A Neural Network Playing Stereotypically Human?

Abstract

In everyday life, whether conscious or not, humans encounter several situations where decision-making and risk assessment are necessary. For these processes, many regions of the human brain work simultaneously. The prefrontal cortex, amygdala, hippocampus, basal ganglia, and insular cortex are the five parts significantly affected by and are crucial for decision-making and risk assessment. Recently, artificial intelligence has been developing to make more effective risk assessments, such as bots which can make the seemingly most beneficial moves in card/board games. Schnapsen, an Austrian version of the card game 66, is an example of a card game that requires each player to conduct a strict risk assessment before making a move. By creating layers in a neural network that consist of multiple Convolutional, Dense, and Long Short Term Memory layers that represent a simplified version of the five regions of the human brain involved in risk assessment. Through these layers, the points and types of moves between AI and human approaches to risk assessment during Schnapsen gameplay can be observed. These are that indeed, humans and AI are similar in their results but, AI appears to be more stereotypically human, than humans.

Keywords: Artificial Intelligence, Neural Network, Schnapsen, Risk Assessment, Human Brain.

1. Introduction

Artificial intelligence, a term rather difficult to define, is often referred to as the theory and development of computer systems able to perform tasks normally requiring human intelligence. In the 21st century, artificially intelligent systems have been developing rapidly in sectors, such as health, business, communication, and many more. One application of AI can be seen in games, where bots are being developed to play against - or to outplay - humans. However, a topic of debate relating to AI bots arises when intuition in gameplay is considered. Because AI lacks consciousness, creating bots to play certain games becomes difficult. For instance, Schnapsen is a card game that has immense possibilities of outcomes; therefore, human intuition is a factor that impacts gameplay significantly.

When it comes to training artificially intelligent systems, Machine Learning and Neural Networks come into play. Neural Networks consist of many layers, modeled on the human brain, and have the intent of teaching AI to make choices that lead to a desired outcome through the use of weighted nodes. This paper aims to discern the parallels between AI decision-making in Schnapsen and human approaches to risk assessment during gameplay. Our approach involves the creation of several Neural Network layers which represent five regions of the brain associated with risk assessment, as an attempt to train a bot to play Schnapsen with a degree of intuition. It should be noted that intuition is defined as the ability to do/act instinctively with reasoning not being a necessity (Cambridge Dictionary, 2019), and therefore, brings up challenges in defining intuition when talking about an AI system. Nevertheless, by striving to implement the impact of stress, anxiety, and competitiveness on human brain regions as Neural Network layers, this paper attempts to train a bot to behave similarly to a human playing Schnapsen. We are focussing our attention on humans and AI playing against two bots namely, Randbot and Rdeep. From these matches key variables are logged to give us insights into comparing their behavior. These variables are the win and loss ratio, the average number of points per round, the average number of move types per round, and the average number of times the player was

in the lead or played as a leader. These variables will give us insights into the parallels we can draw between the two parties since they are commonly assessed factors during (intense) moments of gameplay.

2. Background information

2.1 The game Schnapsen

Schnapsen, an Austrian version of the game 66, is a two-player card game consisting of twenty cards. King, Queen, Jack, Ace, and Ten of every suit are the cards required. Each suit—Ace, Ten, King, Queen, and Jack—comprises cards worth different point values: eleven points for the Ace, ten points for the Ten, four points for the King, three points for the Queen, and two points for the Jack. Every player must have five cards and their hands should be unknown to the opposing player. The remainder of the cards are placed facing down in the talon. One card from the talon is placed facing up underneath all the face-down cards, which becomes the trump card. What makes this card special throughout the game is that it becomes the suit that is ranked higher than the rest. A trump exchange can be made before the start of the game, where the trump card is swapped with the lowest-ranking trump in a player's hand. The end goal is to be the first to reach 66 points.

By taking turns, players can win points by making tricks. A player places a card facing up, and their opponent has to beat the leader's (first person to place a card in the round) card with a higher suit or rank. It is important to note that, after a trick, the leader draws a card from the talon first, while the loser takes the next. This continues until either the deck is exhausted or one player wins. Special moves are available during the game. The leader of a trick can execute a trump exchange if the player is holding the lowest-ranking trump card, they can then replace it with the trump card visible under the talon. Additionally, a player can declare a marriage by revealing and playing both the King and Queen of a single suit. A marriage from the trump suit earns 40 points, while a non-trump marriage earns 20 points, collected upon winning a subsequent trick.

