university of groningen / faculty of science and engineering / bernoulli institute

Playing the Game of Skull - a Multi-Agent Deep Reinforcement Learning approach

Bachelor's Project Thesis

Maria Kapusheva, S3946231, M.M.Kapusheva@student.rug.nl
Supervisors: Dr. Harmen de Weerd

Abstract: The Game of Skull is a multi-player board game, which similarly to poker and other bluffing games, is characterized by its partially observable outcomes. While easy to learn for humans, it poses a challenge to Artificial Intelligence algorithms due to the partial observability and its game mechanics - it has a large number of short stages where different actions are legal. In this paper, we are investigating to what extent Deep Q-Network agents could learn how to play the Game of Skull. Furthermore, by adapting and incorporating the scaffolding learning technique from the field of psychology with our Multi-Agent Deep Reinforcement Learning methods, we are researching if these methods are an effective tool to learn the game and how they compare to the Vanilla DQN agents. According to the results outlined in this paper, the agents successfully learn to play the game and consistently reach the final stage, however, using those algorithms results in deterministic agents that deploy rigid strategies, and can hardly adapt to new playing styles. In addition to that, the agents that learned through the scaffolding technique perform slightly better than Vanilla DQN agents, which is a possible direction for future research on the topic.

1 Introduction

Multi-agent Deep Reinforcement Learning (MADRL) refers to the usage of deep reinforcement learning methods in a multi-agent environment. The investigation of multi-agent problems is slowly gaining traction due to its relevancy in real-world applications (Gronauer & Diepold, 2022). These problems include coordination of autonomous vehicles (Shalev-Shwartz, Shammah, & Shashua, 2016), traffic control (Ma & Wu, 2020) and financial trading models (Lux & Marchesi, 1999). As multi-agent learning is inherently more complex and computationally expensive than single-agent learning, games are often used as stepping stones in investigating that field. For instance, Multi-Agent Reinforcement Learning(MARL) models have been used to tackle some of the most challenging esports such as Starcraft II (Vinyals et al., 2019) and Dota 2 (Berner et al., 2019), achieving performance comparable to the top human players in those games.

Game theory plays a big role in formally representing MARL models. As described by Koller and Pfeffer (1997), one way of classifying games would be whether the agents have perfect or imperfect information for the current state of the world. A large portion of the existing research focuses on games with pimpeformation such as Chess, Shogi, and Go (Silver et al., 2017). However, complete state observability is rare in real-life scenarios, as often either the environment or the interactions between the agents contain some uncertainty. Therefore, imperfect information games are valuable research topics due to their possible applications, e.g. auctions, cybersecurity, negotiation (Brown & Sandholm, 2017). Some examples of these types of games are Poker with all its variations, Bridge, and Liar's Dice.

This research will be focused on one imperfect information game - Skull (Gragera Aguaza, Baffier, & Suppakitpaisarn, 2015). The game of Skull is a multiplayer turn-based card game characterized by little stochasticity and partially observable outcomes. According to Gragera, Baffier, and Suppakitpaisarn (2013), Skull has a shorter learning curve for humans than some other popular turn-based games such as Go and Shogi due to its large number of short stages. The number of possible actions is small in each stage, which makes the gameplay intuitive. However, due to its game mechanics, which will be explained further in this paper, implementing an AI strategy can be challenging.

In this research, the Game of Skull will be formalized as a Partially Observable Stochastic Game (POSG)(Hansen, Bernstein, & Zilberstein, 2004). As a POSG, the Game of Skull provides an appropriate environment and rules for implementing a Multi-Agent Deep Reinforcement Learning algorithm (Du & Ding, 2021). One way of implementing MADRL is by using a Deep Q-network (DQN)

model (Mnih et al., 2013). In a DQN, an action and a state value are used as input for a neural network in a Q-learning framework. The output of the network is the estimate of the reward given the input state and action. However, as described by Heinrich and Silver (2016), DQN learns a deterministic strategy that can impose an obstacle in imperfect-information games. A deterministic strategy could be a disadvantage due to it being very predictable in a deterministic environment and can essentially take away the bluffing aspect of the game. Furthermore, a DQN agent learns the best response to the historical experience, and consequently, if there is a change in the opponent's strategy, the performance of the DQN agent might stagnate. Therefore, it is relevant to investigate how a DQN algorithm performs in a MARL model of the imperfect-information game Skull and more specifically, to what extent can the DQN agents learn to play the game. Furthermore, how would different hyperparameters for the Deep Q-Networks influence the gameplay of two agents that train and play against each other?

When it comes to single-agent environments, OpenAI's Gym (Brockman et al., 2016) has provided a standardized API and a wide collection of environments with a common interface, which prompted and facilitated the research in the field. However, despite Gym's success in single-agent modeling, it is not thoroughly optimized for MARL as described by Terry et al. (2021). For that reason, in this research, we will utilize PettingZoo (Terry et al., 2021), which is designed to be a Multi-agent equivalent to Gym. PettingZoo takes advantage of the Agent-Environment Cycle (AEC), which provides a novel approach for multi-agent games.

The rest of this research will explore the two-player game in three different experiments. The first experiment will be focused on the normal implementation of the DQN algorithm and the effect of different training durations and values of the exploration coefficient. After that, we will introduce a modified version of the DQN algorithm using scaffolding that aims to improve the learning of the different phases. And finally, in the last experiment, a tournament will be held to determine how the algorithms perform against each other.

2 Background

2.1 Game of Skull

Originally, the Game of Skull is an analog multiplayer card game that involves bluffing. The number of players can be between two and six, but in this paper, we will focus on the two-player setup. At the beginning of the game, each player is dealt four cards - one skull and three roses. Afterward, the possible actions a player can take depend on the current phase of the game. The game consists of three different phases. In the first phase, which we will call the *playing cards phase*, the players put a card face-down on the board in turns. Once all players have placed at least one of their cards, they can either play another card or place a bet. If a player decides to place a bet, they would essentially have to declare the number of cards with a rose symbol, that they can flip from the face-down cards on the table. When there is a bet in place, the second phase, or *the betting*, begins. In this phase, which is again sequential, the players can either raise the bet, if they believe that they can flip more roses, or pass a turn. After all of the players, besides the one that raised the highest bet, have passed a turn, the game enters into the last phase - *the flipping phase*. This phase is substantially different than the previous two, as the only allowed action is to flip a card from the ones on the board. Additionally, there are no turns, i.e. only one player (the one that bet the most previously) acts in this phase. This player has to flip the same number of cards as the bet that they raised. If all of the cards they flip are roses, they win the game. Otherwise, they lose if they flip a skull.

As mentioned before the focus of this study will be on the two-player variant of the game. As a result of that, it is appropriate to formalize the game as a two-player zero-sum game. This indicates that the sum of the payoffs of the players will be zero at the final stage, in other words, when one player wins, the other loses, and vice-versa.

