Project Report

Artificial Intelligence

[CSE3705]

Uno AI

By

Ankur Kaushal

210C2030218

Pratyut

210C2030222

Department of Computer Science and Engineering
School of Engineering and Technology
BML Munjal University

November 2023

Declaration by the Candidates

We hereby declare that the project entitled *"Uno AI*" has been carried out to fulfill the partial requirements for completion of the course **Artificial Intelligence** offered in the 5th Semester of the Bachelor of Technology (B.Tech) program in the Department of Computer Science and Engineering during AY-2023-24 (odd semester). This experimental work has been carried out by us and submitted to the course instructor *Dr. Soharab Hossain Shaikh***.** Due acknowledgments have been made in the text of the project to all other materials used. This project has been prepared in full compliance with the requirements and constraints of the prescribed curriculum.

Ankur Kaushal

Pratyut

Place: BML Munjal University

Date: 17 November, 2023

TABLE OF CONTENTS:

1. INTRODUCTION

UNO is a multiplayer card game where the primary goal is to be the first player to empty their hand of cards. At the start of the game, each player is dealt seven cards, and turns involve drawing cards from the deck. If a drawn card matches one in a player's hand, they can play it, reducing their card count. However, if there's no matching card, the player must draw from the deck, potentially increasing the cards they need to discard to win.

A crucial aspect of UNO is the strategic use of action cards, such as skips, reverses, and wild cards, which can disrupt the flow of the game and add an element of unpredictability. The game often leads to exciting and unexpected twists, as players try to outsmart each other by employing various tactics to hinder opponents or protect themselves.

When a player is down to only one card, they must announce "UNO!" to signal their imminent victory. However, the game introduces a twist – if an observant opponent calls out "UNO" before the player does, the player in question is penalized by having to draw four additional cards.

UNO's simplicity, coupled with its potential for strategic depth and the element of surprise, makes it a widely enjoyed and enduring card game across diverse age groups and settings. The game's blend of chance and strategy ensures that each round is a unique and engaging experience for all participants

Fig1: Uno cards

1.1 Game Description

1.1.1 Setup

- The game accommodates 2 to 10 players.
- Seven cards are distributed to each player from the deck, and individuals can only view their own set of cards.
- The undisclosed remainder of the cards is stacked in a draw pile.
- The initial card from the draw pile is placed face-up, marking the commencement of the discard pile.

1.1.2 Deck

The Uno deck is composed of four colors: Red, Yellow, Green, and Blue. In total, the deck encompasses 108 cards, encompassing a variety of card types, such as Reverse, Skip, +4, +2, Wild, and numbered cards ranging from 0 to 9. The cards with colors (Red, Yellow, Green, and Blue) and those distinct from numbered cards are recognized as special cards.

Fig2: All the Special cards

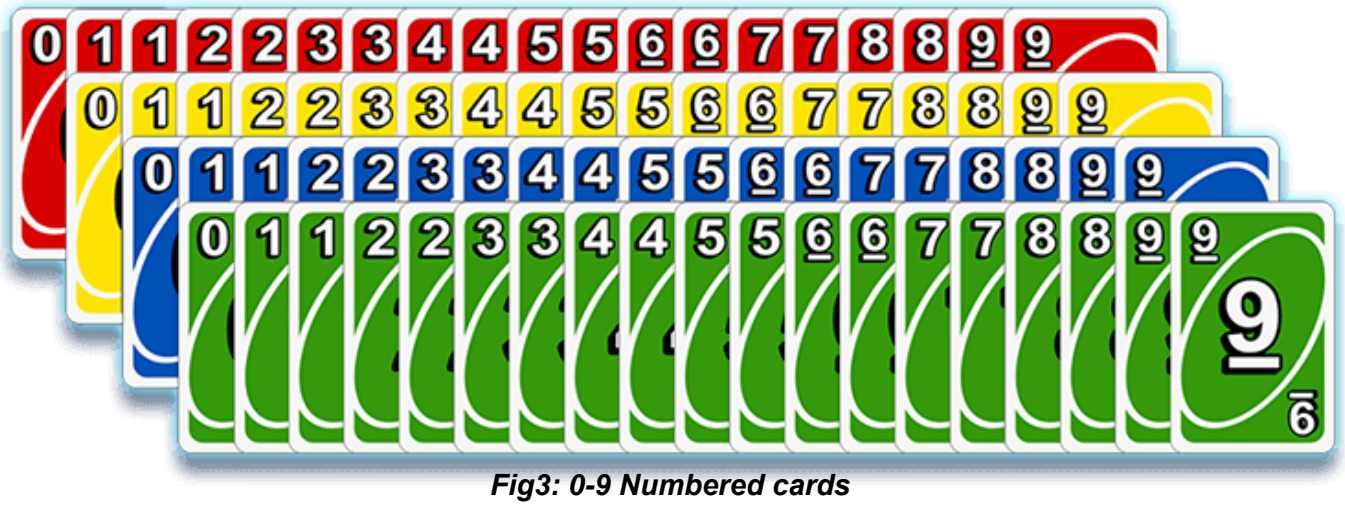


Fig3: 0-9 Numbered cards

Value	Count
Numbers 1 – 10	1 of each color
Reverse	1 of each color
Skip	1 of each color
Draw 2	1 of each color
Wild	4
Draw 4	4

Table 1: Cards of Uno.

1.1.3 Game Play

The game commences with the player situated immediately to the left of the dealer and progresses in a clockwise direction among the players. During a player's turn, the primary aim is to place one of their cards in the discard pile. To achieve this, the player must present a card from their hand that matches the top card on the discard pile in terms of number, color, and/or action.

Should a player lack a matching card, they are obligated to draw a card from the draw pile. If no matching card is obtained, the turn passes to the next player. Alternatively, a player has the option to play a "wild card," allowing them to designate a new color for the top card on the draw pile[6].

Certain cards introduce variations in gameplay. Playing a reverse card results in a reversal of the gameplay direction, while a skip card causes the next player to be skipped. Playing a draw card forces the subsequent player to draw a card from the draw pile.

The ultimate objective of the game is to be the first player to exhaust all of their cards. The game concludes when any player successfully plays all their cards, leaving them with none remaining in their hand.

