Q-Learning based Intelligent Soccer Agent

Parth M. Desai, Master's Student – Spring 2015 – Missouri University of Science and Technology

***Abstract*—This Project aims at developing an intelligent soccer agent based on the Q-learning method of reinforcement learning. The project uses a Grid based Soccer Simulator to simulate the soccer environment. The agent is trained to score a goal when it has ball ownership and to locate and tackle the ball from the opponent when the opponent has the ball. The learning performance metrics of the agent's adaptation to soccer environment will be in terms of the number of games won verses the number of games played and the steps taken in playing each game. The Q-learning agent is successfully implemented and is capable of learning the soccer environment and the dynamic nature of the opponent's movement and hence score maximum goals possible.**

***Index Terms* — Q Learning, Temporal-Difference Learning, Reinforcement Learning, Machine Learning, AI, Grid Soccer, Games.**

I. Introduction

In this project, the problem of building an intelligent player in a soccer game capable of learning over time; how to navigate in the soccer field for two major activities: first, scoring a goal when player owns the ball, and second, tackle the ball when opponent is the ball owner by tracing the opponent is undertaken. The use of Reinforcement Learning (RL) method of Q-learning to develop the learning algorithm for the soccer agent is selected to approach this problem.

Reinforcement learning (RL) is learning from experiences, meaning learning from the consequences of actions taken, rather than from explicit training. The action selection is based on its past experiences and by exploring new choices, similar to the natural learning process of humans i.e. through repeated task execution and improving reactions each time the task is executed. The basic constructs of RL Systems include: Agent, Environment, Policy, Rewards and Value Functions (Delayed Rewards). While defining the reinforcement learning problem, it is required to define the boundaries within which the agent can operate. This involves defining agent's learning using Active (Off-Policy) or Passive (On-Policy) learning, whether the algorithm follows Model based or Model Free design and also defining the environment Exploration / Exploitation policy for allowing the agent to learn over time.

II. Background

Lot of work has been done to build intelligent agents in soccer game. Some of the earlier work done in this direction involves using discrete and continuous time based Q-learning to achieve different tasks for a soccer agent.

In [1], Sawa T. et. al., uses the 2D soccer game environment and proposes a technique of implementing fuzzy Q learning algorithm for behavior acquisition of an agent (player) in the soccer field. In [2], Nakashima T. et. al. implements a Fuzzy Logic based Reinforcement learning algorithm to train a soccer agent to intercept a ball in an continuous state-action space. The learning algorithm is a fuzzy based Q-learning algorithm wherein the agent receives rewards for approaching the ball as well as catching the ball. In [3], Riedmiller M. et. al. discuss the use of Neural Fitted Q Iteration (NFQI) algorithm (NN based RL algorithm) for the task of dribbling (i.e. holding possession of the ball and moving towards the goal). Another implementation of Neural Netowrk based Q-learning agent is described in [4] where Shi-chao Wang et. al. implements the Q-learning algorithm using the conventional Back-Propagation based Neural Network with the advantage of learning continuous state and action spaces with greater efficiency. Various other intelligent soccer agent implementations have been done using methods other than Q-learning such as in [5] which uses Multi-Agent MDP's that are designed in order to co-ordinate the team agents' movement in the field ,[6] discusses the co-operative grid mapping strategy to utilize the field of view of team players to avoid redundant exploration. In addition to these methods, simple Q-Learning applications for robot navigation can be used and modified for the soccer tasks.

III. Q-Learning

For the proposed problem in this project, Q-Learning algorithm is used to implement the agent learning. Q-Learning is a type of Reinforcement learning technique that learns an optimal policy no matter which policy the agent is actually following (i.e., which action 'a' it selects for any given current state 's' of the agent) as long as each possible action is taken at least once in each state. It is a type of temporal difference based model free learning algorithm which evaluates the history of environmental reactions of every action taken at each time step. Over time the algorithm converges and optimizes the action-value function such that it maximizes the Q-Function.

