Strategising RoboCup in Real Time with Uppaal Stratego

Philip I. Holler, Magnus K. Jensen, Hannah Marie K. Lockey and Michele Albano[a]
Department of Computer Science, Aalborg University, Selma Lagerløfs Vej 300, 9220 Aalborg, Denmark

Keywords: Multi-Agent Systems, Multi-Agent Software Platforms, Time Constraints, Agent Models and Architectures.

Abstract: The RoboCup simulator is a playing ground for Agents and Artificial Intelligence research. One of the main challenges provided by RoboCup is generating winning strategies for a set of agents playing soccer, given a partial and noisy view of the game state. Additionally, RoboCup is timing sensitive, and all decisions have to be sent to the server within each tick of 100ms. This paper presents a method for generating strategies by modelling players and scenarios as timed automata in the Uppaal environment. The newest extension of Uppaal, called Uppaal Stratego, allows for synthesising strategies optimising a reward function that is used to guide the decision process. In order to stay within the time frame of 100ms, two approaches were tested, namely forecasting the game state and generating a strategy asynchronously for a later point in time, and generating strategies beforehand and saving them in a lookup table. Four timed automata were developed, and were tested against publicly available methods. We found that strategies could be successfully generated and used within the time constraints of RoboCup using our proposed method. Especially when combined together, our strategies are able to outperform most published methods, but lose against the published world champions.

1 INTRODUCTION

Robotics involves many complex decision problems with tight time constraints that are often difficult to solve using traditional rule based programming, especially when the scenario involves multiple robots. This category of problems extends to higher levels of abstraction, such as strategising to solve problems in a dynamic environment.

The RoboCup federation aims to push the limits of Artificial Intelligence in robotics by providing platforms for soccer competitions between autonomous robots (Robocup Federation, 2020a). In this paper we focus on the strategisation part of the problem, as represented in the RoboCup 2D multi-agent soccer simulation (Robocup Federation, 2020b). This simulation includes a dynamic environment with a large amount of possible configurations that makes linear search for optimal solutions infeasible. The RoboCup player agents, called players for short, are autonomous agents reacting to stimuli from the environment. In order to make the players perform well in the game, it is necessary to quickly create effective strategies for the players.

Uppaal is a modelling and verification tool that allows for modelling the behaviour of timed systems in terms of states and transitions between states. Additionally, Uppaal is able to analyse and verify properties of the models using efficient algorithms (Uppaal, 2019). The latest extension of Uppaal, called Uppaal Stratego, enables strategy generations using different machine learning algorithms such as Q-learning (David et al., 2015). Uppaal Stratego has already been applied to solve problems of real time systems such as floor heating control (Larsen et al., 2016) and traffic light control (Eriksen et al., 2017). In both cases, strategies are generated in real time, but not with the tight time constraints and the large problem spaces that characterise RoboCup.

The aim of this paper is to introduce methods for applying Uppaal Stratego to solve problems that require swift reactions to a dynamic environment. We created a RoboCup team named RoboPaal to showcase our approach, and whose code is available on (RoboPaal team, 2020).

The rest of the paper is structured as follows. Section 2 provides background information regarding RoboCup and Uppaal. Section 3 introduces our general approach to strategising RoboCup, and Section 4 provides insights regarding the design and implementation of RoboPaal. Section 5 present and discusses the results of our experimentation, and Section 6 wraps up the paper.

[a] https://orcid.org/0000-0002-3777-9981

Holler, P., Jensen, M., Lockey, H. and Albano, M.
Strategising RoboCup in Real Time with Uppaal Stratego.
DOI: 10.5220/0010239602730280
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 1*, pages 273-280
ISBN: 978-989-758-484-8

2 BACKGROUND INFORMATION

Definition 1. An **objective** is defined as a goal for a single player. An example of this could be: *move to position x,y on the field*. An objective can require several player actions, i.e. dash, kick and turn, to complete, or none if the objective is already fulfilled.

Definition 2. A **strategy** is a set of objectives assigned to one or more players.

Definition 3. In this paper, a **model** is a timed automata modelled within Uppaal (David et al., 2015).

