Using Approximate Q Learning to Play an Incomplete Information Card Game

McKenna Quam

Northeastern University, Boston, MA
quam.m@northeastern.edu

Abstract

Computer agents sometimes can struggle with games of incomplete information as without knowledge of all variables the agents have less information with which to "solve" for the optimal move. This report investigates using approximate Q learning to play a simple card game called skull; skull has players place cards (either a flower or skull) and take turns betting how many flower cards they can turn over without turning a up skull card. Once a player has successfully completed 2 of their bets they win. This report explores 2 feature sets which fail to win against simpler agents and 1 which succeeds, finding that in generalized reinforcement learning it is more important to pick informative features rather than a large number of features.

Link to project repository here: https://github.com/mckennaquam/Skull-RL-CS-4100#

Introduction

When defining a game state, environments with incomplete information present an interesting issue as how can an agent define an environment which they are not allowed complete information on? This is the motivation behind investigating a card game which has incomplete information, and why I wanted to attempt a solution using approximate Q learning which does not require all information about the state but instead requires the programmer to define what is important about the state via features.

Skull is a game played in rounds with four phases: an adding cards phase, an add or raise phase, a raise phase, and the attempt phase. The actions a player can take are dependent on the game phase and restricted by rules depending on certain states.

All players start the game with 3 flower cards and 1 skull card. A round starts with each player placing at least one card face down in front of them (cards played are in a player's "stack"), this is the adding cards phase. Once each player played 1 card to their stack they take turns either placing another card or starting the bet, this is the add or raise phase.

A player's bet represents how many flower cards they can turn face up from the top of players' stacks without turning up a skull card, starting with their own stack. It is important that players must turn up their own stack before others as that is the bluffing element of the game. A player placing a bet ("I can turn 3 cards…") presumably means their stack does not contain a skull card; however they may be placing a bet in the hopes that another player places a raises and turns their skull card. Once betting has started no more cards can be placed and each player then has the options to raise (bet one higher more than the previous player) or pass, removing themselves from the round.

The raise phase ends once all but one player has passed or a player raises to the number of cards played (the max bet). Then the attempt phase beings, the player who had the highest bet must make the attempt, first flipping their own stack, then flipping over the top cards of other players' stacks until they have flipped the number of flower cards equal to their bet, or they flip a skull card. If a player flips an opponent's skull card, they lose their attempt and have a random card removed from their hand. However, if a player flips a skull card from their own stack they may choose which card is removed [1]. If the player does not flip a skull card, they win their attempt and earn one point. First player to two points wins the game. The game may also end if only one player has cards left, meaning all other players lost four attempts. These two end conditions are referred to as win via score or win via elimination. After the attempt is finished a new round starts with players regaining all their played cards to their hands. The player who made the attempt is the starting player of the next round regardless if they win or lose.

I will now define the variables of which make up the state for each game phase, as well as the actions corresponding to that state, and if there are any restrictions place on those actions. As this game has incomplete information the states will be from the perspective of a theoretical RL agent. The environment has full knowledge of the state.

State Varibles for All Phases

- The current game phase (adding cards, add or raise, raise, and attempt)
- Each players score (can be 0, 1, or 2 victory)

[1] Players will usually remove a flower card unless they only have a flower and skull card left at which point they will remove the skull card, this was a logic programmed into the skull environment used in this paper

- Which players are will within the game (players can be removed from the game by losing all of their cards)
- The cards in hand (start with 3 flower cards and 1 skull card)
- The cards in the player's stack
- The number of cards in opponent i's hand (player is not aware of the types just the amount)
- The number of cards opponent i's stack (player is not aware of the types just the amount)
- The total number of cards played (the sum of the length of all stacks)

Adding Cards

Actions - Place Skull Card, Place Flower Card
Restrictions on actions - cannot place a card which is not contained in their hand. If player lost their skull card in previous round they must place a flower.

Adding or Raise

Actions - Place Skull Card, Place Flower Card, Raise
Restrictions on actions - Player cannot play a card type which is not contained in their hand. If no cards are left in hand player must raise.

