Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

Modifying MCTS for Human-Like General Video Game Playing

Ahmed Khalifa, Aaron Isaksen, Julian Togelius, Andy Nealen
New York University, Tandon School of Engineering
ahmed.khalifa@nyu.edu, aisaksen@nyu.edu, julian@togelius.com, nealen@nyu.edu

Abstract

We address the problem of making general video game playing agents play in a human-like manner. To this end, we introduce several modifications of the UCT formula used in Monte Carlo Tree Search that biases action selection towards repeating the current action, making pauses, and limiting rapid switching between actions. Playtraces of human players are used to model their propensity for repeated actions; this model is used for biasing the UCT formula. Experiments show that our modified MCTS agent, called *BoT*, plays quantitatively similar to human players as measured by the distribution of repeated actions. A survey of human observers reveals that the agent exhibits human-like playing style in some games but not others.

1 Introduction and Background

When developing an algorithm to play a game, the goal is usually to play the game as well as possible. For many games, there is a suitable and unambiguous quantitative metric of success. This might be the score, progress in the game (e.g. by number of pieces captured, level reached, or distance driven), or the highest skill level of an opponent that the player can beat. This metric then becomes the measurement of success for computer algorithms playing the game; the AI agent's role is to maximize the success metric. Such metrics are used for most game-based AI competitions, including those focused on Chess, Go, Simulated Car Racing or StarCraft.

There are however other concepts of success that are more suitable in many cases. In particular, one concept of success is to what extent an artificial player plays in the style of human players (either a particular human or humans in general). In other words, a metric that measures how human-like the agent plays. This is important for several different scenarios. One case is when procedurally generating content, such as levels or items, for a game using a search-based procedural content generation method. Such methods typically rely on simulation-based testing of the content, i.e. an artificial player plays the new content in order to judge its quality [Nelson, 2011; Togelius *et al.*, 2011]. In such cases the content needs to be tested by a human-like artificial player [Isaksen *et al.*, 2015], as a non-human-like AI player might engage with the content in significantly different ways, even if the human-like and non-human-like players have the same general skill level. Automatic playtesting of this kind is also important for human-designed game content, regardless of whether it is designed by professional game developers as part of traditional content production pipeline or by amateurs (e.g. the recent success of Super Mario Maker, which almost entirely is built on player-created content). Human-like playing is also important for creating tutorials and other forms of demonstrations. Finally, it can be important for the immersion and general player experience to have human-like non-player characters in a game. It is commonly thought that the popularity of multi-player games is at least partly due to the lack of believable agents in games [Hingston, 2012]. Similarly, playing against a too-good AI opponent is not enjoyable, so there is often an effort to decrease the skill level of the agent to better match that of the human player. It is worth noting that while the argument here mainly refers to entertainment-focused games, it applies equally well to serious games, virtual reality, games with a purpose, and real-time interactive simulations in general.

In many cases, AI agents that play games well do not play in a human-like manner. This is an informal observation that has been made many times for different game genres. It has been claimed that Chess-playing programs play differently from humans, even when playing at the same skill levels [Simon and Schaeffer, 1990; Sloman, 1999]. When it comes to video games, this has been observed in for example Super Mario Bros [Ortega *et al.*, 2013], simulated car racing [Muñoz *et al.*, 2010], Quake II [Thurau *et al.*, 2004] and Unreal Tournament [Schrum *et al.*, 2011]. Several researchers (see citations in previous sentence) have attempted domain-specific human-like AI players for these games. Two recent series of game-based AI competitions, the 2k BotPrize [Hingston, 2009] and the Mario AI Championship Turing Test [Shaker *et al.*, 2013], have focused specifically on producing artificial players that play in a human-like manner. In these competitions, the human-likeness of the submitted agents were compared directly to humans, and judges were asked which characters were human-controlled and which were AI-controlled. The results suggest that in many cases, it is harder to create a human-like game-playing agent than to create a high-performing agent that wins many games or achieves superhuman scores.

