Battleships

EPR 07

Corinna Schott, Oliver Theobald

February 8, 2020

Battleships falls in the category "easy to learn, hard to master". However, when you don't have to pay attention on setting the right crosses and circles, mastering it becomes a lot easier. Therefore, this is our version of the game "Battleships", programmed in Python 3.

Contents

1 General

Battleships as a game consists essentially of 2 parts:

- Placing your ships and
- trying to shoot the enemy's ships.

It really isn't that complicated. However, finding the ships can be hard, as you have no other tools than your own shots to do so.

2 Rules

As every other game (except for "The Purge" or "The Hunger Games" maybe) there are some rules in place. Most of the rules were given, and some are adjusted.

2.1 Board

- The standard size is 10x10 fields.
- The board size is adjustable
- Our board has a quadratic shape and a side length between 5 and 25.

2.2 Ships

- A ship has a size of 3 to 6 fields.
- A ship can only be placed horizontally or vertically.
- A ship with a size of 3 is named "Destroyer", a size 4 is a "Cruiser", a size 5 is a "Battleship" and a size 6 is a "Carrier".
- Every ship must be placed manually.

2.3 Fleet

- A fleet has at least 2 ships.
- A fleet covers between 10% and 25% of the board.

2.4 Shots

- A coin toss decides who goes first.
- Shots and hits are marked on the board by colors.
- You can't shoot a field you already shot at (makes no sense).
- A player can shoot until he misses a shot (hits water instead of an enemy ship). In that case, it's the other player's turn. Alternatively, the shots a player can fire in a row can be based on the number of his remaining ships.
- It is possible to enable a spray. This means that a shot may not hit the field selected, but one of the surrounding fields. The value can be adjusted as well.

2.5 End the game

- Whoever sinks all ships of his enemy first, wins.
- It is possible to cancel the game. A cancel works like a forfeit, therefore the player who's turn it is loses.
- The boards and player names are displayed after each other, not at the same time.

3 How to play

#TBD

4 How it works

I am not going to explain all of the code for you, since that is not what either of us wants. Instead, I'm gonna explain some special functions here.

4.1 Shots based on the number of remaining ships

This function is pretty simple. Every player has its active fleet stored in a list. When a ship sinks, it is removed from that list.
If the setting is deactivated, it just checks whether the shot is a hit or a miss. If it is a miss, the other player can shoot, but if you hit a ship, you have another try.
If the setting is activated, it just gets the length of the aforementioned list and counts that down.

4.2 Spray

Bullet spray is normal, you can't land every shot where you would like it to hit. Therefore, you can enable spray. In this case it means that at a certain chance (which is normally 15% and can be adjusted) the shot won't hit the field it was supposed to hit but rather one of the eight surrounding fields.
This works by generating random numbers. The first one is used to determine whether spray even applies to the shot or not. If it does, a second number will be generated. This number is in the range from 1 to 8 and will give you the field the shot actually hits, starting east of the original target and going counter-clockwise.

5 Good and bad things

In german there is a saying "Nachher ist man immer schlauer.", which loosely translates to "Everyone is more clever afterwards.", referring to the experience you make in the process which could be useful in a potential repetition of the task.
This was the first time working with a Python GUI, and the third time working with a partner. What things are useful to know for the next time?

5.1 Frontend and Backend

The first thing that I would like to address is the connection between frontend and backend development. In this case this is on one hand the GUI and on the other hand the actual game mechanics.
The GUI needs to use the game mechanics in a way that you essentially need to decide from which way you start.

- One way is GUI –> mechanics. That means that you develop the GUI (how the game should look) and build the game mechanics around that.
- The other way is vice-versa, meaning you build the game mechanics and develop the GUI so that it uses those mechanics.

Both ways can work, but both require a very tight cooperation between GUI and mechanics development.

We decided to split the work: One programs the mechanics, the other one builds the GUI around that. This worked semi-well, where I go to the second point.

5.2 Concept

Having a concept of what you do is important, right? Well, having a concept is only half of that part, the other half is having a good concept.

Everyone knows how battleships works as a game. However, it's easy to lose track of everything you need and everything you don't. In this case, I (who was responsible for the mechanics) wrote a ton of functions and classes and – in the end – didn't know anymore what was important, what not, what I already had, what I still needed etc.

So what do you do in this situation? There is no universal solution for that, but for me, I just did a full reset.
I closed the program, played some video games, opened it up again, turned each line of code into comments, grabbed a beer and just wrote down "If I had to write a concept for this program, what would it look like?". I thought about the classes needed, the functions the game should have, how it would look in the GUI (didn't actually write that down since it wasn't really my part and it's not that important for the mechanics) and how everything interacts with each other.
Then I turned my list into cards of a kanban board and just went through them one by one, writing the code I still needed or – in most cases – just reversing comments of already existing functions.

What I'm trying to say is that having a general idea of what you're doing is good, but not enough. A well-structured list of everything you need is worth so much, it saved me hours of trying to figure out what is missing.

5.3 Trust and Communication

Those are probably the two most important points of this whole section.

First of all, you need to trust the other person that he/she will do his/her part, especially when hard-splitting the work like we did. On the other hand, you need to earn and keep that trust up. This is done by communication. Telling the other one where you are, asking for input when you are stuck, discussing design ideas (how/where a function should be implemented), but also commits (when you are using something like git) show the other one that you aren't just slacking off and playing video games, but that you actually work on the project and try to make it so that it will be functional when approaching the deadline.

Everybody needs to find their own system for this, but if you don't believe that the other person is able to do the remaining part, you are left to decide between hoping that he/she does make it in time or doing it yourself (which isn't the sense of group work). Therefore, just keep up a good communication.

5.4 Conclusion

The next time doing such a group work, I would invest much more time in thoughtfully planning the program and how it works in detail so that coding becomes more of a breeze. This involves both game mechanics and GUI since they are very close to each other.

6 Final words