Games and AI

Assignment 3 - Machine Learning

Joseph Warren s3941911

This project was inspired by a concurrent assignment in my Algorithms and Analysis subject in which we studied the algorithms and strategies that led to the fastest completion of the classic battleship game. I thought it would be interesting to see if AI could learn or supersede the efficiency of these strategies through reinforcement learning on ML agents playing Battleship in the Unity game environment. Steps per episode is the measure of success for my tests as it corresponds to turns taken to complete a game. Steps/turn and episode/game will be used interchangeable throughout the report.

Setup

Battleship Repository: https://github.com/CantankerousCabbage/BattleShip

*Best agent Test38 is the current loaded config. Execution detailed in readme.

Board

Before starting, I needed to build my own version of battleship in Unity. The game consists of a 10x10 board which contains 5 ships of differing size, instantiated and randomly placed at the beginning of each training episode (or game). This models the standard Battleship setup.

The aim is to train my ML agent to get close to or exceed the average turns taken for four different strategies to sink all ships and complete the game.

Strategy 1. Random strategy: Simply randomly firing.

Strategy 2. Hunt/target: Hunt mirrors strategy 1, but once a hit is achieved adjacent tiles are targeted till hit cells are encapsulated in misses.

***Strategy 3. Parity/Target*:** As per Strategy 2, hunt targets every second tile rather than random selection.

Strategy 4. Probability: Calculates all possible ship combinations for any given square selecting the one with the highest probability of containing a ship as as the target.

More information on strategies and their efficiency can be read here[1].

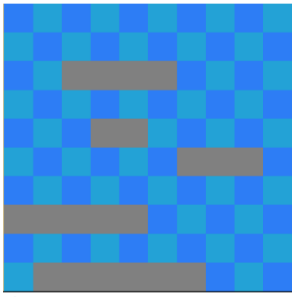

Figure 1. Random Board Setup

Ship	Length
Destroyer	2
Submarine	3
Cruiser	4
Battleship	5
Carrier	6

Table 1. Ship Dimensions

Strategy	AVG Turns
Random	96
Hunt/Target	68
Parity/Target	64
Probabillity	42

Table 2. Strategy Efficiency

Our ML agent fires at the board on each step until all ships are sunk. It may take the maximum 100 turns. Upon completion the game board is reset and all ships randomly placed at new coordinates. This is repeated until training is terminated.

ML-Agent

Before training could commence in earnest, the observation and action methods as well as a reward scheme needed to be decided to allow the agent to play and interpret the game board. I ran trials using default hyperparameters to determine which methods I should implement.

Actions: My initial trial run utilised two discrete branches of size 10 each to represent the x, y coordinates of the board. While this was the most natural representation of target choice, it allowed the agent to select the same target cell repeatedly, resulting in $\approx$ 6000 selection attempts to complete a 100 turn game. This massively slowed training and would have required training beyond the objective to rectify. To circumvent this, I changed the action array to 1 discrete branch of size 100. To prevent duplicate selections a layer mask of previous turns is applied to the action array at the beginning of each step, narrowing available choices to the remaining (yet to be fired upon) board cells. Coordinates are extracted in the form $y \leftarrow \lfloor action/10 \rfloor$, $x \leftarrow action\%10$.

Observations: I tested several observation methods over the course of the first 10 trials to determine what information was of most use to the agent. Trialed setups included:

Stacked Vectors: This setup reported the x, y coordinate of the last action (shot) and its status, hit/miss were represented by 1 and -1 respectively. I tried this with both maximum stacked vectors and with sequence set to 100 so that the agent could piece together a view of the board as it fired. Variations on this observation scheme included counts on miss and hit sets. Due to the board representation being incomplete this observation method faired poorly.

State Representation: To improve the comprehensiveness of the data being supplied to the agent I changed the observation scheme to include all 100 tiles. Each observation reported the status of a given cell prior to action selection as hit, miss or neither represented as 1, -1 and 0 respectively. The index of each observation corresponded to the action index that would target that cell. This view of the entire board state negated the need for sequencing or stacked vectors as it provided a complete view of the board state each turn.

Reward Scheme: The initial reward scheme was simplistic awarding points on a hit and game completion. Test 10 I changed this to reflect the feedback a player would get in game and emphasise the win conditions.

