🖥️🎉 Pair Project Kick Off

Tags	Day 1 · In Class · Mod 1 · Pair Project · Week 2
Due Date	@May 16, 2022 3:00 PM-3:30 PM
Link	
Done	☐
Date Added	@April 1, 2022 5:54 PM
Last Edited	@May 15, 2022 5:07 PM
Related to Turing Calendar (Property)	
Related to Supplemental Reading (Relation)	

Iteration 1

- [ ] create a Ship class

Ship can:

- [ ] track its health
- [ ] take hits
- [ ] report if sunk or not
- [ ] cruiser = Ship.new("type", size)
- [ ] name attribute
- [ ] length attribute
- [ ] health attribute
- [ ] `sunk?` method
- [ ] `hit` method
- [ ] create a Cell class

Cell can:

- [ ] contain a ship **or** nothing
- [ ] accepts a coordinate argument
- [ ] cell = Cell.new("B4")
- [ ] coordinate attribute
- [ ] ship attribute
- [ ] `empty?` method
- [ ] `place_ship(type)` method
- [ ] `fired_upon?` method
- [ ] `fire_upon` method
- [ ] `render` method - returns string representation of the Cell for when we need to print the board
- [ ] `render` method takes boolean argument

- [ ] create a board class

Board can:

- [ ] keep track of all of the cells
- [ ] there should be 16 cell objects (4 x 4 board)
- [ ] cells are tracked in a hash where coordinates of the cell are the keys that point to cell objects
- [ ] `valid_coordinate?(coordinate)` method
- [ ] `valid_placement?(ship, coordinates_as_array)` method
- [ ] valid placement is vertically or horizontally
- [ ] valid placement method should return `false` if user tries to place ship diagonally
- [ ] valid placement should make sure that coordinates are consecutive
- [ ] valid placement should check whether ships are overlapping
- [ ] all of the cells that contain that sunken ship should render as an “X”, not just the cell that resulted in the ship being sunk

```
pry(main)> board = Board.new
# => #<Board:0x00007fcb0e1f6720...>

pry(main)> cruiser = Ship.new("Cruiser", 3)
# => #<Ship:0x00007fcb0e1ffa28...>

pry(main)> board.place(cruiser, ["A1", "A2", "A3"])

pry(main)> cell_1 = board.cells["A1"]
# => #<Cell:0x00007fcb0e1f66a8...>

pry(main)> cell_2 = board.cells["A2"]
# => #<Cell:0x00007fcb0e1f6630...>

pry(main)> cell_3 = board.cells["A3"]
# => #<Cell:0x00007fcb0e1f65b8...>

pry(main)> cell_1.ship
# => #<Ship:0x00007fcb0e1ffa28...>

pry(main)> cell_2.ship
# => #<Ship:0x00007fcb0e1ffa28...>

pry(main)> cell_3.ship
# => #<Ship:0x00007fcb0e1ffa28...>

pry(main)> cell_3.ship == cell_2.ship
# => true

pry(main)> cruiser = Ship.new("Cruiser", 3)
# => #<Ship:0x00007fcb0f0573f0...>

pry(main)> board.place(cruiser, ["A1", "A2", "A3"])

pry(main)> board.render
# => "  1 2 3 4 \\nA . . . . \\nB . . . . \\nC . . . . \\nD . . . . \\n"

pry(main)> board.render(true)
# => "  1 2 3 4 \\nA S S S . \\nB . . . . \\nC . . . . \\nD . . . . \\n"
```

Tips for testing & set-up

- [ ] ranges for placement validation
- [ ] ordinal values for consecutive placement
- [ ] enumerable method called each_cons

- [ ] other enumerable methods (any?, all?, none?)

Iteration 3 - Building the Game

Now it's time to put together the components you've built in the last two iterations to make a working game. You are allowed to build any additional classes or methods you think may be useful to accomplish this. However, this project will be assessed on the spec outlined in the last two iterations, so don't remove any of the functionality from the Ship, Cell, or Board classes.

You are not expected to test anything related to user input and output in this iteration, but try to use TDD as much as possible to help you design your solution.

You are not expected to follow the Game described below exactly. If you want to give your computer player more personality, feel free to do so. **However, the information relayed to the user and received from the user should not change.** For instance, when firing a shot you are not allowed to omit displaying the result of the shot (hit, miss, sink) to the user. For your convenience, at the end of this page you will find a checklist that summarizes all of the functionality you are expected to build.

- [ ] You are expected to build at least one additional class to complete this iteration. This can be a single class that is responsible for running the game. You should have very little code that is not contained in a class.

Main Menu

When the user starts the game, they should see

- [ ] a welcome message that asks them if they want to play or quit.
- [ ] Whenever a game ends, they should return to this message so they can start a new game, or quit.

```
Welcome to BATTLESHIP
Enter p to play. Enter q to quit.
```

Setup

Once the User has chosen to play, you need to place the computer's ships and the players ships to set up the game.