2.1.1 Partially Observable Stochastic Games

When an agent is playing the Game of Skull, they do not know explicitly the actions of the other player. For instance, the cards of the opponent are placed with their face downwards, as well as, the opponent might bet more than they can successfully play in order to deceive others. Therefore, when the game is analyzed from the point of the players it is stochastic, although its action space is deterministic in nature. Furthermore, this also implies that the game has partially observable states since the agents are only able to definitively know their own actions. As a result, we will represent the Game of Skull as a Partially Observable Stochastic Game mark(POSG).

As described by Hansen et al. (2004) POSG is a tuple of $\langle \mathcal{I}, \mathcal{S}, \{b^0\}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R} \rangle$ where:

- $\mathcal{I}$ is a finite set of agents, in this case: $\{i, j\}$
- $\mathcal{S}$ is a set of states
- $b^0 \in \Delta\mathcal{S}$ indicates the initial state

- $\mathcal{A}$ is the set of actions
- $\mathcal{O}$ is the set of observations
- $\mathcal{P}(s'|s,a)$ is the set of transition probabilities given state s and an action a
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathfrak{R}^I$ is the reward function for the agents

2.2 Reinforcement Learning

Before we continue with our implementation, it is important to explain the fundamentals of the RL and DRL in order to provide the appropriate context for this research. In essence, reinforcement learning is the process of learning through interacting with an environment and receiving positive or negative feedback as the result of that interaction. This feedback then propagates change in the agent's behavior depending on the differences between the outcome of their actions and the desired outcome, similar to how humans learn a new task. Respectively, when RL is applied to a multi-agent problem, the joint actions of all of the agents determine the rewards and the state transitions.

Most of the problems in RL are formalized as a Markov Decision Process (MDP), however, for the Game of Skull, we will use a POSG model, which closely resembles a Partially Observable Markov Decision Process (POMDP). The main difference between a POSG and a POMDP is that the former includes a set of agents. Nonetheless, they both are characterized as problems that have the Markov property. Meaning that the current state provides complete information for future decision-making, and therefore a transition model can be successfully used to map a state-action pair at time t to the probabilities of transitioning to a state s_{t+1}. In order to illustrate that better, we can compare an MDP to an environment that does not possess the Markov Property. In such an environment, the probability of moving to a state s_{t+1} from a state s_t when making an action a_t depends on all of the previous actions and spaces.

In contrast, in a Markov environment, the next states depend only on the current state:

$$\mathcal{P}(s'|s,a) = Pr\{r_{t+1} = r, s_{t+1} = s'|s_t, a_t\} \quad (2.1)$$

This results in a more efficient estimation of the state transitions, as the previous game states are effectively irrelevant. This is important in the context of Skull, as it provides clear constraints for how to represent the states of the environment.

In a general POMDP model the state of the environment at timestep t can be represented as state $s_t \in \mathcal{S}$. Consequently, an agent in such an environment observes the state, performs an action $a_t \in A$, and the environment transitions to a new state s_{t+1}. Usually, a reward $r_t \in \mathcal{R}$ is given to the agent after they perform action a_t. However, in the Game of SKull due to the partial observability and the large gamespace of the environment, a reward will be given only after the end of a full game cycle.

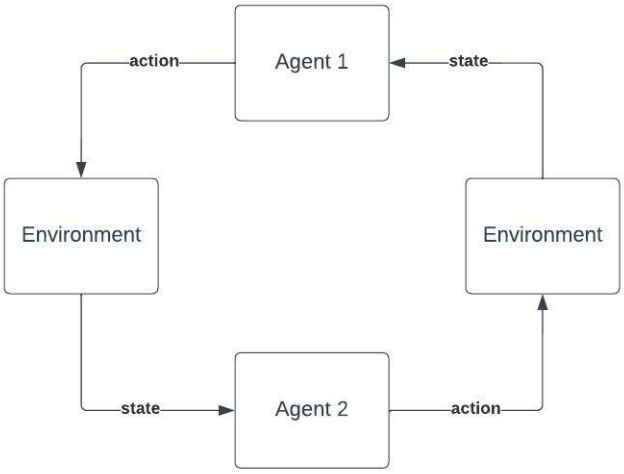

Figure 2.1: Agent-Environment Cycle

During the training of the RL agents, a number of games are repeated. The reason for this is that the agent is trying to gather the information that would allow for devising an optimal policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected sum of the future rewards.

When it comes to multi-agent environments this needs to occur for multiple agents at a time, where each agent should devise their own policy, based not only on their actions but on their opponents' actions as well. In the two-player case, the agent would receive an observation o_t of the state s_t, make an action a_t and the state of the environment will change to s_{t+1}. It is important to note that the observations give some, but not complete, information on the state of the game. Afterward, the other agent receives an observation o_{t+1} of the state s_{t+1}, makes an action a_{t+1} and the state transitions to s_{t+2}. This cycle is illustrated in figure 2.1.

2.2.1 Q-learning

There are multiple ways to solve a RL problem, depending on the available information. For example, if the transition model is available, or the agents are receiving enough information to estimate it, the problem can be solved through model-based learning. In such scenarios, the agents are trying to estimate a model of the environment in order to maximize the rewards of every action entirely based on the transition probabilities. Another approach would be to use model-free algorithms, where the agents are adjusting their policy depending on the consequences of their actions. In this study, the transition model is only known from an outsider's

perspective, therefore we will be approaching the problem with model-free learning. One way for implementing model-free learning is Q-learning. This method allows us to use a state-action-value $Q^{\pi}(s, a)$ to calculate the expected return(see equation 2.2), instead of using the transition probabilities provided by the model $\mathcal{P}(s_{t+1|s_t,a_t})$. In this formulation, γ is used to denote a discount factor between 0 and 1, which determines the evaluation of the rewards, e.g. if γ is higher future rewards would be favored over immediate rewards and vice-versa. This will be further explained later on.

$$\mathcal{Q}^{\pi}(s, a) = E[\sum_{n=0}^{\infty} \gamma^n r_{t+n+1} | s, a, \pi] \tag{2.2}$$

Due to the aforementioned Markov Property, the Q-value of the future state could be used to improve the estimation of the current state. This is referred to as bootstrapping and an agent can use it to learn a policy Q^{π}. This process is formalized by the Bellman equation shown in 2.3.

$$Q^{\pi}(s_t, a_t) = E_{s_{t+1}}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, \pi(s_{t+1}))] \tag{2.3}$$

2.2.2 Deep Q-Networks

For problems with full observability and with small state and action spaces, we can compute the Q-values for each state-action pair and store them in a Q-table. However, due to the complex game space of Skull (Gragera et al., 2013), this is not an appropriate solution to our problem. Therefore, a Deep Reinforcement Learning algorithm such as a Deep Q-Network is a more suitable approach.

