Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

# Лабораторная работа №4
# по дисциплине: «Теория Алгоритмов»
# Тема: *«Бинарные деревья и графы»*

Выполнила ученица 223 группы
Никончук А.П.

Проверил
Сорокин Д.С. _____

Санкт-Петербург,
2015 г.

Задание 1: Необходимо написать функцию, которая создает игральную кость с указанным числом сторон

```
from random import randint

def make_fair_dice(sides):
    assert type(sides) == int and sides >= 1, 'Illegal value for sides'
    def dice():
        return randint(1, sides)
    return dice
```

```
$ python lab_5.py
6
4
1
```

Задание 2: нужно написать функцию roll_dice(), которая позволит бросать кубик с указанным числом сторон нужное число раз. Функция возвращает сумму очков после num_rolls бросков, либо одно очко, если хотя бы в одном из бросков выпала 1

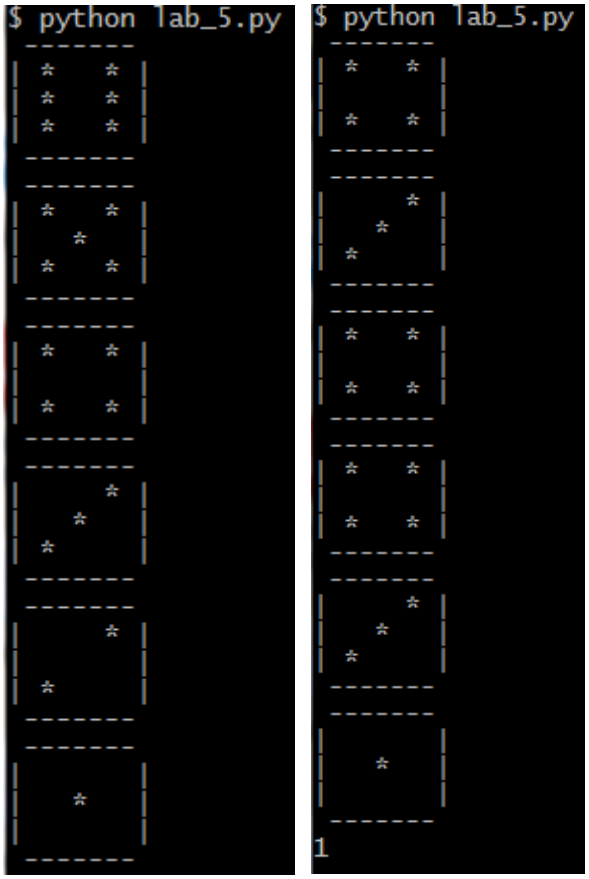
```
def roll_dice(num_rolls, dice=six_sided_dice, who='Grand Jedi Master Yoda'):
    assert type(num_rolls) == int, 'num_rolls must be an integer.'
    assert num_rolls > 0, 'Must roll at least once.'
    score = 0
    while num_rolls!=0:
        b=dice()
        score += b
        if b==1:
            return 1
        num_rolls -= 1
    return score
```

```
$ python lab_5.py
24
```

```
$ python lab_5.py
1
```

Задание 2.5: Напишите функцию draw_number(), используя функцию draw_dice() и добавьте ее в реализацию roll_dice(), чтобы видеть сколько очков выпадает после каждого броска

```
def draw_number(n, dot='*'):
    if n==1:
        return draw_dice(1,0,0,0)
    if n==2:
        return draw_dice(0,1,0,0)
    if n==3:
        return draw_dice(1,1,0,0)
    if n==4:
        return draw_dice(0,1,1,0)
    if n==5:
        return draw_dice(1,1,1,0)
    if n==6:
        return draw_dice(0,1,1,1)
```




```
def roll_dice(num_rolls, dice=six_sided_dice, who='Grand Jedi Master Yoda'):
    assert type(num_rolls) == int, 'num_rolls must be an integer.'
    assert num_rolls > 0, 'Must roll at least once.'
    score = 0
    while num_rolls!=0:
        b=dice()
        print(draw_number(b))
        score += b
        if b==1:
            return 1
        num_rolls -= 1
    return score
```

Задание 3: Напишите функцию take_turn(), которая моделируют ход игрока. В реализации этой функции используйте функцию roll_dice()

```
def FreeBacon(num_rolls, opponent_score):
    if num_rolls==0:
        return int(opponent_score/10)+1
    return False

#print(FreeBacon(0,32))

def Touchdown(score):
    if score%6==0:
        return int(score+(score/6))
    return False

#print(Touchdown(12,6))

def is_prime(n):
    k = 2
    while k < n:
        if n % k == 0:
            return False
        k += 1
    return True

def next_prime(n):
    n=n+1
    while is_prime(n)!=True:
        n+=1
    return n

#print(next_prime(7))

def HogtimusPrime(score):
    if is_prime(score)==True:
        return next_prime(score)
    return False

#print(HogtimusPrime(10))

def take_turn(num_rolls, score, opponent_score, dice=six_sided_dice):
    assert type(num_rolls) == int, 'num_rolls must be an integer.'
    assert num_rolls >= 0, 'Cannot roll a negative number of dice.'
    if commentary:
        print(who, 'is going to roll', num_rolls, 'dice','score',dice())
    score1=roll_dice(num_rolls) + score
    FScore=FreeBacon(num_rolls,opponent_score)+Touchdown(score1)+HogtimusPrime(score1)
    if FScore==0:
        return score1
    return FScore
```

```
$ python lab_5.py
 -------
| *   * |
|       |
| *   * |
 -------
 -------
|     * |
|       |
| *     |
 -------
 -------
| *   * |
|   *   |
| *   * |
 -------
 -------
|       |
|   *   |
|       |
 -------
3
```

