Bayside Basics
Jason Kim, Derek Song

APCS1 pd.1
2018-01-04

Final Project Proposal

Battleship: Human vs Computer

For our final project, we will be attempting to program the popular board game Battleship. Now of course this is a 2 player game, but there is only one screen. Because of this, we think it would be better to attempt to program an AI to be the second player. There will also be 2 different difficulty levels as well.

The setup is just as important. We plan on placing the ships using a "head-direction" method. Basically, the player would indicate the coordinate in which the head of the ship will lie. After that, the player will be asked if he wants the ship to be vertical or horizontal. There will be a diagram to better picture this in the actual program. Throughout the game, we will make the experience as simple as possible by clearly marking everything on our interface.

To-do List

1) Make the UML diagrams for all classes.
2) Do some sketches and visualize how everything should interact.
3) Player class first!
4) Beginner AI class.
5) Advanced AI class.
6) Refinement. Make sure the game is fair but is decently challenging.

Timeline

- By the 6th, have the UML diagrams finished and have a sufficient idea of how everything is going to connect (using sketches).
- By the 8th, have a working Player class and Beginner AI class.
- By the 15th, have a working Advanced AI class.
- By the 16th, have a refined program.

OOP

Overall, Battleship does not require the use of many classes/objects. Instead, there are few object with tons of functionality within them.

- Superclass: Player. Anything playing the game, whether it be a human or computer will be defined under this class.
    - Subclass: Human. Humans will need a few different methods as they are the primary specimen that this program is being built for. The main difference is that a human should be alerted of what ship they have hit if they did hit at all.

    - Subclass: Computer. Computers don't need as much information as Humans do. The main difference we saw was mentioned above already. Also, in general, it's nice to separate the classes this way as if any other oversights were to be realized, it's a lot simpler to implement them.
        - Subclass: Beginner. They hit randomly!
        - Subclass: Advanced. They hit using a strategy that is later explained.

Concepts

- The main concept we are working with here is the 2D Array. The Battleship grids are 2 dimensional and all gameplay occurs on those grids.
- Another concept we want to work with is try and catch. Out of index errors may occur with user input so to make the experience better, we will try to implement a catch.
- We also need to work with boolean expression and conditionals a lot through this as well. As we are trying to develop an AI, we have to toggle its actions and determine when it should do what.

Advanced AI Implementation

- Checkerboard pattern. By going in a checkerboard pattern, you are maximizing efficiency as there are no boats in the game that are shorter than 2 spaces.
- Row check. The AI will start at a row, attempt to hit a few times and if all fails, it will choose a new row to try and attack.
- Center. Generally, the center grid (around 5 by 5) is a hotspot as parts of ships often linger in that area. Starting in this area is generally good.
- Logic! If the AI only has a 5-space ship left, it wouldn't be logical to attack an area that only has a 1 space radius of non-attacked spaces around it. We have to find a way to avoid this.