Welcome!

Welcome to the online documentation for COMS3005A RoboCup Assignment for the RoboCup Soccer AI Assignment. Soccer is a game which requires complex interplay of strategy, coordination, and real-time decision-making. By developing AI that can navigate these challenges, you'll gain hands-on experience with concepts in artificial intelligence, such as pathfinding, role assignment, and strategic planning.

The accompanying documentation will be your guide throughout this assignment. It includes detailed explanations of the core concepts, sample code to help you get started, and step-by-step instructions for implementing key features of your Soccer AI.

Course Outline

The next section in this book contains the assignment outline, which explains the aim and structure of the assignment and lists other important information such due dates and submission instructions. Please be sure to read it carefully before continuing to the main content itself.

1 Project Outline

1.1 RoboCup

RoboCup is an international scientific initiative with the goal to advance the state of the art of intelligent robots. When established in 1997, the original mission was to field a team of robots capable of winning against the human soccer World Cup champions by 2050. Checkout the RoboCup Website for more information on the different leagues.

If you are interested in watching the final of the 2024 Simulation League in Eindhoven you can watch the video below.

1.2 Project Description

For this project you will be working alone or in pairs in the RoboCup Simulation 3D domain. Which looks to realistically simulate a 5 vs 5 game of soccer using the NAO robot platform.

NB: There is a lot of advice, tips and descriptions provided in this project, make sure to read the whole the document first.

This project is broken down into 2 separate components. The first 2 submissions are to help get you integrated into the code-base as well allow you to solve to fundamental problems related to the game of soccer. These two submissions being:

- Submission 1 Role Assignment - Here we look to solve the problem, given a set of positions on the field how can optimally assign a unique player to each of these positions.
- The last component of this project or Submission 2 Full Team - will see you utilising the knowledge gained from the first submission, such that you can create a full team of players and compete against each other in a league. This league will operate in a similar fashion as your 2nd year project.

1.3 Grading

The weighting for each component is given below

Activity	Weight
Submission 1 - Role Assignment	40%
Submission 2 - Baseline	20%
Submission 2 - Tournament	40%

The agents submitted in submission 2 will first be played against a baseline agent, which is the default behaviour of the code that you download. In order to get that 20%, you must defeat this default behaviour by at least 2 goals.

Your marks in the last section (The final 40%) will be determined by your position in the leaderboard generated by the tournament. Your percentage mark of this 40% will be $\frac{classsize - yourposition}{classsize} * 100$

1.4 Timeline

The table below lists the due dates for each of the submissions.

Activity	Due Date
Submission 1	Check moodle
Submission 2	Check moodle

3 Installation

There are three main components that need to be installed in order to work with RoboCup these are;

1. Server (`rcssserver3d`)
2. Visualizer (`RoboViz`)
3. Agents (`WitsFcCodebase`)

This section describes installation on Windows 10/11 and on Ubuntu 22.04 / 24.04. All other platforms are unsupported, although they may work. |

3.1 Windows (WSL)

The first step on Windows is to install the Windows Subsystem for linux. This can be done by following the instructions at the following microsoft link: https://learn.microsoft.com/en-us/windows/wsl/install Note that in general, you can just type:

```
wsl --install
```

Next, download the Linux image for the Wits Robocup game, by going to: https://lamp.ms.wits.ac.za/robocup/robocup-wsl.tar.gz

Once the install has completed, open a command prompt from the start menu, and use this to install the downloaded image. First go to your Downloads folder by typing: `cd Downloads` Then, install the downloaded image by typing

```
wsl --update
wsl --import robocup c:\robocup robocup-wsl.tar.gz
```

Note that this only needs to be done once. In order to start the robocup image in future, you can just open a command prompt and type

```
wsl -d robocup
```

If this image prompts you for a username and password, the username and password are both robocup. The image does not contain the visualizer that lets you view games. This must be downloaded separately as it is not run from inside the image, but from the underlying windows OS.

The Roboviz visualizer can be downloaded
from: https://github.com/magmaOffenburg/RoboViz/releases/tag/2.0.0

3.2 Linux (Ubuntu 22.04 / 24.04)

On linux, the first thing that must be done is installing the robocup soccer server. This can be done by adding the OpenSuse build repository and installing the robocup server from there.

On Ubuntu 22.04, this is done by entering the following commands:

```
echo 'deb
http://download.opensuse.org/repositories/science:/SimSpark/xUbuntu_22.04/
/' | sudo tee /etc/apt/sources.list.d/science:SimSpark.list curl -fsSL
https://download.opensuse.org/repositories/science:SimSpark/xUbuntu_22.04/R
elease.key | gpg --dearmor | sudo tee
/etc/apt/trusted.gpg.d/science_SimSpark.gpg > /dev/null sudo apt update
sudo apt install unzip wget curl rcssserver3d rcssserver3d-dev
```

On Ubuntu 24.04, you can instead do the following:

```
echo 'deb
http://download.opensuse.org/repositories/science:/SimSpark/xUbuntu_24.04/
/' | sudo tee /etc/apt/sources.list.d/science:SimSpark.list
curl -fsSL
https://download.opensuse.org/repositories/science:SimSpark/xUbuntu_24.04/R
elease.key | gpg --dearmor | sudo tee
/etc/apt/trusted.gpg.d/science_SimSpark.gpg > /dev/null
sudo apt update
sudo apt install rcssserver3d
```

Next, you need to install the dependencies for the Wits FC Robocup team by entering the following command

```
sudo apt install build-essential libgsl-dev python3-numpy python3-bind11
python3-psutil
```

You can then download the Wits FC code by typing the following command:
wget https://lamp.ms.wits.ac.za/robocup/WitsFcCodebase.zip

And you can then unzip the file by typing:

```
unzip WitsFcCodebase.zip
```

The Roboviz visualizer can be downloaded
from: https://github.com/magmaOffenburg/RoboViz/releases/tag/2.0.0

3.3 WITS LABS

If you are attempting to do work in the labs you will need to perform the following steps;

```
conda create -n robocup python=3.11 numpy=1.26.4 pybind11
ln -f -s /usr/lib/x86_64-linux-gnu/libstdc++.so.6
/home/vmuser/anaconda3/envs/robocup/bin/../lib/libstdc++.so.6
ln -f -s /usr/lib/x86_64-linux-gnu/libstdc++.so
/home/vmuser/anaconda3/envs/robocup/bin/../lib/libstdc++.so
```

You can then download the Wits FC code by typing the following command: wget https://lamp.ms.wits.ac.za/robocup/WitsFcCodebase.zip The Roboviz visualizer can be downloaded from: https://github.com/magmaOffenburg/RoboViz/releases/tag/2.0.0

4 Running a Team

If you have not yet please download and unzip the codebase by typing the following command: wget https://lamp.ms.wits.ac.za/robocup/WitsFcCodebase.zip

There are three steps involved in running a team. The first step is to run the robocup soccer server. This can be done by opening a terminal (In the WSL instance if you're on Windows) and typing:

```
rcssserver3d
```

You will see a whole pile of debug messages on the screen at this point, and you can just ignore these. Note that in future, you can run the command in the VSCode terminal instead of in a manual WSL terminal.

4.1 VSCode

The best way to work with this code is in Visual Studio Code. Install it on Windows or Linux if you don't already have it. Then go to the folder which contains the team code (`WitsFcCodebase2025/WitsFcCodebase`) open a terminal in that folder and run:

```
code .
```

4.1.1 Linux

In linux, you can just open the folder in VS Code and work with it from there.

4.1.2 Windows (WSL)

In Windows, you need to install the WSL extension from the extensions screen in WSL. Then connect to WSL by pressing Ctrl-Shift-P and typing WSL in the resulting search box. In the popup menu that results, select WSL:Connect to WSL using Distro in WSL, and select robocup.