Schnapsen consists of two phases. For the second phase to begin, the talon has to be empty. In the second phase, the cards of the players often become known to each other.

The phase after the talon closes in Schnapsen is pivotal due to the heightened tension and strategic complexity involved.

This phase amplifies the sense of urgency and stress levels, demanding acute focus and decisive decision-making to navigate the intense competition effectively.

Strategic maneuvering becomes more necessary as players must rely on their existing hand to outmaneuver opponents and secure victory.

The winner earns points based on the margin of victory: three points if the loser won no tricks, two points if the loser's points are less than 33, and one point in any other case. In the event of a draw, neither player earns points.

All phases of the game are relevant to our research, since some people may find either or none of the phases to be risky. Because of this, we decided to capture all aspects of the games being played and average the results. This allows us to not necessarily focus on a single area of the game but, rather, as a whole.

2.2 The parts of the brain involved in risk assessment

To begin with the experiment, understanding the parts of the brain that are involved in risk assessment is crucial. The Prefrontal Cortex, Amygdala, Hippocampus, Basal Ganglia, and Insular Cortex are

deemed the most important. The Prefrontal Cortex was chosen as the starting region, as it has a great influence on reasoning, decision-making, and planning (Arnsten, 2009).
The insular cortex is a portion of the cerebral cortex that is involved in the processing of emotions, including stress. The insular cortex is also involved in the regulation of the autonomic nervous system, which controls the body's response to stress.
The Amygdala is responsible for processing emotions, and reactions, triggering fight-or-flight responses, and linking emotions to senses (Cleveland Clinic, 2023). The Hippocampus' relevance to risk assessment is related to its tasks in memory, learning, and emotional processes like anxiety. Similarly, the Basal Ganglia is a part of emotional processing and decision-making, but it also controls reward and addiction (Alexandru Andrușca, 2015). These parts of the human brain are essential to consider to create Neural Networks that can somewhat replicate the process going on in the human brain when making risky, yet rewarding decisions in card games like Schnapsen.

2.3 Preselected bots

Since the opponent bots incorporate some form of randomness in their strategy, we opted to use a constant seed across our multiple rounds of testing. We incremented the seed every few games because it could only vary so much after a while.
We tested our bot and the human players against both Rdeep and Randbot.

2.4 Rdeep

The Rdeep bot employs an algorithm that combines strategies akin to hill climbing and MCTS to ascertain the optimal move. Rdeep, designed for Schnapsen, formulates a strategy by hypothesizing about the unknown card combinations, executing this assumption to reach an endpoint, and subsequently, assessing its success level. In appraising a presumed state, Rdeep navigates the game to completion under the assumption that each player makes random choices for every move.
Rdeep was given a sample and depth of 6 for all games. We played 150 round-robin [1] matches for both the human and our AI. The following match statistics were gathered in a log file: points per trick, move type per round, leader rate, and lead rate. Finally, we plotted these results for both parties and compared them.

2.5 RandBot

The RandBot is a bot designed to play random moves during a game. It operates in a deterministic manner using the provided random number generator. It selects and plays cards randomly based on the generator's output.

3. The research question

The goal of this paper is to investigate *"What are the parallels between AI decision-making in Schnapsen and human approaches to risk assessment during gameplay?"*

[1] Competitors play in turns against each other. (Merriam-Webster, 2024)

Based on our layout of the network (i.e., the layers), we expect that the AI will perform very similarly to *human* players. We will do this by comparing variables such as the number of points per game, playing a certain number of move types, and win rate.

4. Experimental setup

4.1 Randbot and Rdeep

Both Randbot and Rdeep use some form of randomness in their decision-making process. They differ, however, in their complexity. Randbot simply chooses a valid move from a given move set at random. Rdeep incorporates randomness as a means of predicting future matches. It does this by utilizing the Schnapsen API provided by the university. The API allows bots to interact with the game fairly and it also enables them to gather all the information required to play. It has a method that allows bots to play games behind the scenes and it appropriately outputs the results of those games to the bot, this enables Rdeep to randomly sample matches up to a certain point. Both of those parameters i.e., samples and depth can be adjusted. You may do so to increase performance or perhaps tailor decisions to a certain outcome. However, due to the lack of information throughout most of the first phase of the game, it makes sense to use randomness in some way to enable predictability within the decision-making process.

