Report - Block Basher

Nebulla Nexus

December 1, 2023

Name of Student 1 : Srinivasan M (IMT2021058)
Name of Student 2: P Ram Sai Koushik (IMT2021072)
Name of Student 3: Aditya Nagaraja (IMT2021054)
Name of Student 4: Neelapati Rishi (IMT2021076)

Contents

1 Description

The game represents a classic gaming experience characterized by a player-controlled paddle, a ball, and an arrangement of breakable objects. Players maneuver the paddle along the bottom of the screen to bounce a ball upwards, aiming to destroy a structure of various objects arranged in patterns at the top.

1.1 Key Features:

- **Object Destruction:** The primary objective involves using a ball to break objects, often arranged as bricks, by bouncing it off a paddle.
- **Paddle Control:** Players control a paddle's horizontal movement to prevent the ball from falling off the screen and to redirect it towards the objects.
- **Multiple Levels:** The game typically comprises multiple levels, each presenting different arrangements of objects, varying difficulty, and sometimes introducing new challenges or power-ups.
- **Block Variation:** Some blocks have distinct attributes, such as differing durability or unique effects when destroyed.
- **Progression and Challenges:**As players advance through levels, the game tends to escalate in difficulty, featuring faster ball speeds, tighter patterns, or more resilient objects.

2 Classes

2.1 MenuFrame

- **Attributes:**
  - `private CardLayout cardLayout`: Manages the card layout for panel switching.
  - `private JPanel cardPanel`: Contains panels to display different menu screens.
  - `private DifficultyManager difficultyManager`: Handles game difficulty settings.
  - `private CustomizationManager customizationManager`: Manages game customization options.
- **Methods:**
  - `public MenuFrame()`: Constructor initializing the game menu frame, setting its properties, and creating the menu and settings panels.
    - `private JPanel createMenuPanel()`: Generates the menu panel with "Start Game" and "Settings" buttons, assigning actions for each.
      - `JButton startButton`: Initiates the game frame and prints settings when clicked.
      - `JButton settingsButton`: Switches to the settings panel when clicked.
  - `public void showMenu()`: Displays the menu panel.
  - `private void showSettings()`: Displays the settings panel.
  - `public static void main(String[] args)`: Entry point for the program, initializes and displays the menu frame.

2.2 SettingsPanel

- **Attributes:**
  - `private CustomizationManager customizationManager`: Manages customization options for the game.
  - `private DifficultyManager difficultyManager`: Manages game difficulty settings.

- `private Map<JLabel, JComboBox<String>> labelComboBoxMap`: Maps JLabels to their respective JComboBoxes for settings management.

- **Methods:**
  - `public SettingsPanel(CustomizationManager, DifficultyManager, MenuFrame)`: Constructor initializing the settings panel layout, adding various settings options, and a Save button to save the settings.
  - `private void addLabelAndComboBox(String, String[], GridBagConstraints)`: Adds a label and a combo box for a specific setting using GridBagConstraints for layout.
  - `private void saveSettings()`: Gathers selected settings and updates the CustomizationManager and DifficultyManager accordingly.
  - `private JLabel getLabel(String)`: Retrieves the JLabel associated with a given setting text.
  - `private Color parseColor(String)`: Parses a string input into a Color object, either using Color constants or decoding a hex color value.

2.3 GameFrame

- **Attributes:**
  - `private GameEngine gameEngine`: Represents the game engine JPanel.

- **Constructor:**
  - `public GameFrame()`: Initializes the game frame with title, size, and location settings. Creates a `GameEngine` instance using provided `DifficultyManager` and `CustomizationManager`, and adds it to the frame.

2.4 GameEngine

- **Attributes:**
  - `private PaddleController paddleController`: Manages the paddle controls.
  - `private BallController ballController`: Controls the ball movement and behavior.
  - `private BrickManager brickManager`: Manages the bricks in the game.
  - `private ScoringSystem scoringSystem`: Handles the game's scoring system.
  - `private CustomizationManager customizationManager`: Manages game customization options.
  - `private DifficultyManager difficultyManager`: Handles game difficulty settings.
  - `private UserInterface ui`: Manages the user interface elements.
  - `private InputHandler inputHandler`: Handles user input.
  - `private CollisionConnector collisionConnector`: Manages collision detection.
  - `private InputConnector inputConnector`: Connects input handling components.
  - `private ScoreConnector scoreConnector`: Manages score-related functionality.
  - `private HashMap<Integer, Integer> difficulty_speed`: Stores speed values based on difficulty levels.
  - `private HashMap<Integer, Integer> difficulty_paddle_length`: Stores paddle length values based on difficulty levels.
  - `private HashMap<Integer, Integer> difficulty_paddle_speed`: Stores paddle speed values based on difficulty levels.
  - `private int delay = 8`: Represents the delay interval for the game timer.
  - `private Timer timer`: Manages the game timer.

