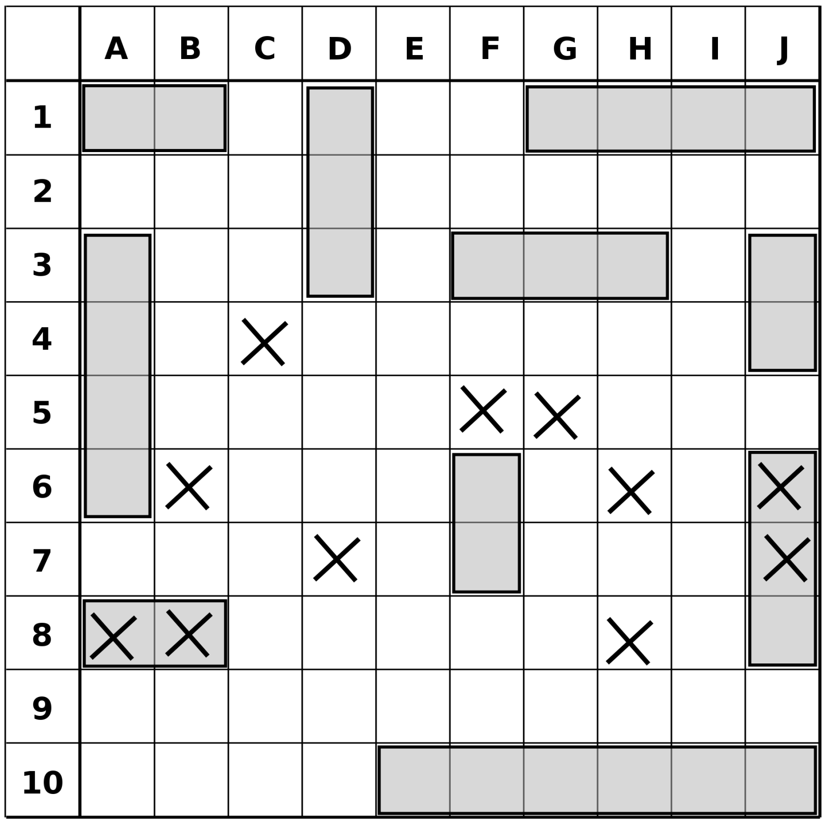
Final project PyLadies Vienna Python Beginners course

Battleship



In this project, we will put together everything we have learned during the course – functions, strings, lists, and everything else – and create a final game. I hope you will like it!

Our goal is to create a clone of the famous game Battleships – which Tyna and Lubo really like. The game itself can be challenging and fun no matter the age of players.

The project itself is not too complex. You have already used the basic principles during the course and homework and you will try to put everything together now. The following text is more like a set of instructions than teaching material and you might find something we didn't speak about during the course. In that case, don't be afraid to search for information, ask fellow students or mentors.

Also since this is the output of the course, try to write your code as cleanly as possible, with comments and meaningful names for functions and variables. This way, you can put it as the base of your new coding portfolio.

Game logic and general recommendations

The main objective is a game controlled by the user, where first ships are placed in positions on a grid and then the computer and player iterate in attempts to guess the correct location of a ship. The game is over when a computer or player destroys all the ships of an opponent. We will get to specific steps later in this document.

First, here are some basic recommendations (not rules!):

We recommend that you create a clean folder, start the project from scratch, and look into the previous codes only if needed.

If the project looks too large at the beginning (any project does, really) it is important to break it down into smaller steps. One of the possible ways could be to follow the steps described later.

As a general suggestion (recommended one), put your code to git (GitHub) as early as possible and commit regularly and often. This will:

1. Enable collaboration inside your team.
2. Allow you to come back to a previous working state of your code.
3. Help you understand the changes and development process in retrospect.
4. Teach good practice of making small incremental changes to your code.

Project sample steps and phases
Following steps 1-7 depend on each other, and provide the final solution gradually. If you complete them, you should be able to play the game!

1. Write a function that creates a grid (map) where ships will be placed. The function should take a list of coordinate pairs as an input, which represents the position of the ship. The grid should be 10 spaces by 10 spaces big and for simplicity at the beginning consider ships only one space big.

   For example draw_battleship_map([(0,0),(1,0),(2,2),(4,3),(8,9),(8,9)]) # you can switch the order of coordinates or change it completely as you like!

   ```
   X . . . . . . . . .
   X . . . . . . . . .
   . . X . . . . . . .
   ```

```
. . . . . . . . . .
. . . X . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . X
. . . . . . . . . .
```

How to do it?

- make a table (list of lists) with only dots, something like: [['.','.','.'],['.','.','.'],['.','.','.']]

- replace the dots with X in the appropriate places

- print the table using 2 nested for loops

1. Create a function that allows the user to select positions for their ships. Store this in a format that your draw function from Step 1 can accept as an argument. Limit the number of ships and verify that a position is not already taken or not outside of a map. After the user selects their ships, draw a map using the function from point 1.
2. Create a function that randomly places computer ships (also check that nothing exists on target position), but does not display the board to the player.
3. Write a function where after asking for input, you check if the opponent's ship was destroyed. If yes, remove the ship from the list of coordinates. create either two functions (one for the player, second for the computer) or try setting default arguments to a single function to play randomly for the computer.
4. Let's put it together: write a main function where the player and computer (after placing their ships) iterate in guessing the opponent's ship location. Keep score and let players know their successes or misses. The game ends when one of the player's lists of ships is empty. Congratulations! You have a game!
5. Now the game is working, but it is time to make it slightly more interesting. Let's put bigger ships into the place. Ships are no longer only single spaces, but create 1x five space big ships, 2x three space big ships, and 3x two space ones. If a player or computer hits a ship, tell them the size of their target.
6. Think about a better strategy for a computer than random and implement it - maybe next to the previous hit?