Monte Carlo Tree Search with Temporal-Difference Learning for General Video Game Playing

Ercüment İlhan
Graduate School of Science, Engineering
and Technology
Istanbul Technical University, Turkey
ilhane@itu.edu.tr

A. Şima Etaner-Uyar
Department of Computer Engineering
Istanbul Technical University, Turkey
etaner@itu.edu.tr

Abstract—**General Video Game Playing (GVGP) is a problem where the objective is to create an agent that can play multiple games with different properties successfully with no prior knowledge about them. Being an important sub-field in General Artificial Intelligence, GVGP has drawn a considerable amount of interest, and the research in this field got intensified with the release of General Video Game AI framework and competition. As of today, even though this problem has been approached with many different techniques, it is still far from being solved. Monte Carlo Tree Search (MCTS) is one of the most promising baseline approaches in literature. In this study, MCTS algorithm is enhanced with a recently developed temporal-difference learning method, namely True Online Sarsa(lambda) to make it able to exploit domain knowledge by using past experience. Experiments show that the proposed modifications improve the performance of MCTS significantly in GVGP, and applications of reinforcement learning techniques in this domain is a promising subject that needs to be further researched.**

I. Introduction

Games have always been an attractive subject for Artificial Intelligence (AI) research because of their controllability and diversity, which makes them perfect benchmarking environments. Over the years, many algorithms were developed which surpassed human-level performance in a broad spectrum of different games ranging from board games such as Go [1] to classical Atari video games [2]. While there has been a serious advancement in the techniques specializing on particular games, developing an algorithm that is capable of playing multiple different games successfully still remains to be a challenging task as an important objective in the quest in solving General Artificial Intelligence.

One of the noteworthy steps taken towards this problem was the creation of the General Game Playing (GGP) concept by the Stanford Logic Group [3]. It allowed many turn-based board games to be represented by a Game Description Language (GDL) and game playing agents to be built based on it to compete against each other. Then more recently, a similar work was done to represent two-dimensional semi-continuous arcade video games with a Video Game Description Language (VGDL) [4] which led to the emergence of General Video Game Playing (GVGP) and the development of General Video Game AI (GVGAI) framework and competitions [5]. These GVGP competitions based on GVGAI are being held on a regular basis after its introduction in some conferences and determined the best agents in this challenging problem to date by evaluating them within a set of different games without any previous game specific knowledge or modifications.

Despite having various approaches such as evolutionary algorithms, tree search methods and their combinations submitted to these competitions, the best competitors still struggle to win more than half of the games they play. When analyzed, the results show that one of the most successful fundamental approaches is Monte Carlo Tree Search (MCTS) [6]. Even in its most basic form with no modifications, it manages to outperform complex controllers in a considerable amount of games. However, some properties of GVGP limit the success of MCTS in this domain. Most importantly, the performance of MCTS is only determined by its initial parameter setup without any way of adapting to the game instances by taking advantage of past experience or by making use of domain knowledge. The aim of this paper is to enhance MCTS with an online on-policy temporal-difference learning technique, namely true online Sarsa(λ), to make it capable of adapting to the games online by exploiting domain knowledge and biasing its rollouts accordingly.

The paper is structured as follows: Section II defines the GVGAI framework and competition in detail. Afterwards, the previous studies done on MCTS in GVGAI domain are reviewed in Section III. In Section IV, the backgrounds of two main algorithms employed in this study, namely MCTS and true online Sarsa(λ), are given. In Section V the proposed method is explained. Section VI describes the experiments and provides an analysis of the results. Finally, the study is concluded with suggestions on future work in Section VII.

II. The General Video Game AI Framework

GVGAI framework is the Java implementation of VGDL (py-vgdl) [5]. In this framework, single/multi-player two-dimensional games that are defined by VGDL can be loaded and run either for the agents or for self-play. Currently, there are more than 90 arcade games in this framework[1] including some of the well known classical Atari games like Missile Command, Pacman and Sokoban. By having such a variety

[1] *The GVG-AI Competition Framework - 2016* is used for the studies and experiments in this paper.

TABLE I

SET 1 OF 10 GAMES, USED AS TRAINING SET IN CIG 2014

Game	Description	Scoring
Aliens	The player must shoot every single alien to win the game. If the player touches the aliens or the missiles they shoot, the game is lost.	● $+1$ for destroying walls. ● $+2$ for killing an alien. ● -1 for touching an alien or its missile.
Boulderdash	The player must dig through a cave and collect all the gems scattered around the map while avoiding falling boulders and enemies to win the game. Making two enemies collide spawns new gems.	● $+1$ for making a new gem spawn. ● $+2$ for collecting a diamond. ● -1 for touching an enemy or a falling boulder.
Butterflies	The player must capture all of the butterflies to win the game. Having a butterfly touch a cocoon spawns new butterflies by destroying it; however, the game is lost if there are no cocoons left.	● $+2$ for capturing a butterfly.
Chase	The player must catch and kill every goat to win the game. The goats run away from the player and if a goat finds a corpse of a slain goat, it turns into an angry goat and starts chasing the player. Angry goats must be avoided, else the game is lost.	● $+1$ for killing a goat. ● -1 for being killed by a goat.
Frogs	The player must reach the predesignated goal to win the game. The path has rivers with moving logs and roads with trucks on them. Rivers and trucks kill the player on contact, and therefore must be avoided.	● $+1$ for reaching the goal. ● -2 for being hit by a truck.
Missile Command	The player must catch and destroy enemies in air before they reach the cities on ground to win the game. The game is lost if all cities are destroyed by the enemies.	● $+2$ for destroying an enemy. ● -1 for every destroyed city.
Portals	The player must reach the predesignated goal to win the game. Portals on the map teleport the player from one place to another while the lasers kill the player on contact.	● $+1$ for reaching the goal.
Sokoban	The player must push all of the boxes in the holes to win the game. Failing to plan accordingly and getting a box stuck in a place makes the game impossible to be won.	● $+1$ for pushing a box down a hole.
Survive Zombies	The player must gather honey to avoid zombies and survive until the timer runs out to win the game. The game is lost if the player gets caught by zombies without any honey.	● $+1$ for collecting honey. ● $+1$ for killing a zombie. ● -1 for being killed.
Zelda	The player must get the key and reach the exit to win the game. There are enemies on the way that can also be killed for extra points.	● $+1$ for collecting the key. ● $+1$ for reaching the exit with the key. ● $+2$ for killing an enemy. ● -1 for being killed.

