# Pokerbots

## Lecture 2

January 9, 2015

# Administrivia

Next Lecture: 1/12, 34-101 1-2:30

Topics:

- Equity, EV
- Basic poker strategies
- Casino, Bot upload instructions

# Administrivia

Outline:

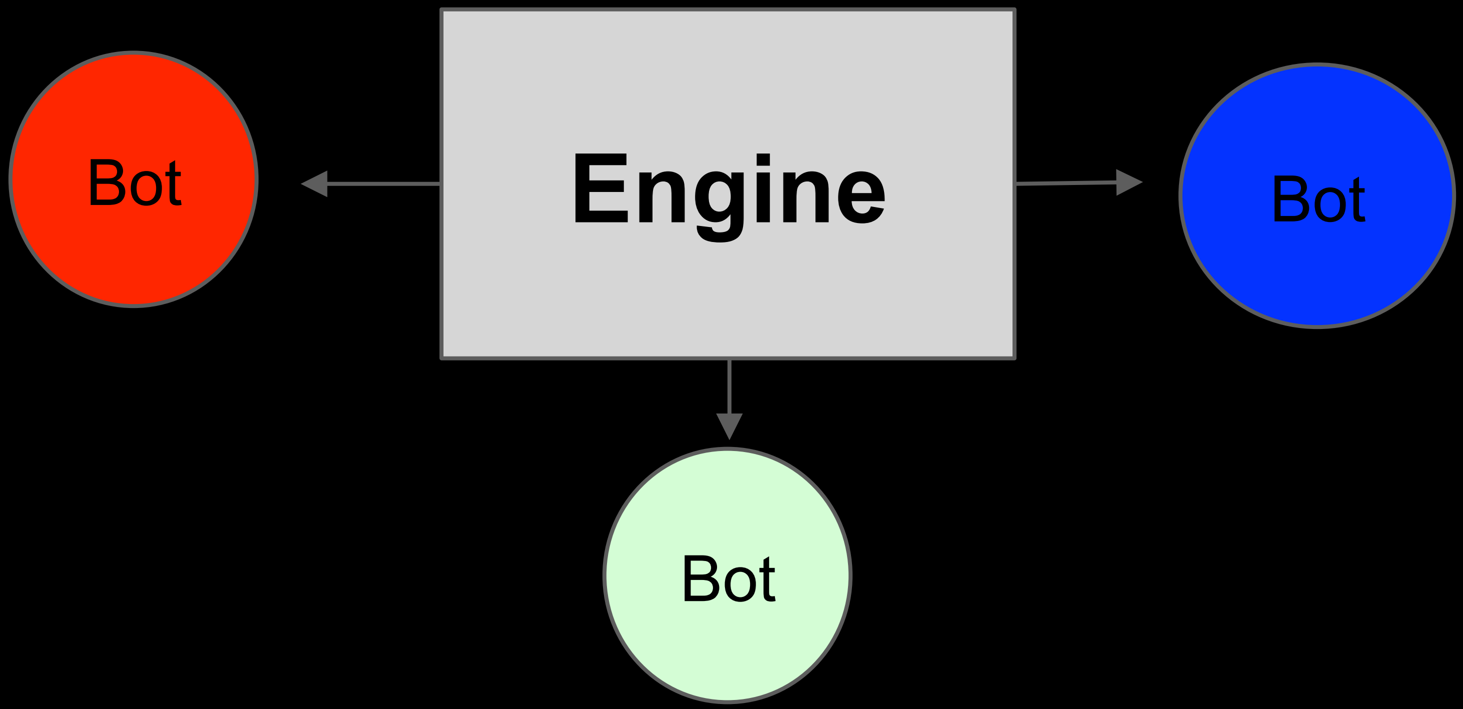
- 3-4pm:
    - Engine
    - Basic Code
    - Equity calculator
- 4-5pm:
    - Dinner
    - Office Hours

