Documentation

Haode Yang

1 Overview

This is the documentation of the code for the algorithm which calculates the expected value of a card game and suggests optimal move at any given position.

2 Rules of the game

- There are a standard deck of 52 cards facing down
- The player can only draw one card at a time
- For each black card he/she draws, he/she earns $1
- For each red card he/she draws, he/she loses $1
- The player may choose to stop drawing cards at any time and walk away with his/her cumulative winnings

3 How to run code

The code contains two functions.

- The **EValueofGame()** function prints and returns **the expected value of the game before drawing the first card**. It also generates 2 matrices **currentValue** and **continueOrNot**. If currently there are C black cards and D red cards facing down, then
  - currentValue[C, D] gives the current expected value of the game
  - continueOrNot[C, D] shows whether the player should continue drawing cards or not
- The **stateAndPolicy(X, Y)** function prints the current expected value of the game and whether the players should continue drawing cards, given that the players has X cards in hand, of which Y are black. It directly accesses the 2 matrices generated by **EValueofGame()**, thus should only be called after **EValueofGame()** is called.

4 How the algorithm is designed

The value of the game is random because the player could stop at any time and walk away with cumulative winnings. Therefore, before calculating the expected value of the game, it's crucial to derive a criterion of when to stop.

We have 26 black cards and 26 red cards in total. When there are C black cards and D red cards facing down, the player has already gained \$$(D - C)$. Given this, he/she would want to know the expected value of the game if he/she continues drawing cards, which we denote by **valueIfPlay**. If that amount is greater than $(D - C)$, then he/she should continue drawing and stop otherwise. Then the current value of the game is

$$currentValue[C, D] = max(D - C,\ valueIfPlay)$$

Now how to calculate **valueIfPlay**? We could use one-step analysis. There is a chance of $\frac{C}{C+D}$ that the next card is black. Then the player will arrive at the position $(C - 1, D)$. There is a chance of $\frac{D}{C+D}$ that the next card is red. Then the player will arrive at the position $(C, D - 1)$. Therefore, if the player continues drawing cards, the expected value of the game will be

$$valueIfPlay = \frac{C}{C+D} * currentValue[C-1, D] + \frac{D}{C+D} * currentValue[C, D-1]$$

The above equation could be solved recursively. However, applying recursion to this problem would waste a lot of time computing results that have already been computed. Thus, I used loop to solve the equation iteratively, starting from $C + D = 0$ to $C + D = 52$ with the constraint that $0 \leq C, D \leq 26$.

Meanwhile, there are several special cases that should be taken care of:

- When $C, D = 0$, the game ends for good, so $currentValue[0, 0] = continueOrNot[0, 0] = 0$.
- When $C = 0$ and $D > 0$, the next card can only be red, so

$$valueIfPlay = currentValue[C, D-1]$$

- When $C > 0$ and $D = 0$, the next card can only be black, so

$$valueIfPlay = currentValue[C-1, D]$$