B351 AI Project: Texas Hold ’em

Jake Sohn[1] and Indiana Reed[1]

[3] *Indiana University, Bloomington, IN, USA*

May 5, 2017

The project was to build an AI for the card game known as Texas Hold ‘em. In order to create this AI, group members Jake Sohn and Indiana Reed programmed in Python. This AI acts as a player in a regular game of Texas Hold ‘em while having different decision factors based on the opponents that it faces. The AI does not read the other player’s cards at any point of the game, and is limited by all the unknowns of a real poker player.

1 Introduction

Games containing an AI provide an easy way for a player to compete, whenever he’d/she’d like. Utilizing AIs for gaming can give players a great way to challenge themselves. For example, many games provide a difficulty level for the AI that the player is playing against, or maybe with. Having such options can allow for a player to advance his/her skills for the game. In an article about an AI being able to beat almost anyone in chess, it was mentioned that the computer’s domination over humanity may start at the chessboard (Murphy, 2015). That is because the AI is able to adapt its knowledge by a deep neural network (learns and makes decisions in a similar way to the individuals that the AI plays) to learn to play at the international master level of chess in 72 hours. On a side note, this type of programming could also advance knowledge needed to help create better programmed self-driving cars.

An AI for a game can even make history. The first AI to beat a world champion chess player was documented on 1997, Deep Blue (created by IBM scientists). This 2,800 pound supercomputer defeated a reigning world chess champion. (Koren, 2016). Not only has an AI competed well with a human in a mind-challenging game like chess, but also a knowledge based game, Jeopardy. Ken Jennings, the man who won the most consecutive rounds of Jeopardy, 74 games, was tasked against an AI in a game of Jeopardy (Goldenberg, 2015). It is important to mention that the knowledge for an AI is not the only important factor for an AI to be a strong game-player. Ken Jennings stated that there was an illusion for those watching Jeopardy, and that was that almost all of the players on Jeopardy knew the correct answers, however the timing on the buzzer was integral for dominating in the game. Watson was built to be fast and smart, and not only does this machine have these capabilities, it is not faced by intimidation or psychological stress (like a human) (Jennings, 2011). Jennings mentioned that Watson was able to send an electric current in order to buzz to answer, which gave it a strong advantage. Jennings was able to find the first (out of three) points multipliers known as a daily double due to his own luck, in a game he played against Watson. In the game, Jennings sought to find all of the daily doubles in order to beat Watson. Jennings however fell second place to Watson. Although Jennings, with his master skills in Jeopardy, did not beat Watson in the match, he was left with an IBM engineer letting him know that he was what inspired the project for Watson. The skills that AIs have in games are not only applicable to the challenges that games have, but are also applicable to problem-solving in the real world. IBM sees a future in such question-answering skills, which is why IBM is using Watson to do such work in fields like medical diagnosis, business analytics, and tech support.

2 Playing Texas Hold ’em as an AI Problem

This game has been transformed into a pure AI agent. That is because a game of Texas Hold ‘em can be simulated by programmer’s request, via this program. The game can handle a variable amount of players. Our AI plays against each of these players, via a simulation. [Strategy 1] In order for the AI to play efficiently, the AI adjusts its playing strategy based on whether or not it recognizes a specific type of poker player. [Strategy 2] Also, the AI player for this game utilizes percentages to decide whether or not it should call, raise, or fold, if the AI has a pair for the pre flop. [Strategy 3] The program reads the 6/7 cards for the AI (accounting the turn and river), in order to accurately identify what the best five card hand is for the AI during the turn/river. The AI does this to simply check for what the best hand is for the AI. [Strategy 4] During the flop, turn, and river stages, the AI will base how good its hand is based on a ranking system. The purpose for having the AI rank its five card hand during the flop/turn/river is to decide what the AI’s chance of success for the hand is, and to decide whether the AI should check, fold, or raise during the flop, turn, and river stages for the hand.

The initial purpose of having the AI recognize different types of players was to ensure that the AI did not fall victim to certain well-developed strategies that players had of their own. For example, in Texas Hold ‘em many poker players like to bluff in order to gain a high amount of chips quickly in a game of poker. In order for the AI to combat this type of player, the AI will establish a read on the given players for a given game of Texas Hold ‘em and recognize if a player is frequently making high bets. The AI will make sure to flag this type of player as a bluffer, and the AI will then adapt its playing strategy so that this player does not capture its chips.

Percentages were used for the AI’s playing strategy when the AI started with a pair for the pre flop. These percentages were used in order for the AI to play wisely, as it was not a good idea for the AI to always raise, bet, or check when the AI started with a pair for the pre flop. If the AI were to raise in this situation every time, a player could easily understand a way that the AI plays. We did not want to give away such a detail for the players playing against the AI, that is why we need a random call, while also being strategic (i.e. utilizing percentages) when deciding whether or not the AI should call a bet, raise, or fold.

