Dynamic Difficulty Adjustment with Limited Player Data

Elliot Moffatt

2nd May 2019

Abstract

Traditionally Video games have one or more static difficulties that a player can choose from. Static difficulties can cause the player to become frustrated if the game is too hard, or bored if the game is too easy. While some game developers may decide that a set static difficulty is required to fit the tone of the game, others may prioritise player enjoyment.

In this paper I detail a framework for a new type of Dynamic Difficulty Adjustment (DDA) system which can be implemented without requiring a large amount of playtest data to balance, instead relying on a Game Developer's understanding of their game and how it can be played. I also implement this system in my own game and analyse the results.

Key Words: Video Games, Dynamic Difficulty Adjustment, Artificial Intelligence

1 Introduction

1.1 Definition of Terms

There is no standardised definition of terms in Game Development and so different terms can have different definitions to different people. Below I have listed my definitions of each term used in this paper:

- **Dynamic Difficulty Adjustment (DDA):** *aka Challenge Tailoring or Adaptive game difficulty balancing.* The process of automatically changing the parameters and behaviours of the game in real time based on the player's ability, with the goal of keeping the player engaged and providing the correct level of challenge.

- **Game Variables:** *aka Game Parameters.* The simple values such as Player Health or Movement speed, or more global values such as strength of gravity.

- **Game Behaviours:** *aka game AI.* Often used as a colloquial term rather than referring to true AI, in the sense that it may describe only automated computation, rather than other standard criteria of true AI such as computer learning.

- **Playstyle:** Some games offer multiple different win conditions that a player can choose from, but often even for games that only have one win condition players can play the game in different ways to complete their task. For example in a Shooter game a player may choose to get close to the enemy to maximise the damage they deal to the enemy, at the cost of taking more damage themselves (colloquially called "Tanking"), or they may choose to shoot from a distance, dealing less damage per second, but massively reducing the risk of them taking damage (a passive playstyle).

- **Player Skill:** An abstract term to describe the player's ability to complete the tasks set out by the game.

- **Bullet Hell:** *aka danmaku.* A sub-genre of the shooter genre. Players must use fast reactions and learn enemy attack patterns to avoid a large number of onscreen projectiles. The main features to note are a top-down perspective, the player being able to move in any 2D direction, as well as the player having finite health. Classic examples include *Ikaruga* and *Touhou*.

- **Cheese Strategy:** In multiplayer games, a cheese strategy refers to a strategy that requires very little skill to execute, but requires a lot of skill for your opponent to overcome (e.g. Zerg Rush in *Starcraft*). In single-player

games, it refers to exploiting the game's systems to overcome a challenge in a way that requires less skill than the developer intended (e.g. exploiting map geometry to stand in a position where you can attack an enemy but they can't attack you)

- **Grinding:** A repetitive and usually uninteresting, but low difficulty, task carried out by the player to increase their character's strength, thereby lowering the difficulty of the game.

1.2 Static Game Difficulty

It is very difficult to formally define game difficulty. An intuitive way to think of the difficulty of any in-game situation is the amount of player skill required to overcome the current challenge presented by the game. There are many different approaches to static difficulty adjustment: The most basic is no adjustment; there is just one difficulty that the player cannot alter *(e.g. Super Mario Bros.)*. There are games with multiple predefined static difficulty levels (e.g. Easy, Medium, Hard). Developers may decide that players can only choose the difficulty at the start of the game, they may allow the player to change the difficulty setting at any time, or they may allow the player to change the difficulty at any time, but only to a lower difficulty level. The issue with having set difficulty levels is that if a player's true skill lies between one of these levels, no difficulty setting will be completely satisfying: The level above their skill will feel too challenging and the level below will feel too easy.

Some games allow the player to customise the difficulty by changing the values of individual settings. For example in the stealth Game *Dishonored 2* you can customise a large range of game variable and behaviour settings, such as how perceptive the enemies are, how long they will search for the player after they detect them, how many will attack the player at once, as well as variables such as how much damage enemies do and how quickly the player regains health.

Even when there is only one difficulty setting, skilled players may impose their own restrictions on top of those enforced by the game's systems. For example, the Nuzlocke challenge in *Pokemon* games or the "No Healing" challenge in *Dark Souls*.

On the other end of the spectrum, when a game is too difficult players have options to lower the skill required to overcome the game's challenges, such as by grinding, or by using Cheese strategies.

1.3 Dynamic Game Difficulty

The main issue with Static Game Difficulty, even in situations where the player can select a difficulty level, is that they require the player to have a good understanding of the game as well as their own skill in order to make the correct choice for difficulty. By using a Dynamic Difficulty Adjustment system, we remove the risk of the player making a poor decision, as well as providing a difficulty level that more closely matches the player's skill.
However, DDA comes with several downsides: First, if the player is aware that a DDA system is in place, lowering the difficulty each time they fail, it diminishes the sense of achievement they feel when they finally complete the task because they know they were given an easier version of it. Conversely, players may exploit this system by repeatedly failing a task as quickly as possible to artificially deflate the difficulty so that they can more easily complete later challenges.

1.4 Aims and Achievements

The direction of this project changed over the course of the year, from a Machine Learning approach to an algorithmic approach. However the general aims remained the same:

1. Produce a game which I can use to test any DDA system I develop
2. Determine a way to model the ideal player performance that leads to the most player enjoyment, to be used as the goal for the DDA system
3. Implement a DDA system into the game
4. Use playtesters to gather explicit game data and implicit tester feedback to determine the success of the DDA system

Over the course of the project I built a Bullet Hell game in the Unity3D engine and designed a DDA system for it, which included a model for ideal player performance in the form of the Skill Functions. Not only did I develop this DDA system for this specific game, I produced a modular framework that is flexible enough to produce DDA

systems for a wide range of games. While I did conduct playtesting and gather feedback and player data, due to time constraints I could only gather data from 22 testers, and the DDA version used had only gone through 2 rounds of beta testing, so was not fully optimised for the game. As a result no strong conclusion could be drawn from the data, although the fact that the results from the unoptimised DDA system were comparable to a static difficulty curve showed promise for a more optimised version of the DDA system.
Finally, I provided potential solutions to issues that arose during testing, and discussed unresolved issues as topics for future research.

2 Literature Review

Initially this Project was titled "Dynamic Difficulty Adjustment by Machine Learning". How the project evolved into its current form will be elaborated in sections 3 and 4. Therefore when I carried out my literature review at the start of this project, there was a strong emphasis on Machine Learning Techniques, however a lot of the analysis made is relevant to the final project and so shall be included here. Before starting this project, I decided upon a series of Research questions that are relevant to designing a DDA system and reviewed the literature for existing solutions to said questions:

2.1 How do you quantify and measure player enjoyment?

To determine if your Dynamic Difficulty System succeeds in making a game more engaging and enjoyable for the player, we must decide on a method to quantify player enjoyment. There have been a variety of different methods used in the past, the most common approaches used player feedback, player metrics and combinations of both:

Sweetser & Wyeth [1] produced a complex model for evaluating player enjoyment for a large variety of games, suggesting many factors that can affect player enjoyment. This model is unnecessarily complex for testing the success of a DDA system as we only need a method to quantify a players preference for one version of a game over another version of the same game.

M. Klarkowski et al [2] tested a method for quantifying the player's sense of flow through a 36-item survey measured on a 7-point Likert scale. Hendrix et al. [3] also used player feedback to determine enjoyment with and without adaptive difficulty but stated that self-reporting alone limits how well you can examine player engagement.

Andrade et al. [4] use questionnaires with questions measured on a Likert Scale, as well as collecting player metrics while they play, logging comments and expressions made during testing, and giving post-experience interviews to collect opinions and suggestions. They also only restricted testers by age rating of the game, so that their testers could have a wide variety of skills.

Colwell & Glavin [5] ran initial tests before the final test phase to alter variables not affected by their DDA system to make sure that all players could reach a certain point in their game, to prevent them from becoming frustrated from losing early. Data from testers was split into groups based on whether the tester had a weak, medium or strong gaming background. As well as asking testers to rank their enjoyment of the game with and without DDA on a scale of one to ten, Colwell argued that for weak players, frustration is more of an issue than boredom, so the DDA system should make the game easier for them and the weak players should reach a higher level with the DDA system than without. Conversely, strong players are more likely to get bored, so the strong players not getting to as high a level when DDA is used indicates a more challenging, and therefore enjoyable, experience.

