POKER-6S

OOA, OOD, OOP

Files Organization

- Download poker.zip from:
  http://samyzaf.com/braude/OOP/PROJECTS/poker.zip
- Unzip this file in drive C (or D), so your project will reside in: C:\poker (or D:\poker)
- You will find there all the files you need for the Poker project
- You can also access individual files from:
  http://www.samyzaf.com/braude/OOP/PROJECTS/poker
- Make sure to edit the **README.txt** file and enter all the required information (name, email, phones, etc.)
- After completing your project, you should zip this directory back to **poker.zip** and upload it to:
  http://www.samyzaf.com/braude/OOP/upload.html

AGENDA

- **This Final course project 2 (the first was Sudoku)**
- Assuming you have already completed the Blackjack project, starting this project should be easier, so we spend less time on preparations
- Your will need to master the game rules by yourself. We use the **five-cards draw** variation, but we simplified it and created our own special version of the poker game which we call: **Poker-6S**
- Use Google, YouTube, and online games to sharpen your acquaintance with this type of Poker (the **five-cards draw** variation).
- You have to build a software model of a simple version of Poker, and then use your model to simulating thousands of Poker games in order to
  - Collect statistical results
  - Test the quality of several playing strategies (before using them in a real games)
- Invest some time on reading the rules and getting knowledge of the game, think about the problems and how to model them, use **UML diagrams**, and then proceed to implementation

PORJECT GOALS

- Make sure you have a full set of class diagrams before you start your coding (you also have to submit it as part of your project!)
- You can fix these diagrams later along the coding stage
- For each class, write a short **ADT** page that describes its functionality (you must also submit an **ADT** for each class that you design)
- Write as many tests as you can which cover each class and each of its significant methods (even how its objects are printed!)
- Make sure you have a full set of tests that will guide you through your implementation phase

Game Strategy

- An important goal of the project is to develop a game strategy that computes your next move from the current game state (a precise definition will follow later on)
- Your strategy will be tested against the other strategies of the course students, and its success will have some weight on the project grade! (that means that all student teams will participate in a virtual tournament of several thousand games on a fast computer)
- Suggestion: A strategy should be a function **f(stage, hand, bets)** where stage is a number 1-6, hand is my current hand, and bets is a list of ['bet', 'fold', 'unknown'] values of all other players ?? Make sure it returns a cards to exchange at stage 4 …

Poker-6S Rules ("Five-Card Draw Poker") 1

- We will use a simple Poker variant called **"Five-Card Draw Poker"** which is best described in the following links:
  http://www.contrib.andrew.cmu.edu/~gc00/reviews/pokerrules
  http://www.pagat.com/poker/rules
- You can download PDF versions here:
  http://www.samyzaf.com/braude/OOP/PROJECTS/poker/Basic_Poker_Rules.pdf
  http://www.samyzaf.com/braude/OOP/PROJECTS/poker/Five_Cards_Draw_Poker.pdf
- Here is a table of all Poker Hand Types:
  http://www.samyzaf.com/braude/OOP/PROJECTS/poker/pokerhands.pdf
- But even this simple version is still not simple enough, so we make it even simpler by removing some complications (like blinds and unlimited bets), and call it **Poker-6S** (6-stages poker). See next slides.

Poker-6S Rules ("Five-Card Draw Poker") 2

- Number of **Players**: 2-6
- Deck of 52 cards (no jokers!)
- To make it simple: **Dealer** does not play! Just deals the cards!
- Player actions: **bet**, **fold**
  We will give up the usual check, call, and raise actions, and use only the **bet and fold** actions
- Bet unit: 1 chip
- No blinds! (we want our game to be simple ☺)
- Each game consists of **6 stages** as described in the next slide
- In case that all players (except one) fold at an early stage, the game may be closed after less than 6 stages

Game Stages ("Five-Card Draw Poker")

- **Stage 1: RoundBet1**
  All players bet 1 chip (to keep it simple)
- **Stage 2: Draw**
  Dealer draws **5 cards** for each player (face down). Each player gets to see his cards and evaluate his hand. No player can see the other players cards!
- **Stage 3: RoundBet2**
  Each player must either **bet** (add 1 chip to the pool) or **fold** (leave the game and loose the chips he put in the pool). If all fold except one, he takes the pool and game is finished.
- **Stage 4: Exchange Cards**
  Each player (at his turn) can now choose 0 to 3 cards in his hand and ask the dealer to replace them with new cards. The discarded cards are put aside and are not returned to the deck!
- **Stage 5: RoundBet3**
  Each player must either **bet** (add 1 chip to the pool) or **fold** (leave the game and loose the chips he put in the pool). If all players fold except one, he automatically gets the pool and game is finished.
- **Stage 6: Showdown**
  Remaining players expose their cards ("face-up") and the best hand takes the pool. If only one player remained, he takes the pool without showing his cards.
  Equal best hands share the pool (the pool is divided equally between them).

