Redundant Action Avoidance and Non-Defeat Policy in the Monte Carlo Tree Search Algorithm for General Video Game Playing

Eduardo Hauck dos Santos[1]* Heder Soares Bernardino[1]

[1]UFJF, Dep. Ciencia da Computacao, Brazil

ABSTRACT

General Video Game Playing (GVGP) proposes as a challenge the development of agents capable of playing real-time video games with previously unknown rules. In recent editions of the General Video Game Playing Competition (GVG-AI), the Monte-Carlo tree search (MCTS) algorithm has been one of the most popular and successful techniques to power such agents. While it achieves good results when compared to other techniques, it carries some weaknesses that can harm its performance in a GVGP environment. This paper introduces the Redundant Action Avoidance (RAA) method to identify redundant actions at the start of a game and to improve Monte Carlo simulations by avoiding sequences of such actions. We also propose a modification to the recommendation policy of the MCTS, the Non-Defeat Policy (NDP) in order to avoid defeats in certain cases where the player is surrounded by multiple dangers. Results from experiments show that both methods are able to improve the general performance of a MCTS-based controller, and that exploring the idea of redundant actions in Monte Carlo simulations can be beneficial in GVGP.

Keywords: General Video Game Playing, Monte Carlo tree search, Artificial Intelligence.

1 INTRODUCTION

Video games have been widely used in academy and industry for evaluating artificial intelligence (AI) techniques. The goal of AI is to create programs that can demonstrate intelligent action such as of the human being, and video games provide a rich platform for testing these programs, allowing the simulation of many events and situations present in real life.

General Video Game Playing (GVGP) proposes the challenge of creating computer programs that can play a wide range of video games with previously unknown rules. The idea of having an intelligent agent that can solve many different challenges in video games, similar to a human being, has been previously associated with an approximation of a General Artificial Intelligence [9].

The GVG-AI competition is the most recent effort to foment research in this area. Rather than to concentrate on the parsing of environment information, the framework allows a bigger focus on the AI part of the controller by providing robust information about the game, such as the player position, position of nearby elements, and the ability to simulate the execution of actions.

One of the most popular algorithms in the GVG-AI competitions is the Monte Carlo Tree Search (MCTS). This algorithm achieved great success when it was first proposed for the board game Go [4]. Likewise, MCTS and its variants usually achieve high ranking positions in GVG-AI [9]. Through expanding a search tree based on a sampling of the search space, the Monte Carlo Tree Search algorithm tries to find the most promising move from the current state.

*e-mail: eduardohauck@gmail.com

The main contribution of this paper is proposing a strategy to identify redundant actions and bias the rollouts performed during the simulation phase of the MCTS algorithm in GVGP. The idea of exploring the redundancy aspect of simulations in GVGP has been previously tackled by using novelty tests to prune expanded redundant states [12] or by applying a penalty to the reward function after executing opposing moves [10]. Here, we define the concept of redundant actions, propose an algorithm to identify such actions, and a method to avoid selecting these actions in the simulation phase of the MCTS, in order to reduce the number of redundant states generated and to improve the overall quality of simulations. This method can be generally applied to every game in the GVG-AI framework to identify and avoid redundant actions during play-outs.

Additionally, we introduce a modification to the recommendation policy of the MCTS algorithm, the Non-Defeat Policy. During the simulation phase, each child node from the root that has seen a defeat state is marked as a defeat node. When executing the recommendation policy, an action that leads to a defeat node is never selected, unless it's the only option. This modification allows the MCTS to take actions that postpone or avoid a defeat state in certain games.

2 REATED RESEARCH

2.1 General Video Game Playing

General Video Game Playing concerns the study of AIs that can play video games with previously unknown rules. As opposed to the traditional board game AIs, video games bring new challenges to the researchers such as a tight computational time budget and new premises such as the possibility of simulating real life events on a virtual environment.

The GVG-AI competition is currently the most prominent competition in GVGP, and offers a framework that allows the development and testing of controllers for 2D arcade style video games. Besides offering a huge library of original and reproduction titles, it also allows the development of new games using a Video Game Description Language (VGDL). For more details on the VGDL and its possibilities, the reader is refered to [11].

The framework also allows the simulation of actions, enabling the controller to see what are the possible outcomes (states) from a given sequence of actions. The *forward model* allows simulations to be executed as many times as needed within a time step. In the Planning track, the main track of the GVG-AI competition, the controller has to use the *forward model* to build up a plan and decide which action to perform at each time step.

