Monte Carlo Tree Search Based Algorithms for Dynamic Difficulty Adjustment

Simon Demediuk*, Marco Tamassia*, William L. Raffe†, Fabio Zambetta*, Xiaodong Li*
and Florian "Floyd" Mueller‡
*School of Science - ‡ School of Media and Communication
Royal Melbourne Institute of Technology RMIT, Melbourne, Australia
Email: firstname.secondname@rmit.edu.au
†Games Studio, Faculty of Engineering and IT
University of Technology UTS, Sydney, Australia
Email: william.raffe@uts.edu.au

Abstract*—Maintaining player immersion is a crucial step in making an enjoyable video game. One aspect of player immersion is the level of challenge the game presents to the player. To avoid a mismatch between a player's skill and the challenge of a game, which can result from traditional manual difficulty selection mechanisms (e.g. easy, medium, hard), Dynamic Difficulty Adjustment (DDA) has previously been proposed as a means of automatically detecting a player's skill and adjusting the level of challenge the game presents accordingly. This work contributes to the field of DDA by proposing a novel approach to artificially intelligent agents for opponent control. Specifically, we propose four new DDA Artificially Intelligent (AI) agents: Reactive Outcome Sensitive Action Selection (Reactive OSAS), Proactive OSAS, and their "True" variants. These agents provide the player with an level of difficulty tailored to their skill in real-time by altering the action selection policy and the heuristic playout evaluation of Monte Carlo Tree Search. The DDA AI agents are tested within the FightingICE engine, which has been used in the past as an environment for AI agent competitions. The results of the experiments against other AI agents and human players show that these novel DDA AI agents can adjust the level of difficulty in real-time, by targeting a zero health difference as the outcome of the fighting game. This work also demonstrates the trade-off existing between targeting the outcome exactly (Reactive OSAS) and introducing proactive behaviour (i.e., the DDA AI agent fights even if the health difference is zero) to increase the agents believability (Proactive OSAS).

I. INTRODUCTION

The concept of flow in games has been argued to be crucial in ensuring the player is immersed in the game. Cowley et al. [1] have mapped Csikszentmihalyi's *Flow theory* [2] to video games for the purpose of showing that a player will continue to play a game that they are immersed in. To ensure that players remain immersed in a particular video game, the level of challenge presented to the player by the game needs to be based on the player's skill level in that game. If the level of challenge is appropriate for the individual player, the player is more engaged. When the game becomes too easy or too hard, the player may become frustrated or disengaged [3]. This process involves changing the strategies and behaviour of the AI opponent or environment to match the skill level of the player. This produces a video game that is more enjoyable and long-lived, since the game can alter its level of challenge as the skill level of the player progresses.

Sweetser et al. [4] developed a model for evaluating games called GameFlow, derived from *Flow Theory*. It was used to predict the success of a game, by evaluating them in the following topic areas: concentration, challenge, player skill, control, clear goals, feedback, immersion and social interaction. Although the work done by Sweetser et al. separates the concept of challenge from the concept of immersion, definitions of immersion found in [5] and [6] include the level of challenge faced by the player as part of what makes a game immersive.

The effect of challenge on enjoyment is supported by work done by Yannakakis et. al. [7], which measured empirically that players found a game's entertainment value to be high when an opponent in the game is at an appropriate level. In early work conducted by Malone [8], it is suggested that player-vs-player competition is motivating simply because it provides a challenge at an appropriate difficulty level. In these competitive games, ranking systems such as Chess Elo [9] and Microsoft's TrueSkill [10] attempt to match players for an appropriate level of challenge based on their skill rank. Players that have a similar skill rank are said to be evenly matched, as each player has a 50% chance of winning the game in question.

For single player video games, the level of challenge is generally the first question asked of the player, in the form of setting the difficulty (e.g easy, medium or hard). These static difficulty settings are used by developers to group players into a limited number of skill ranges. This can be problematic in cases where a player is unaware of their skill level for the game in question. Additionally, as a player's skill level changes through play, since the difficulty remains static, a gap between the players skill and the difficulty of the game can form. Therefore, a mechanism is desirable that can modify the difficulty of the artificial opponents to provide an appropriate level of challenge based on the individual skill level of each player [11].

