Tag using AI: Teaching AI agents to play tag through machine and reinforcement learning

Melissa Tjong, Devyn Myers, Shane Staret, Kevin Doyle
Bucknell University
CSCI 357: AI & Cognitive Science
Professor Dancy
May 17, 2021

Introduction

In recent years, there has been an increase in the use of artificial intelligence in gaming. One use of artificial intelligence in this space is to create characters and experiences in video games that will mimic human behavior. Another application is to simulate the playing of a game and study the choices and strategy that an artificial intelligence agent may make. Through observing this behavior, one can see whether these agents are truly aware of their task and capable of learning in complex environments.

In order to examine the types of strategies that can be adopted during the simulation of the game, we created an intelligent system that simulated a game of tag played between two agents. The premise of tag is relatively simple; it involves one agent, the Tagger (also referred to as the Chaser), chasing another agent, the Runner, whose goal is to avoid the Tagger. When a game of tag is played between children in real life, the environment is an important factor that affects how the Tagger and Runner behave. To mimic this, we created an enclosed environment in Unity with obstacles that the agents have to take into account when navigating around the environment.

Background

There are not any notable implementations of a game of tag being played between two AIs, but there is a popular implementation of AI playing hide and seek created by OpenAI. This example is similar to our own problem, because both the Hider and the Runner avoid a Seeker, or Tagger, that is trying to find them. In OpenAI's example, the agents are playing in an environment with obstacles. The obstacles in the hide and seek environment are movable, which allows the hiders to use them to their advantage. Although the obstacles in our environment aren't movable, which removes a layer of complexity, we wanted to mimic the same approach of using obstacles to the advantage of the runner. The Seekers in this example used a radar-like sensor to detect any objects nearby, which is a strategy that was later used in our system to avoid obstacles.

The agents in OpenAI's hide and seek implementation were trained using reinforcement learning, where a reward of 1 was assigned to hiders if they avoided being found, and -1 if they are found by a seeker. We adopted a similar reward system in our implementation, where the runner was assigned a reward of 1 for avoiding the tagger and a punishment of -1 for being tagged. We similarly used the strategy of reinforcement learning to have the agent learn multiple strategies over time to avoid the Chaser, just as the Hiders developed strategies to avoid the Seekers. It is important to note that only the Runner agents made use of reinforcement learning within our implementation and that the Tagger agent was simply tasked with getting as close to the Runner by evaluating its euclidean distance from the Tagger and determining the movement that would lead it closer to the Runner.

Learning Methodology

Reinforcement learning appears to be the obvious type of machine learning that should be employed for the purpose of training agents to play the game of tag. If supervised learning were to be employed, then the training algorithm would need to be supplied with ideal decisions to make in every possible situation that both the Runner and Tagger agent could find themselves in . Depending on the environment the agents find themselves in, a robust solution for each may not be able to be determined. Thus, supervised learning would not be practical in certain environments, as the variables present within them may not allow a solution for each agent to be determined. While complete solutions may be able to be developed within simple environments, the computational complexity to determine these solutions could be great and the solution acquired may only be useful to that specific environment or others extremely similar to it. Therefore, the game could not be solved for a specific environment and then have that solution extrapolated to many other environments. Instead, a solution would have to be derived for virtually all environments that could exist. Since there are infinitely many 2d finite environments with various obstacle configurations, taking a supervised learning approach is impractical.

