Programming Assignments 4 & 5
Final Assignment
Check webcanvas for due times
There is no late deadline for assignment 5

Assignment Objectives

- Provide practice with object oriented design documentation skills
- Provide practice with new abstract base class skills, and virtual inheritance
- Provide cumulative practice with past class building, aggregation, overloading, and templating skills.

Important Notes

1. Assignment Overview

This assignment requires the creation of formal technical documentation. A rough outline and description of the expected structure have been provided; however, students are encouraged to conduct additional research and expand their work into a comprehensive design document if desired.

2. UML Diagram Requirements

UML diagrams are a required component of this assignment.

- Diagrams may be broken into smaller sections if necessary for clarity, but they must be organized so the reader can easily piece them together to form a complete system overview.
- It is recommended to use draw.io (freely available online), but other tools such as PowerPoint, Keynote, or any preferred drawing software are acceptable.
- The primary evaluation criteria are:
    - All classes in the system are shown.
    - Relationships between classes are clearly illustrated.
    - Correct UML formatting and symbols are used throughout.

3. Writing Style Guidelines

This is a **formal design document**. The following guidelines must be adhered to:

- Use full words (e.g., "you" instead of "u").
- Ensure proper grammar and spelling.
    - While a few minor errors are acceptable, grammar and spelling will factor into the assignment's grading rubric.
    - Students who require assistance are encouraged to utilize third-party grammar tools or the university's Writing Center, which is available through the Student Academic Success Fee.
- Maintain an **objective, technical, and concise** writing style throughout the document.
- Utilize **passive voice** appropriately, similar to the style used in the provided sample documentation.

4. Document Clarity

The document must be written clearly enough that any student who has completed CS202 could implement the code solely based on the design document.

5. Collaboration Policy

Students are permitted to work individually or with **one partner**.

- Groups of three or more are **not allowed**.
- The required amount of work for the assignment is the same, regardless of whether it is completed individually or with a partner.

If working with a partner:

- It is recommended to create a **private GitHub repository** to facilitate collaboration.
- Each partner should create and commit to their **own branch**.

Students interested in simultaneous pair programming are encouraged to use **GitHub Codespaces**, which is available for free to students.

6. Programming Requirements

The following libraries will be required to use:

- `stdlib.h`: Required for random AI selection generation.
- `time.h`: Recommended for seeding the random number generator (`rand` function).

Assignment

In this assignment, you will complete two related tasks: writing a **design document** (PA4) and implementing the corresponding **code** (PA5).

The design document (PA4) will serve as the **roadmap** for your code development in PA5. It is critical to approach this phase thoughtfully and thoroughly. A poorly written or incomplete design document will make the implementation phase significantly more difficult.

PA5 will involve the **implementation of a Treasure Hunt Adventure game**. A detailed description of the game requirements and functionality is provided below.

🎮 Treasure Hunt Adventure Game: Detailed Description

The **Treasure Hunt Adventure game** is a text-based 2D grid game where the player navigates a randomly generated map to find treasures while avoiding traps and an opponent. The opponent reduces the player's health by 30 points if encountered. The player must use the W/A/S/D keys to move around the grid. The goal of the game is to collect **3 treasures** without losing all of their health. If the player's health reaches zero, the game ends in failure. If the player collects all 3 treasures, they win. After each game, the player will have the chance to start a new game again (with a new randomized map) unless they choose to quit (press q).

Keep hunting treasures or quit whenever you choose!

🗺️ Game Map Design

The game world is represented by a **5x5 grid**. Each grid square can be one of three types:

1. **Empty Space (.)** - An empty square that does nothing when the player steps on it.
2. **Trap (X)** - A trap that reduces the player's health by 20 points.
3. **Treasure (T)** - A treasure that increases the player's treasure count by 1.
4. **Opponent (O)** - An AI player that randomly is placed on the board each turn, and reduces 30 health points if interacted with.