# The Engine

# How the Engine works

- Communicate via sockets
- Text-based packets

```
java -jar engine.jar
```



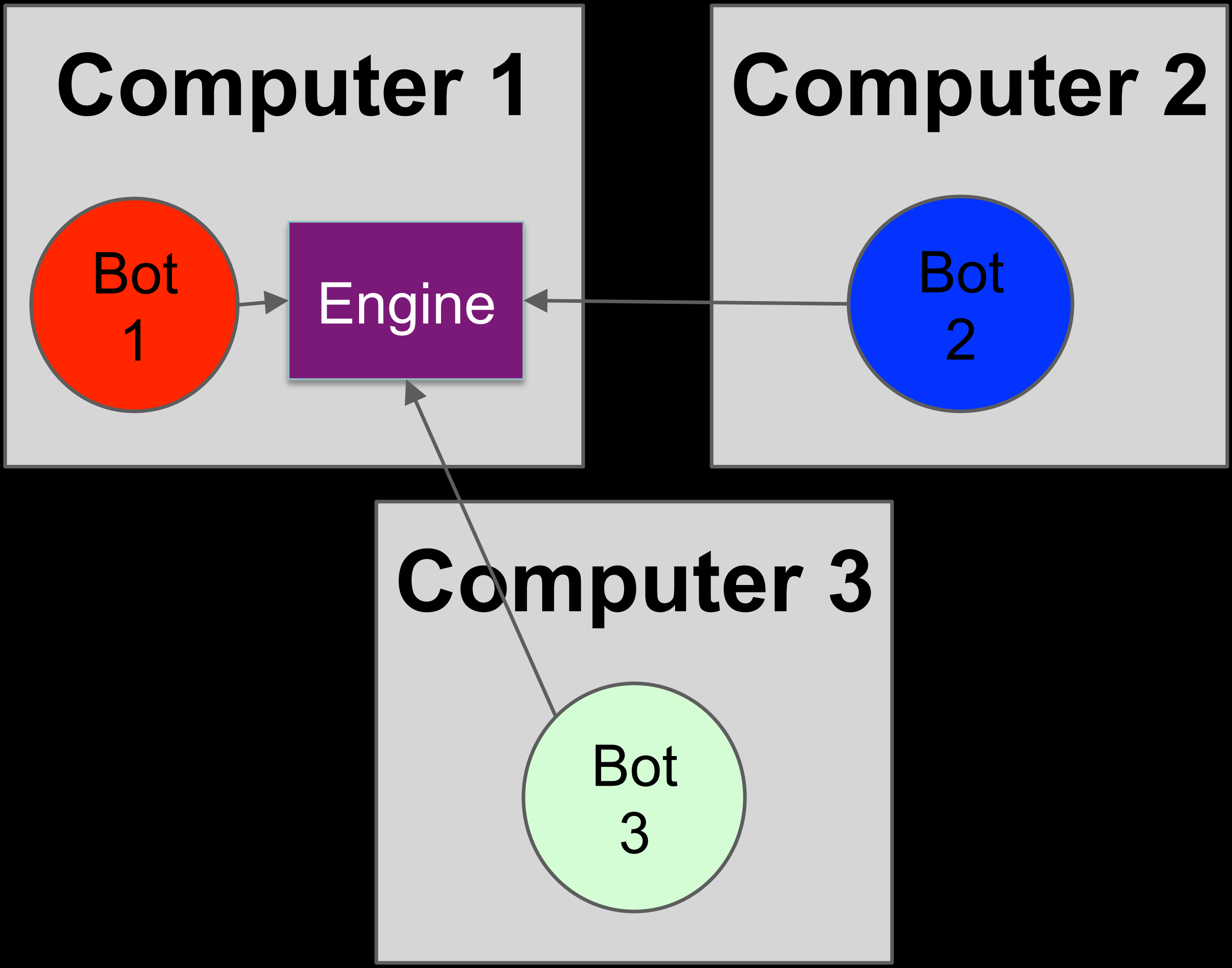
# Engine Configuration

- Configure parameters for testing:
  - Big blind amount
  - Starting stack
  - Number of hands (upper bound)
  - Connection timeout - time given for your bot to initially connect to engine
  - Time bank
  - Enforce time bank
  - Display illegal actions - use this!
  - Triplicate
  - Log File Naming
  - Player types

# Configuration - Player types

- **Folder** - automatically compiles and runs your bot
- **Socket** - connect your bot manually after starting the engine
  - Useful for testing across computers!
- **Random** - test bot which plays with random actions
- **Checkfold** - test bot which always checks or folds

