The Impact of Dynamic Difficulty Adjustment on Player Experience in Video Games

Chineng Vang
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
vang2660@morris.umn.edu

ABSTRACT

Dynamic Difficulty Adjustment (DDA) is a process by which a video game adjusts its level of challenge to match a player's skill level. Its popularity in the video game industry continues to grow as it has the ability to keep players continuously engaged in a game, a concept referred to as Flow. However, the influence of DDA on games has received mixed responses, specifically that it can enhance player experience as well as hinder it. This paper explores DDA through the Monte Carlo Tree Search algorithm and Reinforcement Learning, gathering feedback from players seeking to understand what about DDA is alluring and discouraging. How player experiences are affected by DDA in competitive multiplayer video games is also an area of interest, so a survey of player responses to DDA in multiplayer video games is also examined.

Keywords

Dynamic Difficulty Adjustment, Flow, Monte Carlo Tree Search, Reinforcement Learning

1. INTRODUCTION/BACKGROUND

The challenge a game provides to players is connected to what makes the game fun. Games that are too easy become dull and uninspiring over time. Similarly, games that are too difficult are not encouraging either. Creating a balanced challenge for players is a daunting task for game developers because the skill level of players vary. An obstacle perceived as difficult to one player may be easy to another player and vice versa. The use of DDA allows a video game to gauge the skill level of a player, and then alter game parameters attempting to provide an appropriate challenge to said player. [10]

The goal of DDA is to balance a game so a player remains engaged and motivated to continue playing. The state of mind players enter when this is achieved is called Flow. The idea of Flow and Flow Theory was studied by Mihaly Csikszentmihalyi in the 1970s. He defined Flow as a mental state of energetic focus in which a person is completely immersed in the activity that they are trying to accomplish. [2]

Flow among players is what DDA implementations aim to reach. More specifically, each implementation tries to balance a video game to maintain players in the Flow Channel (see Figure 1). The Flow Channel lies between frustration (when the challenge of a game is too difficult for lower skilled players) and boredom (when the challenge of a game is too easy for higher skilled players). Investigating how players respond to different implementations and aspects of DDA can give game developers valuable insight about what elements of a video game improve a player's game experience and as a result, keeps them in the Flow Channel. [6]

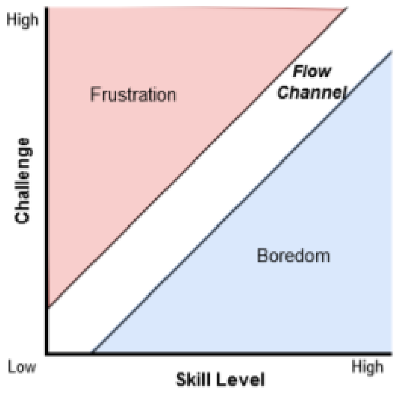


Figure 1: A model of Flow with the Flow Channel being between frustration and boredom. [6]

In this paper, Section 2 uses the Monte Carlo Tree Search (MCTS) algorithm as a foundation for three different versions of DDA. Each version is tested on players, then players give feedback on which version of DDA was most realistic, most difficult, and most enjoyable. Section 3 looks at how reinforcement learning can be used to implement DDA in turn-based games. Three versions of the reinforcement learning algorithm SARSA (explained further in Subsection 3.1) are experimented on players. The success, challenge, skillfulness, and effort that players experience while playing against each version is measured by player feedback. Section 4 explores how DDA affects multiplayer play. Players are surveyed about their preferences of DDA use in multiplayer video games. The key focus of the survey is that players respond from both a high performance perspective, if they are good at the game, and a low performing perspective, if they are not good at the game. The survey finds a stark contrast in how players perceive the use of DDA depending on if they are good at the game or not.

UMM CSci Senior Seminar Conference, November 2021 Morris, MN.

