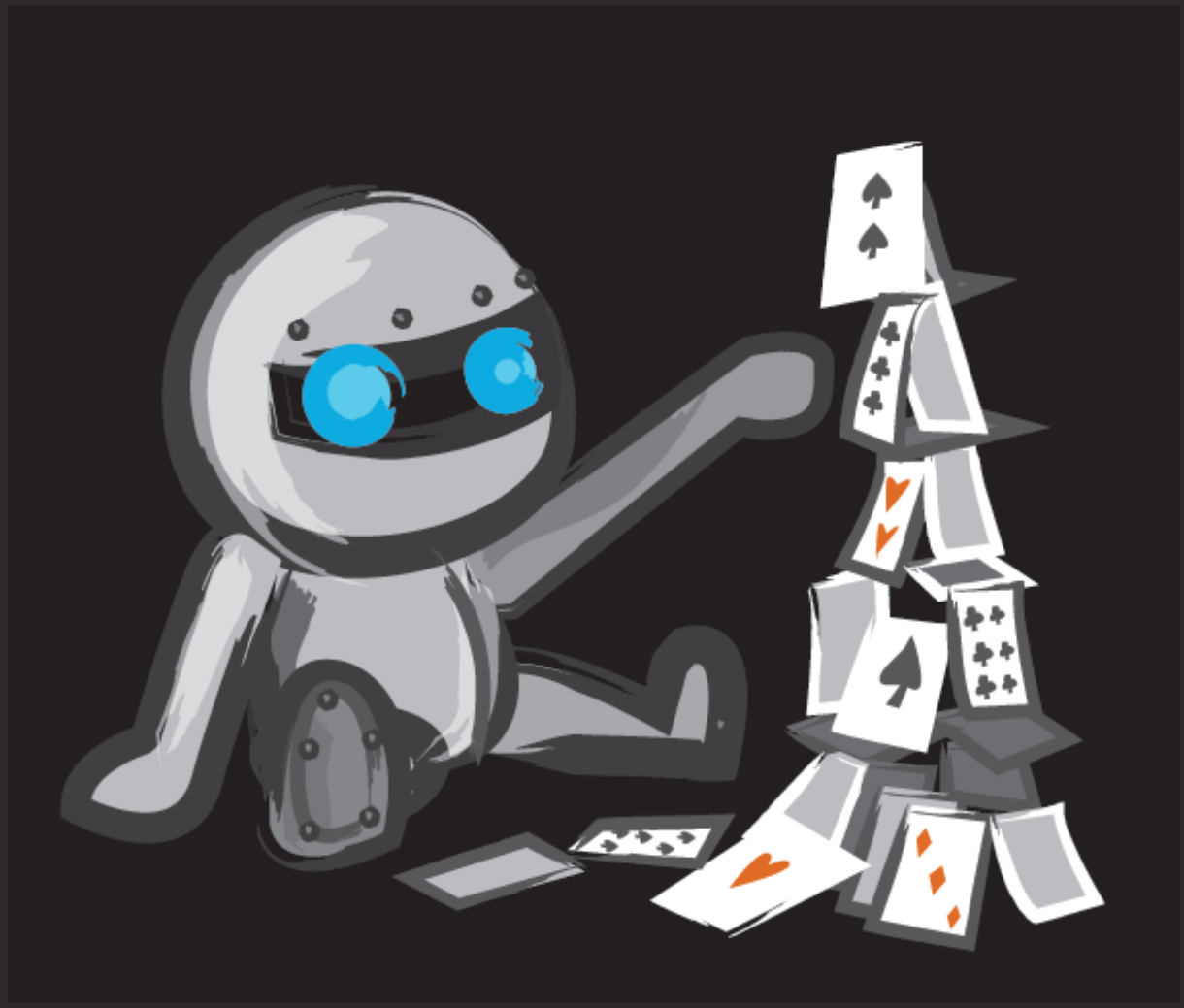
MIT Pokerbots 2015



Administrivia

- Last Lecture!
- Check Piazza for OH next week
- Final event on Feb 2nd, 3:30-5pm at 10-250
- Important dates:
  - Mini Tournament: Tuesday, Jan. 20th, 11:59PM
    - Mandatory bot submission
  - Final Tournament: Thursday, Jan. 29th, 11:59PM
    - Strategy report due Friday, Jan. 30th

