CSE 4106: Artificial Intelligence Laboratory

AI-Driven Games: Blackjack and Memory Card Matching By,

Maimuna Chowdhury

Roll: 1907013

Mahmuda Afrin Tuli

Roll:1907019

Date of Submission: 30-06-2024



Department of Computer Science and Engineering Khulna University of Engineering & Technology Khulna 9203, Bangladesh

**Contents**

1. Introduction
2. Objective
3. Project Overview
4. Methodology
5. Programming Environment
6. Programming Language
7. Key Features
8. Final Output
9. Achievements
10. Limitations
11. Future Scope
12. Conclusion
13. References

**Introduction**

The aim of this project is to develop a Blackjack game using Python and Pygame, incorporating AI players to simulate real opponents. The AI players will use various techniques such as heuristic approaches, fuzzy logic, and alpha-beta pruning to make decisions during the game. This project explores different strategies for AI decision-making in Blackjack, providing an engaging and challenging experience for human players.

The second part of this project develops a card matching game using Python that has AI as the player’s opponent. For the decision making of AI and playing more intelligently three approaches Genetic algorithm, Heuristic search, and Alpha Beta Pruning is used in three levels. This ensures that the game is more challenging for the players.

**Objectives**

* Develop a functional Blackjack game using Python and Pygame.
* Implement AI players that can play against human players and a dealer.
* Utilize heuristic approaches to improve AI decision-making.
* Integrate fuzzy logic to allow the AI to make more nuanced decisions.
* Implement alpha-beta pruning to optimize AI performance.
* Creating a card matching game that is engaging and interactive.
* Designing a visually appealing game board with flip-and-match card mechanics.
* Improving players' memory and concentration.
* Developing an AI using heuristic techniques for decision-making.
* Implementing a genetic algorithm for AI to simulate evolutionary strategies.
* Ensuring balanced gameplay for both human players and AI.

**Project Overview**

The project is divided into two main phases:

1. **Blackjack Version 1**
   * Number cards (2-10) are worth their face value.
   * Face cards (J, Q, K) are worth 10 points each.
   * Aces (A) can be worth either 1 or 11 points, whichever is more favorable for the hand.
   * The game is played between player vs dealer vs AI agent.
   * Initially player, dealer and AI agent are dealt with two cards.
   * In player turn, player can decide hit or stand.
   * In dealer turn, dealer can decide hit or stand.
   * In AI agent turn, AI agent can decide hit or stand.
   * If player card value is greater than 21 then player will bust and dealer and AI agent get point.
   * If player card value is 21 then player will win.
   * If player, dealer and AI agent get same value of card and if it is less or equal 21 then game will tie.
2. **Card Matching Game (Version 2):**

* A card-matching game from a game board of 12 pairs of card backs.
* The game is played between the player and the AI agent.
* The game has two levels.
* In each level the player and the AI has scores for matching the cards successfully. The one with the highest score in the corresponding level wins that level.
* The total score of the player and the total score of the AI in the two levels are compared and the final winner is declared.
* Heuristic approach and genetic algorithms are implemented in different levels to make AI agent play as intelligent as possible.

**Methodology**

**1 Blackjack Version 1**

* 1. **Game Setup**

**Imports and Initialization:** Necessary libraries like pygame, random, and other libraries are imported. Game settings like screen dimensions, fonts, and background images are initialized.

**1.2 Card and Deck Management**

**Card Values**: A dictionary is used to defined card values.

**Deck Setup: The one\_deck and decks define the card deck configuration.**

1.3 **Game State Variables:** Variables like records, score\_board, player\_score, and dealer\_score manage game state.

**1.4 Game Initialization**

The game initializes with a Pygame window, setting up the game environment, including the screen, background color, and frames per second (fps) settings.

**1.5 Initial Deal**

At the start of each game, the player, AI, and dealer are each dealt two cards. This is only done once per game.

1.6 **Game Activation and Score Calculation**

Once the game is active, scores are calculated, and cards are displayed. If the dealer's cards are revealed and their score is less than 17, they draw another card.

**Score Calculation:**

* The player’s score is calculated using their current hand of cards.
* The AI’s score is similarly calculated using the cards in the AI's hand.

**Card Drawing:**

* The cards for the player, dealer, and AI are drawn on the screen.
* If the dealer's cards are not yet revealed, they remain face down; otherwise, they

are shown.

**Dealer's Turn:**

* If the dealer's cards are revealed, their score is calculated.
* If the dealer’s score is less than 17, the dealer draws additional cards until their score is 17 or higher.

**Score Display:**

* The scores of the player, dealer, and AI are displayed on the screen.

**2.7 Players Action:**

* If the hit button is clicked, player get a new card.
* If the stand button is clicked, the reveal becomes true and the value of dealer’s hand is calculated and compared with player vs AI agent vs dealer.
* If the player card value is 21, player will win.
* If the player card value greater than 21, player will bust and dealer and AI agent will get point.

**2.8 Dealers Action**

* If the dealers hand value is less than 17, dealer will take more cards.
* If the player's score reaches or exceeds 21, the dealer's hand is automatically

revealed, and the current hand is deactivated.

**2.9 AI’s Action**

* If label is equal 1, AI take decisions to get more cards or not by applying heuristic approach.
* If label is equal 2, AI take decisions to get more cards or not by applying fuzzy algorithm.
* If label is equal 3, AI take decisions to get more cards or not by applying alpha beta pruning algorithm.

**AI Decision-Making using Heuristic Approach:**

**Purpose**

The AI\_cards function evaluates the AI's current hand and decides whether to hit or stand based on its score, the dealer's visible score, and the probabilities of drawing favorable or unfavorable cards from the remaining deck.

1. **Calculating Scores**
   * The function begins by calculating the score of the AI's hand using the calculate\_score function.
   * If the dealer's hand is not fully revealed, the score is calculated using only the visible part of the dealer's hand (typically the second card onwards).
2. **Initial Decision Based on AI's Score**
   * If the AI's score is 17 or higher, the AI stands immediately, as the risk of busting (exceeding 21) is high.
3. **Probability Calculation**
   * The function calculates the difference between 21 and the AI's current score (need), which represents the ideal value the AI aims to achieve.
   * It then iterates through the remaining cards in the deck to calculate:
   * The less calculates the number of cards with values less than or equal to need.
   * The more calculates the number of cards with values greater than need.
   * The card\_A calculates the number of Aces remaining in the deck.
   * The total number of cards in the current deck is stored in total\_card.
   * The prob\_higher is the probability of drawing a card with a value higher than need.
   * The prob\_lower is the probability of drawing a card with a value less than or equal to need.
   * The risk is the probability of drawing an Ace, which can be valued as either 1 or 11.
4. **Heuristic Decision-Making**
   * Score Less Than 12: Always hit, as the risk of busting is minimal.

