Tic-Tac-Toe using RL

CIS 550: Final Project Presentation

Group_ID: 6
Bala Akshay Reddy Yeruva (01861852)
Panjala Pavan Kumar (01872582)
Pavithra Venkatesan (01766109)
Shravani Lagishetty(01860180)

Objective

Design an artificially intelligent bot to play a classic tic-tac-toe game. We used two different implementations that are commonly used in game playing algorithms.

- Reinforcement Learning represents a strategy oriented approach without the use of recursive algorithms or a complex stored moves database.
- Min-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.

While the two methods vary from each other, we characterize their implementation on a game playing interface and analyze the differences from performance and result oriented perspectives.

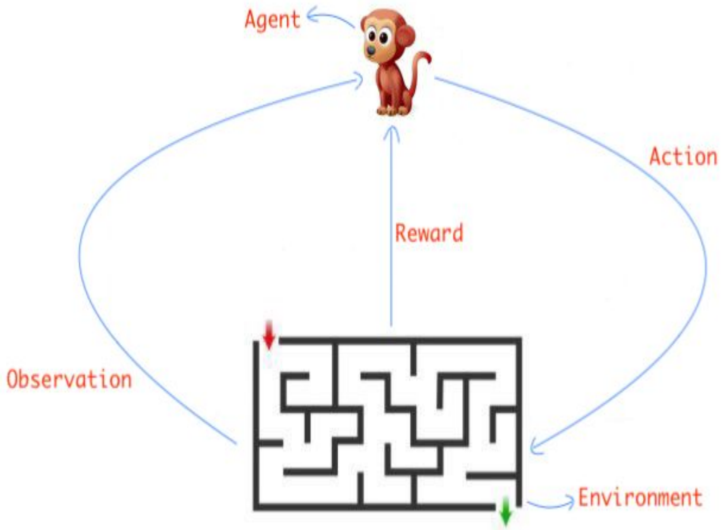
Introduction to Reinforcement Learning

“RL is concerned with how ***agents*** ought to take ***actions*** in an ***environment*** in order to maximize cumulative ***reward***.”

- Exploration or Exploitation

Components:

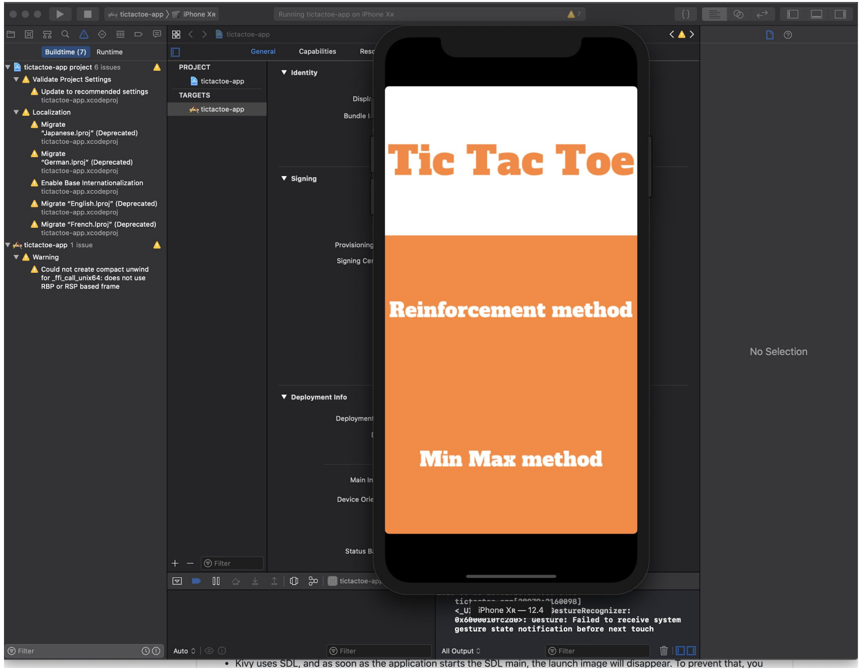
- Agent
- Environment
- Action and Observation
- Reward

