Richard Ma
Example reinforcement learning algorithm scripts: 'tictactoe-reinforcement-learning' by user rfeinman on Github: https://github.com/rfeinman/tictactoe-reinforcement-learning

1. High level overview/abstract of the code:

The scripts implement training functions to teach a learner to play tictactoe using reinforcement learning. The states are the possible configurations of the board, and the actions are the choice of where on the 3x3 grid to place the next move. Rewards are defined as +1 for winning the game, -1 for losing the game, and 0 for a tie or for a continuation of the game (i.e. the action taken doesn't finish the game and the game continues). The learner's knowledge of the state-action table can be filled in with either the Q-learning algorithm or the SARSA algorithm when playing games.

To train the learner, the author implements a teacher class that can follow the optimal strategy for playing the game, thus creating a playable agent that allows training through self-play. Because tic-tac-toe is a simple game with limited dimensionality, the min-max tree can be fully explored and a player using the optimal decision strategy will never lose; there is effectively an analytical method to optimally play the game. The author thus instructs the teacher class to deviate from the optimal strategy with a probability *p* so that the learner has a chance to win and thus explore the corresponding regions of the state-action table.

By playing these games against the teacher and exploring states, actions and rewards, the learner is able to fill in its Q-table. The author includes functions for plotting reward as a function of training time.

2. Identifying core sections of the code (**Original script is plain, my comments are highlighted. All scripts in python**):

This is the function by which the game is played and Q tables are updated to teach the learner

```
def playGame(self, player_first):
    """
    Begin the tic-tac-toe game loop.

    Parameters
    ----------
    player_first : boolean
        Whether or not the player will move first. If False, the
        agent goes first.

    """
    # Initialize the agent's state and action
    if player_first:
        self.playerMove()
    prev_state = getStateKey(self.board)
    prev_action = self.agent.get_action(prev_state)
```

Considering states of the game board: States are defined as 3x3 string arrays, in which each element can either be empty or occupied by one of the two playesr

```
        # iterate until game is over
        while True:
            # execute oldAction, observe reward and state
            self.agentMove(prev_action)
            check = self.checkForEnd('O')
            if not check == -1:
                # game is over. +1 reward if win, 0 if draw
                reward = check
                break
            self.playerMove()
            check = self.checkForEnd('X')
            if not check == -1:
                # game is over. -1 reward if lose, 0 if draw
                reward = -1*check
                break
            else:
                # game continues. 0 reward
                reward = 0
            new_state = getStateKey(self.board)
```

Here rewards are defined: 1 for winning the game, 0 for a draw or continuation of the game, -1 for losing.

```
            # determine new action (epsilon-greedy)
            new_action = self.agent.get_action(new_state)
```

Here is a call to the function to choose the next move based on greedy search: Given the existing Q table, considering the possible actions given the current state, choose the action with the highest reward. If multiple actions have the same reward, one is randomly chosen. The author initializes all actions with 0 reward.

```
            # update Q-values
            self.agent.update(prev_state, new_state, prev_action, new_action, reward)
```

After taking the action, go back and update the Q table using the appropriate equation

```
            # reset "previous" values
            prev_state = new_state
            prev_action = new_action
            # append reward

        # Game over. Perform final update
        self.agent.update(prev_state, None, prev_action, None, reward)
```

Manually specified update to the Q table after the game has finished and the game loop has ended.