Then, select open Folder and select `WitsFcCodebase2025/WitsFcCodebase`

Next, open a terminal, by navigating to Terminal -> New Terminal in VSCode. In the resulting terminal, you can start the agents by typing:

```
./start.sh
```

At this point your team will be running in the game. However, you will not be able to see the game. In order to see the screen, you need to download `RoboViz.tar.gz`, from https://github.com/magmaOffenburg/RoboViz/releases/tag/2.0.0

Note that you will download this on the underlying OS (Windows if you're using WSL), not inside the image. After extracting this, you can run either `roboviz.bat` on Windows or `roboviz.sh` on linux. Note that you will need to have Java (version 17 or higher) installed. You can install a free java jdk from Eclipse Adoptium if you don't have one installed.

Once you have completed these steps you will see a team on the field which resembles the behaviour depicted in the following video. To start the game press `o` on the keyboard and select `Play On`.

5 RoboViz Visualiser

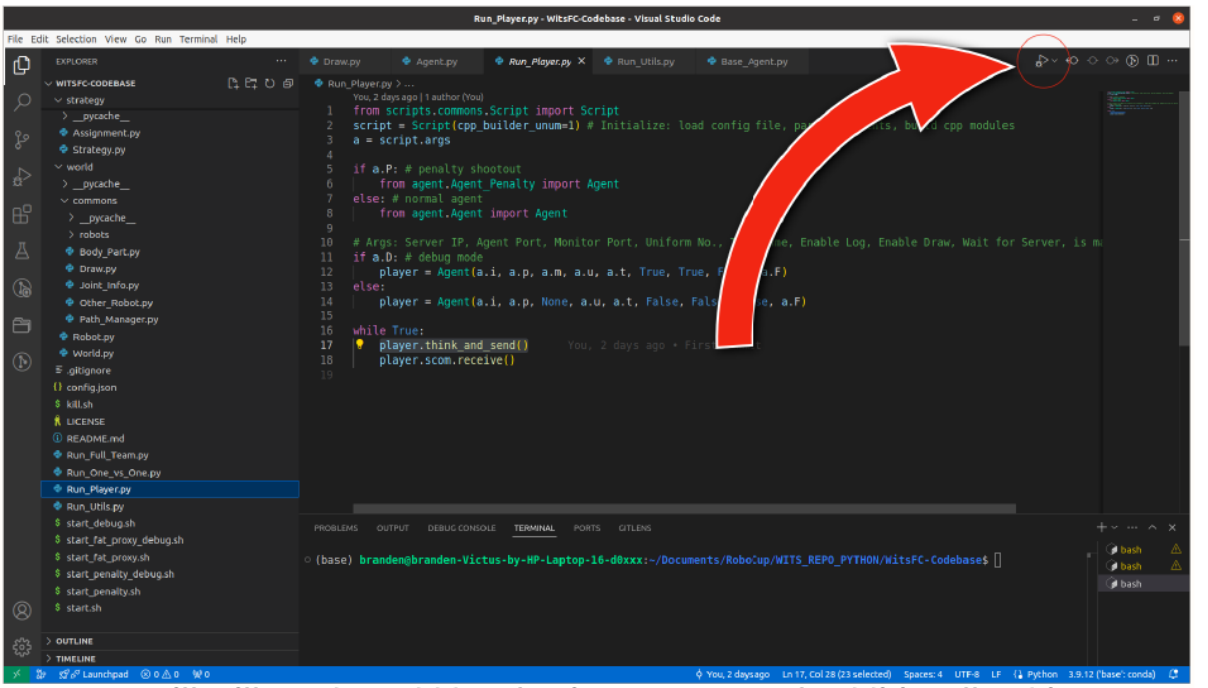
The following image depicts what the RoboViz visualiser would look like if a team was loaded onto the field. The visualiser can be manipulated using both keyboard commands as well as the mouse. Play around with using the arrow keys, holding left click and dragging as well as scrolling with the mouse wheel

In order to start a match we need to change the game mode from `BeforeKickOff` to something else. To do this you can press `o` on the keyboard or by navigating to the top bar, pressing `Server` and then by selecting one of the game modes available. Additionally you can press `8` on the keyboard to change the camera to a top down view of the field.

6 Debugging

Debugging is super useful to help you understand how the different datastructures can be accessed as well as for stepping through any of your implemented behaviours. In Vscode, the first thing you need to do is install the python extension. VSCode should prompt you to do this when you open the codebase. If not, you can go to the extensions tab and install it from there. Then debugging a single agent can be done simply by opening the `Run_Player.py` file and then by pressing the triangle with the small bug icon in the top right of the window as seen below. Note that you must be focused on the `Run_Player.py` file, and if VSCode requires you to select an interpreter at the bottom of the screen, select

```
/usr/bin/python3
```



Note you will still need to add breakpoints as per usual. Additionally, This process will only run a single agent, The next step is in a separate terminal to run `start_debug.sh`. Please make sure that inside `start_debug.sh` that the `teamname` variable matches, otherwise the players will spawn on the wrong side.

6.1 Debugging with Drawings

If you want to use the drawings to help with your debugging, as you should, then just remember you need to run `start_debug.sh`. This is elaborated more in the `Basics` section.

7 Basics

7.1 RoboCup Environment

7.2 Code Structure

The WitsFc Codebase is built using python and contains many files and folders. Although this can be overwhelming you only really need to consider a few particular ones. The sections below outline those particular files.

The execution flow works as follows:

1. `start.sh` is executed using `./start.sh`
2. This script will call `Run_Player.py` for each of your players on the field in its own thread.

3. `Run_Player.py` will instantiate an `Agent.py` object
4. `Run_Player.py` will then call `player.think_and_send()` inside an infinite loop until the process is terminated
    1. Inside `player.think_and_send()` the `strategyData` object will be initialised, we then handle the cases where the player needs to stand up. Finally `select_skill` is called.
    2. Inside `select_skill` we choose between two actions `move` and `kickTarget`. This decision making process will be up to you and what you implement.
    3. Lastly this information is sent to the server and cycle repeats from step 4 above.

7.3 Changing Teamname

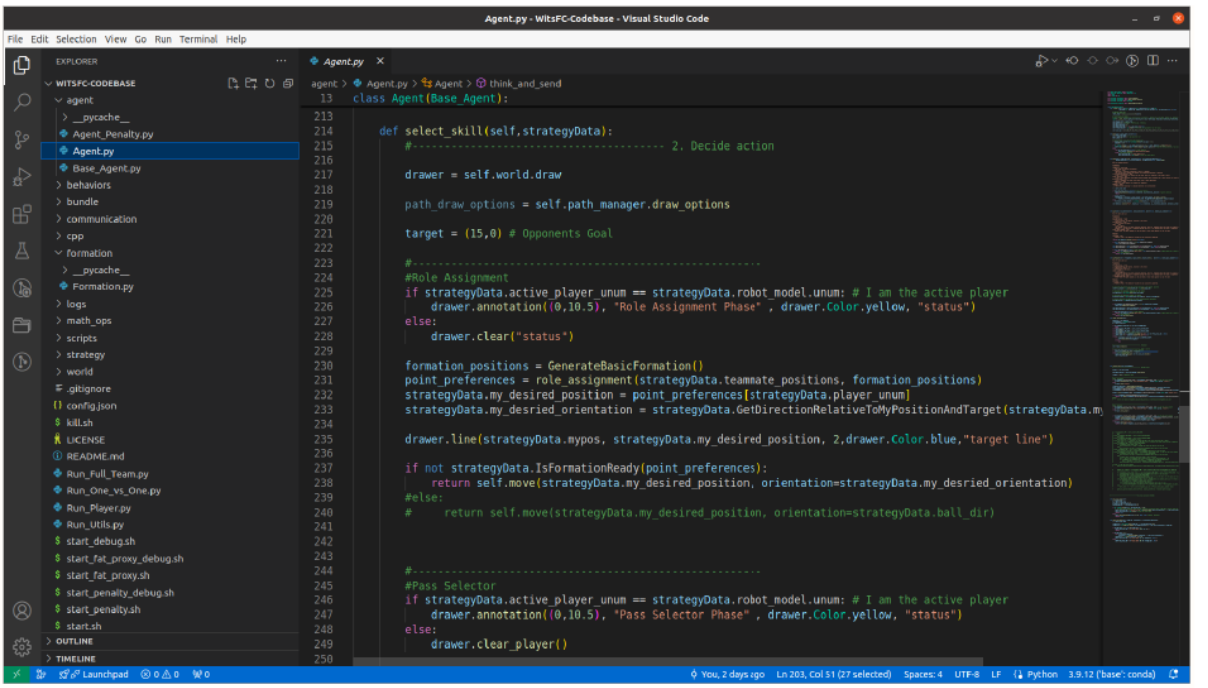
First steps should be to change the teamname that displays when the code runs. Note there is a limited number of characters allowed for this. In order to change the name you will need to update the `StudentName` variable to your name in the following files:

- `config.json`
- `start_debug.sh`
- `start.sh`

The following sections detail different class files within the codebase as well as why they are important.