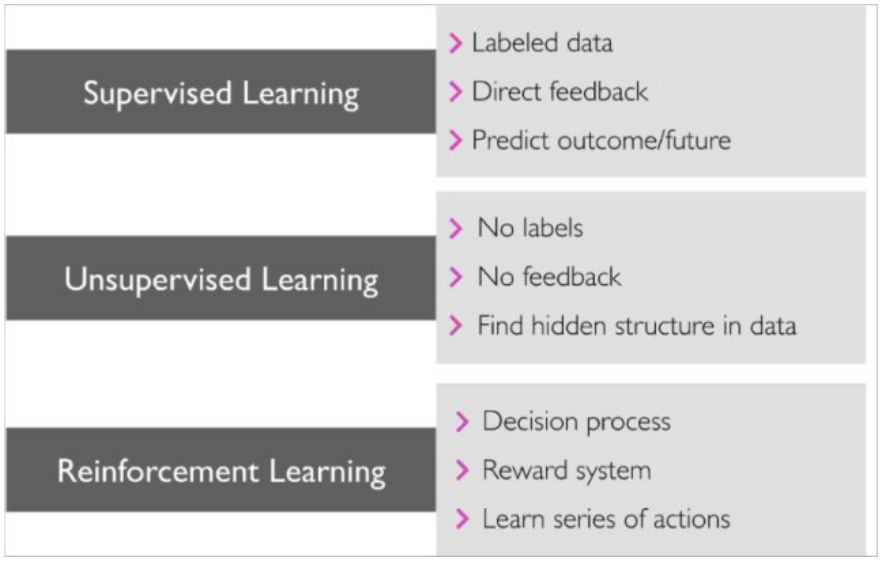
Similarly, unsupervised learning appears to be a less valuable approach than reinforcement learning. Unsupervised learning is typically used when there is not a defined output goal for the training process (Barlow, 1989). In this case, however, there are clear goals for both agents. The Tagger's goal is to reach the current position of the Runner as quickly as possible, whereas the Runner's goal is to avoid the Tagger reaching its position for as long as possible. Therefore, using unsupervised learning in an attempt to train each agent likely would be nonoptimal for achieving our goal. Through unsupervised learning, the system may generate intelligent output, but this output may be irrelevant to each of the agent's goals. Due to this, training through unsupervised learning would be inefficient and may produce a lot of irrelevant output, leading to reinforcement learning being our best option.

Reinforcement learning is a form of machine learning whereby agents learn what to do within a given environment through a reward and punishment system. Reinforcement learning is a form of goal-directed learning in that agents are trained to increase their rewards (rewards are directly linked to the goal that needs to be achieved by the agents) and avoid receiving punishments (Barto & Sutton, 2018, p. 2). How it works differently from both supervised and unsupervised learning is detailed in Figure 1. In the implementation given in this project and many others that make use of reinforcement learning, rewards are given as numerical points and the performance of an agent is dictated by the cumulative score of these points. For reinforcement learning to be effective, it is important that the rewarding system is set-up properly to minimize error and bias. Minimization or elimination of these ensures that the agents are being trained to learn their goals. The reward and punishment system is a parameter for training that must be adjusted throughout different iterations to ensure that it is effectively working. With a faulty

reward and punishment scheme, an otherwise robust training system may fail entirely because its success revolves around the rewarding scheme interfacing well with an agent's desired goal.

Figure 1

Machine Learning Types: High-level Overview



Note. From

Raschka, Sebastian. "3 Different Types of Machine Learning." *KDnuggets*, 2017,

www.kdnuggets.com/2017/11/3-different-types-machine-learning.html.

After investigating the types of machine learning that could be used within this project, reinforcement learning was decided upon. Given that both types of agents have defined goals, it appears that reinforcement learning is the optimal method. There are multiple algorithms used currently in research and industry to implement reinforcement learning (Szepesvári, 2010). Some of these algorithms use a policy-based mechanism to map inputs to outputs within the reinforcement learning process (Kung-Hsiang, 2018). In other words, a function is defined and then used to match inputs with specified outputs. Through refinement of this matching process, reinforcement learning is achieved and used to develop an intelligent agent. Other algorithms are value-based as opposed to policy-based. These algorithms do not maintain a specific policy in storage to map inputs to outputs. Rather, value-based algorithms only store a value function and inputs are used to generate the highest values from the value function (Kung-Hsiang, 2018). Often, algorithms that are value-based (like Q-learning) take the greedy approach to determine the optimal value of the function and then this value is used to estimate a policy (again, the greedy approach) to be used to map inputs to different outputs. So, for a policy-based

algorithm, a policy is supplied that should be used to map inputs to outputs through reinforcement learning while a value-based algorithm attempts to estimate the optimal policy and then generate outputs using that policy.

