TanofWongSquared APCS1 pd 8

Larry Wong, Richard Wong, Stefan Tan 2018--1--3

Final Project Proposal

Battleship Warfare

Our final project proposal is a modified version of the game Battleship. It includes the same setup and gameplay as battleship to battleship as the board game with some variations.

Basic Setups and Rules:

1. Each player receives a 10x10 grid to place their vessels
2. There are going to be 5 different type of vessels (each player receives 1 of each):
    a. Aircraft Carrier
    b. Battleship
    c. Destroyer
    d. Submarine
    e. Cruiser
3. The player will play the Salvo variation of the game
    a.

sal·vo
/ˈsalˌvō/

noun

a simultaneous discharge of artillery or other guns in a battle.
- a number of weapons released from one or more aircraft in quick succession.
- a sudden, vigorous, or aggressive act or series of acts.
"the pardons provoked **a salvo of** accusations"

    b. Each player fires a number of shots equal to their number of ships
4. Each player will have a “markup” grid to place shot locations (hit or miss)
5. The game is completed once all ships on one player’s grid have all sunk

Where the Wild Things Are:

There will a series of changes to spice up war (each item corresponds to item in previous list):

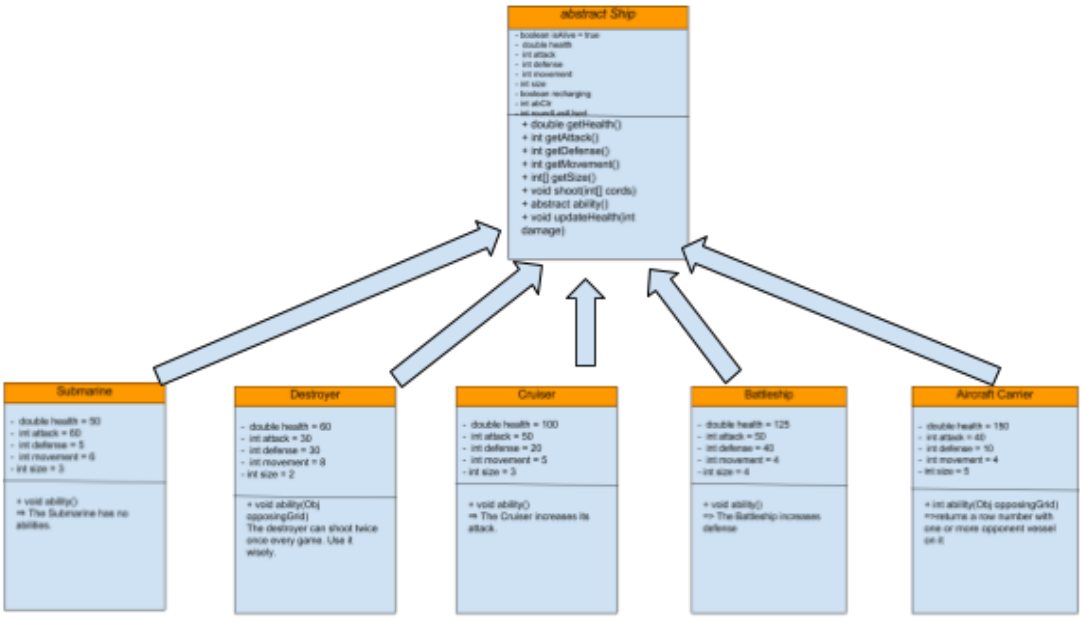
1. Instead of a default 10x10 grid, each player is going to receive a 20x20 grid
    a. Encourages deeper strategies and addiction
2. Each of the vessels are going to receive attributes that will define them:
    a. Variables:
        i. Size - The number of cells a ship takes (width and length)
        ii. Defense - Reduces the damage taken if ship is hit
        iii. Health - How many times a ship can be hit before it sinks
        iv. Movement - How many cells a ship can move when allowed to (discusses further in a later point)
        v. Attack - How much a ship's shot will do if it hits
    b. After a certain number of rounds, each player will be allowed to move their vessels
        i. Distance moved depends on Movement variable
        ii. Use Distance formula ([r][c] == (x,y))
    c. Each ship will contain a special ability
        i. Attack boost, Defense boost, radar, etc…
        ii. Limited by a number per game
        iii. Cannot be used consecutively and prevents ship from shooting their volley
3. Player can permanently make a cell of the grid uninhabitable
    a. Ships will no longer move there

    b. After some rounds, cells with most number of shots there will be unusable
        i. If a shot hits a ship, it will not consider this in this aspect
4. Nothing will update on the :markup" grid, user has free reign
    a. Use different symbols to classify hit or miss
    b. Keeps track of number of shots that landed in a certain area
5. Game is considered done once a player loses all their vessels or grid is no longer habitable

To-Do List:

- Create the following classes:
    - PlayerGrid
        - Set a name
        - Holds 2 2D arrays of size 20x20 (board and markup)
        - Methods to place/move ships and its size takes up that number of spaces in array
        - Methods to edit markup grid
        - Print out both grids
        - Take input(shots) coordinates from opposing PlayerGrid into an 2D array and apply them to itself
            - Hitting ships or damaging playing board
        - Methods to determine uninhabitable space
    - Superclass Ship
        - Subclasses that correspond to the 5 types of ships

    - Create a abstract superclass
        - Accessors
        - Method to shoot, sends info to other grid
        - Abstract method for special ability
        - Checks if it can use ability again
    - Each subclass set variables and has a special ability
- Configure Woo.java to run the game with classes
    - Keeps track of rounds for when it is time to restrict the field or to limit special ability use



Hey, What is that Object Heading Towards Us?

Rough Timeline:

1. Implement PlayerGrid and ensure its functionality (1/4/18 to 1/7/18)
2. Implement class Ship and also working on either an interface or inheritance with abstract methods. (1/7/18 to 1/9/18)
3. Working on the game mechanics and the driver class and also optimize the user interface. (1/9/18 to 1/14/18)
4. Testing and fixing any problems that we may not have noticed before. (1/15/18 to 1/16/18).

We are making meaningful use of OOP by taking advantage of polymorphism in OOP. Instead of writing multiple methods that do the same thing, we could just implement all the shared methods in the parent class while the unique attributes could be implemented in the subclasses. This could be achieved using inheritance, abstract methods, or an interface.