Practical Reinforcement Learning: Teaching AI to Play Conquest of Gaul

Noah Burkhardt, Courtland Climer, Ares Shackleford, Jeffrey Wong
University of Maryland

Abstract

Reinforcement learning has become an increasingly popular subfield of machine learning as of late. The techniques of this subfield have been applied to an array of disciplines with significant recent success in the optimization of video game strategies. This led us to ask if these RL techniques could be used to improve the performance of agents in our custom game and if these same techniques could be used to argue about the fairness of said game. We used Python and OpenAI to create a complex simulator and testing apparatus for our game in order to answer these questions. We used a common analysis technique in RL of getting efficiency of the system by recording the number of wins out of total games simulated with the learned models. We found that ACKTR with 10 million iterations had the highest effectiveness against random and easy enemies. With our results, we have shown that reinforcement learning algorithms, especially ACKTR, can be used to optimize the performance of agents within our game. Our results were also used to argue that the game's ruleset had imbalanced features.

Introduction

Over the past couple of decades, there has been a surge in the development and application of machine learning techniques. These applications include robotics, education, finances, and even video games. Modifications of previous techniques and the improvement of CPU/GPU technology has led to the invention of many new machine learning techniques in recent years. These techniques, largely reinforcement learning techniques, have allowed for the optimization of video game strategies for a plethora of video games. This raises our research questions. First, can reinforcement learning techniques be used to improve the performance of an agent group in the Conquest of Gaul video game? Second, can reinforcement learning techniques be used to find imbalances in the Conquest of Gaul through hyperparameter tuning and the altering of rewards?

We thought this would be an interesting project because we wanted to see how well these techniques could be used in reverse of their usual application. We also thought it would be a lot of fun to build a small simulation suite for these kinds of problems in general and see if we could get an algorithm to train on a custom game of ours. We think this project has some interesting applications in a variety of fields beyond the scope of this project including sociology and economics.

Of course, there have been dozens of other studies and projects that utilize modern reinforcement learning (RL) techniques to optimize performance in a variety of games. Perhaps the most famous examples of these applications are the Lample and Chaplot use of RL methods

to play through the first-person shooter Doom and the Arcade Learning Environment project for learning thousands of Atari games. Simple examples of these techniques have been applied to other classic video games like Snake and Super Mario World and these can easily be found online in a multitude of tutorials. The fundamental challenge of this area of study is to find an effective and often general strategy that allows an agent to perform well in a game, however well is defined by that game's particular ruleset. This often involves the development of game-specific, finely-tuned RL models. The performance of these techniques often heavily relies on the complexity of the enemy AI in the game. (Shao, Tang, et al). Going beyond this fundamental challenge, many of these works have found that they were able to optimize algorithm performance in the games to be comparable to or, in some cases, better than human player performance. For example, the Lample and Chaplot paper found that their learned agent Arnold outperformed human players in both singleplayer and multiplayer scenarios using an Action-Navigation architecture (Lample and Chaplot). Typically, these past works have used deep-Q learning or state-action-reward-state-action algorithms. For example, Lample and Chaplot use a variant of a deep-Q network that utilizes a recurrent neural network on top of the deep-Q network (Lample and Chaplot). Even with that being said, there are still countless RL algorithms that have been tried and tested for video game optimization.

Many studies have also shown just how novel RL models can behave. As stated by ML engineer Mauro Comi, "Sometimes, reinforcement learning agents outsmart us, presenting flaws in our strategy that we did not anticipate (Comi)". This hints at the ability of reinforcement learning as a mechanism for finding imbalanced features and exploits in video games, which we will soon discuss as one of our contributions. It also shows how these techniques can often prove to be unpredictable. Of course, significant work has been done in trying to make these techniques more predictable, but that work lends itself more to XAI and is less relevant for our purposes. Not only have these results proven to be novel, many works also produce models that work on other games and environments with little-to-no tuning. An example of this is the use of the ALE framework for Atari games for a score of mobile games including Flappy Birds and Subway Surfers (Lieber). While these applications certainly did not produce the same high levels of success as when applied to the Atari games, it shows how well generalized these algorithms and techniques can be.