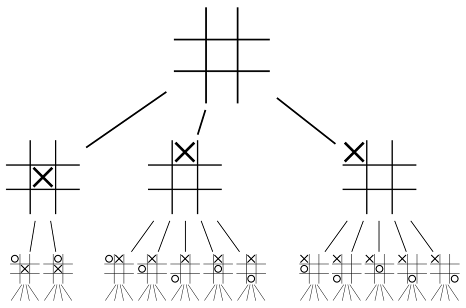


Figure 2: The states and actions for the game tree of Tic-tac-toe. [9]

2. MONTE CARLO TREE SEARCH DRIVEN DDA

The study done by Demediuk et al. implements three versions of DDA based on Monte Carlo Tree Search (MCTS), a search algorithm designed to efficiently discover the most promising courses of action leading the computer (AI) to victory. Each version alters MCTS in different ways to adjust the game difficulty to specific experiment parameters.

Games can be represented by game trees. Each node in a game tree is a game state, certain situations in a game that can be reached by taking certain actions. Each action is represented as branches in the game tree (see Figure 2). It is common for a game to have complex game trees, such as chess and checkers. Complexity comes from the tree being large, as large game trees take a long time to traverse. What makes MCTS beneficial is that unlike other popular search algorithms, the entire game tree does not have to be completely traversed in order for the AI to determine a course of action, improving efficiency.

As MCTS traverses a game tree, it builds a statistics (stats) tree for the game, which is used to help the AI decide what to do. Each node in the stats tree encodes the sequence of actions taken in the game tree up to that node. Each node also has an estimate associated with it, representing the likelihood it leads the AI to a winning state. Based on the stats tree being built, the AI will take actions in the game tree with the highest estimates in the stats tree. There is no stats tree at the start of the game, so MCTS (at the root node of the game tree) chooses random actions to begin a game. Then it repeats four steps to build the stats tree (see Figure 3):

- Selection: Each node in the stats tree holds an estimate of the likelihood that it leads the AI to a winning game state of the game tree. Nodes with higher estimates are called promising nodes and are selected to be investigated further.

- Expansion: Adding a new child node to promising nodes of the stats tree representing an action in the game tree MCTS will investigate.

- Simulation: An entire simulation of the game is tested choosing actions starting from the root node of the stats tree to the node being evaluated. Random actions are taken, following the node being evaluated, to simulate the rest of the game. The simulation produces a value associated to the path taken in the stats tree.

- Back-propagation: The value produced by the simulation is used to update the estimates of the nodes along the path of the stats tree being evaluated. Simulations leading to the AI to win increases the estimates of nodes.

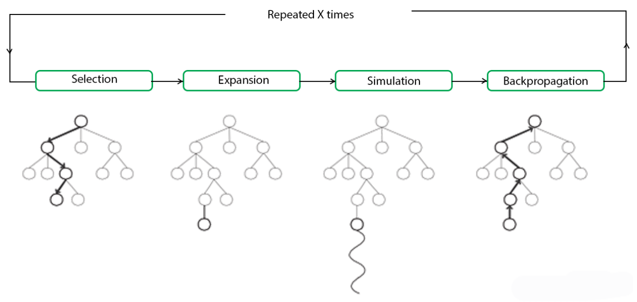


Figure 3: Selection, Expansion, Simulation, and Backpropagation of Monte Carlo Tree Search. [8]

When the AI needs to make a decision, the above process can halt and choose the best course of action available at that time (i.e. choosing the node with the highest estimate). [4]

2.1 DDA Implementations with MCTS

MCTS is used as the foundation for the three versions of DDA. As discussed in Section 1, DDA's goal is to maintain players in the Flow Channel. Demediuk et al. hypothesize that a player winning 50% of the time will accomplish this goal, so whichever version comes closest to winning half their games should provide the most balanced gameplay. Each version either changes how actions are chosen from MCTS or alter how the stats tree is built in MCTS. They then measure how close to the goal each implementation got which is discussed in Subsection 2.2.

2.1.1 DDA with CSAS

The first version of MCTS is based on Challenge Sensitive Action Selection (CSAS) which uses reinforcement learning and Q-tables [5]. Every action taken from a state of a game tree has a reward (also called value). If an action is taken by the AI, the AI receives the corresponding value. The value is a measure to how well the action taken helps the AI towards winning. In reinforcement learning, as the AI traverses a game tree, a Q-table keeps track of these state-action values. The rows of the Q-table hold states and columns hold actions to take at each state. The Q-table helps guide the AI to make decisions as the game progresses, with the beginning of the game having randomized values.

