An Investigation into Adaptive Difficulty

1 CONTENTS

2 INTRODUCTION

2.1 WHAT IS DYNAMIC DIFFICULTY ADJUSTMENT?

Dynamic Difficulty Adjustment (DDA) is a broad term give to a system within a video game that reacts to the competence of the player and then reacts accordingly. The main purpose of an adaptive difficulty system is to retain the players attention and give them an enjoyable user experience. (McClure, 2009)

If a video game is too difficult the player will become frustrated and stop playing the game out of anxiety, this is the same on the other end of the scale, if a player is finding a game to easy then they will become bored of doing the same tasks repeatedly. This can be seen visually in figure 1.

Although adaptive difficulty is used often in the industry it has not been covered academically as much as other aspects of the industry, see table 1. However, this paper will still explore what adaptive difficulty is and how it has been used, is being used and could be used within the industry.

Year	Journal Papers	Conference Papers	Theses	Books	Total
2009	1	2	1	0	4
2010	1	7	1	0	9
2011	2	5	0	1	8
2012	3	8	2	0	13
2013	2	5	3	1	11
2014	1	4	1	1	7
2015	1	5	1	0	7
2016	3	5	3	0	11
2017	5	7	2	0	14

Table 1 – Adaptive difficulty research studies from 2009 to 2017 (Zohaib, 2018)

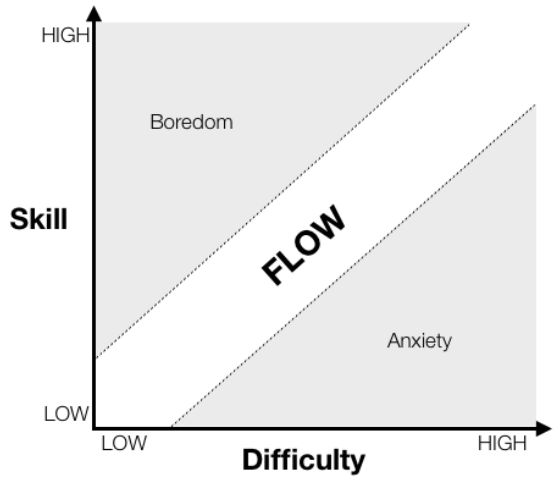

Figure 1 – A graph showing skill in relation to difficulty and how this results in the flow of a video game

2.2 Aim of this Research

To investigate how DDAs have been used in the industry, how they are currently being used and evaluate their links to player retention and experience. Additionally, this paper will evaluate DDA topics within the industry with appropriate academic literature reviews to understand and evaluate:

- DDAs and the different applications within the industry and whether DDAs are applicable to all genres of games.
- The different classifications of DDAs
- How DDAs have been used previously in the industry
- To see if DDAs can be used in sub-genres of the industry such as educational or serious games.

2.3 Objectives of this Research

Research from this paper should create the outline of a DDA system that implements the systems and mechanics described throughout the paper:

- Make use of the available sources to understand the theory behind DDAs.
- Investigate some of the different classifications of DDAs and identify when and why each one should be used.
- Investigate new techniques within the industry that involve the use of DDAs.
- Create a scene in a game engine that implements the research for the second half of this assignment.

3 LITERATURE REVIEW

3.1 USE OF ADAPTIVE DIFFICULTY OVER TIME

Dynamic difficulty adjustment (DDA) is not a new concept within the industry, in fact it has been around for decades with some of the earliest games to use this being Zanac (Compile, 1986) and Gun Fight (Taito, 1975). Zanac made use of a unique artificial intelligence (AI) that would adjust the difficulty based on the skill of the player, the defensive difficulty of the ship and the fire rate of the ship. Gun Fight was released eleven years before Zanac and could balance the game by detecting who had been shot last and putting more obstacles on their side of the screen to give the player more cover.