Another thing that these competitions and related studies have suggested is a preliminary and incomplete list of ways in which the behavior of common game-playing algorithms differ from human gameplay. As these agents are typically not bound by human reaction times, a characteristic "jitteriness" can be observed, where player actions are changed more often than would be practical or possible for a human. A related phenomenon is the lack of pauses; most human players will occasionally stop and pause while evaluating the possibilities ahead. Finally, many algorithms are prone to a kind of cowardice in that high-risk strategies are not pursued (alternatively, one might say that many humans are prone to recklessness).

1.1 General Video Game Playing and MCTS

While there have been a number of attempts to produce human-like agents for specific games, taking into account domain-specific knowledge about the affordances of the game's design, there have been very few investigations of human-like game-playing *in general* [Hingston, 2012; Shaker *et al.*, 2013]. We are not aware of any other work that examines which features are "AI giveaways" across multiple games and how to create agents that can play any game of some given type in a human-like manner. The challenge of playing any game, not just a particular game, is called General Game Playing (GGP) [Genesereth *et al.*, 2005] for combinatorial games and board games or General Video Game AI (GVG-AI) [Perez *et al.*, 2015] for real-time action/arcade games – both are built around AI competitions. For these competitions, agents play not only a single game, but must play games they have not yet seen – they are not known ahead of time by the agent or the developer of the agent. (This differentiates GGP and GVG-AI from e.g. the Arcade Learning Environment, where agents are trained for a finite and well-known set of games [Bellemare *et al.*, 2012].) In this paper, we will focus on human-like agents for the GVG-AI competition.

Most of the best-performing players for GVG-AI are based on Monte-Carlo Tree Search (MCTS) [Perez *et al.*, 2015]. This is a relatively recently invented algorithm for stochastic planning and game-playing [Browne *et al.*, 2012]. Compared to other tree-search algorithms used for game playing there are two main differences: (1) MCTS balances exploration and exploitation when selecting which nodes to expand, leading to a more focused search; and (2) MCTS usually utilizes random playouts instead of (or in addition to) state evaluation functions. MCTS was popularized after revolutionizing computer Go; Go-playing agents that utilize MCTS thoroughly outperform agents based on any other methods, as well as the best human players [Silver *et al.*, 2016]. In the case of General Video Game AI, the relative success of MCTS is largely due to the impossibility of deploying game-specific modifications to the agent; MCTS has proven very useful because of its generality.

Standard MCTS plays real-time video games (e.g. Mario) worse than board games (e.g. Go) for multiple reasons: decisions have to be made in real-time (at approximately once every 40ms), non-player characters can react stochastically, there are often no simple heuristics to evaluate the fitness of a particular action in the short term, a lack of immediate rewards, and there is a large amount of hidden information. In addition, the game tree will often need to be reconstructed from the scratch each time step. Although standard MCTS has all these problems in playing real-time video games, it still performed decently in the 2014 GVG-AI Competition [Perez *et al.*, 2015].

1.2 This Paper

The challenge that this paper addresses is that of creating high-performing human-like players for the GVG framework. The agent needs to be able to perform well on the game as well as play in a human-like manner.

The structure of the paper is as follows. First we describe the GVG-AI testbed and the games we used for data collection. We then describe the quantitative measures we use to characterize human playing behavior and compare it to the game-playing AI agents. Next, we describe a number of modifications to the core MCTS algorithm in order to increase its human-likeness. This is followed by our simulation results, where we use the quantitative measures to compare the agents with humans. Next, we perform a user study where we ask human observers to identify human players among a set of agents and humans playing these games; we discuss the quantitative results of this study as well as the qualitative reasons given by observers for their judgments. We conclude by discussing how this work can be furthered and what it says about the possibility for creating general game AI that plays in a human-like manner.

2 Testbed and Data Collection

We used the GVG-AI competition framework for our experiment, as it includes a large number of video games with the same agent API and can be easily instrumented for data collection. Games are encoded in Video Game Description Language (VGDL) [Ebner *et al.*, 2013; Schaul, 2013], capable of representing a wide variety of 2D arcade games. The central assumptions are that the games feature graphical logic and 2D movement; currently, 60 games are included with the framework, including remakes of arcade classics such as *Seaquest*, *Frogger* and *Space Invaders*.

