The Centipede Game: Nash Equilibria versus Machine Learning

Nathan Whybra

May 3, 2022

Abstract

The Centipede Game is a well known game involving two players. For some finite number of turns the players alternate choosing between two options: “take” or “pass.” The first player that chooses “take” ends the game, wins the game, and also gets a reward. Otherwise, if each player chooses “pass” for every turn in the game, then the game will end and both players get some reward. The longer the game goes on, the higher the reward becomes for the players if they decide to “take.” So the question is, what is the best turn for a player to “take?” The standard game theoretic approach involving subgame perfect Nash equilibria (SPNE’s) predicts that the “optimal” strategy for both players is to immediately “take” in the first turn. The goal of this paper is to create a machine learning model that both plays and learns The Centipede Game, and then compare what the model predicts is the best turn to “take” versus what the game theoretic results predict.

Background: Game Theory

In this section, we briefly discuss the relevant Game Theory necessary for understanding this paper. More specifically, we will begin by defining what a normal-form game is, provide an example of a normal-form game, and then explain the concept of Nash equilibria in normal-form games. Afterwards, we will define what an extensive-form game is, introduce The Centipede Game, explain what a subgame perfect Nash equilibrium (SPNE) is, and then discuss the Nash equilibria and subgame perfect Nash equilibria of The Centipede Game.

Although there are many different versions and classes of games throughout mathematics and Game Theory, some of the simplest games fall under the category of normal-form games of perfect information, which leads us to our first definition.

Definition. *A* ***normal-form game of perfect information*** *is a tuple (N, A, R) where:*

- *N is a subset of natural numbers used for labeling players. For instance if $N = \{1, 2\}$, then the game has two players: Player 1 and Player 2. The set N does not necessarily need to be composed of natural numbers since it's just a set of labels, but the convention is useful for doing mathematics about games.*

- *A is a tuple of sets $(A_1, A_2, ..., A_n)$ with n representing the total number of players in the game. Each A_i is the set of action choices available to Player i in the game. The action choices themselves can be anything really as long as it makes sense in context. There are more nuances to this, but these will be addressed later with examples.*

- *R is a reward function, $R : A_1 \times A_2 \times ... \times A_n \to \mathbb{R}^n$ that assigns rewards to the players depending on what actions they took. The i^{th} coordinate of the output vector in R^n is the reward for Player i, which we will denote as r_i frequently. Intuitively, the larger a player's reward is compared to the other players' rewards, the better that player performed in the game compared to other players.*

- *We make the assumption that each player knows all possible action choices and all possible rewards for every other player. This is the "perfect information" part of the definition.*

The Prisoners' Dilemma

To better understand the above definition, we give an example. The Prisoner's Dilemma is a famous game with the following story. Two prisoners are in a prison cell, and each prisoner has incriminating information about the other. The police make deals with the prisoners. If a prisoner cooperates with police and tattles on the other prisoner, and the other prisoner does not cooperate with the police (or defects), then the prisoner who cooperated gets a large reward and the prisoner who defected gets a poor reward. If both prisoners cooperate with police, then they both get a fair reward. If both prisoners defect, then they both get poor rewards. The real numbers that are used to describe the rewards are open to interpretation. For instance, maybe a small reward corresponds to getting extra jail time and maybe a large reward corresponds to being released from prison. Again, what matters here is the relative sizes of the rewards to each other. Using our definition of a game, one example of a Prisoners' Dilemma game has:

- $N = \{1, 2\}$. The players are Prisoner 1 and Prisoner 2.

- $A = (A_1, A_2)$ with $A_1 = A_2 = \{C, D\}$. Both players can either cooperate (C) or defect (D).

- $R(C,C) = (3,3)$, $R(C,D) = (5,0)$, $R(D,C) = (0,5)$, and $R(D,D) = (1,1)$. For clarity, $R(C,D) = (5,0)$ means if Prisoner 1 cooperates and Prisoner 2 defects then Prisoner 1 gets a reward of 5 and Prisoner 2 gets a reward of 0.

For two player games, oftentimes it is useful to consider a game's equivalent reward matrix (sometimes called the pay-off matrix or game matrix). For this Prisoners' Dilemma, the reward matrix is shown in Figure 1 and is fairly self-explanatory.

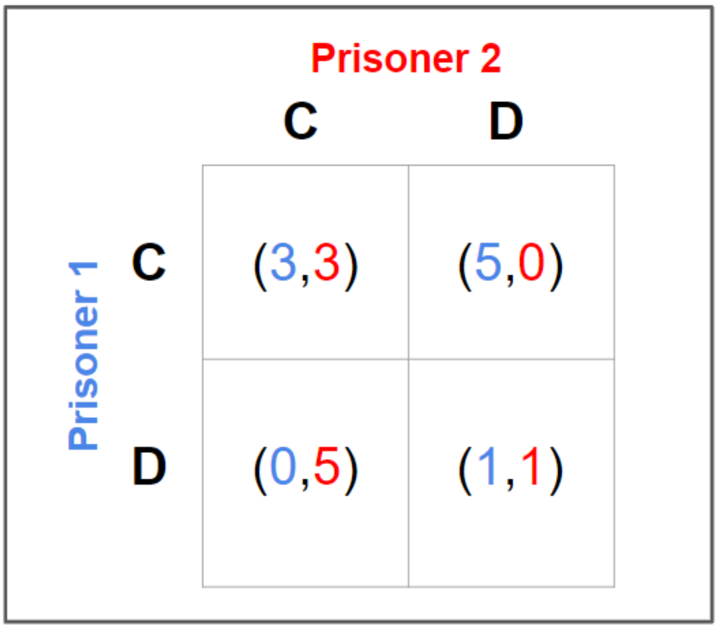

Prisoner 2
C
D
Prisoner 1
C
D
(3,3)
(5,0)
(0,5)
(1,1)

Figure 1: The reward matrix for the Prisoners' Dilemma game considered above. Prisoner 1's rewards are in blue and Prisoner 2's rewards are in red.

What is a Nash Equilibrium?

Suppose you wanted to find the "optimal" or "best" action that each player could take in a game. Optimal can mean a lot of things, but one reasonable notion of optimality stands out in Game Theory. This is the idea: suppose a game has n players $\{1, 2, ..., n\}$ where each player i makes action choice a_i, and each player i gets some reward r_i. We call the tuple $(a_1, a_2, ..., a_n)$ an **action profile** for the game, and it represents the actions taken by each player in the game. Keeping the action choices of all other players fixed, if each player i can't get a better reward by changing their current action choice, then the action profile is considered "optimal." Stated more simply, if no player can profitably deviate from their current action choice, then the action profile for the game is considered "optimal."