1.2. Problem Statement

The objective of this project is to design and implement an artificial intelligence system that can proficiently play Uno, a dynamic and strategic card game. Uno, known for its blend of chance and decision-making, requires strategic card play, adept card matching, and adaptability to opponents' actions. The goal is to develop an AI that not only comprehends the rules of the game but also exhibits human-like gameplay, considering factors such as strategic decision-making, precise card matching, and an understanding of opponents' strategies. The aim is to create an AI system that provides a challenging and enjoyable gaming experience, ensuring seamless integration with the rules and dynamics of Uno while engaging players in a manner that mirrors human intelligence.

1.2.1 Card Matching: The AI needs to recognize and match cards based on color, number, or action, ensuring it follows the game's rules accurately.

1.2.2 Decision-Making: The AI should make intelligent decisions during its turn, considering factors like playing action cards strategically and adapting its strategy based on the current game state.

1.2.3 Opponent Modeling: The AI should be able to analyze and adapt to different playing styles, predicting opponents' moves and adjusting its strategy accordingly.

What We Aim to Achieve:

At the end of this project, we aim to have a fully functional Uno AI system that satisfies the following criteria:

Competitive Gameplay: The AI consistently plays Uno in a manner that challenges and engages players, providing a competitive experience.

Adaptability: The AI adapts its strategy based on changing game conditions, opponents' actions, and different player styles.

2. Methodology

To initiate the development of Uno, we adopt a table-driven approach by establishing a comprehensive set of rules. This ensures that the AI operates in accordance with defined guidelines, allowing for systematic and efficient implementation[4].

Deck: A standard deck with 108 cards: four colors (Red, Yellow, Green, Blue), each with numbered cards (0 to 9), action cards (Skip, Reverse, Draw Two, Draw Four), and Wild cards.

Initial Deal: Each player is dealt seven cards. Remaining cards form the draw pile, and the top card is placed face-up to start the discard pile.

Gameplay: Players take turns clockwise On a turn, play a card matching the top card by color, number, or action. If no match, draw a card; play if possible. Special cards (Skip, Reverse, Draw Two) have specific effects when played.

Wild Cards: Wild cards allow the player to choose the next color.

Skip: Skips the next player's turn.

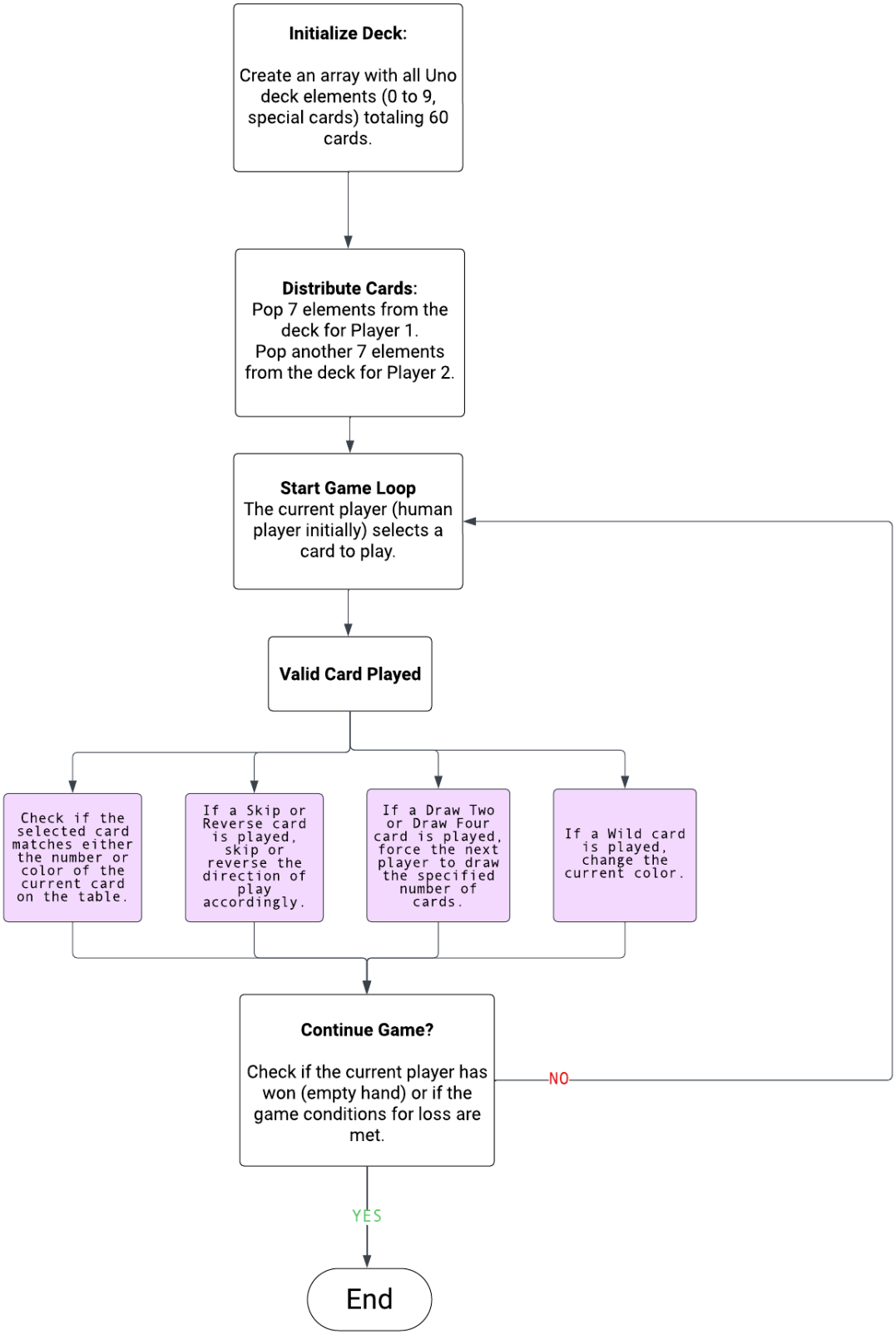
Reverse: Reverses the direction of play.

Draw Two: Forces the next player to draw two cards.

Draw Four: Forces the next player to draw 4 cards

2.1 Table Driven Approach:

Flow Diagram:



2.1.1 Percepts:

Percept	Description
Current Game State	Representation of the Uno game state, including player hands, draw pile, and discard pile.
Top Card on Discard Pile	Attributes of the card currently on top of the discard pile (color, number, or action).
Opponents' Recent Moves	History of recent moves made by opponents, aiding in opponent modeling.