2.2 Monte Carlo Tree Search

Monte Carlo tree search (MCTS) is a tree search algorithm first introduced in 2006 in [3, 4, 7] for the board game Go. It highly improved the performance of the agent when compared to other state-of-the-art techniques at the time. Since then, MCTS has been expanded to other games and domains, and turned into one of the most popular algorithms for General Video Game Playing. In fact, in recent GVG-AI competitions, the controllers with the highest ranking positions on the Planning Track are often based on some variation of the original MCTS [9].

In the MCTS algorithm, an iterative tree search is performed, where each node stores statistical information about play-outs made from that node. Chaslot [2] talk over the application of this technique in game AI, and describes the four basic steps of a MCTS-based algorithm:

- Selection: a selection strategy is applied from the root node until a leaf node is reached. In GVGP, the Upper Confidence bounds applied to Trees (UCT) [7] is often used as the selection strategy.
- Expansion: a new node (action transition) is appended to the tree, unless the game has ended in the current node.
- Simulation: from the newly expanded node, a simulation is executed, performing a number of actions according to a simulation strategy until reaching a specified depth. Due to the cost of biasing the simulation with domain knowledge, a random simulation strategy is often employed in GVGP.
- Backpropagation: a reward associated with the final state of the simulation is backpropagated to all visited nodes. The number of visits of each node is also incremented.

At the end of execution, in GVGP usually given by the end of the available computational time, the action to be taken by the controller is obtained through the best child from the root node. The recommendation policy is usually given by the number of visits (Robust child) or the highest score associated with a node (Max child), or a combination of the two values [1]. For MCTS in GVGP, a common recommendation policy is the Robust child one.

Even though the Monte Carlo tree search has turned into one of the most popular and successful algorithms in GVGP, the original form of the algorithm yields only about 20% of victories [5]. Moreover, it has some issues [8] that can be aggravating on a GVGP environment, such as non-informative and redundant simulations.

3 Proposed Methods

This section describes the methods proposed and integrated to MCTS in this paper. The first method addresses the problem of redundancy in the simulation strategy, while the second modify the recommendation policy to improve performance in corner cases.

3.1 Redundant Action Avoidance

Considering that games on a GVGP environment have previously unknown rules and the fact that we can not make any assumptions about what is the goal of a certain game beforehand, a random simulation policy has been adopted as a common simulation strategy for the MCTS algorithm [5]. Moreover, a complex simulation strategy can increase the computational cost of the simulation, reducing the number of iterations of the algorithm and decreasing the general performance of the controller. On the other hand, it is clear that performing random actions produces a lot of redundant actions (e.g. moving left and right), which can increase the number of uninformative simulations.

Therefore, it becomes interesting to develop a policy to reduce the occurrence of redundant actions, however, without completely eliminating them, since there could exist situations in that executing said actions could be useful (e.g. avoiding many projectiles).

First, redundant actions are defined. A sequence of actions-state (a_1,s_1), (a_2,s_2) is redundant if and only if the execution of both actions from a given state s_1 does not result in a change of the state of the avatar. A change in the state of the avatar can be given by the position, orientation, properties of the avatar, or execution of a certain action. Therefore, two actions a_1, a_2 can be redundant in 4 situtations:

- Position: when the action a_1 causes a change in the avatar from the position p_0 to the position p_1 and the action a_2 causes a new change of position to p_0.
- Orientation: when the action a_1 causes a change in the orientation of the avatar and the action a_2 causes a new change in orientation.
- Properties: when the action a_1 causes a change in the properties value p_0 of the avatar (such as a change in the currently drawn weapon) to value p_1 and the action a_2 causes a new change of property value to p_0.
- Execution: when the action a_1 produces a new object from the avatar in the game (e.g. a shot fired, a sword swinging, etc) and the action a_2 is null (does not cause any change to position, orientation, properties or change in the number/state of created objects from the avatar).

An illustration of two play-outs, one with redundancy based on movement and one without redundancy is given by Figure 1.

Two things are important to note. First, the proposed definition is established for the discrete, grid-based physics adopted in the GVG-AI framework. Second, this definition deals independently with the state of the avatar. We do not compute interactions with the other elements in the game, or the general state of the game itself. In fact, if any kind of interaction (i.e. collision) is detected during action testing, the search for a subsequent redundant action is halted. This decision is based on the high computational cost of tracking the state of too many elements in the game. As observed previously, increasing the computational cost of the simulations can significantly reduce the number of simulations performed during each step of the game.