Ten years after Zanac had been released, Crash Bandicoot (Naughty Dog, 1996) and its subsequent titles made use of a DDA system. An example of how this system worked would be to detect if the player had died to the rolling boulder and if they had then each time the player has died the boulder would be slowed incrementally, this was done to “help weaker players without changing the game for the better players”. (Gavin, 2011)

Resident Evil 4 (Capcom, 2005) made use of DDA, but the system used was called ‘difficulty scale’. This system worked by giving the user a grade that could change over the course of the game, for example, if the player started the game on normal difficulty their initial grade would be four, if they struggled with the game then the grade could drop to a minimum of two and if they found the game to easy then the grade could move to a maximum of seven, this use of a scale meant that the grades could overlap even when playing on different difficulties. This system was so effective that players did not even realize it was being used and was only revealed when it was mentioned in the game’s official strategy guide (Press, Future, 2005). Capcom has continued to use DDAs within the Resident Evil series and they can been seen in the most recent remakes of Resident Evil 2 (Capcom, 2019) and Resident Evil 3 (Capcom, 2020)

3.2 CLASSIFICATION OF DDAS

3.2.1 Probabilistic DDAs

Xue, et al. (2017) conducted a study that addressed the framework of DDA as a problem of optimization and tried to find a set of optimal difficulties over all states. By using a probabilistic two-dimensional graph consisting of the level (k) and trial (t), the game could reach higher numbers of engaged players. Another feature of the probabilistic graph was that the player can only move to one of two adjacent live states from their current live state, it is possible for the player to churn (churn occurs when a player quits the game) at any point as well.

With this above information, three equations can be extrapolated:

Level-up Transition:

$$\Pr(s_{k+1,1}|s_{k,t}) = w_{k,t}(1 - c_{k,t}^{W}) \quad (1)$$

Players can only level up if they complete the level and do not churn and this is what this equation represents. The probability of the player progressing to the next level is the product of the win rate of the level and 1 minus the churn rate after winning.

Retry Transition:

$$\Pr(s_{k,t+1}|s_{k,t}) = (1 - w_{k,t})(1 - c_{k,t}^{L}) \quad (2)$$

Players can only enter the retrial state if they lose the level and do not churn. The probability of a retry is the product of 1 minus the win rate of the level and 1 minus the churn rate of the level after losing.

Churn Transition:

$$\Pr(churn|s_{k,t}) = w_{k,t}c_{k,t}^{W} + (1 - w_{k,t})c_{k,t}^{L} \quad (3)$$

If the player has not made one of the two transitions above it means they must have churned. The probability of churning is the product of the win rate of the level and the churn rate after winning plus the product of one minus the win rate of the level and the churn rate of the level after losing.

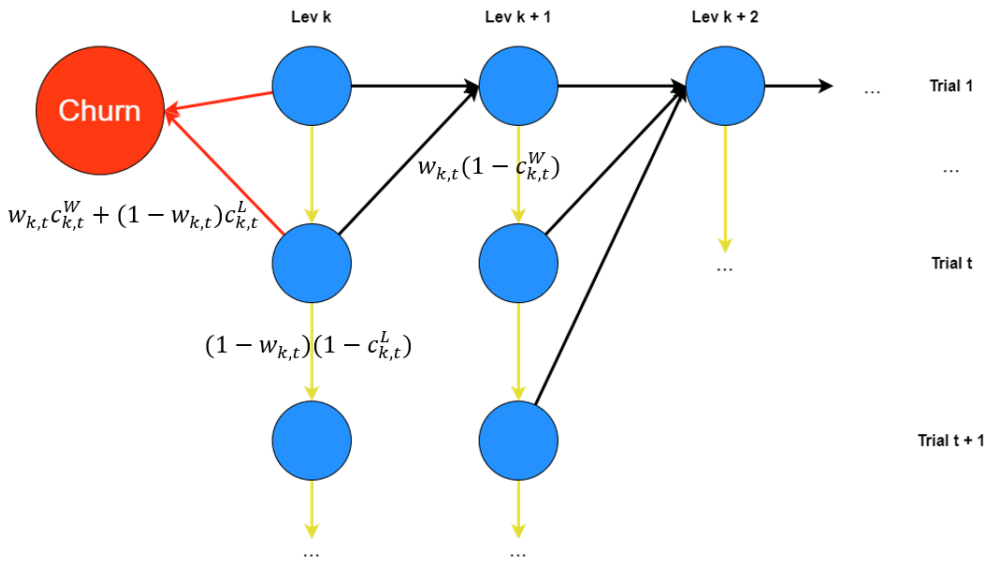

Figure 2 – Probabilistic graph showing the relationship between live states (Xue, et al., 2017)