While these methods have certainly shown success in performance and great improvement, this paper has seen little application of these techniques as an evaluation platform for how fair the ruleset of a particular game is. The balanced nature of a video game, especially that of strategy-based games, is one of the key sources of the entertainment and value of a game. Dynamic games should not feature "easy solution" strategies that make them trivial to perform well in as the challenge of the game is what brings appeal. One also doesn't want the game to be unwinnable and have no viable strategies to utilize. As well, this paper is unique in testing RL strategies on a video game that has never been tested before. This paper then accomplishes two

goals: testing the performance viability of RL methods on the Conquest of Gaul military strategy game and testing how well these methods can be used to detect the fairness of the game.
This leads us to the following two hypotheses:

1. Reinforcement learning techniques can be used to improve the performance of an agent group in the Conquest of Gaul video game.
2. Reinforcement learning techniques can be used to find imbalances in the Conquest of Gaul through hyperparameter tuning and the altering of rewards.

In order to achieve this goal, we devised and implemented a roadmap that was structured as follows:

1. Define and implement the Conquest of Gaul ruleset
2. Build random control agents for baseline comparison of performance
3. Build and test an agent with a reinforcement algorithm equipped
4. Alter rules and hyperparameters to test fairness of game

The remainder of our paper describes the methods we used to construct the Conquest of Gaul and test the performance of reinforcement learning models on it. It then provides the results of the study and wraps up with a discussion of these results.

Methods

We used Python to implement our game and used the Stable Baselines OpenAI library for adding and testing different reinforcement learning methods. To construct our testing environment and connect the game to the RL methods we utilized the Python package Simpy. Our test environment for our research questions was a custom-made military strategy game titled Conquest of Gaul. The base mechanics of the game are:

- There are two teams on a grid of a fixed size, each consisting of one of each unit type (Infantry, Cavalry, and Ballista)
- A turn consists of all units on one team performing an action and then a turn is given to the other team, with turn order starting with “our” team
- The units have four possible actions: movement, attacking, gathering food, and passing
- Units can move to any adjacent tile, attack any adjacent tile or the tile they are currently on, gather, and pass their turn anywhere
- Units move in the following turn order: Cavalry → Infantry → Ballista
- The units perform against other types of units in a Rock-Paper-Scissors format, Infantry beats Ballista, Ballista beats Cavalry, Cavalry beats Infantry
- Fights cause a random amount of troops in the losing unit to be lost.
- Attrition occurs every turn that reduces food supplies on each team by the number of units on that team. Dropping to zero food eliminates all units on the foodless team.
- Seasons: Winter and NotWinter. Winter just amplifies food shortages.
- The game ends when one army wipes the other army out completely.

We developed a complex simulator that allowed us to efficiently run episodes of learning in our game world. The game, described by the rules above, would typically be run for up to 1000 turns; if there was no winner by that point, the game was deemed a tie and shut down. The simulator itself was designed to have three modes. Mode I was a visual mode, which allowed us to test the parameters of the game itself, and additionally to monitor the performance of the agents in the world. It ran a single iteration of the game, slowed down to a speed where we could analyze the decisions the agents were making. Mode II was designed as the testing mode of the simulator. It was passed in a total number of games to be simulated, and two policies (described in detail below) that the two teams would use to choose their actions. Mode III was developed as the training mode for the simulator. It was passed in a number of learning episodes and an agent (one of Cavalry, Infantry, or Ballista) to train, and a specific learning algorithm from the Stable Baselines package. It then produced a file containing the learned model, which could be imported as the learned policy described below.

