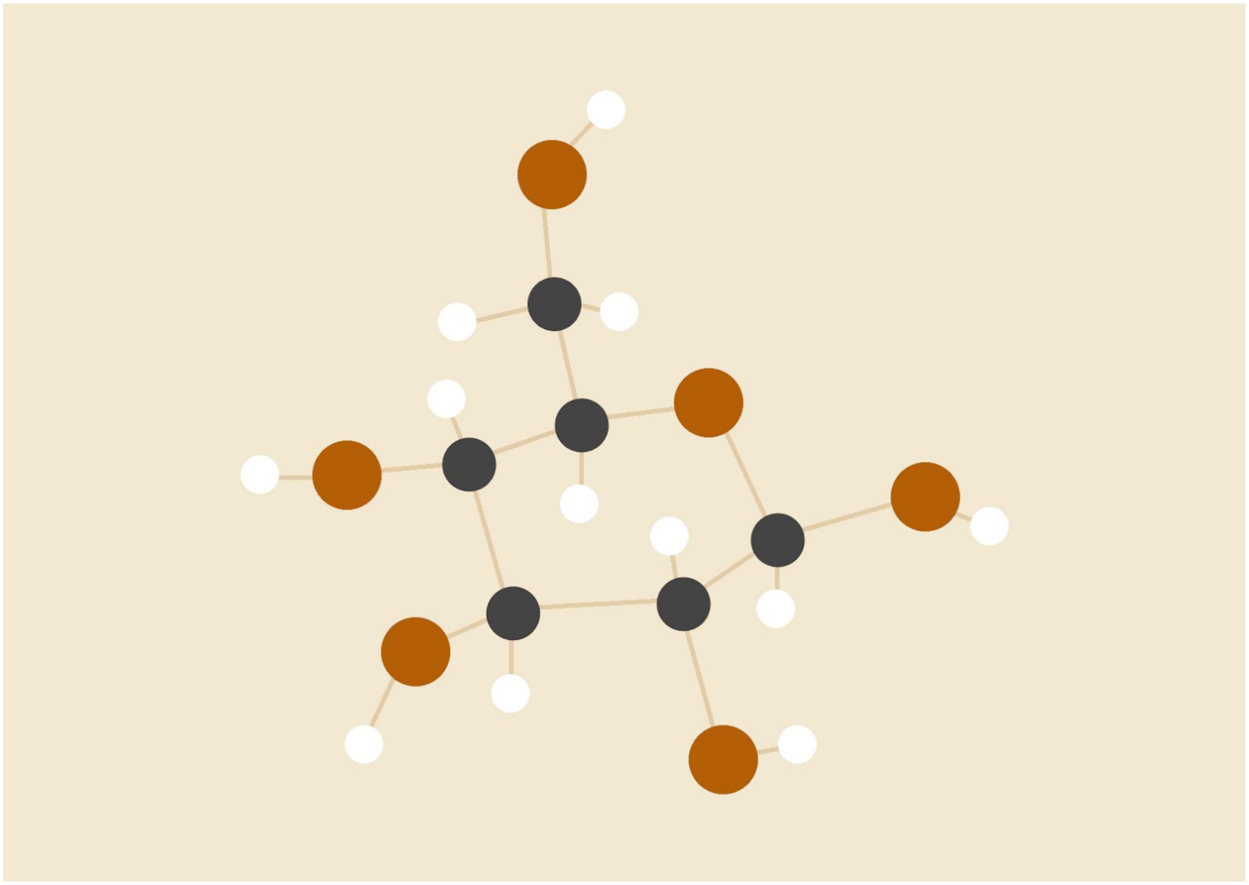
REAL TIME AUTONOMOUS SYSTEM

Multi-agent Collaboration Simulation with RoboCup as example domain



Anurag Shah - B16CS034

Qazi Sajid Azam - B16CS026

DETAILED PROBLEM SPECIFICATION

We have designed a logic based collaborative multi agent system for the soccer environment. Soccer game is a two team game with both teams having 11 players. We have modelled the second team to be a dynamic and obstructive environment and not as a team that wants to score a goal. The first team's global aim is to score goals by avoiding losing the ball to the opposing team. The agents in this team are logic based agents which we will discuss in detail in further sections. We have not used any agent development environment (SPADE or PADE) nor have we used robocup Soccer simulator. We have made our agent development environment and our own soccer simulation software.

