HUBERT, Our FreeCiv AI

Michael Arcidiacono, Joel Joseph Dominic, Richard Akira Heru

Friday 7th December, 2018

Abstract

FreeCiv is an open-source alternative to *Civilization*, which is a turn-based strategy game that allows the player to control an empire by building cities, constructing and moving units, and engaging in diplomacy and war with other nations. We build an artificial intelligence (AI) nicknamed HUBERT to play this game. We use a modified SARSA algorithm with linear approximation of the Q function. We have also tried training HUBERT using a memetic algorithm.

1 Introduction

The *Civilization* video game series is a collection of turn-based strategy games that allow the player to control an empire by building cities, constructing and moving units, and engaging in diplomacy and war with other nations. There are several different ways of winning the game such as through military dominion (i.e. by taking over all the cities of other empires). We use an open-source alternative to *Civilization*, FreeCiv and focus on getting the highest score for this agent. A screenshot of the game in progress can be seen in Figure 1.

We model this game as a Markov Decision Process (MDP) with states, actions and rewards. In our model, the states are quantified as a β vector per city containing the total number of cities, total amount of gold, tax, science, luxury, gold per turn, turn number, the number of technologies, city population, city surplus, city unhappiness level, and city food.

At any given turn, the player can choose to build cities, build units, move units, declare war, trade resources, research technology, and engage in many other possible actions. For our project, our agent decides specifically what unit or building each city will build at each turn while the rest of the actions such as movement of units or diplomacy etc. are controlled by the default AI.

According to FreeCiv Wiki Score page, the score in FreeCiv is computed using the formula

$$\text{SCORE} = c + 2t + 5f + 5w + s$$

where c, t, f, w, s are the number of citizens, the number of technologies, the number of future technologies, the number of wonders and the spaceship score respectively. The spaceship score s is given by the formula

$$s = \text{success rate} \times 0.01 \times \text{number of citizens on the spaceship.}$$

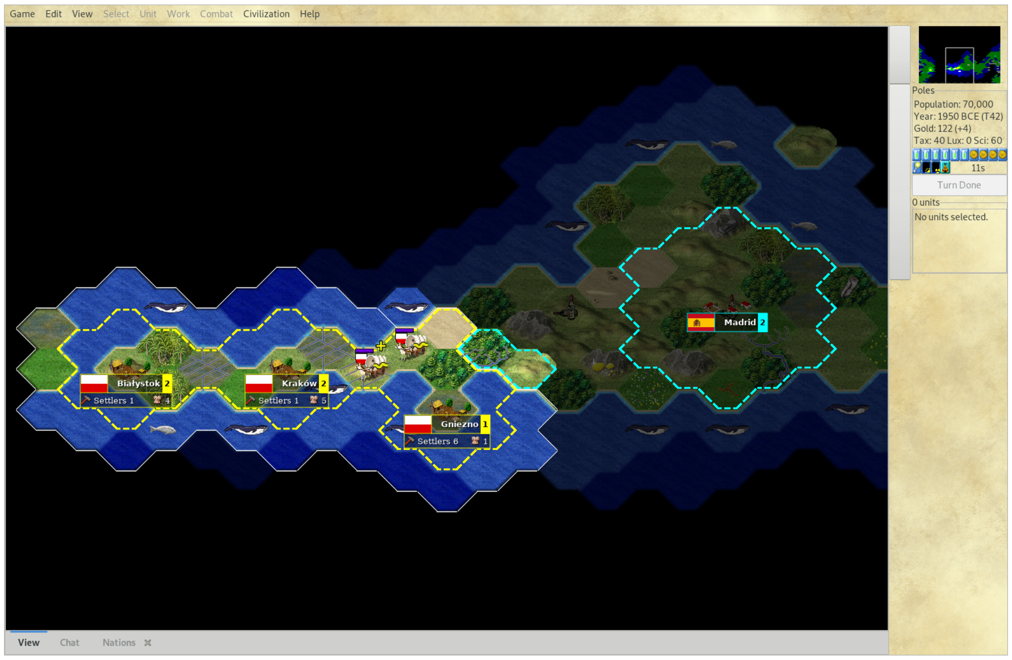

Figure 1: Screenshot of the game

In our model, we use this score to compute the reward.

We build an artificial intelligence (AI) nicknamed HUBERT that can play FreeCiv and win by maximizing its score. FreeCiv has its own AIs - easy, medium, hard. We sometimes train HUBERT against these AIs.

2 Related Works

There have been many attempts at building reinforcement learning models for turn based (or real-time) strategy games. A game that is closely related to Civilization is StarCraft II for which there has been an annual AIIDE (AI for Interactive Digital Entertainment) StarCraft Competition that pits bots against each other. Papers like [6] show the difficulty of building such models especially when faced with such a partially observable, large state and action space that involves multiple agents.