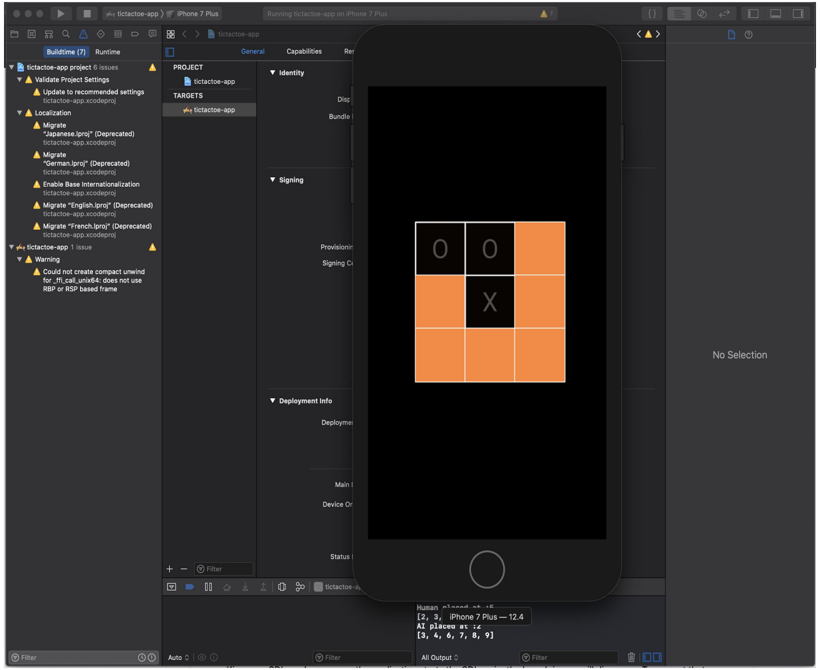


Technologies Used

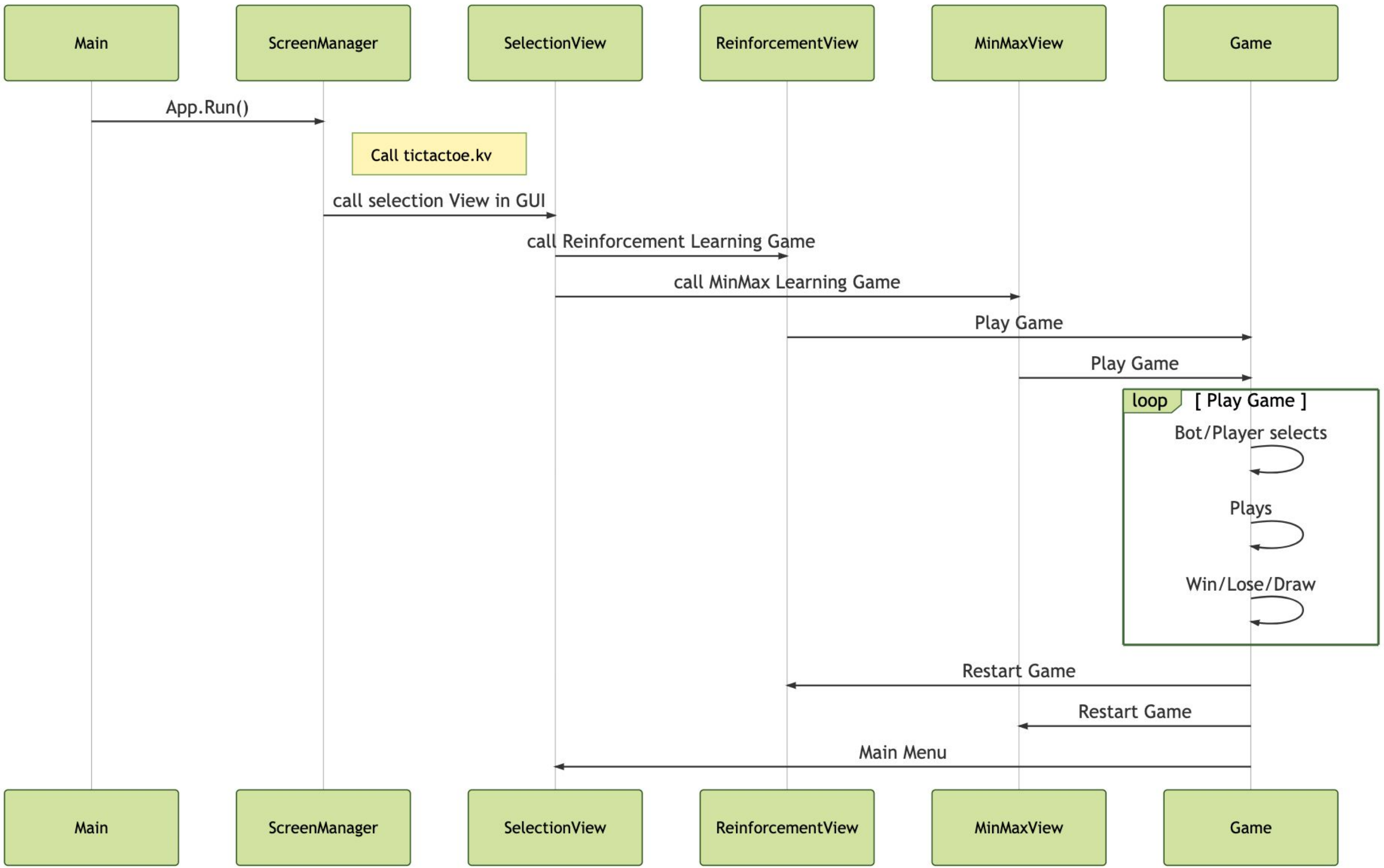
- Language: Python 3.7
- Libraries: Numpy,Math, Itertools etc.
- Framework: Kivy 1.11.1, Toolchain.py
- IDE: PyCharm
- Compiler: Cython
- Build Platform: Xcode 10.3
- Build Target: iOS

