Halite 4 Post Mortem

Syncbot

Written by Houwang

1 Introduction

Halite is an AI programming competition where competitors design artificial intelligence bots to compete in a 4v4 real time strategy game. I will be representing my team to give a brief overview of our bot (provisional 21st silver).

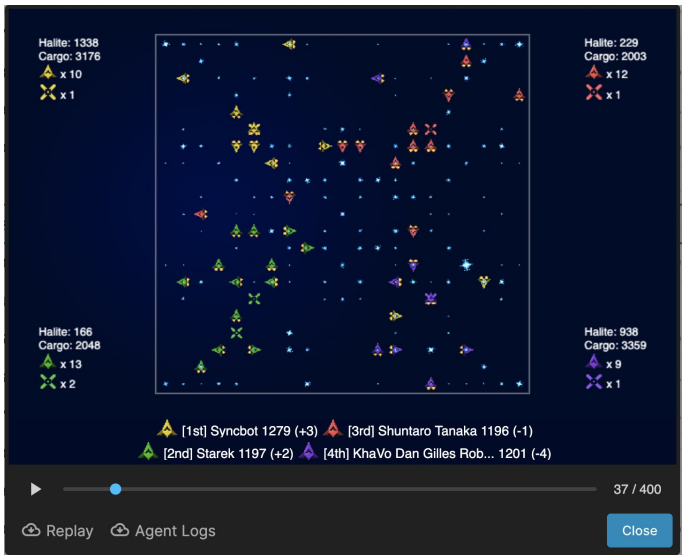
https://www.kaggle.com/c/halite/overview

My team:

https://www.kaggle.com/markhaoxiang
https://www.kaggle.com/idiott
https://www.kaggle.com/chengtianyue
https://www.kaggle.com/tangshuyun
https://www.kaggle.com/atikahamed

2 Rules

This year's game is a variation of Halite 3. Matches take place on a *21 x 21* wraparound grid.



Each game takes place over the course of 400 turns. There are two types of units (shipyards and ships) and the end goal is to obtain the largest amount of stored halite (The resource used in this game). Thus, bots must make intelligent decisions regarding the actions of each unit while accounting for the complex map environment.

