Extended 2d RoboCup Soccer Framework: Methods for Benchmarking Skills and Preparing Datasets

Pavankumar Patel
Lakehead University
ThunderBay, ON
ppatel59@lakeheadu.ca

Ngudup Tsering
Lakehead University
ThunderBay, ON
ntsering@lakeheadu.ca

Souvik Mukherjee
Lakehead University
ThunderBay, ON
smukher2@lakeheadu.ca

Dr. Thiago E Alves de Oliveira
Lakehead University
ThunderBay, ON
talvesd@lakeheadu.ca

***Abstract*—For multi-agent distributed decision making, robotic soccer is a challenging testbed for autonomous learning agents to work in a fast-paced and dynamic environment. RoboCup is a landmark international competition for the evaluation and comparison of teams based on skills and strategies. Benchmarking skills is crucial for analyzing gameplay and developing better skill-specific behaviours. The soccer server has knowledge of the ball, the players, and the overall gameplay, and generates rcg logs that contain the ground truth of all the actions taken during a match. Extracting the commencement of situations of a particular skill spanning multiple rcg logs could be assembled together to form a dataset of skill-specific situations. With an extended server that could initialize its states with such configuration, a match could be started from that situation to only play partial games until the situation lasts, thereby focusing on the efficiency of such skill-specific situations, and generating partial logs. Benchmarking of teams for a particular skill is performed by exposing them to skill-specific situations multiple times, and collecting the partial logs in the result dataset. This paper focuses on extension of the RoboCup soccer server towards benchmarking skills based on skill-specific action situations and generating a dataset of results from partial gameplay for the purpose of comparison amongst different teams in the tournament. The analysis of results from five teams of RoboCup 2019 demonstrates the efficiency of shoot situations considered for the scope of this paper, while also categorizing the situations based on difficulty.**

***Index Terms*—Benchmarking, Dataset, Logs, Log Analysis, Partial Gameplay, RoboCup 2d, Skill-Specific Dataset, Soccer Server.**

I. INTRODUCTION

RoboCup forms a benchmark of successful international competitions and research challenges [1]. It may be seen as a fully distributed, multi-agent domain with both teammates and adversaries. Each agent has only partial knowledge at any given moment. The agents also have noisy sensors and actuators, which affect the simulation similar to the practical world scenario. These dynamics within the robotic soccer environment call for multi-agent based methods to spontaneously collaborate, cooperate, and coordinate with the environment and among each other. Robotic soccer is an active application area of Multi-Agent Systems (MAS) with collective task performing requisites such as in cooperation, coordination, and collaboration. In a coordination process, each agent in a team must cooperate with other agents while facing competitions during a tasks[2]. Decision making in such multi-agent games is deemed crucial and is expected to adjust strategies to achieve subsequent cooperation, coordination, and collaboration[3], [4].

This section discusses the RoboCup soccer domain along with its various components that have significant effects on the operation of the simulation which forms the basic structure of our proposed solution.

A. RoboCup Soccer

RoboCup soccer has been in existence with the vision of fully autonomous humanoid robots competing against human soccer players and winning the latest FIFA world cup winners by the middle of the 21st century [5]. The challenge posed by this vision has inspired researchers around the globe to engage in research and development in the field, primarily in robotics and Artificial Intelligence (AI) and promote state-of-the-art research in the field. While building a robot that plays soccer will not in itself be impactful to the society at large, the accomplishment will certainly be considered as a major achievement, and this mission makes RoboCup a “landmark project”. RoboCup soccer may also be viewed as a standard problem forming an arena for development and testing of various algorithms, architectures, and related research [5].

The RoboCup community hosts a multitude of related robotics research under the umbrella of the RoboCup federation. They include RoboCup Rescue, RoboCup Home, RoboCup Industrial, and RoboCup Junior. The task of soccer is also divided into several sub-tasks like humanoid, standard platform, middle-sized league, small-sized league, and simulation league. Simulation league is further categorized into 2D and 3D simulation sub-leagues [6]. The scope of this paper addresses the 2D simulation subtask of RoboCup soccer and shall be referred hereafter as 2-Dimensional RoboCup Soccer Simulator(2D RCSS).

B. Soccer Simulation 2D Sub League

2D RCSS is the oldest simulation league in robot soccer. Here, two teams of 11 autonomous software programs (agents) each play soccer in a two-dimensional virtual soccer stadium represented by a central server. The agents here are known as individual clients and the gameplay relies on communication between the server and the client. All games are visualized by displaying the simulated field by a soccer monitor on the screen. The soccer server is written to support competition among multiple virtual soccer players in a multi-agent setting with partial knowledge of the environment, coupled with real-time demands as well as semi-structured conditions. This abstraction of the soccer server is an advantage whereby focus can be shifted from hardware issues, background noise, sensory data and components, and other associated robotics problems to more attention for design and deployment of algorithms [4]. Matches are carried out in a client/server fashion where the server provides a virtual field and simulates all movements of a ball and players. A client connects to the server using an User Datagram Protocol(UDP) socket. Via this socket the client sends commands to control a player and receives information from sensors of the player. The 2D sub-league has a more simplified environment than the 3D-league. There are two dimensions, one following the $x-axis$ and one following the $y-axis$ and an angle θ for field of vision [4]. The following Fig 2 shows a diagrammatic representation of 2D RCSS.
The main components of 2D RCSS are discussed below.

*1) **LIBrary for the RoboCup Soccer simulation Client**:* librcsc is a common library for developing 2D soccer simulation software. It contains several library files such as geometry, network interface, communication and synchronization with the simulator, world model, basic actions, log parser, debug message management, formation model, and the like. It encapsulates almost all things related to the communication between the simulator and agent programs[7].

*2) **Server**:* The server is a system that enables various teams to compete in a game. Each client connects to the server via a specific port corresponding to an UDP/IP socket as a separate process, where after all messages are transferred via this port. The players send requests to the server which are received, handled, and updated in the environment by the server accordingly. In addition, the server provides all players with sensory information. The server may be regarded as a real-time system working with discrete cycles[8].

The server provides various models to gain insights into the gameplay. They could be categorized into the following models: sensor models – consisting of aural sensor model, vision sensor model, and body sensor model; movement models – consisting of movement noise model, and collision model; action models – consisting of catch model, dash model, stamina model, kick model, move model, say model, turn model, and turn-neck model [8], [**?**].

*3) **Monitor**:* The Soccer Monitor is a visualization tool that allows to see the proceedings within the server during the game. The information shown on the monitor include score, team names, position of all the players and the ball. They also provide simple interfaces to the server [8], [**?**]. The following Fig 1 shows the monitor environment with the initial formation situation before the start of the game.

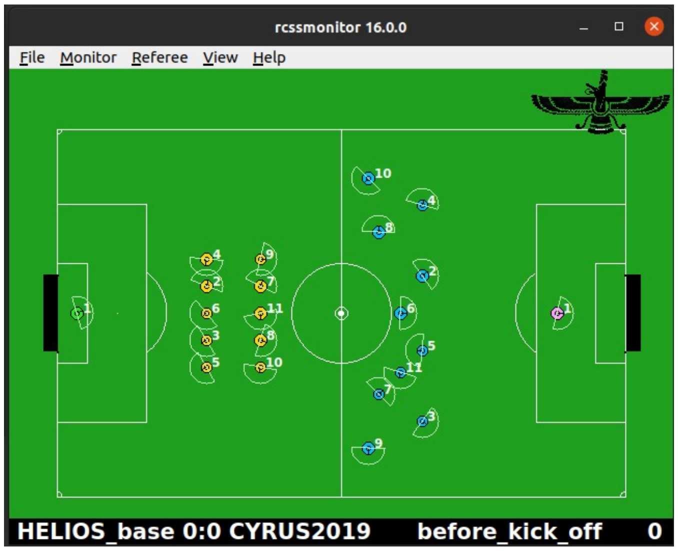

rcssmonitor 16.0.0
File Monitor Referee View Help
HELIOS_base 0:0 CYRUS2019 before_kick_off 0

Fig. 1. Monitor: before_kick_off.

*4) **Client**:* Bundled with the soccer simulator is a client program which implements a primitive client[9].

In the sensor model of each client, there are three different sensors – aural, visual, and body. The aural sensor detects messages sent by the referee, the coaches, and other players. The visual sensor detects visual information about the field. The body sensor detects physical attributes of the player like stamina, speed, and angles [8]. The client is the basic player which is used to develop teams for the soccer matches.