# Testing Across Computers



# Testing Across Computers

- PLAYER_1_TYPE = FOLDERBOT
- PLAYER_2_TYPE = SOCKETBOT
- PLAYER_3_TYPE = SOCKETBOT

## Computer 2

```
./pokerbots.sh -h
18.111.33.113 3000
```

## Computer 3

```
pokerbots.bat -h
18.111.33.113 3001
```

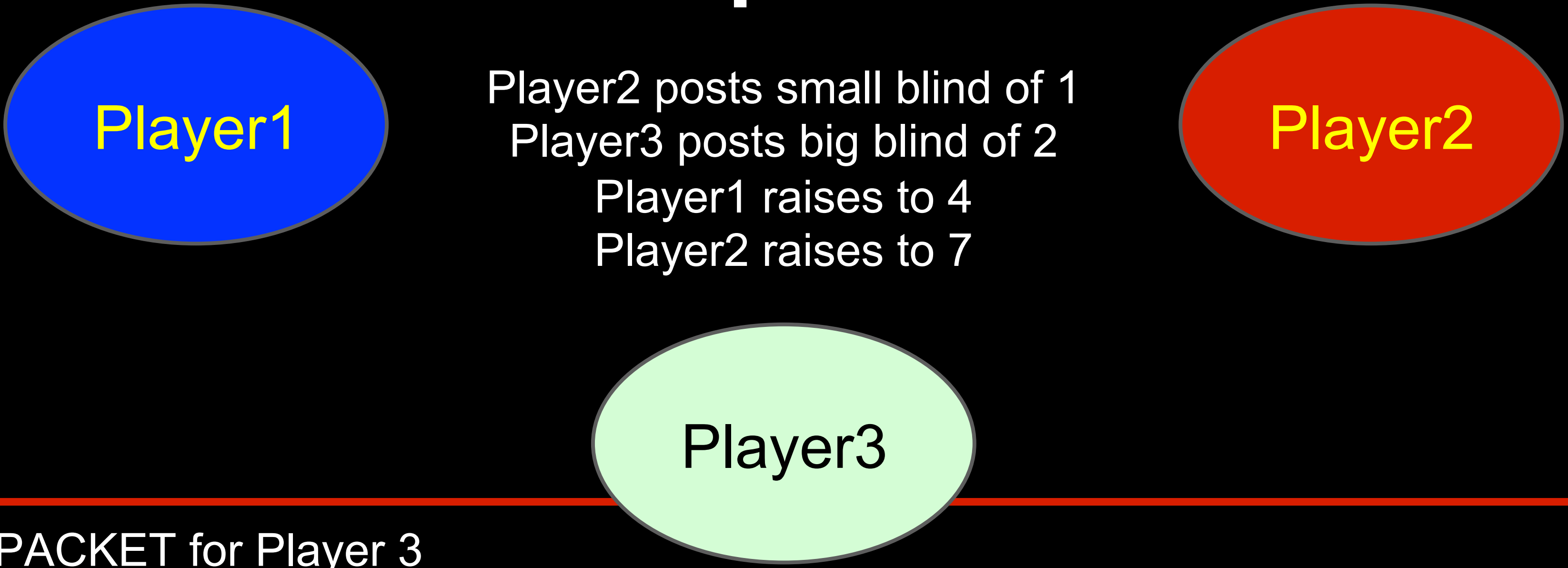
# Engine Packets

- NEWGAME
- KEYVALUE
- NEWHAND
- GETACTION
- HANDOVER
- REQUESTKEYVALUES

# GETACTION packet

- Pot size
- Board cards
- Stack sizes
- Active players
- Last actions
- Legal actions
- Time left

# Example Hand



Player2 posts small blind of 1
Player3 posts big blind of 2
Player1 raises to 4
Player2 raises to 7

PACKET for Player 3

- Pot size: 13
- Board: (empty)
- Stack sizes: 196, 193, 198
- Active players: 3
- Last actions: `POST:1:Player2, POST:2:Player3, RAISE:4:Player1, RAISE:7:Player2`
- Legal actions: `RAISE:10:25, CALL:7, FOLD`
- Time left: 9.65297193

# Example Hand

- PACKET for Player 3
  - Pot size: 13
  - Board: (empty)
  - Stack sizes: 196, 193, 198
  - Active players: 3
  - Last actions: `POST:1:Player2, POST:2:Player3, RAISE:4:Player1, RAISE:7:Player2`
  - Legal actions: `RAISE:10:25, CALL:7, FOLD`
  - Time left: 9.65297193

```
GETACTION 14 0 196 193 198 3 true true true
     4 POST:1:Player2 POST:2:Player3
     RAISE:4:Player1 RAISE:7:Player2
     3 RAISE:10:25 CALL:7 FOLD 9.65297193
```

# Responding to GETACTION

PACKET for Player 3

- Pot size: 13
- Board: (empty)
- Stack sizes: 196, 193, 198
- Active players: 3
- Last actions: POST:1:Player2, POST:2:Player3, RAISE:4:Player1, RAISE:7:Player2
- Legal actions: RAISE:10:25, CALL:7, FOLD
- Time left: 9.65297193

## Valid Responses:

- RAISE:10
- RAISE:16
- RAISE:25
- CALL:7
- FOLD

# Using Datastore

- Don't worry about this until later!
- Bots will be allowed to store some data between matches
- Pass key/value pairs to engine for storage
- No other means of storage is allowed

# REQUESTKEYVALUES packet

- Engine asks bot what information to store after match

- Bot responds with (many) put, delete, or reset requests

    - PUT key value
    - DELETE key - deletes key from storage
    - RESET - clear all values in storage

# KEYVALUE packet

- Engine gives bot all key/value pairs in storage before match

```
KEYVALUE justin_raise_flop 40
KEYVALUE bAllin i hate this bot
```

# Engine summary

- Communication via packets
- Engine can automatically compile/run your bot
- Test across computers!
- Store key/value pairs through the engine

