Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2016, Article ID 7090757, 14 pages
http://dx.doi.org/10.1155/2016/7090757

Research Article

Enhancing Video Games Policy Based on Least-Squares Continuous Action Policy Iteration: Case Study on StarCraft Brood War and Glest RTS Games and the 8 Queens Board Game

Shahenda Sarhan, Mohamed Abu ElSoud, and Hebatullah Rashed

Computer Science Department, Faculty of Computers and Information, Mansoura University, Mansoura 35516, Egypt

Correspondence should be addressed to Shahenda Sarhan; shahenda_sarhan@yahoo.com

Received 7 August 2015; Revised 18 November 2015; Accepted 1 December 2015

Academic Editor: Manuel M. Oliveira

With the rapid advent of video games recently and the increasing numbers of players and gamers, only a tough game with high policy, actions, and tactics survives. How the game responds to opponent actions is the key issue of popular games. Many algorithms were proposed to solve this problem such as Least-Squares Policy Iteration (LSPI) and State-Action-Reward-State-Action (SARSA) but they mainly depend on discrete actions, while agents in such a setting have to learn from the consequences of their continuous actions, in order to maximize the total reward over time. So in this paper we proposed a new algorithm based on LSPI called Least-Squares Continuous Action Policy Iteration (LSCAPI). The LSCAPI was implemented and tested on three different games: one board game, the 8 Queens, and two real-time strategy (RTS) games, StarCraft Brood War and Glest. The LSCAPI evaluation proved superiority over LSPI in time, policy learning ability, and effectiveness.

1. Introduction

An agent is anything that can be viewed as perceiving its environment through sensors and acting in that environment through actuators as in Figure 1, while a rational agent is the one that does the right thing [1].

However, agents may have no prior knowledge on what the right or optimal actions are. To learn the best action selection the agent needs to explore the state-action search space and, from the rewards provided by the environment, the agent can calculate the true expected reward when selecting an action from a state.

Reinforcement learning (RL) is learning what the agent can do and how to map situations to actions in order to maximize the numerical reward signal. Reinforcement learning assists agents to discover which actions yield the most reward and the most punishment after trying them through trial-and-error and delayed reward. Reinforcement learning concentrated more on finding a balance between exploration of anonymous areas and exploitation of its current knowledge [2–4].

Batch reinforcement learning (BRL) is a subfield of dynamic programming (DP) [4, 5] based reinforcement learning that recently has immensely grown. Batch RL is mainly used, where the complete amount of learning experience, usually a set of transitions sampled from the system, is fixed and given a priori. The learning system concern then is to derive an optimal policy out of the given batch of samples [6–8]. So batch reinforcement learning algorithms aim to achieve the utmost data efficiency through saving experience data to make an aggregate batch of updates to the learned policy [7, 8].

Figure 2 classifies different batch RL algorithms based on interaction perspectives into offline and online algorithms. The offline algorithms are also known as pure batch algorithms that mainly work offline on a fixed set of transition samples as Fitted Q Iteration (FQI) [9, 10], Kernel-Based Approximate Dynamic Programming (KADP) [11, 12], and

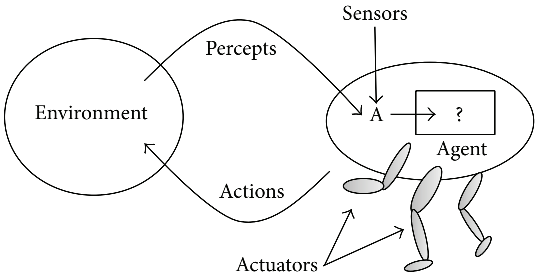


Figure 1: Agent-environment interaction [18].

Least-Squares Policy Iteration (LSPI) [13–15], while online algorithms comprised pure online algorithms, semibatch algorithms, and growing batch algorithms as Neural Fitted Q Iteration (NFQ) [16] that depends on making an aggregate update for several transitions and storing and reusing the experiences after making this update.

Different problems had been solved using BRL based on the RL agents' ability to learn without expert supervision. Game playing problem is one of major areas where BRL represented a magical solution for determining the best policy to be applied in a game. This often depends on a number of different factors, as the space of possible actions is too large for an agent to reason about directly, while still meeting real-time constraints. To cover this many states, using a standard rule based approach would mean specifying a large number of hard coded rules. BRL cuts out this need to manually specify rules, as agents learn simply by playing the game. For example, in backgammon game, agent can be trained by playing against other human player or even other RL agent [7].

As we mentioned obtaining the optimal policy [17] (decision-making function which represents a mapping from states to actions) the game can apply is a major key in evaluating game performance through game-human interaction. The problem arises here as all games especially games with large space of possible actions as real-time strategy (RTS) games and board games suffer through finding the optimal policy and thus the best action. As almost all algorithms count only for discrete states, discrete actions, or continuous states, discrete actions while agents are in such a setting have to learn from the consequences of their continuous actions, in order to maximize the total reward over time. So the researchers in this paper proposed an algorithm based on LSPI considering continuous actions, which we called Least-Squares Continuous Actions Policy Iteration (LSCAPI). The LSCAPI was applied and tested using three game genres: StarCraft Brood War, Glest, and 8 Queens.

This paper is organized as follows. Section 2 covers a brief review of video games concentrating on RTS games and board games. In Section 3, the proposed Least-Squares Continuous Actions Policy Iteration (LSCAPI) algorithm is described in detail. Section 4 comprises the testing of LSCAPI on some real cases. In Section 5 the simulation results and discussion are introduced and in Section 6 the implantation of LSCAPI. Finally Section 7 outlines our conclusions.

2. Background

2.1. Board Games. Board games are the most known and mostly played games over centuries. A board game is a game that involves pieces moved or placed on a board, according to a set of rules. Moves in board games may depend on pure strategy or only pure chance as the rolling dice, or both; in all cases expert opponents can achieve their aims [20].

Earlier board games were represented as a battle between two armies with no rules except points, while modern board games basically relied on defeating opponent players in terms of counters, winning position, or points entitlement thus relying on rules. Many board [21] games as Tic-Tac-Toe [22], chess, and Chinese chess (checker) had a long history in machine learning researches. Trinh et al. in [23] discussed the application of temporal-difference-learning in training a neural network to play a scaled-down version of Chinese chess.

Runarsson and Lucas in [24] studied and likened the temporal difference learning (TDL) using the self-play gradient-descent method and coevolutionary learning, using an evolution strategy for acquiring position evaluation for small Go boards. The two approaches are compared with the hope of gaining a greater insight into the problem of searching for optimal strategies.

Wiering et al. in [25] used reinforcement learning algorithms that can learn a game position evaluation function through learning the backgammon game. They examine three different methods for training games: learning by self-play, learning by playing against an expert program, and learning from viewing experts play against themselves.

And Block et al. in [26] proposed a chess engine which proved that reinforcement learning in combination with the classification of board state leads to a notable improvement, when compared with other engines that only use reinforcement learning, such as Knight-Cap.

Skoulakis and Lagoudakis in [27] demonstrated the efficiency of the LSPI agent over the TD agent in the classical board game of Othello/Reversi. They presented a learning approach based on LSPI algorithm that focuses on learning a state-action evaluation function. The key advantage of the proposed approach is that the agent can make batch updates to the evaluation function with any collection of samples, can utilize samples from past games, and can make updates that do not depend on the current evaluation function since there is no bootstrapping.

Finally Szubert and Jaskowski in [28] employed three variants of temporal difference learning to acquire action value, state value, and after-state value functions for evaluating player moves through puzzle game 2048. To represent these functions they adopt n-tuple networks, which have recently been successfully applied to Othello and Connect 4 board games.

2.1.1. 8 Queens. The 8 Queens puzzle is a context of the N-Queens problem where eight chess queens are placed on an 8 × 8 chessboard [29]. Any of the queens must not attack any of the others, so that no two queens share the same row, column, or diagonal as in Figure 3.