Simulation Environment

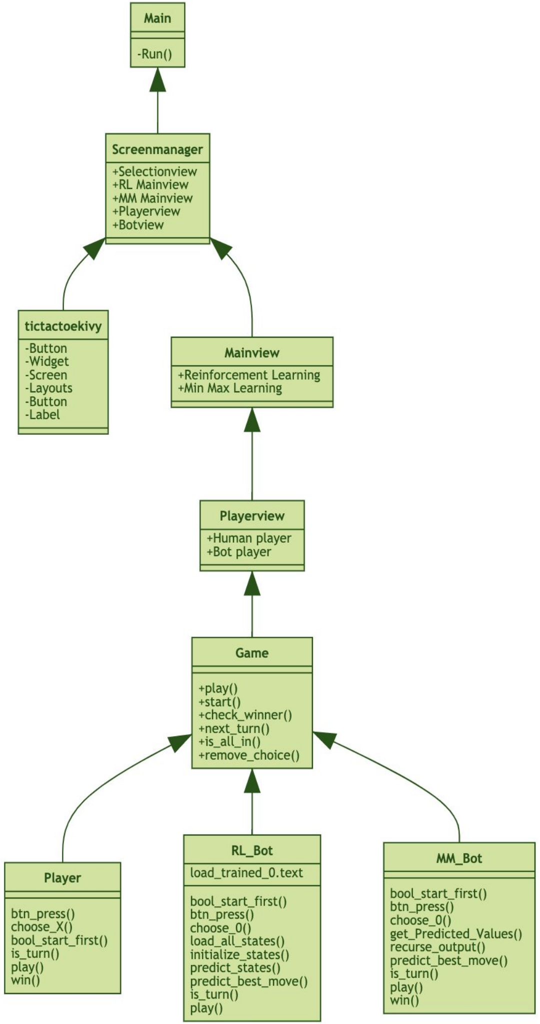




UI Sequence Diagram



UI Class Diagram



Methodology

Case 1: Using Reinforcement Learning

- Setting up the Tic-Tac-Toe environment.
- Training an agent(O) by playing against it(trained agent X) and saving the policy.
- Loading the policy and making the agent play against human(X).

Case 2: Using Min-Max Algorithm

- Setting up the Tic-Tac-Toe environment.
- Applying Min-Max Algorithm for player O.
- Playing against human(X) with model built.

Temporal Difference Learning

- Initially each state is assigned with a constant value ‘0’.
- Agent computes all possible actions it can take in the current state and the new state obtained from the action taken.
- State-Value vectors gives the values of all states in the game.
- Based on the state values and the epsilon value(0.2) from (epsilon decreasing strategy) agent tries to either ***exploit*** or ***explore***.
- After every action the state values are updated using the following formula.