Another important component of algorithms that implement reinforcement learning is if they are based on a model or not. Model-based algorithms attempt to model the dynamics of the environment in which an agent is training within. In other words, "the model learns the transition probability *T(s1|(s0, a))* from the pair of current state s*0* and action a to the next state s*1*. If the transition probability is successfully learned, the agent will know how likely to enter a specific state given [a] current state and action (Kung-Hsiang, 2018)". The increased capabilities of a model-based algorithm comes with drawbacks, however. When dealing with a complex problem, the storage needed to keep track of the varying states, actions, and their dependencies becomes increasingly large, making it impractical to use model-based algorithms for problems that may generate many combinations of states and actions. On the other hand, model-free algorithms do not use any kind of planning for decision-making and they simply determine the best course of action given *only* the current state.

For this solution in particular, we used the Q-learning algorithm. This is a value-based and model-free algorithm, allowing us not to worry about space issues that may arise by using a model-based algorithm. This is also a sufficient algorithm to use to train the agents, as the necessary reward and punishment system can be implemented through a value function, so a policy-based algorithm is not necessary. The Q-learning algorithm is based on the Bellman Equation which is given in Figure 2.

Figure 2

Bellman Equation in Q-Value Form

$$Q^{\pi}(s,a) = \mathbb{E}[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \ldots | s, a]$$

$$= \mathbb{E}_{s'}[r + \lambda Q^{\pi}(s', a') | s, a]$$

Note. From

Kung-Hsiang, Huang (Steeve). "Introduction to Various Reinforcement Learning Algorithms."

Medium, Towards Data Science, 16 Sept. 2018

Implementation

This project was built on a machine learning library called ML-Agents, which acts as a communication layer between the Unity Game Engine and Tensor-Flow. With ML-Agents, we are able to conduct machine learning in a Unity based environment, and still have the same level of control over our Tensor-Flow neural network model.

Our agents exist within a small square-shaped environment, which is completely enclosed by outer walls. As seen in Figure 3, the floor of our environment is completely flat, and in the eyes of our agents the environment is strictly 2D. We used a flat environment in order to restrict the complexity of our problem space, allowing agents to only consider two dimensions of movement instead of three. Within the environment there are also obstacles for the agents to navigate around. These obstacles are static during the training and testing of our agent. One final component of the environment is the addition of a timer, which counts down to zero as the play session goes on. This timer displays the amount of time left in each episode of training, and upon reaching zero the environment is reset.