Raise

Actions - Raise, Pass
Extra state elements

- The current max bet (from 1-the number of cards played)
- The player in attempt (player who raised to the max bet)
- which players have passed (if passed player is out of the round)

Restrictions on actions - No restrictions on actions. If a player raises to the max bet the attempt game phase will automatically trigger therefore a player cannot raise past the max.

Attempt

Actions - Turn top card of player i's stack
Extra state elements

- The current max bet
- The player in attempt (player who raised to the max bet)
- The number of flower cards turned up in this attempt thus far.

Restrictions on actions - Any player who is not in the attempt is skipped. Player in attempt must turn any cards on their own stack first.

I programmed my own environment for skull in python. The class skull_env was interacted with by different agents (detailed in the background and project description sections) through a play_action function. Each agent decided on what action to take from their own choose action action function, which could utilize any state variables from the get_player_knowledge function in the environment.

I choose to use approximate Q learning for this as the state space for skull is very large. While the game itself isn't complex there are lots of variables required to create the each state and each combination of state variables would require its own line, and then for each combination of states there needs to be a line for each possible action. Tabular (exact) Q learning works via taking in samples and then adjusting the Q value of the state action pair based on the reward of the sample and the Q value of the next state.

$$sample \leftarrow R(s, a, s') + \gamma * \max_{a'}(Q(s', a'))$$
$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(sample)$$

In the update s is the beginning state, a is the action taken, s' is the state post action, a' is the actions available from s', and R(s, a, s') is the reward from the action. Additionally γ is a decay rate and α is a learning rate. This process can take a long time as each action at each state needs to be taken multiple times for the agent to learn its converged Q value.

Rather than assigning each state action pair its own Q value we can represent the state as a series of features from which we calculate the approximate Q value as

$$Q(s, a) \leftarrow w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a)$$

Then based on the difference between our sample and our expected approximate Q value each weight is updated according to the corresponding feature present within the state before the action was taken.

$$difference \leftarrow [R(s, a, s') + \gamma * \max_{a'} Q(s', a')] - Q(s, a)$$
$$w_i \leftarrow w_i + \alpha[difference] f_i(s, a)$$

Features are created from state variables then expanded into state action features by having an entry for each possible action where the feature was equal to f_i if evaluating for action i's value and 0 if evaluating for any other action [2].

Training will implement ϵ greedy method for choosing actions where the RL agent will start by choosing a random actions and then after each training episode the ϵ (likelihood of choosing a random move) will be decreased in favor of choosing optimal moves.

For this project 2 approximate Q agents were trained. One to optimize for turning flower cards in the attempt phase and then another for creating optimal behavior in the raise phase to work in conjunction with the algorithm for optimizing the attempt phase. The attempt phase is single player portion of the game with an easily quantifiable goal whereas the raise phase is multi-player and the terminal states which have more ambiguous reward value for the player.

Related Work

There has been research into the game of skull using reinforcement learning by other computer scientists although to my knowledge there has not been work done using approximate Q learning or using more than 2 players within the game.

[2]Example: Consider a state with 2 features a, b and 2 actions 1, 2. The feature vector would be [a, b , 0, 0] when evaluating action 1 and [0, 0, a, b] when evaluating action 2.

Previous reinforcement learning work around skull has focused on the bluffing aspect of the game. Using a tabular Q learning method resulted in a 23% bluff (betting when a skull is placed) rate when the agent was given perfect knowledge of the environment and 12% bluff rate when its information was restricted [1]. The methodology in this study is strange as giving the agent perfect information of the environment is not in line with the rules of skull. Players do not know if their opponent has played a skull card which is what makes the attempt phase of the game difficult. Another study using tabular Q learning on just the raise portion of the game found that in 2 player games bluffing was a successful strategy only when the top card of a players stack is a skull card [2].

Another researcher used a deep Q learning network to train 2 agents to play skull against each other. A deep Q learning network utilizes a neural network with hidden layers to train the reinforcement learning algorithm. They also introduced a technique called scaffolding where the agent learns the attempt phase of the game before learning the previous phases. They were able to successfully train an agent to play skull using both a scaffolding DQN and a vanilla DQN but found that the scaffolded one performed better [3]. The scaffolding technique inspired the creation of 2 agents to better understand how approximate Q learning applied to each phase of the game.

