Cyrus2D Base: Source Code Base for RoboCup 2D Soccer Simulation League

Nader Zare[1](✉), Omid Amini[4], Aref Sayareh[5], Mahtab Sarvmaili[1], Arad Firouzkouhi[6], Saba Ramezani Rad[6], Stan Matwin[1,2], and Amilcar Soares[3]

[1] Institute for Big Data Analytics, Dalhousie University, Halifax, Canada
{nader.zare,mahtab.sarvmaili}@dal.ca, stan@cs.dal.ca
[2] Institute for Computer Science, Polish Academy of Sciences, Warsaw, Poland
[3] Memorial University of Newfoundland, St. John's, Canada
amilcarsj@mun.ca
[4] Qom University of Technology, Qom, Iran
[5] Shiraz University, Shiraz, Iran
[6] Amirkabir University of Technology, Tehran, Iran
{arad.firouzkouhi,saba_ramezani}@aut.ac.ir

Abstract. Soccer Simulation 2D League is one of the major leagues of RoboCup competitions. In a Soccer Simulation 2D (SS2D) game, two teams of 11 players and one coach compete against each other. Several base codes have been released for the RoboCup soccer simulation 2D (RCSS2D) community that have promoted the application of multi-agent and AI algorithms in this field. In this paper, we introduce "Cyrus2D Base", which is derived from the base code of the RCSS2D 2021 champion. We merged Gliders2D base V2.6 with the newest version of the Helios base. We applied several features of Cyrus2021 to improve the performance and capabilities of this base alongside a Data Extractor to facilitate the implementation of machine learning in the field. We have tested this base code in different teams and scenarios, and the obtained results demonstrate significant improvements in the defensive and offensive strategy of the team.

Keywords: 2D Soccer Simulation · RoboCup · Base code

1 Introduction

Soccer is one of the most popular team-based sports in the world. This is a multi-player, real-time, strategic, and partially observable game in which players of each team should cooperate to score more goals. In addition to the cooperative strategy, the players should manage different tactical and technical strategies against their opponent. Designing and implementing this game in a good, realistic graphical simulation environment and encouraging researchers to develop fully autonomous players with human-like skills creates complex challenges for A.I. research. Hence, soccer is considered an exciting environment for developing A.I. and robotic algorithms to solve real-world challenges. The importance of soccer as a game and as a challenging domain for testing the A.I. and machine learning algorithms led to an overreaching vision of a robotic team competing against the best human team by 2050 [1].

The world Cup Robot Soccer Initiative was founded to create a realistic environment similar to real soccer that encourages researchers to employ Robotic and A.I. for solving wide ranges of problems [2]. The first RoboCup was held during the IJCAI-97 [3], and it offered three competition tracks: real robot league, software robots, and expert robot competition. Among them, the Soccer Simulation 2D league (SS2D) [4] provides a wide range of research challenges such as autonomous decision-making, communication and coordination, tactical planning, collective behaviour and teamwork, opponent modelling and behavior predicting [5–12].

In this league, the RoboCup Soccer Simulation Server (RCSSServer) executes and manages a 2D soccer game between two teams of eleven autonomous software programs(agents). It holds the complete knowledge of the game, such as the exact position of every element in the game and their movements. The game further relies on the communication between the server and each agent. Each player receives relative and noisy information about the environment, and based on its logic and algorithms, the agent produces basic commands (like dashing, turning, or kicking) to influence the environment. A visual example of the game is shown in Fig. 1. Another key component of this league is the base code[1] of agents that is responsible for communicating with the server, handling the noisy partial observability of the game, modeling the server world, and making multi-agent decisions throughout the game. Due to the complexity of these tasks, designing an operational base code can astonishingly accelerates the RSS2D teams' progress.

