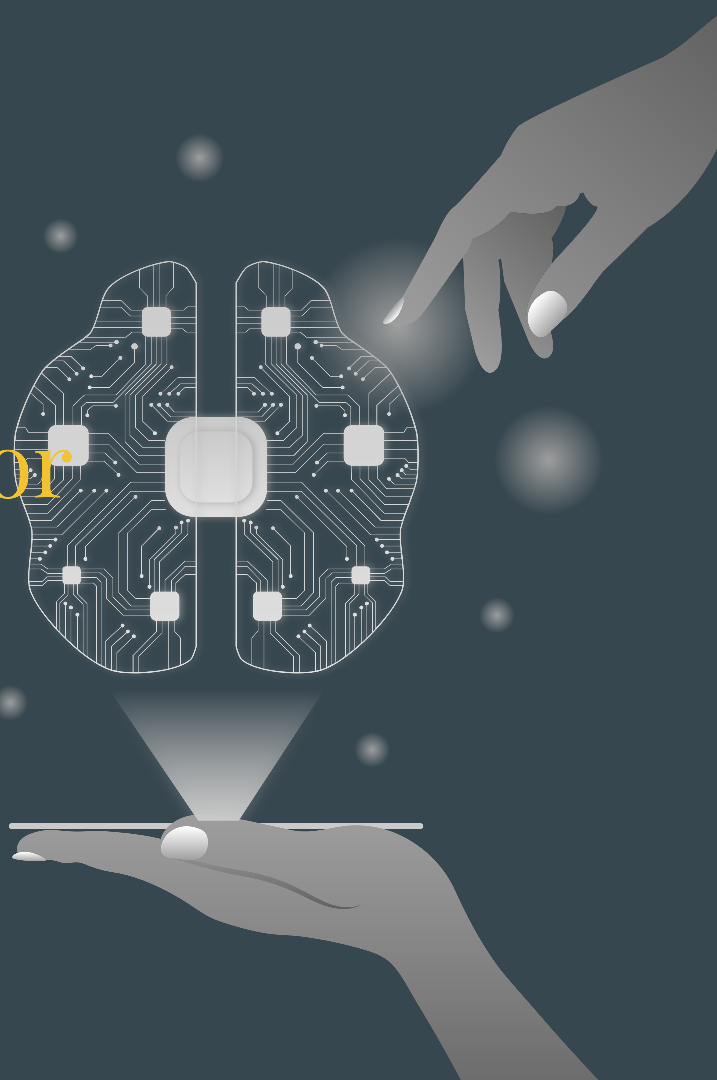
Evaluation of AI Strategies for Blackjack ...



Team 5
111950020 李欣穎
109550072 何嘉婗
109550202 白詩愷
110550092 林啟堯

PROBLEM STATEMENT

Compare two trained Blackjack models and a self-implemented minimax strategy to determine the most effective decision-making and game outcomes.

Objectives

1. Analyze two published ML-based Blackjack strategies

2. Implement a minimax (Expectimax) based strategy

3. Implement an AI agent and compare the methods

4. Identify the most effective AI approach

Timeline_

Justin Bodnar's AI Model

Mason Lee Model

Our Contribution

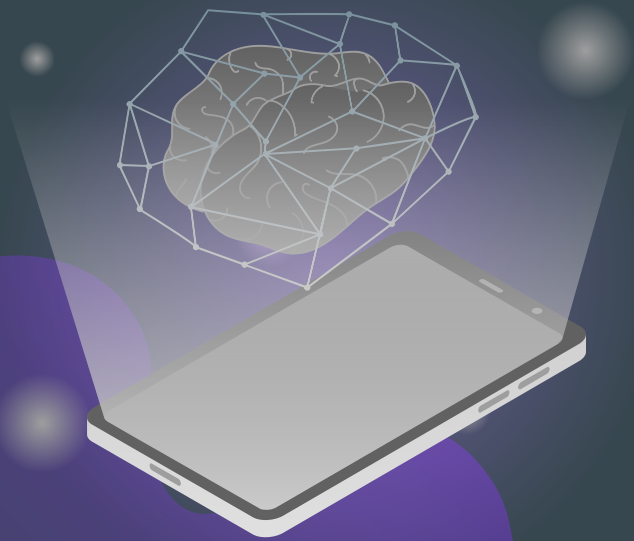
Random and Basic Strategy

Reflect Agent Classifier (RFC)

Baseline-NeuroEvolution of Augmenting Topologies (NEAT)

Expectimax

Result and Analysis



01

Justin Bodnar Model

Goal of the Model

The goal of Bodnar's model is to train network that mimics a basic blackjack strategy that has:

- Input: Information regarding the current game state (e.g., player's hand value)
- Output: Decision to *hit* or *stay*.

