

TEXAS HOLD'EM

WITH MVC AND OBJECTS OF JAVASCRIPT AND PHP



I. PROJECT GOAL

Create Texas Holdem game

By using MVC or Model-View-Controller

Using Objects of JavaScript and php

2. STEPS

Step1

Make a program with C++ to figure out how to embody Texas Holdem card game into a computer program
Analyze it to plan how to utilize JavaScript and php to our project

Step2

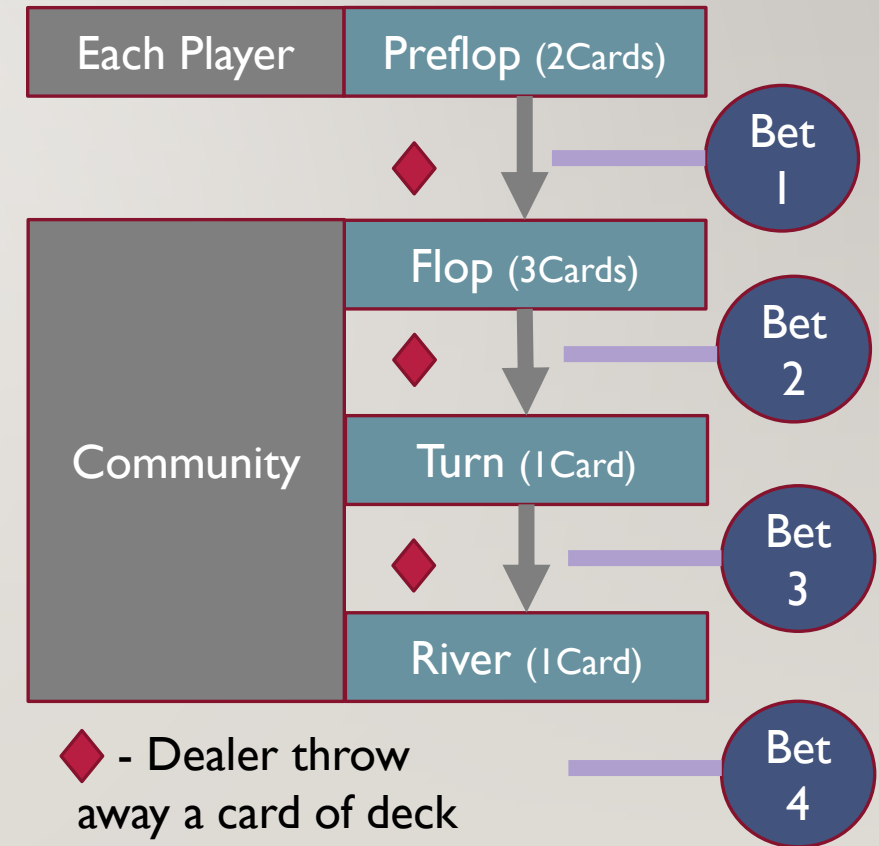
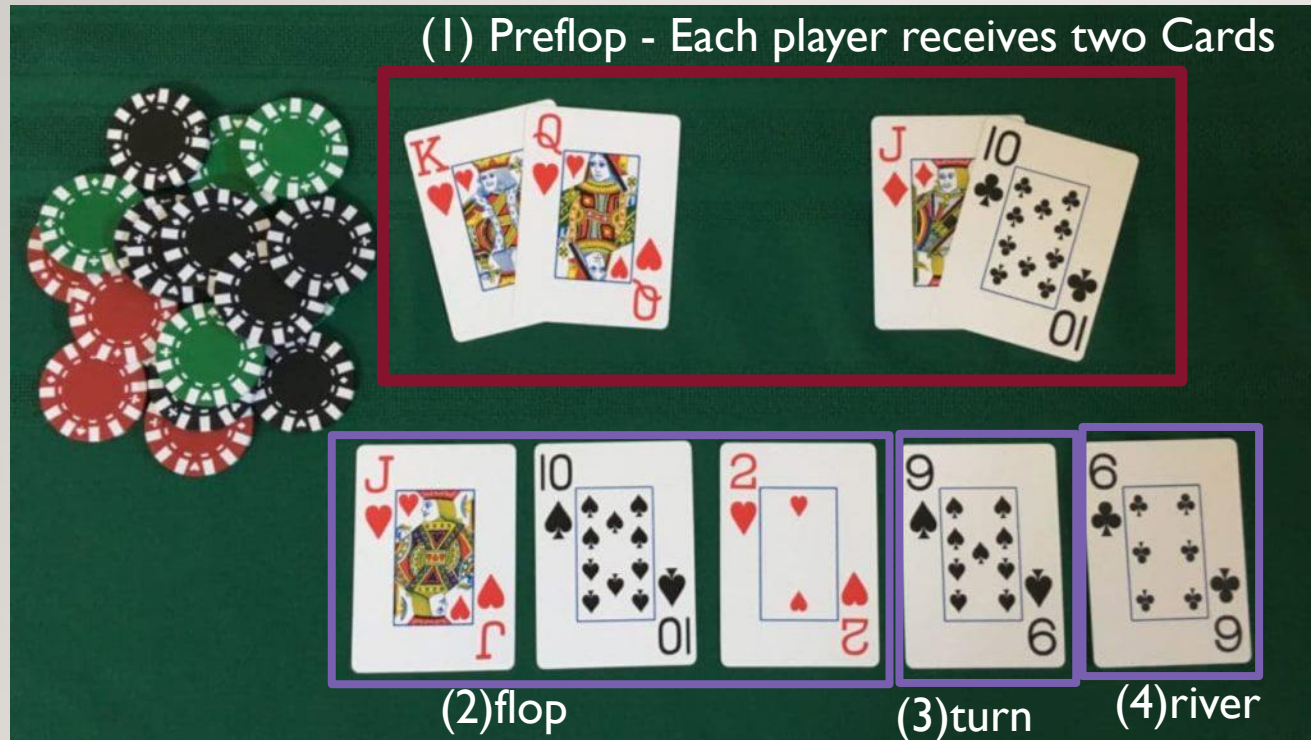
Understand MVC concept throughout comparison between sample game, BattleShip and Texas Holdem

