CIS-17A - Project 2

Battleship CLI Game v1

Alexander Carter

Overview

Project Description

For the first computer science project of this semester, I chose to go after a classic and memorable game of chance, Battleship. Battleship is a strategy game consisting of two players, each with their own grid, often 10x10, in which they can randomly place their fleet of ships at any location. These ships vary in type and size, but often follow this pattern:

1. 1, 5 Wide Ship
2. 2, 4 Wide Ships
3. 3, 3 Wide Ships
4. 4, 2 Wide Ships

On each turn, the player can choose a coordinate of the grid to strike on their oppoinents board. If the coordinate is one of a ship, the other player reports “hit” and the player may take another turn, else the other player reports “miss” and the turn switches. Once all points of a ship are hit, the ship is considered sunk. First player to sink all the opponent’s ships win.

Desired User Experience

1. The user loads up the game in a console (such as Window Command Line, Bash, or their IDE of choice).
2. They are asked for their name and desired board size
3. They are shown a randomly selected board layout. They may choose to regenerate the board layout as many times as they wish until they are satisfied. (This is compensate for lack of free-choice in the board setup phase, as developing a simple interface for this with in CLI would be out of the budget for this project)

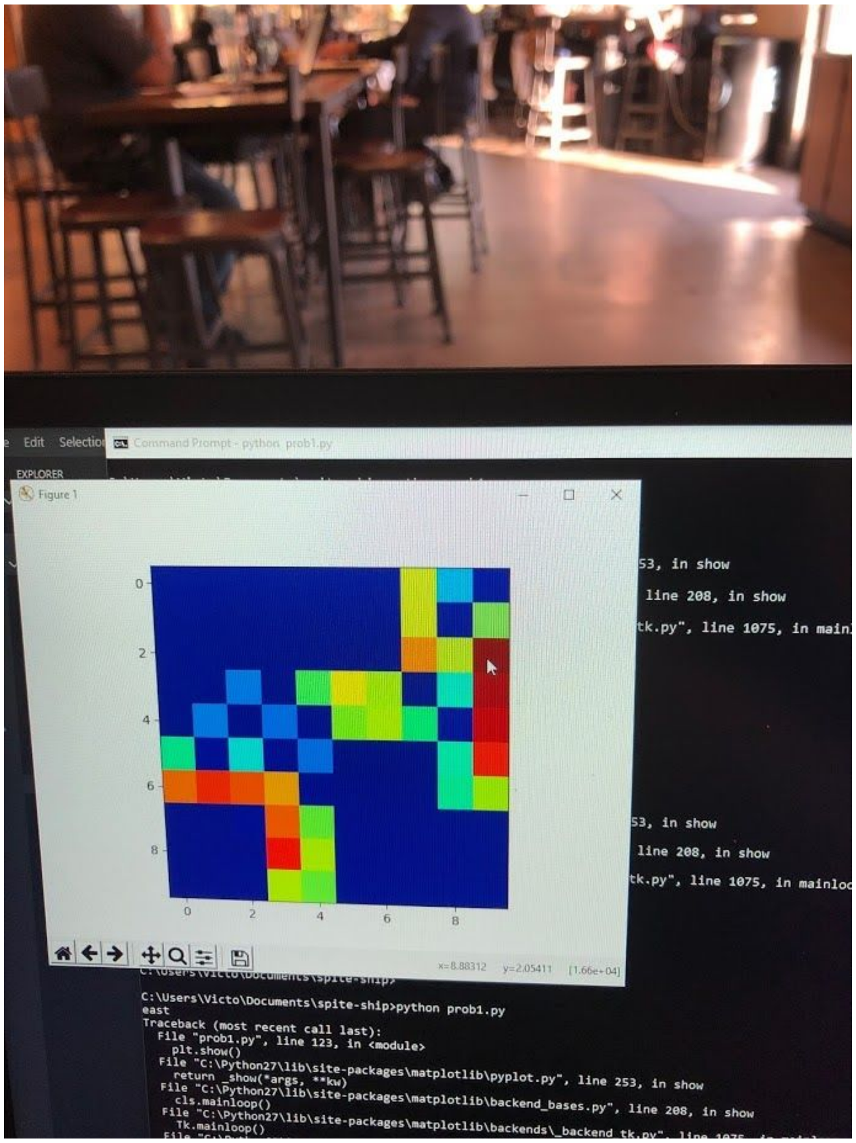
Concepts

Inspiration

Battleship the Game & Probabilistic AI Project

Battleship has always been a fun and unique game to replicate due to its requirement to track multiple states, and how its inherited a probability optimization problem. The winner is often the player who has the ability to narrow down possible coordinates for a ship to be in.

The game of battleship has been adapted to multiple platforms, such as iMessage. This port of the game, and a conversation with a friend about how the game is probability based, lead me to design and develop a simple probability calculator to generate a heat-map of possible ship locations given a field state. (Photos below). This application formed the basis of this one, with much of the game logic and ship placement rules being prototyped.


