Square Dance Game

Introduction

This project implements a simple game where the player controls a square that moves around the screen. The square's size can also be adjusted using specific keys. The goal is to reach a target score to advance through levels.

Game Controls:

- **Arrow Keys:**
    - `Up Arrow`: Move the square up.
    - `Down Arrow`: Move the square down.
    - `Left Arrow`: Move the square left.
    - `Right Arrow`: Move the square right.
- **'z' Key**: Decrease the size of the square.
- **'x' Key**: Increase the size of the square.
- **'q' Key**: Quit the game.

Game Description

- The square starts at the top-left corner of the screen.
- Players can move the square using the arrow keys and change its size using the 'z' and 'x' keys.

- The target score increases with each level, and the player needs to reach that score to advance.
- The game ends if the number of moves exceeds 250.

Directory Structure:

```
/
├── README.md
├── SquareGame.jack
├── Square.jack
├── Main.jack
├── demo.mov
```

Instructions for Playing the Game:

1. Clone or download the repository.
2. Run the `Main.jack` file to start the game.
3. Use the arrow keys to move the square around the screen.
4. Use the 'z' and 'x' keys to adjust the square's size.
5. The target score will increase as you advance through levels.
6. When the target score is achieved, you will move to the next level.
7. Press 'q' to quit the game at any time.

Design:

The game is designed with the following key principles:

1. **Modularity**: The game is split into different classes:
    - `SquareGame`: Manages the overall game flow.

    - `Square`: Handles the behavior and graphics of the square object.
    - `Main`: Initializes and starts the game.
2. **Separation of Logic and GUI**: The game logic is separated from the graphical rendering. `Square` handles the graphical representation, while the game logic resides in `SquareGame`.
3. **Proper Use of Constructors and Disposers**: Each class is properly initialized with constructors and deallocated using disposers to manage memory efficiently.
4. **Score and Leveling System**: The game includes a scoring system, and the player advances to new levels by reaching the target score.

Code Documentation:

Each class and method is documented with comments explaining their functionality. You can refer to these comments within the code for more information.

Example:

```
/**
 * Moves the square up by 2 pixels.
 * If the square would move out of bounds, it does nothing.
 */
method void moveUp() {
    if (y > 1) {
        do Screen.setColor(false);
        do Screen.drawRectangle(x, (y + size) - 1, x + size, y + size);
```

```
            let y = y - 2;
            do Screen.setColor(true);
            do Screen.drawRectangle(x, y, x + size, y + 1);
        }
        return;
    }
```

Application Complexity:

The game presents a simple but engaging challenge. It tests basic game mechanics, including object movement, size adjustments, and scoring.

Virtue:

- **Interactivity**: The player interacts with the game by moving the square and adjusting its size.
- **Challenge**: Each level requires reaching a higher score.
- **Final Touch**: The graphical elements, such as the square and target points, provide an enjoyable visual experience.

Correctness:

The game runs as expected, handling user inputs for movement, size adjustments, and level transitions. It checks for game over conditions and displays the score correctly.

Algorithmic Efficiency:

The game’s graphical animations are optimized to ensure smooth movements and minimal lag during interactions.