



UNIT 9B

Randomness in Computation: Games with Random Numbers

15110 Principles of Computing, Carnegie
Mellon University - CORTINA

1

Simulating a Die



- We want to have a random number between 1 and 6.
- Algorithm:

– Generate a pseudo random number using a PRNG with a very large m .	Range of Number: [0, $m-1$]
– Take the result from the previous step and modulo by 6.	[0, 5]
– Add 1 to the result from the previous step.	[1, 6]

15110 Principles of Computing, Carnegie
Mellon University - CORTINA

2

Using RubyLabs

```

>> include RandomLab
=> Object
>> p = PRNG.new(81, 337, 1000)
=> #<RandomLab::PRNG a: 81 c: 337 m: 1000>
>> rolls = []
=> []
>> 10.times { rolls << (p.advance % 6 + 1) }
=> 10
>> rolls
=> [5, 2, 1, 4, 3, 6, 1, 4, 5, 6]

```

↑
or p.random(1, 6)

Seeding a PRNG

- If we run the same code again, we will get the same sequence since we're seeding with the same integer each time.
- To generate a new seed each time:


```

>> p = PRNG.new(81, 337, 1000)
=> #<RandomLab::PRNG a: 81 c: 337 m: 1000>
>> p.seed(Time.now.to_i % 1000)
=> 574

```
- When would you want to start with the same seed each time?

Simulating a Deck of Cards

- A deck of cards is made up of 52 cards, where each card has a suit and a rank:
 - Suits: Spades (♠), Hearts (♥), Diamonds (♦), Clubs (♣)
 - Ranks: 2, 3, 4, 5, 6, 7, 8, 9, 10, J (Jack), Q (Queen), K (King), A (Ace)
- A standard deck of cards has 1 of each combination of suit and rank.



Cards in RubyLabs

- RubyLabs has an object called a Card that represents a standard playing card.
- ```
>> include RandomLab
=> Object
>> c = Card.new
=> KS
>> c = Card.new
=> 10C
>> c = Card.new
=> 9H
```
- Use of cards requires us to include RandomLab also.

## Cards in RubyLabs (cont'd)

- We can determine the rank or suit of a card:

```
>> c = Card.new
=> 2S
>> c.rank
=> :two
>> c.suit
=> :spades
```

The values for rank and suit are special constants that start with a colon (e.g. `:king`, `:spades`). These are not strings (no quotes).

15110 Principles of Computing, Carnegie Mellon University - CORTINA

7

## Cards in RubyLabs (cont'd)

- We can get a specific card using an index to the `new` function:

```
>> c = Card.new(4)
=> 10S
```

Cards are indexed as follows:

```
A♠ K♠ ... 2♠ A♥ K♥... 2♥ A♦ K♦ ... 2♦ A♣ K♣ ... 2♣
0 1 12 13 14 25 26 27 38 39 40 51
```

15110 Principles of Computing, Carnegie Mellon University - CORTINA

8

## Deck of Cards

- We can create a deck of cards also!

```
>> d = new_deck
=> [AS, KS, QS, JS, 10S, 9S, 8S, 7S,
 6S, 5S, 4S, 3S, 2S, AH, KH, QH, JH,
 10H, 9H, 8H, 7H, 6H, 5H, 4H, 3H, 2H,
 AD, KD, QD, JD, 10D, 9D, 8D, 7D, 6D,
 5D, 4D, 3D, 2D, AC, KC, QC, JC, 10C,
 9C, 8C, 7C, 6C, 5C, 4C, 3C, 2C]
```

- Note that the cards are in the same order as the indexes given in the previous slide.

## Dealing Random Cards

- Suppose we have a card game like Poker where we want to be dealt a “hand” of 5 random cards from the deck.
- What is wrong with the following code?

```
hand = []
5.times { hand << Card.new }
```

## Shuffling the Deck

- We should shuffle a deck and then create a hand from the first 5 cards in the deck.
- There are many ways to shuffle a deck of cards.
- One algorithm:
  - Exchange (swap) the first card with a random card somewhere to its right.
  - Exchange the second card with a random card somewhere to its right.
  - ... Repeat until all cards have been swapped.