of games with different subsets of available actions, win-lose conditions and scoring rules, GVGAI framework clearly presents a challenging environment for the algorithms.

An agent to be evaluated in a *single-player planning track* has to follow the competition rules defined by the GVGAI framework. This evaluation is performed on the competition server by running the agent multiple (generally 25) times independently across a set of 10 games with 5 different levels in each game. At the start of every game run, the agent is given 1 second to be initialized. Afterwards, at every game step which is 40 milliseconds to match the real-time play at 25 frames per second, the agent is required to return an action. Initially and at every game step the agent receives a $StateObservation$ object with high-level momentary information about the game. Just like what human players observe, this object contains only the grid of numbers representing items on the screen, available actions, history of events, score and outcome of the game. An agent can never access the VGDL definitions of these games and has to figure out the objectives of the games on its own. However, a forward model of games are provided to the agent to make it possible to plan ahead. The games are set to terminate after 2000 time steps to avoid infinite runs in games with no lose conditions like Sokoban.

The game set employed in this research is the training set of the GVGAI competition held in 2014 IEEE Conference of Computational Intelligence and Games (CIG 2014). Having enough game variety and being experimented on extensively by most of the previous studies on GVGAI made this set a suitable choice for this research as well. The list of games with descriptions can be found in Table I. It should also be noted that in addition to the details on this table, these games also have different action sets.

III. RELATED WORK

MCTS has proven its efficiency in a wide range of games, and has received various enhancements to be further specialized on most of them [7]. By the introduction of GVGAI, MCTS has become one of the core algorithms around which the research revolves in this problem domain too. Related work given in this section is limited to applications of MCTS in the GVGAI domain with focus on the ones with temporal-difference learning considering the scope of this study.

To this date, MCTS had many extensions for GVGAI such as the addition of path-finding methods [8] [9], modifications directly in the tree policies [10] [11] [12] [13], hybridization with evolutionary algorithms that also involve online or offline learning [14] [15] [16], utilization of planning techniques [17] and multi-objective methods [18], which have boosted its performance in many situations but with no overall best solutions.

On the other hand, while being one of the most successful methods in the game playing domain, MCTS with temporal-difference learning enhancements have only had a few studies done in GVGAI due to the nature of this challenge that prohibits offline game-specific learning. Deleva [19] used temporal-difference with MCTS in the tree building stage and tree policy, by incorporating λ-return to modify how scores are stored and backed-up in the tree nodes; then using Sarsa(λ) to perform an online learning on the tree level. In an other study done by Chu et al. [20], a temporal-difference learning method Q-learning was employed to learn weights of game features recommended in a previous work, to modify MCTS in the rollout stage. As it is claimed in these studies, temporal-difference learning methods have achieved promising results and are as viable as other enhancements in GVGAI. However, these studies could only cover a little portion of these algorithms, without the efficiency of an online on-policy learning method added which is the focus of this study.

IV. Background

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [21] is a tree search algorithm built around the ideas of estimating the true value of an action with random simulations and using these values for an efficient best-first policy. The algorithm works in iterations with anytime fashion, meaning that while its estimations get more accurate as it runs, it can be terminated at any point to return a current best estimation.

MCTS has an empty tree initially and starts its process once it is given a root state node. Afterwards, it executes four main steps as shown in Figure 1. The details of these iterations are as follows:

1) **Selection:** A child node is selected recursively from a root node with *tree policy* until an expandable node is reached.
2) **Expansion:** An unexplored child node is added to the current node to be evaluated.
3) **Simulation:** A simulation (also called a rollout) is performed from the new node in order to determine its value.
4) **Backpropagation:** The value of the new node is back-propagated all the way up to the root node through the selected nodes.

$$UCT_j = \overline{X}_j + C\sqrt{\frac{\ln n}{n_j}} \tag{1}$$

Tree policy plays a critical role in how the tree is built. As this is an important factor in the overall performance of MCTS, it has been studied extensively and different approaches have been proposed [7]. The most popular of these is the *Upper Confidence Bound for Trees* (UCT) algorithm [22]. In this method, a multi-armed bandit problem policy UCB1 [23] was used as a tree policy with the aim of providing a control mechanism in exploration-exploitation. According to UCT, a child node j that maximizes Equation 1 is selected at every selection step. In this equation, $\overline{X}_j$ is the average value of node j determined by the simulations and is normalized to be in $[0, 1]$; n is the total number of selections performed from the parent node, and n_j is how many times the child node j is selected. Clearly, the left term in this equation stands for exploitation while the right term represents exploration, and parameter C controls this relationship.

