Problem 6: The Tiger Casino [Email Submission]

By Aditya Gollapudi

In order to improve endowment returns, Princeton has decided to start a casino whose sole game is Blackjack. Unfortunately, the department put in charge of developing the new casino was HRES, which has always had difficulty with randomness. Knowing this, you decide to see if how money you can make.

Overview of Blackjack Rules

In Blackjack, each card has a value and suit is irrelevant. Cards 2-10 have value corresponding to their label; the Jack (J), Queen (Q), and King (K) have value 10. You may choose whether each Ace has value 1 or 11. The value of a hand is simply the sum of the value of the cards.

The game will work as follows. The dealer will deal two cards face up to you. Then, the dealer will randomly draw two cards from the deck: the first will be face up (i.e. value is known), the second will be face down (i.e. value is unknown). You will have the option to hit as many times as you like where a "hit" is drawing a card randomly from the deck until you either decide to stop or the value of your hand crosses 21, even with all Aces set to 1. If your value exceeds 21 the dealer won't hit at all. Otherwise they will hit until the value of their hand is at least 17 (assuming as many Aces as possible are set to 11 if doing so wouldn't result in a bust). The dealer will then reveal all of their cards

If one player busts (exceeds 21), then the other player wins. If both players bust, then the round is tied. If neither player busts then the player with the higher hand value wins (and if it is tied, then the round is tied). *You are strongly encouraged to read the testing client (which can be downloaded from the problems page) in the language of your choice for a full understanding of how the game is set up.*

Everything You Know About How the Casino Draws Cards

HRES is using a pseudo random number generator (PRNG) to produce a number between 0-51 and return the corresponding card (see next page for how they are mapped). A PRNG essentially only needs to store an internal state; every time it needs to generate another random bit, it will run some deterministic procedure on its state to generate this new bit of randomness. It will also update the internal state in the process. What this procedure is exactly, though, will depend on the PRNG's implementation. If you're completely new to PRNGs, you can read more about them here.

Now HRES has no COS majors on staff, so they have picked one of the three most basic random number generators to serve as the underlying basis for their card drawing procedure (although they may have made *very minor* modifications):

- Linear Shift Feedback Register
- Middle Square Method
- Linear Congruential Register

All three are insecure and thus crackable. Note that their implementation draws **with replacement**, so it is possible, for example, that every card in a round is the two of clubs. And because they expect all Princeton students to use the casino and expect performance issues the internal state of the PRG can be represented in only *16 bits* in order to speed up computation. You will want to eventually figure out which one of the three they're using; after you do, **we encourage you** to Google how to crack the relevant PRNG.

Also note that we will be mapping numbers to cards as follows:

- 0 — King of Clubs
- 1 — King of Diamonds
- 2 — King of Hearts
- 3 — King of Spades
- 4 — Queen of Clubs
- 5 — Queen of Diamonds
- ...
- 51 — Ace of Spades

You will find that these mappings are reflected in the code we have given you.

(There is a next page.)

Overview of the Code Given to You

Upon downloading the zip file linked on the website, you should find a folder with starter code in C++, Java, and Python, along with a folder of 8 randomness files. *Choose the language that you feel most comfortable working with.*

Your job is to implement and submit the `Player` file in your respective language with the best strategy you can come up with. You should not change the name of the object or the 4 methods specified (`shouldHit`, `hit`, `startHand`, `endHand`), nor should you change their signatures. Again, you are encouraged to read the testing file as it contains a number of utilities that you might find useful and are free (in fact encouraged) to copy.

You will notice that there are 8 `txt` files of training data — each of these contains 10,000 card draws from a fixed start state and key of the PRG (the start state and key are not given to you). These are in the precise order the PRNG generated them, where each card draw is on its own line as an int. We have 8 hidden `txt` files against which you will be graded (with different beginning states and keys, but the same PRG). Your grade on each test file will be #(wins) − #(losses), and your overall score will be the sum of your score across all 8 test files. We will run 500 hands on each test file.

You will want to run your code on the sample files we have given you to see if they work. The syntax in each language for doing so is below:

```
python PythonTestClient.py <NUM HANDS> 10000 <PATH TO TEST FILE>
java blackjack/JavaTestClient <NUM HANDS> 10000 <PATH TO TEST FILE>
./CPPTestClient <NUM HANDS> 10000 <PATH TO TEST FILE>
```

By default, the number of hands may not exceed 1/20 of the total number of card draws (10,000), although you may remove this requirement from the testing file if you'd like. If you create your own test file (say, by combining two of the test files we gave you), simply replace the `10000` in the above commands with the number of lines in your test file.

As a fun side note, the top scorer across the entire competition will win 10 cents times their net wins in cash! (If the top scorer is negative, we won't charge you of course.)

Partial Credit and How to Submit: This is a hard problem, so we are giving generous partial credit. Ways to earn partial credit could include *identifying concrete flaws in the PRNG (with evidence)* or *correctly identifying the type of PRNG*. But any solution submitted that ends up winning more than losing will receive more credit than any such writeup.

The only code file you should be submitting (if you are at all) is your `Player` implementation. If you implement any auxiliary functions that are called in the `Player` implementation, put them in the `Player` file you submit. Email any files to coscon.written.submission@gmail.com with *exact* subject *Problem6Submission*. If you must resubmit, *respond to the thread where you sent your original submission; we cannot guarantee that your resubmission will be graded otherwise.*

Real World Relevance and Some Follow-Ups

While you were attacking a casino this time, in the real world attacking pseudo-random generators (or functions) is almost always about breaking cryptographic systems. It is estimated that anywhere between five to ten percent of faulty cryptosystems boil down to an insecure PRNG! For an especially cool example, consider reading about differential cryptanalysis of the old Data Encryption Standard (DES) by Adi Shamir here (note that in this case we are technically trying to attack a pseudo-random *function*, not a PRNG, but these are very closely related).