# Dynamic Difficulty Adjustment & Procedural Content Generation in an Endless Runner

Simon Vidman

**Computer Game Programming, bachelor's level**
**2018**

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

# Abstract

**Dynamic Difficulty Adjustment & Procedural Content Generation in an Endless Runner.**

Games are irritating when they are too hard, and boring when they are too easy. Level Eight is a game development company which is, during this project, developing an endless runner. This thesis describes research and implementation of a dynamic difficulty adjustment system in Level Eights' endless runner by considering several conditions of the player. This was accomplished with the help of the game environment by implementing procedural content generation and combine it with the dynamic difficulty adjustment system.

# Sammanfattning

**Dynamic Difficulty Adjustment & Procedural Content Generation in an Endless Runner.**

Spel är irriterande när de är för svåra, och tråkiga när de är för lätta. Level Eight är ett spelutvecklingsföretag som under detta projekt utvecklar ett ändlöst löparspel. Denna avhandling beskriver forskning och genomförande av ett dynamiskt svårighets justeringssystem i Level Eights ändlöst löparspel genom att överväga flera villkor för spelaren. Detta uppnåddes med hjälp av spelmiljön genom att implementera procedurell innehållsgenerering och kombinera den med det dynamiska svårighets justering systemet.

# Acknowledgments

I would like to start off by giving my thanks to everyone who assisted me along the way.
Special thanks to my examiner Patrik Holmlund for answering questions and giving continuous feedback on this paper.

A big thank you also goes out to Level Eight for giving me a place to do my thesis work and especially Pontus Bengtsson for giving me advice and feedback on the project but also the rest of the team at Level Eight. Lastly, a small thanks go to Jocce Marklund for providing helpful discussions.

# Abbreviations and Terms

- **Branch** - Independent line of development, isolated work
- **C#** - Object-oriented programming language, C-sharp
- **DDA** - Dynamic Difficulty Adjustment
- **PCG** - Procedural Content Generator
- **Chunk** - A pre-built piece of a level
- **GUI** - Graphical User Interface
- **IP** - Intellectual Property
- **Unity** - Game engine
- **L8 -** Level Eight

# Table of Content

# List of Figures

# 1 Introduction

Video games are frustrating when they are too hard and boring when they are too easy, but what makes a game fun and interesting? The nature of fun experiences in games has been the topic of both systematic inquiry and speculations [7]. There are many theories from psychology and game studies focusing on the experiences of playing a game and one of the work is Csikszentmihalyi's concept of flow [8]. Players show a unique learning curve for every game that they play, it thus becomes difficult to entertain each player with preset difficulty levels that are defined by designers [3].

There's also been a lot of discussion about whether games should adapt to the skills of players [25]. However, most current technique limit adaptation to parameter adjustment. But if the parameter adaptation is applied to procedural content generation, then new levels can be generated in real-time in response to a players skill [22]. A game that many people recognize and have dynamic difficulty adjustment is Pocket billiards. The more balls your enemy pockets in, the better are your chances to hit your own balls on your next shot. So the game naturally becomes easier for the losing player and more difficult for the winning player because the scoring game pieces are also obstacles to the opponent [9].

## 1.1 Goal and Purpose

Implementing DDA with PCG in an endless runner game in Unity to keep the player challenged and not bored, was the main goal and purpose of this project.

In conclusion, the goal was:

- Implement dynamic difficulty adjustment
- Implement procedural content generation
- Make dynamic difficulty adjustment & procedural content generation work together.

## 1.2 Limitations

The task was not very clear at the start because I lacked knowledge of dynamic difficulty and L8 were unsure what exactly they wanted for their game. The Endless runner was recently started and was therefore in its early stages, so there were not so many game obstacles to choose from and several gameplay changes were made throughout the weeks. As the game is an endless runner, the difficulty variations to choose from is very limited because the gameplay of an endless runner is fairly uncomplicated.

## 1.3 Level Eight

Level Eight (L8) is an independent game development company located in Umeå, where they, during this project, employed 15 people [10]. It was founded in 2011 by a small core group of eight people who already had extensive experience with handheld gaming and during this project developing games based on the original IPs for the Apple iOS and Google Android platforms. Level Eight has developed 5 games over the past years and their most popular and famous game with over one million daily users is the Robbery Bob series [11]. Their YouTube channel is also quite popular with over 233,000 subscribers and ~41 million views in total [27]. All of the numbers above were accurate during the writing of this project. L8's logotype and Robbery Bob can be seen below in figure 1.1 and 1.2.

