

COMP 1771 – Fall 2008

HW#2: Blackjack Simulations

Instructor:

Prof. Matt Rutherford – mjr@cs.du.edu

Due by:

10:00am on October 2, 2008

Overview

In this homework, we will write software to model some aspects of the card game Blackjack and we will use it to run simulations of Blackjack. This homework is not intended to encourage you to enter a life of gambling. Rather, you will see how unlikely it is that you would ever make money over the long-haul by playing this kind of game.

I quote some of the important aspects of the game below, but as background, I encourage you to read the Wikipedia page on Blackjack:

<http://en.wikipedia.org/wiki/Blackjack>

Blackjack is played one-on-one against the dealer. From Wikipedia:

The hand with the highest total wins as long as it doesn't exceed 21; a hand with a higher total than 21 is said to bust or too many. Cards 2 through 10 are worth their face value, and face cards (jack, queen, king) are all worth 10. An ace's value is 11 unless this would cause the player to bust, in which case it is worth 1. A hand in which an ace's value is counted as 11 is called a soft hand, because it cannot be busted if the player draws another card.

Initially players are dealt two cards, they then must decide whether or not to keep receiving more cards. The terminology used here is:

- **Hit** – take another card
- **Stand** – take no more cards, let their current score stand

In case it is not clear, as the accumulated score of a hand increases, so does the probability of going “bust”.

When played professionally (e.g., in a casino), the dealer is required to play according to a strict set of rules. In particular, the dealer must keep dealing themselves new cards as long as their total value is below a certain threshold. Once that threshold has been reached, they **MUST NOT** deal themselves any more cards. It should come as no surprise that casinos set this threshold to maximize their chances of winning. According to Wikipedia:

There are two slightly different dealer strategies. In the “S17” game, dealer stands on all 17s. In the “H17” game, dealer hits on soft 17s; of course, he stands on hard 17s. (In either case, the dealer has no choice; he must or must not hit.) The H17 game is substantially less favorable to the player. Which game is customary depends on locality. Las Vegas Strip rules are about equally split.

This homework has four main parts that you should work on in the following order:

1. Model a Blackjack “hand” using a class called `Hand`
2. Model the generic Blackjack “strategy” using a class called `Strategy`
3. Simulate different strategies independently to determine their probability of going bust
4. Simulate two strategies head-to-head to determine the probability of winning

Each of these parts is described below in more detail.

Homework Policies

As stated in the class syllabus:

Homework assignments and labs may be discussed in a general fashion with other students. You should not discuss specific solutions or code. You should NEVER copy assignments that have been written by another student or allow another student to copy your assignments. If any of your work includes ideas or quotes from a book, paper, or web site, you should clearly cite the original source.

Your code (four files: `Hand.java`, `Strategy.java`, `BlackjackSim1.java` and `BlackjackSim2.java`) should be emailed as attachments to me at mjr@cs.du.edu before 10am on October 2, 2008. Please use the subject: “COMP 1771 Homework #2”.

Supporting Code

Three files containing supporting code are provided as part of this assignment. They are available here:

<http://mjrutherford.org/teaching/2008/fall/COMP1771/homeworks>

- `Card.java` – implementation of an enumeration of all the card types (not including suites, since they are irrelevant here)
- `Deck.java` – implementation of a deck of cards. The deck is modeled simplistically, it has a single method for returning a random `Card` object.
- `HandTest.java` – a simple test program that you can use to ensure that your implementation of `Hand` is working properly.

Part 1: Modeling a Blackjack Hand

The goal of this part of the assignment is to implement a class called **Hand** that is able to compute the score of a Blackjack hand as new cards are added.

Hand has the following data members (all private, of course):

- **size** – the number of cards dealt so far
- **score** – the current point value of the hand

- **soft** – a boolean flag that indicates whether a hand is “soft” (true) or “hard” (false). A hand becomes “soft” if an Ace is dealt and the Ace’s value of 11 can be used without busting the hand. If the hand is soft and it subsequently is dealt a card that would put the score past 21, the Ace’s 1 value can be used and the hand becomes “hard” again.

