Building a Battlesnake With Different Approaches

Farid Anjidani (V00971584)
farida@uvic.ca

Chris Chiu (V00973917)
christopherchiu@uvic.ca

Zixuan Deng (V00971633)
zixuandeng@uvic.ca

Abstract

In this paper, we explore a multitude of machine learning techniques to compete in the game of Battlesnake, which is a multiplayer version of the traditional game of Snake. We evaluated the different performances by pitting each model into the same arena and discovered that regarding our available computation resources, simplicity was often best. When it came to playing Snake, the snake that was coded with conditional statements often outperformed and outlived the snake that was trained with baseline heuristics in a reinforcement learning setting. Although these results were surprising, we concluded that more work still needs to be done and perhaps adjusting the heuristic values a bit more would lead to different results.

1 Introduction

Over the past decade, the use of artificial intelligence in games has skyrocketed as teams around the world strive to build the best AI to compete against each other. In games like Chess and Go, researchers from IBM and Google were able to develop complex neural network models that are capable of beating the top human grandmasters. And so, inspired by these accomplishments, we decided to experiment with our own machine learning techniques to compete in a simpler game called Battlesnake.

Founded back in 2015 at the University of Victoria, Battlesnake is a multiplayer, battle-royale twist on the classic game of Snake. Every year, teams would get together to design and develop the behaviours of their own Battlesnakes with the objective of competing for limited resources within a limiting arena. The rules of the game are simple; each Battlesnake is placed randomly on a game board and every turn, they must move in one of four cardinal directions. But, they must also be wary of elimination through collision with enemy snake tails, walls or hazards. And they must also watch their health bar, as each turn they waste not collecting food, they will keep shrinking until they are also eliminated. By eating food, the length of a snake will grow and this allows for larger snakes to eat smaller snakes when put in the situation of a head-on collision. As a result, the main challenges faced by developers are these multi-agent environments and the time constraints to choose the best move for each turn. To solve these problems, the top teams will typically employ a mix of strategies ranging from Smart Moving Strategy Snake using Conditional Statements, to search algorithms to more advanced AI neural networks.

2 Related works

In the six years since this event has begun, there have been few papers written related to the strategies employed by Battlesnake competitors. In this section, we will provide a brief overview of some of these team's published works.

In [1], Chung et al. proposed the development of a multi-agent reinforcement learning playground with Human-In-the-Loop Learning (HILL) called the "Battlesnake Challenge". This framework consisted of two components: an offline module for multi-agent reinforcement learning, and an online module to benchmark performance against other public agents. When using the reinforcement learning techniques, they used four modifiable heuristics to improve an agent's behaviour given the state, action, and reward. The heuristics are to avoid hitting the wall, avoid moving in a forbidden manner, starvation, and trapping other snakes. Using these heuristics, the Proximal Policy Optimisation algorithm was then used to train the snake's policy.

In AlphaSnake Zero[2], the authors applied the same approach for the AlphaGo[5] game in to the Battlesnake. They are forced to simplify the network structure for the AlphaSnake due to the lack of the computation resources. In Figure 1, we can see the structure of the AlphaSnake. Since for the Reinforcement Learning problem they needed the true labels for the training, a Monte Carlo Tree Search is used to get an estimate of the optimal policy. After each training process the new network would compete against the previous ones. Then, they replace if it performs better. For input states, they have 21*21*3 images as they fix the snake's head in the middle of the input so for 11*11 map the input size should be 21*21. The second channels are the other snakes and walls. The last channel is for the locations of the food. After training, they achieved 100th place on the Battlesnake leaderboards in summer 2020.

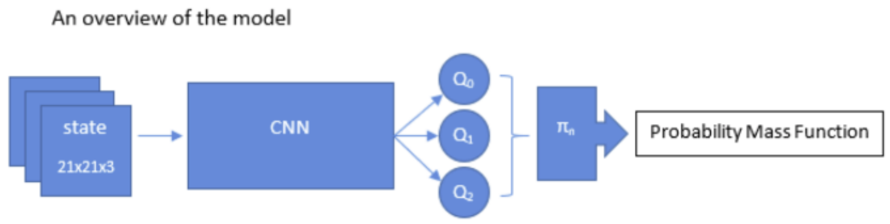


Figure 1: AlphaSnake Zero Model

3 Methodology

In this section, we will elaborate on the different methodologies that we have used to build each of our Battlesnakes.

3.1 Reinforcement Learning using Amazon SageMaker

Reinforcement learning develops strategies for sequential decision-making problems. In reinforcement learning, we have two main parts: agent and environment. The environment gives the agent its state and based on the state the agent makes an action and environment based on that action gives the agent a reward. The agent goal is to make best actions based on the state it receives, and to maximize the expected cumulative reward. The strategy to take the actions is called a policy. Assume you have a starting point t=0 and an ending point t=T (your snake dies). Each game creates an episode that contains a list of states (s_t), actions (a_t), rewards (r_t), and next states (s_{t+1}). Therefore, you can express the cumulative reward as the following equation:

$$\sum_{t=0}^{T} \gamma^t r(S_t, a_t) \tag{1}$$

γ is the discount factor for the future rewards.