With equations one, two and three the reward function can be calculated, this equation can be calculated with the help of the Markov property ("memoryless property of a stochastic process" [Markov, 1954]). The Markov property can be used in this instance as the reward of a level is a future state and for one to calculate a future state the conditions of the present state is needed. The present states are in fact known, meaning the following equation can be created:

$$R_{k,t} = \Pr(s_{k+1,t}|s_{k,t}) \cdot R_{k+1,t} + \Pr(s_{k,t+1}|s_{k,t}) \cdot R_{k,t+1} + 1 \quad (4)$$

To calculate the reward of a level, the probability that the player levels up and the probability that the player looses and retries the level needs to be known, this can be seen visually in figure 2. Both probabilities have been calculated already meaning a substitution can be done to create the following and final reward equation:

$$R_{k,t} = w_{k,t}(1 - c_{k,t}^{W}) \cdot R_{k,t+1} + (1 - w_{k,t})(1 - c_{k,t}^{L}) \cdot R_{k,t+1} + 1 \quad (5)$$

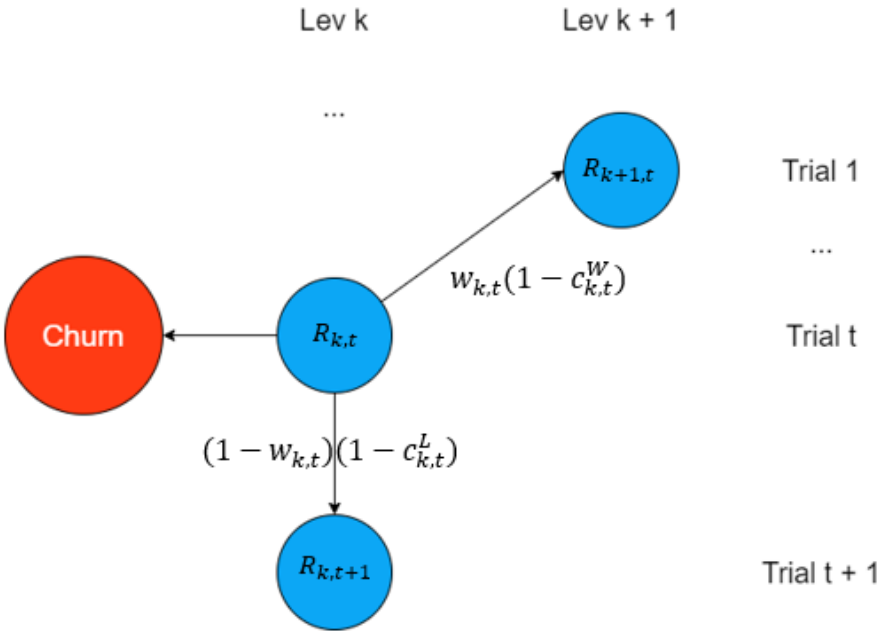

Figure 3 – A more in-depth look at the state transitions with the associated reward states (Xue, et al., 2017)

Finally the problem of optimization can be solved and this can be done by calculating "the maximum reward over all possible difficulty settings" (Xue, et al., 2017), which can be denoted as $R_{k,t}^{*}$:

$$R_{k,t}^{*} = \max \quad w_{k,t}(1-c_{k,t}^{W}) \cdot R_{k+1,t}^{*} + (1-w_{k,t})(1-c_{k,t}^{L}) \cdot R_{k,t+1}^{*} + 1 \tag{6}$$

Seeing as $R_{k,t}^{*}$ is a linear function of $w_{k,t}$ under a constraint meaning equation six can be simplified:

$$w_{k,t}^{*} = \arg max_{w_{k,t}} \quad w_{k,t}((1-c_{k,t}^{W}) \cdot R_{k+1,t}^{*} - (1-c_{k,t}^{L}) \cdot R_{k,t+1}^{*} \tag{7}$$

Equation seven shows that knowing the maximum rewards of the two future state that the difficulty across all win states ($w_{k,t}^{*}$) can be calculated.