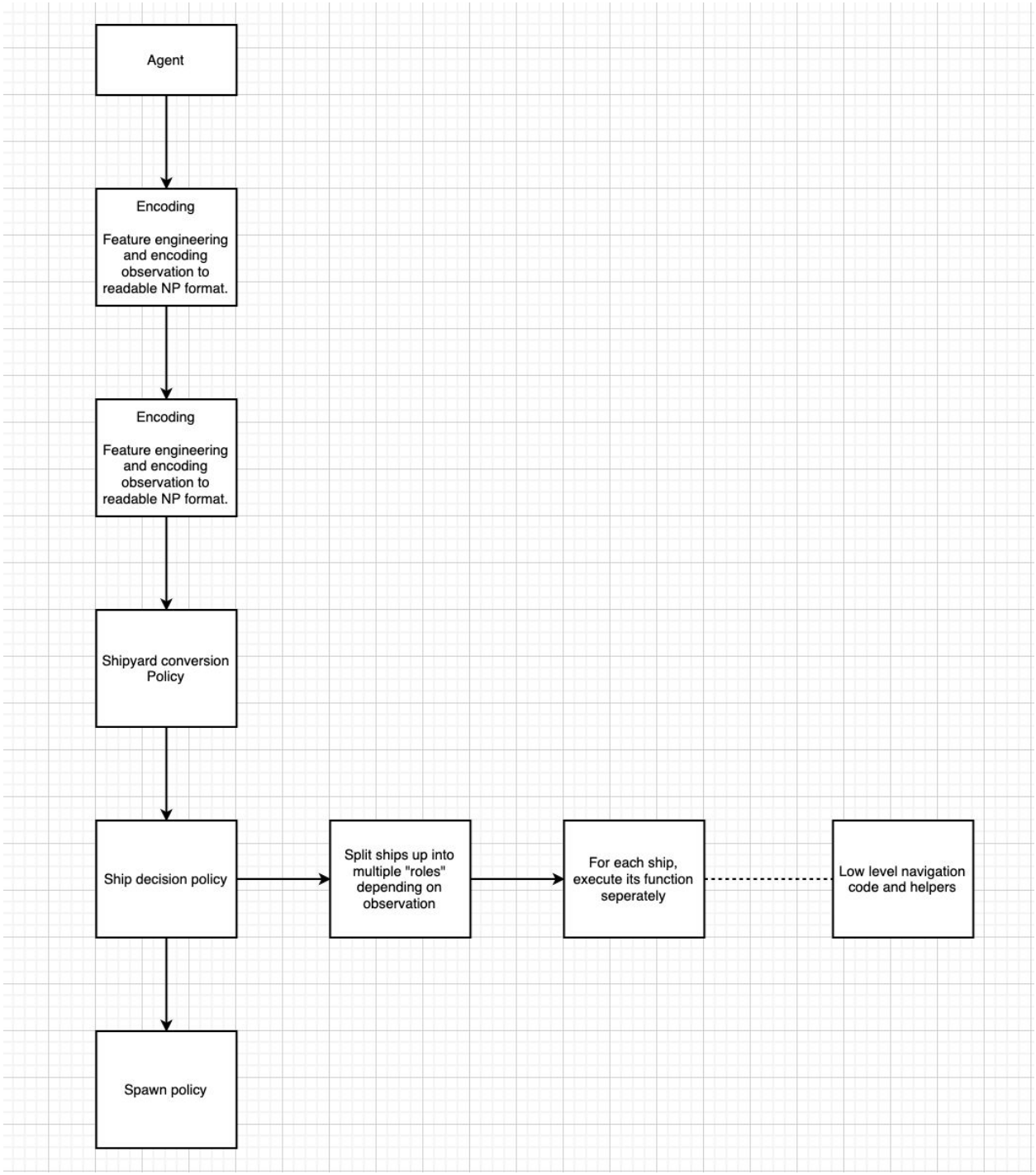
Game Mechanics

https://www.kaggle.com/c/halite/overview/halite-rules

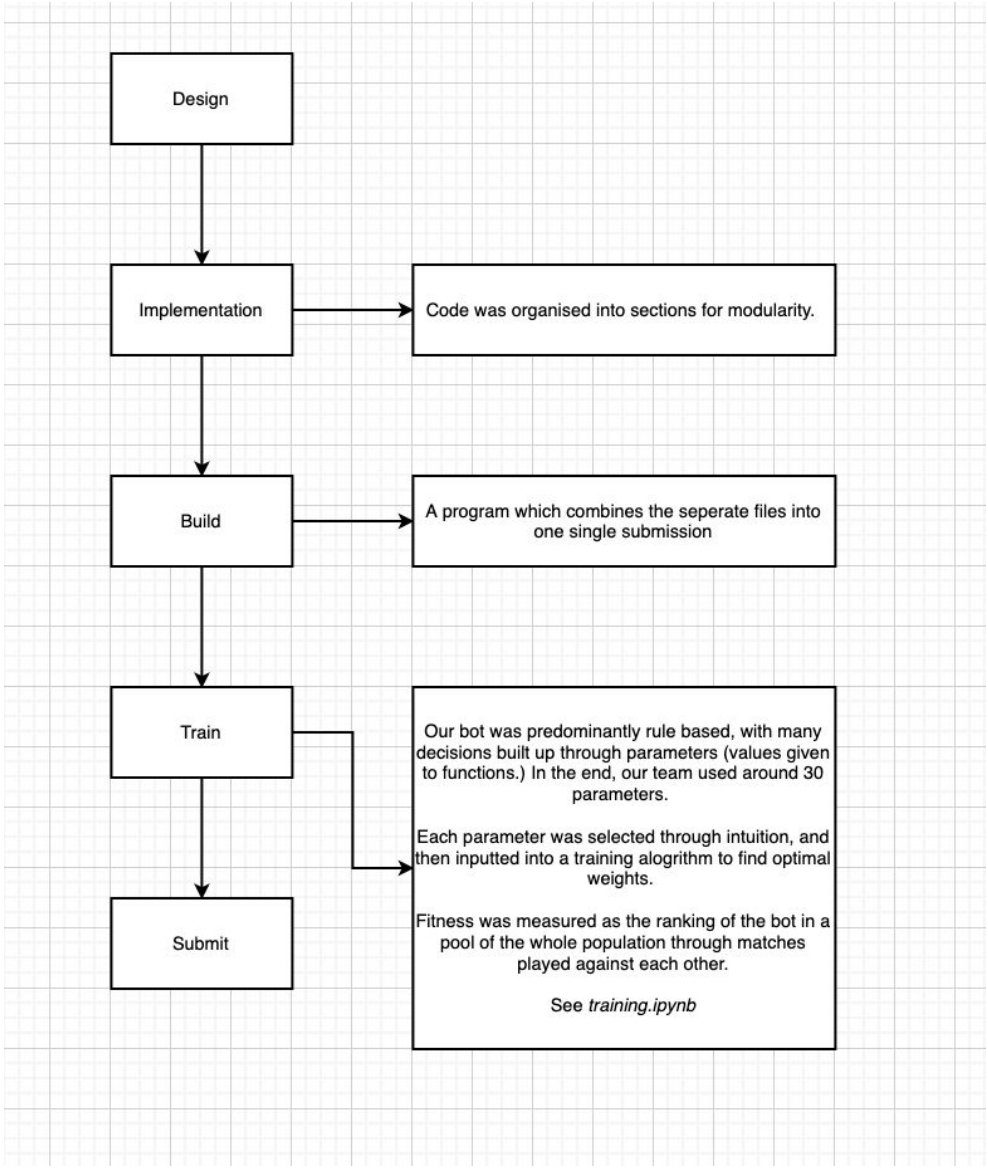
3 Framework

To accomplish the tasks mentioned in section 2, we built two frameworks - a development framework and a bot framework.

3.1 Bot Framework



3.2 Development Framework



4 Approach

4.1 Conversion policy

We took a greedy approach to our conversion policy. Each turn, a halite-value gained for building a shipyard on each cell was calculated. (See *convert.py, shipyard_value()*). This was calculated by considering the distance to the nearest shipyard, the amount of halite nearby, the ship-shipyard ratio, the relative dominance (control of the area by allies) of the cell and the current board step. Next, the highest value cell is chosen.

We try to then convert if the gain of building a new shipyard is higher than the cost of building one (including the opportunity cost of spending the halite to construct a ship).

```
IF conversion_value > ship_value AND conversion_value > 500 + ship_value THEN
        build_shipyard()
```

build_shipyard() marks the chosen cell as a shipyard. Ally ships view this cell as an existing shipyard, and will consider it for use when depositing halite. When allies arrive at this cell to deposit and the shipyard is not built, the ship will convert.

Edge cases such as no shipyards remaining involved moving the closest ship to the targeted shipyard to be built.

4.2 Ship policy

Ships were split into multiple roles, and each role served a specific function. I will go over the main role (miner) as well as give a brief overview of several other roles experimented with various degrees of success.

4.2.1 Miner

The “miner” was used as a general multirole bot that primarily mined. The core intuition behind this bot was to solve the assignment problem (https://en.wikipedia.org/wiki/Hungarian_algorithm) as assigning each ship to a target on the map.

Mining - the reward for mining a cell took inspiration from https://www.kaggle.com/solverworld/optimus-mine-agent (Thanks for sharing KhaVo) halite_per_turn function. This was supplemented through complex heuristics involving distance, safety and halite spread, with emphasis placed on ships considering similar danger regardless of distance.

Attacking - attacking was a value generated as a linear function between the relative dominance of an area, the halite of the cell and the target itself (shipyard / ship halite). The parameters were selected through the use of a genetic function.

Guarding - a reward was given for ships to remain in the friendly shipyard, with the variable parameter as the distance to the nearest opposing ship.