2.1 RoboCup

RoboCup is a set of annual competitions where teams of autonomous robots compete in the game of soccer. RoboCup features tournaments for both physical robots as well as simulated robots in 2D and 3D environments (Robocup Federation, 2020a). This paper focuses on the 2D simulator. Agents in the simulator must tackle problems such as interpretation of noisy data, strategisation and coordination with other robots, and finally time sensitivity as the server enforces a 100ms tick rate. The simulator is publicly available through GitHub (Rodrigues et al., 2020). The **Soccer Server** is the main component of the simulator, and is responsible for storing and updating the game state. All communication between the agents must be done through the server.

Three different types of agents can connect to the server. The **Trainer** and the **Online Coach** have both access to a perfect view of the game state. The Trainer can move objects and change the game state, it cannot be used in official matches, and it is meant to test strategies in controlled environments. The Online Coach can communicate via short messages broadcast to the players (through the server), and can issue substitutions of the players during a match.

The **Players** can communicate with each other (through the server), they periodically receive sensory data, and they send back actions to be performed. Sensory data come of three different types. Visual data consists of distances and relative directions to flags, other players and the ball, and it is distorted depending on how far away the objects are located. Body sensor data include current values of stamina and head angle; auditory data include messages from the referee, other players and the coach. Player actions are executed to influence the game state, and include *dash*, *kick*, *turn*, *turn_neck* and *say*. The kick and dash actions are accompanied by a power parameter between 0 and 100 indicating how hard to kick and how fast to dash. The player state has a stamina level that dictates the effectiveness of the dash and kick actions, which by default starts starts at the upper limit of 8000 stamina, and dash actions consume stamina equal to the power of the action. Stamina regenerates at a rate of at most 30 units per tick throughout the game (The RoboCup Simulator Committee, 2020).

2.2 Uppaal Toolsuite

Uppaal is a modelling and verification toolsuite. The latest edition of Uppaal comprises Uppaal Stratego, which is a tool for generating strategies (Uppaal, 2019). A strategy in Uppaal Stratego consists of a number of transitions in a timed automata depending on the values of variables and clocks, the latter representing the passing of time.

A common issue for model checking of timed automata is the state space explosion, which happens when the state space gets too big to analyse. Within Uppaal Stratego, the strategies are generated according to a query formulated in a query language containing variables or clocks that should be optimised (David et al., 2015). Strategies are generated using different machine learning methods, among which co-variance, Splitting, Regression, Naive, M-Learning and Q-learning (the default method). In this paper, Uppaal Stratego will only be used in its default setting using Q-learning. Models in Uppaal are saved as XML files, which allows for easy direct manipulation of the model. The Uppaal verifier, called `verifyta`, is a binary file used to run strategy generation queries on the models.

Timed automata have been used to strategise real time systems in the past. The work in (Larsen et al., 2016) used Uppaal for online synthesis of short-period strategies on the fly. In fact, the computations needed to learn an effective strategy in Uppaal grows exponentially with the number of states, which grows steepily with the time horizon for the strategy. A heating control strategy is created for the near future, and then recalculated periodically. The traffic controller described in (Eriksen et al., 2017) uses Uppaal to reduce waiting times at traffic lights by continuously creating new strategies for intelligent traffic lights.

3 STRATEGISING RoboCup

Our approach, whose reference architecture is shown in figure 1, employs Uppaal to create strategies for the Players and for the Online Coach.

A player can represent its view of the world as a Uppaal model, which is then used to generate a strategy. The player then translates the strategy into ob-

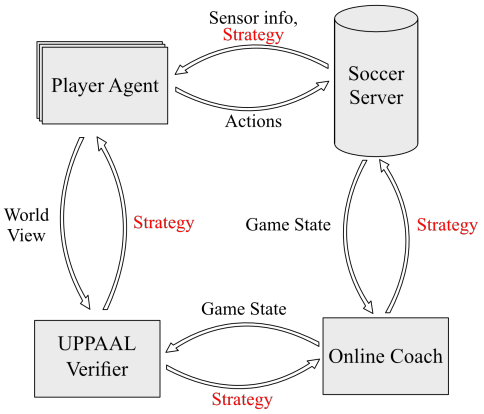

Figure 1: Reference architecture.

jectives and acts upon them by sending actions to the soccer server. This approach allows creating individual strategies for each player, based on its view of the world. The approach for the Online Coach is similar, but it requires the additional step of communicating the strategy to the players with a *Say* action.