C. Gameplay

As discussed before, RCSS 2D enables two teams of 11 autonomous programs (here player agents) each to play a game of soccer with highly realistic rules and real-time gameplay [7]. The special challenge in robot soccer simulation is the complexity in rules and associated simulation constraints that are geared to make the simulation resemble real soccer [3]. This paves way for a multitude of complex tasks while developing and analyzing agents. The simulator implements the rules of the game via the server and also handles interface to the programs controlling each player. A virtual referee is implemented to impose and enforce such rules, and a human referee to supervise the rules and detect fair and just gameplay. Each match is divided into 6000 cycles. Each cycle lasts for 100 milliseconds making every match 10 minutes long. During each cycle each agent is allowed to perform one action [4], [10]. By default, the players use a 90 degree field of view, and receive visual information every 150 milliseconds. Each player receives additional state information including energy levels, referee decisions, and the overall state of the game after each cycle[10]. Rules are segregated into those judged by an

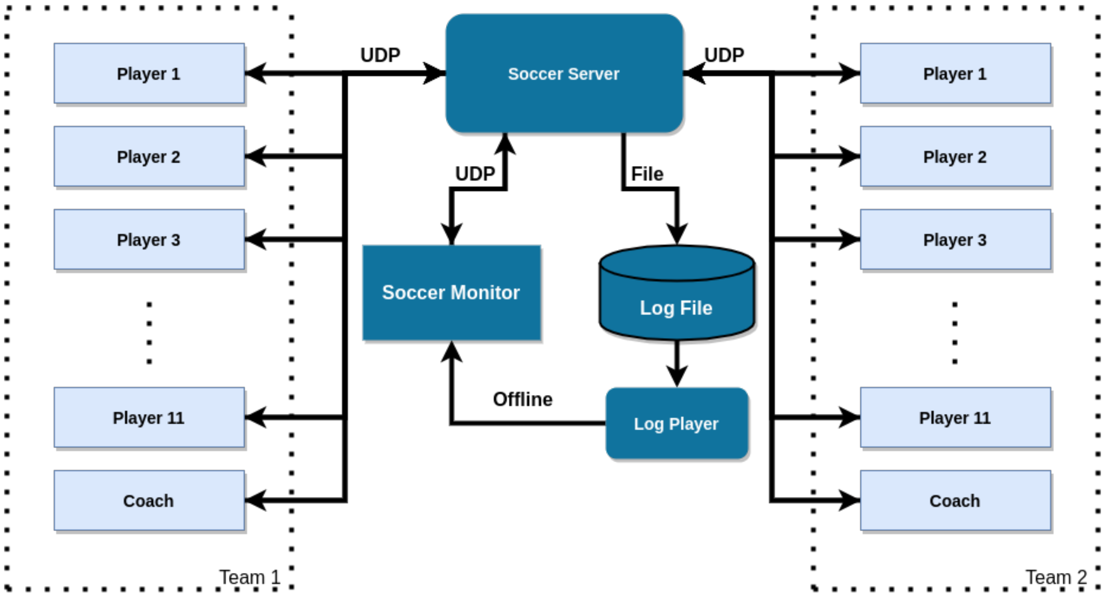


Fig. 2. 2D RoboCup Server Simulator.

automated referee and those judged by a human referee. The automated referee enforces rules pertaining to kick-off, goal, ball or player out of field, players' clearance, controlling play-modes, off-sides, back passes, free kick faults, and half-time and time-up calls. Rules monitored by the human referee are more complex and require external intervention.

D. Connection

As discussed before, matches are carried out in a client/server fashion where the server provides a virtual field and simulates all movements of a ball and players, and the client controls each player. Each client connects to the server via a specific port corresponding to an UDP/IP socket as a separate process, where after all messages are transferred via this port. The players send requests to the server which are received, handled, and updated in the environment by the server accordingly [8], [**?**]. The server protocols may be categorized into: i) player command control which is concerned with connecting, reconnecting, and disconnecting the payer to the server and the server to the client, initializing connections between server and player, and player control; and ii) player sensor protocol between player and server [8]. The client protocols may be categorized into: i) initiation, reconnection, and disconnection commands; ii) control commands which are concerned with actions taken by each player and include – body commands, communication commands, and miscellaneous commands; and iii) sensor information comprising of visual, audio, and body sensor messages that are sent to all the players regularly [9].

E. Binaries

A team binary is a binary file generated upon configuring and building the source code of the respective teams and are typically used towards loading the client teams to the server for subsequent gameplay. Release of binaries was made compulsory since RoboCup 2002, and could be accessed from the archive for analysis of teams previously standing in the tournament[11].

F. Logs

After each game played, the server saves a log of the match that contains all information about the game [11]. RoboCup soccer server produces two log files after each game, namely the rcl and rcg log files. Of these, rcl log files offer a simplified and abstract version of the gameplay and hence is out of the scope of this research. In contrast rcg log files and more detailed and comprehensive, comprising information such as ball position, ball velocity, player position, player velocity, type, body angle, neck angle, stamina, and count details[12]. These logs are stored in the RoboCup archive, and could be downloaded to replay and analyze any previously played matches of 2D RCSS. Log files comprise information about the game, especially about the current positions of all the players and of the ball including their velocity and orientation for each cycle [12]. The purpose of these log files is to assess the team's performance and evaluate its scientific efforts in the context of gameplay and subsequent competitiveness, while also providing opportunities to analytically analyze the proceedings of previous matches [12]. A detailed description of the logs and the information it contains can be found in Appendix 1.

G. Play Modes

Play modes refer to the modes of different situations within the gameplay right from the initialization of the server to the end of the game. The situation as well as situation transitions are determined by the play modes, and hence is an imperative component for analysis of skills, strategies, and gameplay

TABLE I
PLAY MODES[8]

Play Mode	t_c	subsequent play mode	comment
”before_kick_off”	0	‘kick off Side ’	at the beginning of a half during normal play
‘play on’			
‘time over’			
‘kick off Side ’			announce start of play(after pressing the Kick Off button)
‘kick in Side ’			
‘free kick Side ’			
‘corner kick Side ’			
‘goal kick Side ’		‘play on’	play mode changes once the ball leaves the penalty area
‘goal Side ’			currently unused
‘drop ball’	0		
‘offside Side ’	30		for the opposite side

where Side is either the character ‘l’ or ‘r’, OSide means opponent’s side.
t_c is the time (in number of cycles) until the subsequent play mode will be announced

[8]. Play modes are typically controlled by the referee. Table I above shows a list of play modes and their usage.

Having knowledge about the log files and insights into the skills and gameplay could leverage the construction of a customized dataset of skill-specific situations to benchmark skills. Having traced the path of research in this section, we propose our work to be 6-fold to make contributions to the field of 2D RCSS and subsequent building of a framework in progressive order of implementation and execution.

- Analyze the server to draw insights on the necessary operations needed to extend and update the server.
- Log analysis from the archive of game logs to determine the occurrence of skill-specific action situations, and subsequently extract logs based on the desired skill(s).
- Data acquisition and conversion of extracted data into JSON to assemble a dataset of skill-specific action.
- Updating the server to accommodate server state initialization and termination after the desired interval, starting from the situation inception and the occurrence of the situation, and subsequently generate partial logs.
- Create a result dataset from the partial rcg logs generated from the partial gameplay by exposing teams to skill-specific situations multiple times.
- Benchmark teams in skill-specific situations, and evaluate and compare the efficiency.

The remaining of this paper is categorized as follows: section II discusses the literature and reviews the related work in this field. Section III formally introduces the problem statement and formulates the workflow. Section IV gives the system overview. Section V discusses the preparation of skill-specific datasets. Section VI moves on to the experiments conducted and behavioural analysis. Section VII presents the results obtained and discusses them. Finally, Section VIII concludes the paper and describes the scope for future work.

II. LITERATURE REVIEW

Mackworth[13] introduced the idea of using soccer playing robots in research, and successively Kitano et al.[14] proposed the first Robot World Cup Soccer Games and Conferences in 1997. It initially boosted off as the “life of AI after Deep Blue.” 2D RCSS was the most popular sub-task of RoboCup soccer, founded in 1997 and held in Nagoya, Japan. Since then, various leagues and tasks under the umbrella of RoboCup have been hosted worldwide every year that employ research from around the globe [4].

The various leagues of the RoboCup Federation have been seen to include specific evaluation challenges which not only complement the competitions but also enhance the scientific and technological sphere of robotics and AI in general[15]. Typically, a challenge would introduce some new features into the standard competition environment and evaluate how the team would perform under a given new ambience. Previous works that described the innovations in the 2D RCSS including heterogeneous players, stamina and recovery model, tackles and the like, further pointed out implications of the analysis and adaptation of the game strategies and behaviour. The paper described the evaluation challenges introduced by the 2D RCSS league and two subsequent challenges: one intended to systematically trace the advancements in the league, and the other aimed to increase the realism of the environmental conditions during the simulated games [15].