* Score Between 12 and 16: Hit if the probability of drawing a lower value card (prob\_lower + 2 \* risk) is greater than or nearly equal to the probability of drawing a higher value card (prob\_higher).
* Score of 17 or Higher:
* Hit if the dealer's visible score is 7 or higher, as the dealer likely has a strong hand.
* Stand otherwise, to avoid the high risk of busting.

**AI Decision-Making using Fuzzy Logic:**

1. **Calculate Scores:**

Here the AI's score is calculated using calculate\_score(hand). The dealer's score is calculated using the visible part of the dealer's hand.

1. **Defining Fuzzy Variables:**

Fuzzy variables are defined for the AI's score (ai\_score\_lv), the dealer's score (dealer\_score\_lv), and the decision (decision).

1. **Define Membership Functions:**

* For ai\_score\_lv, define membership functions for categories: 'very\_low', 'low', 'medium', 'high', and 'very\_high'.
* For dealer\_score\_lv, define similar membership functions: 'very\_low', 'low', 'medium', 'high', and 'very\_high'.
* For decision, define membership functions for 'stand' and 'hit'.

**4. Create Fuzzy Rules:**

* Rule 1: If AI's score is very low or (low and dealer's score is very high), then hit.
* Rule 2: If AI's score is medium and dealer's score is medium, then hit.
* Rule 3: If AI's score is high and dealer's score is low, then stand.
* Rule 4: If AI's score is very high, then stand.
* Rule 5: If AI's score is low and dealer's score is high, then hit.
* Rule 6: If AI's score is very low and dealer's score is very low, then hit.
* Rule 7: If AI's score is medium and dealer's score is very low, then hit.
* Rule 8: If AI's score is high and dealer's score is very high, then stand.

* Rule 9: If AI's score is low and dealer's score is medium, then hit.
* Rule 10: If AI's score is medium and dealer's score is high, then stand.

**5 Create Control System:**

* The rules are combined into a control system (ai\_ctrl).

**6 Simulate the Control System:**

* **A simulation object (ai\_decision) is created using the control system.**
* The current scores (ai\_score and dealer\_score) is the input into the simulation.
* **The decision value is calculated.**

**7 Decision making:**

* If the decision value is ≥ 0.5 (indicating a hit):
* Check the potential score with the next card.
* If the potential score ≤ 21, deal a card to the AI.
* If hitting would cause a bust, set the decision to stand (decision\_value = 0.0).

**AI Decision Making Using Alpha Beta pruning:**

**1. Parameters:**

* current\_deck: The current deck of cards.
* dealer\_hand: The dealer's hand.
* is\_maximizing\_player: Boolean to indicate if it is the maximizing player's turn.
* depth: The depth of the search.
* alpha: The alpha value for pruning.
* beta: The beta value for pruning.

**2. Base Case:**

* If depth is 0 or the game reaches a terminal state, return the evaluation of the dealer\_hand.

**3. Maximizing Player:**

* The max\_eval is initialized to negative infinity.
* For each card in the current\_deck:
* The card is removed and add it to dealer\_hand.
* Then alphabeta is called recursively with updated parameters.
* The addition is reverted to dealer\_hand.
* The max\_eval and alpha are updated.
* If beta is less than or equal to alpha then loop is terminated.

**4. Minimizing Player:**

* The min\_eval is initialized to positive infinity.
* For each card in the current\_deck:
* The card is removed and add it to dealer\_hand.
* The alphabeta is called recursively with updated parameters.
* The addition is reverted to dealer\_hand.
* The min\_eval and beta is updated.
* If beta is less than or equal to alpha then loop is terminated.

1. **Decision-Making**:

* If AI's score >= 17, return current hand and deck.
* Initialize best\_move and best\_score.
* For each card in current\_deck, simulate move and evaluate using alphabeta.
* Update best\_move and best\_score.
* If best\_move is found, update hand and current\_deck accordingly.

**2.Memory Card Matching Game (Version 2)**

**2.1 Game Board:**

1. A function get\_png\_list(loc) is defined to retrieve all PNG files from a specified directory. This function lists files in the directory, filters for PNG files, and constructs their full file paths.
2. The main function create\_game\_board is defined to set up the game interface.
3. Images for the game cards are retrieved from a specified directory. Pairs of images are created, duplicated, and shuffled to randomize their positions on the game board.

* The default card back image is loaded and this image is initially displayed on all cards. The cards are displayed in a grid layout on the game board.
* The click event is binded to reveal cards.
* The same card sequence is maintained in the two levels.

**2.2 Player Turn:**

* The game starts with the player turn to pick two cards.
* The on\_click function is responsible for handling the player's interaction when clicking on a card in the memory matching game.
* The function checks if it's currently the player's turn. If it's not the player's turn, the function exits without performing any actions.
* The function increments the click count to keep track of how many cards the player has clicked.

**2.2.1 Player clicking the first card:**

* The on\_click function stores the card.
* Increments total clicks.

• Checks if the player has exceeded the target clicks to determine if the level should end.

* Highlights the first card in green to indicate selection.
* Unbinds the click event for the first card to prevent further clicks on it.

**2.2.2 Player clicking the second card:**

* The on\_click function unbinds the click event for the second card to prevent further clicks.
* Highlights the second card in green to indicate selection.

• Increments click count and checks if it exceeds target clicks for game end conditions.

**2.2.3 Matching Cards:**

The function checks if the first card and the second card the player has clicked matches by checking the name of the cards. If the cards match the on\_click function does the following:

* Increments player score and total player score.
* Adds the matched card's text to match list.
* Increments found matches to track completed matches.
* Calls update\_scores() to update the display with the new scores.
* Resets first card and second card to None.
* Resets player count to 0 for the next turn.
* Updates the status bar to indicate it's now the AI's turn.
* Sets player turn to False to switch turns to the AI.
* Unbinds the click event for the matched cards to prevent further clicks on it.

If the first card and second card clicked by the player do not match then the on\_click function does the following:

* Resets the player count to 0, indicating that the player's turn has completed.
* Appends a list containing both cards to unmatched pairs. This list can be used to keep track of pairs of cards that the player has clicked and identified as not matching.
* Updates the status bar to inform the player that it's now the AI's turn and after 1 second (1000 milliseconds), the AI will execute its turn.
* The clicked cards are flipped back and highlighted by white colour.

**2.3 AI’s Turn:**

**2.3.1 Level 1:**

In level 1 Genetic Algorithm is used for the decision making of AI.

**Initial Setup**

* **Available Cards**: Identifies all cards that haven't been matched yet.
* **Check for Game End**: If there are no available cards, declares a winner.

**Genetic Algorithm Functions**

**Fitness(sequence)**

* **Purpose:**

Evaluates how well a sequence (or set of cards) is based on how many matched pairs the sequence has.

* **Implementation:**

A set is maintained to store the card text that has matching cards in the sequence. each card in sequence is iterated, checks if its text is matches the text of another card in the sequence. If it has matching card in the sequence then the card is added to the set and otherwise it is skipped. The fitness function is defined as the length of the set i.e, the number of matching pairs in the sequence.