The DQN is a neural network that maps the input observation to an output action which is essentially represented by the predicted Q-value. As a consequence, the action that an agent would choose would be the one with the highest predicted Q-value. After each training iteration, the Temporal Difference target is calculated and updated using the Bellman equation (see Equation 2.4). Afterwards the DQN can be retrained with the next batch.

$$Q(S_t, a_t) = (R_t + \lambda \max_a Q(S_{t+1}, a)) \tag{2.4}$$

2.2.3 ϵ-greedy

We will be using the ϵ-greedy Exploration strategy (Sutton & Barto, 2018)to choose an action in order to assure exploration in the initial stages of the training and exploitation in the later ones. Usually, it starts with a high value of 1 and it decreases linearly during the training until it reaches the predefined minimum. The probability of an agent choosing to explore a random action is ϵ and the probability to exploit the predicted best action by the DQN is $1 - \epsilon$. The value of ϵ is a probability, therefore it is between 0 and 1.

2.2.4 Experience Replay

In our algorithm, we will be using Experience Replay in this implementation. This a technique, in which the states of the game are stored in batches of size N and then used to train the agent. On one hand, this proves efficient in terms of time complexity, as the agent only needs to be trained at every episode N rather than at each time step. Its other benefits are that, instead of using the collected data once and throwing it away after training, it can be reused multiple times and it improves the stability of the DQN as discussed by Fedus et al. (2020). Because of that, it is less likely that the agents forget how to play the games in the first part of the training.

3 Methods

3.1 Skull Environment

Due to the lack of preexisting environments for the Game of Skull, in this research, we have created and used a custom one built in Python. In order to take advantage of the Agent Environment Cycle(see Figure 2.1) that PettingZoo offers, this custom environment inherits the *AECEnv* class provided in their documentation. An AEC reinforcement learning environment in Python requires a *__init__*(), *reset*(), and *step*() functions. The other important aspects are the representation of the state, rewards, and the action space.

3.1.1 Observations

Due to the partial observability of the environment, the agent's representation of the state is done through observations. In the game of Skull, a player has knowledge about the cards in their own hand, the cards that they played, the number of cards that the opponent played, the current bet, and the number of cards that have to be flipped once the flipping phase starts. The cards that the opponent has played and whether they are bluffing during the betting phase are unknown. The observations only include information about the current state of the game, since, due to the Markov Property, the next state only depends on the current one. In the implementation within the environment class, this is represented through a dictionary consisting of discrete values. The *step*() and the *reset*() function however return a one-dimensional

named tuple with the same values in order to provide a vector input for the DQN algorithm of the agent.

The observations in our implementation consist of the following 12 discrete values:

- **self(self0, self1, self2, self3):** The number of cards that the agent making the observation has played.
- **opponent(opp0, opp1, opp2, opp3):** The number of cards that the opponent has played.
- **current_bet:** The raised bet so far in the game
- **cards_to_flip:** The number of cards that have to be flipped in the last phase of the game
- **skull:** The location of the skull of the current player
- **phase:** The current phase of the game

3.1.2 Action Space

In Skull, the player essentially has 5 actions that he is able to make through a game(playing a skull, playing a rose, betting, passing, and flipping), however, whether or not those moves are legal depends on the phase. For example, an agent cannot start flipping cards when there is no bet in place in the playing cards phase of the game. Therefore, the action is represented through a single value with 12 possible discrete outcomes. The possible actions in this representation are:

1. play skull
2. play rose
3. raise bet
4. pass on betting
5. flip the card in slot 0(from the agent's own cards)
6. flip the card in slot 1(from the agent's own cards)
7. flip the card in slot 2(from the agent's own cards)
8. flip the card in slot 3(from the agent's own cards
9. flip the card in slot 0(from the opponent's cards)
10. flip the card in slot 1(from the opponent's cards)
11. flip the card in slot 2(from the opponent's cards)
12. flip the card in slot 3(from the opponent's cards)

3.1.3 Rewards

As previously explained, the game is a two-player zero-sum game, which means that the agents receive rewards only after a full game cycle is completed. Following that formalization, the rewards of the game are 1 for the agent that won, and -1 for the opponent. Furthermore, the game has differing legal and illegal actions in the three stages. Rather than including a knowledge-based rule-following, we chose to give a reward of -100 when an agent performs an illegal action. If that is the case, the reward for the opponent is 0.

3.1.4 Game mechanics

In our implementation, the $\_\_init\_\_()$ function initializes and sets all of the parameters to their default values. Since the starting state of a Skull game is deterministic, these values are the same for every start of a game. Consequently, the $reset()$ function is essentially the same as the $\_\_init\_\_()$, however, the $\_\_init\_\_()$ function does not return anything, while the $reset()$ returns the initial observation which is used to train the model.

It is important to note that usually in multiplayer games, the agents are still trained through self-play before facing the opponent. In this implementation, however, two agents train while playing against each other, which leads to sequential learning of the two agents due to the turn-based nature of the game. In other words, while illegal actions are expected from both parties, the agent that starts a game first will be able to figure out which actions are illegal before the other agent gets the chance to play. As the turn of the game is not randomized, i.e. agent 0 always plays first, we can see this trend clearly through the training.

3.1.5 Deep Q-Network

As explained previously, we will be using a Deep Q-Network with Experience Replay and an ϵ-greedy policy to train the agents. We chose a three-layer design for the neural network - an input layer, a hidden layer, and an output layer. This was implemented with PyTorch and our DQN class inherits its nn.Module(). The input of the neural network is a one-dimensional vector of size 12, and we directly use the observation returned by the step() and the reset() functions of the environment class for that. The output layer is also a one-dimensional vector of size 12, but it indicates the best possible action based on the input observation. The hidden layer is of size 256. The layers use a Relu activation function. For more information on the layers, weights, and biases of the Deep Q-Network see Listing 1. We use the Mean Squared Error loss function provided by PyTorch's nn.Module() to calculate the loss, as

it is one of the most commonly used functions in RL problems. For the experience replay batches of size 64 were used.

Listing 1: DQN State Dictionary

```
    fc1.weight
torch.Size([256, 12])
    fc1.bias      torch.Size([256])
    fc2.weight
torch.Size([256, 256])
    fc2.bias      torch.Size([256])
    fc3.weight
torch.Size([12, 256])
    fc3.bias      torch.Size([12])
```

3.1.6 Agents

The agent class effectively represents a player in the game of Skull. Using PyTorch in this context allowed us to calibrate precisely the hyperparameters. In the *__init__*() function of this class, we can set up a γ, ϵ value, ϵ decay coefficient, minimum ϵ, learning rate, maximum memory size, as well as the shape of the observation and the number of actions that are available to the agent. Additionally, each agent has a Q_eval device, which is essentially an instance of the Deep Q-Network class explained above.