Prize Structure

- Casino:
  - 1st place: $1000
  - 2nd place: $500
- Mini Tournament:
  - 1st place: $500
  - 2nd place: $250

Prize Structure

- Newbie Tournament

  (Open to sophomore / freshmen teams)

  1st place: $1000

  2nd place: $500

  3rd place: $250

  4th place: $100

Prize Structure

- Final Tournament
  - 1st place: $10000
  - 2nd place: $5000
  - 3rd place: $3000
  - 4th place: $2500
  - 5th place: $2000
  - 6th place: $1500
  - 7th place: $1000
  - 8th place: $500

Six special $500 awards

Administrative Specs

- Zipped bot size: 300 MB
- Unzipped bot size: 1 GB
- Read access: The bot directory you submit
- Write access: None
- Libraries: poker-eval, boost, numpy, scipy, pbots_calc

Tournament Structure

- Bankroll instant runoff tournament structure
- Team with the lowest bankroll gets eliminated every round
- For every match, simulate SnG tournaments until hand limit has been reached
- Payout the cumulative winner one SnG worth of winnings

Mini Tournament Structure

- 1,000 hands until top 9
- 10,000 hands until top 3
- 25,000 hands until winner

Final Tournament Structure

- 1,000 hands until top 27
- 10,000 hands until top 9
- 25,000 hands until top 3
- 100,000 hands until winner

GTO vs. Exploiter

- Game Theory Optimal (GTO) strategy is unexploitable. EV doesn't change based on opponent's actions.
- Exploiter strategy capitalizes on other players' mistakes but performs suboptimal against GTO.
- Even though GTO consistently beats other strategies, Exploiter may have a higher EV!

Example

- There are three players, GTO, Exploiter, and Fish.
- GTO vs. Exploiter: GTO wins 60% of the time
- GTO vs. Fish: GTO wins 60% of the time
- Exploiter vs. Fish: Exploiter wins 85% of the time
- GTO wins 60% of the time on average
- Exploiter wins 62.5% of the time on average
- Fish wins 27.5% of the time on average

How to Exploit Bots?

- Create a historian class that automatically tallies important statistics on other bots
- If a statistic deviates from a standard average, have the bot adapt and perform different actions in order to exploit this vulnerability

Preflop Stats

- From the Button:
  - Limp
  - PFR
  - 4-bet %
  - VPIP
- From the SB/BB:
  - Raise limp %
  - 3-bet %
  - Fold to 4-bet %
  - VPIP

Exploitation Tactics

- 3-Bet % Too high? Tighten raising range
- 3-Bet % Too low? Raise any 2 cards from the button.
- Fold-to-4-Bet % too high relative to 3-Bet %? 4-bet bluff a lot.
- Button PFR too high? Call and 3-bet more often

Flop Statistics

- Continuation Bet %
- Fold to Continuation Bet %
- Check-raise %
- 3-Bet %

Exploitation Tactics

- Continuation Bet % too high? Check-raise him a lot.
- Fold to Continuation Bet % very high? Continuation Bet the flop a large %.
- Check-raise % high? Always Cbet with good hands, never slowplay. Check back often when you don't have a good hand.

General Postflop Statistic

- Aggression Factor (Raise to Call ratio)
- Win $ at SD %
- Went to SD %

Exploitation

- Aggression Factor high – call more, and raise aggressively to not give them a free card when you do have a good hand.
- Win $ at SD % high – don't value bet too thin, fold to their bets. Bluff a lot.
- Went to SD% high – don't expect them to fold on the flop / turn (bet bigger with good hands!)