I chose to use approximate Q learning as I could not find an example of generalized or approximate Q learning being applied to skull as well I was interested in how a reinforcement learning method simpler than a DQN (which uses machine learning) would perform in a game which has both single and multi-player phases.

The choice of these studies to implement a 2 player games of skull is interesting as the publisher's rules state that the game is for 3-6 players [4]. I am not sure how they adapted the rules to fit a 2 player environment and if their accounting for 2 players differs from paper to paper but I have chose to implement a 4 player game of skull (though my environment should be extendable for anywhere from 3-6 players) as that is most realistic to how humans play skull.

Background

Before beginning the process of creating a reinforcement learning agent to first there needed to be interesting opponents for them to play. The goal for the opponent agents was to have them behave in a similar manner to how humans play skull together, as well in the process of attempt to create a "smart" agent, other agents were developed which had non-optimal play styles which were informative to the process of understanding skull strategy for defining the reward parameters of the RL agent.

Random Agent

The random agent chooses a random action for the given game state. The random agent received 25% (equal share) of the wins when playing against random agents in a 4 player game. Winning equal of games is expected when all agents are behaving randomly and is a good introduction to the idea of win share. In this paper having a positive win share means the agent is winning more games than expected (25%) and is out baseline goal when devolving the RL agent. The random agent was used for comparison to other simple agents.

Nice Agent

The nice agent is trying to win via score and therefore does not put their skull card down unless forced. This agent also always raises to give them the best chance of being the agent who performs the attempt this round. The nice agent received 46% (positive share) of the wins against 3 random agents in a 4 player game, and received 0 wins by elimination.

Mean Agent

The mean agent is trying to win via elimination and therefore does not does not put a flower card down unless forced. This agent also always passes to ensure they do not have to turn over their own skull card. The mean agent received just 0.7% (very negative share) of wins against 3 random agents in a 4 player game and received 0 wins by score.

While the nice and mean agents are not in line with how humans play skull, they are useful in thinking about how to design the rewards structure for our approximate Q learning model. The mean agent's small share of the wins shows how winning via elimination while technically possible is difficult to achieve in a competitive multi-agent system. If an agent wants to win via score they need to score 2 points, if an agent wants to win via elimination they need their opponents to lose 12 (in the case of 3 opponents) points. It is much faster to win via score than to win via elimination so most games will be won via score. Therefore in the rewards for Q learning winning attempts will carry a heavier reward than opponents being in attempts when the Q agent has their skull card down.

Smart Agent v1

The random, mean, and nice agents are deterministic which is not how human beings play skull. Rather when playing skull some people are more likely to place their skull card than others, and some people are more likely to raise in the betting phase than others. Therefore to get an "human" simulation it is useful to create an agent with adjustable parameters representing their willingness to place a skull card and their willingness to raise a bet which I will call the "smart" agent.

When a smart agent is asked to play a card in the first add or adding/raise phase a random number is generated from 0-1 and if the number is greater than the user defined "skull threshold" then a skull card is placed (this assumes they have a choice or flower or skull card). A skull threshold of 0 means a skull card is always placed and a threshold of 1 means it is never placed.

For the add/raise phase the agent will place a card or start the bet at equal probability.

For the raise phase if the current maximum bet is less than half the cards on the table the smart agent will always raise. This choice was made due to behavior observed in real skull

games. In a round with 6 cards in play players will not let bets of 1/2/3 go to attempt either because they sincerely believe they can turn more flower cards or because they are bluffing believing an opponent will raise to a higher bet, and be more likely to turn a skull card. If the raise is above half the cards in play a random number from 0-1 is generated and if it is greater than user defined "raise threshold" then the agent raises the bet. A raise threshold of 0 means an agent always raises and a raise threshold of 1 means an agent always passes.