Q-learning involves maintaining a Q-value table for each state action combination. For an Agent P consisting of n states: S_1, S_2,…S_n and m actions: A_1, A_2,… A_m which receives an immediate reward: $r(S_i, A_j)$. for taking action A_j in state S_i. The algorithm uses the next state in order to maximize the cumulative reward. Assuming the next state as S_k, the Q-value of the current state is updated using the relation:

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
    $s \leftarrow s'$;
  until s is terminal

Figure 1: Classical Q-Learning Algorithm.

$$\boldsymbol{Q_{new-curr}(S_i, A_j) \leftarrow Q_{curr}(S_i, A_j) + \alpha.[\, r(S_{i\rightarrow k}, A_j) + \gamma.\underbrace{max}_{A_l}.Q(S_k, A_l) - Qcurr(S_i, A_j)\,]} \quad (1)$$

where "$\mathbf{Q_{new\_curr}(S_i, A_j)}$" denotes Updated (New) Q–value of current state–action pair, "$\mathbf{Q_{curr}\ (S_i, A_j)}$" denotes the Q–value of current state–action pair, "$\boldsymbol{\alpha}$" is the learning rate, "$\mathbf{r(S_{i\rightarrow k}, A_j)}$" is the reward for taking action '$\mathbf{A_j}$' in state '$\mathbf{S_i}$' to go to next state '$\mathbf{S_k}$', "$\boldsymbol{\gamma}$" is the Discount Factor and "$\underbrace{max}_{A_l}\mathbf{.Q(S_k, A_l)}$" denotes Q value of the best state–action pair at the Next state. The standard algorithm used for implementing the Q-learning algorithm is shown in figure 1. For additional details the reader can refer [7] and [8] which gives a detailed explanation of Q-learning and its usage in RL applications.

IV. APPROACH

In this project, a Multi-agent Grid Soccer Simulator (GSS) is used to simulate the soccer environment. The simulator acts as the server which handles and monitors all the playing agent's activities. The source code can be compiled using Visual Basic 2010. For this project, the Q-Learning Soccer Agent is built using MATLAB and it communicates to the GSS server using Java interface. The details of the Simulator can be found in [9].

The intelligence for the Q-learning agent is built by focusing the learning of the agent for two major tasks. First, the agent should learn to successfully dribble the ball and score a goal. A successful dribble is defined as holding the ownership of the ball and reaching the goal. Second, when the opponent has ball ownership, the agent is also required to learn tracing (locating) the opponent, in order to tackle the ball. In order to successfully perform these tasks the logic for agent's learning is divided in to multiple sections.

1. Scanning Environment and Agent information
2. State Determination
3. The Exploration / Exploitation policy
4. Reward Function Definition
5. Q Value Updates
6. History Record keeping and
7. Data Plotting

The GSS source code comes with a built in communication framework between an Agent developed using MATLAB and the GSS server with a set of classes defined for each of the Agent, Environment and Actions. All information about the Grid World is communicated from the server using these classes. This project is focused on building an Agent in MATLAB to learn its action selection strategy using the information available from these class components.

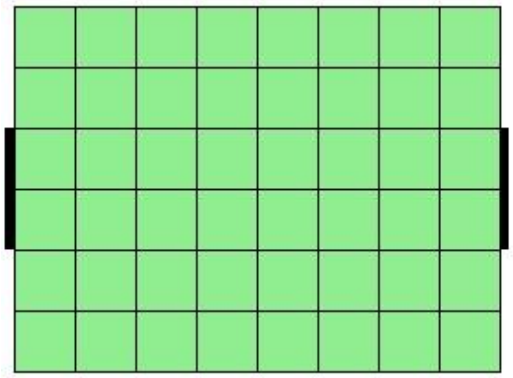

Figure 2: GSS Grid World for Training Q-learning Agent.

The States used for Q-value calculation are the Ball Ownership and the relative distances of the player and the opponent. Here, the relative distances are advised to reduce the size of the Q-table by removing any redundancy in storing the state information. Redundancy in the state information can be described in the sense that for a player at position (2, 2) and opponent at position (4, 4), movement selection criteria will be same as in case the player position (4, 4) and opponent position (6, 6). Thus, it is more efficient to store the state information by considering the opponents position relative to the players position.

The origin of the x-y plane is linked to the player when determining the relative distance to its target. When the Player owns the ball, the relative distance is measured with respect to the goal position. Similarly, when the player is trying to trace the opponent, its relative distance is measured with respect to opponent's position. Since the goal position is not dynamic and remains fixed, it is much easier to trace and can be learned faster as compared to learning to trace the opponent. The other state information is the ball ownership which differentiates the two targets, the player can follow, the goal post or the opponent.