The SageMaker BattleSnake [1] provides a framework to train the snake using Reinforcement Learning and customize the snake base on some heuristics methods as well. With the SageMaker you can select your RL policy and perform further optimization. After making modifications to your snake you can deploy it on the endpoint instance of the SageMaker and it is ready for use in the Battlesnake website. In Figure 2, we can see the runtime AI infrastructure of the AWS. For the RL part it uses the deep Q learning as we know that the states space of the Battlesnake is very big so a deep neural network will be needed. In deep Q-learning the input is an image which is the input state of the problem and the output is the Q values which based on the values the best action will be decided. In the Battlesnake problem we have 4 actions, therefore we only have 4 Q-values which correspond to snake moving Up, Down, Left and Right. Training a snake is also quite easy since the Neural Network is already defined. In our project, we modify some parameters of the training part (like gamma and learning rate) and also we add some heuristics to make the snake as best as possible. The heuristic part is an extra checking over the RL decision. We can check that whether the selected decision based on the Q-values predicted by the Neural Network is a good decision or not. For example the basic heuristic method is banning the bad actions which cause snake die. Thus if the best action based on the Q-values will result in a failure we omit that action and select the second best action based on the Q-values. These including the moving toward another snake or hitting the wall and hitting itself. Another important modification we added to snake is that snake tends to have the longest length but just be one size longer than the second longest snake. Having the largest length is a great opportunity since if they collide head to head the longer snake is the winner. Also making trap for other snakes will become easier. The rationale behind no to eat as much as food as it can, is that experiments show that having very long length compared to other snakes it is not good policy and the probability of hitting itself increases.

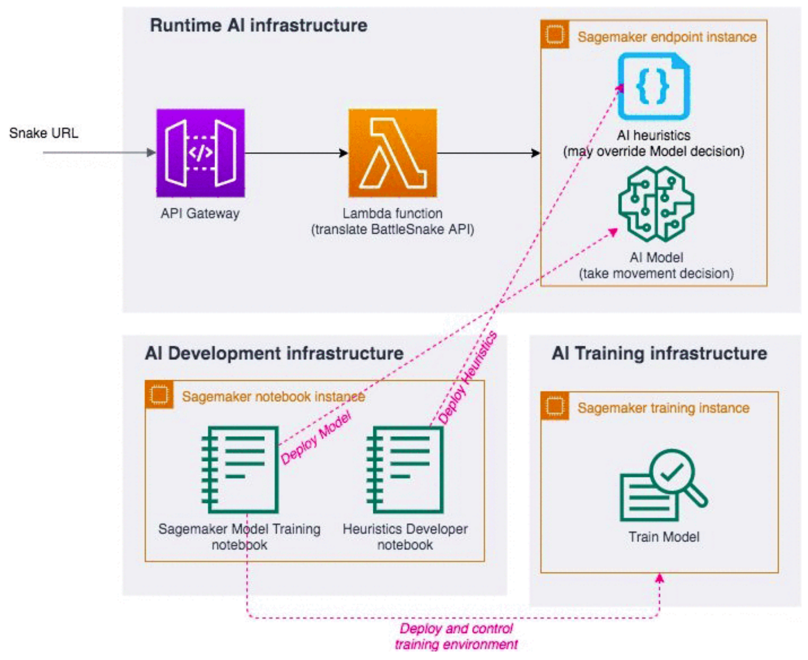

Runtime AI infrastructure
Sagemaker endpoint instance
Snake URL
API Gateway
Lambda function
(translate BattleSnake API)
AI heuristics
(may override Model decision)
AI Model
(take movement decision)
AI Development infrastructure
Sagemaker notebook instance
Deploy Model
Deploy Heuristics
Sagemaker Model Training notebook
Heuristics Developer notebook
AI Training infrastructure
Sagemaker training instance
Train Model
Deploy and control training environment

Figure 2: Sagemaker Infrastructure

3.2 Monte Carlo Tree Search

The next approach we considered exploring was a simple Monte Carlo Tree Search (MCTS) through the Battlesnake game tree [3]. Within this tree, each node would represent the state of the game at each turn and encode information such as the position of food, position of enemy snakes and current direction. Then, this heuristic search algorithm iteratively applies a four-step process until it reaches the solution and determines the optimal policy. The first step in this process is called Selection. In this phase, the algorithm traverses from the root node and chooses a child node using a pre-defined strategy typically given as the following equation:

$$S_i = x_i + C\sqrt{\frac{\ln t}{n_i}} \tag{2}$$

where S_i is the value of the node, x_i is the empirical mean of the node, C is a constant and t is the total number of simulations.
Once the node is selected, that node's children is expanded to discover all possible states arising from that action. This second step is aptly named Expansion. The third step is called Simulation and in this step, the algorithm conducts multiple Monte Carlo simulations to assign a value to all the child nodes that were recently expanded. Finally, in the last step, the algorithm performs back-propagation back to the root node and repeats this whole process again. See Figure 3 for an overview of this search algorithm.