Over the past years, many teams have contributed to the RCSS2D community by releasing their bases, which are mentioned below. One of the first bases was from Carnegie Mellon University, a.k.a "CMUnited" in 2001 [13,14], then a windows-based team was released by "TsinghuAeolus" [15] in 2002. The release of "UvA Trilearn" base [16] in 2003, helped many teams worldwide. "Brainstormers" [17], "WrightEagle" [18] and "Marlik" [19] released their team codes in 2005, 2011 and 2012 respectively. "HELIOS-Base or Agent2D" has been released by the "HELIOS" team from AIST Information Technology Research Institute [20,21]; this is the most important, most relevant, and most frequently used publicly available source code release in soccer simulation 2d. It has been considered as the base code for many prosperous teams such as Cyrus2d [24–26] and Glider2d [27,29]. Later, "Cyrus2D" and "Gliders2D" released their 2014 [22] and 2019 [23] bases respectively that are based on agent2d.

In this paper we are planning to describe and release a more advanced base that is called Cyrus2D in three consecutive versions. We have followed the incremental strategy of evolving base code proposed by [23,30] to exemplify the impact of different approaches and to trace their functionalities. In the first

[1] For simplicity, throughout this paper we will use the "base" term instead of base code.

version(v0.0) we combined the newest release of Helios and Gliders bases with some modifications on their parameters. In the second version(v1.0), we have developed this base code to include three A.I.-based components of Cyrus2D that were successfully implemented in this team. Finally, in the third version(v1.1) we took advantage of Pass Prediction Deep Neural Network module for unmarking decision-making. The performance of these versions went on the rigorous evaluations against the Agent2D, Glider2D bases and the obtained outcomes (number of scored and received goals, and the winning rate) proved the prevalence of our base code. The rest of the paper is organized as follows: Sect. 2 we will define the foundation of our base (version zero), in the next section we will explain the deployment of three ideas(Blocking Strategy, Offensive Risk Evaluation, Simple Unmarking Strategy) on the Cyrus2D v0.0 which results in Cyrus2D v1.0. In Sect. 4, we will present the idea of using Pass Prediction in Unmarking Strategy(Cyrus2D v1.1). In the next section, we will compare Cyrus2D base with other Soccer Simulation 2D bases against best three teams in RoboCup 2021. Finally, we talk about our future works.

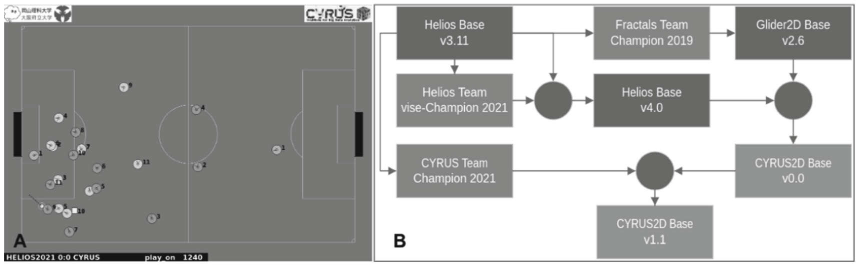

Fig. 1. A: Soccer simulation 2D league. B: The evolution of Helios2D, Glider2D and Cyrus2D base codes

2 Cyrus2D Base Version 0.0

One of the most popular SS2D bases is the Helios Base (agent2d) V3.11 which was released in 2010 [20,21]. This base includes several components such as *librcsc-4.0.0, soccerwindow2-5.0.0 and fedit2-2.0.0.* Gliders and Fractals, who use the Agent2D base, won the championship of RoboCup 2016 and 2019, respectively [27–29]. They also released a simplified version of their teams called Gliders2d base [23,30]. It is an advanced version of Helios base v3.11 with improved formation, passing behavior, and stamina management. It employs a modified version of the Marlik team [19] blocking algorithm, and few unique strategies specifically designed for each team. On the other hand, Helios has started improving its base and components such as *librcsc* based on the new versions of C++ from 2019 [31,32]. In this paper we start by introducing the first version of Cyrus2D base (V0.0). It is established by rewriting the newest version of the Agent2D by merging the latest Gliders2D base (see Figure 1[B]). This base code is fully compatible with the latest version of *librcsc*, but the blocking algorithm and tuning parameters of the Gliders2D base are removed. The Cyrus2D base is released in the Cyrus team repository and will be updated to be compatible with the *rcssserver* and *librcsc*[2]. In order to enhance the functionality of this base, we have implanted three simplified functionalities of Cyrus on this base and we introduce them as the consecutive versions of Cyrus2D base. In the following sections, we will describe these ideas.

