Programming Assignments 3 & 4

Final Assignment

Assignment 3: Due 4/28/2024 @ 11:59 PM

Assignment 4: Due 5/15/2024 @ 11:59 PM

Late/Redo Assignment 3 deadline 5/9/2024 @ 11:59 PM

There is no late/redo deadline for assignment 4; it must be turned in by 5/15 at the absolute latest.

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

- This assignment requires you to write technical documentation. I provide a rough outline and description of headers and what goes beneath them, but you are welcome to do research and expand your work into a full design document if you would like. Make sure you remove instructions from the document before submission.
- UML diagrams are required as part of this assignment. You can break them up if you feel they are too large to understand, but please do so in a way that the reader can follow and piece back together into the larger picture. When I create UML diagrams, I usually use draw.io, which is a software that is freely available online. You are also welcome to use a series of tables in powerpoint/keynote/etc, or a drawing software of your choosing. I'm not picky about anything other than the actual design- does it show all of the classes in your system, their relationships, and provide a general overview that uses the right formatting and symbols?
- This is a **formal** design document. I expect you to use full words (ie, do not use u instead of you). Your grammar and spelling should be checked; a few mistakes are inevitable, but you will be graded with grammar and spelling as a portion of the rubric. If you need extra help with grammar and spelling, there are a variety of third party apps you can use, **in addition to** using the writing center (which you pay for as part of your student academic success fee- seriously, use the writing center if you need it). Your writing style should be objective and technical, and it should be clear and concise. You will notice that the portion of the design document that I supplied uses formal language and uses passive voice- please do the same in your writing.

- Your document should be clear enough that you could hand it to any other student who has taken CS202, and they would be able to implement the code described by you.
- For these assignments, **you may work with a partner**. It is not required, and groups of 3 are not allowed. Please keep in mind that the amount of work is the same for this assignment and the next, regardless of if you have a partner or not.
  - If you work with a partner, I would recommend creating a private github repo so that the two of you can share work. I would recommend each of you having your own branch to make commits to; more on that can be found here.
    - If you are interested in pair programming at the same time, github codespaces is available freely to students.
- You will need to use stdlib.h to randomly generate AI selections. You can read more about it here. You can also include time.h so that you can seed the rand function.

Assignment

In this assignment, you will write a design document (PA 3) and the code that the design document describes (PA 4). The design document will be your roadmap when writing your code in assignment 4, so please take this part seriously or you *will* struggle with assignment 4.

For this assignment, you must write the design documentation and the code necessary to create a program that allows a single user to play Battleship against an AI until the user chooses to quit. In other words, after each game, the user should be asked if they want to play again, and if they do, a new round should be started. I provide the requirements of the program below, but the program's formatting and design choices are entirely up to you.

Battle Ship Notes

If you don't know anything about Battleship, that's fine. I would begin by playing a round of battleship online, so that you can get a feel for how it behaves. Additional rules and instructions can be found here.

Programming Problem

Write a program that allows a player to play game of Battleship. If you haven't ever played before, see the section above. I recommend playing through a few times to get a feel for how it operates, if you've never played.Here's what you need to know about the game:

1. Each player has two boards- even though you'll be playing against the machine, you should still have two boards for the AI (in addition to the 2 boards for the user). I'll refer to them as the player board and the opponent board. Each board is sectioned into squares with 10 columns and 10 rows, forming a square board. Individual squares are identified by letter and number, where columns are letters, and rows are numbers.
2. The player board is where the user (or the AI) places their ships and records any shots taken by the opponent. You will need to select a character for hit and miss, such as X and O. Then, after each turn, the board should be updated with any shots taken by the opposition showing those places where the opponent has taken a shot. This allows you to track when ships are sunk.
    1. There are 5 boats that should be placed on the board. They are:
        1. Carrier- takes 5 squares
        2. Battleship- takes 4 squares
        3. Destroyer- takes 3 squares
        4. Submarine- takes 3 squares
        5. Patrol Boat- takes 2 squares
    2. All boats must be placed on contiguous squares, and they cannot overlap.
    3. At the beginning of the game, the AI should randomly select (using stdlib.h) where to place each of the five boats, and it should be stored in the AIs player board.
    4. At the beginning of the game, the user should be allowed to select the coordinates where they would like to place their five ships, and it should be stored in the user's player board.
    5. The simplest way to track the location of the five ships for AI and player is probably to have each Boat object contain a start index and an end index. Since we know that boats must be contiguously allocated, this will provide us with the full location of the boat.
3. The opponent board is where the user (or the AI) records previous shots taken by the player on the opponent's board, so that the same coordinate is never called more than once when attacking.
    1. At the start of the game, all squares in this board will be blank for AI and user.
4. The game should randomly choose who to start with, the user or AI. That person will take the first shot.
5. After the ships are positioned and the player is selected randomly, the game will occur in a series of rounds.
    1. At the start of each player turn, the player guesses a coordinate where they think the opponent's ship is. If the coordinate has already been guessed, then the user should be prompted to enter another coordinate until they select one that hasn't been called by them.
        1. Guesses for the AI can be generated randomly- just make sure it doesn't choose the same coordinate more than once.

