**OOP-Based 2D Maze Game**

**Programming with C++**

****

**Submitted by**

Name: Prashannaa Silwal

Student ID: 10432

University ID: 2400858

Cyber Security and Digital Forensics

Kathmandu, Nepal

April,2025

**TITLES**

* **Introduction**
* **GameEntity.h**
* **Enemy.h**
* **Player.h**
* **PacmanGame.h**
* **PacmanGame.cpp**
* **main.cpp**
* **README.md**
* **Summary**

**Introduction**

The 2D Pac-Man game is an OOP-based, arcade-inspired project featuring a dynamic 10x10 maze. Players collect items (\*) for points, avoid enemies (X), and reach the exit (E) to progress levels. It includes score tracking, randomized mazes, and file I/O for saving/loading configurations. The game is modular and functional, with potential enhancements like dynamic maze scaling.

**GameEntity.h**  
The GameEntity class serves as a base class for all game entities (e.g., players, enemies) in a gridbased game. It provides shared attributes and functionality while enforcing custom movement logic via a pure virtual function.

**Key Features**

1. **Attributes**:
   * x and y represent the entity's position in a 2D grid.
   * protected access allows derived classes to directly access these attributes.
2. **Methods**:
   * move: A pure virtual function to enforce movement logic in derived classes.
   * getX and getY: Provide read-only access to the entity's current position.
   * setPosition: Updates the position of the entity.
3. **Constructor**:
   * Initializes the entity's position with startX and startY.

**Strengths**

1. **Abstraction**: The move function ensures all derived classes implement their own movement logic.
2. **Encapsulation**: Position-related methods (getX, getY, setPosition) abstract position management.
3. **Reusability**: Common functionality is centralized for use across different entity types.

**Enemy.h**The Enemy class is a derived class of GameEntity that represents an enemy entity in a grid-based game. It implements random movement with collision detection to avoid walls.

**Key Features**

1. **Inheritance**
   * Inherits from the GameEntity class, reusing attributes like x, y and methods like move.
2. **Random Movement**:
   * Movement offsets (dx, dy) are randomly chosen from -1, 0, or 1, allowing the enemy to move horizontally, vertically, or diagonally.
3. **Collision Handling**:
   * Ensures the enemy does not move into walls (represented by '#') by checking the maze grid before updating position.

**Strengths**

1. **Encapsulation**:
   * Uses protected x and y attributes from the base class, allowing reuse of position management.
2. **Collision Logic**:
   * Prevents invalid movement into walls by checking the grid value.
3. **Simplicity**:
   * The random movement logic is straightforward and easy to understand.

**Player.h**  
The Player class represents the player entity in a grid-based game. It is derived from the GameEntity base class and adds features specific to the player, such as score tracking, move counting, and controlled movement.  
  
Key Features

1. **Attributes**:
   * score: Tracks the player's score.
   * moves: Tracks the number of moves the player has made.
2. **Movement**:
   * **move(char direction, const vector<vector<char>>& maze)**:
     + Allows the player to move in four directions (w, s, a, d) based on input.
     + Prevents movement into walls ('#').
     + Updates the player's position and increments the move count.
   * **move(const vector<vector<char>>& maze)**:
     + Dummy implementation, as player movement is handled via the other move method.
3. **Score Management**:
   * **collectItem()**:
     + Increases the player's score by 10 when called.
   * **getScore()**:
     + Returns the current score.
   * **getMoves()**:
     + Returns the total number of moves.
4. **Constructor**:
   * Initializes the player’s position (x, y), score (0), and moves (0).

**Strengths**

1. **Encapsulation**:
   * Keeps score and moves private with public accessor methods (getScore, getMoves).
2. **Collision Handling**:
   * Prevents the player from moving into walls.
3. **Player-Specific Features**:
   * Tracks the score and moves, enhancing gameplay mechanics.

**PacmanGame.h**  
The PacmanGame class encapsulates the logic and mechanics for a PacMan game. It includes game entities such as the player and enemies, along with methods for managing the maze, input, and game progression.

**Key Features**

1. **Attributes**:
   * maze: A 2D grid representing the game board.
   * player: An instance of the Player class representing the player.
   * enemies: A vector of Enemy objects representing the enemies.
   * level: Tracks the current game level.
   * gameOver: A flag indicating whether the game has ended.
2. **Private Methods**:
   * **generateMaze**: Creates the maze layout.
   * **placeCollectibles**: Adds collectibles (e.g., dots or power-ups) to the maze.
   * **placeEnemies**: Positions enemies in the maze.
   * **saveMazeToFile**: Saves the current maze to a file for persistence.
   * **loadMazeFromFile**: Loads a saved maze from a file.
3. **Public Methods**:
   * **PacmanGame**: Constructor to initialize the game.
   * **displayMaze**: Displays the maze on the screen.
   * **handleInput**: Processes player input (e.g., movement).
   * **updateGame**: Updates the game state (e.g., player/enemy movement, collisions).
   * **run**: Main loop to execute the game.