Edit
Selection
Command Prompt - python prob1.py
EXPLORER
Figure 1
0
2
4
6
8
x=8.88312 y=2.05411 [1.66e+04]

```
53, in show
 line 208, in show
tk.py", line 1075, in main

53, in show
 line 208, in show
tk.py", line 1075, in mainlo

C:\Users\Victo\Documents\spite-ship>python prob1.py
east
Traceback (most recent call last):
  File "prob1.py", line 123, in <module>
    plt.show()
  File "C:\Python27\lib\site-packages\matplotlib\pyplot.py", line 253, in show
    return _show(*args, **kw)
  File "C:\Python27\lib\site-packages\matplotlib\backend_bases.py", line 208, in show
    cls.mainloop()
  File "C:\Python27\lib\site-packages\matplotlib\backends\_backend_tk.py", line 1075, in mainlo
    Tk.mainloop()
```

Technology

Data Storage of Game State

InvalidInput Exception

InvalidInput custom exception allows for parsing user input and throwing an exception if user input is incorrect. Catching this exception allows for user feedback.

Vector2

This custom class aids in common vector math. This class stores a custom type (template) x and y value. It also provides helper operator overloads to allow for easy comparisons and setting. In addition, it provides common functions such as clamping a value, allowing for easier and less code when doing operations on the board and sanitizing inputs.

Grid System

The application stores the game board in two individual 2D Integer arrays of a 10x10 size. Each tile of this array describes what belongs in this grid square via an int, usually corresponding to the Tile enum, however this is now always the case as each grid slightly alters it's data storage based on its requirements.

The first of these arrays is the player's main board. This board is responsible for storing all their ship placements. In this case, each value of a tile corresponds to a ships' ID. This allows for easy referencing when querying a tile. This ID can easily be converted to the ship's type for rendering.

The 2nd grid stores the hits and misses for the player, consider this the top board in the real life game. The only values stored in here are the enums EMPTY, MISS, and HIT and is mostly used to help the player track their previous choices and is not necessary for base gameplay logic.

Grid ID System

The application stores all ships, hits, and empty space as integers to allow for easy storage in a 15x15 2D array of integers. Each int type expands to:

- 0 - Empty
- 1 - Aircraft Carrier Tile

- 2 - Battleship Tile
- 3 - Destroyer Tile
- 4 - Cruiser Tile
- 5 - Hit Tile
- 6 - Miss Tile

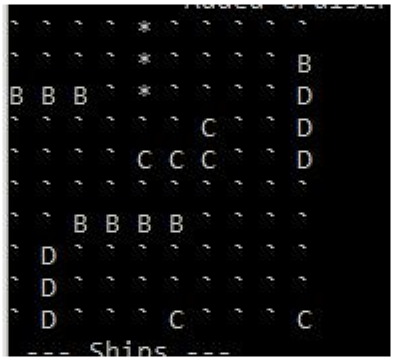
Each of these values have been made into an enum “Tile” to allow for easy association in code. The enums follow:

```
enum Tile{
    EMPTY=0,
    AIRCRAFT_CARRIER=1,
    BATTLESHIP=2,
    DESTROYER=3,
    CRUISER=4,
    HIT=5,
    MISS=6
};
```

Display

Game Board

Each game board can be rendered using the printBoard function, however how each board is rendered can be slightly different. Most boards are rendered by converting each TILE value into a character. For example a EMPTY tile is rendered as “ ` “ while a BATTLESHIP is rendered as “B”

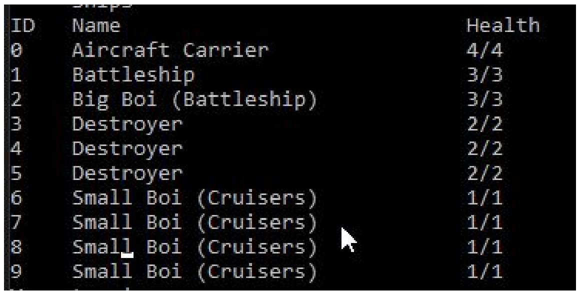


```
` ` ` ` * ` ` ` ` `
` ` ` ` * ` ` ` ` B
B B B ` * ` ` ` ` D
` ` ` ` ` ` C ` ` D
` ` ` ` C C C ` ` D
` ` ` ` ` ` ` ` ` `
` ` B B B B ` ` ` `
` D ` ` ` ` ` ` ` `
` D ` ` ` ` ` ` ` `
` D ` ` ` C ` ` ` C
  --- Ships ---
```

Ship rendering is first started by converting the grid ID to a ship TILE type to allow for this conversion. Ship IDs start at 10+ to avoid confusion with standard tile types.