Figure 1.1 & 1.2: Level Eight's logotype, left figure [10] and Robbery Bob image, right figure [10]

## 1.4 Background

DDA can be seen as early as 1975, in the game Gun Fight by Midway Manufacturing Co [1]. The game would aid whichever player had just been hit by a shot, by placing an additional object on their side to make it easier for them to hide behind objects the next round. Another early game is Archon [24], from 1983. The computer opponent in Archon slowly adapts over time to help players defeat it.

Procedural level generation in video games existed as early as 1978 when Beneath Apple Manor [31] was released and Rogue [30] in 1980. They were one of the first to use procedural generation to construct dungeons for ASCII- or regular tile-based systems [2].

### 1.4.1 Endless Runner

“Endless running” or “infinite running” games are platform games in which the player character is continuously moving forward through a usually procedurally generated, theoretically endless game world [26], as shown in figure 1.3.

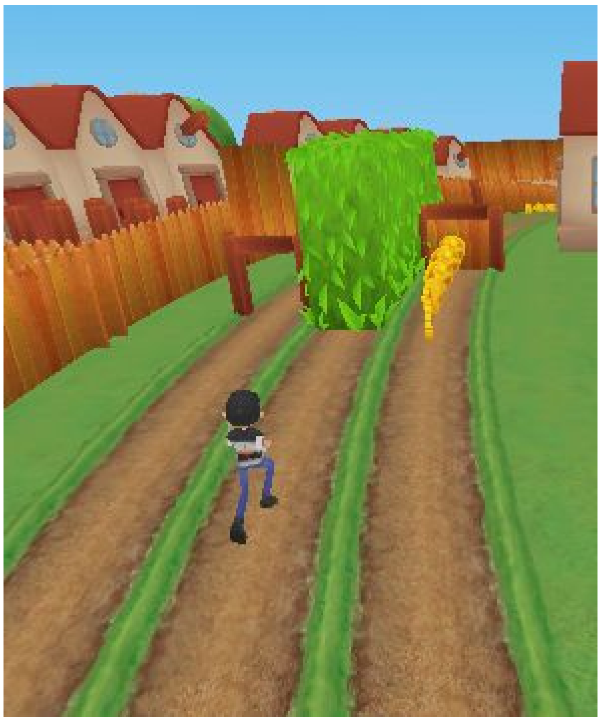


Figure 1.3: Screenshot of the endless runner

The endless runner is well-suited to the small set of controls, they are limited to making the character jump over obstacles, roll under obstacles and slide between different lanes. The main object of endless runners games is to get as far as possible before the character dies because of crashing into an object. As the character runs down the track, the player must collect coins, powerups and avoid obstacles through a combination of the controls mentioned above as the running speed slowly increases.

### 1.4.2 Unity3D

Unity is a cross-platform game engine developed by Unity Technologies, which is primarily used to develop both two- and three-dimensional video games and simulations for computers, consoles and mobile devices [23]. The online documentation is good and detailed, thanks to the international use of the game engine.

### 1.4.3 Atlassian - SourceTree

Atlassian Corporation [12] is an Australian enterprise software company that develops products for software developers, project managers, and content management. One of the products is SourceTree which is a powerful Git and Mercurial desktop client for developers on Mac or Windows. It simplifies how you interact with your Git repositories so you can focus on coding. Visualize and manage the repositories through SourceTree simple Git GUI.

### 1.4.4 Flow

One of the most common approaches to the psychology of the optimal experience is the 'Flow' by M.Csikszentmihalyi [8] [14]. The 'Flow' [6] is the state of consciousness where an individual experiences the peak enjoyment of fulfillment while doing an activity, as described by psychologist Mihaly [8]. The goal of the Flow is to keep the player challenged, interested and not feeling anxious or bored by adjusting the game experience [6], as seen in figure 1.4. But it is not as simple as adding health back when the player is in a bad situation, it is a design problem that involves estimating when and how to interfere with the game.

"*The best moments in our lives are not the passive, receptive, relaxing times… The best moments usually occur if a person's body or mind is stretched to its limits in a voluntary effort to accomplish something difficult and worthwhile.*" - Mihaly Csikszentmihalyi [8]

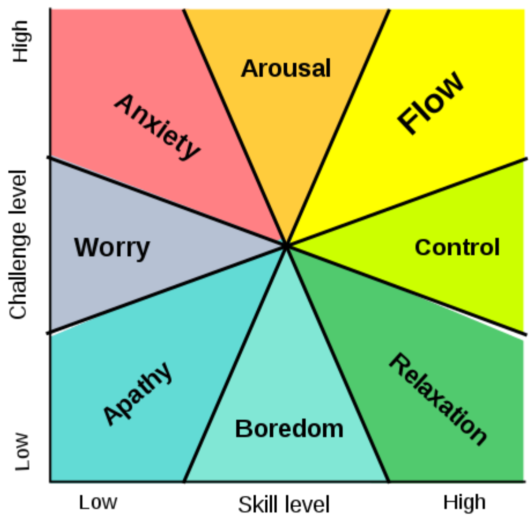


Figure 1.4: The mental state in terms of challenge level and skill level [6]

## 1.5 Related Work

Some examples of games that have taken advantage of DDA or PCG are explained below.