Intuitively, if the outcome of the game for each player is good enough so that each player wouldn't do anything differently given the chance, then the players played the game well. This leads us to our next definition:

Definition. *Suppose $G = (N, A, R)$ is a normal-form game of perfect information with n players and $a^* = (a_1, ..., a_n)$ is an action profile for G where each player i gets reward r_i^*. For each player i and all other possible action choices $\overline{a_i} \in A_i$ available to player i, if we have:*

$$r_i^* \geq (R(a_1, ..., a_{i-1}, \overline{a_i}, a_{i+1}, ..., a_n))_i$$

Then the action profile a^* is called a ***Nash equilibrium*** *of G.*

Part of the "optimality" of a Nash equilibrium comes from our assumption that each player is always going to choose the most rational action choice when given the chance. This is because in our theory, we assume that "perfect information" is available to each player. Basically, each player knows all the possible action choices of the other players and all the possible rewards each player can get from these action choices. Although this is almost never the case for games in real life with inexperienced or new players, it actually sometimes is the case for very experienced players. Players who are experienced with a game have played the game many times and thus have memories of numerous tried action choices and their corresponding outcomes. Therefore, a Nash equilibrium in a game can be thought of as a prediction of what very experienced and logical players would do in a game in real life. This makes sense because, for instance, how differently do you imagine professional soccer players play soccer compared to how 5 year old children play soccer? Later in this paper when we begin discussing The Centipede Game and our machine learning models that learn different Centipede games, this fact will be mentioned again because instead of having perfect information from the start like in the theory, the machine learning models are going to learn from experience just like people in real life probably would.

In general, Nash equilibria are good for predicting what logical players with experience might do in a game, but they are bad at predicting what illogical and inexperienced players might do in a game. Some other aspects of Nash equilibria in games are that they are almost never unique since most games end up having more than one Nash equilibrium. Also just because a Nash equilibrium is supposed to be an "optimal" action profile in a game, this doesn't necessarily mean the profile maximizes the rewards of each player. Before we see an example of this last statement, we should quickly discuss a good way of calculating Nash equilibria in normal-form games.

Calculating Nash Equilibria in Normal-Form Games

For a two player normal-form game, the following algorithm is helpful for computing Nash equilibria.

- Begin by writing the game's equivalent reward matrix (see Figure 1 for an example).
- For each column in the matrix, underline the maximum rewards possible for Player 1 (there can be more than one maximum if there are duplicates).
- Next for each row in the matrix, underline the maximum rewards possible for Player 2 (again, there could be duplicates).
- At this point, the matrix entries that have both Player 1 and Player 2's rewards underlined are the Nash equilibria of the game.

Intuitively, going through each column of the matrix represents Player 1 deviating between their action choices given an action choice by Player 2, and similarly going through each row of the matrix represents Player 2 deviating between their action choices given an action choice by Player 1, so it's fairly easy to see how the action profiles produced by this algorithm are Nash equilibria from the definition. There is also a natural way of extending this algorithm for games with more than two players, but all the games we discuss in this paper are two player games, so we choose to omit the general version of this algorithm from the paper.

For an easy example, one can use the algorithm above to calculate the Nash equilibria of the Prisoners' Dilemma game from earlier. After doing so we see that this version of the Prisoners' Dilemma game has only one Nash equilibrium, namely the action profile (C, C) where both prisoners cooperate with the police. Notice that both prisoners get a reward of 3 in this Nash equilibrium, but some other action profiles allow a prisoner to get a reward of 5. As stated earlier, even though a Nash equilibrium is a kind of "optimal" play, no players here end up getting a maximized reward. Why? Think about it this way: what incentive do the prisoners really have to defect? If a prisoner defects in this game, they can only get 0 or 1 as a reward. On the other hand if a prisoner cooperates in this game, they can only get 3 or 5 as a reward. This means the possible rewards for cooperating are always better than any of the possible rewards for defecting. Therefore both prisoners don't have any good reason to defect, and both decide to cooperate which gives us our Nash equilibrium. Concluding this discussion, we are finally ready to introduce the concept of extensive-form games.

Extensive-Form Games of Perfect Information

Although normal-form games are simple and intuitive, they lack a certain structure that is helpful for thinking about more complicated games. Very simply, an extensive-form game is a game with a tree diagram. The tree has a root (the vertex where the game starts), player vertices or decision nodes (vertices where players take actions), and leaf nodes (vertices where the game ends and players collect their rewards). A major difference between extensive-form games and normal-form games is that in extensive-form games we think about games happening sequentially. That is, players take turns making action choices. The full list of action choices a player takes during the game is called a strategy. We summarize with a definition:

Definition. *An **extensive-form game of perfect information** is a tuple (N, T, V, A, S, R) where:*

- *N is a subset of natural numbers representing the players in the game.*
- *T is a rooted tree, called the Game Tree.*
- *V is a set of labeled vertices in the Game Tree. Some vertices, called decision nodes, are labeled with values from N to specify which players are allowed to make a move at each vertex. The other vertices, called leaf nodes, where players finish the game and collect their rewards.*
- *With n players in the game, A is a tuple of sets $(A_1, ..., A_n)$ where each A_i is the set of action choices available to Player i. Each action is labeled with a vertex, meaning that at different vertices players might have different actions available to them.*
- *With n players in the game, S is a tuple $(S_1, ..., S_n)$ where each S_i is the set of all possible strategies available to Player i. For a player i, a strategy is a sequence of actions with each action coming from a vertex that player i can play on. The strategies must move from the root of the Game Tree towards the leaf nodes, and every strategy for a player always includes the same number of actions as the number of vertices that player can play on, **regardless of whether or not the vertices are visited in the game**.*
- *R is a reward function, $R : S_1 \times S_2 \times ... \times S_n \to \mathbb{R}^n$ that gives each player a reward depending on what strategy they used. Again, the i^{th} coordinate from the output vector in R_n is the reward for player i.*
- *Due to perfect information, every player knows all the possible strategies that any other player can take and all of the corresponding rewards.*

To make things more vivid, Figure 2 below is one example of an extensive-form game. We won't really do much with it because The Centipede Game is also an extensive-form game, but it will be good for familiarization.