Table2: Defining all the percepts

2.1.2 Actions:

Action	Description
Play Card	The AI plays a card from its hand, reducing its card count.
Draw Card	The AI draws a card from the draw pile, potentially increasing its card count.

Change Color (Wild Card)	If the AI plays a Wild Card, it can designate a new color for the top card on the discard pile.
Reverse Gameplay	If the AI plays a Reverse Card, it reverses the direction of gameplay.
Skip Opponent's Turn	If the AI plays a Skip Card, it skips the next opponent's turn.
Draw Cards for Opponent	If the AI plays a Draw Card, the next opponent must draw cards from the draw pile and chance of that player after drawing card is skipped

Table3: Defining all the Actions

2.1.3 Percept to Action Mapping:

Matching Card in Hand:
- If there's a matching card in the AI's hand for the top card on the discard pile, the AI plays that card.

No Matching Card in Hand:
- If there's no matching card, the AI draws a card.

Wild Card Play:
- If the AI decides to play a Wild Card, it designates a new color for the top card on the discard pile.

Reverse Card Play:
- If the AI plays a Reverse Card, it reverses the direction of gameplay.

Skip Card Play:
- If the AI plays a Skip Card, it skips the next opponent's turn.

Draw Card Play:
- If the AI plays a Draw Card, the next opponent must draw cards from the draw pile.

2.1.4 Moves

- **Play a card from its hand:** Choose a card from the Uno AI's hand and play it onto the discard pile, considering the current active color and special card effects.
- **Draw a card from the deck:** If unable to play a card from its hand, the Uno AI should draw a card from the deck and evaluate its options.
- **Challenge a "Draw Two" or "Skip" call:** If the Uno AI suspects that another player's "Draw Two" or "Skip" call is invalid, it can challenge the call, forcing the player to prove their hand or draw two/skip a turn.

2.2 Intelligent Agent

The Q-Learning algorithm[13] emerges as a strategic solution for optimizing decisions in the context of the Uno AI project. Its distinctive strength lies in its ability to navigate between immediate rewards and delayed reinforcements. In the Uno AI scenario, at each step, the agent observes the current game state vector (xt) and strategically selects and applies an action (ut). Upon transitioning to the next state (xt+1), the agent receives reinforcement (r(xt, ut)). The training objective is to determine a sequential series of actions that maximizes the cumulative sum of future reinforcements, ultimately leading to an intelligent decision-making process for the Uno AI and facilitating the most efficient path from the initial game state to a winning outcome.

In its basic form, Q-learning works in a similar way. However, while MC waits for the completion of each episode before updating q-values, Q-learning updates them with a lag of one step, at each step.

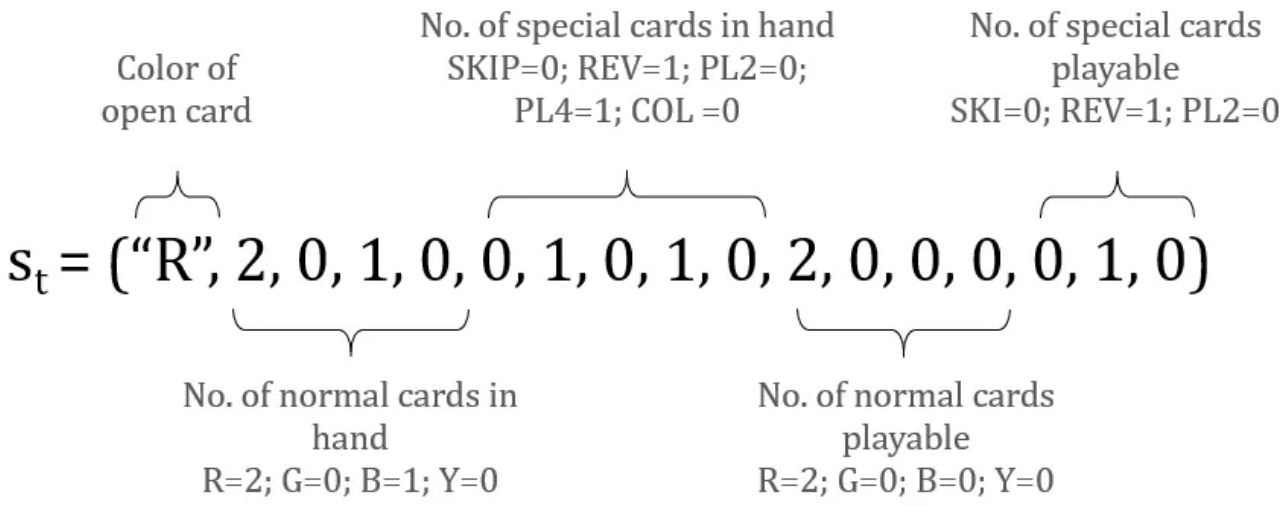
Q-value update function
The q-value is thereby dependent on the step-size parameter, the reward of the next step r, and the q-value of the next step at state s-hat and action-hat.

Algorithms consequently take the same 2 parameters which have the following effects:

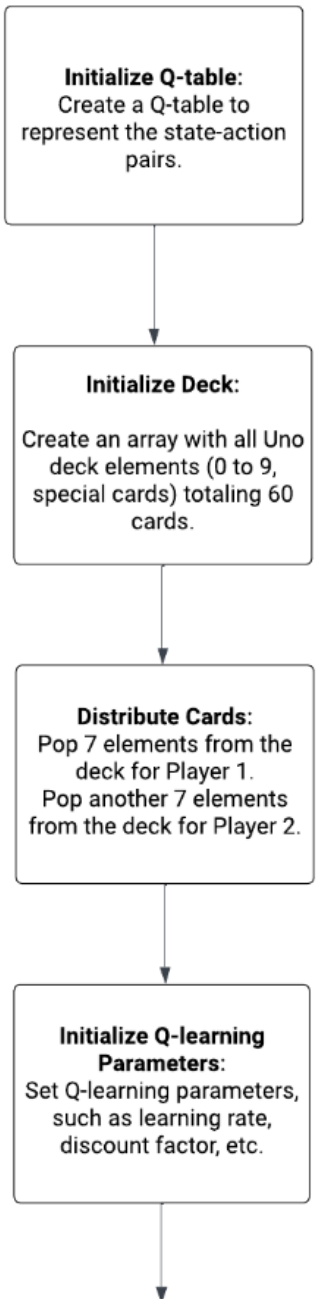
Alpha: A higher step size parameter increases the change in q-values at each update while prohibiting values to converge closer to their true optimum
Epsilon: A higher epsilon grants more exploration of actions, which do not appear profitable at first sight. At the same time, this dilutes the optimal game strategy when it has been picked up by the agent.

