Dynamic Difficulty Adjustment in Flappy Bird

Carl Klier, Vibhav Nandavaram Abbai Srivaths, Neill Robson
Department of Computer Science
NC State University
{caklier, vsrivat, nlrobson}@ncsu.edu

Abstract

Video games must strike a balance between leaving players bored by the game's ease, or overwhelmed by its difficulty. Existing strategies addressing this problem are largely static, where difficulty curves and customization options are hard-coded in advance in hope that the result will appeal to a wide enough audience. Real-time, dynamic strategies for difficulty adjustment show promise for drawing a more diverse audience, but they require fine-tuning to be effective. By implementing a Dynamic Difficulty Adjustment algorithm in a simple video game, we hope to distill the essential elements of a well-optimized strategy that can be applied to larger projects.

Introduction

Video games have become an extremely widespread mode of entertainment in recent years (McGonigal 2011), as well as being implemented in the fields of education and healthcare (Bleakley et al. 2015). In the light of the current pandemic, more people than ever before have turned to gaming to cope (Marston and Kowert 2020). With this influx of new players, video game developers have come to recognise the challenge of balancing gameplay to be appealing to both novice players and skilled veterans. In recent years, there has been increased discourse over the accessibility of video games to a wider audience. The difficulty of a video game is one aspect that developers need to constantly tune and balance so that novices, casual players, and veterans can all experience a satisfying and fair challenge. Conventionally, most video games come with difficulty settings or levels that players can choose to play on to their liking. These difficulty levels are hardcoded into the game, and the gameplay properties of each difficulty level are static as a result. Players normally have to manually change difficulty settings should they wish to change their gameplay experience.

However, static difficulty levels can result in players struggling to find the correct level of difficulty that suits their specific playstyle and desired challenge. In his book "Flow: The Psychology of Optimal Experience", Csikszentmihalyi describes a flow channel between the states of boredom and frustration in which one has an optimally challenging yet fulfilling experience (2018). This can be adapted to video games, where the state of boredom arises when the player is faced with opponents that are too easy to beat, and the state of frustration comes about when the player faces opponents that are seemingly too difficult or unfair.

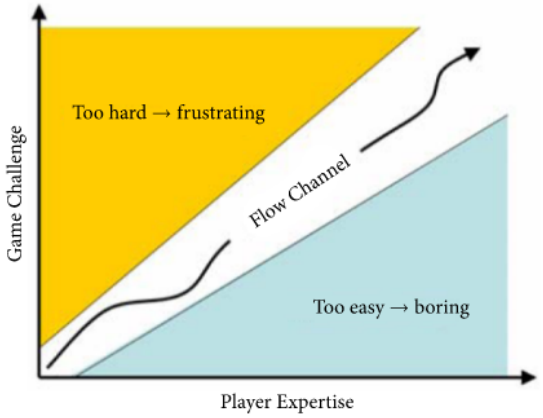

Figure 1: Flow Control Graph

From this scenario, it would make sense to implement a system that adjusts gameplay parameters and difficulty in real time to a player's skill level and ingame actions. This is called Dynamic Difficulty Adjustment (DDA), which is a technique that games can use to automatically change game parameters, opponent behavior, and scenarios in real time in response to a player's skill level and ability. DDA constantly adjusts the game's difficulty to ensure the player is never bored by how easy the game is, or put off by how hard it is to win—thus ensuring that the player remains in the flow channel, engaged with the video game.

Background

DDA is an AI based system designed to change in-game scenarios and parameters to match player performance, ensuring that players of any skill level remain engaged throughout a game session (Sepulveda, Besoain, and Barriga 2019). This can be achieved if DDA adheres to the three basic concepts:

- The game needs to track player skill and ability
- The game needs to actively adapt its difficulty to match

the skill level of the player in order to keep the player in the flow channel

- The changes in difficulty should not be perceptible to the players and should maintain a degree of consistency with successive game sessions

For DDA to adapt to the player's performance, one would need to model the factors that decide a game's difficulties and what specific performance metrics the DDA algorithm must consider before changing the difficulty. There are a variety of different methods that academia and industry have explored to achieve DDA techniques in video games, such as (Lach 2015):

- Dynamic scripting
- Multi Layered Perceptions
- Reinforcement Learning
- Neural Networks

The principle behind DDA is based on the understanding that the longer the user engages with a video game, the more likely that the video game will be successful. However, one of the most important things is to ensure that the video game continues to challenge the player in a fair and balanced way, potentially adapting to the change in the player's proficiency and skill. Conventionally, video games have either implemented difficulty levels or completely done without them, sticking to a single gameplay experience. Difficulty levels usually come with changes to specific game or environment variables, such as (Aponte, Levieux, and Natkin 2009):

- Different enemy/powerup placement
- Changes in damage dealt by enemies
- New Enemy Attack patterns
- New zones/levels, or changes to existing ones

Problem

In this project, we aim to assess if DDA improves the engagement and performance of players at different skill levels. To analyse the effect of DDA on players, we seek to implement DDA in Flappy Bird, a game notorious for its brutally coarse control scheme yet addictive gameplay loop. We hypothesize that the implementation of DDA in Flappy Bird may improve the performance of players. Their changes in performance will be measured across multiple sessions of at least ten runs each, where each session will have subjects playtest either the baseline version of the game or the DDA-augmented version.