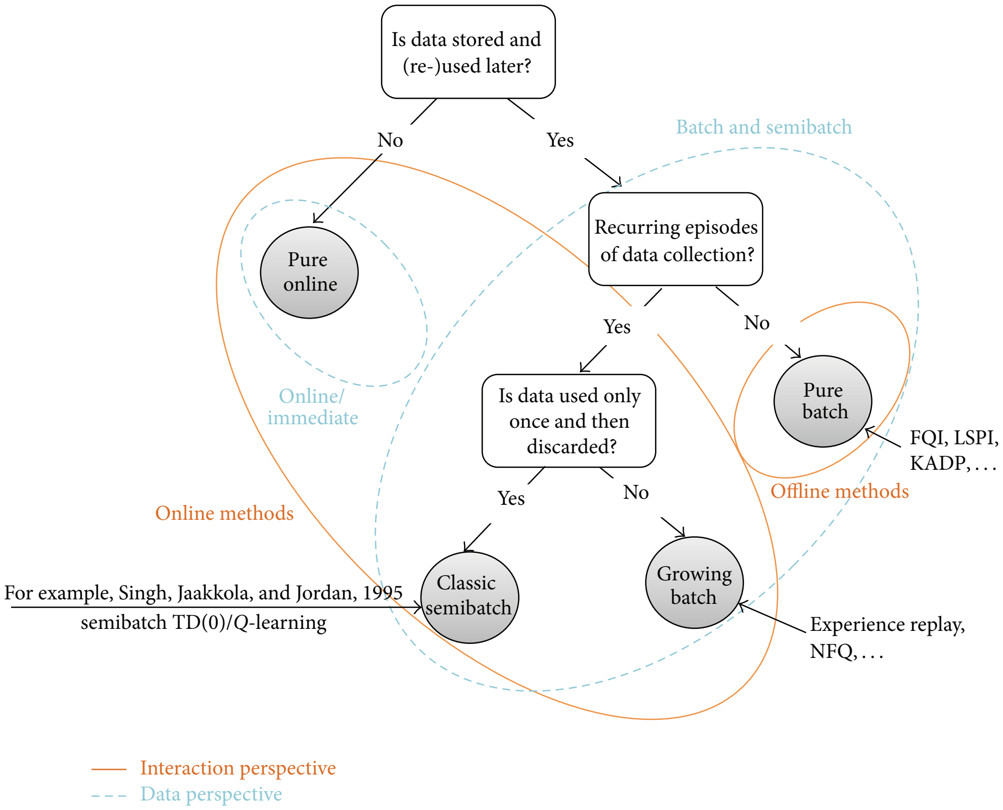

Is data stored and (re-)used later?
No
Yes
Batch and semibatch
Pure online
Recurring episodes of data collection?
Yes
No
Online/ immediate
Is data used only once and then discarded?
Pure batch
FQI, LSPI, KADP, . . .
Offline methods
Yes
No
Online methods
For example, Singh, Jaakkola, and Jordan, 1995 semibatch TD(0)/Q-learning
Classic semibatch
Growing batch
Experience replay, NFQ, . . .
Interaction perspective
Data perspective

Figure 2: Classification of batch versus nonbatch algorithms [7].

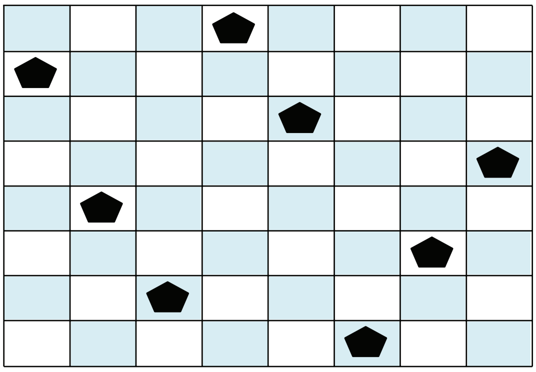


Figure 3: 8 Queens [19].

Many algorithms had been proposed for solving 8 Queens as in Lim and Son [30], who applied the *Q*-learning as a problem solving algorithm for the N-Queens and compared it with the traditional existing methods for solving N-Queens problem.

2.2. Real-Time Strategy Games. Real-time Strategy (RTS) games are a subgenre of strategy games where players need to build an economy and military power in order to defeat their opponents. In RTS games players race and struggle against enemy factions by harvesting resources scattered over a terrain and producing buildings and units and help one another in order to set up economies, improve their technological skill and level, and win battles, until their enemies are extinct [31–34]. The better the balance you get among economy, technology, and army, the more the chances you have to win RTS games [34].

Marthi et al. in [35] applied hierarchical reinforcement learning in a limited RTS domain. This approach used reinforcement learning augmented with prior knowledge about the high-level structure of behavior, constraining the possibilities of the learning agent and thus greatly reducing the search space.

Gusmao in [36] considered the problem of effective and automated decision-making in modern real-time strategy games through the use of reinforcement learning techniques. The researcher proposed a stable, model-based Monte Carlo method assuming that models are imperfect, reducing their influence in the decision-making process. And its effectiveness is further improved by including a novel online search procedure in the control policy.

Leece and Jhala in [37] presented a simplified game that mimics spatial reasoning aspects of more complex games, while removing other complexities through analyzing the effectiveness of classical reinforcement learning for spatial management in order to build a detailed evaluative standard across a broad set of opponent strategies. The authors also demonstrated the potentiality of knowledge transfer to more complex games with similar components.

In 2015, Sethy et al. [38] proposed reinforcement learning algorithm based on *Q*-learning and SARSA with the generalized reward function to train agents. The proposed model

Figure 4: Glest RTS game.

Figure 5: StarCraft Brood War RTS game.

was evaluated using Battle City RTS game proving superiority over the state of the art in enhancing agent learning ability.

2.2.1. Glest. Glest is an open source 3D real-time strategy game, where the player can have armies of two different factions, tech, and magic. Tech mainly included warriors and mechanical devices, while magic relied on mages and summoned creatures in the battlefield [34, 39]. In Glest warriors fight by weapons and mangonel, while magicians cast spells and magic forces as shown in Figure 4. The final goal for both factions is to extinct all enemy units, finish the episode, or die.

Glest actions are structured in three levels. The first consists of primitive low-level actions, such as modifications of basic variables that describe low-level microstates. In the second level of actions, grouped primitive actions form a more abstract task or rule. There are thirteen different rules in Glest as Worker Harvest, Return Base, Massive Attack, Add Tasks, Produce Resource Producer, Build One Farm, Produce, Build, Upgrade, and Repair [33, 39, 40].

The third level represents a strategy tactic level, grouping second-level actions-rules with respect to the timer of the game. Each second-level action-rule is associated with a certain time interval. Depending on the module of the timer a certain rule will be picked up and applied by the tactic of the game within the rule's interval [40].

Glest has four main variables: Kills, Units, Resources, and Timer. Each state is characterized by the values of the four variables, where "Kills" refers to the number of kills that the player has achieved, "Units" counts the units the player has produced, "Resources" are the resources the player had harvested, and finally "Timer" counts the game time units.

Every variable is connected with some related actions except for "Timer"; its only role is to count time. "Units" is connected with Build One Farm, Produce, Build, Upgrade, Expand, and Repair actions, while "Kills" is connected with Scout Patrol, Return Base, Massive Attack, and Add Tasks. Finally "Resources" are connected with Worker Harvest, Refresh Harvester, and Produce Resource Producer.

Glest recently attracts the attention of different researchers as in 2009 Dimitriadis [33] investigated the design of reinforcement learning autonomous agents that learn to play Glest RTS game through interaction with the game itself. He used the well-known SARSA and LSPI algorithms to learn how to choose among different high-level strategies with the purpose of winning against the embedded AI opponent, while Aref et al. in 2012 proposed a new model called (REAL-SEE) for exchanging opponent experiences among real-time strategy game engines of Glest [34].

2.2.2. StarCraft Brood War. StarCraft Brood War shown in Figure 5 is an expansion pack released in 1998 for the award-winning real-time strategy game StarCraft. Brood War gameplay remains fundamentally unchanged from that of StarCraft but with new difficult campaigns, map tilesets, music, extra units for each race, upgraded advancements, and less practical rushing tactics where missions are no longer entirely linear [41, 42]. Through the Brood War a single agent can make high-level strategic decisions as attacking an opponent or creating a secondary base and mid-level decisions as deciding what buildings to build [42, 43].

Brood War has gained popularity as a test bed for research as it exhibits all of the issues concerning most interested AI research areas in RTS environments including pathfinding, planning, spatial and temporal reasoning, and opponent modeling. Also Brood War has a huge number of online game replays that have been used as the case base for case-based reasoning (CBR) systems to analyze opponent strategies [42, 43].

