Post-competition

I would say, there were 3 possible approaches for this competition:

* Heuristics
* Reinforcement Learning
* Machine Learning using data from the replays of the top performer players.

Since I joined late, I opted to do heuristics, since there was already a public kernel to use as a baseline. Heuristics works fine. Actually the winning solutions were using heuristics and not ML, but it is clearly not an approach that a data scientist should take. It can be very time demanding, frustrating and not very useful in terms of learning. Next time, I should do ML or RL. Actually if u check the code from the 1st solution (11k lines of code), u see how much work you need to put in to have a winning bot using only heuristics.

# Insights from top performer competitors

<https://www.kaggle.com/c/halite/discussion/183312>

<https://www.kaggle.com/c/halite/discussion/183389>

* One of the approaches is to use data from the replays of the top performer players, train the model with that, and translate the predicted probabilities into ship actions.
* Another approach, which works better, is to have a ml similar to what we have commented before, but equipped with fine-tuned heuristics on some key policies such as base spawning, ship converting, base protecting,… which directly override the outputs of the model when needed.
* The winning solution <https://github.com/ttvand/Halite> and <https://www.kaggle.com/c/halite/discussion/183543>, comments that tried to use Deep RL but failed to do so. In his own words: “It turns out that the credit assignment part is very hard to get right with an arbitrary number of units (ships/bases), a long episode duration and a dynamic opponent pool. Not having infinite compute is also annoying.”
* At the end the winning strategy was the combination of many, many carefully combined heuristics in combination with learning the opponent behavior wherever possible in order to obtain a dynamic and opponent specific strategy. He stores information of how the opponent reacts towards certain situations and, from that, he scores the opponent to know how aggressive is he.
* He also analyses following steps after taking an action to avoid actions that can get the ship killed. Ie, he assigns a low level to the actions that can lead to a no valid escape of the ship if the opponent plays optimally (minimax strategy: we consider the oponnet takes the best action possible).
* 1st solution: effectively had my internal leaderboard and ran Bayesian Optimization to decide on the hyperparameters that work best against the current internal pool. The pool contains a wide range of tactics / exploiters, as well as earlier iterations and the best public leaderboard agents. In the last week, I abandoned the internal validation and only then optimized the strategy for the opponent distribution on the leaderboard.
* Another small comment. I should avoid working from the kaggle kernels. It is just a code editor (a bad one), and as such, has nothing to compare against an IDE like pycharm. Coding from there becomes more time consuming and frustrating than it should be, because on many occasions it goes really slow, it freezes and many other annoying things.
* Basic solution that achieves great results: <https://github.com/soosten/Halite-Bot>
* Next time, I should define the agent in a .py file instead of a .ipynb
* I should create a settings.py file where I define all the constants of the heuristics, such as ATTACK\_DISTANCE and so on.
* I should split the work in several .py files. For instance, one file to define the convert strategy, another to define the defense, and so on. I should have a class for each one, The main file should import all these.