Each agent has three important functionalities - storing transitions for the Experience Replay, choosing an action, and learning. Choosing an action is either done randomly while the epsilon is high in order to promote exploration in the initial episodes of the training or based on the predicted best action by the DQN of the agent in the later epochs. After each step, the agent stores the transitions of one state to the other in its state, new state, reward, action, and terminal memory slots. These transitions are eventually used for learning as they are sent in batches to the DQN in the form of tensors. Subsequently, the loss is calculated and the epsilon is decreased if it has not already reached the minimum possible value.

4 Experiments

Three experiments were designed in order to test the extent to which an agent can learn to play the game in such a setting. In all three experiments, Agent 0 refers to the agent that plays the first move in the game, and Agent 1 to the second player.

4.1 Experiment 1

The first experiment is aimed to investigate how a Vanilla DQN algorithm performs in the setting and what is the behavior of the agents after they learned to play the game in such a way. We conducted three trials with a different number of games played in each of them. Naturally, we chose different values for the epsilon decay parameter in the three trials, as we changed the number of games played. We chose very small numbers to represent that because exploration is highly desired when learning the game. This is the case due to different actions being legal and illegal in the three stages of the game. After each trial, the playing behavior of each agent will be investigated over 100 games, where the ϵ will be set to 0 to avoid random actions during the testing.

	Training 1	Training 2	Training 3
Average Score Agent 0	-0.36	0.84	0.28
Average Score Agent 1	0.02	-0.3	-5.48

1. Training agent0 and agent1 with $eps\_decay = 5 * 10^{-4}$ for 10 000 games
2. Training agent0 and agent1 with $eps\_decay = 5 * 10^{-5}$ for 100 000 games
3. Training agent0 and agent1 with $eps\_decay = 2 * 10^{-5}$ for 300 000 games

4.1.1 Results

We ran all three trials of the experiment. The general pattern of learning of the agents throughout training is relatively similar - Agent0 has lower scores than Agent1 in the first part of training, which then increases while Agent1's scores decrease. During the testing, Agent 0 was exclusively the winner for all games in trial 2 and trial 3(see Figure 4.4 and 4.6). In trial 1, however, as we can see on Figure 4.2, Agent 1 won all of the games. Regarding the playing behavior, the agents in each trial, were playing the same game over and over. In trial 1, the bet at the last stage was 4 and the game ended due to Agent 1 flipping 4 roses. In trial 2 and 3, the final bet was 1.

4.1.2 Discussion

After running the experiment, we can see that a longer training period does not necessarily indicate a better performance at the last stages of training. Furthermore, the learning patterns are heavily influenced by the design of the game mechanics in this research. Due to Agent0 always having the first turn it learns how to play the game first, therefore it is also not surprising that in two out of the three trials, Agent0 has a higher average score than Agent1. However, as expected the vanilla DQN algorithm is very rigid in terms of learning a strategy, which is shown by the fact that the agents repeat the same gameplay. Considering that the game is deterministic, the action space is deterministic,

