King Abdullah University of Science and Technology

KAUST Academy

Final Report

Name: Abdulmalik Almadhi*, Saeed Bawazeer*, Alyahya Akkam*

E-mail: AbdulmalikMishari@gmail.com, sab3495@gmail.com, alyahyakkam@gmail.com

Table of Contents

Introduction 3
Requirements and Design.....3
  Game Logic 3
  Environment 3
Implementation.....4
Conclusion.....5

Introduction

Baloot, a widely cherished trick-taking card game, holds significant cultural importance across the Middle East, especially in Saudi Arabia. Traditionally played by four participants split into two teams, the game's objective centers around outscoring the opposing team by successfully navigating a series of contracts or modes. Despite its regional popularity, the strategic depth and complexity of Baloot pose considerable challenges for artificial intelligence (AI) and reinforcement learning (RL) applications.

In this project, we focused on "Sun," a pivotal subsection of Baloot, to develop a scalable multi-agent environment as a proof of concept. By narrowing our scope to this specific aspect of the game, we aimed to create an RL agent—SunAgent—capable of mastering Sun's intricate strategies and gameplay dynamics. This foundational work not only demonstrates the feasibility of AI mastery over Sun but also paves the way for the future development of an agent equipped to tackle the full breadth of Baloot.

Requirements and Design

Game Logic

The game logic of 'Sun' involves four players split into two teams, with teammates positioned opposite each other. The game is played with a deck of 32 cards, divided evenly among the players, over eight rounds. In each round, player 0 begins by playing an initial card. Subsequently, all other players must play a card of the same suit if possible; if not, any played card will not count towards winning the round. The player who plays the strongest card of the same suit as the initial one collects all the cards and their corresponding points for that round. This player then starts the next round. The process repeats for the remaining seven rounds. The game has a total of 130 points, where the sum of all cards is 120 points, and winning the last round will reward you an extra 10 points. The card values are as follows:

- Seven, Eight, and Nine are worth 0 points
- Jack is worth 2 points
- Queen is worth 3 points
- King is worth 4 points
- Ten is worth 10 points
- Ace is worth 11 points

Environment

As for the environment, we determined that it must support multiple agents and be coded with scalability in mind, particularly when we add more subsections of the game Baloot. The interactions between agents and the environment should be sequential, with rewards distributed after each round and a winner declared at the game's end. The observations given to the agents should be partial, including only their hand, the card played in the current round, and a history of all cards played. We believe this promotes strategic thinking by providing past cards. Furthermore, we will add an action mask to the observations to prevent any illegal moves, such as playing a card of a different suit when a card of the same suit is available. The action space will consist only of playable cards. For rendering, we aim to display all cards,

how they were played by each player, and a score that is updated after each round.

Training & Model

For training, we decided to use a single model to represent all current agents, simplifying the training process. We chose the MaskedPPO model because it is the only one available that supports action masking. While we could have developed a custom model from scratch, time constraints prevented us from doing so.

Implementation

The project was implemented in Python 3.11. We used Pygame library for rendering the game, Petting Zoo & Gymnasium for creating the environment, and Stable Baselines3 for a model to train on. This is the [github repo] if you want any specifics.

Game Logic

We implemented the Sun game logic within its own class, using a dataclass for the cards. Upon instantiation of this class, a new game is created by shuffling a deck of 32 cards and dealing eight cards to each player. Player 0 initiates the first round. Each player is then provided with a subset of possible moves from his hand, determined by a function named possible_moves, which indicates the cards legally playable during that round. Any card played is removed from the player's hand and added to a played_cards list, formatted as a list of tuples (player, card_played). The end of a round is checked when played_cards reaches four entries. At this point, we tally the points, determine the round winner who will start the next round, and update the rounds_played counter. This process repeats until the game concludes, indicated by the rounds_played counter reaching eight.

Rendering

We then developed a Render class, utilizing Pygame to visualize and record the gameplay. This class accepts a dictionary containing initial player hands, the starting player for each round, the sequence of cards played in each round, and the scores per round. With these parameters, we initialize a Pygame display and a camera setup to record each frame. Initially, we display all player hands, shift played cards to the center of the screen, and update the score after each round. The recording captures one frame per second, resulting in a total of 33 frames. You can find a snippet of a rendered video in the drive link.