In addition to the simulator itself, we developed a set of policies that could be plugged in modularly to give the agents different means for decision making. The first policy was a random policy, which simply chose a random move out of a list of allowed moves. The second policy, labelled ‘easy’, performed random moves, unless it perceived that its team was low on the food resource. In that case, it performed the GatherFood action to keep its team from losing troops due to attrition. The third policy, labelled ‘hard’, used the same food mechanic as the easy policy, but only performed the Attack action when it perceived that it was in range of an agent where attacking would be advantageous. The final policy, the learned policy, provided us with a bridge between the Stable Baselines reinforcement learning models and our simulator. It simply imported the output from the learning session, and queried the model whenever an action was to be made. As a point of reference, the ‘easy’ policy won about 70% of the time against the random policy, and the ‘hard’ policy won about 90% of the time against the random policy.

Due to the fact that we chose a reinforcement learning problem, we did not have a formal set of “training data”. Instead, our training data were simply the simulations we ran our learning algorithms against. While we could have gathered data, such as average and total rewards, during the learning sessions, we opted not to as it did not seem to add much value to our findings. We determined that the win percentage would be a sufficient measure of an agent’s performance. In a similar light, the training data was simply the set of simulations against which we ran our trained agents.

We chose to use pre-existing implementations of reinforcement algorithms instead of writing our own or tweaking existing algorithms. This decision was made so that we could spend more time on the game optimization and balance directions for our project. Specifically we trained algorithms based on Deep Q-learning Networks (DQN), Actor Critic methods (A2C, ACER, and ACKTR), and Policy Optimization methods (PPO1, TRPO).

This also served as our analysis method for determining the capabilities of these techniques/models and finding answers to our research questions. Our procedures all produced a

statistic we called effectiveness. Effectiveness against a certain policy was simply the approximate probability that a particular agent would win against another agent. Since the 'easy' policy described above won 70% of the time against the random policy, we say that its effectiveness compared to the random policy was 0.7. In the results section below, we show the effectiveness measure produced for a few noteworthy algorithms that we trained.

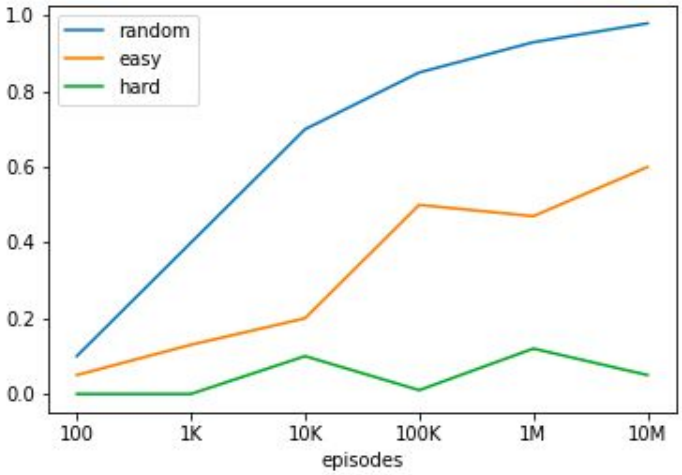
Results

Utilizing the techniques outlined in our methodology, our findings were as follows. We largely had poor results with the DQN. It often took a much larger amount of episodes to produce results and these results were often around a 0.0-0.05 effectiveness. Similarly with A2C, and PPO1, we found that it performed with less than 0.1 effectiveness. Our successes were typically with the ACKTR, ACER, and TRPO algorithms. Our most successful result against a random enemy was ACKTR with 10 million iterations at a 0.98 effectiveness. Against easy enemies, this was the same model but with only a 0.60 effectiveness. For hard enemies, that model but only trained with a million episodes performed the best at a 0.12 effectiveness.

The reward policy was: (+/-) 1000 win/loss, -1 illegal action, +1 GatherFood, +1 Move, +5 Attack, -1 Pass, unless specified otherwise. A condensed copy of our notable results is shown below.