Step3

Apply objects created by C++ to JavaScript and php with the concept of MVC

UNDERSTANDING OF TEXAS HOLD'EM I







- HOW TO DISTRIBUTE CARDS



UNDERSTANDING OF TEXAS HOLD'EM 2

- RANK OF HANDS

- Texas Hold'em is one of Poker game which has a unique rank of hands created by **combinations of pairs and sequences** of face and suit with five cards

Rank of Hands	Examples	
(Royal) Straight Flush	Straight + Flush	
Four of a kind		
Full House		
Flush	Five cards with same kind of suit	
Straight	Five cards in a sequence	
Three of a kind, Two pairs, One Pair		

UNDERSTANDING OF TEXAS HOLD'EM 3

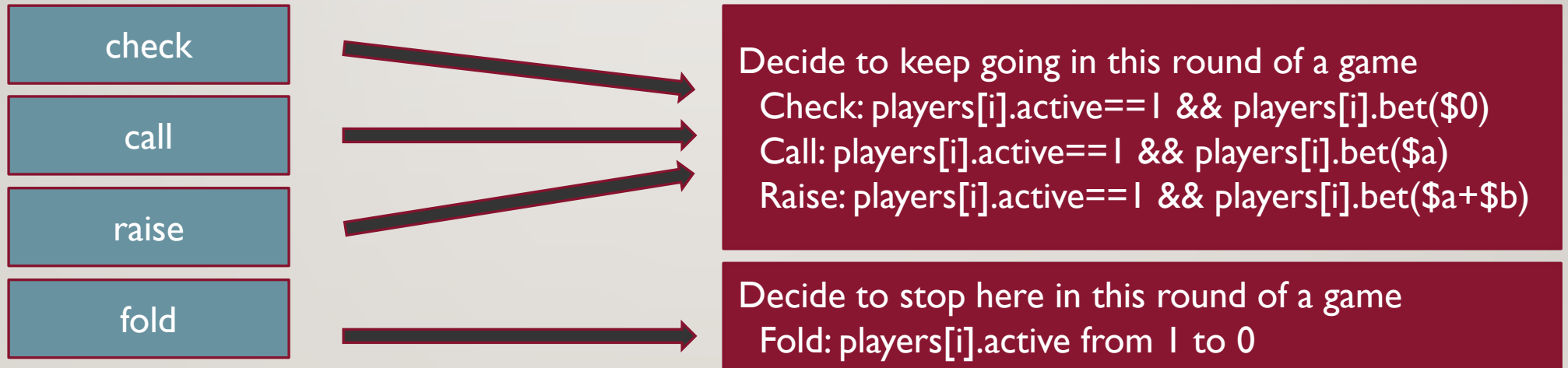
- GIVEN OR PREDESTINED CONDITION

- The number of players
- The result of shuffle
- Game Rule
 - e.g. Big Blind should initially put money into the pot mandatorily (Small Blind $\sim \frac{1}{2}$)
 - Dealer distributes cards with a clock-wise order
 - Procedure \sim “preflop”, “flop”, “river”, “turn”
 - After each procedure, dealer throw away a card
- Betting strategy of another players (Bluffing vs Reasonable Decision) – That is, it's not only about a calculation of probability. Rather, it's about a figuring out another player's betting strategy and its pattern. In this reason, our project is limited with human players.

