Stanford CME 241 (Winter 2021) - Assignment 2

Snake and Ladders

Chih-Hsuan 'Carolyn' Kao (chkao831@stanford.edu)

Feb 3rd, 2021

In [1]:

```
import sys
sys.path.append('/Users/chih-hsuankao/Desktop/CME241/RL-book/')
```

In [2]:

```
from dataclasses import dataclass
from IPython.display import Image
import itertools
import matplotlib.pyplot as plt
import numpy as np
from rl.distribution import Categorical, Constant
from rl.markov_process import MarkovProcess, FiniteMarkovProcess
from rl.markov_process import Transition, FiniteMarkovRewardProcess, RewardTransition
from typing import Mapping, Tuple, Dict
```

1. Model the game of Snakes and Ladders (single-player game) as a Markov Process. Write out it's state space and structure of transition probabilities.

In [3]:

```
Image(filename = "SnakeLaddersScreenshot.png", width=350, height=350)
```

Out[3]:

As shown, there are 101 states in the game, since we have a starting state of 0. Technically speaking, the state space is every integer from 0 to 100. 100 is the terminal state. In terms of the transition probability, from S_t to $S_t + 1, S_t + 2 ..., S_t + 6$, it would be 1/6 for normal cases.

In my design, for simplicity, I simply assume that to win, the player doesn't need to land exactly on square 100 -- in my design, I allow anyone with a roll that lands on or passes 100 to win the game (i.e. the bouncing back rule is not included).

2.Create a transition map: Transition data structure to represent the transition probabilities of the Snakes and Ladders Markov Process so you can model the game as an instance of FiniteMarkovProcess.

In [4]:

```python
@dataclass(frozen=True)
class SnakePosition:

    pos: int # player's current state (position) on the board

```

In [5]:

```python
@dataclass
class SnL_MP(FiniteMarkovProcess[SnakePosition]):
    """
    Finite Markov Process of Snakes and Ladders
    """
    def __init__(self,
                 board_size,
                 snl_states
    ):
        self.board_size: int = board_size # Dimension of the board
        self.snl_states: Mapping[int, int] = snl_states # transition map


        super().__init__(self.get_transition_map())


    def get_transition_map(self) -> Transition[SnakePosition]:
        """
        return transition map
        """

        # Initialize an empty dictionary of Mapping whose keys are the states in S
        # Maps to set of states it transitioned to from state w/corresponding probability
        # Maps to None if given terminal key
        d: Dict[SnakePosition, Optional[Categorical[SnakePosition]]] = {}

        # Iterate through board positions
        for position in range(0,self.board_size-1):
            # Initialize the map with set of states to be transitioned to
            prob_map: Mapping[PlayerState, float] = {}

            board_size = self.board_size
            special = position in self.snl_states # special states with snakes & ladders
            current_state = SnakePosition(position)

            # an integer that serves as a threshold to handle ending cases
            final_edge = min(position+7, board_size+1)

            if special:
                state = self.snl_states[pos]
            else:
                state = position

            prob = 1.0/6.0
            # Iterate through possible next states
            for j in range(state+1, final_edge):
                # for edge cases (ending)
                if j == final_edge - 1:
                    prob = 1 - (final_edge - state - 2)/6
                    prob_map[SnakePosition(j)] = prob
                # for other normal cases
                else:
                    prob_map[SnakePosition(j)] = prob

            d[SnakePosition(position)] = Categorical(prob_map)
            # Won the game -- ended
            d[SnakePosition(board_size)] = None

            return d

```

In [6]:

```python
snl_cases = {3: 39, 7: 48, 12: 51, 20: 41, 25: 57, 25: 57, 25: 57, 28: 35,
             28: 35, 38: 1, 45: 74, 49: 8, 53: 17, 60: 85, 65: 14, 67: 90,
             69: 92, 70: 34, 76: 37, 82: 63, 88: 50, 94: 42, 98: 54}

snl_mp = SnL_MP(100,snl_cases)
```

4. Extend the Snakes and Ladders FiniteMarkovProcess to an appropriate FiniteMarkovRewardProcess instance.

In [7]:

```
@dataclass
class SnL_MRP(FiniteMarkovRewardProcess[SnakePosition]):
    """
    Finite Markov Process of Snakes and Ladders
    """
    def __init__(self,
                 board_size,
                 snl_states
    ):
        self.board_size: int = board_size # Dimension of the board
        self.snl_states: Mapping[int, int] = snl_states # transition map


        super().__init__(self.get_transition_map())


    def get_transition_map(self) -> Transition[SnakePosition]:
        """
        return transition map
        """

        # Initialize an empty dictionary of Mapping whose keys are the states in S
        # Maps to set of states it transitioned to from state w/corresponding probability
        # Maps to None if given terminal key
        d: Dict[SnakePosition, Optional[Categorical[SnakePosition]]] = {}

        # Iterate through board positions
        for position in range(0,self.board_size-1):
            # Initialize the map with set of states to be transitioned to
            prob_map: Mapping[PlayerState, float] = {}

            board_size = self.board_size
            special = position in self.snl_states # special states with snakes & ladders
            current_state = SnakePosition(position)

            # an integer that serves as a threshold to handle ending cases
            final_edge = min(position+7, board_size+1)

            if special:
                state = self.snl_states[pos]
            else:
                state = position

            prob = 1.0/6.0
            # Iterate through possible next states
            for j in range(state+1, final_edge):
                # for edge cases (ending)
                if j == final_edge - 1:
                    prob = 1 - (final_edge - state - 2)/6
                    prob_map[SnakePosition(j)] = prob
                # for other normal cases
                else:
                    prob_map[SnakePosition(j)] = prob

            d[SnakePosition(position)] = Categorical(prob_map)
            # Won the game -- ended
            d[SnakePosition(board_size)] = None

            return d
```