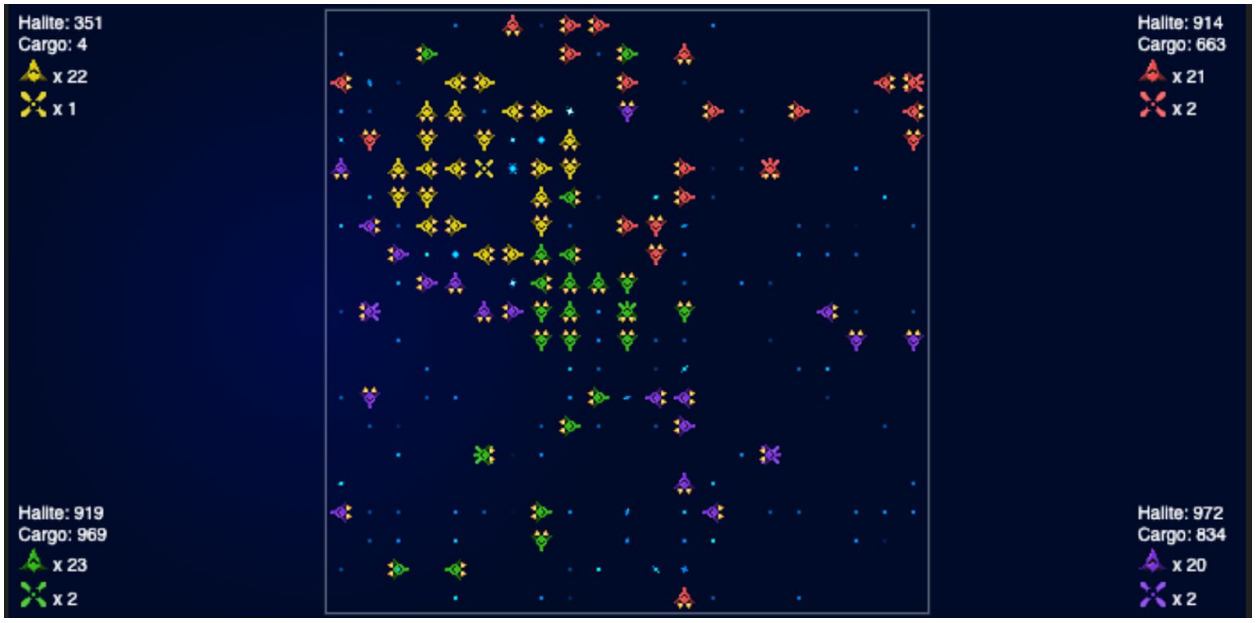
Return reward - the deposit reward for a ship to return involved the distance, stored halite and whether or not the bot wished to construct new ships. (So ships will return more often in the early game when there is a higher opportunity cost of continuing to mine)

Control - for several of our bots (Submission16943752), ships were encouraged to park on empty cells guarding nearby halite areas.

4.2.2 Attacker

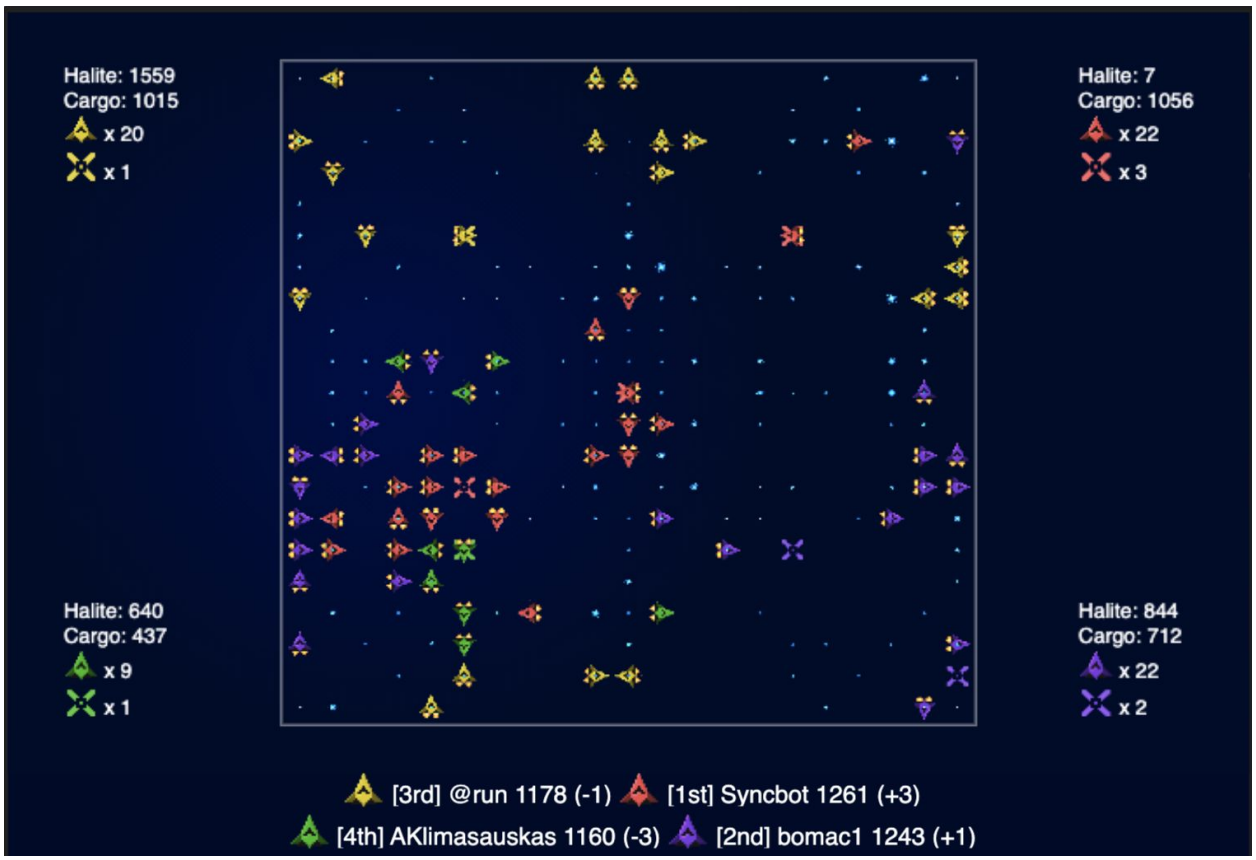
Where each empty ship attempts to attack the nearest non-empty enemy ship, with a multiplier for several ships to attack the same ship.