This version of MCTS (called CSAS) has the AI begin a game by choosing the mean valued action in the Q-table instead of the highest valued action. The mean is usually a middle ground, presenting an average challenge to players (see Figure 4). After a certain number of mean valued actions have been taken, CSAS evaluates if the difficulty needs to be increased or decreased depending on the player's

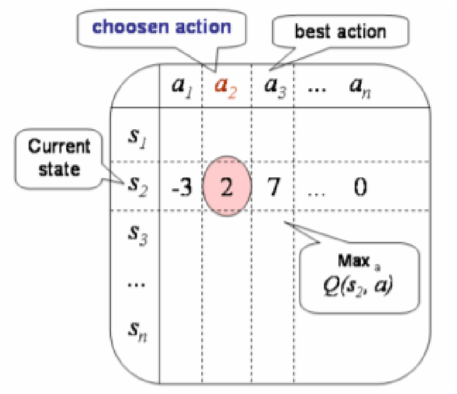


Figure 4: A sample Q-table with state-action values. The mean valued action of the state is circled. [4]

gameplay. If there needs to be an increase, values one level higher than the current one will be chosen in the stats tree. Likewise, values one level lower than the current one will be chosen if the difficulty needs to decrease. If the game is already well balanced in difficulty, there will not be any change. [4]

2.1.2 DDA with ROSAS

The second version of MCTS is based off of Reactive Outcome Sensitive Action Selection (ROSAS). Recall that MCTS chooses the action with the highest estimate in the stats tree. ROSAS deviates from choosing the action with the highest estimate and chooses an action trying to get the health bars of the AI and player to match. In a fighting game, each character has a health bar with a specified number of hit points. A character losing all their hit points means that character loses. Based on the hit points in the health bar for the AI and the player, the action trying to match their hit points (referred to as matching health bars) will be taken. Higher valued actions will be chosen if the AI is losing, becoming a more difficult opponent for the player. Lower valued actions will be chosen if the computer is winning, giving the player the advantage. ROSAS differs from CSAS because ROSAS makes adjustments to the player with every action trying to match its health bar to the player's. CSAS takes more time to adjust to the player, waiting until a specified number of actions are taken before reevaluating difficulty. [5]

2.1.3 DDA with ATROSAS

The third version of MCTS is based on Adaptive True Reactive Outcome Sensitive Action Selection (ATROSAS). ATROSAS has the same goal as ROSAS, which is to have the AI and player match health bars. CSAS and ROSAS alter how MCTS chooses action, but ATROSAS changes how MCTS builds the stats tree. In the Simulation step of MCTS, if the node being evaluated causes the difference in hit points between the AI and player to shrink, then the node receives a higher value since it is making the game closer. If the node causes the difference in hit points to increase, then the node receives a lower value.[1]. Essentially, instead of the estimates in the stats tree representing the likelihood of winning, ATROSAS alters this so that they represent the likelihood of matching the health bars of the AI and player. Like MCTS, ATROSAS then chooses the action with the highest estimate, the action most likely to match the health bars. [4]

2.2 Experiment/Results

The experiment had 31 participants of varying skill level play the fighting game against each version of MCTS as well as a standard MCTS version of the game. The participants then gave feedback on their experience answering the following questions:

- On a scale of 1-5 how difficult was this opponent? (1 being least difficult, 5 being most difficult)
- On a scale of 1-5 how enjoyable was playing against this opponent?
- On a scale of 1-5 how realistic of a challenge was this opponent?

Observing the three versions of MCTS, the ATROSAS version won 51% of the time and was voted as the most difficult version of the three. ROSAS won 46% of the time, and CSAS won 41% of the time. To reiterate, the goal of DDA is to keep the player in the Flow Channel. This was hypothesized to be accomplished with a challenge where the AI and player each win 50% of the time. So it was expected that ATROSAS would come closest to providing the most balanced gameplay (which would have been seen in high ratings of player feedback) because it came closest to winning 50% of the time. But a surprising finding of the study is that even though ATROSAS came closest to winning 50% of the time, ROSAS turned out to have the highest average of players vote it the most enjoyable and realistic to play against (see Figure 5).