Example Historian

Initialize variables at the start of every match

```
// game stats
numHandsPlayed = 0; //hands played so far
winCount = 0; //your win rate
instantFold = 0;
// pre-flop stats
pfrCount = 0; //pre-flop raise
vpipCount = 0; //call or raise pre-flop
initFoldCount = 0; //related to vpip, counts when they don't play their hand.
threeBCount = 0; // re-raise pre-flop
myPFRCount = 0; //how many times we pfr
my3BetCount = 0; //how many times we 3-bet
pfrFoldCount = 0; //fold to initial pre-flop raise
f3Count = 0; // fold to a re-raise pre-flop
callRaiseCount = 0;
hasCalledPreflop = false;
poss3Bet = 0;
poss2Bet = 0;

//post-flop stats
aggressionFactor = 0.0; //(Raise% + Bet%) / Call% , calculated for post-flop only
raiseCount = 0;
betCount = 0;
callCount = 0;
actionCount = 0; // post-flop rates
wtsdCount = 0; //how often opponent goes to showdown after seeing flop (can be used with aggression)
showdownCount = 0; //how often we go to showdown in total
seenFlopCount = 0; //number of times we've made it to the flop
cbCount = 0; //how often opponent continuation bets as the pre-flop raiser
twoBCount = 0; //how often opponent makes another continuation bet after the first one (second barrel)
myCBet = 0; //how many times we c-bet
myBarrel = 0; //how many times we double barrel
fbCount = 0; // fold to continuation bets
f2Count = 0; // fold to second barrels
sdwCount = 0;
foldCount = 0;
checkCount = 0;
oppCheckThisStreet = false;
```

Start with default “placeholder” values for opponents

```
// default stats (the "placeholder" values we use when calculating stuff")
private double vpip = 0.8;
private double pfr = 0.5;
private double threeB = 0.45;
private double pfrFold = 0.3;
private double threeBFold = 0.4;
private double wtsd = 0.5;
private double cbet = 0.2;
private double db = 0.15;
private double cbFold = 0.4;
private double bFold = 0.4;
private double aggression = 1.5;
private double sdw = 0.5;
private double aggrofreq = 0.5;
private double callraise = 0.0;
private double checkraise = 0.0;
private double twoB = 0.15;
```

Update statistics at the end of every hand

```
void update(PerformedAction[] lastActions) {
    this.lastActions = lastActions;
    for (int i = 0; i < lastActions.length; i++) {
        PerformedAction pa = lastActions[i];
        if(pa.getType().equalsIgnoreCase("POST")) {
            currentState = GameState.PREFLOP;
        }
        if (currentState == GameState.PREFLOP) {
            processPreflopAction(pa);
        } else {
            processPostFlopAction(pa);
        }
        if(pa.getType().equalsIgnoreCase("WIN")) {
            if(pa.getAmount() == bb * 3 / 2 && pa.getActor().equalsIgnoreCase(oppName)) {
                instantFold++;
            }
        }
    }
}
```

Code logic of counting different statistics through actions

```
void processPreflopAction(PerformedAction pa) {
    if (pa.getType().equalsIgnoreCase("RAISE")) {
        if((double)(pa.getAmount() - dory.theirBetsThisStreet) / (brain.potSize - pa.getAmount()) < 0.2)
            return;
        if (pa.getActor().equalsIgnoreCase(oppName)) {
            if(theirRaiseCount == 0) {
                preFlopRaiser = true;
                pfrCount++;

                if(hasCalledPreflop) {
                    callRaiseCount++;
                }
            }
            else if(theirRaiseCount > 0 && myRaiseCount > 0) {
                threeBCount++;
            }
            else if(theirRaiseCount == 0 && myRaiseCount == 1) {
                twoBCount++;
            }
            theirRaiseCount++;
        }
        else {
            if(myRaiseCount == 0) {
                //System.out.println("POSS 2 BET");
                poss2Bet++;
            }
            else if(myRaiseCount > 0 && theirRaiseCount > 0) {
                poss3Bet++;
            }
            myRaiseCount++;
        }
    }

    else if (pa.getType().equalsIgnoreCase("FOLD")) {

    }

    else if (pa.getType().equalsIgnoreCase("CALL")) {
        if(pa.getActor().equalsIgnoreCase(oppName)) {
            hasCalledPreflop = true;
        }
    }
```