* **Returns:**

The fitness score of 0 or more.

**Selecting parents**

* **Purpose:**

Selects parents from a population of card sequences based on their fitness scores.

* **Implementation:**

Adjusts fitness scores by adding 1 to each for ensuring non-zero weights and uses random choices to select two sequences (parents) weighted by their fitness scores.

* **Returns:**

Two selected parent sequences.

**Crossover**

* **Purpose**:

Combines genetic material from two parent sequences to produce offspring (children).

* **Implementation**:

Chooses a random crossover point and swaps sub sequences between parents to create two new sequences (children).

* **Returns**:

Two children sequences resulting from the crossover operation.

**Mutate**

* **Purpose:**

Introduces random changes (mutations) to a sequence.

* **Implementation:**

Iterates through each card in sequence and, based on a mutation rate, replaces cards with random cards that are not match list.

* **Returns:**

The mutated sequence.

**generate\_initial\_population():**

* **Purpose**:

Creates an initial population of card sequences (individuals) for the genetic algorithm.

* **Implementation**:

Uses random selection to create sequences of cards that are not in match list. Each sequence has the length of half of the number of cards.

* **Returns**:

A list of sequences (population) of population size 20.

**AI Decision Making:**

The ai\_turn\_genetic() function does the following:

* Initializes the available cards excluding those in match list.
* Generates an initial population of card sequences.
* Iterates through a number of generations and does the following:
  + Calculates fitness scores for each sequence in the population.
  + Selects parent sequences based on fitness scores.
  + Performs crossover and mutation to generate new sequences.
  + Updates the population with the new sequences.
* Selects the best sequence (highest fitness) from the final population.
* Selects the first card from the best sequence randomly and flips the card and highlights the card using green colour.
* Increments the total click count and checks if the maximum number of clicks has been reached. If so, declares a winner.
* Selects the second card from randomly from rest of the cards in best sequence other than the first card and flips the card and highlights it using green colour.

**2.3.2 Level 2:**

In level 2 Genetic Algorithm along with heuristic approach is used for the decision making of AI.

All the functions of genetic algorithm are implemented in the same way as in the Level 1.

The ai\_turn\_genetic\_heuristic() function does the following:

* Selects the first card randomly from the best sequence and flips the card and highlights it using green colour.
* Increments the total click count and checks if the maximum number of clicks has been reached. If so, declares a winner.
* Checks if the selected first card is in one of the pairs of unmatched pairs. If so, then the other member of that unmatched pair is excluded from the best sequence to avoid the possibility of choosing a unmatching pair of cards.
* Selects the second card randomly from the resulting best sequence if it is not the first card.
* Flips the second card and highlights it using green colour.

**Matching and Unmatching conditions in all levels:**

If the first and second cards that AI has picked is matched then the AI turn does the following:

* Increments the AI's current score and total score.
* Adds the matched card's text to the match list.
* Increment the count of found matches.
* Resets the AI's click count.
* Loops through blank cards to re-enable clicking on cards that are not in the match list.
* Updates the status bar to indicate it's the player's turn.

If the first and second cards that AI has picked does not match then the AI turn does the following:

* Resets the AI count to 0, indicating that the AI's turn has completed.
* Appends a list containing both cards to unmatched pairs. This list can be used to keep track of pairs of cards that the AI has clicked and identified as not matching.
* Updates the status bar to inform the player that it's now the player's turn and the player can choose cards by clicking on them

**Programming Environment**

1. Visual Studio Code

**Programming Language**

1. Python 3.11.3

1. : The system calibrates the background to enhance hand segmentation accuracy.

**Key Features**

**Blackjack Game:**

* AI competes with players using advanced heuristic and fuzzy logic.
* Implementation of alpha-beta pruning for efficient decision-making.
* Dynamic and adaptive gameplay experience.

**Card Matching Game:**

* In level 1 Genetic algorithm is employed for the AI to evolve and improve strategies.
* In level 2 Heuristic approach along with genetic algorithm is used so that the AI can choose the optimal move.

**Pseudocode Functions**

**Blackjack Version 1:**

1. **Heuristic AI Strategy:**

Function heuristic\_ai(hand):

sum = calculate\_hand\_value(hand)

If sum < 17:

return "hit"

Else:

return "stand"

1. **Fuzzy Logic AI Strategy:**

Function fuzzy\_logic\_ai(hand\_value, risk):

Define fuzzy variables:

- Hand value: low, medium, high

- Risk: low, medium, high

Define fuzzy rules:

- If hand value is low and risk is low, then action is hit

- If hand value is medium and risk is medium, then action is hit

- If hand value is high and risk is high, then action is stand

- (Add more rules as needed)

Compute fuzzy output using rules:

- Evaluate current hand value and risk

- Apply fuzzy logic rules to determine the action

If action suggests low risk:

return "hit"

Else:

return "stand"

1. **Alpha-Beta Pruning Strategy:**

Function alpha\_beta\_pruning(state, depth, alpha, beta, maximizing\_player):

If depth == 0 or game\_over(state):

return evaluate\_state(state)

If maximizing\_player:

max\_eval = -∞

For each move in possible\_moves(state):

evaluation = alpha\_beta\_pruning(apply\_move(state, move), depth - 1, alpha, beta, False)

max\_eval = max(max\_eval, evaluation)

alpha = max(alpha, evaluation)

If beta <= alpha:

break

return max\_eval

Else:

min\_eval = ∞

For each move in possible\_moves(state):

evaluation = alpha\_beta\_pruning(apply\_move(state, move), depth - 1, alpha, beta, True)

min\_eval = min(min\_eval, evaluation)

beta = min(beta, evaluation)

If beta <= alpha:

break

return min\_eval

4. **Card Handling Functions:**

**Calculate Hand Value**

Function calculate\_hand\_value(hand):

value = 0

num\_aces = 0

For each card in hand:

If card is an Ace:

num\_aces += 1

Else:

value += card\_value(card)

value += num\_aces \* 11

While value > 21 and num\_aces > 0:

value -= 10

num\_aces -= 1

return value

**Card Value:**

Function card\_value(card):

If card is a face card (J, Q, K):

return 10

Else:

return numeric\_value\_of(card)

**Possible Moves:**

Function possible\_moves(state):

moves = []

If can\_hit(state):

moves.append("hit")

If can\_stand(state):

moves.append("stand")

Return moves

**Apply Move:**

Function apply\_move(state, move):

new\_state = copy\_of(state)

If move is "hit":

draw\_card(new\_state)

Else If move is "stand":

pass // No change in state

Return new\_state

**Evaluate State:**

Function evaluate\_state(state):

player\_value = calculate\_hand\_value(state.player\_hand)

dealer\_value = calculate\_hand\_value(state.dealer\_hand)