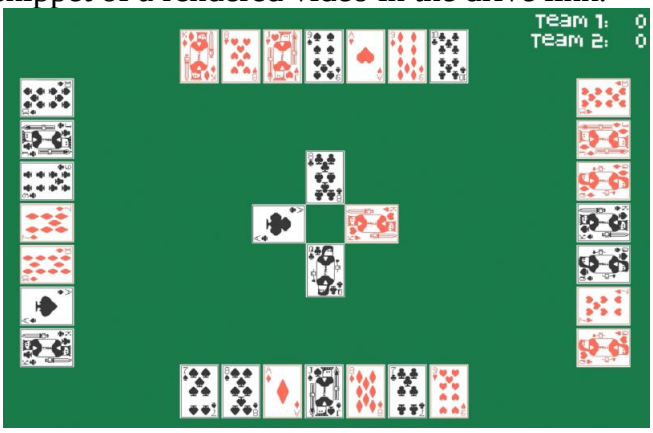


Figure 1 SunAgent Render Sample

Environment
We utilize the Agent Environment Cycle (AEC) framework for our environment, allowing agents to act sequentially with updated observations and rewards prior to each action. The observation space includes a dictionary with observation and action_mask as keys. Observation is represented as a MultiBinary grid of 5 rows and 32 columns; a 1 indicates the presence of a card, while a 0 indicates its absence. The first row displays the player's hand, rows two to four show cards played by other agents in the current round, and the final row tracks the history of all cards played in the game. The action_mask, a numpy ndarray with 1 row and 32 columns, uses 1 to show playable cards and 0 for others. Our action space, a Discrete of 32, corresponds to the indices of cards, facilitating correct legal move predictions when combined with the action_mask. We've implemented reward engineering: winners receive points equal to the card points collected per round, while losers get negative rewards equivalent to the card points they played. The game's final reward is calculated by subtracting the cumulative squared negative rewards from the player's total score, encouraging agents to maximize rewards when confident of winning and to minimize losses otherwise.

Training
For training, we wrap our environment with the SB3ActionMaskWrapper, sourced from the PettingZoo library. This wrapper makes our environment compatible with Stable Baselines3 models. We utilized the MaskedPPO model and trained it 10 million steps within the environment. The results are in the following drive link:

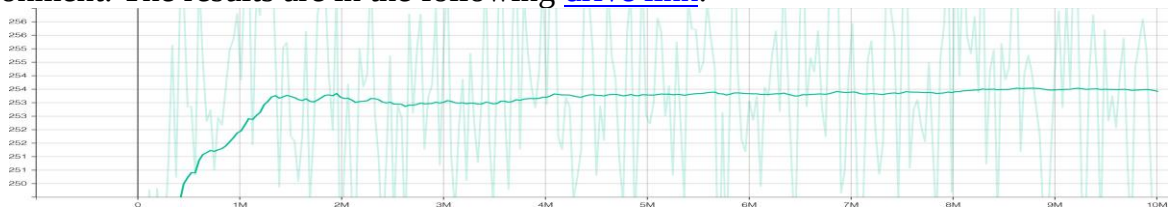


Figure 2 SunAgent Reward Plot

Conclusion
Our experiments with SunAgent have yielded promising results, demonstrating the potential of reinforcement learning in mastering the game of Sun. However, there remains significant room for improvement. Due to time constraints, we relied on external libraries such as Stable Baselines3 for the training process. While this approach enabled us to advance our project within the available timeframe, it also imposed certain limitations, particularly regarding the choice of models. For instance, we intended to implement a recurrent model to allow the agent to retain memory of previous game states—a crucial feature for optimizing decision-making—but were unable to do so due to time restrictions.

Despite these challenges, our project made substantial progress. The multi-agent environment we developed serves as a robust foundation for future research and development in this area. We believe that our work will be valuable for anyone seeking to explore or improve upon AI-driven strategies in the game of Baloot, and we are optimistic about the future possibilities this project unlocks.