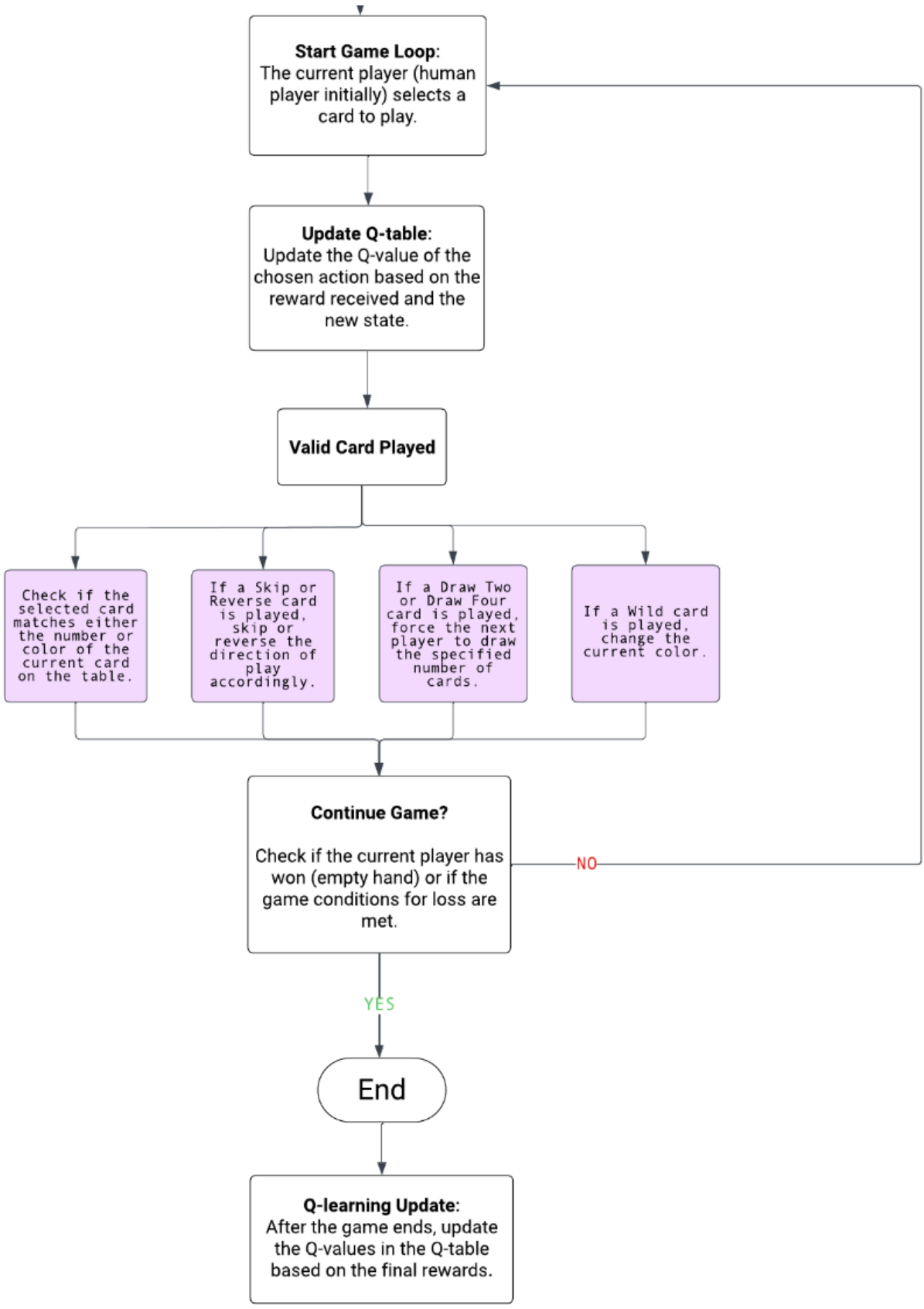
The strategy employed for the Uno AI centers around leveraging the Q-Learning algorithm to instill intelligent decision-making. The key elements of our strategy include:

We are focusing on creating an AI which can easily identify the moves and play accordingly with these steps:



Flow Diagram:




Start Game Loop:
The current player (human player initially) selects a card to play.
Update Q-table:
Update the Q-value of the chosen action based on the reward received and the new state.
Valid Card Played
Check if the selected card matches either the number or color of the current card on the table.
If a Skip or Reverse card is played, skip or reverse the direction of play accordingly.
If a Draw Two or Draw Four card is played, force the next player to draw the specified number of cards.
If a Wild card is played, change the current color.
Continue Game?
Check if the current player has won (empty hand) or if the game conditions for loss are met.
NO
YES
End
Q-learning Update:
After the game ends, update the Q-values in the Q-table based on the final rewards.

2.2.1 State Representation for Uno AI:

The state representation involves capturing essential information that defines the current game context. The state includes:

In a supervised machine learning set-up, a function is fitted to map the available features to the output variable. RL on the other hand evaluates each action sequentially and individually at each step. Therefore states, actions, and rewards need to be defined.

A state can represent any information available to the decision-maker that is useful to describe the current situation of the game. This could be the type of cards he holds, the number of cards the opponent holds, or information regarding cards that have already been played.

At first sight, UNO might appear to be a very simplistic card game, due to its limited set of rules. However, when figuring out the possible combinations of cards a player can hold, it quickly gets out of hand. To be precise there are about 10^{163} combinations. Since the RL-agent has to learn an optimal action for each state, it makes sense to limit the number of states.

Thus, I formed a 16-digit state identification, that captures which cards the agent holds and which he can play, differentiating only between colors, and special cards (skip, reverse, etc.)

Player's Hand:

- A vector representing the cards in the player's hand. Each element in the vector corresponds to a specific card, with information about color, number, and type (action or number card).