Algorithm, Iters	Reward Policy	Eff. on Random	Eff. on Easy	Eff. on Hard
DQN, 1M	---	0.13	0	0
A2C, 1M	---	0.05	0	0
ACKTR, 1M	---	0.93	0.47	0.12
ACKTR, 10M	---	0.98	0.60	0.05
PPO1, 1M	---	0.09	0	0
TRPO, 1M	---	0.96	0.45	0.01
ACER, 1M	---	0.93	0.4	0.01
ACER, 1M	-100 illegal action, +100 move, +500 attack, -500 pass	0.92	0.43	0
ACER, 1M	-5 illegal action, +5 Move, + 5 Attack	0.08	0	0

Additionally, as shown in the graph below, as training episodes increased, effectiveness also increased. This particular graph is using the ACKTR algorithm with the default reward policy, and varying amounts of episodes. This relation is unsurprising, as the additional training episodes allowed the agents to perceive a greater portion of the total state space, and learn to make the most optimal actions possible.



Discussion

In summary, our results were quite clear. The amount of iterations used and the effectiveness of the model have a positive relationship with one another. The models with the lowest effectiveness were with DQN, A2C, and PPO1 with effectiveness less than 0.1 effectiveness. The models that performed the best were with the ACKTR, ACER, and TRPO algorithms where ACKTR under 10M iterations performed the best with up to .98 effectiveness. Unsurprising for all learned models, each one would tend to do the best against random enemies, then easy enemies, then hard enemies.

The findings show that, indeed, reinforcement learning can be used to improve the performance of the agents in Conquest of Gaul. This was shown by our results as some of the models tested increased performance by 48% (from 50% to 98%) when measuring against the control (random) performance against a random enemy. Even more, some of our models doubled performance (from 30% to 60%) when measuring against the control (random) performance against an easy enemy. Using these results, we can make some informed statements about how well balanced the game's ruleset actually was. The idea was to make the game similar to chess, a game where no one strategy always wins or wins a significantly disproportionate amount of the time. This was found to not be the case in our game. This can clearly be seen by the incredibly high level of success of the ACKTR model when one would expect something closer to a 50/50 for a truly "balanced" game regardless of which model was used. This definition of balanced/fair

only really applies to military strategy games where both teams are on complete equal footing and so this certainly does not invalidate the results or speak to the fairness of video games or games outside of the aforementioned category.

The significance of these results can be seen through the application to a new game/simulation system and the discussion of the impact of that result. In past works, they have applied these algorithms to other games. As seen by our results, we've extended this to not only include an application to our military strategy game, but also to include a discussion as to why this means our game was not fairly constructed in terms of military game strategies.

In this paper, we have defined, built, and optimized a military strategy game called Conquest of Gaul. We did this using several Python libraries, namely Simpy, OpenAI, and Stable Baselines. We tested a multitude of methods including Deep Q-learning Networks (DQN), Actor Critic methods (A2C, ACER, and ACKTR), and Policy Optimization methods (PPO1, TRPO). With our results, we have shown that reinforcement learning algorithms, especially ACKTR, can be used to optimize the performance of agents within our game. Furthermore, we have argued that this result, paired with the success of several other algorithms, means that the game's ruleset had imbalanced features.

Of course, there were multiple limitations in our work. For one, the amount of states in our video game made reinforcement learning more difficult. In reinforcement learning, the more states there are, the more that training needs to be done in order to gain any substantial new strategies. The amount of states that we had was further complicated by the fact that we had a dynamic action space. Depending on the state, some of the actions would not always be available for the player causing the training to become even harder to train. Another problem that we ran into was that since we needed to run through so many iterations to gain effective strategies, the amount of time some of the training algorithms took longer than any of us anticipated. In most cases, the wait was worth it because the policy that was generated as a result would produce high win rates, however, in several cases, the win rates were not nearly as promising despite the time put into training the algorithm. These two factors combined limited the amount of training we could accomplish with our environment. We initially planned to have even more complex actions and rules in the game, but that ended up being too complicated to write an enemy ruleset for and made the state space far too large for our machines, resulting in another limitation.

