Solving an Erdos-Selfridge-Spencer game using deep reinforcement learning : Attacker Defender

Report by Antoine Habis & Geert-Jan Huizing

Introduction

Goal In this report we will be analyzing and partially implementing the 2018 article *Can Deep Reinforcement Learning Solve Erdos-Selfridge-Spencer Games?*[1]. The authors proposed using Erdos-Sefridge-Spencer games to assess the strengths and limitations of deep reinforcement learning methods. Indeed, benchmarks like the Atari games exist, but they are often very complex and not fully characterized, making it difficult to know whether the algorithm made an optimal decision.

Notation A big part of this report will consist of summarizing the aforementioned article. To highlight the sections where we try to replicate the authors' experiments and offer our personal results and interpretations, we will add a bar on the left, like this.

What are Erdos-Selfridge-Spencer (ESS) games?

Definition (combinatorial games) A combinatorial game is a two player game where :

- The game is deterministic: there is no randomization mechanism such as flipping a coin or rolling a die.
- There is perfect information in the game: each player knows all the information about the state of the game, and nothing is hidden.

Definition (ESS games) ESS games are a class of combinatorial games where

- Two players take turns selecting objects from some combinatorial structure
- Optimal strategies can be defined by potential functions derived from conditional expectations over random future play

Why are ESS games interesting? According to the article, the following points make Erdos-Sefridge-Spencer games a good fit for evaluating deep reinforcement learning algorithms.

- They offer a simply parametrized family of environments
- Optimal behavior can be completely characterized
- The environment is rich enough to support interaction and multiagent play

Here we are going to focus on one particular game: Spencer's attacker-defender game.

[1] Maithra Raghu Alex Irpan Jacob Andreas Robert Kleinberg Quoc V. Le Jon Kleinberg, ICML 2018

1 The Erdos-Selfridge-Spencer Attacker-Defender Game

The Attacker-Defender game was introduced by Joel Spencer in *Theoretical Computer Science* (1994) under the name "tenure game". Two players compete – an attacker and a defender. The attacker's goal is to move a set of pieces to the end of the board, and the defender's goal is to destroy all the pieces before the attacker reaches the end.

Rules of the game The game has K levels and N points are spread into these levels at the beginning. The attacker's goal is to get at least one of his pieces to level K. If he does, he wins. At each turn, the attacker has to propose a partition A/B of his pieces. Then the defender choose to destroy either A or B. If the attacker has no more pieces, the defender wins, otherwise the pieces left can jump to a higher level.

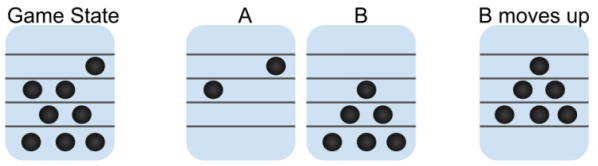

Figure 1: One turn of the game: (i) the attacker partitions his pieces into two sets A and B (ii) the defender chooses to destroy set A (iii) the remaining set B moves up

Parameters We can see easily see that the complexity of the game can be influenced by varying the parameters N and K.

1.1 Some Preliminary results

Theorem 1 Consider an instance of the Attacker-Defender game with K levels and N pieces, with all N pieces starting at level 0. Then if

$$N < 2^K$$

the defender can always win.

Proof. If all pieces start at level 0 and the defender always destroys the larger set then after K steps the number of pieces left $n_k \leq \frac{N}{2^K}$ and if $\frac{N}{2^K} < 1$, as $n_k \in \mathbb{N}$, it means that $n_k = 0$, so the defender wins. □

However the pieces can be initialized at any level, not only the level 0. We need a generalization of theorem 1.

