project-root/

│

├── src/ # Source files

│ ├── models/ # Game models (Player, Enemy, etc.)

│ │ ├── Player.ts

│ │ ├── Enemy.ts

│ │ ├── Wall.ts

│ │ ├── Treasure.ts

│ │ └── GameObject.ts (baseclass)

│ │

│ ├── core/ # Core classes and utilities

│ │ ├── GameController.ts

│ │ ├── GameMap.ts

│ │ └── InputHandler.ts

│ │

│ ├── assets/ # (Optional) Sprites, images, sounds

│ │ └── (image files, sound files)

│ │

│ ├── main.ts # Entry point

│ └── config.ts # Game configuration constants

│

├── dist/ # Compiled JavaScript files (output of tsc)

│ └── (compiled .js files)

│

├── index.html # HTML file with <canvas> tag

├── tsconfig.json # TypeScript configuration

└── package.json # npm package details

**1. GameObject (Base Class)**

This would be the **abstract** base class for all game objects (player, enemy, walls, treasures). It should include common properties like position and a method to render the object on the canvas.

**Properties:**

* x: number: X coordinate of the object.
* y: number: Y coordinate of the object.
* color: string: The color or texture for the object.

**Functions:**

* render(ctx: CanvasRenderingContext2D): Abstract method to render the object on the canvas (to be implemented by subclasses).

**2. Player (Subclass of GameObject)**

This class would represent the **player** (the dark green circle).

**Additional Properties:**

* radius: number: The size of the player.
* speed: number: The movement speed of the player.

**Functions:**

* move(direction: string): Function to update the player’s position based on input (up, down, left, right).
* collectTreasure(treasure: Treasure): Logic to collect treasures when overlapping with a Treasure.

**3. Enemy (Subclass of GameObject)**

This class represents the **enemy** (red circle).

**Additional Properties:**

* radius: number: The size of the enemy.
* speed: number: The speed of the enemy’s movement.

**Functions:**

* move(): Logic for automatic movement, likely to chase the player or patrol around.
* attackPlayer(player: Player): Logic to damage or impact the player when colliding.

**4. Wall (Subclass of GameObject)**

The brown squares would be represented by the Wall class. These are static obstacles.

**Additional Properties:**

* width: number: The width of the wall.
* height: number: The height of the wall.

**Functions:**

* No movement is needed, just render(ctx: CanvasRenderingContext2D).

**5. Treasure (Subclass of GameObject)**

The yellow squares represent **treasures**. These disappear when collected by the player.

**Additional Properties:**

* isCollected: boolean: Tracks whether the treasure has been collected.

**Functions:**

* collect(): Marks the treasure as collected and removes it from the canvas.

**6. GameMap**

This class represents the **game map/grid** (the overall layout, like grass, walls, positions of objects, etc.). It manages the collection of all game objects and draws the background (grass) and game objects on the canvas.

**Properties:**

* width: number: Width of the map.
* height: number: Height of the map.
* objects: GameObject[]: A collection of all game objects (player, walls, treasures, enemies).

**Functions:**

* renderMap(ctx: CanvasRenderingContext2D): Renders the map background (grass) and all game objects.
* checkCollisions(): Check for collisions between player and walls, treasures, or enemies.

**7. GameController**

This class manages the **game state** and input handling.

**Properties:**

* player: Player: Reference to the player.
* map: GameMap: Reference to the game map.
* enemies: Enemy[]: Collection of enemies.

**Functions:**

* startGame(): Initializes and starts the game loop.
* handleInput(): Handles player movement based on key presses.
* gameLoop(): The main game loop, responsible for updating the game state and redrawing the canvas (e.g., move enemies, check for collisions, update the player).

**Class Diagram Summary:**

1. **GameObject** (Base class)
   * Player (Subclass)
   * Enemy (Subclass)
   * Wall (Subclass)
   * Treasure (Subclass)
2. **GameMap** (Manages the grid and all game objects)
3. **GameController** (Handles input, the game loop, and game state)

**Suggested Flow:**

1. **GameController** initializes the game by creating a Player, Enemy, Wall, and Treasure objects, places them on the GameMap, and starts the game loop.
2. The game loop repeatedly calls render() on the GameMap, which in turn calls render() on all the GameObject subclasses to draw them on the canvas.
3. **Input Handling**: Player’s movement and interaction (like collecting treasures) is handled by the GameController based on keypresses.
4. **Enemy Logic**: Enemies move automatically (e.g., chasing the player or following a patrol path).
5. **Collision Detection**: The GameMap checks for collisions between the player and other objects (walls, treasures, enemies).

**Key Functionality Breakdown:**

* **Player**: Handles movement and collecting treasures.
* **Enemy**: Moves automatically and interacts with the player.
* **Walls**: Static obstacles that block movement.
* **Treasure**: Collectible items that disappear when collected.
* **GameMap**: Manages the overall environment, including rendering and collision detection.
* **GameController**: Manages the game state, player input, and game loop.