More specifically, there have also been numerous attempts to build AIs for FreeCiv and Civilization II and IV. Branavan employed Non-linear Monte Carlo Search for models on FreeCiv in [3] to find optimal actions for units. Additionally, Amato uses Q-learning and Dyna-Q learning in [1] to choose high-level strategy directives of the agent. Wender also uses Q-learning, Q(λ), one-step SARSA and SARSA(λ) in [7] to explore optimal city site selection for new cities. Arnold also explored city site selection in [2] with genetic algorithms. Arago's Hiro AI has in fact, been able to beat 80 percent of human opponents in FreeCiv [5].

3 Approaches

3.1 SARSA with Linear Approximation

The first model that we implemented uses SARSA algorithm with linear approximation. Instead of using the typical update rule for SARSA,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)),$$

we approximate the state action value $Q(s, a)$ by the dot product of θ and $\beta(s, a)$, where θ is a weight vector and β is a function that takes as input a state s and an action a and returns a vector, and update the weight vector θ instead using this rule,

$$\theta \leftarrow \theta + \alpha(r_t + \gamma\theta^T\beta(s_{t+1}, a_{t+1}) - \theta^T\beta(s_t, a_t))\beta(s_t, a_t),$$

where α, γ, s_t and a_t denote the learning rate, the discount factor, the state and action at time t respectively. The pseudocode for this algorithm 1 is a modified version of Algorithm 5.3 and Algorithm 5.5 found in [4].

Algorithm 1 SARSA with Linear Approximation

```
1: function SARSA with Linear Approximation
2:     $t \leftarrow 0$
3:     $s_0 \leftarrow$ initial state
4:     Initialize $Q$
5:     loop
6:         Choose action $a_t$ based on $\theta_a^T\beta(s_t)$ and some exploration strategy
7:         Observe new state $s_{t+1}$ and reward $r_t$
8:         $\theta \leftarrow \theta + \alpha(r_t + \gamma\theta^T\beta(s_{t+1}, a_{t+1}) - \theta^T\beta(s_t, a_t))\beta(s_t, a_t)$
9:         $t \leftarrow t + 1$
10:    end loop
11: end function
```

3.2 Memetic Algorithm

We then try to improve our algorithm by combining the previous approach with a population-based approach. In each round of the simulation, we play a few HUBERTs against each other, sometimes with some default AIs, in a battlefield. Each HUBERT is going to update its weight vector θ based on Algorithm 1. At the end of the simulation, the HUBERT with the highest score is deemed as the winner and its weight vector is going to be used as the starting weight vectors of the HUBERTs in the next round of simulation. Some of the HUBERTs are mutated with some small probability by adding some noise to the values in the vector θ before the next simulation begins. In short, our implementation of the memetic algorithm selects only the fittest individual to be trained at each iteration of the simulation.

3.3 Change in Reward Function, States and Environment

Besides changing the models for HUBERT, we also changed the reward function several times and the properties of the state. We also switched between different number and types of opponents to make the problem more tractable.

3.3.1 Relative Score

Another adjustment we attempted was to use the gain in score after each turn relative to the gain in score of another player with the highest score as our reward instead of the absolute gain of the player's score after each turn as the reward.

3.3.2 Delayed Reward

We update θ based on the reward only after 75 turns, where the reward for each action is the score at the end of these 75 turns.

3.3.3 Increase variables in State

Initially, our state just consisted of HUBERT's number of cities, amount of gold, gold per turn and city population. However, we added more variables to the state so that it can approximate the actual state better.

4 Results and Analysis

4.1 Intermediate Results

Our memetic algorithm and SARSA models performed poorly when our reward function was the increase in score a certain number of turns after a unit or building was built. These models in fact, performed worse than our random opponent. Changing the reward function to the relative function above did not help the performance significantly either. Upon these results, we decided to simplify the reward function and switch back to the SARSA model.

4.2 Final results

To evaluate the performance of our AI, HUBERT, we test HUBERT by playing it against various opponents: Random, Easy AI, Medium AI, and Hard AI. Random chooses a random action at each turn whereas the other AIs are FreeCiv's default AIs. For each of these opponents, HUBERT plays against it 100 times and we record the number of times it finishes with a higher score than the opponent. The results are tabulated in Table 1. As we can see from Table 1, HUBERT's performance after some training is better than Random, comparable to Easy AI and Medium AI, and slightly worse than Hard AI.

Figure 2 shows the running averages of scores of various players, HUBERT, Hard AI, and Random over multiple training iterations. As we can see from the graph, HUBERT is learning to score higher over time. We can also see that HUBERT's performance and

Table 1: HUBERT's performance against various opponents

Opponent	Winning rate
Random	70%
Easy AI	54%
Medium AI	57%
Hard AI	40%

Random's performance are comparable at the beginning, both worse than Hard AI's performance. However, as HUBERT learns, it becomes better than Random, even beating Hard AI sometimes.

