CptS 111 – Fall 2024
Zach & Logan
Battleship

Project Report
Due: Monday, 12.9.24, before 11:59 pm

Instructions

Fill out this project report and submit it together with your other project files. Compare your project proposal as you complete this report and comment on any changes made to your project and the reasons for them.

1. Names of team members (if a member didn't participate, indicate this)
   Zachary Cornthwaite
   Logan Harris

2. Project title
   Battleship in Python

3. What project does
   Traditional game of battleship, coded in python. Meant to work on the same machine (you take turns on your machine. Clears the terminal after each players inputs, so no peeking! Biggest twist is that the board state is visualized in matplotlib, rather than in the terminal (most common). Keeps track of some basic stats that will be illustrated at the end.

   Also, there is an easter egg hidden in the code!!!

4. Project tasks; each task represents a single function (minimum of 6 functions plus `main()`); explain any changes to your original proposal and reasons for the changes.

   Admittedly, we edited our project a lot from our proposal. We frankly saw this coming, as we knew that our project was a little ambitious. The original project idea stemmed from Logan, as during his freshmen year, his roommate took CPT_S 111 (I believe) which forced students to make battleship in the C language. Logan wanted to do this, but in python, as he always found the idea cool. Zach and Logan never took CPT_S 111, so it seemed like a good challenge. With this said, our project was ambitious, and we definitely watered down some functionality, and scrapped a few functions, but our functionality kept close to what we originally proposed:

   1. Main() = This task was the creation of our main function, which runs our entire game. Our function stayed pretty faithful to what we wanted it to be in our proposal. It runs the game, calls other functions, and keep game stats. The only little addition is that we nested an "easter egg" in it, that will break the loop if the easter egg is found. Besides that, it does what we wanted it

to do, which it acts as the game master, calling the functions when they are needed, starts the game, and ends the game.

2. Visualize_board = This task was creating the function to display our board, which is a spin-off of our originally named function "display_board." This function was a little ambitious, and we decided to split this original function idea into visualize_board and status_update. Basically, when making the "place_ships()" function, Logan realized that he would have to visualize the board each time a player places their ships, which made it very obvious that he needed to make a function to call, with basic board information, to visualize the board. Therefore, this is a very important function call when place_ships() is running. Besides breaking this function up into multiple functions, it does exactly what it is supposed to. It takes "raw" board information from main() and other appropriate function, and acts as a "wildcard" function that will visualize the board. It makes our game much more flexible and not as "hard coded."
3. Place_ships = This task was the creation of the place_ships function, which sets up the game. After the main() function does a coin flip to determines who starts first, the main() function immediately calls the place_ships() function. Place_ships() function does exactly what we initially planned for it to do, however we separated the visualization part into its own function (visualize_board) as Logan realized how much of a necessity it was for that to be its own callable function. We did a few quality of life edits where after each ship placement it shows the board so you know where you placed it, and asks if you would like to edit your placement. There was a lot of testing on this particular function as we had troubles getting the board to visualize correctly (essentially, the board has to be inverted, so instead of starts at a normal bottom left origin, it starts at the top left, like actual battleship). Does what we planned though, besides that.
4. Take_turn = This task relates to making the "take_turn()" function, or essentially the function that takes player inputs, and tests it against their opponent's board state (this is the HIT and MISS function, depending on if you missed or hit the other player's ships). This does essentially, exactly as we planned it to, however we made essentially 2 board states. We have where each player's ships are placed as "opponent_board", and then we have "tracking_board" which shows you where you hit and missed. The function is simple, it takes your coordinate input, and tests it against the nested list of your opponent's board. If something is there, it says hit, and marks your tracking_board. If not, it says miss, and marks the tracking_board. There are multiple calls to the visualize_board function to visualize all these tracking/player boards. Besides the addition of a "tracking_board" this task stayed consistent with our proposal.
5. Status_update = This task relates to creating the status_update function, which is our first real step away from our original proposal. Instead of having a separate function that checks whether you hit or miss, and what you sunk, etc. etc. (as our original proposal wanted to do), we add these functions into the take_turn function. Furthermore, we didn't really have time to add the "you sunk my [battleship]" function into the game, so we essentially decided to play classic battleship, in which you actually don't know if you sunk the ship or not. Therefore, instead, this is more a "dumb" function that is called at the end of each person's turn to illustrate to them their progress in winning the game. It basically just calls the visualization_board at the end of each players turn (in the main() game loop) and conveys the board state. That's basically it. A

very big waterdown from our original proposal, but most of the proposed functionality of this function was picked up either by the main() function or other functions, or we decided to remove those functions all together due to deadlines.