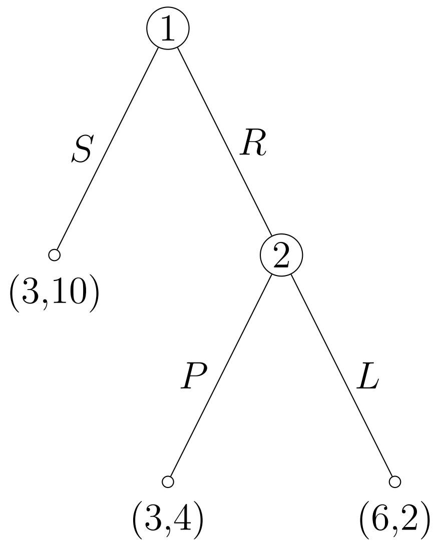


Figure 2: A simple example of an extensive-form game with two players. Player 1 moves from vertex 1 and Player 2 moves from vertex 2. In this game tree, the node at the very top is the root node (meaning the game starts there) and the nodes with the tuples of numbers attached to them are the leaf nodes with the corresponding reward for each player.

The Centipede Game

Finally we get to The Centipede Game! The Centipede Game is an extensive-form game with two players and has the following story. In the first turn of the game, Player 1 and Player 2 are presented a sum of money. Player 1 can either choose to take, and get more money than Player 2, or pass. If Player 1 takes the money the game is over. If Player 1 passes, the sum of money is altered and now Player 2 can decide to take or pass on it. If Player 2 takes, they get more money than Player 1 and the game ends. If Player 2 passes, again the sum of money is altered, and now the second turn of the game starts. This process repeats until one of the players decides to take the money, or until the game runs out of turns. If each player passes for every turn in the game, both players leave with an equal amount of money (Note: We choose the rewards for each player in this case to be smaller than the rewards given if the players were to take during their last available opportunity. This is because if these rewards were larger, it would be obvious that both player's should work together to get the highest possible reward). The longer the game goes on, the higher the reward becomes if a player decides to take the money. A two turn Centipede Game is shown below in Figure 3.

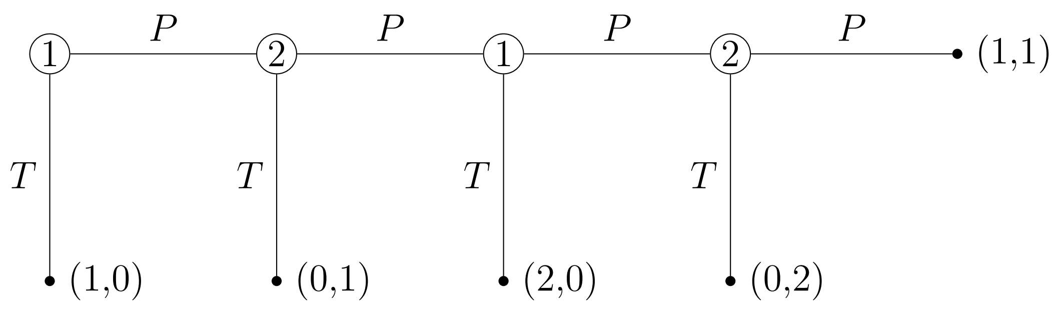
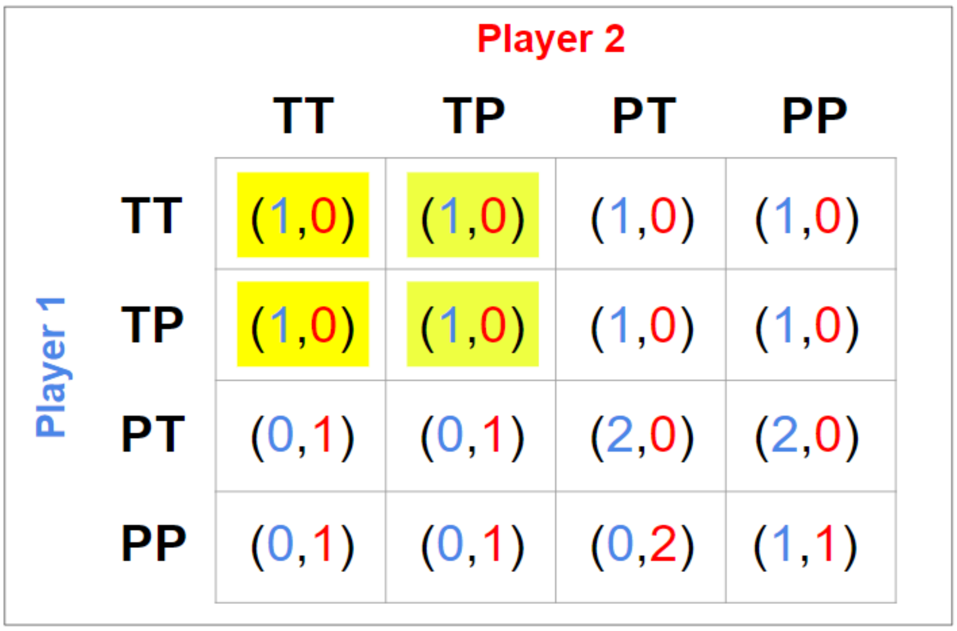


Figure 3: A two-turn Centipede Game. Here T means "take" and P means "pass." If Player 1 makes it to turn 2 they can win big! Should they dare?

The interesting thing about The Centipede Game is that the players can make their reward higher by choosing to pass, but (as we will see very soon) the Game Theory says that the players should always take in the first turn!

Nash Equilibria in The Centipede Game

Consider the Centipede Game in Figure 3. We want to try and find its Nash equilibria. To make this task easy, we can actually turn this Centipede game into an equivalent normal-form game like in Figure 4 below, and then solve for Nash equilibria the same way we did for normal-form games.



Player 1 \ Player 2	TT	TP	PT	PP
TT	(1,0)	(1,0)	(1,0)	(1,0)
TP	(1,0)	(1,0)	(1,0)	(1,0)
PT	(0,1)	(0,1)	(2,0)	(2,0)
PP	(0,1)	(0,1)	(0,2)	(1,1)

Figure 4: The equivalent normal-form game (reward matrix) for the Centipede game from Figure 3. The Nash equilibria are highlighted in yellow.

An important distinction here is that in an extensive-form game players make an action anytime the game gets to one of their vertices in the game tree, meaning players can make more than one action choice in an extensive-form game. However the way we defined action choices in normal-form games, players only make one action choice per game. The correspondence between extensive-form and normal-form games are that the set of all of possible *strategies* for a player in an extensive-form game will be the set of all possible action choices for that player in the equivalent normal-form game.