The necessity and existence of a dataset of required information for behavioural analysis and subsequent research and development is deemed crucial. A collection of log files from the gameplays of top teams formed a baseline for further development and deployment of skills, strategies and gameplay. This is because game log files contain ground truth information obtained from recording games, and could be produced from the simulator and recorded in a binary format. The article[10] used 10 different teams to conduct a match against each other, resulting in 45 unique pairings. For each pairing, 25 matches of 10 minutes each were run, leading to 1125 matches and more than 180 hours of gameplay. The resultant data is stored in the form of Comma Separated Values (CSV) files amounting to 17 GB of compressed data (229 GB unzipped) as well as standard soccer simulation

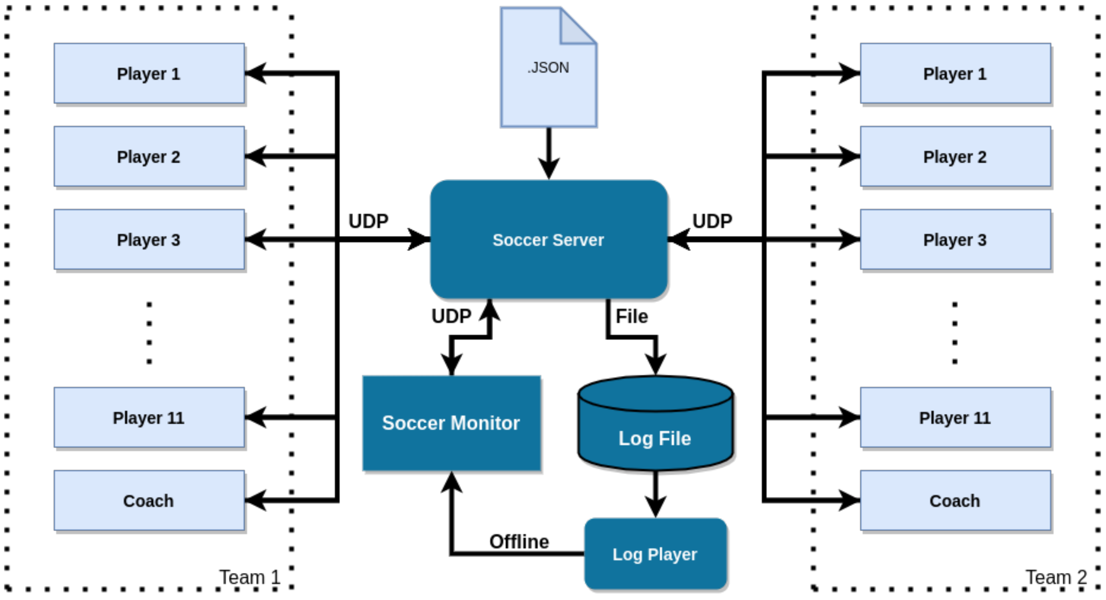


Fig. 3. Updated 2D RoboCup Server Simulator.

formats (RCL and RCG). The dataset possessed both ground truth data as well as noisy, local, and incomplete precepts of each robot, and this is what makes the data unique.

Logs could be reviewed by developers in order to analyze the game, debug the code and explore probable problems to improve the gameplay. The article proposes using a data mining approach for developers to be able to process this data very quickly [12]. Log files gather information about ball position, velocity and decay, and also player type, position, velocity, decay, body direction, head direction, stamina (energy), recovery and a variable number of count information. In other words, using data mining can help to analyze and process log files and produce useful and easily understandable information and patterns from a large amount of data. Ball and players' positions are selected in rcg files, followed by tracing the ball owner at different times and subsequently observing actions. Actions included pass (give the ball to a teammate), dribble (move with the ball), goal (shoot the ball into opponent's goal), unsuccessful pass, unsuccessful dribble and unsuccessful shoot; where "unsuccessful behavior" meant the behavior that causes the opponents to own the ball. Then a dataset was formed by "effective players' positions" as state and "performed action" as class attributes. Agent2d was used as the base team for building the dataset which was further employed to classify and form a decision tree. Trees were built to analyze shoot, dribble, pass, and predict opponent behaviour [12].

The Skills, Tactics, and Plays (STP) [16] software architecture has been employed many-a-times for tackling the complex problem of robotic soccer hierarchically. STP is a three-tier hierarchical architecture. Skills signify coded policies that represent low-level tasks which are typically used repeatedly such as dribbling the ball, navigating to a point, etc. Tactics combine skills into single robotic behaviour like attacker, defender, and the like. Deep Q-learning based Reinforcement Learning (RL) algorithms have been explored to redress the problem of hierarchical STP architecture in learning low level skills as well as intermediate level tactics [17]. Single agent is utilized to learn small skill sets, while it is theoretically explained that a multi-agent network maybe deployed to learn the more complicated tactics in the gameplay.

In order to develop an insight of behavior and strategies of the teams in the RoboCup soccer domain, it was necessary and crucial to know the current work of the specific teams done in the past, which would further contribute towards qualitative and quantitative analysis of the gameplay These research reports, known as Team Description Papers (TDPs), would contribute towards the analysis of skills and associated actions.

After a broad review of the related work, it is evident that a difference in performance of teams owing to the difference in strategies; difference in strategies owing to difference in skills; difference in skills owing to difference in methodologies of analyzing, segregating, developing and implementing the set of skills. Improved skills in a MAS environment ensures enhanced tactics, thus resulting in superior gameplay. Consequently, analyzing occurrence of skill-specific situations and associated actions from log files, extending the server to accommodate initialization and termination of server state, and combining the two to generate logs from partial gameplay is crucial. The assimilation of such partial logs in a structured dataset of skill-specific actions for benchmarking and further evaluation is of significant research interest, pertaining to the scope of this paper.

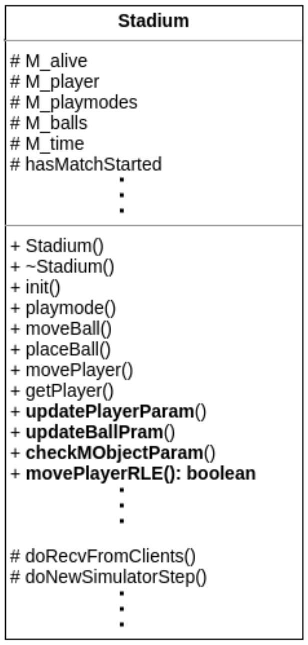


Fig. 4. Updated Stadium Class.

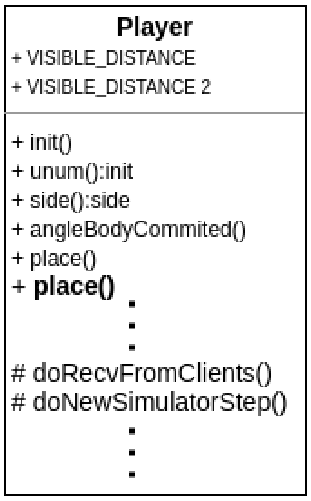


Fig. 5. Updated Player Class.

III. Problem Statement and Formulation

The aforementioned concepts, components, attributes, functionalities, and related work could be integrated into a problem where simulation and gameplays are analyzed, situations are identified, skill-specific data is extracted, and the 2D RCSS server framework is extended to accommodate updating the server for partial gameplay. The resulting framework would utilize skill-specific situational dataset to form partial gameplay for benchmarking skills and evaluate behavioural aspects and gameplay of different teams. The framework could also serve as a testbed for the development of enhanced tactics in gameplay.

The problem can be formulated in the following three categories:

1) Skill-specific Dataset Preparation:
    a) Log analysis from the archive of previous matches, wherein numerous log files were studied to recognise patterns of skill, and subsequently extract logs based on the desired skill(s).
    b) Data acquisition and conversion of extracted data into JSON to assemble a dataset of skill-specific situations.
2) Extending 2d RCSS Server:
    a) Analysis of the existing 2D RCSS server to draw insights on the necessary attributes needed to extend and change.
    b) Input analysis to determine the nature and format of the information required by the server.
    c) Updating the server with configuration to accommodate initialization of server state and allow gameplay from the desired situation.
3) Benchmarking Teams in a Specific Skill:
    a) Repeatedly execute partial gameplay for a stipulated number of cycles and subsequently generate partial logs.
    b) Create the result dataset from the rcg logs generated from the gameplays.
    c) Benchmark teams in skill-specific situations, and evaluate and compare the efficiency.

IV. System Overview

The system overview may be broadly classified into the following segments: server analysis, input analysis, design update, and partial gameplay. Each of these segments is explained in detail in the following subsections.

A. Server Analysis

The code for the entire simulation was written in C++, and divided into over 100 files. There were more than ten files with over 1000 lines of code, but there existed a few classes that were considered more important than the others and would therefore be the main focus of the study [4]. The significant classes could be studied from Unified Modeling Language (UML) diagrams showing the association between classes, different methods of each class and their contribution towards shaping the server. The Doxygen documentation tool[18] was used to generate the class diagrams.
A thorough study of the class diagrams and their respective codes revealed three classes to have significance on the operations of the ball, the players, and the stadium environment as a whole. They were the Stadium class, the MPObject class, and the Player class. The class diagrams of these classes are shown in Appendix 3. In addition, the Stdtimer class was studied as part of the analysis.