Definition 1 Given a game state $S = (n_1, n_2, ..., n_k)$, with n_i the number of pieces at level i, we define the potential of the state as: $\phi(S) = \sum_{i=0}^{K} n_i 2^{-(K-i)}$, i.e. $\phi(S) = w^T S$ with $w_l = 2^{-(K-l)}$

Theorem 2 Consider an instance of the Attacker-Defender game that has K levels and N pieces, with pieces placed anywhere on the board, and let the initial state be S_0. Then

1. If $\phi(S_0) < 1$, the defender can always win
2. If $\phi(S_0) \geq 1$, the attacker can always win.

Let us now prove theorem 2.

Proof of theorem 2, part 1 By hypothesis, $\phi(S_0) < 1$ (0).

We can show that by destroying the partition with the highest potential at each step, the defender will win.

The attacker proposes a partition A/B. Without loss of generality we suppose $\phi(A) \leq \phi(B)$ (1).
We have that $\phi(A) + \phi(B) = \phi(S_0)$ (2).
The defender destroys B and the pieces advance of one level. The potential of the next set of pieces if $2\phi(A)$.

Thanks to (1) then (2) and finally (0), $2\phi(A) \leq \phi(A) + \phi(B) < 1$. Hence recursively for each subsequent state S, $\phi(S) < 1$

To win, the attacker would have to place a piece on level K, which would produce a potential of at least 1, which we just showed is impossible. Recursively we get that the attacker can not win. □

Proof of theorem 2, part 2 To prove part 2 of the theorem, Spencer defined an optimal strategy for the attacker that relies on a greedy algorithm which would demand too much computing power in our case. Later in this paper the authors defined instead a new optimal strategy, the prefix-attacker.

2 Deep Reinforcement Learning on Attacker-Defender

2.1 Framing the problem

Tweaking complexity We have seen previously that the attacker-defender game's difficulty is controlled by the initial potential $\phi(S_0)$ and the number of levels K. For greater values of K, in addition to longer games and thus less feedback, we get way more possible game trajectories and thus a higher complexity.

Inputs In order to use deep reinforcement learning on this game, we need a way to represent the information available to the player.

- For the attacker, a good way to represent the game state is a $K+1$ dimensional vector containing the number of pieces on each level from 0 to K.
- For the defender, we can use a similar vector of dimension $2 \times (K+1)$ representing both partitions A and B.

Start state For a given K, the start state S_0 can be initialized randomly so as to match a given potential.

2.2 Training a defender

Defining an attacker strategy to play against In the appendix, the authors explain the strategy they used for the attacker when training the defender : the attacker plays optimally most of the time but occasionally uses a suboptimal disjoint support strategy.

Let us define the disjoint support strategy : it is a variation on Theorem 3's prefix attacker, with the following difference.

- The regular prefix attacker aims to partition the pieces in two sets A and B such that $\phi(A) \approx \phi(B)$
- The disjoint support attacker picks A and B such that $\phi(A) \approx \alpha(\phi(B) + \phi(A))$ where α is uniformly chosen in [0.1, 0.2, 0.3, 0.4] at each turn.

The authors claim that training the defender agent to react to sets of uneven potential makes the final strategy more generalizable.

Linear optimal policy As shown in the proof of Theorem 2, we can make an optimal decision based on the sign of $\phi(A) - \phi(B) = w^T(A - B)$, which is linear. This gives us a *linear optimal policy.*

Increasing difficulty The closer the start potential $\phi(S_0)$ is to 1, the closer the strategy has to be to optimality in order to win. Hence for a fixed K, high start potential values make the game more difficult. As mentioned before, a higher K will also increase difficulty because the rewards will be sparser and it allows more possible game states.

Exploring different training methods The authors applied different reinforcement learning methods to train the defender.

- Proximal Policy Optimization (PPO)
- Advantage Actor Critic (A2C)
- Deep Q-Networks (DQN)

Linear model Since an optimal strategy can be expressed as a linear function, we can think that a linear model (i.e. single layer) should be enough to train the agent. However the authors' results with a linear model are disappointing.