Tailoring the difficulty of a game to each player is a non-trivial task. Dynamic Difficulty Adjustment (DDA) [12] has

978-1-5386-3233-8/17/$31.00 ©2017 IEEE

been applied in various forms in different game genres in order to address this issue. DDA systems aim to vary the level of difficulty of a game according to the current player's skill level, both at the beginning and during game play. The level of difficulty that a DDA system provides is not one of the fixed levels (easy, medium or hard) but rather a custom difficulty level that matches the players current skill. The difficulty that is appropriate for a player, is one in which the player has a 50% chance to either win or lose the game. As the players skill level changes so does the level of the DDA system, such that the level of difficulty perceived by the player is constant. There has been a focus in the literature on tailoring a game's difficulty to the level of the player through the use of Artificial Intelligence agents [11], [13]–[15] or Artificial Intelligent controlled game environments [12], [16].

This paper presents a number of improvements to the current state of the art DDA systems through the development of a DDA Artificially Intelligent (AI) agent which implements Monte Carlo Tree Search (MCTS) [17] with novel heuristic playout evaluation functions and action selection criteria. Through modifying the way in which Monte Carlo Tree Search selects the next action, our DDA AI agents are able to vary the level of difficulty of the Artificial Intelligent opponent in real-time to provide an appropriate level of difficulty to the individual player. This work uses the FightingICE engine as a test-bed for our agents; the engine as been used in recent years for AI agent competitions. Unlike the main stream of the competition, which aim at producing a winning bot, our agents aim at finishing games with the smallest possible health points gap. We conduct experiments in the FightingICE 2D real-time fighting game against both artificial and human opponents to demonstrate the ability of the agents to change the level of difficulty to match that of the current opponent.

This paper is structured as follows: Section II gives a brief background on traditional MCTS algorithms. Section III introduces Reactive Outcome Sensitive Action Selection (Reactive OSAS, or ROSAS) and Proactive OSAS (POSAS), presenting the modifications to the action selection policy and playout evaluation heuristics to MCTS. Section IV describes the experiments conducted against other automated bots and human players as well as the results of these experiments, while Sections V and VI give a final discussion of these results and their implications for future work.

II. BACKGROUND

Monte Carlo Tree Search (MCTS) is a tree-based search algorithm used in decision processes [17]. MCTS iteratively builds a search tree where each edge represents an action. Note that nodes do not encode states; rather, they encode the sequence of actions from the root to the node itself.

Unlike other search algorithms such as Minimax, MCTS does not build the entire tree, nor does it build it uniformly. Instead, the search is usually interrupted at a predefined depth, and a "playout" is used to assess the quality of the solution at the leaf node. A playout is a simulation of the process at hand starting from the current state, which is represented by the root

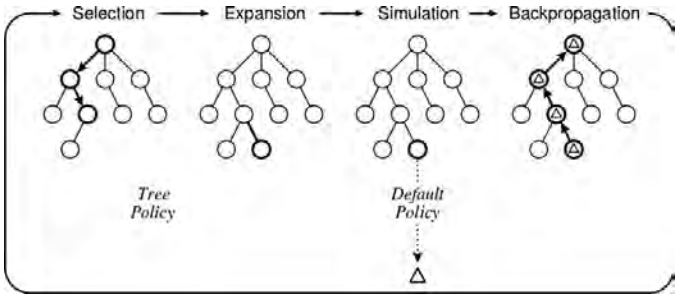

Fig. 1. One iteration of the general MCTS approach (from Browne et. al. [17])

node. In the simulation, the agent executes first the actions in the path root–leaf, and then a sequence of random actions. The outcome of the simulations is used as a heuristic to assess the quality of the leaf node. Notice that, if the playout is run until the end of the process, this mechanism does not require a heuristic designed through expert knowledge to evaluate the outcome of the playout: instead, the actual outcome of the process can be used (e.g., final score, or victory/loss). If the playout is interrupted before the end of the process, as we do in this work, a simple heuristic can be used.

To build the tree, MCTS repeatedly iterates through 4 phases (see Figure 1), until the time allotted for the computation expires. The first of these phases is "selection", where the tree is descended starting from the root node, selecting the most promising nodes, until a leaf node is reached. The second phase is "expansion", where a new child of the leaf node is created (if the leaf is within the depth limit) and descended into. The third phase is "simulation", where the actions of the path root–leaf are executed in the simulator, followed by a sequence of random actions; this is called a playout. The fourth and final phase is "back-propagation", where the result of the playout is used to update the information of the nodes that were traversed.