Top Card on Discard Pile:

- Attributes of the card currently on top of the discard pile. This includes color, number, and type (action or number card).

Opponents' Hands:

- Information about the number of cards in each opponent's hand. This provides insight into their game progress.

Direction of Gameplay:

- An indicator of the current direction of gameplay (clockwise or counterclockwise).

Draw Pile Status:

- Information about the draw pile, such as the number of remaining cards.

2.2.2 Action Space Definition for Uno AI:

The action space for the Uno AI comprises a set of possible moves that the agent can take during its turn. These actions include:

Play Specific Card:

- The agent chooses to play a specific card from its hand. This action involves selecting a card that matches the top card on the discard pile in terms of color, number, or action.

Draw from Deck:

- The agent decides to draw a card from the draw pile. This action is taken when there is no matching card in the player's hand.

Play Wild Card:

- If the agent holds a Wild Card, it can choose to play it, allowing the selection of a new color for the top card on the discard pile.

Reverse Gameplay:

- If the agent has a Reverse Card, it may choose to play it, thereby reversing the direction of gameplay.

Skip Opponent's Turn:

- If the agent possesses a Skip Card, it can decide to play it, skipping the next opponent's turn.

2.2.3 Q-Table Initialization:

Define States and Actions:

- Identify various game situations (states) and potential moves (actions) in Uno, such as playing a card or drawing from the deck.

Determine Table Size:

- Decide the dimensions of the Q-table based on the identified states and actions, resembling a chart with rows and columns.

Assign Initial Values:

- Allocate initial Q-values to each cell in the chart, starting with random numbers. This provides a baseline for the Uno AI to commence its learning process.

Balance Exploration:

- Foster exploration of different actions early on by selecting initial values that strike a balance between experimenting with new strategies and adhering to learned behaviors.

Handle End-of-Game:

- If the Uno game has potential end-game scenarios, set Q-values for actions in those situations to zero since no further decisions are required.

Allow Dynamic Updates:

- Ensure the Q-table is adaptable and can be updated as the AI learns from playing Uno. This flexibility enables the table to evolve based on rewards and the game's progression.

2.2.4 Training Episodes for Uno AI:

Simulated Uno Environment:

- Create a virtual Uno setting that replicates the essential elements of the game, including player hands, the discard pile, draw pile, and rule dynamics.

Episode Initialization:

- Commence each training episode by setting up the game environment with a shuffled deck, random player positions, and an initialized game state.

2.2.5 Agent's Observation:

- At each time step within the episode, the Uno AI observes the current game state, taking into account details such as the player's hand, the top card on the discard pile, and information about opponents.

Decision-Making Process:

- Based on the observed state, the AI employs its decision-making strategy to choose actions, including playing specific cards, drawing from the deck, or executing special moves.

Reward Assignment:

- After taking an action, the AI receives a reward based on the game's progression. Rewards are carefully designed to encourage favorable actions and discourage suboptimal ones, guiding the AI toward strategic decision-making.

Q-Learning Updates:

- Utilize the Q-Learning algorithm to iteratively update the Q-values in the agent's Q-table. This involves incorporating information about the current state, the action taken, the received reward, and the maximum Q-value for the next state.

Iteration Within Episodes:

- Repeat the observation, decision-making, reward assignment, and Q-Learning update process for a predefined number of time steps within each training episode.

Episode Conclusion:

- Conclude the training episode when a terminal game state is reached or after a specified number of time steps. This marks the completion of one simulated Uno game within the training session.

Cumulative Learning:

- Over multiple training episodes, the Uno AI accumulates knowledge about optimal strategies, refining its decision-making capabilities through the reinforcement learning process.

Adaptation to Varied Scenarios:

- Ensure that the AI adapts its decision-making strategy to diverse scenarios encountered during different training episodes, fostering flexibility in gameplay.

2.2.6 Evaluation and Fine-Tuning for Uno AI:

Performance Assessment:

- Evaluate Uno AI's performance by simulating games. Assess adherence to rules, ability to identify optimal moves, and competitiveness against opponents.

Metric Definition:

- Define success metrics like correct move percentage, win rate, and efficiency in playing all cards to measure Uno AI's performance.

Parameter Analysis:

- Fine-tune key parameters (learning rate, exploration-exploitation balance) based on evaluation. Optimize for better decision-making and adaptability.

Reward Structure Refinement:

- Analyze the effectiveness of the reward system. Refine it to reinforce strategic moves and discourage less optimal actions, enhancing overall strategy.

Q-Table Optimization:

- Review and optimize the Q-table using observed states and actions. Adjust Q-values to better reflect optimal strategies, aligning decision-making with strategic goals.

Handling Special Cards: Implement specific strategies for handling special cards, such as "Skip," "Draw Two," and "Reverse," to maximize their impact.

Adaptive Learning: Incorporate adaptive learning mechanisms to allow the Uno AI to adjust its strategies based on the playing styles of different opponents.

Long-term Planning: Implement long-term planning techniques to enable the Uno AI to consider the consequences of its actions and make strategic decisions.

3. Implementation - Technology Stack

C++ ProgrammingLanguage: C++ is a versatile and widely used programming language renowned for its applicability in machine learning and reinforcement learning applications. It boasts a robust ecosystem of libraries and tools tailored to meet the unique demands of these computational tasks.

Python Programming Language: Python is a versatile and widely used programming language renowned for its applicability in machine learning and reinforcement learning applications. It boasts a robust ecosystem of libraries and tools tailored to meet the unique demands of these computational tasks. Python's readability, extensive community support, and ease of integration make it a popular choice for developing and implementing algorithms in the fields of machine learning and reinforcement learning. Its rich set of libraries, such as TensorFlow, PyTorch, and scikit-learn, further contribute to Python's prominence in the development of advanced computational models.

VS code: Employed for interactive development, experimentation, and visualization of Uno AI components during the implementation phase.

Documentation:
Sphinx: Utilized for generating project documentation, providing a comprehensive reference for codebase structure and functionalities.

Collaboration and Communication:
GitHub: Utilized as a collaborative platform for code hosting, issue tracking, and project management.

4. Conclusions

In the pursuit of creating an intelligent Uno AI, we have embarked on a journey marked by meticulous design, strategic implementation, and iterative refinement. Our focus on move identification and strategic play has driven the development of an AI that excels in effortlessly recognizing optimal moves and executing them with finesse. The Uno AI's training episodes, simulated in a carefully crafted environment, have laid the groundwork for adaptive decision-making, refining its strategy over multiple iterations.