Positive Points:

- Hitting a Ship: $0.5 - 1$
- Sinking a ship: $0.5 - 1$
- Winning the game. Scaled by speed. $(100 - turns) * 0.1$

Negative Points:

- Turn based decrement: $0.05 - 0.2$

The above scheme was utilised for the majority of the training with variations to weighting throughout depending on the emphasis of a particular training run. A reward for exploring adjacent hit tiles was used for several sessions but removed. Not only was it ineffective but it was training the agent to emulate the hunt mode rather than giving it the framework to devise the strategy itself. Imitation training could be employed to its fullest effect by providing the agent with the probabilities for a hit in a given cell as per the probability strategy.

Training

With action, observation and reward schemes broadly established I started to systematically improve the hyperparameters governing my behaviour policy. As I was running my tests off two computers adjustments to a parameter were generally mirrored in the reverse on the other machine to verify the intended effect. Firstly, to stabilise training updates I increased the buffer size drastically and decreased the learning rate. This decrease the range of results between episodes and allowed me to attempt to improve the policy. To encourage exploration and policy evolution, I increased beta drastically early on. This was paired with increases in time horizon and batch size to

100 to capture the relevant experiences in an entire episode. Gamma was gradually reduced to minimum over the training period as immediate rewards (hits) equated to long term success. Gamma was gradually reduced to minimum over the training period as immediate rewards (hits) equated to long term success. Epoch was increased over the course of training to allow more passes through the experience buffer and offset the lower learning rate. Epsilon was increased to the maximum recommend range to allow more deviation and corresponding policy evolution. Paired with a high beta this increased the initially policy loss and shortened the time required to improve turns taken. Increasing layers generally had a negative effect but maximised hidden units increased improvement before entropy plateau was reached and learning abated. Normalisation was set to false as our observations were already normalised and after initial experimentation with observations and in turn tweaking memory size and sequence these were reverted to their default values.

Results

All results: https://drive.google.com/file/d/1ZnHlKvb2LRsq3tzzw1HGW2PbNRjyLnR_/view?usp=sharing

*Tests are not always in chronological order as they were carried out on two machines simultaneously.

Results were drawn on ≈30 sessions to test various hyper parameters while fine tuning the reward scheme. The initial ten tests were used to calibrate actions, observations and get used to the affect of hyperparameters. The initial ten setup tests ran for ≈100k steps. This was partly due to memory bottleneck due to un freed unity objects on board reset. Post set up and on resolution of the memory issue promising parameter setups ran from 4 to 8 million steps while regressive parameters were generally terminated at 1million steps.

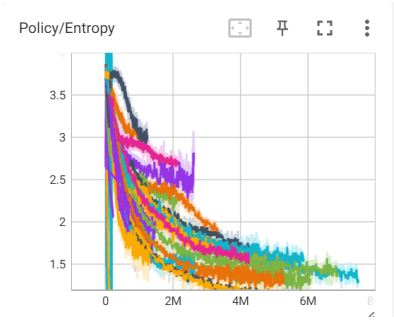

Figure 5. Entropy All

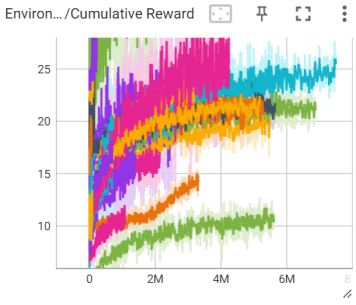

Figure 6. Reward All

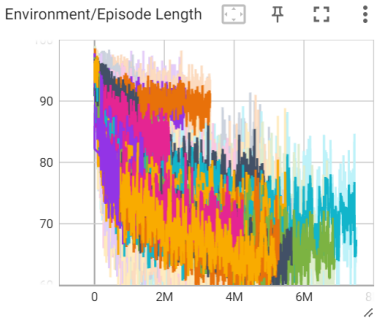

Figure 7.Episode Length All

The import result metrics are episode entropy and episode length. Episode length corresponds game turns taken to win and is inversely proportional to the cumulative reward. The decrease in episode length over time as the is matched by a decrease in entropy. Both plateau at the same time as the agent stops learning.

Most training runs are altered based on the previous most successful policy. Once the standard policies episode length is exceeded the new policy is adopted as the basis for future tests. Four pivotal training sessions were 11, 13, 19 and 38.

11 - was the first run with the stablished action, observation and reward scheme. Results when graphed showed the first stable upward curved gradient trending downwards.

13 - Saw the first milestone in improved performance due to doubling of the buffer size and a corresponding increase in batch size as well as scaling of end game reward from $100 - turns \implies (100 - turns) * 0.1$.

19 - Beta increased ten times from $1.0^{-4} \implies 1.0^{-3}$. Also saw the first increase large increase in hidden units $96 \implies 256$.