We first collected human play traces for analysis, to compare with a standard MCTS agent. For this purpose, we distributed a small program which allowed humans to play the selected games and saved their play traces to a server. Each row in the game analytics database stores a play trace that contains the sequence of actions at every frame (including a nil-action if nothing was performed on a frame), the final score, and whether the game was won before 1000 frames or lost by player death or time expiring on the 1000th frame. We collected 209 play traces from 11 different human players, all members of our lab. We also recorded 50 play traces from the standard MCTS agent included with GVG-AI. All players and the MCTS agent played the first level of three different games:

- **Aliens:** VGDL implementation of Space Invaders. The avatar can move left, move right, and fire bullets. The goal is to kill all the aliens spawned at the top of the level. Aliens try to kill the player by moving across the screen and downwards while intermittently dropping bombs.
- **PacMan:** VGDL implementation of Pac-Man. The avatar moves in all four directions around a maze, eating

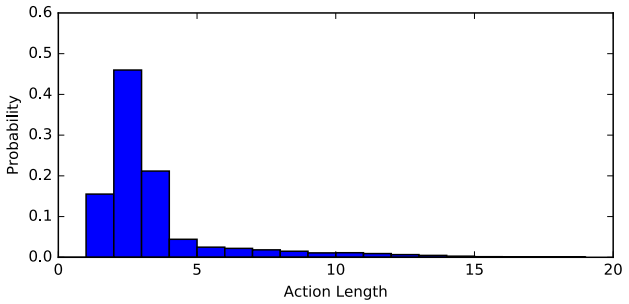

Figure 1: Action length histogram P_a for human players.

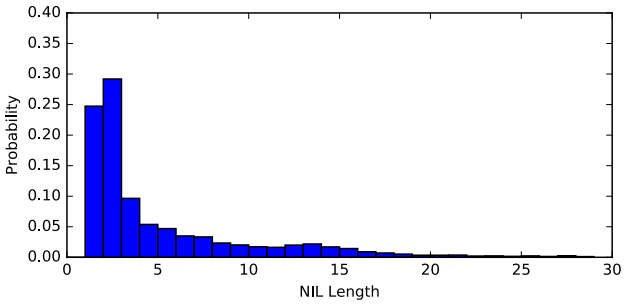

Figure 2: Nil-action length histogram P_n for human players.

dots, fruits, and power pellets. The goal is to eat all dots while avoiding being caught by chasing ghosts.

- **Zelda:** VGDL implementation of the dungeon system in the early The Legend of Zelda games. The avatar can move in all four directions and attack in the direction they are facing with a sword. The goal is to grab a key and then reach the exit without dying. Monsters move randomly, killing the player if they overlap locations. The player can either avoid monsters or kill them for extra points by using the sword.

3 Quantitative Measures of Player Behavior

We analyzed the collected data and compared the differences between Human and standard MCTS play traces. This comparison showed significant difference in several different aspects. We noticed that the standard MCTS rarely repeated the same action (e.g. Left, Right, Up, Down, and Attack) two frames in a row, while human players tended to repeat actions several frames in a row. MCTS also rarely repeated the nil-action, while human players would perform a nil-action for multiple frames in a row. As a result, the standard MCTS has a high tendency to switch from one action to another much more often than human players do. The following subsections explains these aspects in detail.

3.1 Action Length

Action Length is number of repetition of each action, excluding nil actions. In real-time games, humans have a tendency to press a button for more than one frame due to non-instantaneous reaction time [Schmidt and Lee, 1988]. Agents, on the other hand, can react far more rapidly, and therefore tend to make many single frame actions of length 1.

To calculate Action Length, we examine the time series of actions taken by the player from each play trace. We count the total number of frames an action was held (x is recorded when an action is used for x frames) and add it to a histogram. After dividing by the histogram sum, the final action length probability histogram is P_a, shown in Figure 1.