- Stadium class was one of the most imperative classes containing significant information regarding the Ball object, the Player objects, and the environment parameters. It housed all the object references of the ball and all players and necessary information of gameplay.

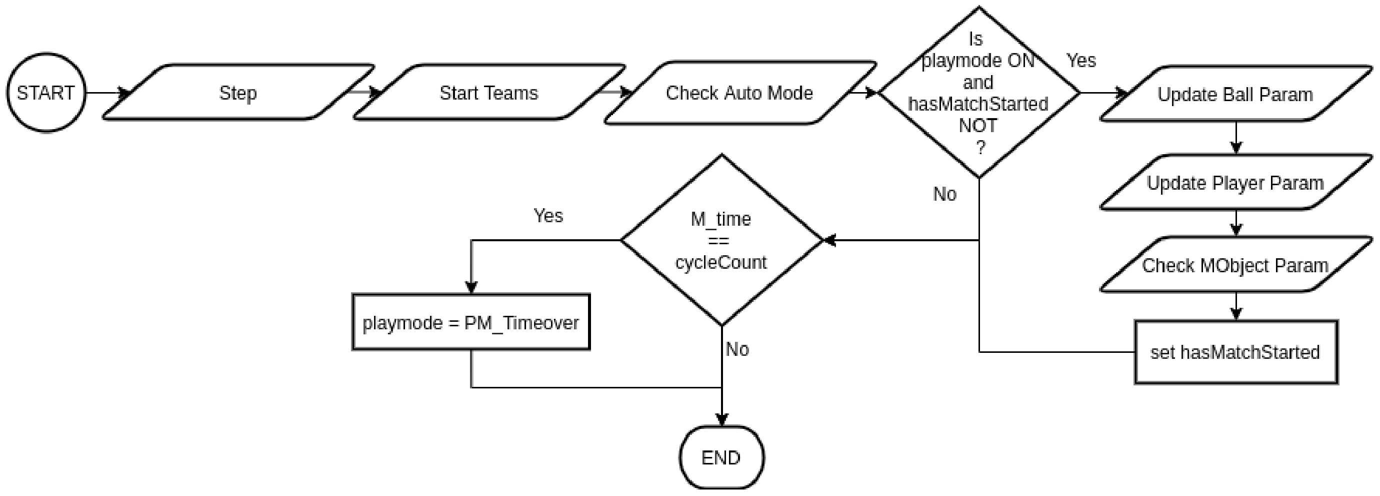


Fig. 6. Updated Flow Diagram: doNewSimilaterStep().

- The MPObject class served as the parent method for the ball and all player objects and included attributes such as Stadium reference, position and velocity, and methods to access them. It also offers other methods to represent the behavior of a movable object: moveTo() and moveToCollision().
- The object of a Player class represents a player in a match. It comprises attributes like player roles and player information, and methods like kick, dash, tackle, and turn.

The server acts as a central processing unit of the whole 2D RCSS and encapsulates all the components of it. In particular, the Stadium class was studied and found to be the most significant class for the construction of the framework whereby the server state could be initialized with the log configuration and proceed with gameplay.

B. Input Analysis

Input analysis studied the information to extract from the log files to accommodate the server state initialization. The data of a situation is extracted from the log and stored that it has the following components:

- Cycle number corresponded to the cycle index in which the skill-specific action situation occurred, and is an integer value.
- Cycle interval corresponded to the number of cycles that were desired to be run in the server, and in essence, comprised the tentative number of cycles: from at the cycle where the skill-specific action situation occurred till the cycle where the skill was exhibited.
- Ball parameters corresponded to the attributes of the ball object to be updated as part of configuring the server with the cycle where the skill-specific action situation occurred, and contained the ball positions (x and y) and the ball velocities (along x and y).
- Player parameters corresponded to the attributes of the player object to be updated as part of configuring the server with the cycle where the skill-specific action situation occurred, and contained the player positions (x and y), velocities (along x and y), head and body angles, stamina information, player being faced towards, and various count variables.

The aggregation of such input parameters were included in a dataset for subsequent usage. The JSON file format with the aforementioned parameters is shown in Appendix 2.

C. Design Update

In order for a server to be initialized with a log configuration with states of ball and players so that a specific situational condition is reproduced, changes were implemented in the existing server. The framework was extended to accept an input log configuration as a JSON file defined in the previous section that represents a situation from gameplay. The updated system diagram is shown in Fig 3.

The situation configuration would be utilized when the teams are connected to the server and the match starts. In doing so, the states of the ball and players were initialized to reposition them to the desired location in the field as per the situation configuration. The updated class diagram of Stadium included attributes and methods as shown in Fig 4. Similarly, the Player class was also updated as shown in Fig 5.

Updated flow diagram of doNewSimilaterStep presented in Fig 6 exhibit where the changes have been made to accommodate the requirements.

D. Partial Gameplay

In the partial gameplay, the updated server could be initialized based on the input log configuration representing the inception of a skill-specific situation and proceed to continue with gameplay. The situation configuration fed to the server had a stipulated number of cycles to determine the total interval of gameplay and when to terminate. The gameplay thus initiated would result in a partial match being played, and subsequent rcg logs being generated at the default location as partial logs. These logs would be further deployed to

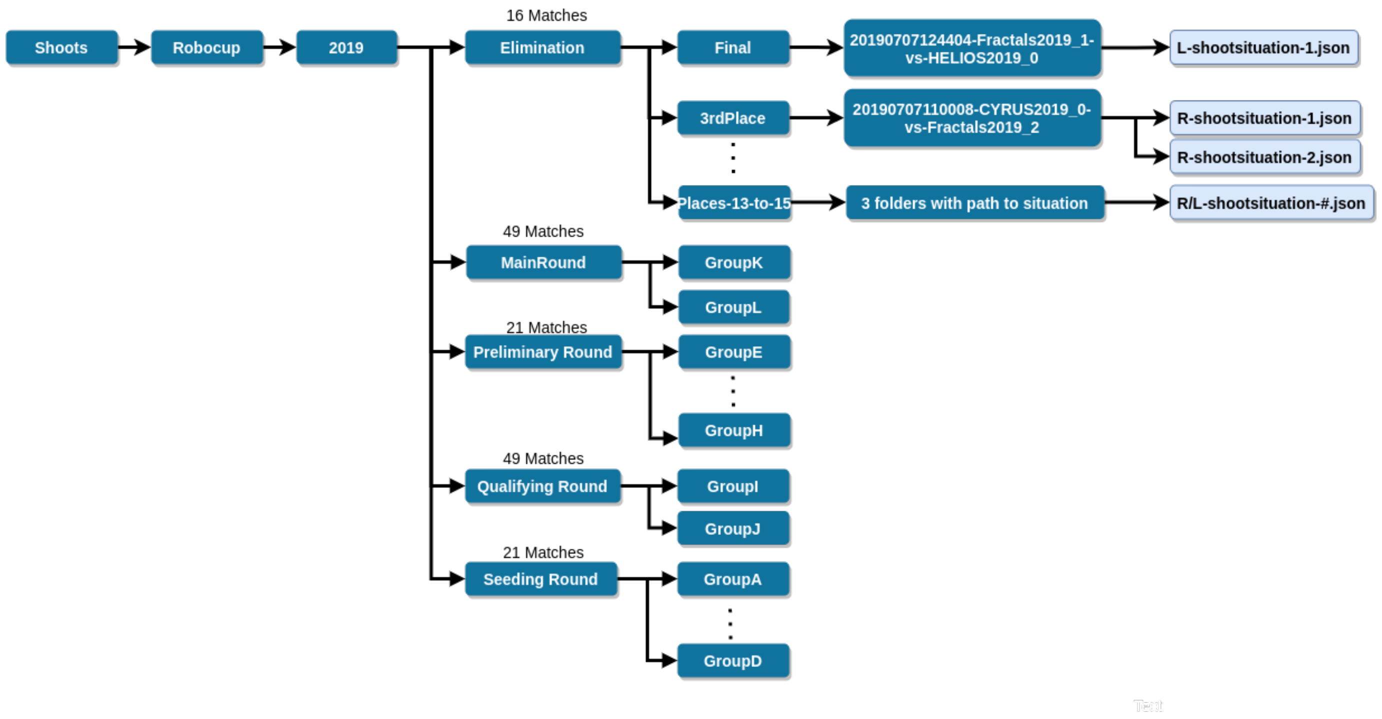


Fig. 7. Folder Structure: Shoot Situations Dataset.

draw insights about the exhibition of the skills in the desired situation.