The evaluation and fine-tuning phase, marked by comprehensive performance assessments and parameter adjustments, has been instrumental in enhancing the AI's proficiency. Metrics defining success, such as correct move percentages and win rates, have guided the refinement process, ensuring the Uno AI aligns with strategic goals. The utilization of reinforcement learning techniques, particularly Q-learning, has empowered the AI to optimize decision-making, ultimately providing users with an intelligent and challenging opponent.

Our technology stack, carefully selected for its efficiency and scalability, has played a pivotal role in the project's success. From Python and TensorFlow for machine learning components to Visual Studio Code for development, each tool has contributed to a cohesive and productive workflow.

As we conclude this phase of the project, the Uno AI stands as a testament to our commitment to innovation and excellence. Its adaptability, strategic prowess, and seamless integration into gameplay scenarios make it a compelling addition to the realm of AI-driven gaming. Moving forward, continuous learning and refinement will remain at the forefront, ensuring the Uno AI evolves alongside emerging strategies and dynamic gameplay scenarios.

In summary, this endeavor has not only resulted in the creation of a sophisticated Uno AI but has also provided valuable insights into the intricacies of reinforcement learning, adaptive decision-making, and the application of AI in the gaming landscape. The Uno AI project stands as a testament to the possibilities that arise when technology and strategy converge, offering a glimpse into the future of intelligent and immersive gaming experiences.

5. References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,
Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. CoRR, abs/1708.05866, 2017.
[3] Pablo Barros, Ana Tanevska, and Alessandra Sciutti. Learning from learners: Adapting reinforcement learning agents to be competitive in a card game, 2020.
[4] Henry Charlesworth. Application of self-play reinforcement learning to a four-player game of imperfect information. CoRR, abs/1808.10442, 2018.
[5] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfectinformation games. CoRR, abs/1603.01121, 2016.
[6] Mattel Inc. Uno Instructional Sheet, 2001.
[7] Qiqi Jiang, Kuangzheng Li, Boyao Du, and Hao Chen amd Hai Fang. Deltadou: Expert-level doudizhu ai through self-play, 2019.
[8] Mykel Kochenderfer, Tim Wheeler, and Kyle Wray. Algorithms for Decision Making. GitHub, 2020.
[9] Yuxi Li. Deep reinforcement learning. CoRR, abs/1810.06339, 2018.
[10] Xiaobai Ma, Katherine Driggs-Campbell, Zongzhang Zhang, and Mykel J. Kochenderfer. Monte-carlo tree search for policy optimization, 2019.
[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan
Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, Feb 2015.
[12] Andrew, A. M. (1999). REINFORCEMENT LEARNING: AN INTRODUCTION by Richard S. Sutton and Andrew G. Barto, Adaptive Computation and Machine Learning series, MIT Press (Bradford Book), Cambridge, Mass., 1998, xviii + 322 pp, ISBN 0-262-19398-1, (hardback, £31.95). Robotica, 17(2), 229–235. https://doi.org/10.1017/s0263574799211174
[13] Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292. https://doi.org/10.1007/bf00992698