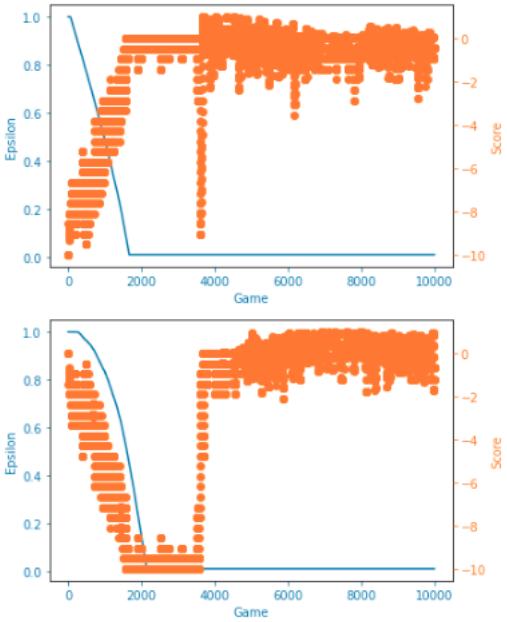

Figure 4.1: Experiment 1 Trial 1 Training Agent0 & Agent1

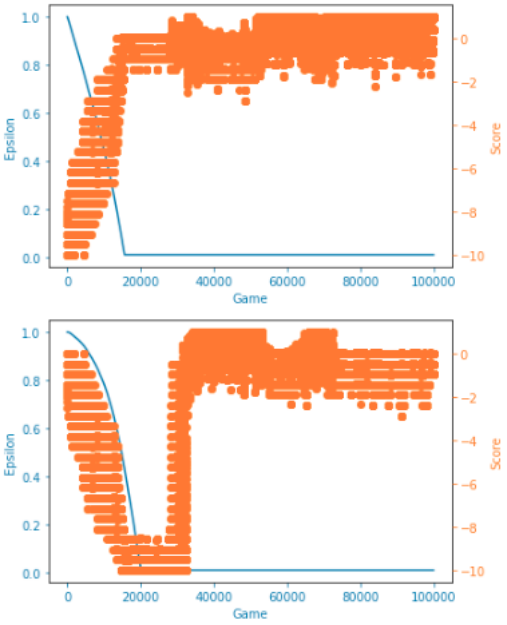

Figure 4.3: Experiment 1 Trial 2 Training Agent0 & Agent1

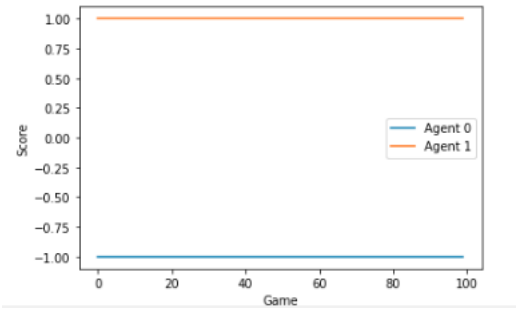

Figure 4.2: Experiment 1 Trial 1 Testing Agent0 & Agent1

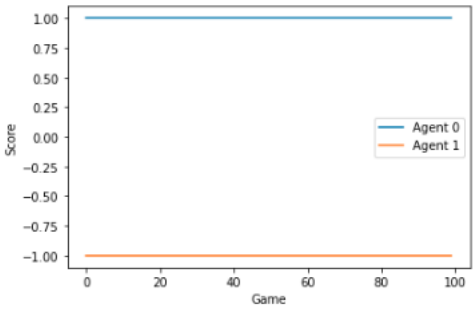

Figure 4.4: Experiment 1 Trial 2 Testing Agent0 & Agent1

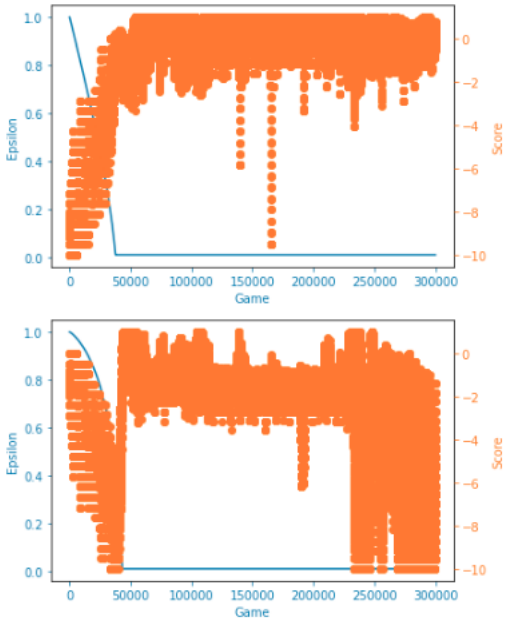

Figure 4.5: Experiment 1 Trial 3 Training Agent0 & Agent1

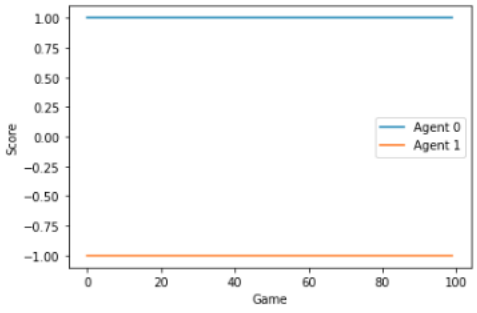

Figure 4.6: Experiment 1 Trial 3 Testing Agent0 & Agent1

and the agents act in the same manner once they are trained, we can conclude that the agents also act deterministically.

4.2 Experiment 2

The second experiment focuses on a different approach. As the game consists of three phases, the training is split into three parts. Due to our formalization of the problem as a zero-sum one, we cannot provide the agents with intermediate rewards. Thus, we adopted an approach similar to the scaffolding technique (Zydney, 2012) in psychology. Initially, the agents are trained for 100 000 epochs only in the third stage of the game. The initial observations are randomly generated complying with the rules for a valid state in this phase. The turn of the agents is also random because this stage is essentially a single-player subgame. Afterward, the models are saved, the epsilon coefficient is reset and the agents are retrained from the second phase onward. The initial observations are again randomly generated based on the rules of the game. Finally, we save the models and retrain them once more but on the whole game, i.e. all three phases.

1. Training agent0 and agent1(same hyperparameters) with $eps\_decay = 0.0001$ for 200 000 games only on the third stage

2. Reusing the model and training on the second and third stages with the same *eps_decay*

3. Reusing the model and training on all stages with the same *eps_decay*

Afterwards, the final models will be tested for a 100 games with ϵ set to 0.

4.2.1 Results

During the first part of this experiment, where the agents learn by only playing the last phase of the game, we can already see a substantially different learning pattern compared to the previous experiment (see Figure 4.7). The agents are no longer learning sequentially, but rather in a randomized manner, which results in very similar learning curves throughout the training period of this part of the game.

A similar pattern can be noticed during the second part of training (see Figure 4.8), where the agents start from the second phase and continue until the end of the game. However, during the last part of training, where the initial turn was no longer randomized, these agents learn in a similar way to the agents in Experiment 1 (see Figure 4.9). After Agent 1's initial lead in the games, we can see a drop in its performance, and a consequential

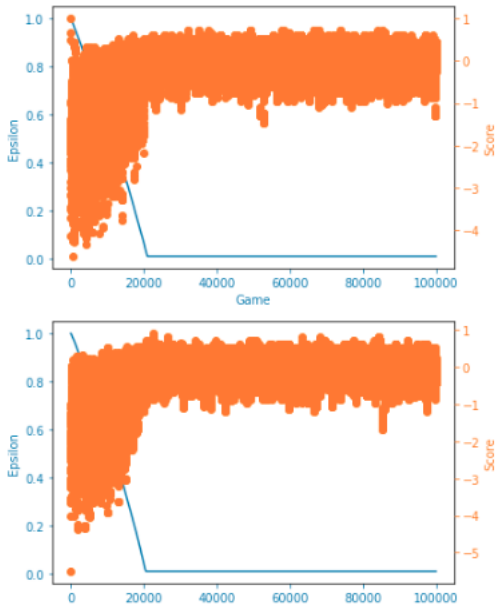

Figure 4.7: Experiment 2 Part 1 Training Agent0 & Agent1

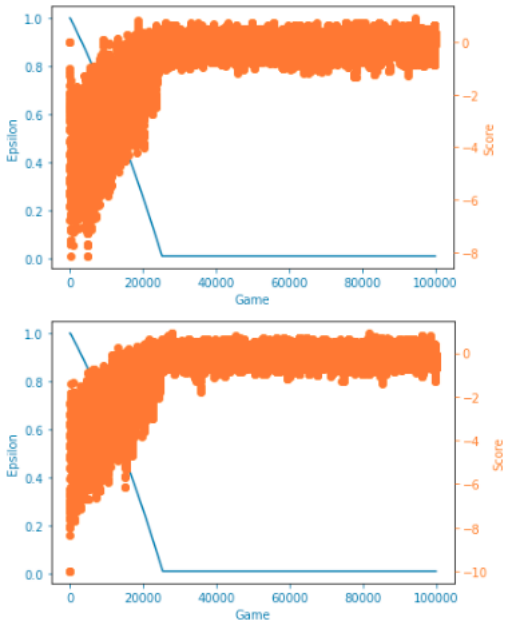

Figure 4.8: Experiment 2 Part 2 Training Agent0 & Agent1

improvement of the performance of Agent 0, which continues until the end of the training period. The average scores were 0.15 and -1.25 for Agent 0 and Agent 1 respectively.

Subsequently, the agents were tested once again on a 100 games (see Figure 4.10). The agents were once again repeating the same gameplay over and over, with Agent 0 being exclusively on top. In all of the games the bet at the final stage was 1, which means that Agent 0 only had to flip one rose card to win the game. That indicates that Agent 1 was effectively losing the game over and over by passing on their last turn.