Calculate opponent statistics

```
public double getPFR() {
    double pfrRate = ((double)pfrCount) / Math.max(1, (numHandsPlayed-instantFold));
    return (25*pfr + pfrRate*(numHandsPlayed-instantFold)) / ((numHandsPlayed-instantFold) + 25);
}

public double getCallRaise() {
    double callRaiseRate = ((double)callRaiseCount) / Math.max(1, (numHandsPlayed-instantFold));
    return (50*callraise + callRaiseRate*(numHandsPlayed-instantFold)) / ((numHandsPlayed-instantFold) + 50);
}

public double getCheckRaise() {
    double checkRaiseRate = ((double)checkRaiseCount) / Math.max(1, seenFlopCount);
    return (50*callraise + checkRaiseRate*seenFlopCount) / (seenFlopCount + 50);
}

public double get3BetRate() {
    double threeBRate = ((double)threeBCount) / Math.max(1, poss3Bet);
    return (50*threeB + threeBRate*poss3Bet) / (poss3Bet + 50);
}
public double get2BetRate() {
    double twoBRate = ((double)twoBCount) / Math.max(1, poss2Bet);
    //System.out.println("TWOBCOUNT: " + twoBCount + " poss2Bet: " + poss2Bet);
    return (50*twoB + twoBRate*poss2Bet) / (poss2Bet + 50);
}
public double getAggression() {
    double rate = (double) (raiseCount + betCount) / Math.max(1, callCount);
    return (200 * aggression + rate * actionCount) / (200 + actionCount);
}
public double getAggressionFrequency() {
    int actions = raiseCount + betCount + callCount + foldCount + checkCount;
    //System.out.println("R: " + raiseCount + " B: " + betCount + " C: " + callCount + " F: " + foldCount + " Check: " + checkCount);

    double rate = (double) (raiseCount + betCount) / Math.max(1, actions);
    return (100 * aggrofreq + rate * actions) / (100 + actions);
}
```

Save statistics for future encounters!

```
else if ("REQUESTKEYVALUES".compareToIgnoreCase(word) == 0) {
    // At the end, engine will allow bot to send key/value pairs to store.
    // FINISH indicates no more to store.
    outStream.println("DELETE " + maj.oppName);
    outStream.println("PUT " + maj.oppName + ":PFR " + maj.getValueToSave("PFR"));
    outStream.println("PUT " + maj.oppName + ":SDW " + maj.getValueToSave("SDW"));
    outStream.println("PUT " + maj.oppName + ":AGGRO " + maj.getValueToSave("AGGRO"));
    outStream.println("PUT " + maj.oppName + ":AGGROFREQ " + maj.getValueToSave("AGGROFREQ"));
    outStream.println("PUT " + maj.oppName + ":CALLRAISE " + maj.getValueToSave("CALLRAISE"));
    outStream.println("PUT " + maj.oppName + ":CHECKRAISE " + maj.getValueToSave("CHECKRAISE"));
    outStream.println("PUT " + maj.oppName + ":3B " + maj.getValueToSave("3B"));
    outStream.println("PUT " + maj.oppName + ":2B " + maj.getValueToSave("2B"));
    outStream.println("FINISH");
}
```

How do we use this information?

Meet bAllin

- bAllin uses a liquid equity model to make decisions
- After every action our opponent makes, bAllin adjusts its equity to reflect its own chances of winning