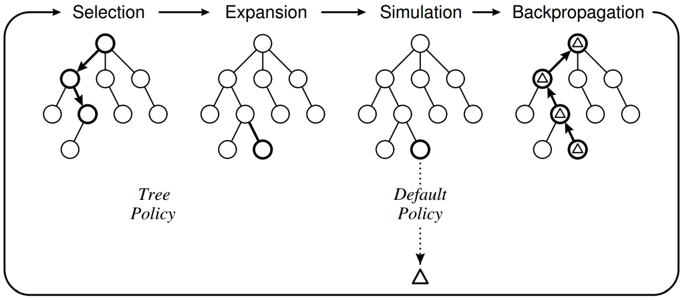

Fig. 1. Main steps of MCTS [7].

Default policy which controls how simulations are run is another fundamental part of MCTS. By default, this policy makes uniformly random selection of child nodes until it reaches a terminal state or a predetermined depth in the tree.

Algorithm 1 MCTS state evaluation method.

```
function EVALUATE($StateObservation$)
    $reward \leftarrow StateObservation.GameScore$
    if $StateObservation$ is terminal then
        if game is won then
            $reward \leftarrow reward + HIGH\_VALUE$
        else
            $reward \leftarrow reward - HIGH\_VALUE$
        end if
    end if
    return $reward$
end function
```

Vanilla MCTS (or sample MCTS) available in the GVGAI framework is the baseline algorithm used in this research. This controller follows the previously defined tree policy and default policy. Parameter C is set to a widely used value $\sqrt{2}$ to maintain exploration-exploitation balance, and the maximum depth for rollouts is limited by 10 due to the 40 milliseconds of action time constraint. As the rollouts rarely manage to reach a terminal state, a custom state evaluation method is also defined as seen in Algorithm 1 in order to produce an outcome for non-terminal states.

B. True Online Sarsa(λ)

Temporal-difference (TD) learning is a core concept in reinforcement learning (RL) that combines Monte Carlo ideas with dynamic programming techniques [24]. Over the years, it has had many improvements and served as a base for many popular algorithms like TD(λ), State-Action-Reward-State-Action (Sarsa) and Q-learning. The method employed in this study is the recently introduced variant *true online Sarsa(λ)* [25]. Compared with its counterparts, true online Sarsa(λ) stands out by its simple implementation, low computational cost, ability to perform online learning only with temporal local information, which makes it an applicable and promising choice in GVGAI domain.

True online Sarsa(λ) is an on-policy control algorithm designed for discrete-time *Markov decision processes* (MDPs). MDPs are defined with 5 main elements: S, A, p, r, γ. S is the set of states; A is the set of all actions; p is the probabilistic transition function $p(s'|s,a)$, indicating the probability of going into state $s' \in S$ from $s \in S$ when action $a \in A$ is taken; r is the reward function $r(s,s',a)$, returning a reward for transition from s to s' under action a; γ is the discount factor, controlling how future rewards are evaluated.

$$G_t = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i} \tag{2}$$

In such MDPs, *return* value G_t which represents the discounted sum of rewards from time step t is calculated as in Equation 2. If the process has a terminal state, this summation is also terminated.

$$v_\pi(s) = \mathbb{E}\{G_t | S_t = s, \pi\} \tag{3}$$

True online Sarsa(λ) tries to accomplish two main tasks in MDPs. First one of these is learning the expected value function v_π of an MDP under policy π as defined in Equation 3.

$$v_{\pi_*}(s) = \max_\pi v_\pi(s) \tag{4}$$

The second task, which is more challenging, is determining the optimal policy π_* with the highest value in each state as in Equation 4.

Based on these objectives, the basic temporal-difference learning method has received many improvements and extensions over time, resulting in more complex algorithms for different purposes, with true online Sarsa(λ) being one of them with the pseudo-code shown in Algorithm 2. The new variables introduced here are: $\boldsymbol{\theta}$, weights; $\boldsymbol{e}$, eligibility-traces; $\boldsymbol{\psi}$, feature representation, which is also used in forms of $\boldsymbol{\psi}(s)$ and $\boldsymbol{\psi}(s,a)$, denoting the corresponding state s and action a; Q, approximated state value; δ, td-error; α, learning rate; λ trace decay parameter. Some of the important properties of this algorithm is highlighted in the remaining part of this section. For the complete and detailed explanation of the derivation, the reader is referred to [25].

$$V(s, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s) \tag{5}$$

Most of the games and real-world problems have large state spaces. Consequently, the majority of research in reinforcement learning that deals with these kind of problems is focused on using value function approximation instead of state lookup tables to be able to generalize between these states. In true online Sarsa(λ) the case of linear function approximation is covered. According to this, value of a state s is calculated by multiplying the weight vector θ with the corresponding state representation which is generated by the $\phi(s)$ function as in Equation 5, converting the policy evaluation task into learning the weight vector $\boldsymbol{\theta}$.

