The Gin - Interactive AI Card Game

TensorFlow Environment-Agent Trained Card Game

Kanglin Xu
Dept. of Computer Science
Texas Tech University
Lubbock, United State
kanglin.xu@ttu.edu

Tomas Rohatynski
Dept. of Computer Science
Texas Tech University
Lubbock, United State
trohatyn@ttu.edu

***Abstract*—This report documents the introduction, description, motivation and more depth in development aspect of our project, the gin, an interactive AI card game. The project aims to test the limit and find the difference in decision-making between reinforcement learning trained AI and humans in a card game named 'straight gin.'**

***Index Terms*—deep learning, reinforcement learning, tensor flow**

I. Introduction

The goal of our project is to train a machine learning model (MLM) using TensorFlow to play the best game of gin physically possible, so that we can observe the limits of current open-source MLM libraries – which in our case is TensorFlow's Python library.

We are defining playing the best possible game of gin by balancing luck and skill, because in gin you need four out of seven cards to be exact in number to declare a legitimate win, but because of the seven cards in a hand being shuffled for randomness, it is possible for the MLM to draw a winning hand. In the event of such a case, there was no action that occurred prior to a win (reward), making that single iteration of the 10,000 completely useless for training purposes.

Essentially, the best possible game for the MLM to play is one where it does not at least win right off the bat, since it has to take – a soon to be defined – minimum number of actions for that iteration to yield learning progression in the epoch.

On the surface, the game has little room for strategy and more to do with the luck of the draw. Due to the nature of this project, we can solve this contingency while simultaneously running the risk of: Discovering unique case strategies to apply in a physical environment, defining a set of outlier environmental variables (starting hands/decks), while answering why and how they impacted the training process, demonstrating the ability of how simplicity can be used as the medium to support a complex machine learning network, allowing leeway for teaching/presentation purposes relating to machine learning, finding what limitations there were (if any) with TensorFlow during the training phase of our MLM, and answering if our methodology was truly the best (open-source) approach.

Our approach will be done by creating a set of parameters that our model will explore, with the hope that after 10,000 training iterations – it can maintain a superhuman win rate (which will be defined later on). The roadmap for this is simple: First, using Python we will define the environment for the machine learning model to train in. Next is designing a TF (TensorFlow) MLM to play gin against a copy of itself. Then through trial and error, remove/add/optimize the MLM learning parameters until they learn to play at our soon to be defined, "superhuman win rate". Finally, create support for a human to play one-on-one against our model to showcase its progress.

II. Description

The goal is to train a artificial neural network agent to be able to interactively play against a human. To achieve this we will leverage several interesting Machine Learning tools, utilities and concepts. Those include **Python** with object oriented programming, **Tensorflow** learning and inference, **Reinforcement Learning** with multiple **Agents**, and data visualization using matplotlib and custom graphics displays. We will also make use of numpy for speed optimization.

A. Tools & Approach Discussion

1) Python: Due to the nature of anything related to machine learning being very dynamic with usually many parameters that need to be tuned or the need to entirely change the approach to the problem, a scripting language like Python is ideal. This is because if a change is needed, there is no need to recompile the entire project, it is as simple as making the change and re-running the code. This allows us to test changes faster to arrive at a working solution with less time spent waiting for the code to compile and more time actually analyzing the code and results. Another benefit is that for projects that build upon very complex concepts like machine learning, but do not necessarily need that much code because it is covered by libraries like TensorFlow, a easy to read high-level scripting language like Python allows to focus on the main problem at hand instead of various complicated syntax and programming structures. Python also has tools like matplotlib that make data visualization easy to add, and another tool called NumPy that allow leveraging the speed of C for processing intensive tasks which will help us get more training done in less time.

2) Object Oriented Programming: In order to effectively train a machine learning network, it is crucial to run as much data in parallel as possible. OOP helps make this easier thanks to being able to create self-contained objects that keep track of the data of each game, thus greatly simplifying organizing the data and reducing the risk of intertwining data of different games. Since the object classes serve as templates, it is easy to quickly create many parallel tasks by simply instantiating the desired number of objects.

3) TensorFlow[1]: This is one of the most fully-fledged and flexible machine learning libraries for Python. It accommodates any type of learning paradigm, network nodes and layouts, and preprocessing and postprocessing operations one can think of. Thanks to its great range of functionality it makes it easy to evaluate and combine different methodologies while still working within that same library and not having to bridge multiple libraries together. Using TensorFlow will aid us both in the prototyping stage with its flexibility and in the final code with how mature and fast it is. Once we have trained the neural network to our satisfaction using the training harness that TensorFlow provides, we will the run the network in inference mode which will allow us to watch it play or play against a human.

4) Multi-agent Reinforcement Learning: When training an artificial neural network we want it to analyze as much data as possible so that it can train on a varied set of data and build enough experience off of it to deal with new cases that it has not seen before. When working with big data acquiring enough examples for the network to analyze is easy, but in the case of a card game there are only so many unique cards and that is usually nowhere near enough data. The alternative could be for the network to play against a human, and learn from the action taken by the human player, unfortunately this introduces human bias which is undesirable when training a neural network model. Besides that it would take many hours of human interaction to train it. An optimal approach is thus for the neural network to generate the data it trains on itself, in layman's terms the neural network can learn by playing against a copy of itself. This way the neural network can gather thousands of hours of playing experience in a manner of minutes comparing to playing at a human pace. The reinforcement part of this learning approach is that the agents receive a reward/penalty based on how well they did in the match, they then try to maximize this reward parameter. This allows them to develop strategies against each other on their own, which can potentially go beyond strategies that humans can develop. The reward is also the caveat of reinforcement learning, because an improperly tuned reward function might cause the agent to overshoot their learning and become to used to playing a copy of itself and not be adaptable enough to play against a human. Beyond the configuration of the neural network layers itself, the reward function will be one of the most crucial aspects to tune in order to reach desirable training outcomes[2, 3].

5) Testbed: Our testbed consists of a simple Terminal User Interface (TUI) that allows a human to play against another human or against the Agent. This way one can play against another human first to learn more about the game, and then play against the Agent for a challenge. The TUI displays the current player's hand, the last discarded card, and how many cards there are in the deck and discard pile respectively. Then the TUI prompts the current player to first choose to draw a card from the deck or discard pile. Then if the card the player drew forms a winning hand the game ends, otherwise the player is prompted to discard one of their cards and for the next player to continue the game.

B. Gin Card Game

We have decided to train the artificial neural network to play Gin (straight gin variant), this is because this card game has properties that play well with neural network models. Specifically, the number of observations and actions is always the same, which is a prerequisite for any artificial neural network. Besides this Gin also has a very clear win/loss condition, that does not leave any ambiguities as to how one got to a winning hand. A example of winning hand is shown in figure 1.