Now looking at Figure 4, as we hinted earlier, all of the Nash equilibria in this game happen when the players decide to take in the first turn... but we're being a bit disingenuous here. By fiddling with the rewards of the game, we can make Centipede games that have a Nash equilibrium where the players don't take in the first turn. To get an even better understanding of this game, we use a refinement to the concept of Nash equilibrium which we describe in the following section.

Subgame Perfect Nash Equilibria

We begin this section with a definition.

Definition. *A **subgame** of an extensive-form game G is the extensive-form game defined by starting G at some player vertex or decision node (could still be the root node), and ignoring the parts of G that happened before that vertex. The way this is defined, the game G itself is also a subgame.*

For example, these are all of the subgames of the specific Centipede Game we've been talking about:

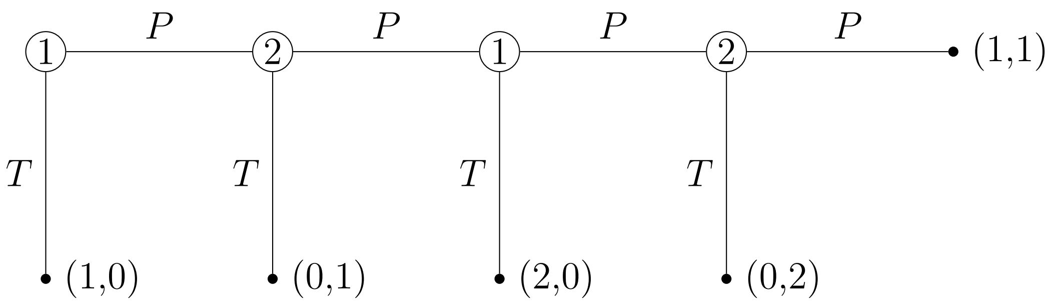


Figure 5: The subgame starting at Player 1's first vertex, ie. the full game.

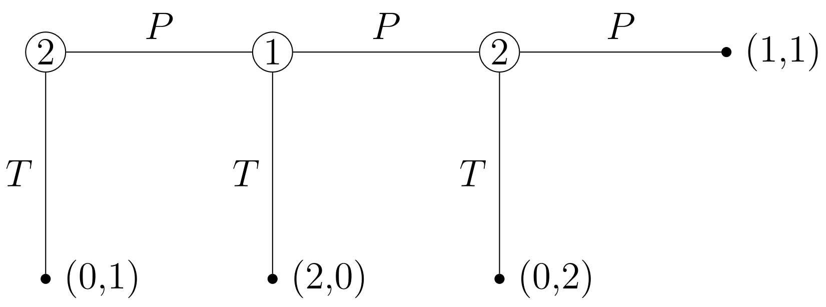


Figure 6: The subgame starting at Player 2's first vertex.

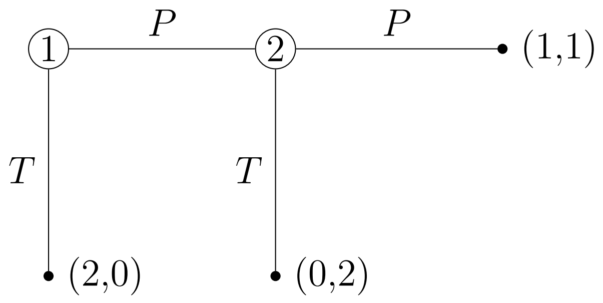


Figure 7: The subgame starting at Player 1's second vertex.

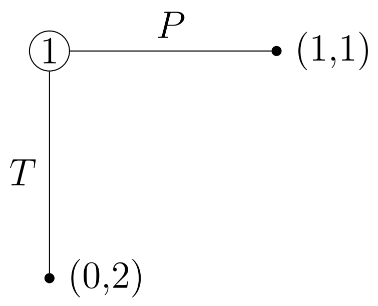


Figure 8: The subgame starting at Player 2's second vertex.

This leads us to our next definition:

Definition. *A* ***subgame perfect Nash equilibrium*** *or* ***SPNE*** *of an n player extensive-form game G is a strategy profile $s^* = (s_1, ..., s_n)$ such that s^* is a Nash equilibrium in every subgame of G. Here s^* is a strategy profile for the complete game G, but not for the proper subgames of G specifically. To resolve this, we say that s^* is a Nash equilibrium in a subgame H if there exists a Nash equilibrium in H that matches s^* when H and G overlap.*

The concept of a subgame perfect Nash equilibrium is a refinement to the concept of a Nash equilibrium because firstly, any subgame perfect Nash equilibrium of a game is also a Nash equilibrium of that game. Secondly, subgame perfect Nash equilibria capture the idea of "perfect recall" where players "remember" their past action choices as they progress through the game, and use their "memories" to play "better" as the game progresses. Another interpretation is that in a SPNE, players act like they would act in hindsight.

Calculating Subgame Perfect Nash Equilibria

There are two basic ways we can calculate the subgame perfect Nash equilibria of an extensive-form game G. One way is to make the equivalent normal-form game for each subgame of G, calculate the Nash equilibria of these subgames like we normally would, and then keep the strategy profiles for G that match up to the Nash equilibria of all the subgames. This method is intuitive but can be pretty time consuming, especially for larger games (imagine a 1000+ turn Centipede Game). The faster, and usually better, way of calculating SPNE's is by using a method called **backwards induction**. To do backwards induction, we begin by looking at the terminal nodes in a game tree (the player vertices adjacent to leaf nodes). Then using our idea that players want the largest reward available to them, we work our way backwards through the game and keep track of the action choices players would choose given the chance. It is easiest to explain backwards induction by doing an example, so again consider the same Centipede Game we've been using so far.

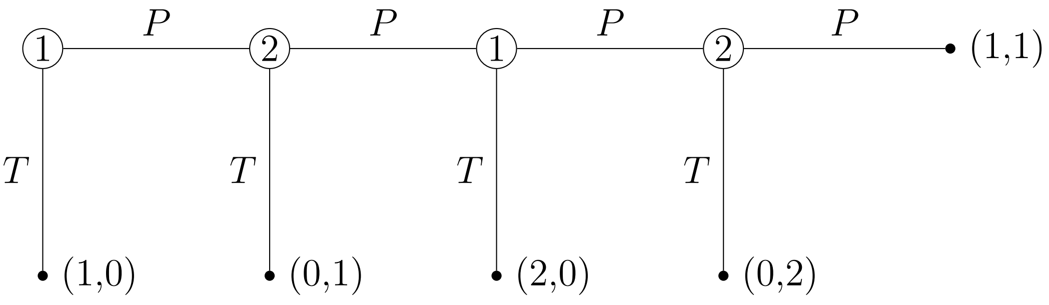


Figure 9: The Centipede Game... again.