- `private int ballposX = 350`: X-coordinate of the ball.
- `private int ballposY = 500`: Y-coordinate of the ball.
- `private int ballXdir = 1`: X-direction of the ball's movement.
- `private int ballYdir = 2`: Y-direction of the ball's movement.
- `private boolean play = false`: Represents the game's play state.
- `private boolean victory = false`: Indicates the victory state.
- `private boolean defeat = false`: Indicates the defeat state.
- `private Timer victoryDelayTimer`: Manages the delay for victory or defeat messages.

- **Methods:**
  - `public void paint(Graphics g)`: Handles the rendering of various game elements such as the paddle, ball, bricks, and score on the panel. It ensures proper updating and display of the game components by utilizing the provided graphics context.
  - `public void initializeGame()`: Initializes the game components and settings. This method sets up the bricks, scoring system, user interface, collision handling, input handling, paddle, and ball controllers based on the game's difficulty level and customization options.
  - `public void actionPerformed(ActionEvent e)`: Controls the game's flow by managing actions triggered by the game timer. It orchestrates the movement of the ball, updates the paddle, and continuously checks for victory or defeat conditions. Upon game start, this method ensures the smooth running and synchronization of various game components.
  - `public void checkVictory()`: Determines whether the player has achieved victory in the current level by checking if all bricks have been destroyed. Upon victory, it sets the appropriate state for showing victory or game-beat messages.
  - `public void showVictoryMessage()`: Displays a victory message in a dialog box when the player clears a level. Offers options to proceed to the next level or exit the game.
  - `public void showGameBeatMessage()`: Shows a congratulatory message in a dialog box when the player beats the entire game. Provides options to restart from level 1 or exit the game.
  - `public void checkDefeat()`: Determines whether the player has been defeated by checking if the ball has crossed the paddle. If defeated, it sets the appropriate state for displaying the defeat message.
  - `public void showDefeatMessage()`: Presents a defeat message in a dialog box when the player loses the game. Offers options to restart the current level or exit the game.
  - `public void keyPressed(KeyEvent e)`: Processes key-press events for controlling various game aspects. It handles input for starting the game and controlling the paddle's movement.

2.5 CollisionConnector

- **Methods:**
  - `public void checkCollisions(ballController,brickManager, paddleController,scoreConnector)`
    - Checks for collisions between the ball, bricks, and paddle.
    - Handles collision detection between the ball and the paddle, updating ball direction upon collision.
    - Manages collision detection between the ball and bricks, updating brick durability, score, and ball direction accordingly.

2.6 InputHandler

- **Methods:**
  - `public String keyPressed(KeyEvent e)`: Accepts a KeyEvent and determines the key that was pressed by the user. It checks for specific key codes such as `KeyEvent.VK_LEFT` for the left arrow key, `KeyEvent.VK_RIGHT` for the right arrow key, and `KeyEvent.VK_ENTER` for the enter key. It returns a string indicating the pressed key ('left', 'right', 'enter', or 'none' if no specific key is pressed).

2.7 InputConnector

- **Attributes:**
  - `private InputHandler inputHandler`: Manages the input handling within the game.
- **Methods:**
  - `public String keyPressed(KeyEvent e)`: Forwards the key-pressed event to the associated input handler's `keyPressed` method and returns the result.

2.8 Brick

- **Attributes:**
  - `private int durability`: Represents the strength or durability of the brick.
  - `private Color color`: Stores the color of the brick.
  - `private Double x`: Represents the x-coordinate position of the brick.
  - `private Double y`: Represents the y-coordinate position of the brick.
  - `public static Hashtable<Integer, Color> brickDurability`: Static map associating brick durability values with their respective colors.
- **Constructor:**
  - `public Brick(int durability, Color color, Double x, Double y)`: Initializes a brick with specific durability, color, and position.
- **Method:**
  - `public void updateBrick(int value)`: Updates the brick's durability and color based on the provided value from the 'brickDurability' map.