Smart Agent v2

The Smart agent performed much better than the previous deterministic agents, however in simulating against random agents the methodology behind a static skull threshold proved incomplete for simulating a player who is "more likely to play their skull card". Consider a skull threshold of 0.3 on an agents first placement they are more likely to place their skull card than if they picked randomly ($0.3 > 1$ in 4), but if they place a flower card on their next placement they will be less likely than if they picked randomly ($0.3 <$ 1 in 3). For smart agent v2 rather than having a static skull threshold instead each time the agent choose to place a card the probability of placing the skull card was determined by a user defined skull factor which then calculated the probability of placing a skull card as 1 + skull factor / number of cards in hand + skull factor. This methodology is borrowed from Laplace smoothing, the agent is observing more skull cards in their hand than they actually have. Following the same example from before, an agent with a skull factor of 1 would have a 0.4 (2 out of 5) chance of placing a skull card and which then increases to a 0.5 (2 out of 4) chance on the second placement choice.

In simulated trials (each trial containing 10,000 games) for both smart agents against 3 random agents Smart v1 had best results at skull threshold of 0 and a raise threshold of 0.55 with a 56% win share. For Smart v2 the best results were at skull factor of -0.5 and a raise threshold of 0.77 with a 51% win share. Both agents performed better than the mean and nice agents. These results indicate that the optimal strategy for skull is to never place your skull card and sometimes raise when the bet is risky. However, these are the best results against random agents, if these agents were to play against humans their opponents would quickly realize they can score free points as the agent never places a skull. What these results show is that these probabilistic agents preform better than the simpler deterministic agents, and that adjustments in how likely an agent is to play a skull or raise will affect their win share. Full results of the Smart v2 trials shown in figure 1.

For my the RL agent training I used 3 Smart v2 agents which had these settings: 0 skull factor & 0.4 raise threshold (plays cards randomly, is more likely to raise), 0 skull factor & 0.6 raise threshold (plays cards randomly, less likely to raise), and 2 skull factor & 0.6 raise threshold (more likely to play skull and less likely to raise). This is a balance of different play styles for the RL agent to play against and therefore should produce a robust agent.

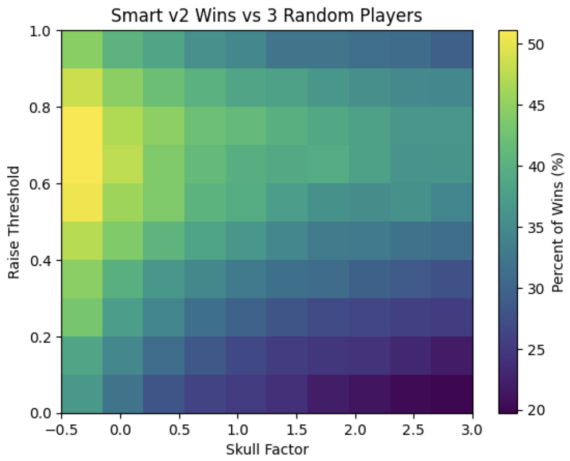

Figure 1: Any percent of wins above 25% represents a positive win share, performing better than expected

Project Description

I created 2 versions of my Q approximate agent. One which only learned how to play the attempt phase of skull: the goal of the agent was to turn the most flower cards. Building off the algorithm of the first agent, the second agent learned how to play the attempt and raise phases of skull: the goal of the agent was to have the highest win share amoung the agents it competed against.

Q Learning Agent V1 - Attempt only

The first version was a version of the "nice" agent (never play skull card and always raise) when choosing an action in the attempt phase used Q approximate learning to determine which pile to turn. For each action of turning player i's stack there were 2 features: the number of cards in player i's hand, the number of cards in player i's stack. They were equal to the value of the features when evaluating action turn i and 0 otherwise [3].

