DETAILED PROJECT DESCRIPTION

1. Introduction

The "SHERLOCK: A STUDY IN PINK – Part 2" project is a highly complex game logic system built on an object-oriented programming foundation with dynamic memory management. It simulates a criminal pursuit through a maze, featuring Sherlock, Watson, and the criminal as main characters, along with various supporting entities such as robots and items. This document provides a detailed description of the system's functional requirements, from constructing map components to handling complex interactions among the different objects in the game.

2. Project Scope

The system is divided into the following main modules:

- **Map Components:** Manages map elements consisting of traversable paths (Path), real walls (Wall), and deceptive walls (FakeWall), with special detection requirements for Watson based on his EXP.
- **Map and Position:** Constructs the map as a two-dimensional array of objects, each initialized at a specific position, and includes the `Position` class with methods for string formatting and position validation.
- **Moving Entities:** Defines an abstract class `MovingObject` and its concrete subclasses (Sherlock, Watson, Criminal). Each entity follows its own movement rule (`moving_rule`) and special conditions (HP/EXP, Manhattan distance, etc.).
- **Array of Moving Entities:** Manages a collection of moving objects with methods for adding new entities, checking capacity, and displaying information.
- **System Configuration:** Reads a configuration file containing parameters for map size, positions, number of objects, and initial settings for the characters.
- **Robots and Items:** Implements a variety of robot types (RobotC, RobotS, RobotW, RobotSW) with unique movement rules and distance calculations, and integrates an item system (MagicBook, EnergyDrink, FirstAid, ExcemptionCard, PassingCard) that modifies character stats when used.
- **Inventory and Item Exchange:** Each character has an individual bag (`SherlockBag`, `WatsonBag`) for storing, retrieving, and exchanging items when meeting another character.
- **Game Process – StudyInPinkProgram:** Coordinates the entire movement process, collision handling, robot creation (after every three valid moves by the Criminal), and game state updates until termination conditions are met (HP = 0 or the criminal is caught).

- **Testing:** Provides the `TestStudyInPink` class to thoroughly test all modules and ensure system accuracy.

3. Detailed Functional Requirements

3.1. Map Components

- **MapElement**
  - **Attribute:** `protected ElementType type` (an enum with values: `PATH`, `WALL`, `FAKE_WALL`).
  - **Constructor:** Takes one parameter of type `ElementType`.
  - **Virtual destructor** and a method `getType()` to return the element type.
- **Subclasses**
  - **Path, Wall:** Created according to system requirements.
  - **FakeWall:**
    - Has a private attribute `req_exp` computed by `(r * 257 + c * 139 + 89) % 900 + 1`, where `r` and `c` are row and column indices.
    - *Note:* Sherlock can always detect a FakeWall, while Watson must have sufficient EXP to pass through it.

3.2. The Map (Class Map)

- **Attributes:**
  - Number of rows (`num_rows`) and columns (`num_cols`), and a two-dimensional array storing `MapElement` objects polymorphically.
- **Constructor:**
  - Builds the map with `Wall` and `FakeWall` objects at specified positions, while the remaining positions are `Path`.
- **Destructor:**
  - Frees all dynamically allocated memory.
- **Method:**
  - `isValid(Position, MovingObject*)` checks whether a position is valid for a given moving object, based on the object's characteristics and the map element type.

3.3. Position (Class Position)

- **Attributes:**
  - `r` and `c`, describing the row and column.

- **Constructors:**
  - One that accepts `(r, c)` and another that accepts a string in the format "(r,c)".
- **Methods:**
  - Getters and setters for `r` and `c`;
  - `str()` returns a string in the format "(r,c)";
  - `isEqual(int, int)` compares positions;
  - A static constant `npos` signifies an invalid position (r = −1, c = −1).