UNDERSTANDING OF TEXAS HOLD'EM 4

- WHAT PLAYERS CAN DECIDE

- Join/quit/just watch a game
- Betting strategy: check / call / raise / fold



(\$a= amount that previous player bet, \$b=additional amount current player bet)

OBJECT I

- (I) CARD

Class
Hierarchy

- c Card
- c Dealer
- ▼ c Player
- c Hands

(I) Card Membership List

Card(int num)	Card	
getFace() const	Card	inline
getFaceName() const	Card	inline
getNumber() const	Card	inline
getPict() const	Card	inline
getSuit() const	Card	inline
getSuitName() const	Card	inline
setFace()	Card	
setPict()	Card	
setSuit()	Card	
toString()	Card	

(3) Hands Membership List

OBJECT 2

- (2) PLAYER, (3) HANDS : PLAYER

Class Hierarchy

c Card
c Dealer
▼ c Player
c Hands

(2) Player Membership List

addBal(int pot)	Player	Inline
addMyCards(Card card)	Player	
bet(int amount)	Player	Inline
getActStatus()	Player	Inline
getBalance() const	Player	Inline
getMyCards()	Player	Inline
getName() const	Player	Inline
getPlayerAct()	Player	Inline
Player()	Player	
putInThePot()	Player	Inline
resetMyCards()	Player	Inline
setInThePot()	Player	Inline
setPlayerAct()	Player	Inline
setPlayerBal()	Player	Inline
setPlayerInact()	Player	Inline
setPlayerName(string name)	Player	Inline
~Player()	Player	Inline

addBal(int pot)	Player	Inline
addMyCards(Card card)	Player	
bet(int amount)	Player	Inline
checkStraight()	Hands	
getActStatus()	Player	Inline
getBalance() const	Player	Inline
getFaces()	Hands	Inline
getHands()	Hands	
getHandsName()	Hands	
getMyCards()	Player	Inline
getName() const	Player	Inline
getPlayerAct()	Player	Inline
getSuits()	Hands	Inline
Hands()	Hands	
Hands(const Hands &orig)	Hands	
Player()	Player	
putInThePot()	Player	Inline
resetHands()	Hands	Inline
resetMyCards()	Player	Inline
setFaces()	Hands	
setInThePot()	Player	Inline
setPlayerAct()	Player	Inline
setPlayerBal()	Player	Inline
setPlayerInact()	Player	Inline
setPlayerName(string name)	Player	Inline
setSuits()	Hands	
~Hands()	Hands	Inline
~Player()	Player	Inline

OBJECT I

- (4) DEALER

Class Hierarchy

- c Card
- c Dealer
- ▼ c Player
- c Hands

(4) Dealer Membership List

bettingPrompt1()	Dealer
bettingPrompt2(int num, int amount)	Dealer
bettingPrompt3(int num, int amount)	Dealer
calBal()	Dealer
Dealer(int num)	Dealer
decideWinner()	Dealer
displayPlayersInfo(int num)	Dealer
flop()	Dealer
getBigBlind()	Dealer <small>inline</small>
getDeck()	Dealer <small>inline</small>
getNumAct()	Dealer
getNumPlayers() const	Dealer <small>inline</small>
getPlayers() const	Dealer <small>inline</small>
getPotAmount()	Dealer <small>inline</small>

getRound() const	Dealer <small>inline</small>
getRound()	Dealer <small>inline</small>
getSmallBlind()	Dealer <small>inline</small>
nextRound()	Dealer <small>inline</small>
preflop()	Dealer
resetGame()	Dealer
resetPot()	Dealer <small>inline</small>
river()	Dealer
setBlind()	Dealer
setCards()	Dealer <small>inline</small>
setIniCont(int num)	Dealer <small>inline</small>
setPlayers(int num)	Dealer
setRound()	Dealer <small>inline</small>
shuffle()	Dealer
turn()	Dealer
~Dealer()	Dealer <small>inline</small>