The selection phase is particularly critical to the performance of MCTS. Due to the stochastic nature of playouts, the scores found are bound to be noisy. Consequently, at the early stages of the exploration of a node, the score of its children are not fully reliable. This, in turn, can affect the remainder of the exploration if the exploration follows the highest value. To address this, Upper Confidence Bound (UCB1) [18] applied to Trees (UCT) [19] drives exploration balancing the score of a node with the number of times it has been visited compared to its siblings. Specifically, the formula used is:

$$\mathrm{UCT} = \overline{X}_j + 2C_p\sqrt{\frac{\ln n}{n_j}}, \tag{1}$$

where $\overline{X}_j$ is the average reward from action j, n_j is the number of times action j was tried at the node, n is the sum of n_j for all actions and $C_P > 0$ is a constant.

The four phases can be repeated as many times as the time-budget allows. After the tree is built, a child action of the root node is selected according to some technique. The most

popular is to choose the action that was visited most often. This strategy seems to be at least as effective as choosing the action with the highest average score. The techniques that we propose in this paper are variations of MCTS in that they use either a different action-selection strategy or a different heuristic to evaluate the results of playouts. These changes have dramatic effects on the behaviour of an agent driven by MCTS.

III. DYNAMIC DIFFICULTY ADJUSTMENT AI AGENTS

MCTS has been used to develop DDA AI agents in the past by Hao et. al. [20]. Their methodology restricts the time allotted for building the search tree by using a pre-calculated formula that best achieves the desired outcome in their experiments. This approach has several limitations: it requires the development of a different equation for each game type, and is more time consuming and complex to calculate when compared to our method of changing the MCTS action selection policy or playout evaluation heuristic. Another significant limitation of their implementation is its dependence on the computational power of the hardware that runs MCTS. Finally, the approach is not principled, and as such is prone to producing unbalanced behaviours. In fact, in some advantageous states, this technique may find a strong action even under tight computational constraints; similarly, in disadvantageous states, the technique may struggle to find a strong enough action even under generous computational constraints. This is because the formula used is not dependent on the starting state, and as such does not differentiate between advantageous and disadvantageous ones.

Another approach to DDA that is related to ours is Challenge Sensitive Action Selection (CSAS), proposed by Andrade et al. [21]. CSAS employs action values to determine the action to use, and uses Q-learning [22] to estimate such values. At play-time, after Q-learning has been trained, given the current state s of the game, CSAS ranks the actions a based on the estimates $Q(s,a)$ provided by Q-learning. The action corresponding to the current level is then chosen to be performed. The level is initially 0.5, indicating that the action in the 50th percentile is to be chosen (that is, the action corresponding to the median value). The level is increased or decreased periodically, every τ time steps, depending on the current situation of the game. In their tests, they used a fighting game, and used the health difference to decide whether to increase, decrease or leave unaltered the level. The main limitation of the approach is that the parameter τ needs to be tuned, different values compromising between rapid change of level (which causes jittering) and slow change of level (which can potentially be too slow to track the opponent level before the game finishes). Another limitation derives from the lack of principle on which to base the system: the median action can be a very strong action in certain states and a very weak one in others; in fact, the rank of the action is not a good indication of its strength.

In this section we describe our approach to using Monte Carlo Tree Search (MCTS) with Upper Confidence Bound 1 applied to Trees (UCT) [19] for DDA AI agents. Our approach is inspired by CSAS, and addresses its main limitations. We are not proposing new MCTS algorithms; rather, we propose agents that use MCTS as a search technique with the aim of producing an appropriate level of challenge to the opponent. Our contribution consists in two action selection strategies (that MCTS uses to select the action after the tree has been built) and in two heuristic playout evaluation strategies (that MCTS uses to evaluate the state resulting from playouts) that give rise to behaviours that fit a DDA AI agent.

A. Reactive Outcome-Sensitive Action Selection

Our first DDA AI agent, Reactive Outcome-Sensitive Action Selection (ROSAS), is developed with the goal of matching as closely as possible the current skill level of the opponent. ROSAS builds a search tree using MCTS, where playouts are evaluated with the aim to promote strong actions: this is the same as normal MCTS. For instance, in a 2D real-time fighting game, the health points difference between players can be used. In particular, the score is to be positive if the DDA AI agent is winning, negative if it is losing. This is the same heuristic that an AI trying to win would use. However, after the search tree is built, ROSAS selects the action whose score is the closest to zero; i.e., the action most likely to produce an outcome with a zero health points difference. The formula to select the action is:

$$\text{action} = \arg\max_a -|r[a].\text{score}|, \tag{2}$$

where r is the root node of the tree and $r[a]$ is the child of r corresponding to action a.

