

Project 4: Blackjack

Due: 11:59 PM, Apr 23, 2009

Contents

1	Introduction	1
2	CS141 Blackjack	2
3	What To Do	3
4	Handin	3
5	Support	3
6	Extra Credit	4
7	Resources	4



“You cannot beat a roulette table unless you steal money from it.”

—Albert Einstein

1 Introduction

Expert game playing is one of the claims to fame of modern AI. But the majority of AI’s popular success stories involve deterministic games, like chess and checkers. In this assignment, you will build an intelligent agent that is capable of playing the classic card game of Blackjack, which, unlike chess and checkers, is rife with randomness. You will achieve this goal by formulating the game as a Markov Decision Process (MDP), and solving for an optimal policy.

2 CS141 Blackjack

Blackjack is card game commonly played in casinos, in which one or more players compete against a select player called the dealer. The object of the game is achieve a point value as close to 21 as possible without going over. The CS141 version of Blackjack is a slightly simplified version of the typical casino game. Each hand proceeds as follows:

1. Each player places his or her bet before the hand begins.
2. The initial cards are dealt to the players:
 - Each player is dealt two cards face up.
 - The dealer is dealt one card face up and one face down, the latter of which is revealed to the players later in the hand.
3. The cards have the following point values:
 - Cards of rank 2–10 have point values equal to their rank.
 - Picture cards (Kings, Queens, Jacks) have point values equal to 10.
 - Aces have point values equal to 11, unless the player has a score above 21, in which case Aces take on a value of 1.
4. If any of the players is dealt a *natural*—two cards that sum to 21—that player need not make any decisions. A natural pays 3:2, unless the dealer is also dealt 21, in which case no money changes hands.
5. Otherwise, after the initial cards are dealt, the players choose between two different actions:
 - Stick: If a player chooses to stick, the player’s turn is over.
 - Hit: If a player chooses to hit, the player is dealt another card face up. If the player’s point value exceeds 21 as a result of the newly dealt card, the player *busts* and loses his or her bet. If the player’s point value is less than or equal to 21, the player again has the option of hitting or sticking.
6. After all the players have either gone bust or opted to stick, the dealer reveals his hidden card, and takes the following actions, deterministically:
 - If his score is ≤ 16 , the dealer hits.
 - If his score is ≥ 17 , the dealer sticks.
7. If the dealer busts by drawing cards that lead to a point value greater than 21, all the players who have not themselves busted win money equal to their bets, except players with a natural who win money equal to 150% of their bets.

If the dealer accumulates a point value between 17 and 21, all the players with point values greater than the dealer win money equal to their bets, except players with a natural who win money equal to 150% of their bets.

All the players with scores less than the dealer lose their bets. All the players with scores equal to the dealer *push*, or tie, and don’t win or lose any money.

3 What To Do

Your task in this assignment is to formulate blackjack as an MDP and solve it. One effective way of computing an optimal strategy in an MDP is by implementing a Monte-Carlo control algorithm.

1. **Part I:** The first thing that you have to do is to formulate CS 141 blackjack as an MDP assuming an infinite deck and a fixed bet. This involves deciding what the state space will be, what the available actions are at each state, what rewards are associated with each state (or state-action pair), and what transitions are possible between states. You need not explicitly specify all transition probabilities. *Do this on paper first please!*

Now that you understand how to represent Blackjack as an MDP, implement the MDP interface, which contains methods for representing the information contained in an MDP. Your implementation should map `States` to `ActionValueMaps`. Make sure you subclass both of these abstract classes before starting on your MDP implementation.

2. **Part II:** Solve the control problem (i.e., compute an optimal policy) for your formulation of Blackjack as an MDP by writing a Java class that extends the abstract class `BlackjackPlayer`. Your implementation should use either *Q*-Learning or SARSA to solve this problem. For this part, continue to assume an infinite deck and a fixed bet.

Note: In the lecture notes, *Q*-Learning and SARSA continue forever, but we would like you to choose a reasonable stopping point and return an approximate solution.

3. **Part III:** Drop the assumption that the deck is infinite and begin counting cards. Do not add card counts to the state space; that would make the state space far too large. Instead, keep track of the number of cards remaining in the deck(s), and then calculate transition probabilities based on your counts. Be aware that your `BlackJackPlayer` should be able to deal with more than one deck (52 cards) at a time. The number of 52-card decks is passed in as a parameter to the `BlackJackSimulator`'s constructor.

Finally, add an initial betting decision to your model and your `BlackjackPlayer`. Think about using a betting function that uses the probability information stored in your MDP.

4 Handin

Hand in any code necessary for your implementation of the abstract class `BlackjackPlayer`. In addition, hand in your written formulation of Blackjack as an MDP. This formulation can include a figure and/or a mathematical description, but note that your handin must be electronic. As usual, to hand in your assignment, type `cs141_handin blackjack` from your working directory.

5 Support

The first thing you want to do to get started on this assignment is to install the support code by running `/course/cs141/bin/cs141install blackjack`. This will place the files in your projects directory (`course/cs141/projects/blackjack`). The support code consists of a few `.java` files

and a `Makefile`. The `.java` files contain the classes necessary for testing your blackjack player using the simulator and GUI provided.

The `BlackjackPlayer` class is an abstract class that defines the interface between the simulator and your code. You must subclass this class to implement your blackjack player. All of the call backs and methods are described in that class, so please refer to the code for further instructions. The class `BlackjackSimulator` contains a method `simulateHand()`, which is called repeatedly to test your algorithm. In the `main()` method of `BlackjackSimulator` you will see instructions for adding your implementation of the abstract class `BlackjackPlayer` to the simulation. You must modify the code as indicated to test your algorithm with the simulator and GUI provided.

Also, note that the `Action` and `State` classes are abstract, and will need to be subclassed.

In addition to a simulator we have provided a GUI so that you can watch your algorithm in action. The GUI code lives in the directory `/course/cs141/lib/bjGUI`, and is invoked by default in the simulator. The GUI repeatedly calls the `simulateHand()` method on the simulator and displays the results on the screen. It also allows you to pause the simulation and step back through each of the states. We have also provided a text-based display for those of you who prefer a more traditional viewing experience. Instructions for switching between the two can be found in the `main()` method of `BlackjackSimulator`.

You can compile your code at any time using the `Makefile`, which can be invoked by typing the command `make` into a terminal. You can run your code by typing the command `make run` into a terminal. Make sure that you are executing the command from the directory that contains your code and the `Makefile`. It is strongly recommended that you use the `Makefile` to compile and run your program. You cannot simply compile this program with the familiar `javac *.java`. You can view the TA's Demo blackjack player by typing `make rundemo` in your shell.

Note: If you want to develop in Eclipse, you can install the GUI code in your own directory by running `/course/cs141/bin/cs141install bjGUI`. Having the GUI code at your disposal will make it easier to reference in Eclipse.

6 Extra Credit

Easy Extend your Blackjack player and the simulator to allow for *doubling-down*. This means that after looking at only your first two cards, you may double your initial bet and only take one more card, at which point the players turn is over.

Hard Extend your Blackjack player and the simulator to allow for *splitting*. Splitting is an action that can only be taken after the initial cards are dealt, and only when the two cards have the same rank. The cards are split into two hands and the player must place an additional bet equal in value to their original bet. A new card is dealt to accompany each of the split cards and the two hands are played separately.

7 Resources

1. Blackjack Rules Online
<http://www.blackjackinfo.com/blackjack-rules.php>

2. Reinforcement Learning

<http://www-anw.cs.umass.edu/~rich/book/the-book.html>