Another noteworthy development in this algorithm is the usage of *eligibility-trace* vectors, denoted by the λ symbol in the algorithm name. According to this technique, a new vector $\boldsymbol{e}$ with the same structure the feature vector has, is employed to hold frequency and recency information of corresponding features, controlled by the λ parameter. This is a very useful extension in cases with delayed rewards where credit assignment is mandatory.

Algorithm 2 True online Sarsa(λ) [25]

```
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{init}$
**for all** episodes **do**
    obtain initial state $S$
    select action $A$ based on state $S$
    $\boldsymbol{\psi} \leftarrow$ features corresponding to $S$, $A$
    $\boldsymbol{e} \leftarrow 0; Q_{old} \leftarrow 0$
    **while** terminal state has not been reached **do**
        take action $A$, observe next state $S'$ and reward $R$
        select action $A'$ based on state $S'$
        $\boldsymbol{\psi}' \leftarrow$ features corresponding to $S'$, $A'$
        $Q \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}$
        $Q' \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}'$
        $\delta \leftarrow R + \gamma Q' - Q$
        $\boldsymbol{e} \leftarrow \gamma\lambda \boldsymbol{e} + \boldsymbol{\psi} - \alpha\gamma\lambda(\boldsymbol{e}^\top \boldsymbol{\psi})\boldsymbol{\psi}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\delta + Q - Q_{old})\boldsymbol{e} - \alpha(Q - Q_{old})\boldsymbol{\psi}$
        $Q_{old} \leftarrow Q'$
        $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi}'; A \leftarrow A'$
    **end while**
**end for**
```

V. Proposed Method

MCTS in its vanilla form (see Section IV-A for details), with fixed depth rollouts that use random uniform policy performs poorly in some cases. Firstly, whenever a rollout ends up in a non-terminal state, it has to rely on a momentary game score rather than the actual game outcome, which is the $GameScore$ provided in the $StateObservation$ in this case. In addition, no matter how close it gets to an item that can increase the score when interacted with, the actual $GameScore$ does not change unless there is an actual score gain. This clearly shows a requirement for a custom state evaluation method that considers domain knowledge in calculations. In addition, even with an efficient state evaluation, rollouts need to follow a policy utilizing this information, as having them completely random will fail to provide meaningful feedback in most of the cases. A way of achieving these, performing a policy evaluation and an on-policy control respectively, is to implement a learning procedure based on past experience. Therefore, by treating the rollouts of MCTS as separate episodes of past experience, true online Sarsa(λ) algorithm was executed in every rollout with ϵ-greedy selection policy with constant ϵ in place of default policy. When taking actions, snapshot of learned weights from

TABLE II

Results of 100 runs for the games in set 1 with 5 levels each - columns with symbols represent statistical significance comparison of the corresponding algorithm with V-MCTS, determined by Welch's t-test with p-value < 0.05. Symbols and their meanings: $=$, outcomes are equal; $+$, TD-MCTS is better than V-MCTS, but is not significantly different; $++$, TD-MCTS is better than V-MCTS, and is significantly different; $-$, TD-MCTS is worse than V-MCTS, but is not significantly different; $--$, TD-MCTS is worse than V-MCTS, and is significantly different.

	Win Percentages					Average Scores				
Game	**V-MCTS (50)**	**TD-MCTS (50)**		**TD-MCTS (25)**		**V-MCTS (50)**	**TD-MCTS (50)**		**TD-MCTS (25)**	
Aliens	$\mathbf{100 \pm 0}$	$\mathbf{100 \pm 0}$	$=$	$\mathbf{100 \pm 0}$	$=$	341.9 ± 1	$\mathbf{351.8 \pm 0.7}$	$++$	348.5 ± 0.7	$++$
Boulderdash	14.6 ± 1.3	$\mathbf{17.8 \pm 1.7}$	$+$	14.6 ± 1.4	$=$	$\mathbf{70.4 \pm 0.9}$	64.7 ± 1	$--$	58 ± 1.3	$--$
Butterflies	$\mathbf{99.4 \pm 0.3}$	$\mathbf{99.4 \pm 0.3}$	$=$	97.6 ± 0.6	$--$	$\mathbf{160.9 \pm 1.8}$	148.4 ± 1.6	$--$	153.5 ± 1.4	$--$
Chase	5.2 ± 1	7 ± 1.2	$+$	$\mathbf{8 \pm 1.2}$	$+$	$\mathbf{14.6 \pm 0.4}$	13.9 ± 0.5	$-$	13.1 ± 0.5	$--$
Frogs	8.8 ± 1.3	36 ± 1.4	$++$	$\mathbf{44 \pm 1.8}$	$++$	0.4 ± 0.1	1.8 ± 0.1	$++$	$\mathbf{2.2 \pm 0.1}$	$++$
Missile Command	61.8 ± 1.7	$\mathbf{97.8 \pm 0.6}$	$++$	93.6 ± 1.1	$++$	22.9 ± 0.6	$\mathbf{42.5 \pm 0.4}$	$++$	36 ± 0.4	$++$
Portals	12.8 ± 1	$\mathbf{15.8 \pm 0.9}$	$++$	15.2 ± 1.1	$+$	0.6 ± 0.1	$\mathbf{0.8 \pm 0}$	$++$	$\mathbf{0.8 \pm 0.1}$	$+$
Sokoban	27.0 ± 1.3	$\mathbf{32.2 \pm 1.3}$	$++$	29 ± 1.2	$+$	$\mathbf{6.7 \pm 0.1}$	5.5 ± 0.1	$--$	5.2 ± 0.1	$--$
Survive Zombies	$\mathbf{48.6 \pm 1.2}$	46.4 ± 1.4	$-$	44 ± 1.5	$--$	$\mathbf{14 \pm 0.5}$	13.5 ± 0.6	$-$	11.5 ± 0.6	$--$
Zelda	35.6 ± 1.9	$\mathbf{53 \pm 2.2}$	$++$	51.4 ± 2.2	$++$	25.5 ± 0.4	25.3 ± 0.5	$-$	$\mathbf{27.7 \pm 0.5}$	$++$
Overall	41.4 ± 1.1	$\mathbf{50.5 \pm 1.1}$	$++$	49.7 ± 1.2	$++$	65.7 ± 0.6	$\mathbf{66.8 \pm 0.6}$	$+$	65.6 ± 0.6	$-$