4.2.2 Discussion

During the first two parts of the experiment, we can see how a randomized starting turn completely changes the learning patterns compared to a fixed turn. Furthermore, this results in a more normally distributed training through this period, where there is no clear player leading at any given point between Agent 0 and Agent 1. However after letting the agents play the whole game with a fixed turn, we can see the familiar pattern from Experiment 1 in the beginning of the training. In addition to that, the results and the playing behaviour also overlap with the ones from Experiment 0 - for every game without illegal actions, the same gameplay is repeated and one of the agents wins consistently. Therefore, we can again conclude that the agents act deterministically 99% of the time as they only have 0.01(ϵmin) chance of performing a random action. However, the results from the randomized turn in the first two parts of this experiment lead

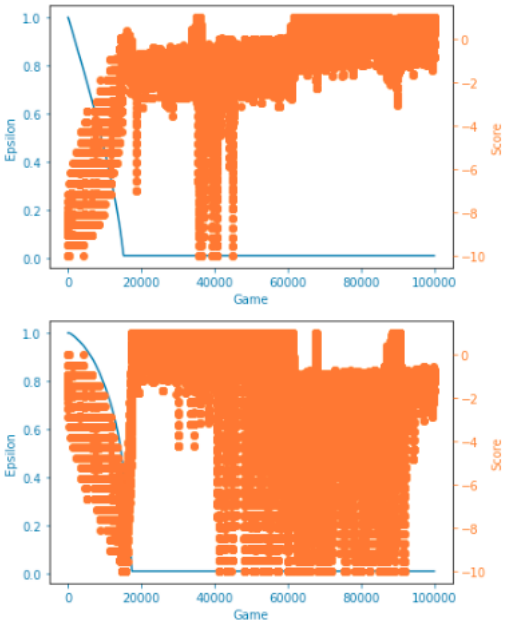

Figure 4.9: Experiment 2 Part 3 Training Agent0 & Agent1

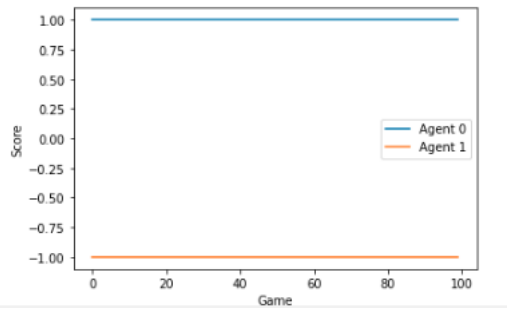

Figure 4.10: Experiment 2 Testing Agent0 & Agent1

to the question if the stochasity introduced by making the starting state uncertain, would lead to a substantially different behaviour.

4.3 Experiment 3

The goal of this experiment is to determine whether the different ways of learning the game that we outlined in Experiments 1 and 2 provide an advantage in a competitive setting. For this purpose, during our first two experiments, we saved agents at different checkpoints in the last stage of the training, depicted in Figure 4.11. As we saved both agent 0 and agent 1 at each checkpoint in each experiment, we ended up with 20 agents. However, as the turn in the game is constant, i.e. agent0 always starts the game, and in addition to the conclusion after Experiment 1 and 2 that the agents act deterministically, we will always let an agent0 start first and play against an agent1 in the tournament. Furthermore, as agents from the same experiment have already been tested against each other, we will only explore games where agents from different experiments compete. Therefore, there will be in total 50 games. The exact pairing in those games is explained in Table 4.1.

4.3.1 Results

The experiment resulted in a large number of games ending due to an invalid action - 19 out of 50, with the larger part of them(14 games) being when the starting agent was from the first experiment. The complete results of this experiment can be found in Appendix B.

- **Games 1-25:** *Agent 0 from Experiment 1, Agent 1 from Experiment 2*

  During the first 25 games, Agent 1(from Experiment 2) caused 5 of them to end due to making an illegal move, while Agent 0(from Experiment 1) played legally in all of them. In addition to that, during the valid rounds from these 25 games, Agent 0 won 13 times, while Agent 1 won only 5 times. However, while agents playing first from Experiment 1 Checkpoints 1-3 were exclusively winning, the ones from Checkpoints 4 and 5 were losing against the agents from the second experiment. In terms of their playing behavior, it is worth noting that each Agent 0 rarely changed their card placing strategy, and consequently, that led to the same behavior in the multiple Agents 1, that played against the same opponent. The bets that were played were around 3 on average, reaching a maximum of 5 in the first two games and a minimum of 1 in games 4 and 19.

- **Games 26-50:** *Agent 0 from Experiment 2, Agent 1 from Experiment 1*

  Throughout games 26 to 50, the agent that played first was from the second experiment, and its opponent was from the first experiment respectively. Agent 1 performed an invalid action in all 14 games that ended for this reason. In terms of the 11 valid games, Agent 0 from Experiment 1 was almost exclusively winning, more specifically, Agent 0 won a total of 10 games. In terms of playing behavior, the agents place their cards in a similar way as the agents in games 1 to 25, however, the placed bet in all valid games but game 31 is 1.

Overall, out of the 31 valid games, the agents from the second experiment won a total of 17 times, while the ones from the first experiment - 14 times.

4.3.2 Discussion

In this experiment, we can see a distinct strategy in all of the games that are started and won by Agent 0. This strategy was not explicitly present when the first player was from Experiment 1. Mainly, the bet does not rise above 1, and all of the games are won due to Agent 0 flipping all cards without revealing a skull. This perhaps indicates that the Agents that learned through scaffolding found a loophole in the strategy, by deducing that the higher bet leads to a higher chance of flipping a skull. It is important to note that this behavior was not present while the agent playing first was trained in the first experiment.

Nonetheless, it cannot be ignored that a large part of the games that were played in Experiment 3 ended due to an illegal action being made by one of the agents. This is supported by the findings of Heinrich and Silver (2016), which note that the DQN algorithm leads to learning a deterministic strategy, and therefore it is sub-optimal against new opponents. Due to that, although the agents that learned through the scaffolding technique performed better overall, we cannot derive any definitive conclusions due to the small margin in win

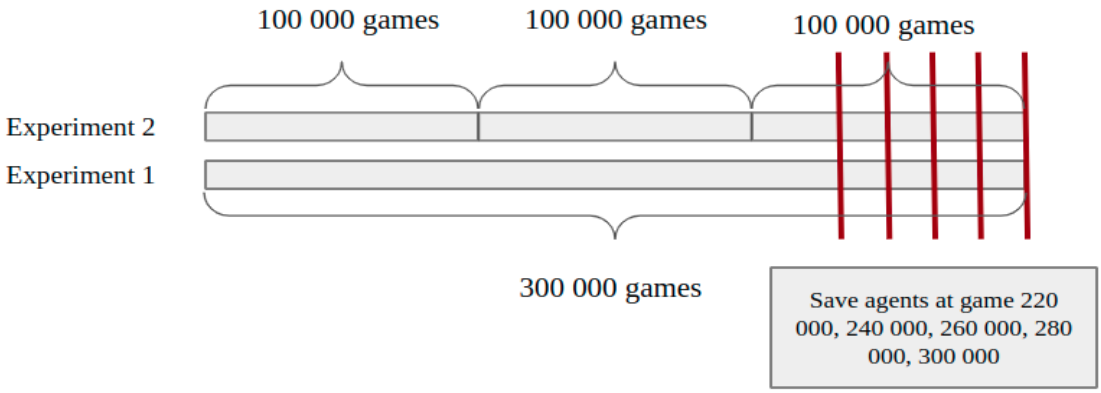

Figure 4.11: Experiment 3 - Setup

	Agents	Opponents					
Agent 0 Experiment 1 vs Agent 1 Experiment 2	Checkpoint 1	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 2	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 3	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 4	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 5	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
Agent 0 Experiment 2 vs Agent 1 Experiment 1	Checkpoint 1	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 2	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 3	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 4	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5
	Checkpoint 5	Checkpoint 1	Checkpoint 2	Checkpoint 3	Checkpoint 4	Checkpoint 5

Table 4.1: Experiment 3 - Tournament

rates and a large number of invalid games. In addition, it seems like there is a first-player advantage in the game. In both combinations of players, the one that acts first outperforms the other.

