Problem 1.

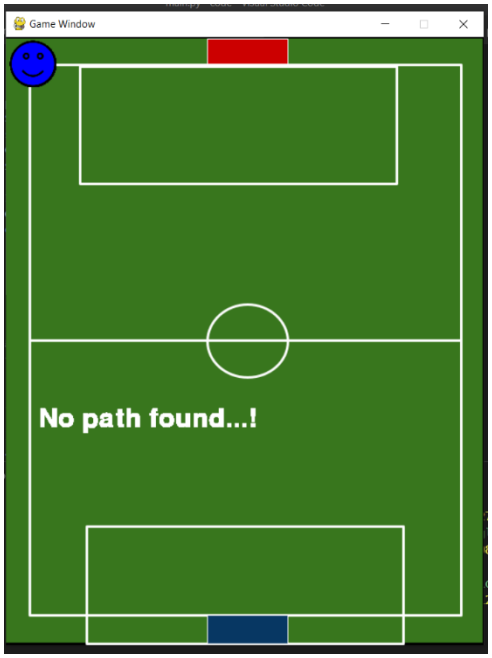
Programming: Let's Play Soccer [Marks 100]

The task is to model an agent to perform assisted goal shootout in the game of soccer. This is a simple implementation of policies; no learning is expected by the agent.

- There are 2 teams, team RED and team BLUE
- Team RED has 3 players and team BLUE has 4 players (1 Kicker and 3 in team RED play area).
- The players cannot move from their respective places once the game starts.
- Team BLUE is performing an assisted goal shootout but from the Centre Circle position. (Condition 1)
- So, one player from the BLUE team must remain at the centre circle to take an assisted goal shoot.
- One player from each team will be staying in the Team RED goal box and will not leave it. (Condition 2)
- Apart from the centre kicker from team BLUE players, the rest of the players will remain in the upper part as shown (RED Team area). (Condition 3)
- Shootouts must be done from the centre by the team BLUE player as shown in the below image.
- How to Play:
    - The assisted goal shoot will be taken by a TEAM BLUE player from the centre.
    - The kicker is our agent.
    - The agent needs to decide the shortest goal path, this will be your heuristic cost. (has to be assisted goal) (Condition 4)
    - With every run, the position of players will be changed, which has to be randomized and should satisfy the previous condition. (Condition 5)

Solution.

For this question, I have used the 'pygame' python library to simplify loading assets and rendering the players and the field. The implemented program starts at the empty screen,



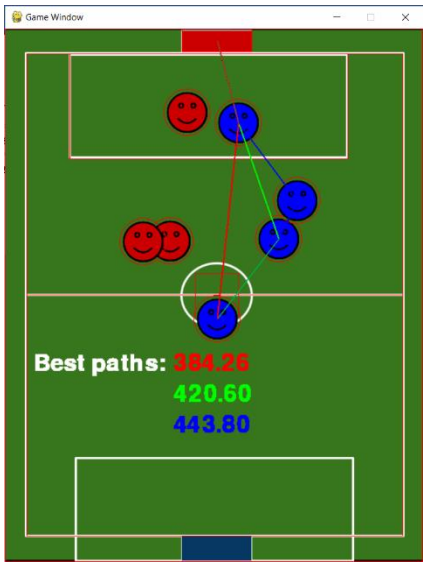
This screen represents the uninitialized state of the program. To start the program, press the 'X' key.

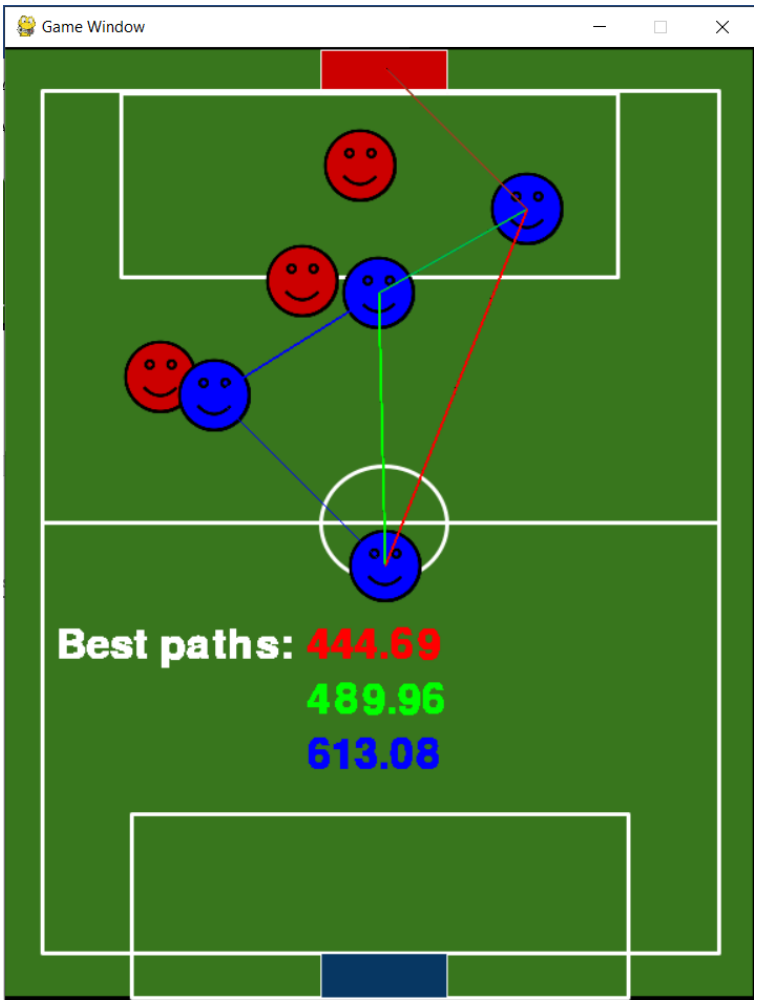
The various **controls** for the program are as follows:-

