COP Term Project: Analysis and Design

Definition

* The proposed system definition is a simulation of the classic board game ‘Battleship’, where two players compete against each other by trying to sink enemy ships through guessing their positions based off of a 10x10 grid.
  + Five ships, with varying lengths of 2 – 5 are placed in random areas by the players before game play, where no ship can overlap the other and the ships cannot cross the 10x10 boundary.
  + Players take turned guessing the locations of enemy ships and either hit, miss, or sink (if the last hit corresponds to the final empty slot of the ship) the enemy boats.
  + A player wins the game if they manage to sink all of their opponent’s ships.
* The system will provide the user with an option to either play with another person or against a computer, and will end when either the person quits, wins, or loses against the opponent.
* The planned program is important because it will allow the team members of the group to practice essential programming techniques, using data structures, managing memory, error catching, object orientation, and other fundamentals, while also learning new skills in graphical display, computer AI, and teamwork. As for significance to a larger extent, this program will be able to benefit many who wish to play the game but do not have access to the physical version of the board game; in addition this strategic game is fun to play and stretches the brain.

Analysis

* Inputs
  + On the start-up of the program, the user will be asked to choose whether or not to play against another person, or against a computer. Once the user makes that decision, they are brought to another screen showing the Battleship board and the game will begin.
  + The user will need to input their decisions for the placement of their ships by clicking the mouse on the desired positions, and if they are valid choices, the game will start by asking the user to guess where the enemy ships are located, again by mouse click.
    - The locations and statuses of the users’ ships will be stored in a map-like data structure, along with the computers’.
  + Then, the computer will take their turn to guess the position of the user’s boats, recording the corresponding hit, miss, or sink. The turns continue to alternate until either all boats are sunk by one player or the user decides to quit.
  + The format for the ship data:
  + The format for the grid data:
* Outputs
  + The program will generate outputs of a hit, miss, or sink on either the user’s ship or the computer’s ship. This will be shown graphically on the Battleship board at each turn.
  + Only the Battleship screen of the user will be shown on the program, in true simulation of the game; though data from the ships and grids are collected for both the user and the computer, the user will only be able to see the grid where their ships and their guesses are visible. In this way, only a subset of all the data is shown to the user, as it is a game where the opponent’s data is not known.
* Constraints
  + There will be a time-limit constraint to enable the game play to flow. If the user does not make a decision in the time frame, it will automatically switch turns and the computer will then make a turn.
  + Ship lengths cannot change, along with ship location after initial placement.
* Assumptions
  + On the part of the developers, the AI is assumed to try and win the game, along with the user. In this sense, its performance will try to optimize the hits and sinks of enemy ships.
  + The general rules of Battleship are followed and implemented in this simulation.
* Modifications
  + When a ship is either hit or sunk, their metadata will need to be changed along with the grid data. This will need to happen automatically when the user or the computer guesses correctly on a given turn.
* Flow & Logic
  + During the course of the game, the data between the ships and the grid themselves need to be aligned. As a user or computer guesses a location on the grid, the grid will need to update either a hit or miss, and the ship data will then need to be updated as well depending on the location of the hit; sinks will also need to be treated differently.
  + Both the ships and the grid will act as objects that interact with each other
  + If there is a sink, the data structure holding all of the ships will need to delete the ship from the container (a player wins when there are no more ships on the opponent’s side).
* Use Cases
  + As there are no discriminations between users playing Battleship, unless there is a difficulty component (which may or may not be implemented due to the time constraint of the project), there will be no user cases and no different types of access on the part of users to perform the tasks of the game.

Design

* Control, UI
  + The user interface will consist of the welcome screen leading to the Battleship screen. The latter will update turn-by-turn as the game progresses, until program ends when either the user or computer wins the game.
  + The game screen will have the intention of emulating almost all of the design elements that the original board game possesses. However, there will be neat features that will hopefully be able to be implemented as the project progresses: in choosing the ship placements, i.e. there will be a shadow image of the ship that is being placed on the UI so that the user can see exactly where on the grid the ship will be placed.
  + In controlling the data, many cases have to be considered (when a user tries to choose an invalid ship placement, they have guessed the position already, etc.).
* Data
  + Ships: aircraft carrier (length 5), battleship (length 4), submarine (length 3), destroyer (length 3), patrol boat (length 2)
    - Attributes: length, status of every position in that length (hit or not hit), sink or not sink
    - Methods:
  + Grids: both the one visible to the user, and also the hidden one known only to the computer
    - Attributes: each position has an associated status (hit, miss, sink, empty, not yet hit)
    - Methods:
      * Insert ship coordinates based on ship attributes
      * Insert ship coordinates based on ship attributes
      * Record hit/miss and update ship object
      * Record sinking ship and update (delete) ship object
* Algorithms
  + Algorithm to represent the choices of the computer player
  + Algorithm to represent the data of the grid and change the coordinates of the grid into Battleship coordinates
  + Algorithm to hit ships, miss ships, sink ships, and update the objects accordingly
* UML

|  |
| --- |
| Grid |
| - int[] grid |
| + void displayGrid()  + void convertCoordinates()  + void hitShip()  + void missShip()  + void sinkShip() |

|  |
| --- |
| Ship |
| - int length  - int[] status |
| + void getLength() |

* The required number of modules will be:
* For each module, the classes and methods are:
* The shared classes and methods across all modules are:

Execution Plan

* The division of the coding tasks will mostly be based on procedural aspects of the program: data structure of the grid, data structure of the ship, graphical aspect of the program, AI component, methods to record a hit or miss, method to record a sink, methods to check error cases (of user ship placement, etc.).
* The makefile will:
* In order to complete the implementation/testing on each module of the project, the team proposes to work separately and then group together to merge work once every week; the deadlines for testing will be moved back a week to account for any problems in the final implementation, and several smaller deadlines will be set in order to finish major portions of the program (namely, the grid and ship design, the AI component, etc.).