3.2 Nil-Action Length

Nil-Action Length is the number of frames that an empty nil-action is repeated. In real time games, humans have a tendency to occasionally wait (i.e. do no actions) to think or change direction. However, AI agents typically have a fast response time, so there is no need to pause and think.

To calculate Nil-Action Length, we again examine the time series of actions taken by the player from each play trace, but now we only count how long a nil-action is repeated. The final nil-action probability histogram is P_n, shown in Figure 2.

3.3 Action to New Action Change Frequency

The Action to New Action Change Frequency metric measures the number of times that a player switches from one action to another (e.g. from Up to Left in consecutive frames), divided by the total number of frames in the play trace. AI agents in real time games, free to select the optimal move on every frame, often change actions quite frequently which increases the jittery effect we are trying to avoid. High values indicate high-jitter and generally only occurred in AI agents. We do not use this in our model to select actions, but we use this metric to compare agents to humans in Section 6.

4 Proposed MCTS Modifications

Monte Carlo Tree Search (MCTS) is a stochastic tree search algorithm which is widely used for general game playing as well as playing specific games such as Go; in applicable cases, it asymptotically approaches the same result as a Minimax algorithm [Browne *et al.*, 2012]. Instead of exploring the tree in a depth-first or breadth-first manner, MCTS stochastically explores new areas of the search tree that have not been visited previously, while also exploiting promising areas that have already been discovered but not fully explored. In order to understand the modifications, we first describe the four main steps of the standard MCTS algorithm:

1. **Selection:** the algorithm selects a node in the tree to expand. In this step MCTS tries to balance between exploitation and exploration using the Upper Confidence Bound (UCB) equation.

$$UCB_j = \bar{X}_j + C\sqrt{\frac{ln\ N}{n_j}} \quad (1)$$

where $\bar{X}_j$ is the average node value for all the runs that pass through this node and is responsible for exploitation (high averages will be exploited more often), C is a constant to encourage exploration, N is the total number of

visits to all children of node j, and n_j is the number of visits for this node.

2. **Expansion:** MCTS picks a random unvisited child of node j to expand.
3. **Simulation:** MCTS plays the game using random moves until it reaches a termination state (win/loss) or decides to approximate the simulation using a heuristic.
4. **Backpropagation:** MCTS updates all ancestor nodes of the child using results from the simulation step – including the average scores $\bar{X}_j$, total visits N, and child visits n_j for each node up the tree.

The GVG-AI framework comes with a standard MCTS implementation as one of the AI agents, with some changes to support video games instead of the typical combinatorial games that MCTS is more commonly used for. Instead of playing until it reaches a final state, it only simulates it for a fixed number of actions and then uses a heuristic function to evaluate the reached state. The function gives a large positive value for a winning terminal state, a large negative value for a losing terminal state, or the total score which the agent achieves during the game if a terminal state is not reached.

Next, we describe the changes we made to the GVG-AI MCTS algorithm to make it act in a more human-like manner. We incorporate these modifications into an agent we call *BoT*.

4.1 Human-Modeling

We first added a term H_j to the standard UCB equation to model more human-like behavior, with a tuning constant α to balance between exploration, exploitation, and human-like modeling.

$$UCB_j^+ = UCB_j + \alpha H_j \quad (2)$$

H_j itself is a function that depends on the state of the game and the previous move that has been selected. The function helps the MCTS selection process model human-like actions. For example if the previous move is action of length 1 and the current state is the same action, this means the value should be high to encourage picking the same action (as explained in Section 3.1). As the action is held longer, H_j will decrease.

The parameters for H_j are calculated from the histograms recorded from human play traces in Section 3; we tune the MCTS algorithm to follow similar human distributions for action length and nil-action length. We first select the appropriate histogram based on the previous action in the tree: if a normal action we use $P = P_a$ and if a nil-action we use $P = P_n$. We then count l, how long the current action has been held, and calculate the cumulative probability $T(l) = \sum_l P(l)$. The probability of changing to a different action after holding for l frames is then $T(l)$ while the probability of continuing to hold the action is $1 - T(l)$. Thus, H_j is calculated from a probabilistic data-driven model and depends on recent ancestors in the MCTS game tree.