- **Press X to**: Relocate players randomly.
- **Press B to**: Show/hide the bounding boxes of objects.
- **Press Esc to**: Quit the program.

Once the program has starter press **X** to relocate players randomly. Doing this automatically computes the top 4 shortest paths to score a goal and displays the path lengths as well as coloured lines of the paths to follow.

If less than 4 paths were found, the program shows however many that were found in decreasing order of length.





Pressing **B** shows/hides the bounding boxes which can be used to verify weather any viable paths have been missed or if any invalid paths are shown.

The environment is modelled as a data structure containing the following information,

```
13  @dataclass
14  class Environment:
15      '''
16      Rpresents the environment that this policy works in.
17      '''
18      red_players: List[Player]
19      blue_players: List[Player]
20      kick_team: Team
21      field: SoccerField
22      kicker: Player = None
```

Figure 1. Environment - policy.py

Here, the environment contains the list of all players in their teams, a reference to the kicker player, a reference to the team to which the kicker belongs and a reference to the field which contains location data regarding the different sections and goal boxes on the field.

To find the shortest path from the **kicker** to the centre of the goal, the program performs a Depth First Search over all positions of the blue team players. Each position checks whether a pass can be sent from the current position to any of the teammates that have not been explored during DFS. To check is a pass is possible, the program performs 2 tasks, given a 2D point P_1 and P_2,

- For each point P_0 as position of player that can receive a pass check if
  - Minimum distance of the line from P_1P_2 from $P_0 > radius_{player}$
  - Smallest rectangle containing P_1 and P_2 also contains P_0.

```
117         def check_collide_connect(p1: Point, p2: Point):
118             '''
119             Checks if 2 points can be connected without collision with any other players.
120             '''
121             l = Line(p1, p2)
122             for p in collide_pos:
123                 if p != p1 and p != p2:
124                     d = l.dist_of_point(p)
125                     ray_rect = Rectangle.enclosing_points([p1, p2])
126                     ray_rect.top_left -= pl_radius
127                     ray_rect.bottom_right += pl_radius
128                     if d <= pl_radius and ray_rect.contains_point(p):
129                         LOG.debug(f'{l} colides with {p} for distance {d}.')
130                         return False
131             return True
```

Figure 2. Circle to Line Collision Detection - policy.py

If both of the above conditions are TRUE then the circle or given radius at P_0 collides with the ray from P_1 to P_2. Using the above collision logic, the program can perform a simple DFS search to find all possible paths connecting the kicker to the centre of the goal. After all paths are found, the program simply filters the paths that satisfy out constraints and returns this list.

Running the program

To run the program the following pre-requisites must be installed on your system,

- **Python**: ^3.9.2 (https://www.python.org/downloads/release/python-392/)
- **Poetry**: ^1.0.0 (https://python-poetry.org/docs/#installation)

The **poetry** library is used to ensure package compatibility and system cleanliness since it automatically makes use of python virtual environments. **NOTE** that all other dependencies (like pygame etc.) are installed in the '.venv' folder and the user does not have to handle anything except installing python and poetry.

Once **poetry** is installed, open a command prompt and navigate into the code folder (the top-level folder containing pyproject.toml). Then simply run the command 'poetry run main'.