the previous game-step is used for the rollouts to ensure that states are evaluated and compared equally.

While the main flow of these two algorithms are kept the same, hybridizing them in the GVGAI domain efficiently took some additional modifications. These are explained in the following sections.

A. State Representation

Game instances in the GVGAI domain can have large grids with many different item types resulting in huge state spaces when represented in a tabular way, making it hard to learn and generalize between states for the algorithms. Therefore, as suggested in true online Sarsa(λ), linear value function approximation is used with the state representation method.

According to this method, the $StateObservation$ grid with the items on it, is processed to obtain the smallest euclidean distances between objects of every item type including the avatar itself. For a game with N different item types (including the avatar), this representation has a length of $\binom{N}{2}$. Then, these distances are normalized with respect to the maximum euclidean distance in the game grid to normalize them between 0 and 1. Afterwards, these values are subtracted from 1 to be converted into proximities. Finally, the resulting vector is element-wise squared giving the final representation.

The weights $\boldsymbol{\theta}$ that are subject to a learning procedure are all initially set to 1.0 instead of 0.0 to provide a curiosity effect to make the avatar likely to get closer to the items to discover the interactions. This is a desired behavior in almost every game.

B. Custom Rollout Returns

$$v_\pi(s_t) = \sum_{i=1}^{T-t} \gamma^{i-1} R_{t+i} + \gamma^{T-t} \max_{a' \in \mathrm{A}} \boldsymbol{\theta}^\top \boldsymbol{\psi}(s_T, a') \qquad (6)$$

As stated previously, rollouts are treated as separate episodes (sub-MDPs) in this approach. Thus, unlike vanilla MCTS, where the final state is evaluated with a function, outcomes of rollouts are calculated as the discounted sum of rewards from the initial state s_t to the final state s_T under the rollout policy π; then, the approximated value of this final state is added up, as in Equation 6 where T ($> t$) is the maximum time-step (or the maximum rollout depth). This procedure can be viewed as taking a few steps in reality and then predicting the remaining steps. Discount factors are also employed in the backup steps, ensuring that same levels of depth have the same impact on the tree node scores.

Final form of this proposed algorithm can be viewed in Algorithm 3 and Algorithm 4, divided in two parts. At the start of each game, 1 second of free time is used to take random actions repeatedly to discover every possible item type in the game to build a feature base. Afterwards, every time the framework gives the $StateObservation$ and asks for an action, this object is passed to SEARCH as a game state s, starting the algorithm. Newly introduced parameters in here are as follows: v, tree node; D, depth of a tree node; N, visit count of a tree node; X, cumulative value of a tree node; P, score of a state.

VI. Experiments

The experiments in this study were performed on the game set 1 described in Table I by having them played 100 times on each of the 5 different levels by the following controllers: V-MCTS and TD-MCTS. First controller V-MCTS stands for vanilla MCTS provided in the GVGAI framework, with its default parameters C set to $\sqrt{2}$ and maximum rollout depth set to 10. Second controller TD-MCTS stands for the proposed algorithm, MCTS with temporal-difference learning. For this controller, parameters of the core MCTS part were kept the same as V-MCTS; and the learning parameters α, λ, γ, ϵ (of ϵ-greedy) were determined empirically and set as 0.05, 0.2, 0.99, 0.1 respectively.

For both of these tested algorithms, a fixed number of 50 rollouts at every game-step are used instead of the 40 milliseconds time limit in order to have consistency between tests. This number was determined by averaging the number