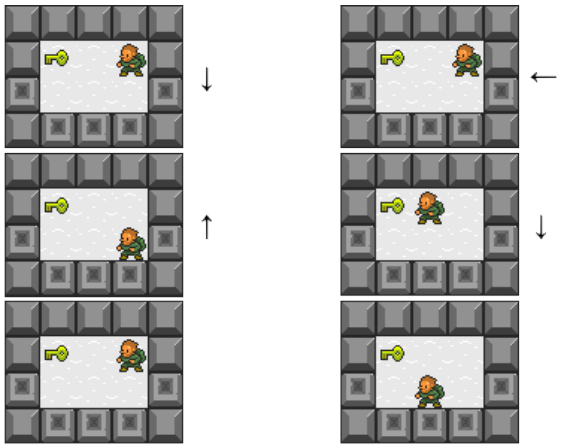

Figure 1: Two play-outs of the same game. The arrows indicate the action performed from the current state. In (a), one redundant move is executed in the second frame. In (b), a similar play-out but without any redundant action.

In this work we tackle movement, properties and execution actions. These are the less expensive actions to identify and to bias during simulation, i.e., it is possible to know which actions are redundant just by checking the previous step. For orientation actions, at least the two last actions would need to be checked in order to do so, due to the fact that movement and orientation are usually changed with the same key presses. This problem will be addressed in a future work.

Initially, we define a uniform probability distribution to every available action transition as

$$P(a,s) = \frac{1}{n}, \quad (1)$$

where $P(a,s)$ is the probability of choosing a certain action a from state s and n is the number of available actions in the game. Given that a certain action a is chosen for the state s, and there is an action a' so that a' is a redundant action of a, the probability of choosing the action a' for the state s' is given by

$$P(a',s') = \frac{1}{n}(1-p), \quad (2)$$

where p is a penalty coefficient. The penalized value is redistributed among the remaining actions so that it sums 1. Here, as we have only one redundant action for each action, the probability adopted for the non-redundant actions is given by

$$P(a'',s') = \frac{1}{n}\left(1+\frac{p}{n-1}\right), \quad (3)$$

where a'' represents a non-redundant action of a. The uniform distribution is maintained when no redundant actions are identified for a certain game.

If the initial position of the avatar is surrounded by obstacles, it will not be possible to identify some or any redundant actions for the game. Regardless of the policy adopted to identify these actions in-game, it is not possible to provide any guarantee about the possibility of finding all redundant actions for a given match. In practice, the proposed solution was able to identify redundant actions for most of the games tested.

3.2 Non-Defeat Policy

The MCTS algorithm tries to find the best action to take from the current state by performing random simulations of d depth and backpropagating the reward for the state given at this depth. However, in GVGP there are situations in which a precise sequence of keys must be pressed in order to avoid an instantaneous loss.

When this type of situation occurs during the MCTS execution and none of the random simulations were able to find at least one sequence of actions that would avoid a defeat in the game, all children nodes from the root node will likely have the same reward value and action selection will be random.

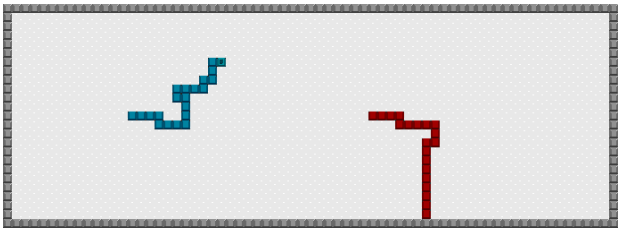

Figure 2: Illustration of a situation where MCTS (player red) often fails at finding the best action and end up hitting the wall.

We propose a simple method to avoid those instant death moments, the Non-Defeat Policy. Whenever the selection phase takes place, and a simulation is made from the root node to one of their children, we check the state of the children and, if it is a state where the player lost the game, we mark it as a defeat node. This verification is repeated at the beginning of every simulation in the game.

At the end of execution, the Non-Defeat Policy changes the default recommendation policy in order to consider only nodes that have not been marked as defeat nodes. The choice from the recommendation policy remains unchanged when all children nodes are defeat nodes. This strategy should avoid most of the simple situations where MCTS is defeated, such as the one depicted in Figure 2.

