Informatyka techniczna i sztuczna inteligencja
Technical informatics and Artificial Intelligence
Engineer of XXI Century'2024

Adaptive Difficulty Controls for First Person Shooters Game

Adrian Kubiczek [1], Marcin Bernaś [2*]

[1] *Faculty of Mechanical Engineering and Computer Science, University of Bielsko-Biala, Willowa 2, 43-300 Bielsko-Biala, Poland*
[2] *Faculty of Mechanical Engineering and Computer Science, University of Bielsko-Biala, Willowa 2, 43-300 Bielsko-Biala, Poland*
* *Corresponding author, mbernas@ubb.edu.pl*

Abstract: The article presents the author's method of controlling game parameters using a neural network. A literature analysis was performed, and then for the purposes of the work, a game level was designed, in which the player can pass the game level in various difficulty modes. Then, an adaptive control level was proposed with a neural network tuned by heuristics and random player transitions. As a result of tuning, a network has been learned that allows you to adjust the gameplay parameters to the player's requirements. For the created environment, research was carried out on a selected group of players. The results confirm the legitimacy of using adaptive difficulty levels in games. The paper also indicates the directions of development.

Keywords: 3D games; neural networks; adaptive control;

Adaptacyjne sterowanie poziomem trudności w strzelankach FPS

Adrian Kubiczek [1], Marcin Bernaś [2*]

[1] *Uniwersytet Bielsko-Bialski, Wydział Budowy Maszyn i Informatyki, Willowa 2, 43-300 Bielsko-Biala, Polska*
[2] *Uniwersytet Bielsko-Bialski, Wydział Budowy Maszyn i Informatyki, Willowa 2, 43-300 Bielsko-Biala, Polska, mbernas@ubb.edu.pl* [3]
* *Corresponding author, mbernas@ubb.edu.pl*

Streszczenie: W artykule przedstawiono autorską metodę sterowania parametrami gry za pomocą sieci neuronowej. Przeprowadzono analizę literatury, a następnie na potrzeby pracy zaprojektowano poziom gry, w którym gracz może przejść poziom gry w różnych trybach trudności. Następnie zaproponowano poziom sterowania adaptacyjnego z siecią neuronową dostrojoną przez heurystykę i losowe przejścia między graczami. W wyniku strojenia powstała sieć, która pozwala dostosować parametry rozgrywki do wymagań gracza. Na potrzeby stworzonego środowiska przeprowadzono badania na wybranej grupie graczy. Wyniki potwierdzają zasadność stosowania adaptacyjnych poziomów trudności w grach. W artykule wskazano również kierunki rozwoju.

Słowa kluczowe: Gry 3D; sieci neuronowe; Sterowanie adaptacyjne;

1. Introduction

Computer games have become an integral part of modern culture and entertainment, gaining huge popularity around the world. The popularity of computer games can be attributed to a number of factors, such as the increasing availability of technology, the development of the internet, and innovative approaches to interactive content creation [1][2]. One of the key aspects of successful video games is their ability to attract and engage a wide community of gamers. Game developers use a variety of strategies to encourage players to return to their titles on a regular basis [3]. Some of the most common methods include an attractive storyline, complex and interesting game mechanics, regular content updates, and the integration of social elements such as the ability to cooperate or compete with other players online [4].

DOI: https://doi.org/10.53052/9788367652292.09

One of the most important challenges game developers face is adjusting the difficulty level to match the skill and preferences of the players. The traditional approach is to set fixed difficulty levels from which the player can choose the one that best suits their skills. However, this solution does not always provide an optimal experience for all players.