At this stage, we only used modification as described in this section. The resulting AI agents exhibited actions conforming to the sampled human distribution but they did not appear to be fully human. Useless moves like walking into walls, a lack of long term planning and cowardice are some reasons for the agents to appear unbelievable. We therefore added the following techniques to increase agents believability.

4.2 Trivial Branch Pruning

Trivial Branch Pruning eliminates any ineffective movements, such as walking into walls, immediately reversing direction, and attacking indestructible objects, which are occasionally selected by MCTS. An ineffective action is when the player's orientation and position does not change after executing a move action. A branch with an ineffective action will not be further explored. The rules for action pruning were specified a priori as we want our method to be applicable to games without a learning or training phase; they could conceivably also be learned from data.

4.3 Map Exploration Bonus

The Map Exploration Bonus encourages MCTS to explore new locations in the game map by giving a bonus for visiting map tiles that have been less frequently visited. This was introduced to help the agents improve their long term planning. We modified Equation 2 to add a bonus exploration term.

$$UCB_j^{++} = UCB_j^+ + E \cdot \left(1 - \frac{visited(x_j, y_j)}{\max_i(visited(x_i, y_i))}\right) \quad (3)$$

where E is a constant for the contribution of the bonus exploration term and $visited(x_j, y_j)$ is the how many times the current map tile has been visited by the player.

4.4 MixMax

MixMax is inspired from Jacobsen et al. [2014], where several techniques were applied to enhance the performance of MCTS playing Infinite Mario Bros. MixMax was originally suggested as a method for overcoming cowardly behavior in Mario – the avatar would unproductively avoid gaps and enemies. The same problems appear in many GVG games, where the agent often flees from enemies instead of killing them.

Instead of using the average value $\bar{X}_j$ from the UCB equation, a mixed value is used between the average $\bar{X}_j$ and the maximum child value. Equation 4 shows the new exploitation part of Equation 1, replacing $\bar{X}_j$ with $\bar{X}_j^*$.

$$\bar{X}_j^* \leftarrow Q \cdot max + (1 - Q) \cdot \bar{X}_j \quad (4)$$

where Q is the mixing parameter between both terms, max is the maximum value achieved at that child and $\bar{X}_j$ is same average value as originally described for UCB.

A value of $Q = 0.25$ has an approximately best effect, chosen by visual inspection of the output behavior to make it appear most human. High values of Q cause the agent to be overly courageous and often dies from enemy attacks, while low values of Q cause the agent to be too cowardly.

5 Simulation Results

This section compares the quantitative measurements explained in Section 3 for four types of players: (a) standard MCTS, (b) AdrienCtx, the winner of the GVG-AI competition in 2014 [Perez *et al.*, 2015], (c) our BoT algorithm, and (d) actual human play. We compared them on the same three games: Aliens, PacMan, and Zelda.

Figure 3 compares different agents to human players by Action Length. As can be seen, both AdrienCtx and standard

MCTS favors short moves of length 1 while humans favor repeating the same action. Our BoT algorithm (in blue) increased the tendency of selecting the same action which leads to having a closer distribution to human players (in red).

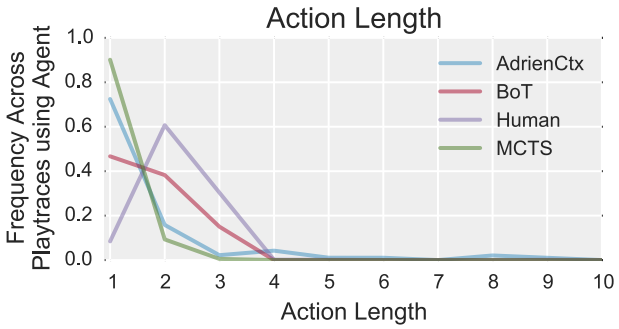

Figure 3: Action Length distribution for human players, standard MCTS, AdrienCtx, and our BoT agent.