The rules of Gin are as follows[4, 5]:

1) Deck of cards with jokers removed is used (52 cards)
2) Each player is dealt 7 cards
3) One more card is drawn and placed face-up next to the deck
4) The player left of the dealer goes first
5) Player must pick either a face-down card from the deck, or a face-up card next to the deck
6) Player must discard 1 card to the face-up pile next to the deck
   - If a card was drawn from the face-up pile, then that same card can not immediately be discarded
   - If a card was drawn from the face-down deck, it may be immediately discarded
   - Otherwise, a card that is in the players hand at the start of their turn may be discarded
   - At the end of their turn, the player must always have exactly 7 cards in their hand
7) The first player to have one set or run of 3 cards and one set or run of 4 cards wins
   - Valid set is when the player has 3 or 4 cards of the rank (and different suits), e.g. two of clubs, hearts, and spades
   - Valid run is when the player has 3 or 4 cards in sequence of the same suit, e.g. two, three, and four of clubs
   - Ace cards can either be counted as before 'two' or after 'king', wrapping around (e.g. king, ace, two) is not allowed
   - Player indicates that they have won by discarding the card that is not part of their winning face-down on the discard pile

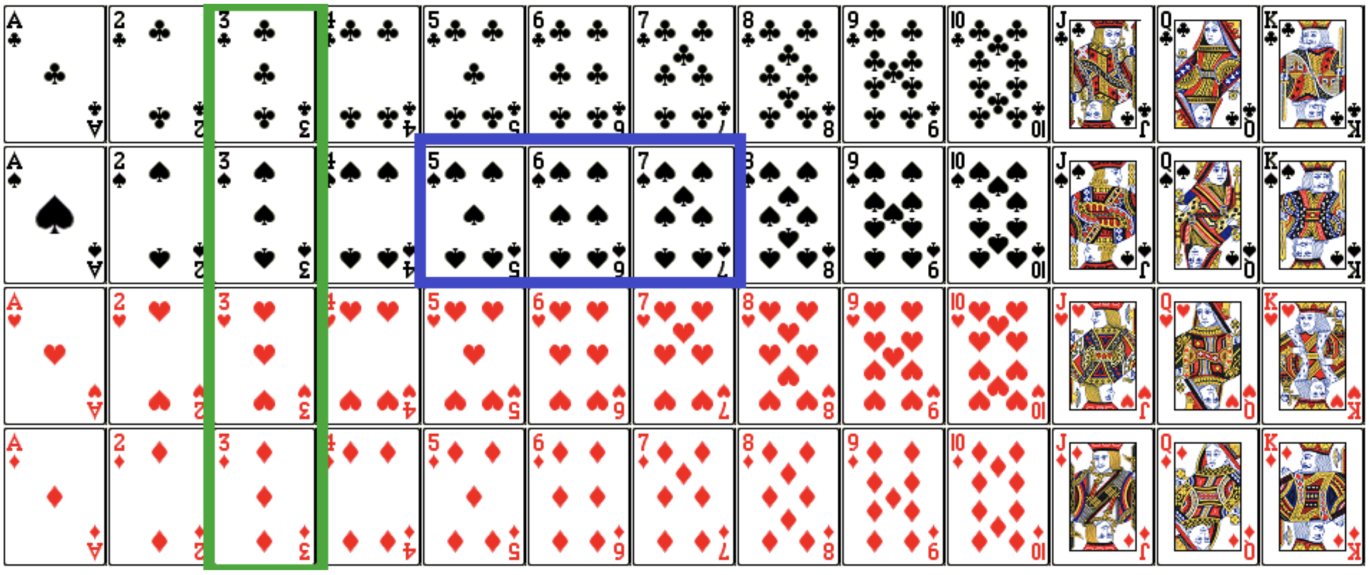

Fig. 1. Win Case Demonstration

III. Motivation

In today's world, efficiently utilizing limited resources to maximize outcomes in the shortest possible time is imperative. This principle aligns closely with the objectives of the card game 'Gin', where the goal is to form specific combinations of cards with minimal moves. Our project focuses on developing a reinforcement-learning algorithm-trained artificial neural network (ANN) for the card game 'Gin', enabling human players to compete against AI. The game involves strategic observation of opponents' discarded cards and careful management of one's deck to form sets and runs. The training involves two ANN agents aiming to complete sets and runs with seven cards in as few turns as possible, enhancing their decision-making by analyzing other players' actions. This project serves dual purposes: educational and entertainment.

A. Educational Objectives

From an educational standpoint, the project tests and contrasts the decision-making limits of ANN agents within a constrained resource environment against proficient human players. Developed within TensorFlow's versatile environment-agent framework, this project allows users to customize action environments, agent steps, and reward systems tailored to specific objectives. The training process employs unsupervised learning to explore the ANN agent's decision-making capabilities. Our goal is to observe the differences in strategy between ANN agents and human players, particularly in how ANN agents analyze opponents' actions to avoid duplicating their combinations. Moreover, we aim to investigate how ANN agents handle scenarios where they compete directly with humans for similar combinations, providing insights into strategic adaptability.

B. Entertainment Goals

For entertainment, our aim is to develop a more human-like ANN agent in the realm of card games by adjusting its memory capabilities. Typically, ANN agents pose a significant challenge due to their superior memory capacity, allowing them to recall all discarded cards and accurately predict remaining deck compositions to strategize their moves. However, we plan to limit the ANN agent's memory space to make their behavior more akin to human players. This limitation is intended to create scenarios where both ANN agents and human players may coincidentally aim for the same combinations, thereby enhancing the game's entertainment value and unpredictability.

IV. UML Diagrams

A. Class Diagram

- AutoTrainEnvironment
  - Represents an environment where a neural network is trained by playing against a copy of itself
  - deck: deck of cards in play
  - hands: hands of each agent
  - agents: agents playing against each other
  - train(): a single training step consisting of each agent taking one action with training enabled
- HumanPlayEnvironment
  - Represents an environment where a single human plays against a trained neural network
  - deck: deck of cards in play
  - hands: hands of each agent
  - agents: agents playing against each other
  - human_action(): the action of the human player which is called twice, first to draw a card and then to discard a card