The parameters of the attempt phase Approximate Q algorithm are as follows

- ϵ started at 1 (always random) and decreased by 1 / (1 + number of training episodes) after each episode
- γ (q value discount) was set 0.9
- α (learning rate) was set at 0.5

The rewards for attempt phase are as follows:

- non-terminal state
  - Turn flower, 10
- terminal state
  - Win challenge, 100
  - Self lose challenge, -1000
  - Lose challenge, -100

[3] In a 4 player game evaluating the action turn 2 the feature vector would be [0, 0, 0, 0, # of cards in player 2 hand, # of cards in player 2 pile, 0, 0].

Algorithm 1: Simulate Games with Approx Q Leaning in Attempt

Input: Array of players with a Q agent, Skull Env
Parameter: number of training eps, γ decay rate, α learning rate, rewards for (state, action, next state) in attempt phase
Output: An agent trained have a high success rate turning flowers in attempt

Perform game loop for the number of training eps
while Game is still running **do**
  before state ← env variables known to player
  if player is Q agent and is in attempt **then**
    choose action with greedy ϵ
    Exp Q value ← max over i players (w_i * # of cards in i's stack + w_{i+1} * # of cards in i's hand)
    next state ← env variables after attempt action by q-agent
    diff ← (reward + γ * Exp Q value (next state)) - Exp Q value
    For player i who's stack was turned
    $w_i \leftarrow w_i + \alpha$ * diff * # of cards in i's stack in before state
    $w_{i+1} \leftarrow w_{i+1} + \alpha$ * diff * # of cards in i's hand in before state
    decrease ϵ value
  else
    choose and perform action (may use before state to inform action choice)
  end if
  Check if game reached terminal state, end game loop if in terminal state
end while

These were set in the environment. They are meant to represent that losing on turning your own skull card is worse as you have the knowledge that it is there.

Algorithm 1 shows how the logic for the training the attempt phase weights was implemented into the standard game loop. I trained the agent over 1000 games then tested the agent over another 1000 games. I used the same number of train and test episodes for all Q learning agents. When testing the agent the weights for the features were no longer updated and the action with the max approximate Q value was chosen each time.

Q Learning Agent V2 - Attempt and Raise

I then created a second version of the Q learning agent which had Q learning implemented for both the attempt and raise phases of the game.

Unlike the attempt phase the raise phase is played in a multi-agent environment so the result of any player action is not immediate, and instead be being evaluated right after the action is taken needs to be evaluated at the start the Q agent's next turn. Because of this the algorithm cannot use the action state rewards which are set up in the skull environment as the agent are not evaluating the state right after their action. Instead rewards are determined within the weight update function and are as follows:

- non-terminal state
  - continue to be in raise, 0
- terminal state
  - other player in attempt and q player has skull down, 10
  - other player in attempt and q player no skull down, -1
  - Q player win attempt, 100
  - Q player lose attempt, 0

Originally the "Q player losing an attempt" reward was set to -1 but this lead to the Q agent having a negative win share of games (will discuss further in the experiments section).

Updates to the game loop to train both the raise and attempt phases are outlined in **Algorithm 2** which has formatted such that it is at the end of the document. I used the same ϵ, γ, and α for the raise and attempt training, however I kept separate epsilons for the 2 phases as the raise phase will be trained every round whereas the attempt phase will only be trained when the Q agent is in attempt which will only happen some rounds.

For the second version of the Q learning agent I experimented with 3 different sets of features for the raise phase. There are 2 actions in the raise phase (raise, pass) and so the first set of features corresponded to the raise action and were 0 otherwise and the second set of features corresponded to the pass action and were 0 otherwise [4]. I used the same set of attempt features and attempt rewards from the first version of the Q learning agent for the attempt phase.

Feature Set 1

- Feature 0 - 0 if no skull down, 1 if we have a skull down
- Feature 1 - # of cards Q player has down unless there is a skull card played then it is 0
- Feature 2 - 1 if we are the player in attempt 0 otherwise
- Feature 3 - total number of cards opponents have
- Feature 4 - current max bet

Feature Set 2

- Feature 0 - 0 if no skull down, 1 if we have a skull down
- Feature 1 - total number of cards played
- Feature 2 - current max bet

Feature Set 3

- Feature 0 - 0 if no skull down, 1 if we have a skull down
- Feature 1 - total number of cards played Feature 1 - the % we have to turn for the next bet, calculated as (max bet + 1) / cards on table

Their win shares will be discussed in the experiments section.