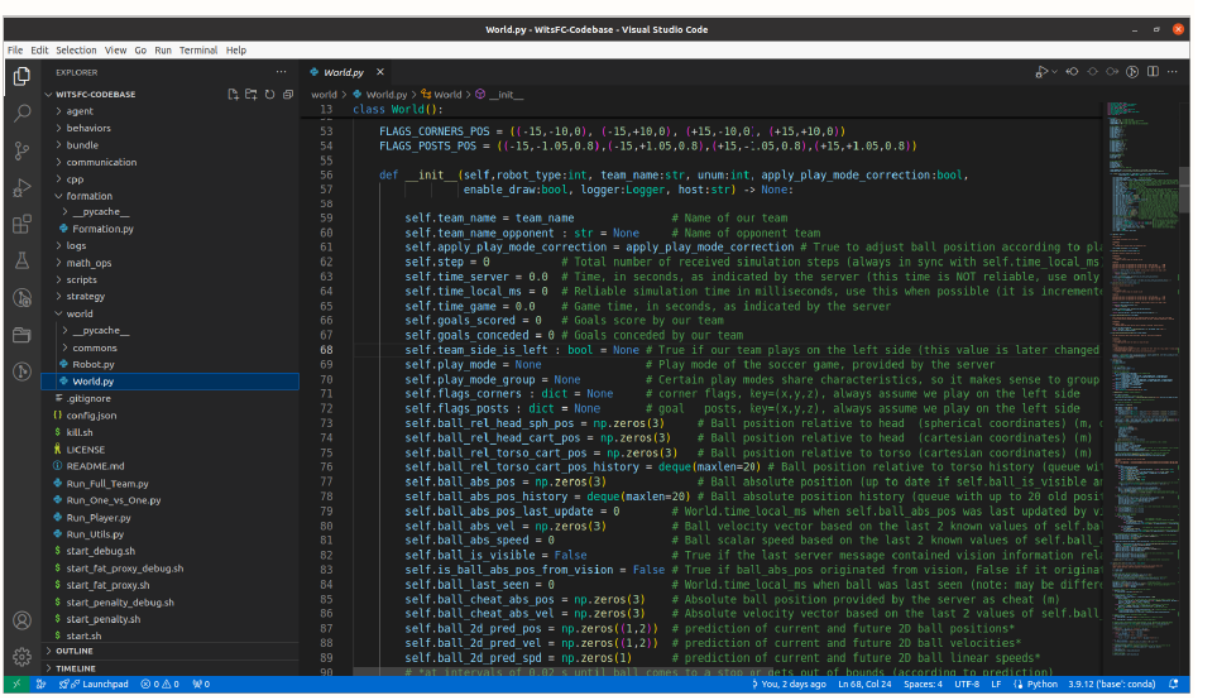
7.3.1 Agent.py

This file found in `WitsFcCodebase/agent/` contains the primary function in which you will be working, this being `select_skill` as well as the implementations of the `move` and `kickTarget` action, although these should not be altered.



7.3.2 World.py

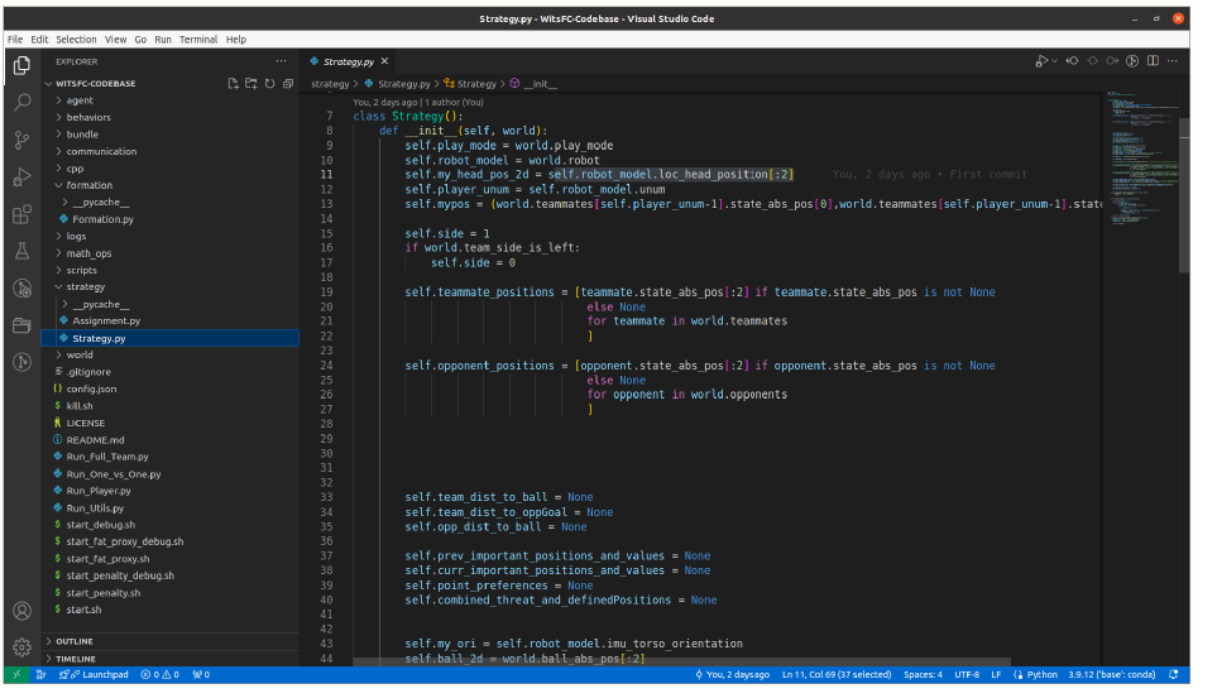
This file found in `WitsFcCodebase/world/` contains all the information regarding the current state of the game. This includes information like the current score and game mode as well as the information about each individual player on the field.