In 2012, Wender and Watson introduced an evaluation of the suitability of Q-learning and SARSA reinforcement learning algorithms to perform the task of micromanaging combat units in the StarCraft Brood War RTS game [44], while in 2014 Siebra and Neto proposed a modeling approach for the use of SARSA in enabling the computational agents evolving their combat behavior according to actions of opponents to obtain better results in later battles [45].

3. Least-Squares Continuous Actions Policy Iteration (LSCAPI)

Reinforcement learning and video games have a long beneficial conjoint history as games are fruitful fields for testing reinforcement learning algorithms. In any application of reinforcement learning, the choice of algorithm is just one of many factories that determined success or failure. The choice of the algorithm is not even the most significant factor. The choice of representation, formalization, the encoding of domain knowledge, and setting of parameters can all have great influence.

```
**Input:** discount factor $\gamma$
$\phi_i: X \rightarrow R, i = 1, \ldots, N, \psi_j: U, j = 0, 1, 2, \ldots, M_P$
$l \leftarrow 0$, initialize policy $h_0$
measure initial state $x_0$
**for** step $k = 0, 1, 2, \ldots$ **do**
$u_k = \phi^T(x, \overline{u})\theta_l, \psi = 1$; a uniform random action in $U, \psi = -1$
apply $u_k$, measure state $x_{k+1}$, and reward $r_{k+1}$
start LSTD-Q policy evaluation
    $\Gamma_0 \longleftarrow 0, \Lambda_0 \longleftarrow 0, z_0 \longleftarrow 0$
$\Gamma_{k+1} \leftarrow \Gamma_k + \phi(x_k, u_k)\phi^T(x_k, u_k)$
$\Lambda_{k+1} \leftarrow \Lambda_k + \phi(x_k, u_k)\phi^T(x_{k+1}, h(x_{k+1}))$
$z_{k+1} \leftarrow z_k + \phi(x_k, u_k)r_{k+1}$
finalize policy evaluation
$\frac{1}{k+1}\Gamma_{k+1}\theta_l = \gamma\frac{1}{k+1}\Lambda_{k+1}\theta_l + \frac{1}{k+1}z_{k+1}$
policy improvement
$h_{l+1}(x) \longleftarrow u_L + (u_H - u_L)\frac{1 + \text{argmax}_{\overline{u}}\phi^T(x, \overline{u})\theta_l}{2}\forall x$
**until** $h_{l+1}$ is a satisfactory
$l \leftarrow l + 1$
**end for**
**Output:** $\widehat{h^*} = h_{l+1}$
```

Algorithm 1: Offline Least-Squares Continuous Actions Policy Iteration.

In this research the researchers proposed an offline pure batch algorithm based on the Least-Squares Policy Iteration algorithm. Least-Squares Policy Iteration is a relatively new, model-free, approximate policy iteration algorithm for control. It is an offline, off-policy batch training method that exhibits good sample efficiency and offers stability of approximation [13–15].

Least-Squares Policy Iteration evaluates policies using the least-squares temporal difference for Q-functions (LSTD-Q) and performs exact policy improvements. To find the Q-function of the current policy, it uses a batch of transition samples, with LSTD-Q, to compute the parameter vector. Then, an improved policy in this Q-function is determined, to find the Q-function of the improved policy and so on [5]. Most releases of LSPI use discrete actions although for control problems; continuous actions are needed. As systems need to be stabilized, any discrete action may cause unneeded chattering of the control action.

The idea of LSCAPI as described in Algorithm 1 concentrates on solving discrete action problem through comparing actions to get the largest Q-value action to be applied. But first scalar control actions U are considered to deal with the actions as a continuous action chain where ($U = [u_L, u_H]$).

A new parameter for evaluating actions other than state parameters is proposed called orthogonal polynomials Ψ. Ψ is evaluated by two values $\{-1, 1\}$ considering if the action is selected for applying in the game through the current step or not. Fitted action is evaluated by 1 where the engine will return the true value that was used to compute the policy. The other value is -1 where the action is discarded and a new action is selected from the action space U.

As soon as the fitted action is selected the LSTD-Q [13] function evaluates the policy by creating parameter θ from the input transition using the selected fitted action.

Finally the difference between the previous and the new selected actions is calculated and stored with the new action as in (1). This step optimizes the needed storage space for actions by only storing the difference between actions not the actions themselves:

$$\begin{aligned}\widehat{Q}(x, u) &= \sum_{j=0}^{M_P} \psi_j(\overline{u}) \sum_{i=1}^{N} \overline{\phi}_i(x)\,\theta_{[i,j+1]},\\ \widehat{Q}(x, u) &= \phi^T(x, \overline{u})\,\theta.\end{aligned} \tag{1}$$

4. Testing LSCAPI on Real Cases

For more explanation the LSCAPI algorithm was tested on 8 Queens, Glest, and StarCraft Brood War games and implemented using Microsoft visual studio 2013 C# engine.

4.1. 8 Queens

4.1. 8 Queens. Through this section the proposed algorithm will be used for solving the 8 Queens problem and tracing each step separately. But first there are some assumptions to clarify:

(i) X is the states in the game which are eight states (eight rows).

(ii) U is the actions which are continuous as every state has a chain of actions for it.

(iii) $N = 8$ considering 8×8 board and $M_P = 64$ which represents the probability of actions that the 1st queen can be taken on the board.

(i) 1st Queen. As clarified in Figure 6, the first queen will be set randomly on the first row of the board, where the initial state $\phi_1 = s_0 = 1$ indicates that s_0 is in row 8 and in column

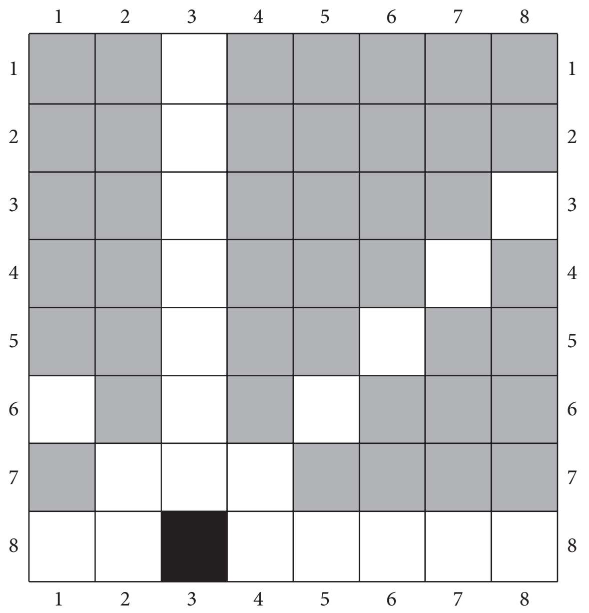


Figure 6: 1st queen.

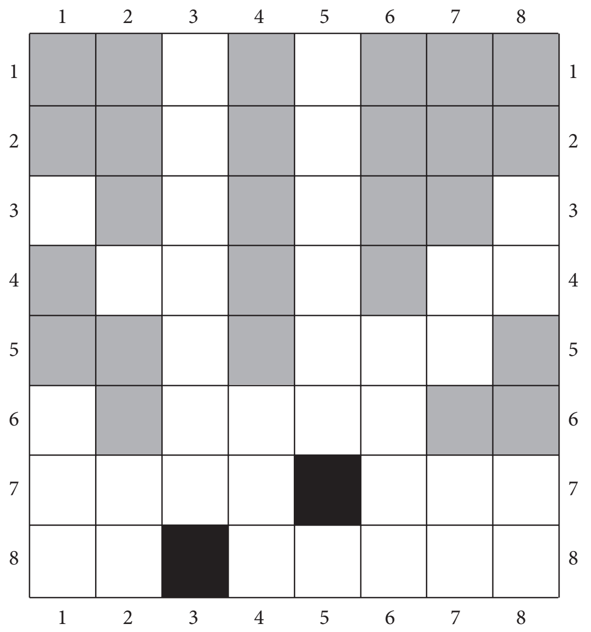


Figure 7: 2nd queen.

3 ($j = 3$). The probability [row, column] of the 1st queen is $(8, 3)$, which indicates that the action for placing the 1st queen in column 3 is taken.