Modelling the whole game state within Uppaal is unfeasible since a game of RoboCup can have up to 11 players for each team and each player is characterized by more than 40 variables, leading to massive state explosion. For this reason, one primary problem we studied was the choice of a proper part of the game state to model. We considered specific scenarios (free kicks), characteristics of a player (its stamina level), and choices (passing the ball or trying to keep it).

We created two distinct families of methods for generating strategies for RoboCup with Uppaal. A first family called **Online Strategy** considers to generate a strategy on the fly when it is needed. The approach must adhere to tight time constraints (100ms). The models for generating online strategies are altered using the data received from the server, and the output strategies focus on objectives of immediate applicability, such as finding the optimal target for passing the ball. If strategy generation takes too long, the strategy might be ready too late to be useful. We reduce this risk by forecasting the game state, even though we could generate a strategy for a game state that never materialises.

A second family called **Offline Strategy** considers to generate strategies before a game is run. Relevant offline strategies are stored in a lookup table. Offline strategies cannot feasibly be generated for all scenarios of the game, due to the large amount of possible states. Instead, this kind of strategy can be developed for scenarios that are predictable and repeated often. At run time the agent can then search through the offline strategies, and apply them if they match the current game state.

4 DESIGN & IMPLEMENTATION

This section describes how the architecture and all of its modules were realized in the Python programming language.

4.1 Players

The biggest development effort was devoted to the **Players**. Each of them is run on an independent thread and can communicate with the server only (see Section 2.1). Each player consists of two sub-threads: the **Communication thread** for communicating messages with the soccer server, and the **Thinker thread** that is responsible for parsing the received messages and generating actions given a game state. The two threads communicate through a thread-safe queue.

As mentioned in Section 2.1, players receive inaccurate sensory data from the server, or even not receive some data at all, which can lead to generating strategies based on wrong information. An example of the noisy data is the distance to the ball and to every other object, which is judged incorrectly by up to 10 percent (Akiyama, 2010). To mitigate this problem, the players estimate their own positions and directions by using trilateration on all visible flags on the field, and then leverage this information to estimate the position of the rest of the objects.

We also mitigate the noise problem by keeping a history of the perceived game state, to spot and correct inconsistencies in the data received by the server. Finally, we associate a time stamp with every sensory data received from the server, to both estimate the location of non perceived objects, and to forecast the future position of ball and players for online models such as the possession model (see Section 4.3.2).

Every time a message from the server is parsed by a Thinker thread, the player assesses if the current game state matches the requirements for one or more strategies. If the game state matches the requirements for an offline strategy, the strategy is retrieved from the lookup table and objectives are immediately applied for the player. If the game state matches an online strategy configuration, then the agent constructs an Uppaal model to generate the strategy (see Section 4.3). Since generating an online strategy may take longer than 100ms, it is done asynchronously using a separate thread. Upon completion, the Uppaal module returns a series of objectives to the player.

Players can also receive objectives for a strategy generated by the online coach, as *say* messages (see Section 4.2). In case no suitable strategy is found, objectives are determined by default hard coded behaviours.

4.2 Online Coach and Trainer

Similarly to the player, the **Online Coach** and the **Trainer** are run as two sub-threads (Communication thread and Thinker thread). Their data on game state are complete and not distorted, which eliminates the need for the techniques from Section 4.1. If the data received in one tick trigger the generation of a strategy, a Uppaal model is generated, run, and the Objectives it produces are communicated to the players (see for example Section 4.3.1).

4.3 Implemented Strategies

To showcase our approach, we implemented one offline strategy generation module (the **Goalie Defence Model**) and three online strategy generation modules (the **Pass-chain Model**, the **Stamina Model** and the **Possession Model**).