3.4. Moving Entities (Abstract Class MovingObject)

- **Attributes:**
  - `index`: the object's index in the array;
  - `pos`: current position;
  - `map`: pointer to the `Map` object;
  - `name`: the object's name.
- **Constructor:**
  - Initializes these attributes (with `name` defaulting to an empty string if not provided).
- **Methods:**
  - Pure virtual `getNextPosition()` to compute the next move;
  - `getCurrentPosition()` returns the current position;
  - Pure virtual `move()` executes a move;
  - Pure virtual `str()` returns a formatted string describing the object.

3.5. Moving Characters

- **Sherlock**
  - **Constructor:**
    - Accepts `moving_rule`, `init_hp`, `init_exp`. HP is capped at 500, and EXP is capped at 900.
  - **`getNextPosition()`** uses the movement rule string cyclically.
  - If the next position is invalid, returns `npos`.
  - **`move()`** updates Sherlock's position if the move is valid.
  - **`str()`** returns: `Sherlock[index=<index>;pos=<pos>;moving_rule=<moving_rule>]`.
- **Watson**
  - Similar to Sherlock but with a slightly different string format in `str()`.
- **Criminal**
  - Moves to the position that maximizes the sum of Manhattan distances from Sherlock and Watson.

- In case of ties, the order of priority is: `U` (Up), `L` (Left), `D` (Down), `R` (Right).
- **`str()`** returns: `Criminal[index=<index>;pos=<pos>]`.

3.6. Array of Moving Entities (ArrayMovingObject)

- **Purpose:**
  - Manages an array of moving objects (`MovingObject*`) with attributes:
    - `arr_mv_objs`, `count`, `capacity`.
- **Methods:**
  - **Constructor:** initializes the array with a specified capacity.
  - **Destructor:** deallocates memory.
  - `isFull()` checks if the array has reached capacity.
  - `add(MovingObject*)` adds an object if there is space, then returns `true`; otherwise returns `false`.
  - `str()` returns a string in the format: `ArrayMovingObject[count=<count>;capacity=<capacity>;<MovingObject1>;...]`.

3.7. System Configuration (Class Configuration)

- **Function:**
  - Reads a configuration file specifying:
    - Map dimensions (rows and columns);
    - Maximum number of moving objects;
    - Lists of positions for Walls and FakeWalls;
    - Initialization parameters for Sherlock and Watson (moving_rule, position, HP, EXP);
    - The Criminal's initial position;
    - The number of steps (`NUM_STEPS`).
- **Constructor:**
  - Processes the file and initializes attributes accordingly.
- **`str()`** returns a string enumerating all parameters in the prescribed order.

3.8. Robots and Items

- **Robot**
  - **Types:** RobotC, RobotS, RobotW, RobotSW, each with unique attributes (e.g., they can move on FakeWall but not on Wall) and pointers to `Criminal`, `Sherlock`, or `Watson` as required.
  - **Movement methods:**

        - `getNextPosition()` and `move()` each follow specific rules:
            - **RobotC**: Moves according to the Criminal's position.
            - **RobotS, RobotW, RobotSW**: Compute positions based on optimal Manhattan distance, with a priority starting at "Up" and moving clockwise.
    - **`str()`** returns a string in the format: `Robot[pos=<pos>;type=<robot_type>;dist=<dist>]` (RobotC omits `dist`).
- **Items (BaseItem and subclasses)**
    - **Types:** MagicBook, EnergyDrink, FirstAid, ExcemptionCard, PassingCard. Each defines two pure virtual methods: `canUse(...)` and `use(...)`.
    - Each item has its own usage conditions (e.g., MagicBook requires `exp ≤ 350`, EnergyDrink requires `hp ≤ 100`, etc.).
    - Items are created according to a "dominant digit sum" rule derived from multiplying the row and column indices of the robot's spawn location.