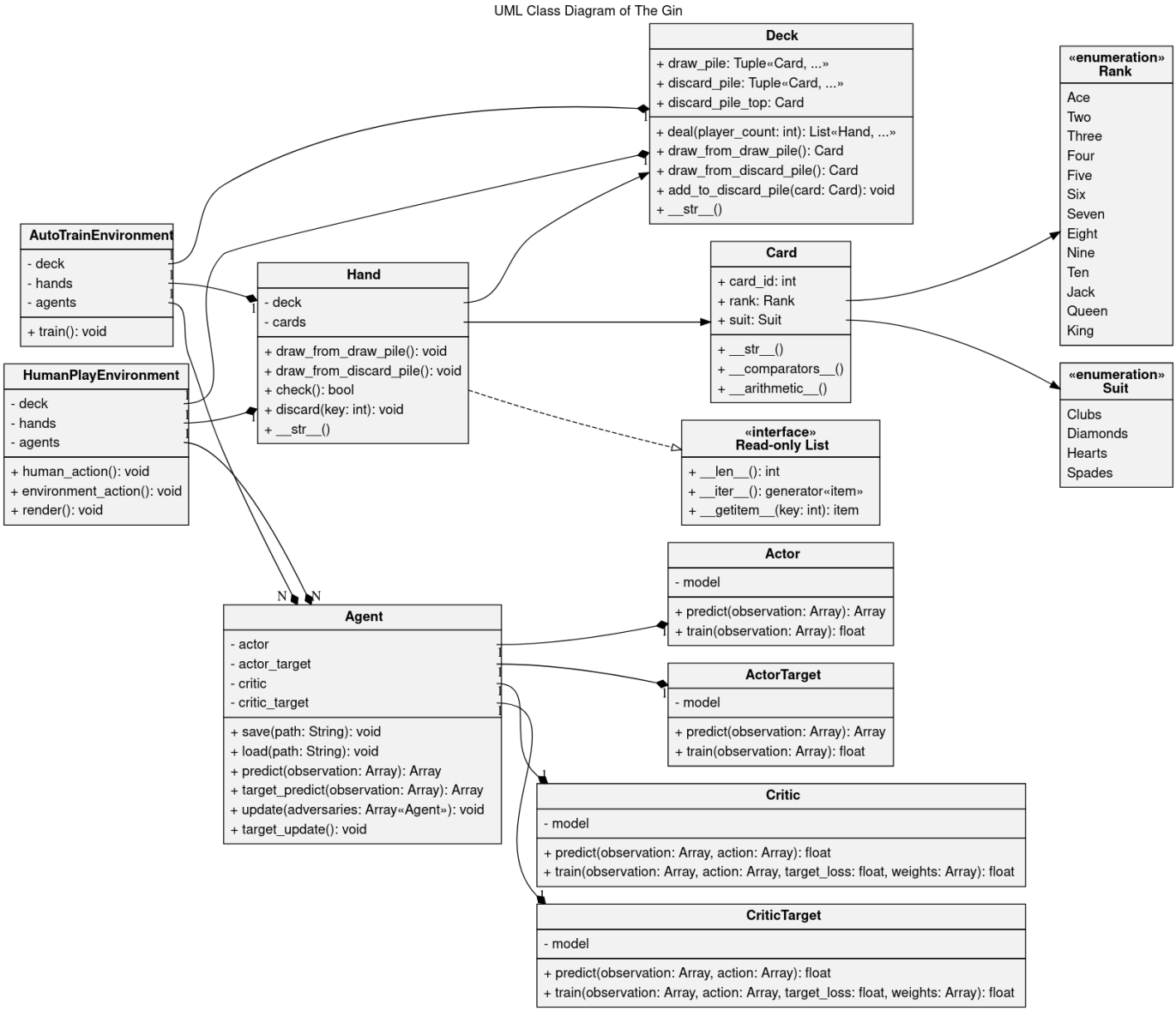

Fig. 2. Class Diagram

  - environment_action(): the two actions (draw then discard) of the trained agent are done automatically when this method is called
  - render(): displays a updated Terminal User Interface representing the the human player's hand, the last discarded card, and the number of cards in deck and discard pile respectively

- Agent
  - Represents a MADDPG agent that can act within one of the environments
  - actor / actor_target / critic / critic_target: the various neural networks that build up the MADDPG agent
  - save(path): save all the neural networks to a file
  - load(path): load all the neural networks from a file and replace the ones that are referenced by the object
  - predict(observation): make a prediction based on the observation, which can be used as an action in the environment
  - target_predict(observation): make a prediction based on the observation, which can be used as an action in the environment, but use the target network instead which has more immediate experience development
  - update(adversaries): train the actor and critic network one step by letting the critic network judge the actor network's action compared to the other agents and apply the gained experience to the target networks, and move some of the experience from the target to the main networks
  - target_update(): train the target networks using gained experience, and move some of that experience

