ROBOCUP - Soccer Simulator2D Research Project

Summer 2024, Final Report

Khaled Boubekeur, Zhiheng Zhu under the supervision of Joseph Vybihal

School of Computer Science
Faculty of Science
McGill University

August 15th, 2024

1 Introduction

RoboCup is an international robotics competition with the aim of developing autonomous robots that can compete in multiple disciplines such as in the Soccer Simulator2D. The ultimate goal of the RoboCup Soccer2D initiative is to develop a team of fully autonomous robots that compete against other teams. However, as the previous person who worked on this project noticed there was a lack for proper team coordination and strategy. This project aims to address this issue by getting the robots to work together as a team and to develop a strategy that will allow the team to communicate and coordinate effectively. Thus, our objective was to develop a team of robots that can work together and communicate effectively as well as to create new player's positions and roles that will allow the team to have better goal scoring opportunities and to better defend against the opposing team.

2 Setup

2.1 Environment preparation

For the code to run properly, it is necessary to follow the instructions for the operating system you are using. The setup was tested on Ubuntu 22.04, MacOS Sequoia 15.1 and Windows 11 with 'rcssserver-19.0.0' and 'rcssmonitor-19.0.1'.

Linux Installation :

1. Install the dependencies.

```
sudo apt update
sudo apt install build-essential automake autoconf libtool
flex bison libboost-all-dev
```

2. Download the latest releases of the server and the monitor.
3. Extract the files.
4. Run `./configure` followed by `make` in both the `rcssserver-[version no]` and `rcssmonitor-[version no]` directories.

MacOS Installation :

1. Install the dependencies as described in the README[1] (check that Bison version is the latest one and not 2.x.x using bison -V).

2. Download the latest releases of the server and the monitor.

3. Set the environment variables for Qt5:

```
export LDFLAGS="-L/opt/homebrew/opt/qt@5/lib"
export CPPFLAGS="-I/opt/homebrew/opt/qt@5/include"
export PKG_CONFIG_PATH="/opt/homebrew/opt/qt@5/lib/pkgconfig"
echo 'export PATH="/opt/homebrew/opt/qt@5/bin:$PATH"' >> ~/.zshrc
```

4. Run `./configure --with-boost="/opt/homebrew/Cellar/boost/[version-no]"` and `make` in both the `rcssserver-[version no]` and `rcssmonitor-[version no]` directories (Please note that depending on your MAC version non-M-chip Macs, you might have to change /opt/homebrew to /usr/local/ also type this command manually don't copy paste).

Windows Installation :

The installation process for Windows is different. Refer to the tutorial by Carleton University for detailed instructions.

2.2 Running the code

2.2.1 Clone the Code

1. Clone the code from the GitLab repository in the same directory as the server and the monitor:

```
git clone https://gitlab.cs.mcgill.ca/jvybihal/soccer-tournament
```

2.2.2 Compile the Java Code (Optional if using IDE)

1. Navigate to the `soccer-tournament` directory and compile each directory using the following commands:

```
javac RCSS2D/src/*.java
javac RCSS2D/src/<directory>/*.java
```

2. Note: Use the first line for the `src` directory and the second for any other subdirectory.

2.2.3 Run the Tester Codes

1. You may now run any runnable code, such as the tester codes (`gktester`, `midtester`, `teamtester`), using the following command:

```
java RCSS2D.src.<tester-code>
```

2. If successful, players should appear in position. If not, simply press `Ctrl+C` to shut down the code and re-run it.

2.3 Running the Simulation

2.3.1 Start the Simulation

1. Go to `Referee > Kick Off`. The left team kicks off and the simulation runs for 3000 cycles, then the right team kicks off and the simulation runs for another 3000 cycles.

2. To interrupt the simulation, simply press `Ctrl+C` in the CLI.

2.3.2 Visualize Real-Time Data

1. Real-time data can be visualized using the Monitor. To do so, run the Monitor using the following command:

   - By toggling `View > Status Bar`, you can see all of the field's information allowing you to monitor the game and get all relevant coordinates for debugging purposes which can also be obtained via the command line.
   - Another set of options is `View > Preferences` (also accessible with `Ctrl+V`): where you can see useful information such as the flags, the offside line to better tune the players' positions and roles.