In the Uppaal models, the possible objectives that the player can pursue are represented as controllable transitions. For example a transition might represent passing the ball to a specific player. The Uppaal verifier `verifyta` is executed on the model to synthesise a strategy, which provides an estimated utility that can be gained by taking each possible transition in the model. Uncontrollable factors, such as the movement of opponents, are represented as dotted lines (see figures 2 – 4). These transitions can influence the state of the model and the estimated reward of the controllable actions, but will not appear in the output strategy file since they do not correspond to actions that can be taken.

4.3.1 Pass-chain Model

Similarly to the real world, it is impossible for a player to kick the ball hard enough for it to cross the entire field and score a goal. Instead, the players must either dribble or pass the ball to each other multiple times, while avoiding interception by the opposing team, until they are close enough to the goal. The proposed pass-chain model is used by the online coach, which observes the game state and builds a model. The model generates a list of players, and the coach communicates to each player the next player to pass to via a *Say* message. The last player in the list is instructed to dribble the ball forward.

The model consists of a series of team players, instantiated from the template in figure 2, and the one-state model in figure 3 for measuring rewards. Opponents are represented as a list of coordinates. A player may either be in possession of the ball or free. From the possession state, the player may either choose to dribble or pass the ball to a teammate. In the latter

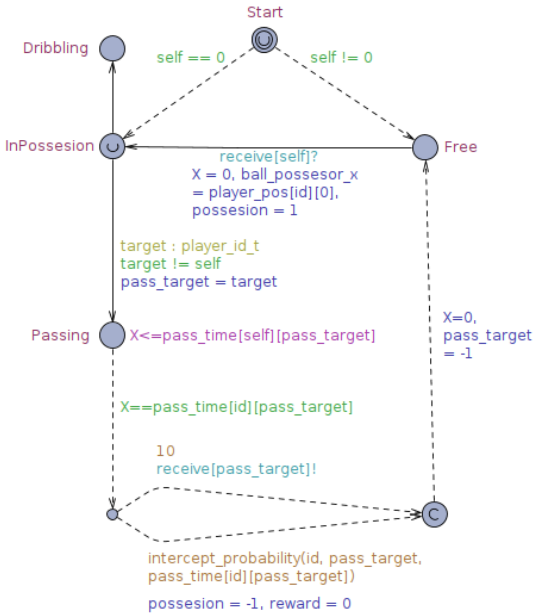

Figure 2: Player template of the pass-chain model.

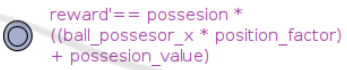

Figure 3: Ball/reward template of the pass-chain model.

case, there is a probability that the ball is lost to the opponent, which is estimated based on the positions of the pass target and of the opponents. The reward is modelled by means of a clock variable having a rate determined by how far the ball is currently positioned from the opponents' goal, and by which team is in possession of the ball. The query for generating the strategy tries to optimize the reward value over a period of 10 simulated seconds.

4.3.2 Possession Model

When a player is in possession of the ball, it must decide whether to pass the ball to a teammate or to dribble forward using a series of low power kicks. The possession model estimates the effectiveness of these choices. The model is highly dependant on a large number of uncontrollable and unpredictable variables such as the current positions of the opponents. Thus, it is infeasible to precompute offline strategies for all possible scenarios, which makes online strategy generation the only option for this model. On the other hand, generating the strategy as it is needed can involve computation time spanning one or more server ticks, which is sometimes enough for the opponents to take control of the ball. Thus, in several scenarios (e.g.: when chasing the ball, when attempting an interception, when the ball is moving towards

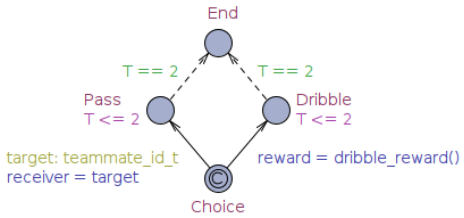

Figure 4: Possessor template from the possession model.

the player) players estimate periodically the time that they will come into contact with the ball, to generate a strategy a few ticks before the estimated contact time. The model is built using an extrapolation on the position of all visible moving objects.