[4] In the case of 3 features evaluating for the raise action would have the feature vector [f_1, f_2, f_3, 0, 0, 0] and evaluating for the pass action would be [0, 0, 0, f_1, f_2, f_3]

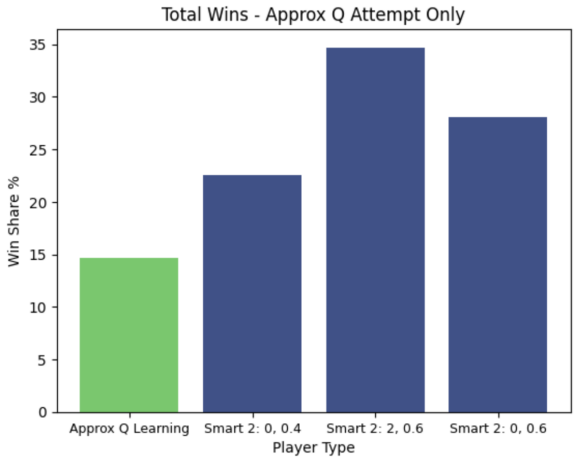

Figure 2: Results from 1000 test games after training the attempt phase for 1000 games

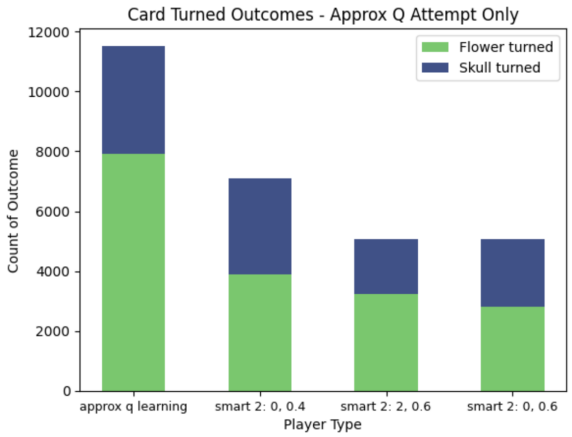

Figure 3: Count of results from every attempt for 1000 test games, Q agent trained for approx only

Experiments

Q Learning Agent V1 - Attempt only

The Q learning agent trained on the attempt phase had the lowest win share out of all of the agents in the simulation (see figure 2). This would indicate that the training has failed, however the the goal of this agent was not to win games but turn the most flower cards up in the attempt phase.

Looking at the results of actions in attempt phase we can split them into 2 categories, turned a flower (win challenge, turn flower) or turned a skull (lose challenge, lose game) plotting all the results of attempts of the during the test games we can see that Q agent is making the most attempt and turning the greatest number of flower cards (see figure 3). This is true for both raw counts and percentage with the Q agent turning a flower card 68% of the time.

The reason that the Q agent is failing to win games is that because it is programmed with the same methodology as the "nice" agent. First the Q agent doesn't place it's skull unless it has to, this is because the only goal of the agent is to win attempts and placing a skull on it stack would mean they would automatically lose. However this means that when other agents are in attempt they score free points off of the Q agent making it easier for them to complete attempts. Second the Q agent always raises, this means that it's bets are significantly higher than other the other agents. They are turning more flower cards but they also need to turn more flower cards than the other agents.

The agent which is winning the most games is the Smart v2 agent with the parameters of 2 skull factor and 0.6 raise threshold [5] which I am going to call the top agent to differentiate it. The top agent's dominance in the test games shows the importance of skull card placement in skull. All of the top agents opponents are placing their skulls less frequently which means that when the top agent makes an attempt they are less likely to fail, conversely when other agents are in attempt they are more likely to fail due to the top agent's skull card. These results are interesting as they contradict what was found during the background section where the optimal strategy against random agents was to never place a skull. These are also more in line with my personal experience playing skull where often the player who is most frequently placing their skull at the start of the game wins.

