Technical University of Cluj-Napoca
Faculty of Automation and Computer Science
Department of Computer Science

2017/2018

Software Engineering

Name: Acu Raul-Mihai
Group: 30432
E-mail: acuraulm@gmail.com

RandoBot

Prof. Dr. Eng. Eneia Nicolae Todoran

‘

Table of contents

1. Abstract

a. Project problem statement

RandoBot is a game developed in Python, which is based on two probabilistic PRISM case studies and aims to act like a user-friendly interface for those interested in the correctness and performance of randomized distributed algorithms for various scientific departments or domains. The goal of the game is for the robot to reach the top-right corner of the board within a given time. Using a third PRISM case study, the simple Peer-to-Peer protocol, the game can be easily deployed for playing or further development on any platform. Furthermore, to increase the accessibility and flexibility of the project, everything runs on virtual machines in cloud.

b. PRISM case studies

- Randomized distributed algorithms: Dice Programs
- Planning and synthesis: Grid World Robot
- Performance and reliability: Simple peer-to-peer protocol

c. Technologies used

- Google Cloud
- Python Socket/TCP Programming
- NumPy

2. Specification

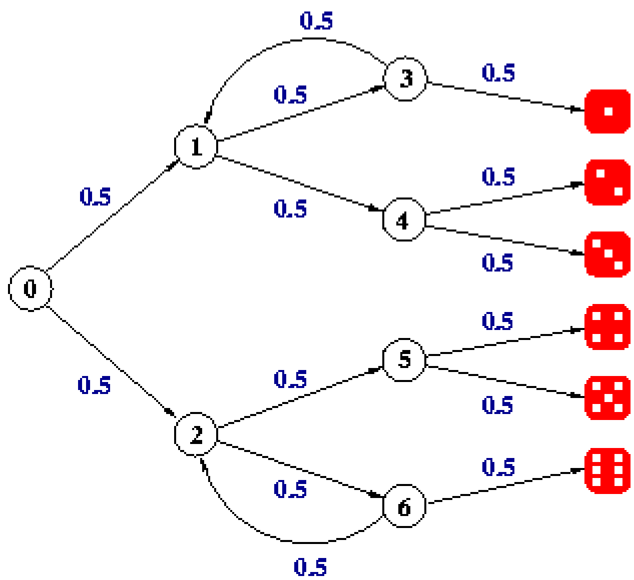
a. PRISM model

Dice programs model

(Knuth & Yao)
This case study considers two probabilistic programs, due to Knuth and Yao [KY76], which model fair dice using only fair coins. We have reimplemented work done by Joe Hurd, the key difference being that the latter uses a *theorem prover* (HOL) whereas we use a *model checker* (PRISM) to show the correctness of such probabilistic programs.

Value of a die

The following program models a die using only fair coins. Starting at the root vertex (state **0**), one repeatedly tosses a coin. Every time *heads* appears, one takes the upper branch and when *tails* appears, the lower branch. This continues until the value of the die is decided.



The PRISM code for this program is as follows.

```
// Knuth's model of a fair die using only fair coins
dtmc

module die

	// local state
	s : [0..7] init 0;
	// value of the dice
	d : [0..6] init 0;

	[] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
	[] s=1 -> 0.5 : (s'=3) + 0.5 : (s'=4);
	[] s=2 -> 0.5 : (s'=5) + 0.5 : (s'=6);
	[] s=3 -> 0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1);
	[] s=4 -> 0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3);
	[] s=5 -> 0.5 : (s'=7) & (d'=4) + 0.5 : (s'=7) & (d'=5);
	[] s=6 -> 0.5 : (s'=2) + 0.5 : (s'=7) & (d'=6);
	[] s=7 -> (s'=7);

endmodule

rewards "coin_flips"
	[] s<7 : 1;
endrewards
```

This model has 13 states and in PRISM it takes 4 iterations to find these reachable states.
To prove the above program is correct, we show that the probability of reaching a state where d=k for k=1,...,6 is 1/6.
In PRISM this corresponds to calculating the probability of satisfying the formula

P=? [**F** s=7 & d=k] for k=1..6.

Performing this verification in PRISM using iterative numerical methods, we find that the probability for each k is indeed 1/6 (up to an accuracy of six decimal places), with each case requiring 22 iterations.
As shown in [KY76], this program takes on average 11/3 coin tosses to output a dice throw and this value is optimal.
The expected time can be calculated by in PRISM through the formula

R=? [**F** s=7]

Performing this verification in PRISM using iterative numerical methods, we find that the expected number of coin tosses is indeed 11/3 (up to an accuracy of six decimal places), requiring 21 iterations.

Sum of two dice

To generate the sum of two dice throws using the above program using PRISM, we combine two such processes in asynchronous parallel composition as follows:

```
// sum of two dice as the asynchronous parallel composition of
// two copies of Knuth's model of a fair die using only fair coins

mdp

module die1

        // local state
        s1 : [0..7] init 0;
        // value of the dice
        d1 : [0..6] init 0;

        [] s1=0 -> 0.5 : (s1'=1) + 0.5 : (s1'=2);
        [] s1=1 -> 0.5 : (s1'=3) + 0.5 : (s1'=4);
        [] s1=2 -> 0.5 : (s1'=5) + 0.5 : (s1'=6);
        [] s1=3 -> 0.5 : (s1'=1) + 0.5 : (s1'=7) & (d1'=1);
        [] s1=4 -> 0.5 : (s1'=7) & (d1'=2) + 0.5 : (s1'=7) & (d1'=3);
        [] s1=5 -> 0.5 : (s1'=7) & (d1'=4) + 0.5 : (s1'=7) & (d1'=5);
        [] s1=6 -> 0.5 : (s1'=2) + 0.5 : (s1'=7) & (d1'=6);
        [] s1=7 & s2=7 -> (s1'=7);

endmodule

module die2 = die1 [ s1=s2, s2=s1, d1=d2 ] endmodule

rewards "coin flips"
        [] s1<7 | s2<7 : 1;
endrewards
```

To prove the above program is correct for calculating the sum of two dice, we show that, both the minimum and maximum probability of reaching a state where s1=7, s2=7 and d1+d2=k for k=2,...,12 is:

k:		probability:
2		1/36
3		1/18
4		3/36
5		1/9
6		5/36
7		1/6
8		5/36
9		1/9
10		3/36
11		1/18
12		1/36

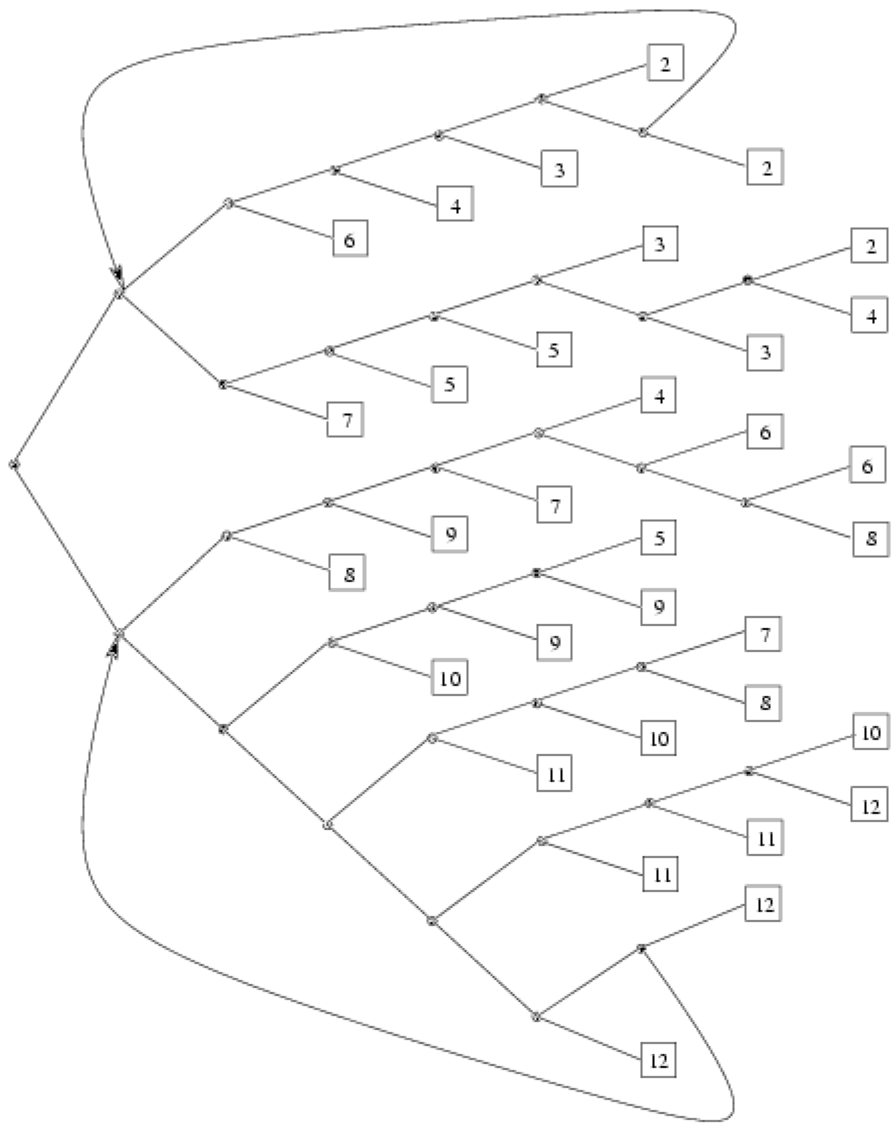
In PRISM this corresponds to verifying the formulae:

Pmin=?[**F** s1=7 & s2=7 & d1+d2=k] and **Pmax**=?[**F** s1=7 & s2=7 & d1+d2=k]

for k=2,...,12.
Performing this verification in PRISM we find that the probabilities for each k is as shown in the table and each calculation requires 29 iterations.
Again we can calculate the expected number of coin flips, in this case the minimum and maximum number of coin flips can be calculated through the following formulae:

Rmin=?[**F** s1=7 & s2=7] and **Rmax**=?[**F** s1=7 & s2=7]

Verifying these properties we find that the both the minimum and maximum number of coin flips to find the sum of two dice is 22/3 (two times the expected time to find the value of one dice).
We note that this approach is not optimal and [KY76] shows that the optimal program has an expected 79/18 coin flips and is realized by the following program:



The PRISM code for this program is given below. Note that verifying the formula **R**=?[**F** s=34] with PRISM does indeed show that the expected number of coin flips in this case is 79/18 (up to an accuracy of six decimal places), requiring 21 iterations.

```
// optimal program for the sum of two dice

dtmc

module sum_of_two_dice

	// local state
	s : [0..34] init 0;
	// total of two dice
	d : [0..12] init 0;
```

```
        [] s=0  -> 0.5 : (s'=1)  + 0.5 : (s'=2);
        [] s=1  -> 0.5 : (s'=3)  + 0.5 : (s'=4);
        [] s=2  -> 0.5 : (s'=5)  + 0.5 : (s'=6);
        [] s=3  -> 0.5 : (s'=7)  + 0.5 : (s'=34) & (d'=6);
        [] s=4  -> 0.5 : (s'=8)  + 0.5 : (s'=34) & (d'=7);
        [] s=5  -> 0.5 : (s'=9)  + 0.5 : (s'=34) & (d'=8);
        [] s=6  -> 0.5 : (s'=10) + 0.5 : (s'=11);
        [] s=7  -> 0.5 : (s'=12) + 0.5 : (s'=34) & (d'=4);
        [] s=8  -> 0.5 : (s'=13) + 0.5 : (s'=34) & (d'=5);
        [] s=9  -> 0.5 : (s'=14) + 0.5 : (s'=34) & (d'=9);
        [] s=10 -> 0.5 : (s'=15) + 0.5 : (s'=34) & (d'=10);
        [] s=11 -> 0.5 : (s'=16) + 0.5 : (s'=17);
        [] s=12 -> 0.5 : (s'=18) + 0.5 : (s'=34) & (d'=3);
        [] s=13 -> 0.5 : (s'=19) + 0.5 : (s'=34) & (d'=5);
        [] s=14 -> 0.5 : (s'=20) + 0.5 : (s'=34) & (d'=7);
        [] s=15 -> 0.5 : (s'=21) + 0.5 : (s'=34) & (d'=9);
        [] s=16 -> 0.5 : (s'=22) + 0.5 : (s'=34) & (d'=11);
        [] s=17 -> 0.5 : (s'=23) + 0.5 : (s'=24);
        [] s=18 -> 0.5 : (s'=25) + 0.5 : (s'=34) & (d'=2);
        [] s=19 -> 0.5 : (s'=26) + 0.5 : (s'=34) & (d'=3);
        [] s=20 -> 0.5 : (s'=27) + 0.5 : (s'=34) & (d'=4);
        [] s=21 -> 0.5 : (s'=34) & (d'=5) + 0.5 : (s'=34) & (d'=9);
        [] s=22 -> 0.5 : (s'=28) + 0.5 : (s'=34) & (d'=10);
        [] s=23 -> 0.5 : (s'=29) + 0.5 : (s'=34) & (d'=11);
        [] s=24 -> 0.5 : (s'=30) + 0.5 : (s'=34) & (d'=12);
        [] s=25 -> 0.5 : (s'=1)  + 0.5 : (s'=34) & (d'=2);
        [] s=26 -> 0.5 : (s'=31) + 0.5 : (s'=34) & (d'=3);
        [] s=27 -> 0.5 : (s'=32) + 0.5 : (s'=34) & (d'=6);
        [] s=28 -> 0.5 : (s'=34) & (d'=7) + 0.5 : (s'=34) & (d'=8);
        [] s=29 -> 0.5 : (s'=33) + 0.5 : (s'=34) & (d'=11);
        [] s=30 -> 0.5 : (s'=2)  + 0.5 : (s'=34) & (d'=12);
        [] s=31 -> 0.5 : (s'=34) & (d'=2) + 0.5 : (s'=34) & (d'=4);
        [] s=32 -> 0.5 : (s'=34) & (d'=6) + 0.5 : (s'=34) & (d'=8);
        [] s=33 -> 0.5 : (s'=34) & (d'=10) + 0.5 : (s'=34) & (d'=12);
        [] s=34 -> (s'=34);


endmodule


rewards "coin_flips"
```

```
        [] s<34 : 1;
endrewards
```

Python implementation and experiments

Value of a die

```python
def rollOneDie():
    state = 0
    die = 0
    while (state != 7) and (die == 0) :
        if state == 0 :
            state = np.random.choice([1, 2], p=[0.5, 0.5])
        if state == 1 :
            state = np.random.choice([3, 4], p=[0.5, 0.5])
        if state == 2 :
            state = np.random.choice([5, 6], p=[0.5, 0.5])
        if state == 3 :
            state = np.random.choice([1, 7], p=[0.5, 0.5])
            if state == 7:
                die = 1
        if state == 4 :
            state = 7
            die = np.random.choice([2, 3], p=[0.5, 0.5])
        if state == 5 :
            state = 7
            die = np.random.choice([4, 5], p=[0.5, 0.5])
        if state == 6 :
            state = np.random.choice([2, 7], p=[0.5, 0.5])
            if state == 2:
                die = 6
          if state == 7:
            break

    return die
```

Value of two dice

```python
def rollTwoDice():
    state = 0
    dice = 0
    while (state != 34) and (dice == 0):
        if state == 0:
            state = np.random.choice([1, 2], p=[0.5, 0.5])
        if state == 1:
            state = np.random.choice([3, 4], p=[0.5, 0.5])
        if state == 2:
            state = np.random.choice([5, 6], p=[0.5, 0.5])
        if state == 3:
            state = np.random.choice([7, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 6
        if state == 4:
            state = np.random.choice([8, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 7
        if state == 5:
            state = np.random.choice([9, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 8
        if state == 6:
            state = np.random.choice([10,11], p=[0.5, 0.5])
        if state == 7:
            state = np.random.choice([12, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 4
        if state == 8:
            state = np.random.choice([13, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 5
        if state == 9:
            state = np.random.choice([14, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 9
        if state == 10:
            state = np.random.choice([15, 34], p=[0.5, 0.5])
```

```
            if state == 34:
                dice = 10
        if state == 11:
            state = np.random.choice([16, 17], p=[0.5, 0.5])
        if state == 12:
            state = np.random.choice([18, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 3
        if state == 13:
            state = np.random.choice([19, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 5
        if state == 14:
            state = np.random.choice([20, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 7
        if state == 15:
            state = np.random.choice([21, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 9
        if state == 16:
            state = np.random.choice([22, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 11
        if state == 17:
            state = np.random.choice([23, 24], p=[0.5, 0.5])
        if state == 18:
            state = np.random.choice([34, 25], p=[0.5, 0.5])
            if state == 34:
                dice = 2
        if state == 19:
            state = np.random.choice([34, 26], p=[0.5, 0.5])
            if state == 34:
                die= 3
        if state == 20:
            state = np.random.choice([34, 27], p=[0.5, 0.5])
            if state == 34:
                dice = 4
        if state == 21:
            dice = np.random.choice([5, 9], p=[0.5, 0.5])
            state = 34
        if state == 22:
            state = np.random.choice([28, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 10
        if state == 23:
            state = np.random.choice([29, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 11
        if state == 24:
            state = np.random.choice([30, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 12
        if state == 25:
            state = np.random.choice([1, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 2
        if state == 26:
            state = np.random.choice([31, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 3
        if state == 27:
            state = np.random.choice([34, 32], p=[0.5, 0.5])
            if state == 34:
                dice = 6
        if state == 28:
            dice = np.random.choice([7, 8], p=[0.5, 0.5])
        if state == 29:
            state = np.random.choice([33, 34], p=[0.5, 0.5])
            if state == 34:
                dice = 11
        if state == 30:
            state = np.random.choice([34, 2], p=[0.5, 0.5])
            if state == 34:
                dice = 12
        if state == 31:
```

```
            dice = np.random.choice([2, 4], p=[0.5, 0.5])
        if state == 32:
            dice = np.random.choice([6, 8], p=[0.5, 0.5])
        if state == 33:
            dice = np.random.choice([10, 12], p=[0.5, 0.5])
            state = 34
        if state == 34:
            break
    return dice
```