The possessor template of the model is represented in figure 4. The generated strategy file contains rewards for performing passes to each of the visible teammates, and for performing a forward dribble. The rewards are based on the estimated probability that the action will succeed, as well as how much the ball will advance towards the opponent's goal.

4.3.3 Goalie Defence Model

The goalie is the last defence when the opposing team is close to scoring a goal. A good strategy for this situation aims to maximise the chance of a save by positioning the goalie in the best spot.

When an opposing striker approaches the goal, we assume that, if the striker shoots, it will be within the cone to the goal. Figure 5 represents the cone between *B1* and *B2*, and *G1* and *G2* are the shortest (perpendicular) segments from the goalie to the edge of the cone. The objective of the goalie is to keep the longest segment between *G1* and *G2* as short as possible, since if one of these were long, the goalie would take longer to intercept the ball if kicked along that trajectory.

Two timed automata for the goalie and the striker respectively are run in parallel. All transitions in the striker model are uncontrollable, since the striker is not under RoboPaal's control. The model for the striker is represented in figure 6. At each step both goalie and striker can choose to accelerate or decelerate in some direction. After this, the striker non-

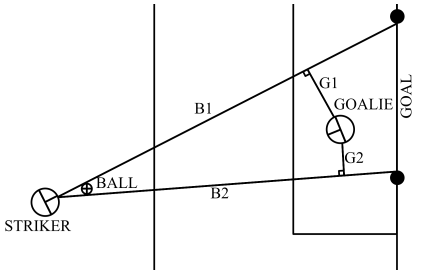

Figure 5: Goalie defender idea illustrated.

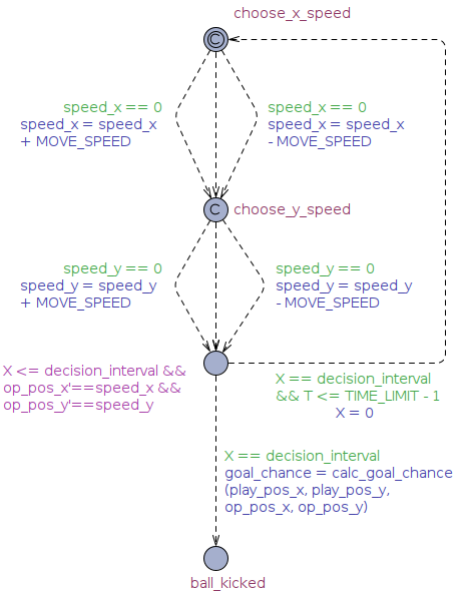

Figure 6: Striker part of the goalie model.

deterministically decides to either kick the ball or move further, and the goalie can respond by either moving or standing still. When the striker eventually kicks, the goal chance is calculated as a function of the goalie and the striker positions. The query used to synthesise the strategy aims to generate a strategy minimizing the goal chance.

Since the goalie needs fast reaction times and the state space is relatively small, this strategy was made as an offline strategy. The penalty area was divided into a grid of 1x1 meter squares, and a strategy was generated for each combination of goalie and striker positions within the grid. All the combinations were then saved in a dictionary, with the key being the positions of striker and goalie and the value being movement instructions for the goalie.

4.3.4 Stamina Model

If players dash as fast as they can throughout a game, they will run out of stamina quickly. As mentioned in Section 2.1, the player's stamina is a value between 0 and 8000 and it get lowered when dashing, and the effect of all player's actions is reduced when the stamina drops below 1000. We introduce a model for online stamina strategy generation. The strategy is generated every 11 seconds, or 110 ticks. The model uses stamina intervals 0 to 9 to reduce the number of possible states, and it searches for the maximal dash power that keeps the stamina level above 2000.

5 EXPERIMENTAL RESULTS

This section presents the experimental results we collected regarding the performance of RoboPaal. Every time the soccer server is quit, log files are generated. The log files can be used for analysing how teams performed in the game.