4 Computational Experiments

4.1 Setup

The *vanilla* MCTS controller provided by the GVG-AI competition was adopted as the baseline for the experiments. The proposed methods were incorporated into the *vanilla* MCTS, and 5 different configurations were tested: four controllers employing the Redundant Action Avoidance method with penalty coefficients 0.4, 0.5, 0.6 and 1, and one controller employing the Redundant Action Avoidance method with penalty coefficient 0.5 and the Non-Defeat Policy. The MCTS parameters used for all configurations are the same as of the *vanilla* MCTS of the GVG-AI framework. A detailed description of the parameters used follows.

The UCT was adopted as selection strategy, with constant $C = 2$ and rewards normalized between [0,1]. The simulation depth was set to 10. A random simulation strategy was adopted, with the exception of the configurations using the Redundant Action Avoidance method, in which case the probability of performing a certain action during a simulation is given by Equation 1, Equation 2 or by Equation 3, according to the state of the game. The robust child policy was used as the recommendation policy, with the exception of the configuration using the Non-Defeat Policy, in which case the child with the highest number of visits is chosen only if it is not a defeat node. An open loop approach was used for the algorithm, with only statistics of the states being stored in each node.

To ensure enough diversity in the set of games used in this experiment, the same subset of games from [6] was adopted, which combines games from two different classification methods proposed for the games in the GVG-AI framework. Table 1 shows the selected games for this experiment.

Id	Name	Type	Id	Name	Type
0	Aliens	S	4	Bait	D
13	Butterflies	S	15	Camel Race	D
22	Chopper	S	18	Chase	D
25	Crossfire	S	36	Escape	D
29	Digdug	S	46	Hungry Birds	D
49	Infection	S	58	Lemmings	D
50	Intersection	S	60	Missile Command	D
75	Roguelike	S	61	Modality	D
77	Seaquest	S	67	Plaque Attack	D
84	Survive Zombies	S	91	Wait for breakfast	D

Table 1: Indexes, names and types of games chosen for this experiment. Legend: D - Deterministic, S - Stochastic

Each game was run 20 times for all of its 5 levels, totaling 100 runs for each game for each controller. The number of victories was used to compare their performance. The budget used for planning at each step followed the GVG-AI competition rules: 40ms of allowed time to plan. If at some step the controller takes from 40ms to 50ms, its action for that step is set to null, and if it takes more than 50ms, the game is halted and the controller disqualified.

4.2 Results

The percentage of victories per game can be seen in Table 2. The subject with $p = 0$, equivalent to the vanilla MCTS, already has 100% victories for games Alien (Id 0) and Intersection (Id 50). The modifications proposed, over all parameters, were able to keep this percentage of victories, indicating that we did not harm the performance for these games. Four other games deserve special attention: Digdug (Id 29), Escape (Id 36), Lemmings (Id 58), and Roguelike (Id 75). These are all games that offer a varying of different

hard challenges for the vanilla MCTS. None of the proposed modifications were able to increase the performance obtained in these games, but since the modifications do no address these challenges, this result was expected.

Game	p = 0	p = 0.4	p = 0.5	p = 0.5+NDP	p = 0.6	p = 1
0	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
4	8%	**10%**	**10%**	**10%**	7%	6%
13	97%	95%	**99%**	98%	97%	**99%**
15	6%	**7%**	4%	**7%**	4%	3%
18	7%	**11%**	8%	6%	5%	8%
22	17%	21%	14%	22%	22%	**23%**
25	2%	**5%**	3%	**5%**	3%	4%
29	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
36	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
46	5%	**6%**	**6%**	5%	4%	5%
49	95%	**98%**	96%	**98%**	96%	94%
50	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
58	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
60	61%	65%	**67%**	62%	66%	64%
61	27%	26%	24%	27%	**29%**	25%
67	89%	86%	89%	91%	91%	**93%**
75	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
77	58%	58%	**59%**	58%	49%	56%
84	42%	42%	37%	**47%**	43%	46%
91	9%	17%	11%	12%	**19%**	6%
# bests	6	**12**	11	11	8	9
# better/equals $p = 0$	17	16	**19**	15	14	

Table 2: Percentage of victories for each game, for each configuration tested. The first one (p = 0) is equivalent to the vanilla MCTS.