2.9 BrickManager

- **Attributes:**
  - `private Color bgcolor`: Stores the background color of the bricks.
  - `private Brick bricks[][]`: Represents the grid of bricks.
- **Constructor:**
  - `public BrickManager(int row, int col, Color bgColor)`: Initializes the brick manager with the specified number of rows, columns, and background color. It creates the grid of bricks and initializes them.
- **Methods:**
  - `public Brick[][] getBricks()`: Retrieves the grid of bricks.

- `private void createBricks(Integer x, Integer y)`: Generates bricks based on the given row and column count. It assigns random durability values to the bricks and stores them in the grid.
- `public void drawBricks(Graphics2D g)`: Renders the bricks on the game screen using the provided graphics context. It draws the bricks' shapes, colors, and borders based on their durability status.
- `public boolean allBricksHit()`: Checks if all bricks have been hit (their durability is zero). Returns true if all bricks are destroyed, otherwise returns false.
- `private int getRandomNumber()`: Generates a random number representing the durability of bricks. It returns a random value between 1 and 3.

2.10 PaddleController

- **Attributes:**
  - `private int paddlePosition`: Represents the current position of the paddle.
  - `private int paddleSpeed`: Indicates the speed of the paddle movement.
  - `private int paddleWidth`: Represents the width of the paddle.
  - `private InputConnector inputConnector`: Connects the paddle controller to handle input events.
- **Methods:**
  - `public void moveLeft()`: Moves the paddle to the left if it's within the game boundaries.
  - `public void moveRight()`: Moves the paddle to the right if it's within the game boundaries.
  - `public void handleInput(KeyEvent e)`: Handles the key-pressed event by determining whether to move the paddle left or right based on the input.
  - `public void paddleDisplay(Graphics2D g)`: Displays the paddle on the game screen using the provided graphics context.

2.11 BallController

- **Attributes:**
  - `private int ballPositionX`: Represents the current x-coordinate position of the ball.
  - `private int ballPositionY`: Represents the current y-coordinate position of the ball.
  - `private int ballSpeed`: Indicates the speed at which the ball moves.
  - `private int ballDirX`: Specifies the ball's movement direction along the x-axis.
  - `private int ballDirY`: Specifies the ball's movement direction along the y-axis.
  - `private CollisionConnector collisionConnector`: Manages collision detection and handling for the ball.
  - `private Color color`: Represents the color of the ball.
  - `private int gameover`: Stores the game state flag indicating whether the game is over.
- **Methods:**
  - `public void move(BrickManager brickManager, PaddleController paddleController, ScoreConnector scoreConnector)`: Manages the movement of the ball. It checks for collisions with bricks and the paddle, updates the ball's position based on its speed and direction, and handles collisions with the game boundaries. Additionally, it contains logic to determine if the ball has hit the bottom wall, indicating the game's end state.
  - `public int GameOver()`: Returns a flag indicating whether the game is over.
  - `public void ballDisplay(Graphics2D g)`: Renders the ball on the game screen using the provided graphics context.

2.12 ScoringSystem

- **Attributes:**
  - `private int score`: Stores the current score of the game.
- **Methods:**
  - `public void displayScore(Graphics2D g)`: Displays the current score on the game screen using the provided graphics context. It sets the color, font, and position to render the score.

2.13 ScoreConnector

- **Attributes:**
  - `private ScoringSystem scoringSystem`: Holds an instance of the ScoringSystem class to connect and manipulate the game's score.
- **Methods:**
  - `public void updateScore(int val)`: Updates the game score by adding the provided value to the current score via the associated ScoringSystem instance.
  - `public void scoreDisplay(Graphics2D g)`: Calls the ScoringSystem's method to display the current score on the game screen using the provided graphics context.

2.14 CustomizationManager

- **Attributes:**
  - `private String paddleDesign`: Stores the design or style of the paddle.
  - `private Color ballColor`: Represents the color of the ball.
  - `private Color backgroundTheme`: Stores the background color or theme of the game.

2.15 DifficultyManager

- **Attributes:**
  - `private int level`: Stores the current difficulty level of the game.
  - `private boolean gameBeat`: Indicates whether the game has been completed or beaten.
- **Method:**
  - `public void incrementLevel()`: Increments the difficulty level and manages the '`gameBeat`' flag.