Temporal Difference Learning (contd.)

State-Value vector, V(s)

$$V(s) \leftarrow V(s) + \alpha \times (V(s^f) - V(s))$$

Where V(s) => current state

$V(s^f)$ => new state

$\boldsymbol{\alpha}$ = 0.2 in our case => step-size parameter / learning rate(>0)

Training Phase

The code snippet on the right is taken
From training.py for training Agent ‘O’

TD() defines the Temporal Difference
earning algorithm using reinforcement
earning for Tic-Tac-Toe.

The resultant rewards/policies are stored
In ‘trained_O.txt’ : for testing

```
def TD(sv, each_player, epsilon):
    actions = []
    curr_state_values = []
    empty_cells = []
    for i in range(3):
        for j in range(3):
            if sv[i][j] is ' ':
                empty_cells.append(i * 3 + (j + 1))

    for empty_cell in empty_cells:
        actions.append(empty_cell)
        new_state = new(sv)
        play(new_state, each_player, empty_cell)
        next_sid = list(states_dictionary.keys())[list(states_dictionary.values()).
        index(new_state)]
        if each_player == 'X':
            curr_state_values.append(sv_X[next_sid])
        else:
            curr_state_values.append(sv_O[next_sid])

    print('Possible Action moves = ' + str(actions))
    print('Action Move values = ' + str(curr_state_values))
    best_move_id = np.argmax(curr_state_values)

    if np.random.uniform(0, 1) <= epsilon:  # Exploration
        best_move = random.choice(empty_cells)
        print('Agent decides to explore! Takes action = ' + str(best_move))
        epsilon *= 0.99
    else:  # Exploitation
        best_move = actions[best_move_id]
        print('Agent decides to exploit! Takes action = ' + str(best_move))
    return best_move
```

Testing Phase

This phase involves making our trained agent ‘O’ play with Human ‘X’.
In order to begin play, the policies/ state values trained for agent ‘O’ are loaded.

```
# Loading policy or the trained state values
sv_O = np.loadtxt('trained_O.txt', dtype=np.float64)
```

Min-Max Algorithm

```python
def MM(state, each_player):
    # Minimax Algorithm

    winner_loser, done = cur_state(state)
    if done == "Done" and winner_loser == 'O':  # Agent win
        return 1
    elif done == "Done" and winner_loser == 'X':  # Human win
        return -1
    elif done == "Draw":  # Draw
        return 0

    moves = []
    empty_cells = []
    for i in range(3):
        for j in range(3):
            if state[i][j] is ' ':
                empty_cells.append(i * 3 + (j + 1))

    for empty_cell in empty_cells:
        steps = {}
        steps['index'] = empty_cell
        new_state = new(state)
        play(new_state, each_player, empty_cell)

        if each_player == 'O':
            result = MM(new_state, 'X')
            steps['score'] = result
        else:
            result = MM(new_state, 'O')
            steps['score'] = result

        moves.append(steps)
```

```python
# Finding the next best move
best_move = None
if each_player == 'O':
    best = -infinity
    for steps in moves:
        if steps['score'] > best:
            best = steps['score']
            best_move = steps['index']
else:
    best = infinity
    for steps in moves:
        if steps['score'] < best:
            best = steps['score']
            best_move = steps['index']

return best_move
```

DEMO

https://github.com/Pavivenkatesan/TicTacToe-RL-MM-.git

Results and Interpretation

We implemented Tic-Tac-Toe

- Using Reinforcement Learning
- Using Min-Max algorithm (without RL)

As shown in the demo, the RL agent tend to fight better against human than Min-Max agent giving us more efficient results.

Hence, Model-free learning approaches like Temporal difference learning seems to work better than Model-based learning Min-Max algorithm w.r.t Tic-Tac-Toe.

Future Scope

The game is one of the simple demonstration of Reinforcement Learning.

RL combines the functional approximation of Deep Learning with the optimization of decision-based systems to bring the next wave of technology-fueled innovation.

Future applications of reinforcement learning may include,

- Advanced online multi-player gaming system like Google Stadia
- Smart traffic signals
- Autonomous robots
- Advanced self-driving cars
- Smart prosthetic limbs

Thank you