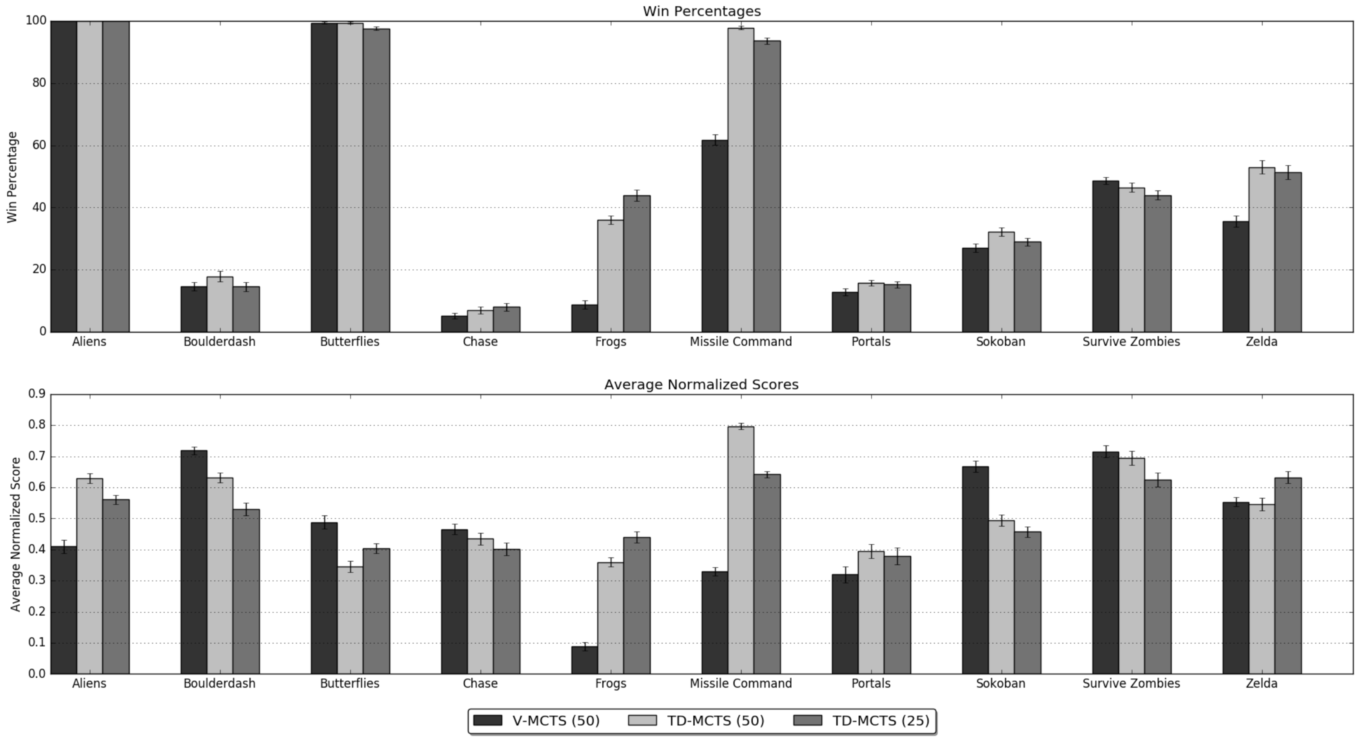

Fig. 2. Win percentages and average normalized scores with standard errors of V-MCTS (50), TD-MCTS (50) and TD-MCTS (25) algorithms.

of rollouts V-MCTS can do across all games using the default time budget of 40 milliseconds. In addition, considering the additional computational cost the learning steps in TD-MCTS brings, it is also tested with 25 rollouts per step setting to effect of trade-off between learning process and extra number of rollouts. This relationship is determined by the implementation and also by the complexities of the games. The number of rollouts used in the algorithms are shown in parentheses next to their names in Table II and Figure 2.

Results of the experiments with their standard errors are shown in Table II. The table has two sections for the percentage of victories and average scores achieved in the games, with the highest values in the corresponding rows denoted in bold. While performance is determined by the number of victories, it is also beneficial to examine the scores for the behavior of the algorithms. In addition to these, TD-MCTS columns have an additional cell to the right of every result, showing the statistical significance comparison with the corresponding V-MCTS result, determined by Welch's t-test with p-value < 0.05. Symbols used in this cell and their meanings are as follows: $=$, outcomes are equal; $+$, TD-MCTS is better than V-MCTS, but is not significantly different; $++$, TD-MCTS is better than V-MCTS, and is significantly different; $-$, TD-MCTS is worse than V-MCTS, but is not significantly different; $--$, TD-MCTS is worse than V-MCTS, and is significantly different. These results are also shown in Figure 2 in graph form where the normalized versions of scores (with respect to minimum and maximum scores achieved by any algorithm in each game) are used for better comparability.

As it can be seen, TD-MCTS with 50 rollouts have significantly outperformed V-MCTS in 5 games (*Frogs*, *Missile Command*, *Portals*, *Sokoban*, *Zelda*) by only being slightly worse in *Survive Zombies*, achieving 50.5% win percentage overall compared to V-MCTS with 41.1%, showing a considerable improvement. Similarly, TD-MCTS with 25 rollouts has still managed to surpass V-MCTS performance significantly in 3 games (*Frogs*, *Missile Command*, *Zelda*) while being significantly worse in 2 games (*Butterflies*, *Survive Zombies*) and achieving 49.7% win percentage. The games TD-MCTS have performed well are *Aliens*, *Butterflies*, *Frogs*, *Missile Command* and *Zelda*; despite its comparison with V-MCTS. Win conditions in these games are closely related with the proximities between one or more items in the games, which is covered by the feature set used and learned by TD-MCTS successfully. In games like *Boulderdash* and *Chase* where long-term decisions and pathfinding ability matter proposed modifications could not provide any significant improvements.

On the other hand, there is no clear winner in the average scores obtained in games. Aiming for the greedy decisions that give a score increase rather than doing long-term planning to maximize score leads to an early win in most of the cases. For example, in *Butterflies*, TD-MCTS chases down all the butterflies and wins the game earlier while V-MCTS unintentionally takes its time to gain a higher score before finishing the game.