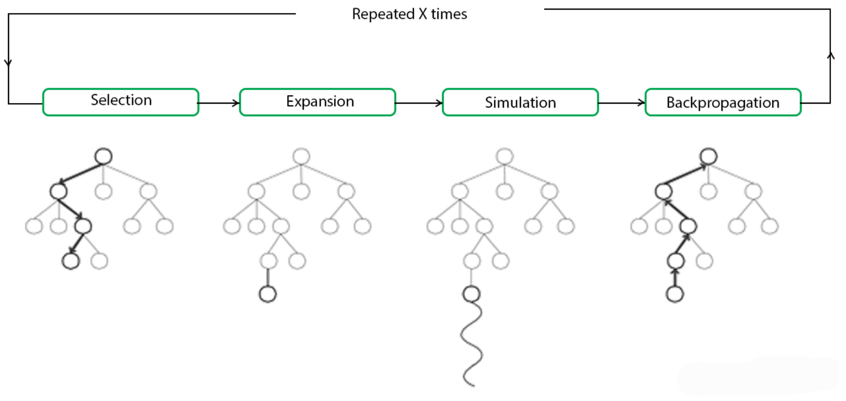


Figure 3: An Overview of the Monte Carlo Tree Search Algorithm [3]

3.3 Smart Moving Strategy Snake using Conditional Statements

The last approach we considered is to add some rules for snake moving. This approach has many rules that could let the snake avoid edges of the map, collisions with other snakes and itself, and paths that are closed to other snakes moving paths, spaces that are dangerous. And it has some features such as consider movements when other snakes are in collision range, find winning movement when other snakes have only one available movement, find available food while moving [2]. And basically, this approach has set those rules to make sure the snake could be the last existing snake in the game, though it may not have the most food compares with other snakes. However, as time goes by, the snake will grow up and one strategy to win the battlesnake is to survive until the end by avoiding other snakes.

4 Experiments and Results

4.1 AlphaZero Snake using Monte Carlo Tree Search

We trained the AlphaZero [4] on google TPUs using colab with some changes in hyper parameters such as depth and breadth of Monte-Carlo Tree Search for the training becomes faster. After a few hours of training the model reached an acceptable performance but we did not use this for our final model due to following two facts. First, the model is not very efficient for instance the input shape is 21 *21 which is unnecessary. The other reason is that they do not provide any interface for Battlesnake webesite interface and the report on their GitHub repository was not completely clear. Therefore, we could not write a code to change the input and output format to standard Battlesnake format.

4.2 Smart Moving Strategy Snake using Conditional Statements

We have tried Conditional Statements snake called "Snaky97". This snake has referred to some features and moving rules described in the methodology part. Unlike the AlphaZero Snake, it does not use any reinforcement learning or monte carlo tree search algorithms. Therefore, it does not have any training time, and this approach tried run pretty fast and can even run on a low-power hardware. The performance of this approach is good and we have done multiple games, "Snaky97" usually survival in the end , and win the game.

4.3 Reinforcement Learning

As mentioned earlier, we created two snakes: one using the RL and another one with using just conditional statements but with predicting future and preferring some moves called smart moves over some others. Also we used a public snake in the battlesnake website called "amazingly snakey". The

training metrics	Episode 33
best snake episode len max	351
Forbidden move max	4
Forbidden move mean	0.69
Forbidden move min	0
Killed another snake max	4
Killed another snake mean	0.54
Killed another snake min	0
Snake hit body max	5
Snake hit body mean	1.32
Snake hit body min	0
Snake hit wall max	2
Snake hit wall mean	0.12
Snake hit wall min	0
Snake was eaten max	4
Snake was eaten mean	1.12