Q Learning Agent V2 - Raise and Attempt

In the initial trials for extending the Q agent to learn in the raise phase a negative reward was assigned to the terminal case of the Q agent losing an attempt, this caused all feature sets to have a negative win share in their test cases. This is because the magnitude of the negative reward for failing an attempt was greater (-10) than the magnitude of the negative reward for another player being in an attempt without the Q player having a skull card placed (-1). This meant in the cases where the Q agent did not have a skull card placed the agent could either risk getting a very negative reward by raising and putting themselves into the attempt or they could get a smaller negative reward by passing, therefore optimal behavior is to always pass regardless of skull card placement. To win skull the agent must make attempt and will likely lose attempts, therefor losing an attempt should not carry a negative reward. After changing the reward to 0 the third set of features gained the majority of the wins.

Looking at all 3 of feature sets implemented only 1 of the sets was able to gain a positive win share in its test games. Versions 1 and 2 had similar performance only winning 10% of their games, version 3 was able to win 38% of games, which is greater than the win percentage of the top agent in first attempt only approximate Q learning trial. The Smart 2 agent with the skull factor 2 and raise factor 0.6 was again the best performer of the opponents.

I did not train any kind of reinforcement learning for the adding cards phases and wanted to test how the 3rd feature set would perform for different skull thresholds. For the initial testing I had a skull threshold of 0.5 meaning the Q learning agent was equally likely to place a flower or skull card. I choose this as to train the cases having a skull card placed

[5] more likely to place skull card and less likely to raise

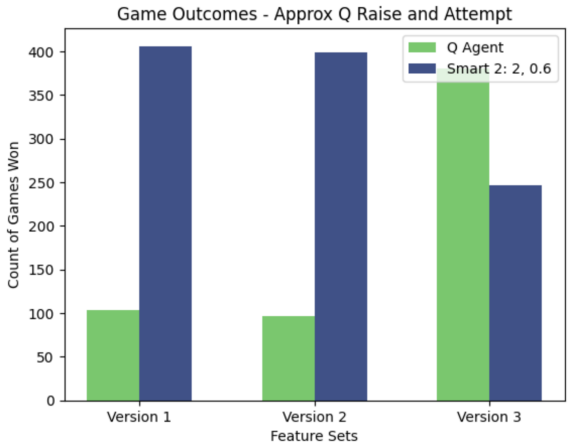

Figure 4: Count of results from every attempt for 1000 test games, across the 3 sets of raise test features

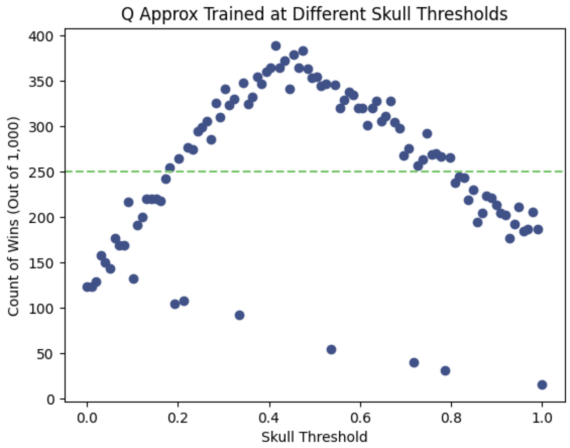

Figure 5: Wins for Q agent at different skull thresholds from 0-1, anything above the green like is a positive win share.

and not having a skull card placed in equal amounts, but that might not be the optimal probability for skull card placement. I ran trials for 100 skull threshold from 0-1 (0 being always place and 1 being never placed) and found that their win share followed a general parabola shape which the best performing q agent having a skull threshold of 0.4 (a little more likely to place skull than not). This aligns with previous results from trials showing that skull placement while not a good path to elimination victory is important for stopping opponents from winning via score.