If player\_value > 21:

return -∞ // Player busts

Else If dealer\_value > 21:

return ∞ // Dealer busts

Else:

return player\_value - dealer\_value

**Game Over Check:**

Function game\_over(state):

If calculate\_hand\_value(state.player\_hand) > 21:

return True

Else If calculate\_hand\_value(state.dealer\_hand) > 21:

return True

Else:

return False

**Blackjack Version 2:**

**Player’s Turn in the two levels:**

Function on\_click(event):

If it is the player's turn:

Increment player click count

Flip the clicked card to show the front

If this is the first card:

Store this card as the first card

Highlight the first card

Else:

Compare with the previously clicked card

If they match:

Update scores and status

Set AI's turn

Else:

Store unmatched pair

Flip back both cards after a delay

Set AI's turn

**AI’s Turn is in level 1:**

**Function fitness(sequence):**

Initialize an empty set for matches

For each card in the sequence:

Check for matching pairs in the sequence

Add matches to the set

Return the number of matches

**Function select\_parents(population, fitness\_scores):**

Adjust fitness scores (add 1)

Select two parents from the population based on fitness scores

Return the selected parents

**Function crossover(parent1, parent2):**

Randomly select a crossover point

Create two children by swapping parts of the parents' sequences

Return the two children

**Function mutate(sequence):**

For each card in the sequence:

With probability Mem.mutation\_rate, replace with a new random card

Return the mutated sequence

**Function generate\_initial\_population():**

Initialize an empty population list

For each individual in the population:

Create a sequence of random cards

Add the sequence to the population

Return the population

**Function ai\_turn\_genetic():**

Get available cards

Generate initial population

For each generation:

Compute fitness scores for each individual

Create a new population using selection, crossover, and mutation

Determine the best sequence from the final population

Perform actions based on the best sequence

Update the game state and handle player turns

**AI’s Turn is in level 2:**

**Function ai\_turn\_genetic\_heuristic():**

Get available cards

Generate initial population

For each generation:

Compute fitness scores for each individual

Create a new population using selection, crossover, and mutation

Determine the best sequence from the final population

Use heuristic approach to adjust available cards

Perform actions based on the adjusted sequence

Update the game state and handle player turns

**Final Output**

These are the outputs of our project.

Blackjack Vesion1:



Fig 1: Initial Window of BlackJack game.

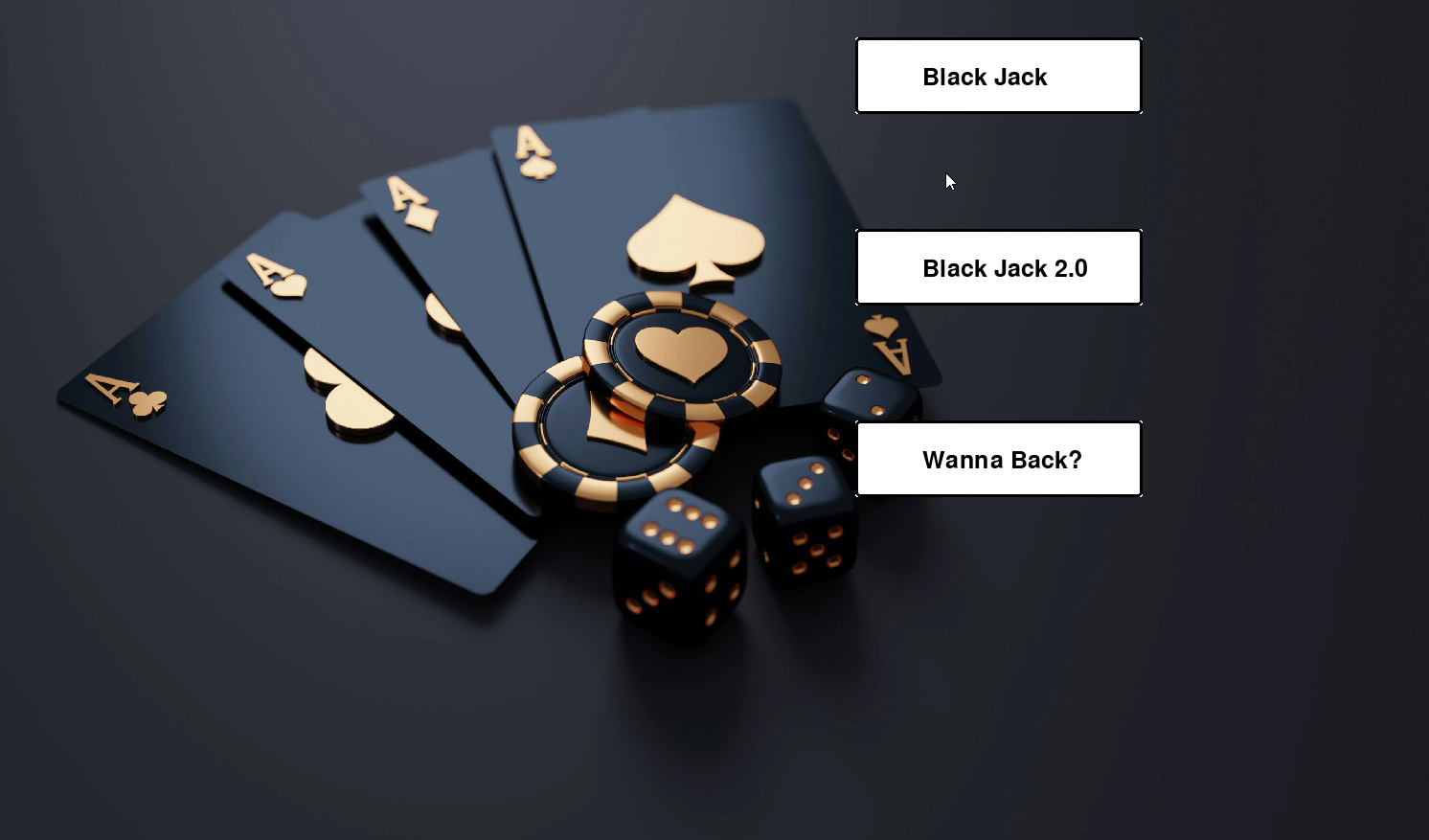


Fig 2: Two versions of the game.

