2023-02-08

Unit 8 Project

This week and next

Wed 2/8: Unit 8 Project

Fri 2/10: Unit 8 Project

Mon 2/13: Unit 8 Review + more time on Unit 8 Project

Wed 2/15: Unit 8 Test. **Unit 8 Project DUE Wed morning**

Fri 2/17: NO CLASS

Mon 2/20: NO CLASS


BATTLESHIP
1 2 3 4 5 6 7 8 9 10
A
B
C
D
E
F
G
H
I
J

```
YOUR INSTRUCTIONS:
This is a functioning implementation of the Hasbro classic board game, Battleship.
You can consult the official rules for Battleship here:
https://www.hasbro.com/common/instruct/battleship.pdf

This is a one-player version of the game where the user plays against the computer.
However, the computer opponent is very simplistic... it just fires at random.
The game also doesn't let the player layout the ships exactly as they want them.
You'll be fixing this and making other enhancements.
```

```
Where shall we fire, captain? E9
PLAYER ATTACKED OPPONENT AT E9
  HIT: SUBMARINE
  OPPONENT'S SUBMARINE HAS BEEN DESTROYED!
OPPONENT ATTACKED PLAYER AT A6
  MISS
```

	1	2	3	4	5	6	7	8	9	10
A				–						
B		–		A*					–	
C			–	A*					S*	–
D			–	A*	–				S*	
E				A*	–			–	S*	
F			B*	A*						
G			B*		–	C*	–			
H			B*	–	D*	C*			–	
I			B*	–	D*	C*		–		
J			–		–					

YOUR OPPONENT

  [*] Hit
  [–] Miss

[?][A] Aircraft Carrier (5)
[?][B] Battleship (4)
[?][C] Cruiser (3)
[?][S] Submarine (3)
[?][D] Destroyer (2)

[?]DESTROYED

	1	2	3	4	5	6	7	8	9	10
A					–	–				–
B	–				C*	–		D*	–	
C		–	S		C*	–	–	D*		
D			S		C*	–	–	–	–	
E			S	–	–	A*	–		–	–
F						A*				
G						A*				
H	B	B	B	B		A*				
I		–			–	A*	–	–		–
J									–	