3 Code

We decided to use the template, simply because it would give us a starting point. We started by editing the template and making improvements that would help us in debugging. We decided to write our AI in a subclass of the player class. This way, we could test our AI against the default random players and ensure it could beat them. We added a name representation to the default player class, so we could more easily identify players during rounds. Another modification was adding a history of actions parameter to the CallAI function. Lastly, we changed some formatting of the printlines and added a wait for input in-between hands. This allowed us to step through each hand one by one and see results.

Our first task was to make sure our AI could return its best possible hand. We solved this by reading through the documentation for the deuces library and utilizing the evaluate function to rank all hand combinations we could produce. To find all combinations, we used itertools to produce a list of all 5 card hands (7 choose 5). Then, we evaluated each one and returned the one with the best rank (lower being better in the deuces library).

Each time our AI is called to make a decision, we update our aggression value, which has a base value of 1. We check to see if there are aggressive players in the current hand and multiply our own aggro by 1.1 for each aggressive player. Our AI keeps track of opponent’s aggressiveness by using a dictionary with key value pairs in the form of opponent: aggrovalue. This way we will be ready to call an opponent’s bluffs if they have shown to raise a lot. We also update our aggro based on whether we are up in chips compared to the rest of the field. If we are up overall, we multiply aggro by 1.1, otherwise we multiply by 0.9. We use the aggro value as a modifier for random chances, bet amounts, and hand ranks.

We have different strategies for different stages of the game. If the hand is in the preflop stage, we use if statements to check for a pocket pair or high card (face only). To check cards we use the getrankint method from the deuces library. We have set outcomes for each of the possibilities. Pairs are split into two categories: 10 and above, or 9 and below. Some of these outcomes have a chance for a raise. Any time we do a dice roll (random.randint()) we multiply by our aggro value. We also multiply any raise amounts, and sometimes maxbid by the aggro modifier.

For the flop, turn, and river stages of the hand, our strategy is implemented as follows. We rank how good our hand is using the evaluate function from deuces, then convert that rank to a percentage using the getfivecardrankpercentage() method. We then flip this percentage by subtracting it from 1. We flip it because we want this number to reflect our chances of winning a hand, where 1.0 is the best chance and 0.0 is the worst. Then, we modify the value by multiplying the hand rank by our aggro factor. If we are playing aggressively, our AI plays as if it has a better hand. Conversely, it will play very conservatively when behind as if its hand is worse.

We then choose our action based on where hand rank lies. If it is under 0.1, or worse than the 10th percentile, we check with a maxbid of 0 (fold if anyone raises). If we are between the 10th and 20th percentile, we check with a medium maxbid. If we are above the 20th percentile, we raise an amount based on hand rank.

4 Results

In the end, our AI was able to defeat the default random strategy players every time. We consider that a success! The aggro rating system seemed to work out really well, and we would love a chance to test it against real players given more time. It could also better portray aggressiveness by looking at the raise sizes of opponents, not just how often they raise.

On the flipside, our hand ranking system could definitely be more nuanced. We would have liked to have separate outcomes for each class of hand: straight, flush, etc. instead of our 0-1 gradient system.

5 Bibliography

Murphy, Mike. "An AI Computer Learned How to Beat Almost Anyone at Chess in 72 Hours." Quartz. Quartz, 16 Sept. 2015. Web. 04 May 20

Koren, Marina. "When Computers Started Beating Chess Champions." The Atlantic. Atlantic Media Company, 10 Feb. 2016. Web. 04 May 2017. 17.

Goldenberg, David. "Why Ken Jennings's 'Jeopardy!' Streak Is Nearly Impossible To Break." FiveThirtyEight. FiveThirtyEight, 06 May 2015. Web. 04 May 2017.

Jennings, Ken. "Watson Jeopardy! Computer: Ken Jennings Describes What It's Like to Play Against a Machine." Slate Magazine. N.p., 16 Feb. 2011. Web. 04 May 2017.

