Battleships

Summary

We recreated the classic board game Battleship in Python. In this game, a player plays against artificial intelligence to see who can sink the other's ships first. We chose this project for a few reasons. First, we all wanted to make something that was fun and interactive. We also all knew of Battleship, and it seemed like a fun way to apply everything that we learned in CSMC105. We broke up the project into different sections and tackled them separately. This was probably the most efficient way to complete this project, as group meeting times were sometimes hard to coordinate. However, after we each completed our parts, we spent the last few days of the project compiling and splicing everything together and making sure that there were no bugs.

Background

Battleship is a guessing and strategy game simulating a battle between naval forces. In the game, each player places five ships of varying sizes on their own 10x10 grid, which they can see but their opponent cannot. They then take turns to guess a grid cell where they think their opponent has placed a ship. If there is a ship in this location, it is 'hit' on that cell, and the shooter is told that their shot was a hit. Otherwise, the shot misses and the shooter is told so. Once a ship has been hit in all of its cells (ranging from 2 for the Patrol Boat to 5 for the Aircraft Carrier) it is sunk, and the shooter is told that they have sunk the ship. The winner is the first person to sink all five of their opponent's ships.

Terminology:

Ship: One of five objects placed by each player. Ships are sunk as units. Each player has the following ships: Aircraft Carrier (5 cells), Battleship (4 cells), Destroyer (3 cells), Submarine (3 cells), and Patrol Boat (2 cells). The friendly player's ships are marked by 'O'.

Board: A 10x10 grid of cells on which ships are placed and shots are marked. Each player has their own board, which they can see, and which their opponent cannot.

Cell: One of the hundred squares on a player's 10x10 board.

Shot: Each player makes one 'shot', or guess, per turn.

Hit: A shot that has struck a ship. These are reported and marked on the board by 'X'.

Miss: A shot that has not struck a ship. These are reported and marked on the board by '+'.

Sink: When all of a ship's cells have been hit, the ship is sunk. This is reported to the shooter and marked by 'V'.

Motivation

This open ended project prompt gave us the opportunity to explore different plans and ideas. After discussing all the possible roads we could take, we chose to recreate the classic game of Battleship in Python. We decided on this game because we all felt that it would be a fun and challenging project for us. Battleship is an enjoyable game and though arduous at times, coding it was just as enjoyable. We also figured that taking on this project would help us exercise and apply our understanding of the various topics covered in class. We utilized matrices to create the boards, dictionaries to store ship locations and check for sink and win, lists to allow ordered ship placements and to keep track of damaged ships, etc. When deciding upon Battleship, we also took into consideration how conveniently we could divide up the different parts of the code for different group members to work on and make it all come together.

Main Contributions

Nathan Wang

- Created printing statements for both the player's board and the AI's board
- Defined the matrices that the boards refer to, as well as the variables
- Designed the format of the boards that the user sees
- Various other UI elements
- Created playerShot - the function that allows a player to shoot at the enemy board
- Created enemyShot - the function that allows an AI enemy to shoot at the player board
    - Random shooting
    - 'Smart shooting' after hitting with a random shot (assisted by group members)
- Debugged and cleaned code with other group members
- Compiled and spliced code with other group members
- Playtested with other group members

Caspian Macky

- Designed ship placement functions, including automated player placement and AI opponent ship placement.
- Created AI to intelligently and unpredictably place ships on boards.
- Revised and debugged AI shooting algorithm to be more alert and tactical

- Reworked printed statements' timing for ease of use
- Assisted with debugging and cleaning code
- Assisted with collating final product from individual work
- Assisted with playtesting

Tazkia Afra

- Coded function to check when a ship has been sunk
- Coded function to check when the game has been won
- Set up the early parts of the game with the display of rules and rock, paper, scissors
- Coded function to indicate when a ship has sunk by marking them with 'V's
- Assisted with the main code
- Assisted with debugging and commenting
- Assisted group members with playtesting

Methods

Unfortunately there simply isn't enough room for us to describe everything we tried, at any level of detail- *Battleship* was an extremely intensive project! What follows is a list of some of our prouder accomplishments. Side note: The only package we imported was 'random'. Everything else was coded by us, with no outside help.

Board Matrices

Our display boards were extremely aesthetic and clean. We took advantage of the fact that PyCharm (and other text editing apps) allot the same amount of space for every letter. This meant that we could create a very neat and organized grid for the user to see.

Ship Location Dictionaries

We were able to create dictionaries of lists of tuples: one dictionary per side of five entries (one per ship). Each entry had the ship name, which was used to reference it and also to print detailed sink reports. Each entry's value was a list of tuples, with each tuple being the coordinates of a cell occupied by the ship. This data structure was easy to work with and allowed for simple sink detection and reporting.

Intelligent AI Ship Placement

The AI went from placing ships randomly in valid locations, to obeying simple rules such as 'don't put ships parallel in adjacent rows/columns', to dynamically

evaluating ship placement in order to ensure that ideas resulting in adjacent ships were usually, but not always, discarded. This was achieved by creating a list of tuples, each tuple being the coordinates of a cell that was adjacent to an existing ship, and checking each potential placement to see if it overlapped with this ‘bad ideas list’. This was also applied to allow the player to automatically place their ships if they were feeling lazy.

Tactical AI Shooting

We implemented tactical AI shooting using global variables to store ‘target’ coordinates over repeated calls of the AI’s shooting function. The AI was able to detect when the target had been sunk (and find a new one, if any were available), and to designate new coordinates as the target if a ship was found. It also fired tactically around the target to ensure it would sink the ship.

Setbacks

Smart Shooting AI

By far one of the largest setbacks we had was working out all of the kinks and bugs for the smart AI shooting. We encountered a problem where if a boat was touching an edge of the board, the AI would sometimes shoot on the other edge of the board (since it would just go from one end of the list to the other). There were a lot of if-statements we had to implement in order to prevent the AI from doing something it was not supposed to do.

Smart Ship Placement

The first iteration of AI ship placement was fully random, but this method (while unpredictable) resulted in bad tactical choices. We had to fully redesign this AI algorithm from the ground up twice, but we're happy with our end product.

Conclusion and Future Work

We thoroughly enjoyed working on this project and the end result is something we are very proud of. For future work, we could utilize Pygames and implement advanced GUI to make the game of Battleship more aesthetically pleasing and more interactive.

Table of Contributions

Member	Contributions

Nathan	- Created printing statements for both the player's board and the AI's board
	- Defined the matrices that the boards refer to, as well as the variables
	- Designed the format of the boards that the user sees
	- Various other UI elements
	- Created playerShot - the function that allows a player to shoot at the enemy board
	- Created enemyShot - the function that allows an AI enemy to shoot at the player board
	  - Random shooting
	  - 'Smart shooting' after hitting with a random shot (assisted by group members)
	- Debugged and cleaned code with other group members
	- Compiled and spliced code with other group members
	- Playtested with other group members

Caspian	- Designed ship placement functions, including automated player placement and AI opponent ship placement. - Created AI to intelligently and unpredictably place ships on boards. - Revised and debugged AI shooting algorithm to be more alert and tactical - Reworked printed statements' timing for ease of use - Assisted with debugging and cleaning code

	- Assisted with collating final product from individual work - Assisted with playtesting
Tazkia	- Coded function to check when a ship has been sunk - Coded function to check when the game has been won - Set up the early parts of the game with the display of rules and rock, paper, scissors - Coded function to indicate when a ship has sunk by marking them with ‘V’s - Assisted with the main code - Assisted with debugging and commenting - Assisted group members with playtesting