Potential Benefits

Humans in a comfortable, secure environment are typically fulfilled by pursuing internal thoughts and desires, rather than external rewards and consequences. Psychologists have proposed a three-pronged model for this internal motivation: a secure human being is fulfilled by pursuits that maximize autonomy, mastery, and purpose (Pink 2009). Autonomy is simply the desire to make one's own decisions, and purpose is an abstract concept of ethical significance. Mastery, however, is the drive for constant improvement and growth in one's skill level. A sense of mastery comes from being neither bored nor overwhelmed with challenge. Achieving such a balance in a video game could provide immense benefits to the user experience and cognitive science research communities.

Since player diversity is so large, having a predefined difficulty setting does not necessarily equate to an equal experience for all players. Each person playing the game has a unique set of skills and playing style that can cause them to have a different reaction to in-game scenarios (Xue et al. 2017). Dynamic difficulty adjustment helps to alleviate some of the negative consequences of inherent player diversity by better tailoring the game to the individual in order to maximize engagement and provide a better overall experience. Games are the most fun when they propose a challenge that matches well with a player's skill level. The goal of dynamic difficulty adjustment is to ensure that a game's difficulty is neither too hard, causing the player to become overwhelmed and therefore quit the game, nor too easy, causing the player to become bored and also therefore quitting the game. A successful dynamic difficulty adjustment algorithm sufficiently challenges the player, but gives them a reasonable chance of success, to increase the overall time spent playing the game (2019). In this way, dynamic difficulty adjustment opens up new opportunities for games to become more accessible, for players to feel a bigger sense of enjoyment, and for the game to ultimately achieve more widespread success.

Research Questions

The following research question is posed under the assumption that better player performance will lead to higher player engagement. We claim that, the higher the player's score, the more tasks they have completed in the game.

- Does player performance—measured in terms of obstacles cleared—exhibit sustained growth in the DDA-augmented form of Flappy Bird?

We hypothesize that adding DDA to Flappy Bird will improve the engagement and performance of players at a wide variety of skill levels. The performance metric described in our research question and outlined in greater detail below will be used to analyze the correctness of our hypothesis.

Preview of Findings

Player's playing Flappy Bird with our DDA algorithm enabled achieved higher scores than players playing a baseline version of the game with constant difficulty. However, our results do not show that player performance exhibits sustained growth. Over the course of multiple runs of the game by different players, we did not see a steady increase in player scores. However, since player scores with DDA enabled were overall higher, based on our assumptions made in the section above, adding our DDA algorithm to Flappy Bird increased player engagement.

Literature Review and Related Work

A study in 2013 (Alexander, Sear, and Oikonomou 2013) explored the effects of game difficulty found that players prefer

that the difficulty of a game be reflective of their gaming experience instead of their actual proficiency and gaming skill. This presents us with the reason why it is important that any DDA implementations be subtle enough that the player does not find the game less interesting to play, or that they are not being given a chance by the game to improve before the game's systems "shift the goalposts" for the player's victory. This notion is further supported in a journal paper by developers from Electronic Arts (Xue et al. 2017), stating that DDA methods exhibit "diversity in the levers that adjust difficulty", ensuring that the game can predict and intervene in gameplay should the predicted future game state be an undesirable one. DDA is inherently a temporal player model, where the game's systems have to capture changes in player behaviour and experience over the course of experiences, needing to forecast several steps into the future of the player's gameplay trajectory. As the player keeps playing, the DDA algorithm needs to solve an optimization problem to maintain the optimal difficulty (Zook et al. 2012).

Another important facet to implementing DDA, is the intent of dynamically changing the difficulty of the game. One reason for using DDA would be to ensure that players progressively improve their performance over time, while they remain in the flow channel of engagement (Csikszentmihalyi 2018). Another reason for implementing DDA is to ensure that the overall difficulty of the game is tuned to push players to buy in game currency or microtransactions, such as in mobile games. For instance, Candy Crush Saga, and Bejeweled (Zook et al. 2012) both have DDA algorithms tuned to ensure that churn rates of players are reduced. In the case of Candy Crush Saga, DDA ensures that there is enough escalating yet optimum increases in puzzle difficulty without entirely tiring out the player, and yet the player does not feel like the puzzles are unrewarding to solve. This Goldilocks zone of difficulty has the highest probability investing in microtransactions to bypass difficult puzzles instead of outright abandoning the game. However, in recent years, the implementation of DDA has been regarded with suspicion and controversy with EA patenting a specific DDA implementation strategy at increasing monetization in games (Arts 2016). This move has been met with backlash from the gaming community resulting in a lawsuit (Yin-Poole 2020) that was later withdrawn over the potential for exploitation over overt monetization and addition of lootboxes. This is another reason why DDA algorithms need to be subtle in their implementation, and the intent behind their implementation is also important to establish.