DESIGN

1. **Agent Architecture in general**
   Agents are designed to have different behavior according to their role in the game - offenders or defenders. Agents have a global aim to score a goal for their team. Agents also have 3 subgoals which determine how they will play the game. The subgoals differ for the attackers and defenders.
   a. Attacker Subgoals: Ball holder
      To determine whom to pass the ball or to move with the ball we calculate score with the weighted sum of following 3 criteria:
      i. Distance to goal
      ii. Sum of Opponents distance from himself
      iii. Distance from ball

      The ball is passed to the team member with maximum score. If the player with the ball himself has the maximum score, he moves ahead with the ball. If the player with the ball is close to the goal then he can shoot the goal.

   b. Attacker Subgoals: Other members
      The other team members will also move with a combination of following intents. The resulting movement is a weighted sum of 3 motions:
      i. Move towards the goal
      ii. Move towards the ball
      iii. Avoid forming crowded clusters

   c. Defender subgoals

The defender's motion is controlled by the following intents. The resulting motion is a weighted sum of 2 different motions:

i. Get between the player holding the ball and the goal
ii. Avoid forming crowded clusters

2. **Multi-agent Collaboration Protocol**
Collaboration Protocol is like a Contract Net Protocol. After each timestep the player with the ball conducts a Second Price Sealed Bid Auction to determine whom should the ball be passed. The other players bid their true value of having the ball at their current position which is calculated considering different factors like the position of opponent players, their distance to goal, their distance to the person passing the ball and whether he is clear to pass i.e. can the player with ball pass it to him without the opponent players' interception. A slight variation from the Second Price Sealed Bid Auction as here the player having the ball also participates in the auction. The ball is passed to the player with maximum score. If the player with the ball has the highest score, he moves towards the goal by avoiding the opponent players. We have used indirect communication (shared memory) to conduct the auction.
3. **Behavioural Models**
We have defined a few different behaviors for attacker and defender agents. We have defined following classes for the same
   a. Attackers
      i. Utility Based
      A score value is calculated based on distance from opponents, distance from goal, and distance from goal. The ball is passed to the player with maximum score. If the current owner of the ball has the maximum score, then he moves towards goal with the ball. Other players move towards the goal and the ball and avoid clustering.
      ii. Attack Based
      The player with the ball passes the ball to the nearest player in its team and who is ahead of him. It takes the ball ahead if no one is ahead of it. Other players who are nearest to it aim to run ahead of it and be in free area so that the ball can be passed to them. The player near to goal within a certain distance scores the goal
      iii. Defensive Play
      The attacking team takes up a defensive strategy when they have scored more goals then the other team and their aim switches to

keeping possession over the ball and not getting scored. This class reflects the property.

b. Defenders

   i. Random

      The defender moves randomly and acts like dynamic moving obstacles which the attacking team should avoid while achieving their goal.

   ii. Logic Based

      To prevent the other team from scoring, the agents have to position themselves between the goal and the ball owner. They also maintain sufficient distance among themselves so as to cover a larger area and can deal better if the ball gets passed to another player.

4. **Planning Scheme for Achievement of the goal**

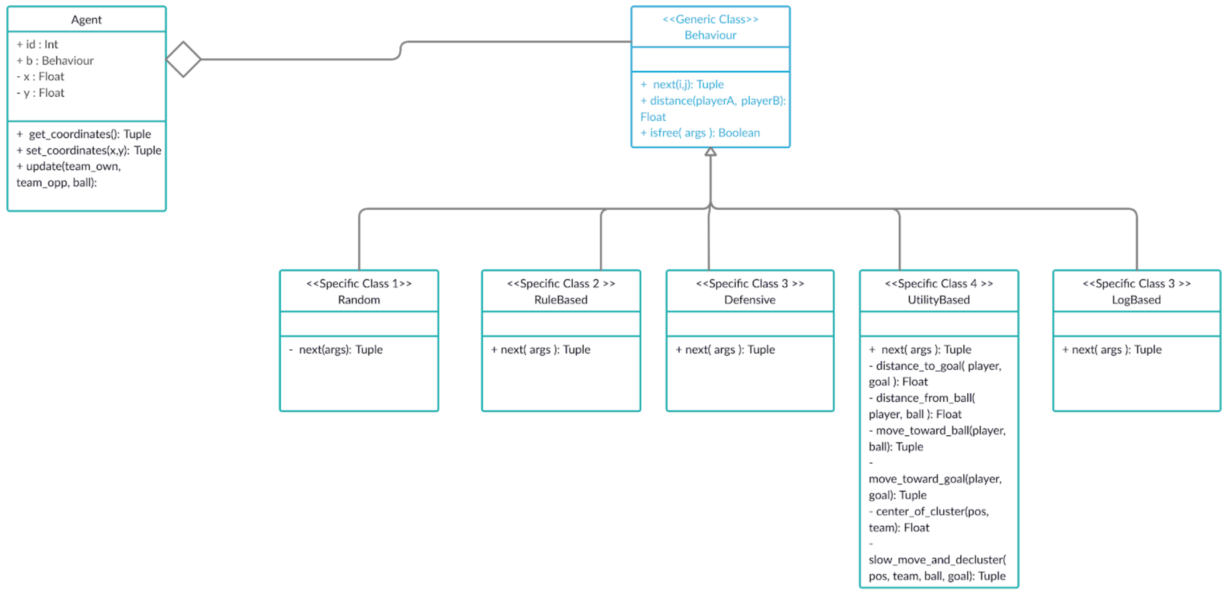
   To score a goal, the attacking agents must proceed towards the goal, because they can only shoot the goal when they are close enough. They must also maintain sufficient distance among themselves because they have to pass the ball to another player if too many defenders get close to the current ball holder. Finally, when a player with the ball is close enough to the goal, he can shoot and the game ends. All these plans are implemented in code as discussed above.