Regarding the illegal actions, they were mostly made in the second and third phases of the game when they were caused by the Experiment 1 agent. On the other hand, in all 4 invalid games that an Experiment 2 agent caused, the invalid move was made in phase 1. Since agents from Experiment 1 caused 14 out of the 19 games to end illegally, we suppose that it was caused because the Experiment 1 Agents were not introduced to a large variety of phase 2 and phase 3 gameplays. And respectively, perhaps the games that ended in phase 1 were caused because the Experiment 2 Agents were not trained sufficiently in the first phase by that checkpoint of their training.

5 Discussion

The first and the second experiments that were conducted were in general successful in terms of investigating the learning and playing behavior of DQN agents in the context of Skull. They were consistently reaching an end game. Regarding the small number of games that ended due to an illegal move, we suppose that it was caused by encountering a completely unfamiliar observation space. In this research, we made very specific decisions regarding the checkpoints for saving the agents, the learning steps, and the hyperparameters of the neural network. Some of them were chosen based on preexisting literature, others due to the results of the first and the second experiment. The number and the position checkpoints were chosen with the purpose of saving the most amount of versions of agents that are trained on the whole game while allowing for large enough training periods in between the checkpoints so that the agents could learn more, or start implementing a different strategy. Perhaps different design choices could improve the agent's performance, but it would not lead to a drastically different strategy.

Due to the game having simple rules in general, perhaps a knowledge-based approach in combination with reinforcement learning could prove more efficient in minimizing invalid games. A point for future investigation would be to map actions to a specific phase and encode the information before letting the agents play, similar to the approach used by Nan, Perumal, and Zaiane (2022). Possibly, this might reduce the duration of the training in addition to the number of invalid games.

Furthermore, in terms of playing behavior both the agents in Experiments 1 and 2 performed similarly when tested - they were playing the same game over and over. These results overlap with the conclusions in the analysis of the game by Gragera et al. (2013) and provide an overview of what a straightforward learning curve for a DQN agent looks like. Perhaps an algorithm that estimates an optimal non-deterministic strategy could prove more efficient in the context of playing against multiple different opponents or at least very different strategies. Moreover, the Constructivist Anticipatory Learning Mechanism proposed by Perotto, Buisson, and Alvares (2007) seems like a promising learning mechanism for partially observable ad partially deterministic environments such as the Game of Skull. Another approach for training more well-versed agents would be to introduce more stochasticity in the environment. For instance, during the first and the second part of the training in Experiment 1, we observed more randomly distributed results due to the 50% chance of an agent being the first player instead of having a fixed turn. Such an investigation could lead to agents that play equally "well" when starting first and when starting second, instead of being specialized in being the first or the second player.

Subsequently, the scaffolding, which indeed allowed the agents to successfully and consistently reach a final state of the game, essentially broke down the game into simplified problems that the agents would have to learn one at a time before tackling the game as a whole. This approach produced slightly better results than the Vanilla DQN algorithm, and although not enough to draw any meaningful conclusions, a more in-depth investigation of this method in different environments or in combination with different reinforcement learning algorithms would be an interesting topic for future research.

Undoubtedly, the most interesting result in this paper was that the scaffolding method lead to agents finding a "loophole" in the game while the Vanilla DQN agents did not despite playing in the same environment and being trained with the same algorithm. In addition to that, the scaffolding agents performed fewer illegal actions against an entirely new opponent than the Vanilla DQN agents in the same setting. The duration of the training for these two experiments was the same, therefore, this result could be attributed to the fact that the scaffolding agents had the opportunity to deploy the ϵ-greedy exploration strategy independently in each of the three phases. This led them to explore more in the second and third phases of the game. Moreover, due to the random generation of the initial states in the first and second part of the training in Experiment two, they perhaps were exposed to a larger number of possible observation states which lead to slightly better learning of the game. In comparison, the epsilon of the agents from the first experiment reached its minimum at

approximately 1/6 of the training, and through this period the games were rarely entering stage 2 and stage 3. Consequently, for almost 90% of the training, their probability of acting randomly was only 1%. This suggests that they mostly explored the different strategies in the first phase, and acted even more deterministically in the later phases. In conclusion, the scaffolding method could be a valuable tool in environments with a large number of sequential stages as it allows an exploration strategy to be deployed in each one of them.

6 Conclusions

The experiments that were conducted in this research have been mostly successful in playing the Game of Skull through MADRL methods. Despite the fact that the agents were playing deterministically, which would not result in optimal gameplay against a human, there are some key takeaways that can be drawn from this study.

1. Both the Vanilla and the scaffolding DQN algorithms could be successfully used to teach an agent how to play the Game of Skull.
2. Using a Deep Q-Network is not the most optimal algorithm for learning how to play the Game of Skull against new opponents, as it results in an agent with a rigid strategy optimized to play against another deterministic agent.
3. Using the scaffolding technique in problems with a large number of short stages might prove beneficial for finding unexpected strategies.
4. A scaffolding DQN algorithm seems to perform slightly better than a Vanilla DQN in the context of the Game of Skull.
5. A partially observable game with a deterministic initial state and deterministic actions seems to lead to deterministically behaving agents, despite the partial observability.
6. In similar problems scaffolding is effective in terms of introducing more data to the agents during training, therefore it is a more suitable approach than the Vanilla DQN algorithm. However, more research is needed to determine its applicability in other scenarios.

References

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., ... others (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI gym. *arXiv preprint arXiv:1606.01540*.

Brown, N., & Sandholm, T. (2017). Safe and nested subgame solving for imperfect-information games. *CoRR*, *abs/1705.02955*. Retrieved from `http://arxiv.org/abs/1705.02955`

Du, W., & Ding, S. (2021). A survey on Multi-Agent Deep Reinforcement Learning: from the perspective of challenges and applications. *Artificial Intelligence Review*, *54*(5), 3215–3238.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., & Dabney, W. (2020, 13–18 Jul). Revisiting fundamentals of experience replay. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th International Conference on Machine Learning* (Vol. 119, pp. 3061–3071). PMLR. Retrieved from `https://proceedings.mlr.press/v119/fedus20a.html`

Gragera, A., Baffier, J., & Suppakitpaisarn, V. (2013). A bounds-driven analysis of "Skull and Roses" cards game. *2013*, 102–105.