A vastly different and noteworthy method of determining the Fun of a game level was developed by Sorenson, Pasquier & DiPaola [6], who designed a challenge metric $c(t)$ which returns a value for challenge at any given time t. They then defined a term "anxiety" to refer to the accumulated challenge over time. They calculate anxiety by integrating the challenge metric over a time period. They then defined "Fun" as a function of anxiety, and used the design of Super Mario Bros. levels, which are widely-regarded as well-designed, to split their procedurally designed levels into rhythm groups. The "Fun" of each rhythm group could then be computed and the sum of all the rhythm groups' Fun values gives a value for the "Fun" of the level.

Some authors, rather than directly trying to quantify enjoyment, made assumptions about

correlations between enjoyment and challenge. For example, Shi & Chen [7] assume that maximum gameplay enjoyment occurs with moderately challenging content, and Tan et al. [8] consider evenly matched games where there is a winner to be the most fun, but games with lots of draws to be frustrating.

2.2 How can we quantify difficulty?

Different game genres will allow for different methods of altering difficulty [4]: in some games it is sufficient to change a few of the games variables, such as to increase the number of Enemies and Hazards in a platforming game [7], whereas in others, the behaviours of NPC and Enemy AI may need to be altered. Once we have decided how we are going to alter the difficulty of our game, we must find a way to quantify that difficulty.

Colwell & Glavin [5] used a simple function using variables from the current game state, with output ranges corresponding to different player performance levels to determine how much to increase the difficulty of the next level by.

Andrade [4] use a Heuristic function to map a game state into a value quantifying difficulty. Zook & Riedl [9] were able to quantify player performance in each RPG battle encounter into a number in the range 0 to 2. They found a medium inverse correlation between subjective, self-reported difficulty and objective, measured in-game performance for their RPG combat game.

A Isaksen et al. [10] created an AI that modelled a human player by introducing error into input timings and limiting actions per second to a number achievable by humans. By having this AI run a level many times, and gathering data on how long it would survive, they could then produce functions that used this data to output a Hazard rate for the level. Shi & Chen [7] also used survival rates of pre-created Super Mario Bros agents to measure difficulty of their Procedurally Generated Super Mario Bros. Levels. An alternative method of determining the difficulty of a Super Mario Bros Level was proposed by Sorenson, Pasquier & DiPaola [6], who modelled the difficulty of a Super Mario Bros level at any specific point in time by creating a function which takes as inputs the size of gap being jumped and the widths of platforms on either side of the gap. The function treats enemies as gaps (since both must be jumped over) and simply adds a constant to account for the added difficulty of the enemy moving.

2.3 Can we account for player learning when quantifying difficulty?

When planning out my project, I thought that player learning may interfere with the success of a DDA system, since a player may improve at a game quicker than the DDA system can detect and increase the difficulty. However, having searched the literature, it seems that this is rarely an issue, and when the problem does arise it is easy to solve:

Zook & Riedl [9] found that players showed minimal learning in their experimental game and that additional work would be needed to determine if their GameTailor system could accurately predict player learning, whereas Colwell & Glavin [5] state that player learning is only an issue when testers have a weak gaming background, and that it is not an issue when using testers with a strong gaming background.

Andrade et al. [4] propose the solution of providing a tutorial that players must pass before full testing begins. This ensures that they have finished their initial learning phase and entered their stable learning phase, where learning is much less significant.

2.4 Can we rank gameplay mechanics and behaviours in terms of their impact on the games difficulty?

Finding a solution to this research question is less critical than answering the others, however being aware of possible solutions could allow us to determine which mechanics and behaviours have little impact on the difficulty and so don't need to be included in our DDA system, which will reduce the complexity of creating the system.

A Isaksen et al [10], after creating a Player Model and a function for computing Hazard rate, ran Single- and Two-Dimensional Sampling, allowing them to see which parameters had the greatest impact on difficulty, as well as allowing them to notice relationships between parameters.

Colwell & Glavin [5] used multiple rounds of playtesting to determine which variables had the largest impact on difficulty, as well as to determine which variables the player can easily

notice changes in.

2.5 What AI and Machine Learning techniques can be used to alter the games difficulty?

Below I will outline how different techniques have been used to produce DDA systems for different game genres:

Um, Kim & Choi [11] created a DDA system for Tetris by defining 5 difficulty levels. They divided the time domain into

fixed length timeslots, where each timeslot can have a different difficulty level, and changes in difficulty levels across several timeslots are named Patterns of Controlling Difficulty (PCD)
They created a function that outputs player fitness (a measurement of game difficulty). They then used a developer-defined value for ideal fitness and designed an algorithm to make current player Fitness converge to ideal Fitness.
The algorithm used is a micro-genetic algorithm, which has a smaller population and requires fewer iterations than standard genetic algorithms. The algorithm evaluates the player's Fitness for different PCD's, as well as the player's Fitness for the Game as a whole and gives either an easy or a hard pattern next to make the player's Fitness converge to the ideal Fitness.
This technique has the same acceptance-speed problem as the genetic algorithm, and additionally the issue that pre-defined difficulty levels are still static, so the difficulty cannot exceed that of the highest pre-defined static difficulty level, which may not be challenging enough for very skilled players.

G. Andrade [4] discuss several techniques they could have used for their DDA system. They decided that dynamic scripting was a poor choice since it does not scale well with game complexity. They also considered using Genetic algorithms, with use of online co-evolution to speed up the learning process, however this approach has the limitations that the intelligent agents employing the genetic algorithm cannot regress in skill, unlike a player. This means that the agent must start its evolution at the simplest level, which can make the early-game boring for skilled players. Additionally, its use of pre-defined models restricts its ability to learn, which could prove problematic against very skilled players.
For their Fighting game, where the player and the AI have the same mechanics, Andrade et al. found that the best option was an agent using Reinforcement Learning, with offline training so that the agent can quickly adapt to a players skill as soon as the game starts. The agent can then choose more or less optimal strategies to keep the agents performance close to the players performance.

Colwell & Glavin [5] created a Simplistic Tower Defence game. They first iterated through many versions of the game, altering different variables, until they found good base values. They also determined that their DDA system should only alter number of enemies per wave. They then created a function which took the game state (player and gate health) and used the output value to sort the players into performance band. The performance band determines how many extra enemies would spawn next wave on top of the base value. This is a very simple form of DDA, with only 2 input values and 1 output value.

Shi & Chen [7] split difficulty of a Super Mario Bros. game level segment into 5 difficulty levels based on how many of each hazard are in that segment. The difficulty level of the next segment was determined by the performance in the previous segment.

For their Car Racing game, Tan, Tan & Tay [8] used a set of behaviours that when constantly used against the human opponent, would consistently outperform them. They then used ideas from evolutionary computation and reinforcement learning to create 2 adaptive algorithms: adaptive uni-chromosome controller (AUC) and adaptive duo-chromosome controller (ADC). These algorithms alter the probabilities of each behaviour being used after each point scored, to increase or decrease the probability of the AI winning the next point, based on the AIs score relative to the player, in order to keep the score difference close and the game more intense. These algorithms had no training phase, adaption occurs during the game session.

Zook & Riedl [9] created a Game Challenge Tailoring system, GameTailor, for a game with a series of RPG combat encounters. GameTailor uses collected player data and uses Tensor Factorisation to predict player responses to new content. It then proactively picks future

content that is likely to give a player performance defined by a designer-requested performance curve.

Sorenson, Pasquier & DiPaola [6] used evolutionary computation to procedurally generate game levels, with level designs being individuals in the population, and a player enjoyment model is used for the fitness function. They tested their technique on *Super Mario Bros.* and *Zelda* style levels. The method was successful in creating feasible *Super Mario Bros.* levels but had little success in creating feasible *Zelda* levels, due to their higher complexity.