The player starts at the top-left corner of the map (position (0,0)) and must move through the grid to find treasures. If the player steps on a trap, their health is reduced by 20 points. When they step on a treasure, their treasure count increases by 1. When a player lands on a space with an opponent, their health is reduced by 30 points.

Important Map Generation Rule:
When generating the map, treasures must not be completely surrounded by traps. Each treasure must have at least one open adjacent path (north, south, east, or west) that is not a trap.

Recommended Numbers for a 5x5 Map:

- 5 Treasures (T)
- 5–7 Traps (X)
- 1 Opponent (O)
- Remaining spaces are Empty (.)

🕹️ Game Mechanics

- **Health**: The player starts with 100 health points. If the player's health reaches 0, the game is over and they lose.
- **Treasures**: The player needs to collect 3 treasures to win the game.
- **Opponent**: The opponent moves randomly to a different spot after every player moves. If the player encounters the opponent, they lose 30 health points.
- **Movement**: The player can move in four directions:
    - **W**: Move up (North)
    - **A**: Move left (West)
    - **S**: Move down (South)

    - **D**: Move right (East)
- **End Conditions**:
    - **Win**: The player collects all 3 treasures.
    - **Lose**: The player's health reaches 0.

📜 Game Flow

1. **Initialization**:
    - The player starts with 100 health and 0 treasures.
    - A 5x5 grid is randomly populated with . (empty spaces), O (opponent), X (traps), and T (treasures).
    - The player's starting position is always at (0,0).
2. **Player Movement:**
    - The player inputs one of the movement keys (W, A, S, D) to move around the grid.
    - After every move, the game checks the new square for a trap or treasure.
        - **Trap (X)**: Decreases health by 20 points.
        - **Treasure (T):** Increases the treasure count by 1.
        - **Opponent(O):** Decreases health by 30 points.
3. **Opponent Movement:**
    - After every player moves, the opponent moves randomly to a different location (except the player's current position and treasure/trap's position).
4. **Continuous Play:**
    - After each game (either win or lose), the player will be asked if they want to play again.
    - If the player chooses yes, a new map is generated and a new game starts.
    - If the player chooses no, the game exits.
5. **Map Visibility:**
    - The map hides traps, treasures, and the opponent at the beginning.
    - Traps are revealed permanently after being triggered.
    - Treasures disappear after being collected.

🏆 End Conditions

- **Win:** The player collects 3 treasures.
    - A "Congratulations" message is displayed and the game ends.
- **Lose:** The player's health drops to 0.
    - A "Game Over" message is displayed and the game ends.

Afterward, the player can choose whether to continue hunting for more treasures or quit.

📊 Sample Step-by-Step Game Run

(Example hidden map for understanding; not shown to the player)

```
[ . ][ . ][ T ][ . ][ . ]
[ . ][ . ][ X ][ . ][ . ]
[ . ][ . ][ . ][ T ][ . ]
[ X ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ T ]
```

Opponent position should be hidden but here it's shown for you to understand

Starting State:

```
Health: 100 | Treasures: 0/3
[ P ][ . ][ . ][ . ][ . ]
[ O ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
Move (W/A/S/D): D
```

- Move Right → Empty Space → Nothing happens.

Next Move:

```
Health: 100 | Treasures: 0/3
[ . ][ P ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ O ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
Move (W/A/S/D): D
```

- Move Right → 🎯 Treasure Found! (+1 Treasure)
- Treasure Count: 1/3

Next Move:

```
Health: 100 | Treasures: 1/3
[ . ][ . ][ P ][ O ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
Move (W/A/S/D): S
```

- Move Down →Trap → Health - 20

Next Move:

```
Health: 80 | Treasures: 1/3
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ P ][ . ][ . ]
[ . ][ . ][ O ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
Move (W/A/S/D): S
```

- Move Down → Opponent → Health - 30.