1) Server Initialization and Termination: The server state would be initialized with the configuration parameters from the JSON file when the play mode would be set to "on", and the simulator would be allowed to play the match thereafter. The match would run till the stipulated number of cycles as determined by the cycle interval inside the JSON file. Finally, the server would terminate after running for the defined interval by updating the play mode to "time over". This would safely terminate the server and generate log files based on the gameplay.

2) Partial Log Generation: The execution of the partial match for the given number of cycles would terminate the partial gameplay and generate log files based on only that interval, and not the whole game. The logs would be saved in rcg format in the default path. These partial logs would thus contain information regarding the skill-specific situation being executed for further evaluation.

V. Preparing Dataset

The dataset preparation process was divided into three main segments: log analysis - included study of log files for recognizing and analyzing skill-specific action situations; data acquisition - included detection of a cycle of occurrence of desired skill-specific action to be converted into JSON, and generation of skill-specific dataset - included aggregation of all JSON files into a dataset.

A. Log Analysis

An rcg log contains information about the game, especially about the current positions of all the players and of the ball including their velocity and orientation for each cycle. The logs provide insights about the behavior and strategy of teams. Patterns could be inferred from these logs to recognize specific skills. The desired attributes from the input analysis would not be found in all the cycles, and hence the valid log cycles with the appropriate parameters should be chosen which would exhibit occurrence of skill-specific situation. These skill-specific patterns would be identified in the cycles. Furthermore, the interval between the occurrence of the situation and the execution of the related action is also evaluated.

B. Data Acquisition

Data acquisition signified the extraction of data from the log files to create JSON files for subsequently configuring the server to initiate gameplay from the desired cycle. For this purpose, the choice of the appropriate cycle would be crucial based on the analysis of desired actions. Extraction of the relevant ball and player data from the cycle that exhibits the situation action inception would be used to extract cycle attributes into respective JSON files. Subsequently all such JSON files corresponding to skill-specific situation actions would be aggregated into a skill-specific dataset of action situations, each entry of which would correspond to a cycle in the actual logs exhibiting the action.

C. Skill-Specific Dataset

The JSON files created in the above process would be aggregated into a dataset of skill-specific situations. The dataset would be segregated into folders. The folder structure from the original RoboCup archive was preserved to ensure easy reference to particular occurrences of skill-specific action situations in the original archive. The nomenclature convention of all entries in the dataset would correspond to the sequence of occurrence of a particular situation within the match in the original log. Further information regarding the skill-specific dataset in case of goal situation is discussed in a later section.

D. Dataset of Situation

The partial logs generated would be collected and stored in a result dataset for the purpose of benchmarking and other analysis. All resulting data would be stored in rcg log file format. This dataset thus obtained would contain the logs generated from partial gameplay of an array of such skill-specific situations, and could serve as a baseline for further analysis of the performance of the skills in the desired situations.

VI. Experiment

Shoot skill situations were chosen to evaluate the associated shooting efficiency of teams. In order to carry it out, all the log files of the 2019 games were analyzed, and a skill-specific dataset of shoot situations were built. Thereafter, on the updated server, teams were made to play partial games to generate partial logs based on shoot situations.

A. Choice of Teams

1) RCSS 2D 2019: The 2019 RoboCup was held in Sydney with 15 teams participating, and Fractals winning the title. The other teams were HELIOS, CYRUS, YuShan, MT, FCP_GPR, FRA-UNIted, HillStone, ITAndroids, MT, Razi, Receptivity, Ri-one, RoboCIn, Titas da Robotica, and HfutEngine [6]. The tournament was divided into 5 rounds with varying numbers of matches among qualified teams at each level. The seeding round consisted of 4 groups. Group A held 6 matches, group B held 6 matches, group C held 3 matches, and group D held 6 matches. The preliminary round comprised 4 groups again, with 6 matches each in group E, F, and G, and 3 matches in group H. The qualifying round followed with group I playing 21 matches, and group J playing 21 matches. The main round had 28 matches in group K, and 21 matches in group L. The elimination round which determined the positioning of the 15 teams, and included 16 knockout matches. Altogether, there were 156 matches and witnessed 472 goals of which 289 goals were scored by the right team, and 183 goals were scored by the left team during all matches.

2) Team Selection: For the purpose of benchmarking, 5 teams from RoboCup 2019 were shortlisted based on their statistical standing. These teams secured ranks 2 to 6 in the tournament, and contributed to 45 percent of the total goals scored in the tournament. The teams were CYRUS, HELIOS, YuShan, MT, and Receptivity. Though Fractals won the 2019 RoboCup, it was excluded from the evaluation for it required changes in the *librcsc* files commonly shared by all the teams. The left team always starts a match. When the teams receive a KICK_OFF command, the left team may or may not take a long time to start a match depending on how the teams are implemented. This introduces inconsistency in the amount of time the partial match can be played, as the interval is fixed based on the input log configuration from the JSON file. Therefore, a fixed left team as an opponent is chosen for consistent behaviour. On the right, teams for benchmarking are connected. Receptivity was always chosen as the left team while the remaining 4 teams served as right teams for the gameplay. Thus, keeping a constant left team while changing the opponents would provide a stable baseline for evaluation and comparison for the 289 right shoot situations that are used for the benchmark analysis.

B. Data Extraction from Logs

Acquisition and subsequent extraction of data from cycles containing shoot situations to prepare JSON files, and consequently incorporate it into a dataset of log configurations to initialize the server state. It involves the following three steps.

1) Validation of Logs : The log files contain 6000 or more cycles out of which not all cycles could be considered as they might not carry any information about ball and player parameters as desired for updating the server and accommodating server states. Hence only valid log cycles were taken into consideration.

2) Assessment of Cycle Intervals: As mentioned above, cycle interval refers to the number of cycles between situation action inception and completion of the situation. Hence it would refer to the interval between the execution of a shoot action situation and the occurrence of a goal. Inspecting the matches showed a maximum of 12 cycles from shoot to goal. Hence a safe interval of 15 cycles was chosen to be considered to fetch 472 shoot situations. After considering buffer time required by each member of the team to be connected to the server for initialization and save termination, a default interval of 45 cycles was decided for the shoot situations analysis.

3) Shoot Pattern Identification: A successful shoot would lead to a goal, which would, in turn, mean the transition of play mode to $goal\_r$ and $goal\_l$. Hence, the presence of this play mode was searched within the logs to locate the occurrence of a goal. Within the aforementioned safe interval, a player status of 0×3 was probed occurring immediately before a successful goal in order to identify the successful kick responsible for scoring the goal. 2 cycles prior to the position of the kick in the log was taken as the final consideration for the occurrence of shoot situation. At a time when player status for kick was overwritten, then the cycle found after tracing back default safe interval was considered.

C. Dataset of Shoot Situations

The suitable parameters as specified in the JSON format in the previous sections were picked from the cycle deemed responsible for the successful shoot and was used to create

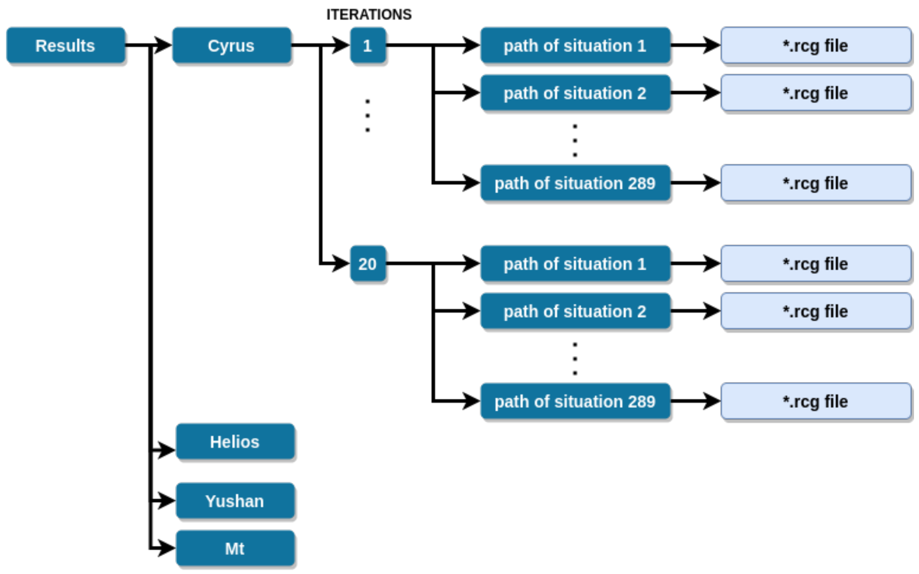


Fig. 8. Folder Structure: Result Dataset.

respective JSON files, with one JSON file being built per occurrence of the shoot action situation. String manipulation was used to extract attributes, therefore any update on rcg format might break this process at a later stage. The dataset of goal situations thus contained 472 such entries corresponding to 472 goals scored in the tournament. The folder structure of the dataset is shown in Fig 7. Nomenclature of the file names is: the R or L represents the right or left team scoring that goal in that situation, the skill-specific situation name is then appended, in this case, shootsituation, followed by the ordinality of the goal. For instance, R-shootsituation-1.json is a situation where the R team scored the first goal of the match.