7.3.3 Strategy.py

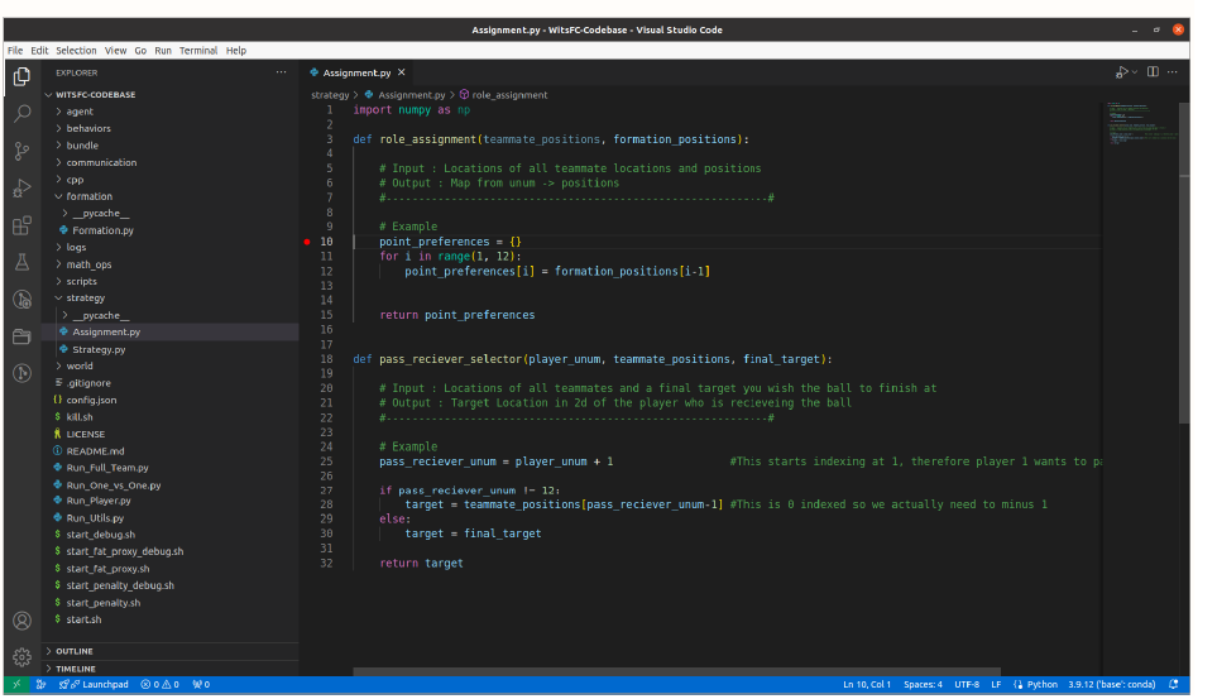
This file found in `WitsFcCodebase/strategy/` contains the implementation of the strategy class which contains useful datastructures for decision making such as a list of (x,y) coordinates called `teammate_positions` which correspond to the

positions of all your teammates. This class can be considered a subset of the `World` class with extra variables useful for designing an intelligent soccer team. This class also has some useful methods and is where you should implement all your helper functions.



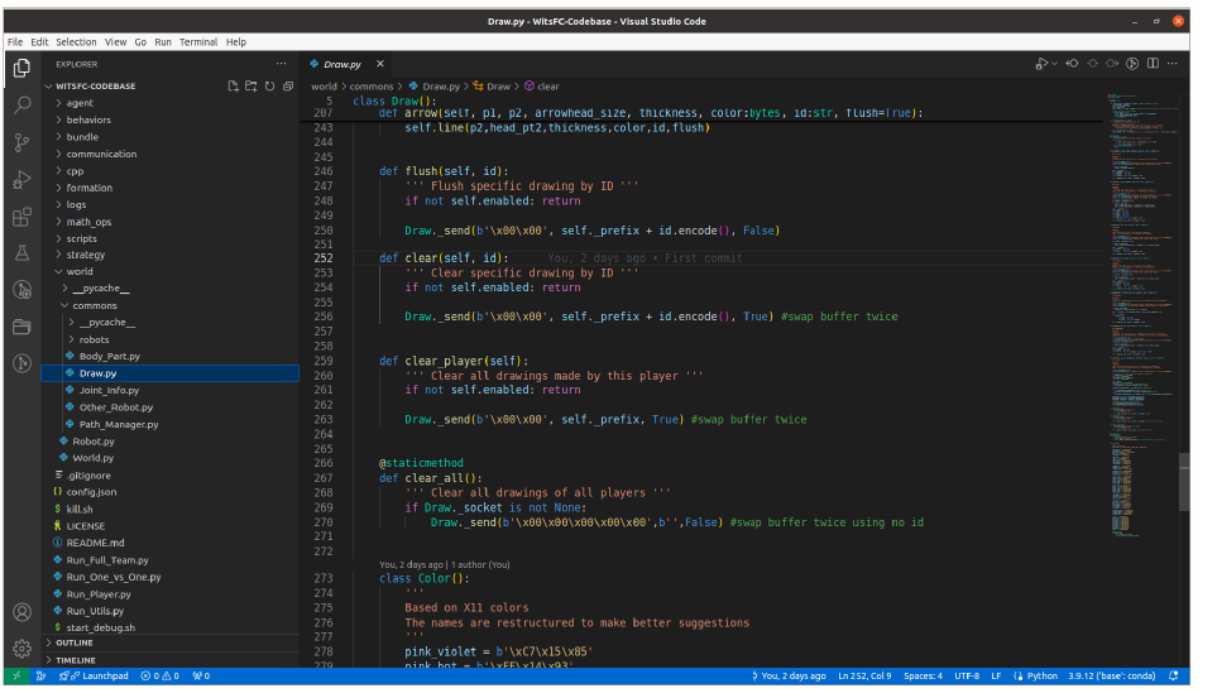
7.3.4 Assignment.py

This file found in `WitsFcCodebase/strategy/` contains the function prototypes that you will need to implement for Submission 1.



7.3.5 Draw.py

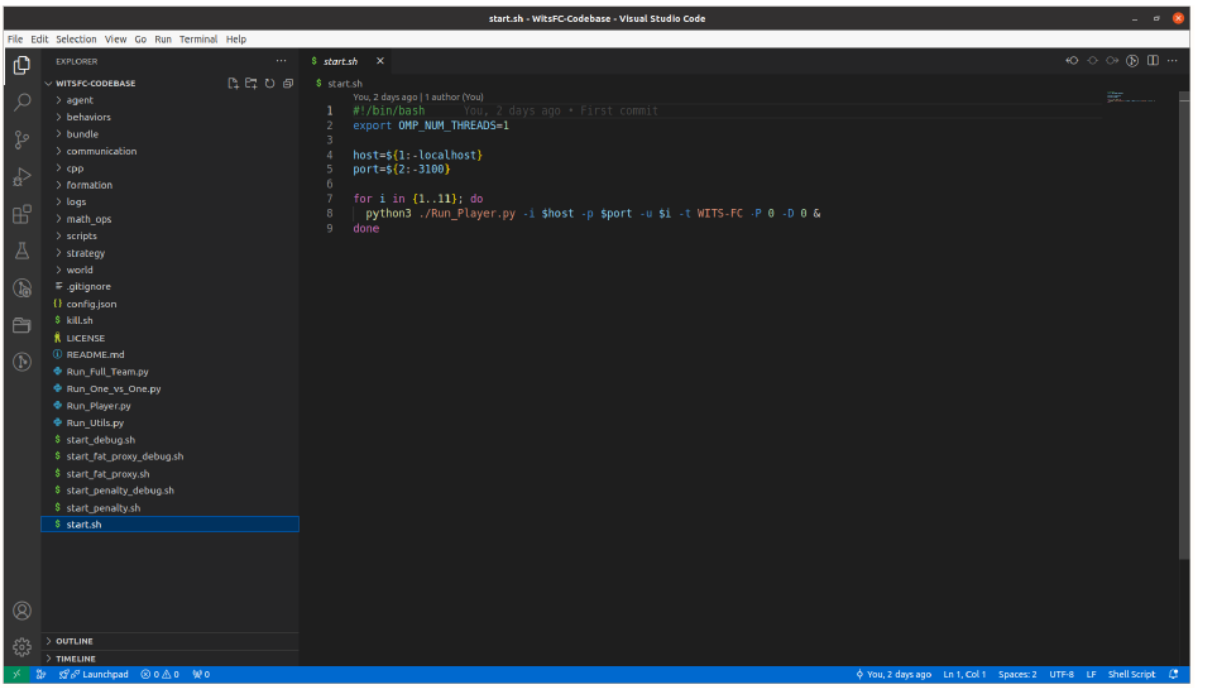
It is very useful to use drawn visualisation to depict the behaviour in which you are trying to implement. This file found in `WitsFcCodebase/world/commons/` contains the functions which you can call inside your code to draw objects like `text`, `lines`, `circles` ect.