B. Proactive Outcome-Sensitive Action Selection

A limitation of ROSAS is that it produces non-proactive behaviour; this reactive behaviour results in an agent that only responds, never trying to take a lead. This can feel very unnatural and could even be exploited by players to win, by waiting for the last second to deal damage to the agent driven by ROSAS.

To address this limitation, we develop a second DDA AI agent, Proactive Outcome-Sensitive Action Selection (POSAS). This agent is similar to ROSAS: it builds a tree using MCTS and selects an action with the goal of matching the opponent level. However, ROSAS takes a different approach to the final selection: all actions in a defined interval around zero are all considered equally valuable and one is randomly selected. The formula used is:

$$\text{action} = \arg\max_a -(|r[a].\text{score}| - I_h)^+, \tag{3}$$

where r is the root node of the tree, $r[a]$ is the child of r corresponding to action a, I_h is the half-size of the interval around zero in which all actions are considered equally valuable, and $(\cdot)^+$ indicates the ramp function; i.e., a function behaving like the identity function for positive numbers and returning 0 for negative numbers.

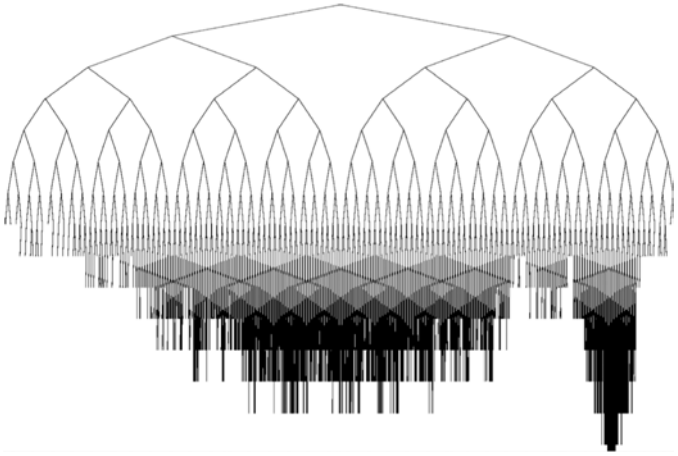

Fig. 2. Example of asymmetric tree (from Browne et. al. [17]).

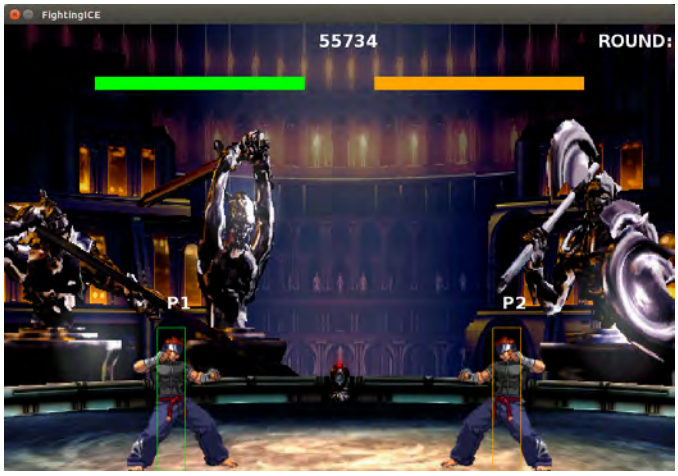

Fig. 3. Screen capture of the fightingICE game.

This mechanism allows ROSAS to take the initiative and gain some advantage over the opponent. While this does not target the desired outcome exactly, it behaves in a more realistic fashion and is less exploitable by players.

In our 2D real-time fighting game, I_h is set at $\pm 10\%$ of maximum player health.

C. True ROSAS/POSAS

The main strength of MCTS is that it explores the search tree asymmetrically. This means that the regions of the tree that are most promising will be expanded and explored more. Figure 2 shows an example of such a tree. While this is useful to make the most well-informed decision, ROSAS and POSAS cannot take advantage of this; in fact, the opposite happens: the region of the tree they are most interested, that with balanced outcomes, is under-explored. This is due to the playout evaluation strategy, which values advantageous outcomes more, leading to a deeper exploration in those areas.

To address this limitation, we propose two additional DDA AI agents, True ROSAS and True FOSAS. Both these agents change the heuristic strategy of MCTS such that the tree is expanded in a way that aligns with their respective goals.