We performed two sets of experiments. The first set focused on specific scenarios, with each experiment repeated a number of times with different random seeds to reach statistical significance. In the second set of experiments, we let RoboPaal play against teams retrieved from public repositories.

We defined a number of metrics in order to measure the benefits provided by the strategies. The **possession** metric measures for how long the ball is kept, and it is used for the performance of the pass-chain model (section 4.3.1). The **field progress** metric measures the meters progressed toward the center of the goal of the opposing team without losing possession of the ball, and it is used to judge the performance of the pass-chain model. For the stamina model, we measured the metrics **minimum stamina** (the stamina of the team player with the minimum stamina level) and the **average stamina** (average stamina of all players in a team). The **goalie intervention** metric measures if the goalie is within 1.2 meters of the ball, i.e close enough to catch the ball, at any point during the attack, and it is used for to judge the goalie defender strategy.

5.1 Goalie Defender Strategy

For the goalie defender model (see section 4.3.3), we generated random striker attacks with the striker at a random position on the y-axis and at the edge of the penalty area on the x-axis. The ball is spawned right in front of the striker. The goalie is given 75 ticks to position according to the strategy, since in a real game the goalie would position itself gradually as the striker approaches the penalty area. Later on, both the goalie and the striker are allowed to proceed normally. The strategy is compared with: an **idle strategy** where the goalie positions itself in the middle between the two posts of the goal; a **following strategy** where the goalie follows the ball's y-coordinate.

Table 1 reports the results. We tested the Null Hypothesis 1, considering a significance level of 0.05. The binomial distribution is run with $n = 100$, expected probability 0.48, and actual number of successes 58. We got a p-value of 0.029, which allowed us to reject hypothesis 1 and to conclude that the defender strategy performs statistically better than the idle strategy.

Table 1: Successful goalie interventions out of 100 attacks.

	Successful interventions
Idle in front of goal	48
Goalie Defender Strategy	58
Manual Y-axis adjustment	60

Null Hypothesis 1. The goalie defender strategy does not perform better, than the idle strategy.

We then tested Null Hypothesis 2 using a binomial distribution with a significance level of 0.05. The test was run with $n = 100$, expected probability 0.60, and the actual number of successes set to 58. The results of the test did not allow us to reject hypothesis 2.

Null Hypothesis 2. The manual y-axis adjustment and goalie defender strategy perform identically given the same random attacks.

5.2 Stamina Strategy

The performance of the stamina model (see section 4.3.4) was tested by means of a beep test, and of full games.

The beep test is a real world fitness test for athletes, where the players have to run continuously back and forth between the two posts with less and less time to reach the next target (Wood, 2008). We measure the performance of our strategy against a baseline where a player simply dashes at full power all the time. Figure 7 shows that the average stamina at each tick is kept higher by our strategy, and figure 8 shows that the running performance of the stamina strategy overpowers the baseline since tick 2000. The fluctuations around tick 3000 are caused by the server replenishing the stamina of the players at half time. The decrease in stamina just before half time and end game is due to the player being allowed to use the rest of his stamina when the game is about to end.

We also tested the stamina strategy in 40 full games between two teams of identical implementations, except for one team using the stamina strategy.

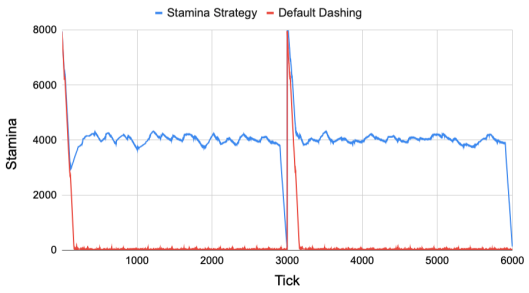

Figure 7: Stamina comparison of stamina strategy against dashing at full speed.