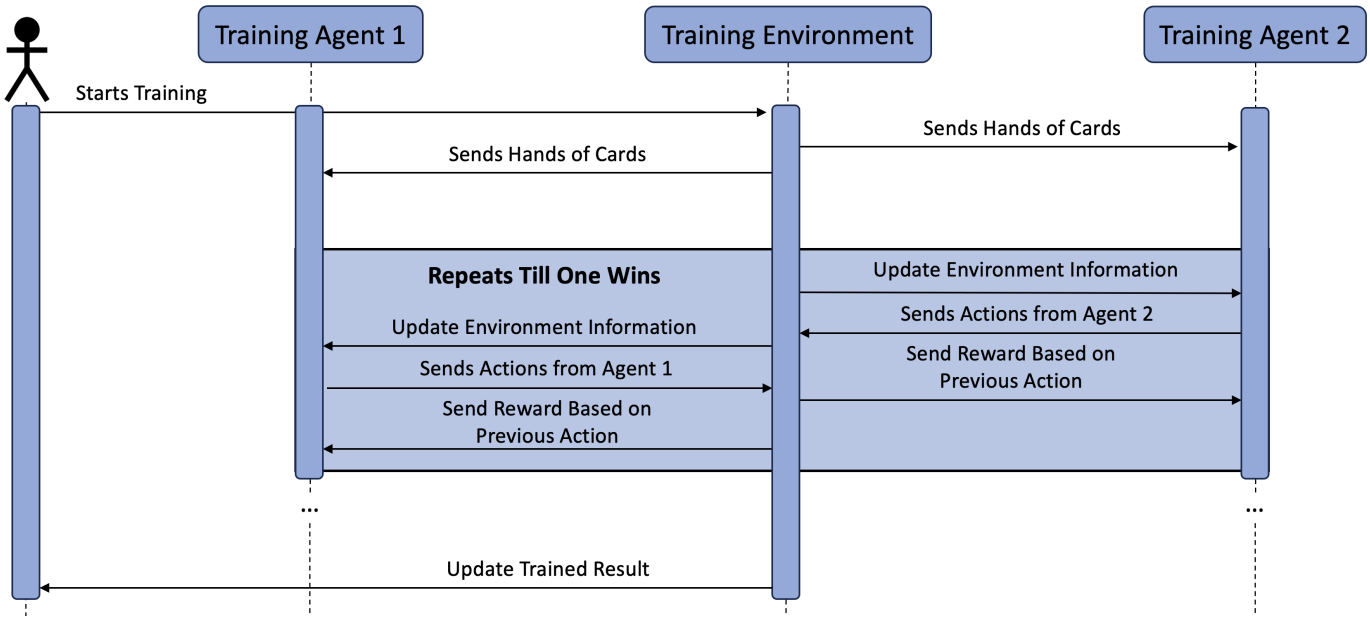

Fig. 3. Sequence Diagram - Training

to the main networks

- Actor/ActorTarget
  - Represents the actor neural network (Actor and ActorTarget are identical networks)
  - model: the TensorFlow model for the actor
  - predict(observation): make a single prediction based on an observation which can then be used as an action in one of the environments
  - train(observation): train the network by a single step by letting its corresponding critic network judge its action
- Critic/Critic Target
  - Represents the critic neural network (Critic and CriticTarget are identical networks)
  - model: the TensorFlow model for the critic
  - predict(observation, action): make a single prediction based on an observation and an action of an actor in order to judge the actor for training
  - train(observation, action, target_loss, weights): train the network by a single step by comparing the output of the critic, critic_target, and experience collected in the replay buffer
- Deck
  - Represents the deck in use for the current game
  - draw_pile: cards ready to draw that are face-down
  - discard_pile: discarded cards that are face-up
  - discard_pile_top: shorthand for the card at the top of the discard pile that can be drawn in place of the draw_pile
  - deal(player_count): instantiates player_count hands and gives each hand 7 cards
  - draw_from_draw_pile(): remove a card from the top of the draw_pile and return it
  - draw_from_discard_pile(): remove a card from the top of the discard_pile and return it
  - add_to_discard_pile(): put a card at the top of the discard pile
  - __str__(): print all the cards in the deck
- Hand
  - Represents a hand owned by a player in the game
  - deck: a reference to the deck in order to draw and discard cards from/to it
  - cards: the cards on the hand
  - draw_from_draw_pile(): remove a card from the decks draw_pile and add it to this hands cards
  - draw_from_discard_pile(): remove a card from the decks discard_pile and add it to this hands cards
  - check(): determine if this is a winning hand
  - discard(key): remove the specified card index from this hand and add it to the deck's discard_pile
  - __str__(): print all the cards on this hand
  - <<¡interface>> Readonly List: this class implements a readonly lists in order to get, iterate, and manipulate cards from this hand without worrying about accidentally mutating the hand itself, the Hand class can be directly interacted with just like any other python list with the exception of deleting or modifying values as they need to be copied first
- Card
  - Represents a card in the game
  - card_int: the id of the card which is calculated from the card's rank and suit when it is instantiated
  - rank: the Rank enum representing the rank of this card

  - suit: the Suit enum representing the suit of this card
  - __str__(): print the rank and suit of this card
  - __eq__(), __lt__(), etc. (__comparators__()): comparator operations that allow two cards to be compared directly using their card_id's without first having to get the card_id
  - __add__(), __sub__(), etc. (__arithmetic__()): arithmetic operators that instantiate a new Card with card_id resulting from the arithmetic operation between a card and an integer which allows directly comparing a card to e.g. an incremented card without first getting the card_id, then doing the math, and the instantiating a new Card separately
- <<enumeration>>Rank
  - Represents all possible ranks a card can have in the game
  - Ace = 0, Two = 1, . . . , King = 12
- <<enumeration>>Suit
  - Represents all possible suits a card can have in the game
  - Clubs = 0, Diamonds = 1, Hearts = 2, Spades = 3

B. Sequence Diagram

1) Training: In the Sequence Diagram of Training (Figure 2), the diagram consists of one user and three systems. The process starts with the user initializing the training request to the Training Environment, and the Training Environment sends a hand of cards to both Training Agent 1 and Training Agent 2. Continuously, it goes into the training loop and ends when one Training Agent satisfies the win condition. Within the training loop, one of the Training Agents will receive updated environment information from the Training Environment and reply with their action. Then, the Training Environment sends the updated environment information to the other agent and gets replies with its action. At the end of the mutual turn, the environment analyzes their previous actions and sends a reward based on it. When the training loop ends, the Training Environment returns the updated trained result to the user, and the process ends.

2) Playing: In the Sequence Diagram of Playing (Figure 3), the diagram consists of one user and two systems. The process starts with the user entering the game, and the environment will send a hand of cards to both the Trained Agent and the Player. Then, the player will send an action to the Environment, and the Environment will update the environment information to the trained agent and take an action from the agent. After the action from the agent, the Environment updates the environment information to the player and takes action from the player again. The process iterates until the Trained Agent or the Player satisfies the win condition, and the loop ends. At the end of the process, the Environment sends the result to the Trained Agent and the Player.

C. User Diagram

In the User Diagram (Figure 4), the Gin consists of three users in total: the player, the trained agent (or the second player), and lastly, the environment. The system itself also includes eight functions that will be involved during the gameplay. The player has access to the functions "Start Game" and "Game Mode" to specify the level of difficulties and number of players. The "Start Game" function will send information and trigger the function "Initialize Environment," which initializes and sends the game information to the Environment along with information from the function "Game Mode." Then, the Environment will have access to the "Deck and Hands Initialize" function to initialize the game. As the game starts, the Player and the Agent will have access to the functions "Draw Card" and "Discard Card" to draw a card from the face-down deck or discarded pile and discard a card from hand. After each turn, the Environment will have access to both the functions "End Turn" and "Win Check" to check the win condition of the current player and end their turn if the win condition is not satisfied. The functions "Draw Card," "Discard Card", "Win Check", and "End Turn" works in a flow.

V. Potential Solution

The solution that we have in mind is to pursue a Multi-Agent Reinforcement Learning environment using MADDPG (Multi-Agent Deep Deterministic Policy Gradients) as our Agent model layout. We are currently working on re-implementing MADDPG based on an example we found on GitHub [6] in a way that will better align with our environment, and this way we can also better understand how it works under the hood. The basis of MADDPG (see Fig. 5) is that each agent consists of 4 neural networks - Actor, ActorTarget, Critic, and CriticTarget - as well as a replay buffer. The Actor - Critic pair allows the Agent to judge its own action compared to its adversaries every time it takes an action. The Target networks are there to gradually move the experience from the immediate actions to the main network, thus improving training stability and helping prevent overfitting.

Our plan is to define a loop of two such MADDPG agents playing against each other, and learning from each other's actions. This way they can learn and build experience from hundreds of games without any human input. At certain intervals the first/primary Agents' networks will be saved to a file, which can later be recalled to either resume training from that point or to play against a human - which is our end goal.