True ROSAS evaluates nodes using the formula:

$$\text{node.score} = -|h_s|, \tag{4}$$

where h_s is the heuristic strategy employed; for our 2D real-time fighting game, this is the health points difference. When the tree is fully built, True ROSAS selects the action with the most visits (or the one with highest score). This ensures the tree expansion is in-line with the action selection policy of the ROSAS algorithm, ensuring the most relevant areas are explored more in depth.

True POSAS, similarly, evaluates nodes using the formula:

$$\text{node.score} = -(|h_s| - I_h)^+, \tag{5}$$

where h_s is the heuristic strategy employed, I_h is the half-size of the interval around zero in which all actions are considered equally valuable, and $(\cdot)^+$ indicates the ramp function; i.e., a function behaving like the identity function for positive numbers and returning 0 for negative numbers.

This maintains the goal of POSAS while expanding the search tree in the areas that relate to the heuristic strategy rather than the most promising actions for victory.

IV. Experimental Results

In this section we describe the two experiments we conducted to investigate the effectiveness of the DDA AI agents and their results. We developed the four agents ROSAS, POSAS, True ROSAS and True POSAS to play a 2D fighting game [23] called FightingICE (see Figure 3). FightingICE is real-time fighting game similar to Street Fighter [24] and Mortal Kombat [25]. Two players face off in a 2D arena and using a combination of movement, jumping, crouching, blocking, punching, kicking and special attacks try to lower the opposing player health points. The game is played over timed rounds, and the round ends when either the timer ends or one player's health points reach zero.

A. Bot Trials

To test the DDA AI agents ability to adapt to an individual opponent and provide an appropriate level of challenge, we played all four agents against a number of bots from the 2016 CIG FightingICE competition and recorded their performance.

To provide the appropriate level of difficulty, the bot and our agent need to be evenly matched with either having a 50% of winning. The outcome ROSAS targeted was a zero health points difference between itself and the bot that it was playing against, as previous work in this area [26] has shown that this indicates the opponents are evenly matched throughout the game. For example, if the opponent bot had a lead of 20 health points, ROSAS would select an outcome that would reduce the health point lead of the opponent bot to or as close to zero as possible. True ROSAS builds the search tree based

Agent Name	Mean	Median	St. dev.	TOST ± 20 p-value	TOST ± 50 p-value
CSAS	-13	1	71	0.439	0.191
ROSAS	-1	-1	17	0.033	0.000
POSAS	-11	16	39	0.360	0.049
True ROSAS	-1	0	9	0.001	0.000
True POSAS	-9	-15	34	0.308	0.023

TABLE I

AGGREGATED MEAN AND MEDIAN FINAL HEALTH DIFFERENCE, STANDARD DEVIATION, AND TWO ONE-SIDED T-TESTS (TOST) p-VALUE FOR DIFFERENT HEALTH POINT INTERVALS, BETWEEN THE AGENT AND ALL BOTS. AGGREGATED OVER ALL 500 GAMES.

Agent Name	Win Rate
CSAS	36%
ROSAS	40%
POSAS	37%
True ROSAS	24%
True POSAS	39%

TABLE II

AGGREGATED WIN RATE (PERCENTAGE) FOR OUR AGENTS AND ALL BOTS. AGGREGATED OVER ALL 500 GAMES.

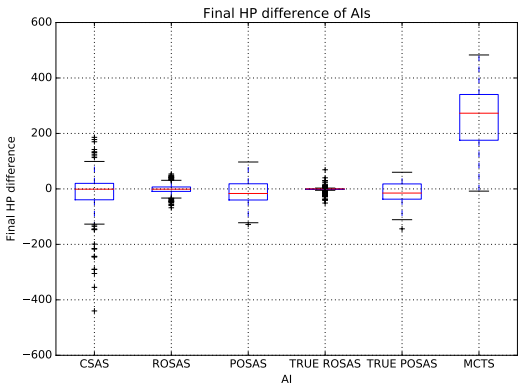

Fig. 4. Boxplot of the final health point difference between our agents, CSAS and traditional MCTS and all bots, for all games.

Agent Name	Mean	Standard Deviation
ROSAS	+16	±67
POSAS	-4	±48
True ROSAS	-4	±52
True POSAS	+1	±39

TABLE III

AGGREGATED MEAN FINAL HEALTH DIFFERENCE, STANDARD DEVIATION AGENTS AND THE HUMAN PLAYERS. AGGREGATED FOR ALL 5 PLAYERS.

on the same outcome targeted by ROSAS and then selects the highest valued action.