Table 1: Trained snake's State after 33 episodes of training

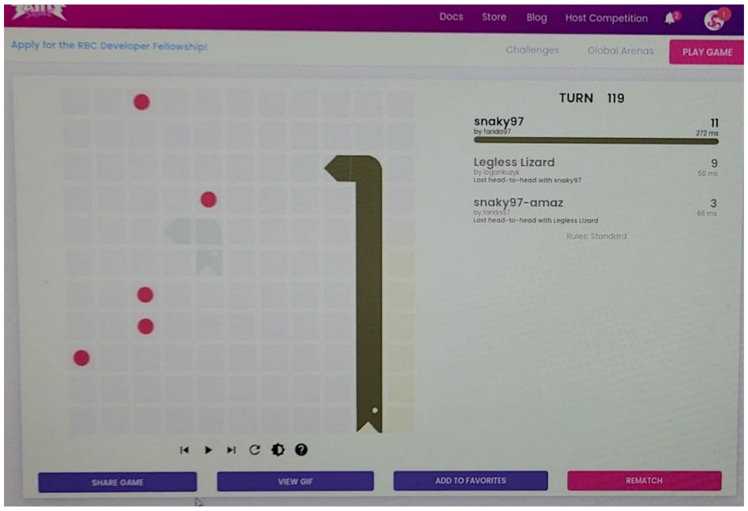
trained snaked is called "Snaky97-amaz" and the Smart-Moving Strategy Snake using Conditional Statements is called "Snaky97".
The training process for 30 episodes took about 2 hours in the table 1 we can see the training result of the "Snaky97-amaz". In each episode the number of Forbidden should decrease and we should see an increase in number of timestamps(number of moves for a game). As we disucssed in the methodology section "Snaky97-amaz" tries to have longer length in comparison to other snakes. The videos of the battles are attached to the report file. One surprising fact is that training model is not as good as the "Snaky97" and in figure 4 we can see the result in battle with "legless lizard" which is very powerful snake. The reasons behind this is the length of training due to the not enough computation resources. Also the heuristics for the trained model can be further expand for example by selecting the next move based on a policy between the trained result and the heuristic itself. Thus, training itself might not lead to the best result and some heuristic can make it very powerful.

4.4 Decision Trees

As a bonus experiment, we also tried using a decision tree to implement the snake behaviours because ultimately, the snake just needs to make one of four decisions at each turn: left, right, up, and down. We could theoretically encode each state into a large number of features but the size of a dataset to train a working model would need to be exponentially large. For example, there would be features to represent each square in the grid and signal whether that space contained food, an enemy snake, or an empty square. Then there would be a target feature that states whether this led to a winning outcome and/or the direction the snake headed in. Due to this complexity and perhaps due to copyright and privacy concerns, there were no comparable datasets available on successful Battlesnake games. So, instead we manually collected data from a short 11-turn game played by Kevin Durcant's snake who placed in the top 10 on the global leaderboards. This dataset consisted of 11 examples and 122 features where 121 of those features represents each cell in the 11x11 grid and the last feature is the resulting direction the snake took. The trained decision tree did not perform well and the validation accuracy was 50% likely because of the extremely sparse and small dataset used.

5 Conclusion and Future Work

Overall, both trained snakes "Snaky97-amaz" and "Snaky97" performs well. However, "Snaky97-amaz" does not perform better than "Snaky97". This is surprising because "Snaky97-amaz" uses reinforcement learning method, and spend two hours to train the model, but "Snaky97" even do not need any training. However, "Snaky97-amaz" could have more improvement if we have more computation resources and do more modification on heuristics. In the future, we could combine Smart-Moving Strategy Snake using Conditional Statements approach with Reinforcement Learning via Amazon Sagemaker.



TURN 119

snaky97 by farida97 11 272 ms

Legless Lizard by logankuzyk 9 50 ms
Lost head-to-head with snaky97

snaky97-amaz by farida97 3 66 ms
Lost head-to-head with Legless Lizard

Rules: Standard

Figure 4: Snaky 97 vs legless lizard. The legless is the yellow snake in the right of the picture

In the future, we would like to implement some of our failed approaches and continue work on the AlphaZero snake using the Monte Carlo Tree Search. And we would like to use genetic algorithms for hyper-parameter optimizations, experiment with more hyperparameters/neural network structures, and usually these algorithms have high requirement for hardware. Another limitation for this project is there are not many related works. Battlesnake is pretty new and even the demo snake python file in Battlesnake offical website changes quickly. There is still a long way to go about battlesnake.

References

[1] Chung, J., Luo, A., Raffin, X., & Perry, S. (2020). Battlesnake Challenge: A Multi-agent Reinforcement Learning Playground with Human-in-the-loop. arXiv preprint arXiv:2007.10504.

[2] Altersaddle. (n.d.). Altersaddle/untimely-neglected-wearable: A simple Battlesnake written in Python. Retrieved from https://github.com/altersaddle/untimely-neglected-wearable

[3] “ML: Monte Carlo Tree Search (MCTS).” GeeksforGeeks, 14 Jan. 2019, www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/.

[4] Fool-Yang. “Fool-Yang/AlphaSnake-Zero: A Reinforcement Learning AI That Plays the Synchronous Strategy Game Battlesnake Based on the ALPHAGO ZERO’S Algorithms.” GitHub, github.com/Fool-Yang/AlphaSnake-Zero.

[5] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of go without human knowledge. nature, 550(7676), 354-359.