IMPLEMENTATION

1. **Description of the Software architecture of the system**

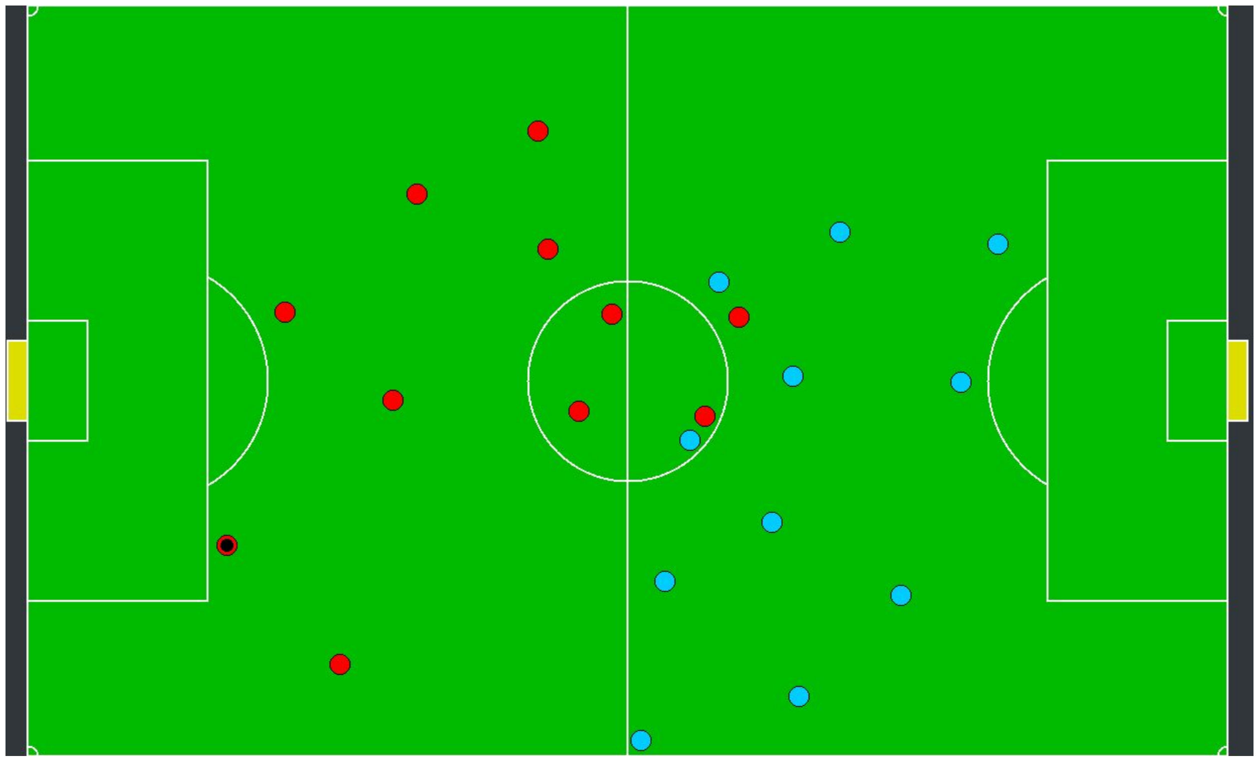
   We have implemented the whole system on our own without any external library. We have used Python 3.7 as our coding language. The UI is designed using Tkinter

   Here is a UML class diagram describing our software architecture.


Agent
+ id : Int
+ b : Behaviour
- x : Float
- y : Float
+ get_coordinates(): Tuple
+ set_coordinates(x,y): Tuple
+ update(team_own, team_opp, ball):
<<Generic Class>> Behaviour
+ next(i,j): Tuple
+ distance(playerA, playerB): Float
+ isfree(args): Boolean
<<Specific Class 1>> Random
- next(args): Tuple
<<Specific Class 2 >> RuleBased
+ next(args): Tuple
<<Specific Class 3 >> Defensive
+ next(args): Tuple
<<Specific Class 4 >> UtilityBased
+ next(args): Tuple
- distance_to_goal(player, goal): Float
- distance_from_ball(player, ball): Float
- move_toward_ball(player, ball): Tuple
- move_toward_goal(player, goal): Tuple
- center_of_cluster(pos, team): Float
- slow_move_and_decluster(pos, team, ball, goal): Tuple
<<Specific Class 3 >> LogBased
+ next(args): Tuple

2. **Sample output**

   The screenshot below shows a sample output. The players are shown as red circles (attacking team) and blue circles (defending team). The terminal prints the ball-pass events for reference.



3. **Code with documentation for execution**

To run the code, the requirements are Python 3 and Tkinter. It can be installed using following commands on Ubuntu system:

```
sudo apt install python3
sudo apt install python3-tk
```

The code can be executed from its directory using:

```
python3 main.py
```

Github link - https://github.com/DeveloperAndCoder/Autonomous-Robocup