Ship Status

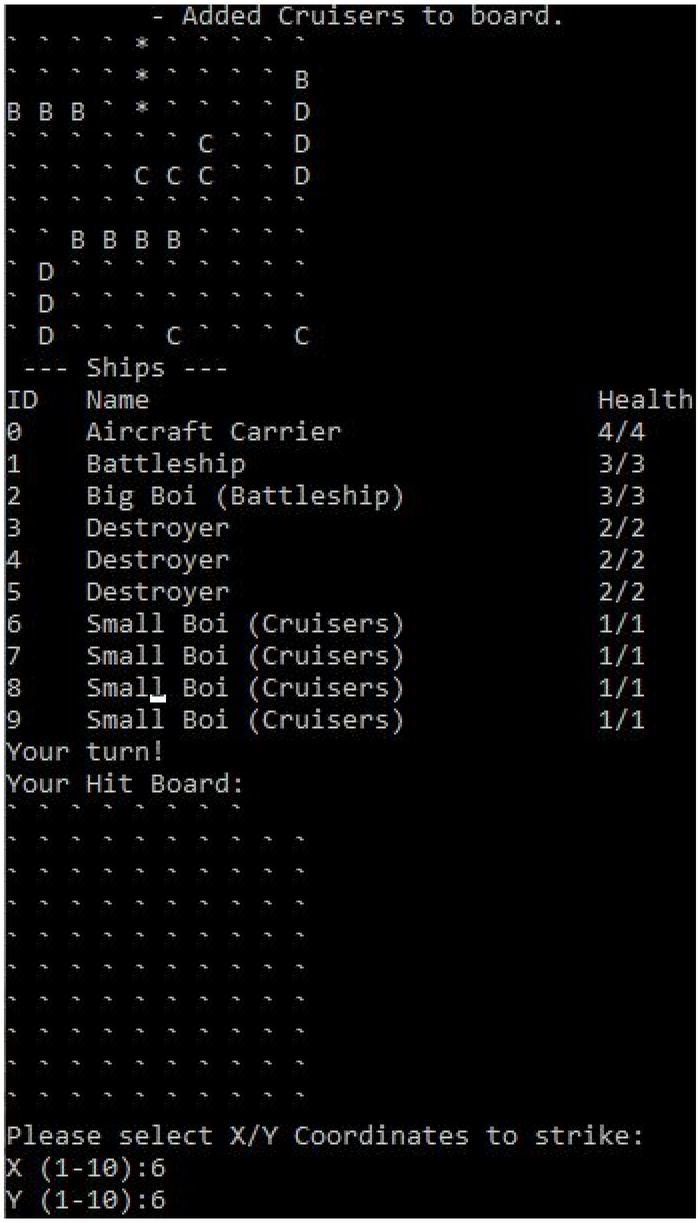
Ship status list is a quick and easy way to display each of a player’s ship status. This simply prints each ship, its name, and its hits/health.



```
ID   Name                            Health
0    Aircraft Carrier                4/4
1    Battleship                      3/3
2    Big Boi (Battleship)            3/3
3    Destroyer                       2/2
4    Destroyer                       2/2
5    Destroyer                       2/2
6    Small Boi (Cruisers)            1/1
7    Small Boi (Cruisers)            1/1
8    Small Boi (Cruisers)            1/1
9    Small Boi (Cruisers)            1/1
```

Documentation and Images

Main Game Display:



```
         - Added Cruisers to board.
` ` ` ` * ` ` ` ` `
` ` ` ` * ` ` ` ` B
B B B ` * ` ` ` ` D
` ` ` ` ` ` C ` ` D
` ` ` ` C C C ` ` D
` ` ` ` ` ` ` ` ` `
` ` B B B B ` ` ` `
` D ` ` ` ` ` ` ` `
` D ` ` ` ` ` ` ` `
` D ` ` ` C ` ` ` C
 --- Ships ---
ID   Name                            Health
0    Aircraft Carrier                4/4
1    Battleship                      3/3
2    Big Boi (Battleship)            3/3
3    Destroyer                       2/2
4    Destroyer                       2/2
5    Destroyer                       2/2
6    Small Boi (Cruisers)            1/1
7    Small Boi (Cruisers)            1/1
8    Small Boi (Cruisers)            1/1
9    Small Boi (Cruisers)            1/1
Your turn!
Your Hit Board:
` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `
` ` ` ` ` ` ` ` ` `

Please select X/Y Coordinates to strike:
X (1-10):6
Y (1-10):6
```

Reflection

Improvements and Plans

- Add TCP based multiplayer between two CLI instances
- Proper rendering, either via OpenGL or Websockets and JavaScript
- Probabilistic AI from the original prototype.
- Custom Ship Placing
- New Save File System to support classes
- Enter player name

Shortfalls

- Player can’t choose exactly where to place ships
- The CLI interface can be difficult to understand at a glance