Once the state information is available, it is now required to define the action to be selected at the current state. For this project, the ***ε-greedy*** policy is implemented to solve the exploration / exploitation dilemma. The ***ε-greedy*** policy is an action selection policy followed at every state such that with a probability of $\boldsymbol{\varepsilon}$ the agent takes random actions (among East, South, West and North) and with a probability of $\boldsymbol{(1-\varepsilon)}$ the agent takes actions based on Maximum Q-value of the state-action pair at the current state.

After defining the states and actions, the reward assignment for an action taken at a particular state needs to be identified. In this project, instead of storing rewards in the reward table, a reward function was implemented which decides how the rewards will be assigned for any action the agent takes within the environment. This allows memory savings to some extent as the reward table is largely dependent on the size of the Q-table.

The reward assignment is divided into 3 different strategies:
Goal Directed Rewards: Agent receives 100 reward for scoring a goal, -1 reward for going out of field bounds and 0 rewards for any movement within the field, with ball ownership. This is shown in figure 3.

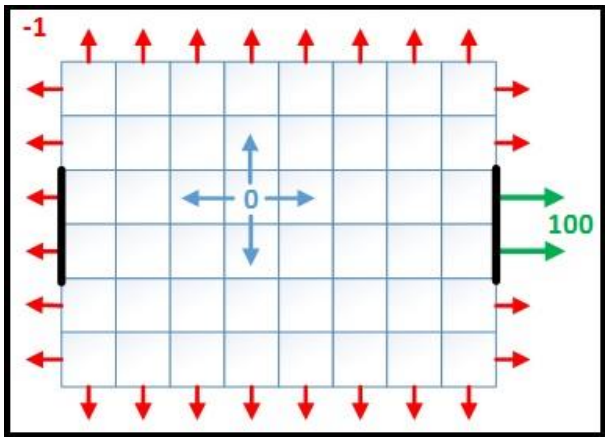

Figure 3: Goal Directed Reward Assignment.

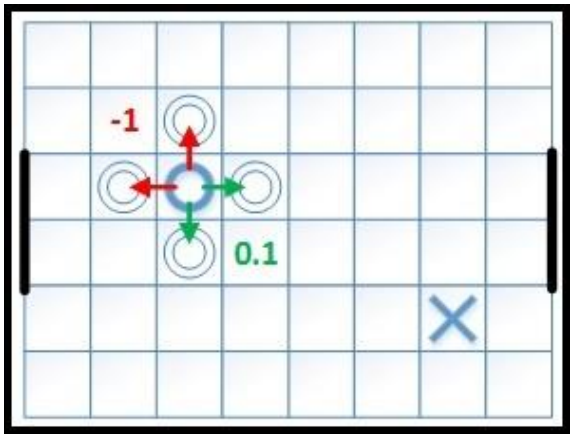

Figure 4: Chase Directed Reward Assignment.

Chase Directed Rewards: In this case, the ball is owned by the opponent. The Manhattan distance [10] between player and opponent position is calculated. If this distance reduces as compared to the previous state, the agent receives reward of 0.1 otherwise the agent receives a reward of -1. This is shown in figure 4.

Ball Ownership Rewards: For successfully tackling the ball from opponent and retaking ownership of the ball there is 0 reward. However, if the agent loses ownership of the ball, it receives a reward of -1 in addition to any other rewards.

Different reward values and some additional conditions were defined to experiment the change in learning performance of the Q-learning agent as shown in Table 1. The Reward Strategy 3 was used for the final implementation.

The Q-value for the current state-action pair is calculated based on the rewards assigned as per Reward Strategy 3 and using the Equation 1 above. Different values of learning rate and discount factor were experimented.

Once, the Q-values are updated in current state, before entering the next iteration, the current states, actions, score, and etc. information are stored for usage in the Q-value calculation of the next state. This is because the learning algorithm is such that, the Q-value of last state is updated in the current iteration. Finally, the data required to plot the performance of the agent is recorded in terms of No. of Games Won and Lost, Steps taken per game and the Games Played to Games Won Ratio.

	Reward Strategy 1	**Reward Strategy 2**	**Reward Strategy 3**
Score Goal	100	5	**100**
Move Out of Field	-1	-0.01	**-1**
Move Inside Field	0	-0.01	**0**
Not Owning Ball	-1	---	**---**
Move Closer to Opponent	0.1	0.005	**0.1**
Moving Away from Opponent	---	-0.01	**-1**
Tackle (From Front, Top or Bottom)	0.2	---	**---**
Tackle (From Behind)	-1	---	**---**
Stealing Ball	0.2	---	**---**
Losing Ball	-2	Reward - 0.02	**Reward - 1**

Table 1: Reward Assignment Strategies.