3 Cyrus2D Base Version 1.0

3.1 Blocking Strategy [3]

As the environment of SS2D is highly dynamic and unpredictable, an innovative defensive strategy can increase the winning chance of the team. To establish the defensive strategies, we need to understand defensive actions and how players can cooperatively perform to minimize the risk of receiving the goal. Blocking and marking are two main defensive actions that prevent the opposing team from controlling the ball and playing with it. Blocking stops the progress of the opponent's ball holder on the field, and marking prevents the passing of the ball to the opposing team players. Therefore, when one of our agents tries to block the ball holder, the other players should choose to mark the opponent players. In the Cyrus2D base, we implemented multi-agent blocking decision-making. The blocking function or "Blocking Simulator" is called when the opponent owns the ball. It simulates the dribbling behavior of the opponent ball holder called the "dribbling curve" and then finds a position that one of our players can arrive in, before arrival of the opponent's ball holder and (our) players. To simulate the dribbling curve, it predicts the first position of the ball that the opponent's player can kick the ball. In the next step, it predicts the following ball positions of dribbling behavior. The dribbling speed is considered 0.7 m/s. To find the dribbling direction, we evaluate ten positions around the ball position using the reversed formulation of "Field Evaluator" in Helios base. To improve the performance of the Blocking algorithm, we implemented some conditions to prevent players from using extra stamina or going far from their home position.

3.2 Offensive Risk Evaluation [4]

To score more goals, the team's ball holder must move the ball towards the opponent's goal area, and a final striker must shoot the ball towards the goal. Dribbling and passing are examples of possible actions that can lead the ball towards the goal. Henceforth, the ball holder must choose the best action between the possible passes and dribbles. For this purpose, we need to scrutinize our base code and improve the implementation of the offensive strategy.

[2] https://github.com/Cyrus2D/Cyrus2DBase.

[3] This Algorithm Is Implemented in Src/bhv_basic_block.cpp.

[4] This Algorithm Is Implemented in Src/chain_action/action_chain_graph.cpp.

The Agent2D base has a decision-making algorithm called *Chain-Action*, which uses a modified version of Breadth-First Search to decide an action for the ball owner in an action graph tree. The Chain-Action has *action generator* modules such as Pass-Generator, Short-Generator, and Dribble-Generator. An *action generator* module receives a state of the game and then generates all possible actions in that state. The Chain-Action also includes a simple *predictor module* that receives a state and an action; then, it generates a new state. It simulates the possible outcome of the game after applying the received action [26].

After predicting a new state, Chain-Action evaluates the state based on the ball position using a module called *Field-Evaluator*. This module receives a state and uses the X coordinate of the ball and its distance to the opponent's goal to measure its value. To expand the tree to the next level, the chain-action chooses a node with the maximum value. An example of this procedure is shown in Fig. 2.

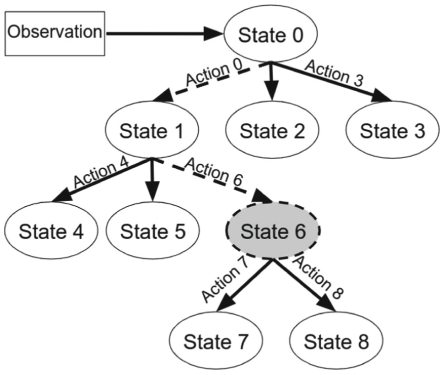

Fig. 2. Example of Chain-Action. A multi-branch tree search is performed. Each edge presents an action and each node corresponds to a state instance.

We improved the Field-Evaluator module by including a term that is subtracted to its calculation algorithms which is called *Offensive Risk Evaluation (ORE)*. This value is calculated based on the minimum number of cycles that the opponent players need to reach the ball in the input state of Field-Evaluator.

The Field-Evaluator first calculates the minimum number of cycles c that the opponent player needs to reach the ball. Then it uses an array with seven elements where the n-Th element would be the ORE term if the opponent reaches the ball in n-Th-cycles. To populate this array, we took advantage of the genetic algorithm[5].

