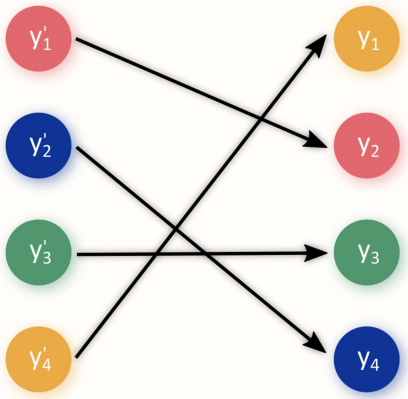
9 Submission 1 - Role Assignment

9.1 Introduction

In this submission, we tackle the critical problem of role assignment within Soccer AI—specifically, the challenge of assigning unique players to unique locations on the field. Effective role assignment is essential for optimizing team performance, as it ensures that each player is positioned strategically to execute their role within the team's overall strategy.



9.2 Algorithm

In this submission, we tackle the **Stable Marriage Problem**, a classic problem in computer science and game theory that focuses on creating a **stable one-to-one matching** between two sets of elements, in our case, agents and roles-based positions on the field.

A matching is considered stable if there is no pair of elements who would both rather be matched to each other than to their current matches. Solving this problem efficiently ensures a robust allocation without any incentive to deviate from the assigned match.

This concept is widely applicable from assigning students to schools, partners in a team, or even matching players to specific AI strategies based on preference.

To solve this problem, we implement the **Gale-Shapley Algorithm**, also known as the **Deferred Acceptance Algorithm**. This algorithm ensures that the final match is stable and fair based on preferences of both sides.

9.2.1 Step 1. Define a Preference Lists

- Begin by creating two equal-sized sets:
  - Set A (`player_position`)
  - Set B (`formation_position`)
- Each element in both sets should have a ranked list of preferences for elements in the opposite set.

```
# Example preference lists for 3 players and 3 roles
players_preferences = {
    0: [2, 0, 1],
    1: [0, 1, 2],
    2: [1, 2, 0]
}

roles_preferences = {
    0: [1, 0, 2],
    1: [0, 2, 1],
    2: [2, 1, 0]
}
```

- Preferences are ranked from most preferred to least preferred.
- Preferences should be calculated based on the relative distance between two positions.

9.2.1.1 Euclidean Distance (Straight-Line)

Suppose each player and formation position has **(x, y)** coordinates.

- **Player:** `(1, 1)`
- **Formation positions:** `(2, 1)`, `(0, 3)`, `(4, 5)`

Distances:
- To `(2, 1)` → $\sqrt{(2-1)^2 + (1-1)^2} = 1$
- To `(0, 3)` → $\sqrt{(0-1)^2 + (3-1)^2} = \sqrt{5} \approx 2.24$
- To `(4, 5)` → $\sqrt{(4-1)^2 + (5-1)^2} = 5$

Preference list (closest first):
`[(2, 1), (0, 3), (4, 5)]`

9.2.2 Step 2. Initialize All Players and Roles as Free

- Create a list of unmatched players.
- Keep track of each role's current match (initially None).

```
unmatched_players = [0, 1, 2]
current_matches = {0: None, 1: None, 2: None}
```

9.2.3 Step 3. Proposals

- While there are unmatched players:
  1. Each unmatched player proposes to their highest-ranked role that they haven't proposed to yet.
  2. Each role reviews their current match and the new proposal.
  - If the role is free, they accept the proposal.
  - If already matched, they choose between the current match and the new proposer based on their preference list.
    - If the new proposer is preferred, the current match is rejected, and the role accepts the new proposer.
    - If the current match is preferred, the new proposer is rejected.

9.2.4 Step 4. Repeat Until Stable

- Continue the proposal and evaluation loop until no unmatched players remain.
- At this point, each player has been matched to a role in a way that no blocking pairs exist.

9.2.5 Step 5. Return Final Matches

- The resulting mapping from players to roles is your final stable matching.

```
# Example output:
# Player 0 matched to Role 2
# Player 1 matched to Role 0
# Player 2 matched to Role 1

stable_matches = {
    0: 2,
    1: 0,
    2: 1
}
```

9.3 Implementation Details

Inside the code-base you will find an `Assignment.py` located inside the `WitsFcCodebase/strategy/` folder. Here you find the following function prototype for the role assignment submission.

```
def role_assignment(teammate_positions, formation_positions):
```

This function receives as input `teammate_positions` and `formation_positions` which you need to use in order output `point_preferences` which is a dictionary where the `key` represents the `unum` (1 to 5) and the associated `value` component being the location this player is assigned to.

9.3.1 Input

```
#Both teammate_positions and formation_positions are a list of 2D positions. Here each element in the list is a ndarray which contains a list object. This list containst two integers which correspond to an x and y coordinate. Therefore
teammate_positions = [ndarray([int,int]), .... , ndarray([int,int])]
```

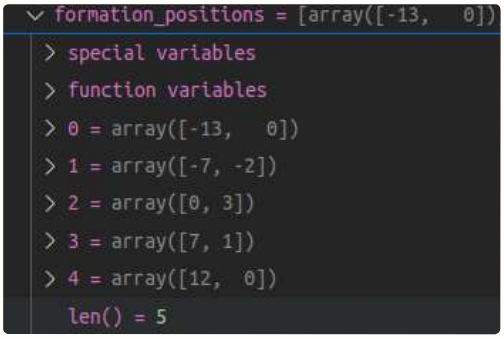
In order to reference individual components of this data-structure you can use the following:

```
teammate_position_1 = teammate_positions[0]
x_position_1 = teammate_position_1[0]
y_position_2 = teammate_position_1[1]

# You can also directly reference the same position as follows
x_position_1 = teammate_positions[0][0]
y_position_1 = teammate_positions[0][1]

# formation_positions is represented in the same form so you can do the same kind of referencing
x_formation_1 = formation_positions[0][0]
y_formation_1 = formation_positions[0][1]
```

The following image demonstrates what an example input looks like when debugging



9.3.2 Output

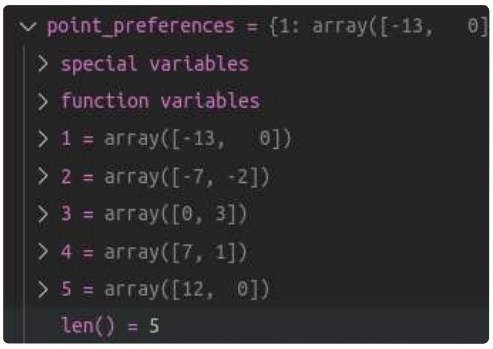
For this assignment you need to return a populated dictionary object called `point_preferences`. This object has been declared for you inside the `role_assignment` method.

```
point_preferences = {}
# point_preferences should have the following structure when returned point_preferences = {key : value, ... , key : value}. Here key is an integer that corresponds to the player id (unum) and value is an ndarray() which contains a list [] (ndarray([])). This list containst two integers which correspond to an x and y coordinate (ndarray[int,int]). Therefore point_preferences = {int : ndarray[int,int], ... , int : ndarray[int,int]}
```

In order to assign a value to this dictionary you can therefore do the following:

```
point_preferences[i] = ([x,y])
```

The following image demonstrates what an example output looks like when debugging



9.4 Submission Requirements

To submit your code

1. Right click on the WitsFcCodebase folder and zip it
2. upload to moodle submission link

« 7 Basics

10 Submission 2 - Full Soccer Team »

"**COMS3005A RoboCup Assignment**" was written by Branden Ingram.

This book was built by the bookdown R package.