Figure 3

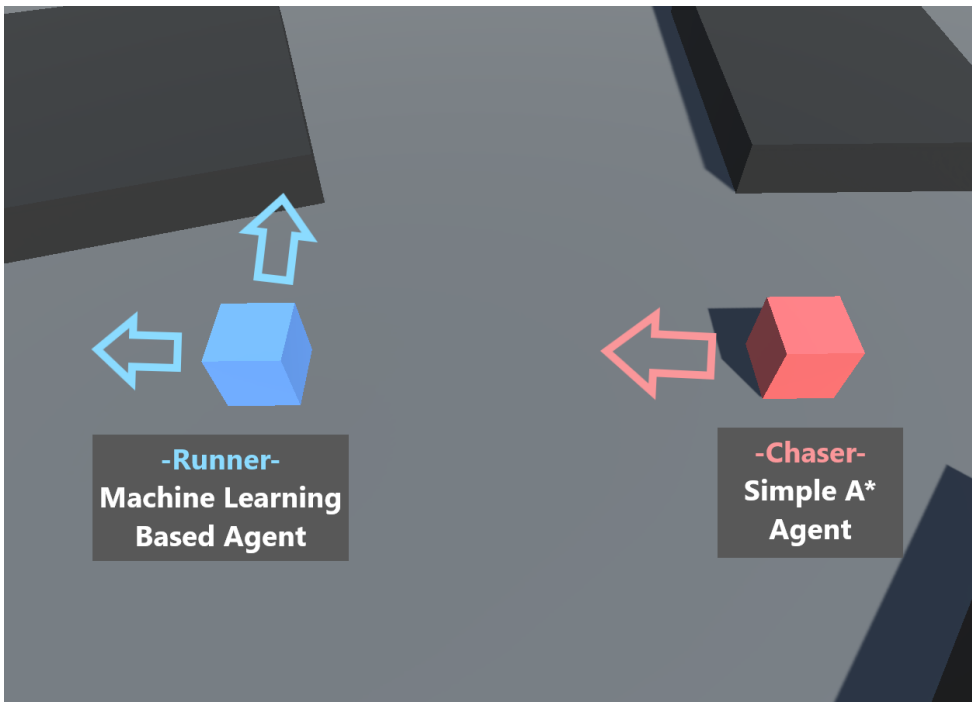
Top-down Environment View

As for the agents themselves, there are two classes: Taggers and Runners. These agents can be seen in Figure 4. The tagger is a simple A*-based navigation agent, which will always move towards the runner. Since the tagger uses A* to navigate throughout the environment, it has a built-in understanding of how to move around obstacles and reach the runner. We chose to use this simple solution for the tagger agent so that all of the focus for our machine learning implementation could be on one agent: the runner. Our runner agent uses an ML-Agents interface to send observations to the neural model, and receive consequent outputs for how the runner should react. Throughout the entirety of our testing, the runners' observations varied in what they represented. However, the output from the runner model always translates directly to the movement of the agent on its x and z coordinates. The end goal of the runner

agent is to evade the tagger agent until the environment's timer runs out. If the timer hits zero before the runner is tagged, the runner receives a reward and a new round of play begins. If the tagger succeeds in tagging the runner before the timer runs, the runner receives a punishment and again a new round of play begins.