Even though the ATROSAS implementation was the strongest of the three versions, as it had the highest win rate, ROSAS was ultimately favored by the players despite it not coming closest to a win rate of 50%. This result alludes to how DDA can certainly make the AI match (or even surpass) a player's skill. However, the players' favor of ROSAS over ATROSAS suggests that players want to feel challenged to a certain extent as well as win more than half their games. This illustrates that if DDA is too strong, it can negatively impact a player's game experience. DDA is supposed to provide a challenge, not aim to win. But if a player wins often, which is an observation of the CSAS version only winning 41%, player engagement may drop off.

3. REINFORCEMENT LEARNING IN TURN-BASED GAMES

Pokémon is a popular video game that utilizes a turn-based battle system. In turn based games, either the player or AI will choose an action, then the other will respond with their own choice of action.[2] Difficulty in Pokémon is intro-

[1]For specific details about how the value is calculated, see [4]
[2]In the experimental fighting game in Subsection 2.1, the game is in real-time. It updates continually, with no pauses or phases unlike with turn-based games

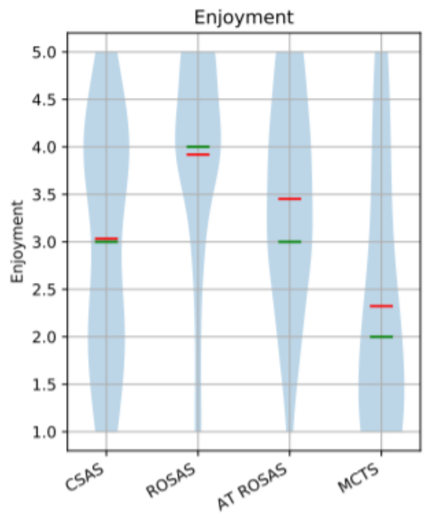
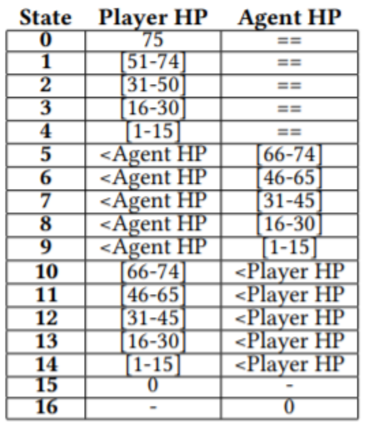


Figure 5: Particpants' response to enjoyment of the DDA implementations. The blue area is the distribution of scores. The red line is the mean, and the green line is the median. [4]



State	Player HP	Agent HP
0	75	==
1	[51-74]	==
2	[31-50]	==
3	[16-30]	==
4	[1-15]	==
5	<Agent HP	[66-74]
6	<Agent HP	[46-65]
7	<Agent HP	[31-45]
8	<Agent HP	[16-30]
9	<Agent HP	[1-15]
10	[66-74]	<Player HP
11	[46-65]	<Player HP
12	[31-45]	<Player HP
13	[16-30]	<Player HP
14	[1-15]	<Player HP
15	0	-
16	-	0

Figure 6: States of the environment. Each state is a different combination of hit points between the player and agent. [6]

duced by battling stronger Pokémon, strength being determined by levels and statistics such as attack and defense points to name a few. A limitation to this approach is that the strength of these opponents are fixed and do not adapt to the player's skill or experience. This lack of flexibility can be uninteresting to players as the game can become far too easy or conversely, far too difficult. In any case, it is not an effective contributor to maintaining players in Flow.

Pagalyte et al. [6] take a different approach with turn-based games by using a reinforcement learning algorithm to implement DDA on an experimental turn-based battle simulator. In the simulator, the player's character and AI's character are battling each other, trying to make the other's hit points reach 0. The simulator is essentially a turn-based fighting game.