Now that the method of the optimization has been described, the optimization of the DDA is now simply finding the maximum value of $R_{1,1}$ in relation to ($w_{k,t}^{*}$).
Furthermore, a test was carried out to prove if the proposed theory could reach the desired outcome. Xue, et al. (2017) performed a test using a mobile game by EA (Electronic Arts), within this game a random seed from 0-99 is chosen to generate the board, after collecting data on the average difficulty of each seed it was discovered which seeds were the hardest and easiest. Using this information and the players progression and the results from the optimal difficulty ($w_{k,t}^{*}$), a suitable

difficulty can be chosen for the player. Furthermore, two test groups were used to see if there was any increased player retention and by looking at tables two and three, a clear increase in player retention can be seen. The mean gain in table one is 6.2% and 6.9% in table two, therefore it can be concluded that the DDA works due to the higher player retention and also answer the initial question set out by (Xue, et al., 2017) asking if DDA could be solved as an optimization problem.

Phase	Platform	Default	DDA	Delta	Gain
1	iOS	1,118,237	1,167,616	+49,379	+4.4%
	Android	789,640	825,182	+35,543	+4.5%
2	iOS	855,267	915,334	+60,067	+7.0%
	Android	1,137,479	1,228,473	+90,995	+7.9%
3	iOS	711,749	763,508	+51,759	+7.2%
	Android	1,285,448	1,366,820	+81,373	+6.3%

Table 2 – Total number of rounds played daily in the default group and the DDA group (Xue, et al., 2017)

Phase	Platform	Default	DDA	Delta	Gain
1	iOS	3,684,082	3,847,516	+163,435	+4.4%
	Android	2,686,781	2,814,953	+128,172	+4.8%
2	iOS	2,916,570	3,148,722	+232,152	+7.9%
	Android	3,787,414	4,129,762	+342,348	+9.0%
3	iOS	2,582,809	2,788,690	+205,881	+8.0%
	Android	4,619,907	4,956,680	+336,773	+7.3%

Table 3 – Total durations of gameplay in the default group and the DDA group (Xue, et al., 2017)

3.2.2 Dynamic Scripting

"Dynamic scripting is an online unsupervised learning approach for games" (Zohaib, 2018). The basis of dynamic scripting is to use a weight system that is attached to all the newly created opponents in the form of a script, the script chooses which abilities the enemy will be given based on weights, the weight algorithm uses the success and failure rates of an ability and will adjust the weights accordingly, for an illustration of how this system may be utilised in a commercial game see figure 3. This system has three enhancements that can be implemented (more detail in section 3.2.2.1) that can make for refreshing gameplay while also adjusting the difficulty for the player.

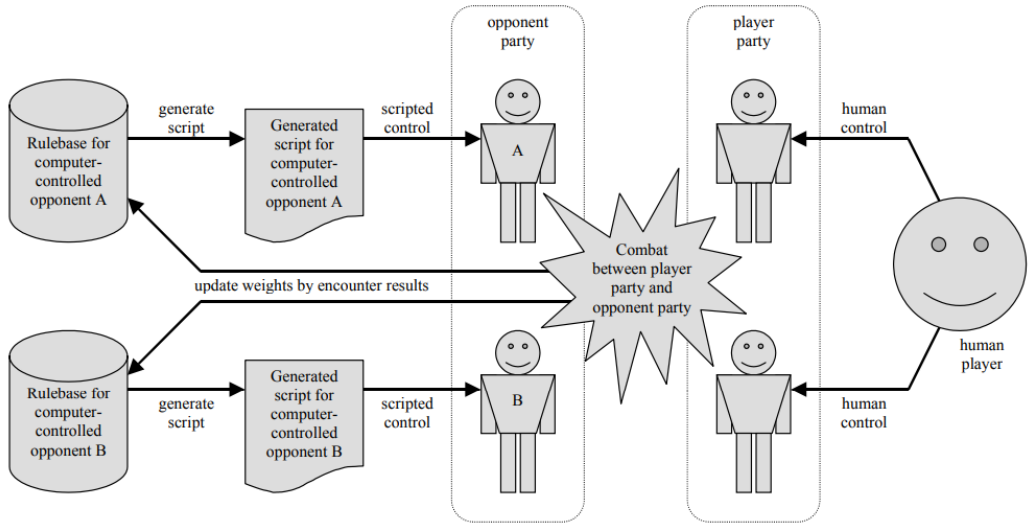

Figure 4 – A visual representation of how dynamic scripting may be used in a commercial game. (Pieter Spronck, 2004)

3.2.2.1 High-fitness Penalising

In high-fitness penalising the weight balancing is proportional to the fitness value (the fitness values are used to know statistics of the player or party such as how much health the player has or how much damage the party has done as a whole) resulting in the rewards also being proportional to the fitness value. This type of approach can be used to make a mediocre difficulty of games by rewarding the mediocre weights and punishing the superior and inferior values.