For our task, the genetic representation is a list of seven values. A solution must be an array of seven values between 0 to 50, in descending order, as opponents closer to the ball are more dangerous. The fitness function is the average

[5] We reduced the array size to seven because our GA algorithm with several settings found that the eighth and following cells of the best arrays will be 0.

goal difference of Cyrus2D base in 100 games against random opponents from 10 teams of RoboCup competitions in 2021. To initialize the first population, we randomly generated 100 solutions. After evaluating with the fitness function, we generate 80 new children, from 160 parents which are selected randomly with probability based on their fitness score. After cross-over, we update the new children to possible solutions by making sure each value is less than or equal to the value before it. In the next step, some of their genes can be mutated with a low random probability. The mutation is done in a manner that the mutated solution is still considered possible.

If the generated solution is not in descending order, to preserve the validity of the solution, we replace the first occurrence of illegal value with a value lower than the previous element for example if the generated solution looks like *solution* = [10 18 5 4 3 2 1], the validity procedure transforms it to *fixed solution* = [10 9 5 4 3 2 1]. Afterwards, we create the new population of 100 by selecting 20 of the best chromosomes of the previous generation and adding the 80 new children.

We repeat this process until the population converges or until 100 iterations are evaluated.

3.3 Unmarking Strategy [6]

Unmarking is the player's ability to move, avoid being marked, and relocate himself in a space where he could receive a pass from the ball possessor.

In the unmarking algorithm, a player who wants to unmark is called the "unmarker", and the player who will pass the ball to the unmarker is called the "passer". The passer player can be a player who owns the ball or does not have ball possession at the moment, but it is possible to be a ball possessor in the future. An unmarking Strategy identifies the passer, and after identifying the passer, the unmarker should find a position to receive a pass from the passer in future cycles. An effective unmarking should consider the actions of other agents and the cooperation between them.

Cyrus2D base version 1.0 includes a simple unmarking strategy. In this algorithm, all players do unmarking for the ball possessor to receive a pass from him. After identifying the passer, the unmarker simulates ten targets in ten directions around him according to its previous movement. After generating 100 targets, it ignores targets close to teammate or opponent players and targets far from its home position. The home position is the target position of a player that is calculated based on team's formation. The next step simulates eight lead passes from the passer to itself in every target to find which target it can receive a pass. A pass has a score calculated using the "Field Evaluation" formula in the Helios base. The score of each target is calculated based on the scores of possible received passes in the position and the minimum distance of opponent players to the position. Eventually, the target with the maximum score will be selected as the unmarking target.

[6] This Algorithm Is Implemented in Src/bhv_unmark.cpp.

4 Cyrus2D Base Version 1.1 [7]

In Cyrus2D base Version 1.0, we improved the offensive strategy, using a novel Blocking behavior, and a simplified Unmarking strategy. In this section, we will explain the improvement on the Unmarking strategy using a module called Pass Prediction. The Pass Prediction module includes a trained DNN, that receives a state of the game, and identifies which player will be the pass receiver in that state. This module enables us to generate a tree that assigns a passer to each player in the future cycles of the game. To generate a data set for training the DNN, we employed Data Extractor module.

4.1 Data Extractor

As in real soccer, passing is one of the possible actions that can lead the ball to the goal. Predicting the pass target player, from the point of view of the ball possessor, has many benefits in defensive and offensive algorithms. In this paper, the ball possessor is the player who can kick the ball in the current cycle or receive the ball in the future cycles.

To predict the behavior of (our) ball possessor, we were required to create the dataset of game states from this player point of view. For this purpose, we embedded a Data Extractor module in each one of the players and then we recorded the features of game states and their corresponding label. The label shows the uniform number of the player who is target of the best pass [12,25,26].

To generate a data set, our player (ball holder) feeds the state of the game and its selected pass receiver uniform number to the Data Extractor when the ball is in its kickable area. After that it saves the features and the label in a CSV file. Later this dataset will be used to train the Pass Prediction model.