Historically, there have been several different dynamic difficulty adjustment strategies. For instance, the Hamlet system is a set of libraries embedded into Valve's Source Engine, the game engine most famously known for being used in the Half-Life game series (Hunicke and Chapman 2019). The Hamlet system has functionality for tracking game statistics in real time, defining and executing difficulty adjustment strategies, and so on. These functionalities are designed to predict if the player is "flailing"—meaning the player is repeatedly edging towards a state that they cannot progress forward from. Some of the specific systems that the Hamlet library uses to detect flailing behavior, include:

- The player's current location in the level
- How often they have died at a specific level in the game
- How often they have repeated the current encounter
- Their current health levels, ammo count, and inventory levels

If a player repeatedly dies at a specific encounter, the Hamlet system intervenes in an iterative and subtle manner, which means that health packs could have greater chances to drop from crates, the strength of the player's ammo might be increased, or the overall damage taken from enemies might be reduced slightly as well. In recent years, these functionalities have been implemented using AI and ML techniques as well (Hunicke and Chapman 2019).

For our project we aim to design and implement our own DDA algorithm and analyse its effect on gameplay and engagement on Flappy Bird, a mobile game. We chose Flappy Bird specifically for the simple gameplay system, easy to track player performance, and potential for experimentation and tweaking the difficulty of the game. While Flappy Bird might not require a prediction based model to achieve DDA, we can still engineer our own version of as system that adjusts dynamically to the performance of a player similar to the Hamlet libraries from Valve's Source Engine.

Tasks and Techniques

Our desire is to augment the Flappy Bird game with a dynamic difficulty adjustment routine. This algorithm should be as non-invasive as possible: we don't want to add any mechanics or behaviors to the game that players would not expect, and we don't want the difficulty adjustments themselves to be obvious to the engaged player. The following sections outline the details of the original Flappy Bird game, as well as how we propose to achieve the minimal and sufficient DDA implementation described here.

Defining the Game

The goal of Flappy Bird is to direct the avian character through obstacles that appear on the screen. The game screen follows the bird, moving to the right (and nearer to the obstacles) at a constant rate. Gravity affects the bird, so it will rapidly descend to the ground if the player does not intervene. However, the player may control the bird through a single tap or button directing the bird to "flap" once (giving it a brief upward force against gravity). If the bird collides with the ground or any side of the obstacles, the game is over.

The obstacles are quite simply pairs of pipe-shaped walls, extending from the sky and ground, making a passageway somewhere in the middle of the screen. The player is scored by how many pairs of these pipes they can progress through without colliding. In the original game, every increment of ten pipes represented a new tier of achievement: players would receive a bronze, silver, gold, and platinum medal for reaching ten, twenty, thirty, and forty pipes cleared, respectively (Moscaritolo 2014).

In the original game, the only variable that changed as the game progressed was the vertical position of each passageway on the screen. No matter how long a player survived, the

height of each passageway remained unchanged, as well as the horizontal distance between each passing obstacle pair. No variations in the gravitational force, strength of each flap, or other environmental features were observed. Based on the condition for the highest achievement tier (passing forty obstacles), we claim that the static difficulty level was an intentional choice by the developer. A "full run" of the game (forty obstacles cleared) would only take around eighty seconds to complete—too short of a time span to include significant difficulty variations.

Clearly, a game with a static difficulty level requires some level of modification to make DDA a meaningful addition. As discussed in the Introduction, DDA typically impacts a game's attributes, behaviors, and events (2019). We believe that, in the case of Flappy Bird, a minimal and viable DDA implementation need only touch the game's attributes to effectively (and discreetly) improve player performance and engagement. As subsequent sections will discuss, we only propose to modify the environmental factors (pipe distance, gravity, etc.) over time, and *not* add any new behaviors or events to the game.

Performance Metrics for our DDA framework

In a conventional game of Flappy Bird, a player continuously generates certain values that can help a DDA algorithm assess the performance of a player and dynamically adjust difficulty. These values could be categorised as "run-specific" variables generated over the course of a single run, and "session-specific" variables that change over the course of a set of runs. To make the distinction clearer between the two types of variables, let us consider a hypothetical player who plays 10 consecutive games of vanilla Flappy Bird.

In one specific run of the game, the player may or may not be able to clear all 40 obstacles needed to "finish" the run. This may be due to inconsistently clearing the pipe obstacles, or simply failing to tap the screen enough times to keep the bird in the air. Here, we can observe and record "run-specific" variables, such as:

- The number of times the player taps the screen in between obstacles
- The average position between the pipes at which the player clears an obstacle
- The average height of the Bird in a run

These variables can help better inform the DDA algorithm which aspects of the game need to be dynamically adjusted to maintain a fair but challenging run. For instance, if the player is rapidly tapping the screen, it can possibly mean that the player is struggling to maintain the bird's altitude. The average position between the pipes at which the player clears an obstacle can indicate if the player is easily clearing the pipes roughly in the middle of the gap, just barely making over the bottom pipe, or nearly hitting the top pipe. The above example variables can inform how the game adjusts the gravity of the game environment as the run progresses, or can also allow for the DDA algorithm to adjust the distance between the top and bottom pipes in obstacles.

On the other hand, "session-specific" variables would be collected over the course of a set of runs, and can indicate how player performance affects consecutive runs as well. Some of these variables include:

- The average length of runs
- The average number of obstacles cleared in a session
- The time gap between successive runs
- The average location of player deaths

These variables can be used to analyse how effective the DDA algorithm is overall, as we can infer the engagement of the player over a session. Should the average number of obstacles cleared decrease over the course of a session, it can indicate a decline in player performance or interaction with the game. The time gap between successive runs can also indicate how quickly players want to engage with the game after possibly losing or finishing a run successfully. The average location of player deaths refers to where the player often hits an obstacle or falls to the ground with respect to the number of obstacles passed. Another way to regard the location of player deaths would be to analyse specifically where on an obstacle the player is making contact. For instance, if a player makes contact with the left side of the obstacle's bottom pipe, the DDA algorithm could adjust the gap between the vertical pipes with an arbitrary multiplier relative to the position of contact and height of the bottom pipe.