Figure 4

Annotated Image of Agents



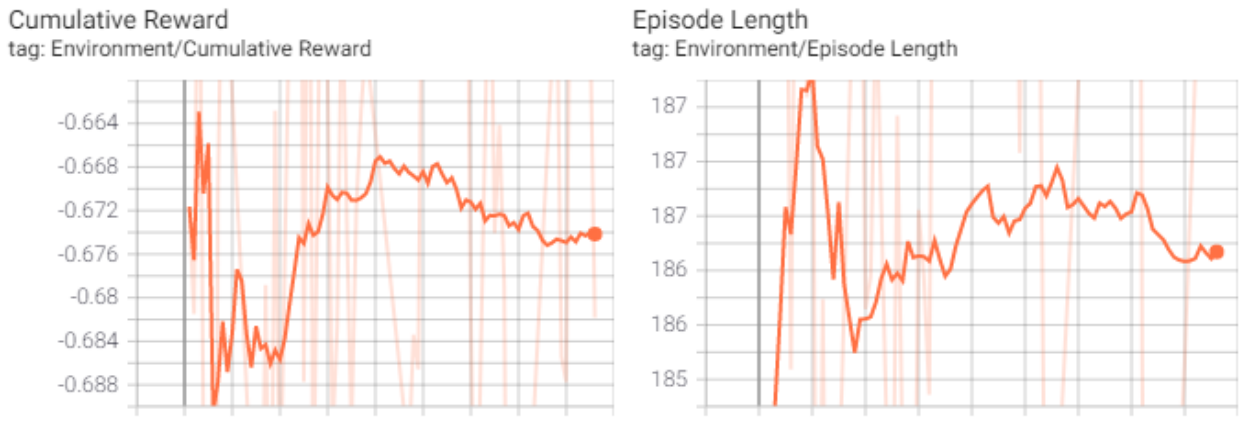
First Iteration

Through the development of this project, we had several different iterations of the agents, with changes to inputs and environment being made along the way. In this first iteration, our runner agent had a simple set of 4 inputs: its own x and z position, and the chaser's x and z coordinates. These observations are floating-point values, and are relative to the global origin in the scene. However, what we had not considered when implementing these observations is that using a global position adds complexity when training several agents in parallel, since each agent's environment is in a different coordinate range. As a result, our trained runner agent had great difficulty understanding its environment, and frequently ran directly into walls. The runner agent's strategy was effectively to move in one direction, and hope to avoid being tagged before the timer hit zero. We can see in Figure 5 that very little progress was made in terms of cumulative reward or episode length. Ideally, we would like to see both of these values increase

over time, as the runner agent learns to evade the chaser for longer, and consequently receives more rewards for evading successfully.

Figure 5

Chart of Cumulative Reward and Episode Length During First Iteration



Second Iteration

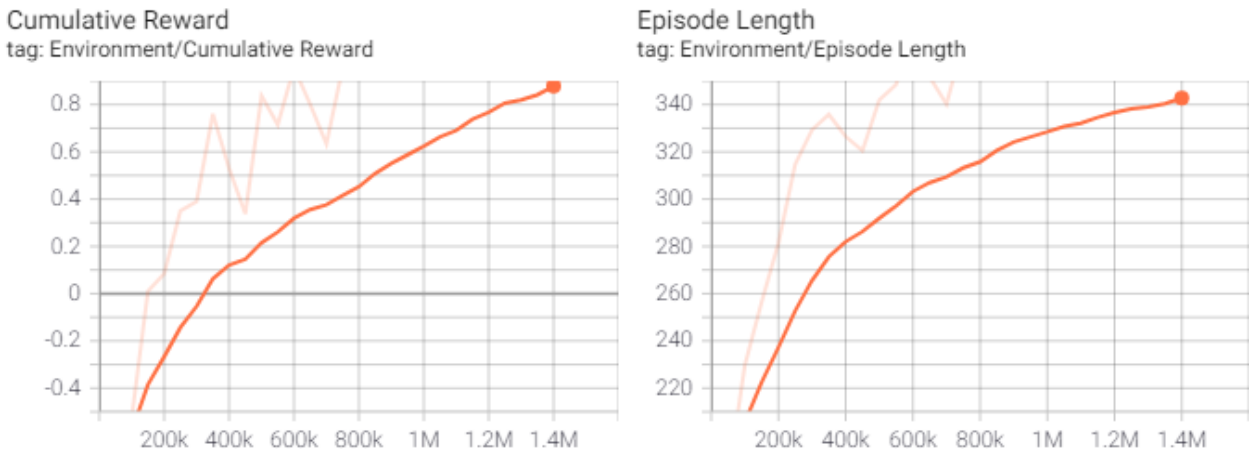
In the second iteration, we applied the lessons learned about using global position, and instead used positions relative to the agent's environment. With this change, each runner's observations about its own positions and the chaser's position are always in the same value range, regardless of its position in the entire Unity scene.

Another important change made during this iteration was to count any collisions with the outer wall as getting tagged. This was meant to encourage the agent to move around its environment more carefully, and not just move in one direction blindly.

The results from this iteration were much more promising than the previous. The Runner agent displayed a much greater understanding of its environment, and moved around obstacles semi-effectively. We can see in Figure 6 that the model was progressing in a positive direction, and the agent was learning to evade for longer amounts of time. It also learned to avoid touching the outer wall in addition to avoiding the tagger agent. One interesting observation about these results was that the runner gained a favorite hiding spot on the right side of the environment. In many cases, it would dash from the other side just to reach this hiding spot. While amusing, this also showed that the runner agent is probably not generalized enough, and is relying too heavily on a single learned trick to maximize its reward.