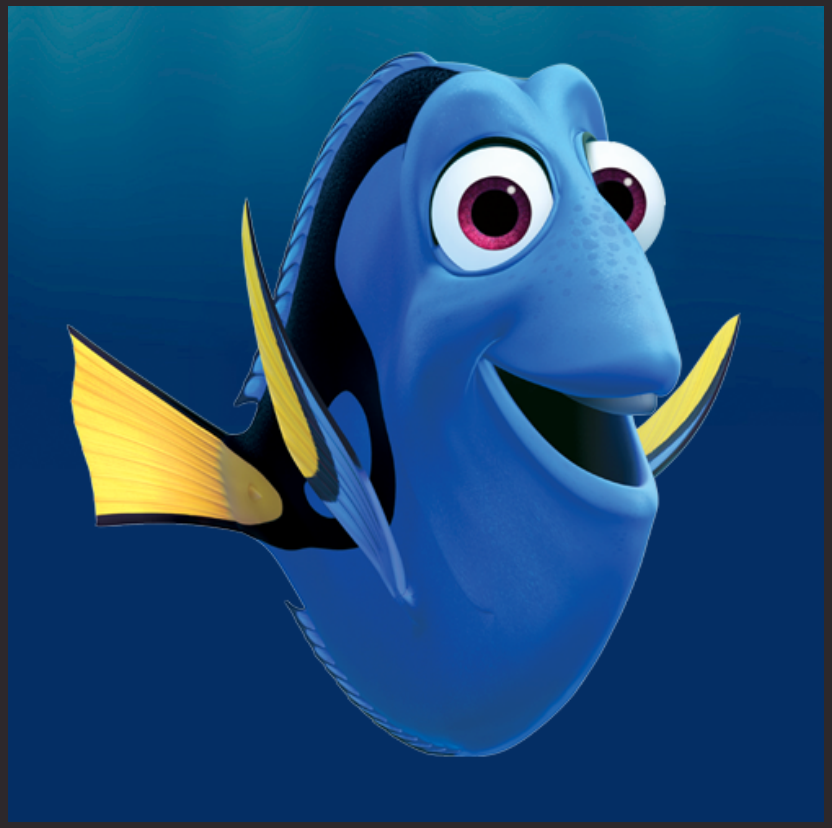
Chief Keef, the face of team bAllin

How bAllin makes decisions

```
protected Action actPreFlopButton() { //small blind acts first
    if(eL(1)) {
        return checkFold();
    }
    else if(eL(2)) {
        return call();
    }
    else if(eL(3)) {
        return call();
        //return putPotPercentage(dory.liqEquity(), eVals[2], eVals[3], pVals[0], pVals[1]);
    }
    else if(eL(4)) {
        return putPotPercentage(dory.liqEquity(), eVals[3], eVals[4], pVals[2], pVals[3]);
    }
    else if(eL(5)) {
        return putPotPercentage(dory.liqEquity(), eVals[4], eVals[5], pVals[4], pVals[7]);
    }
    else if(eL(6)) {
        return putPotPercentage(dory.liqEquity(), eVals[5], eVals[6], pVals[7], pVals[10]);
    }
    else {
        return putPotPercentage(dory.liqEquity(), eVals[6], eVals[7], pVals[10], pVals[13]);
    }
}
```

Meet Dory

- Dory is bAllin's assistant who keeps track of traits of the current hand such as raise history and pot size
- She adjusts the equity given by the calculator based on statistics given by the historian



Stuff Dory keeps track of

```
public class Dory {
    ArrayList<Integer>[] theirRaiseHistory; //array of ArrayLists representing the raise history of the opp
                                            //theirRaiseHistory[0] contains all raises of opp preflop, etc
    ArrayList<Integer>[] myRaiseHistory;
    boolean[] checkHistory;                 //has opponent checked this street?
    int[] betHistory;                       //opponent can only bet once
    double pfRaiseFactor = 0.00026;
    double postFlopRaiseFactor = 0.0005;
    double pfCallFactor = 0.05; //has no effect
    double pfCheckFactor = 0.25;
    double callRaiseFactor = 0.02; // has no effect

    double aggroFactor = 0.2; //has no effect
    double sdwFactor = 0.3;

    double pfrDivFactor = 1.0;
    double aggroDivFactor = 1.0; //has no effect
    double aggroFreqDivFactor = 1.0;

    public GameState currentState;
    private Brain brain;
    private Historian maj;

    public double changeEquity, changeEquityCall, changeEquityTemp;
    public int myBets, theirBets, myBetsThisStreet, theirBetsThisStreet;
```

Example of adjusting equity

```
private void theirBetAction(PerformedAction performedAction) {
    theirBetsThisStreet += performedAction.getAmount();

    double potBet = (double)performedAction.getAmount() / (brain.potSize - performedAction.getAmount());

    if(potBet < 0.2)
        return;

    theirRaiseHistory[currentState.ordinal()].add(performedAction.getAmount());

    double PFR = adjustPFR(maj.getPFR());
    double AGGRO = adjustAggro(maj.getAggression());
    double AGGROFREQ = adjustAggroFreq(maj.getAggressionFrequency());

    double logFactor = HelperUtils.logisticSmall(3.0, 3.0, potBet) *
                       Math.min(100.0, (HelperUtils.logistic(400.0, 400.0, performedAction.getAmount()) + 300.0) / 4);

    if(currentState == GameState.PREFLOP) {
        changeEquity -= pfRaiseFactor * logFactor / PFR;
    }
    else {
        changeEquity -= postFlopRaiseFactor * logFactor / AGGROFREQ;
    }

    changeEquityCall = 0.0;
}
```

Pros and Cons of bAllin

- Pros:
  - Very flexible model, easily adjustable and adaptable to multiple inputs
  - Plays very solid ABC poker, driven entirely by our interpretation of equity at any given moment
- Cons:
  - Many parameters, very hard to find an optimal setup against multiple opponents
  - If our interpretations are wrong, our liquid equity would deviate from our actual equity more than the equity calculator, creating unoptimal decisions