Players Order

- It is clear that the last player has an advantage: if all previous players fold, he automatically gets the pool, even if he has the worst hand!
- It is therefore necessary to change their order on each game
- The simplest fair shuffling scheme is as follows:
  [p1, p2, p3, p4] => [p2, p3, p4, p1]
- It can be easily obtained by the following Python code:
  players = players[1:] + players[0:1]
- Game state can be represented by a tuple like (1,0,1,1,2,2), where
  0 = fold
  1 = bet
  2 = did not play yet
- If we have six players, then player 4 can only know what the first three players did, since player 5 and 6 did not move yet (2)
- The state should be a data member of the Game class?
  Who is going to update it?

Poker Strategy

- A Poker strategy is a function that gets the current game state and tells the player his next move:

  f(stage, hand, bets)

- The game state should include:
  - The player hand (5 cards)
  - The current stage (1-6)
  - What the previous players did (bets/folds)
- This is just an initial suggestion and you may have better ideas when you design and write your code

GameSequence and Game

- Make sure to make a distinction between a game and a GameSequence (a sequence of games)
- A **GameSequence** is a sequence of games with a fixed group of players
- Some of the players may not be able to participate in all games if they go broke in the middle
- Make sure to shuffle the players at the start of each game

SIMULATION

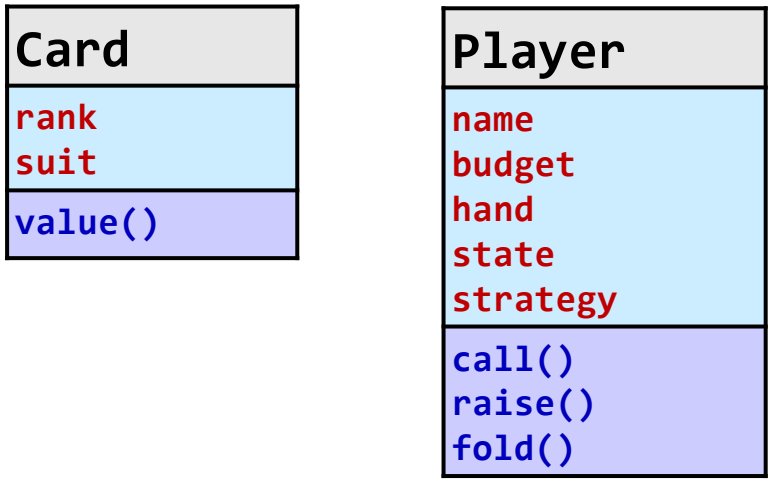
- Our main goal in this project is to test several player strategies by simulating a few thousand games with our software environment
- We will later define what kind of simulations we want to do and may supply several examples of strategies

CONTEST

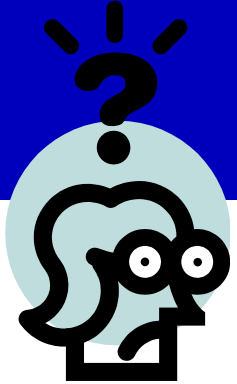
- Each team should come up with its best strategy
- We will conduct a virtual contest between all teams to see who suggested the best strategy
- Naturally, the team with the best strategy will be enttled to a higher project grade …
- More details soon

OOD

OBJECT ORIENTED DESIGN



Classes ?



- Here are some ideas for classes we want to consider – just a suggestion! Nothing final yet … you will decide …
- Do some thinking on what classes you think we should have? And what sort of attributes and methods should they have?

Card
rank suit
value()

Deck
cards
shuffle() draw_card()

Player
name budget hand state strategy
sort() exchange() call() raise() fold()

Dealer
name? state deck
shuffle() draw_card?

Game
dealer players log
open() close() run() history() is_finished ???

Hand
cards soft
add(card) value()

Pool
bets?
deposit()? value()

Any other classes ???


BACKUP

AGENDA

- **OOD** brainstorming in class (but please start thinking about **OOD** before the class)
- We need to decide what are our classes? How do they relate to each other?
- **OOP**
  - After OOD we need to implement our specification in some programming language
  - Naturally we will start with Python
  - Your last assignment in this course is to convert our Python implementation to another language such as Java, C++, or C# - we will discuss this in class

AGENDA

- Remember our long term goal: create a convenient software environment for simulating thousands of Poker games in order to test player strategies (so we know how good they are before we use them in a Casino …:-)
- Please start by designing a few more classes toward this goal
- To get you started, here are client tests and two suggestion for classes that give you a taste for what we are trying to do
- Remember that writing tests (many of them) before you write classes can actually help you make better design choices!

Advanced Challenge: Machine Learning

- The previous experiments are useful for comparing existing strategies
- How about playing millions of games and improving our best strategy?
- After playing millions of games, we may find that our best strategy (strategy2) has some defects and can be fixed by some small changes to the tables
- Probabilistic strategies: after many games we can learn things such as: when player total is soft 18 and dealer is soft 15, then play should hit at probability 0.76 and stand in probability 0.26. These are probabilistic strategies
- Ideas for a future final project …