A limitation of these trails the Q agent is likely benefiting from is the Smart 2 agents deterministic behavior of always raising when the current bet is less than half of the cards played. This means that our agent does not need to bluff as when they have a skull placed. This goes against conventional human wisdom of skull which says that raising while having a skull card placed is an good strategy to get opponents to fail their attempts. As well the Smart 2 agents choose stacks to turn randomly meaning they cannot use the fact that the Q agent is always passing to inform their decision of if the Q agent's pile is safe to turn. An extension on this study could see a version of the Q agent which is only rewarded if an opponent turns their skull cards rather than a reward rewards the agents for all opponent attempts with their skull card dwon regardless of it the agent turns it over. Alternately, a reward structure which rewards the Q agent greater magnitude when an opponents bet is higher as higher bets means the opponent is more likely to turn the Q agents skull and would incentive the Q agent to raise while their skull card is placed to drive up the bet.

Conclusion

I hypothesize that part of the reason that feature set 3 was the only one to produce a positive win share is because it is the set with the least number of features at only 2. Approximate Q learning is good at reinforcement learning in large environments with many state variables provided the features that are chosen are distinct enough to create Q values which reflect the actual value of the state. Feature sets 2 and 3 use the same state variables (if Q agent has placed a skull card, the current bet, and number of cards played) to inform their features however feature set 2 expresses the max bet and the number of cards on the table as 2 separate features where feature set 3 combines them into 1 feature which is the percent cards needed to be turned to fulfill the raised bet. This more accurately expresses what the agent is concerned with which is the percentage of the played cards they think is safe to turn over and minimizes possible value conflation by limiting features. When selecting features for approximate Q learning it is important to consider not just which features will be informative for the agent but also those features will interact with each other when being combined with the weights.

Unlike other researchers who have previously worked on creating agents to play skull, this project was not concerned with the bluffing aspect of the game but rather how skull is a game of incomplete information, with agents being unaware if opponents stacks are safe to turn, and if approximate Q learning can be used to maximize the value of attempt phase and of the game overall. I found that having an good policy in the attempt phase was not enough to produce game wins, but only when defining a good policy in both the raise and attempt phase was the Q agent able to reach a positive win share.

The approximate Q learning agent was able to generate strong results both in the single agent and multi-agent game phases of skull, indicating that approximate Q learning is powerful enough to be used in many different styles of game environment. Further study can be done into approximate learning to play skull including teaching the agent to bluff effectively, expanding beyond simple opponents by having reinforcement learning agents play each other, and having approximate Q agents play humans rather than computer agents.

Bibliography

[1] Alexandru Dumitru Ditu. Skull and Roses: Multi-agent Reinforcement Learning approach of modelling Bluffing and Theory of Mind. PhD thesis, University of Groningen, 2024.

[2] Maria Kapusheva. Playing the Game of Skull-A Multi-Agent Deep Rein- forcement Learning approach. PhD thesis, University of Groningen, 2023.

[3] Hana Lee. Reinforcement Learning Applied to a Game of Deceit. PhD thesis, Stanford University, 2017.

[4] Lui-meme. Skull rulebook, 2013.

Algorithm 2: Simulate Games with Approx Q Leaning in Attempt and Raise

Input: Array of players with a Q agent, Skull Env
Parameter: number of training eps, γ decay rate, α learning rate, rewards for (state, action, next state) in attempt phase and raise phase
Output: An agent trained have a high success rate winning games of skull

Perform game loop for the number of training eps
while Game is still running **do**
    before state $\leftarrow$ env variables known to player

    if update raise weights **and** player q turn **and** (player in raise **or** terminal raise case) **then**
        get new state
        update weights based on difference between new state Q value and before state Q value (last time Q player took an action in raise)
        decrease ϵ value for raise training
        update raise weights $\leftarrow$ False
    end if

    if player is Q agent and is in attempt **then**
        choose action with greedy ϵ for attempt training
        find Q value for action and before state
        perform action
        update weights based on difference between new state Q value and before state Q value
        decrease ϵ value for attempt training
    else if player is Q agent and is in raise **then**
        choose action with greedy ϵ for raise training
        find Q value for action and before state
        perform action
        update raise weights $\leftarrow$ True
    else
        choose and perform action (may use before state to inform action choice)
    end if
    Check if game reached terminal state, end game loop if in terminal state
end while