3 DDA using Machine Learning

As stated in Section 2, this Project was titled *"Dynamic Difficulty Adjustment by Machine Learning"*. Initially I had to decide how to build my Game and DDA System: whether to program in Python or to use a Game Engine. While using Python would make the Machine Learning aspect easy though use of packages such as Tensorflow, The majority of games are built with game engines, so to demonstrate that my DDA system could be widely adopted, my demonstration game should be built in a game engine rather than using python.
Next I had to choose which Game Engine to use. While Unreal Engine 4 and CryEngine 3 both had AI features, they did not appear to have support for Machine Learning. Unity3D however offers the Machine Learning Agents (ML-Agents) toolkit, an open-source plugin for Unity which supports training agents using deep Reinforcement Learning, imitation learning and other machine learning method via Python APIs.
Having Decided to use Unity, I went about building my demo game that I would later implement my DDA system into. In retrospect I should have looked deeper into the viability of the ML-Agents toolkit before finalising Unity as my choice of game engine.
I quickly ran into issues when trying to make use of the ML-Agents toolkit. The toolkit was still in Beta and was being frequently updated but the official Documentation was not being kept up to date. Additionally, given how new the ML-Agents toolkit was, and the fact that it was being frequently updated, there were very few 3rd party tutorials and those I found were outdated. Even when experimenting with the example projects that came with the toolkit, not even editing any of their code, the toolkit would throw errors and not train the agents. After several weeks of trying to get the ML-Agents toolkit working correctly I decided to change the direction of the project and use an algorithmic approach to DDA instead of a machine learning one.
In hindsight I should have been more wary about trying to develop my project using software that was still in beta. I should have thoroughly checked the viability of the ML-Agents toolkit before beginning to develop the demo game in Unity. If I had determined that the ML-Agents toolkit was not a viable option earlier in the project, I would have been more willing to look more deeply into the Machine Learning options for other game engines, but I was wary that persevering with the machine learning approach to DDA may be simply a sunk-cost fallacy, and instead decided to shift to an algorithmic approach.

4 A modular framework for Dynamic Difficulty Adjustment

4.1 Motivation for my solution

After deciding to swap from a Machine Learning approach to an algorithmic approach to DDA, I again consulted the literature. When reading a paper by Electronic Arts, Inc. developers [12] I noted that to test their approach to DDA, which used probabalistic functions, they had a huge amount of data available to them (Their values for daily gameplay time in minutes were in the millions). This approach could not be applied nearly as successfully for small developers that don't have access to as much data. I also noted that their DDA system was very seperate from the mechanics of their game, using only the probabilities of players transitioning between different states (levels), rather than making use of metrics that can be gathered while the player is actually playing each level.

After reading that paper, I considered the idea that a developer's deep knowledge of their game and how the player can play it can be used as a substitute for probabalistic functions that require large data sets. I recalled the famous quote made by video game designed Soren Johnson in his blog [13]: *"Given the opportunity, players will optimise the fun out of a game"*. If a player can figure out how to optimise a specific strategy for a game, that strategy should be able to be expressed mathematically in terms of player performance

and actions. With a deep knowledge of their game, a developer could explicitly define each playstyle available in their game, know what game variables and behaviours each playstyle is weak or strong against, and write functions for how well the player is utilising each playstyle.
In the following subsections I talk through the details of each module of the DDA framework, and provide an overview of the algorithm as a whole. It is worth noting that the concept of a developer-knowledge focussed DDA system could not be found in the literature, and as such all contributions and results are my own unless I explicitly cite another source.

4.2 DDA segments

This DDA system works additively, making adjustments to the difficulty at set intervals. These intervals, which I've called "DDA Segments" are a set length of time, decided by the developer to suit the type of game, where over the interval the game is "polled" multiple times to gain metrics about the player that will be used for the Playstyle and Skill function calculations. At the end of the segment the Playstyle and Skill functions are computed and the State Update Function is called, then the next DDA Segment begins.

4.3 Game Difficulty State

The Developer decides which game variables and behaviours the DDA system will be allowed to change. For Game variables, through playtesting they will decide upon a default value that is fixed, and then will create Multiplier variables that can be manipulated by the DDA system to change the difficulty, i.e.

$$Difficulty = Multiplier\ Variable \times Base\ Difficulty$$

This DDA framework assumes that the Multiplier variable will be in a range 0 to 1, but if the developer wants a different range, say 0.75 to 1.25 it is trivial to create an intermediate function that remaps to that range, i.e.

$$Difficulty = f(Multiplier\ Var) \times Base\ Difficulty$$

For Game behaviours, if the behavior can have varying strength then this strength can be determined by the Multiplier Variable, else if the behaviour can only be active or inactive the Multiplier variable can represent the probability of that behaviour being active in the next DDA segment.

4.4 Playstyle Functions

The Developer defines a number of playstyles the player may use. These playstyles can be independent of each other or dependent inverse playstyles. the Developer can also define playstyles that are strongly correlated, however it may be better to combine them into one if there is a strong correlation. The Developer must then formally define the playstyle using the player metrics they've gathered, either using metrics from just the last segment, metrics from the whole game session, a weighted average of the two, or some other option. The output of these functions must be a float in range 0 to 1, with 0 meaning the player does not utilise this playstyle at all and 1 meaning the player completely uses this playstyle. These functions can be purely continuous, or can be more complex, utilizing conditional statements and step functions. Examples are given in Section 5.

4.5 Skill Functions

Each Playstyle has an associated Skill Function. The Skill function uses player metrics decided upon by the developer to represent the Skill of the player relative to the current challenge of the game. The output of this function should be a float in the range -1 to 1.
This output represents how balanced the game is (under the assumption that the player us utilising the Playstyle with which this Skill function is associated), with an output of 1 meaning the game is very unbalanced in favour of the player (The player will certainly beat the current challenge) and an output of -1 meaning the game is very unbalanced against the player (The player will certainly lose the current challenge). An output of 0 means the game is perfectly balanced and therefore no changes to the difficulty should be made.
Simply put, the goal of the DDA system is to reach a Skill Function output of 0.

4.6 Counter Matrix

The Counter Matrix is a 2-Dimensional array where for each Playstyle P_i and each Game Variable BV_j (or game behaviour), the Counter Matrix Value $C_{i,j}$ represents how strongly the Variable "Counters" the Playstyle (i.e. how much more difficult is it to play using that playstyle when that Variable is high relative to its default value (or that behaviour is active, relative to when the behaviour is inactive). $C_{i,j}$ is in the range -1 to 1 where a value of -1 means that the Variable being high (or behaviour

being active) makes the Playstyle a lot easier to use, a value of 1 means the Variable being high (or behaviour being active) makes the Playstyle a lot harder to use, and a value of 0 means the Variable/Behaviour has no impact on the difficulty of the Playstyle.

4.7 State Update Function

For each Playstyle P_i and each Game Variable (or Behaviour) V_j, with Counter Matrix Value $C_{i,j}$ and Skill Value S_i, the Multiplier Variable for the Game Variable (or Behaviour) M_j is updated as follows:

$$M_j \mathrel{+}= U_{i,j}$$

Where $U_{i,j}$ is some function $\mathbf{U}(S_i, P_i, C_{i,j})$ with the following properties:

- $(S_i = 0$ **or** $P_i = 0$ **or** $C_{i,j} = 0) \Leftrightarrow U_{i,j} = 0$
- $|U_{i,j}|\, is\ an\ increasing\ function\ of\ |S_i|$
- $|U_{i,j}|\, is\ an\ increasing\ function\ of\ |P_i|$
- $|U_{i,j}|\, is\ a\ decreasing\ function\ of\ |C_{i,j}|$
- $sgn(S_i \times C_{i,j}) = sgn(U_{i,j})$

4.8 Algorithm for the DDA system

The Pseudocode for the DDA system is given in Algorithm 1.

Algorithm 1 DDA Algorithm

```
procedure SETUP
    for all C_{i,j} do
        C_{i,j} ← Developer-chosen Value

procedure DDA
    while Game is running do
        repeat
            Poll for player data
        until Segment ends

        for all Playstyles P_i do
            P_i ← PlaystyleFunction(i)
            S_i ← SkillFunction(i)
        for all Game Variables V_j do
            for all Playstyles P_i do
                M_j += U_{i,j}
```

5 Implementation

To test this DDA Framework, I implemented it in my own game, a Bullet Hell game titled *BULLETHELL* built in the Unity3D game engine.

5.1 Description of Game

BULLETHELL consists of 10 levels, where in each level the player can move the player Character (a green sphere) up, down, left and right. The player character automatically shoots projectiles forward, and the player can choose to fire the projectiles behind instead.
In each level, there are one or more "Nemeses" (red Cubes). By default, the Nemeses move in a fixed pattern around the screen and have a default pattern for firing their own projectiles. The Nemeses have different movement patterns in each level, and there are also differences in parameters such as projectile speed, enemy movement speed and enemy health. The base difficulty increases throughout the game (So level 10 is much harder than level 1 by default).
The player character loses health by being hit by an enemy projectile, and the enemy loses health by being hit by player projectiles. The enemy also loses a small amount of health passively over time. If the player's health drops to 0, they lose the level. They can retry each level as many times as they like. The game starts off with just one level unlocked. Each subsequent level is unlocked by the player killing all the Nemeses in the previous level (the player's health is returned to its maximum on starting a new level). Once the player completes all 10 levels, The game is beaten.