In our hypothetical scenario, a session of 10 runs for a player may vary greatly in terms of these variables, allowing us to ascertain the player's skill level, and how the DDA algorithm reacts to the player's actions. A skilled player would gradually find that over the course of a session with successful runs, the game begins to increase gravity, narrow the gaps in obstacles, increase the speed of the game, or even spawn the obstacles closer together. A novice would instead see their session of the game change on these same fronts, but in ways that balance the difficulty to their level. In the case of both these categories of variables, we can develop a framework for our DDA algorithm to tweak specific properties of Flappy Bird itself.

Our DDA Strategy

We propose using a multiplier and metrics framework for implementing dynamic difficulty adjustment (Sutoyo et al. 2015).

To create the adjustment values for our algorithm, we will use multipliers that affect parameters of objects in our game. The three parameters that could change depending on player skill are:

1. the space of the gap between two pipes vertically
2. the distance between sets of vertical pipes
3. the gravitational force acting upon the bird.

However, after doing some initial play testing comparing a version of the game that has static gravity versus a version of the game in which gravity changes, we realized that modifying gravity (either increasing or decreasing) makes the game more difficult since it throws off the natural rhythm that the player falls into with making the bird flap its wings to rise up against gravity. One of the more important aspects

to DDA is that a DDA algorithm shouldn't modify the fundamental mechanics of the game, and after playing those two versions of the game, we felt like consistent gravity was a fundamental mechanic of flappy bird. Thus, our algorithm focuses on only modifying the space of the game between two pipes vertically and the distance between sets of pipes.

The multipliers that are used to modify the parameters above are:

- a pipe gap multiplier which changes how much we increase the pipe gap each run of the game
- a pipe distance multiplier which changes how far the distance is between sets of pipes

Initially, all the multipliers are set to 1. One of our goals with our DDA algorithm was to help worse performing players feel accomplished by helping them to achieve the first medal, a bronze medal, for achieving a score of 10. Thus, if the player dies and they did not get a score above 10, we increase the pipe gap and the pipe distance multiplier by some small amount, in our case, .01 and .0075 respectively.

If a player is currently playing and they are having a really good run, we don't want the game to stay at that current level of difficulty. We want to increase the difficulty mid-run so that the player doesn't feel bored in the middle of their current playthrough. In this case, starting at a score of 15 and then every 5 more pipes passed successfully, our algorithm decreases the pipe gap multiplier and the pipe distance multiplier by .013 and .01 respectively.

At the end of each death sequence, before the next run, we update the parameters by multiplying their original value by their related multiplier and add that to the original value. For example, if the player died by hitting a pipe, the new pipe gap parameter would be equal to the pipe gap parameter + pipe gap parameter * pipe gap multiplier where an increase in the pipe gap parameter lessens the difficulty. In this way, over time, the game's difficulty will better match the player's skill level.

In addition, one of the problems we began to see occur was "slingshotting." This is where players would start off with low scores so our DDA algorithm would make the game easier. But the game would get too easy and players would get really high scores, so our algorithm would kick in and make the game harder causing the player to get really low scores again. To prevent this "slingshotting" and to prevent the multipliers from contributing excessively to this problem, if a player got a score of over 30 and died, we would reset the multipliers back to 1. This helped in that if the game was already hard for a player (since after a score of 15 the multipliers start decreasing), the game wouldn't continue to get harder while the multipliers took a while to get back to being above 1.

It's important to note that these values used to modify our parameters were picked after just acquiring a little bit of playtesting data. With more data from players, these increases and decreases to the multiplier values could be better tuned to achieve a more effective DDA algorithm. It's also important that these values aren't large enough that they modify the game too much and change the fundamentals of game-play. In addition, we don't want the variable modification to be noticeable by the player in keeping with one of the standards of a DDA algorithm.

Evaluation Methods

A player's performance in Flappy Bird is measured only by the number of pipe barriers passed without collision. This metric implicitly takes session duration into consideration, as pipes appear on screen at a fixed rate. The longer a player is able to survive, the higher their final score will be.

Due to the conflation of these two variables, we propose that a simple rolling average of a player's most recent scores is an adequate litmus test of DDA's impact on performance and engagement. The number of data points collected per session tacitly encodes how long a player spent engaged with the game, and the slope of the regression line provides a coarse indicator of performance improvement across the session.

Note that both the quantity of data points per session *and* their slope must be taken into consideration during analysis. On one hand, a player could play hundreds of games and still fail at the first pipe (no performance improvement). The other (more foreboding) potential is that DDA could be excessively generous to players, causing their score to skyrocket on their second playthrough (creating a steep positive slope) without effecting a real increase in skill level. As mentioned in the Potential Benefits section, we are operating under the assumption that human players are motivated by the urge to become more skilled in a task. We expect that a player sensing the development of skill-based mastery will not "churn" as readily, leaving the game out of boredom. Since perceived mastery is distinct from reported performance—anyone can receive a fabricated award—we must include the session duration (number of data points) in our consideration.