The POSAS agent targets a health point difference in the range of ±10% the maximum health points. This is done by generating a list of all the actions that will result in the desired outcome range and randomly selecting one; if none are present, the action with the value closest to the interval is chosen. True POSAS, similarly, builds its search tree based on the same outcome range as POSAS and then randomly selects one of the highest valued actions, or the closest one to the target range.

The experiment conducted involved the four agents playing 100 games against each of the following bots: paranahueBot, BANZAI, Poring, DragonSurvivor and RandomCommands[1]. Each game consisted of three 90 second rounds with both players starting with 500 health points.

Additionally we develop an agent using Challenge Sensitive Action Selection [21] (CSAS) to compare against our DDA AI agents. To provide a fair comparison, our implementation of CSAS uses MCTS to compute the actions value and rank, as opposed to Q-Learning [22], as in the original work. We tested the CSAS agent against the same bots as the other agents in the same setting.

Table I shows the mean and standard deviation of the final health difference between our agents and the competition bots, aggregated over all 500 fights that our agents played. A positive mean indicates that the agent, on average, finished with more health points than the opponent, and a negative mean indicates the opposite. Table I also shows the p-values as computed by Two One-Sided t-Tests (TOST) on the final health difference compared for different intervals.

[1]RandomCommands plays by performing random actions, a list of the random commands can be found at [23].

B. Player Trials

To test whether our agents perform suitably well against human players we conducted a small internal study where we played our agents against 5 people. The experiment involved the players first playing 2 games, one against RandomCommands and one against Jaybot2016 [23] to get a feel for the game. The players then played four games, one game against each of our agents in random order. All players were familiar with the genre of fighting game, but were unsure of their level of skill. As before, each game consisted of three 90 second rounds with the health points for both players set at 500. The results are shown in Table III. Due to the small number of games played, statistical test were not conducted on these results. This experiment has only anecdotal value, and a larger study is required to draw a conclusion.

V. DISCUSSION

The CSAS methodology could have benefited from replacing RL with MCTS: there is an improvement in both the mean and standard deviation when compared to the numbers reported in the original work [2]. This could also be due to the difference between the two gaming platforms. We were not able to obtain a copy of the game used in their study. MCTS did not require us to design an appropriate space representation, making the difference less likely to be related to poor decisions in this regard on our side.

We analysed the final health difference of the bot matches that we ran in the experiments, to gain confidence that our

[2]We took into account that the experiments performed in [21] used a different maximum health compared to our experiments.

agents were performing as we designed them. We computed the p-values of the samples collected using Two One-Sided t-Tests (TOST); the results are shown in Table I. We looked at two different health point intervals, the first interval we looked at was ± 20. This interval represents 4% of the total health points of the players, and it is important that our ROSAS and True ROSAS agents are playing within this interval as it is very close to the target of a zero health point difference. The results of the tests show that we have confidence that the True ROSAS agent (p-value=0.001) and the ROSAS agent (p-value=0.033) have a mean final health point difference within this interval. These results also indicate that our ROSAS agent and True ROSAS agent perform better than CSAS when aiming for a zero health point difference against the bots. The POSAS agent and the True POSAS agent target a health point difference range of $\pm 10\%$, this is represented by the interval ± 50. We have confidence that both the POSAS agent (p-value=0.049) and True POSAS agent (p-value 0.023) are playing within the target range of health point difference, when playing against the bots. No statistical test were run on our human trial data because of the small sample size. Whilst the sample is not sufficiently large to draw any conclusion, the results shown in Table III suggest that the agents, when playing against human players, can achieve similar performance to those achieved when playing against bots. A larger study will be conducted in future work to confirm this hypothesis.

Table I shows that our agents have an improved mean and standard deviation when compared to CSAS. Even though ROSAS and True ROSAS produce a mean closely matching zero and a small standard deviation, the agents will only ever respond after the player achieves a health point lead; i.e., they will not take any initiative to engage with the player when there is no health point difference. When used in tests against bots this is acceptable; however, human players may feel unfairness at the resulting rubber-banding effect, not to mention their suspension of disbelief is likely to be disrupted. It was also suggested by human players, anecdotally, that ROSAS had the most unrealistic behaviour of all the agents. POSAS and True POSAS provide a more interactive opponent that will initiate actions against the player even when there is no health point difference, and will at times stay timid when there is a health point difference due to its action selection range. While this results in a larger standard deviation when compared to ROSAS, it produces better perceived behaviours by human players.