Figure 4 compares different agents to human players by Nil-Action Length. Standard MCTS (green) has mostly short nil-actions – almost all of them of length 1. On the other hand AdrienCtx (purple) has the longest nil-action of length 11. This agent stands still for 11 frames at the beginning of each game in order to analyze the world and never performs a nil-action again, which explains this particular time distribution. BoT (blue) has a higher tendency to select the nil-action which leads it to having a similar distribution to human players (red).

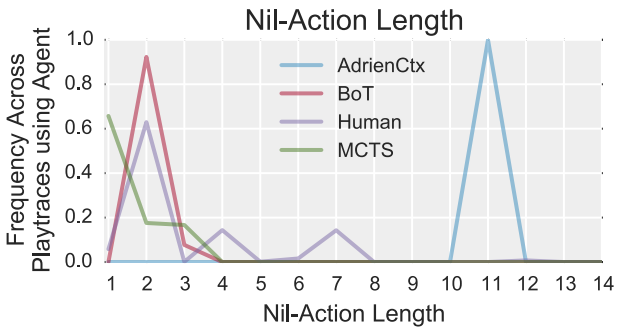

Figure 4: Nil-Action Length distribution for human players, standard MCTS, AdrienCtx, and our BoT agent.

Figure 5 shows how different agents compare with human players, using Action to New Action Change Frequency histograms. Human players, indicated in light blue, rarely switch from an action to a new action. However, standard MCTS and AdrienCtx both switch quite often from one action to another, as indicated in the histograms. Our agent, BoT, has a distribution much closer to humans. This reduces the amount of jitter, which we believe makes a more human-like agent.

Our aim is not just mimicking human distributions, but to combine human-like action distributions with the generality and positive behaviors of MCTS. An algorithm that simply sampled actions from the human distribution would fit perfectly but would be an atrociously bad game player. In order to make sure that the modified MCTS is still generic, we also use score and win rate as a performance metrics.

	Pacman	Zelda	Aliens
Human	0.00% 4.26	12.4% 0.57	22.5% 13.09
MCTS	0.66% 4.63	0.66% 0.64	32.0% 21.77
BoT	0.00% 5.33	7.33% 0.40	0.0% 25.19
AdrienCtx	24.6% 15.57	33.3% 2.08	32.0% 0.56

Table 1: Win percentage and average score for humans, standard MCTS, AdrienCtx, and our BoT agent.

Table 1 compares the performance for three algorithms to human players. The first number is the win rate while the other is the average score across all plays of that game/agent pairing. These metrics show that our BoT algorithm is scoring as well as standard MCTS. It is clear from the values that our BoT performs at least as well in both PacMan and Zelda, but worse in Aliens. The likely reason for the bad performance in Aliens is that spatial exploration is unfortunately always preferred over shooting due to the Map Exploration Bonus.

6 User Study

To verify that our algorithms perform in a more human-like manner than the standard MCTS, we performed a Turing-test-like user study. We evaluated our algorithms on three games: Zelda, Pacman, and Boulderdash. Boulderdash is a recreation of the classic game *Boulder Dash* by First Star Software. The player has to dig through dirt, collect gems and reach the exit, while avoiding falling rocks and moving monsters. This game was selected because the differences in playing style could be seen more clearly in this game than in Aliens.

For each game, we compared four different players: (a) Standard MCTS, (b) AdrienCtx, the winner of the GVG-AI competition in 2014 [Perez *et al.*, 2015], (c) our BoT algorithm, and (d) actual human play. For each of these conditions, we generated 3 videos using the GVG-AI system, for a total of 3 games * 4 algorithms * 3 videos = 36 videos.