5.2 Game Behaviours

I decided to not have the DDA system for this game alter any game variables, just Game Behaviours. I could then manually set the game's variables to set the difficulty curve for the game. Since all behaviours can only be active or inactive, the DDA system sets the probabilities of each behaviour being active in the next DDA segment. The behaviours are split up into movement pattern behaviours and Bullet Pattern Behaviours:

Movement Patterns

- Default: The Nemesis moves between set waypoints on the screen in a set order. Active iff no other Movement behaviours are active.

- Random Waypoints: The Nemesis moves between the set waypoints, but in a random order.
- Stationary: The Nemesis does not move
- Chase: The Nemesis moves towards the player.

Only one movement pattern can be active at a time, with priority being:
Chase > Stationary > Random Waypoints > Default

Bullet Patterns
There are 6 Bullet Patterns, viewable in Appendix A. Any number of them can be active at the same time, with the exception that the Default pattern is only active iff none of the other patterns are active.
Due to the fact that the Counter Matrix considers the difficulties of each behaviour to be independent to one another (an issue that will be discussed further in Future Research), I have hard-coded in adjustments to the behaviours so that they behave differently when multiple are active. Specifically, Their fire rate relative to the base fire rate is adjusted so that the more Bullet patterns that are active, the lower the fire rate is for each of them.

5.3 Playstyles

During the Polling period of the DDA segment, the game polls for data every 0.5 seconds. The poll computes the distance between the player and the (closest) Nemesis, and the distance the player has moved since the last poll. Once the Polling period ends, the average distance between player and enemy, and average distance travelled between polls is calculated.
I defined two pairs of inverse playstyles. The first pair is **Stationary** and **Mobile**. A Stationary Playstyle is one where the player is moving very little, whereas a player with a Mobile playstyle moves a lot.
Since The two playstyles are opposites, I can compute them at the same time using Algorithm 2:

Algorithm 2 Stationary-Mobile Playstyle function

procedure
$LowerBound, LB \leftarrow 0.5$
$UpperBound, UB \leftarrow 5.5$
$D \leftarrow$ Average Distance Travelled
if $D \leq LB$ **then**
Stationary $\leftarrow 1$
Mobile $\leftarrow 0$
else
Stationary $\leftarrow 0$
if $LB < D \leq UB$ **then**
$Mobile = \frac{D - LB}{UB - LB}$
else
Mobile $\leftarrow 1$

The second pair of playstyles refer to how close to the enemy the player chooses to play. The **Tanky** playstyle is for players who choose to play close the the enemy to maximise their damage output, but at the cost of being more difficult to dodge the enemie's projectiles. The opposite playstyle is **Evasive**, where the player stays away from the enemy to make dodging easier, but at the cost of dealing less damage. Again, I compute both playstyles at the same time using Algorithm 3.

Algorithm 3 Tanky-Evasive Playstyle function

procedure
$TankyMax, T_{Max} \leftarrow 3$
$Middle, M \leftarrow 6$
$EvasiveMax, E_{Max} \leftarrow 10$
$D \leftarrow$ Average Distance between player and enemy
if $D \leq M$ **then**
$Evasive \leftarrow 0$
if $D \leq T_{Max}$ **then**
$Tanky \leftarrow 1$
else
$Tanky \leftarrow \sqrt{\frac{M - D}{M - T_{Max}}}$
else
$Tanky \leftarrow 0$
if $D \geq E_{Max}$ **then**
$Evasive \leftarrow 1$
else
$Evasive \leftarrow \sqrt{\frac{D - M}{E_{Max} - M}}$

Screenshots of each playstyle can be seen in Appendix A

5.4 Skill Function

Due to the simplicity of the game, I decided that all playstyles should share the same skill function. Initially, I designed the playstyle under the assumption that the game is perfectly balanced when the player's and Enemy's health are the same, but in practice that means that half the time, the player

is losing the level when the enemy has a very small amount of health left, which i a very frustrating experience. Therefore I changed the Skill Function so that it the game is considered balanced when the player has 10% more of its max health than the enemy does, e.g. if the enemy is at 50% health the player should be at 60%, and the player should defeat the enemy and beat the level with just 10% of their health left. Additionally, to increase the rate at which the DDA system adjusts the difficulty, The Skill function assumes that if the player has 50% or more health than the enemy, the game is fully unbalanced in favour of the player, and if the enemy has 30% or more health than the player then the game is fully unbalanced against the player.

Algorithm 4 Skill Function

procedure

$Skill_{max} \leftarrow 0.5$
$Skill_{zero} \leftarrow 0.1$
$Skill_{min} \leftarrow -0.3$

$$HP_{Player} \leftarrow \frac{Current\ Player\ Health}{Starting\ Player\ Health}$$

$$HP_{Enemy} \leftarrow \frac{Current\ Enemy\ Health}{Starting\ Enemy\ Health}$$

$\Delta HP \leftarrow HP_{Player} - HP_{Enemy}$

if $\Delta HP > Skill_{max}$ **then**
$Skill \leftarrow 1$
else if $\Delta HP \geq Skill_{zero}$ **then**

$$Skill \leftarrow \frac{\Delta HP - Skill_{zero}}{Skill_{zero} - Skill_{min}}$$

else if $\Delta HP \geq Skill_{min}$ **then**

$$Skill \leftarrow \frac{\Delta HP - Skill_{zero}}{Skill_{zero} - Skill_{min}}$$

else
$Skill \leftarrow -1$

5.5 Counter Matrix

The Counter Matrix used for *BULLETHELL* is displayed in Table 1.

The values in the Counter Matrix were selected based upon my own testing of the game using each Playstyle. Some values were easy to decide upon, for example the Tracking Fire Behaviour was specifically designed to Counter the Stationary playstyle, so was given a value of 1, whereas other behaviours were more difficult to select values for.

Table 1: *Counter Matrix for BULLETHELL*

Game Behaviour	**Player Playstyle**			
	Tanky	Evasive	Stationary	Mobile
Stationary	-1.0	-0.5	-1.0	0.0
Skip Waypoints	0.8	0.2	0.1	0.5
Chase	1.0	1.0	1.0	1.0
Reverse Rotation	0.2	0.2	0.3	0.1
Directional Fire	0.4	0.3	0.1	0.2
Burst Fire	0.9	0.2	0.2	0.2
Tracking Fire	1.0	0.3	1.0	0.2
Lazer Fire	1.0	0.5	0.7	0.8

5.6 State Update Function

The difficulty state update function $U_{i,j}$ for Game Behaviour B_j and Playstyle i with value P_i, with Skill Function Output S and Counter Matrix value $C_{i,j}$ is shown in Algorithm 5

Algorithm 5 State Update Function

procedure U$(S, P_i, C_{i,j})$
$\alpha \leftarrow 0.02$
$\beta \leftarrow 0.5$
$\gamma \leftarrow 1$
$\delta \leftarrow 0.5$
if $S == 0$ **or** $P_i == 0$ **or** $C_{i,j} == 0$ **then**
return 0
else

$$output \leftarrow sgn(S \times C_{i,j}) \times \frac{\alpha \times |S|^{\beta} \times {P_i}^{\gamma}}{|C_{i,j}|^{\delta}}$$

if $output < 0$ **then**
return $output \times 1.5$
else
return *output*

The constants α, β, γ and δ are used to scale the effects of each input, as well as the overall size of the change to the difficulty the function makes. It is also worth noting that if the skill value is negative, the output magnitude is multiplied by 1.5. I decided to do this so that the difficulty can be reduced slightly faster than it can be increased, because I felt that players will become frustrated by a high difficulty quicker than they will become bored by a low difficulty, so the system needs to reduce the difficulty quicker than it increases it.

6 Testing of the Implementation

6.1 Procedure

To test the effectiveness of my DDA system, I created a second system that works independently of player performance. This system has a fixed difficulty, where I manually set the activation probabilities for each Game Behaviour, with the probabilities increasing exponentially in proportion to the level number.
I gave my DDA system the codename "Mode Y" and the Static difficulty system the codename "Mode X". I hosted the game on my GitHub page, and in the instructions told players that both Mode X and Mode Y were DDA systems I had built. By making the Static difficulty system a placebo, and giving the modes non-descriptive names, I reduced the bias of players giving more favourable feedback towards my DDA system simply because they want to give me positive data.
When the game launches, the Mode (X or Y) that is initially selected is random (Note that players can switch Modes at any point from the main menu). This is because while I did include a tutorial to help reduce the effects of player learning on the results, I expected that the order that the player plays Mode X and Mode Y will effect both their objective progress data and subjective feedback data.