3.1 DDA Implementation with Reinforcement Learning

In reinforcement learning, the AI needs to create a strategy of how it will choose an action depending on the situation that it is in. This strategy is called a policy. This decision making by the AI follows the framework of a Markov decision process which has the following elements:

- Agent: The decision maker for the AI. It is represented by the AI's character in the game.
- Environment: A contained world in which the player and agent interact with each other. The environment has game states that the agent can reach by taking certain actions. In this case, we are considering the battle simulator to be the environment.
- State(s): A situation in the environment which the agent can reach by taking certain actions.
- Actions: The possible decisions the agent can choose from at a certain state of the environment.
- Reward: The outcome of the agent choosing an action at a certain state. The reward is given to the agent, informing it about whether or not the action taken is helps the agent towards winning. [3]

In the simulator (the environment), there are 16 states defined by the health bar of the player and agent. Each health bar has 75 hit points, and the 16 states correspond to a different combination of the how many hit points the player and agent have (see Figure 6).

DDA in the simulator is implemented with the SARSA algorithm. SARSA (state-action-reward-state-action) is a reinforcement learning algorithm with the purpose of determining a policy for the agent to follow.[3] When the agent needs to choose an action to take, it will either choose an action with the highest reward aiming to win, or explore other states in the environment by randomly choosing an action to take (with the agent not trying to win). These concepts are called exploitation and exploration respectively. [11]

The experiment tests different rates of exploration and exploitation to observe which configuration(s) best keep players in a state of Flow. The rates being tested are (50,50), (30,70), and (70,30). (70,30) denotes that 70% of the time, the agent will explore and 30% of the time it will exploit.

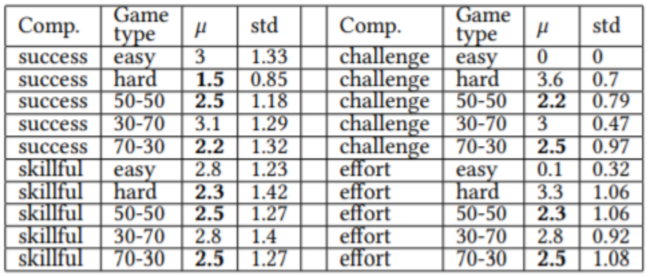
3.2 Experiment/Results

Two studies were conducted using the battle simulator implemented with the SARSA agent. The scope of this paper is focused on one of those studies, which is about the gameplay experience of players. The research question associated with the study is: Does a reinforcement learning approach improve the game experience in terms of a player's flow?

Ten players of different ages and skill levels participated in the study. Five types of games were played in the battle simulator:

- Easy: The simulator does not include SARSA and the game is preset to some easy difficulty
- Hard: The simulator does not include SARSA and the game is preset to some hard difficulty
- 50-50: The simulator includes SARSA. The exploration vs exploitation rate is set to (50,50)

[3]The mathematical details of SARSA are beyond the scope of this paper. More details on it can be found in [7]



Comp.	Game type	μ	std		Comp.	Game type	μ	std
success	easy	3	1.33		challenge	easy	0	0
success	hard	**1.5**	0.85		challenge	hard	3.6	0.7
success	50-50	**2.5**	1.18		challenge	50-50	**2.2**	0.79
success	30-70	3.1	1.29		challenge	30-70	3	0.47
success	70-30	**2.2**	1.32		challenge	70-30	**2.5**	0.97
skillful	easy	2.8	1.23		effort	easy	0.1	0.32
skillful	hard	**2.3**	1.42		effort	hard	3.3	1.06
skillful	50-50	**2.5**	1.27		effort	50-50	**2.3**	1.06
skillful	30-70	2.8	1.4		effort	30-70	2.8	0.92
skillful	70-30	**2.5**	1.27		effort	70-30	**2.5**	1.08

Figure 7: The feedback from players in the study from Subsection 3.2. Each row consists of a gameplay component (success, skill, challenge or effort), game type, mean and standard deviation. The mean values in bold are between or equal to 1.5 and 2.5 which is how balanced gameplay is defined. [6]

- 30-70: The simulator includes SARSA. The exploration vs exploitation rate is set to (30,70)
- 70-30: The simulator includes SARSA. The exploration vs exploitation rate is set to (70,30)

The Easy and Hard game types both do not have any DDA influence. This is tied to the idea of presetting a game to remain easy or hard the entire game, which is a technique of introducing challenge aside from DDA. Since the game types without DDA do not adjust the player's performance, it is hypothesized that the game types including SARSA (i.e. game types with DDA) would lead to a more balanced gameplay experience for the players.