Figure 6

Chart of Cumulative Reward and Episode Length During Second Iteration



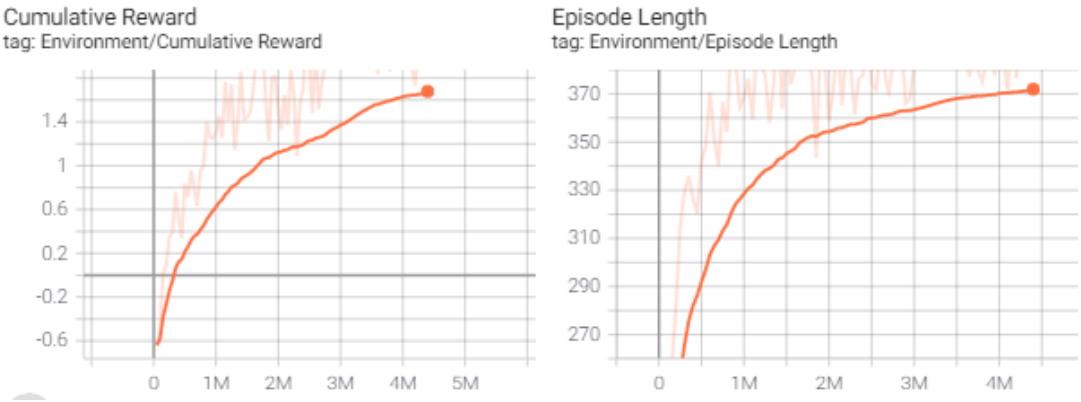
Third Iteration

Our third iteration was primarily aimed at trying to teach the runner agent a generalized understanding of the game's objective. We also wanted to see if the runner could evade for longer lengths of time. To achieve this, some major changes were made to the environment. All obstacles were removed to give the agents a simpler play space. We also increased the length of time for each play session from 30 seconds to 60 seconds. Finally, the runner is now punished less for getting tagged if it survives longer in the environment. Again, all of these changes were made with the goal of forcing the runner agent to become more generalized, and develop an adequate skill set for the game of tag.

Unfortunately, after training we found that the runner had still developed a single move-set which it relied on. Specifically, the runner would move down, to the right, and then up again in a rectangular motion. After reaching the top of this learned path, the runner agent would simply stop moving, and not once was it able to successfully evade the tagger for 60 seconds. We can see in Figure 7 that our training did result in longer episode length than previous iterations, but the method by which this was achieved is still not suitable for a general tag-playing agent.