Without an existing corpus of session data to analyze, it is difficult to propose a statistically-sound method of joining these two variables in analysis. One alternative is to run several play sessions with a control group on a version of Flappy Bird without DDA. Using the average play session length as a pivot, we can drop any experimental sessions whose length did not meet or exceed that average control session duration. Similarly, when looking at the slopes of performance improvement over time, we can take a weighted average of slopes within each group where slopes associated with longer total play times are emphasized.

All of these considerations necessitate extensive "playtesting" of both versions of the game, preferably blind to the players themselves. Given the limited time available to us as both developers and researchers, we opted to publish a Web-accessible version of our game that can be distributed to anyone with a web browser. After tying each participant to a unique identifier, we would ask them to open the game (and play ten or more rounds) at least once per day for three days. The date of the session (and the rolling performance averages) are automatically recorded by the system for every player.

Collision and other Performance Variables

In order to track the performance of the players and understand trends in their runs over the course of a play session, we introduced a set of variables in specific sections of the game. These variables are meant to function as parameters for the DDA algorithm to make the game easier for low performing players.

For instance, we added a variable called collisionPosition which detects if the player collides with either the pipes, or the ground. CollisionPosition can help better inform the DDA algorithm whether to modify the game's gravity, time intervals in which pipes spawn on screen, and even the height of the pipe gaps as well. This variable is currently added to save data at the end of each run, and the DDA algorithm makes the necessary changes before the next run begins. Other variables or changes are planned that can detect the height at which the player collides with a pipe, and also whether the player collides with the top or the bottom pipe as well.

CollisionPosition and the player's High score are currently the main variables that we are using to chart player performance. The original version of Flappy Bird awards the player medals for reaching a specific high score. For our initial implementation, we chose the lowest high score of ten required to get a Bronze medal as the threshold score the DDA algorithm helps players achieve. For each consecutive run in which a player fails to clear the threshold score of ten, the DDA algorithm checks for whether the player collided with the ground or the pipes through collisionPosition. This then leads the algorithm to either gradually decrease the game's gravity, or additionally also gradually increase pipe height and time intervals in which the pipes spawn to make the game easier to play. Conversely, the algorithm also makes the game harder as detailed previously in our DDA strategy.

Collecting these variables over the course of a run allows for us to chart the performance of players over the course of a play session, though it also requires us to find a way to save the variable values in the browser itself over time.

Save Data Strategy

During the course of development, we found that tracking player performance history would pose a significant hurdle in several areas:

- How will a player be uniquely identified and paired with a data set, if runs are spread over multiple sessions and potentially multiple devices?
- How do we keep the data size manageable over a large number of runs and play sessions?
- How can we collect the data from our participants (and ourselves) without exposing the participant to their own performance history? This exposure could affect how a player behaves in future runs, e.g. causing them to purposely slam into the side of a pipe in order to achieve a certain parameter change.
- How can we minimize the amount of unintentional user error in the system?

Although the limited scope of the project prevented us from addressing all of these issues in detail, we outline some of our considerations and strategies in this section.

Because our Flappy Bird implementation is browser-based, storing save data in the browser's cookies was a natural choice. The original implementation already put the player's high score into a browser cookie, and we only needed to extend that functionality. When a new browser accesses the site for the first time (indicated by the lack of a cookie), a UUID is generated identifying the player and stored in the new save-data cookie.

Cookies last over many browser sessions by definition, but they are not easily shared between devices. A consequence of this fact is that a singular player playing the game on device 1 would be recorded as a different player if they played the game on a different device 2. We decided to compromise in this regard, with the justification that different devices often necessitate different motor skills for gameplay (tapping the screen versus clicking a mouse). Furthermore, if we so desired, we could mark multiple UUIDs with a single player identity if, during the collection process, we asked players to provide their name or other identification information.

Early on in the development process, we saw a tight correlation between the data that we were collecting and processing to adjust gameplay difficulty, and the data that we were saving in the browser's cookie storage. Since the difficulty information was entirely derivable from a simple history of basic performance metrics (e.g. duration of run, cause of death), we strictly limited our save data to these "bare-bones" performance indicators and then used them both for in-game difficulty adjustment *and* post-experiment data collection/analysis.

In addition to giving ourselves a strict "budget" of what data to collect, we also Base64-encode the data before storing it or presenting it to the user. This has the dual benefit of further compressing the data for storage/transmission and giving it a one-pass obfuscation to prevent players from unintentionally being affected by seeing their own performance results. Of course, such an encoding does not prevent the curious/malicious player from simply decoding the presented string and seeing the raw JSON object storing their data. We did nothing to prevent such behavior aside from asking for truthful reporting, due to the limited scope of this project.