Here is the process:

- Looking at Figure 9, focus your attention to the very right where Player 2's second vertex is. We start here because this vertex is terminal and has the most leaf nodes. If Player 2 was here, they'd like to take the money because if they take here they get a reward of 2 and if they pass they get a reward of 1 which isn't as good.
- Moving backwards towards the root node, our next stop is Player 1's second vertex. Since Player 1 is rational, they know that if they pass then Player 2 will just take afterwards and Player 1 will get nothing. To avoid that, Player 1 wants to take at this vertex to get the reward of 2.
- Moving backwards again, we are now at Player 2's first vertex. Similarly here, Player 2 knows that Player 1 is going to take if they decide to pass, so to avoid getting nothing Player 2 wants to take and will accept the reward of 1 here.
- Moving backwards one last time, we are finally at the root node of the game, ie. Player 1's first vertex. Again Player 1 knows that if they decide to pass then Player 2 will just take. So like before, Player 1 wants to take and will accept the reward of 1 here.

Putting everything together we recover only one SPNE strategy profile, namely $s^* = (\text{TT}, \text{TT})$. So the unique SPNE of this Centipede game is when both players take the money whenever possible. In general, SPNE's don't have to be unique and usually in a backwards induction there would be more splitting paths and more actions to keep track of, but the Centipede Game is relatively simple. This is the result we alluded to earlier in our discussion before calculating the Nash equilibria of the game. As a sanity check, notice that s^* is one of the Nash equilibria we calculated earlier (as it should be). This is a weird result, because it's telling us that experienced and rational players with hindsight will always just take the money immediately and not even try to get the higher rewards later on. In our Centipede game example the rewards are relatively small, but we could have made the later rewards much larger compared to the reward available in the first turn, and still we would recover the same SPNE. Here is a relevant theorem:

Theorem. *In any Centipede game (the way we have defined it) with a finite number of turns, there is a unique subgame perfect Nash equilibrium where each player takes in every turn.*

Proof. Repeat the same backwards induction procedure we just did for the two turn version of the Centipede game.

$\square$

This finally brings us to the goal of this paper: to test whether or not a trained machine learning algorithm, more specifically a reinforcement learning algorithm, plays The Centipede Game the way the SPNE predicts. We now transition into the machine learning part of this paper.

Background: A Brief Description of Reinforcement Learning

Although very vague, the following definition gives a general feel for what's involved in a reinforcement learning algorithm.

Definition. *A reinforcement learning algorithm is a machine learning algorithm with the following things in mind.*

- *An agent, or player.*
- *An environment where the agent is expected to perform.*
- *A set of states S representing different aspects of the environment.*
- *A collection of action sets $A(S) = \{A(s)\}_{s \in S}$ which represent actions available to the agent when in a given state s of the environment. The set $A = \bigcup_{s \in S} A(s)$ represents all of the possible actions the agent could take.*
- *A reward signal, or reward function $R : S \times A \to \mathbb{R}$ that gives the agent a reward for taking an action in a given state.*
- *A policy function $\pi : S \to A$ which tells the agent what action to take when in a given state.*

Sometimes the goal of a reinforcement learning algorithm is to find an optimal policy π^* which optimizes the cumulative reward earned by the agent. Other times, the algorithm is designed to learn without optimizing the policy. Generally speaking, any learning algorithm concerned with an agent learning from interacting with an environment falls under the umbrella of reinforcement learning.

To avoid making this paper unnecessarily long, we choose to specifically describe the learning algorithm we used to learn Centipede games. Our algorithm in fact will be very similar to the k-armed Bandit algorithm described in Chapter 2 of [1].

Learning The Centipede Game

To learn different Centipede games, our algorithm will work based off of the following representation.

- There will be two agents: Player 1 and Player 2.
- There will be just one state for each player: the initial state of the Centipede game before any player makes a move.

- If the specific Centipede game in question has n turns, then there will be 2^n actions available to each player. Each action corresponds to a possible strategy for a player from the extensive-form of the Centipede game. For example in a two turn Centipede game, the set of possible actions for each player would be $\{TT, TP, PT, PP\}$. We will call the set of actions available to both players A.
- The reward given to each player is calculated by playing a Centipede game with the strategies (or actions in the reinforcement learning context) the players decided to use, and observing the rewards given to each player.

There are a few things we haven't specified yet: the policies for each player and the method in which each player actually learns. In order to explain what the policies are, we need to explain the learning aspect first.

Learning By Estimating Rewards

The way we have represented The Centipede Game, every time a player takes an action they will get a reward. The thing is, depending on what action the other player chooses, the reward given to the player is not guaranteed to be the same every time the player chooses the same action. For this reason, we'd like to maintain estimates of the rewards we expect a player to receive for choosing specific actions. Once we have this, the player can make a decision about what action to choose next based on what the current reward estimates are. To achieve this, for each player i we make a matrix Q_i where the rows of the matrix represent the possible states a Player i could be in and where the columns of the matrix represent the possible actions Player i could take, and then setting $Q_i(s, a)$ to be our current estimate of the reward given to Player i for taking the action a in the state s. Since there is only one state in our model, the Q matrix (or Q table) for each player reduces to a single row vector ($Q_i(s, a) \to Q_i(a)$).