(ii) 2nd Queen. After placing the 1st queen, the probability of actions for the second queen is $M_P = 42$, eliminating the probabilities that other queens could be on the same row, column, or diagonals. In the current state a matrix of continuous actions will be initiated holding the five available probabilities of actions $[(7, 1); (7, 5); (7, 6); (7, 7); (7, 8)]$ that the second queen could take in the next row.

Through applying LSCAIP, the value of ψ will be set to -1 if action $(7, 1)$ is selected as it increases the probability of having pairs of attacking queens and set to 1 if the action $(7, 5)$ is selected as it provides more possible actions to take. Based on LSCAPI the action with the higher Q-function $(7, 5)$ will be selected. State $\phi_2 = s_1 = 2$ means that s_1 is in row 7 and in column 5 ($j = 5$). So the action with the largest Q-function was to place the queen in position $(7, 5)$ as shown in Figure 7.

(iii) 3rd Queen. After placing the 2nd queen, we will move to the next state, the third row, having 26 probabilities of actions to place the 3rd queen on board and only 3 probabilities of actions in the continuous actions matrix $[(6, 2); (6, 7); (6, 8)]$. The previous steps will be repeated from state 2 to select the action with the higher Q-function indicating the action generating more possible positions on the board to place the 4th queen. As in Figure 8, the 3rd queen will be placed on position $(6, 2)$, where state $\phi_3 = s_2 = 3$ means that s_2 is in row 6 and column 2.

The same is done in the next five states to get the policy of the game and solve the problem by placing the eight queens in their true places without errors as shown in Figures 9, 10, 11, 12, and 13.

Table 1 describes some of the other solutions generated by LSPI and LSCAPI for the 8 Queens game.

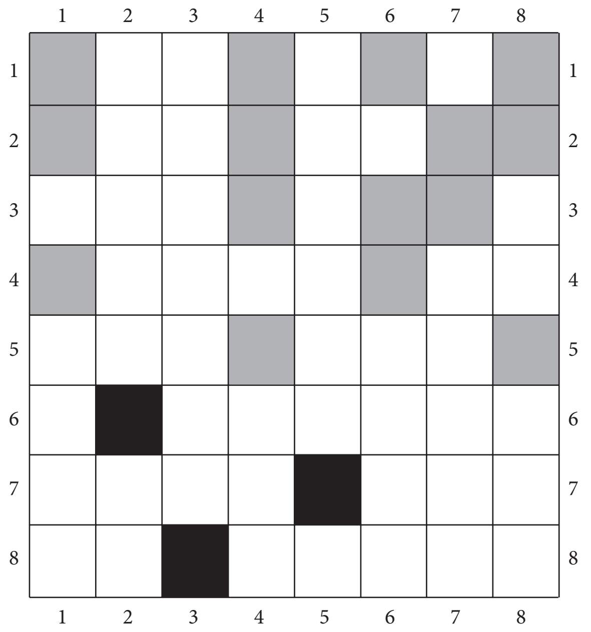


Figure 8: 3rd queen.

The performance of any action in the chain is measured based on the number of nonattacking queens. The minimum is equal to zero where all queens attack each other. The maximum is 28 where there are no attacking queens. The performance is calculated based on (2) to determine the fitness of the action compared to alternative actions:

$$P_A = \frac{\text{No. of Non-attacking pairs}}{\text{No. of pairs}}. \quad (2)$$

Table 2 represents the performance evaluation of actions taken by LSCAPI in case 1 $\{6, 3, 7, 2, 8, 5, 1, 4\}$. Every row has

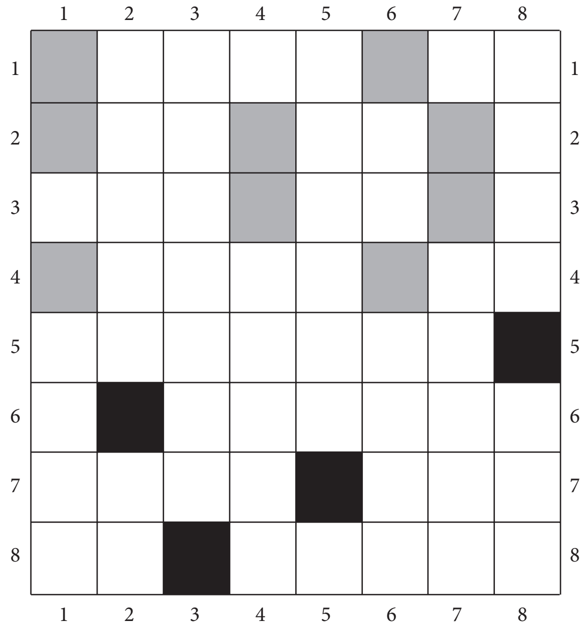


Figure 9: 4th queen.

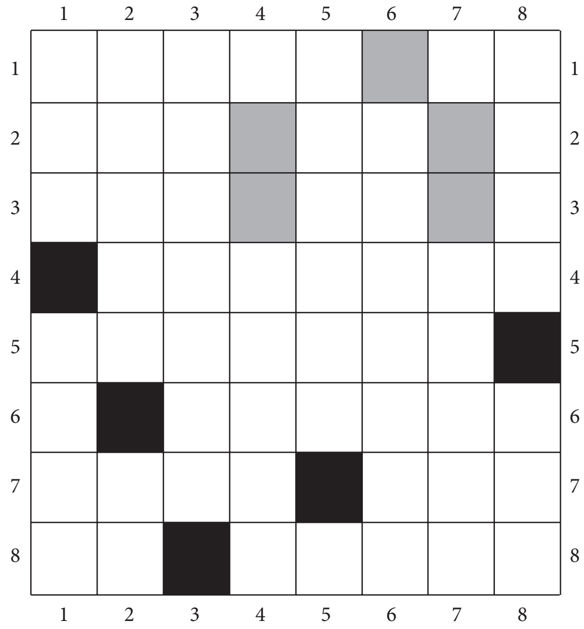


Figure 10: 5th queen.

chain of fitted actions evaluated by (2). Notice that if two fitted actions have the same values, the fitted one is chosen depending on the previous chosen actions. Table 3 represents the performance evaluation of discrete actions taken by LSPI.

It is noticed that the case performance achieved by LSCAPI is higher than the case performance achieved by LSPI from which we can also determine that actions selected by LSCAPI are much better than LSPI that finally produce better game performance.

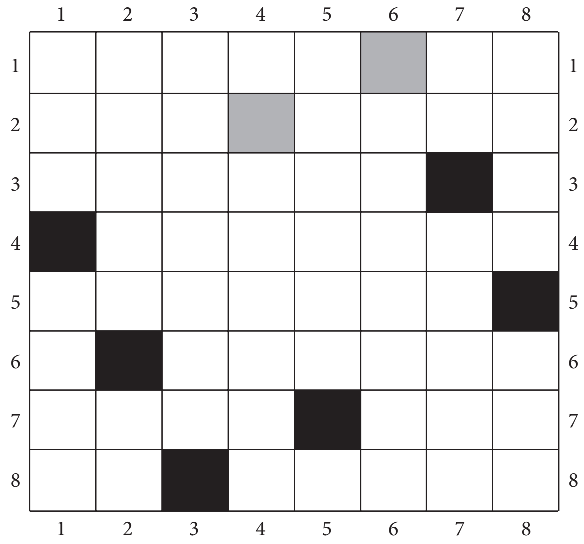


Figure 11: 6th queen.

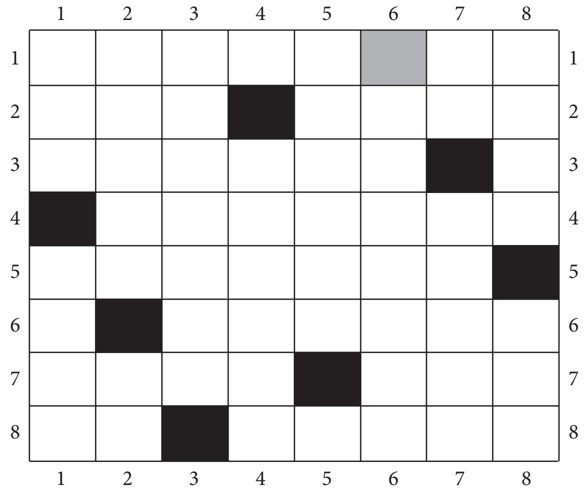


Figure 12: 7th queen.

Table 1: Problem solution using LSPI versus LSCAPI.