Looking at the percentage of victories for the modifications proposed, it is possible to see that there were improvements for every parameter tested. The highest percentage of victories is observed when p = 0.4 or p = 0.5. However, the one with the highest improvement over the vanilla MCTS was the subject with p = 0.5 combined with the Non-Defeat Policy, getting an equal or better performance in 19 of the 20 games tested. Configurations of p = 0.6 and p = 1 presented good results nevertheless, but indicates that higher values of penalty do not increase the overall performance of the Redundant Action Avoidance method.

In a general sense, the Redundant Action Avoidance method tend to improve the performance of the MCTS in most games, as most of them have multiple redundant actions and therefore tend to perform many uninformative simulations during execution. There is no guarantee, however, that avoiding these kind of actions will actually lead to better results. In some games, performing redundant actions can be useful. In practice, the results show that the performance can be improved with the use of this kind of method.

The games in which the Non-Defeat Policy yield an improvement (Ids 4, 25, 49 and 84, with the exception of 15), are all games in which the avatar can be surrounded by multiple dangers and that the vanilla MCTS may fail to choose an action that does not result in a game over. The strategy proposed can postpone or avoid the defeated states in those situations.

With the exception of the games that result in 100% or 0% of victories, at least one of the configurations tested was able to increase the performance in comparison with the vanilla MCTS. It is important to note that due to the stochasticity of execution, small variations on the percentage of victories can occur at each execution. That is why, in order to mitigate this effect, the games were run 100 times for each controller.

5 Concluding Remarks and Future Works

This paper introduced the definition of redundant actions in General Video Game Playing, and proposed an algorithm to identify these actions for a given game and a method to bias the rollouts in the simulation phase of the MCTS algorithm in order to reduce the number of redundant actions and improve the general quality of simulations. A modification to the recommendation policy of the MCTS is also proposed in order to avoid some defeat states.

Results show that the general performance of a MCTS-based controller can be improved by applying a penalty to the occurrence of redundant action. The proposed recommendation policy was also able to increase the performance over certain games.

The Redundant Action Avoidance method is a first step in an effort to improve the quality of simulations in GVGP by biasing the path taken during the simulation using the concept of redundant actions. There are many improvements that can be added to this method. First, the identification of redundant action always fail when there are obstacles adjacent to the avatar spawn point. A strategy could be developed to move the avatar around until there are no obstacles nearby, in order to properly identify all redundant actions. Second, the changes in orientation could be integrated in the algorithm. In board-physics games like the ones tested in the GVG-AI environment, this might not be a frequent problem, however, for the general case, it is fundamental to integrate orientation actions into redundancy avoidance. Finally, the proposed strategy can be tested in other GVGP environments. The adaptation of this method to a 3D environment is also a challenge proposed as future work.

GVGP has shown to be a hard problem, with factors such as time limitation and heuristic independent algorithms. The research and publications in the field has increased with the development of the GVG-AI framework, and different algorithms and strategies have been tested in this environment. This work can be a first step towards a strategy that can be integrated into different algorithms in order to improve the general quality of techniques in GVGP.

References

[1] G. Chaslot. Monte-carlo tree search. *Maastricht: Universiteit Maastricht*, 2010.

[2] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.

[3] G. Chaslot, J.-T. Saito, B. Bouzy, J. Uiterwijk, and H. J. Van Den Herik. Monte-carlo strategies for computer go. In *Proc. of the BeNeLux Conference on Artificial Intelligence*, pages 83–91, 2006.

[4] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2006.

[5] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius. Investigating mcts modifications in general video game playing. In *Conf. on Compt. Intelligence and Games (CIG)*, pages 107–113. IEEE, 2015.

[6] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana. Analysis of vanilla rolling horizon evolution parameters in general video game playing. In *European Conference on the Applications of Evolutionary Computation*, pages 418–434. Springer, 2017.

[7] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *European Conf. on Mach. Learning*, pages 282–293. Springer, 2006.

[8] D. Perez, S. Samothrakis, and S. Lucas. Knowledge-based fast evolutionary mcts for general video game playing. In *Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2014.

[9] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. 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*, 2015A.

[10] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas. Open loop search for general video game playing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 337–344. ACM, 2015B.

[11] T. Schaul. A video game description language for model-based or interactive learning. In *Conference on Computational Intelligence in Games (CIG)*, pages 1–8. IEEE, 2013.

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