NOTE: – the Hand class does not need to store references to the **Card** objects that are deal to it. The Hand class has the following public methods (that you have to implement properly):

```
public class Hand
{
    public int getScore() { ... }
    public int getSize() { ... }
    public void initialize() { ... }
    public int isBust() { ... }
    public int isSoft() { ... }
    public void playNextCard() { ... }
    public void updateScore(Card c) { ... }
}
```

The methods are described here:

- **getScore** – accessor method for the **score** data member
- **getSize** – accessor method for the **size** data member
- **initialize** – method that deals the first two cards to the hand
- **isBust** – utility method that returns true if the hand is bust, false otherwise
- **isSoft** – accessor method for the **soft** data member
- **playNextCard** – method that receives a random card from the `Deck` class and updates the Hand’s score
- **updateScore** – method that updates the Hand’s internal score and other variables according to what **Card** is passed in.

NOTE1: – `HandTest.java` contains a simple Java application that you can run to test your implementation of Hand. It does not guarantee success, but if you can get it to run without aborting (with assertions enabled), you will be doing pretty well.

NOTE2: – you have to use exactly these method names for the class you implement otherwise **HandTest** will not compile.

Part 2: Modeling a Blackjack Strategy

As described above, a blackjack strategy is used by players and dealers to determine when they stop receiving new cards. A strategy is parameterized by the score at which they stop asking for new cards, and whether they stop at that score if the hand is **soft** as well as **hard**.

For this part you will implement a `Strategy` class. The class has at least the following data members (all private). You may add any more you need to help:

- **hand** – a reference to a Hand object that will be used internally
- **standOnValue** – the score at which the strategy dictates the player stand
- **standOnSoft** – a flag indicating that the player should stand on soft as well as hard if the score is **standOnValue**.

The class should have methods similar to:

```
public class Strategy
{
    public boolean isBust() { ... }
    public boolean getScore() { ... }
    public void play() { ... }
    public void setStandOnSoft(boolean flag) { ... }
    public void setStandOnValue(int value) { ... }
}
```

The methods are described here:

- **isBust** – uses the internal `Hand` object to return its **isBust** value.
- **getScore** – uses the internal `Hand` object to return its score
- **play** – this method causes the strategy to repeatedly call `Hand.playNextCard()` until its parameters dictate that it should stand. At the end of this method call, the `Strategy` object will not let its hand play any more cards.
- **setStandOnSoft** – mutator method for the **standOnSoft** data member
- **setStandOnValue** – mutator method for the **standOnValue** data member

NOTE: – you might want to consider creating a class `StrategyTest` to test the functionality of this class before moving on to the next parts, but this is not required.

Part 3: Simulate Strategies Independently

In this part, you will write a Java application (class with a main function) called `BlackjackSim1`. This application will use the `Strategy` class to simulate strategies with **standOnValues** from 11 to 20, and with **standOnSoft** both true and false (hint: use loops to run all these different simulations). Each strategy will be simulated 5000 times and the probability of going bust for each strategy variant will be printed.

Part 4: Simulate Strategies Head-to-Head

In this part, you will write a Java application called `BlackjackSim2`. This application will use the `Strategy` class to simulate a player using a strategy and a dealer using a (possibly different) strategy. The simulation will compute the probability of the dealer winning in each scenario.

Keep in mind that the dealer can win if they have a higher score than the player or if the player goes bust (regardless of what happens to the dealer). If the player and the dealer have the same score, nobody wins.

Use loops to simulate all combinations of the player and dealer strategies using **standOnValues** of 16 to 18, with **standOnSoft** both true and false. (I.e., you will have 36 simulations). Each simulation should run 5000 times.

This application will print out the probability of the dealer winning in each of the 36 scenarios above.

Grading

Module	Percentage
Correct implementation of <code>Hand.java</code>	40%
Correct implementation of <code>Strategy.java</code>	30%
Correct simulation of strategies independently	15%
Correct simulation of strategies head-to-head	15%