In Figure 4 we see the distribution of the final health point difference between the agents and the bots. From this box-plot it is evident that True ROSAS is able to best track the desired outcome of a zero health points difference. On the other hand, CSAS shows a large number of outliers, which may result in a number games against players which are at an inappropriate level of difficulty.

Our algorithms closely match the desired outcome of a zero health points difference, suggested in previous work [21] as an appropriate level of difficulty. None of the algorithms, however, achieve a win rate of 50% (see Table II) which has been suggested in the literature [7], [8], [10] as an appropriate level of difficulty. This is due to the inherent nature of the algorithms responding to the players action; this, in a situation of low health for both players, will result in defeat more often than victory for the algorithms. The algorithms could be modified to target an outcome above a zero health point difference to achieve a win rate of 50%. However, the value of a 50% win rate over that of a zero health points difference throughout the game has not been established. Since the value of a 50% win rate is in the uncertainty of the outcome, we suggest an other (possibly more accurate) metric for such uncertainty could be the HP difference between the players. In fact, a difference close to zero throughout the game is a symptom of an unpredictable outcome; assuming the game does not break the player suspension of disbelief. For this reason we speculate that there could be more value in keeping the player on the edge of losing rather than achieving an exact 50% win rate

An intrinsic limitation of all DDA AI agent systems is that the difficulty level provided by the agents is limited by the maximum level the AI can play. In this case, our DDA AI agents are all limited by the maximum strength of MCTS. Figure 4 shows the results from an agent that uses MCTS with the aim of winning rather than providing an adaptive level of difficulty. In these experiments the MCTS agent had no problem winning against the different bots used. However, it may be the case that a stronger bot or a professional human player can consistently beat this traditional MCTS agent. In this case our agents will not be provide a suitable level of difficulty for that bot or player.

VI. CONCLUSIONS

This paper presents a novel approach to Dynamic Difficulty Adjustment systems, through the use of MCTS search trees, by changing the method in which the search tree is built and how actions are selected. Unlike existing approaches, this method requires smaller changes to the traditional MCTS algorithm used to develop expert AI agents. Our agents change the way the actions are selected by focusing on a desired outcome rather than victory.

The proposed DDA AI agents have shown that they are capable of providing an appropriate level of difficulty for a range of different bots and human players. They also show improved performance when compared to existing methods for DDA in the fighting genre, additionally they are less complex in implementation than previous MCTS-based DDA algorithms.

Although this implementation works well for the fighting game genre, the strategies employed in this type of game are very short term. Traditional MCTS has been shown to be an effective method in developing AI agents in games that have long term strategies and vast search spaces [27], as well as, in games in which a heuristic can describe the game outcome [17]. In future work, we will be investigating how our approach can be applied to games that require longer term strategy, using a turn-based game like Carcassonne [28].

Furthermore, with realism in mind, our proactive agents can be modified to provide even better opponents. For example, once a health point difference is selected in with the defined range around a zero health point difference, actions should adhere to such decision for more time than they currently do (one decision every frame). Notice that this is not the same as the evaluation cycle in CSAS; the difference is that our proactive agents select a *target* health point difference, while CSAS selects the *level* of actions (of which the rank is a potentially inaccurate proxy), which could lead to an increase in health points difference until the rank is readjusted. Additionally, the range used for decisions could be made to scale relatively to the skill level of the opponent, to ensure that the algorithm takes a more realistic initiative against more skilled opponents; i.e., the agent should take a larger health points lead against skilled players, who are more likely to make a strong come-back.

The current literature suggests that a win rate of 50% is an appropriate level of difficulty for matching players in competitive games. However, this may not be true for entertainment purposes; the authors suggest that keeping players on the edge may have increased entertainment value. Further investigation of this is required to inform Dynamic Difficult Adjustment system developers of the best methodology for entertaining their players.

The best approach may again be different for training or educational purposes; a slightly higher than 50% player win-rate may increase player motivation and avoid disheartening players (especially those with a fear of failure) [29] during training by ensuring that they win more often than they lose.

As an extension of this work, we hypothesize that the actions selected by our agents against the human players can be used to determine the players skill level for the purposes of reporting player skill progression and training players to be better at the game through Dynamic Difficulty Adjustment. By computing the average strength of actions chosen by our algorithms, we could measure players skill level in a similar way to how we have previously proposed [30].