The user study was performed within a web browser. Each participant was shown two videos chosen from the same game, side by side and labeled A and B. The participant was asked "Which of these videos appear to show a human player?" and given 4 possible answers: A, B, Both, Neither. They were also given a free form field labeled "How did you decide? (optional)" so we could better understand how they made their choices. The participants were asked to rate at least 10 games and results were not included if participants only evaluated one game. We had 34 participants in our study, with a median number of games evaluated per participant of 10 and a mean of 9.18, for a total of 312 evaluations. Since each evaluation contained two games, our study has $n = 624$ samples.

Table 3 shows the results of our user study. For each cell, we show the fraction of times the video was indicated as coming from a human (either indicated as A, B, or Both) divided by the number of times it was shown to the participants. In

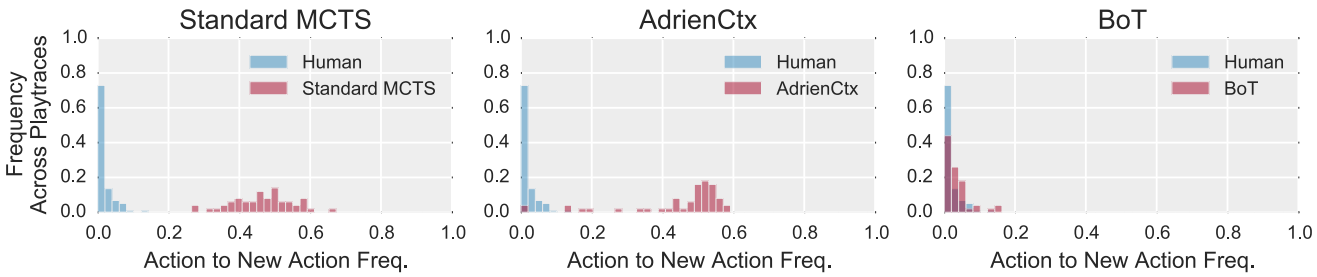

Figure 5: Action to New Action Change Frequency for Pacman. Humans have a very low frequency for this metric (light blue). (a) Standard MCTS and (b) AdrienCtx have much higher values, leading to jittery AIs. (c) BoT has a distribution much closer to human players.

the first row, we see that at best humans correctly identify human play with 88.7% accuracy for Zelda, and at worst with 70.2% accuracy for Pacman. In the second row, we see that Standard MCTS performs quite poorly, at best attributed as a human 33.3% of the time for Pacman and at worst 8.3% of the time for Boulderdash. For the two remaining algorithms, BoT performs the best at Pacman, coming close to matching the same attribution rate for human players, while AdrienCtx performs the best for Zelda and Boulderdash.

To test for significance, we use a binomial test for each game and agent, with the null hypothesis that the percentage of human-like attributions matches the percentage for Standard MCTS. Except for Pacman/AdrienCtx, we have a p-value $< .01$, rejecting the null hypothesis for all other agents and games at the $\alpha = .01$ significance level. This shows a significant result for $n = 624$ that the algorithm has an effect on how participants attribute human-like behavior to an AI.

We also analyzed the observers' optional free-text responses (163 were given) on the videos to see what are the common problems that make humans believe the game performance they are observing is not from a human. Answers were coded for the occurrence of keywords or expressions with essentially the same meaning. Table 2 shows the frequency of top qualitative keywords that identify that the player is an AI agent.

Keyword	Number of occurrences
Jitteriness	40 / 163
Useless Moves	39 / 163
No long term planning	25 / 163
Too fast reaction time	21 / 163
Overconfidence	10 / 163

Table 2: Most common qualitative reasons for claiming an observed player is not human, taken during the user study.

By combining our own observations with an analysis of the reasons given by the respondents, the overall findings can be explained as follows. Standard MCTS exhibits most of the problems that were noted in the beginning of this paper: jitteriness, performing ineffective actions such as running into and hitting walls, and often has no apparent long-term goal. AdrienCtx avoids most of these problems, but is given away by having obvious superhuman reflexes, including the ability to counteract multiple enemies much faster and more reliably than a human. In fact, some observers thought the AdrienCtx agent was cheating. BoT has the advantage that it has more human-like reaction times because of its tendency to repeat moves and wait. It also gives the appearance of following up on a course of action because of its "momentum" (tendency to continue moving in the same direction), which helps in particular in Pacman.