3 Previous Work

The previous person that worked on this project noticed that there were some definite issues with the implementation, the lack of actual players actions and the fact that the project had more of a final goal objective and lacked individual players' responsabilities. To address the situation he decided to create different classes of players, each with distinct goals and actions. They split the `ActivePlayer` class into three positions:

- **Attacker**: Moves forward, waits for passes, intercepts nearby balls, and attempts to score.

- **Midfielder**: Intercepts the ball, carries it forward, and passes to the attacker if blocked.
- **Defender**: Stays near the goal, aligns with the ball, and tackles approaching opponents.

Each player has specific goals and actions to move to their respective positions, defined by colored areas. Notable design choices include:

- The attacker's position is independent of the ball but should avoid being offside.
- The midfielder maintains a certain distance from the ball to avoid clustering.
- The defender's zone allows space for the goalkeeper and adjusts based on the ball's position.

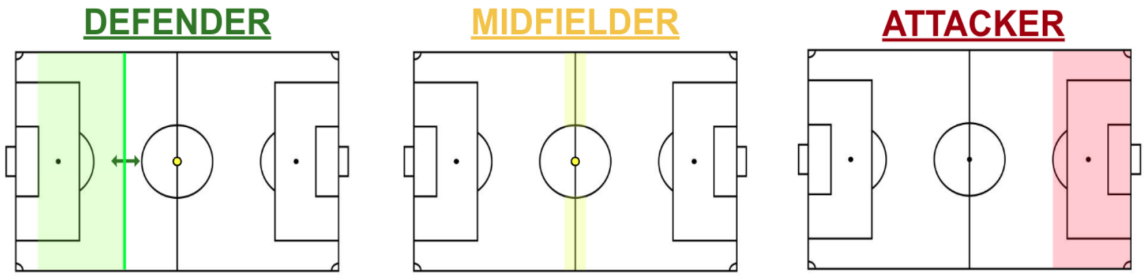

Figure 1: Relative position of each Player on team Left

This approach was successful in creating a more organized team, but there were still some issues with the players' actions and the lack of communication between them. The players were not able to pass the ball effectively, and the team was not able to score goals consistently. The previous person that worked on this project also noticed that the players were not able to defend effectively, and the team was not able to prevent the opposing team from scoring goals. However, the most critical situation was that that players were clustering around the ball which led to the players not following their roles and would result in many ball resets (if the ball does not move for a set amount of time the game will reset and the players will all be dispersed). To address these issues, new player's positions were created and our role would be to fix this issue by implementing an anti-clusteing system that would allow the team to have better goal scoring opportunities and to better defend against the opposing team.

4 Classes and Methods

To understand the code and the various dependencies in detail, you can refer to the previous reports and get a grasp of the different classes and methods that were used in the project. The most important classes and methods are:

- **Player**: The main class that contains the player's attributes and methods.
- **The different classes of players**: `Attacker`, `Midfielder`, `Defender`, `Goalkeeper`, `Winger`.
- **StateKeys**: This enum class contains the different states that a player can be in.
- **Actions**: This package contains all the different actions that a player can take.
- **GlobalMap.java**: This class contains the global map of the field and the different zones for each player and is updated at every cycle. This ensures that the global map is consistently accurate and up-to-date, reflecting the current state of the game.
- **LocalView.java**: This class returns the data from the server in a more readable format.
- **PlayerMath.java**: This class contains the mathematical calculations for the players' position and orientation.
- **Graphs and GOAP**: The project uses a graph-based approach to determine the players' actions and the order in which they should be executed. The GOAP system is used to determine the players' actions based on the current state of the game and the desired goal scoring opportunities.