Our Current Results

Using our implementation of our DDA algorithm for Flappy Bird, we were able to gather player data from multiple different players. We had some players play through runs of Flappy Bird running out DDA algorithm and some players play a baseline version of Flappy Bird in which the values of the variables remained constant. Over the course of the playertester's session, we recorded game metrics using our save data strategy mentioned above. A single run of the game saved gameplay statistics that looked like:


```
{
    "collisionPosition": 11,
```

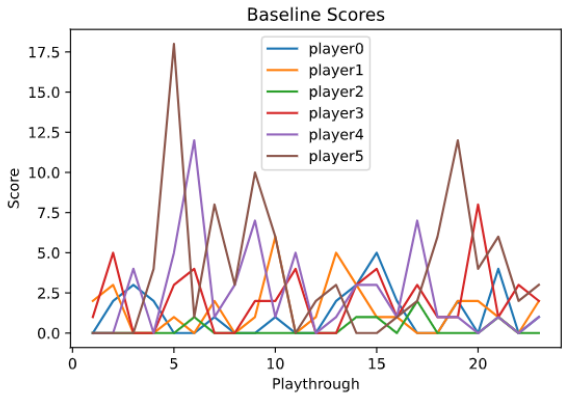

Figure 2: Player scores from baseline version

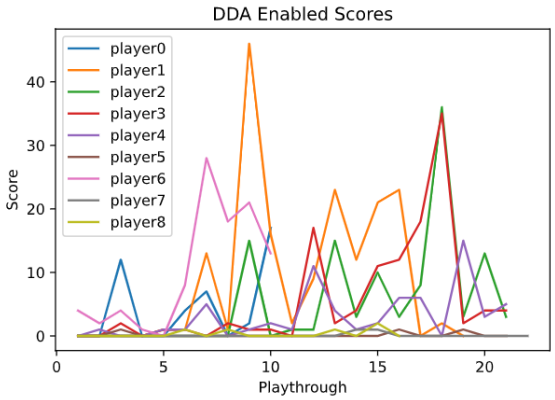

Figure 3: Player scores from DDA enabled version


```
    "duration": 16416,
    "gravity": 0.25,
    "pipeInterval": 1450,
    "pipeheight": 92,
    "score": 6,
    "startTime": 1617827921756
}
```


The main recorded metric we examined was score as our research question and hypothesis revolved around a player's performance increasing over time as the DDA algorithm helped low skilled players achieve better scores. We also looked at how the height of the pipe gap, called pipeheight here, and the distance between pipes, called pipeInterval, for a player increased or decreased over the course of their game playthroughs.

Interpretation of Results

Looking at our results in Figure 3, which were derived from playtesters playing Flappy Bird running our DDA algorithm, we can see that scores start low and then, around playthough 5, our multipliers get large enough that we start to see an increase in player scores. However, contrary to our hypothesis, our results do not show that our DDA algorithm does