V. Results and Discussion

Multiple experiments were performed while building the agent. These experiments involved first verifying successful goal scoring when Q-learning agent has ball ownership, followed by locating and chasing the opponent owning the ball to tackle and re-taking ownership of the ball. Finally, a combined code was developed for the agent to perform both actions whenever required.

In order to obtain sufficient information from the simulator, for building the intelligence of the Q-learning agent, the algorithm uses the following components from each MATLAB class:

Agent: Game Step (Cycle), Agent's Position (Row, Column), Opponent's Position (Row, Column), Current Ball Owner and Current Score. **Environment:** Field Size, Goal limits (Upper and Lower Row Limit). **Action:** Number of Actions – 4 (East, South, West, North)

Also, the GSS configuration settings used for this project are: VisibleRadius: 0, NumRows: 6, NumCols: 8 and GoalWidth: 2. All other settings are unaltered. This allows a soccer field size to be limited to a 6 x 8 grid world with the goal of size 2. The game grid is as shown in figure 2.

For implementing the ***ε-greedy*** policy, the initial value of ε is taken to be 0.999 and is decremented by a factor of 0.001 after a fixed number of steps taken by the player irrespective of episode number. This decrement in ε is continued until ε = 0.1 beyond which the value of ε is reduced by 10% every fixed number of steps.

The values of learning rate and discount factor were finalized after observing the effects of changing them to different values [11] (Other values tried mentioned in brackets).

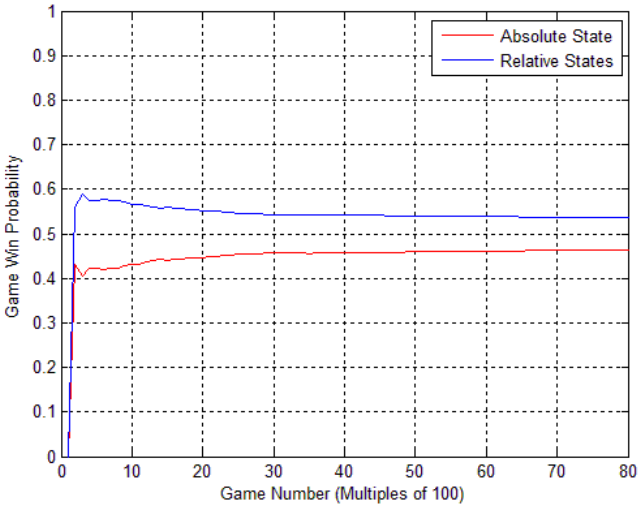

Figure 5: Q-Learning Agent's Learning Performance comparison for training using Absolute States and Relative States.

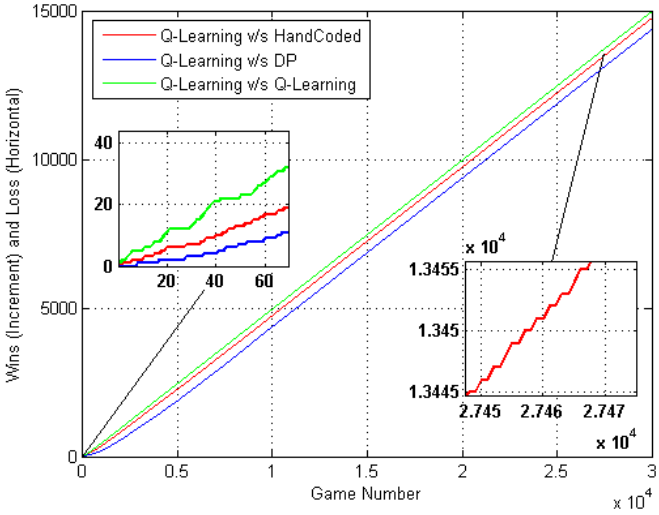

Figure 6: Q-Learning Agent's Wins and Losses.

Learning Rate: 0.05 (0.7, 0.1, 0.3)
Discount factor: 0.3 (0.8, 0.1)

The values were finalized based on two major criteria: convergence speed and difference between learning accuracy plotted in each case. Different reward assignments strategies experimented are summarized in table 1 shown above. Reward Strategy 3 was finalized for providing better performance in comparison to other strategies.