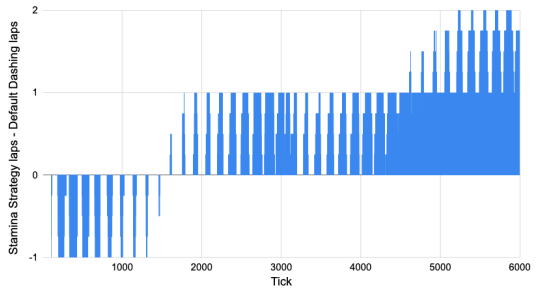

Figure 8: Lap comparison of stamina strategy against dashing at full speed.

The results can be seen in Table 2. We proposed the Null Hypothesis 3, and test it with a significance level of 0.05 against a binomial distribution, with $n = 40$, expected probability of 0.5 wins and actual number of successes 28. This yields a p-value of 0.008853, which is enough to confirm that the stamina strategy increases the chance of winning.

Null Hypothesis 3. The stamina strategy does not increase the chance of winning.

Figure 9 reports on the average stamina of the most tired player at every tick over the 40 games (the **average stamina** metric yields similar results), and it shows that players can preserve stamina while still winning more games. Another indicator that the stamina strategy can indeed make a difference is the fact that the team using it scores most of the goals in the latter half of each half time, which is when some players without the stamina strategy are out of stamina (see figure 9). An extreme example of this effect was noticed in one of the 40 games, where the stamina strategy team won 1-11 and only 1 of the 11 goals scored by the winning team fell within the first third of the half times.

Table 2: Game results of 40 games with one team using the stamina strategy.

Team	Goals Scored	Wins
Stamina Strategy	247	28
Default Dashing	160	11

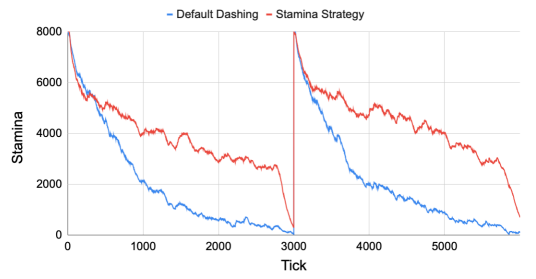

Figure 9: Average stamina of lowest stamina player in the 40 games.

5.3 Possession Strategy

Table 3 shows the cumulative results of 50 full games, where the team with the possession model performed significantly worse than the default implementation. The poor performance could be a result of considering backwards passes in the strategy, which in certain situations appears to be a good idea, but often results in losing the ball closer to a player's own goal.

Table 3: Results of 50 games with possession model against default.

Team	Goals Scored	Wins
Possession Strategy	167	11
Default	233	28

5.4 Pass-chain Strategy

The strategy was tested by setting up scenarios with one team using the strategy and one having default behaviour. 5 players were part of each team, and each scenario was run for 100 ticks. The metric used is field progress, calculated as an average on 200 runs per strategy. As seen in table 4 the pass-chain strategy has a better average field progress.

Table 4: Results of 200 games in average field progress.

Teams	Average Field Progress
Pass-Chain	17.81
Default	16.58

To test the results statistically, a Wilcoxon signed-rank test was used. We tested the data against the null hypothesis 4, with significance level 0.05. The outcome of the test was a p-value of 0.52218, which is not $p < 0.05$, thus we could not reject the null hypothesis.

Null Hypothesis 4. The pass-chain performance is identical to the default behaviour in terms of field progress.

5.5 Games against Other Implementations

Finally, we have tested RoboPaal against other implementations. Since the stamina strategy and goalie defender strategies appear to improve performance (see sections 5.1 and 5.2) we utilised them for our team, and we played using all the official rules.

The Keng implementation is available on GitHub (Keng, 2015), and it was developed as part of a course in machine learning at the Lafayette College

Table 5: Results of games against other implementations.

RoboPaal vs Keng	RoboPaal vs HfutEngine
19 - 0	0 - 34
21 - 0	0 - 34
21 - 0	0 - 36
16 - 0	0 - 36
15 - 0	0 - 35

in the United States. This implementation won the class tournament, and it employs a hybrid of reflex, utility and goal-based agents using swarm intelligence. Keng's players were developed manually and in the Python language, just like our ones. The games were very one sided and all ended with a win for our implementation, as seen in table 5.