Following the loading of JSON configuration to initialize server state, and terminate server after partial gameplay, the partial game logs were also stored in a dataset of goals for further benchmarking and testing of the updated simulation for shoot successfulness evaluation of the teams in consideration. This resulting dataset also had 289 entries, and in both cases, the dataset was segregated into folders based on the original folder structure of RoboCup 2019 in the archive.

D. Benchmarking

For the purpose of benchmarking teams, CYRUS, HELIOS, YuShan, and MT were selected, keeping Receptivity as the opponent. The purpose of benchmarking was to evaluate the shooting efficiency of teams in successful goal situations and compare the results. The respective team binaries were picked from the archive and connected to the extended 2D RCSS framework. The server states were initialized with the shoot situation configuration JSON file corresponding to the 289 right shoot situations to analyze gameplay. The partial gameplay lasted for a default interval of 45 cycles. For a statistically sound outcome, each of these situations was executed for 20 iterations. The results of all such situations were stored to create a dataset in the form of rcg log files. The folder structure containing the results is shown in Fig 8.

VII. Results and Discussion

For 20 iterations of 289 shoot situations, four teams - CYRUS, HELIOS, YuShan, and MT played against one opponent - Receptivity in the extended server for benchmarking their shooting skill.

A. Shooting Efficiency

As the situations exposed to the teams were actual shoot situations extracted from the matches of 2019, shooting efficiency is, therefore, taken as the ratio of total goal scored by the team to the total situations exposed.

$$Shooting\ Efficiency = \frac{Total\ Goals\ Scored}{Total\ Situation\ Exposed} \quad (1)$$

The graph in Fig 9 represent goal distributions by respective teams in all shoot situations.

TABLE II
Shooting Efficiency

Teams	Goals Scored	Situations Exposed	Efficiency
HELIOS	2068	5776	0.3580
CYRUS	2326	5780	0.4024
YuShan	968	5757	0.3418
MT	1795	5746	0.3123

From the Table II, the shooting efficiency ranged between 0.3123 and 0.4024, which is particularly low given that the situations were indeed actual shoot situations. Though HELIOS secured the second rank in the 2019 2D RoboCup tournament and CYRUS came third, the latter had better shooting efficiency of 0.4024.

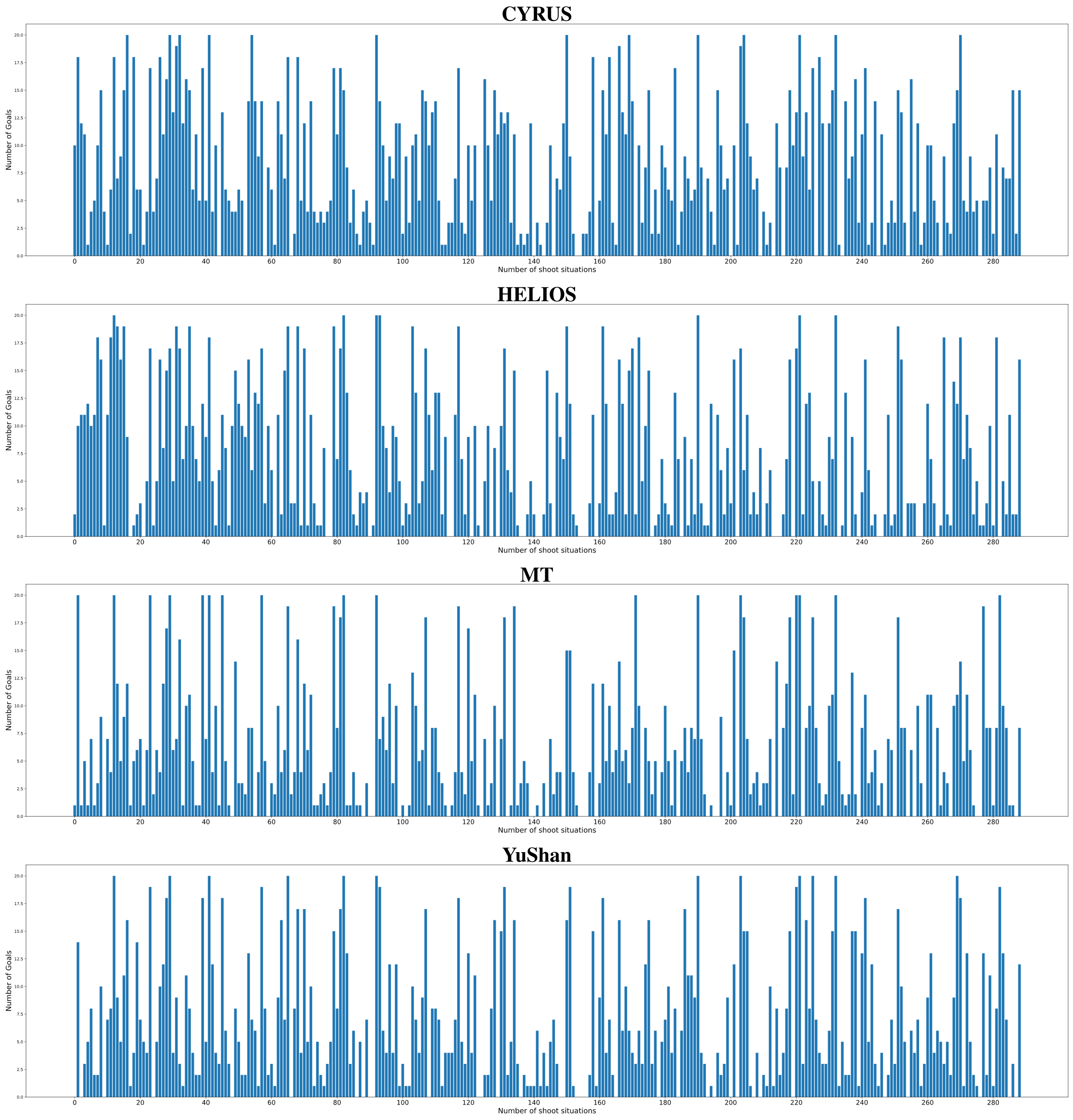


Fig. 9. Goal Distribution.

B. *Shoot Efficiency in Previously Exposed Situations*

From all the situations, those situations were considered where the teams have scored in the original match. Shooting efficiency is thus the ratio of total goal scored in previously exposed situations to the total previously exposed situations. The Fig 10 of graph represents the goal distributions by respective teams highlighting the situations that teams had been previously exposed.

From the Table III, efficiency ranged between 0.3041 and 0.3814. YuShan performed better than the rest of the teams in this situation with a shooting efficiency of 0.3814. Even when the teams were exposed to those situations where they have scored in the original matches, their shooting efficiencies were dismal indicating non-deterministic nature of decision in the