Figure 7

Chart of Cumulative Reward and Episode Length During Third Iteration



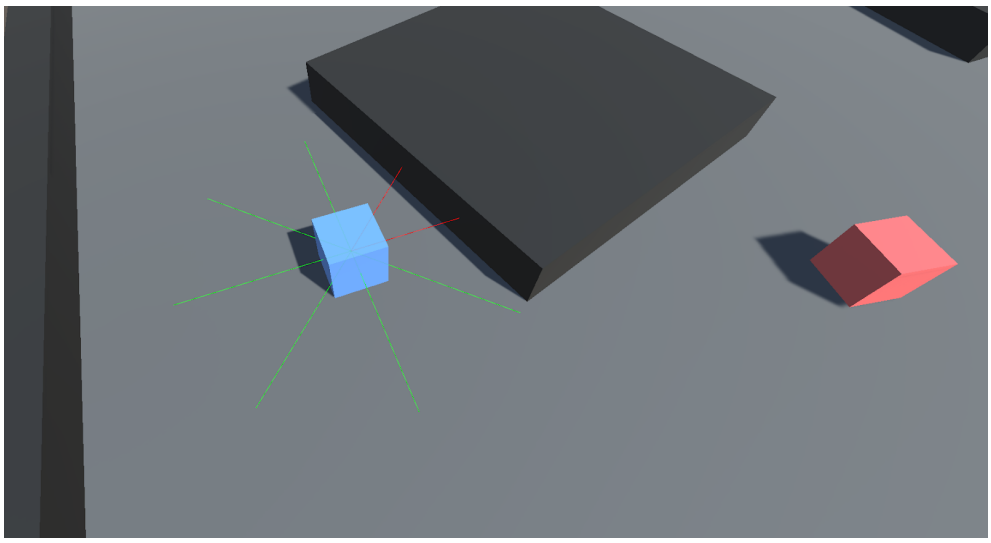
Fourth Iteration

Our fourth and final iteration had the same goals as the previous iteration. We wanted to help the agent learn the core objective of tag, and display that knowledge through a more generalized move-set. Our strategy to achieve this was to hide some information about the environment from the runner agent.

Instead of having the runner agent observe its position, it now has eight sensors around itself which detect when an obstacle is nearby. These sensors can be seen in Figure 8. This forces the agent to not consider the coordinate locations of its environment, but rather focus on its limited view of its surroundings. Another accompanying change was to how the runner observes the tagger's location. The runner no longer sees the tagger's position relative to the environment, and instead observes a vector representing the tagger's position relative to the runner.

Figure 8

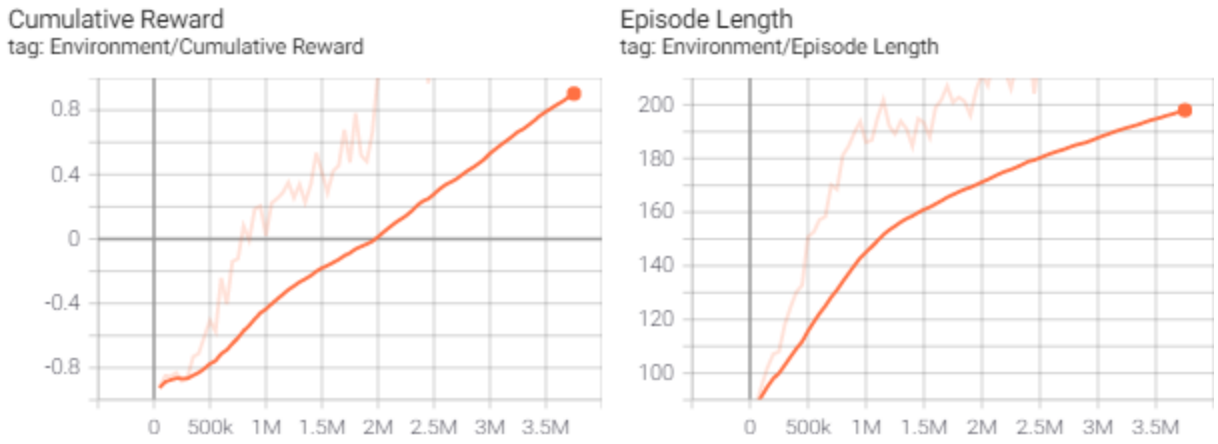
Runner Agents Obstacle Sensors



The results for this iteration are somewhat mixed. We no longer see much of a pattern in the runners movements. It also still seems to understand that its goal is to move away from the tagger agent, and also avoids touching any obstacles. However, the runner struggled to escape the corners of the environment after being chased into them. Every single episode of training seemed to end with the runner agent being cornered and not knowing that it can still move to escape. It's hard to make any solid conclusions about the state of our runner agent in this iteration, but the lack of movement patterns may show promise that we are getting closer to a more generalized agent. It may even be that better results could have been achieved with more time training. When looking at Figure 9, it certainly seems like the cumulative reward and episode length are still trending upwards. However, training during this iteration was already taking several hours for these results, and time limitations prevented longer training sessions.

Figure 9

Chart of Cumulative Reward and Episode Length During Fourth Iteration



Ethical Considerations

The use of AI in gaming has been shown to provide many enhancements to the industry. These create a sort of Utopian society within the gaming community. The use of AI to challenge the user to the best of their ability has a large influence on how the gaming industry is becoming more utopian. AI agents in games have become powerful enough to beat even the best opponents. We have seen it beat grand champion chess players, professional poker players and much more (McDonald, 2019). However, the use of AI to challenge the human opponent instead of beating them is very useful. To challenge the user to the best of their ability makes games more fun, which is the end goal of many video games. If the industry continues to focus on their users having fun over beating their users, the gaming industry is due for success.