4.3 Brainbot

Brainbot was inspired by the human brain and was intended to exhibit similar gameplay behaviors to humans during risk assessment. It works by combining multiple layers through the TensorFlow Keras' Sequential model. Each layer corresponds to a specific region in the brain. Firstly, there is the prefrontal cortex, which consists of a SimpleRNN[2] and LSTM[3] layer. The SimpleRNN layer handles what we believe to be the initial impressions one gets when receiving input, tanh[4] was used as its activation function because it gives a range of both positive and negative numbers, which can be interpreted as good or bad i.e., a mix of emotions. The number of units was kept relatively small because not much thought goes into this stage in the decision-making process. This layer is then followed by an LSTM, which although very similar to the SimpleRNN, handles input more intricately and can therefore be thought of as a stage of deeper analysis. Decisions are being tailored during this stage and emotions can still be of influence so, tanh is also used as its activation function.
Secondly, there is the amygdala. This region is meant to capture elements of fear and reward and for this purpose, a mixture of Dense, Conv2D, and LSTM layers was chosen. The Dense layer functions as a means of capturing high-level features, the big picture so-to-speak. It uses a sigmoid[5] as its activation function, since everything comes down to a range between 0 and 1 it is considered to be well-rounded and more rational compared to the alternative options. The Conv2D layer serves to capture non-linearity within the data and capture relationships within the sequences of the input. Relu[6] was used as its activation function due to it giving a larger range of positive values, this matters at this stage because although we feel as though certain things make us feel negative or positive, the amount of that emotion we feel is irrespective of that. We finish it off with an LSTM, which serves to capture temporal behaviors and mixed emotions, hence, a tanh function was used here as well.

---

[2] Recurrent Neural Network. A deep learning algorithm.
[3] Long short-term memory. An RNN architecture that is beneficial for sequence task predictions (Saxena, 2021).
[4] Hyperbolic tangent function.
[5] An activation function that adds non-linearity to a machine learning model.
[6] Rectified linear unit.

Thirdly, the AI has a region that mimics the hippocampus, a region focused on memory and spatial navigation. It is very similar to the preceding region, where it uses a combination of Conv2D, LSTM, and Dense layers. The order, however, has changed. The Conv2D and LSTM layers serve similar purposes as in the amygdala, but the Dense layer now utilizes Relu as its activation function. Because we want a measure of that response the input had, which in turn impacts memory, not necessarily the relationship between the individual and that memory in this instance.
Fourth, there is the basal ganglia. This region is responsible for handling procedural learning, habit formation, and motor functions. So, it uses an LSTM, Dropout, and once again, an LSTM as its layers. The LSTM at this stage serves to capture even deeper relationships between the sequential nature of the data. And, more importantly, the Dropout layer serves to reinforce learning, by introducing what we consider to be errors. It is the only quantity of response we are interested in here, hence the use of Relu functions.
Lastly, it uses the insular cortex. Which for the bot's purposes can be thought of as the final step in the decision-making process. It utilises a Conv1D, LSTM and Dense layer. Everything at this stage is meant to come to a single decision, so a Conv1D serves as a means of reducing the spatial features within the data and capturing any final relationships within those features. Followed by an LSTM, which serves a similar function to the Conv1D. Finally, it ends on a Dense layer, which gives us its ultimate decision.

5. Results

Both the human players and Brainbot played 150 matches against Randbot and Rdeep and their results were captured in log files and formatted for display. The following are statistics throughout the games played.

Human

Bot	Win Rate	Loss Rate	Regular Moves	Trump Exchange Moves	Marriage Moves
Randbot	0.78	0.22	3.805	0.0	0.447
Rdeep	0.399	0.601	3.872	0.0	0.095