Experiments

Property Details

Property:
P=? [F s=34&d=x]

Defined constants:
x=5

Method:
Verification

Result (probability):
0.111111082136631 (value in the initial state)

Okay

Property Details

Property:
P=? [F s=7&d=x]

Defined constants:
x=5

Method:
Verification

Result (probability):
0.16666650772094727 (value in the initial state)

Okay

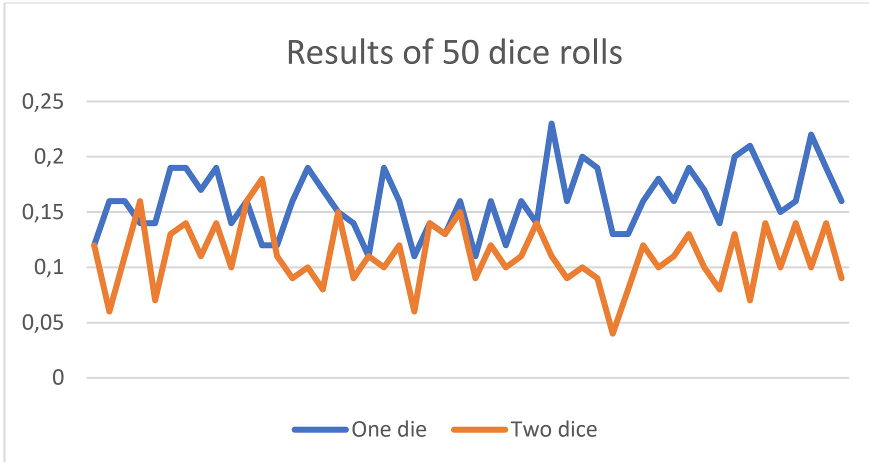
Step		sum_of_two_dice		Rewards
Action	#	s	d	["coin_flips"]
	0	0	0	1
sum_of_two_dice	1	1	0	1
sum_of_two_dice	2	4	0	1
sum_of_two_dice	3	8	0	1
sum_of_two_dice	4	13	0	1
sum_of_two_dice	5	34	5	0
sum_of_two_dice	6	34	5	?

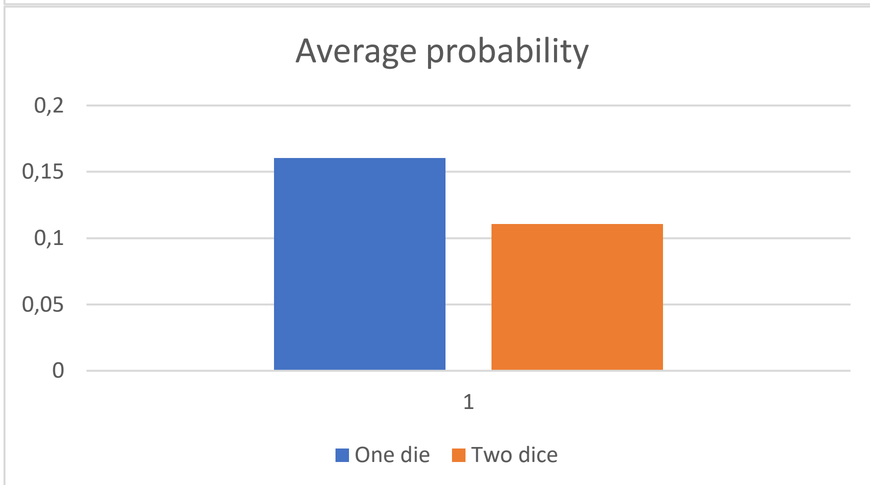
Python implementation for the probability of obtaining a certain number on the die:

```
def calculateProbability(x):
    arrayResultsOneDie = []
    arrayResultsTwoDice = []
    probOfNumber = 0
    probOfNumberTwo = 0
    for k in range(0,100):
        arrayResultsOneDie.append(rollOneDie())
        arrayResultsTwoDice.append(rollTwoDice())
    for i in range(0, len(arrayResultsOneDie)):
        if arrayResultsOneDie[i] == x:
            probOfNumber += 1
    for j in range(0, len(arrayResultsTwoDice)):
        if arrayResultsTwoDice[j] == x:
            probOfNumberTwo += 1
    print("The probability of getting", x," using one die
                                                is:",probOfNumber / len(arrayResultsOneDie))
    print("The probability of getting", x, " using two dice
                                             is:", probOfNumberTwo / len(arrayResultsTwoDice))

Results:
The probability of getting 5  using one die is: 0.16
```

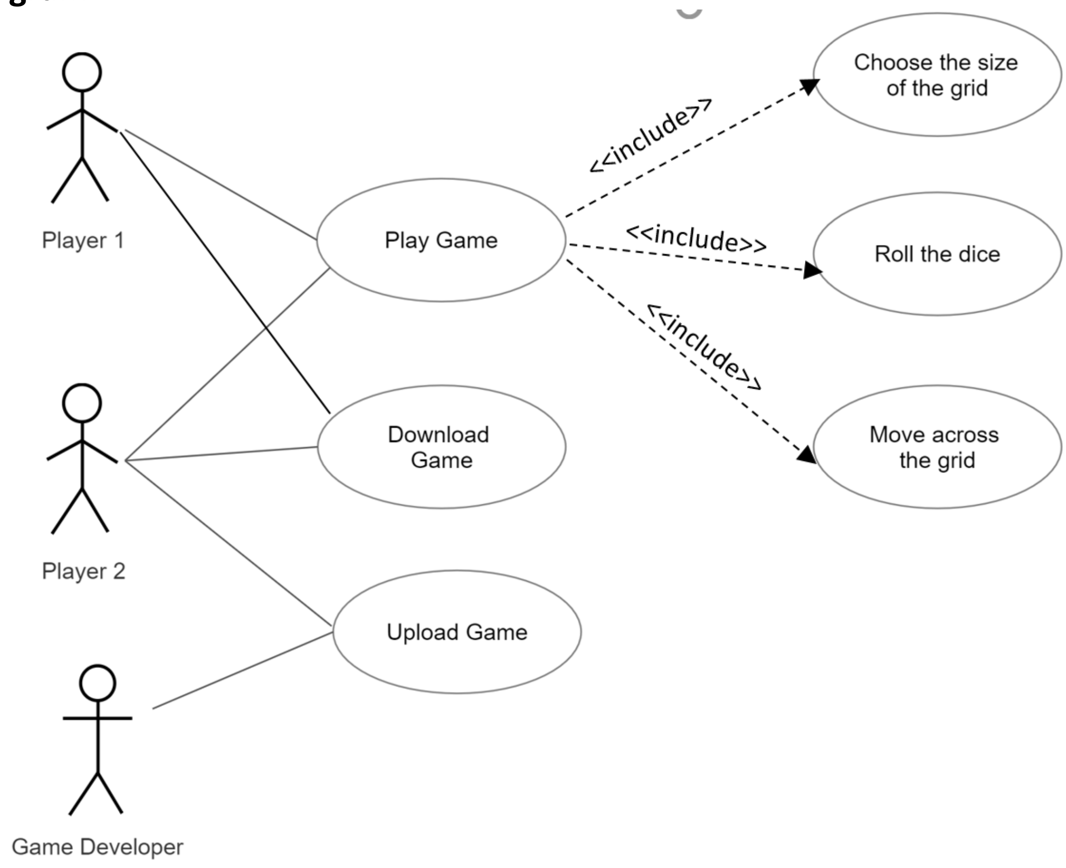
```
The probability of getting 5  using two dice is: 0.12
```

Of course, this might give different results every time we call the function, because it is not a straight-forward and easy way to compute the probability using dynamically and randomly results in Python. Therefore, we will run this multiple times and compute the average, in that way, the probability should be as accurate as possible.