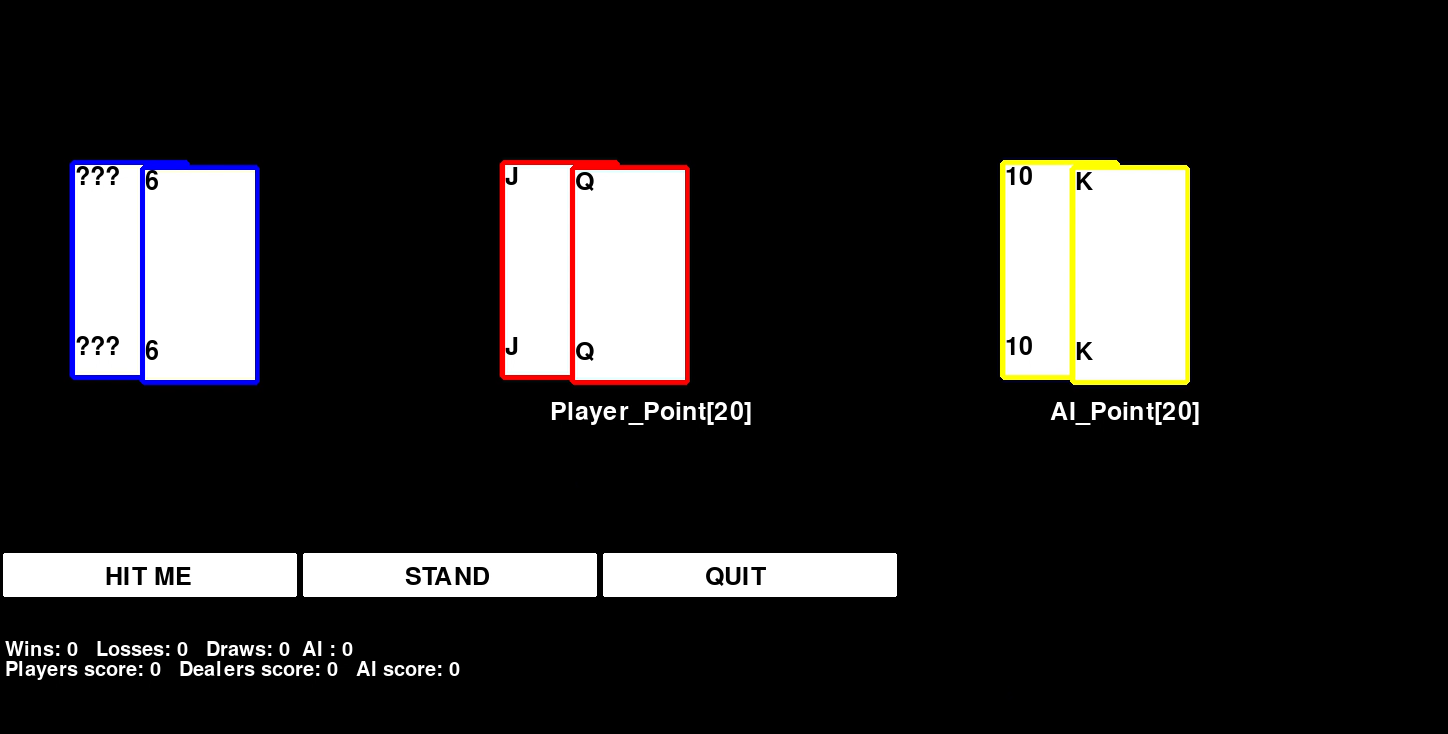


Fig 3: Showing Cards and Scores of Dealer, Player and AI.

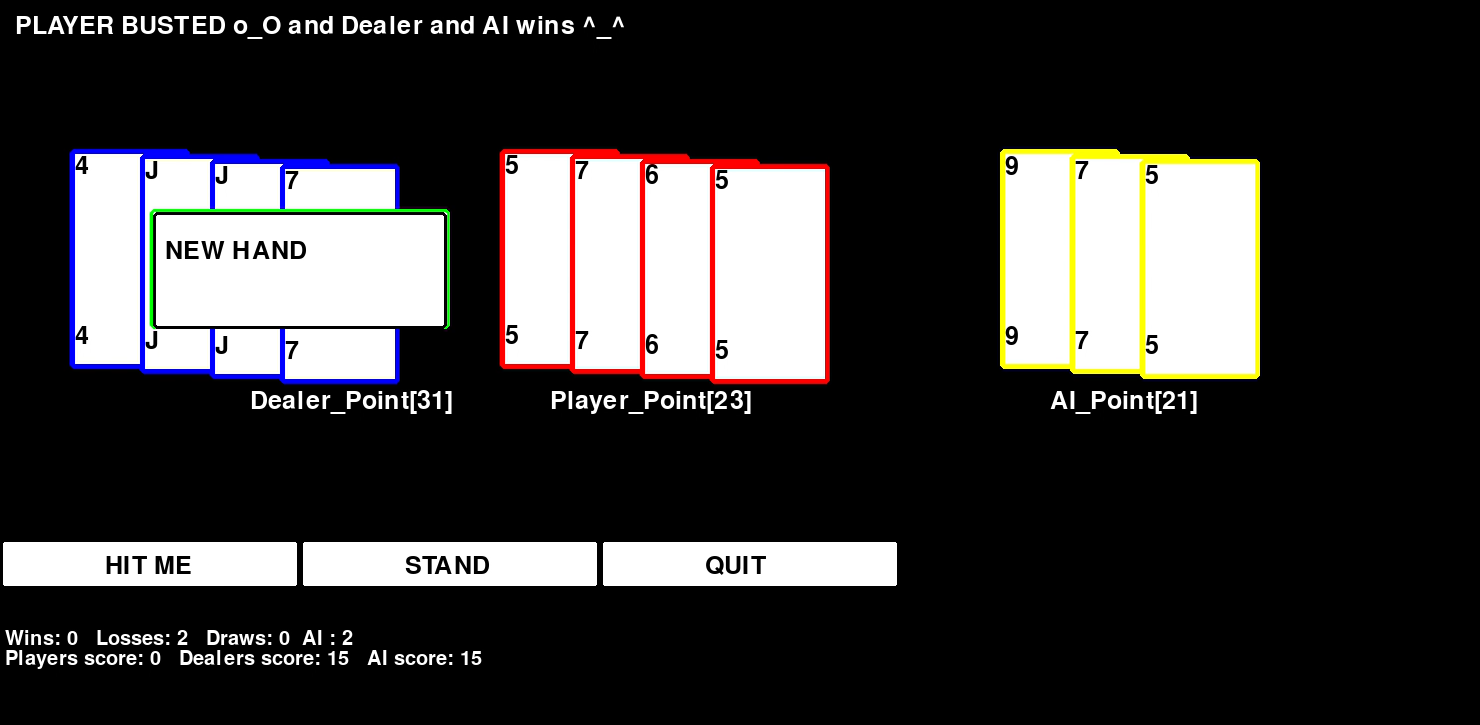


Fig 4: Showing New hand to start another round of game.