Four components of player gameplay were the focus of this study: success, skill, challenge, and effort. Every player played each game type three times. Once they finished, they gave feedback on a questionnaire including the following questions associated with each component: Did you feel successful playing against each game type? How skillful or competent did you feel playing against each type? Was the game type challenging for you? Do you think a lot of effort was needed to play against each game type?

The questionnaire collected feedback with the Likert scale where 0 means "not at all" and 4 means "extremely". From Figure 7, the mean values in bold are between or equal to 1.5 and 2.5. Values in this range represent that players perceived the component as balanced under the game type. For example, row five of the first column of Figure 7 has a mean of 2.2. So in the game type including SARSA and an exploration vs exploitation rate of (70,30), players felt a balanced amount of success, since the mean is 2.2 and $1.5 \leq 2.2 \leq 2.5$.

The challenge and effort components have similar trends. The Easy game type offered players the least challenge and effort, while the Hard game type offered players the greatest challenge and required the players' greatest effort. In comparison to their counterparts, the game types including DDA were able to provide a more balanced experience in terms of challenge and effort.

A particular observation is that the 30-70 game type had no scores in what was deemed a balanced range, although it did have high scores for both challenge and success. This could imply that even though the 30-70 game was more challenging, players perceived a greater amount of success as a result of overcoming the challenge. This is not necessarily a negative outcome of the experiment, but it does not align with the goal of providing balanced gameplay. Regarding the other game types including DDA, the 50-50 and 70-30 game types had mean values between 2.2 and 2.5 for each component. The consistency of the data being within the range of a balanced experience suggests that a reinforcement learning agent can certainly provide an engaging and balanced challenge to players, helping sustain them in the Flow Channel.

4. DDA AND MULTIPLAYER PLAY

In a single player game, the challenge to a player is the AI. The level of difficulty can be controlled. In multiplayer games, games where players face off against one another, the challenge to a player is the other player(s). The question for game developers is then how to implement DDA and provide a balanced challenge for each player, but at the same time not inhibit the gameplay experience of other players.

In a study by Baldwin et al., around 150 participants were surveyed about their perspective on usage of multiplayer DDA (MDDA) in video games. Player performance has moments of high level and low level gameplay, especially when competing against other people. To account for the range of gameplay, players gave feedback from a high performance perspective (HPP) and low performance perspective (LPP). The players are questioned about instances of MDDA. An instance is defined as "a gameplay feature in competitive multiplayer video games designed to reduce the difference in challenge experienced by all players through adjusting the potential performance of certain players". [1] Essentially, instances are opportunities to balance a game so all players face the same level of challenge. An example of this is in the racing game Mario Kart. Players that are losing the race can receive a blue shell which slows down the racer in first place when used. This allows the other racers the chance to catch up to the leader.

4.1 MDDA Framework

An MDDA framework is defined with seven components revolving around how and when instances are utilized in multiplayer play. The survey asked the participants questions revolving around these components.

The first three components focus on when players are able to benefit from an MDDA instance and who receives it. The first component is Determination. A design decision must be made about if the MDDA instance should be instituted pre-gameplay or during gameplay. Pre-gameplay means the instance is instituted before a game starts and is based on a player's past performance(s). During gameplay means the instance is used during gameplay and is based on the current performance of a player. A related component to Determination is Automation. Automation deals with if the system gives players instances, or if players can give themselves instances (like a handicap). In some fighting games, players can choose to begin with a lowered health bar, giving the other player a handicap. In this case, the instance is manually applied. The third component is the Recipient of an MDDA instance. This component is only applicable if there are teams of players. Will a struggling player on a team receive an MDDA instance? Or will an entire team receive the boost?