15110 Principles of Computing, Carnegie  
Mellon University - CORTINA

11

## Building the Function

- For the first card (at index 0) in deck **d**, how do we generate a random index for a card to its right?  
**r = random(1, d.length-1)**
- How do we swap the first card with the randomly-selected card?  
**temp = d[0]**  
**d[0] = d[r]**  
**d[r] = temp**  
or we can use *parallel assignment* in Ruby...  
**d[0], d[r] = d[r], d[0]**

15110 Principles of Computing, Carnegie  
Mellon University - CORTINA

12

## Building the Function (cont'd)

- For the second card (at index 1) in deck **d**, how do we generate a random index for a card to its right?

```
r = random(2, d.length-1)
```

- How do we swap the first card with the randomly-selected card?

```
temp = d[1]
```

```
d[1] = d[r]
```

```
d[r] = temp
```

or we can use *parallel assignment* in Ruby...

```
d[1], d[r] = d[r], d[1]
```

## Building the Function (cont'd)

- For the third card (at index 2) in deck **d**, how do we generate a random index for a card to its right?

```
r = random(3, d.length-1)
```

- How do we swap the first card with the randomly-selected card?

```
temp = d[2]
```

```
d[2] = d[r]
```

```
d[r] = temp
```

or we can use *parallel assignment* in Ruby...

```
d[2], d[r] = d[r], d[2]
```

## In general...

- For the card at index `i` in deck `d`, how do we generate a random index for a card to its right?  
`r = random(i+1, d.length-1)`
- How do we swap the first card with the randomly-selected card?  
`temp = d[i]`  
`d[i] = d[r]`  
`d[r] = temp`  
 or we can use *parallel assignment* in Ruby...  
`d[i], d[r] = d[r], d[i]`

## Shuffling the entire deck and dealing five cards...

```
def permute!(d)
 for i in 0..d.length-2 do
 r = random(i+1, d.length-1)
 d[i], d[r] = d[r], d[i]
 end
 return d
end
```

```
>> hand = permute!(new_deck).first(5)
=> [3H, AD, 3S, 3D, 7D]
```



## Poker: Detecting a Flush

- In poker, a flush is a hand where all of the cards have the same suit.
- One possible algorithm:
  - If all of the cards have a suit of spades, return true.
  - If all of the cards have a suit of hearts, return true.
  - If all of the cards have a suit of diamonds, return true.
  - If all of the cards have a suit of clubs, return true.
  - If none of the above tests returns true, return false.

## Poker: Detecting a Flush (cont'd)

```
def all_spades?(hand)
 for i in 0..hand.length-1 do
 return false if hand[i].suit != :spades
 end
 return true
end
```

`all_hearts?`, `all_diamonds?` and `all_clubs?`  
are written similarly.

## Poker: Detecting a Flush (cont'd)

```
def flush?(hand)
 if all_spades?(hand) return true
 if all_hearts?(hand) return true
 if all_diamonds?(hand) return true
 if all_clubs?(hand) return true
 return false
end
```

## Simple dice game

- A player has two die. On each roll, if the player does not roll “doubles” (same value on each die), then the player wins the sum of the die values. Otherwise, the player earns a “strike”. The game ends once the player has three strikes.
- Write a function that returns the amount the player wins in a simulated simple dice game.

## Rolling a die

```
def roll()
 return rand(6) + 1
end
```

## One round of the game

```
die1 = roll()
die2 = roll()
if die1 == die2 then
 strikes = strikes + 1
else
 sum = sum + die1 + die2
end
```

## Putting it together

```
def simple_game()
 strikes = 0
 sum = 0
 while (strikes < 3) do
 die1 = roll()
 die2 = roll()
 if die1 == die2 then
 strikes = strikes + 1
 else
 sum = sum + die1 + die2
 end
 end
 return sum
end
```

15110 Principles of Computing, Carnegie  
Mellon University - CORTINA

23

## What is the average winnings for 1000 players of this game?

```
>> games = []
=> []
>> 1000.times { games << simple_game() }
=> 1000
>> games
=> [61, 86, 127, 140, ... , 114, 292]
>> total = 0
=> 0
>> games.each { |score| total += score }
=> [61, 86, 127, 140, ... , 114, 292]
>> total/1000.0
=> 106.731
```

15110 Principles of Computing, Carnegie  
Mellon University - CORTINA

24