Gragera Aguaza, A., Baffier, J.-F., & Suppakitpaisarn, V. (2015). Skull and roses card game. In N. Lee (Ed.), *Encyclopedia of Computer Graphics and Games* (pp. 1–4). Cham: Springer International Publishing. doi: doi:10.1007/978-3-319-08234-9_19-1

Gronauer, S., & Diepold, K. (2022). Multi-agent deep reinforcement learning: A survey. *Artificial Intelligence Review*, *55*(2), 895–943.

Hansen, E. A., Bernstein, D. S., & Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *AAAI* (Vol. 4, pp. 709–715).

Heinrich, J., & Silver, D. (2016). *Deep reinforcement learning from self-play in imperfect-information games.* arXiv. Retrieved from `https://arxiv.org/abs/1603.01121` doi: doi:10.48550/ARXIV.1603.01121

Koller, D., & Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *Artificial Intelligence*, *94*(1), 167-215. (Economic Principles of Multi-Agent Systems) doi: doi:https://doi.org/10.1016/S0004-3702(97)00023-4

Lux, T., & Marchesi, M. (1999). Scaling and criticality in a stochastic multi-agent model of a financial market. *Nature*, *397*(6719), 498–500.

Ma, J., & Wu, F. (2020). Feudal multi-agent deep reinforcement learning for traffic signal control. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (pp. 816–824).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning.* arXiv. Retrieved from `https://arxiv.org/abs/1312.5602` doi: doi:10.48550/ARXIV.1312.5602

Nan, A., Perumal, A., & Zaiane, O. R. (2022). Sentiment and knowledge based algorithmic trading with deep reinforcement learning. In *International Conference on Database and Expert Systems Applications* (pp. 167–180).

Perotto, F. S., Buisson, J.-C., & Alvares, L. O. C. (2007). Constructivist anticipatory learning mechanism (calm): Dealing with partially deterministic and partially observable environments. In *International Conference on Epigenetic Robotics (EpiRob), Piscataway, NJ, USA, 29/10/2007-02/11/2007* (pp. 117–127).

Shalev-Shwartz, S., Shammah, S., & Shashua, A. (2016). Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... others (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., ... others (2021). Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, *34*, 15032–15043.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... others (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, *575*(7782), 350–354.

Zydney, J. M. (2012). Scaffolding. *Encyclopedia of the Sciences of Learning*, 2913–2916.

	Experiment 1	**Experiment 2**
Learning rate α	0.03	0.03
Discount rate γ	0.99	0.99
Maximum epsilon ϵ	1	1
Minimum epsilon ϵ	0.01	0.01
ϵ**- decay rate**	5e-4(1st trial) 5e-5(2nd trial) 5e-6(3rd trial)	5e-5

A Appendix

Game		Agent 0	Agent 1	Final observation space	
Agent 0 Exp 1 Checkpoint 1 vs Agent 1 Exp 2 Checkpoint 1-5	1	Winner	Loser	[1, 1, 1, 0, 1, 1, 1, 1, 5, 0, 0, 2]	Agent 0 did not flip a Skull
	2	Winner	Loser	[1, 1, 1, 0, 1, 1, 1, 1, 5, 0, 0, 2]	Agent 0 did not flip a Skull
	3	Winner	Loser	[1, 1, 1, 0, 1, 1, 1, 1, 3, 0, 0, 2]	Agent 0 did not flip a Skull
	4	Winner	Loser	[1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	5		Illegal action		
Agent 0 Exp 1 Checkpoint 2 vs Agent 1 Exp 2 Checkpoint 1-5	6	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	7	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	8	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	9	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	10		Illegal action		
Agent 0 Exp 1 Checkpoint 3 vs Agent 1 Exp 2 Checkpoint 1-5	11	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	12	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	13	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	14	Winner	Loser	[0, 1, 0, 1, 0, 1, 1, 1, 3, 0, 1, 2]	Agent 0 did not flip a Skull
	15		Illegal action		
Agent 0 Exp 1 Checkpoint 4 vs Agent 1 Exp 2 Checkpoint 1-5	16	Loser	Winner	[1, 1, 1, 0, 1, 1, 1, 1, 2, 0, 0, 2]	Agent 1 did not flip a Skull
	17	Loser	Winner	[1, 0, 1, 1, 1, 1, 1, 1, 2, 0, 0, 2]	Agent 1 did not flip a Skull
	18	Loser	Winner	[1, 0, 1, 1, 1, 1, 1, 1, 2, 0, 0, 2]	Agent 1 did not flip a Skull
	19	Winner	Loser	[1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	20		Illegal action		
Agent 0 Exp 1 Checkpoint 5 vs Agent 1 Exp 2 Checkpoint 1-5	21	Loser	Winner	[0, 1, 1, 1, 0, 1, 0, 1, 2, 0, 1, 2]	Agent 1 did not flip a Skull
	22	Loser	Winner	[0, 1, 1, 0, 0, 1, 0, 1, 2, 0, 1, 2]	Agent 1 did not flip a Skull
	23	Loser	Winner	[0, 1, 1, 0, 0, 1, 0, 1, 2, 0, 1, 2]	Agent 1 did not flip a Skull
	24	Loser	Winner	[0, 1, 1, 1, 0, 1, 0, 1, 2, 0, 1, 2]	Agent 1 did not flip a Skull
	25		Illegal action		
Agent 0 Exp 2 Checkpoint 1 vs Agent 1 Exp 1 Checkpoint 1-5	26	Winner	Loser	[1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	27		Illegal action		
	28	Winner	Loser	[1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	29		Illegal action		
	30		Illegal action		
Agent 0 Exp 2 Checkpoint 2 vs Agent 1 Exp 1 Checkpoint 1-5	31	Loser	Winner	[1, 0, 1, 1, 0, 1, 0, 0, 2, 0, 0, 2]	Agent 1 did not flip a Skull
	32		Illegal action		
	33	Winner	Loser	[0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	34	Winner	Loser	[0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	35		Illegal action		
Agent 0 Exp 2 Checkpoint 3 vs Agent 1 Exp 1 Checkpoint 1-5	36	Winner	Loser	[1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	37		Illegal action		
	38	Winner	Loser	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	39		Illegal action		
	40		Illegal action		
Agent 0 Exp 2 Checkpoint 4 vs Agent 1 Exp 1 Checkpoint 1-5	41	Winner	Loser	[1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	42		Illegal action		
	43	Winner	Loser	[1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	44		Illegal action		
	45		Illegal action		
Agent 0 Exp 2 Checkpoint 5 vs Agent 1 Exp 1 Checkpoint 1-5	46	Winner	Loser	[1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	47		Illegal action		
	48	Winner	Loser	[1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2]	Agent 0 did not flip a Skull
	49		Illegal action		
	50		Illegal action		