### 1.5.1 Borderlands

The very successful game series Borderlands [15] shows that procedural content generation can even be used in AAA productions to generate more diversity in the gameplay. Borderlands has a countless number of different weapons that are generated procedurally by pieces as shown in figure 1.5. By doing this they create an illusion of almost unlimited technical variety, making the game experience better.

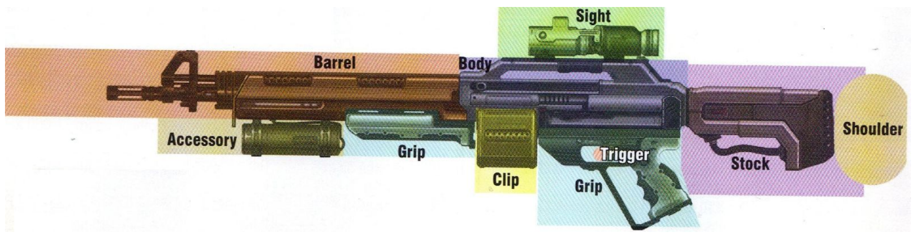


Figure 1.5: Various weapon components that can be combined [28]

### 1.5.2 Crash Bandicoot

Dynamic difficulty adjustment [17] is a system introduced in Crash Bandicoot 2: Cortex Strikes Back [16] and then also implemented in many later Crash Bandicoot platforming games [17], which is designed to help players struggling with a certain level or section of a level. If a player dies repeatedly in a certain section of a level, Crash will be given a ‘powerup’ and will continue to on subsequent deaths until reaching the next checkpoint. Also, there are some crates in various levels that are programmed to become checkpoints after several deaths [17].

### 1.5.3 Infinite Mario Bros

Infinite Mario Bros [4] uses dynamic difficulty adjustment and procedural content generation. One of the inputs to the system is a set of parameters which specify probabilities for a specific event to occur, such as placing a gap. By modifying these probabilities, the system can create a large amount of levels with varying difficulty. So IMB scales up the difficulty by increasing the frequency of gaps, the average size of gaps, a variation of ground height, and the number of enemies.

### 1.5.4 Left 4 Dead

Left 4 Dead [18] has a system called ‘The Director’ that features a dynamic system for game dramatics, pacing, and difficulty. Instead of having set spawn points for all the zombies (enemies), the Director places zombies in varying positions and numbers based upon each player’s current location, skill, status, situation, creating a new experience for each play-through.

### 1.5.5 Mario Kart

A well-known example is a racing go-kart game called ‘Mario Kart’ [19] made by Nintendo [20]. Mario Kart uses a technique called ‘rubber-banding’ in a few different ways. One of them is adjusting the opponents' speed depending on the player’s performance. If the player is doing well, all the opponents will speed up, if the player is not doing well, the opponents will slow down. Another adjustment is the amount and types of power-ups that are available to the player during a race. If the player is not doing well, the player will get more and better power-ups.

## 1.6 Method

The work process during this project consists of a few phases. The first phase was to research, learn and understand more about the main subject and create an overview over the process. This involved reading a lot of relevant blogs, forums, and papers, such as “Staying in the flow using procedural content generation and dynamic difficulty adjustment” by Parekh, R [3]. Dynamic difficulty adjustment is usually built for just one specific game so there is no universal *‘this is how everyone does it’* or a specific algorithm that everyone uses that can be applied to every game.

After the research phase, the focus was to get familiar with the endless runner and unity engine. The engine is fairly easy to acquire a general sense of how to use, but the game was in early development stage so changes were made continuously every day, so playing the game at least once a day was helpful, and take a brief look at the code.

The final phase was finding what different kind of variables I had to keep an eye on in order to calculate and implement the player performance which I could then use to adjust the difficulty, and also build chunks with varying difficulties.

## 1.7 Social, Ethical, and Environmental Considerations

### 1.7.1 Social considerations

One of the largest social impacts would be the same as with many other games. If the user gets addicted to the game it may have a negative impact on the person's personal life, such as sleep deprivation or disrupted food cycle.