4.2.3 Farmer



(Yellow) Taking advantage of the unwillingness of opposing ships to attack empty ships in order to create a wall.

4.2.3 Swarmbot



Camping the enemy team (Red camping green) by building a shipyard near the opposing shipyard.

4.3 Spawn policy

4.3.1 Custom spawn

The first spawning policy we attempted was a greedy policy. Each turn, a ship_value (The expected halite to be gained from a ship before the game ends) is calculated. If this value is larger than 500, a ship is constructed.

```
def ship_value():
   if len(state['myShips']) >= 60:
       return 0
   res = state['haliteMean'] * 0.25 * (state['configuration']['episodeSteps']- 30 -
state['board'].step) * weights[4][0]
   res += (len(state['ships']) - len(state['myShips'])) ** 1.5 * weights[4][1]
   res += len(state['myShips'])  ** 1.5 * weights[4][2]
   return res
```

Where *weights* are parameters trained by a genetic algorithm. (3.2)

4.3.2 Learned spawn

The second spawn involved the imitation of top teams such as mzotkiew through a multilayer perceptron.

Features:
Halite in bank
Mean halite on map
Board step
Total number of ships
Total number of friendly ships
Boolean of whether ship construction results in ally collision

Architecture:
nn.Linear(6, 4),
nn.ReLU(),
nn.Linear(4, 4),
nn.ReLU(),
nn.Linear(4, 1)

Final accuracy achieved is 98%.

4.3.3 Edge case

We would try to build a ship to avoid shipyard death if the shipyard was the last ally shipyard remaining.

4.4 Navigation

We used Dijkstra's algorithm, and the cost of travelling between cells is the danger of travelling onto such a cell. The bot tried to avoid all collisions, as it is not profitable to collide with an enemy ship for a trade.

Friendly movements were taken in priority; higher priority ships moved first. Where a movement would cause a friendly collision, the target friendly ship that will be collided into is given the next highest priority. All ships that have already moved have targets marked as obstacles. This system allows for *swapping* where friendly ships exchange positions as well as *collision avoidance*.

5 Final Words and Code Release

First of all, I'd like to thank the incredible Kaggle and Two Sigma staff for their amazing support and organization of this event. I would also like to thank my teammates and the community for the great experience of spending 2 months competing!

Our code is available here -

Github (Genetic Trainer, Bot and misc):
(Farmbot is our newest bot)
(Trainer is training.ipynb)
https://github.com/IDI0T/halite20

Notebook(Spawning Emulator):
https://www.kaggle.com/markhaoxiang/syncbot-spawning-emulator