        2. Guesses from the user should be taken as input. The format you do that in is up to you.
    2. The opponent announces whether or not the square is occupied by a ship.
        1. If yes, this is a hit. The game should mark the player's opponent board with a symbol (like X) to indicate that there was a hit in that coordinate on the board.
        2. If no, this is a miss. The game should mark the player's opponent board with a symbol (like O) to indicate that there was a miss in that coordinate on the board.
        3. The game should mark the opponent's player board with the same symbol as was placed on the player's opponent board, so that they can track if a ship has sunk. Ships are considered sunk when all of the coordinates containing a ship have been hit.
            1. If a ship has sunk, the game should tell the user that the ship has sunk. For example You sunk my destroyer! might appear after you hit all three coordinates containing a destroyer on the opponent's board. Regardless of the actual message output at sink, the game should include the name of the ship type so the user knows which ships remain.
        4. The opponent's turn should now take place, following steps 1-3.
6. The objective of the game is to sink all of the opponent's ships by hitting all of the squares on the board containing the ships. The first player to do so wins.

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 but **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.
    - If you are stuck on where to place the overload and how to use it, I would suggest overloading the + operator so that when it acts on a coordinate, an x is added to signify a hit.
- 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).
- 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.

- 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 turn (not AI) should display the updated boards to the user so that they can decide where to attack. The way that you represent a board, its squares, and its hits and misses is up to you.

Programming assignment 3: Due 4/28/2024 by 11:59 PM

Your design document must follow the same format as the provided template. When you are finished, save your work as a pdf and submit it via webcampus as Assignment 3. If you are working with a partner, please make sure both names are on the assignment- only one of you needs to submit. We will give you feedback, and you are welcome to refactor the document and resubmit for credit by 12/12/2023.

Programming assignment 4: Due 5/15/2024 by 11:59 PM

All .cpp, .h, and makefiles necessary to run the program should be compressed into a zip file. Your executable should be named *battleship*. Submit the zip file as Assignment 4. Please note that **assignment 4 has no late submission deadline**- this is due to the tight turnaround between the assignment and end of the semester. You are still more than welcome to go to office hours and discuss ways to improve your program if you feel like you need additional guidance or are worried that the program isn't finished. There will not be an end to end grading script for this assignment; instead, we will manually play through the game and assign points using the assignment 4 rubric.

Extra Credit: Due 12/12/2023

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 PA4 and the extra credit- please submit your original PA4 without any extra credit components to the PA4 portal. Submit your augmented PA4 program to the extra credit portal.

Each assignment up until now has been worth 100 points, for a total of 400 points in the assignments category. Completing the extra credit in this section will allow you to increase your overall assignment category points by up to 50 points (a 12.5% bump), up to the 400 points available in the category. In other words, the extra credit will not allow you to exceed 100% in the assignments category.

Below, I list different functionality to add to the program (in addition to the requirements above). The extra credit points are listed next to the functionality. You may choose to do only some, or all of them.

1. [15 points] Implement a way to track player wins and losses, like an arcade machine. A file containing the top 10 winning streaks and the name of the user who won should be saved and loaded when the program is started. For example, sarad 10 wins. The user, in addition to being able to play the game, should be allowed to choose to look at the top 10 winning streaks. The user should be prompted for a username at the start of the game- the username does not need to be unique. Each time the program is run, that player should start with 0 wins, and each time they win, that should be incremented. If the number of wins is greater than anyone on the winners board, then the player's username and win streak should be recorded as part of the wins file.

2. [25 points] Implement a 2 player option. The program described above only allows for 1 player to compete against a simple AI. Instead, modify the program so that the user can still play against AI OR they can choose to play against someone else. The turn order in that case will remain the same, but rather than randomly generating and selecting, the second player should be prompted for input on where to place ships and where to shoot their shot.

3. [10 points] Modify the AI so that it makes intelligent choices. In other words, if there is a hit somewhere on the board but the boat hasn't been sunk yet, the AI should try to choose a coordinate touching the location where the boat was hit instead of randomly choosing a spot.

As part of the submission of the extra credit, include a file named *augmentations.txt*. Inside, list which of the extra credit options you implemented. **Failure to submit this txt file will result in the extra credit not being graded.**

Academic Honesty

Please be aware that we will be using both a plagiarism detector and AI detector on your work-students found to be in violation of the University's academic honesty policy will be referred to the office of student conduct.

Academic dishonesty is against university as well as the system community standards.

Academic dishonesty includes, but is not limited to, the following:

Plagiarism: defined as submitting the language, ideas, thoughts or work of another as one's own; or assisting in the act of plagiarism by allowing one's work to be used in this fashion.

Cheating: defined as (1) obtaining or providing unauthorized information during an examination through verbal, visual or unauthorized use of books, notes, text and other materials; (2) obtaining or providing information concerning all or part of an examination prior to that examination; (3) taking an examination for another student, or arranging for another person to take an exam in one's place; (4) altering or changing test answers after submittal for grading, grades after grades have been awarded, or other academic records once these are official.

Cheating, plagiarism or otherwise obtaining grades under false pretenses constitute academic dishonesty according to the code of this university. Academic dishonesty will not be tolerated and penalties can include cancelling a student’s enrollment without a grade, giving an F for the course, or for the assignment. For more details, see the University of Nevada, Reno General Catalog.