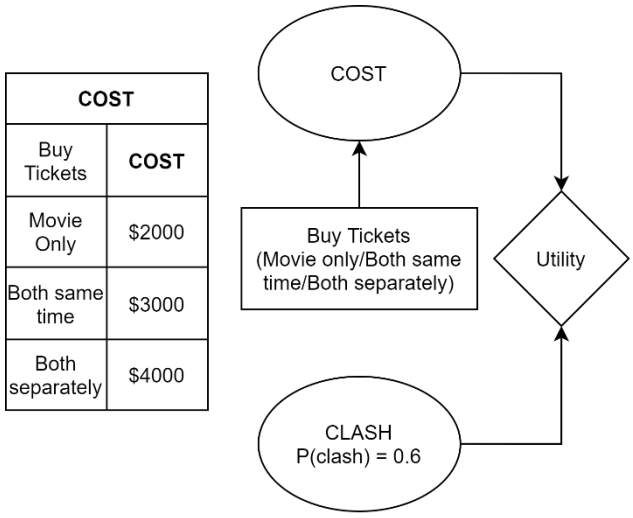
Problem 3.

Topic: Making Simple Decision Networks

The new movie of the Marvel Universe "Falcon and the winter soldier" is releasing soon in the nearest IMAX theatres and you being a big fan of the marvel universe, want to watch the movie at IMAX theatres only. On the same day of the movie release, you come to know that there is a concert of your favourite pop star, happening in the same city of 1 hr duration (consider timing is not known and entry will be allowed max 10 min before the start of the concert). Now, you have 2 options. Either you can buy tickets for both now, at some discounted rates or you can buy them separately after your Marvel Universe Movie so that you can avoid time clashing and wasting your money on buying the ticket. It is given that, the probability of finding the time for both the events is 0.4. A single ticket costs you around $Rs.\ 2000/-$ each and combined ticket costs you around $Rs.\ 3000/-$. The Value of going to the movie is $Rs.\ 2000/-$. Which ticket you should buy? Calculate the expected value of buying a combined ticket for the probability of 0.4.

Solution.

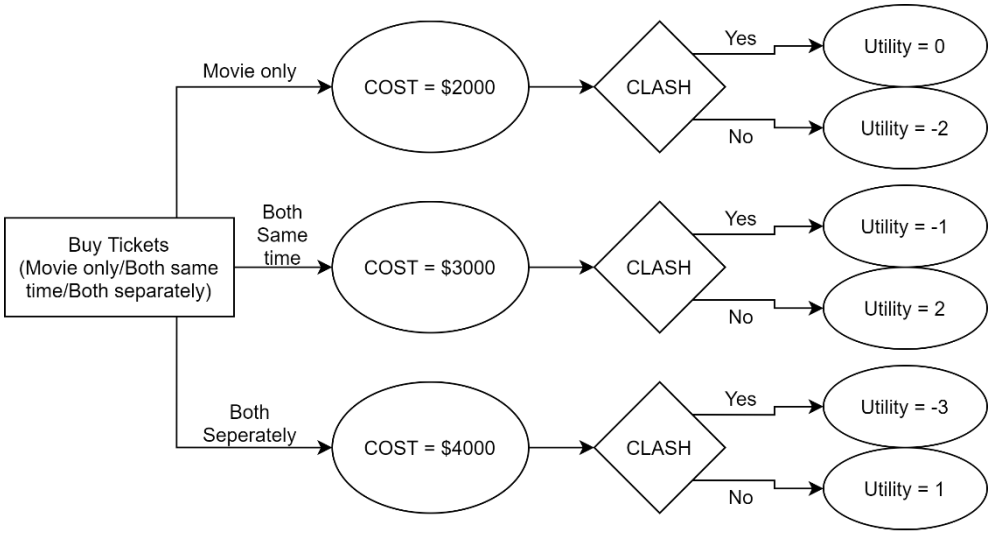
The decision network for this problem is as follows,



COST	
Buy Tickets	**COST**
Movie Only	$2000
Both same time	$3000
Both separately	$4000

Utility		
CLASH	COST	**Utility**
F	$2000	-2
F	$3000	2
F	$4000	1
T	$2000	0
T	$3000	-1
T	$4000	-3

The decision tree is as follows,



Since the value of each ticket is $Rs.\ 2000/-$, buying them separately costs $Rs.\ 4000/-$. However, the combined ticket costs $Rs.\ 3000/-$, this means a profit of $Rs.\ 1000/-$. But this profit is only applicable if there is no clash in the event schedules. In fact, if there is a clash, the best action is to buy only one ticket at the cost of $Rs.\ 2000/-$. Thus, if there is a clash, then the combined ticket causes a loss of $Rs.\ 1000/-$.

$$P(no\_clash) = 0.4 \tag{3.1}$$
$$U(no\_clash) = 1000 \tag{3.2}$$
$$U(clash) = -1000 \tag{3.3}$$
$$\Longrightarrow Value\ of combined\ ticket = P(no\_clash) * U(no\_clash) + P(clash) * U(clash) \tag{3.4}$$
$$\Rightarrow Value\ of\ combined\ ticket = 0.4 * 1000\ -\ 0.6 * 1000 = -200 \tag{3.5}$$