26 - Hidden units further increased $256 \implies 512$. Epsilon $0.2 \implies 0.3$. Normalisation to false. Enabled more rapid policy evolution.

38 - Reward strength $1.0 \implies 1.2$. Learning schedule set to constant. Preventing depreciation of learning rate led to marginal improvement on test 26 results. Though not evident from the graph it has an improved average of ≈ 62.

Test 11, **Test 13**, **Test 19**, **Test 26**, **Test 38**

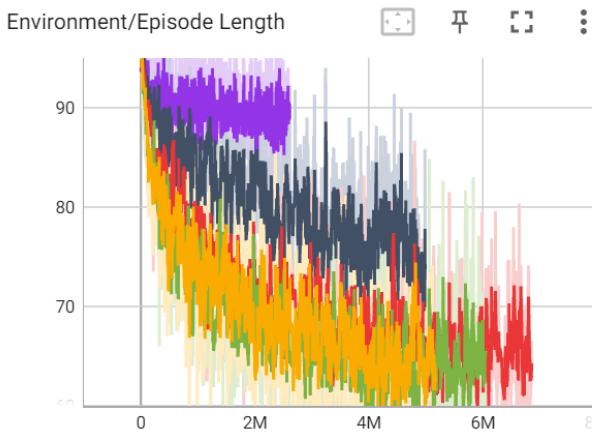

Figure 8. Episode Progression

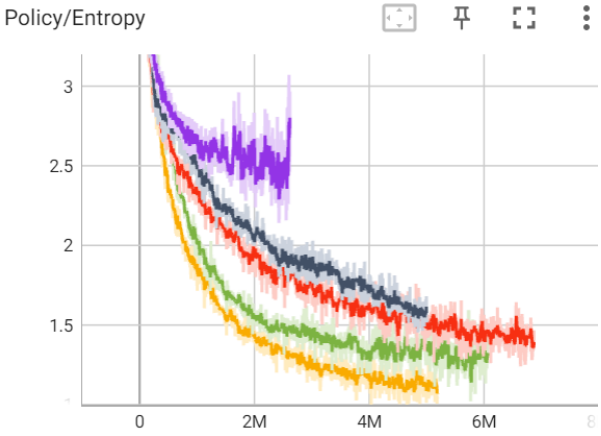

Figure 9. Entropy Collapse

Initially with minimal (no directed) adjustments to the hyperparameters the agent failed to complete the board consistently in sub 90 turns. Increases to buffer and corresponding increases to batch size and number of epochs significantly improved learning by providing ample experiences for policy improvement. Increasing beta and epsilon then allowed more exploration and deviation from the initially policy for more rapid policy improvement. The third significant factor was increase in hidden units to provide ample neurons to process the large number of inputs/actions. Turn improvement was incremental over the tests with the best being achieved in test 38.

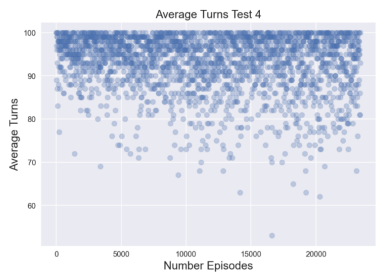

Figure 10. Test 4 Results

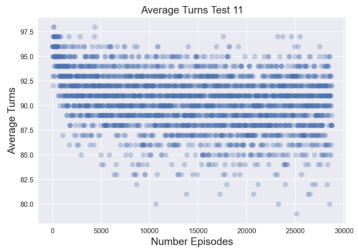

Figure 11. Test 11 Results

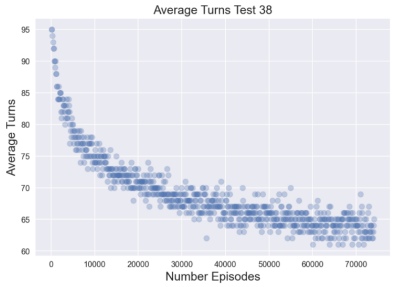

Figure 12. Test 38 Results

The agent in the initial training sessions adopts a random strategy, resulting in average game length of above 90 turns. From the diagram below we can see agent 11 fires at random. Multiple hit locations (red squares) are not encapsulated in misses (white squares) before other non-adjacent cells are fired upon. No coherent strategy is utilised.