Next Move:

```
Health: 50 | Treasures: 1/3
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ X ][ . ][ . ]
[ O ][ . ][ P ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
Move (W/A/S/D): D
```

- Move Right → 🎯 Treasure Found! (+1 Treasure)
- Treasure Count: 2/3

Next Move:

```
Health: 50 | Treasures: 2/3
[ . ][ . ][ . ][ O ][ . ]
[ . ][ . ][ X ][ . ][ . ]
[ . ][ . ][ . ][ P ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
Move (W/A/S/D): S
```

- Move Down → Empty Space → Nothing happens.

Next Move:

```
Health: 50 | Treasures: 2/3
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ X ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ P ][ . ]
[ O ][ . ][ . ][ . ][ . ]
Move (W/A/S/D): S
```

- Move Down → Empty Space → Nothing happens.

Next Move:

```
Health: 50 | Treasures: 2/3
```

```
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ X ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ . ][ . ][ . ][ . ][ . ]
[ O ][ . ][ . ][ P ][ . ]
Move (W/A/S/D): D
```

- Move Right → 🎯 Treasure Found! (+1 Treasure)
- Treasure Count: 3/3

🎉 **Game End:**

```
Congratulations! You have found all the treasures!
You Win!
```

It will continue until you run out of health or collect all the treasures.

You are allowed to use any code from the lab problems/class lectures/previous PA assignments.

📊 **Key Features**

- **Dynamic Map Generation**: The grid and the map elements (empty spaces, traps, and treasures) are generated randomly at the start of each game.
- **Real-time Interaction**: The player controls the game via keyboard input (W, A, S, D) and gets immediate feedback on their actions.
- **Health and Treasure Tracking**: The player's health and treasure count are constantly updated based on the player's actions.
- **Win/Loss Conditions**: The game ends either when the player collects all treasures or their health reaches zero.

Requirements

- The *only* libraries you should use are stdlib for random, time for seeding rand, and iostream. As usual, you are welcome to include the std namespace.
- There are lots of ways that you can write this program, and you are welcome to use DynamicArray from assignment 3. **You are not allowed to use vectors from the vector library.**
- There should be at least 4 classes in your program. You may have more, but that is the bare minimum.
- At least 1 class should be an abstract base class.
- The program must have at least 1 operator overload.
    - You can use the + operator to add treasure for that specific player object. Or you can use the - operator to decrement the health from the player object. These are just some ideas, you are open to apply operator overload in other cases as well.
- Every class in the program should have an insertion stream overload function; it should be used any time you want to display the contents of the object (ie, do not use the getters to cout). For example, when you are going to display the board after each move or display the player information.
- The program should have at least 1 template function. Remember, templates are supposed to make code more generalizable- template functions are usually used when you have 2+ functions doing the same thing with different parameter data types. It can be a very simple template method and your requirement will be fulfilled.
- All objects and arrays should be dynamically allocated. Objects or variables that are not arrays should be passed by reference when the function needs a deep copy.
- Each user move (not the opponent) should display the updated boards to the user so that they can decide where to move.

Extra Credit:
Your assignment is only eligible for extra credit if all requirements of assignment 5 are met and the program more or less functions as intended (ie, if there's a weird seg fault somewhere that doesn't particularly impact game play, it's fine to pursue the extra credit.) There is a separate submission portal for PA5 and the extra credit- please submit your original PA5 without any extra credit components to the PA5 portal. Submit your augmented PA5 program to the extra credit portal.

The following features can be implemented for additional credit:

- **User Profile and History Saving (20 pts)**
  At the start of the game, the program should prompt the user to enter their name. If the user name already exists, the program must load the player's previous game history.

Otherwise, a new profile should be created. The player's progress and game history must be updated and saved after each session. There should be a file for each user with their history in the workspace.

- **Top 10 Winning Streak Leaderboard (30 pts)**
  The program must maintain a file that stores the top 10 winning streaks, including the corresponding user names.
  When the program starts, it should load the existing leaderboard. Players should have the option to view the top 10 winning streaks from the main menu.

If a player's number of wins exceeds a currently listed player (e.g., surpasses Jane's 10-win streak), the leaderboard should be updated accordingly. The player's information must be inserted in the correct rank, and lower-ranking entries should be shifted down appropriately.