4.2 Unmarking Strategy with Help of Pass Prediction Module

To improve the Unmarking Strategy and sketching the flow of the game, each player of Cyrus2D base tries to simulate a tree that includes the probable passes and their outcoming states. Each node of the tree contains a state of the game where one of our players is the ball owner in that state. The edges from the current state shows a probable pass in the future. The root node of the tree is the first state of the game where one of our players can kick the ball. A player can not be ball possessor in more than one node of the tree. To create the tree, the unmarker feeds the state of the root node to the Pass Prediction module, and receives the probability of players for receiving a pass. This module includes a trained DNN that can receives features of the game generated by Data Extractor and gives probability of players to receive a pass in the given state. In the next step, the unmarker selects two passes with the maximum probability higher than a limit and inserts them into a list called Pass List.

[7] This Algorithm Is Implemented in Src/bhv_unmark.cpp, Src/data_extractor/DEState.cpp and Src/data_extractor/offensive_data_extractor.cpp.

Then after, it pops the pass with highest probability from the Pass List. Next, it simulate the outcome that it send the outcome state to Pass Prediction module and eventually insert best passes from the outcome of Pass Prediction module. This procedure continues until the number of tree nodes is equal to ten or there is not any pass in the Pass List. Figure 3 shows an overview of the Unmarking Decisioning. After termination of this procedure, the umarker agent looks for its corresponding node in the tree, then it chooses the parent player as the "Passer" for the unmarking procedure in order to receive the pass from that player in the future.

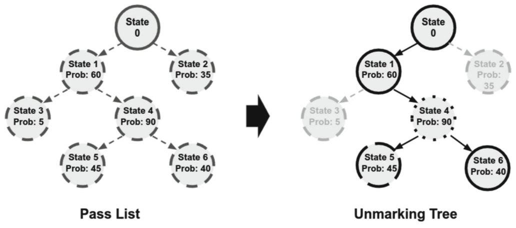

Fig. 3. Overview of the unmark decisioning algorithm. The left tree shows the result states and their points. The Bold circles (full, dotted, and long dashed) in the right tree present the selected best nodes from the candidate list. The dashed circle one indicates the node that the unmarker is the ball owner in its state, and its parent node is the dotted circle.

5 Results

5.1 Training DNN for Cyrus2D V1.1 [8]

For generating a data-set for training the "Pass Prediction DNN", we ran 500 games against Helios Base v3.11 and newest version, Gliders2D base v1.6 and v2.6, Cyurs 2021, Helios 2021, and YuShan 2021. We obtained total 1,429,032 data instances. We split them into two subsets, 85% for training and 15% for testing. The prediction model (DNN) has three layers of 128, 64, 32 and 11 neurons, with RELU activation function and a softmax function at the last layer. The validation accuracy of the trained neural network on the test data was 68.1%. We used Python TensorFlow Keras library [33] for training the model, and we implemented a library called CppDnn [34] to use the trained model in C++. The CppDnn is a C++ library powered by Eigen [37]; this library creates a deep neural network model by reading the weights of a trained DNN model.

To evaluate the impact of the implemented features and algorithms and comparing the Cyrus2D base with the HELIOS base(Ag) and Gliders2d Base(G2D), we ran X-number games between two versions of HELIOS base(3.1.1/newest), two versions of Gliders2d Base (1.6/2.6), six versions of Cyrus2D base (C2D0 = Cyrus2D zero, C2DB = Cyrus2D zero base with Blocking Strategy, C2DR

[8] All Scripts for Training Are Available in Scripts/training_unmark.

Table 1. Win rate

Team	H2D 3.11	H2D new	G2D 2.6	Cyrus21	Helios21	YuShan21	Average
H2D 3.11	–	26.2	3.5	0.0	0.0	0.0	4.9
H2D new	73.8	–	9.8	0.2	0.0	0.0	14.0
G2D 1.6	95.5	85.4	30.3	0.4	0.0	0.2	35.3
G2D 2.6	96.5	90.2	–	1.9	0.0	1.0	31.6
C2D 0.0	100.0	98.1	78.5	7.2	0.0	4.3	48.0
C2D B	99.6	97.2	77.0	6.4	0.0	3.6	47.3
C2D R	99.4	97.4	81.0	7.9	0.2	6.3	48.7
C2D U	100.0	99.0	80.8	5.6	0.2	5.8	48.6
C2D 1.0	99.3	98.6	79.9	8.6	0.3	4.6	48.6
C2D 1.1	99.8	99.6	84.1	5.8	0.8	4.9	49.1