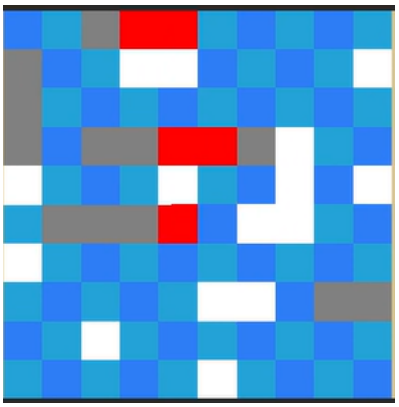

Figure 13. Agent 11-Early game

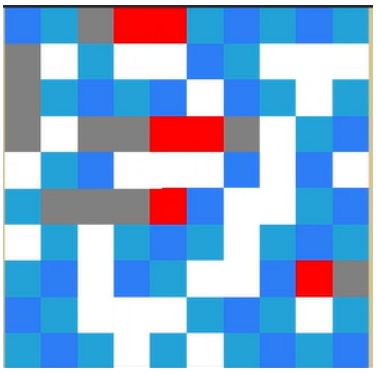

Figure 14.Agent 11-Mid Game

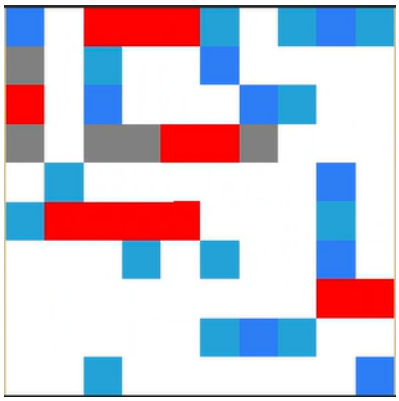

Figure 15. Agent 11-Late Game

In contrast the best performing agent (randomly selected game) utilises the parity rule shooting at every secondary cell, closes out hits along a given axis as well as opening the game by initially firing on the high probability tiles in the center of the board.

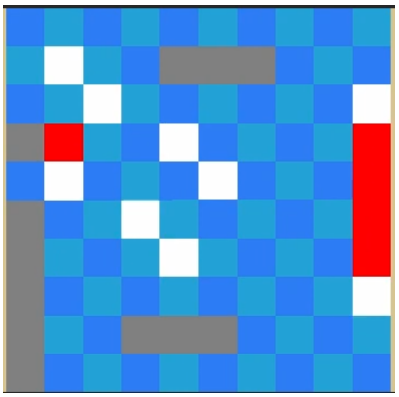

Figure 16. Agent 38-Early Game

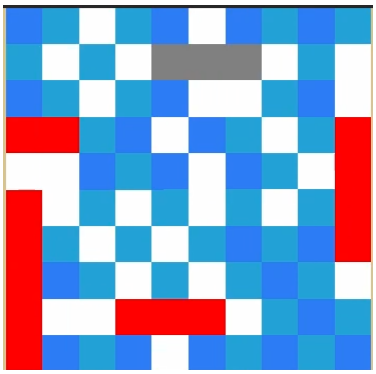

Figure 17. Agent 38-Mid Game

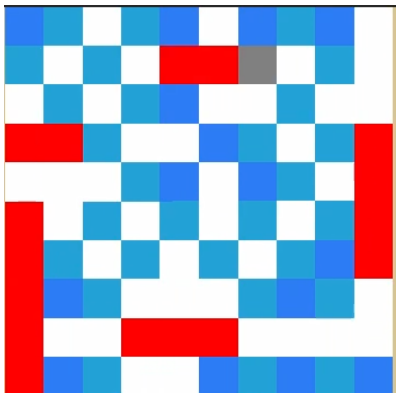

Figure 18. Agent 38-Late Game

Conclusion

Overall the best performing agent reached an average of ≈62 turns for game completion by the end of training. This was a huge improvement on the initial test which averaged ≈98 turns. The improvements were incrementally improved by fine tuning the agents hyperparameters. Equally important was the reward scheme that emphasised turn efficiency though a scaling win reward and per turn reward penalty. Masked actions provided efficient action selection, whilst the observations provided a comprehensive view of the board state on which to base them. The downside of of having 100 inputs and observations was that policy adjustment took a reasonable long time, usually 1million plus steps to make significant improvements. Agent 38 exceeded or was on par with 3 of the four strategies, marginally exceeding the average turn completion of the second best strategy, parity/hunt. This indicates a high level of competency beyond that exhibited by many human players. It did, however, fail to get anywhere close to the probability based strategy despite showing aspects of it in its initial tile selection for most games. This potentially could have been achieved using imitation oriented training by calculating the probability externally and exposing it to the agent through the observation or reward scheme, it would have trivialised the training process by providing the solution as opposed to a framework.

References

1. http://www.datagenetics.com/blog/december32011/