7.3.6 Run Scripts

In order to run your team whether for testing or when we evaluate them in the tournament we utilise run scripts. There are a number of these provided which can be used for different things

- `start.sh` - This is the default script that can be used to run a single team using `./start.sh` in the terminal.
- `start_debug.sh` - This performs the same operation as `start.sh`, however, it will allow drawings to be rendered. This if you want to the drawings from `draw.py` you need to run this script using `./start_debug.sh`
- `start_debug_teammates.sh` - This launches 4 players and should be run in conjunction to the debugging process described in the debugging section.
- `Run_Player.py` - You can also run an individual player directly by running `python Run_Player.py` in the terminal. This can be useful for debugging purposes.
- `Run_Two_Teams.py` - This functions the same as `Run_Player.py` but for 5 players and for both teams. However, this just runs your current team as the opponent. However be adviced each of these python process will be run on the same thread and will have major performance issues.
- `Run_Full_Team.py` - This functions the same as `Run_Player.py` but for 5 players. However be adviced each of these python process will be run on the same thread and will have major performance issues.



7.4 Primitive Actions

Below describes, the only two actions you have available to use, which will need to be returned from the `select_skill` function. Meaning at some point in your `select_skill` function you must either return `move()` or `kickTarget()`. Failing to do so will result in undesired behaviours such as crashing or falling over.

7.4.1 move

The move action allows a player to walk towards a given target. This is handled by calling the `move` function given a `target_2d` which is an array of values. For example `(10,0)` if we called `move((10,2))` the agent would move to coordinate `x=10` and `y=2`. This is basic usage of the function, however, it can be expanded to include more components such as the ones shown below:

```
def move(self, target_2d=(0,0), orientation=None, is_orientation_absolute=True,
         avoid_obstacles=True, priority_unums=[], is_aggressive=False, timeout=3000):
    '''
    Walk to target position

    Parameters
    ----------
    target_2d : array_like
        2D target in absolute coordinates
    orientation : float
        absolute or relative orientation of torso, in degrees
        set to None to go towards the target (is_orientation_absolute is ignored)
    is_orientation_absolute : bool
        True if orientation is relative to the field, False if relative to the robot's torso
    avoid_obstacles : bool
        True to avoid obstacles using path planning (maybe reduce timeout arg if this function is called multiple
    priority_unums : list
        list of teammates to avoid (since their role is more important)
    is_aggressive : bool
        if True, safety margins are reduced for opponents
    timeout : float
        restrict path planning to a maximum duration (in microseconds)
    '''
```

You will see in the template code provided that there are a few calls to the `move` function found in the `select_skill`.

```
if not strategyData.IsFormationReady(point_preferences):
    return self.move(strategyData.my_desired_position, orientation=strategyData.my_desried_orientation)
```

In the above example we tell the player to walk to the location assigned to `my_desired_position` while maintaining an orientation facing `my_desired_orientation`.

7.4.2 kickTarget

The second action relates to kicking the ball. This is performed by making a call to the `KickTarget` function. This function takes as input the `strategyData` object, a 2D array corresponding to `mypos_2d` as well as 2D array corresponding to a kick target location represented by `target_2d`.

```
def kickTarget(self, strategyData, mypos_2d=(0,0),target_2d=(0,0), abort=False, enable_pass_command=False):
    '''
    Walk to ball and kick

    Parameters
    ----------
    strategyData : object
        containing game state variables
    mypos_2d : array_like
        2D target in absolute coordinates
    target_2d : array_like
        2D target in absolute coordinates
    abort : bool
        True to abort.
        The method returns True upon successful abortion, which is immediate while the robot is aligning itself.
        However, if the abortion is requested during the kick, it is delayed until the kick is completed.
    avoid_pass_command : bool
        When False, the pass command will be used when at least one opponent is near the ball

    Returns
    -------
    finished : bool
        Returns True if the behavior finished or was successfully aborted.
    '''
```

You will see in the template code provided that there are a few calls to the `kickTarget` function found in the `select_skill`.

```
#---------------------------------------------------
# Example Behaviour
target = (15,0) # Opponents Goal

if strategyData.active_player_unum == strategyData.robot_model.unum: # I am the active player
    drawer.annotation((0,10.5), "Pass Selector Phase" , drawer.Color.yellow, "status")
else:
    drawer.clear_player()

if strategyData.active_player_unum == strategyData.robot_model.unum: # I am the active player
    pass_reciever_unum = strategyData.player_unum + 1 # This starts indexing at 1, therefore player 1 wants to pass to player 2
    if pass_reciever_unum != 6:
        target = strategyData.teammate_positions[pass_reciever_unum-1] # This is 0 indexed so we actually need to minus 1
    else:
        target = (15,0)

    drawer.line(strategyData.mypos, target, 2,drawer.Color.red,"pass line")
    return self.kickTarget(strategyData,strategyData.mypos,target)
else:
    drawer.clear("pass line")
    return self.move(strategyData.my_desired_position, orientation=strategyData.ball_dir)
```

Here we determine a pass recipient (`pass_reciever_unum`) to be the player whose player number is one plus the closest players number. Calculated as:

```
pass_reciever_unum = strategyData.player_unum + 1 # This starts indexing at 1, therefore player 1 wants to pass to player 2
```

From that we determine the `target` to be the location of the `pass_reciever_unum` found in `teammate_postions`. Calculated as:

```
target = strategyData.teammate_positions[pass_reciever_unum-1] # This is 0 indexed so we actually need to minus 1
```

Finally we kick the ball into the goal is Player 11 has the ball as seen below:

```
target = (15,0)
```

If the player returning the `kickTarget` action is not within range to actually successfully perform a kick then the player will first walk towards the ball until they can kick. This is handled for you and does not need to be taken into consideration.

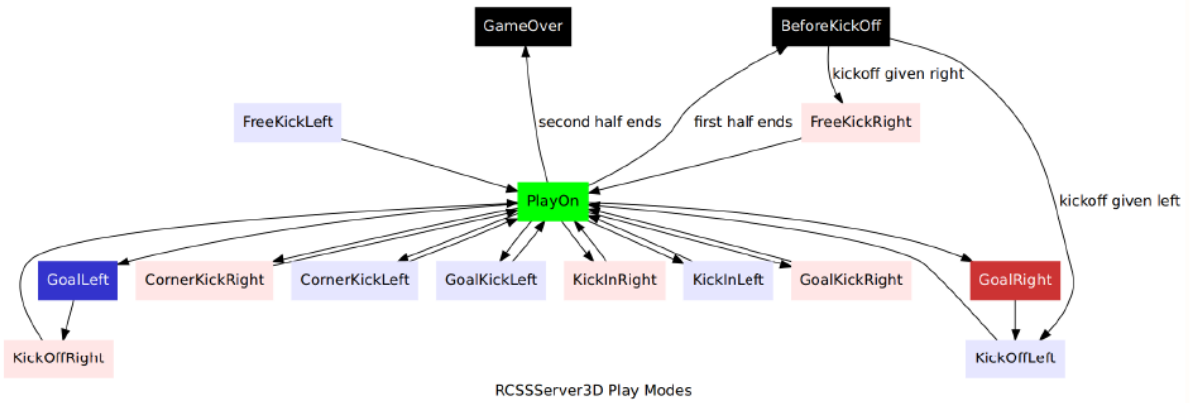
Note: The passing behaviour described above constitutes the behaviour of Baseline 1

7.4.3 getup

If your player falls over for any reason they will begin to execute the `getup` behaviour. Therefore you do not have to worry about it.

7.5 Game Modes

During the course of a match the players will have to adapt to number of different game modes. These include ones related to Corner Kicks, Goal Kicks, Kick Ins ect. For the most part the game will run in what is called Play On. However, as seen below there can many transitions to and from different game modes.