3.2.2.2 Weight Clipping

Within this system the maximum value of the weight is the deciding factor is the optimal level for a tactic and the maximum value is automatically varied. By doing this it is almost guaranteed that the highest weight values will be chosen as they are the most effective. This works vice versa as a low maximum value of a weight will see it being chosen less as it is non optimal and cannot grow due to the lower maximum therefore creating varied scripts that create a balanced game.

3.2.2.3 Top Culling

A similar concept to weight clipping but the weights can exceed their maximum values, but in doing this it means the weight cannot be chosen for the generated script forcing the opponent to use weaker tactics. In turn if these weaker tactics lead to the opponent loosing then the maximum weights will be decreased meaning more effective tactics can be utilised once again.

Finally, in testing done by (Pieter Spronck, 2004) this form of DDA proved to be effective. However, the three approaches were tested, and high-fitness penalising failed the tests while weight clipping and top culling were found to be effective both in the initial simulation and while being test using Neverwinter Nights (BioWare, 2002).

3.2.3 Reinforcement Learning

With the growing popularity of the video game industry, this then results in more players, this growing player base means a wider variety of play styles and sees the use of static AI that cannot adapt to multiple play styles, and becomes outdated. Furthermore, the use of adaptive AI has been shown to increase player enjoyment as players enjoyed the game more when their opponents adapted to their play styles as it made the match a fair contest Hagelbäck & Johansson (2009).

Expanding on what Hagelbäck and Johansson stated about adaptive AI, Tan, Tan and Tay developed an adaptive AI for making play more even. Within their study they made use of two different algorithms, the first was a adaptive unichromosome controller (AUC) and te second was an adaptive duochromosome controller (ADC).

An AUC “stores one chromosome which encodes seven real numbers (probabilities of activating each behaviour)” (Tan, et al., 2011). When a way point is crossed there is a probability of a behaviour being activated, the chromosome tracks how well the player is performing and created a strategy that would be sufficient at beating the user. When using an AUC, Tan, Tan and Tay concluded that a more entertaining experience was created as it distributed the wins and losses well and the AUC dealt with a variety of opponents.

An ADC works like an AUC but instead of one chromosome there are two chromosomes, and it does not assume the complement of an expected winning strategy to be a losing strategy. One chromosome tracks a winning state while the other one tracks the loosing state simultaneously. If the behaviour wins a waypoint then the winning behaviours in the winning chromosome is the only chromosome to be updated. On the other hand, if a behaviour loses a waypoint then the losing chromosome is the only chromosome that is updated.

In conclusion, Tan, Tan and Tay saw that the occurrence of wins and losses were evenly distributed and that 70.22% of the time the score differential was kept to 4 or less. Additionally, the AUC was seen to be less memory intensive while the ADC

reduced the number of drawn games leading to less players becoming frustrated. Finally, the AUC and ADC would choose different combinations of behaviours to counter different opponents based on the mean score and winning percentage of that opponent.

3.2.4 Self-Organising System and Artificial Neural Networks

A Self-Organising System (SOS) is "a group of entities that presents global system traits through local interactions while not having centralized control" (Zohaib, 2018). This SOS tries to adjust the difficulty by making use of Non-Player Characters (NPCs) and tracking human player traits with the help of an Artificial Neural Network (ANN) as ANNs can adapt to the varied skill levels of players, this is visually represented in figure 5.

In a study by Ebrahimi and Akbarzadeh-T (2014) they used Pac-Man (Namco, 1980) to conduct their research. To start they split agents into two categories, the first being local agents and the second being global agents. Local agents "act based on percept's that collect them from their neighbours" (Ebrahimi & Akbarzadeh-T., 2014) and global agents cover more extended space a.k.a. act globally.