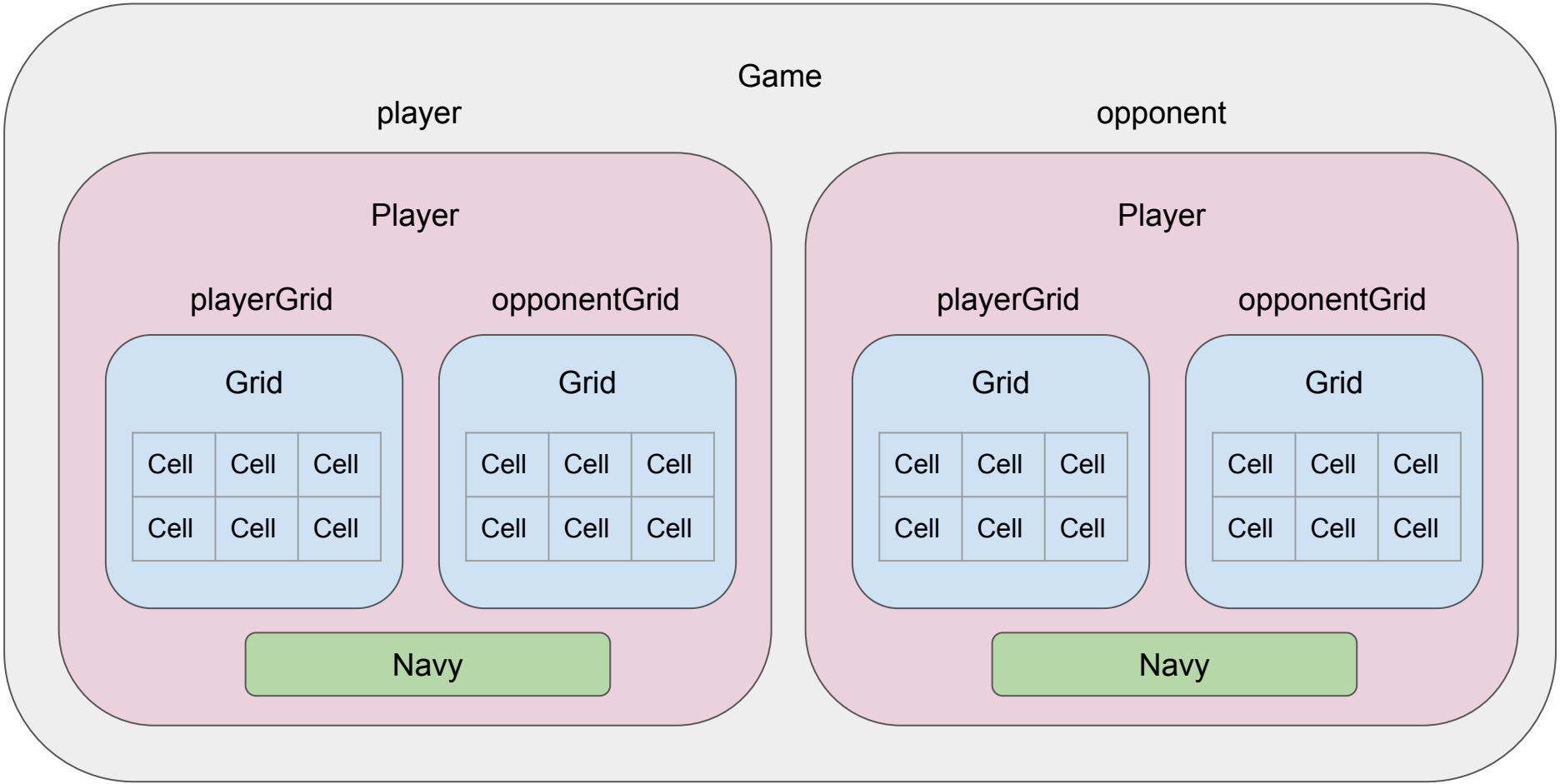
YOUR NAVY

  [*] Hit
  [–] Miss

[?][A] Aircraft Carrier (5)
  [B] Battleship (4)
[?][C] Cruiser (3)
  [S] Submarine (3)
[?][D] Destroyer (2)

[?]DESTROYED

```
ALL YOUR OPPONENT'S SHIPS ARE DESTROYED – YOU WIN
Aircraft Carrier (B4) (C4) (D4) (E4) (F4)
Battleship (F3) (G3) (H3) (I3)
Cruiser (G6) (H6) (I6)
Submarine (C9) (D9) (E9)
Destroyer (H5) (I5)
ggrossman@PK61V7VQ7Q battleship %
```


Game
player
opponent
Player
playerGrid
opponentGrid
Grid
Cell
Cell
Cell
Cell
Cell
Cell
Grid
Cell
Cell
Cell
Cell
Cell
Cell
Navy
Player
playerGrid
opponentGrid
Grid
Cell
Cell
Cell
Cell
Cell
Cell
Grid
Cell
Cell
Cell
Cell
Cell
Cell
Navy

1. The game exits after you win or lose. Change it so the game asks whether you'd like to play again. If the player says yes, the game should restart from a blank state. (If you're writing a lot of code to reset the game, consider other options... this should only take a few lines!) **(10 points)**

2. Add a cheater "peek" command that lets you see the positions of your opponent's ships. **(10 points)**

3. Make it so pressing Return instead of entering a valid move makes a random move. **(10 points)**

4. The player's ships are laid out randomly right now. Ask the user when the game starts whether they want random or manual layout. If they choose manual layout, ask them for the starting position and whether to lay out horizontal/vertical for each ship in the navy. Print an error and ask again if they choose an impossible position (goes off edge of grid or conflicts with another ship.) **(30 points)**

5. Implement the "AI" for choosing the computer's next move by implementing the selectTarget() method of class Player. This does not have to be perfect, but it should do a better job than firing randomly.

At a minimum, if the computer has scored a hit and the ship isn't sunk yet, the computer should fire on a not-yet-fired-upon location adjacent to such a hit. Full points will be given for that. See the two pseudocode examples at the bottom of these instructions.

A sophisticated implementation will be able to see that a series of hits are horizontal or vertical, and know to fire upon the next adjacent horizontal or vertical space. **(30 points)**

6. The AI for choosing the computer's move should be able to function as a hint mode for the player too. Add a "suggest" command which uses the AI to suggest a next move for the player. **(10 points)**