6. Appendix

Table Driven:

```
#include <iostream>
#include <queue>
#include <algorithm>
#include <vector>
#include <string.h>
#include <random>
#include <chrono>

using namespace std;

struct card{
  string color;
  int number;
  string action;
};


void showq(queue<card> gq)
{
          queue<card> g = gq;
          while (!g.empty()) {
       card fe = g.front();
                    cout << fe.color << " : " << fe.number << " : " << fe.action << endl;
                    g.pop();
          }
}

void print(vector<card> v)
{
    for (card i: v) {
       if(i.color != ""){
          cout << i.color << " ";
       }
       if(i.number != 0){
          cout<<i.number << " ";
       }
       if(i.action != ""){
          cout<<i.action << " ";
       }
```

```
        cout << " | ";
    }
    cout<<endl;
}

void pr(card c){
    if(c.color != ""){
        cout << c.color << " ";
    }
    if(c.number != 0){
        cout << c.number << " ";
    }
    if(c.action != ""){
        cout << c.action << " ";
    }
    cout<<endl;
}

void displayGameState(vector<card> ai , vector<card> human){
    //cout<<"AI has cards : "<<ai.size()<<" left."<<endl;
    cout<<"A.I. hand ---"<<endl;
    print(ai);
    cout<<"HUMAN hand ---"<<endl;
    print(human);
}

int main()
{
    vector<card> deck;

    string color[4] = {"Red","Blue","Green","Yellow"};
    string special_card[3] = {"skip","reverse","+2"};

    for(int i=0;i<4;i++){
        for(int j=1;j<=10;j++){
                deck.push_back(card{color[i],j,""});
        }
        for(int j=0;j<3;j++){
                deck.push_back(card{color[i],0,special_card[j]});
        }
    }
    for(int i=0;i<4;i++){
                deck.push_back(card{"",0,"wild_card"});
```

```
}
for(int i=0;i<4;i++){
            deck.push_back(card{"",0,"+4"});
}


cout << "DECK.SIZE() : " << deck.size() << endl;

unsigned seed = chrono::system_clock::now().time_since_epoch().count();
shuffle (deck.begin(), deck.end(), default_random_engine(seed));

queue<card> sdeck;
for(int i=0;i<deck.size();i++){
   sdeck.push(deck[i]);
}

vector<card> ai;
vector<card> human;

for(int i=0;i<7;i++){
   ai.push_back(sdeck.front());
   sdeck.pop();
}

for(int i=0;i<7;i++){
   human.push_back(sdeck.front());
   sdeck.pop();
}

// initializing top card

card topCard = sdeck.front();
sdeck.pop();

bool isAi = true;
int input;
vector<card> playedCards;
bool skip = false;
int draw = 0;
string changeColor = "";

while(true){
   if(human.size() == 0){
```

```
    cout << "HUMAN WINS" << endl;
    break;
}
else if(ai.size() == 0){
    cout << "A.I. WINS" << endl;
    break;
}
cout << "\n==========================================================" <<endl;
cout << "\nCurrent Game state ----" << endl;
displayGameState(ai,human);
cout << "\ntopCard --" << endl;
pr(topCard);

if(sdeck.size() == 0){
    shuffle (playedCards.begin(), playedCards.end(), default_random_engine(seed));
    for(;playedCards.size()!=0;){
        sdeck.push(playedCards.front());
        playedCards.erase(playedCards.begin());
    }
}

//A.I.

if(isAi){

    if(skip){
        cout << "A.I. SKIPPED" << endl;
        skip = false;
        for(;draw>0;draw--){
                ai.push_back(sdeck.front());
                sdeck.pop();
        }
        isAi = false;
        continue;
    }
    else{
        cout << "\nnot skipping" << endl;
    }

    cout << "\n ---- A.I.'s move ---- " << endl;
    string color = topCard.color;
    int number = topCard.number;
    string action = topCard.action;
```

```
card selectedCard;
int selectedCardNumber = 0;


if(topCard.number == -1){
    for(int i=0;i<ai.size();i++){
            if((ai[i].color.compare(color) == 0) && (color != "")){
                    cout << "color found" <<endl;
                selectedCard = ai[i];
                selectedCardNumber = i+1;
                break;
            }
            else if ((ai[i].action.compare(action) == 0) && (action != "")){
                    cout << "action found\n" <<endl;
                selectedCard = ai[i];
                selectedCardNumber = i+1;
                break;
            }
    }
}
else{
        for(int i=0;i<ai.size();i++){
            if((ai[i].color.compare(color) == 0) && (color != "")){
                    cout << "color found" <<endl;
                selectedCard = ai[i];
                selectedCardNumber = i+1;
                break;
            }
            else if (ai[i].number == number && number != 0){
                    cout << "number found" <<endl;
                selectedCard = ai[i];
                selectedCardNumber = i+1;
                break;
            }
            else if ((ai[i].action.compare(action) == 0) && (action != "")){
                    cout << "action found\n" <<endl;
                selectedCard = ai[i];
                selectedCardNumber = i+1;
                break;
            }
        }
}
```

```
        if(selectedCardNumber == 0){
            for(int i=0;i<ai.size();i++){
                if(ai[i].action.compare("")!=0 &&ai[i].color.compare("") == 0 && ai[i].number == 0){
                    selectedCard = ai[i];
                    selectedCardNumber = i+1;
                    break;
                }
            }
            if(selectedCardNumber == 0){
                cout << "A.I. drawing card" << endl;
                ai.push_back(sdeck.front());
                sdeck.pop();
                isAi = false;
                continue;
            }
        }


        cout << "Card Seleted : ";
        pr(selectedCard);
        cout << "Card Number is : " << selectedCardNumber <<endl;
        playedCards.push_back(topCard);
        ai.erase(ai.begin()+(selectedCardNumber-1));

        if(selectedCard.color.compare("") == 0 && selectedCard.number == 0 &&
selectedCard.action.compare("") != 0){
                    if(selectedCard.action.compare("+4") == 0){
                        changeColor = ai[0].color;
                        skip = true;
                        draw = 4;
                        selectedCard.color = changeColor;
                        selectedCard.number = -1;
                    }
                    else{
                        changeColor = ai[0].color;
                        skip = false;
                        draw =0;
                        selectedCard.color = changeColor;
                        selectedCard.number = -1;
                    }
                }
        else if(selectedCard.action.compare("") != 0){
```

```
            if(selectedCard.action.compare("skip") == 0){
                    skip = true;
                    draw = 0;
                    changeColor = "";
                }
                else if(selectedCard.action.compare("reverse") == 0){
                    skip = true;
                    draw = 0;
                    changeColor = "";
                }
                else{
                    changeColor = "";
                    skip = true;
                    draw = 2;
                }
    }
    topCard = selectedCard;
    isAi = false;
}

//HUMAN

else{

    if(skip){
        cout << "HUMAN SKIPPED" << endl;
        skip = false;
        for(;draw>0;draw--){
            human.push_back(sdeck.front());
            sdeck.pop();
        }
        isAi = true;
        continue;
    }

    cout << "\nHuman Enter the card number 1-" << human.size() << endl;
    cout << "Enter 0 for draw card" << endl;
    cin >> input;
    if(input == 0){
        cout << "drawing card" << endl;
        human.push_back(sdeck.front());
        sdeck.pop();
        isAi = true;
```

```
            continue;
        }
        else if(input < 0 || input > human.size()){
            cout << "--- Invalid Move PLAY VALID MOVE again ---" << endl;
            continue;
        }
        else{
                string color = topCard.color;
                int number = topCard.number;
                string action = topCard.action;
                bool valid = false;
                if
                (
                (
                        (human[input-1].color.compare(color) == 0 || human[input-1].number ==
number)
                         &&
                        (human[input-1].color.compare(color) == 0 ||
human[input-1].action.compare(action) == 0)
                )
                 ||
                      (human[input-1].color.compare("") == 0 && human[input-1].number == 0 &&
human[input-1].action.compare("") != 0)
                )
                {
                      valid = true;
                }
                if(valid){

                    cout << "Card Seleted" << endl;
                    pr(human[input-1]);
                    playedCards.push_back(topCard);
                    topCard = human[input-1];

                    if(human[input-1].color.compare("") == 0 && human[input-1].number == 0 &&
human[input-1].action.compare("") != 0){
                      if(human[input-1].action.compare("+4") == 0){
                            cout << "enter the color : \nRed\nBlue\nGreen\nYellow" <<endl;
                            cin >> changeColor;
                            skip = true;
                            draw = 4;
                            topCard.color = changeColor;
                            topCard.number = -1;
```

```
                    }
                    else{
                        cout << "enter the color : \nRed\nBlue\nGreen\nYellow" <<endl;
                        cin >> changeColor;
                        skip = false;
                        draw =0;
                        topCard.color = changeColor;
                        topCard.number = -1;
                    }
                }

                else if(human[input-1].action.compare("") != 0){
                    cout << "executing ifelseif for action card" << endl;
                    if(human[input-1].action.compare("skip") == 0){
                        changeColor = "";
                        skip = true;
                        draw = 0;
                    }
                    else if(human[input-1].action.compare("reverse") == 0){
                        changeColor = "";
                        skip = true;
                        draw = 0;
                    }
                    else{
                        changeColor = "";
                        skip = true;
                        draw = 2;
                    }
                }

                human.erase(human.begin()+(input-1));

                isAi = true;
            }
        else{
            cout << "Invalid card selected play a valid card" << endl;
        }
    }
  }

    return 0;
}
```