Cases	Problem solution							
	1	2	3	4	5	6	7	8
				LSCAPI				
Case 1	5	1	8	6	3	7	2	4
Case 2	6	3	7	2	8	5	1	4
Case 3	5	7	2	6	3	1	8	4
Case 4	4	2	5	8	6	1	3	7
Case 5	5	3	1	6	8	2	4	7
				LSPI				
Case 1	2	7	5	8	1	4	6	3
Case 2	5	2	8	1	4	7	3	6
Case 3	6	3	1	8	4	2	7	5
Case 4	7	4	2	5	8	1	3	6
Case 5	1	7	5	8	2	4	6	3

Table 2: LSCAPI case (1).

State (row)	Action performance inside the chain (column)							
8	0	0	0	**4**	0	0	0	0
7	**1/21**	0	0	0	0	0	0	0
6	0	0	0	0	**1/21**	0	0	0
5	0	0	0	0/21	0	0	0	**2/21**
4	0	**3/21**	0	3/21	0	0	0	0
3	3/21	0	0	0	4/21	0	**5/21**	0
2	3/21	5/21	**6/21**	2/21	0	0	0	6/21
1	0	0	0	0	0	**6**	0	0
	1	2	3	4	5	6	7	8
Total performance = 18/21 = 0.857								
Problem	6	4	5	2	1	5	6	4
Solution	6	3	7	2	8	5	1	4

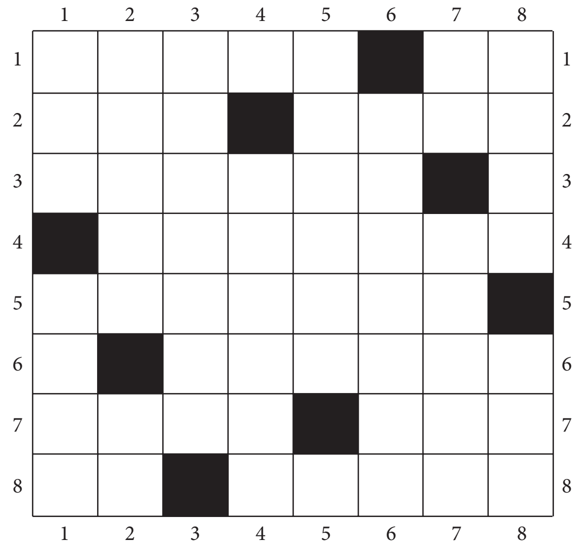


Figure 13: 8th queen.

Table 3: LSPI case (1).

State (row)	Action performance inside the chain (column)							
8	0	0	0	0	0	**6**	0	0
7	0	0	**0/23**	0	0	0	0	0
6	**1/23**	0	0	0	0	1/23	0	0
5	0	0	0	0	0	0	0	**2/23**
4	0	0	0	0	**4/23**	0	0	3/23
3	4/23	**4/23**	0	0	0	2/23	0	3/23
2	3/23	5/23	3/23	**6/23**	3/23	0	0	0
1	0	0	0	0	0	0	**7**	0
	1	2	3	4	5	6	7	8
Total performance = 17/23 = 0.739								
Problem	7	8	2	5	8	1	6	3
Solution	7	4	2	5	8	1	3	6

Table 4: LSCAPI three towers case.

Game						
Available actions	AT	B	MA	P	PRP	WH
Actions priority						
State 1	PRP	MA	B	AT	P	0
	Kills = 250 *Units* = 200 *Resources* = 1100					
State 2	PRP	MA	B	WH	0	0
	Kills = 900 *Units* = 850 *Resources* = 3000					
Player						
Available actions	AT	B	MA	P	PRP	WH
Actions priority						
State 1	WH	PRP	B	MA	AT	P
	Kills = 150 *Units* = 200 *Resources* = 400					
State 2	PRP	WH	MA	B	P	0
	Kills = 300 *Units* = 300 *Resources* = 800					

4.2. Glest. The evaluation of LSCAPI in Glest is a bit different than in 8 Queens as in Glest the LSCAPI works on improving the game engine policy against the player. The researchers in evaluating LSCAPI in Glest concentrated on three parameters $\Delta kills$, $\Delta Units$, and $\Delta resources$ as in Table 4, where $\Delta kills$, $\Delta Units$, and $\Delta resources$ represent the gained or lost knights, units, and resources as a result of moving from state S_i to S_{i+1} (executing an action).

In LSCAPI algorithm we will get action for a state not as discrete but as continuous chain of actions. Every action in a chain has the value of polynomial 1 or −1 not as a calculation but only for selecting the fittest action in this state. Using continuous chain will improve efficiency of the game and speed it up. The LSCAPI arranges the most fitted action then the next action and so on based on priority. It saves time to select the fitted action and enhance performance. Fitted actions are evaluated by the most reward resulting from them.

In Table 4 a sample case of Glest game called three towers was introduced. LSCAPI arranges the available actions in a priority continuous chain. This method helps to get the actions with the higher performance in less time.

In this case the available actions were Add Tasks (AT), Build (B), Massive Attack (MA), Produce (P), Produce Resource Producer (PRP), and Worker Harvest (WH). These actions are arranged based on their priority; the action with a high priority will be in the beginning of the chain and so on until the only fitted actions that will be used in state are complete.

In "game" state 1 the priority of continuous actions chain is (PRP, MA, B, AT, P); we use only five actions from the six input actions in this case with the shown reward. In state 2 we use only four actions of them, while in player the priority chain of actions in state 1 includes all of the input actions and in state 2 it includes only five of them.

Meanwhile in the tower of souls case as in Table 5 the available actions were Produce Resource Producer, Build One Farm, Refresh Harvest, Massive Attack, Return Base, and Upgrade.

In game state 1 the priority of continuous actions chain is (MA, PRP, RH, RB, and BOF); we use only five actions from the six input actions in this case with the shown reward. In

Table 5: LSCAPI tower of souls case.

			Game			
Available actions	PRP	BOF	RH	MA	RB	Up
			Actions priority			
State 1	PRP	MA	BOF	Up	RB	0
	Kills = 350 *Units* = 300 *Resources* = 1700					
State 2	MA	PRP	Up	RB	0	0
	Kills = 1300 *Units* = 950 *Resources* = 5000					
			Player			
Available actions	PRP	BOF	RH	MA	RB	Up
			Actions priority			
State 1	MA	PRP	RH	RB	BOF	0
	Kills = 200 *Units* = 150 *Resources* = 900					
State 2	PRP	RB	BOF	MA	Up	0
	Kills = 500 *Units* = 400 *Resources* = 2300					

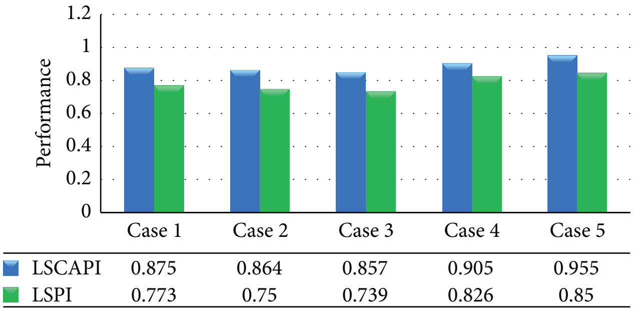


Figure 14: LSPI versus LSCAPI generated solution cases performance.

state 2 we use only four actions of them, while in player the priority chain of actions, as in the table, in state 1 includes all of the input actions and in state 2 it includes only five of them. Finally the calculations of the reward function to evaluate the action selection mechanism are based on

$$\text{reward} = \Delta\,(\text{kills}) + \Delta\,(\text{units}) + \Delta\,(\text{resources}) \quad [33]\,, \tag{3}$$

where

$$\begin{aligned}
\Delta\,(\text{kills}) = {} & [\text{kills}\ \ (\text{agent new state}) \\
& - \text{kills}\ \ (\text{agent old state}) \\
& - \text{kills}\ \ (\text{enemy new state}) \\
& - \text{kills}\ \ (\text{enemy old state})] \times 100 \quad [33]\,, \\
\Delta\,(\text{units}) = {} & [\text{units produced}\ \ (\text{agent new state}) \\
& - \text{units produced}\ \ (\text{agent old state}) \\
& - \text{units produced}\ \ (\text{enemy new state}) \\
& - \text{units produced}\ \ (\text{enemy old state})] \times 50 \quad [33]\,, \\
\Delta\,(\text{resources}) = {} & [\text{resources}\ \ (\text{agent new state}) \\
& - \text{resources}\ \ (\text{agent old state}) \\
& - \text{resources}\ \ (\text{enemy new state}) \\
& - \text{resources}\ \ (\text{enemy old state})] \quad [33]\,.
\end{aligned} \tag{4}$$

Table 6 demonstrates an example for calculating reward function, based on the values obtained from the number of kills, units, and resources in each two consecutive states as in (3). The reward values support the superiority of the LSCAPI over LSPI with a reasonable difference.