The recent use of AI has created a shift in what games are today. Julian Togelius, a professor at NYU and co-director of the University's game innovation lab describes this shift, "Chess and Go were great, but they've had their run and we're done with that now. Stop playing chess and Go." Meaning chess and card games like Go are too simple for the technology that we have today. However, there have been improvements due to the use of AI in the gaming industry that surpasses the simplicity of these games. AI is now being used in game as a procedural generator meaning, " these games develop new levels algorithmically rather than via manual coding — responding to players' choices and creating new game elements on the fly (McDonald, 2019)." Improvements such as procedural generation are what will keep the gaming industry successful and lead it to a sort-of utopian future.

Conclusion

After progressing through multiple iterations of our agent, we determined that it was difficult to get the agent to understand the concepts of the game of tag, namely avoiding the chaser. In many trials, the agent seemed to adopt the same strategies repeatedly, such as hiding in one specific part of the map or using the same pattern of movement to escape the chaser. Repetition of these strategies maximized the reward the agent received, but did not show an understanding of how the game of tag worked. The runner also had difficulty avoiding the chaser during longer training sessions. Another difficulty that was faced was the modification of how the agent gathered information from the environment. It was determined that using relative positions rather than global positions was better suited to our needs. The specialized behavior adopted by the agent was also a result of the unchanging 2d environment that was used in our training. In future iterations of this model, we hope to use randomized environments in order to prevent non-generalized behavior from our runner agent.

References

Are video games and screens an addiction? (n.d.). Retrieved from https://www.mayoclinichealthsystem.org/hometown-health/speaking-of-health/are-video-games-and-screens-another-addiction

Barlow, H.B. "Unsupervised Learning." *Neural Computation*, MIT Press, 1 Sept. 1989, www.mitpressjournals.org/doi/10.1162/neco.1989.1.3.295.

Barasch, A. (2019, March 31). AI Ethics, Computer With Souls, Self-Playing Games. Retrieved from https://variety.com/2019/gaming/features/ai-ethics-computer-with-souls-self-playing-games-1203176874/

Kolter, Zico. "Introduction to Reinforcement Learning." *Carnegie Mellon University*, Carnegie Mellon University, 2017, www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjklpOx7NHwAhXSm-AKHZk8CMYQFjALegQIAxAD&url=https%3A%2F%2Ficaps18.icaps-conference.org%2Ffileadmin%2Falg%2Fconferences%2Ficaps18%2Fsummerschool%2Flectures%2FLecture5-rl-intro.pdf&usg=AOvVaw1QdLATeGNaM60Rn1O4Z2Ww.

Kung-Hsiang, Huang (Steeve). "Introduction to Various Reinforcement Learning Algorithms." *Medium*, Towards Data Science, 16 Sept. 2018, towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-Arsa-dqn-ddpg-72a5e0cb6287.

McDonald, G. (2020). Why AI will make your video games better. Retrieved May 17, 2021, from https://expmag.com/2020/03/why-ai-will-make-your-video-games-better/.

Raschka, Sebastian. "3 Different Types of Machine Learning." *KDnuggets*, 2017, www.kdnuggets.com/2017/11/3-different-types-machine-learning.html.

Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: an Introduction*. The MIT Press, 2018.

Szepesvári, Csaba. *Algorithms for Reinforcement Learning*. Morgan & Claypool Publishers, 2010, www.morganclaypool.com/doi/abs/10.2200/S00268ED1V01Y201005AIM009.

Togelius, J. (2019, April 17). How games will Play You. Retrieved May 17, 2021, from https://ai.shorensteincenter.org/ideas/2019/4/7/how-games-will-play-you