- PPO gets an average reward around 0.50 for potential $p = 0.8$ with both $K = 15$ and $K = 20$. When we increase the potential to $p = 0.99$, the average reward drops to -1, which means it looses almost all the time.
- A2C does not even reach a positive average reward for potentials between 0.8 and 1
- DQN offers the best performance : with $K = 15$, the average reward approaches 0.75, except for potential 0.99, where it is around 0. But when K is increased to 20 the average reward reaches only between 0.25 and 0.5, depending on potential.

In conclusion the linear model using DQN can deliver interesting results when the problem is easy (low K, low potential).

Using deeper models Even though we could expect a linear model to be enough, the authors found that using more layers led to an increase in performance.

- PPO still does not perform well with high potentials, but overall the average rewards are much better than with the linear model.
- A2C still performs poorly for high K or high potential, but in the simplest case ($K = 15$, $p = 0.8$), it reaches an average reward of 0.75. Overall the results are better than its linear counterpart.
- DQN performs really well in this setting. It manages to get an average reward close to 1 even for high K or high potential. We can observe however that the training converges much faster in simple configurations (low K, low potential).

In conclusion, the deep models perform much better than the previously tested single-layer models. The best performances were achieved using DQN.

2.3 Our results

Implementation In order to leverage the power of the OpenAI Gym framework, we chose to use the Attacker-Defender Gym environment found on Github at the address `https://github.com/maxencemonfort/dqn-attack-defense/`, and tweaked it a bit for our purpose. You can find the code for the following experiments on our Github repository `https://github.com/gjhuizing/Deep-RL-Erdos-Selfridge-Spencer`.

Experimental settings Following the results presented in the article, we chose to train a defender using DQN, and modeling the decision function with a Multi-Layer Perceptron. As our computing resources were limited, we chose to keep the same range of potentials as the article (0.8 to 0.99) but to lower the values of K, to make the experiment less demanding to run. As a result, we also lowered the number of training steps to 4000, which we found to be sufficient to train the model. We used a moving average to smoothe the reward graph and give an idea of the average reward.

Results In fig. 2 we show the learning curves for different values of K and p. These curves led us to the following observations:

- Generally speaking, models with higher potentials need more steps to converge and have a lower final average reward.

- The range of final average rewards matches what we could see in the article, varying between 0.75 and 1 depending on K and p.
- When K increases, the final average reward tends to be lower, which was expected, but the model converges faster, which is counter-intuitive. Our interpretation is that when the setting is too easy, it is possible to win with relatively bad strategies, which makes it harder to discover an optimal strategy.
- Our limitations on the value of K making the game "too easy" may also explain why we observe the general trends of the article, but that some individual results seem odd. For instance with $K = 5$, the average reward converges faster for $p = 0.95$ than for $p = 0.9$.

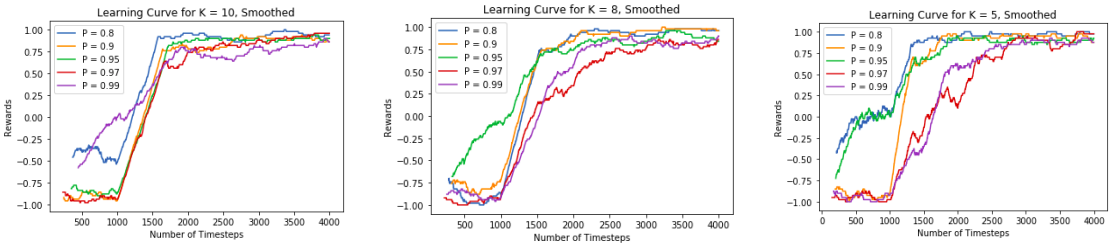

Figure 2: Defense

3 Generalization in RL and Multiagent Learning

Does our defender generalize well? The fundamental observation justifying this section is that *our defender tends to overfit to the attacker's strategies*. It does not generalize well to attack strategies not seen during training.