Model Architecture

The model proposed uses Monte Carlo simulations of Blackjack games to create training data.

The model runs on a neural network:

- Level 1 » Player and hand value
- Level 2 » Player hand + dealer's face-up card
- Level 3 » All card seen

02

Mason Lee Model

Goal of the Model

The goal of this project is to train an AI agent to play Blackjack by evolving its decision-making strategy using NEAT (NeuroEvolution of Augmenting Topologies).

input: Current game state (player's hand, dealer's visible card, usable ace info, etc.)
Output: AI decides action: hit, stand, double

Model Architecture

Evolve neural networks based on fitness (win rate, performance)

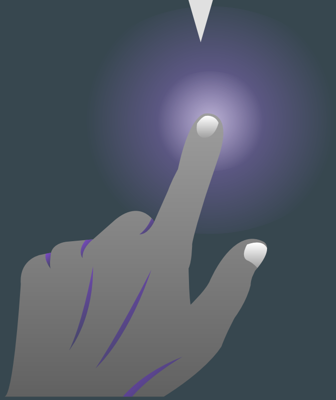
1 ➻ Initialize game
2 ➻ Player decision making
3 ➻ Dealer logic
4 ➻ Determine result

Our Contribution;

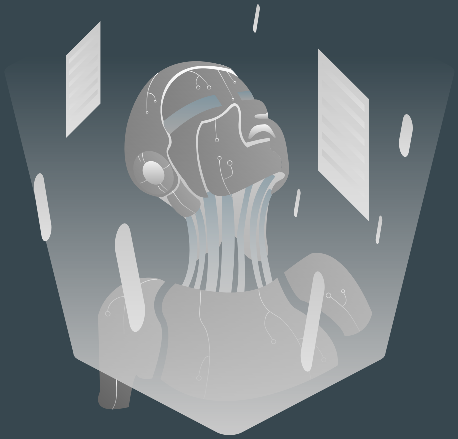
- Implement Agent/ Training /Compare Win Rate of multiple AI strategies
  (Random, Basic, RFC, NEAT)
- Implement these AI agents into an application about Blackjack

AI Strategies;

Random and Basic Strategy



Reflect Agent Classifier(RFC)



NeuroEvolution of Augmenting Topologies (NEAT)

Baseline-Random and Basic Strategy

The Random Agent makes decisions purely based on randomness. At each decision point, the agent randomly chooses between two actions: Hit/Stand

The Basic Strategy Agent strictly follows the static Blackjack decision table from the paper.

B.2. **One Up-Card.** The following table give the basic strategy for the one up-card variation played using one deck.

Player \ Dealer	2	3	4	5	6	7	8	9	10	11
Hard 4 - 12	H	H	H	H	H	H	H	H	H	H
Hard 13	H	H	H	H	H*	H	H	H	H	H
Hard 14	S*	H*	H*	S	S	H	H	H	H	H
Hard 15	S	S	S	S	S	H	H	H	H	H
Hard 16	S	S	S	S	S	H	H	H*	H*	S
Hard 17 - 20	S	S	S	S	S	S	S	S	S	S
Soft 12 - 16	H	H	H	H	H	H	H	H	H	H
Soft 17	S	S	S	H	H	H	H	H	H	H
Soft 18	S	S	S	S	S	S	S	H	S	S
Soft 18 - 20	S	S	S	S	S	S	S	S	S	S

TABLE 7. Basic strategy for the one up-card variation (single deck). The * signifies that the optimal decision depends on the specific layout given this hand total.

Baseline-Random Forest Classifier

The RFC Agent is a supervised learning model trained using Random Forest Classifier.

Extracted from game state for each action: player_sum, dealer_up, is_soft, num_cards

The goal is to learn the best mapping from state features to player action based on historical data.

Training Process:

- For each game: iterate through actions_taken sequence.
- For each action:
  - Record current state features and action.
  - Simulate "drawing a card" by appending a placeholder (0) when hitting.

Model Output: Given the state features, predict whether to Hit or Stand

Baseline-NeuroEvolution of Augmenting Topologies (NEAT)

The NEAT Agent uses NeuroEvolution of Augmenting Topologies (NEAT) to evolve a neural network that directly maps state features to actions.