VI. User Interface

A. Enter Page

Figure 6 shows the Enter Page user interface. 'The Gin' at the middle top is the title of our game/program. Then, it follows with a quick description of our card game, 'Interactive AI Card Game.' After that, it will show an option that asks the user to pick the game mode, where entering 1 will be for a single player and 2 will be for two players. Choosing the option below will lead the user into the game accordingly.

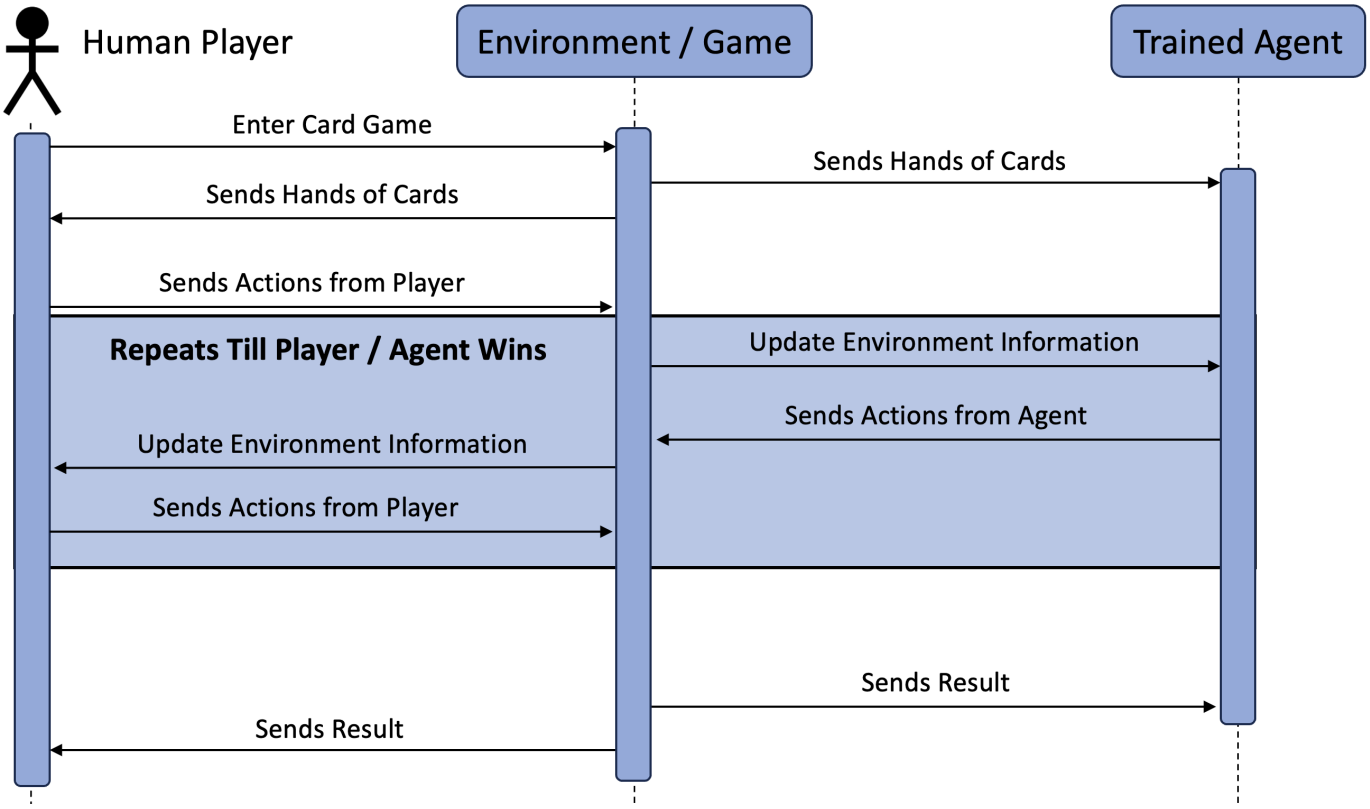

Fig. 4. Sequence Diagram - Playing

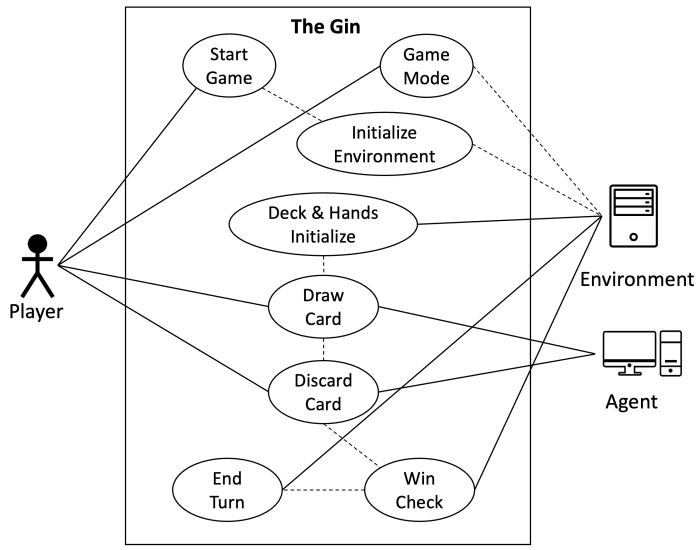

Fig. 5. User Diagram

B. Agent vs. Agent and Player vs. Agent

Figure 7 shows the game interface for agents playing against agents. 'The Gin - Interactive AI Card Game' is another quick description of the card game. In the agent vs. agent interface, the user will be given information on both the agent's hand of cards and the observation of the environment, including the number of remaining cards in the deck and the most recent discarded cards. Then, at the bottom, we will show the most recent action. Figure 8 shows the game interface for player vs. agent. The user interface for Player vs. Agents is similar to Agent vs. Agent, with the difference that the player only has access to their hand of cards.

VII. Implementation Overview

This project aims to design and train an artificial neural network to play the card game gin. Similarly to the Alpha Go Zero developed by Google, our project aims to train the agent by having it play against itself and not import any game data from other players. The implementation is first designed based on MADDPG (Multi-Agent Deep Deterministic Actor-Critic Policy Gradient), where the learning involves multiple agents and environments and acts based on the other agent's actions and environment policies. Later on, we modified the design to adapt DDPG for Multi-agent learning. To briefly summarize the essential implementation, we first define the rules of the card game gin in the environment, then design the agent that analyzes the environment and acts upon it. Furthermore, the agent inputs the action based on observing the environment. The other agents input the actions based on the previous actions, and the environment will reward them based on their actions. The goal of each agent is to finish the combination of cards before the other agent does so. Within