OUTPUT:

```
┌──(kali㉿kali)-[~/Desktop/aipr]
└─$ ./game
DECK.SIZE() : 60

==================================================

Current Game state ----
AI has cards : 7 left.
HUMAN hand ---
Yellow skip  | +4  | wild_card  | Blue 9  | Red 2  | Red 6  | Green 1  |

topCard --
Red 1

not skipping

 ---- A.I.'s move ----
color found
Card Seleted : Red 4
Card Number is : 1

==================================================

Current Game state ----
AI has cards : 6 left.
HUMAN hand ---
Yellow skip  | +4  | wild_card  | Blue 9  | Red 2  | Red 6  | Green 1  |

topCard --
Red 4

Human Enter the card number 1-7
Enter 0 for draw card
5
Card Seleted
Red 2

==================================================

HUMAN hand ---
```

Red 5 |

topCard --
Red 6

not skipping

 ---- A.I.'s move ----
color found
Card Seleted : Red 9
Card Number is : 1

==

Current Game state ----
AI has cards : 2 left.
HUMAN hand ---
Red 5 |

topCard --
Red 9

Human Enter the card number 1-1
Enter 0 for draw card
2
--- Invalid Move PLAY VALID MOVE again ---

==

Current Game state ----
AI has cards : 2 left.
HUMAN hand ---
Red 5 |
topCard --
Red 9
Human Enter the card number 1-1
Enter 0 for draw card
1
Card Seleted
Red 5
HUMAN WINS

Q learning:

```
import pandas as pd
import numpy as np
import random

import src.state_action_reward as sar


class Agent(object):
    def __init__(self, agent_info:dict):
        """Initializes the agent to get parameters and create an empty q-tables."""

        self.epsilon     = agent_info["epsilon"]
        self.step_size   = agent_info["step_size"]
        self.states      = sar.states()
        self.actions     = sar.actions()
        self.R           = sar.rewards(self.states, self.actions)

        self.q = pd.DataFrame(
            data    = np.zeros((len(self.states), len(self.actions))),
            columns = self.actions,
            index   = self.states
        )

        self.visit = self.q.copy()


class QLearningAgent(Agent):

    def __init__(self, agent_info:dict):

        super().__init__(agent_info)
        self.prev_state  = 0
        self.prev_action = 0

    def step(self, state_dict, actions_dict):
        """
        Choose the optimal next action according to the followed policy.
        Required parameters:
            - state_dict as dict
            - actions_dict as dict
        """
```

```
    # (1) Transform state dictionary into tuple
    state = [i for i in state_dict.values()]
    state = tuple(state)

    # (2) Choose action using epsilon greedy
    # (2a) Random action
    if random.random() < self.epsilon:

        actions_possible = [key for key,val in actions_dict.items() if val != 0]
        action = random.choice(actions_possible)

    # (2b) Greedy action
    else:
        actions_possible = [key for key,val in actions_dict.items() if val != 0]
        random.shuffle(actions_possible)
        val_max = 0

        for i in actions_possible:
            val = self.q.loc[[state],i].iloc[0]
            if val >= val_max:
                val_max = val
                action = i

    return action

def update(self, state_dict, action):
    """
    Updating Q-values according to Belman equation
    Required parameters:
        - state_dict as dict
        - action as str
    """
    state = [i for i in state_dict.values()]
    state = tuple(state)

    # (1) Set prev_state unless first turn
    if self.prev_state != 0:
        prev_q = self.q.loc[[self.prev_state], self.prev_action].iloc[0]
        this_q = self.q.loc[[state], action].iloc[0]
        reward = self.R.loc[[state], action].iloc[0]

        print ("\n")
        print (f'prev_q: {prev_q}')
```

```
            print (f'this_q: {this_q}')
            print (f'prev_state: {self.prev_state}')
            print (f'this_state: {state}')
            print (f'prev_action: {self.prev_action}')
            print (f'this_action: {action}')
            print (f'reward: {reward}')

            # Calculate new Q-values
            if reward == 0:
                self.q.loc[[self.prev_state], self.prev_action] = prev_q + self.step_size * (reward + this_q
- prev_q)
            else:
                self.q.loc[[self.prev_state], self.prev_action] = prev_q + self.step_size * (reward -
prev_q)

            self.visit.loc[[self.prev_state], self.prev_action] += 1

        # (2) Save and return action/state
        self.prev_state  = state
        self.prev_action = action
```

Output:

---------- TURN 2 ----------

Current open card: GRE 3

AIs hand:
RED 9
RED 1
RED 3
GRE 8
YEL 7
WILD PL4
RED 2

AIs playable hand:
RED 3
GRE 8
WILD PL4

AI plays RED 3

---------- TURN 3 ----------

Current open card: RED 3

Humans hand:
WILD COL
GRE 0
BLU 8
GRE 5
BLU 7
GRE SKI

Humans playable hand:
WILD COL
enter the card position : 1

Human plays WILD COL
enter the card's color :
RED
YEL
BLU

GRE
GRE

---------- TURN 4 ----------

Current open card: GRE COL

AIs hand:
RED 9
RED 1
GRE 8
YEL 7
WILD PL4
RED 2

AIs playable hand:
GRE 8
WILD PL4

AI plays WILD PL4
AI chooses RED

prev_q: 0.0
this_q: 0.0
prev_state: ('GRE', 2, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0)
this_state: ('GRE', 2, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0)
prev_action: RED
this_action: PL4
reward: 0.0

Human has to draw 4 cards
Human draws YEL SKI
Human draws RED 9
Human draws BLU SKI
Human draws BLU 1

---------- TURN 45 ----------

Current open card: RED 0

Humans hand:
RED 5

Humans playable hand:
RED 5
enter the card position : 1

Human plays RED 5
Human has won!

prev_q: 0.0
this_q: 0.0
prev_state: ('RED', 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
this_state: ('RED', 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
prev_action: RED
this_action: RED
reward: 0.0
Enter 1 to break else Enter anything else to continue : 1
PS C:\Users\praty\OneDrive\Desktop\uno-card-game-rl>

For further info visit this Repository.
Github Repo Link: https://github.com/PratyutCS/uno-ai