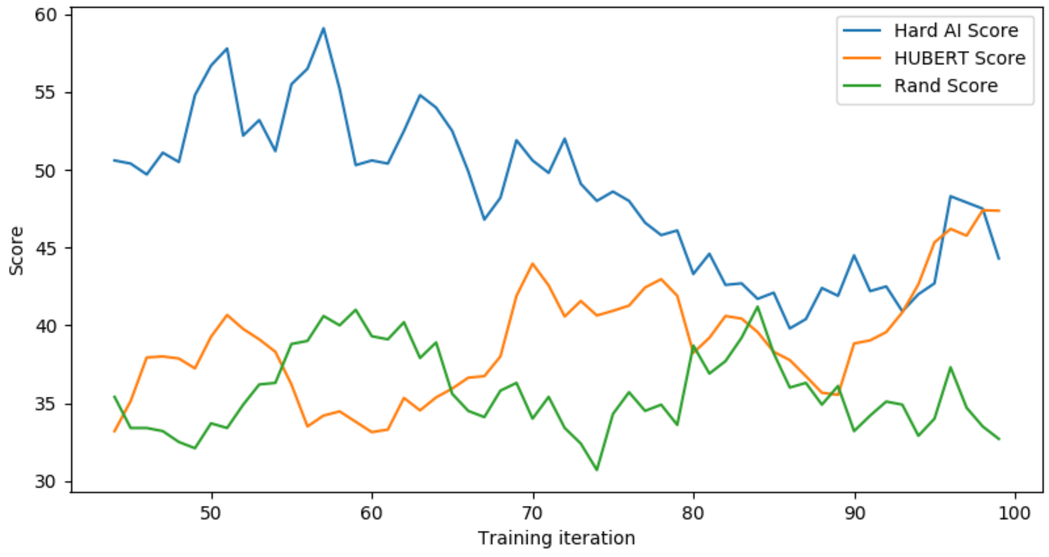

Figure 2: Graph of running averages of scores over training iterations

4.3 Interpretation of results

Upon closer examination of the vector θ, which is used to determine the best action at a given state, we realize that after some training, the values seem reasonable. A notable example is the θ value associated with choosing to build the unit settlers. Settlers can create new cities and producing them is generally a good strategy early in the game. Later in the game, however, producing them is discouraged because there is likely not much land left to colonize. Additionally, cities can only produce settlers quickly if they have a certain amount of population available. The θ values for producing settlers reflect these heuristics; the bias for producing settlers is large (48.79) while the weights associated with turn number (-0.83) and city population (-0.027) are negative.

This shows that our training algorithm is actually learning strategies that a human may find reasonable.

5 Conclusion and Future Work

In conclusion, we have successfully trained HUBERT to play FreeCiv satisfactorily. While algorithms without delayed rewards generally do not perform well given the nature of the game as most actions take time to produce their rewards, by using delayed reward scheme, we manage to train HUBERT to achieve a performance that is better than Random, comparable to Easy AI and Medium AI, and slightly worse than Hard AI. After all, HUBERT is not the Highly Unreliable Bot Extremely Resistant to Training we thought it would be. In fact, HUBERT even beat us when we played against it for a limited number of turns.[1]

This project can be extended in many directions. If we had more training time, we could have used a more complicated genetic algorithm in our training. This includes training more individuals by performing local search using them and sampling from the top few individuals, instead of just starting from the best individual after each iteration, performing genetic crossover of top performing individuals in addition to individual mutation that is already included in the model we have tried. However, we noted that genetic algorithm can take a very long time to train and produce good results. Furthermore, we could also expand our search state space, consider more potential actions and use other delayed reward schemes to tackle this problem.

6 Contribution of Group Members to the Project

Michael wrote all the code. Richard and Joel contributed ideas[2] and wrote most of the report. Richard inserted relevant graph, table, algorithm, and other figures to the report as well.

References

[1] Christopher Amato and Guy Shani. High-level reinforcement learning in strategy games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 75–82. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[2] Felix Arnold, Bryan Horvat, and Albert Sacks. Freeciv learner: A machine learning project utilizing genetic algorithms. *Georgia Institute of Technology, Atlanta*, 2004.

[3] SRK Branavan, David Silver, and Regina Barzilay. Non-linear monte-carlo search in civilization ii. In *IJCAI*, pages 2404–2410, 2011.

[4] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application.* MIT press, 2015.

[5] Jess Vilvestre. Watch arago's hiro beat out human gamers in freeciv, Dec 2016.

[6] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. 2017.

[7] Stefan Wender. *Integrating Reinforcement Learning into Strategy Games.* PhD thesis, University of Auckland, 2009.

[1] But we believe we can beat HUBERT over the long run :)

[2] and nourished Michael physically and spiritually