Brainbot

Rdeep training set

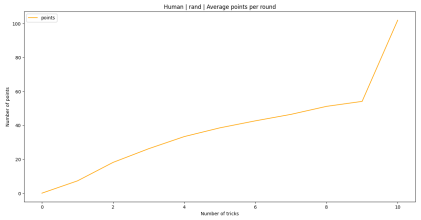
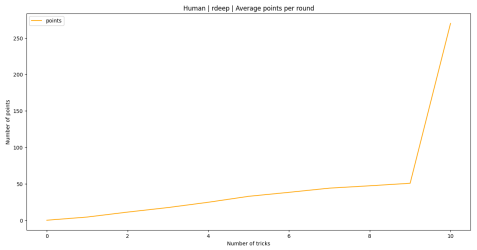
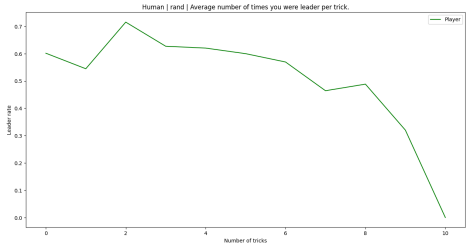
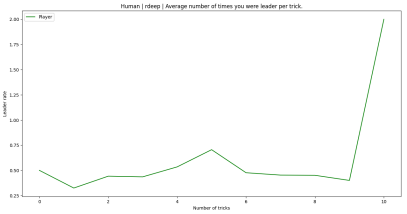
Bot	Win Rate	Loss Rate	Regular Moves	Trump Exchange Moves	Marriage Moves
Randbot	0.46	0.54	2.06	0.0	0.913

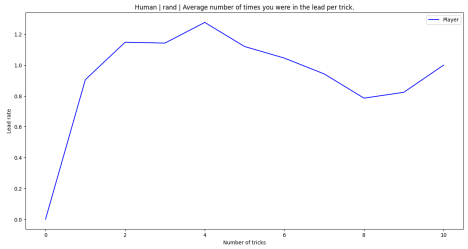
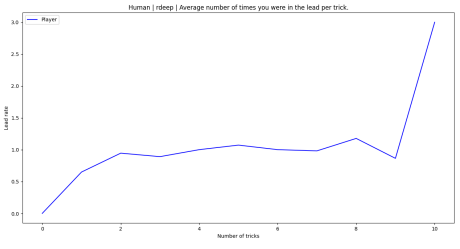
Rdeep	0.087	0.913	2.28	0.0	0.36

Randbot training set

Bot	Win Rate	Loss Rate	Regular Moves	Trump Exchange Moves	Marriage Moves
Randbot	0.513	0.487	2.36	0.0	0.847
Rdeep	0.173	0.827	2.667	0.0	0.247

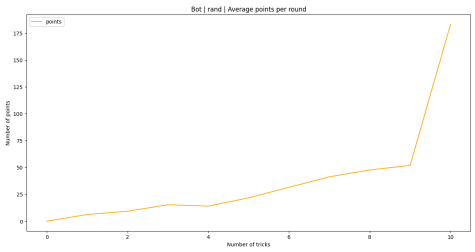
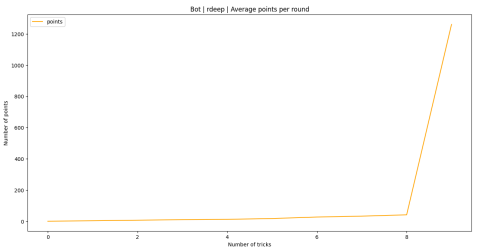
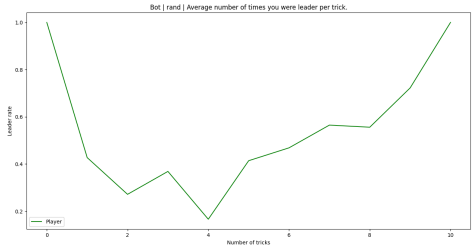
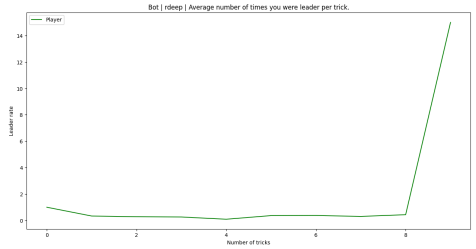
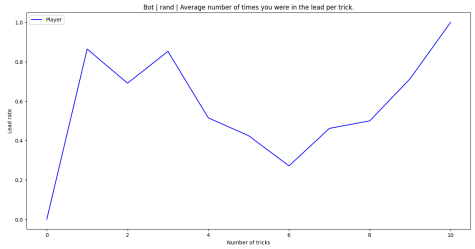
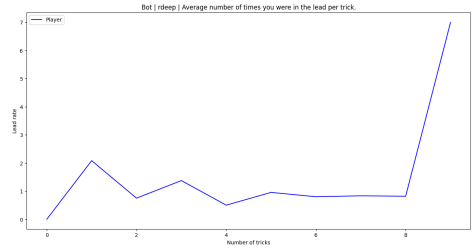
Human