The players were instructed to play through the game with the first gamemode selected. Once they were done playing with the first gamemode (either beat the game or quit early due to frustration or boredom) they were asked to fill out the first half of the survey. The survey was comprised of multiple choice questions (*MC*), 7-point linear scales (*LS*) where the player has to identify how strongly they agree with the statement provided, a couple of short answer questions for collecting objective data and a final long answer question where the player could leave any relevant comments not covered by the questions.

The first half of the survey contained the following questions:

- (*MC*) Player age range
- (*LS*) Player's experience with games in general (1: none - 7: very experienced)
- (*LS*) Player's experience with the Bullet Hell genre (1: none - 7: very experienced)
- (*MC*) Which Mode they played first
- Number of Levels beat
- Number of lives lost
- (*LS*) How challenging the early-, mid- and late-game levels were (3 seperate questions, 1: not challenging, 7: very challenging)
- *MC* How close the last level they completed felt (0: don't remember, 1: very close, 2: quite close, 3: not very close, 4: not at all close)

They were also asked how much they agreed with the following statements (on a 7-point Linear scale), taken from Jackson's [14] paper. These questions are used to determine the player's sense of flow.

1. *"I was challenged, but I believed my skills would allow me to meet the challenge"*
2. *"I made the correct movements without thinking about trying to do so"*
3. *"I knew clearly what I wanted to do"*
4. *"It was really clear to me that I was doing well"*
5. *"My attention was focused entirely on what I was doing"*
6. *"I felt in total control of what I was doing"*
7. *"I was not concerned of what others may have been thinking of me"*
8. *"Time seemed to alter (either slowed down or sped up)"*
9. *"I really enjoyed the experience"*

The players were then asked to take a break so that any built up stress from the first mode does not carry over into the second, then play through the second mode and return to the survey. The second half of the survey collected the same data for the second game mode played as for the first, as well as the following multiple choice comparison questions, with the option of selecting Mode X, Mode Y, or no difference/preference:

- Which Mode had an easier early-, mid- and late-game (3 seperate questions)
- Which Mode had the most close levels (i.e. levels completed with barely any player health left)
- Which Mode they enjoyed the most

I chose to use this approach because it provides the opportunity for cross-examination between explicit game data and implicit player feedback, which upon evaluation could have the potential to reveal a trend that would lead to further refinement of the Skill Function.

6.2 Feedback

22 Players took part in the survey. 4 were 25 to 34 years old, and 18 were 15 to 24 years old. All participants answered 5 or more to the question on how much experience they have with games in general, indicating that all participants have a lot of experience with games.

A lot of the questions asked in the survey are subjective. The responses to these should not be given much weight on their own, but should be used only as supplementary evidence for any conclusions made using the objective data: Levels beat and Lives lost. As an example of how unreliable this data is, consider the feedback given by Player ID 12: When asked flow question 9 individually for each mode, they gave a higher score for Mode Y than for Mode X, indicating that they enjoyed Mode Y more, yet when asked explicitly which mode they preferred, they chose Mode X.

The Raw data is shown in Table 2. The table also includes the difference in answers between Mode Y and Mode X for each participant. On initial inspection there is no clear trend that emerges. I considered applying the Wilcoxen signed-rank statistical test since my data is pairs of data points, but decided against using such statistical tests since my sample size is small and most of the data is subjective, which means doing a statistical test is likely to output a false positive which would wrongly affect my conclusions.

I next decided to split the participant pool into disjoint sets in 2 different ways to test 2 seperate hypotheses:
The first Hypothesis, H_1, is that despite the Tutorial, player learning during gameplay affected the results. Therefore I split the pool into 2 sets:

- Players who played Mode X first
- Players who played Mode Y first

The second Hypothesis, H_2, is that players who fully completed the game on both modes, getting to experience the full exponential difficulty curve of Mode X and giving Mode Y's DDA system the full amount of time to adjust the difficulty will have more contrasting opinions of the 2 modes. To test this I split the pool into 3 groups:

- Group A: Players who only played one mode (No comparison data available)
- Group B: Players who did not finish the game in *both* modes
- Group C: Players who finished all the levels in both modes

6.3 Analysis of the results

Having reviewed the raw data as a whole, as well as having reviewed it for each Group, I have decided that no strong conclusions can be drawn just from considering the raw statistics. The differences in both objective and subjective data between modes X and Y are very small.
There is, however, one interesting fact that can be drawn from the results: 4 of the 11 participants who played Mode X first quit and did not play mode Y, whereas only 1 of the 11 who played mode Y first quit and did not play mode X. This suggests that Mode Y indeed fulfils its purpose of making players less likely to quit, however due to the small sample size I am unwilling to draw this conclusion. It is also worth considering that the version of my DDA system used for Mode Y was only the result of 2 rounds of testing: the first round with only myself as a tester, and the second with 3 other testers relaying feedback to me. In commercial use this DDA system would require many more waves of testing, so the fact that the results of this system after just 2 waves of testing were comparable to a standard, traditional difficulty curve shows promise that with further testing and refinement this system could surpass a traditional static difficulty curve.

In retrospect, while distributing the game online gained me more pariticipants, it meant that the data gathered was a lot less conclusive. I believe being present when the participant plays the game to gather their feedback as they play, rather than hoping they recall correctly after playing, would provide more reliable data. Additionally, I recorded the levels beat and lives lost on the main menu so it was easy to gather objective data without trusting the player to remember. I should have listed more objective data on that homescreen, such as player health at the end of each level, so that I would not need to rely on subjective questions such as "How challenging did you find the mid-game levels", but instead could just review the objective data provided.

7 Future Research

In this section I will be discussing the issues this DDA framework currently has, and proposing

some potential solutions that could be tested in future research.

Having finished defining the framework, implementing it into *BULLETHELL* and analysing feedback, I realised that one of the assumptions I made when initially planning this DDA system is not necessarily true: I assumed that the most enjoyment can be had out of a game when the game's difficulty level matches the player's skill level. While this is certainly true for certain games and certain types of players, it has a major issue: This DDA system *erodes the difficulty curve*. Because the system adapts to the player's current skill level, there is no reason for the player to learn and improve at the game. If the player keeps failing a challenge the game sets out enough times, never improving or trying to understand the game better, the DDA system can still eventually lower the difficulty level enough that the player can overcome the challenge. Of course, certain players will be fine with this, caring more about the explicit in-game reward received by overcoming the challenge rather than the implicit sense of accomplishment and pride in having improved at the game, but for other players this sense of pride and accomplishment is what motivates them. This DDA system is designed to adjust the difficulty so that the player just barely overcomes each challenge, but for some players that will not be satisfying enough: they want to fail the challenge several times, but know that they are slowly improving so that when they finally overcome the challenge it feels a lot more satisfying. Ultimately it is up to the developer to determine what kind of player they are aiming their game at and to decide whether the current version of this DDA system will be sufficient, but for those developers that wish to maintain a more structured difficulty curve I propose the following ideas for follow-up research to the DDA system described in this paper:

7.1 Proposal A: Careful selection of DDA variables and ranges

The first way to prevent this DDA system from degrading the difficulty curve does not require changing the current algorithm, but rather requires the developer to more carefully choose which game variables and behaviours the DDA system alters. As described previously, when implementing this system into *BULLETHELL*, I chose to only alter the game behaviours, and manuallly set static variables for each level. I did this thinking that I could control the difficulty curve of the game via the game variables, while letting the DDA system make small adjustments to the difficulty via changing game behaviour activation probabilities. As it turned out, the Game Behaviours had a much more powerful impact on the game's difficulty than expected, meaning that any increase in difficulty caused by the static game variables could be negated by the DDA system.
By more carefully testing the effects of game variables and behaviours on the game's difficulty, developers can design a difficulty curve and be certain that the DDA system will not erode the curve, therefore making the DDA system act more as a "helping hand" to struggling players rather than a substitute for player improvement. For example, lets consider how this system could work in the *Dark Souls* game series:

Figure 1: *A screenshot from Dark Souls 3, showing a Boss arena. The player's health, magic and stamina can be seen in the top left corner, and the Boss's health bar is on the bottom of the screen. Source: Softpedia*

Here is a simplified explanation of how the gameplay loop works in Dark Souls: The player plays as a knight with a sword and shield. The player fights through an area with lots of regular enemies until they reach the Boss arena, where they must beat the boss (The boss is much more difficult to fight, with a larger healthpool and/or more powerful attacks). Once the player beats the boss they unlock the next area and the cycle repeats. Players gain experience points by defeating both regular enemies and bosses. They can use this experience to upgrade their character's health and stamina as well as their physical and magic attack power. Dark Souls has a static difficulty, however if a player is struggling with a certain boss fight they can "grind" by repeatedly killing the weaker

enemies in the area to increase their player stats slightly, making the boss fight slightly easier. However, despite having higher character stats, the player will not be able to beat the boss without learning the bosses attack patterns and improving at responding correctly to each attack.
Where Dark Souls allows players to change the difficulty of the boss fight, my DDA system could be used instead: By using the Skill function to detect that the player is performing poorly in the boss fight (i.e. repeatedly dying to the boss), the game could adjust the player's health and/or damage stats to give them a slightly better chance of winning the fight. By carefully selecting the Multiplier range of the health and damage variables so that they can't increase too much, as well as keeping the Boss's variables and behavours fixed, my system would give players better odds in the fight without having to carry out the boring task of grinding, but still force them to learn how to react to the Boss's behaviours, thus maintaining the Difficulty curve.