Testing on a different strategy To demonstrate this issue, the authors trained the defender against an optimal attacker, and tested its performance against the disjoint support attacker. As we have seen earlier, the latter is theoretically easier to beat, since it is suboptimal. However the experiment (see fig. 3) shows that the model fails to generalize to a different opponent strategy, even when the opponent is easier to beat.

Figure 3: The defender overfits to the attacker's strategy

3.1 Training the attacker

In order to avoid this over-fitting issue, the idea is to train the attacker too.

Finding a decision policy In Spencer's original work, he proposed a greedy optimal decision policy to partition the pieces. However in practice this would give us an action space way too big to train the agent well. The following theorem gives a novel way to construct an optimal attacker, that induces an action space of size K only, i.e. linear complexity instead of exponential.

Theorem 3 For any Attacker-Defender game with K levels, start state S_0 and $\phi(S_0) \geq 1$, there exists a partition A, B such that $\phi(A) \geq 0,5$, $\phi(B) \geq 0,5$, and for some level l, partition A contains *some* pieces of level $i > l$, and B contains *all* pieces of level $i < l$.

Proof.
$\forall l \in \{0, 1, ..., K\}$ A_l be the set of all pieces from level K to level l, l being excluded. We have that $\phi(A_{t+1}) \leq \phi(A_t)$, $\phi(A_K) = 0$ and $\phi(A_0) = \phi(S_0) = 1$.

There exists l s.t, $\phi(A_l) < 0.5$ and $\phi(A_{l+1}) \geq 0.5$. If we have the equality then we just stop and set $A = A_{l+1}$ and B the complement.

Otherwise $\phi(A_l) < 0.5$ and $\phi(A_{l+1}) > 0.5$:

A_l contains all the pieces from level K to l so $\phi(A_l)$ is an integer multiples of $2^{-(K-l)}$ meaning that

$\exists m, n \in \mathbb{N}$ s.t $\phi(A_l) = m2^{-(K-l)}$ and $\phi(A_l) = n2^{-(K-l)}$

We are guaranteed that level l has $m - n$ pieces, and that we can move $k < m - n$ pieces from A_{l-1} to A_l such that the potential get equal to 0.5 □

Decision policy induced by theorem 3 Thanks to this theorem, the attacker can always keep it's potential superior to 1 every step meaning that the attacker will always win if its initial potential is superior to 1.

Experimental results In both algorithms (A2C and PPO), they found that there was a large variation in performance when changing K.

From $K = 5$ to $K = 25$, the Average Reward at the end of the training drops from ≈ 1 to ≈ 0 with A2C.

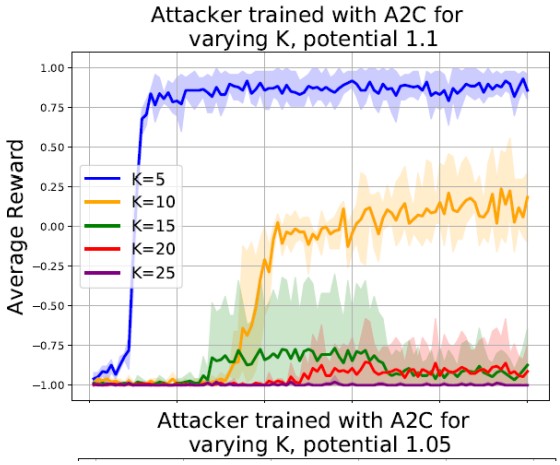

Figure 4: Attacker trained with A2C

Increasing K affects both reward sparsity and action space size.
However with a fixed value of K, they observe small changes when the potential changes (small variability) but a greater variance of the Average Reward.