The next question is, how do we calculate the reward estimates $Q_i(a)$? In practice there are many ways to do this, but we pick a relatively simple way. Before the players play the game at all, initially set $Q_i(a) = 0$ for every $a \in A$. After playing the game some number of times, suppose that Player i chooses to take action a. Let k be the total number of times that Player i chose action a so far (including this time), and let R_i^j be the reward that Player i received after choosing a for the j^{th} time (with $1 \leq j \leq k$). Also, let $Q_i^j(a)$ denote the value of $Q_i(a)$ after the j^{th} time Player i chooses action a, then we update $Q_i(a)$ by the following rule.

$$Q_i^k(a) = Q_i^{k-1}(a) + \frac{1}{k}(R_i^k - Q_i^{k-1}(a))$$

Why do this? We're trying to estimate the reward R given to Player i for choosing action a. In other words, we'd like to know $\mathbb{E}\left[R_i \mid a\right]$ (the expected value of the reward given to Player i for choosing action a). A natural way to approximate this would be to average all of the rewards Player i has received so far from choosing action a, ie. we would set:

$$Q_i^k(a) = \frac{1}{k}\sum_{j=1}^{k} R_i^j$$

But notice:

$$\begin{aligned}
Q_i^k(a) &= \frac{1}{k}\sum_{j=1}^{k} R_i^j \\
&= \frac{1}{k}\left(\sum_{j=1}^{k-1} R_i^j + R_i^k\right) \\
&= \frac{1}{k}\left(\frac{k-1}{k-1}\sum_{j=1}^{k-1} R_i^j + R_i^k\right) \\
&= \frac{1}{k}\left((k-1)Q_i^{k-1}(a) + R_i^k\right) \\
&= Q_i^{k-1}(a) + \frac{1}{k}(R_i^k - Q_i^{k-1}(a))
\end{aligned}$$

So the update rule described above is equivalent to just taking the average of all the rewards Player i received for choosing action a so far. The reason we prefer the update rule over the traditional formula for the average is because it takes less time computationally to use the update rule as opposed to computing the entire sum every time Player i chooses action a. Throughout reinforcement learning, there are other kinds of update rules and many of them take a similar form to the update rule that we are using. A benefit of using this update rule is that by the strong law of large numbers, as $k \to \infty$, we have $Q_i^k(a) \to \mathbb{E}\left[R_i \mid a\right]$. This is not always ideal for reasons we won't get into, but for The Centipede Game it should work out fine. We are now ready to talk about the policies of each player.

ϵ-Greedy Policies: Exploration versus Exploitation

We've already discussed how the players learn their expected reward for choosing an action, but now we need to discuss how players decide what action to take in the first place. Intuitively, one might think the players should just choose the action a with the highest $Q(a)$ value. In other words, players should choose the action that they think will give them the highest reward. Let π_i be the policy of Player i, then this would correspond to:

$$\pi_i = \arg\max_{a \in A} Q_i(a)$$

If there is more than one action a that maximizes $Q(a)$, just pick one of the actions at random. Here we say that the players are **exploiting** their knowledge since they are using their current information about their expected rewards to make their decision about what action to choose next. The problem with this however is that under these policies as stated, the players may start taking the same few actions over and over again without ever trying anything else. This is an issue because sometimes players can get fixated on actions that give them okay rewards in the beginning of the game when in reality there might be other actions that give much higher rewards on average. To encourage the players to try other actions, or **explore**, we add an extra parameter to the polices above. Namely, we introduce a parameter $\epsilon \in (0, 1)$ that represents the probability that the players pick an action randomly, regardless of the Q table values. In practice, the value of ϵ to use will vary depending on the situation. Making this adjustment, the policies from earlier become:

$$\pi_i = \begin{cases} \arg\max_{a \in A} Q_i(a) & \text{with probability } 1 - \epsilon \\ \text{some random } a \in A & \text{with probability } \epsilon \end{cases}$$

In this paper we choose the random actions uniformly from the set A, meaning any $a \in A$ has equal probability of being chosen. A common challenge in reinforcement learning is trying to balance the amount that agents exploit versus the amount that they explore. We are now ready to state our Centipede game learning algorithm in its entirety.

The Algorithm

```
$Q_1(a) \leftarrow 0$ for every $a \in A$
$Q_2(a) \leftarrow 0$ for every $a \in A$

$K(a) \leftarrow 0$ for every $a \in A$
$L(a) \leftarrow 0$ for every $a \in A$

**for** $N$ times **do**
    $a_1 \leftarrow$ action chosen by $\pi_1$
    $K(a_1) \leftarrow K(a_1) + 1$

    $a_2 \leftarrow$ action chosen by $\pi_2$
    $L(a_2) \leftarrow L(a_2) + 1$

    $r_1 \leftarrow$ reward for Player 1 using $a_1$ in Centipede game
    $r_2 \leftarrow$ reward for Player 2 using $a_2$ in Centipede game

    $Q_1(a_1) \leftarrow Q_1(a_1) + \frac{1}{K(a_1)}(r_1 - Q_1(a_1))$

    $Q_2(a_2) \leftarrow Q_2(a_2) + \frac{1}{L(a_2)}(r_2 - Q_2(a_2))$
**end for**
```

In the above algorithm, N is the number of times we want the model to play the Centipede game, $K(a)$ is the number of times Player 1 chose action a, and $T(a)$ is the number of times Player 2 chose action a.

Results

We tested our reinforcement learning model on a 2 turn Centipede game, a 6 turn Centipede game, and a 20 turn Centipede game. For each Centipede game tested, we trained the model in 9 different batches with each batch lasting for 1000 games. Also for each Centipede game tested, the first batch had $\epsilon = 0.9$, the second batch had $\epsilon = 0.8$, ..., and the ninth batch had $\epsilon = 0.1$, where ϵ was the explore probability for each player. The purpose of this was so that in the first few batches (high values of ϵ), players could focus more on exploring rather than exploiting and become familiar with the kinds of rewards possible from choosing each action. Then in the later batches (low values of ϵ), players could start focusing more on exploiting instead of exploring so that they could start playing the way that they thought was best. After each batch, we calculated the percentage of times each player decided to "take" in the first turn. The following figures summarize the results.