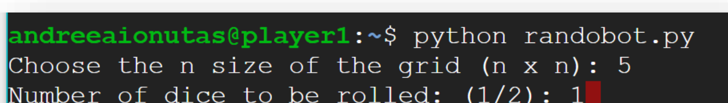




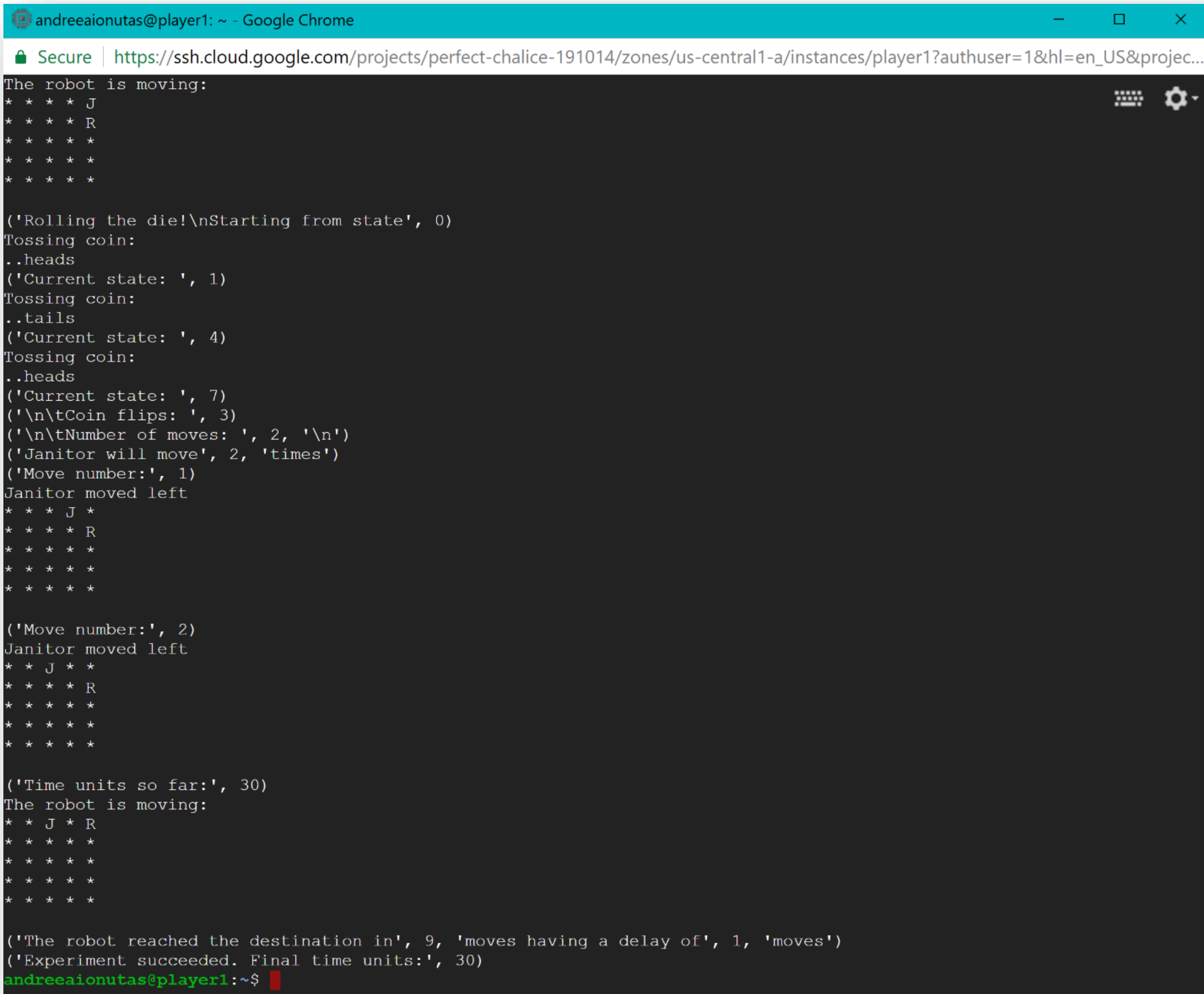
b. Use case diagram



Basic Scenario:
Actor (player) downloads the game from another player or from the Game Developer.
Player starts playing the game. He has to introduce the number of grids for the game and the number of dice to be rolled (1 or 2).



The game will set a goal to be reached in a number of steps and unit times and start to move the robot and the janitor according to the game rules.



3. Architecture and design

The design solution must be:

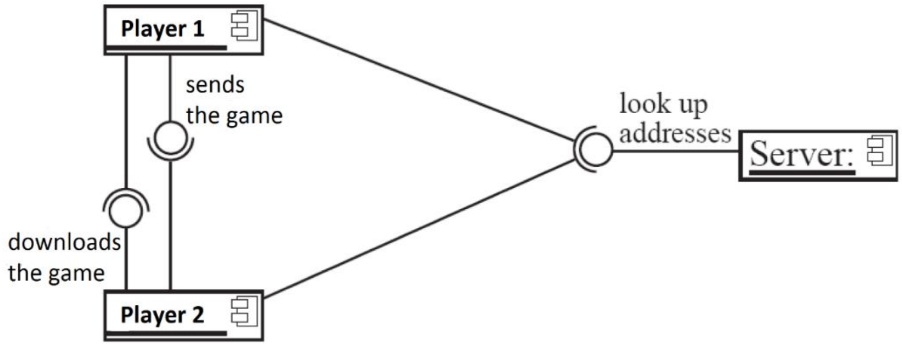
- ***validated***
- ***compared with competing solutions*** (other randomized distributed algorithms)

In this presentation we focus on designing reliable software components deployed on a computing architecture. We employ an approach based on formal analysis and design - PRISM model checker, process algebras www.prismmodelchecker.org:

- Implementation-independent design
- Model based performance analysis, reliability.

Develop functional models and quantitative aspects involving real-time and statistical issues. Implementation: Python.

Architectural patterns: Peer-to-Peer



4. PRISM model validation

RandoBot probabilities and statistics (Python implementation):

Number of runs	Grid size	Number of dice	Success probability	Average time units	Average coin flips
100	3x3	1	0.83	57	46
	3x3	2	0.96	89	63
	4x4	1	0.86	96	89
	4x4	2	0.90	134	116
	5x5	1	0.94	33	29
	5x5	2	0.99	66	37
	6x6	1	0.73	131	211
	6x6	2	0.86	263	276
1000	3x3	1	0.88	56	48
	3x3	2	0.94	111	61
	4x4	1	0.83	107	90
	4x4	2	0.91	212	116
	5x5	1	0.94	34	29
	5x5	2	0.98	67	36
	6x6	1	0.72	143	211
	6x6	2	0.82	291	276

Success probability:

We obtained these results by using the following python implementation: We create an array in which we store the results of the getProbability() function that returns 1 for success and 0 for failure. Then we count the number of 1's in the array and we divide by the number of runs (also the length of the array).

```
arrayProbability = []

for k in range (0, nrOfRuns):
    arrayProbability.append(getProbability(gridSize, nrOfDice))
iterator = 0
while(iterator<len(arrayProbability)):
    if arrayProbability[iterator] == 1 :
        successes += 1
    iterator += 1
```

```
rateOfSuccess = 100*successes/len(arrayProbability)
print(rateOfSuccess)
```

```
Results for 100 runs:                               Results for 1000 runs:
3x3 with 1 die:  0.83 prob. of success              3x3 with 1 die:  0.88 prob. of success
3x3 with 2 dice:  0.96 prob. of success             3x3 with 2 dice:  0.94 prob. of success
4x4 with 1 die:  0.86 prob. of success              4x4 with 1 die:  0.83 prob. of success
4x4 with 2 dice:  0.90 prob. of success             4x4 with 2 dice:  0.91 prob. of success
5x5 with 1 die:  0.94 prob. of success              5x5 with 1 die:  0.94 prob. of success
5x5 with 2 dice:  0.99 prob. of success             5x5 with 2 dice:  0.98 prob. of success
6x6 with 1 die:  0.73 prob. of success              6x6 with 1 die:  0.72 prob. of success
6x6 with 2 dice:  0.86 prob. of success             6x6 with 2 dice:  0.82 prob. of success
```

Average time units:

A time unit means an action taken by either the Janitor or the Robot.