The next two components are about how to access an MDDA instance. The Skill Dependency component focuses on whether or not a player needs to act with some degree of skill in order to utilize an MDDA instance. If the instance requires a player be skill dependent, then it provides players with the opportunity to benefit from the MDDA instance, the instance is not guaranteed. If no skill is needed, then the instance is skill independent. User action is the next component. To activate an MDDA instance, does the player need to interact with the game in any way? Or does the system enhance a player's gameplay without the player doing anything? User Action is not to be confused with Skill Dependency. User Action is simply about whether or not a player needs to interact with the game, such as pressing a button, to activate an MDDA instance. Skill Dependency can be described as a player having to earn an MDDA instance.

The remaining two components are about the Duration and Visibility of an MDDA instance. The Duration component is about how long an MDDA instance lasts. The instance can either be used once (single-use), more than once (multi-use), or used continuously over a brief period of time (time-based). The final component is Visibility. This concentrates on which players are explicitly told by the system that an MDDA instance is being received. Either the player receiving instance is notified, non-recipients are notified (which can include the recipient), or the system notifies no one about the instance being given.

4.2 Participant Feedback

The feedback from participants can be sectioned into three categories of interest: player control, personal benefit, and awareness of MDDA instances.

4.2.1 Player Control

A trend from the feedback emerged among the components of Duration, Skill Dependency, and User Action. From a HPP, players reported a positive game experience when MDDA instances were skill dependent, required user-action, and were single-use. They would prefer that all players have a single opportunity (not guaranteed) to receive assistance at a time of their choice. This resonates with the idea of skillful players wanting minimal system assistance.

From a LPP, players did not provide as much inclination as players from a HPP in regard to control of MDDA instances. Unlike their counterparts, they had a greater tolerance for MDDA influence in multiplayer video games as it helps them perform better. One consensus between HPP and LPP players is the Automation component of an MDDA instance. By letting the game automate MDDA instances, players cannot abuse game assistance to elevate their gameplay. Additionally, players were discouraged by being given the choice to use an MDDA instance as choosing to use it connotes weak gameplay skills.

4.2.2 Personal Benefit

Players from a LPP found overall enjoyment in components allowing MDDA instances to help them perform better than they could without it. This means preferring multi-use MDDA instances over single-use and time-based ones. Skill independence was also favored by LPP players, because they could take advantage of an MDDA instance without having to earn it. Players from a HPP found enjoyment when the impact of MDDA instances were minimal as it allows their skill to not be overshadowed by game assistance. Examples again being single-use instances that are skill dependent.

4.2.3 Awareness of MDDA Instances

The Visibility component was important to players from both a HPP and LPP, because it allows transparency with MDDA usage. If LPP players are told they are being given an MDDA instance, then they have a better chance to make use of the assistance. If LPP players are aiming to improve at a game, then being told when they are being assisted can help them gauge their skill level. If HPP players are informed of an MDDA instance presence, then they can try to account for the added effects of a player receiving the MDDA instance. This also can give them a boost of confidence in their own skill at the game, knowing they do not have to rely on system assistance.

Section 2 and Subsection 3.1 both conducted experiments with implementations of DDA. Despite only a survey revolving around MDDA instances being done, players' feedback on their past experience and biases when it comes to MDDA is valuable. Game designers can gain substantial insight into what players are desiring in terms of game balancing during multiplayer play.

5. CONCLUSIONS AND FUTURE WORK

DDA usage continues to grow in the video game industry. The studies analyzed in this paper have shown how DDA can be effective in providing a balanced and enjoyable game experience for players, but game developers must consider whether or not it impedes too heavily on players' gameplay. DDA implementations that are too strong, with the intention of winning rather than providing a challenge, can discourage lesser skilled players from continuing to play. Especially in multiplayer play, players who perform highly and value control are hesitant to accept their gameplay (or their opponents) being interfered with, because success is not entirely dependent on one's own skill.