5 New Implementations

1. **Create new field `type` and `IsClosest`**
    - We created the `type` to represent different player types and designed various movement strategies for players based on these types. `IsClosest` is a flag that indicates whether a player is the closest to the ball. This flag can be updated frequently.
2. **Modify `MoveToBallAction`**
    - We updated the `MoveToBallAction` to include logic that checks the distance between players and ensures they maintain a minimum distance from each other.

3. **Create New Distance Retrieval Actions**

    - We added new actions in the `PlayerMath` module to compute the closest player to the ball every time the rcssserver would receive a ping from the ongoing game. This computation was essential for determining which players needed to adjust their positions to avoid clustering instead of relying on the previous priority-based system resulting in a more dynamic and efficient anti-clustering system with a 95% success rate.

4. **Maintain Minimum Distance Range**

    - By using the new actions in `PlayerMath`, we ensured that players from the same team maintained a minimum distance from each other. This helped in avoiding clustering around the ball such that there wouldn't be more than 2 players on the ball at any given time.

5. **Preserve Formation**

    - The modifications were designed to maintain the formation set in the tester method. This ensured that players followed their roles and positions effectively, leading to better goal-scoring opportunities and improved defense.

However, the new anti-clustering system was not perfect and there were still some issues with the players' actions and the lack of communication between them. The players were not able to pass the ball effectively, and the team was not able to score goals consistently. The main issue was an imbalance between the teams as one team was constantly attacking while the other being overly defensive. In order to assess this issue we had to tinker with the current GOAP priority system by modifying some priorities even though we are not sure what the values exactly mean as the documentation for that part was non-existant. We think that the different priority values are used to determine the importance of each action and the order in which they should be executed with higher values being executed first.

We also implemented a new player type the winger. The winger's role is to stay wide on the field, provides width to the team's attack, and crosses the ball into the box. The winger's position is independent of the ball but should avoid being offside. The winger's zone allows space for the attacker and adjusts based on the ball's position. Wingers are a mix between attackers and midfielders, by receiving the ball they make forward runs and have the same goal chances as attackers but to the contrary of attackers, they have the ability to pass the ball to players in the back.

Winger			
GOALS	**StateKeys**	**Effect**	**Actions**
FocusOnBallGoal	ball_centered_in_FOV	true	GetBallInFOV, CenterBallInFOV
PassBallGoal	kick_performed	true	ScanForTeammates, PassBallToTeammate
KickBallGoal	kick_performed	true	KickBallToGoal
InterceptBallGoal	has_ball	true	MoveToBall
GetInWingPositionGoal	in_wing_position	true	MoveToMidPosition
ResetPositionGoal	in_new_position	true	TeleportAction

Table 1: Midfielder Goals, StateKeys, Effects, and Actions

Once a winger crosses the midpoint, he can pass to the attacker or midfielder if he can't score. This restriction ensures that the winger focuses on creating scoring opportunities by delivering the ball to the most forward player which is in most cases the closest player. Usually, the winger is positioned more forward than the attacker to have better crossing opportunities. This forward positioning allows the winger to exploit the width of the field and deliver precise crosses into the box. However, this function is not implemented perfectly; sometimes the wingers cannot pass the ball reliably.

To maintain tactical discipline and prevent the winger from straying too far from his designated zone, we limit the winger's position on the opposite side of the field. Specifically, the winger's movement is constrained within the following coordinates: approximately $y = \pm 10$ to $y = \pm 34$ and $x = 0$ to $x = 52$. This positional limitation ensures that the winger remains in an optimal area to support the attack while avoiding offside positions and maintaining the team's overall shape.