7.2 Proposal B: An extension to the algorithm

Proposal A has a couple of issues: firstly, it can be difficult to set the range that you allow variables to change: too large a range and they overpower the curve, too little and it will be too similar to a static difficulty. Secondly, it requires you to split the variables that affect difficulty into two groups: those that are static (to maintain the difficulty curve) and those that are used by the DDA system. An alternate system which extends the existing Framework and maintains the difficulty curve without limiting the number of game variables and behaviours the DDA system can alter is as follows:

Instead of having the difficulty settings carry over between each ingame challenge (level, boss etc), use the changes the DDA makes for the previous challenge to influence the starting difficulty for the next challenge.
Lets consider a simple situation with just one variable being altered. Let that variable's default value for challenge i be X_i and lets say that the developer allows the DDA system to change the value to within the range $X_i - \alpha_i X_i$ to $X_i + \alpha_i X_i$. When the player has finally overcome the challenge, the DDA system had adjusted X_i to a new V_i, and we can represent V_i as $V = X_i + k_i \alpha_i X_i$ where k_i is in the range -1 to 1. Then to adjust the starting difficulty for the next challenge, we compute

$$\widehat{X_{i+1}} = X_{i+1} + k_i \alpha_{i+1} X_{i+1}$$

and the new range is between $\widehat{X_{i+1}} - \alpha_{i+1} X_{i+1}$ and $\widehat{X_{i+1}} + \alpha_{i+1} X_{i+1}$. Thus we can also compute

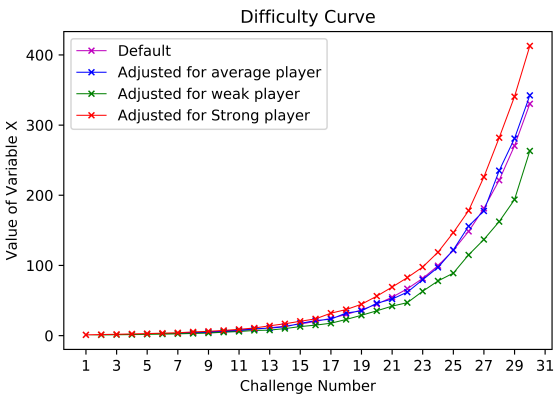
$$\widehat{\alpha_{i+1}} = \frac{\alpha_{i+1}}{1 + k_i \alpha_{i+1}}$$

Figure 2 shows how this system could scale the difficulty curve of a variable X defined by

$$X = e^{0.2 \times Challenge\ number}$$

with fixed $\alpha = 0.2$ and shows how the curve changes for an average player (random k in range -0.3 to 0.3), a weak player (random k in range -1 to -0.6) and a strong player (random k in range 0.6 to 1). For an average player, their curve does not differ much from the default, but for stonger players their value of X by challenge 30 is about 60% larger than the value for a weak player. The weak player is given a much lower difficulty than the strong player, but the difficulty curve is maintained and without the weak player improving they will not overcome all of the challenges.

Figure 2



This approach still has some issues: firstly, it assumes that each challenge is disjoint, whereas in reality it is possible for challenges to overlap. Secondly, it assumes that the game is fully linear: That the developer knows what order the player will face each challenge and that the player will not return to any previously faced challenges. More research will need to be carried out to determine what is required to make this system compatible with non-linear and open-world games.

7.3 Issues with the Counter Matrix

Another issue with the current form of the DDA Framework laid out in this report is the Counter Matrix: The Counter Matrix has 2 major flaws: The first is that it assumes all the difficulty parameters altered by the DDA system are independent, whereas it is likely that the increase in difficulty caused by 2 game behaviours being active at the same time is greater than the sum of the difficulty increases of each one being active individually. As mentioned previously, In my implementation I hardcoded alterations to the behaviours so that the strength of each individual behaviour was lowered as more behaviours were active at the same time. A topic for future research could be how to adjust the counter matrix to account for how combinations of game behaviours affect each other.

Additionally, the values for the counter matrix were selected by myself, using my own personal experience of the game. Another topic for future research could be how to personalise these values to each individual player, by developing a method that detects when the effect a behaviour/variable has on a player's performance does not match the expected effect defined in the Counter Matrix.

8 Conclusion

In this report I have discussed existing Dynamic Diffficulty Adjustment systems and have proposed a framework for implementing a DDA system into a range of games which relies more heavily on a developer's understanding of their game than on large quantities of player data. I tested this framework by implementing it into my own game, and comparing it to a version of the game with a static difficulty. Due to time contraints the version of the DDA system was not well optimised and so the results were inconclusive. Additionally I discussed methods to adapt this framework to prevent eroding the difficulty curve, and discussed topics of future research.