The studies looked at in this paper used simple experimental games in their experiments. Future research about DDA currently being used in complex games and different genres of games can help us better understand what elements of a game keeps players in the Flow Channel and potentially lead to wider acceptance of DDA. [6] The motivations of why players play video games is another topic of interest. The idea of casual play focuses on adventuring and playing relaxed. How does DDA impact those types of players? [4] In any case, the future of DDA in video games holds plenty to be excited about when considering the increasing complexity of video games as well as the evolving technology used to create it.

Acknowledgments

I would like to thank my advisor Kristin Lamberty, Elena Machkasova, Nic McPhee, and Tim Snyder for their insight and wisdom over the research process.

References

6. REFERENCES

[1] Alexander Baldwin, Daniel Johnson, and Peta Wyeth. Crowd-Pleaser: Player Perspectives of Multiplayer Dynamic Difficulty Adjustment in Video Games. In *Proceedings of the 2016 Annual Symposium on*

Computer-Human Interaction in Play, CHI PLAY '16, page 326–337, New York, NY, USA, 2016. Association for Computing Machinery. Accessed: 9-September-2021. URL: https://doi-org.ezproxy.morris.umn.edu/10.1145/2967934.2968100, https://doi.org/10.1145/2967934.2968100 doi:10.1145/2967934.2968100.

[2] Daniel Berube. The Flow Theory Applied to Game Design, Apr 2019. Accessed: 17-October-2021. URL: https://thinkgamedesign.com/flow-theory-game-design/.

[3] deeplizard. Markov Decision Processes (MDPs) - Structuring a Reinforcement Learning Problem Youtube, Sep 2018. Accessed: 15-October-2021. URL: https://www.youtube.com/watch?v=my207WNoeyA.

[4] Simon Demediuk, Marco Tamassia, Xiaodong Li, and William L. Raffe. Challenging AI: Evaluating the Effect of MCTS-Driven Dynamic Difficulty Adjustment on Player Enjoyment. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW 2019, New York, NY, USA, 2019. Association for Computing Machinery. Accessed: 3-September-2021. URL: https://doi-org.ezproxy.morris.umn.edu/10.1145/3290688.3290748, https://doi.org/10.1145/3290688.3290748 doi:10.1145/3290688.3290748.

[5] Simon Demediuk, Marco Tamassia, William L. Raffe, Fabio Zambetta, Xiaodong Li, and Florian Mueller. Monte Carlo Tree Search Based Algorithms for Dynamic Difficulty Adjustment. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 53–59, 2017. Accessed: 3-September-2021. https://doi.org/10.1109/CIG.2017.8080415 doi:10.1109/CIG.2017.8080415.

[6] Elinga Pagalyte, Maurizio Mancini, and Laura Climent. Go with the Flow: Reinforcement Learning in Turn-based Battle Video Games. In *Proceedings of the 20th ACM International Conference on Intelligent Virtual Agents*, IVA '20, New York, NY, USA, 2020. Association for Computing Machinery. Accessed: 7-September-2021. URL: https://doi-org.ezproxy.morris.umn.edu/10.1145/3383652.3423868, https://doi.org/10.1145/3383652.3423868 doi:10.1145/3383652.3423868.

[7] Pankaj Porwal. SARSA (State Action Reward State Action) Learning - Reinforcement Learning - Machine Learning - Youtube, Apr 2020. Accessed: 20-October-2021. URL: https://www.youtube.com/watch?v=FhSaHuC0u2M.

[8] Rahul Roy. Ml: Monte Carlo Tree Search (MCTS), Jan 2019. Accessed: 2-November-2021. URL: https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/.

[9] Stannered. File:tic-tac-toe-game-tree.svg, Apr 2007. Accessed: 10-October-2021. URL: https://commons.wikimedia.org/wiki/File:Tic-tac-toe-game-tree.svg.

[10] Matheus Weber and Pollyana Notargiacomo. Dynamic Difficulty Adjustment in Digital Games Using Genetic Algorithms. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 62–70, 2020. Accessed: 7-September-2021. https://doi.org/10.1109/SBGames51465.2020.00019 doi:10.1109/SBGames51465.2020.00019.

[11] Wikipedia. State–action–reward–state–action, Sep 2021. Accessed: 10-October-2021. URL: https://en.wikipedia.org/wiki/State-action-reward-state-action.