Refactoring Report

Project Name: City Escape Game (Group 9)
Authors: Woochan (Andy) Jung, Ehsanullah Mohammadi

A. Scope of Refactoring

Affected Modules/Classes in Game Logic:

- Map class
- Cop class
- Character class

Affected Modules/Classes in Game UI:

- MainScreenPanel class

B. Original Issues

Game Logic:

1. **Large Methods**
    - The `aStarPathfinding` method in the `Map` class was performing multiple responsibilities, including initialization, node evaluation, path reconstruction, and neighbour processing, leading to reduced readability and maintainability.
2. **Code Duplication**
    - Neighbour evaluation logic and path-checking conditions were repeated in multiple areas, causing unnecessary redundancy.
3. **Limited Encapsulation**
    - The logic for moving cops relied on multiple methods being separately invoked, increasing the complexity of the code and reducing encapsulation.
4. **Repetitive Direction Logic**
    - The logic for determining direction and moving cops toward the thief was handled in a verbose manner, causing unnecessary repetition.

Game UI:

1. **Lack of Code Reusability**
    - Repeated logic for panel creation and gradient background setup increased redundancy and maintenance difficulty.
2. **Reduced Readability and Maintainability**
    - Inline logic, hardcoded values, and a lengthy constructor made the code harder to read, debug, and modify.
3. **Limited Extensibility**

    - Hardcoded button actions and inconsistent styling made it challenging to reuse or extend the class without significant modifications.
4. **Violation of Separation of Concerns**
    - The constructor handled both UI setup and behaviour definition, mixing responsibilities and making the class harder to manage.

C. Refactored Changes

Game Logic:

1. *Modularized A Pathfinding Method**
    - **Original Issue**: The `aStarPathfinding` method was responsible for multiple tasks, including initializing nodes, processing neighbours, and reconstructing the path. This violated the Single Responsibility Principle and made the code harder to follow.
    - **Refactored Change**:
        - Divided the method into smaller, focused methods:
            - `initializeOpenList` for node initialization.
            - `isTargetReached` to check if the target node was reached.
            - `processNeighbors` to handle neighbour evaluation.
            - `reconstructPath` to handle path reconstruction.
        - Each method handles a single responsibility, improving readability and maintainability.
2. **Enhanced Neighbor Evaluation Logic**
    - **Original Issue**: The neighbour evaluation logic was repeated in multiple areas, making the code harder to update and maintain.
    - **Refactored Change**:
        - Centralized neighbour evaluation logic in `processNeighbors`.
        - Introduced `isBetterPath` to encapsulate the logic for comparing nodes in the open list, reducing redundancy.
3. **Simplified Cop Movement Logic**
    - **Original Issue**: The `executeCopMove` method handled direction determination and cop movement separately, increasing verbosity (how wordy it is) and reducing cohesion.
    - **Refactored Change**:
        - Integrated the `simulatePosition` method into the cop movement logic to simplify position updates.
        - Refactored the `executeCopMove` method to determine the direction and directly set the new position.
        - Removed moveup, left, down , right methods to reduce verbosity.
4. **Improved Direction Handling**
    - **Original Issue**: The logic for determining movement direction based on the current and next positions was verbose and scattered across the codebase.
    - **Refactored Change**:

        - Consolidated the direction determination logic into a reusable method, `determineDirection`.
        - Improved the clarity and modularity of movement handling.

Game UI:

1. **Modularized Panel and Background Creation**
    - **Original Issue**: Repeated panel setup and gradient logic.
    - **Refactored Change**:
        - Introduced `createPanelWithBackground` to centralize panel creation with a consistent background.
        - Moved gradient background logic into a helper method `setGradientBackground` for better encapsulation.
2. **Centralized Styling with Constants**
    - **Original Issue**: Hardcoded colours, fonts, and dimensions scattered across the code.
    - **Refactored Change**:
        - Defined constants (e.g., `BACKGROUND_COLOR`, `TITLE_FONT`) for all styling elements to ensure consistency and make updates easier.
3. **Simplified Listener Logic**
    - **Original Issue**: Mouse listener logic for buttons was inline, adding complexity to the constructor.
    - **Refactored Change**:
        - Extracted listener logic into dedicated methods (e.g., `createPlayMouseListener`, `createExitMouseListener`) to separate behaviour from UI setup.
4. **Standardized Button Creation**
    - **Original Issue**: Button styling and creation logic were verbose and inconsistent.
    - **Refactored Change**:
        - Centralized button creation into `createButtonLabel`, ensuring consistent design and behaviour across all buttons.
5. **Decoupled Button Behavior from Panel**
    - **Original Issue:** Hardcoded actions for buttons (e.g., System.exit(0)) limited reuse.
    - **Refactored Change:**

      Used an ActionListener for the "PLAY" button to decouple its behaviour, enabling flexibility for different implementations.

D. Results of Refactoring

Game Logic:

1. **Improved Readability**

    - Breaking down large methods into smaller, focused methods enhanced code readability and made it easier to debug and understand.
2. **Reduced Redundancy**
    - Centralized logic for neighbour evaluation and direction handling eliminated code duplication.
3. **Enhanced Maintainability**
    - By encapsulating pathfinding and movement logic, future changes can now be made using specific methods without impacting the rest of the codebase.
4. **Better Cohesion**
    - The refactored code ensures that each method handles a single, well-defined responsibility, improving cohesion across the affected classes.

Game UI:

1. **Improved Reusability**
    - Encapsulated common logic (panel creation, gradient setup) into reusable methods, reducing code duplication.
2. **Enhanced Readability and Maintainability**
    - Simplified the constructor and reduced inline logic, making the class easier to read, debug, and modify.
3. **Better Extensibility**
    - Centralized styling and decoupled button behaviour allow easy modifications and adaptation of the class for other screens.
4. **Increased Cohesion**
    - Ensured methods handle single, well-defined responsibilities, adhering to the Single Responsibility Principle.