### 1.7.2 Ethical considerations

The application saves no user information and this is at no risk of exposing any. No personal or sensitive information such as name, address etc. was collected. Dynamic Difficulty Adjustment will also make the game open to a broad player base because it adapts to every individual player.

### 1.7.3 Environmental considerations

Since the system is created for a game that is released on a digital platform it lessens the need to make physical copies and the game will require very low specifications so there will most likely be no need to purchase a new mobile or device.

# 2 Design and Implementation

The general design of the implementation was to create a dynamic difficulty adjustment system combined with a procedural content generator that would work in Unity3D. It had to be written in the programming language C# because the rest of the company is using C#.

## 2.1 Dynamic Difficulty Adjustment

Dynamic difficulty adjustment is a general term for methods that alter the gameplay to fit the player's performance. DDA is also known as Dynamic Game Balancing (DGB) or Dynamic Game Difficulty Balancing. More in-depth it is the process of automatically changing parameters, scenarios, and behaviors in a game in real-time based on the player's ability & skills. In order to avoid making the player bored (if the game is too easy) or frustrated (if it is too hard). The main goal of dynamic difficulty adjustment is to keep the user interested from the beginning to the end, providing a good level of challenge throughout the whole level. There is no universal algorithm for DDA because it is always designed depending on the type of game.

Deciding what game parameters to take into consideration for calculating players performance is a difficult task and needs considerable experimentation because the result may look very different depending on the parameter. Most games have different kind of parameters and there are only a few available in an endless runner game but some of them are the following:

- The number of coins presented.
- The number of coins collected.
- The number of powerups presented.
- The number of powerups collected.
- The number of successful dodges.
- Total distance traveled.
- Time alive.

But it is very hard to take all of the parameters into account and create something balanced out of it, so there should only be one primary unit of a parameter. Therefor, in this thesis, coins will be the main parameter, so the rest of the parameters are taken in terms of the main parameter. Some game elements that might be manipulated for dynamic difficulty in an endless runner are:

- The speed of the character.
- The frequency of obstacles.
- The frequency of powerups.
- The frequency of enemies (None in this game).
- Powerups duration

The higher speed the character has, the less reaction time the player gets, which makes it more difficult. Same with the frequency of obstacles, if there are more obstacles the likelihood of crashing into something will increase. Meanwhile increasing the chance of a powerup appearing will help and make the game less difficult for the player.

$$Performance = function\ (Skill,\ Difficulty) \tag{2.1}$$

Performance can be defined as a function of skill and difficulty. Above in equation 2.1, you can see the players performance (P) defined mathematically as a function of a given skill (S) and difficulty (D). Which can be used to identify the current difficulty zone.

To calculate the performance we need to add some penalty. Every 10 seconds a penalty is calculated, based on how many coins and powerups you picked up and missed. The penalty can be formulated as equation 2.2:

$$\begin{aligned} Penalty = {} & (20\%\ of\ total\ coins\ collected\ *\ number\ of\ times\ powerup\ lost) \\ & +\ (5\%\ of\ total\ coins\ collected\ *\ number\ of\ coins\ lost) \end{aligned} \tag{2.2}$$

After the penalty, the performance of the player is directly calculated, given as equation 2.3:

$$Performance = ((Total\ coins\ collected\ -\ Penalty)\ /\ Total\ coins\ presented)\ *\ 100 \tag{2.3}$$

There is a penalty for every death depending on the time you stayed alive, the shorter you survived, less difficult. The longer you survived, more difficult. Also the multiplier increases as the more deaths you have, an example can be given as equation 2.4:

$$\varphi = (Time\ alive < Lower\ threshold)$$
$$\Psi = (Deaths * 0.25f)$$
$$Ꚍ = -(Deaths * 0.1f)$$

$$(\varphi \rightarrow \Psi) \wedge (\neg\varphi \rightarrow Ꚍ) \tag{2.4}$$

To receive a more accurate difficulty we take the average of three performance calculations and if the average is between some thresholds we decrease or increase the difficulty, so the final calculation can be seen as equation 2.5.

$$\alpha = (Average\ performance > Increase\ difficulty\ threshold)$$
$$\beta = Increase\ the\ difficulty$$
$$\gamma = Decrease\ the\ difficulty$$

$$(\alpha \rightarrow \beta) \wedge (\neg\alpha \rightarrow \gamma) \tag{2.5}$$

## 2.2 Procedural Content Generation

PCG is an approach to create content automatically with the use of algorithms instead of manual effort which contributes to effective resource usage and expandability. Having the game build the whole level by itself instead of placing it all manually saves a lot of time and work and you get more replayability because of the random factors.