Algorithm 3 TD-MCTS algorithm part 1.

```
function Search($s$)
    $\boldsymbol{\theta}_S \leftarrow \boldsymbol{\theta}$
    initialize tree with $v_r$
    $s(v_r) \leftarrow s; a(v_r) \leftarrow null$
    $\boldsymbol{e}(v_r) \leftarrow 0$
    $D(v_r) \leftarrow 0; G(v_r) \leftarrow 0$
    $N(v_r) \leftarrow 0; X(v_r) \leftarrow 0$
    while computational budget is available do
        $v \leftarrow$ TreePolicy($v_r$)
        $Q_v \leftarrow$ Rollout($v$)
        Backup($v, Q_v$)
    end while
    return $a(\operatorname*{argmax}_{v' \in \text{ children of } v_r} N(v'))$
end function

function TreePolicy($v$)
    while $v$ is not a terminal node do
        if $v$ is fully expanded then
            $v \leftarrow$ BestChild($v$)
        else
            return Expand($v$)
        end if
    end while
    return $v$
end function

function BestChild($v$)
    return $\operatorname*{argmax}_{v' \in \text{ children of } v} [G(v') + \gamma \frac{X(v')}{N(v')+\varepsilon}] + C\sqrt{\frac{\ln N(v)}{N(v')+\varepsilon}}$
end function

function Expand($v$)
    add child $v'$ to $v$
    choose $a \in A_{untried}(v)$ uniformly at random
    $\boldsymbol{\psi} \leftarrow$ features corresponding to $s(v)$, $a$
    $s(v') \leftarrow f(s(v), a)$
    $R, R_e \leftarrow$ GetRewards($s(v), s(v')$)
    $a(v') \leftarrow a$
    $\boldsymbol{e}(v') \leftarrow \gamma\lambda\boldsymbol{e}(v) + \boldsymbol{\psi} - \alpha\gamma\lambda(\boldsymbol{e}(v)^\top \boldsymbol{\psi})\boldsymbol{\psi}$
    $G(v') \leftarrow G(v) + R_e; D(v') \leftarrow D(v) + 1$
    $N(v') \leftarrow 0; X(v') \leftarrow 0$
    return $v'$
end function

function Backup($v, Q$)
    while $v$ is not null do
        $N(v) \leftarrow N(v) + 1$
        $X(v) \leftarrow X(v) + Q$
        $Q \leftarrow \gamma Q$
        $v \leftarrow$ parent of $v$
    end while
end function
```

Algorithm 4 TD-MCTS algorithm part 2.

```
function Rollout($v$)
    $\boldsymbol{e} \leftarrow \boldsymbol{e}(v); Q_{old} \leftarrow 0; Q_s \leftarrow 0; t \leftarrow 0$
    $D \leftarrow D(v)$
    $s \leftarrow s(v)$
    $a \leftarrow$ EpsilonGreedy($s, \boldsymbol{\theta}_S$)
    $\boldsymbol{\psi} \leftarrow$ features corresponding to $s$, $a$
    while $s$ is non-terminal and $D < D_{max}$ do
        $s' \leftarrow f(s, a)$
        $D \leftarrow D + 1$
        $R, R_e \leftarrow$ GetRewards($s, s'$)
        $Q_s \leftarrow Q_s + \gamma^t R_e$
        $a' \leftarrow$ EpsilonGreedy($s', \boldsymbol{\theta}_S$)
        $\boldsymbol{\psi}' \leftarrow$ features corresponding to $s'$, $a'$
        $Q \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}$
        $Q' \leftarrow \boldsymbol{\theta}^\top \boldsymbol{\psi}'$
        $\delta \leftarrow R + \gamma Q' - Q$
        $\boldsymbol{e} \leftarrow \gamma\lambda\boldsymbol{e} + \boldsymbol{\psi} - \alpha\gamma\lambda(\boldsymbol{e}^\top \boldsymbol{\psi})\boldsymbol{\psi}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\delta + Q - Q_{old})\boldsymbol{e} - \alpha(Q - Q_{old})\boldsymbol{\psi}$
        $Q_{old} \leftarrow Q'; \boldsymbol{\psi} \leftarrow \boldsymbol{\psi}'; a \leftarrow a'; s \leftarrow s'; t \leftarrow t + 1$
    end while
    if $s$ is non-terminal then
        $Q_s \leftarrow Q_s + \gamma^t \max_{a' \in \text{ A}} \boldsymbol{\theta}_S^\top \boldsymbol{\psi}(s, a')$
    end if
    return $Q_s$
end function

function EpsilonGreedy($s, \boldsymbol{\theta}$)
    sample $\rho$ uniformly at random in $[0, 1]$
    if $\rho > \epsilon$ then
        return $\operatorname*{argmax}_{a' \in \text{A}} \boldsymbol{\theta}^\top \boldsymbol{\psi}(s, a')$
    else
        return choose $a \in$ A uniformly at random
    end if
end function

function GetRewards($s, s'$)
    return GetScores($s'$) $-$ GetScores($s$)
end function

function GetScores($s$)
    $P \leftarrow GameScore$ of state $s$
    $P_e \leftarrow P$
    if $s$ is terminal then
        if game is won then
            $P_e \leftarrow P_e + HIGH\_VALUE$
        else
            $P_e \leftarrow P_e - HIGH\_VALUE$
        end if
    end if
    return $P, P_e$
end function
```