4.3. StarCraft Brood War. The StarCraft Brood War applied the same case form as Glest, so we will just concentrate on the calculations of the reward function. The reward function of Brood War is representing the difference between the damage done by the agent and the damage received by the agent in each two consecutive states, to evaluate the action selection mechanism as in

$$\begin{aligned}
\text{Reward}_{t+1} = {} & \sum_{i=1}^{m} \Big(\big(\text{enemy\_unit\_health}_{i_t} \\
& - \text{enemy\_unit\_health}_{i_{t+1}} \big) - \big(\text{agent\_unit\_health}_{i_t} \\
& - \text{agent\_unit\_health}_{i_{t+1}} \big) \Big) \quad [42]\,.
\end{aligned} \tag{5}$$

Table 7 demonstrates an example for calculating the reward of agent in case of applying the LSPI and the SARSA (already applied in the StarCraft Brood War engine) action selection mechanisms, through the Protoss versus Zerg Case. This case included four ground units, Probe, Dragon, Drone, and Larva, considering that Drone and Larva units belong to Zerg which represents the game agent, while Probe and Dragon units belong to Protoss which represents the game enemy.

We can notice that the reward values in the Brood War are within the range of [0, 1] which is much smaller in range than Glest. Also from the table it is cleared that the reward of the agent resulting from applying actions supported by LSCAPI is much higher than the agent reward in case of SARSA.

5. LSCAPI Implementation

The LSCAPI was implemented using Microsoft visual studio 2013 C# and packaged to the Glest and StarCraft Brood War games engines while the 8 Queens was a fully standup application based on LSPI and LSCAPI.

6. Results and Discussion

6.1. 8 Queens. The overall performance of 5 different solutions generated by the LSCAPI and LSPI is calculated through the implementation and shown in Figure 14 from which we can detect that the LSCAPI generated actions and solutions that achieved better performance against opponents than LSPI which also indicates better policy through the game.

Meanwhile in Figure 15 the time taken by each one of these solutions through LSPI and LSCAPI is presented. It is

Table 6: Three towers case.

	LSCAPI				
Evaluation metrics	State 1 (old)		State 2 (new)		Reward
	Game (agent)	Player (enemy)	Game (agent)	Player (enemy)	
Kills	250	150	900	300	20,000
Units	200	200	850	300	7,500
Resources	1100	400	3000	800	70
		Total reward			*Reward* = 27,570
	LSPI				
Evaluation metrics	State 1 (Old)		State 2 (New)		Reward
	Game (agent)	Player (enemy)	Game (agent)	Player (enemy)	
Kills	300	100	700	120	18,000
Units	200	152	660	240	3,400
Resources	2500	200	4000	300	100
		Total reward			*Reward* = 21,500

Table 7: Protoss versus Zerg Case.

	LSCAPI				
Evaluation metrics	State 1 (old)		State 2 (new)		Reward
	Zerg (agent)	Protoss (enemy)	Zerg (agent)	Protoss (enemy)	
Drone units health	0.85		0.6		0.2
Larva units health	0.95		0.8		
Probe units health		0.85		0.65	
Dragon units health		0.9		0.5	
	SARSA				
Evaluation metrics	State 1 (old)		State 2 (new)		Reward
	Zerg (agent)	Protoss (enemy)	Zerg (agent)	Protoss (enemy)	
Drone units health	0.65		0.5		0.08
Larva units health	0.65		0.6		
Probe units health		0.9		0.85	
Dragon units health		0.85		0.58	

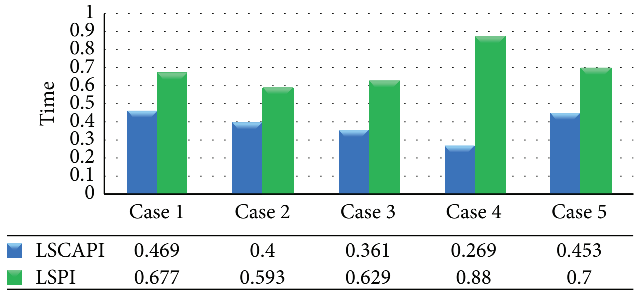


Figure 15: LSPI versus LSCAPI solution generating time.

noticed from Figure 15 that LSCAPI not only generated policies that achieved a higher performance, but also generated them in less time as in case 4 solution of LSCAPI where the performance reaches 0.905 in 0.269 seconds while in LSPI case 4 performance only reached 0.28 in 0.88 seconds which is nearly the triple of the time needed to generate the LSCAPI solution.

Table 8 demonstrates the number of generated solutions by LSPI and LSCAPI in 10 different simulation attempts and the time taken by each attempt. The LSCAPI in attempt 4 generated 34 different arrangements of the 8 Queens in 9.16 seconds while LSPI generated only 10 solutions in 8.8 seconds.

Finally we can assure that LSCAPI achieved a really obvious superiority over LSPI generating more accurate solutions in less time which leads to better policies through game solving.

6.2. GLEST. To evaluate Glest, the reward of the agent in cases under consideration generated by LSPI and LSCAPI was calculated based on (3) and listed in Table 9 demonstrating that LSCAPI achieved much higher rewards to agents after performing the selected actions by LSCAPI which leads to better policies through facing opponents. For example, LSCAPI achieved 15000 in case 8 while LSPI selected actions only achieved 8500 which is nearly the half.

As we also can see from Table 9, LSCAPI cases with higher reward are associated with less number of used actions. This means that the reward has an inverse relationship with the number of used actions so that when the number of actions decreases the reward increases and vice versa.

Table 8: Number of solutions and their time generated by LSCAPI versus LSPI.

Simulation attempts	LSCAPI		LSPI	
	Number of solutions	Time in seconds	Number of solutions	Time in seconds
1	22	10.31	13	08.80
2	22	08.80	15	08.89
3	25	09.03	14	08.81
4	34	09.16	10	08.80
5	20	09.06	12	08.39
6	25	09.05	13	08.79
7	31	09.35	19	08.85
8	25	08.99	17	09.15
9	24	09.03	16	08.45
10	35	09.18	12	08.64

Table 9: Reward function calculations of LSCAPI versus LSPI.

Cases	LSCAPI algorithm			LSPI algorithm
	Used actions	Ratio	Reward	Reward
1	20/24	0.833	27,570	21,500
2	19/24	0.792	30,010	25,000
3	16/24	0.667	33,000	26000
4	21/24	0.875	18,000	13,000
5	23/24	0.958	12,000	8,500
6	22/24	0.917	14,530	11,500
7	20/24	0.833	29,000	22,000
8	22/24	0.917	15,000	8,500
9	21/24	0.875	17,000	14,000
10	22/24	0.917	15,700	13,000
11	22/24	0.917	14,000	12,000

Table 10: Agent reward calculations of LSCAPI versus SARSA.

Cases	Agent reward in case of LSCAPI	Agent reward in case of SARSA
1	0.8	0.45
2	0.65	0.37
3	0.67	0.58
4	0.73	0.6
5	0.71	0.66
6	0.88	0.65
7	0.83	0.56
8	0.75	0.44
9	0.68	0.53
10	0.72	0.47

Finally Figure 16 presents the game army score achieved by LSCAPI policy learning against the score achieved by learning LSPI policy through the Duel scenario which is a medium difficulty level scenario. LSCAPI and LSPI policy scores were evaluated in 15 test games played taking into account that each policy game testing is played after every 10 games of learning.

Figure 16 clarified that game army using LSCAPI policy achieved much higher score than that achieved by LSPI, which means that LSCAPI really helps the game agents to efficiently face opponents and defeat them.