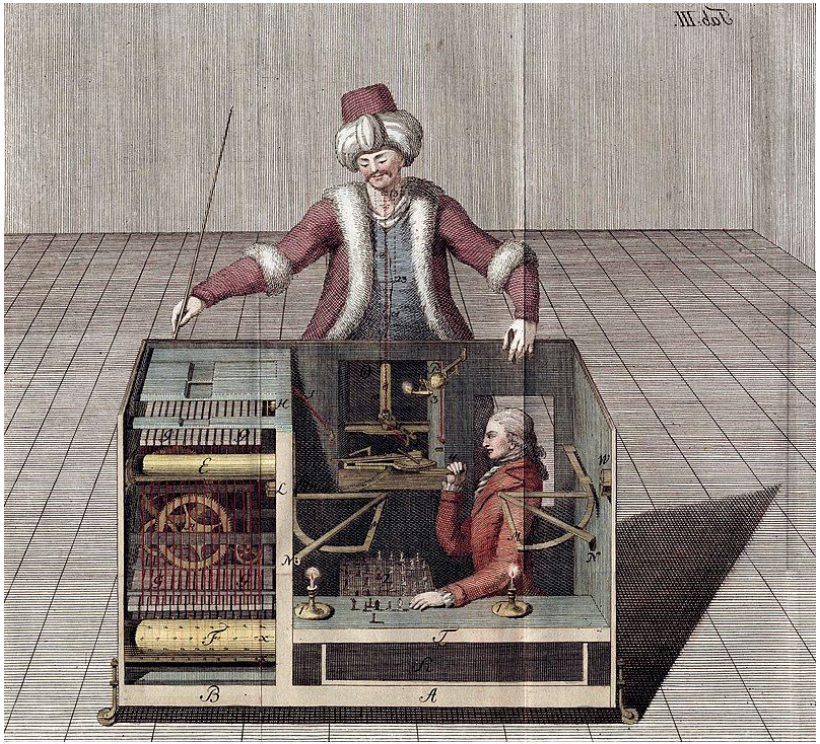
Deep Blue vs. Garry Kasparov Feb 10, 1996 – May 11, 1997

Player.java

Player.java

```
31
32  public Location selectTarget() {
33    // TODO Implement your computer opponent "AI" here to
34    // make this better than just shooting randomly.
35    Location location;
36    while (true) {
37      location = Location.random();
38      if (!opponentGrid.getCell(location).isFiredOn()) {
39        break;
40      }
41    }
42    return location;
43  }
```

Note: Your AI should NOT cheat... even though your code can access the location of the player's ships, the AI should play like a player that can't see the exact locations and has to figure out where the ship is by trial and error.



```
BASIC COMPUTER OPPONENT ALGORITHM PSEUDOCODE:

This is example pseudocode for a basic computer opponent AI. If you do just this
level of computer opponent and it works without bugs, you'll get full points for #5.

1. Search the entire grid representing your player's view of the opponent (opponentGrid)
   for a cell that has been fired upon, that is a hit, and where the ship has not
   yet been sunk.
   - When such a cell is found, look one cell in all directions, left/up/right/down,
     for a cell that has not yet been fired on. If found, return that as the location
     to fire on.
   - If no such adjacent cell is found, resume the loop of looking for cells
     with hits on ships that haven't been sunk yet.
2. If no cells were found to fire on in step 1, fire randomly on a location that
   has not yet been fired on.
```

```
SOPHISTICATED COMPUTER OPPONENT ALGORITHM PSEUDOCODE:

Here's example pseudocode for a computer opponent AI. This is just one possible
approach; you can implement it this way or in a way of your choosing, as long as
it's more effective than purely random fire!

We look for "lines" of adjacent cells in the opponentGrid where every cell in
the line meets this criteria:
- It's the same ship, that is, cell.getShip() returns the same value.
- The cell has registered a hit, that is, cell.isHit() returns true.
- The ship has not yet been sunk, that is, cell.getShip().isSunk() returns false.

To each sides of such a line are either the end of the grid, or a cell that doesn't
match this criteria.
```

```
Algorithm: Try each of these steps in turn.
1. Search the grid for horizontal lines with length >= 2. When one is found:
   - If there is a not-fired-on cell to the line's left, return as location to fire on.
   - Or if there is a not-fired-on cell to the line's right, return as location to fire on.
2. Search the grid for vertical lines with length >= 2. When one is found:
   - If there is a not-fired-on cell directly above the line, return as location to fire on.
   - Or if there is a not-fired-on directly under the line, return as location to fire on.
3. Look for any cell that is a "hit" with a ship that not yet been sunk.
   - If found, find any adjacent cell left/up/right/down that has not yet been fired on,
     and return that cell as the location to fire on.
4. If none of steps 1-3 found a cell to fire on, fire on a random location that has
   not yet been fired on.
```

Hints from Chris

- #2: Using `Player.getDamageReport()` is not going to get you want you need; It does not display ship positions
- #4: For folks writing ship placement code - If you want to save some time check out `Ship.placeShip()` and `Grid.isLegal()`
- #5-6: Your AI - `Player.selectTarget()` - **SHOULD NOT** be accessing `Cell.getShip()` - restrict your usage to `Cell.isFiredOn()`, `Cell.isHit()`, and `Cell.isMiss()`
- UPDATE from Gary: It is OK to call `Cell.getShip()`, since you are told when you get a hit which ship it is, e.g. Tom: "D-4." Frank: "Hit. Destroyer." But don't look at the `locations` array in Ship because that would be cheating! The AI can know which ship was hit, but not where it is other than where it's already landed hits.