6 Appendix

```
#Indiana Reed (ijreed) & Jake Sohn (jfsohn)
#CSCI-B351 Final Project
#Texas Holdem AI

import os
from deuces import Card
from deuces import Evaluator
from deuces import Deck
import itertools
import random

maxbet = 100

class Player:
        name = "" #gave players a name for easier debugging
money = 0
curHand = []
def __init__(self, startMoney, name):
self.money = startMoney
                self.name = name
        def __repr__(self): #represented as their name
                return self.name
def returnBestHand(self, board): #CHANGE THIS FUNCTION TO RETURN THE BEST HAND WHEN ASKED AT THE RIVER
return board
def callAI(self, state, actions): #CHANGE THIS FUNCTION. NEVER REFERENCE OTHER PLAYER'S CARDS.
                #You are not allowed to cheat! (obviously)
                #e.g. If we see curState.players[x].curHand, that's unacceptable.
raiseAmount = random.randint(0, 100)
maxbid = max(raiseAmount, random.randint(0, 100))
if maxbid > self.money: #do not remove
```

```
maxbid = self.money
if raiseAmount > self.money: #do not remove
raiseAmount = self.money
possibleActions = ["check", ["raise", raiseAmount]] #can only check or raise,
                #since only one action is processed. Fold if max bid is lower than the biggest raise.
return [ possibleActions[random.randint(0,len(possibleActions) - 1)], maxbid ]

#Our Poker AI
#made it a subclass of Player
#so we can play it against the default Player and see it's better than random!
class OurPokerAI(Player):
        myeval = Evaluator()
        #aggressiveness rating for each opponent (ex. opponentAggro[Player1] = 0.75)
        opponentAggro = {}
        #how aggressive our AI is currently playing, used as a modifier value
        aggro = 1
        #how many actions the AI makes
        turns = 0

        #takes the board (cards on table)
        #ranks all hand+board card combos and returns the best one
        def returnBestHand(self, board):
                cards = self.curHand + board
                all5cardcombos = itertools.combinations(cards, 5)
                bestcombo = []
                minimum = 7462 #worst (highest) hand rank
                for combo in all5cardcombos:
                        #Card.print_pretty_cards(bestcombo)
                        score = self.myeval.evaluate(list(combo), list())
                        #print(score)
                        if score < minimum:
                                minimum = score
                                bestcombo = combo
                #print("Best Combo:")
                #Card.print_pretty_cards(bestcombo)
                return bestcombo

        #takes a state and returns an action
        def callAI(self, state, actions):
                #CHANGE THIS FUNCTION. NEVER REFERENCE OTHER PLAYER'S CARDS.
                #You are not allowed to cheat! (obviously)
                #e.g. If we see curState.players[x].curHand, that's unacceptable.

                #count turn
                self.turns += 1
                #reset aggro
                self.aggro = 1

                #update opponents' aggro ratings based on actions
                for action in actions:
                        #print (action)
                        if action[0] not in self.opponentAggro:
                                self.opponentAggro[action[0]] = 0
                        else:
                                #if they checked, pass
                                if (action[1][0] == 'check'):
                                        pass
                                #otherwise if they raised, increase their aggro factor by 1
                                else:
```

```
                        self.opponentAggro[action[0]] = self.opponentAggro[action[0]]
        #if there is an aggressive opponent in the hand, be ready to call their bluffs
        for player in state.curPlayers:
                if player != self:
                        if (self.opponentAggro[player]/self.turns) > 0.35:
                                self.aggro *= 1.1

        #are you winning or losing?
        #based on how many chips you have vs. the field
        winning = 0
        for player in state.curPlayers:
                if player != self:
                        if self.money < player.money:
                                winning -= 1
                        elif self.money > player.money:
                                winning += 1
        winning = (winning > 0)
        #if you are winning, play more aggressively
        #otherwise, play more conservatively
        if (winning):
                self.aggro *= 1.1
        else:
                self.aggro *= 0.9

        #action is a string: raise, check,or fold
        action = ""
        raiseAmount = 1
        maxbid = 1

        if (state.curStage == "preflop"):
                #if you have a pocket pair
                if (Card.get_rank_int(self.curHand[0]) == Card.get_rank_int(self.curHand[1])):
                        #pair of 10's or higher
                        if (Card.get_rank_int(self.curHand[0]) >= 8):
                                #40% chance to raise up to 50, call up to 100
                                if (random.randint(0,100)*self.aggro > 60):
                                        raiseAmount = int(random.randint(1, 50)*self.aggro)
                                        maxbid = 100
                                        action = "raise"
                                else:
                                        maxbid = 100
                                        action = "check"
                        #pair of 9's or lower
                        else:
                                #call up to 30
                                maxbid = int(30 * self.aggro)
                                action = "check"
                #if you have ace high
                elif (Card.get_rank_int(self.curHand[0]) == 12):
                        #call up to 30
                        maxbid = int(30 * self.aggro)
                        action = "check"
                else:
                        #otherwise, call up to 25
                        maxbid = int(25 * self.aggro)
                        action = "check"
        else: #if flop, turn, or river
                #rank percentage of hand (0.0-1.0 where 1.0 is the best hand)
                handRank = 1.0 - self.myeval.get_five_card_rank_percentage(self.myeval.evaluat
                        #adjust hand rank based on aggression (play like you have a better/worse hand)
                        handRank *= self.aggro
                        if (handRank < 0.1):
                                #if hand is in the bottom 10th percentile, fold
                                action = "fold"
                        elif (handRank >= 0.1 or handRank <= 0.2):
                                #if hand is in the 10 - 20th percentile, check with a maxbid of 100
                                maxbid = 100
                                action = "check"
                        elif (handRank >= 0.2):
                                #if hand rank is greater than the 20th percentile, raise an amount bas
                                raiseAmount = int(handRank * maxbet)
                                #keep maxbid at 100 to stay in the hand if possible
                                maxbid = 100
                                action = "raise"

                #check to make sure you have enough money for maxbid and raiseAmount
                if maxbid > self.money: #do not remove
maxbid = self.money
if raiseAmount > self.money: #do not remove
        raiseAmount = self.money

                #return actions
                if (action == "raise"):
                        return [["raise", raiseAmount], maxbid]
                elif (action == "check"):
                        return ["check", maxbid]
                elif (action == "fold"):
                        #print("Billy folds")
                        return ["check", 0]
                #else: #action == ""
                        #default: choose randomly
        #possibleActions = ["check", ["raise", raiseAmount]] #can only check or raise,
                        #since only one action is processed. Fold if max bid is lower than the biggest
        #return [ possibleActions[random.randint(0,len(possibleActions) - 1)], maxbid ]

#holds state, you'll need to reference this in callAI
class State:
stages = ["preflop", "flop", "turn", "river"]
pot = 0
curStage = ""
players = []
curPlayers = []
board = []

def __init__(self, players):
self.players = players

#deal out cards to the board
def deal(cardAmt):
draw = deck.draw(cardAmt)
if isinstance(draw, int):
curState.board.append(deck.draw(cardAmt))
else:
curState.board.extend(deck.draw(cardAmt))
print("Board: ")
Card.print_pretty_cards(curState.board)

#get bet amounts from each player
```

```
def bet():
actions = []
maxRaise = 0
for player in curState.curPlayers:
action = player.callAI(curState, actions)
actions.append([player, action])
if action[0][0] == "raise" and action[0][1] > maxRaise:
maxRaise = action[0][1]
for action in actions:
if maxRaise > action[1][1]:
curState.curPlayers.remove(action[0])
else:
action[0].money -= maxRaise
curState.pot += maxRaise
print("Player Actions: ")
        for action in actions:
                print (str(action[0]) + ":\n  " + str(action[1][0]) + ", maxbid: " + str(action[1][1])
print("Pot: " + str(curState.pot))
print("Current Players: ")
for player in curState.curPlayers:
                print(player)
print("   Hand: ")
Card.print_pretty_cards(player.curHand)
print("   Money: " + str(player.money))

#Setup for poker game
evaluator = Evaluator()
curPot = 0
players = [Player(2000, "Player1"), Player(2000, "Player2"), OurPokerAI(2000, "Billy")] #CAN CHANGE
print ("Players", players)
curState = State(players)

i = 0
while len(players) > 1:
        #raw_input("Press Enter to continue...") #easier to debug when looking hand by hand
        print("")
        print("======== new hand ========")
i += 1
curState.pot = 0
#ante
for player in curState.players:
ante = min(player.money, 50)
player.money -= ante
curState.pot += ante
        #prepare board and deck
deck = Deck()
deck.shuffle()
        #testCard = deck.draw(1)
        #print(Card.get_rank_int(testCard))
        #Card.print_pretty_card(testCard)
        curState.board = deck.draw(0)
for player in curState.players:
player.curHand = []
player.curHand = deck.draw(2)
        #create players for this round
curState.curPlayers = curState.players[:]
        #go through betting stages
for stage in range(0, len(curState.stages)):
curState.curStage = curState.stages[stage]
```

```
                print("======== " + curState.curStage + " ========")
if curState.curStage == "flop":
deal(3)
elif curState.curStage == "turn":
deal(1)
elif curState.curStage == "river":
deal(1)
bet()
#check if only one player is left
if len(curState.curPlayers) == 1:
print("Round Over")
curState.curPlayers[0].money += curState.pot
print("Winner: ")
                        print(curState.curPlayers[0])
Card.print_pretty_cards(curState.curPlayers[0].curHand)
print("New Stack: " + str(curState.curPlayers[0].money))
break
                #If river check who won
if curState.curStage == "river":
print("Round Over")
scores = []
for player in curState.curPlayers:
                                #fixed this line so it actually evaluates hands and calculates scores
scores.append(evaluator.evaluate(list(player.returnBestHand(curState.board)), list()))
                        #print (scores)
winner = scores.index(min(scores))
                        curState.curPlayers[winner].money += curState.pot
                        print("Winner: ")
                        print(curState.curPlayers[winner])
Card.print_pretty_cards(curState.curPlayers[winner].curHand)
print("New Stack: " + str(curState.curPlayers[winner].money))
break
        #remove broke players
for player in curState.players:
if player.money == 0:
curState.players.remove(player)

#print winner of game
print(str(curState.players[0]) + " has won!")
```