We obtained the following results by storing in an array the time units returned by the getTimeUnits() functions, then summing them up and dividing by the number of runs which is in fact the length of the array. (Mean value)

```
arrayTimeUnits = []

for k in range (0, nrOfRuns):
    arrayTimeUnits.append(getTimeUnits(gridSize, nrOfDice))
iterator = 0
while(iterator<len(arrayTimeUnits)):
        timeUnitsSum += arrayTimeUnits[iterator]
    iterator += 1

averageTimeUnits = timeUnitsSum/len(arrayTimeUnits)
print(averageTimeUnits)
```

```
Results for 100 runs:                               Results for 1000 runs:
3x3 with 1 die:  57.73 av. time units               3x3 with 1 die:  56.876 av. time units
3x3 with 2 dice:  89.11 av. time units              3x3 with 2 dice:  111.157 av. time units
4x4 with 1 die:  96.75 av. time units               4x4 with 1 die:  107.588 av. time units
4x4 with 2 dice:  134.64 av. time units             4x4 with 2 dice:  212.679 av. time units
5x5 with 1 die:  33.98 av. time units               5x5 with 1 die:  34.336 av. time units
5x5 with 2 dice:  66.23 av. time units              5x5 with 2 dice:  67.444 av. time units
6x6 with 1 die:  131.79 av. time units              6x6 with 1 die:  143.16 av. time units
6x6 with 2 dice:  263.8 av. time units              6x6 with 2 dice:  291.455 av. time units
```

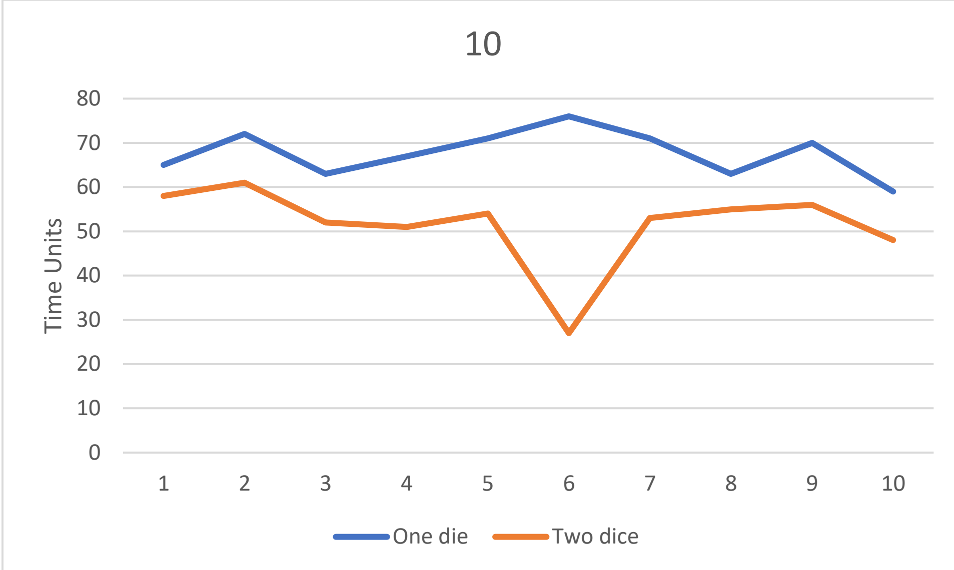
Average coin flips:

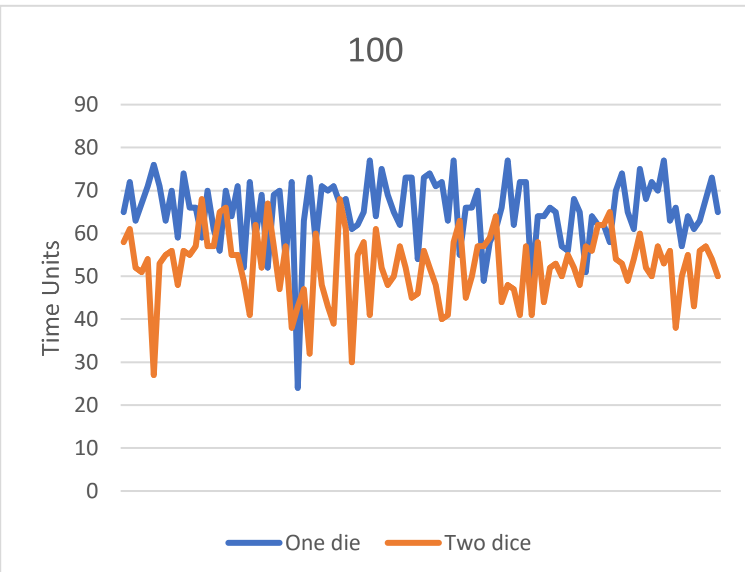
The number of coin flips represents the required number of state-changes to obtain a value on the die, such that the janitor will be able to take a number of actions. The results were obtained using the following implementation in python:

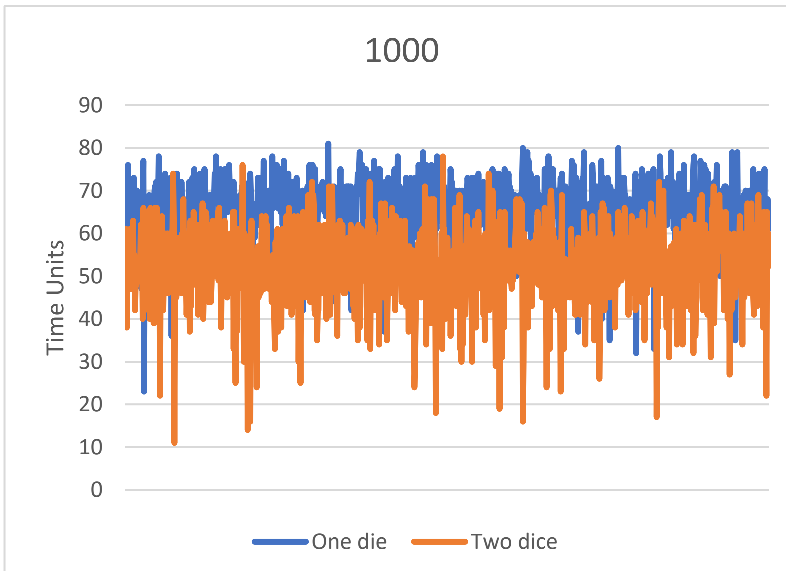
```
arrayCoinFlips = []

for k in range (0, nrOfRuns):
    arrayCoinFlips.append(getCoinFlips(gridSize, nrOfDice))
iterator = 0
while(iterator<len(arrayCoinFlips)):
        CoinFlipsSum += arrayCoinFlips[iterator]
    iterator += 1
averageCoinFlips = CoinFlipsSum/len(arrayCoinFlips)
print(averageCoinFlips)
```

```
Results for 100 runs:                               Results for 1000 runs:
3x3 with 1 die:  46.81 av. coin flips               3x3 with 1 die:  48.058 av. coin flips
3x3 with 2 dice:  63.02 av. coin flips              3x3 with 2 dice:  61.351 av. coin flips
4x4 with 1 die:  89.92 av. coin flips               4x4 with 1 die:  90.445 av. coin flips
4x4 with 2 dice:  116.62 av. coin flips             4x4 with 2 dice:  116.091 av. coin flips
5x5 with 1 die:  29.31 av. coin flips               5x5 with 1 die:  29.021 av. coin flips
5x5 with 2 dice:  37.44 av. coin flips              5x5 with 2 dice:  36.725 av. coin flips
6x6 with 1 die:  211.78 av. coin flips              6x6 with 1 die:  211.68 av. coin flips
6x6 with 2 dice:  276.38 av. coin flips             6x6 with 2 dice:  276.696 av. coin flips
```

The following graph show the difference obtained by repeatedly playing the game on a 5x5 grid world, by using one die and two dice, analyzing the time units required for the robot to reach the destination.