2.16 UserInterface

- **Methods:**
  - `public void paddleDisplay(PaddleController, Graphics2D)`: Displays the paddle using the provided PaddleController and Graphics2D.
  - `public void drawBricks(BrickManager, Graphics2D)`: Draws the bricks using the provided BrickManager and Graphics2D.
  - `public void scoreDisplay(ScoreConnector, Graphics2D)`: Displays the score using the provided ScoreConnector and Graphics2D.
  - `public void ballDisplay(BallController, Graphics2D)`: Displays the ball using the provided BallController and Graphics2D.

3 Testing Report

3.1 Brick Manager

Here we are testing for whether all bricks have been hit or not, using the **allBricksHit()** function.

- Description of **allBricksHit()**:
  The function checks whether the durability attribute stored in the 2D array bricks[][] is greater than zero or not, (in the case of durability > 0, it means that atleast one brick is still intact)
- Test for **allBricksHit()** == false, when the bricks are intact
  Here we check whether the Assertion of the **allBricksHit()** funbction equals false.
- Test for **allBricksHit()** == true, when all bricks are hit.
  Here we check whether the Assertion of the **allBricksHit()** funbction equals true.

This section contains the description for the Unit and Integration testing by us for our Block-Basher Game. They are as follows:

3.2 Test Case 1: `testAllBricksHitWhenAllBricksIntact`

3.2.1 Description

This test case checks the behavior of the `allBricksHit()` method when the bricks present on the screen aren't broken by the ball.

3.2.2 Procedure

1. Create a `BrickManager` object. It creates a 2×2, Matrix that will contain the bricks.
2. Retrieve the bricks array.
3. Execute the `allBricksHit` method.

3.2.3 Expected Result

The result should be `false` as at least one brick is still intact.

3.2.4 Actual Result

The result matches the expected result.

3.3 Test Case 2: `testAllBricksHitWhenAllBricksHit()`

3.3.1 Description

This test case checks the behavior of the `allBricksHit()` method when the all the bricks are hit in the game.

3.3.2 Procedure

1. Create a `BrickManager` with 1×1 matrix that will contain the bricks.
2. Retrieve the bricks array and update the single brick to be hit.
3. Execute the `allBricksHit()` method.

3.3.3 Expected Result

The result should be `true` as all bricks are hit.

3.3.4 Actual Result

The result matches the expected result.

3.4 Test Case 3: `checkCollisions()`

3.4.1 Description

This test case checks the behavior of the `checkCollisions` method when the ball collides with the paddle.

3.4.2 Procedure

1. Set up the necessary objects: `CollisionConnector`, `BallController`, `BrickManager`, `PaddleController`, `ScoreConnector`.
2. Set the ball position for a collision with the paddle.
3. Set the paddle position for the collision.
4. Execute the `checkCollisions` method of `CollisionConnector`.

3.4.3 Expected Result

The ball direction in the Y-axis (`BallDirY`) should be reversed (`-1`) after the collision with the paddle.

3.4.4 Actual Result

The result matches the expected result.

3.5 Test Case 4: `testCheckCollisionsBricks()`

3.5.1 Description

This test case checks the behavior of the `checkCollisions` method when the ball collides with bricks.

3.5.2 Procedure

1. Set up the necessary objects: `CollisionConnector`, `BallController`, `BrickManager`, `PaddleController`, `ScoreConnector`.
2. Set the ball position for a collision with bricks.
3. Get the initial durability of a brick.
4. Execute the `checkCollisions` method of `CollisionConnector`.

3.5.3 Expected Result

There should be changes in the brick durability, score update, and ball direction after the collision with bricks.

3.5.4 Actual Result

The result matches the expected result. The initial durability of the brick is decreased after the collision.

3.6 Test Case 5: `testCheckPass()`

3.6.1 Description

This test case checks the behavior of the `checkDefeat()` method in `GameEngine` when the ball position does not reach the defeat threshold.

3.6.2 Procedure

1. Create a `GameEngine` object with a specific difficulty and customization manager.
2. Set up a `BallController` with a starting position that avoids reaching the defeat threshold.
3. Set the `BallController` for the `GameEngine`.
4. Execute the `checkDefeat()` method.

3.6.3 Expected Result

The `isDefeat()` method should return `false` as the ball position is within the playable area.