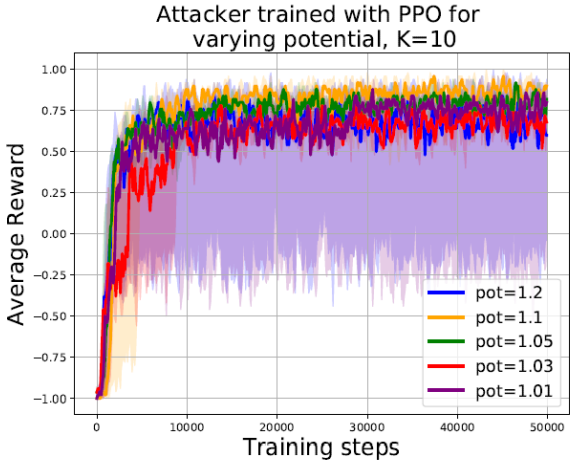

Figure 5: Attacker trained with PPO

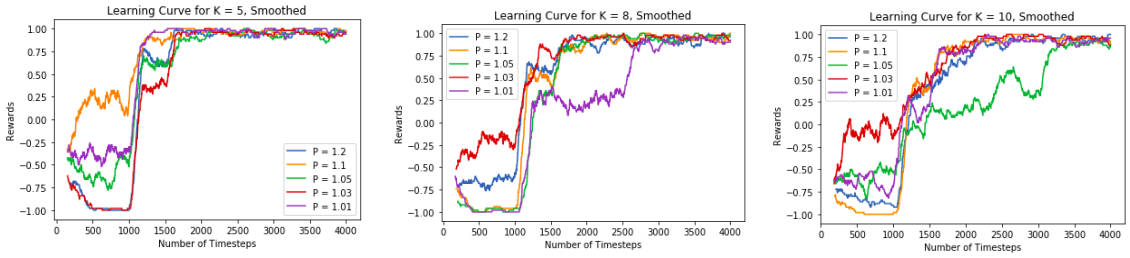

Figure 6: Our attacker training

Our results We recover results that are similar to the ones obtained in the paper. We couldn't go any further than $K = 10$ because of issues with the memory space. We tried using DQN again even though the authors claimed it had poor performance, and you can see in fig. 6, for easy settings (low K, low potential) it seems to work well enough.

We can see that the variance inducted by small values of the potential (very close to 1) is way smaller with PPO than with A2C.

3.2 Multiagent Play

The attacker and defender were trained jointly through multiagent play. These are the main takeaways:

- The attacker does not perform as well in the multiagent setting as it did in a single agent setting against an optimal defender.
- The defender trained through multiagent play generalizes much better than the defender trained against the hardcoded optimal attacker (see fig. 7).

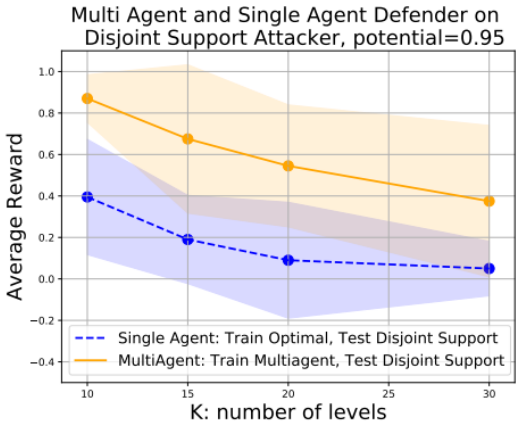

Figure 7: Defender generalization

4 Training with Self Play

Principle of self-play In the above section the attacker and the defender were trained with different neural nets. The idea here is to use a single one.

Self Play with Binary Search The algorithm we use is based on the results of Theorem 3. The decision of the attacker is to find a balanced partition according to Theorem 3. More precisely, given an initial partition A/B into a prefix and a suffix of the pieces sorted by level, we determine which set has higher potential, and then recursively find a more balanced split point inside the larger of the two sets.

Algorithm 1
initialize game
repeat
... Partition game pieces at center into A, B
... **repeat**
..... Input partition A, B into neural network
..... Output of neural network determines next binary search split
..... Create new partition A, B from this split
... **until** Binary Search Converges
... Input final partition A, B to network
... Destroy larger set according to network output
until Game Over: use reward to update network parameters with RL algorithm

Results: The results are way better with self play. We can see that the algorithm is almost not affected by the variation of K and ϕ. When increasing K from 5 to 25, the Average Reward at the end of the training remains approximately the same (≈ 1) and the same thing occurs when varying the start potential.