Table 2. Goals scored (Goals Conceded)

Team	H2D 3.11	H2D new	G2D 2.6	Cyrus21	Helios21	YuShan21	Average
H2D 3.11	—	1.6(2.7)	0.5(3.0)	0.2(6.2)	0.1(13.0)	0.2(7.5)	0.4(5.4)
H2D new	2.7(1.6)	—	0.7(2.3)	0.3(5.9)	0.1(11.1)	0.3(6.4)	0.7(4.5)
G2D 1.6	3.5(0.7)	2.6(1.0)	0.8(1.4)	0.5(5.3)	0.1(6.5)	0.2(4.8)	1.3(3.3)
G2D 2.6	3.0(0.5)	2.3(0.7)	—	0.5(3.5)	0.1(5.5)	0.2(3.8)	1.0(2.3)
C2D 0.0	4.3(0.2)	2.8(0.3)	1.1(0.4)	0.6(2.8)	0.2(3.6)	0.3(1.9)	1.6(1.5)
C2D B	4.2(0.2)	2.9(0.2)	1.1(0.4)	0.6(2.7)	0.2(3.8)	0.3(2.4)	1.6(1.6)
C2D R	4.1(0.2)	2.9(0.3)	1.1(0.4)	0.6(2.6)	0.2(3.8)	0.3(1.7)	1.5(1.5)
C2D U	4.4(0.2)	3.2(0.3)	1.3(0.5)	0.6(2.9)	0.2(4.0)	0.4(2.1)	1.7(1.7)
C2D 1.0	4.8(0.2)	3.6(0.2)	1.3(0.4)	0.7(2.6)	0.2(3.9)	0.4(2.5)	1.8(1.6)
C2D 1.1	4.4(0.2)	3.2(0.2)	1.2(0.4)	0.6(2.8)	0.2(3.8)	0.3(2.3)	1.7(1.6)

= Cyrus2D zero base with ORE, C2DU = Cyrus2D zero base with UnMarking Strategy, C2DV1.0 = Cyrus2D version one with all of the previous features and C2DV1.1 = V1.0 with pass prediction) against Helios bases [20], Glider2d Base(v2.6) and three of the best teams in RoboCup (Helios2021 [35], YuShan [36], Cyrus2021 [25]).

Table 1 shows the expected winning rate of all version of Cyrus against opponent teams. The winning rate is calculated by $num\_wins/(num\_games - num\_draws)$. Table 2 presents the average number of our scored goals and conceded goals respectively.

The results demonstrate Cyrus2D base v1.1 prevalence over other released bases. For instance the Cyrus2D base wins Helios and Gliders2D bases in more than 99% and 84% of games respectively. The average win-rate of Cyrus2D against best three RoboCup teams is 3.76 (0.2% to 3.8%) percent higher than the winning rate of Helios base against those teams, and 2.86 (2.9% to 3.8%) percent higher than Gliders2D base.

6 Conclusion

In this paper, we aimed to introduce three versions of Cyrus2D base code and their particular features. The first version of Cyrus2D base was created by combining the latest release of Helios Agent2D and Gliders2D bases. For this version, we removed some of the fine tuned parameters. In the next version, Cyrus2D v1.0, we have upgraded the Blocking, and offensive strategy by using the Offensive Risk Evaluation and unmarking behavior. In the Cyrus2D v1.1 we improved the unmarking behavior using the Pass Prediction. To evaluate the performance of Cyrus2D, we ran 500 games against Gliders2D, Helios base, and best three teams in RoboCup 2021. The obtained results shows significant improvement on win-rate, scored goals and conceded goals. For our future work, we are planning to enhance the Cyrus2D base in terms of chain action movement prediction, and marking by using multi-agent decision-making.

Acknowledgements. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). We thank the HELIOS and Gliders teams for their code bases and extraordinary contributions to the SS2D league.