The learning performance of the MATLAB Q-learning Agent built for this project was compared three different opponents. First Opponent is the MATLAB agent itself, which verifies the agent's learning performance against an opponent of similar intelligence. The other two opponents are some pre-programmed agents available with the simulator. These agents are named as Hand-Coded Agent (HC Agent) (Programmed using a simple algorithm to go towards the goal irrespective of the Q-learning agent's position when it has ball ownership, otherwise to chase the Q-learning agent and tackle) and a Dynamic Programming Agent (DP Agent) trained using Policy Iteration. The performance metrics used for comparison were:

1. Number of games won / lost against number of games played.
2. Number of Steps taken per game.
3. Game Win Probability (per every 100 games Played).

Experiment 1: Training the Q-learning agent to only score a Goal when it has ball ownership based on conditions defined by Goal directed Rewards. No specific action defined to chase the opponent. States (Agent's position (Row and Column)). The experiment was successful in making the agent learn to score a goal when it is assigned the ball at the beginning of a new game.

Experiment 2: Training the Q-learning agent to score a Goal when it has ball ownership as well as chase opponent to tackle if it does not own the ball using chase directed rewards strategy. States considered are (Opponent Position (Row and Column) and Agent's Position (Row and Column)).

Results were unsuccessful at first, as the number of states were very large. It was also observed that the Q-learning agent was unable to trace the opponent even after allowing sufficient game runs, in majority of the trials. In addition, the reward assignment was not training correctly and Q-table updates were flawed. One major observation in the method for chase directed reward assignment was the effect of using the Euclidean Distance for calculating the increase / decrease in distance between player and opponent. Since, here a grid based environment is used, the Euclidean Distance would not help as it is suitable for continuous environments. Thus, it was decided to use the Manhattan distance to determine the distance from the opponent so that it can be easily interpreted by this world. However, the performance of the Player did not improve much and the training time was still large.

Experiment 3: After obtaining results from experiment 2, the states were required to be redefined to better represent the agent and opponent behavior. The new states were defined considering the relative distance of the opponent's position from the Q-learning agent's current position. As explained previously, this eliminates the training of redundant states. A comparison was made to verify the performance improvement achieved by considering relative distance between Q-learning agent and opponent as system states instead of the absolute position values of Q-learning agent and opponent as system states. Figure 5 confirms that the win ratio is comparatively very large for the agent trained using relative position states and also the learning speed is faster.

The selection of opponent's relative position with respect to Q-learning agent's position and the reward assignment strategy 3 resulted in better performance in learning in terms of learning speed and maximizing game wins as compared to other options experimented. The Q-learning agent was then tested with three different opponents. Figure 6 shows the Games Won / Lost with each opponent. As highlighted in the figure, initially there are lot of losses recorded for the Q-learning agent (denoted by horizontal lines), after learning is complete, the number of incremental ramps increase, indicating more wins recorded for the Q-learning agent. Table 2 summarizes the games won / lost recorded at fixed intervals. It can be observed that after 25000 games, the Q-learning agent either maintains or reduces the games win-loss difference.

Figure 7 shows the steps taken per game. As the training progresses, it takes relatively less number of steps to score a goal. Thus, Q-learning agent's learning converges to take the

Total Games Played	Games Won			Difference between Wins & Losses		
	QL Agent	HC Agent	DP Agent	QL Agent	HC Agent	DP Agent
1000	453	334	209	90	382	582
2500	1196	1002	762	108	496	976
5000	2448	2205	1879	104	590	1242
10000	4949	4682	4307	102	636	1386
25000	12450	12200	11790	100	600	1420
50000	24950	24730	24290	100	540	1420
100000	49950	49770	49290	100	460	1420

Table 2: Game Wins for different Opponents.