3.6.4 Actual Result

The result matches the expected result.

3.7 Test Case 6: `testCheckDefeat()`

3.7.1 Description

This test case checks the behavior of the `checkDefeat()` method in `GameEngine` when the ball collides with the bottom wall.

3.7.2 Procedure

1. Create a `GameEngine` object with a specific difficulty and customization manager.
2. Set up a `BallController` with a starting position that collides with the bottom wall.
3. Set the `BallController` for the `GameEngine`.
4. Execute the `checkDefeat()` method.

3.7.3 Expected Result

The `isDefeat()` method should return `true` as the ball collided with the bottom wall.

3.7.4 Actual Result

The result matches the expected result.

3.8 Conclusion

1. The unit tests for the `BrickManager` class were successful. The class behaves as expected in both scenarios, where some bricks are intact and when all bricks are hit.
2. The unit tests for the `CollisionConnectorTest` class were successful. The class handles collisions appropriately, reversing the ball direction after colliding with the paddle and updating brick durability upon collision.
3. The unit tests for the `GameEngineTest` class were successful. The class correctly handles defeat conditions, considering both scenarios where the ball does not reach the defeat threshold and when it collides with the bottom wall.

4 Snapshots of the Game

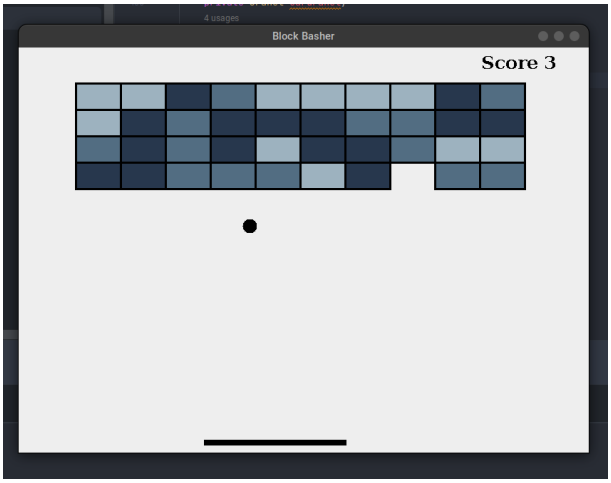

Figure 1: Brick Game

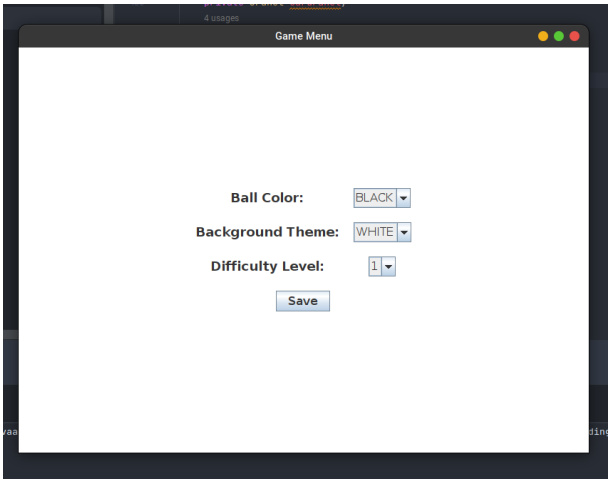

Figure 2: Settings Menu

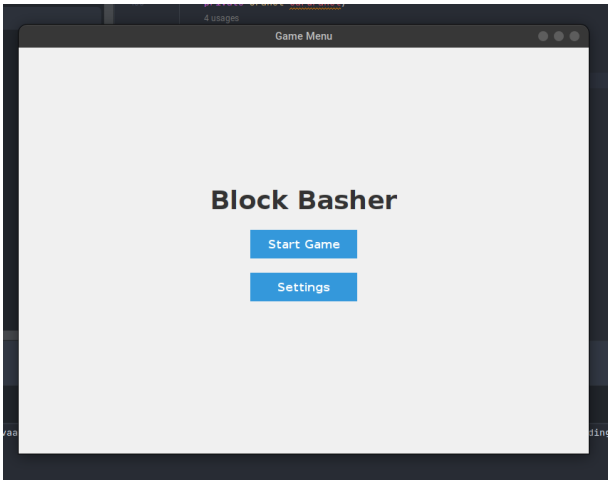

Figure 3