RCSSServer3D Play Modes

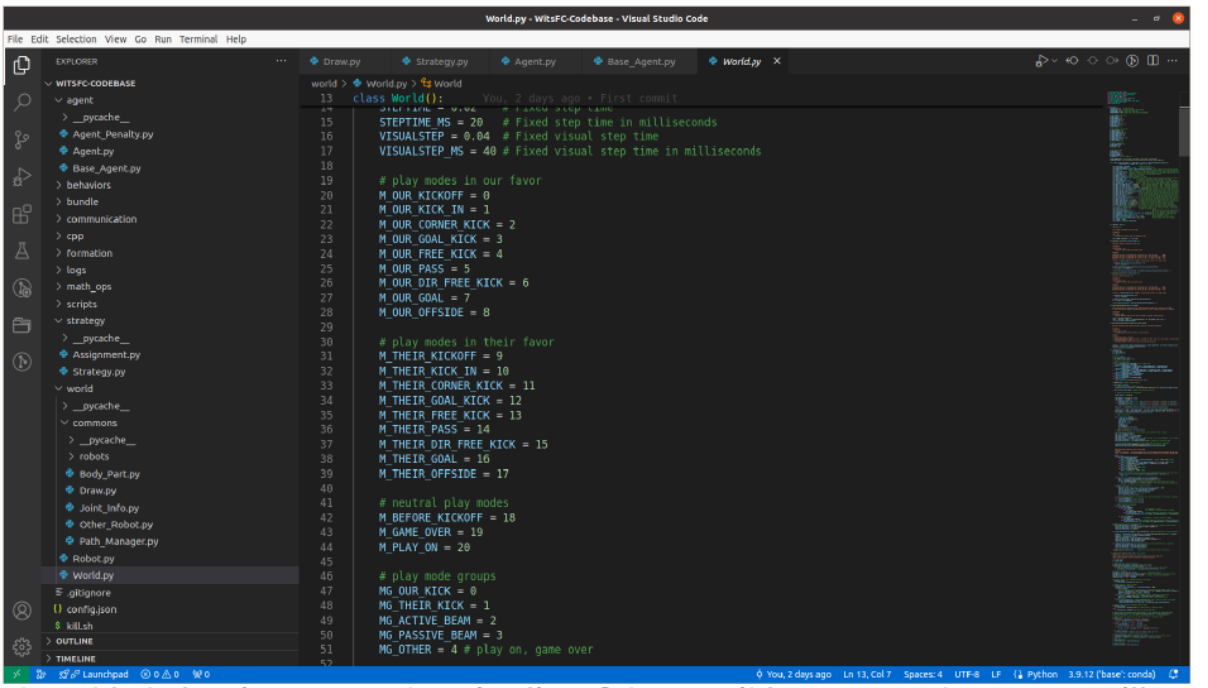
The current play mode value can be referenced from inside the `Strategy` class and is stored inside the `play_mode` variable as seen below.

```
class Strategy():
    def __init__(self, world):
        self.play_mode = world.play_mode
        self.robot_model = world.robot
```

This variable can then be compared against a set of all game mode constants which is stored inside the `World` class. Below we can see that we are trying to check if the current `play_mode` is `M_GAME_OVER`. This example is found in the `think_and_send` function inside `Agent.py` .

```
if strategyData.play_mode == self.world.M_GAME_OVER:
    pass
```

The figure below shows the names given to all the different play modes stored as constants within the `World` class.



The table below is a comprehensive list of the possible game modes a team will encounter.

Play mode	Description	Conditions
BeforeKickOff	Before the match	The ball is at (0,0), the midfield and may not be moved. Players may use their beam effectors. Game time does not progress. This state is only left when a user instructs the simulator to start.
KickOff_Left	Kick off for the left team	The left team have a period in which to make their first kick. During this time the right team are lot allowed to cross the centre line, or inside the centre circle. The left team may not cross the centre line, unless they are within the goal circle.
KickOff_Right	Kick off for the right team	The right team have a period in which to make their first kick. During this time the left team are lot allowed to cross the centre line, or inside the centre circle. The right team may not cross the centre line, unless they are within the goal circle.
PlayOn	Regular gameplay	
KickIn_Left	Kick in left team	The right team have kicked the ball off the side of the field. The ball is positioned on the sideline at the position it left the field. The right team are not allowed within a fixed radius of the ball, and the left team have a period of time in which to kick the ball back into play.
KickIn_Right	Kick in right team	The left team have kicked the ball off the side of the field. The ball is positioned on the sideline at the position it left the field. The left team are not

Play mode	Description	Conditions
		allowed within a fixed radius of the ball, and the right team have a period of time in which to kick the ball back into play.
CORNER_KICK_LEFT	Corner kick left team	
CORNER_KICK_RIGHT	Corner kick right team	
GOAL_KICK_LEFT	Goal kick for left team	
GOAL_KICK_RIGHT	Goal kick for right team	
OFFSIDE_LEFT	Offside for left team	Currently unused.
OFFSIDE_RIGHT	Offside for right team	Currently unused.
GameOver	After the match	Play has finished. Agents may still move about, but no actions will have an effect upon the result of the match.
Goal_Left	Goal scored by the left team	This state exists for a few moments, before transitioning to KickOff_Right.
Goal_Right	Goal scored by the right team	This state exists for a few moments, before transitioning to KickOff_Left.
FREE_KICK_LEFT	Free kick for left team	The right team are not allowed within a fixed radius of the ball, and the left team have free access. PlayOn commences when the left team touches the ball, or if they fail to do so after a fixed period.
FREE_KICK_RIGHT	Free kick for right team	The left team are not allowed within a fixed radius of the ball, and the right team have free access. PlayOn commences when the right team touches the ball, or if they fail to do so after a fixed period.
NONE	No or unknown play mode	Agents should never receive this play mode.

7.6 Field Layout

The following Figures depict the field as well as give you a better visualistation of the coordinate system in place.

In the figure above you can see the visualisation of the field before any agents have been added. On the field you can also see 30 visual stripes of alternating green shades. Now these stripes aren't only to simulate that grass effect of a real field but also as a visual key to see different positions on the field. Since our field width denoted by FIELD_X = 30 is equal to 30 each stripe represents a unique integer value from -HALF_FIELD_X to HALF_FIELD_X or -15 to 15 with the center being at FIELD_CENTER_X or 0. Additionally we have drawn lines on the field between two pairs of points namely (-5,10),(-5,-10) and (5,10),(5,-10).

An Important thing to note is that this visualisation would be flipped for the team loaded in on the right hand side. This means that the position of your own goal is always at (-15,0) and the opponents goal is always at (15,0). The figure below depicts the visualisation of those same coordinates from the perspective of the team on the right hand side. The left hand side team does not draw anything to the visualiser. The same code as above was utlised.

7.7 Initial Position of Players

When you load a team you will find that the individual players are all placed at certain locations. This can be considered there `initial formation`. These initial positions can be modified by modifying the `init_pos` variable in the `Agent` class inside `Agent.py`. Currently you will find the following initial formation:

```
self.init_pos = ([-13,0],[-7,-2],[0,3],[7,1],[12,0])[unum-1] # initial formation
```

This assigns a 2d position for each of our 5 players for example `Player 1` will be assigned an initially position of `[-13,0]`.

7.8 Rule and Fouls

There are multiple “cardable” offenses and fouls which a player can recieve. In such cases the penalised player is teleported to the side of the field where they will then be able to instantly come back on.

7.8.1 Kick ins

There are a number of game modes which result in a kick in needing to be performed. The list of these can be seen below:

```
# play modes in our favor
M_OUR_KICKOFF = 0
M_OUR_KICK_IN = 1
M_OUR_CORNER_KICK = 2
M_OUR_GOAL_KICK = 3
M_OUR_FREE_KICK = 4
M_OUR_PASS = 5
M_OUR_DIR_FREE_KICK = 6
M_OUR_GOAL = 7
M_OUR_OFFSIDE = 8

# play modes in their favor
M_THEIR_KICKOFF = 9
M_THEIR_KICK_IN = 10
M_THEIR_CORNER_KICK = 11
M_THEIR_GOAL_KICK = 12
M_THEIR_FREE_KICK = 13
M_THEIR_PASS = 14
M_THEIR_DIR_FREE_KICK = 15
M_THEIR_GOAL = 16
M_THEIR_OFFSIDE = 17
```

In such situations a circle will form around the ball which only allows the team performing the kick in, to go inside it. This is to prevent the opposition from disrupting the play.

Note you can not score directly from any Kick In mode

7.8.2 Double Touch

During set plays (Kick Ins, Goal Kicks, Kick Off ect.) the same player can not touch the ball twice, in such cases the team who did not commit the offence will recieve a KICK IN of their own.