Blackjack Version2:

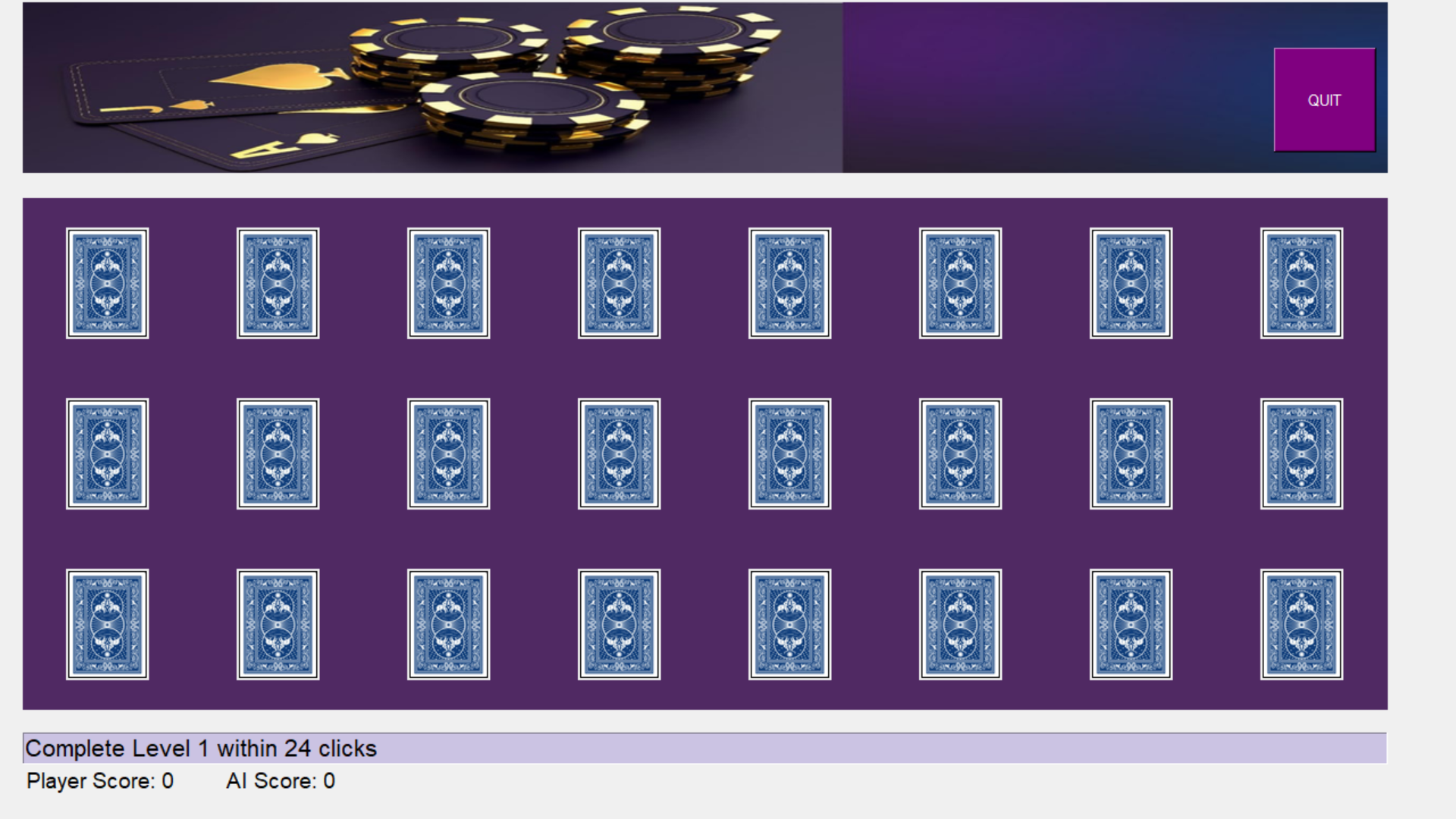


Fig 5: Initial game state showing player score 0 and AI score 0.

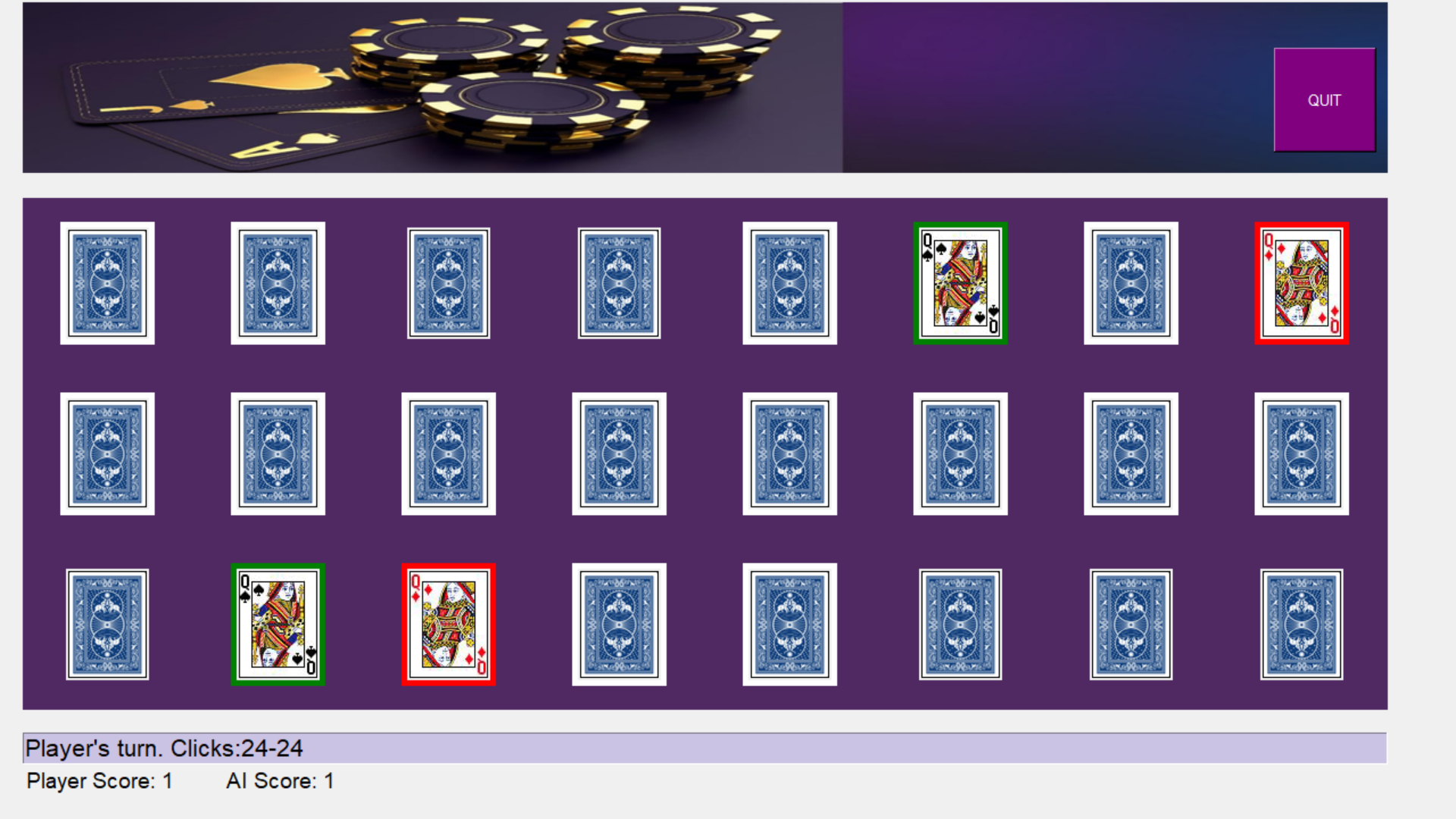


Fig 6: Game state showing player score 1 and AI score 1.