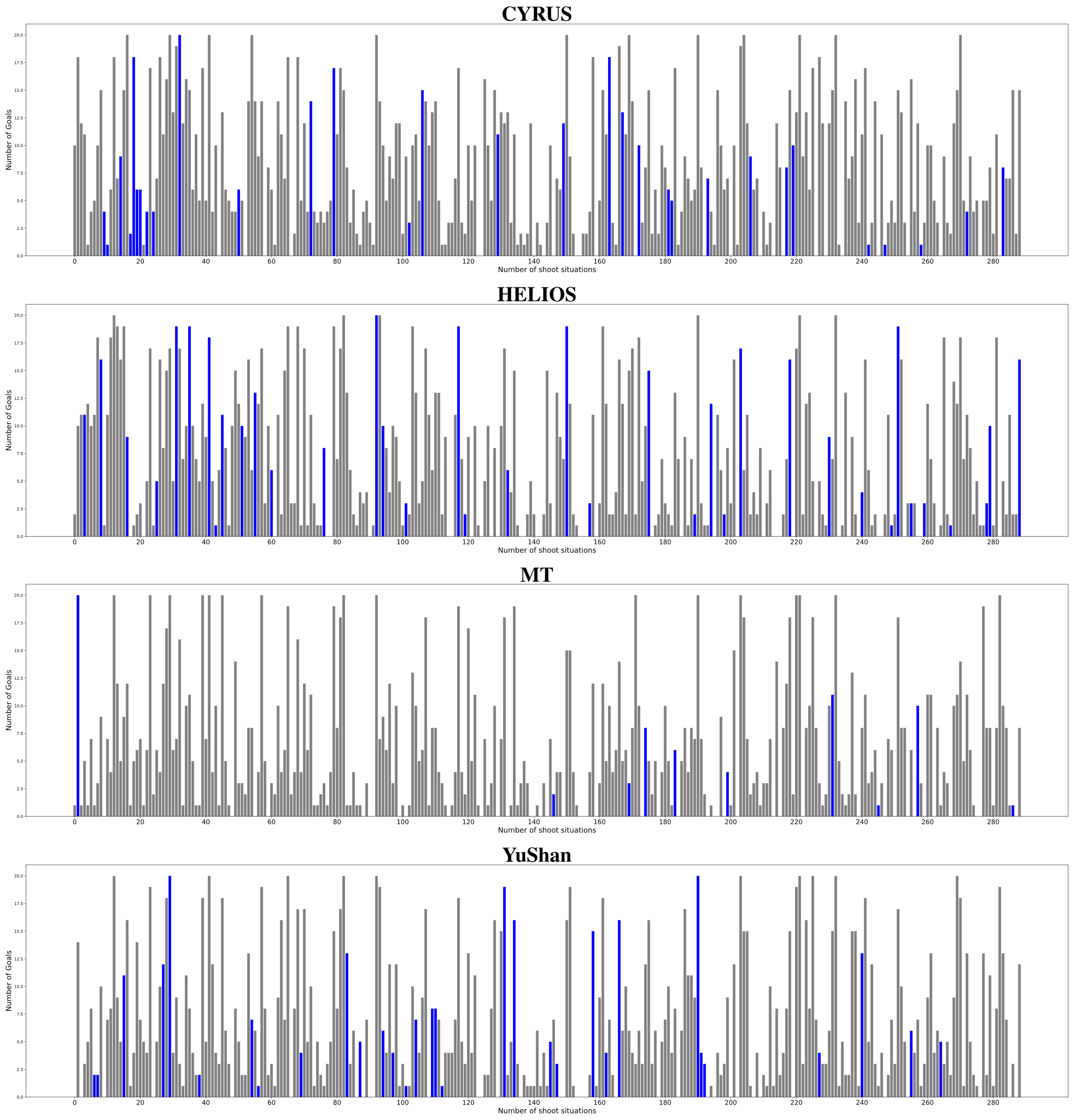


Fig. 10. Goal Distribution: Previously Exposed Situations Highlighted.

gameplay.

$$Shooting\ Efficiency = \frac{Total\ Goal\ Scored\ in\ Previously\ Exposed\ Situations}{Total\ Previously\ Exposed\ Situations} \tag{2}$$

C. Situation Segregation

From the result, there were 23 situations where 2 or fewer goals were scored by any team on any iteration and 12 situations where goals were almost always scored by all the team on all iterations with 17 or more goals. Extending this idea, there were certain situations where scoring a goal was fairly easy for all teams, and certain situations were potentially hard.

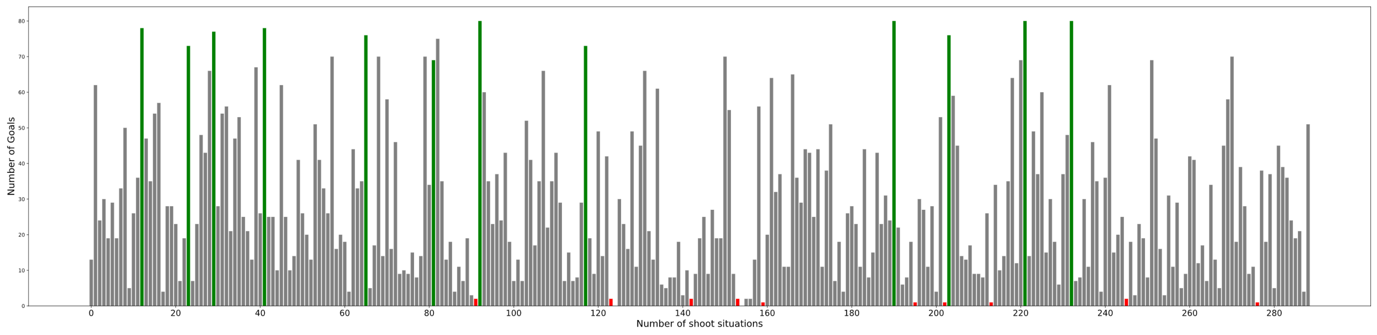


Fig. 11. Goal Distribution: Segregated Situation Highlighted.

TABLE III
SHOOT EFFICIENCY IN PREVIOUSLY EXPOSED SITUATIONS

Teams	Goals Scored	Previously Exposed Situations	Efficiency
HELIOS	322	979	0.3289
CYRUS	219	660	0.3318
YuShan	288	979	0.3814
MT	66	217	0.3041

The graph in the Fig 11 represents the goal distribution by all teams in all situations, and the colour coding to differentiate easy (green) and hard (red). The empty spaces between the situations are situations with zero goals.

To further analyse these situations, two situations from each category were selected at random and were exposed to the participating teams for 100 iterations.

TABLE IV
SHOOTING EFFICENCY(SITUATION SEGREGATION)

Team	Easy		Hard	
	Situation I	Situation II	Situation I	Situation II
HELIOS	0.99	0.95	0.14	0.12
CYRUS	0.93	0.87	0.06	0.03
YuShan	0.98	0.90	0.04	0.05
MT	1.0	0.97	0.01	0.05
Avg	0.975	0.923	0.063	0.063

From the Table IV, the situations that were categorized hard had an average shooting efficiency of 0.0625 in contrast to 0.948 for easy situations succinctly agreeing to the categorization.

VIII. Conclusion and Future Work

This paper presented a method to extend the RoboCup soccer server towards benchmarking specific skills and preparing skill-specific datasets based on the analysis of rcg log files. Initial work comprised of research of related literature in the field of robotic soccer towards the development of a framework and generating skill-specific situational datasets to be able to serve as a benchmark for evaluation of behavioral aspects and strategic gameplay of different teams. An dataset was prepared for shoot specific situations which contained JSON files relating to every occurrence of a goal corresponding to a shoot action. The 2D RCSS framework was updated to accommodate ball and player parameters towards server state initialization and start the gameplay from a desired state, and subsequently terminate the server after a stipulated cycle interval. This partial gameplay generated logs which were consequently stored in a dataset in the form of rcg logs. This dataset corresponded to a dataset of skill-specific situations which had been utilized towards benchmarking the team. The resultant dataset generated from benchmarking had 23059 goals situations with a total of 8157 successful situations. After successful accomplishment of the experiments, it was concluded that the extension of the server to accommodate partial gameplays to be crucial towards developing skill-specific situational datasets for further evaluation and comparison.

Multiple routes of future work are proposed beyond the scope of this paper. The framework in the paper has only dealt with a dataset of shoot situations to evaluate shoot efficiency. This dataset could also be employed to assess defensive strategies as well. The skill-specific situational datasets could also be extended towards developing multiple other offensive, defensive, or general skills. The dataset of shoot situations considered matches from RoboCup 2019 tournament. This could also be extended to include shoot situations from other matches from the past. Finally, Reinforcement Learning(RL) as a technique to develop a team was not in the scope of this paper. The dataset generated from the framework can be used in conjunction with clients in developing a state-of-the-art RL based teams .

IX. Acknowledgement

The authors of this article would like to extend sincere gratitude to Dr Thiago E Alves de Oliveira, Lakehead University, for his continuous support and invaluable guidance towards the completion of this project and allowing us to access all the resources at his disposal.

X. Conflict of Interest

The authors of this article declare that there is no conflict of interest regarding the publication of this paper.

References

[1] K. Kurach, A. Raichuk, P. Stańczyk, M. Zając, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet, and S. Gelly, "Google Research Football: A Novel Reinforcement Learning Environment," 2019.

[2] H. Shi, Z. Lin, K. S. Hwang, S. Yang, and J. Chen, "An Adaptive Strategy Selection Method with Reinforcement Learning for Robotic Soccer Games," *IEEE Access*, vol. 6, pp. 8376–8386, 2018.

[3] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," *Autonomous Robots*, vol. 27, no. 1, pp. 55–73, 2009.

[4] P. Nycander and P. Se, "RoboCup 2D Soccer Simulation League An analysis of the team WrightEagle and the creation of a basic team," *Dissertation*, 2013.

[5] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "RoboCup," 1997.

[6] Robocup, *RoboCup Federation official website*, 2020 (accessed October 9, 2020). https://www.robocup.org/.

[7] H. Akiyama and T. Nakashima, "HELIOS base: An open source package for the RoboCup soccer 2D simulation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8371 LNAI, no. October, pp. 528–535, 2014.

[8] M. Chen, K. Dorer, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, J. Murray, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin, "RoboCup Soccer Server 7.07, Manual," p. 150, 2003.

[9] H. Rodrigues, L. Oliveira, J. Bampton, and S. Wallace, "herodrigues/robocup2d-tutorial."