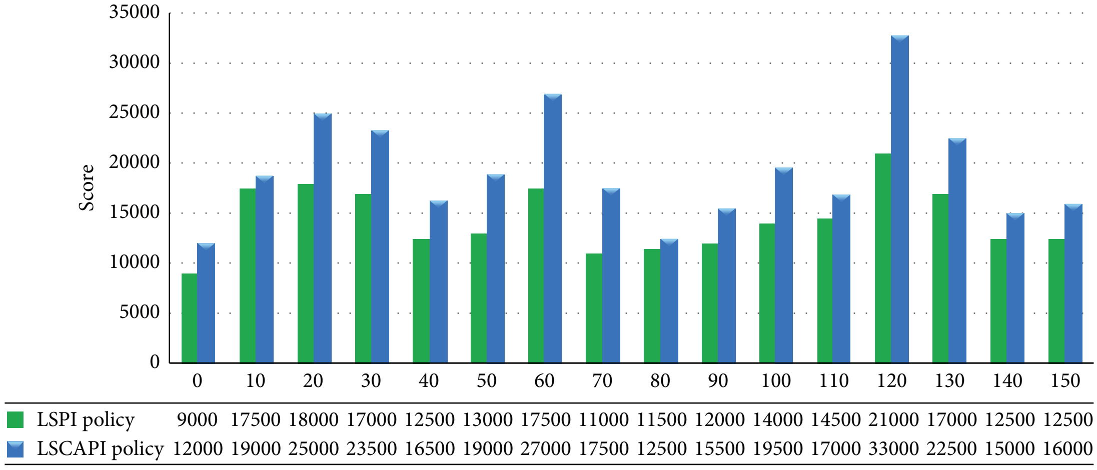
6.3. StarCraft Brood War. To evaluate Brood War, the reward function of the agent in the cases under consideration generated by LSCAPI and SARSA was calculated and listed in Table 10. The rewards values from Table 10 illustrated that LSCAPI achieved superiority over SARSA concerning the agent rewards, leading to better policies through facing opponents as in case 8, where LSCAPI achieved 0.75 as an agent reward value while SARSA action only achieved 0.44.

Finally Figure 17 presents the game army score achieved by LSCAPI policy learning against the score achieved by SARSA through the Terran scenario 1 which is a medium difficulty level scenario. LSCAPI and SARSA policy scores were evaluated in 15 test games played taking into account that each policy game testing is played after every 10 games of learning.

Figure 17 illustrated that LSCAPI achieved much higher scores through the 15 test games than those achieved by SARSA. Also we can notice that SARSA score values increase with a decreasing rate which finally turned into a fixed value whatever the number of trials. On the other side LSCAPI score values grow with an increasing rate indicating that the learning rate is increasing based on the increasing states of agent rewarding as a result of better actions selection. From all of the foregoing, we can assure that LSCAPI represents a real help to the game engines to easily and efficiently face opponents and defeat them.

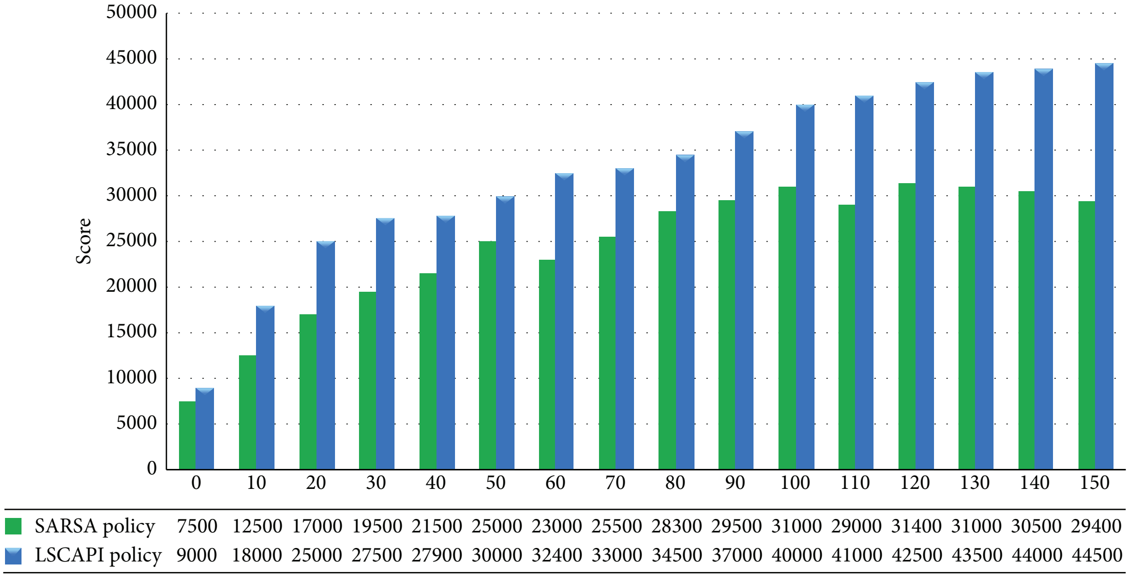
7. Conclusions and Future Work

Mapping from states to actions is the base function of game engine to face and react towards opponent, which is known as game policy. In this paper, we had studied the impact of batch reinforcement learning on enhancing game policy and proposed a new algorithm named Least-Squares Continuous Actions Policy Iteration (LSCAPI). The LSCAPI algorithm relied on LSPI considering handling continuous



	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
🟩 LSPI policy	9000	17500	18000	17000	12500	13000	17500	11000	11500	12000	14000	14500	21000	17000	12500	12500
🟦 LSCAPI policy	12000	19000	25000	23500	16500	19000	27000	17500	12500	15500	19500	17000	33000	22500	15000	16000

Figure 16: LSPI versus LSCAPI policy score on Duel scenario.



	0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
🟩 SARSA policy	7500	12500	17000	19500	21500	25000	23000	25500	28300	29500	31000	29000	31400	31000	30500	29400
🟦 LSCAPI policy	9000	18000	25000	27500	27900	30000	32400	33000	34500	37000	40000	41000	42500	43500	44000	44500

Figure 17: SARSA versus LSCAPI policy scores during Terran scenario 1.

actions through a tradeoff between available actions and electing the action that scores higher reward to the game agent.

LSCAPI was tested on two different types of games: board games represented in 8 Queens and RTS games represented in Glest and StarCraft Brood War open source games. The proposed algorithm was evaluated based on the agent reward values, scores, time, and number of generated solutions. The evaluation result indicated that LSCAPI achieved better performance, time, policy, and agent learning ability than original LSPI.

In the future we plan to pursue testing LSCAPI on more complicated games as chess and poker to check its impact on game policy especially that the nonplaying character in these two games heavily relies on the game policy.

Conflict of Interests

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

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Pearson, 3rd edition, 2009.

[2] E. Kok, *Adaptive reinforcement learning agents in RTS games [M.S. thesis]*, Utrecht University, Utrecht, The Netherlands, 2008, Thesis number INF/SCR-07-73.

[3] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass, USA, 1998.

[4] F. Guenter, M. Hersch, S. Calinon, and A. Billard, "Reinforcement learning for imitating constrained reaching movements," *Advanced Robotics*, vol. 21, no. 13, pp. 1521–1544, 2007.

[5] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, CRC Press, New York, NY, USA, 2010.

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

[7] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement Learning: State of the Art*, M. Wiering and M. van Otterlo, Eds., Springer, 2011.

[8] S. Kalyanakrishnan and P. Stone, "Batch reinforcement learning in a complex domain," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07)*, pp. 650–657, ACM, New York, NY, USA, May 2007.

[9] D. J. Lizotte, M. Bowling, and S. A. Murphy, "Linear fitted-q iteration with multiple reward functions," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3253–3295, 2012.

[10] A. Antos, R. Munos, and C. Szepesvari, "Fitted Q-iteration in continuous action-space MDPs," in *Advances in Neural Information Processing Systems 20*, pp. 9–16, MIT Press, Cambridge, Mass, USA, 2008.

[11] X. Xu, H. Zhang, B. Dai, and H.-G. He, "Self-learning path-tracking control of autonomous vehicles using kernel-based approximate dynamic programming," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN '08)*, pp. 2182–2189, Hong Kong, June 2008.

[12] X. Xu, C. Lian, L. Zuo, and H. He, "Kernel-based approximate dynamic programming for real-time online learning control: an experimental study," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 1, pp. 146–156, 2014.

[13] M. Lagoudakis and R. Parr, "Model-free least-squares policy iteration," in *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, 2001.