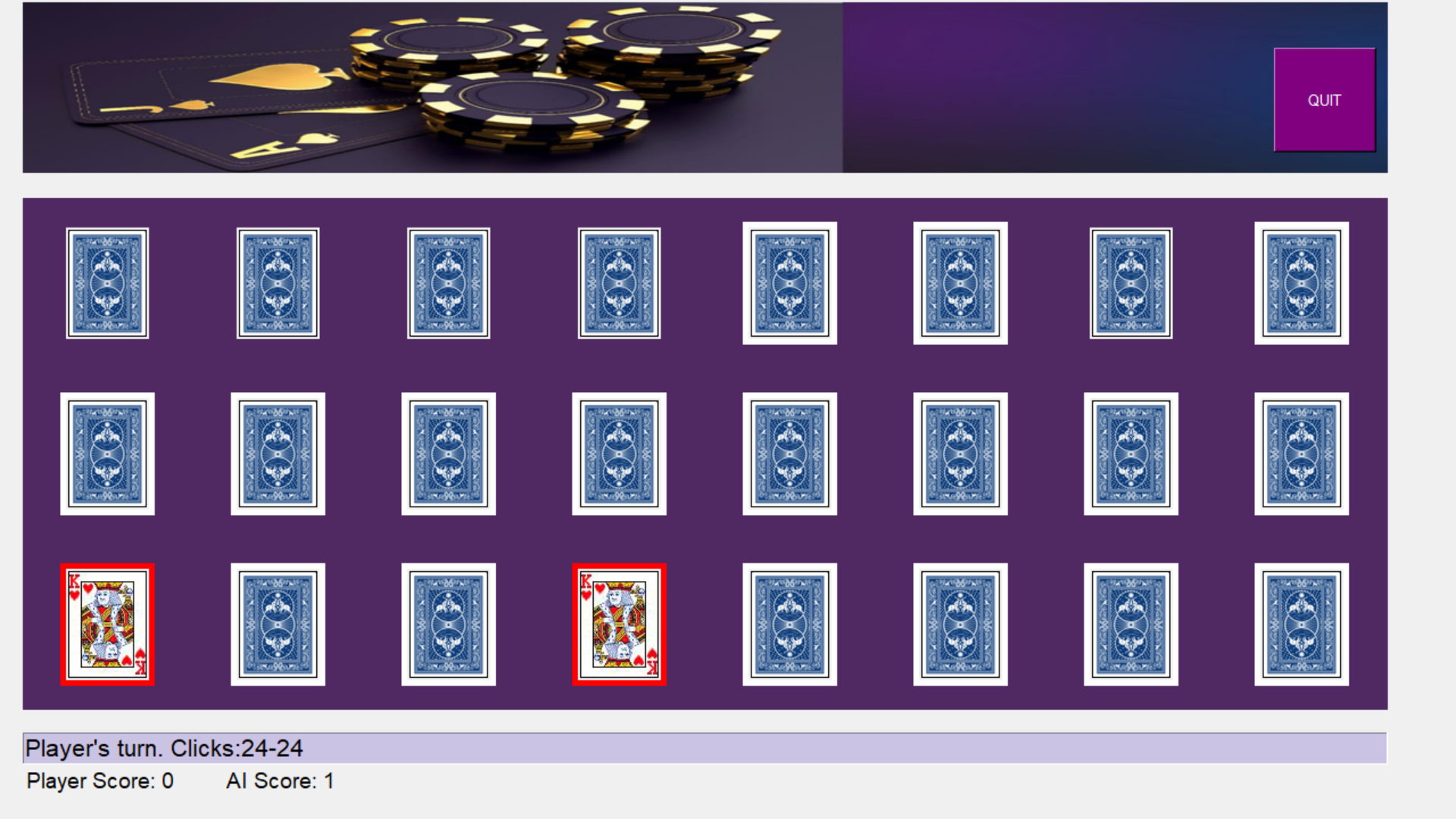


Fig 7: Game state showing player score 0 and AI score 1.