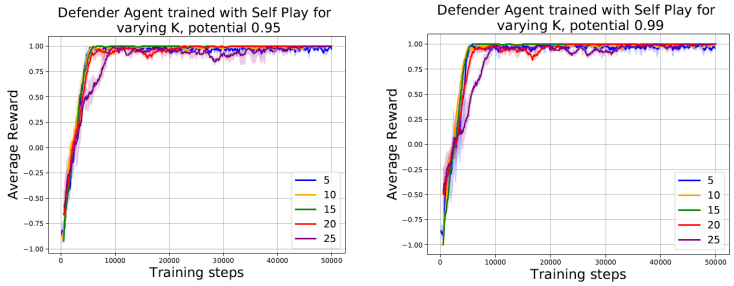

Figure 8: Self Play

5 Supervised Learning Versus Reinforcement Learning

Motivation In previous sections, we have seen that the Attacker-Defender game is a good benchmark to evaluate reinforcement learning methods. But since we have an optimal policy to determine the next move given a particular game state, we can also try using supervised learning to predict the best next move for a given state.

How to apply supervised learning The idea is to build a large dataset of game states. Since for each state we know the best move we can evaluate the model's performance.
The team behind AlphaGo also compared supervised learning and reinforcement learning using such a dataset. However, their target move was only a heuristic based on existing human strategies, whereas in our case we know the actual optimal move.

Experimental settings Since the goal is to compare reinforcement learning and supervised learning the experiments carried out by the authors go as follow:

- Train a defender decision policy using *reinforcement learning*. For each turn save the game state $X_i = (A_i, B_i)$.
- For each observation X_i compute the best next move y_i according to our optimal decision policy.
- Train a defender decision policy using *supervised learning* using the observations X and the best moves y
- Compare the models' predictions $y_{supervised}$ and y_{RL} to the ground truth y to determine the models' performances.

In order to get a fair comparison, the same deep architecture is used to model the decision policy in both cases.

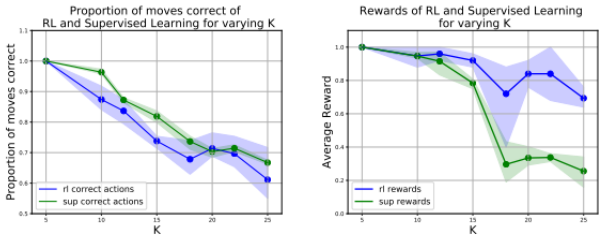

Figure 9: Compared performance of RL and supervised learning

Results The plot on fig. 9 shows two interesting results.

- The supervised model is more likely to predict the optimal move. This was to be expected because it was specifically trained to match the optimal policy on a per-move basis
- However the RL model is more likely to win the game, even if the individual decisions that make up the game are farther from the optimal ones. The difference becomes more noticeable for bigger values of K.

The authors also proved that despite the fact that the RL model makes more errors, it makes less fatal errors (errors that are directly responsible for losing the game).

Takeaways Supervised learning allows efficient learning of an optimal model. However it fails at the "big picture" : the reinforcement learning model is more likely to win the game, even with less optimal individual decisions. Intuitively this makes sense : we trained the supervised model to perform well on a per-step basis, whereas we trained the RL model learning to win the game. These are two different objectives, which reflect the different strengths of both approaches.

We should ask about generalization One thing we are used to in the context of supervised learning is the use of a test set to evaluate generalization, as most supervised models are very prone to overfitting. We were thus surprised that this did not come up in the article's comparison of supervised learning and RL. A sensible thing to do would be to compare the generalization performances of both models. We could even use the approach described in section 3, i.e. testing against a different opponent to evaluate generalization.