MVC

- BATTLESHIP VS. TEXAS HOLD'EM I

	Battle Ship	Texas Holdem
Player's Discretion	Guess = the location of a table	Decide to keep going or Stop the game with betting strategy
Given Condition	Locations of ship created randomly	Order of cards throughout shuffling
View::displayMessage function (msg)	1. Model::fire function(guess) (1) "Oops, you already hit that location" (Input validity) (2) "You sank my battleship!" (3) "HIT!" and "You missed!" 2. Controller::processGuess(guess) (1) "You sank all my battleships ..."	1. Deal::preflop(), flop(), river(), turn() (1) "player[i] checked" (2) "player[i] called \$x" (3) "player[i] raised \$x" (4) "player[i] folded" 2. Controller::decideWinner(Hands); (1) "Player[i] won \$x in 10 th round" 3. Controller::checkPlayers(Deal); (1) "player[i] joined/quited" (2) "player[i] won and earned total \$X" (ini & terminate)

ATTACHMENT I - CSS & HTML OF TEXAS HOLD'EM

* - plan

Name: ...
CurrentBalance: \$
Money in the Pot: \$

<div>P8</div>	<div>J♣</div>	<div>9♣</div>	<div>•••••</div>	<div>P1</div>	<div>10♥</div>	<div>A♣</div>	<div>•••••</div>	<div>P2</div>	<div>8♣</div>	<div>10♠</div>	<div>•••••</div>	<div>P3</div>	<div>4♥</div>	<div>6♣</div>	<div>•••••</div>
000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015
<div>Total amount in the pot && list amount of each player</div>						<div>3♦</div>	<div>K♣</div>	<div>4♣</div>	<div>7♦</div>	<div>6♠</div>					
100	101					106	107	108	109	110	111	112	113	114	115
<div>P7</div>	<div>5♠</div>	<div>3♥</div>	<div>•••••</div>	<div>P6</div>	<div>6♥</div>	<div>Q♦</div>	<div>••~••</div>	<div>P5</div>	<div>9♥</div>	<div>8♦</div>	<div>••~••</div>	<div>P4</div>	<div>K♥</div>	<div>10♣</div>	<div>••~••</div>
200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215

Total amount in the
pot
&& list amount of
each player

Log
P1 checked
P2 raised to \$20
P3 folded
P4 folded
P5 called
P6 quit
.....
.....
.....
.....
.....
.....
.....

Player[i]

Check /call

Raise

Fold

MVC

- BATTLESHIP VS. TEXAS HOLD'EM 2

	Battle Ship	Texas Holdem
View::display[picture]	Model::fire function(guess) calls view::displayHit(location) and view::displayMiss(location)	Deal::preflop, flop, river, turn function() calls view::displayCard(location, players[i].myCards[j].picture)
Initiate a game	HandlersInit.js with guess input	<ul style="list-style-type: none">- HnadlersInit.js with number of players input- After initiating the game, we have to handle betting buttons similarly (waiting next players decision or dealer's action)
Model	<ol style="list-style-type: none">1. Locate ships randomly to cells of the table2. Reflect player's guess to the location3. Show the result of player's guess	4 Classes so far : Card => 2. Player => 3. Deal => 4. Hands

MVC

- BATTLESHIP VS. TEXAS HOLD'EM 3

	Battle Ship	Texas Holdem
Controller.js	<ol style="list-style-type: none">1. It says when this game is done such as the case that player hits all the ships.2. Translate user's guess ("A2") to array location of table("02")	<ol style="list-style-type: none">1. Round control (static NROUND ++) => Card distribution order change => rotate the location of big and small blind => As NROUND increases, base amount goes up2. In the betting process, it changes the status of the player (players[i].active = 1 → 0)3. It Decides a Winner and verifies the changes of balance for each player (bal > 0, betting amount < bal)4. When a player quit, pop the player[i] object5. When a player took all the balance of other players or all the players quit, it terminates the game.6. It asks if players want to do another game and prepare next game with resetting players.

IMPROVEMENTS

- TO DO LIST FOR FINAL

- Figuring out a way to communicate between php and JavaScript
- Players can log in and have their own different view. They also can see other players game without joining the game.
- Exceptional case study, e.g. all-in situation (call betting amount > players[i].balance), immediate winning decision when all other players folded
- Display various message depending on each event like examples in the previous table
- Do...while memory clearing issue

Do{

 play game (preflop() → flop() → turn() → river() → decideWinner() → resetRound())

 nRounds++;

 rotate distributing cards order;

}while(all players not quit || more than two players still play game)