Computer Ship Placement

- [ ] The computer player should place their ships.
- [ ] The baseline computer should simply use random placements but still adhere to the valid placement rules from iteration 2.

Player Ship Placement

Next, the user places their ships.

- [ ] User should receive a short explanation of how to place their ships along
- [ ] with a visual representation of their board (which should be empty).

```
I have laid out my ships on the grid.
You now need to lay out your two ships.
The Cruiser is three units long and the Submarine is two units long.

  1 2 3 4
A . . . .
B . . . .
C . . . .
D . . . .
Enter the squares for the Cruiser (3 spaces):
>
```

When the user enters a valid sequence of spaces,

- [ ] the ship should be placed on the board,
- [ ] the new board with the ship should be shown to the user,
- [ ] and then the user should be asked to place the other ship.

```
Enter the squares for the Cruiser (3 spaces):
> A1 A2 A3

  1 2 3 4
A S S S .
B . . . .
C . . . .
D . . . .
Enter the squares for the Submarine (2 spaces):
>
```

- [ ] If the user enters an invalid sequence, they should be prompted again:

```
Enter the squares for the Submarine (2 spaces):
> C1 C3
Those are invalid coordinates. Please try again:
> A1 B1
Those are invalid coordinates. Please try again:
> C1 D1
```

The Turn

Players take turns firing at one another by selecting positions on the grid to attack.

A single turn consists of:

- Displaying the boards
- Player choosing a coordinate to fire on
- Computer choosing a coordinate to fire on
- Reporting the result of the Player's shot
- Reporting the result of the Computer's shot

Displaying the Boards

- [ ] At the start of the turn, the user is shown both boards.
- [ ] The user should see their ships but not the computer's ships:

```
=============COMPUTER BOARD=============
  1 2 3 4
A M . . M
B . . . .
C . . . .
D . . . .
==============PLAYER BOARD==============
  1 2 3 4
A S S S .
B . M . .
C M . S .
D . . S .
```

Player Shot

- [ ] The player should be asked for a coordinate to fire on.
- [ ] If they enter an invalid coordinate, they should be prompted until they enter a valid one:

```
Enter the coordinate for your shot:
> D5
Please enter a valid coordinate:
> Z1
Please enter a valid coordinate:
> A4
```

Computer Shot

- [ ] The computer should choose a random space on the board.
- [ ] The computer should not fire on a space that has already been fired on.

Results

- [ ] The results of the shots should be displayed:

```
Your shot on A4 was a miss.
My shot on C1 was a miss.
```

The game needs to handle all of the following results:

- A shot missed
- A shot hit a ship
- A shot sunk a ship
- Coordinates that have already been fired upon

- [ ] You need to add something that informs the user if they attempt to fire at a coordinate that has already been fired upon. You may choose to either prompt them again for a coordinate they haven't fired on, or let them choose it again and inform them in the results phase that they selected this coordinate again.

End Game

The game is over when either the computer or the user sinks all of the enemy ships.

- [ ] When this happens, the user should see a message stating who won:

`You won!`
or
`I won!`

- [ ] Then, they should be returned to the **Main Menu** asking them if they would like to play or quit.

Functionality Checklist

This checklist summarizes all of the functionality you are expected to build. This will be used to assess the completion of your project:

Main Menu:

- [ ] User is shown the main menu where they can play or quit

Setup:

- [ ] Computer can place ships randomly in valid locations

- [ ] User can enter valid sequences to place both ships
- [ ] Entering invalid ship placements prompts user to enter valid placements

Turn:

- [ ] User board is displayed showing hits, misses, sunken ships, and ships
- [ ] Computer board is displayed showing hits, misses, and sunken ships
- [ ] Computer chooses a random shot
- [ ] Computer does not fire on the same spot twice
- [ ] User can choose a valid coordinate to fire on
- [ ] Entering invalid coordinate prompts user to enter valid coordinate
- [ ] Both computer and player shots are reported as a hit, sink, or miss
- [ ] User is informed when they have already fired on a coordinate
- [ ] Board is updated after a turn

End Game:

- [ ] Game ends when all the user's ships are sunk
- [ ] Game ends when all the computer's ships are sunk
- [ ] Game reports who won
- [ ] Game returns user back to the Main Menu

Iteration 4 - Additional Features

Variable Board Size

When starting a game, a user should be able to indicate the height and width of the board. All of your validations should still work with these variable board dimensions

Custom Ships

When starting a game, a user can optionally create the ships they will play with. They should be able to give each ship a name and length, and be able to create as many of these ships as they want.

Intelligent Computer

The computer should make an educated guess of what coordinate to fire on.