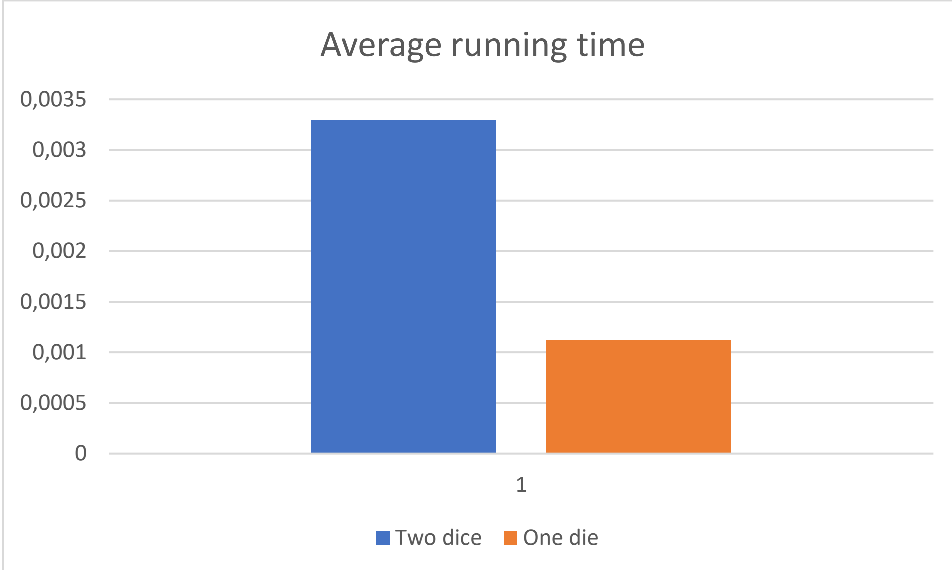


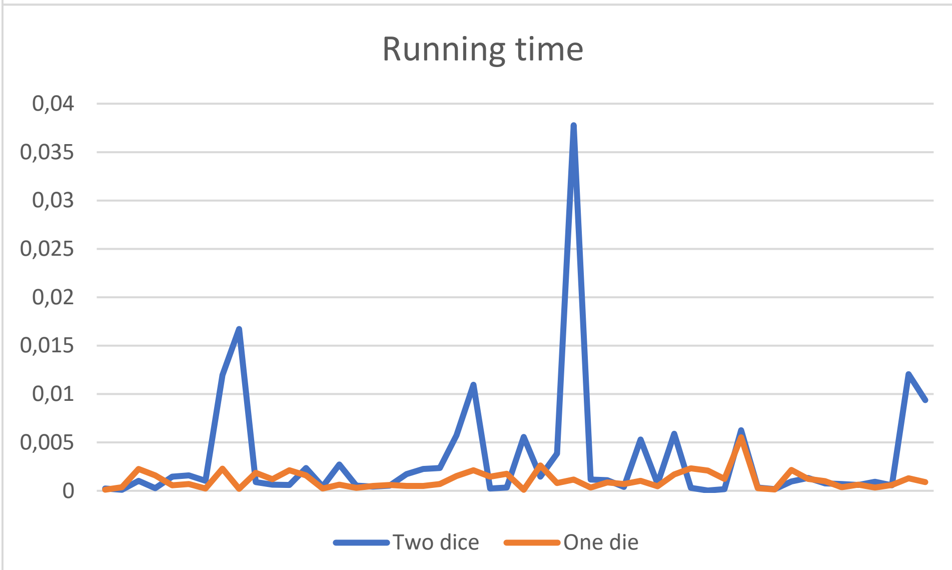




RandoBot performance tests:

We computed the performance difference between using one die and two dices, based on the required running time by their implementation in python.





