Reinforcement Learning and Decision Making **Homework #1**

Planning in MDPs

Description

You are given an N-sided die, along with a corresponding Boolean mask vector, `is_bad_side` (i.e., a vector of ones and zeros). You can assume that $1 < N \leq 30$, and the vector `is_bad_side` is also of size N and 1 indexed (since there is no 0 side on the die). The game of DieN is played as follows:

1. You start with 0 dollars.
2. At any time you have the option to roll the die or to quit the game.
    A. **ROLL**:
        a. If you roll a number not in `is_bad_side`, you receive that many dollars (e.g., if you roll the number 2 and 2 is not a bad side -- meaning the second element of the vector `is_bad_side` is 0, then you receive 2 dollars). Repeat step 2.
        b. If you roll a number in `is_bad_side`, then you lose all the money obtained in previous rolls and the game ends.
    B. **QUIT**:
        a. You keep all the money gained from previous rolls and the game ends.

Procedure

- You will implement your solution using the `solve()` method in the code below.
- Your return value should be the number of dollars you expect to win for a specific value of `is_bad_side`, if you follow an optimal policy. That is, what is the value of the optimal state-value function for the initial state of the game (starting with 0 dollars)? Your answer must be correct to 3 decimal places, truncated (e.g., 3.14159265 becomes 3.141).
- To solve this problem, you will need to determine an optimal policy for the game of DieN, given a particular configuration of the die. As you will see, the action that is optimal will depend on your current bankroll (i.e., how much money you've won so far).
- You can try solving this problem by creating an MDP of the game (states, actions, transition function, reward function, and assume a discount rate of $\gamma = 1$) and then calculating the optimal state-value function.

Resources

The concepts explored in this homework are covered by:

- Lecture Lesson 1: Smoov & Curly's Bogus Journey
- Chapter 3 (3.6 Optimal Policies and Optimal Value Functions) and Chapter 4 (4.3-4.4 Policy Iteration, Value Iteration) of http://incompleteideas.net/book/the-book-2nd.html (http://incompleteideas.net/book/the-book-2nd.html)
- Chapters 1-2 of 'Algorithms for Sequential Decision Making', M. Littman, 1996

Submission

- The due date is indicated on the Canvas page for this assignment. Make sure you have your timezone in Canvas set to ensure the deadline is accurate.
- Submit your finished notebook on Gradescope. Your grade is based on a set of hidden test cases. You will have unlimited submissions - only the last score is kept.
- Use the template below to implement your code. We have also provided some test cases for you. If your code passes the given test cases, it will run (though possibly not pass all the tests) on Gradescope.
- Gradescope is using *python 3.6.x* and *numpy==1.18.0*, and you can use any core library (i.e., anything in the Python standard library). No other library can be used. Also, make sure the name of your notebook matches the name of the provided notebook. Gradescope times out after 10 minutes.

In [5]:
```
#################
# DO NOT REMOVE
# Versions
# numpy==1.18.0
################
import numpy as np

class MDPAgent(object):
    def __init__(self):
        pass

    def solve(self, is_bad_side):
        """Implement the agent"""
        pass
        return True
```

Test cases

We have provided some test cases for you to help verify your implementation.

In [ ]:

```
## DO NOT MODIFY THIS CODE.  This code will ensure that your submissio
## will work proberly with the autograder

import unittest

class TestDieNNotebook(unittest.TestCase):
    def test_case_1(self):
        agent = MDPAgent()
        np.testing.assert_almost_equal(
            agent.solve(is_bad_side=[1, 1, 1, 0, 0, 0]),
            2.583,
            decimal=3
        )

    def test_case_2(self):
        agent = MDPAgent()
        np.testing.assert_almost_equal(
            agent.solve(
                is_bad_side=[1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
            ),
            7.379,
            decimal=3
        )

    def test_case_3(self):
        agent = MDPAgent()

        np.testing.assert_almost_equal(
            agent.solve(
                is_bad_side=[1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0,
            ),
            6.314,
            decimal=3
        )

unittest.main(argv=[''], verbosity=2, exit=False)
```