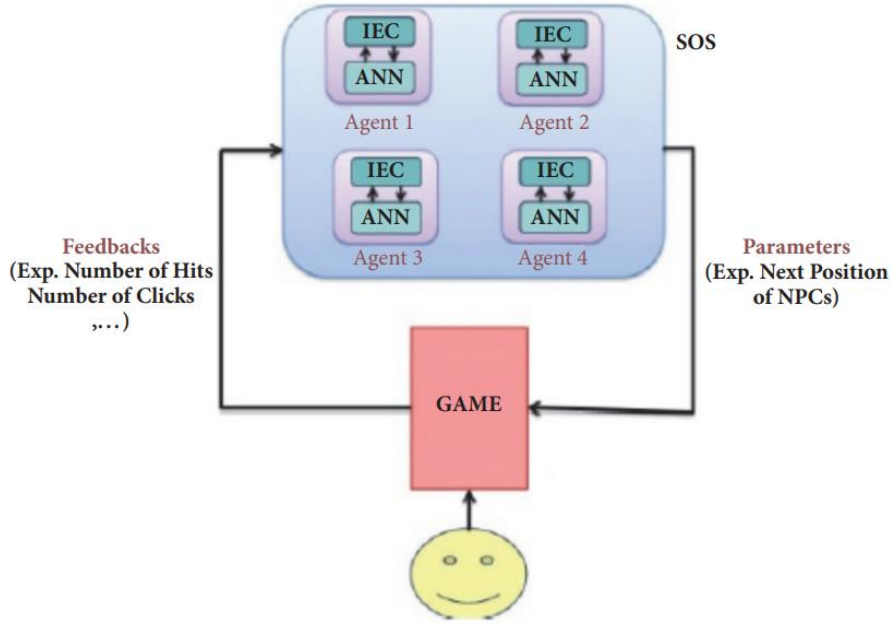

Figure 5 – Visual representation of how the agents will works with the IEC and ANN.

In the research four types of pacman were give different levels of intelligence, these different levels were:

1. Cost-based (CB) pacman will choose its next move based on an inverse relationship with the cost of the next cell, for example, the higher the cost of moving to the next cell the less likely the pacman is to make this move.
2. Nearest distance based (NDB) pacman works on simple logic by deciding what move to make next based on which move will create the most distance from the nearest ghost.
3. Distance-based (DB) pacman works in a similar way to CB pacman but the move to the next cell is proportional to the distance from the ghost cells, each neighbour cell calculates the sum of distances from all ghosts and the cell with the highest total distance is the cell chosen for the next move.
4. Random (RD) pacman is the simplest implementation as it randomly chooses on of its neighbouring cells to move to, all cells have an equal probability to be chosen and if a wall cell is chosen then pacman will not move.

Two types of tests were carried out, offline and online testing. The offline training was used to train neural networks before the online training because performance will be increased while not using random chromosomes. These chromosomes are the basis for the online training of the neural controllers, where an Interactive Evolutionary Computation (IEC) was used, the IEC is an approach to Evolutionary Computation (EC) that replaces the human evolution function with a fitness function.

From multiple tests it was determined that DB pacman had the best results and CB pacman had the worst results. Additionally, an SOS works better as a globalised one then a localised one and this system was able to adapt to several levels of skills.

3.3 DDAs within Serious Games

An aspect of the industry that could benefit greatly from the implementation of a DDA would be serious games and the sub-genre of educational games. The reason a DDA would be appropriate is due to the fact anyone playing an educational game have different levels of cognitive ability meaning it may take them a shorter or greater amount of time to perform a task.

In a paper by Lach (2017) they investigated using modified evolutionary algorithms (EA) within an educational game that is used to teach mathematics. Five modifications were proposed in this paper:

1. *Saved Solutions Table (SST)* – A SST would be used to store all the fitness values of the player. Although an SST would require more storage space than some other solutions, but the exchange is worth the sacrifice in storage as new levels can be evaluated by looking at the saved fitness values in the SST.
2. *Failed Difficulty Point for the Game Element* – Within this approach the trade of is more memory space with less training data. During this method, game elements (GE, each level has these which "can be adjusted to the skills of the player" [Lach, 2017]) are counted and stored with their accompying difficulty points (DP) and the number of lost game levels when this specific DP was used. DP has a failure ratio threshold of lim_F and if this threshold is exceeding then DP is said to have failed, if this is the case then the failure ratio is recursively checked from a start point of easiest DP until a failure ratio below lim_F is found. With this method the number of impossible solutions can be eliminated quicker.
3. *Optimisation with Threshold* – Within this method there is a threshold for the fitness value, lim_O, and when the fitness value is less then lim_O local optimisation is chosen. This optimisation chooses neighbouring solutions and evaluates them and the best one is chosen; this is done until the lim_{Oi} cannot find any better neighbours and the number of optimisations is caped at lim_{Oc}. The only problem with this method is the EA may not find a solution for the fitness value below lim_O.
4. *Optimisation with History* – Similar to optimization with threshold but this is optimized to find the best solution with a fitness value from the previous generation f_{MIN-1}. Additionally, the optimizations for a generation may not exceed lim_{Ogc}.
5. *Optimisation with Probability* – This optimization works with the minimal fitness value f_{MIN-1} but with the weight w_O. A solution is chosen from a population of optimizations with the given probability p_O. To reduce the number of evaluated solutions, a random DP is chosen and then the DP one