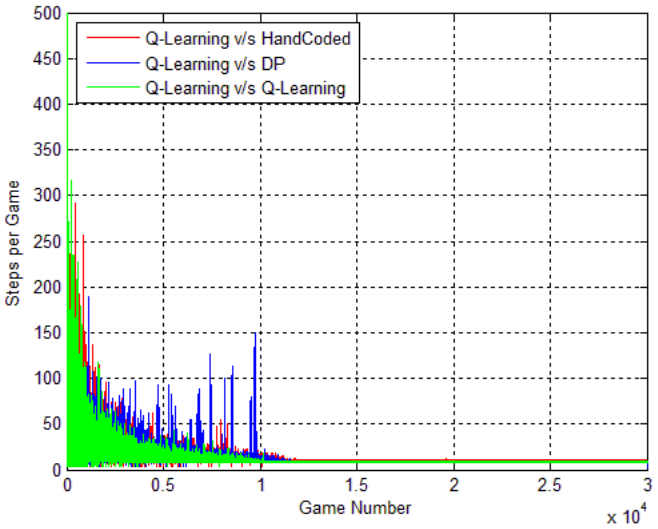

Figure 7: Q-Learning Agent: Number of Steps per Game.

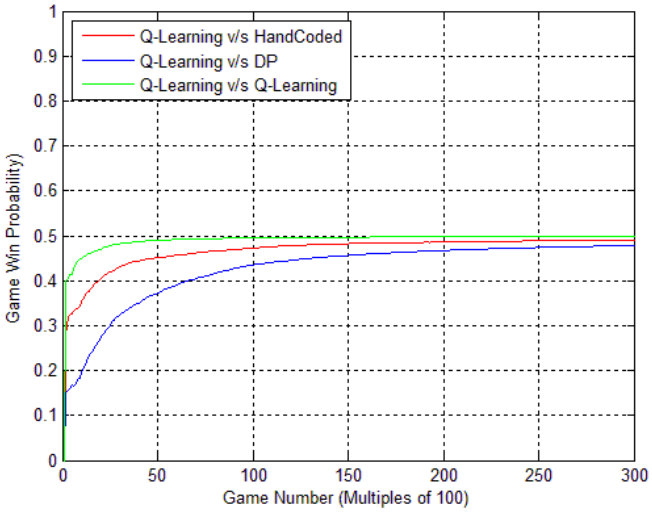

Figure 8: Q-Learning Agent: Probability of Game Wins (Measured per 100 Games)

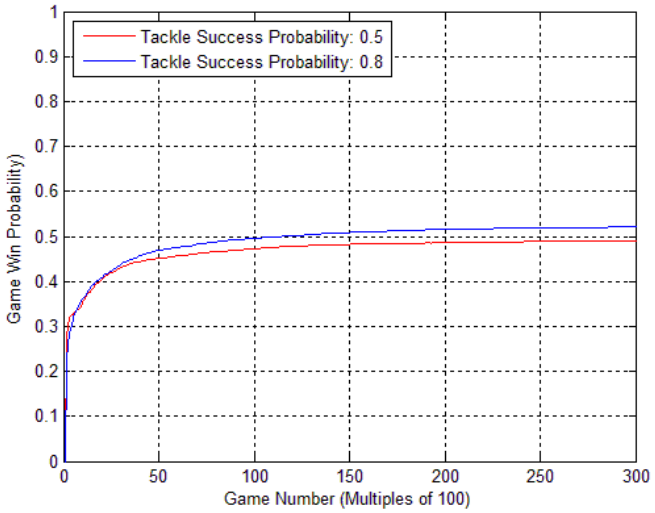

Figure 9: Performance comparison for different Tackle Success probability rate.

best possible action which results in increase in the number of Game wins. Figure 8 shows the Probability of number of Games won compared to the number of games played recorded every 100 games played. The curve shows that if more games are allowed to be played, the Player will have probability of greater than 0.5 to win a game which is the main objective of this project i.e. to learn to win as many games as possible.

During experimentation is was also observed that the GSS Server determines the next possible ball owner in an event of a conflict (or tackle) situation. The GSS server code is by default are programmed with a probability of 0.5 that the ball owner will not change and only a probability of 0.5 that the ball owner changes. This is important as it largely affects the performance of the learning agent in the sense that after learning even-though the Q-learning agent has learned to place itself for a conflict (tackle) situation, the success of the tackle is dependent on the probability defined by the server.

All previous learning was performed when the tackle success probability was set to 0.5 by GSS Source code. To check the change in learning performance, the server code was modified in order to change the conflict success probability rate from the default values to a higher success rate i.e. in an event of a conflict, the ball owner will change with probability of 0.8 and will remain the same with probability of 0.2. The Game win probability curve for comparing the results of different tackle success rates is shown in figure 9.