5. Implementation

RandoBot.py

```
import numpy as np
import time as tm

robotPos = []

def rollOneDie():
    state = 0
    flips = 0
    die = 0
    print("Rolling the die!\nStarting from state", state)
    print("Tossing coin: ")
 #   tm.sleep(0.3)
    state = np.random.choice([1, 2], p=[0.5, 0.5])
    if state == 1:
        print("..heads")
    elif state == 2:
        print("..tails")
    flips += 1
    print("Current state: ", state)
    while (state != 7) and (die == 0) :
        if state == 1:
```

```
            flips += 1
    #        tm.sleep(0.3)
            print("Tossing coin:")
     #       tm.sleep(0.3)
            state = np.random.choice([3, 4], p=[0.5, 0.5])
            if state == 3 :
                print("..heads")
            elif state == 4 :
                print("..tails")
            print("Current state: ", state)
        if state == 2 :
            flips += 1
      #      tm.sleep(0.3)
            print("Tossing coin:")
       #     tm.sleep(0.3)
            state = np.random.choice([5, 6], p=[0.5, 0.5])
            if state == 5 :
                print("..heads")
            elif state == 6 :
                print("..tails")
            print("Current state: ", state)
        if state == 3 :
            flips += 1
         #   tm.sleep(0.3)
            print("Tossing coin:")
        #    tm.sleep(0.3)
            state = np.random.choice([1, 7], p=[0.5, 0.5])
            if state == 1:
                print("..heads")
            elif state == 7:
                print("..tails")
                die = 1
            print("Current state: ", state)
        if state == 4 :
            flips += 1
            state = 7
         #   tm.sleep(0.3)
            print("Tossing coin:")
         #   tm.sleep(0.3)
            die = np.random.choice([2, 3], p=[0.5, 0.5])
            if die == 2:
                print("..heads")
            elif die == 3:
                print("..tails")
            print("Current state: ", state)
        if state == 5 :
            flips += 1
            state = 7
         #   tm.sleep(0.3)
            print("Tossing coin:")
        #    tm.sleep(0.3)
            die = np.random.choice([4, 5], p=[0.5, 0.5])
            if die == 4:
                print("..heads")
            elif die == 5:
                print("..tails")
            print("Current state: ", state)
        if state == 6 :
            flips += 1
         #   tm.sleep(0.3)
            print("Tossing coin:")
         #   tm.sleep(0.3)
            state = np.random.choice([2, 7], p=[0.5, 0.5])
            if state == 2:
                print("..heads")
                die = 6
            elif state == 7:
                print("..tails")
```

```
            print("Current state: ", state)
    if 0<die<7 :
        print("\n\tCoin flips: ", flips)
        print("\n\tNumber of moves: ", die,"\n")
        return die
    else :
        return 1


def rollTwoDice():
    state = 0
    flips = 0
    dice = 0
    print("Rolling the dice!\nStarting from state", state)
    while (state != 34) and (dice == 0):
        if state == 0:
            print("Tossing coin:")
            state = np.random.choice([1, 2], p=[0.5, 0.5])
            if state == 1 :
                print("..heads")
            elif state == 2 :
                print("..tails")
            print("Current state: ", state)
            flips += 1
        if state == 1:
            print("Tossing coin:")
            state = np.random.choice([3, 4], p=[0.5, 0.5])
            if state == 3 :
                print("..heads")
            elif state == 4 :
                print("..tails")
            print("Current state: ", state)
            flips += 1
        if state == 2:
            print("Tossing coin:")
            state = np.random.choice([5, 6], p=[0.5, 0.5])
            flips += 1
            if state == 5 :
                print("..heads")
            elif state == 6 :
                print("..tails")
            print("Current state: ", state)
        if state == 3:
            print("Tossing coin:")
            state = np.random.choice([7, 34], p=[0.5, 0.5])
            flips += 1
            if state == 7:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 6
            print("Current state: ", state)
        if state == 4:
            print("Tossing coin:")
            state = np.random.choice([8, 34], p=[0.5, 0.5])
            flips += 1
            if state == 8:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 7
        if state == 5:
            print("Tossing coin:")
            state = np.random.choice([9, 34], p=[0.5, 0.5])
            flips += 1
            if state == 9:
                print("..heads")
            elif state == 34:
```

```
                print("..tails")
                dice = 8
            print("Current state: ", state)
        if state == 6:
            print("Tossing coin:")
            state = np.random.choice([10,11], p=[0.5, 0.5])
            flips += 1
            if state == 10 :
                print("..heads")
            elif state == 11 :
                print("..tails")
            print("Current state: ", state)
        if state == 7:
            print("Tossing coin:")
            state = np.random.choice([12, 34], p=[0.5, 0.5])
            flips += 1
            if state == 12:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 4
            print("Current state: ", state)
        if state == 8:
            print("Tossing coin:")
            state = np.random.choice([13, 34], p=[0.5, 0.5])
            flips += 1
            if state == 13:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 5
            print("Current state: ", state)
        if state == 9:
            print("Tossing coin:")
            state = np.random.choice([14, 34], p=[0.5, 0.5])
            flips += 1
            if state == 14:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 9
            print("Current state: ", state)
        if state == 10:
            print("Tossing coin:")
            state = np.random.choice([15, 34], p=[0.5, 0.5])
            flips += 1
            if state == 15:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 10
            print("Current state: ", state)
        if state == 11:
            print("Tossing coin:")
            state = np.random.choice([16, 17], p=[0.5, 0.5])
            if state == 16 :
                print("..heads")
            elif state == 17 :
                print("..tails")
            print("Current state: ", state)
            flips += 1
        if state == 12:
            print("Tossing coin:")
            state = np.random.choice([18, 34], p=[0.5, 0.5])
            flips += 1
            if state == 18:
                print("..heads")
            elif state == 34:
```

```
                print("..tails")
                dice = 3
            print("Current state: ", state)
        if state == 13:
            print("Tossing coin:")
            state = np.random.choice([19, 34], p=[0.5, 0.5])
            flips += 1
            if state == 19:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 5
            print("Current state: ", state)
        if state == 14:
            print("Tossing coin:")
            state = np.random.choice([20, 34], p=[0.5, 0.5])
            flips += 1
            if state == 20:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 7
            print("Current state: ", state)
        if state == 15:
            print("Tossing coin:")
            state = np.random.choice([21, 34], p=[0.5, 0.5])
            flips += 1
            if state == 21:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 9
            print("Current state: ", state)
        if state == 16:
            print("Tossing coin:")
            state = np.random.choice([22, 34], p=[0.5, 0.5])
            flips += 1
            if state == 22:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 11
            print("Current state: ", state)
        if state == 17:
            print("Tossing coin:")
            state = np.random.choice([23, 24], p=[0.5, 0.5])
            if state == 23 :
                print("..heads")
            elif state == 24 :
                print("..tails")
            print("Current state: ", state)
            flips += 1
        if state == 18:
            print("Tossing coin:")
            state = np.random.choice([34, 25], p=[0.5, 0.5])
            flips += 1
            if state == 34:
                print("..heads")
                dice = 2
            elif state == 25:
                print("..tails")
            print("Current state: ", state)
        if state == 19:
            print("Tossing coin:")
            state = np.random.choice([34, 26], p=[0.5, 0.5])
            flips += 1
            if state == 34:
                print("..heads")
```

```
                dice = 3
            elif state == 26:
                print("..tails")
            print("Current state: ", state)
        if state == 20:
            print("Tossing coin:")
            state = np.random.choice([34, 27], p=[0.5, 0.5])
            flips += 1
            if state == 34:
                print("..heads")
                dice = 4
            elif state == 27:
                print("..tails")
            print("Current state: ", state)
        if state == 21:
            print("Tossing coin:")
            dice = np.random.choice([5, 9], p=[0.5, 0.5])
            flips += 1
            if dice == 5:
                print("..heads")
            elif dice == 9:
                print("..tails")
            state = 34
            print("Current state: ", state)
        if state == 22:
            print("Tossing coin:")
            state = np.random.choice([28, 34], p=[0.5, 0.5])
            flips += 1
            if state == 28:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 10
            print("Current state: ", state)
        if state == 23:
            print("Tossing coin:")
            state = np.random.choice([29, 34], p=[0.5, 0.5])
            flips += 1
            if state == 29:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 11
            print("Current state: ", state)
        if state == 24:
            print("Tossing coin:")
            state = np.random.choice([30, 34], p=[0.5, 0.5])
            flips += 1
            if state == 30:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 12
            print("Current state: ", state)
        if state == 25:
            print("Tossing coin:")
            state = np.random.choice([1, 34], p=[0.5, 0.5])
            flips += 1
            if state == 1:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 2
            print("Current state: ", state)
        if state == 26:
            print("Tossing coin:")
            state = np.random.choice([31, 34], p=[0.5, 0.5])
            flips += 1
```

```
            if state == 31:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 3
            print("Current state: ", state)
        if state == 27:
            print("Tossing coin:")
            state = np.random.choice([34, 32], p=[0.5, 0.5])
            flips += 1
            if state == 34:
                print("..heads")
                dice = 6
            elif state == 32:
                print("..tails")
            print("Current state: ", state)
        if state == 28:
            print("Tossing coin:")
            dice = np.random.choice([7, 8], p=[0.5, 0.5])
            flips += 1
            if dice == 7:
                print("..heads")
            elif dice == 8:
                print("..tails")
            state = 34
            print("Current state: ", state)
        if state == 29:
            print("Tossing coin:")
            state = np.random.choice([33, 34], p=[0.5, 0.5])
            flips += 1
            if state == 33:
                print("..heads")
            elif state == 34:
                print("..tails")
                dice = 11
            print("Current state: ", state)
        if state == 30:
            print("Tossing coin:")
            state = np.random.choice([34, 2], p=[0.5, 0.5])
            flips += 1
            if state == 34:
                print("..heads")
                dice = 12
            elif state == 2:
                print("..tails")
            print("Current state: ", state)
        if state == 31:
            print("Tossing coin:")
            dice = np.random.choice([2, 4], p=[0.5, 0.5])
            state = 34
            flips += 1
            if dice == 2:
                print("..heads")
            elif dice == 4:
                print("..tails")
            print("Current state: ", state)
        if state == 32:
            print("Tossing coin:")
            dice = np.random.choice([6, 8], p=[0.5, 0.5])
            flips += 1
            if dice == 6:
                print("..heads")
            elif dice == 8:
                print("..tails")
            state = 34
            print("Current state: ", state)
        if state == 33:
            print("Tossing coin:")
```

```
            dice = np.random.choice([10, 12], p=[0.5, 0.5])
            flips += 1
            if dice == 10:
                print("..heads")
            elif dice == 12:
                print("..tails")
            state = 34
            print("Current state: ", state)
        if state == 34:
            break
    if 0<dice<13 :
        print("\n\tCoin flips: ", flips)
        print("\n\tNumber of moves: ", dice,"\n")
        return dice
    else :
        return 1

def print_matrix(xJ, yJ, xR, yR):
    for i in range(n):
        for j in range(n):
            if i == xR and j == yR:
                Matrix[i][j] = "R"
            elif i == xJ and j == yJ:
                Matrix[i][j] = "J"
            else:
                Matrix[i][j] = "*"
    for row in Matrix:
        print(' '.join([str(elem) for elem in row]))
    print('')


try:
    n = int(input("Choose the n size of the grid (n x n): "))
    nrOfDice = int(input("Number of dice to be rolled: (1/2): "))
    if n==5 :
        maxTimeUnits = 100
    else :
        maxTimeUnits = n*n + (79/18)*n*4
    print("The goal is to reach the destination with maximum", (2*n), "moves in maximum",
maxTimeUnits, "time units.")
    tm.sleep(4)
    Matrix = [[0] * n for i in range(n)]
    for x in range(n):
        robotPos.append((n-1, x))
    for y in range(n-2, -1, -1):
        robotPos.append((y, n-1))
    v1 = 0
    deniedMoves = 0
    timeUnits = 0
    xR, yR = robotPos[v1][0], robotPos[v1][1]
    xJ, yJ = 0, n-1
    print_matrix(xJ, yJ, xR, yR)
    while not(xR == 0 and yR == n-1):
        if nrOfDice == 1:
            numberOfMoves = rollOneDie()
        elif nrOfDice == 2:
            numberOfMoves = rollTwoDice()
        else:
            print("Invalid number of dice")
            exit(3)
        print("Janitor will move", numberOfMoves, "times")
        for i in range(0, numberOfMoves):
            timeUnits += 1
            print("Move number:",i+1)
            randomPos = np.random.choice([0, 1, 2, 3], p=[0.25, 0.25, 0.25, 0.25])
            if randomPos == 0:  # left y--
                if (yJ - 1) < 0:
                    print("Janitor cannot move left, (Out of boundary)")
```

```
                elif (yJ - 1 == yR) and (xJ == xR):
                    print("Janitor cannot move left, (Occupied by Robot)")
                else:
                    yJ = yJ - 1
                    print("Janitor moved left")
                    print_matrix(xJ, yJ, xR, yR)
            if randomPos == 1:  # up x--
                if (xJ - 1) < 0:
                    print("Janitor cannot move up, (Out of boundary)")
                elif (xJ - 1 == xR) and (yJ == yR):
                    print("Janitor cannot move up, (Occupied by Robot)")
                else:
                    xJ = xJ - 1
                    print("Janitor moved up")
                    print_matrix(xJ, yJ, xR, yR)
            if randomPos == 2:  # right y++
                if (yJ + 1) >= n:
                    print("Janitor cannot move right, (Out of boundary)")
                elif (yJ + 1 == yR) and (xJ == xR):
                    print("Janitor cannot move right, (Occupied by Robot)")
                else:
                    yJ = yJ + 1
                    print("Janitor moved right")
                    print_matrix(xJ, yJ, xR, yR)
            if randomPos == 3:  # down x++
                if (xJ + 1) >= n:
                    print("Janitor cannot move down, (Out of boundary)")
                elif (xJ + 1 == xR) and (yJ == yR):
                    print("Janitor cannot move down, (Occupied by Robot)")
                else:
                    xJ = xJ + 1
                    print("Janitor moved down")
                    print_matrix(xJ, yJ, xR, yR)
        v1 += 1
        if xJ == robotPos[v1][0] and yJ == robotPos[v1][1]:
            v1 -= 1
            print("Robot cannot move, (Occupied by Janitor)")
            deniedMoves += 1
        xR, yR = robotPos[v1][0], robotPos[v1][1]
        timeUnits += 1
        print("Time units so far:", timeUnits)
        print("The robot is moving: ")
        print_matrix(xJ, yJ, xR, yR)
        if xR == 0 and yR == (n-1) :
            print("The robot reached the destination in", v1+deniedMoves, "moves having a delay
of", deniedMoves,"moves")
            if (maxTimeUnits-timeUnits) < 0 :
                print("and", maxTimeUnits-timeUnits, "time units")
            if maxTimeUnits-timeUnits < 0 :
                print("Experiment failed. Timelimit exceeded")
            else :
                print("Experiment succeeded. Final time units:", timeUnits)
except ValueError:
    print("Not a number")
```