Randbot	Rdeep
 *Points per round*	 *Points per round*
 *Leader per round*	 *Leader per round*

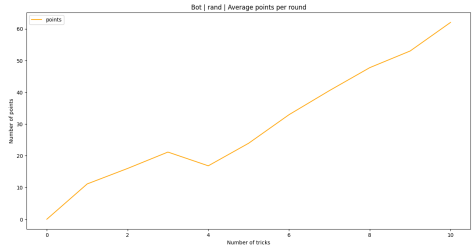
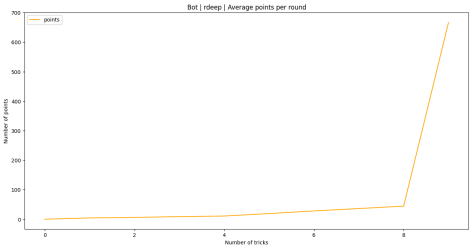
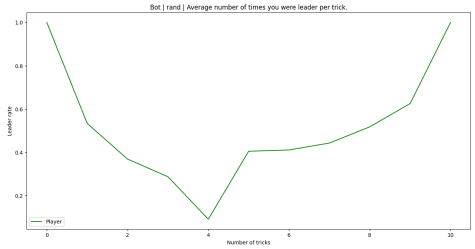
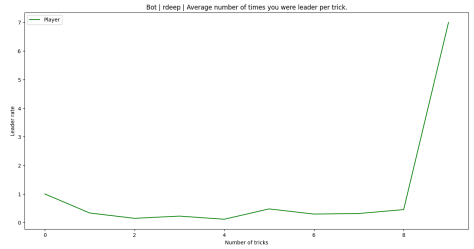
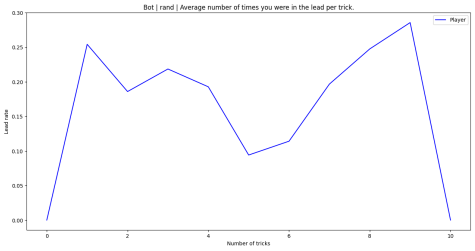
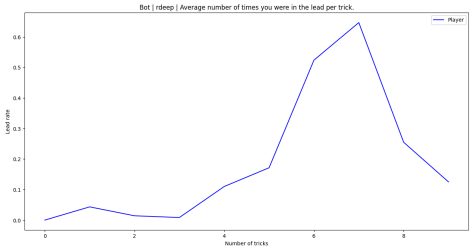
 *In the lead per round*	 *In the lead per round*

Brainbot

Randbot training set

Randbot	Rdeep
 *Points per round*	 *Points per round*
 *Leader per round*	 *Leader per round*
 *In the lead per round*	 *In the lead per round*

Rdeep training set

Randbot	Rdeep
	
Points per round	*Points per round*
	
Leader per round	*Leader per round*
	
In the lead per round	*In the lead per round*

6. Conclusions

From the results in section 5, we can conclude that Brainbot depended on the bots it was trained on and the parameters within the layers. This certainly reflects the behaviors found in humans as it was a product of their environment and settings (nature and nurture). If humans played randomly against each other, we would see about a 50/50 match in results. Similarly, if a strategy is introduced, such as being trained on Rdeep and playing against it, we find that the tougher opponent still introduces a challenge, but promotes better performance.
AI systems are inherently similar to humans due to them being fundamentally inspired by the neurons in organic brains.

7. Future work

We wanted to use Reservoir networks, which perfectly fit into the nature of our question. Due to the parallel nature of the brain's inner workings, it would allow us to more realistically capture the relationships with each layer and their outcomes. A deeper analysis of each layer would have enabled us to draw more (dis)similarities between the human brain and AI, which could have deepened our understanding of the relationship between each region of the brain. Secondly, we would have included Attention for our RNN but, we could not get them to work due to a lack of understanding of the TensorFlow API. One area of concern is that of the "leader per round" statistic, we think they differ because the humans we tested were more inclined to think in the short term, whereas the AI tended to focus on key stages in the match. This further supports our hypothesis because humans focus on many things at a time, we believe that if we introduced some mechanism for multi-tasking that statistic would look similar for both parties.
For Rdeep, we could have changed its sample and depth size to increase variety in our statistics; however, we found that due to the large amount of time and number of games, it would have taken for the human players to go through, it was not feasible within the given time frame. We would like to experiment with these numbers in the future.
We would also like to include other training sets to give us varying degrees in our reports since having Randbot and Rdeep play against themselves resulted in some linearity between the games being played.