Our results Following our remarks in last paragraph, we decided to compare the generalisation properties of reinforcement learning and supervised learning with an experiment similar to the one in the article. This is our protocol :

- Train a defender for a short number of steps and save the observations. These will constitute the test set X_{test}. The training *does not* matter, we only need the observations.
- Train a defender for a bigger number of steps and save the observations. These will constitute the training set X_{train}. The training here *does* matter, and we use the trained model to compute the predicted actions $y_{RL,test}$.
- We compute the optimal actions y_{train} and y_{test}.
- We train a multi-layer perceptron classifier with the same parameters as in the RL setting. We train it with X_{train} and y_{train}, then we compute $y_{sup,test}$.
- We evaluate the correct proportion for multiple values of K

The results are shown on fig. 10. We can see that even when using new data, the supervised method still has a much better per-step performance than the RL method. Our results stay similar to those in the article, at least for small values of K. We could not increase K more for computing limitations.

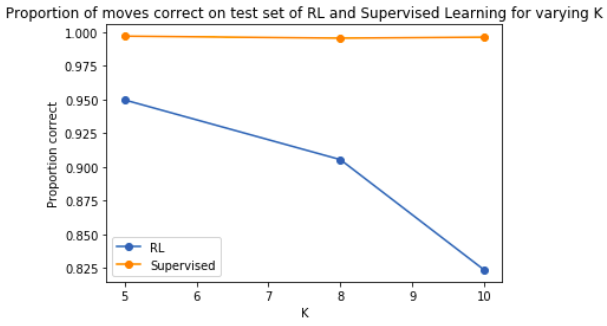

Figure 10: Our results on supervised vs RL

6 Conclusion

- The Erdos-Selfridge-Spencer game attacker-defender is an easy way to rate and compare the efficiency of reinforcement learning algorithms such as DQN, PPO or A2C.
- Thanks to different tuning parameters such as K or the start potential, we are able to find the strengths and weaknesses of these learning algorithms. The paper also provides an optimal behavior under some given constraints to evaluate generalization over raw performance.
- On the other hand, it gives a theoretical insight to implement multiagent play and self play with binary search but it also try to compare the efficiency of Deep Learning versus Reinforcement Learning.

What we learned during this assignment Analyzing this article and playing around with the Attacker-Defender game allowed us to learn the following:

- *Evaluating reinforcement learning methods is not as easy as it seems.* By design, many games are very complex and offer scarce feedback. This means that there is no obvious way to analyse the agent's actions. Games like the Attacker-Defender are very well defined and the fact that there exists an optimal decision offers a perfect measuring stick for performance.
- Generalization in RL is tricky : training an agent against a strong opponent does not mean it will perform better against a weaker opponent. In this case it was *better to train it against mixed strategies*. We learned that overfitting is as much of a problem in reinforcement learning as it is in supervised learning.
- *Reinforcement learning can outperform supervised learning*, even when large amounts of labeled data are available. Supervised learning will probably be closer to optimality on a per-move basis, but still lose more often.

Analyzing an example of a game using our graphical interface

Here is an example of a game with a starting potential equal to 1. The attacker wants the two dots to reach the arrival and he knows that the defender will choose the partition with the higher potential so he isolates the two dots as partition A and the rest as partition B.

$\phi(A) = 2 \times \frac{1}{2^2} = 0.5$
$\phi(B) = 2 \times \frac{1}{2^3} + 2 \times \frac{1}{2^4} + 2 \times \frac{1}{2^5} + 2 \times \frac{1}{2^6} + 2 \times \frac{1}{2^7} + 3 \times \frac{1}{2^8} + 3 \times \frac{1}{2^8} + 1 \times \frac{1}{2^9} = 0.505$

The defender chooses B because he is playing optimally but will loose because in the next step the attacker can choose one dot or the other, he will win.

Figure 11: Attack Strategy with $\phi_{init} = 1$