Задание 4: Напишите функции num_allowed_dice() и select_dice(), которые упростят реализацию функции play(). Функция num_allowed_dice()является реализацией правила №1 (Hog Tied), функция select_dice() является реализацией правила №2 (Hog Wild). Обе функции принимают по два аргумента: очки игрока и очки соперника

```python
def HogTied(score,opponent_score):
    if (score+opponent_score)%10 == 7:
        return True
    return False

#print(HogTied(10,7))

def num_allowed_dice(score, opponent_score):
    if HogTied(score, opponent_score):
        return 1
    return 10

#print(num_allowed_dice(10,7))

def HogWild(score, opponent_score, n=7):
    if (score+opponent_score)%n==0:
        return True
    return False

##print(HogWild(11,10))

def select_dice(score, opponent_score):
    if HogWild(score,opponent_score):
        return 'four_sided_dice'
    return 'six_sided_dice'

#print(select_dice(21,8))
```

```
$ python lab_5.py
1
six_sided_dice
```

Задание 5: Напишите функцию play(). Игра проходит в поочередной смене ходов. Каждый из игроков использует свою стратегию (аргументы strategy0для 1-го игрока и strategy1 для 2-го игрока). Игра продолжается до тех пор, пока один из игроков не достигнет goal очков. Когда игра заканчивается функция play() возвращает 0, если победил первый игрок, 1, если победил второй игрок

```
def play(strategy0=always_roll(1), strategy1=make_comeback_strategy(), goal=100):
    score0, score1 = 0, 0
    print('who: 0')
    print(score0)
    print('who: 1')
    print(score1)
    while score0<=goal or score1<=goal:
        score0+=take_turn(5,score0,score1)
        print('who: 0')
        yield score0
        if score0>=100:
            print('0 is winner. score:',score0)
            return False
        print('who: 1')
        score1+=take_turn(strategy1(score1, score0),score1,score0)
        yield score1
        if score1>=100:
            print('1 is winner. score:',score1)
            return False

def pl():
    p = play()
    while p!=False:
        print(next(p))

pl()
```

```
$ python lab_5.py
who: 0
0
who: 1
0
who: 0
29
who: 1
2
who: 0
64
who: 1
31
who: 0
129
0 is winner. score: 129
```

Задача 6: Напишите функцию make_comeback_strategy(), которая возвращает следующую стратегию: если игрок проигрывает своему оппоненту не более margin очков, то он делает num_rolls + 1 бросков,
иначе num_rolls бросков. Другими словами, если мы начинаем отставать от нашего соперника более чем на margin очков, то идем на риск и делаем на один бросок больше, при этом увеличивается вероятность выпадения кубика с 1

```
def make_comeback_strategy(margin=10, num_rolls=5):
    def mcs(score, opponent_score):
        if opponent_score-score>=margin:
            return num_rolls+1
        return num_rolls
    return mcs

strategy = make_comeback_strategy()
print(strategy(3,14))
```

```
$ python lab_5.py
6
```

Задача 7: Напишите функцию make_mean_strategy(), которая возвращает следующую стратегию: игрок совершает 0 бросков, если выполняются следующие условия:

- По правилу Free Bacon мы получим не менее чем min_points очков.
- Получив min_points сумма наших очков и очков оппонента будет множителем 7 (Hog Wild) и/или оканчиваться на 7 (Hog Tied), таким образом, мы мешаем нашему сопернику набирать очки.

Если эти условия не выполняются, то мы бросаем кубик num_rolls раз

```
def make_mean_strategy(min_points,num_rolls=5):
    def mms(score, opponent_score):
        if ((opponent_score//10)+1)>min_points:
            if (HogTied ((score+1),opponent_score) != False) or\
            (HogWild ((score+1),opponent_score) != False):
                return 0
        return num_rolls
    return mms

strategy = make_mean_strategy(10)
print(strategy(6,21))
```

```
$ python lab_5.py
5
```

Задача 8: Напишите функцию make_average(), которая принимает функцию fn в качестве аргумента и возвращает функцию, которая принимает столько же аргументов, сколько и функция fn. Эта функция вызывает функцию fn num_sample раз и возвращает среднее арифметическое всех вызовов.

```
def make_average(fn, num_samples=100):
    def ma(*args):
        s=0
        for i in range(num_samples):
            s += fn(*args)
        return s/ float(num_samples)
    return ma
```

Задача 9: используйте функцию run_experiments() из файла hog.py, чтобы проверить какая из
стратегий make_comeback_strategy() или make_mean_strategy() работает лучше

```
Win rate against the baseline using 10 value: 0.5
Best always_roll strategy: 1
```