Neuroevolution Workflow:

- Input Features
  - Normalized values: player_sum / 21, dealer_card / 10, is_soft, num_cards / 5
- Network Evolution: NEAT starts with simple networks and evolves both weights and topologies over generations.
- Fitness Evaluation:
  - Each genome (network) is evaluated by running 1000 games.
  - The fitness score is based on the win rate.
- Decision Logic:
  - For a given game state, the evolved neural network predicts the probability of Hit vs. Stand.
  - The agent selects the action with the higher output value.

Main Approach-AI Agent and Training Process

- Random Agent

  At each decision step, randomly selects Hit or Stand.

  No learning, no logic.

- Basic Strategy Agent

  Follows Blackjack basic strategy table, derived from the reference paper.

  Rules include:

Player \ Dealer	2	3	4	5	6	7	8	9	10	11
Hard 4 - 12	H	H	H	H	H	H	H	H	H	H
Hard 13	H	H	H	H	H*	H	H	H	H	H
Hard 14	S*	H*	H*	S	S	H	H	H	H	H
Hard 15	S	S	S	S	S	H	H	H	H	H
Hard 16	S	S	S	S	S	H	H	H*	H*	S
Hard 17 - 20	S	S	S	S	S	S	S	S	S	S
Soft 12 - 16	H	H	H	H	H	H	H	H	H	H
Soft 17	S	S	S	H	H	H	H	H	H	H
Soft 18	S	S	S	S	S	S	S	H	S	S
Soft 18 - 20	S	S	S	S	S	S	S	S	S	S

- Purely rule-based, no learning.

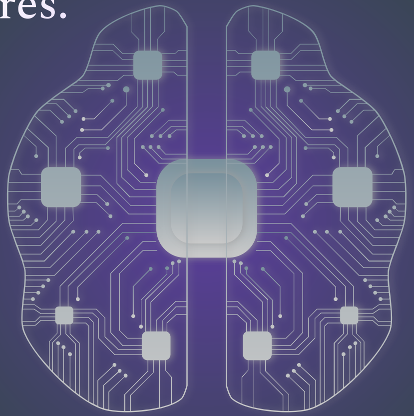
Main Approach-AI Agent and Training Process

RFC Agent (Random Forest Classifier) Supervised learning model trained with Random Forest Classifier.

Training Process:

1. Load pre-simulated game data (blackjack_simulator.csv).
2. Extract state-action pairs for each decision step: Input features: player_sum, dealer_card, is_soft, num_cards. Target label: action (0=Hit, 1=Stand).
3. Train RandomForestClassifier to predict actions based on state features.

Inference Logic:
Given a new game state, the model predicts whether the player should Hit or Stand.



Main Approach-AI Agent and Training Process

NEAT Agent (NeuroEvolution of Augmenting Topologies)
Uses NEAT to evolve a neural network for Blackjack decision-making.

Training Process:
1. Input features: player_sum / 21, dealer_card / 10, is_soft, num_cards / 5
2. NEAT evolves neural network topology and weights over generations.
3. Fitness evaluation:
   Run 1000 games per genome.
   Fitness = cumulative reward (based on win/loss outcomes).

Inference Logic: The neural network outputs a probability:
If output > 0.5, choose Hit; else Stand.

Main Approach-Graphical User Interface (GUI) (blackjack_gui.py)