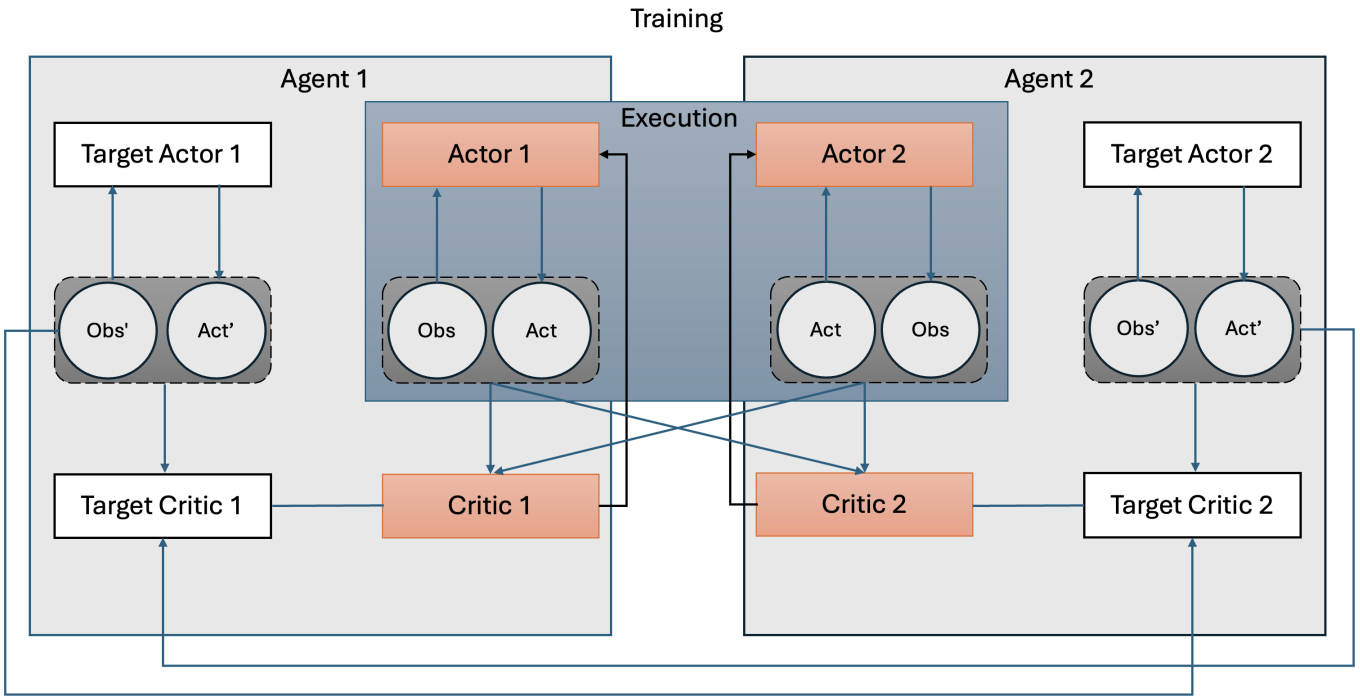

Fig. 6. Multi-Agent Reinforcement Learning Environment

Fig. 7. Enter Page

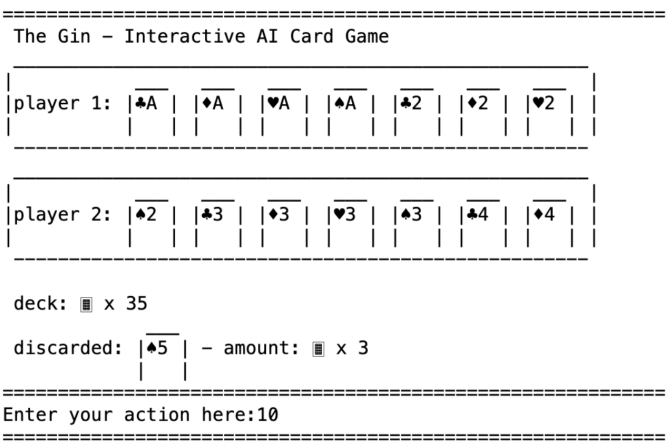

Fig. 8. Agent vs. Agent

the project, we also analyze each agent's actions and improve our reward definition to fine-tune the training.

VIII. Development Techniques

We developed the project using Python, and implemented all of the data structures using Object-Oriented Programming. Python allowed us to quickly iterate on changes and test them without having to recompile the project or ensure that all of the code was valid; this way, we could make progress quickly and easily discover when an approach was or was not working. Because our project involved several representative data structures like hands of cards, cards themselves, players, decks, etc., using OOP was the most effective way to implement everything in a well-organized, easy-to-understand, and modified manner. [1, 7]

We used Git through GitHub to store and synchronize our project. This allowed us to easily keep track of the project's latest version and record who changed what in the code. This way we could work on the project and test it without meeting up every time.

For the Machine Learning part of the project we made use of TensorFlow and Reinforcement Learning via DDPG provided by TensorFlow. The benefit we found in using TensorFlow was

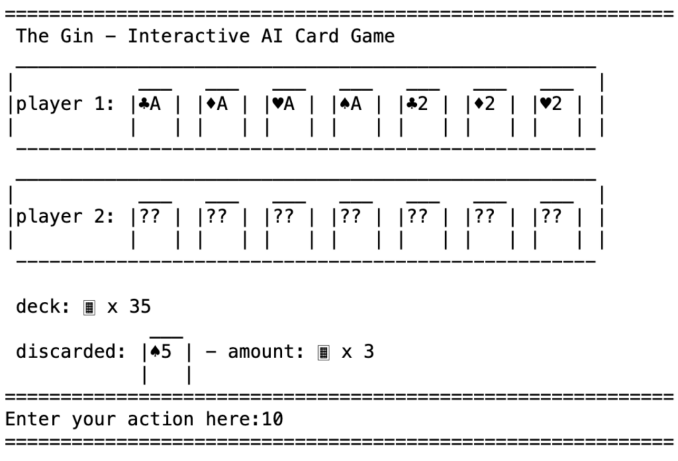

Fig. 9. Player vs. Player

that the library implements all machine learning algorithms and strategies one can think of, this way we could try different approaches while still working in the same library and could therefore avoid having to bridge multiple different libraries. Reinforcement Learning is the key to our project where the neural network can train by playing against a replica of itself. This way the neural network can learn from the thousands of hours of human play-time equivalent in a matter of hours. This way we could achieve meaningful results without having to invest many hours of our time into the training and could focus on writing the code itself and fine tuning the network instead. DDPG is one of the many Reinforcement Learning Agent training environments. The characteristic of DDPG is that it allows for continuous control in the environment, where the agent is expected to take multiple actions in a row to control its environment. [1, 7]