The first main idea was to randomly spawn content all over the lanes based on Perlin Noise [21] and the difficulty provided by the dynamic difficulty calculations. The big issue with spawning content randomly all over the lanes, is that you need an enormous amount of different conditions to make sure the level is playable. This is because the contents are likely to spawn in such a way it blocks all possible paths. After some discussions and research by studying games like Subway Surfers [29], a new decision was made, building the levels with chunks. Chunks are hand-built sequences that can be seamlessly spawned after each other and because they are built by hand they can never be impossible to complete (unless you build them impossible of course). The chunks will also appear as random because you will never know when that specific chunk is appearing. The chunks are built with a custom *'Chunk Editor'* where you start off with an empty grass field, seen in figure 2.7. You can then increase or decrease the length of the grass field and also place the obstacles, coins or powerups at a desirable position, as figure 2.6.

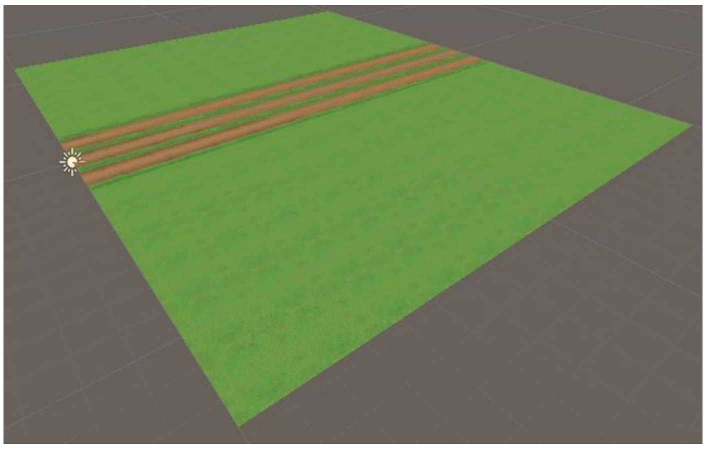
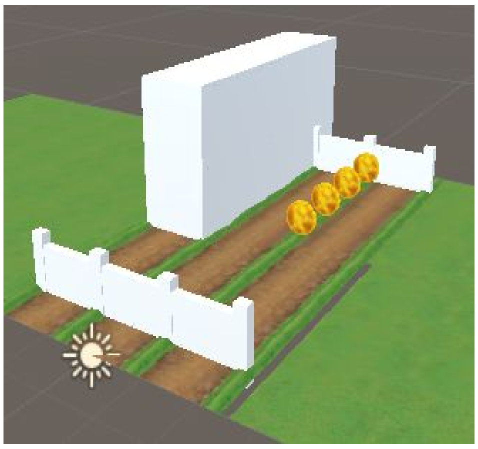
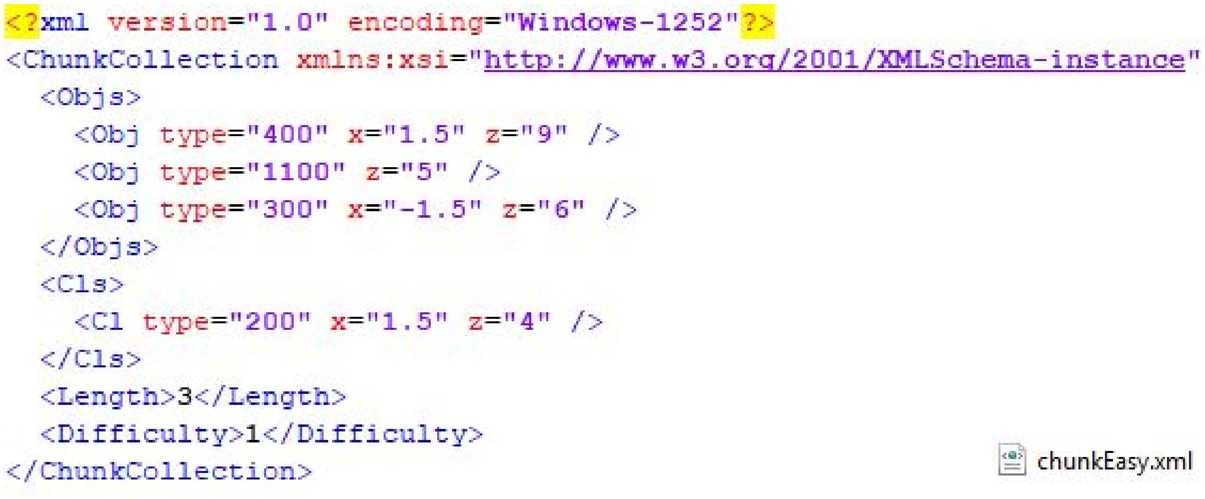



Figure 2.6 & 2.7: Grass field with obstacles, figure left and empty grass field, figure right.