above and below is also checked if it will improve the solution, however, this is only done if the fitness value exceeds $w_O \cdot f_{MIN-1}$.

Seven tests were carried out using different combinations of the five methods and it was found that the use of all five methods provided the best results but took longer due to the computational times. On the other hand, the use of methods one and two gave the worst results as the threshold was either not met in time or met to late for the algorithm to initiate. All these results proved that it is possible to decrease the time a player must wait to find a suitable game level meaning this type of approach to educational and serious games could be beneficial given more research and development.

4 CONCLUSION

To conclude this research, it has been seen that DDAs are a generalized term and that there are many ways to classify and implement them. In this paper four DDAs where classified, (Zohaib, 2018) talked about eight different classifications and there may be more that are being researched or have not been published yet. The type of DDA that may be used in a game has to be carefully selected as each DDA serves a select purpose, for example, a probabilistic DDA would not be best suited for a game that has no levels as there would be no retrials of levels. A probabilistic DDA would be more appropriate in a serious educational game that has level progression because the DDA would be able to identify if the player is struggling on a select set of questions or topics and adjust the difficulty of the questions accordingly.

Furthermore, the use of DDAs may become something developers need to seriously consider when making a new game because the use of a DDA can lead to higher player retention (as was seen in table 2 and table 3) and this is needed in the current market of the industry as more games are fighting for the same player base, as seen in figure 6 on Steam, the number of games published ten years ago was 276 as apposed to nine years later when 8,290 games were published. However, not all players may enjoy a game having a DDA as it may lead to them feeling cheated out of a challenge or experience and this is a situation the developers must consider when creating a game as the use of a DDA may lead to lower player retention then not using one at all.

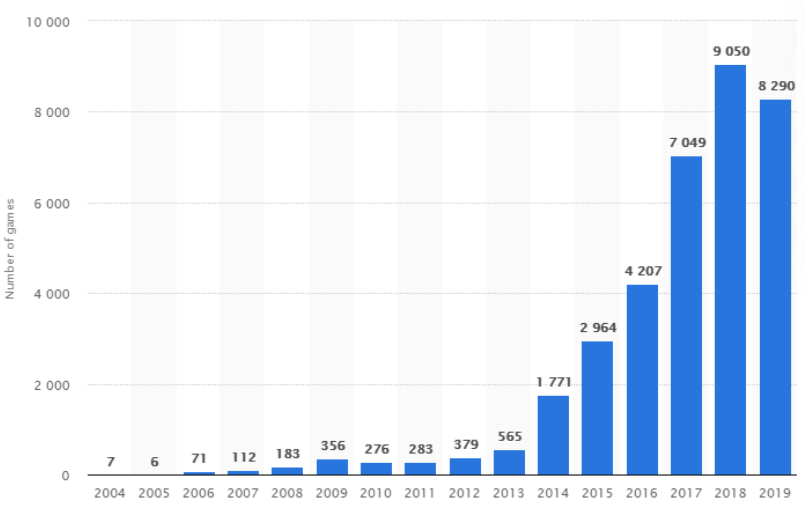

Figure 6 – Number of games published on steam between 2004 to 2019. (Statista, 2020)

5 OUTPUT DESIGN

5.1 OUTLINING THE SOLUTION

For the practical element, a scene will be created that will contain an endless runner that will demonstrate how a DDA works and how it can be implemented. The scene will contain obstacles such as jumps that require timing, otherwise the player will fail, the player will also have obstacles that they need to dodge and a type of collectible they can pick up which will aid in telling the DDA how well the player is performing.