Player 1: 2 Turn Centipede Game

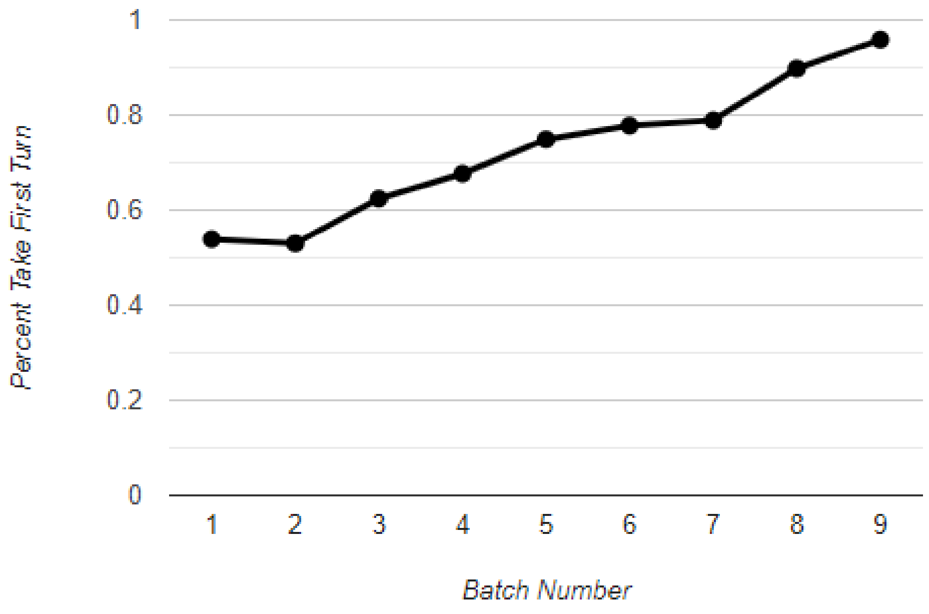


Figure 10: The percentage of times Player 1 chose to take in the first turn as a function of the Batch Number in the 2 turn Centipede game. By the end of training, the model nearly always takes in the first turn.

Player 2: 2 Turn Centipede Game

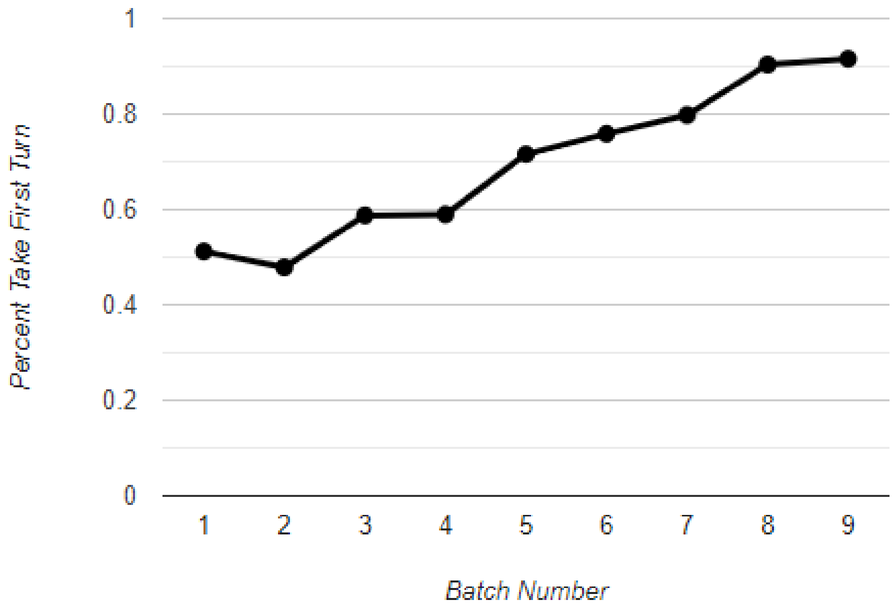


Figure 11: The percentage of times Player 2 chose to take in the first turn as a function of the Batch Number in the 2 turn Centipede game. By the end of training, the model nearly always takes in the first turn.

Player 1: 6 Turn Centipede Game

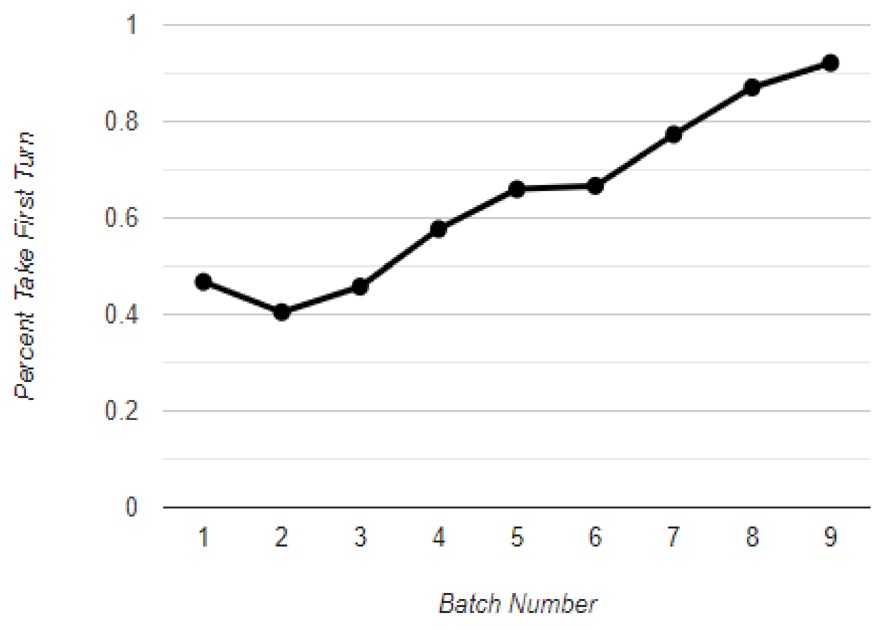


Figure 12: The percentage of times Player 1 chose to take in the first turn as a function of the Batch Number in the 6 turn Centipede game. By the end of training, the model nearly always takes in the first turn.

Player 2: 6 Turn Centipede Game

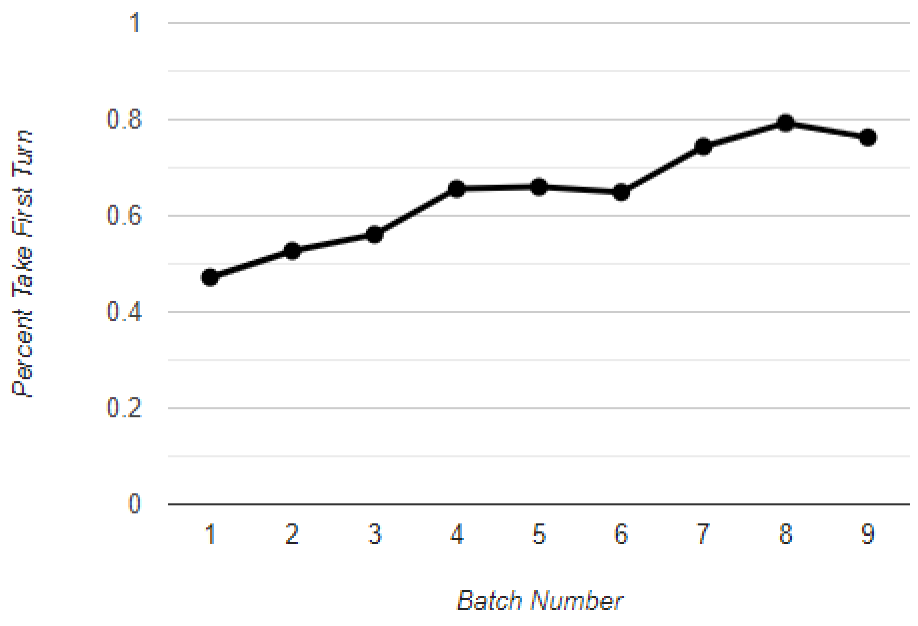


Figure 13: The percentage of times Player 2 chose to take in the first turn as a function of the Batch Number in the 6 turn Centipede game. By the end of training, the model takes in the first turn around 80 percent of the time.