References

1. Burkhard, H.D., Duhaut, D., Fujita, M., Lima, P., Murphy, R., Rojas, R.: The road to RoboCup 2050. IEEE Robot. Autom. Mag. **9**(2), 31–38 (2002)
2. Noda, I. and Matsubara, H.: Soccer server and researches on multi-agent systems. In Proceedings of the IROS-96 Workshop on RoboCup, pp. 1–7 (1996)
3. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: Robocup: the robot world cup initiative. In: Proceedings of the 1st International Conference on Autonomous Agents, pp. 340–347 (1997)
4. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: RoboCup: a challenge problem for AI. AI Mag. **18**(1), 73–73 (1997)
5. Noda, I., Stone, P.: The RoboCup soccer server and CMUnited clients: implemented infrastructure for MAS research. Auton. Agents Multi-Agent Syst. **7**(1–2), 101–120 (2003)
6. Riley, P., Stone, P., Veloso, M.: Layered disclosure: revealing agents' internals. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 61–72. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44631-1_5
7. Stone, P., Riley, P., Veloso, M.: Defining and using ideal teammate and opponent models. In: Proceedings of the 12th Annual Conference on Innovative Applications of Artificial Intelligence (2000)
8. Butler, M., Prokopenko, M., Howard, T.: Flexible synchronisation within robocup environment: a comparative analysis. In: Stone, P., Balch, T., Kraetzschmar, G. (eds.) RoboCup 2000. LNCS (LNAI), vol. 2019, pp. 119–128. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45324-5_10
9. Reis, L.P., Lau, N., Oliveira, E.C.: Situation based strategic positioning for coordinating a team of homogeneous agents. In: BRSDMAS 2000. LNCS (LNAI), vol. 2103, pp. 175–197. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44568-4_11

10. Prokopenko, M., Wang, P.: Relating the entropy of joint beliefs to multi-agent coordination. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002. LNCS (LNAI), vol. 2752, pp. 367–374. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45135-8_32
11. Prokopenko, M., Wang, P.: Evaluating team performance at the edge of chaos. In: Polani, D., Browning, B., Bonarini, A., Yoshida, K. (eds.) RoboCup 2003. LNCS (LNAI), vol. 3020, pp. 89–101. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25940-4_8
12. Zare, N., Sarvmaili, M., Sayareh, A., Amini, O., Matwin, S., Soares, A.: Engineering features to improve pass prediction in soccer simulation 2d games. In: Alami, R., Biswas, J., Cakmak, M., Obst, O. (eds.) RoboCup 2021. LNCS (LNAI), vol. 13132, pp. 140–152. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-98682-7_12
13. Stone, P., Asada, M., Balch, T., Fujita, M., Kraetzschmar, G., Lund, H., Scerri, P., Tadokoro, S., Wyeth, G.: Overview of Robocup-2000. In: Stone, P., Balch, T., Kraetzschmar, G. (eds.) RoboCup 2000. LNCS (LNAI), vol. 2019, pp. 1–29. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45324-5_1
14. Stone, P., Riley, P., Veloso, M.: The CMUnited-99 champion simulator team. In: Veloso, M., Pagello, E., Kitano, H. (eds.) RoboCup 1999. LNCS (LNAI), vol. 1856, pp. 35–48. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45327-X_2
15. Yao, J., Chen, J., Cai, Y., Li, S.: Architecture of TsinghuAeolus. In: Birk, A., Coradeschi, S., Tadokoro, S. (eds.) RoboCup 2001. LNCS (LNAI), vol. 2377, pp. 491–494. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45603-1_66
16. Kok, J.R., Vlassis, N., Groen, F.: UvA Trilearn 2003 team description. In: Polani, D., Browning, B., Bonarini, A., Yoshida, K. (eds.) Proceedings CD RoboCup 2003. Springer, Padua (2003)
17. Riedmiller, M., Gabel, T., Knabe, J., Strasdat, H.: Brainstormers 2d - team description 2005. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) Proceedings CD RoboCup 2005. Springer (2005)
18. Bai, A., Chen, X., MacAlpine, P., Urieli, D., Barrett, S., Stone, P.: WrightEagle and UT austin villa: RoboCup 2011 simulation league champions. In: Röfer, T., Mayer, N.M., Savage, J., Saranlı, U. (eds.) RoboCup 2011. LNCS (LNAI), vol. 7416, pp. 1–12. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32060-6_1
19. Tavafi, A., Nozari, N., Vatani, R., Yousefi, M.R., Rahmatinia, S., Pirdir, P.: MarliK 2012 soccer 2D simulation team description paper. In: RoboCup 2012 Symposium and Competitions: Team Description Papers, Mexico City, Mexico (2012)
20. Akiyama, H., Nakashima, T.: HELIOS base: an open source package for the robocup soccer 2d simulation. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013. LNCS (LNAI), vol. 8371, pp. 528–535. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44468-9_46
21. Akiyama, H.: Agent2D base code. http://www.rctools.sourceforge.jp (2010)
22. Khayami, R., et al.: CYRUS 2D simulation team description paper 2014. In: RoboCup 2014. Joao Pessoa, Brazil (2014)
23. Prokopenko, M., Wang, P.: Gliders2d: source code base for RoboCup 2D Soccer simulation league. CoRR abs/1812.10202 (2018)
24. Zare, N., et al.: Cyrus Soccer 2D Simulation Team Description Paper: In: RoboCup 2013, p. 2013. Eindhoven, Netherlands (2013)
25. Zare, N., Sayareh, A., Sarvmaili, M., Amini, O., Soares, A., Matwin, S.: CYRUS 2D soccer simulation team description paper 2021. In: RoboCup 2021 Symposium and Competitions, Worldwide (2021)

