Using Reinforcement Learning for Inferring Board Game Strategies

EE496 Final Report
Fall 2024

Department of Electrical and Computer Engineering
University of Hawai'i at Manoa

Aaron Ramos

December 23, 2024

Faculty Advisor: Professor Narayana Prasad Santhanam

1 Introduction

The main objective of this project is to create a player model that can learn various strategies for the board game, *Huligutta*, also known as *Goats and Tigers* by employing Reinforcement Learning (RL). To complete this task, we aim to develop player models that reach 100% win rates. By doing so, this project demonstrates the ability to use reinforcement learning for higher decision-making and can be used for more meaningful applications. For example, renewable energy systems can be optimized such that storage, management, and energy collection are completed optimally through intelligent decision-making, driven by RL.

Similar projects handling related topics in RL and artificial intelligence include Google Deepmind's **AlphaGo**, **AlphaZero**, **MuZero** and open-source projects like **OpenSpiel**. The projects construct player models capable of playing different board games by integrating deep neural networks (DNN) with various learning algorithms and numerous test environments; all of which are related to a strong reinforcement learning model. However, these projects have not implemented the Goats and Tigers game with their RL models, demonstrating that our work can be considered original due to its distinct rules, unorthodox map, and overarching goal.

This project was built using concepts taught in computer engineering courses (e.g., basic coding, object-oriented programming, logic of linear algebra and probabilities): **EE160**, **EE260**, **EE205**, **MATH307/EE345**, and **EE342**. These classes, experience in **Python** and **Github**, and access to the university's high performace computing cluster **(Koa HPC)** are foundational for understanding and contributing to this project.

2 Background

Huligutta is a strategic board game played by two players. There are 2 types of pieces that each player takes unique control of: *Goats* and *Tigers*.

The game is played on a map of empty nodes,

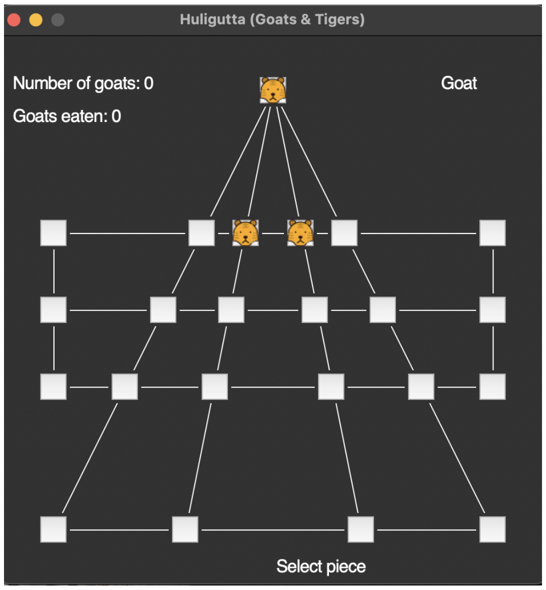

Figure 1: Initial game state configuration

best described as a triangle intersecting a wide rectangular grid. Two nodes can be identified as *adjacent* if they are next to each other and connected according to the line displayed on the map.

2.1 Gameplay

The game begins with 3 Tigers already placed on the board, following the configuration shown in Fig. 1. This is the maximum number of Tigers allowed. The Goat player must engage in a "place phase", where Goats can be placed on empty nodes until the maximum amount (15) is reached. At this point, the Goat player enters the "move phase", where additional goat pieces can no longer be placed but existing goat pieces can be moved instead. Players take turns moving their pieces to achieve their piece-specific win condition. Both Goats and Tigers are only allowed to move adjacently and only to empty nodes.

2.1.1 Win Conditions

Tigers win if they *capture* 6 Goats. A Tiger can capture or "eat" a Goat by "jumping across" them onto a free space, removing the Goat. Tigers are not able to capture Goats if the space to be jumped to is occupied. Note that Fig 2 demonstrates a state where a Tiger can capture the Goat above it after 5

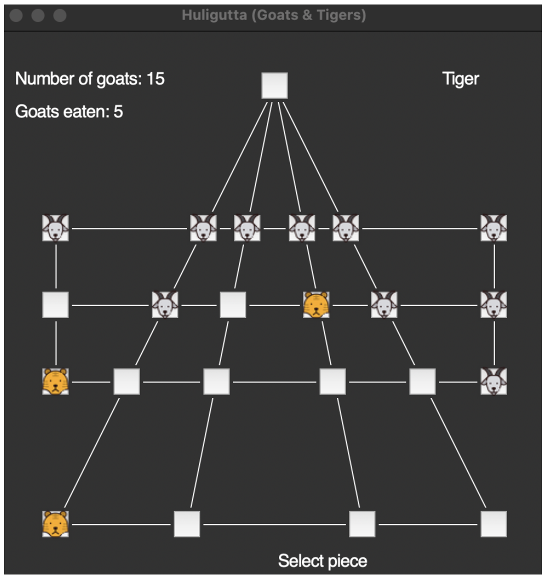

Figure 2: Tiger win condition

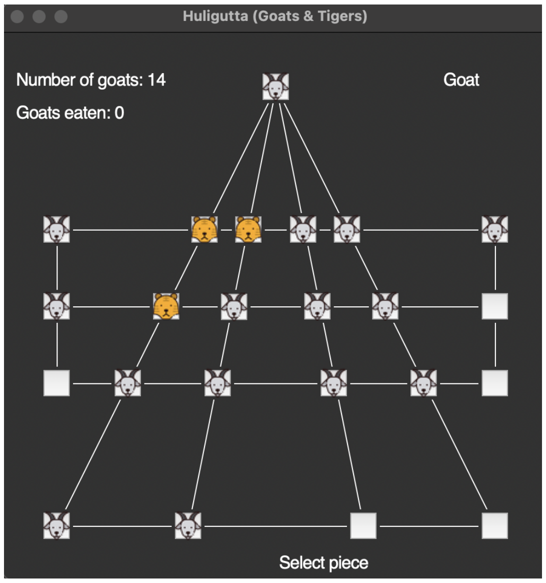

Figure 3: Goat win condition

other Goats have already been eaten, meaning that the Tiger wins after the next capture.