A probabilistic approach, like that described in 3.2.1 will be used as the basis of the DDA. Using data such as how many obstacles the player has collided with will be used to calculate the probability of the user colliding with the next obstacle (equation 8) and in turn the probability of the player dodging an obstacle can be calculated (equation 9). From this information the DDA can adjust the game, if the player is dodging a lot of obstacles the game may be sped up and more obstacles will be spawned making the game more challenging for the player.

$$\Pr(hitting\ an\ obstacle) = \frac{total\ no.of\ obstacles\ hit}{total\ no.of\ obstacles}$$

Equation 8 – Probability of the player hitting an obstacle

$$\Pr(dodging\ an\ obstacle) = \frac{total\ no.of\ obstacles - total\ no.of\ obstacles\ hit}{total\ no.of\ obstacles}$$

Equation 9 – Probability of the player dodging an obstacle

5.2 WHAT SOFTWARE WILL BE USED

To create the scene Unity (Unity Technologies, 2019) will be used as the user interface is intuitive, the component system makes it easy to follow how objects work with one another and its use of C# allows for a more understandable implementation of the DDA. To code the DDA Visual Studio 2019 (Microsoft, 2019) will be used due to its ability to work seamlessly with Unity and how user friendly the program is.

Although these are the main two programs that will be used to create the practical solution, additional programs such as Adobe Photoshop may be used to create UI assets of textures if they are needed in the scene and there is enough free time for these assets to be created.

6 REFERENCES

BioWare, 2002. *Neverwinter Nights.* Paris: Infogrames.

Capcom, 2019. *Resident Evil 2.* Chūō-ku, Osaka: Capcom.

Capcom, 2020. *Resident Evil 3.* Chūō-ku, Osaka: Capcom.

Compile, 1986. *Zanac.* Minato, Tokyo: Pony Canyon.

C. P. S., 2005. *Resident Evil 4.* Chūō-ku, Osaka: Capcom.

Ebrahimi, A. & Akbarzadeh-T., M.-R., 2014. Dynamic Difficulty Adjustment in Games by Using. *Proceedings of the 2014 Iranian Conference on Intelligent Systems, ICIS 2014.*

Gavin, A., 2011. *Making Crash Bandicoot - part 6.* [Online]
Available at: https://all-things-andy-gavin.com/2011/02/07/making-crash-bandicoot-part-6/
[Accessed 15 10 2020].

Hagelbäck, J. & Johansson, S. J., 2009. Measuring player experience on runtime dynamic difficulty scaling in an RTS game. In: Milano: s.n., pp. 46-52.

Lach, E., 2017. *Dynamic Difficulty Adjustment for Serious Game Using Modified Evolutionary Algorithm.* zakopane, s.n.

McClure, D., 2009. *Game Career Guide.* [Online]
Available at:
https://www.gamecareerguide.com/features/805/adaptive_.php#:~:text=Adaptive%20difficulty%20is%20a%20term,that%20the%20games%20challenges%20pose.
[Accessed 6 10 2020].

Microsoft, 2019. *Visual Studio 2019.* Redmond: Microsoft.

Namco, 1980. *Pac-Man.* Ōta, Tokyo: Namco.

N. D., 1996. *Crash Bandicoot.* San Mateo: Sony Computer Entertainment.

Pieter Spronck, I. S.-K. E. P., 2004. Online Adaptation of Game Opponent AI with Dynamic Scripting. *Games & Simulation,* Volume 3, pp. 45-53.

Statista, 2020. *Number of games released on Steam 2004-2019.* [Online]
Available at: https://www.statista.com/statistics/552623/number-games-released-steam/
[Accessed 09 28 2020].

Taito, 1975. *Gun Fight.* Shinjuku, Tokyo: Taito.

Tan, C. H., Tan, K. C. & Tay, A., 2011. Dynamic Game Difficulty Scaling Using Adaptive Behavior-Based AI. *IEEE Transactions on Computational Intelligence and AI in Games,* 3(4), pp. 289-301.

Unity Technologies, 2019. *Unity 2019.1.9.* San Francisco: Unity Technologies.

Xue, S. et al., 2017. *Dynamic Difficulty Adjustment for Maximized Engagement,* Geneva: International World Wide Web Conference Committee.

Zohaib, M., 2018. Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. *Advances in Human-Computer Interaction,* pp. 1-12.