References

1.Alireza Falahiazar, Saeed Setayeshi, Yousef Sharafi,J.: Computational model of social intelligence based on emotional learning in the Amygdala. Math. Computer Sci. 14(2015)77-86

2.Chen, X., Yang, T. A neural network model of basal ganglia's decision-making circuitry. Cogn Neurodyn 15, 17–26 (2021)

3. Arnsten, Amy F. T. "Stress Signaling Pathways That Impair Prefrontal Cortex Structure and Function." Nature Reviews Neuroscience, vol. 10, no. 6, June 2009, pp. 410–422

4. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain''. Psychological Review 65 (6): 386–408

5.Alexandru Andrușca. "Basal Ganglia." ***Kenhub*****, Kenhub, 26 May 2015,**

www.kenhub.com/en/library/anatomy/basal-ganglia.

6.Arnsten, Amy F. T. "Stress Signalling Pathways That Impair Prefrontal Cortex Structure and Function." ***Nature Reviews Neuroscience*****, vol. 10, no. 6, June 2009, pp. 410–422,**

www.ncbi.nlm.nih.gov/pmc/articles/PMC2907136/#:~:text=The%20prefrontal%20cortex%20(PFC)%20intelligently,brain%20regions%20(BOX%201)., https://doi.org/10.1038/nrn2648.

7.Cambridge Dictionary. "INTUITION | Meaning in the Cambridge English Dictionary." *Cambridge.org*, 2019, dictionary.cambridge.org/dictionary/english/intuition.

8.Cleveland Clinic. "The Amygdala: A Small Part of Your Brain's Biggest Abilities." *Cleveland Clinic*, 11 Apr. 2023, my.clevelandclinic.org/health/body/24894-amygdala.

9.Saxena, Shipra. "What Is LSTM? Introduction to Long Short-Term Memory." Analytics Vidhya, 16 Mar. 2021, www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/#:~:text=LSTM%20(Long%20Short%2DTerm%20Memory.