[14] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[15] L. Buşoniu, A. Lazaric, M. Ghavamzadeh, R. Munos, R. Babuška, and B. De Schutter, "Least-squares methods for policy iteration," in *Reinforcement Learning*, vol. 12 of *Adaptation, Learning, and Optimization*, pp. 75–109, Springer, Berlin, Germany, 2012.

[16] M. Riedmiller, "Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method," in *Machine Learning: ECML 2005*, Springer, Porto, Portugal, 2005.

[17] B. King, A. Fern, and J. Hostetler, "On adversarial policy switching with experiments in real-time strategy games," in *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS '13)*, pp. 322–326, Rome, Italy, June 2013.

[18] 2015, https://johnbaps.wordpress.com/2014/03/20/intelligent-agents-in-artificial-intelligence/.

[19] 2014, http://cs.smith.edu/~thiebaut/transputer/chapter9/chap9-4.html.

[20] I. Ghory, "Reinforcement learning in board games," Tech. Rep., Computer Science Department, Bristol University, Bristol, UK, 2004.

[21] M. Genesereth, N. Love, and B. Pell, "General game playing: overview of the AAAI competition," *AI Magazine*, vol. 26, no. 2, 2005.

[22] P. Ding and T. Mao, *Reinforcement Learning in Tic-Tac-Toe Game and Its Similar Variations*, vol. 1, Thayer School of Engineering at Dartmouth College, Hanover, NH, USA, 2009.

[23] T. B. Trinh, A. S. Bashi, and N. Deshpande, "Temporal difference learning in Chinese Chess," in *Tasks and Methods in Applied Artificial Intelligence*, vol. 1416 of *Lecture Notes in Computer Science*, pp. 612–618, Springer, Berlin, Germany, 1998.

[24] T. P. Runarsson and S. M. Lucas, "Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 628–640, 2005.

[25] M. Wiering, J. Patist, and H. Mannen, "Learning to play board games using temporal difference methods," Tech. Rep. UU-CS-2005-048, Institute of Information and Computing Sciences, Utrecht University, 2007.

[26] M. Block, M. Bader, E. Tapia et al., "Using reinforcement learning in chess engines," *Research in Computing Science*, vol. 35, pp. 31–40, 2008.

[27] I. E. Skoulakis and M. G. Lagoudakis, "Efficient reinforcement learning in adversarial games," in *Proceedings of the IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI '12)*, vol. 1, pp. 704–711, IEEE, Athens, Greece, November 2012.

[28] M. Szubert and W. Jaskowski, "Temporal difference learning of N-tuple networks for the game 2048," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG '14)*, pp. 1–8, IEEE, Dortmund, Germany, August 2014.

[29] G. Schrage, "The eight queens problem as a strategy game," *International Journal of Mathematical Education in Science and Technology*, vol. 17, no. 2, pp. 143–148, 1986.

[30] S. Lim and K. Son, "The improvement of convergence rate in n-queen problem using reinforcement learning," *International Journal of Information Technology*, vol. 11, no. 5, pp. 52–60, 2005.

[31] 2014, http://en.wikipedia.org/wiki/Real-time_strategy.

[32] M. Buro, "Real-time strategy gaines: a new AI research challenge," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, pp. 1534–1535, August 2003.

[33] K. Dimitriadis, *Reinforcement Learning in Real Time Strategy Games Case Study on the Free Software Game Glest*, Department of Electronic and Computer Engineering, Technical University of Crete, Chania, Greece, 2009.

[34] M. Aref, M. Zakaria, and S. Sarhan, "Real-time strategy experience exchanger model [real-see]," *International Journal of Computer Science Issues*, vol. 8, no. 3, supplement 1, pp. 360–368, 2011.

[35] B. Marthi, S. Russell, and D. Latham, "Writing stratagus-playing agents in concurrent ALisp," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*, pp. 67–71, Edinburgh, Scotland, 2005.

[36] A. Gusmao, *Reinforcement learning in real-time strategy games [M.S. thesis]*, Aalto School of Science, Department of Information and Computer Science, 2011.

[37] M. Leece and A. Jhala, "Reinforcement learning for spatial reasoning in strategy games," in *Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE '13)*, pp. 156–162, October 2013.

[38] H. Sethy, A. Patel, and V. Padmanabhan, "Real time strategy games: a reinforcement learning approach," *Procedia Computer Science*, vol. 54, pp. 257–264, 2015.

[39] 2014, http://glest.org/en/index.php.

[40] 2014, http://glest.org/en/techtree.php.

[41] K. Efthymiadis and D. Kudenko, "Using plan-based reward shaping to learn strategies in starcraft: broodwar," in *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG '13)*, pp. 1–8, IEEE, August 2013.

[42] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar," in *Proceedings of the IEEE International Conference on Computational Intelligence and Games (CIG '12)*, pp. 402–408, Granada, Spain, September 2012.

[43] J. Eriksson and D. Ø. Tornes, *Learning to play starcraft with case-based reasoning: investigating issues in large-scale case-based*

planning [Master of Science in Computer Science], Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, 2012.

[44] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar," in *Proceedings of the IEEE International Conference on Computational Intelligence and Games (CIG '12)*, pp. 402–408, IEEE, Granada, Spain, September 2012.

[45] C. Siebra and G. Neto, "Evolving the behavior of autonomous agents in strategic combat scenarios via SARSA reinforcement learning," in *Proceedings of the 13th Brazilian Symposium on Computer Games and Digital Entertainment (SBGAMES '14)*, pp. 115–122, Porto Alegre, Brazil, November 2014.

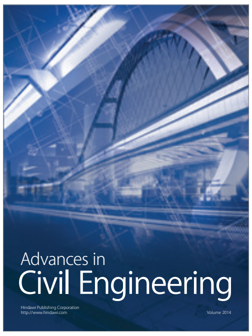
Journal of
Engineering

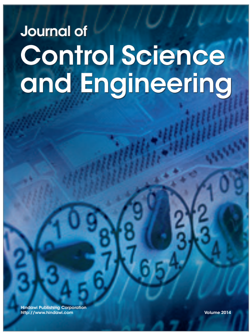
The Scientific World Journal

International Journal of
Rotating Machinery

Journal of
Sensors

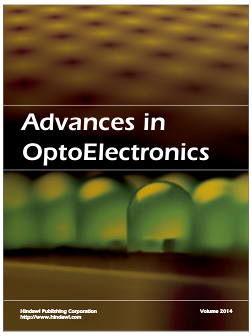
International Journal of
Distributed Sensor Networks


Advances in
Civil Engineering


Journal of
Control Science and Engineering

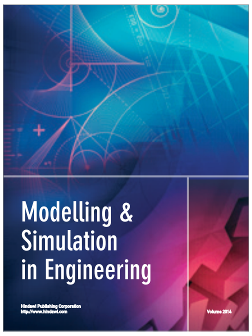
Journal of
Robotics

Journal of
Electrical and Computer Engineering

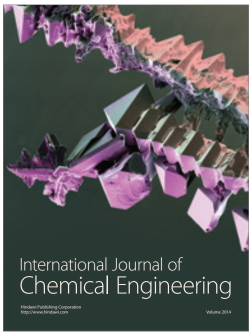

Advances in
OptoElectronics

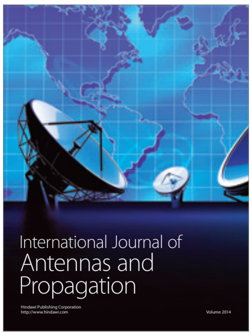
VLSI Design

International Journal of
Navigation and Observation

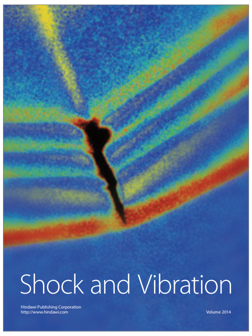

Modelling & Simulation in Engineering

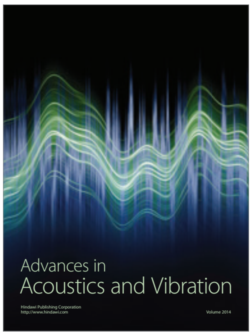
International Journal of
Aerospace Engineering


International Journal of
Chemical Engineering


International Journal of
Antennas and Propagation

Active and Passive
Electronic Components


Shock and Vibration


Advances in
Acoustics and Vibration