The chunks are exported and imported in the form of XML files as in figure 2.8.



```xml
<?xml version="1.0" encoding="Windows-1252"?>
<ChunkCollection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Objs>
    <Obj type="400" x="1.5" z="9" />
    <Obj type="1100" z="5" />
    <Obj type="300" x="-1.5" z="6" />
  </Objs>
  <Cls>
    <Cl type="200" x="1.5" z="4" />
  </Cls>
  <Length>3</Length>
  <Difficulty>1</Difficulty>
</ChunkCollection>
```

chunkEasy.xml

Figure 2.8: XML Chunk in notepad++

The parameters in the XML file has different meanings:

- **Obj type** - Defines what enum type of obstacle.
- **Cl type** - Defines what enum type of pickup.
- **Length** - The length of the chunk.
- **Difficulty** - The difficulty of the chunk.
- **x, y, z** - position in x, y, z-axis of the obstacle / pickable.

## 2.3 Implementing the System

The implementation of this project was made in C# with the engine Unity3D, inside a branch based on the development-branch, so if something would break then no one else would become affected and whenever there are updates on the development-branch it's easy to pull the new commits.

First, a variable manager is made to keep track of all the different variables. The ones that are kept track of are the following:

- ***int _score*** - Based on distance traveled.
- ***int _coins*** - Coins picked up by the player
  - A script that checks if the player has run past (missed) or collected the coin, is placed on the coin-prefab.
- ***int _coinsMissed*** - Coins missed by the player.
- ***int _coinsTotal*** - Total amount of coins.
  - _coinsTotal = _coins + _coinsMissed.
- ***int _powerup*** - Powerups picked up by the player.
  - A script that checks if the player has run past (missed) or collected the powerup, is placed on the powerup-prefab.

- ***int _powerupMissed*** - Powerups missed by the player.
- ***int _powerupTotal*** - Total amount of powerups.
  - _powerupTotal = _powerup + _powerupMissed.
- ***int _deaths*** - Total amount of player deaths.
  - An event listener to check if the player has died.
- ***int _timeAlive*** - The time the player has been alive in the current round.
  - If the player is alive, += Time.deltaTime
- ***int _timeAliveAverage*** - The average time stayed alive in a round.
  - Every single _timeAlive is put in a list and divided by the size of the list.

All of the above variables are declared as private because the '_'-prefix indicates it as private, but they have public getters & setters to easily set or return a value outside of the variable manager. The manager also contains two reset-functions for the variables. *'resetStats'* that sets every single variable to zero and *'resetStatsForPerformance'* that sets variables specific to the performance calculation to zero.

Next up is building the dynamic difficulty adjustment structure. What we need to calculate is the penalty, the performance, and the difficulty.

To calculate the difficulty we need the performance, and to calculate the performance we need to calculate the penalty. So first we start off by calculating the penalty, and we want to do this every 10 seconds. The variables we will need for this is:

- The number of coins collected
- The number of coins missed
- The number of total coins presented
- The number of missed powerups

Pseudo-code for the penalty implementation in C# can be seen in figure 2.9.

```
InvokeRepeating("calculateThePenalty", 10, 10);                    // Run every 10
seconds

private int calculateThePenalty() {
   float powerupPenalty= ( 20% of coins collected ) * powerups missed;
   float coinPenalty = ( 5% of coins collected  ) * coins missed;
   int penalty = Mathf.RoundToInt ( powerupPenalty + coinPenalty );   // Round to nearest int
   return penalty;
 }
```

Figure 2.9: Pseudo-code for penalty implementation

After we have calculated the penalty we continue by calculating the performance which can be implemented as figure 2.10.

```
private int calculateThePerformance (int penalty) {
   float perf = coins collected - penalty;
   perf = perf / total coins presented;
   perf = perf * 100.0f;                        // Change range to 0-100
   int performance = Mathf.RoundToInt (perf )    // Round to nearest int

   resetStatsForPerformance ();                 // Reset stats for next calculation
   return performance;
}
```

Figure 2.10: Pseudo-code for performance implementation

The performance explains how good the player performed on a range 0 to 100 based on the coins and powerups. Next up, we want to store the performance inside a list, and once the list reaches a size of three, run the '*chooseDifficulty'* function as in figure 2.11.

```
private void chooseDifficulty() {
   for ( int i = 0; i < _performanceList.Count; i++ ) {
      averagePerformance += _performanceList[i];
   }
   averagePerformance = averagePerformance / _performanceList.Count;
   _performanceList.Clear ();

   if ( averagePerformance  >  increase difficulty threshold ) {
      Increase the difficulty
   }
   else {
      Decrease the difficulty
   }
}
```

Figure 2.11: Pseudo-code for difficulty implementation

Then, with the difficulty provided by *'chooseDifficulty'* we either increase or decrease the level difficulty based on thresholds.