3.9. Inventory (BaseBag and Derived Classes)

- **BaseBag**
    - Provides methods:
        - `insert(BaseItem*)`,
        - `get()`,
        - `get(ItemType)`,
        - `str()` – displays items from the head to the tail of a singly linked list.
- **SherlockBag & WatsonBag**
    - Each has a different capacity (13 items for Sherlock, 15 for Watson).
    - When Sherlock and Watson meet, they exchange items as follows:
        - Sherlock gives all PassingCards (if any),
        - Watson gives all ExcemptionCards (if any).
        - If multiple cards exist, each one is transferred in order from the start of the bag.

Detailed item information:

Item	Effect	Usage Condition
MagicBook	Contains ancient magical knowledge that allows Sherlock and Watson to quickly boost their experience, recovering an extra 25% EXP when used.	`exp ≤ 350`

Item	Effect	Usage Condition
EnergyDrink	An energy drink that restores an additional 20% HP when used.	`hp ≤ 100`
FirstAid	A first-aid kit that restores an additional 50% HP when used.	`hp ≤ 100` **or** `exp ≤ 350`
ExcemptionCard	Exempts the user from HP and EXP penalties when failing a challenge at a given position.	Only **Sherlock** can use this card, and only if **Sherlock's HP** is an **odd number**.
PassingCard	Allows the user to bypass a challenge without performing it. The card has a `challenge` attribute (e.g., "RobotS" to bypass RobotS). If the card type is "all," it can be used for any challenge. Otherwise, if the card type does not match the encountered challenge, the user loses 50 EXP even though the exemption still applies.	Only **Watson** can use this card, and only if **Watson's HP** is an **even number**.

3.10. Game Process – StudyInPinkProgram

- **Module Integration**
  - Initialization: Reads the configuration file, constructs the map, creates the moving objects (Criminal, Sherlock, Watson), and adds them to the `ArrayMovingObject`.
- **Movement Loop**
  - **If Sherlock encounters**:
    - **RobotS**: Sherlock must solve a puzzle to defeat RobotS. If his EXP > 400, he succeeds and acquires the robot's item; otherwise, he loses 10% EXP.
    - **RobotW**: Sherlock automatically defeats RobotW and obtains its item without a fight.
    - **RobotSW**: Sherlock can only win if his EXP > 300 and HP > 335. If he succeeds, he gains the robot's item; if he fails, he loses 15% HP and 15% EXP.
    - **RobotC**: Indicates Sherlock is adjacent to the criminal. If Sherlock's EXP > 500, he captures the criminal (no item gained from the robot). Otherwise, the criminal escapes. Nevertheless, the robot is destroyed and Sherlock obtains its item.
    - **FakeWall**: As described above.

    - **If Watson encounters**:
        - **RobotS**: Watson does nothing and does not acquire any item.
        - **RobotW**: Watson must fight RobotW; he wins only if HP > 350. If he wins, he acquires the item; if he loses, Watson's HP decreases by 5%.
        - **RobotSW**: Watson can only win if his EXP > 600 and HP > 165. If he succeeds, he acquires the item; otherwise, he loses 15% HP and 15% EXP.
        - **RobotC**: Watson is adjacent to the criminal but cannot capture him because RobotC blocks him. He still destroys the robot and obtains its item.
        - **FakeWall**: As described above.
    - Before and after each encounter, the character checks their inventory to see if any item can be used.
- **Termination Conditions**
    - The program ends if either Sherlock or Watson's HP reaches 0, or if the criminal is captured according to the rules.
- **Display Methods**
    - `printStep`, `printResult` output detailed information about the system's state at each step.

---

4. Additional Requirements

- **Development Environment**
    - The program must compile and run on a Unix platform using `g++` with the `-std=c++11` option. Only two files can be modified:
        - `study_in_pink2.h`
        - `study_in_pink2.cpp`
- **Include Directives**
    - In `study_in_pink2.h`, there must be exactly one `#include "main.h"`.
    - In `study_in_pink2.cpp`, there must be exactly one `#include "study_in_pink2.h"`.