**Strengths**

1. **Encapsulation**:
   * Combines all core game logic into a single class, maintaining modularity.
2. **Game Management**:
   * Provides methods for generating, saving, and loading the maze, which is crucial for game flow.
3. **Player and Enemy Integration**:
   * Integrates Player and Enemy classes for entity-specific behavior.
4. **Input Handling**:
   * Processes player inputs via handleInput.

**PacmanGame.cpp**  
This implementation defines the core logic for the PacmanGame. It manages the maze, player, enemies, and game mechanics, providing a functional and interactive Pac-Man-style game.

**Key Features**

1. **Maze Generation**:
   * generateMaze: Dynamically generates a 10x10 maze with walls, an exit (E), and random internal walls.
   * placeCollectibles: Randomly places 5 collectibles (\*) within the maze.
   * placeEnemies: Spawns 2 enemies (X) at random positions.
2. **Game State Management**:
   * saveMazeToFile and loadMazeFromFile: Save and load the maze configuration to/from a file (maze.txt).
   * displayMaze: Clears the console and displays the current maze, player (P), enemies (X), score, moves, and level.
3. **Player and Enemy Interaction**:
   * handleInput: Handles player movement, collectible collection, and level progression upon reaching the exit.
   * updateGame: Updates enemy movement and checks for collision with the player, setting gameOver if an enemy reaches the player.
4. **Game Loop**:
   * run: Main game loop that displays the maze, processes player input, updates the game state, and pauses for 200ms in each iteration. Ends the game when gameOver is set to true.

Strengths

1. **Dynamic Maze and Collectibles**:
   * Each level generates a new maze with randomized walls and collectibles, keeping gameplay fresh.
2. **File Persistence**:
   * Allows saving and loading the maze, making it possible to resume gameplay or analyze the maze layout.
3. **Player and Enemy Integration**:
   * Incorporates the Player and Enemy classes seamlessly into the gameplay mechanics.
4. **Level Progression**:
   * Progresses to a new level when the player reaches the exit, encouraging continuous gameplay.
5. **Interactive Gameplay**:
   * Uses \_kbhit and \_getch for real-time input handling, enhancing interactivity.

**main.cpp**

The provided main function serves as the entry point for the Pac-Man game. It initializes an instance of the PacmanGame class and starts the game loop using the run() method.

**Key Features**

1. **Game Initialization:**
   * Creates an instance of the PacmanGame class, initializing all necessary game components (maze, player, enemies, collectibles, etc.).
2. **Game Execution:**
   * Calls the run() method of the PacmanGame class, which contains the main game loop for handling input, updating the game state, and rendering the maze.
3. **Program Termination**:
   * Returns 0 to indicate successful program execution.

Strengths

1. **Simplicity**:
   * The main function is clean and concise, delegating all game logic to the PacmanGame class.
2. **Modularity**:
   * The use of a dedicated PacmanGame class ensures that the main function remains focused on initiating the game rather than implementing game-specific logic.
3. **Ease of Testing**:
   * The function can easily be extended or modified to test other aspects of the game, such as initializing different game configurations.

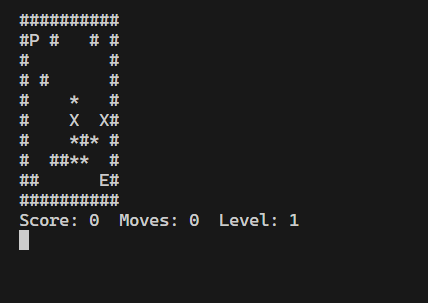
**README.md**This file explains the features and contains info on how to play the game.

**Summary**

This project is a 2D PacMan game built using OOP principles. Players navigate a 10x10 maze grid, collecting items (\*) for points, avoiding enemies (X), and reaching the exit (E) to progress to the next level.

The game features dynamic maze generation, score and move tracking, and level progression. Enemies move randomly, adding an element of challenge. The game also includes file I/O for saving and loading maze layouts.

While the game is functional and modular, improvements could include dynamic maze scaling, smarter enemy AI, platform-independent libraries, and better difficulty scaling. Overall, it is a well-structured and entertaining game with room for enhancement.

****