list of high scores.

UI :

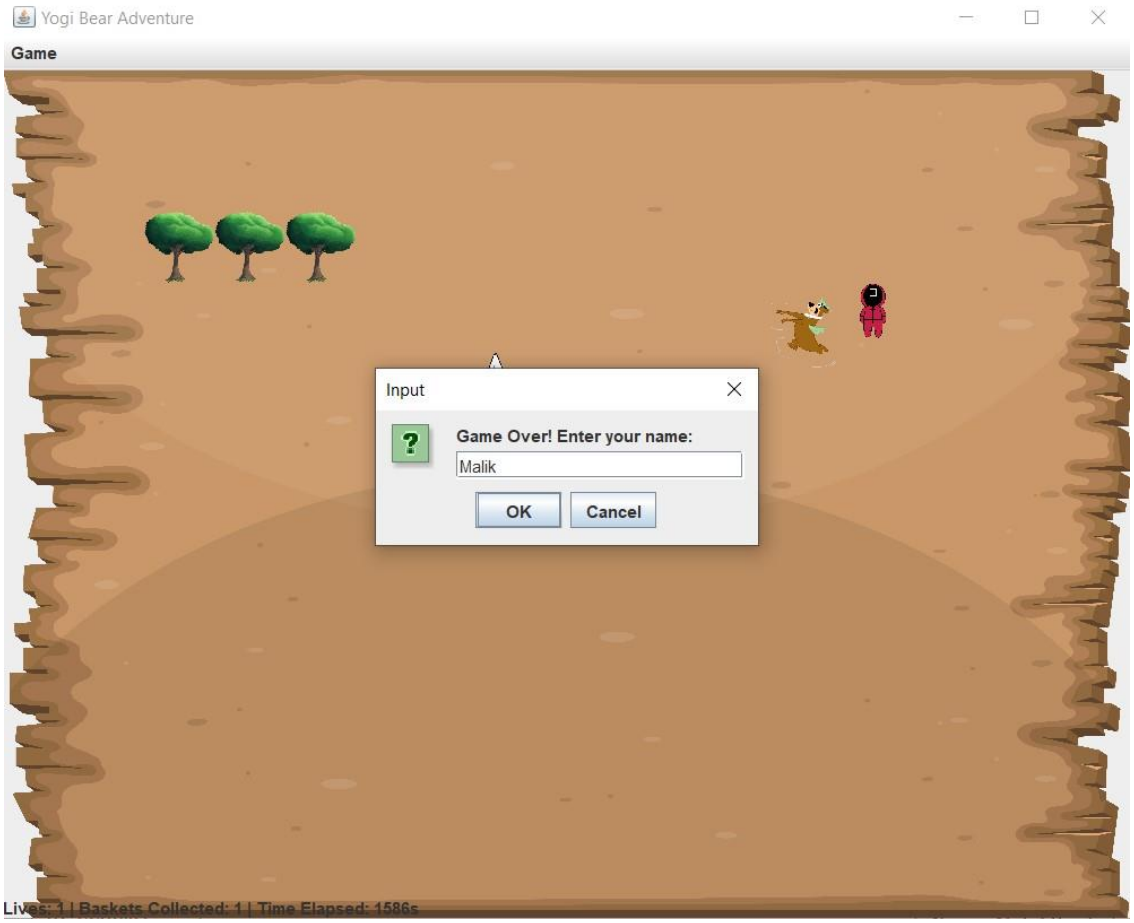




ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit



Test Cases Black-Box Test Cases

1. **TestBasketCollection**

   As a: Player
   I want to: Collect all picnic baskets to progress to the next level.
   GIVEN: Yogi is near a basket.
   WHEN: Yogi moves to the basket's position.
   THEN: The basket should be removed from the board, and the score should increase.

2. **TestRangerCollision**

   As a: Player
   I want to: Avoid rangers to preserve lives.
   GIVEN: A ranger is one unit away from Yogi.
   WHEN: Yogi moves closer to the ranger.
   THEN: Yogi should lose a life and respawn at the entrance.

3. **TestGameOver**

   As a: Player
   I want to: See the game end when all lives are lost. **GIVEN:** Yogi has one life left.

WHEN: A ranger catches Yogi.
THEN: The game should end, and a message should prompt for the player's name.

4. **TestHighscoreEntry**
   As a: Player
   I want to: Save my score when the game ends.
   GIVEN: The game is over.
   WHEN: I enter my name in the prompt.
   THEN: The score should be saved in the database.

5. **TestInvalidMove**
   As a: Player
   I want to: Ensure invalid moves are blocked.
   GIVEN: Yogi tries to move outside the board.
   WHEN: The move is attempted.
   THEN: The game should show an error message.

White-Box Test Cases

1. **TestUpdateRangers**
   As a: Developer
   I want to: Verify that rangers move according to their patterns.
   GIVEN: A ranger has a horizontal movement pattern. **WHEN:** The board updates.
   THEN: The ranger's position should change correctly.

2. **TestCollisionDetection**
   As a: Developer
   I want to: Ensure collision detection works correctly.
   GIVEN: Yogi and a ranger occupy adjacent cells.
   WHEN: The game checks for collisions.
   THEN: The collision should be detected, and Yogi should lose a life.

3. **TestResetGame**
   As a: Developer
   I want to: Verify the game resets correctly.
   GIVEN: The game has ended.
   WHEN: The reset function is called.
   THEN: The board and player state should return to their initial values.

4. **TestWinCondition**
   As a: Developer
   I want to: Ensure the game detects a win correctly.
   GIVEN: All baskets are collected.
   WHEN: The game checks for a win. **THEN:** The next level should load.

ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit

5. **TestHighscoreDatabase**

   As a: Developer

   I want to: Verify that high scores are saved and retrieved correctly.

Assignment 2 P07

ID: EE8LMT

Name: Abdul Basit

GIVEN: A new score is saved.
WHEN: The high-score table is displayed.
THEN: The saved score should appear in the correct position.