Last off is the content generation. As randomly spawning content all over the lanes is very hard to accomplish we chose to go as previously discussed with building chunks by using an in-house *'Chunk editor'*. With the chunk editor, we can increase or decrease the length of the chunks with a single button, place obstacles, coins or powerups wherever we decide. Secondly, set a difficulty on the chunks based on how they are built. A lot of obstacles usually means it is harder and fewer obstacles usually means it is easier. So now we can spawn chunks in real-time based on how the player is currently performing.

# 3 Results

The goal of the project was to implement dynamic difficulty adjustment with procedural content generation in Level Eight's endless runner. A player would play the game for a while, the dynamic difficulty adjustment system calculated the performance of the player and then with the help of the dynamic content generator spawn levels according to the player performance. At the end of the implementation stage, this was the result that we ended up with. The result differed some from the main idea that was planned in the beginning.

## 3.1 End Results

In the final version, the dynamic difficulty adjustment system could calculate the player performance based on coins and powerups collected with coins as the main parameter. Then adjust the difficulty of the upcoming chunks by using the performance provided.

# 4 Discussion

The task of creating a working dynamic difficulty adjustment system with procedural content generation can result in many different ways depending on the game. The goal that was set at the beginning of the project was achieved and worked as expected. Though the first idea was to create a procedural content generator but as some time passed and work had been done we realized the outcome of randomly spawning content all over the lanes wouldn't be appealing to the player, and probably very hard to make playable. So the "randomly spawning content all over the lanes"-project was discarded and a new idea was brought to life. The idea was building small chunks by hand and spawn them seamlessly based on their difficulty which in the end is a more solid solution.

All the chunk difficulties are set by a personal opinion which can have an impact on the gameplay because what the creator sets as a 5 (medium) might feel like a 10 (hard) for someone else. So the result of doing this can be that the player dies early even though that wasn't planned because the chunk is too hard.

## 4.1 Future work

There is of course always room for further improvements and additions. Some future work for this implementation could be:

- Increase difficulty length.
    - Increase the length of the difficulty from 1-10 to a wider range such as 1-100, so we can receive a more accurate and deep difficulty because the 1-10 range only contains 10 difficulty options meanwhile 1-100 has 100 options. Or even change the difficulty to a real number instead of an integer, in order to get an infinite amount of alternatives.

- Increase complexity.
    - The algorithms are fairly simple and could be made more complex and have more contributing factors. This could be both a bad and a good idea depending on the amount of complexity.

- Amount of chunks.
    - The more chunks that are available the more we have to choose from, which will increase the variation.

- Chunk difficulty based on user-input
    - Currently, all the chunk difficulties are set by a single opinion, which isn't accurate at all. A more solid idea would be having a couple of people, play some specific chunks at 50% speed, and afterward they rate the difficulty. Then take an average range of the inputs.

# 5 Conclusion

In this thesis a dynamic difficulty adjustment system was presented by considering several conditions of the player. This was done with the help of the game environment by using procedural content generation to ensure the game is balanced yet still welcoming to beginners and challenging for advanced players respectively. The player’s stats were used to determine and affect the difficulty of the game, some of the collected stats were total coins collected, coins missed, powerups collected, powerups missed.

# 6 References

[1] Wikipedia. (2018). Dynamic Difficulty Balancing. Available at: https://en.wikipedia.org/wiki/Dynamic_game_difficulty_balancing [Accessed 2018-04-06]

[2] Lee, J. (2014). How procedural generation took over the gaming industry. Available at: https://www.makeuseof.com/tag/procedural-generation-took-gaming-industry/ [Accessed 2018-04-06]

[3] Parekh, R. (2017). Staying in the flow using procedural content generation and dynamic difficulty adjustment. Available at: https://web.wpi.edu/Pubs/ETD/Available/etd-042717-113745/unrestricted/raparekh.pdf [Accessed 2018-04-06]

[4] Lewis, C. (2010). Infinite Mario Bros. Available at: https://github.com/cflewis/Infinite-Mario-Bros [Accessed 2018-04-10]

[5] Patel, A. (2013). Noise Functions and Map Generation. Available at: https://www.redblobgames.com/articles/noise/introduction.html [Accessed 2018-04-09]

[6] Csíkszentmihályi, M. (1990). Flow: The Psychology of Optimal Experience. Harper & Row.. ISBN: 978-0-06-016253-5.