Goats win if they *stalemate* the Tigers, i.e. the tigers cannot move. This requires the game to reach a state in which all 3 Tigers on the board are cornered by the Goats and cannot capture nor move to an empty node, as shown in Fig. 3.

3 Design Tools

Assets and tools used include **Python**, **Visual Studio Code** (or any code editor), **GitHub**, the **Koa HPC Cluster**, and **OpenSpiel**. The initial hard-coded game of Goats and Tigers was developed by Professor Narayana Prasad Santhanam using Python and managed using GitHub.

This semester, project developments occurred independently within the students' personal devices due to the new design approach to further the current RL implementation using OpenSpiel. OpenSpiel is an application programming interface (API) with many reinforcement learning algorithms and environments that can be made applicable to Goats and Tigers. This also meant that the capable Koa HPC Cluster would be used more often for version control and project management.

4 Design Methodology

Prior models focused on winning as the Goat player, with the Tiger player coded to follow a *greedy* strategy; it moves 'randomly' and captures Goats whenever possible.

This methodology used value iteration to train player models that would "look ahead" and perform optimal moves based on analysis of future possible game states. The most recent iteration of this is a Goat model, 'twoStepGoat', that can observe the game 2 steps into the future.

However, expanding this predictive capability to look at ahead at a larger scale is not sustainable, i.e., evaluating a vast quantity of game states would be computationally expensive, undermining the purpose of the project. So, a new approach is being pursued using the OpenSpiel API. By porting Goats and Tigers into the OpenSpiel API, we can analyze and use existing reinforcement learning algorithms and environments to develop player models that explore further into the future while decreasing the required cost of computing power. Another benefit of using OpenSpiel is that a model of the Tiger player can now be trained and developed in addition to the Goat model.

4.1 Reinforcement Learning

RL models are trained by receiving rewards and punishments, optimizing its choices to acquire more rewards. Here, rewards are based on game states and piece actions. State set $\underline{\mathbf{S}}$ consists of all game states, accounting for piece positions, goats eaten, and phase of the game (place/move). The action set $\underline{\mathbf{A}}$ is a set of all possible actions of all pieces on the board. Thus, the reward function $R(s,a)$, determines a positive reward value (1) to a winning action **a** from a state **s**, a negative value (-1) to losing actions, and a neutral value (0) for all other actions and states.

4.1.1 Value Iteration

For the Goat model to reach winning moves, the stochastic policy $\pi(a|s)$ is used. This is a conditional distribution on the set of actions possible to take from a given state. To find preferred states, value iteration is used for an initial game state s_0 before the first goat is placed. Applying the stochastic policy on the initial state shows the value of a state to be $V^{\pi(s_0)} = P(win|S) - P(lose|S)$, an average of future rewards over all game states branching from the initial state. The maximized amount of rewards returned is $\pi^* = argmaxV^{\pi(s_0)}$, with the ideal state value being V^{π^*}. Rather than directly computing V^{π^*}, we start with a heuristic V_0, which implicitly uses the policy based on the state reached by action **a** $s(a)$.

$$\pi(a|s) = softmaxV(s(a)) \tag{1}$$

Applying the softmax function yields probabilities assigned to each move based on the possible reward, seen in Equation 1. Then, the Goat model can choose a move based on a 1-step look-ahead.

4.1.2 2-Step Look-ahead

A new heuristic V_{i+1} is derived from the 1-step look-ahead and the previous process is repeated for all next possible moves. The state value function now contains two inputs for the next immediate action and the subsequent actions. Summing these together and applying the natural logarithm gives Equation 2, a new probability value for a 2-step look-ahead. Note that $V_i \to V^{\pi^*}$ as $i \to \infty$, where looking more steps ahead causes the state value to approach its ideal. The Goat model now wins against a greedy Tiger 95% of the time.

$$V_{i+1}(s) = ln \sum_{a_1,a_2} e^{V(s(a_1,a_2))} \tag{2}$$

4.2 OpenSpiel Results

Next, 2 groups were formed: **porters** and **analysts**. Porters must interface a version of the game compatible with OpenSpiel, while analysts sift through the API to understand observer functions and how other game models have been trained.

Thus far, porters made a copy of tic-tac-toe on OpenSpiel. Its code will be altered so that it plays Goats and Tigers, while configuring API functionality specific to the game. Analysts reviewed RL algorithms for games like breakthrough and are becoming familiar with the OpenSpiel environment.

5 Discussion

5.1 Constraints and Limitations

Previous versions of the Goat player used hard-coded rules for RL. Increasing the scope of the look-ahead leads to expensive computational costs. For OpenSpiel, the unique shape and logic of the map poses a problem for porting, since we need to ensure that pieces cannot illegally traverse.

5.2 Future Work and Improvements

We are to successfully port Goats and tigers into OpenSpiel. This will reduce computing costs. Once successful, the next step would be to continuously train Goat and Tiger models that can employ a variety of strategies to a successful degree.

6 Conclusion

The EE496 project uses RL concepts like value iteration to train models competent at Goats and tigers. Its rules and design tools used were mentioned. The current Goat model can look 2 steps ahead with a 95% winrate. Progress of interfacing with the OpenSpiel API, limitations, and improvments were described.