Table 2: *Raw Feedback Data*

General																						
Player ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
exp with games	7	7	2	7	5	7	5	6	7	6	6	5	7	7	7	7	7	6	6	7	7	7
exp with bullethells	7	2	2	4	2	2	2	1	2	3	5	3	2	2	5	6	2	3	3	4	5	7
Challenge																						
Mode X		only	only	only	only	1st	1st	2nd	2nd	2nd	2nd	1st	2nd	2nd	2nd	1st	1st	2nd	1st	2nd	2nd	1st
levels beat		2	3	5	8	3	3	4	4	4	6	8	10	10	10	10	10	10	10	10	10	10
lives lost		0	5	1	3	8	3	1	1	11	2	12	4	2	7	6	5	5	4	3	2	0
early-game challenge		3	7	2	2	4	7	3	4	5	3	4	2	1	2	1	3	3	2	3	2	1
mid-game challenge		dnf	dnf	dnf	4	dnf	dnf	dnf	dnf	dnf	dnf	4	7	3	3	3	4	5	3	4	4	2
late-game challenge		dnf	dnf	dnf	dnf	dnf	dnf	dnf	dnf	dnf	dnf	dnf	6	4	6	6	5	6	5	4	6	2
closeness of last lvl		0	2	2	1	1	1	4	2	2	4		1	3	1	1	3	2	2	2	1	3
Mode Y	only																					
levels beat	10					3	6	2	7	9	2	4	9	9	10	10	10	10	10	10	10	10
lives lost	3					8	1	1	2	13	1	3	5	3	8	2	4	3	2	7	2	0
early-game challenge	1					5	3	4	1	2	7	5	3	3	5	4	3	3	2	2	3	1
mid-game challenge	3					dnf	dnf	dnf	2	4	dnf	dnf	5	5	1	6	4	4	3	3	2	3
late-game challenge	6					dnf	dnf	dnf	dnf	dnf	dnf	dnf	dnf	dnf	3	6	5	5	6	4	6	3
closeness of last lvl	3					2	1	3	2	2	4	1	1	1	0	2	3	2	1	2	2	4
Y-X																						
levels beat						0	3	-2	3	5	-4	-4	-1	-1	0	0	0	0	0	0	0	0
lives lost						0	-2	0	1	2	-1	-9	1	1	1	-4	-1	-2	-2	4	0	0
early-game challenge						1	-4	1	-3	-3	4	1	1	2	3	3	0	0	0	-1	1	0
mid-game challenge													-2	2	-2	3	0	-1	0	-1	-2	1
late-game challenge															-3	0	0	-1	1	0	0	1
closeness of last lvl						1	0	-1	0	0	0		0	-2		1	0	0	-1	0	1	1
Comparison																						
easier early game						0	y	x	y	y	x	y	y	0	y	x	0	0	x	y	y	x
easier mid game						dnf	y	dnf	dnf	y	x	0	y	x	y	x	y	y	x	y	y	0
easier late game						dnf	dnf	dnf	dnf	y	dnf	0	x	x	x	y	y	y	y	0	y	x
most close levels						0	0	y	x	0	x	0	x	y	0	x	x	x	0	0	x	y
preferred gamemode						0	y	x	x	x	x	x	x	y	x	0	x	x	y	y	x	y
Thoughts&feels																						
X																						
q1		5	2	7	2	5	5	2	6	5	5	7	6	4	4	6	6	5	3	3	6	2
q2		5	3	7	1	3	3	5	4	2	3	7	4	5	3	2	2	3	4	1	7	7
q3		6	4	7	5	3	3	7	4	4	5	5	5	6	4	5	7	4	6	5	6	7
q4		5	3	1	2	4	4	3	5	5	5	4	2	6	2	5	2	5	3	3	5	7
q5		7	4	6	3	4	3	7	5	6	6	5	7	7	3	6	6	6	4	2	7	7
q6		6	3	5	5	4	5	7	5	4	3	4	4	6	5	1	7	7	5	4	6	2
q7		6	6	7	7	4	7	7	5	1	4	4	4	7	4	7	7	7	7	5	6	7
q8		4	4	1	2	1	3	4	6	4	4	2	7	3	2	4	1	1	1	2	6	4
q9		5	2	7	4	4	2	6	5	5	5	4	6	7	6	4	6	7	4	5	7	2
Y																						
q1	5					4	5	2	5	5	3	4	6	6	3	6	6	5	5	3	2	4
q2	2					3	3	2	6	2	3	3	2	3	3	2	2	3	4	1	5	7
q3	5					2	5	7	4	2	3	4	2	6	6	6	5	5	6	7	7	7
q4	1					3	3	3	4	5	3	4	1	5	3	5	2	5	4	5	5	7
q5	3					4	5	7	3	3	2	3	6	7	2	6	7	6	4	2	6	7
q6	4					3	4	6	5	3	3	4	2	6	5	1	6	7	4	6	5	2
q7	7					4	7	7	5	6	4	3	6	7	5	7	7	7	7	5	6	7
q8	1					1	2	4	5	4	1	2	6	2	1	4	1	1	1	2	6	4
q9	3					4	3	5	4	4	2	5	5	6	6	5	6	6	5	5	5	3
Y-X																						
q1						-1	0	0	-1	0	-2	-3	0	2	-1	0	0	0	2	0	-4	2
q2						0	0	-3	2	0	0	-4	-2	-2	0	0	0	0	0	0	-2	0
q3						-1	2	0	0	-2	-2	-1	-3	0	2	1	-2	1	0	2	1	0
q4						-1	-1	0	-1	0	-2	0	-1	-1	1	0	0	0	1	2	0	0
q5						0	2	0	-2	-3	-4	-2	-1	0	-1	0	1	0	0	0	-1	0
q6						-1	-1	-1	0	-1	0	0	-2	0	0	0	-1	0	-1	2	-1	0
q7						0	0	0	0	5	0	-1	2	0	1	0	0	0	0	0	0	0
q8						0	-1	0	-1	0	-3	0	-1	-1	-1	0	0	0	0	0	0	0
q9						0	1	-1	-1	-1	-3	1	-1	-1	0	1	0	-1	1	0	-2	1

Remark: dnf = did not finish. players could not answer this question because they did not complete all of the levels the question relates to.

Table 3: *Averaged Data for each group*

	Mode X first	Mode Y first	Group A	Group B	Group C
General					
Size of Group	11	11	5	9	8
exp with games	5.91	6.64	5.6	6.22	6.75
exp with bullethells	3.18	3.55	3.4	2.44	4.38
Challenge					
Mode X					
levels beat	6.55	7.80	4.5	5.78	10.00
lives lost	4.27	3.80	2.25	4.89	4.00
early-game challenge	3.27	2.80	3.5	3.67	2.13
mid-game challenge	3.33	4.33	4	4.67	3.50
late-game challenge	4.50	5.33		5.00	5.00
closeness of last lvl	1.60	2.20	1.25	2.25	1.88
Mode Y					
levels beat	7.57	8.00	10	5.67	10.00
lives lost	2.86	4.36	3	4.11	3.50
early-game challenge	3.29	3.09	1	3.67	2.88
mid-game challenge	4.00	3.22	3	4.00	3.25
late-game challenge	5.00	4.80	6		4.75
closeness of last lvl	2.00	2.00	3	1.89	2.00
Y-X					
levels beat	-0.14	0.00		-0.11	0.00
lives lost	-2.57	0.70		-0.78	-0.50
early-game challenge	0.14	0.50		0.00	0.75
mid-game challenge	1.00	-1.00		0.00	-0.25
late-game challenge	0.50	-1.00			-0.25
closeness of last lvl	0.33	-0.22		-0.25	0.29
Comparison (X:Y:noPref)					
easier early game	(3:2:2)	(2:5:2)		(2:5:2)	(3:3:2)
easier mid game	(2:2:2)	(2:5:0)		(2:3:1)	(2:5:1)
easier late game	(1:3:1)	(3:2:1)		(2:1:1)	(2:5:1)
most close levels	(2:1:4)	(4:2:3)		(3:2:4)	(4:1:3)
preferred gamemode	(2:3:2)	(7:2:0)		(6:2:1)	(4:3:1)
Thoughts&feels					
X					
q1	4.55	4.60	4	5.00	4.38
q2	4.00	3.70	4	4.00	3.63
q3	5.27	5.00	5.5	4.67	5.50
q4	3.64	4.10	2.75	4.22	4.00
q5	5.00	5.60	5	5.56	5.13
q6	4.27	5.10	4.75	4.67	4.63
q7	6.27	5.00	6.5	4.78	6.25
q8	2.45	3.90	2.75	3.78	2.63
q9	4.00	5.90	4.5	4.89	5.13
Y					
q1	4.86	4.09	5	4.44	4.25
q2	3.43	2.91	2	3.00	3.38
q3	5.00	4.91	5	3.89	6.13
q4	4.00	3.64	1	3.44	4.50
q5	5.14	4.27	3	4.44	5.00
q6	3.43	4.73	4	4.00	4.50
q7	6.00	5.91	7	5.44	6.38
q8	2.14	3.00	1	3.00	2.50
q9	4.43	4.64	3	4.22	5.13
Y-X					
q1	0.00	-0.60		-0.56	-0.13
q2	-0.57	-0.70		-1.00	-0.25
q3	-0.14	-0.10		-0.78	0.63
q4	-0.14	-0.20		-0.78	0.50
q5	0.14	-1.20		-1.11	-0.13
q6	-0.57	-0.30		-0.67	-0.13
q7	-0.14	0.80		0.67	0.13
q8	-0.14	-0.70		-0.78	-0.13
q9	0.71	-1.10		-0.67	0.00

Note: The average Y-X values were calculated as the average of the differences, not the difference of the averages

References

[1] P. Sweetser and P. Wyeth. Gameflow: A model for evaluating player enjoyment in games. *ACM Computers in Entertainment*, 3(3), July 2005. Article 3A.

[2] Madison Klarkowski, Daniel Johnson, Peta Wyeth, Simon Smith, and Cody Phillips. Operationalising and measuring flow in video games. In *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, OzCHI '15, pages 114–118, New York, NY, USA, 2015. ACM.

[3] M. Hendrix, T. Bellamy-Wood, S. McKay, V. Bloom, and I. Dunwell. Implementing adaptive game difficulty balancing in serious games. *IEEE Transactions on Games*, 2018. DOI 10.1109/TG.2018.2791019.

[4] G. Andrade, G. Ramalho, A. Sandro Gomes, and V. Corruble. Dynamic game balancing: an evaluation of user satisfaction. *American Association for Artificial Intelligence (www.aaai.org)*, 2006.

[5] Anthony M. Colwell and Frank G. Glavin. Colwell's castle defence: A custom game using dynamic difficulty adjustment to increase player enjoyment. *CoRR*, abs/1806.04471, 2018.

[6] N. Sorenson, P. Pasquier, and S. SiPaola. A generic approach to challenge modeling for the procedural creation of video game levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):229–244, September 2011.

[7] P. Shi and K. Chen. Learning constructive primitives for real-time dynamic difficulty adjustment in super mario bros. *IEEE Transactions on Games*, 10(2):155–169, June 2018.

[8] C. H. Tan, K. C. Tan, and A. Tay. Dynamic game difficulty scaling using adaptive behavior-based ai. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(4):289–301, December 2011.

[9] A. Zook and M. O. Riedl. Temporal game challenge tailoring. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):336–346, December 2015.