IX. Functionalities

A. What has been done and What remains to be done

The primary objective of this project is to develop an artificial neural network (ANN) capable of playing the card game Gin, specifically designed to challenge human players. Up to this point, we have successfully implemented the core functionalities necessary for the game. Looking ahead, our focus will be refining the training process to enhance the agent's proficiency in gameplay. In detail, the current system enables a player to compete against the trained agent, with the interface displaying each move, along with the details of the cards in hand and on the table. The agent is now adept at forming valid card combinations and discarding cards strategically. However, there are several key areas that we plan to develop further. Firstly, we intend to introduce varying difficulty levels, allowing players to select an appropriate challenge level that matches their skill. This will involve developing different strategic models for the ANN, each calibrated to a specific level of gameplay complexity. Additionally, we aim to enhance the user interface to make the game more engaging and user-friendly. This could involve graphical improvements, more intuitive layout designs, and real-time feedback mechanisms to aid players in understanding the game dynamics as they play. Finally, ongoing efforts are being made to fine-tune the agent's decision-making processes. This includes improving its ability to predict and counter human player strategies, requiring more sophisticated data analysis and algorithmic adjustments. Through these enhancements, we hope to create a more challenging and enjoyable experience for players, pushing the boundaries of what artificial intelligence can achieve in competitive card games.

X. Implementation Issues and Difficulties

A. Tensorflow Environment

Our most significant challenge arose from internal incompatibilities among the various TensorFlow modules we utilized. Given its vast array of algorithms, TensorFlow comprises a large codebase with contributions from numerous developers, often leading to discrepancies in module compatibility. These inconsistencies were further compounded by occasional gaps in the documentation, or even complete absences, which hindered our ability to reliably integrate different computational approaches.

This issue necessitated creative solutions to adapt parts of our training environment, ensuring compatibility between distinct modules. Often, we were compelled to delve into TensorFlow's source code directly to diagnose and resolve errors, a process that was both time-consuming and technically demanding. Common problems included data type mismatches and discrepancies in tensor shape expectations, such as conflicts between modules that required tensors to be in 2D versus 1D formats.

To mitigate these issues, we started documenting our findings and solutions to foster a smoother development process for our project and to aid future projects that may encounter similar challenges. We also began contributing to community forums and discussions, sharing our insights and learning from others facing similar obstacles. This collaborative approach not only helped improve our project but also contributed to the broader TensorFlow community, enhancing the overall robustness and usability of the platform.

B. Reward Function on Agent's Behavior

During the development of our project, we encountered challenges with fine-tuning the behavior of the agent's actions. Initially, the reward structure was simple: agents earned points for quickly completing their card combinations and winning the game, with points deducted as the number of turns increased. While this approach aligned with the ultimate goal of winning, it inadvertently led to suboptimal learning outcomes. For instance, agents developed a tendency to draw exclusively from the discard pile, neglecting the face-down pile, and often discarded necessary cards for their combinations. Recognizing these issues, we revised the reward function to encourage more nuanced strategies. In the new system, agents receive rewards for forming incremental combinations and selecting cards that enhance these combinations, while penalties are

	Stage 1	Stage 2	Stage 3
Agent	•Observe hand of 7 cards[3], and last card discarded [1]. •Draw a new card from either face-down deck[2], or last discarded card[1].	•Check win condition with cards on hand [3] and drawn card [4].	•Discard one of the 8 cards on hand [3][4].
Environment	•Discarded deck [1]. •Face-down deck[2]. •Cards on hand[3].	•Drawn card [4].	•Discarded card adds to discarded deck.

Fig. 10. Implementation Overview

imposed for discarding cards that could disrupt existing patterns. This adjustment markedly improved strategic behavior, with agents learning to draw from the correct pile and discard less essential cards. Looking ahead, we aim to soanalyzete the reward function further to foster predictive capabilities in the agents. The goal is for agents to anticipate and counteract their opponents' strategies, effectively blocking them from winning. This next phase will involve complex modeling of opponent behavior and strategic countermeasures, potentially enhancing the agents' competitive edge and decision-making skills in dynamic game scenarios.

XI. Experimental Result and Analysis

The results of the training are very promising, and the agent was able to learn several strategies that it demonstrated when playing against a human player. In Figure 11 it is shown how its strategy evolves over 10 epochs with a higher learning rate against a random action policy. The strategy development demonstrated by this graph shows how the neural network has explored both a strategy of trying to inhibit the opponent from scoring points (at the cost of itself not gaining as many points) as well as trying to maximize how many points it gets (at the cost of allowing the opponent to gain more points).

Strategies that were observed when a human was playing against the agent were: Drawing appropriate cards to build sets and runs Predicting cards in the face-down draw pile Understanding what cards the opponent needs to win, drawing those cards when able, and holding onto them so that the opponent can not draw them and win

The agent had significant trouble understanding which card to discard each turn, and as a result would discard cards that it would need to win. Also, while the strategy to hold on to cards the opponent needs is smart, the issue with it is that as long as the opponent is also holding on to cards from that set or run no-one can win and the game ends in a draw.

It appears that this behavior and how the agent learns it primarily hinges on how the reward for its actions is specified. Right now, every move results in a penalty of 1 point, and winning adds a reward of 100 points, with the game automatically ending in a draw after the 100th turn. Thus a win will give more points the faster it is achieved, and a draw results in -100 points which is worse than losing which results in some amount of points between -1 and -99. This reward setup might be confusing the training, because it trains it to lose if it is not certain that it can win.

A way to improve on this would be to add a penalty of -100 points to ensure that losing is the least preferred outcome. Another issue is also that we did not implement a reward or penalty for drawing and/or discarding proper cards. For example discarding a card from a set or a run should yield a

