* Head-to-head
* Describe model:
  + Network structure
  + Input tensor
  + Exploited Symmetries
  + Observed gameplay (chains)
* Discuss learning statistics

PROBLEM STATEMENT

DESGIN CHOICES AS RESEARCH QUESTIONS

CONCLUSIONS FROM EXPERIMENTS TOGETHER WITH SCIENTIFIC SUPPORTED MOTIVATION

CONCRETE + PRECISE methods, formulas

For this part, the challenge is to create an agent that can play on a bigger board and also on any size. To tackle the challenge of playing on a bigger board size, we are using a deep Q-network (DQN). The agent is implemented using the PyTorch version of OpenSpiel’s DQN. Being able to play on multiple board sizes is considered a challenge. Therefore there are two viable options presented: training a model for a 7x7 board or finding a way for the agent to tackle the problem of different board sizes.

A working solution to create an agent that can play on any size is to train a model for the maximum sized board, which is 15x15, using OpenSpiel’s DQN-agent. The idea is that the DQN-agent plays on 15x15-board but is only allowed to take actions that are legal on the board where the actual game is played. For example, if a game is played on a 7x7-board, the DQN-agent that has been trained for a 15x15-board, will return the best action that is legal for the 7x7-board.

When a game starts, the 15x15-model is loaded into the DQN-agent. Then, a separate environment is created for the specific board size of the game. All legal moves for the smaller board are converted to the corresponding actions of the bigger board. When *step* is called on the agent, the legal actions of the small board are given to the step function of the big 15x15-board agent. The DQN-agent’s step function returns the best epsilon-greedy action for the given legal moves. The action is converted back to the action of the smaller board. Finally, the action is applied to both boards. The final state of the two boards and how the smaller board fits into the bigger board can be seen in figure 1. The smaller board starts in the most upper left corner of the 15x15-board as it can then be extended to any other board size. However, he success of the agent very much depends on how well the main 15x15-model can generalize so it can easily adept to various board sizes.

Creating a DQN-agent requires some design choices. All of which have an impact on performance and correctness. Part of choosing these design chooses is balancing effectiveness of the agent and the training time, so we can make regular adjustment and experiment with hyperparameters.

The network of the DQN-agent consist of 2 hidden layer of a 100 nodes per layer. The number of input nodes is equal to the size of the observation tensor. The number of output nodes are the possible actions that can be taken. The learning rate is 0.001 and the discount factor is 0.99. A range of batch sizes and replay buffer capacities are used to acquire a good result. For the final model, a replay buffer capacity of 150 was used and a batch size of 50. While training the model a mean episode reward function is used to evaluate the trained agent every 250 episodes. INPUTTENSOR? Furthermore, the agent do not make any use of any symmetries of the board. Every 1000 episodes, the models for the agent that goes first and for the agent that goes second are saved.

**Results training**

In figure 2 we can see the evaluation of the agents every 250 episodes against random bots. The rewards of every state for the agents is summed and a mean is taken of a 1000 played games. Figure 2 shows that the mean reward slowly increases until 4000 episodes where the reward rapidly increases to 0.92. After 10000 episodes, the mean reward stabilizes but start to slowly decrease. This could be due to overfitting.

**Head-to-head**

As already mentioned, during training, the agent is evaluated against random bots. With the full 15x15 model trained, we can now show how the agent would perform against random bots on different board sizes. To get a broad idea of the performance of the agent, the agent is played against random bots on different board sizes. In figure 3, we can see the number of wins the agents is able to achieve on square boards when a 100 games are played. It is clear that the agent is able to win the most on a 15x15-board. This makes sense, as the main model was trained for this board size. We can also conclude a lower board size, such as 5x5, does not bode very well for the agent. The agent barely manages to do slightly better than a random bot. Sizes 7x7 and 10x10 have a higher win rate than 5x5 but not as much as for the 15x15-board size.

The agent is also tested on less conventional non-square boards. We can see in figure 4 that the agent more or less has the same win rate for all the non-square board. From figure 3 and figure 4 ,we can see that the agent handles the square boards a bit better than the non-square boards.

**Results gameplay**

As far as gameplay tactics, the against does not seem to have follow a specific tactical plan such as chaining. This can be expected as the evaluation mean is 0.75 for a 15x15 board and even less for smaller boards. If we look at the win rates in figure 3 and 4, then it is not unusual that we can assume, the model did not develop a tactic. If the agent struggles against random bots, the chance is small it developed a tactical plan.

**Conclusion**

**Maarten**

3

A picture containing screenshot, square, rectangle, design

Description automatically generated