Alec Gironda and Luke Lorenz
CSCI 0311 Final Project Abstract
12/11/22

Background:

Both classic and modern board games are areas of interest in the field of AI research (Heyden, 2009). These games present a variety of challenges such as incomplete information, extensive search spaces, branching factors, varying numbers of players, and stochasticity (De Mesentier Silva et al., 2017). One such game is Camel Up. In this game, players seek to maximize their earnings by betting on camels as they race around a track. See the *Gameplay* section of the accompanying poster for the game's rules. While a probabilistic AI for Camel Up has been developed (Baron, 2018), we sought to combine an expectimax algorithm with reinforcement learning to enhance overall play. Inspiration for this implementation came from previous research on stochastic board games, such as Risk, which have been optimized using reinforcement learning (Blomqvist, 2020). To meet the scope and time constraints of our project, we limited the rules to a two-player version without second-place payouts or oasis tiles.

Methods:

Our initial approach focused solely on reinforcement learning to find the optimal policy at any game state. We built a RandomPlayer (Table 1) and simulated 1000 games where two RandomPlayers played one another. During those games, we stored key value pairs of the game state nodes visited and the money earned at the end of the game. The stored nodes were used as training for a neural network model and we used money earned at the end of the game as labels. Our network consisted of an input layer, 1 dense layer of 100 units, and a single node output layer, and was implemented with the Keras framework and a Tensorflow backend. Utilizing this strategy, we were able to develop an agent that chose its actions based on the child game state with the highest predicted reward according to the neural network.

After limited success with a strictly reinforcement learning based agent, we sought to incorporate an expectimax algorithm into our agent's model. Inspired by GitHub user trbarron's Sir_Humpfree_Bogart AI, we built an agent that calculated the expected values (EV) of leg bets (see *Gameplay* on the accompanying poster for a description of leg bets), and made moves based on the highest EVs across the bets. We calculated these EVs by considering all permutations of dice that had yet to be rolled and all possible numbers that could be rolled on those (3 sided) dice. With n dice left, there are $n! \cdot 3^n$ possible ways to roll numbers on the dice. We simulated all of the camels' movement for each of these possibilities, and calculated each leg bet's expected value with the following equation:

$$EV = \frac{(camel\ wins)(leg\ bet\ payout)}{n! \cdot 3^n}$$

Our final agent, SmartPlayer, chooses to take the leg bet with the highest EV greater than 1, and if there are no bets with an EV greater than 1, SmartPlayer rolls or makes an overall winner or loser bet according to the neural network trained through simulated games. We set this threshold at 1 because the EV of rolling is 1 (a player is paid 1 coin for rolling), and we believe that leg bets are the strongest moves (see *Results*). Instead of training our final agent's neural network on simulated games between two RandomPlayers as we had in our initial model, we trained SmartPlayer on 100 games between MaxPlayer and RandomPlayer (see Table 1 for player descriptions).

To analyze the success of SmartPlayer, we recorded the wins, losses, and ties of SmartPlayer, MaxPlayer, and RandomPlayer in simulated games against a RandomPlayer. Finally, we leveraged pygame to visualize SmartPlayer's situational decision making.

Results:

We simulated 100 games of RandomPlayer against another RandomPlayer. The first player won 49 games, and the second won 48 games, and they tied 3 times. Next, we simulated 100 games of MaxPlayer against RandomPlayer. MaxPlayer won 81 games, RandomPlayer won 18 games, and there was 1 tie. Finally, we simulated 100 games of SmartPlayer against RandomPlayer, and SmartPlayer won 96 games, RandomPlayer won 3 games, and there was 1 tie. Based on these outcomes, SmartPlayer outperformed MaxPlayer, and MaxPlayer outperformed RandomPlayer. These results are displayed in Figure 3 on the accompanying poster.

We found SmartPlayer was successful because it prioritized leg bets. Prioritizing leg bets is advantageous, as it allows a player to quickly accumulate large sums of money. A player can win up to 10 coins each leg by correctly betting on the winning camel in the leg over multiple legs in a race. For comparison, a player could earn up to 8 coins for an overall race winner or loser bet, and a player only earns 1 coin for rolling a die. MaxPlayer also prioritized leg bets, hence both agents' dominance over RandomPlayer. SmartPlayer's proclivity for leg bets is displayed in Figure 1 on the accompanying poster.

Additionally, SmartPlayer's neural network preferred early betting on winners and losers over rolling. It is advantageous to bet on a race winner or loser earlier in the game because the potential payout is 8 coins if you are the first player to make an overall bet, and only 5 coins if you are the second to make a bet. Assuming each camel has about a ⅕ chance of winning at the very start, the EV of an overall bet that pays 8 coins is greater than a die roll with an EV of 1 ($8 \cdot \frac{1}{5} = \frac{8}{5}$ and $\frac{8}{5} > 1$). We attribute SmartPlayer's dominance over MaxPlayer to its prioritization of these early overall bets.

A strategic flaw in our AI is that it takes bets to maximize payout when it is already winning by a large margin. This gives the other player opportunities to win leg bets and close that margin. Instead, our agent should choose to roll to try to end the game quickly. SmartPlayer's neural network is designed to maximize money earned, so in order to resolve this flaw, we would need to train it to maximize games won instead. We would also need to deprioritize leg betting and allow the neural network to choose whether or not to leg bet, unlike SmartPlayer's neural network which only chooses between rolling and taking overall bets.

Conclusions:

This project demonstrates the power of reinforcement learning and expectimax for optimizing board game strategy. However, we believe that we only began to scratch the surface of what is possible in the realm of Camel Up. Training SmartPlayer's neural network on more simulated games, extending to 4 players, adding second place payouts, adding oasis tiles, and addressing the strategic flaw mentioned in *Results* are all ways we could improve our agent. While SmartPlayer's neural network currently only chooses between taking overall race bets and rolling, we hope to expand the agent's decision making capabilities to some of the extensions mentioned above. We view this as a stepping-stone for advanced machine learning techniques for even more complex board games in future research.

Supplementary Figure:

Player	**Function**
RandomPlayer	Agent who takes completely random moves during each turn.
MaxPlayer	Agent who will choose to take highest Expected Value (EV) leg bet on their turn if there exists a leg bet with EV > 1. Otherwise, there is a 10% chance they will randomly bet on an overall winner or loser, or roll, and there is a 90% chance that they will roll.
SmartPlayer	Requires a neural network as an input. This agent will choose to take the highest EV leg bet on their turn if there exists a leg bet with EV > 1. Otherwise, the player will use its pre-trained neural network model to predict which child will have the greatest money output and choose that child.

Table 1. Classes of players used in the project.

References:

Baron, Tyler. “Sir Humpfree Bogart” (2018).
https://github.com/trbarron/Camel_Up_Cup_2K18/blob/master/Sir_Humpfree_Bogart.py

Blomqvist, E. (2020). Playing the Game of Risk with an AlphaZero Agent (Dissertation).

“Camel Up Rules”. F.G. Bradley’s, https://www.fgbradleys.com/game_rules.asp

De Mesentier Silva, Fernando & Lee, Scott & Togelius, Julian & Nealen, Andy. (2017).
AI-based playtesting of contemporary board games. 1-10. 10.1145/3102071.3102105.

Heyden, Cathleen. (2009). Implementing a Computer Player for Carcassonne.

Linderman, Michael. (2022). Middlebury College Artificial Intelligence Presentation
“Introduction to AI”