Appendix

A.1 Brainbot code

Here is the code used for Brainbot, the entire repository can be found on our Git Hub, which can be found through the following link https://github.com/jalenna/project_is, since our codebase is quite large we thought it would be better for the paper to only display this section of the code to give a sense of how the bot operates.

```
from src.brain.regions.insular_cortex import InsularCortex
from src.brain.regions.basal_ganglia import BasalGanglia
from src.brain.regions.hippocampus import Hippocampus
from src.brain.regions.amygdala import Amygdala
from src.brain.regions.first_layer import FirstLayer
from src.brain.regions.prefrontal_cortex import PFC
from src.utils.CONSTANTS import GLOBAL_SETTINGS

import tensorflow as tf
```

```
SETTINGS: GLOBAL_SETTINGS = GLOBAL_SETTINGS()


class Brain:
    """
    The neural network comprised of "brain-regions."
    """

    def __init__(self) -> None:
        """Constructs the brain of this neural network."""

        input_layer: FirstLayer = FirstLayer()
        input_layer._input(SETTINGS.TRAINING_SET_FILE_PATH)

        self.X, self.y = input_layer.get_features_and_labels()

        # Initialize the model
        self.regions: tf.keras.Sequential = tf.keras.Sequential()

        self.regions.add(input_layer.get_layer())

        # Define the shape of our data
        shape = (len(self.X[0]), len(self.X[0][0]))

        # Add brain regions here
        # The order can be found in BRAIN_OVERVIEW.md

        pfc: PFC = PFC(shape)
        for NN in pfc.get_layer():
            self.regions.add(NN)

        amygdala: Amygdala = Amygdala(shape)
        for NN in amygdala.get_layers():
            self.regions.add(NN)

        hippocampus: Hippocampus = Hippocampus(shape)
        for NN in hippocampus.get_layers():
            self.regions.add(NN)

        basal_ganglia: BasalGanglia = BasalGanglia(shape)
        for NN in basal_ganglia.get_layers():
            self.regions.add(NN)

        insular_cortex: InsularCortex = InsularCortex(shape)
        for NN in insular_cortex.get_layers():
            self.regions.add(NN)

        return None
```

```
def train(self) -> None:
    """Trains the model."""

    self.regions.compile(
        optimizer="adam", loss="mean_squared_error", metrics=["accuracy"]
    )
    self.regions.fit(self.X, self.y, epochs=SETTINGS.TRAINING_ITERATIONS)

    # Optional, print the overview of the model
    self.regions.summary()

    return None

def save_model(self) -> None:
    """Saves the model.

    Saves the model to the specified path in src.utils.CONSTANTS.py
    """

    self.regions.save(SETTINGS.MODEL_SAVE_PATH)

    return None

def save_weights(self) -> None:
    """Saves the weights of the model.

    Useful for analyzing individual weights/connections.

    Saves the weights to the specified path in src.utils.CONSTANTS.py
    """

    self.regions.save_weights(SETTINGS.MODEL_WEIGHTS_PATH)

    return None

def load_model(self) -> None:
    """Loads and sets the model of this instance of the brain."""

    self.regions = tf.keras.models.load_model(SETTINGS.MODEL_SAVE_PATH)

    return None

def load_weights(self) -> None:
    """Loads the weights of the model in case the model is loaded into memory."""

    if self.regions is None:
        return None
```

```
        self.regions.load_weights(SETTINGS.MODEL_WEIGHTS_PATH)

        return None

    def run_all(self) -> int:
        """Runs all methods.

        Useful for debugging and experimenting.

        Arguments:
            None
        Returns:
            int: Exit code
        """

        print("Running all methods...")

        self.train()
        self.save_model()
        self.save_weights()
        self.load_model()
        self.load_weights()

        print("Ran all methods successfully.")

        return 0


# Code relevant to the bot

from schnapsen.game import Bot, PlayerPerspective, Move
import src.utils.VU.ml_utils as ml_utils
import numpy as np


class BrainPlayingBot(Bot, Brain):
    """
    This class loads a trained ML model and uses it to play.
    """

    def __init__(self, name: str = "BrainBot") -> None:
        """
        Create a new MLPlayingBot which uses the model stored in the model's location.

        Arguments:
            name optional(str): The name of the bot, default = "BrainBot"

        Returns:
            None
```

```
        """

        super().__init__(name)

        # load model
        self.load_model()

        # Or load the weights
        # self.load_weights()

    def get_move(
        self, perspective: PlayerPerspective, leader_move: Move | None
    ) -> Move:
        # get the sate feature representation
        state_representation = ml_utils.get_state_feature_vector(perspective)

        # get the leader's move representation, even if it is None
        leader_move_representation = ml_utils.get_move_feature_vector(leader_move)

        # get all my valid moves
        my_valid_moves = perspective.valid_moves()

        # get the feature representations for all my valid moves
        my_move_representations: list[list[int]] = []
        for my_move in my_valid_moves:
            my_move_representations.append(ml_utils.get_move_feature_vector(my_move))

        # create all model inputs, for all bot's valid moves
        action_state_representations: list[list[int]] = []

        if perspective.am_i_leader():
            follower_move_representation = ml_utils.get_move_feature_vector(None)
            for my_move_representation in my_move_representations:
                action_state_representations.append(
                    state_representation
                    + my_move_representation
                    + follower_move_representation
                )
        else:
            for my_move_representation in my_move_representations:
                action_state_representations.append(
                    state_representation
                    + leader_move_representation
                    + my_move_representation
                )

        # Convert actions to a numpy array
        action_state_representations_np = np.array(
            action_state_representations, dtype=np.float32
```

```
    )

    action_state_representations_np = np.expand_dims(
        np.array(action_state_representations_np), axis=1
    )

    model_output = self.regions.predict(action_state_representations_np, verbose=0)

    winning_probabilities_of_moves = [
        outcome_prob[1] for outcome_prob in model_output
    ]

    # Find the move with the highest probability of winning
    highest_value: float = -1
    best_move: Move = my_valid_moves[0]

    for index, value in enumerate(winning_probabilities_of_moves):
        if value > highest_value:
            highest_value = value
            best_move = my_valid_moves[index]

    return best_move
```

A.2 Training set

We are using two training sets of ten thousand games played each. One was Randbot against Randbot and the other was Rdeep against Rdeep. We initially chose these as a means of testing our bot. These matches were converted to feature vectors and then captured and stored in log files.