	Pacman	Zelda	Boulderdash
Human	70.2% 40/57	88.7% 47/53	80.7% 46/57
MCTS	33.3% 13/39	10.0% 5/50	8.3% 5/60
BoT	60.0% 33/55	26.5% 13/49	25.0% 14/56
AdrienCtx	38.2% 21/55	50.0% 26/52	39.0% 16/41

Table 3: Percentage and fraction of evaluations indicating the presented video came from a human.

7 Conclusions

MCTS is currently the most successful algorithm—or perhaps algorithmic framework—for playing unseen games in the GGP and GVG-AI competitions and associated software frameworks. However, in its standard form, the algorithm does not play in a human-like manner. In this paper, we have quantitatively and qualitatively characterized the differences in playing style between humans and MCTS-based agents in three representative games within the GVG-AI framework. We have also suggested a number of modifications to the core of standard MCTS algorithm in order to make its behavior appear more human-like; the core modifications are a new term in the UCT equation that biases action selection towards observed human histograms. The effectiveness of the modifications, collectively known as "BoT", was ascertained through computational testing as well as a Turing-test-like user study. We believe the BoT agent we have presented in this paper could be useful for automated testing and demonstration of games in the GVG-AI framework, generation of new games, and for pointing a way forward for developing methods for human-like game-playing across games and game genres.

References

[Bellemare *et al.*, 2012] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

[Browne *et al.*, 2012] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, et al. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[Ebner *et al.*, 2013] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. *Dagstuhl Follow-Ups*, 6, 2013.

[Genesereth *et al.*, 2005] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62, 2005.

[Hingston, 2009] Philip Hingston. The 2k botprize. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009.

[Hingston, 2012] Philip Hingston. *Believable Bots*. Springer, 2012.

[Isaksen *et al.*, 2015] Aaron Isaksen, Dan Gopstein, and Andy Nealen. Exploring game space using survival analysis. *Foundations of Digital Games*, 2015.

[Jacobsen *et al.*, 2014] Emil Juul Jacobsen, Rasmus Greve, and Julian Togelius. Monte mario: platforming with mcts. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 293–300. ACM, 2014.

[Muñoz *et al.*, 2010] Jorge Muñoz, German Gutierrez, and Araceli Sanchis. A human-like torcs controller for the simulated car racing championship. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 473–480. IEEE, 2010.

[Nelson, 2011] Mark J Nelson. Game metrics without players: Strategies for understanding game artifacts. In *Artificial Intelligence in the Game Design Process*, 2011.

[Ortega *et al.*, 2013] Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.

[Perez *et al.*, 2015] Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon Lucas, Adrien Couëtoux, Jeyull Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. 2015.

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

[Schmidt and Lee, 1988] Richard A Schmidt and Tim Lee. *Motor control and learning*. Human kinetics, 1988.

[Schrum *et al.*, 2011] Jacob Schrum, Igor V Karpov, and Risto Miikkulainen. Ut 2: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 329–336. IEEE, 2011.

[Shaker *et al.*, 2013] Noor Shaker, Julian Togelius, Georgios N Yannakakis, Likith Poovanna, Vinay S Ethiraj, Stefan J Johansson, Robert G Reynolds, Leonard K Heether, Tom Schumann, and Marcus Gallagher. The turing test track of the 2012 mario ai championship: entries and evaluation. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Simon and Schaeffer, 1990] Herbert A Simon and Jonathan Schaeffer. The game of chess. Technical report, DTIC Document, 1990.

[Sloman, 1999] Aaron Sloman. *What sort of architecture is required for a human-like agent?* Springer, 1999.

[Thurau *et al.*, 2004] Christian Thurau, Christian Bauckhage, and Gerhard Sagerer. Learning human-like movement behavior for computer games. In *Proc. Int. Conf. on the Simulation of Adaptive Behavior*, pages 315–323, 2004.

[Togelius *et al.*, 2011] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):172–186, 2011.