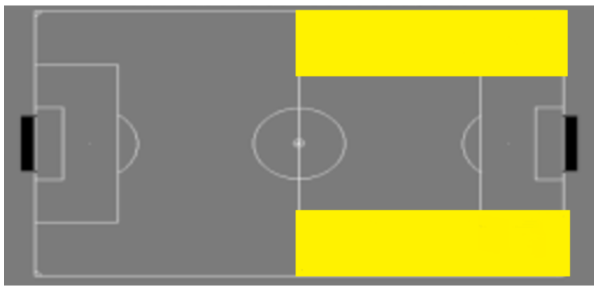

Figure 2: Winger's Positional Limitation

We also wanted to implement the body sensor, but someone tasked with implementing it did not do their part of the work and notified us a bit too late. The body sensor

would have allowed us to monitor the player's physical status, including stamina, speed, head angle, and various action counts, providing valuable insights for optimizing player performance and strategy.

6 Future Work

The project has made significant progress in addressing the clustering issue and improving the team's overall organization and communication. However, there are still some areas that need further improvement: as we said in this report, the GOAP system is not well documented and we are not sure what the different priority values mean. It would have been great if we had a better understanding of the GOAP system and how it can be used to improve the team's performance.

We would also like to implement the body sensor to monitor the player's physical status, optimize player performance/strategy and to make the game a lot more realistic as soccer has many external variables and is a lot more volatile than how the current system is: for example the wingers could have acceleration implemented in order to make them dash faster initially and then slow down once the stamina runs out. Finally, by implementing the body sensor there could also be better movement displacement and would allow a better refinement to the anti-clustering system and result in an improvement of the team's overall organization and communication.

As for future new strategies we would like to implement new tactics that would result in a more complex and dynamic game. For example, we could implement a new formation such as the 4-4-2, the 4-3-3, the 4-3-2-1, etc. which would allow the team to have more attacking options and to better defend against the opposing team. We could also implement new player's roles such as the sweeper or the libero which would allow the team to have better goal scoring opportunities and to better defend against the opposing team. Finally, we could implement new actions such as the through ball which would allow the team to have better goal scoring opportunities and to better defend against the opposing team.

Continuing to enhance movement strategies and logic for different players will be a priority, as will enhancing team collaboration. Utilizing the `doSay(String)` method might be an effective solution for improving communication among players. Moreover, expanding the capabilities of the Winger class by adding more goals and actions will significantly improve their effectiveness on the field and introducing more player types, status keys, goals, and actions will further enrich the gameplay and strategic diversity of our system.

7 Conclusion

In conclusion, the project has made significant progress in addressing the clustering issue and improving the team's overall organization and communication. The new anti-clustering system has been successful in preventing players from clustering around the ball and has resulted in a more organized team. The new winger player type can stay in their zone, run to the ball, and kick it toward the goal. However, there are still some areas that need further improvement, such as the GOAP system and the body sensor implementation. Overall, the project has been successful in improving the team's performance and creating a more dynamic and efficient team. And as the previous report states it would also be better to have multiple members working on the project as it would allow for a more efficient and faster development of the project and would allow for more ideas to be implemented.

8 References

1. Carleton University, "RoboCup Soccer Simulator 2D Installation Guide", https://carleton.ca/robohub/robocup-soccer-simulator-2d-installation-guide/

2. Melissa Katz and Joseph Vybihal, ROBOCUP Soccer Simulator2D Research Project. https://gitlab.cs.mcgill.ca/jvybihal/soccer-tournament/-/blob/main/Reports/COMP_396___Final___Melissa_Katz__updated.pdf

3. Raphael Julien and Joseph Vybihal "ROBOCUP - Soccer Simulator2D Research Project", https://gitlab.cs.mcgill.ca/jvybihal/soccer-tournament/-/blob/main/Reports/COMP396___Final_Report___Raphael_Julien.pdf

4. Raphael Julien, Melissa Katz, Joseph Vybihal, Pedro Khalil Jacob. McGill soccer tournament repository, main branch README https://gitlab.cs.mcgill.ca/jvybihal/soccer-tournament/-/blob/main/README.md

5. RoboCup Soccer Simulator Github page. https://rcsoccersim.github.io/

6. RoboCup Soccer Simulator Users Manual. https://rcsoccersim.readthedocs.io/en/latest/index.html

7. RoboCup Users Manual, Action Models. https://rcsoccersim.readthedocs.io/en/latest/