REFERENCES

[1] B. Cowley, D. Charles, M. Black, and R. Hickey, "Toward an understanding of flow in video games," *Computers in Entertainment (CIE)*, vol. 6, no. 2, p. 20, 2008.

[2] M. Csikszentmihalyi, *Flow : the psychology of optimal experience*, 1st ed., ser. Harper Perennial modern classics. New York: Harper Perennial, 2008.

[3] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.

[4] P. Sweetser and P. Wyeth, "Gameflow: a model for evaluating player enjoyment in games," *Computers in Entertainment (CIE)*, vol. 3, no. 3, pp. 3–3, 2005.

[5] C. Jennett, A. L. Cox, P. Cairns, S. Dhoparee, A. Epps, T. Tijs, and A. Walton, "Measuring and defining the experience of immersion in games," *International journal of human-computer studies*, vol. 66, no. 9, pp. 641–661, 2008.

[6] E. Brown and P. Cairns, "A grounded investigation of game immersion," in *CHI'04 extended abstracts on Human factors in computing systems*. ACM, 2004, pp. 1297–1300.

[7] G. N. Yannakakis and J. Hallam, "Towards capturing and enhancing entertainment in computer games," in *Hellenic Conference on Artificial Intelligence*. Springer, 2006, pp. 432–442.

[8] T. W. Malone, "What makes things fun to learn? heuristics for designing instructional computer games," in *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*. ACM, 1980, pp. 162–169.

[9] A. Elo, "New USCF rating system," *Chess Life*, vol. 16, pp. 160–161, 1961.

[10] R. Herbrich, T. Minka, and T. Graepel, "Trueskill: A bayesian skill rating system," in *Advances in Neural Information Processing Systems*, 2006, pp. 569–576.

[11] A. L. Alexander, T. Brunyé, J. Sidman, and S. A. Weil, "From gaming to training: A review of studies on fidelity, immersion, presence, and buy-in and their effects on transfer in pc-based simulations and games," in *The interservice/industry training, simulation, and education conference (I/ITSEC), NTSA, Orlando, Florida*, 2005.

[12] R. Hunicke and V. Chapman, "Ai for dynamic difficulty adjustment in games," in *Challenges in Game Artificial Intelligence AAAI Workshop*, 2004, pp. 91–96.

[13] G. N. Yannakakis and J. Hallam, "Real-time game adaptation for optimizing player satisfaction," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 2, pp. 121–133, 2009.

[14] P. M. Avery and Z. Michalewicz, "Adapting to human gamers using coevolution," in *Advances in Machine Learning II*. Springer, 2010, pp. 75–100.

[15] A. Baldwin, D. Johnson, and P. A. Wyeth, "The effect of multiplayer dynamic difficulty adjustment on the player experience of video games," in *CHI'14 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2014, pp. 1489–1494.

[16] G. N. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *Affective Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 147–161, 2011.

[17] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

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

[19] L. Kocsis and C. Szepesvri, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*, ser. Lecture Notes in Computer Science, J. Frnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Springer Berlin Heidelberg, 2006, vol. 4212, pp. 282–293.

[20] Y. Hao, S. He, J. Wang, X. Liu, W. Huang *et al.*, "Dynamic difficulty adjustment of game ai by mcts for the game pac-man," in *Natural Computation (ICNC), 2010 Sixth International Conference on*, vol. 8. IEEE, 2010, pp. 3918–3922.

[21] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Challenge-sensitive action selection: an application to game balancing," in *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*. IEEE, 2005, pp. 194–200.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[23] R. U. Intelligent Computer Entertainment Lab, "Fighting game ai competition," *http://www.ice.ci.ritsumei.ac.jp/ ftgaic/*, 2017.

[24] Takashi Nishiyama, "*Street Fighter*," Game [Arcade], Osaka, Osaka Prefecture, Japan, August 1987, CAPMCOM, Osaka, Japan.

[25] E. Boon and T. John, "*Mortal Combat*," Game [Arcade], Chicago, Illinois, U.S., October 1992, midway Games, Chicago, Illinois, U.S.

[26] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Extending reinforcement learning to provide dynamic game balancing," in *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 7–12.

[27] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[28] Z.-M. Games, "Carcassonne," *http://www.zmangames.com/carcassonne-universe.html*, 2017.

[29] J. E. Brophy, *Motivating students to learn*. Routledge, 2013.

[30] S. Demediuk, W. L. Raffe, and X. Li, "An adaptive training framework for increasing player proficiency in games and simulations," in *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*. ACM, 2016, pp. 125–131.