7.8.3 Charging and Touching

This is a cardable offensive which results in the offending player being beamed off the field. It occurs when an player bumps into the back side of another player This can sometimes be quite fickle and we have seen that other forms of contact such as side on and even accidental contact can lead to a charging offensive.

7.8.4 offside

There is no rule governing offsides, however, during and before kick off you cannot move into the opponents half. Failing to do so will result in the offending player being teleported(beamed) back onto their own half.

7.9 Drawing Visualisation Aids

The below video which corresponds to an example behaviour which firstly naivelly performs role assignment and then pass selection demonstrates the drawing capabilities of the codebase.

Here we can see examples of text and line drawing. text drawing is handled by the `annotation` function which can be found in the `Draw.py` file. It is implement as follows:

```
def annotation(self, pos, text, color:bytes, id:str, flush=True):
    '''
    Draw annotation

    Examples
    ----------
    Annotation in 3D: annotation((1,1,1), "SOMEtext!", Draw.Color.red, "my_annotation")
    Annotation in 2D (z=0): annotation((1,1), "SOMEtext!", Draw.Color.red, "my_annotation")
    '''
    if not self.enabled: return
    if type(text) != bytes: text = str(text).encode()
    assert type(color)==bytes, "The RGB color must be a bytes object, e.g. red: b'\xFF\x00\x00'"
    z = pos[2] if len(pos)==3 else 0

    if self._is_team_right:
        pos = (-pos[0],-pos[1],pos[2]) if len(pos)==3 else (-pos[0],-pos[1])

    msg = b'\x02\x00' + (
    f'{f"{pos[0]  :.4f}":.6s}'
    f'{f"{pos[1]  :.4f}":.6s}'
    f'{f"{z       :.4f}":.6s}').encode() + color + text + b'\x00'

    Draw._send(msg, self._prefix + id.encode(), flush)
```

Here you can see it takes a position where it should be rendered as well as the actual text to be displayed, the colour it should be written in and a unique identifier for the drawing. This identifier can be used to edit or clear the drawing later.

The second example drawing is that of a line, which once again is found in the `Draw.py` file and is implemented as follows :

```
def line(self, p1, p2, thickness, color:bytes, id:str, flush=True):
    '''
    Draw line

    Examples
    ----------
    Line in 3D: line((0,0,0), (0,0,2), 3, Draw.Color.red, "my_line")
    Line in 2D (z=0): line((0,0), (0,1),   3, Draw.Color.red, "my_line")
    '''
    if not self.enabled: return
    assert type(color)==bytes, "The RGB color must be a bytes object, e.g. red: b'\xFF\x00\x00'"
    assert not np.isnan(p1).any(), "Argument 'p1' contains 'nan' values"
    assert not np.isnan(p2).any(), "Argument 'p2' contains 'nan' values"

    z1 = p1[2] if len(p1)==3 else 0
    z2 = p2[2] if len(p2)==3 else 0

    if self._is_team_right:
        p1 = (-p1[0],-p1[1],p1[2]) if len(p1)==3 else (-p1[0],-p1[1])
        p2 = (-p2[0],-p2[1],p2[2]) if len(p2)==3 else (-p2[0],-p2[1])

    msg = b'\x01\x01' + (
    f'{f"{p1[0]  :.4f}":.6s}'
    f'{f"{p1[1]  :.4f}":.6s}'
    f'{f"{z1     :.4f}":.6s}'
    f'{f"{p2[0]  :.4f}":.6s}'
    f'{f"{p2[1]  :.4f}":.6s}'
    f'{f"{z2     :.4f}":.6s}'
    f'{f"{thickness :.4f}":.6s}').encode() + color

    Draw._send(msg, self._prefix + id.encode(), flush)
```

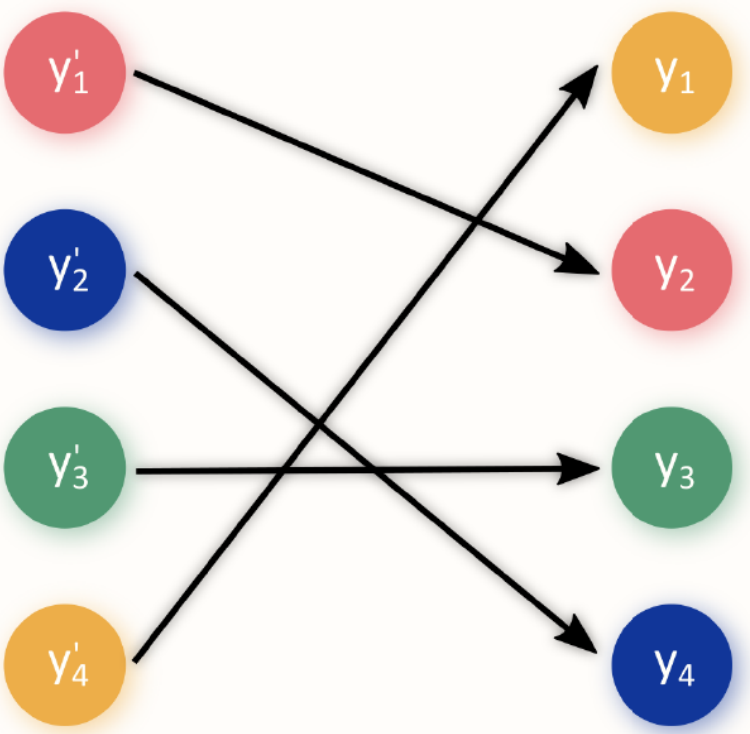
Both these functions can be seen in action in the Pass Selector section of `select_skill`, here we use the `drawer` object to access the drawing functions. Note the usage of `clear_player` and `clear` for the purpose of clearing all the drawings for a particular player as well as clearing a particular drawing based on an id (`pass line`).

```
#---------------------------------------------------
#Pass Selector
if strategyData.active_player_unum == strategyData.robot_model.unum: # I am the active player
    drawer.annotation((0,10.5), "Pass Selector Phase" , drawer.Color.yellow, "status")
else:
    drawer.clear_player()



if strategyData.active_player_unum == strategyData.robot_model.unum: # I am the active player
    target = pass_reciever_selector(strategyData.player_unum, strategyData.teammate_positions, (15,0))
    drawer.line(strategyData.mypos, target, 2,drawer.Color.red,"pass line")
    return self.kickTarget(strategyData,strategyData.mypos,target)
else:
    drawer.clear("pass line")
    return self.move(strategyData.my_desired_position, orientation=strategyData.ball_dir)
```

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.