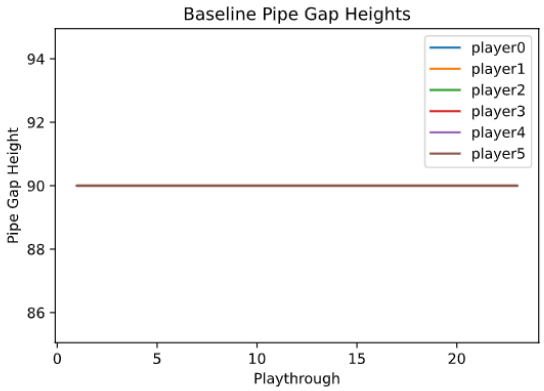

Figure 4: Pipe Gap Height from baseline version

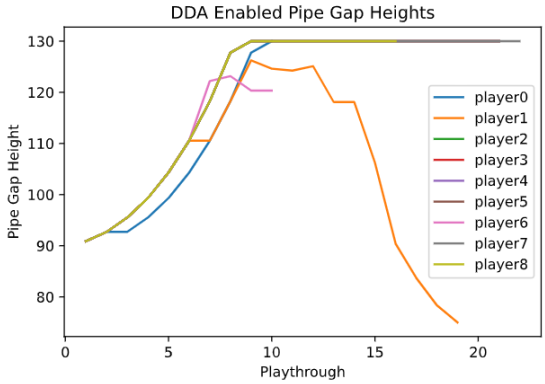

Figure 5: Pipe Gap Height from DDA enabled version

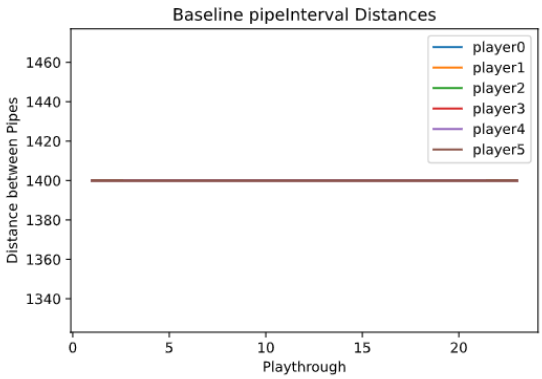

Figure 6: Pipe Interval Distance from baseline version

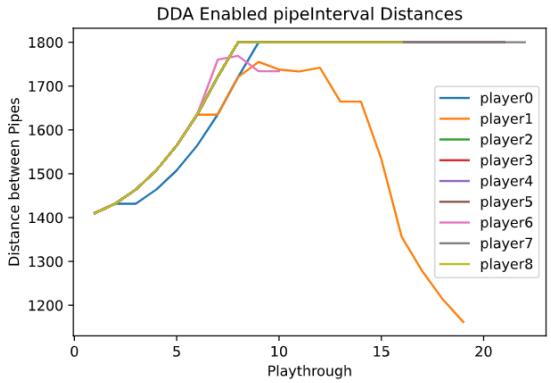

Figure 7: Pipe Interval Distance from DDA enabled version

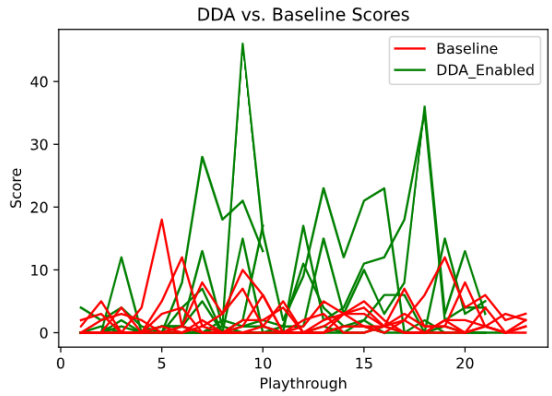

Figure 8: Scores from DDA enabled version against scores from baseline version

a good job improving engagement and performance over a longer period of playthroughs as player scores do not show sustained growth. In addition, some players failed to exhibit any growth in scores and also failed to achieve our desired goal of earning a bronze medal.

But, as Flappy Bird is an inherently difficult game, the graph in 8 shows that, overall, our algorithm does help players get higher scores than they would normally get playing a non-DDA assisted version. And that can be interpreted, by our initial assumption, as an overall increase in engagement when compared to the baseline version of the game.

Figure 5 shows the value of the pipe gap height parameter at the end of each playthrough. You can see that player1—the orange line—had a significant drop off in the pipe gap height starting at around playthrough 9. This correlates to player1 getting a high score of over 40 in playthough 9. This means our DDA algorithm saw player1 doing really well and tried to make the game harder for him over time by decreasing the pipe gap multiplier and thus the distance between the top and bottom pipes. While the pipe gap continued to decrease, so did player1's scores. This shows that our algorithm did indeed make the game harder for player1 after he had an easy time achieving the highest medal in the game. The same can be seen in Figure 7.

Figure 5, when looking at the players with straight lines from playthrough 8 onwards, shows that some players, even with our DDA algorithm enabled, were not able become successful enough at the game for our algorithm to begin to make the game more difficult for the player by eventually decreasing the gap between the pipes. There does exist a delay between the *multiplier* beginning to decrease, and the actual *game parameters* decreasing in turn. Maybe with more practice and by playing more than just 20 playthoughs, these players would become better at the game and get to a point where eventually our algorithm would start to increase the difficulty of the game. In addition, with a more fine tuned and better suited DDA algorithm, we could help these players become more successful at the game and achieve higher scores.

While our data does show some initial promise of a successful DDA algorithm, we remain confident that with more player data and more thorough testing and analysis, a future iteration of our DDA algorithm could confirm our hypothesis.

Future Work

As we mentioned in the introduction, our primary qualitative goal with this project is to discover and sustain a flow channel, or "Goldilocks zone," of difficulty in Flappy Bird by observing and responding to player performance. In measuring the efficacy of our efforts, we primarily focused on quantitative performance measures such as the number of barriers passed or play session durations. However, our data failed to capture the emotional aspects of how different players define their flow channels. For example, when conducting informal conversations with play testers outside a given session, some discussed their enjoyment and (self-identified) addiction to the game, despite having consistently *decreasing* scores across their runs. In other words, one of our im-

plicit assumptions—that engagement and performance level were directly correlated—showed evidence of being false during our data collection.

Even before making changes or augmentations to our DDA algorithm, it would behoove us to explore these emotional responses further. By conducting background surveys and performing dedicated user-experience trials, we could form a more nuanced mental model of what drives players toward game engagement, and optimize our algorithms to those goals rather than simply an increasing score card. Indeed, as we mentioned in our literature review, a wide body of research has gone into using DDA to maximize microtransaction profits (Zook et al. 2012): a metric that could implicitly drive performance *down* as players are encouraged to "pay to win."

Our initial playtests revealed that certain gameplay variables could be modified to make the game easier, but only when the player became comfortable with the modifications. There was a non-trivial adaptation time for players when, for example, the gravitational force was reduced: they would receive lower scores for several runs before working up to (and surpassing) their old high scores. Because our algorithms were reliant on instant performance responses (performance improvement on the first run after e.g. gravity decreased), these variables with longer turnaround had to be left constant to avoid positive feedback loops. However, we believe that much potential exists in building out the DDA algorithm to handle these cases. Especially given our realization that constantly improving performance is not always the most engaging activity, a temporary setback caused by a defamiliarizing change could actually bring players joy in both the short-term trial and the subsequent triumph.

Although Flappy Bird served as an excellent starting point for our research, it also lacks several crucial features for DDA in its original form. The lack of a formal win condition or levels (with changing difficulty) significantly limit the number of opportunities available for us as developers to take a "performance inventory" and adjust difficulty correspondingly. In comparison, recall that Valve's Hamlet (Hunicke and Chapman 2019) would identify specific "problem points" in an experience with an evolving (yet scripted) difficulty, and adjust the difficulty in that localized area to improve a player's experience. If we were to take more artistic liberties with the Flappy Bird concept, and augment it with some of the features that these more complicated games have, we could also explore the localized difficulty adjustment strategies that other game companies have found success using. For example, simply by adding consistency in the generated pipe sequences to give the feeling of clearing a fixed level, we could assign more weight to a player continually dying between e.g. the final two pipes of a sequence, and increase their gap heights accordingly.