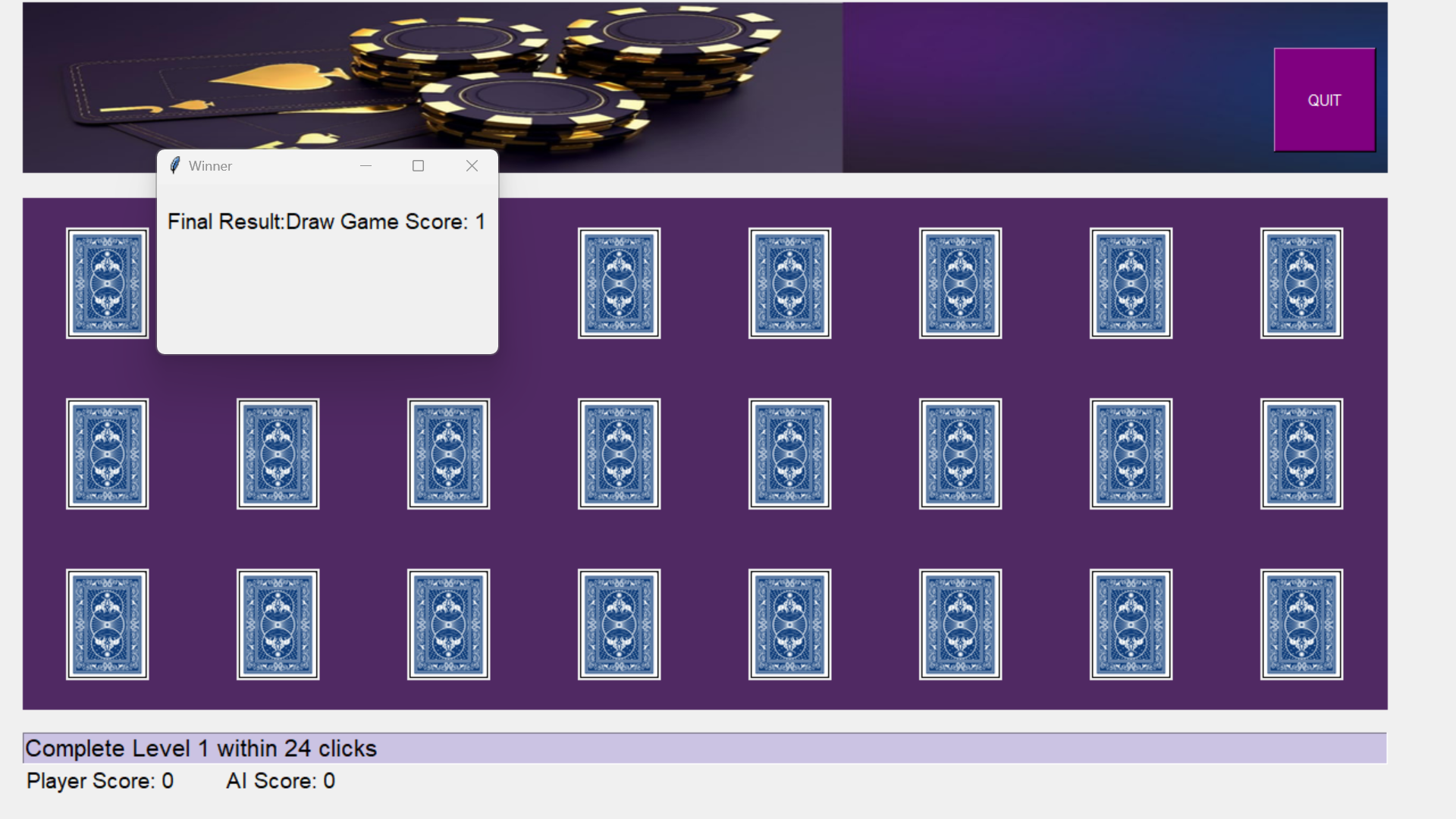


Fig 8: Game state showing the draw game.

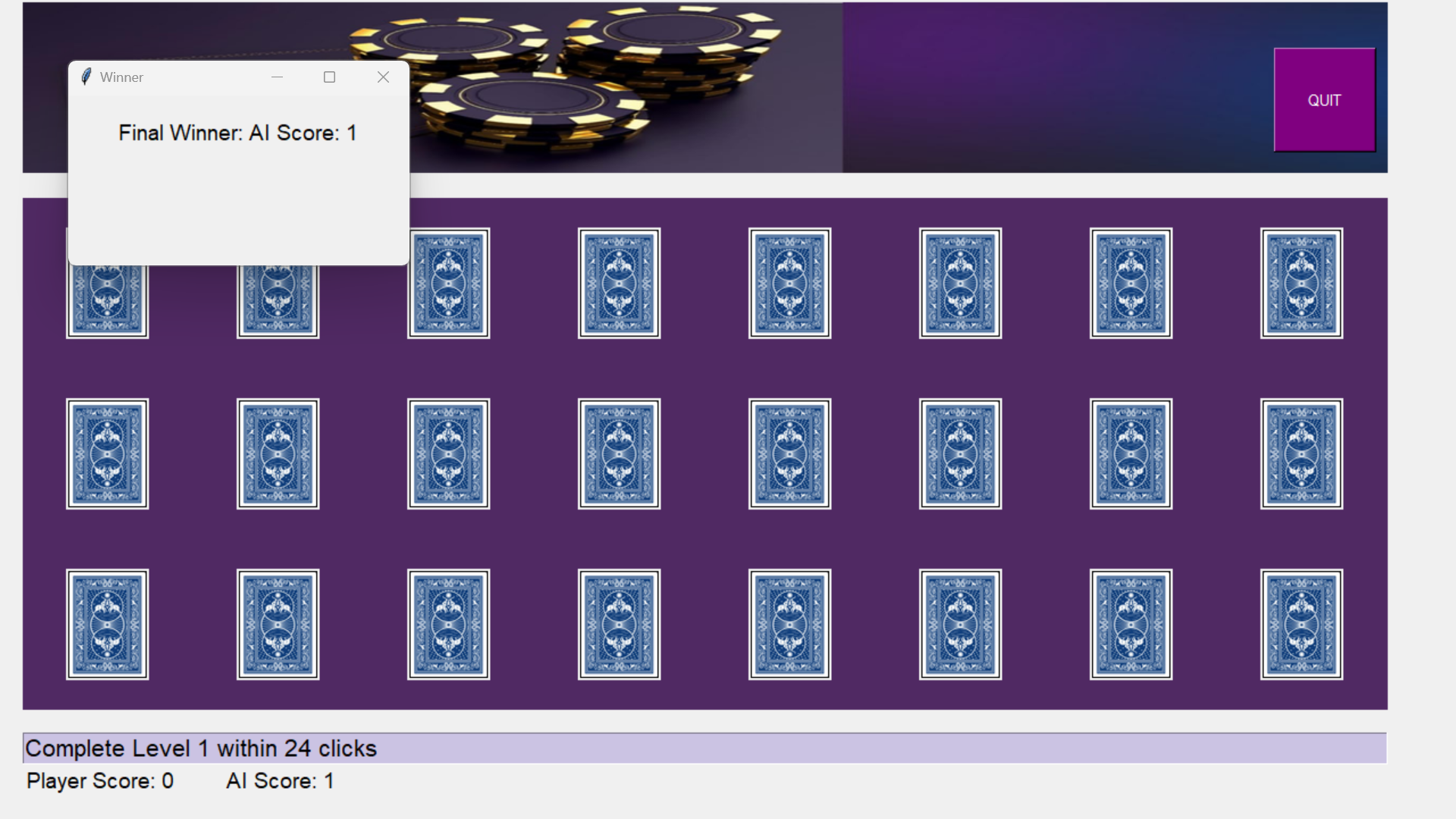


Fig 9: Game state showing AI agent as the final winner

**Achievements**

**Successful Integration of AI Techniques:**

* Demonstrated the application of heuristic approaches, fuzzy logic, and alpha-beta pruning in the Blackjack game.
* Implemented genetic algorithms in the Card Matching game for evolving strategies.

**Enhanced Player Experience:**

* Provided a challenging and adaptive AI opponent in both games.
* Improved gameplay dynamics through intelligent decision-making.

**Limitations**

**Complexity and Computation:**

The implementation of advanced AI techniques increases the computational complexity, which can affect performance on lower-end hardware.

**Scalability:**

The AI strategies, while effective, may require fine-tuning to scale with different levels of difficulty and varied game scenarios.

**Future Scope**

**Enhanced AI Learning:**

Integrate machine learning techniques to allow the AI to learn and adapt to players' strategies over time.

**Multiplayer Support:**

Expand the games to support multiplayer modes, allowing multiple players to compete against the AI.

**Advanced Visuals and User Experience:**

Improve the graphical interface and user experience to make the games more engaging and visually appealing.

**Conclusion**

This project successfully demonstrates the integration of advanced AI techniques in game development. By implementing heuristic approaches, fuzzy logic, alpha-beta pruning, and genetic algorithms, the project showcases the potential of AI in creating intelligent and adaptive gameplay experiences. The games developed provide a robust platform for further exploration and refinement of AI methodologies in interactive environments.

**References**

1. [**https://www.youtube.com/watch?v=e3YkdOXhFpQ**](https://www.youtube.com/watch?v=e3YkdOXhFpQ)