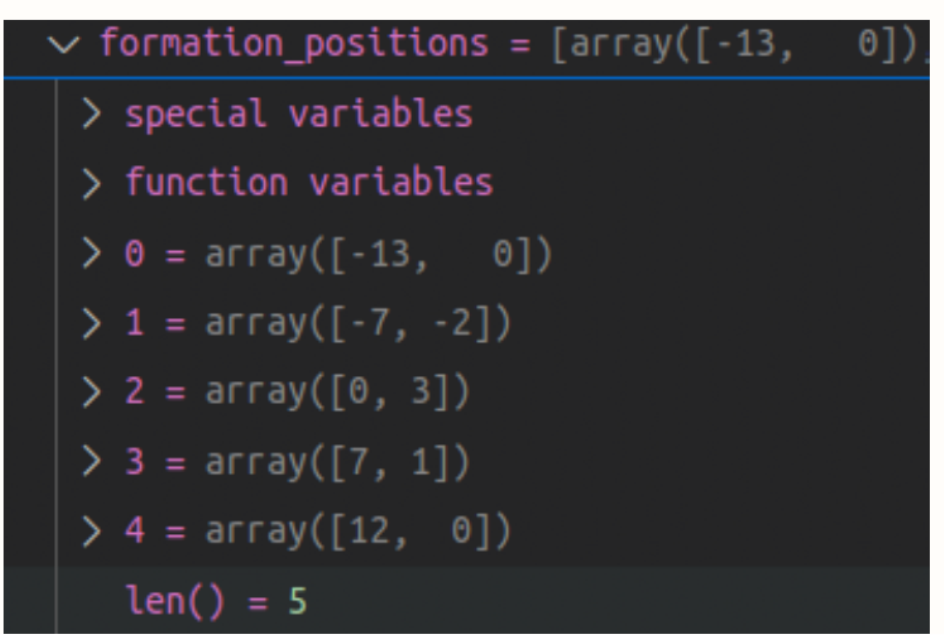
9.3.1 Input

```
#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

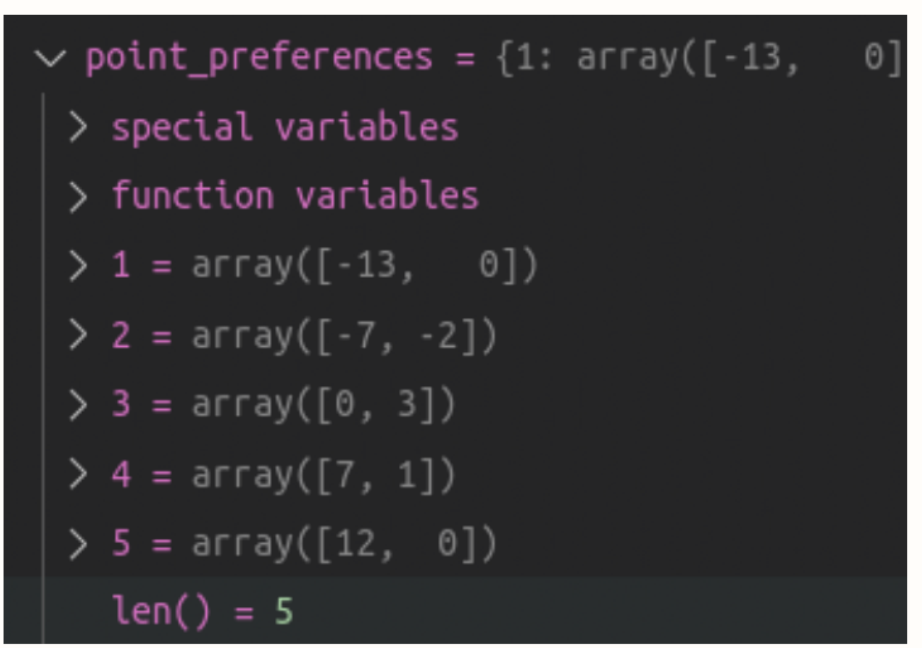
For this assignment you need to return a populated dictionary object called `point_preferences`. This object has been declared for you inside the `role_assignment` method.

```
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

10 Submission 2 - Full Soccer Team

10.1 Introduction

The last submission for this assignment involves you taking what you done for the previous submission and expanding it to develop the artificial intelligence of a whole team which is capable of playing soccer. This means that you need to expand your team to handle different game modes, consider opponents positions, generate formations all in aid to develop a team capable of beating your friends and more importantly your enemies.

10.2 Implementation Details

The RoboCup code base comes down to making a choice between two actions; walking to a target or kicking to a target. Ultimately your team needs to perform the decision making to handle making that choice. Therefore, no matter what the game state is your code inside the `select_skill` function needs to return `kickTarget` or `move` with the appropriate parameters.

Feel free to create and add any additional `.py` files as long as they exist somewhere inside the project folder.

Things to consider:

1. Your code only has a limited amount of time to perform task (). If your code exceeds this time, it will not crash but rather the players will fall over frequently or behave abnormally.
2. Make sure you have behaviours that can handle the different game modes, we recommend a simple function that can handle the different groups of game modes.
3. In the first submission you looked to find an optimal assignment of players to positions, now you will need to consider how can we create a set of positions. To this end you we recommend considering the following:
    - Hardcoding an intelligent formation
    - Perhaps your formation should be dependent on where the ball is
4. Another aspect revolves around decision making, what should each team member do and when. This can be handled by asking questions of the world. For example `am I the closest player to the ball` or `is there a teammate nearby`. Incorporating these types of decision making mechanism can be achieved using the following:
    - Finite State Machines
    - Decision Trees
    - If Statements ;)

10.3 Fixing Falling Over

The reason why your players are falling over is that your code is too slow to finish in a single server tick. One solution which we will allow is to run the server in asyn mode. Which means the server will wait for all players to send a message before moving on.

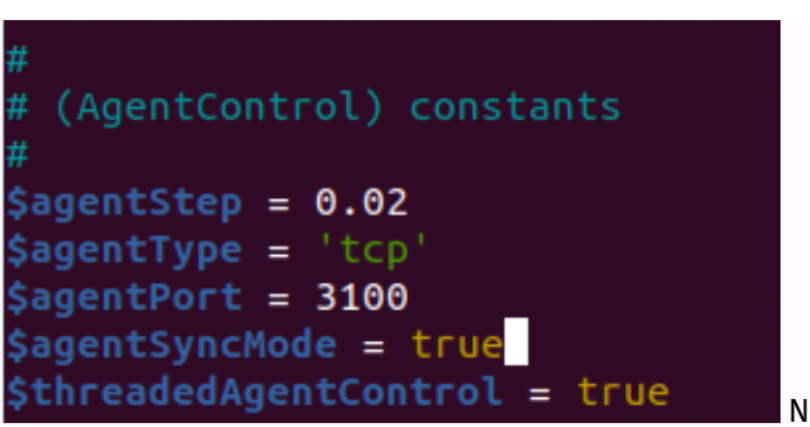
To enable this you need to follow the following steps:

1. Inside the same terminal that you would use to run the server type the following command:

```
nano ~/.simspark/spark.rb
```

This will open the following looking file

2. Use the arrow keys to navigate down the file until you get to `$agentSyncMode = false`. You will need to change this to TRUE as can be seen below



Note for the following instructions the precise key commands will depend on which program you are

using as well as if you are using Vscode from inside WSL. The main thing you need to do is execute `Write-Out` followed by `Exit`.

3. Then type `Ctrl-O` to save. It will ask if you are sure then press `Enter`.
4. Lastly type `Ctrl-X` to close the file.

Now you should be able to run your agents normally, if they are still falling over then there is an issue with your code.

10.4 Submission Requirements

To submit your code

1. First you need to change the team_name. To do this open `start.sh` and modify the string that says `StudentName`. This can be found on line 8, simply replace that with your team name. For simplicity, please use your student number as the teamname.

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

10.5 Tournament Structure

Once your submission has been made your team will be added into the pool of other teams. The tournament uses a Swiss style structure which is a popular format used in various competitive settings, including chess, eSports, card games, and academic competitions. It is designed to pair participants against others with similar performance levels, ensuring a fair and competitive experience for all players throughout the tournament.

10.5.1 Key Features of the Swiss Tournament Structure:

1. Fixed Number of Rounds:

- The tournament consists of a predetermined number of rounds, regardless of the number of participants. Typically, the number of rounds is chosen based on the number of participants to ensure that a clear winner can be determined.

2. No Elimination:
   - Unlike knockout tournaments, no player is eliminated after a loss. All participants play in each round, allowing everyone to compete in all rounds of the tournament.
3. Pairing Based on Performance:
   - In the first round, participants are usually paired randomly or based on seeding (if prior rankings are available). In subsequent rounds, players are paired against opponents with similar scores (e.g., a player with 3 wins and 1 loss will be paired with another player who has 3 wins and 1 loss).
   - The goal is to match players who are performing similarly, ensuring more balanced and competitive matches as the tournament progresses.
4. Scoring System:
   - Participants earn points based on their performance in each round (e.g., 1 point for a win, 0.5 points for a draw, and 0 points for a loss). The total score after all rounds determines the final rankings.
   - Tiebreakers, such as the Buchholz system (sum of opponents' scores), may be used to rank players with identical scores.
5. Final Standings:
   - After the predetermined number of rounds, participants are ranked based on their total score. The player with the highest score is declared the winner. If there is a tie, tiebreaker methods are applied to determine the final standings.