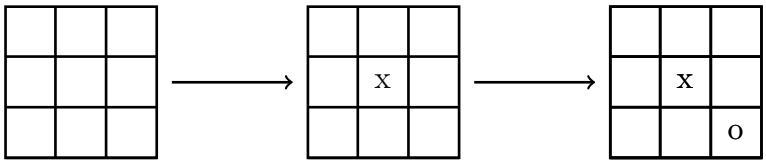
1 Backward Induction: Tictactoe

Consider the two player zero sum game (sequential move) of tic tac toe where,

- Players: {x, o}, player x plays first.
- Actions: $\{0, 1, \ldots, 8\}$, where each action represents the square in which the move will be played as shown below.

0	1	2
3	4	5
6	7	8

- Set of histories: H
  A history $h \in H$ is a sequence of actions taken from the starting state. Note, a history is different from a board position since several histories could lead to the same board position.
  Example: $h = [4, 8]$



- Terminal histories: Z
  The game ends in a win for either player or a draw at any terminal history $z \in Z$.
- Player function: $player(h)$ gives the player who makes the move at a given history $h \in H \setminus Z$.
- Action function: $actions(h)$ gives the valid actions (empty squares) at a given history $h \in H \setminus Z$.
- Utility function of player i: $utility_i(z)$ gives the utility for player i for a given terminal history $z \in Z$.
- Randomized strategy of player i: gives a probability distribution over valid actions i.e., $\Delta actions(h)$ for a given history $h \in H$ where player i plays.

1.1 Task

Find the subgame perfect equilibrium strategy for both players in the tic tac toe game using backward induction . Note this involves find the strategy for both players at every history $h \in H \setminus Z$.

1.2 Code

- `History` class:
  - `history`: list to keep track of sequence of actions

- `board`: board position (list) corresponding to the history
- `player`: player whose move it is at the given history

There are also several helper functions defined but not implemented. This might be needed when coding `backward_induction`. Feel free to implement this in anyway if needed.

- Global variables `strategy_dict_x` and `strategy_dict_o`: These will be updated in the recursive backward_induction function to keep track of the strategies which are a mapping from histories to probability distribution over actions. Note testing will be done w.r.t these variables (returned by `solve_tictactoe` function which calls `backward_induction`).

  1. These are dictionary with keys as string representation of the history list e.g. if the history list of the history class object is [0, 4, 2, 5], then the key is `"0425"`. Each value is in turn a dictionary with keys as actions `0 to 8 (str "0", "1", ..., "8")` and each value of this dictionary is a float (representing the probability of choosing that action). Example: `{"0452": {"0": 0, "1": 0, "2": 0, "3": 0, "4": 0, "5": 0, "6": 1, "7": 0, "8": 0}}`
  2. Note, the strategy for each history in `strategy_dict_x` and `strategy_dict_o` is probability distribution over actions. But since tictactoe is a Perfect information extensive form game (PIEFG)[1], there always exists an optimal deterministic strategy (SPE). So your strategy will be determistic (0s, 1s). The above data structure just keeps it more general.

- Function `backward_induction`
  Input: `History` class object
  Return: float

  - Implement backward induction for tictactoe.
  - Update the global variables `strategy_dict_x` or `strategy_dict_o` which are a mapping from histories to probability distribution over actions. Do this before you return the best utility.

Refer to code and doc strings for more information. Implement backward_induction function to get the strategies for both players. Also if needed, implement the blank functions part of history class.

1.3 Test

There are no public test cases for this question on gradescope. However, the given code saves your computed `strategy_dict_x` and `strategy_dict_o` in `policy_x.json` and `policy_o.json`. You can run `play_tictactoe.py` as follows and play against your computed policies to test it. This requires `pygame` package.

```
python3 play_tictactoe.py --BotPlayer x --BotStrategyFile policy_x.json
```

or

```
python3 play_tictactoe.py --BotPlayer o --BotStrategyFile policy_o.json
```