26. Zare, N., et al.: improving dribbling, passing, and marking actions in soccer simulation 2D games using machine learning. In: Alami, R., Biswas, J., Cakmak, M., Obst, O. (eds.) RoboCup 2021. LNCS (LNAI), vol. 13132, pp. 340–351. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-98682-7_28
27. Prokopenko, M., Wang, P., Obst, O., Jaurgeui, V.: Gliders 2016: integrating multi-agent approaches to tactical diversity. In: RoboCup 2016 Symposium and Competitions: Team Description Papers, Leipzig, Germany (2016)
28. Prokopenko, M., Wang, P.: Disruptive innovations in RoboCup 2D soccer simulation league: from Cyberoos'98 to gliders2016. In: Behnke, S., Sheh, R., Sarıel, S., Lee, D.D. (eds.) RoboCup 2016. LNCS (LNAI), vol. 9776, pp. 529–541. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68792-6_44
29. Prokopenko, M., Wang, P.: Fractals 2019: Guiding self-organisation of intelligent agents. In: RoboCup 2019 Symposium and Competitions, Sydney, Australia (2019)
30. Prokopenko, M., Wang, P.: Fractals2019: combinatorial optimisation with dynamic constraint annealing. In: Chalup, S., Niemueller, T., Suthakorn, J., Williams, M.-A. (eds.) RoboCup 2019. LNCS (LNAI), vol. 11531, pp. 616–630. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35699-6_50
31. Akiyama, H.: Agent2D base code. https://github.com/helios-base/helios-base (2010)
32. Akiyama, H.: LibRCSC, component of Agent2D base code. https://github.com/helios-base/librcsc (2010)
33. Martín Abadi, et al. TensorFlow: large-scale machine learning on heterogeneous systems (2015). Software available from tensorflow.org
34. Nader, Z., et al.: CPPDNN: A C++ library to use a trained DNN by Tensor Flow Keras. https://github.com/Cyrus2D/CppDNN
35. Yamaguchi, M., Kuga, R., Omori, H., Fukushima, T., Nakashima, T., Akiyama, H.: Helios 2021: team description paper. In: RoboCup 2021 Symposium and Competitions, Worldwide (2021)
36. Cheng, Z., Zhang F., Guang, B., Wang, L.: YuShan2021 team description paper for RoboCup2021. In: RoboCup 2021 Symposium and Competitions, Worldwide (2021)
37. Guennebaud, G., Jacob, B., et al.: Eigen v3 (2010). http://eigen.tuxfamily.org