Client.py

```
import socket
import sys
import threading
import pickle

try:
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error, msg:
   print 'Failed to create socket. Error code: ' + str(msg[0]) + ' ,Error message: '+ msg[1]
   sys.exit()
```

```
sListen=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sListen.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
portL = 8888

try:
   sListen.bind(('',portL))
except socket.error, msg:
   print 'Bind failed. Error Code: ' + str(msg[0]) + ' Message: ' + msg[1]
   sys.exit()
sListen.listen(10)
print "Socket now listening"


def client(host, port, s, portL):
   try:
      remote_ip = socket.gethostbyname(host)
   except socket.gaierror:
      print 'Hostname couldn\'t be resolved. Exiting'
      sys.exit()

   s.connect((remote_ip, port))
   print 'Socket connected to ' + host + ' on ip ' + remote_ip
   reply = s.recv(4096)
   print reply

   handleInput()

   s.close()


#Handle user input
def handleInput():
   quit = False
   while not quit:
      input = raw_input(">> ")
      if not input:
         continue
      elif input[0] is 'U':
         fileName = raw_input('Enter file name: ')
         filePath = raw_input('Enter path: ')
         message = 'SHARE_FILES\n'+fileName+' '+filePath

      elif input[0] is 'R':
         nickname = raw_input('Enter a nickname: ')
         message = 'REGISTER\n'+nickname

      elif input[0] is 'S':
         fileName = raw_input('Enter file name to be searched: ')
         message = 'SEARCH\n'+fileName
         try:
            s.sendall(message)
         except socket.error:
            print 'Send failed'
            sys.exit()
         reply = s.recv(4096)
         if reply.split('\n')[0] == 'ERROR':
            print reply.split('\n')[1]
            sys.exit()

         usersHavingFile = eval(reply)
         if not usersHavingFile:
            print 'File not found'
            continue

         message = 'The following users have the file:\n'
         for user in usersHavingFile.keys():
            message = message + usersHavingFile[user]['nick'] + ' (' + user + ') (' +
usersHavingFile[user]['filePath'] + ')\n'
```

```
         print message

         tryDownload(fileName, usersHavingFile)
         continue

      elif input is 'E':
         quit = True
         continue

      else:
         print 'Unknown command'
         continue

      try:
         s.sendall(message)
      except socket.error:
         print 'Send failed'
         sys.exit()

      reply = s.recv(4096)
      print reply

def tryDownload(fileName, usersHavingFile):
   response = raw_input('Write \"Q\" followed by the client IP for downloading file from that client\n')
   response = response.strip()

   if response[0] == 'Q':
      s1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      peerIP = response.split(' ')[1]
      s1.connect((peerIP, portL))
      queryMessage = 'DOWNLOAD\n' + fileName + '\n' + usersHavingFile[peerIP]['filePath']
      try:
         s1.sendall(queryMessage)
      except socket.error:
         print 'Send failed'
         sys.exit()
      fw = open('Received'+fileName,'wb+')
      flag = 0
      chunk = s1.recv(100)

      while chunk.strip() != 'SHUT_WR':
         s1.send('received')
         if chunk.split('\n')[0] == 'ERROR':
            print chunk.split('\n')[0]+' '+chunk.split('\n')[1]
            flag = 1
            break
         fw.write(chunk)
         chunk = s1.recv(100)
      if flag != 1:
         print "\nFile saved in current folder"
      fw.close()
      s1.close()

def listenForSharing(sListen):

   while 1:
      conn, addr = sListen.accept()
      data = conn.recv(1024)
      if data.split('\n')[0]=='DOWNLOAD':
         fileName = data.split('\n')[1]
         filePath = data.split('\n')[2]
         print filePath+fileName
         try:
            fr = open(filePath+fileName,'rb')
         except:
            conn.sendall('ERROR\nNo such file available')
            continue
```

```
            chunk = fr.read()
            conn.send(chunk)
            ack = conn.recv(100)
            conn.sendall('SHUT_WR')
            fr.close()
    sListen.close()


try:
    host = sys.argv[1]
    port = int(sys.argv[2])
    print host
    print port

    if __name__ == '__main__':
        t = threading.Thread(target = client, args=(host,port,s,portL))
        t.start()
        t2 = threading.Thread(target = listenForSharing, args = (sListen,))
        t2.daemon = True
        t2.start()
except:
    sListen.close()
```

Server.py

```
import socket
import sys
from thread import *
import functions

host = '10.132.0.2'
port = 55555

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
print 'Socket created'
try:
    s.bind((host,port))
except socket.error, msg:
    print 'Bind failed. Error Code: ' + str(msg[0]) + ' Message: ' + msg[1]
    sys.exit()

print 'Socket bind complete'

s.listen(10)
print 'Socket now listening'
activePeers = []
users = {}
def clientthread(conn,addr):
    conn.send('Commands:\n 1. R - register with an username to the server\n 2. U - upload a file to
the server\n \
        3. S - search for the given file. Check if any peer has it\n 4. E - exit peer client')
    activePeers.append(addr[0])
    while 1:
        data = conn.recv(1024)
        if not data:
            break

        if data.split('\n')[0] == 'REGISTER':
            functions.register(conn, addr, data.split('\n')[1])
        elif data.split('\n')[0] == 'SHARE_FILES':
            functions.share(conn,addr,data.split('\n')[1])
        elif data.split('\n')[0] == 'SEARCH':
            functions.search(conn,addr,data.split('\n')[1],activePeers)

    activePeers.remove(addr[0])
    conn.close()
```

```
while 1:
   conn, addr = s.accept()
   print 'Connected with ' + addr[0] + ':' + str(addr[1])

   start_new_thread(clientthread, (conn,addr))


s.close()
```

Functions.py

```
import pickle

def register(conn, addr, nick):
   try:
      users = pickle.load(open("users","rb"))
   except:
      users = {}
      pickle.dump(users,open("users","wb"))
   try:
      nickname = users[addr[0]]['nick']
      conn.sendall('User already registered with nickname '+nickname)
   except:
      users[addr[0]] = {}
      users[addr[0]]['nick'] = nick
      users[addr[0]]['fileList'] = {}
      conn.sendall('You have been registered with nickname '+nick)

   pickle.dump(users,open("users","wb"))


def share(conn, addr, file):
   try:
      users = pickle.load(open("users","rb"))
   except:
      conn.sendall('You need to register first')
      return
   try:
      nickname = users[addr[0]]['nick']
   except:
      conn.sendall('You need to register first')
      return

   fileName = file.split(' ')[0]
   filePath = file.split(' ')[1]
   users[addr[0]]['fileList'][fileName] = filePath
   pickle.dump(users,open("users","wb"))
   conn.sendall('File '+fileName+' added')



def search(conn, addr, fileName, activePeers):
   try:
      users = pickle.load(open("users","rb"))
   except:
      conn.sendall('ERROR\nNo users registered till now')
      return

   usersHavingFile = {}
   userList = users.keys()
   for user in userList:
      found = False
      if fileName in users[user]['fileList'].keys():
         if user in activePeers:
            usersHavingFile[user] = {}
            usersHavingFile[user]['nick'] = users[user]['nick']
```

```
            usersHavingFile[user]['filePath'] = users[user]['fileList'][fileName]
   conn.sendall(str(usersHavingFile))


def checkDB(conn):
   try:
      users = pickle.load(open("users","rb"))
      conn.sendall(str(users))
   except:
      conn.sendall('File doesn\'t exist')
```