We tested our implementation against the team that won the 2019 World Championship, namely the HfutEngine team (Lu et al., 2019). The players were developed using neural networks combined with the Helios Base player code (Zhiwei-Le et al., 2015), which is a heavily optimised C++ code library that takes care of most agent mechanisms and allows teams to focus on the AI part of the players. As seen in table 5 HfutEngine wins with a big lead against our RoboPaal implementation.

Both the comparisons have, however, limited validity since there are large differences in the ability of the players to analyse sensory input and perform maneuvers such as dribbles, intercepts and passes. The Keng implementation is created from scratch and the players turn and move significantly slower than the RoboPaal agents. Inversely, the HfutEngine players are much more adept than the RoboPaal agents. A more interesting comparison would involve future work aimed at developing players using our strategies over the Helios Base.

6 CONCLUSION

Generating strategies in real time for RoboCup players using timed automata modelled in Uppaal was proved to be possible. Challenges like time-sensitivity can be mitigated using game state forecasting combined with asynchronous computing, or offline strategy generation. In this work, we were able to implement both kinds of strategies and compared them in real games against existing RoboCup teams.

Future work considers to port our team to a better codebase to be able to compete with world champions. In a wider sense, we also plan to apply the two methods (state forecasting, and offline computation) to other problems.

ACKNOWLEDGEMENTS

This work was partly founded by the TECH faculty project "Digital Technologies for Industry 4.0", Aalborg University.

REFERENCES

Akiyama, H. (2010). Robo cup soccer simulation 2d league winning guide. ftp://ftp.iij.ad.jp/pub/sourceforge.jp/rctools/46021/RoboCup2DGuideBook-1.0.pdf.

David, A., Jensen, P. G., Larsen, K. G., Mikučionis, M., and Taankvist, J. H. (2015). Uppaal stratego. In Baier, C. and Tinelli, C., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *LNCS*, pages 206–211. Springer Berlin Heidelberg.

Eriksen, A. B., Huang, C., Kildebogaard, J., Lahrmann, H., Larsen, K. G., Muñiz, M., and Taankvist, J. H. (2017). Uppaal stratego for intelligent traffic lights. http://people.cs.aau.dk/~muniz/strategoTraffic.pdf.

Keng, W. L. (2015). Cs 420, artificial intelligence, robot soccer project, lafayette college department of computer science. https://github.com/kengz/robocup-soccer/blob/master/report/paper.pdf.

Larsen, K. G., Mikučionis, M., Muñiz, M., Srba, J., and Taankvist, J. H. (2016). Online and compositional learning of controllers with application to floor heating. http://people.cs.aau.dk/~muniz/LMMST-TACAS-16.pdf.

Lu, K., Ma, J., Cai, Z., Wang, H., and Fang, B. (2019). Hfutengine simulation 2d team. http://archive.robocup.info/Soccer/Simulation/2D/binaries/RoboCup/2019/PreliminaryRound/HfutEngine_SS2D_RC2019_R1_BIN.tar.gz.

Robocup Federation (2020a). A brief history of robocup. https://www.robocup.org/a_brief_history_of_robocup.

Robocup Federation (2020b). Robocup 2d simulation league. https://www.robocup.org/leagues/24.

RoboPaal team (2020). Repository for robopaal codebase. https://github.com/philipholler/RocoCup_Soccer_P6.

Rodrigues, H., Akiyama, H., Zare, N., and Obst, O. (2020). The robocup soccer simulator. https://github.com/rcsoccersim.

The RoboCup Simulator Committee (2020). The robocup soccer simulator documentation. https://rcsoccersim.github.io/manual/.

Uppaal (2019). About uppaal. http://www.uppaal.org/about.

Wood, R. (2008). Beep test variations and modifications. https://www.topendsports.com/testing/beep-variations.htm.

Zhiwei-Le, Keting-LU, Wang, G., Hao-Wang, and Baofu-Fang (2015). Hfutengine2015 simulation 2d team description paper. https://www.robocup2015.org/show/article/95.html.