6. Turn_time = relates to the task of creating the turn_time function, which is also another big step away from our proposal. In our initial proposal we wanted to have a turn_timer that would end each player's turn if they took too long (so the game didn't stall), however the design choice of using matplotlib as our visualization tool essentially paused the game each time someone took their turn, which make this entire function impossible (as least with our skills and knowledge). Therefore, we took the python time function, and decided to just use this function as a call during the main game loop to track how long the game was. This was a massive waterdown from our initial proposal, what it still gives us statistics, which is what we wanted, but it doesn't end anyone's turn if they take too long.
7. Check_game_over = This task relates to the creation of the check_game_over function, which was added onto our scope once we started making the main() function. We realized that once we started making the main function, that a lot of the functions we wanted main to do essentially had to become separate function that the main function could call in its main loop. Check_game_over became one of those, as we originally wanted the main function to be able to tell when the game is over. Instead, we made the main function and its main loop just call this function, which looks at each players board and checks if there are any ships still alive. If both players have ships alive, the game continues. Checks at the end of each player turn. Pretty straight forward, but we had to make this into a function that could be called in the main loop.
8. Final_scoreboard = This task relates to making the final scoreboard, which we made a function call from main. Essentially, main is hard coded to track basic stats as the game goes on, and main will call this function, with all the game stats, and this will print it out in a neat format. It will format all the numbers. Pretty straight_forward, but again, we initially wanted main() to do this by itself, but we realized that we needed to make it into a function for easier code running.
9. Check_easter_egg = this tasks relates to our last function, which is an extra easter egg which we added. Logan insisted that he wanted to add an easter egg in the game. Therefore, we added an easter egg that would illustrate credits and a large Coug logo made in ascii art, only if the player shot in the 4 corners. If the player shot in each corner as their only moves, then the game would end, announce them as the winner, and then write credits to Zach and Logan, along with a "GO COUGS" and a large Coug ascii art. All ascii art was created with a simple online ascii art generator. This was made into its own function, for similar reasons for the other functions. We wanted this to be in main, but it became too complex for us to hard write it into the main function, therefore, we made it its own "checking" function which takes the board state, and runs it through an if statement to see if the easter egg requirements are met. Does this in the main game loop after each players turn. If it it is met, then the main game loop breaks, and the game ends like normal, but with the extra easter egg ascii print, which comes from this function.

5. Explain the closed-box test cases you used to determine whether your code is working correctly. If not all your test cases worked, i.e., your code has some bugs, explain what isn't working and why you think it isn't.

   All of our testing, or most of it, was done closed box in means that each of our function were created outside of our main function, and then we brought all of our functions together into one file to make our finished project.

   For example, for our "place_ship()" function, which was made mostly by Logan, we hard coded test inputs at the bottom of the file, which we would make a comment (using hashtags, ###), and would introduce them back into our code for testing, and then make them comments again if not being tested. For place ships, it was easy, as the only input needed for that function is a player name, but other functions were harder, such as the take turn function, and funny enough, for me (Logan), the hardest function to test was the check_easter_egg function, as it had a very complicated ascii art that needed to be printed, and python kept throwing errors on the symbols that we used to make the art. On top of this, it was hard to close box test this, as it was one of the last functions we made, and it depended on the main() function working, therefore, if I wanted to make sure this function worked, I needed to ruin the game for at least 8ish turns to ensure that I could insert the easter egg conditions into the game, and see if it would return the correct output. Which for a long time, it didn't. In the case of this problem, it was due to the fact that the ascii art was using symbols which python identifies as code, such as break out cases like the backslash, we forced me to revise the ascii art without using slashes.

   Another case was the visualization of the board using matplot, which took a really long time. Although rather simple to test as the place_ships function was created outside our main function, therefore close box, it was hard to get everything to work correctly as typically battleship is played form the top left corner (A-J going down, and 1-10 going across, from the top left). However, matplot uses the typical math way of graphing, which starts from the original, so this took a lot of testing, running over and over, to get corrected. We tried many different tests, such as having the backend of the game (the actually board data and math) be one way, and then changing it once inputting it into matplot, but after a lot of help from copilot and google, we found a way to get our data to take letter inputs, then translate into appropriate index values, before plotting. Therefore, we have a matplot that looks like battleship, and starts from the top left.

   There we other more unique problems here and there, but generally speaking, they were all tested the same way as we made each function outside of the main function, then we uploaded the working function to github, and then combined them all together at the end.

Save your Word document as a **pdf** file and zip it together with your other project files which must be submitted before 11:59 pm, Monday, 12.9.24.