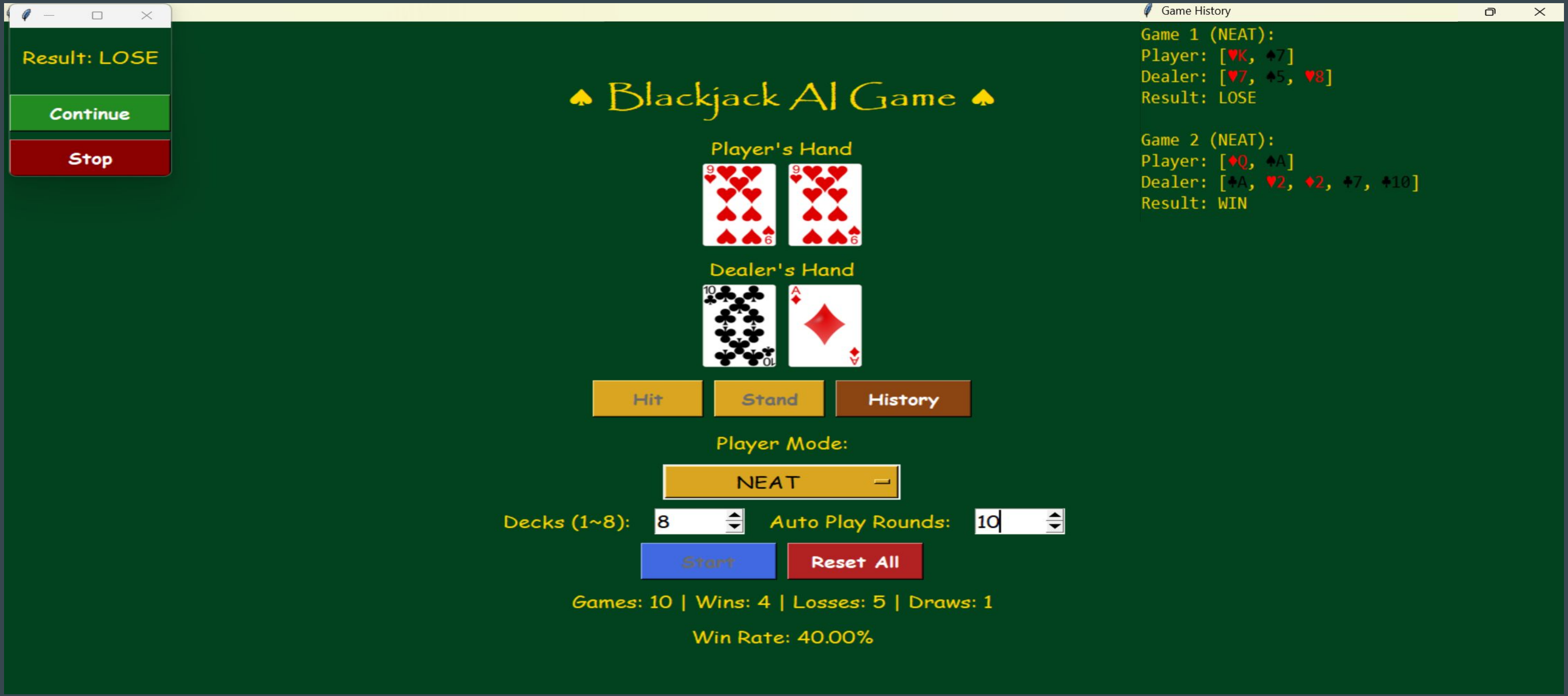
GUI Features

Component	Description
Mode Selection	Dropdown menu: Manual, Random, Basic, RFC, NEAT
Manual Play Buttons	Hit, Stand
AI Simulation Control	Auto-run games in batch, update results without dialog boxes
Card Display	Visual display of cards using **Vector Playing Cards** images
Game Status Display	Shows player/dealer hands, scores, wins, losses, draws
Result Dialog (Manual)	After each manual game, popup shows result and allows new game

Integration Logic:
GUI interacts with BlackjackEnv to get game state and render visuals.
When AI mode is selected, the GUI runs simulations automatically.
The GUI updating statistics and history in real-time.
Manual mode supports full gameplay with user decisions.

Each agent is tested over 100 trials, each trial simulating 1000 games.
Metrics collected: Win rate. Results visualized as line plots in compare.png

Main Approach-Graphical User Interface (GUI) (blackjack_gui.py)



Expectimax Agent

Objective:

Compare expected outcomes of 'Hit' and 'Stand' to decide optimal move.

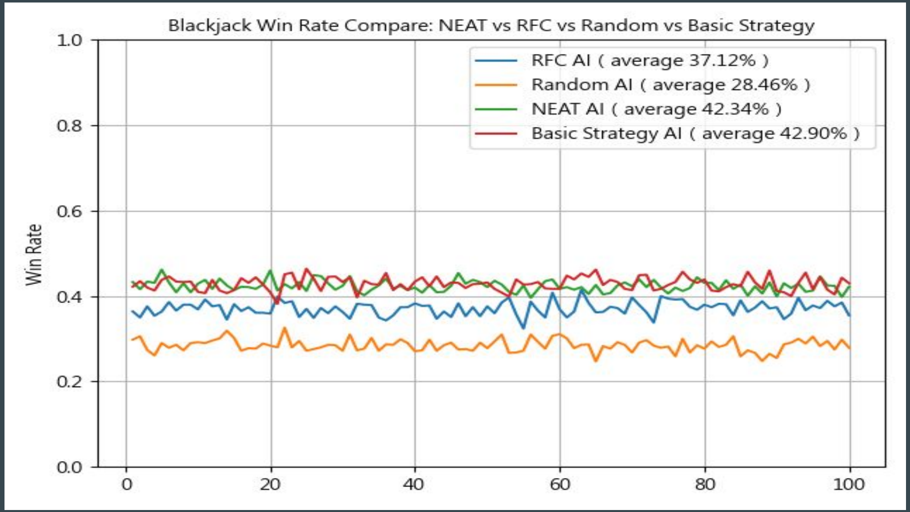
Approach:

- 1st Loop (Stand):
  - Loop through all dealer possible values (2–11).
  - If dealer < 17, assume dealer draws to 17.
  - Compute difference:
    Agent Value - Dealer Value
  - Accumulate differences to get Stand EV.
- 2nd Loop (Hit):
  - Agent draws all possible cards (excluding dealer's card).
  - Add drawn card to agent's hand.
  - Re-run 1st loop logic for each new agent value.
  - Accumulate differences to get Hit EV.
- Final Decision:
  - Compare averages:
    Hit EV vs Stand EV
  - Choose the action with the higher expected value.

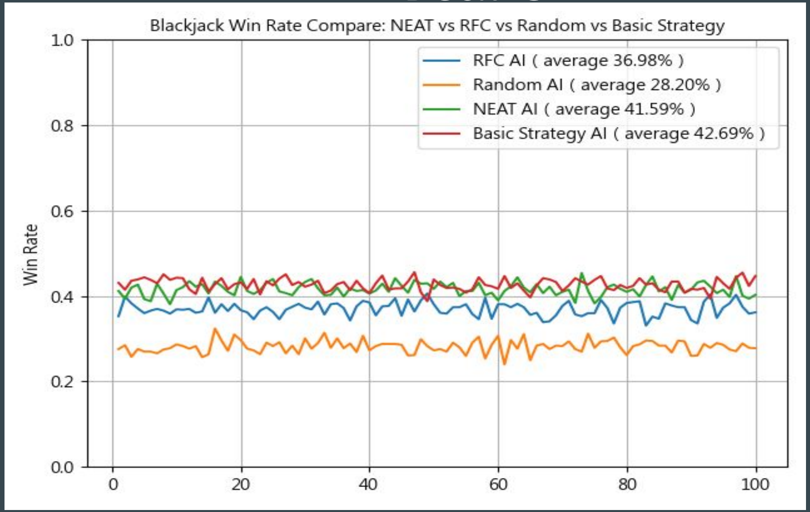
Result and Analysis

AI Agent

Deck=1



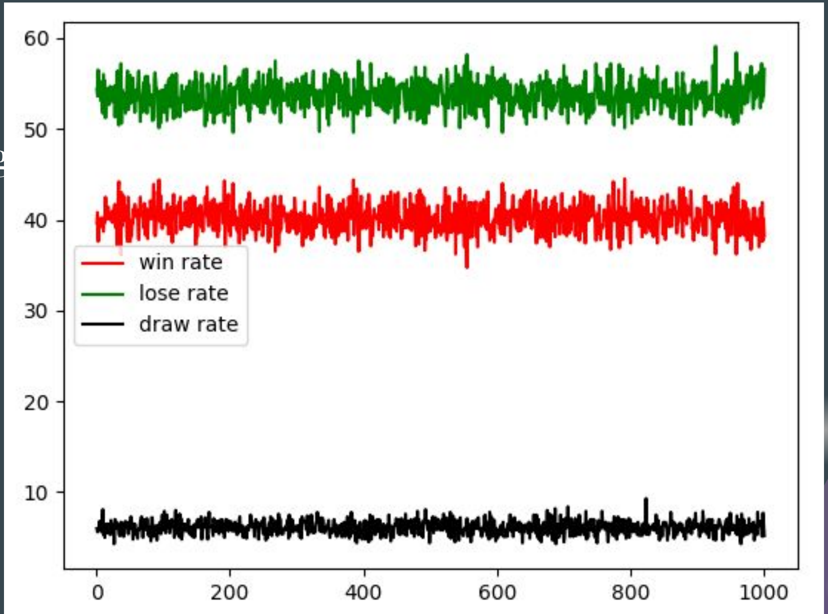
Deck=8



Expectimax Agent

Win Rate:

- Approximately 40% win rate
- Notable, considering Blackjack's high uncertainty and the agent's lack of advanced features (e.g., card counting



Stability:

- Variance < 1.5% across multiple runs
- Indicates a consistently performing agent, not overly affected by random fluctuations

Interpretation:

- Expectimax performs well despite the game’s probabilistic nature
- Suggests strong decision-making logic in terms of evaluating *Hit* vs *Stand*

The Better Model?

After testing, the result is:
When comparing winning rate:
Basic strategy > NEAT > Expectimax > RFC > Random

Thank You!