[10] A. Isaksen, G. Gopstein, J. Togelius, and A. Nealen. Exploring game space of minimal action games via parameter tuning and survival analysis. *IEEE Transactions on Games*, 10(2):182–194, June 2018.

[11] S. Um, T. Kim, and J. Choi. Dynamic difficulty controlling game system. *IEEE Transactions on Consumer Electronics*, 53(2):812–818, May 2007.

[12] Su Xue, Meng Wu, John Kolen, Navid Aghdaie, and Kazi A. Zaman. Dynamic difficulty adjustment for maximized engagement in digital games. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 465–471, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.

[13] Soren Johnson. Gd column 17: Water finds a crack, June 2011. Available at https://www.designer-notes.com/?p=369, accessed 2019-03-31.

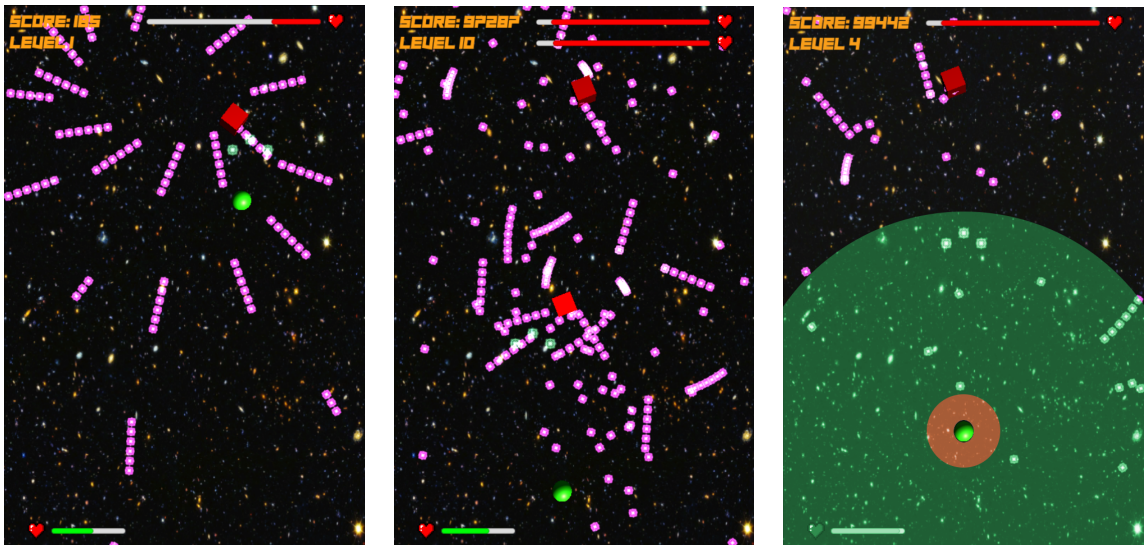
[14] Susan A. Jackson and Kerbert W. Marsh. Development and validation of a scale to measure optimal experience: The flow state scale. *Journal of Sport & Exercise Psychology*, 18(1):17–35, 1996.

9 Appendix A: *BULLETHELL* screenshots

Figure 3: *Main Menu*

Figure 4: *Different Playstyles*

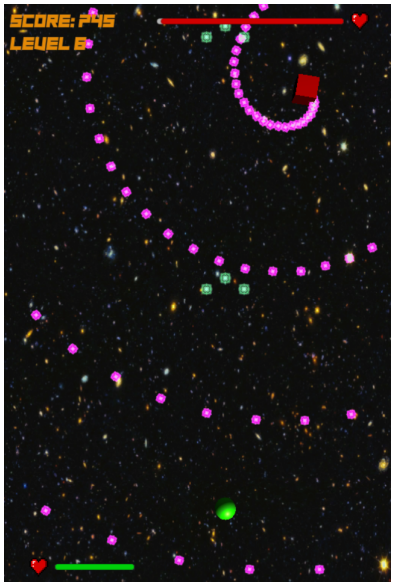
(a) **(b)** **(c)**



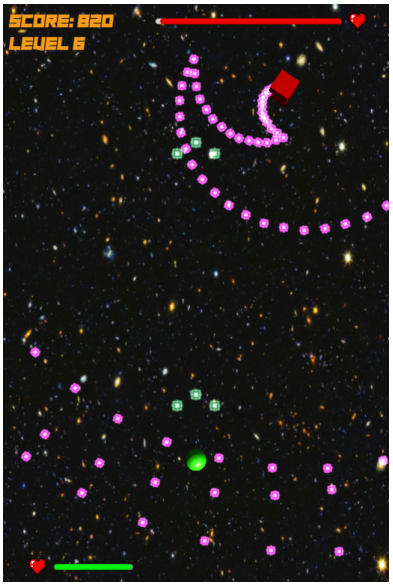
(a) *Shows a player with a Tanky Playstyle.*
(b) *Shows a player trying to use an evasive playstyle while the game is at maximum difficulty.*
(c) *Depicts the space a player could move into in 1 second if they have a stationary playstyle (orange) or a mobile playstyle (green)*

Figure 5: *Different Bullet Pattern Game Behaviours*

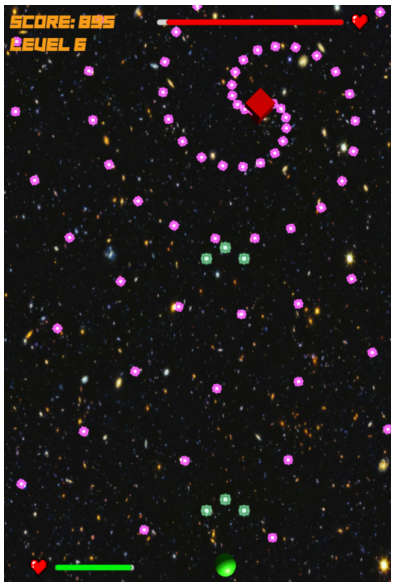
(a) *Default (no behaviours active)*



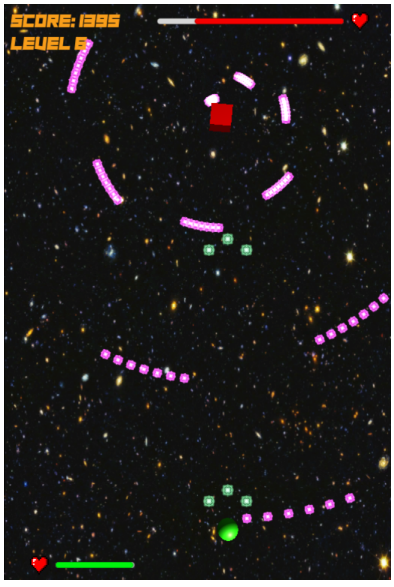
(b) *Reverse Rotation*



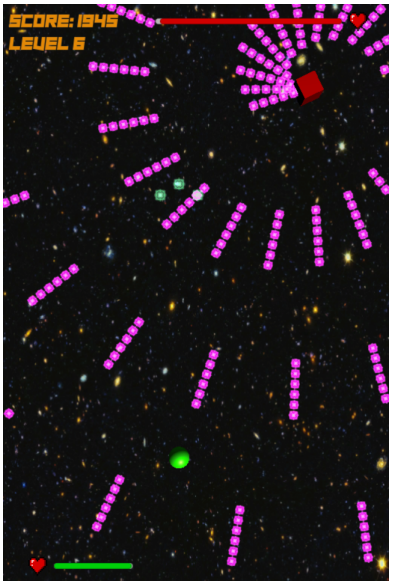
(c) *Directional Fire*



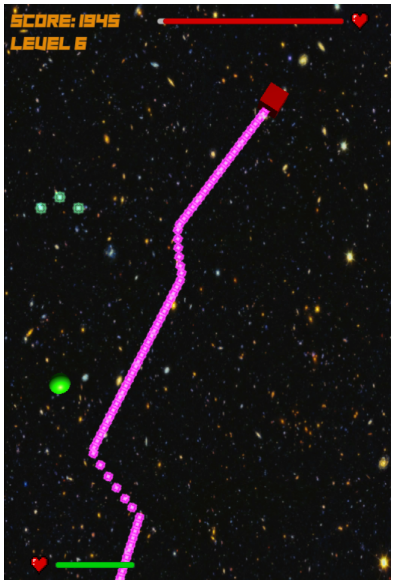
(d) *Burst Fire*



(e) *Lazer Fire*



(f) *Tracking Fire*



Remark: Screenshots were taken with the "Stationary" Movement Pattern Game Behaviour active to make Bullet patterns easier to see. In actual gameplay the Nemesis will be moving, making the pattern less predictable