VI. Conclusion

The objective of the project to train soccer player to score maximum wins as fast as possible in a Grid based world using Q-learning was successfully achieved using MATLAB. Experiments proved that the definition of states and rewards assignment largely affects the learning performance of the Q-learning Agent. Here, it was observed the states represented in terms of relative position between two players, has increased the learning convergence rate as well as saved memory by

avoiding storage of redundant state information. Memory saving was also achieved by implementing a reward function rather than maintaining a reward table. Different experiments, proved that when playing matches with different opponent types, the Q-learning Agent learns to increase or maintain a constant game win probability rate.

REFERENCES

[1] Sawa, T.; Watanabe, T., "Learning of keepaway task for RoboCup soccer agent based on Fuzzy Q-Learning," *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on* , vol., no., pp.250,256, 9-12 Oct. 2011

[2] Nakashima, T.; Udo, M.; Ishibuchi, H., "Knowledge acquisition for a soccer agent by fuzzy reinforcement learning," *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, vol.5, no., pp.4256, 4261 vol.5, 5-8 Oct. 2003

[3] Riedmiller, M.; Hafner, R.; Lange, S.; Lauer, M., "Learning to dribble on a real robot by success and failure," *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, vol., no., pp.2207, 2208, 19-23 May 2008

[4] Shi-chao Wang; Zheng-xi Song; Hao Ding; Hao-bin Shi, "An Improved Reinforcement Q-Learning Method with BP Neural Networks in Robot Soccer," Computational Intelligence and Design (ISCID), 2011 Fourth International Symposium on , vol.1, no., pp.177,180, 28-30 Oct. 2011

[5] Sari, S.C.; Kuspriyanto; Prihatmanto, A.S., "Joint action optimation for robotic soccer multiagent using reinforcement learning method," *System Engineering and Technology (ICSET), 2012 International Conference on* , vol., no., pp.1,7, 11-12 Sept. 2012

[6] Foroughi, B.; Safdari, R.; Kashkoee, R.; Salehi, M.E., "Action selection using cooperative occupancy grid map for humanoid soccer robot," *Electrical Engineering (ICEE), 2014 22nd Iranian Conference on*, vol., no., pp.1052, 1057, 20-22 May 2014

[7] Christopher J.C.H. Watkins and Peter Dayan, "Technical Note: Q-Learning", Machine Learning, Vol. 8, No. 3. (1 May 1992), pp. 279-292.

[8] Sutton, Richard S., and Andrew G. Barto. "Temporal-Difference Learning" *Reinforcement Learning an Introduction.* Cambridge, Mass.: MIT, 1998. Print.

[9] Iravanian S., Araghi S. "Grid-Soccer Simulator". Internet: http://gridsoccer.codeplex.com/, Jul. 21, 2012 [Mar. 25, 2015].

[10] Wikipedia. "Taxicab Geometry". Internet: http://en.wikipedia.org/wiki/Taxicab_geometry, Apr. 08, 2015 [Apr. 15, 2015]

[11] Wilson, D. R., & Martinez, T. R. "The need for small learning rates on large problems". Proceedings of the 2001 International Joint Conference on Neural Networks (IJCNN'01), pp. 115 – 119, 2001.

[12] Prokhorov, D.V.; Wunsch, D.C., "Adaptive Critic Designs," Neural Networks, IEEE Transactions on , vol.8, no.5, pp.997,1007, Sep 1997.

[13] F. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," IEEE Computational Intelligence Magazine, vol.4, no.2, pp.39-47, May 2009.

[14] Suton, Richard S., and Andrew G. Barto. "Monte Carlo Methods" *Reinforcement Learning an Introduction.* Cambridge, Mass.: MIT, 1998. Print.

[15] D.Tamilselvi, S. Mercy Shalinie, and G. Nirmala, "Q Learning for Mobile Robot Navigation in Indoor Environment", IEEE-International Conference on Recent Trends in Information Technology, ICRTIT, 978-1-4577-0590-8, pp 324-329, 2011.

[16] Mhammad, J.; Bucak, O., "An improved Q-learning algorithm for an autonomous mobile robot navigation problem," Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on , vol., no., pp.239,243, 9-11 May 2013

[17] Knar, A.; Chakraborty, I.G.; Singh, S.J.; Jain, L.C.; Nagar, A.K., "A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot," Systems, Man, and Cybernetics: Systems, IEEE Transactions on , vol.43, no.5, pp.1141,1153, Sept. 2013