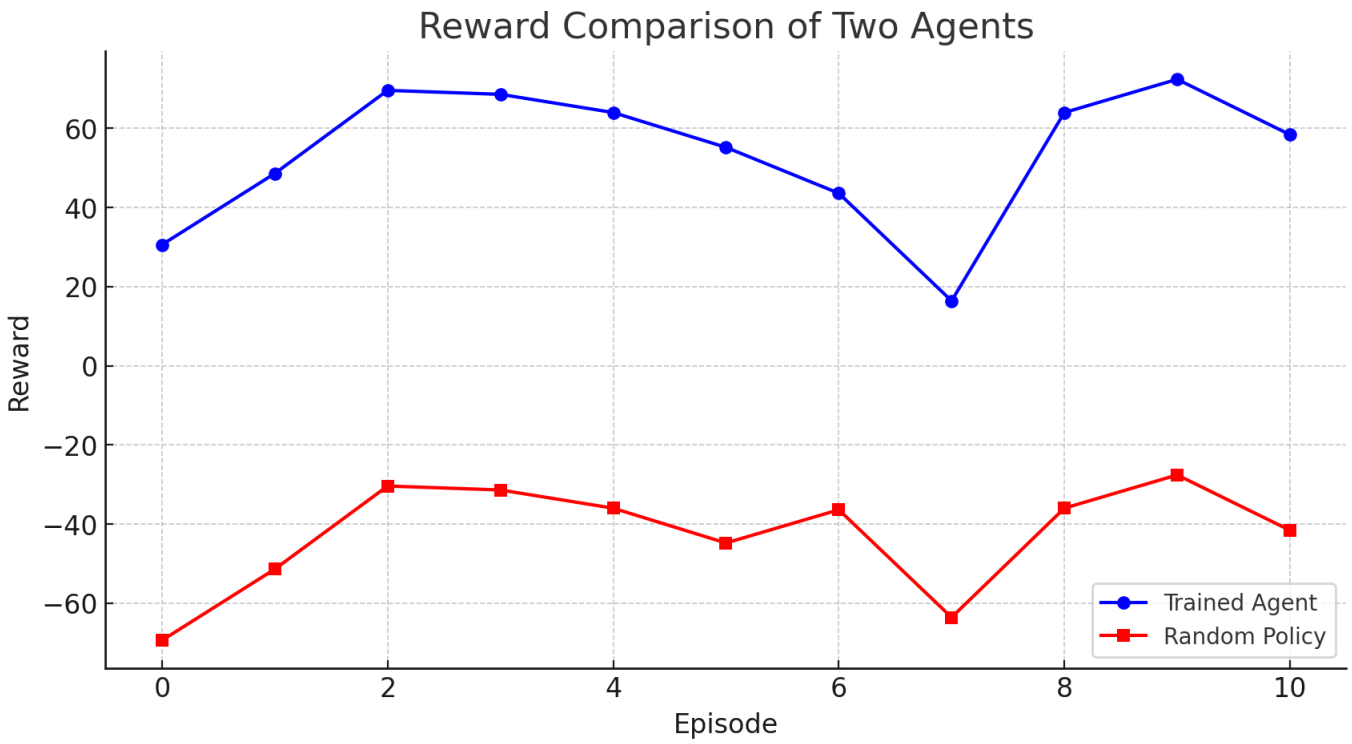

Fig. 11. Reward Comparison of Two Agent

penalty, likewise drawing a card that does not belong to a set or a run should also yield a penalty. The opposite should also hold true, to reward drawing cards that belong to a set or a run and so on. This approach might unfortunately inhibit the strategy of holding cards an opponent needs. Because of the complexity involved with checking all of these properties we were not able to implement, and much less test, this approach in time.

Another approach could be genetic evolution instead of DDPG. The issue with DDPG is that it works based on mapping rewards to certain states and actions, which means there is no direct connection between for example drawing and then discarding a certain card, and then next turn drawing a certain card that results in a win. These are 3 distinct moves, and DDPG relies entirely on the neural network to connect those dots and does not see the entire play as one unit. Genetic evolution would instead create many random agents, that would then all play against each other, then the bottom percentage would be discarded, and finally the remaining agents would have their networks crossed with each other until desired playing proficiency is reached. This way each playing strategy, unique to each agent, would be seen as a single unit. On top of this a genetic approach would allow parallelization of the training as multiple pairs of agents can play against each other simultaneously, unlike in the current DDPG approach that only allows for a serial approach due to working with only one network at a time. This is an approach that we would like to explore as it appears very promising, which we might attempt on our own after the Capstone Project class is over.

XII. Validation

For validation, our target users are enthusiasts of the card game Gin who are interested in challenging a sophisticated AI agent. Initially, our validation process involved observing two agents competing against each other. This stage allowed us to analyze their actions in-depth, facilitating further tuning of the training process while ensuring that the agents' strategies were sound and aligned with genuine game tactics. In the final stage of our validation, team members engaged in multiple rounds of play against the AI to test its proficiency firsthand. What stood out during these sessions was the AI's unique playing style, which differed markedly from traditional human strategies. This distinctive approach provided valuable insights into the AI's learning process and decision-making patterns. Throughout these validation games, we meticulously documented the AI's actions and the game outcomes. This data collection was critical for subsequent analysis and refinement of the AI's performance. By understanding the nuances of the AI's strategies, we could identify any quirks or inefficiencies in its gameplay, allowing us to fine-tune the agent further. Moving forward, we plan to expand our validation by inviting external players with varying levels of expertise to challenge the AI. This broader testing will help us assess the AI's adaptability across different skill levels and refine its strategies to be more versatile and competitive against a wider range of human opponents. This iterative process of testing, analysis, and adjustment is essential for achieving an AI agent that can

Fig. 12. Game Play Demonstration

truly mimic or surpass human-level play in Gin. The game play are shown in figure 12.

XIII. Member Contribution

Each member are assigned task and completed the project beyond expectation.

Contributions are detailed below:

- **Kanglin Xu**
  - Group Web Page: 75%
  - Project Presentation: 75%
  - Project Report: 60%
  - Project Design: 50%
  - Agent Script: 25%
  - Game Interface: 75%
  - Meeting Arrangement: 50%
  - Readme: 50%
- **Tomas Rohatynski**
  - Group Web Page: 25%
  - Project Presentation: 20%
  - Project Report: 35%
  - Project Design: 50%
  - Agent Script: 75%
  - Game Interface: 25%
  - Meeting Arrangement: 50%
  - Readme: 50%
- **Zech Dunlap (Previous Member)**
  - Project Presentation: 5%
  - Project Report: 5%

XIV. Conclusion

The report comprehensively outlines the motivation behind the project, detailing the objectives, expected outcomes, development and difficulties. It also covers the project's implementation, including the tools and methodologies employed. Notably, the report incorporates UML diagrams such as user, case, and sequence diagrams, which elucidate the project's structure and workflow. The core of the project involves developing an artificial neural network-trained agent capable of playing straight gin against a human opponent. This agent is crafted using Python and leverages the TensorFlow Machine Learning framework for robust training. The training methodology integrates object-oriented programming principles, reinforcement learning across multiple agents, sophisticated data visualization techniques, and optimization strategies to enhance performance. Later on in the report, it delve into the potential solutions, discussing the system architecture and offering a more detailed view of the project implementation, thus setting the stage for future enhancements and applications. Finally, the report conclude with result analysis, comparison and validation with intended user.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] J. Hu, M. P. Wellman *et al.*, "Multiagent reinforcement learning: theoretical framework and an algorithm." in *ICML*, vol. 98, 1998, pp. 242–250.

[3] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Applied Sciences*, vol. 11, no. 11, p. 4948, 2021.

[4] Me!, "Gin rummy," Dec 2018. [Online]. Available: https://www.rummyrulebook.com/pages/gin-rummy/

[5] ——, "Straight gin," Jan 2019. [Online]. Available: https://www.rummyrulebook.com/pages/straight-gin-rummy/

[6] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Neural Information Processing Systems (NIPS)*, 2017.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.