[10] O. Michael, O. Obst, F. Schmidsberger, and F. Stolzenburg, "RoboCup-SimData: A RoboCup soccer research dataset," *arXiv*, pp. 1–6, 2017.

[11] S. Glaser, "archive.robocup.info." Last accessed: 2020-01-12.

[12] M. Karimi and M. Ahmadzadeh, "Mining RoboCup Log Files to Predict Own and Opponent Action Available Online at www.ijarcs.info International Journal of Advanced Research in Computer Science Mining RoboCup Log Files to Predict Own and Opponent Action," no. July 2014, 2017.

[13] Y. Zhang and A. K. Mackworth, "A constraint-based robotic soccer team," *Constraints*, vol. 7, pp. 7–28, 2002.

[14] I. Noda, S. Suzuki, H. Matsubara, M. Asada, and H. Kitano, "Overview of robocup-97," in *RoboCup-97: Robot Soccer World Cup I* (H. Kitano, ed.), (Berlin, Heidelberg), pp. 20–41, Springer Berlin Heidelberg, 1998.

[15] M. Prokopenko, P. Wang, S. Marian, A. Bai, X. Li, and X. Chen, "RoboCup 2D soccer simulation league: Evaluation challenges," *arXiv*, no. June, 2017.

[16] B. Browning, J. Bruce, M. Bowling, and M. Veloso, "STP: Skills, tactics, and plays for multi-robot control in adversarial environments," *Proceedings of the Institution of Mechanical Engineers. Part I: Journal of Systems and Control Engineering*, vol. 219, no. 2, pp. 33–52, 2005.

[17] D. Schwab, Y. Zhu, and M. Veloso, "Learning Skills for Small Size League RoboCup," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11374 LNAI, pp. 83–95, 2019.

[18] D. v. Heesch. https://www.doxygen.nl/index.html. Last accessed 2020-12-03.

Appendix 1 - Description of Log Parameters

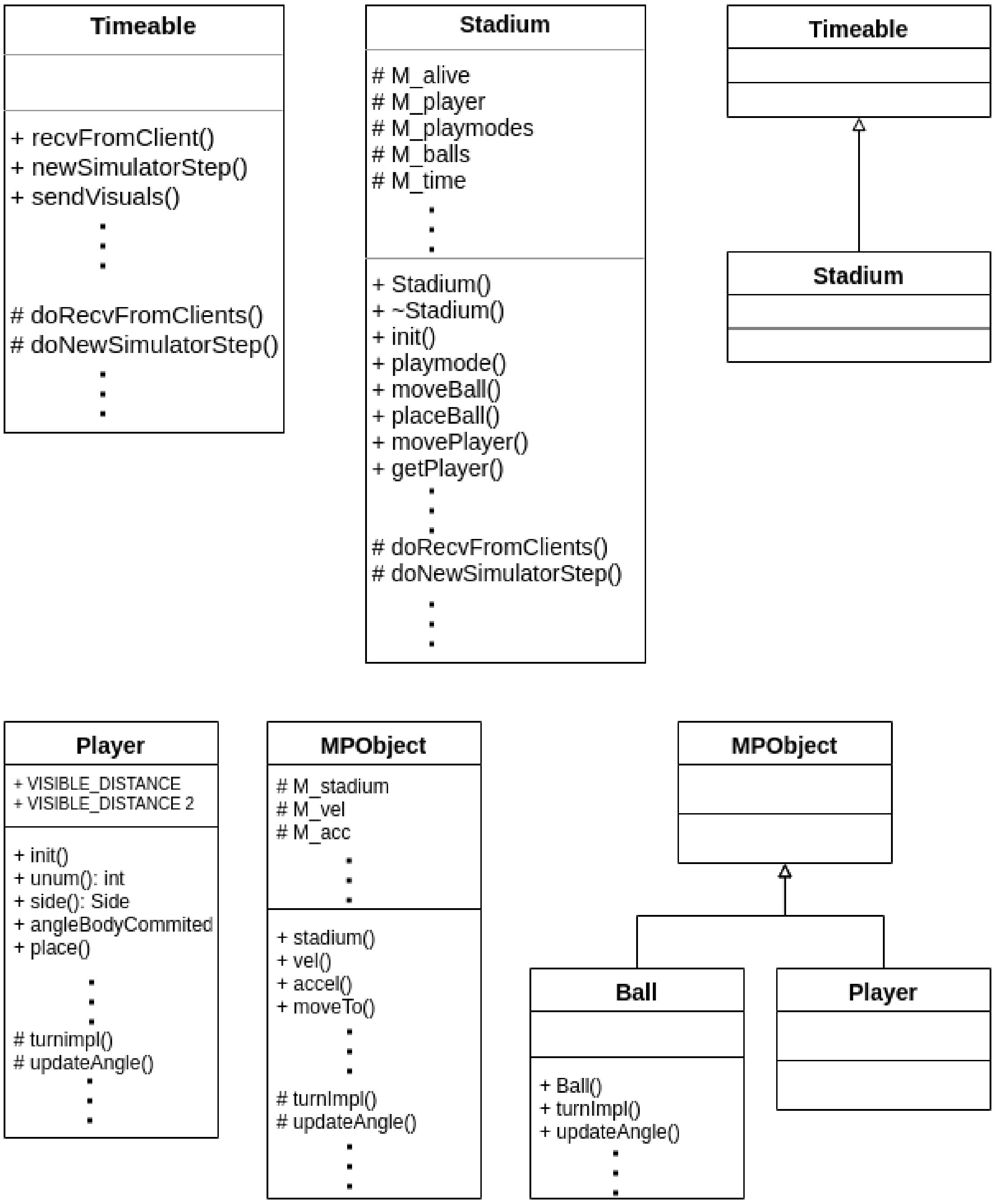
(show 2220	:	cycle number
((b)	:	ball parameters
36.4673	:	ball position along x axis
27.0083	:	ball position along y axis
0.494	:	ball velocity along x axis
0.0785)	:	ball velocity along y axis
((l 1)	:	player uniform number and side
17	:	player parameters
0x9		
-25.2002	:	player position along x axis
6.5416	:	player position along y axis
0.1348	:	player velocity along x axis
0.0066	:	player velocity along y axis
3.46	:	player body angle
-4	:	player head angle
(v h 180)	:	player view width
(s	:	stamina parameters
7932.16	:	player stamina
0.92242	:	player effort
1	:	player recovery
68799.2)	:	player turn neck
(f l 11)	:	player facing player uniform number and side
(c	:	count parameters
0	:	player kick count
1010	:	player dash count
1483	:	player turn count
0	:	player say count
1	:	player turn neck count
2494	:	player catch count
87	:	player move count
103	:	player change view count
0	:	player point to count
0	:	player attention to count
1588))	:	player tackle count

Appendix 2 - Input JSON format

```
{
  "cycle": 2241,
  "interval": 45,
  "ball":
    {
          "posx": -41.521,
          "posy": 0.4036,
          "velx": 0.1717,
          "vely": -0.1734
     },
   "players":
    {
          "player":
               [
                    {
                         "unum": 1,
                         "side": "l",
                         "posx": -48.1578,
                         "posy": -1.5131,
                         "velx": -0.115,
                         "vely": 0.2078,
                         "bangle": 118.892,
                         "hangle":-90.0,
                         "viewWidth": 60,
                         "stamina": 7945.0,
                         "effort": 1.0,
                         "recovery": 1.0,
                         "turnNeck": 76479.7,
                         "facePlayerSide": "l",
                         "facePlayerNum": 2,
                         "kickCount": 8,
                         "dashCount": 487,
                         "turnCount": 2101,
                         "sayCount": 3,
                         "turnNeckCount": 6,
                         "catchCount": 2605,
                         "moveCount": 117,
                         "changeView": 269,
                         "pointtoCount": 0,
                         "attentoCount": 485,
                         "tackleCount": 837
                      },
                      {...}

               ]
     }
}
```

Appendix 3 - Important Class Diagram


Timeable
+ recvFromClient()
+ newSimulatorStep()
+ sendVisuals()
# doRecvFromClients()
# doNewSimulatorStep()
Stadium
# M_alive
# M_player
# M_playmodes
# M_balls
# M_time
+ Stadium()
+ ~Stadium()
+ init()
+ playmode()
+ moveBall()
+ placeBall()
+ movePlayer()
+ getPlayer()
# doRecvFromClients()
# doNewSimulatorStep()
Timeable
Stadium
Player
+ VISIBLE_DISTANCE
+ VISIBLE_DISTANCE 2
+ init()
+ unum(): int
+ side(): Side
+ angleBodyCommited
+ place()
# turnimpl()
# updateAngle()
MPObject
# M_stadium
# M_vel
# M_acc
+ stadium()
+ vel()
+ accel()
+ moveTo()
# turnImpl()
# updateAngle()
MPObject
Ball
+ Ball()
+ turnImpl()
+ updateAngle()
Player