VII. Conclusions

This paper introduced a new hybrid method, combining true online Sarsa(λ) and MCTS in the GVGAI domain. By applying a generalizable feature extraction based on proximities between item types to game states, a learning procedure is embedded in the rollouts of MCTS to perform state value evaluations using a linear function approximation, and biasing these rollouts ultimately. Through the modifications proposed in this study, MCTS was successfully made to take advantage of domain knowledge it encounters using it as basic heuristic in its decisions. This algorithm is then tested and compared with the vanilla MCTS over the GVGAI framework set 1 of 10 games with 100 runs for each, resulting in 5000 plays in total. The results show that the proposed approach provides significant improvements over vanilla MCTS in the majority of the games, even in the case with only half the number of rollouts. Additionally, following the self-generated heuristics makes the proposed method perform slightly worse than vanilla MCTS in some of the games where the victory conditions are not related with the exploited features, as random search yields better results than a misleading heuristic.

There are some possible extensions to this study. Firstly, there are many parameters (α, λ, γ, ϵ) to tune in the learning portion of the algorithm. Considering the time required to evaluate a set of parameters (by playing every game in a set multiple times) and the fact that each one of these parameters affect how the other parameters behave, it is a challenging task to find the actual best parameter set. A game-based study on these parameters would be a valuable work. Moreover, developing a game-based adaptive parameter tuning could increase the performance further. Secondly, as experiments show, how the algorithm performs is closely related to the feature representation method (which was a vector of highest proximities between every item type pair in the game in this study). Extending this representation to include extra information such as resources, path planning data with different distance metrics would be an interesting direction to take.

It was seen that the proposed approach incorporating an embedded online temporal-difference learning algorithm is as viable as the previously proposed major modifications for MCTS in the GVGAI *planning track* challenge and is worthy to be further studied. Furthermore, with the addition of the new *learning track* challenge in GVGAI, temporal-difference learning techniques are clearly going to get much more attention in the near future.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015.

[3] M. Genesereth and N. Love, "General game playing: Overview of the AAAI competition," *AI Magazine*, vol. 26, pp. 62–72, 2005.

[4] T. Schaul, "A video game description language for model-based or interactive learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*. Niagara Falls: IEEE Press, 2013.

[5] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C. U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, Sept 2016.

[6] D. P. Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General video game AI: Competition, challenges and opportunities," in *AAAI*, D. Schuurmans and M. P. Wellman, Eds. AAAI Press, 2016, pp. 4335–4337.

[7] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, S. Tavener, D. Perez, S. Samothrakis, S. Colton, and et al., "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI*, 2012.

[8] B. Ross, "General video game playing with goal orientation," Master's thesis, University of Strathclyde, 2014.

[9] C. Y. Chu, H. Hashizume, Z. Guo, T. Harada, and R. Thawonmas, "Combining pathfinding algorithm with knowledge-based monte-carlo tree search in general video game playing," in *2015 IEEE Conf. on Computational Intelligence and Games*. IEEE, 2015, pp. 523–529.

[10] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying MCTS for human-like general video game playing," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16. AAAI Press, 2016, pp. 2514–2520.

[11] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius, "Investigating MCTS modifications in general video game playing," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 107–113.

[12] H. Park, H. Kim, and K. Kim, "GreedyUCB1 based monte-carlo tree search for general video game playing artificial intelligence," *KIISE Transactions on Computing Practices*, vol. 21, no. 8, pp. 572–577, 2015.

[13] D. J. N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands, "Enhancements for real-time monte-carlo tree search in general video game playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–8.

[14] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary MCTS for general video game playing," in *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 2014, pp. 1–8.

[15] I. Bravi, A. Khalifa, C. Holmgård, and J. Togelius, "Evolving UCT alternatives for general video game playing," in *The IJCAI-16 Workshop on General Game Playing*, 2016, p. 63.

[16] H. Horn, V. Volz, D. Pérez-Liébana, and M. Preuss, "MCTS/EA hybrid GVGAI players and game difficulty estimation," in *2016 IEEE Conference on Computational Intelligence and Games*, Sept 2016, pp. 1–8.

[17] M. de Waard, "Monte carlo tree search with options for general video game playing," Master's thesis, Universiteit van Amsterdam, 2016.

[18] D. Perez-Liebana, S. Mostaghim, and S. M. Lucas, "Multi-objective tree search approaches for general video game playing," in *Evolutionary Computation, 2016 IEEE Congress on*. IEEE, 2016, pp. 624–631.

[19] A. Deleva, "TD learning in monte carlo tree search," Master's thesis, University of Ljubljana, 2015.

[20] C. Y. Chu, S. Ito, T. Harada, and R. Thawonmas, "Position-based reinforcement learning biased MCTS for general video game playing," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–8.

[21] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game AI," in *AIIDE*, C. Darken and M. Mateas, Eds. The AAAI Press, 2008.

[22] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proceedings of the 17th European Conference on Machine Learning*, ser. ECML'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 282–293.

[23] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2-3, pp. 235–256, May 2002.

[24] R. S. Sutton, "Learning to predict by the methods of temporal differences," in *Machine Learning*. Kluwer Academic Publishers, 1988, pp. 9–44.

[25] H. van Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, "True online temporal-difference learning," *Journal of Machine Learning Research*, vol. 17, no. 145, pp. 1–40, 2016.