[7] José, R & Filipe T. (2014). Procedural Level Balancing in Runner Games. Available at: http://www.sbgames.org/sbgames2014/files/papers/computing/full/303-computingfullpages.pdf [Accessed 2018-04-16]

[8] Oppland, M. (2016). Mihaly Csikszentmihalyi: All about flow & positive psychology. Available at: https://positivepsychologyprogram.com/mihaly-csikszentmihalyi-father-of-flow/ [Accessed 2018-04-16]

[9] Saltsman, A. (2009). Game Changers: Dynamic Difficulty, Available at: https://www.gamasutra.com/blogs/AdamSaltsman/20090507/83913/Game_Changers_Dynamic_Difficulty.php [Accessed 2018-04-16]

[10] Level Eight. (2018). Level Eights homepage. Available at: http://www.leveleight.se/ [Accessed 2018-04-02]

[11] Level Eight. (2018). Robbery Bob: Man of Steal. Available at: http://www.leveleight.se/products/robbery-bob/ [Accessed 2018-04-02]

[12] Atlassian Corporation. (2018). Atlassian homepage. Available at: https://www.atlassian.com/ [Accessed 2018-04-17]

[13] Atlassian Corporation. (2018). Atlassian SourceTree. Available at: https://www.sourcetreeapp.com/ [Accessed 2018-04-17]

[14] Claremont Graduate University. (2018). Mihaly Csikszentmihalyi. Available at: https://www.cgu.edu/people/mihaly-csikszentmihalyi/ [Accessed 2018-06-02]

[15] Gearbox Software. (2018). Borderlands the game. Available at: https://borderlandsthegame.com/ [Accessed 2018-04-17]

[16] Bandipedia. (2018). Crash Bandicoot 2: Cortex Strikes Back. Available at: http://crashbandicoot.wikia.com/wiki/Crash_Bandicoot_2:_Cortex_Strikes_Back [Accessed 2018-04-17]

[17] Bandipedia. (2018). Crash Bandicoot Dynamic Difficulty. Available at: http://crashbandicoot.wikia.com/wiki/Dynamic_Difficulty_Adjustment [Accessed 2018-04-17]

[18] Thompson, T. (2015). In the directors chair: Left 4 Dead. Available at: https://aiandgames.com/in-the-directors-chair-left-4-dead/ [Accessed 2018-06-02]

[19] Nintendo. (2018). Nintendo - Mario Kart. Available at: https://mariokart8.nintendo.com/ [Accessed 2018-06-02]

[20] Nintendo. (2018). Nintendo homepage. Available at: https://www.nintendo.com/ [Accessed 2018-04-17]

[21] Zucker, M. (2001). What is Perlin Noise? Available at: https://mzucker.github.io/html/perlin-noise-math-faq.html#whatsnoise [Accessed 2018-06-02]

[22] Slashdot. (2010). Infinite Mario With Dynamic Difficulty Adjustment. Available at: https://games.slashdot.org/story/10/09/08/055245/infinite-mario-with-dynamic-difficulty-adjustment [Accessed 2018-04-17]

[23] Unity. (2018). Unity3D home page. Available at: https://unity3d.com/ [Accessed 2018-04-19]

[24] Edwards, Benj. (2005). The Secrets of Archon. available at: http://www.vintagecomputing.com/index.php/archives/44 [Accessed 2018-04-26]

[25] Slashdot. (2009). Should Computer Games Adapt To the Way You Play. Available at: https://games.slashdot.org/story/09/10/13/078247/Should-Computer-Games-Adapt-To-the-Way-You-Play [Accessed 2018-05-02]

[26] Chong, B. (2015). Endless Runner Games: How to think and design (plus some history). Available at: https://www.gamasutra.com/blogs/BenChong/20150112/233958/Endless_Runner_Games_How_to_think_and_design_plus_some_history.php [Accessed 2018-06-02]

[27] Youtube. (2018). Level Eights Youtube Channel. Available at: https://www.youtube.com/user/LevelEightVideos/about [Accessed 2018-05-03]

[28] Borderlands Wiki. (2015). Weapons. Available at: http://borderlands.wikia.com/wiki/Weapons#cite_note-chimeric-rifle-1 [Accessed 2018-05-15]

[29] Kiloo. (2017). Subway Surfers. Available at: https://subwaysurfers.com/ [Accessed 2018-05-20]

[30] Moby Games. (2018). Rogue. Available at: https://www.mobygames.com/game/rogue [Accessed 2018-05-31]

[31] Worth, D. (2018). Beneath Apple Manor. Available at: http://www.mobygames.com/game/beneath-apple-manor [Accessed 2018-05-31]