# Writing a bot

```
while True:
  packet = readFromEngine()
  if packet is GETACTION packet:
    // implement logic here
    return "CHECK"
  if packet is REQUESTKEYVALUES
        packet:
    return "FINISH"
```

# Parsing the Packets

- Get the data from a string to a more useful format

```
packet_values = data.split(' ’)
if packet_values[0] == 'GETACTION':
  ''’
  parse the fields of the GETACTION packet
  ''’
  offset = 0 # helps handle optional values
  pot_size = packet_values[1]
  num_board_cards = int(packet_values[2])
  if num_board_cards > 0:
    board = packet_values[3:3+num_board_cards]
    offset += num_board_cards
  stack_sizes = packet_values[3+offset:6+offset]
...
```

# Making Basic Decisions

- Use asserts to debug
- Make modular functions to keep code clean

```
if 'BET' in avail_actions:
  reply('BET’, min_bet, s)
elif 'RAISE' in avail_actions:
  reply(‘RAISE’, min_raise, s)
elif 'CALL' in avail_actions:
  reply('CALL’, call, s)

def reply(action, amount, socket):
  if action == ‘RAISE’:
       assert amount >= min_raise, ‘Raise too low’
       assert amount <= max_raise, ‘Raise too high’
  ...
  socket.send(action + ‘:’ + str(amount) + ‘\n’)
```

# Equity Calculator

# Equity Calculator

- Definition
- Usage (examples)
- Library intro
- Implementation overview

# What is Equity?

Equity = % of pot you expect to win
(assuming go to showdown)

$$= \frac{\sum \text{expected pot fraction won each hand}}{\text{\# hands possible}}$$

It's as simple as that!

# Equity Example 1

Example: What's my equity in the following?

Me: 4sQh

Op: 8h9h

Board: 2hQcAh5s7h

# Equity Example 2

Example: What's my equity in the following?

Me: 4sQh

Op: 89h

Board: 2hQcAh5s7h

# A Quick Note About EV...

- Equity ≠ Expected Value
- Your equity is a fraction of the pot you expect to win, with a value between 0 and 1
- EV is how much you expect to gain/lose, with respect to your stack size (usually with regard to taking a certain action)
- e.g. if I have an equity of .90, the pot is at 15, and to call I need to pay 5, my EV is

  $$(.9)(15)+(.1)(-5) = 13$$

# Reality of equity calculation

- Very important
- Not very interesting to implement
- Hard to implement correctly, with fast evaluations

# Pokerbots Equity Calculator

- Enumeration and Monte Carlo Simulation
- Focused on Texas Hold'em
- Multi-language support (C, java, python)
- Multi-platform support (kinda)
- Alpha release - coming soon!

# Hand Range Syntax

- 2-card hand ranges are straightforward:
    - 8sTd - single hand range
    - 8sTd, 8sTc - 2 hand range
    - 8Ts - 4 hand range: [8sTs, 8cTc, 8dTd, 8hTh]
    - 8To = [8sTc, 8sTd, 8sTh, 8cTs, ...]
    - 88 = [8s8c 8s8d 8s8h 8c8d 8c8h 8d8h]
    - 8T = [8Ts, 8To]
    - JJ+ = [JJ, QQ, KK, AA]
    - 88-TT = [88, 99, TT]
    - xx = random (all possible 2-card hands)

# Combinatorial Growth

Last example, 44 possible hands
Suppose preflop, 4sQh v. 8h9h: 1,712,304
Suppose preflop, 4sQh v. 89s: 6,849,216
Suppose preflop, 4sQh v. 99+: 56,506,032
Suppose preflop, 4sQh v. ??: 2,097,572,400

# Combinatorial Growth

- Evaluation is a combinatorial problem - enumeration may take too long!
- Use Monte Carlo Simulation

# Calculator Install

- Written in C, with Java and Python wrappers
- Relies on poker-eval (http://goo.gl/2n5GO)
  - Installs on all platforms (we'll provide directions)
- With poker-eval installed, can then build calculator library
- Scons builds shared library as well as c, python and java examples.
- https://github.com/mitpokerbots/pbots_calc

# Using the Calculator

- Calculator exports one primary function:

```
calc(hand_str, board_str, dead_str, iters, res)
```

- `hand_str` - string of all hands (ranges)
- `board_str` - string of any board cards
- `dead_str` - string of any discarded cards
- `iters` - maximum number of iterations
- `res` - location to store results
- Returns integer error code (0 on failure, 1 for success)

# Calculator Wrapup

- Simple, yet reasonably fast implementation
- Spares you need to write your own
- Open source (GPL) so you can incorporate it directly into your bot
- Ability to extend and improve for your needs
- Compatible with many different development languages and OS (in progress)
- Still in alpha
  - Let us know if you have trouble/ find bugs