Finally, the obvious prognosis of such explorations is applying our relatively simple DDA algorithm to other games altogether. Agility-based game experiences are abundant on mobile app stores, and many of them have open-source alternatives that can be freely experimented with and expanded. The prospect of taking a game that has already brought many players joy, and giving them the additional satisfaction of optimal challenge and attainable expertise, was and continues to be one of our primary motivations as researchers into the topic of DDA.

Conclusion

Dynamic difficulty adjustment strives to make the gaming experience more engaging for players across all skill levels. It helps make games more accessible and more fun for a diverse group of people. For a simple game like Flappy Bird, there are only a few parameters that we need to adjust in order to change the difficulty of the game. Using a simple metric and multipliers framework we created a dynamic difficulty algorithm that responded to how and where a player dies. After examining a combination of data points, including player score and play session length, we observed only marginal amounts of increased player engagement. However, with a more nuanced exploration of the psychology behind our players' experiences, we believe that a better, more engaging version of Flappy Bird is within reach. Our goal is ultimately to give skilled players the opportunity to feel challenged, and challenged players the opportunity to develop skill.

Our implementation of Flappy Bird, named FlappAI Bird, can be found at the following URL:

"https://github.com/neillrobson/flappai-bird".

Other Sources of Contribution

In order to save development time, we sought out an existing Flappy Bird game implementation that:

- Was faithful to the minimal gameplay specification of the original,
- Was highly accessible to users, as a cross-platform mobile app or browser game,
- Was protected under a permissive open-source license.

We found that Floppy Bird, a clone of the game using basic web browser technologies (Briefkani 2020), fulfilled all of these requirements. Our DDA augmentation is developed on top of Briefkani's existing work on his implementation of the game.

References

Alexander, J.; Sear, J.; and Oikonomou, A. 2013. An investigation of the effects of game difficulty on player enjoyment. *Entertainment Computing* 4: 53–62. doi:10.1016/j.entcom.2012.09.001.

Aponte, M.-V.; Levieux, G.; and Natkin, S. 2009. Scaling the Level of Difficulty in Single Player Video Games. *Springer* 12.

Arts, E. 2016. Dynamic Difficulty Adjustment.

Bleakley, C. M.; Charles, D.; Porter-Armstrong, A.; McNeill, M. D.; McDonough, S. M.; and McCormack, B. 2015. Gaming for health: a systematic review of the physical and cognitive effects of interactive computer games in older adults. *Journal of Applied Gerontology* 34(3): NP166–NP189.

Briefkani, N. 2020. Floppy Bird. https://github.com/nebez/floppybird.

Csikszentmihalyi, M. 2018. *Flow: The Psychology of Optimal Experience*. CreateSpace Independent Publishing Platform. ISBN 1717428517, 9781717428516.

Hunicke, R.; and Chapman, V. 2019. AI for Dynamic Difficulty Adjustment in Games. *Advances in Human-Computer Interaction* 7.

Lach, E. 2015. A quick method for dynamic difficulty adjustment of a computer player in computer games. In *International Conference on Artificial Intelligence and Soft Computing*, 669–678. Springer.

Marston, H. R.; and Kowert, R. 2020. What role can videogames play in the COVID-19 pandemic? *Emerald Open Research* 2.

McGonigal, J. 2011. *Reality is Broken: Why Games Make Us Better and How They Can Change the World*. Random House. ISBN 1409028984, 9781409028987.

Moscaritolo, A. 2014. Flappy Bird Tops App Store Charts, Headed to Windows Phone. *PC Magazine* .

Pink, D. H. 2009. *Drive: The Surprising Truth About What Motivates Us*. Canongate Books Ltd.

Sepulveda, G. K.; Besoain, F.; and Barriga, N. A. 2019. Exploring Dynamic Difficulty Adjustment in Videogames. *2019 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)* doi:10.1109/chilecon47746.2019.8988068. URL http://dx.doi.org/10.1109/CHILECON47746.2019.8988068.

Sutoyo, R.; Winata, D.; Oliviani, K.; and Martadinata, D. 2015. Dynamic Difficulty Adjustment in Tower Defence. *Procedia Computer Science* 59: 435–444. doi:10.1016/j.procs.2015.07.563.

Xue, S.; Wu, M.; Kolen, J.; Aghdaie, N.; and Zaman, K. A. 2017. Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, 465–471. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee. ISBN 9781450349147. doi:10.1145/3041021.3054170. URL https://doi.org/10.1145/3041021.3054170.

Yin-Poole, W. 2020. FIFA scripting lawsuit withdrawn after EA provides "detailed technical information and access to speak with our engineers". URL https://www.eurogamer.net/articles/2021-03-03-fifa-scripting-lawsuit-withdrawn-after-ea-provides-plaintiffs-with-detailed-technical-information-and-access-to-speak-with-our-engineers.

Zook, A.; Lee-Urban, S.; Drinkwater, M. R.; and Riedl, M. O. 2012. Skill-based Mission Generation: A Data-driven Temporal Player Modeling Approach. In *Proceedings of the The third workshop on Procedural Content Generation in Games - PCG'12*, 1–8. Raleigh, NC, USA: ACM Press. ISBN 978-1-4503-1447-3. doi:10.1145/2538528.2538534. URL http://dl.acm.org/citation.cfm?doid=2538528.2538534.