The goal of future research would be to address and resolve some of our limitations. The main limitations stated previously were about not having enough time to train the agent as much as we desired, however, now that we have the game developed and know which methods take longer than others we could alleviate that problem in future studies of this system. Also, future work could implement the more complex actions that we wanted to put into the game in the first place like building fortifications or having more complex interactions between allied units. Studying how these complex actions would change the optimal strategies of our game would be a very interesting future research topic. Future work could also focus on how to apply our findings to fields unrelated to video games like finance or sociology.

References

M. A. Samsuden, N. M. Diah and N. A. Rahman, "A Review Paper on Implementing Reinforcement Learning Technique in Optimising Games Performance," 2019 IEEE 9th International Conference on System Engineering and Technology (ICSET), Shah Alam, Malaysia, 2019, pp. 258-263, doi: 10.1109/ICSEngT.2019.8906400.

M. Comi, "How to teach AI to play Games: Deep Reinforcement Learning," Towards Data Science, 2018, https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a.

O. Lieber, "Reinforcement Learning for Mobile Games," Towards Data Science, 2019, https://towardsdatascience.com/reinforcement-learning-for-mobile-games-161a62926f7e.

G. Lample and D. S. Chaplot, "Playing FPS Games with Deep Reinforcement Learning," Carnegie Mellon University, 2017, https://www.cs.cmu.edu/~dchaplot/papers/aaai17_fps_games.pdf.

C. Watkins and P. Dayan, "Q-learning", Machine learning, 8(3-4):279–292, 1992.

R. Sutton and A. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.

V. Minh, K. Kavukcuoglu, D. Silver, et. al., "Playing Atari with Deep Reinforcement Learning," NIPS Deep Learning Workshop, 2013, https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.

K. Shao, Z. Tang, Y. Zhu, et. al., "A Survey of Deep Reinforcement Learning in Video Games," IEEE, 2019, https://arxiv.org/pdf/1912.10944.pdf.

Contribution Statement

Ares Contribution: Overall, my contribution was 20%. The focus of my work was spent researching the OpenAI gym toolkit in order to easily implement reinforcement learning algorithms. My goal was to create a workable base gym environment for a game. Once, I made a stable foundation with OpenAI, my team expanded on what was there in order to adjust it to our specific needs. I also validated some of the training algorithms that were conducted to make sure that our data was as accurate as possible. Alongside that I wrote some of the paper.

Court Contribution: Overall, my contribution was 30%. I came up with the idea for the project. I worked on designing the ruleset of the game. I assisted Noah in working on the enemy AI for the game. I worked with Ares to explore using the OpenAI gym and how it would need to be connected in order to work with our game and simulation environment. I tested several different RL algorithms and recorded results to be used in the Results section of our paper. I also wrote a significant amount of the paper including the background report as I spent a significant amount of time looking into which tools and algorithms would be good for testing (which involved finding other projects and papers that accomplished similar tasks). I also held meetings (digitally) with the various members of our group to discuss directions for the project, development options, and how progress was going.

Jeffrey Contribution: Overall, my contribution was 10%. I tested some algorithms and wrote nothing in the paper. I also aided Noah and the others in combining our simulator and the OpenAI gym environment.

Noah Contribution: Overall, my contribution was 40%. My focus was on writing the simulator that was used during training and testing. I determined the most optimal packages to use, designed the foundations of the enemy rulesets, and designed the graphical aspect of the simulator. Most importantly, all of the game and simulator logic was programmed in by me. The OpenAI gym environment that we plugged into the simulator was also designed and implemented by me. In addition, I ran most of the tests we used for the different parameters and reward policies, so that we could find the most optimal models to use, and did a substantial amount of research on which RL algorithms would best suit our environment. Finally, I wrote a couple of the sections of the paper, and worked with Court to help coordinate meetings with the team members.