Player 1: 20 Turn Centipede Game

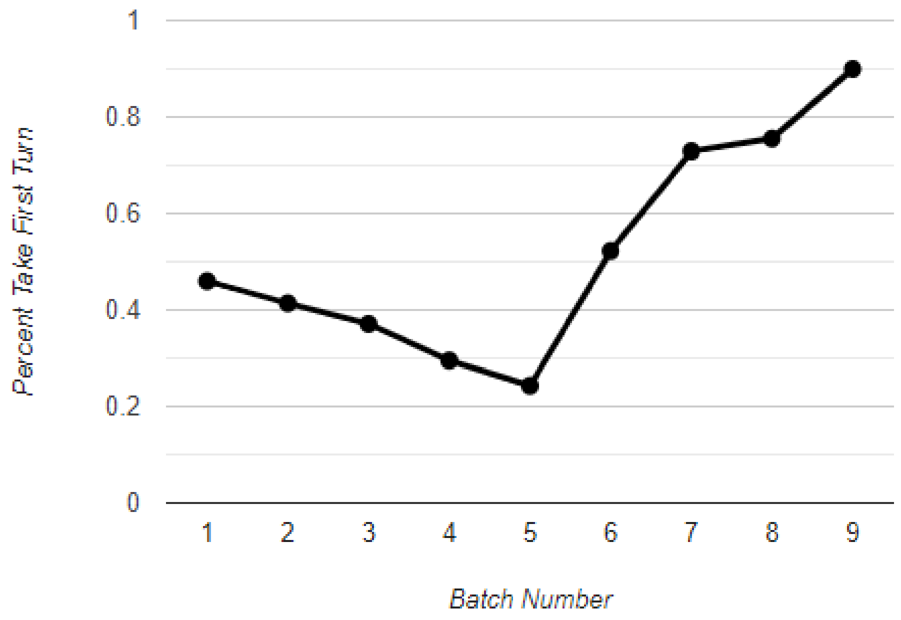


Figure 14: The percentage of times Player 1 chose to take in the first turn as a function of the Batch Number in the 20 turn Centipede game. For high values of ϵ (low Batch Number) the model progressively takes in the first turn less and less, but by the end of training, the model nearly always takes in the first turn.

Player 2: 20 Turn Centipede Game

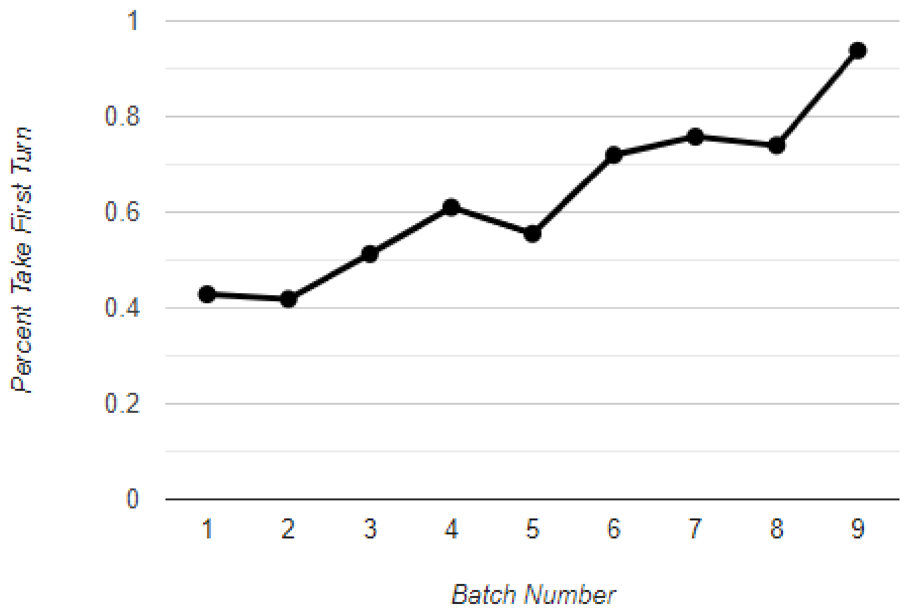


Figure 15: The percentage of times Player 2 chose to take in the first turn as a function of the Batch Number in the 20 turn Centipede game. By the end of training, the model nearly always takes in the first turn.

Concluding Remarks

Our models seem to agree with the Game Theory. For each Centipede game tested, by the end of training, Player 1 chose to take in the first turn over 90% of the time. The same was almost true for Player 2, however by the end of training in the 6 turn game, Player 2 chose to take in the first turn just under 80% of the time. We don't think this difference is that significant though, especially since Player 2 chose to take in the first turn over 90% of the time in the 20 turn game... so this difference can most likely be corrected by more training. More insight into this comparison might be made by trying this experiment again but with a more complicated learning model. For instance it might be interesting to model The Centipede Game using a Generative Adversarial Network (GAN) because neural network's are notoriously good at noticing hidden patterns in data, and GAN's specifically have been shown to be excellent models for games. However, our hypothesis is that these models would still probably end up agreeing with the Game Theory. We think the reason is a matter of experience. Although empirical studies [2] have shown that regular people tend to play differently than the Game Theory predicts, most people don't play Centipede games thousands of times trying to learn every possible thing that can happen to them in the game. Summarizing, inexperienced players probably won't play like the Game Theory predicts. Since ultimately any machine learning model would gain immense experience playing the Centipede game, it is likely that other models would behave similarly to the model we used in this paper.

Citations

1. Sutton, R. S. Bach, F. Barto, A. G. Reinforcement Learning: An Introduction; MIT Press LTD: Massachusetts, 2018.

2. McKelvey, R. Palfrey, T. “An experimental study of the centipede game.” Econometrica, 1992.