In response to these challenges, you can choose a technique called Dynamic Difficulty Adjustment (DDA) [5]. It allows you to customize the challenges you face in real time based on your current skills and progression in the game. Difficulty manipulation can take many forms, from subtle changes in enemy behaviour, to adjusting the amount of resources available to the player, to dynamic modifications to missions and challenges [6]. Studios still rarely decide to use DDA in their productions. This may be due to the structure of the mechanics itself or the nature of the computer game. However, there are still examples of big titles that are a great study of how dynamic difficulty adjustment can improve the player experience - for example, "Left 4 Dead" or "Resident Evil 4". The AI (Artificial Intelligence) system responsible for managing the zombie hordes adjusts their number, aggressiveness and attack strategies depending on the skills and progress of the players. This allows even experienced players to enjoy a challenge that constantly adapts to their level. By examining the mechanics used in such titles, it can be concluded that the effective use of artificial intelligence to dynamically adjust difficulty can significantly improve the experience of players, providing them with appropriately balanced challenges and satisfaction from the gameplay. By analysing the implementation of DDA in games of various natures, it is also possible to draw conclusions about the effectiveness of various adaptive difficulty control strategies and identify best practices that can be used in the computer game being created. Because artificial intelligence (AI) is a discipline of computer science that specializes in creating intelligent systems and algorithms, it is capable of effectively replacing tasks that traditionally require human attention [7]. In the context of computer games, it often plays a key role, controlling the behaviour of computer opponents, allies or other elements of the game world.

AI systems in games can be programmed to respond to different stimuli, make decisions, develop strategies, and learn from experience [8]. The most fitting model in this context seems to be the neural network, which is one of the techniques of artificial intelligence. A neural network is a mathematical model inspired by the biological nervous system that consists of interconnected artificial neurons. Neural networks are capable of learning from data, so they can adapt to changing conditions and make decisions based on the analysis of multiple parameters. In the case of computer games, neural networks can be used to analyse player behaviour and dynamically adjust the difficulty level on the fly. In order to assess the effectiveness of the neural network in the context of DDA, research will be carried out on a selected group of people.

This research will enable the collection of data on players' experiences and reactions to different levels of game difficulty, adjusted by the AI. The analysis of the collected information will allow to assess the effectiveness and satisfaction of users with the dynamic game adjustment system used, which in turn will contribute to the further improvement of this technology in the context of computer games. This paper presents a comprehensive study of the topic of adaptive difficulty control in computer games with the use of artificial intelligence. The presentation of theoretical foundations, analysis of the selected case and the results of the research will allow for the formulation of practical recommendations for game developers. The development and popularization of adaptive difficulty systems can contribute to the creation of more rewarding and engaging games that will potentially be able to better respond to the diverse needs of players [9][10].

2. Related works

Various DDA methods have been proposed in the literature [11], of which the following are based on artificial intelligence. The only common aspect of all the methods given is the requirement to measure (in a way that can be hidden or explicit) the level of difficulty encountered by the player at any given time. These measures are estimated by heuristic functions, also known as challenge functions, which assign a value for any given game state that indicates the level of difficulty the player is feeling at any given time.

In the first approach, DDA can be defined as an optimization framework, where the global goal is to maximize player engagement throughout the game [12]. The player's progress in the game is modelled as a probabilistic graph, consisting of different player states. During gameplay, players move from one state to another, and the probabilities to transition between states depend on the difficulty of playing in those states. Maximizing player engagement is equivalent to maximizing the number of playthroughs in the progression graph. This goal comes down to the difficulty function of the game in different states, which can be solved with dynamic programming. Although the DDA framework is discussed in the context of level-based games, it is universal and can be extended to other game genres.

A study by Pedersen, Togelius, and Yannakakis [13] examined the relationship between level design parameters in platform games, player experience, and individual game characteristics. The design parameters studied were related to the size and placement of levels, and the components of the player experience included frustration, fun, and challenge. The neural network model, which mapped the player's behaviour characteristics, emotion, and level design parameters, was trained using data from game sessions and evolutionary learning of user preferences. Dynamic scripting is a method of online unsupervised learning for games. It is computationally fast, robust, efficient, and effective [14]. It supports multiple rules in the game, running one for each type of opponent. These rules are manually designed using domain-specific information. When a new opponent is created, the rules that create the script to guide the opponents are retrieved from the rule database based on their type. The probability of selecting a script rule depends on the weight value assigned to the rule. The rule base adjusts by changing the values, reflecting the failure or success rates of the associated script rules. Most games use the concept of inventory, which is a store of items that the player collects and collects throughout the game. The relative size or absence of items in inventory directly affects the players' experience. Games are designed to control the exchange of items between the player and the world. These links between the "producer" and the "consumer" can be seen as an economy or a dynamic system.

Hamlet - the DDA system built by Hunicke and Chapman [15], uses methods taken from operations research and inventory management. It studies and adjusts the supply and demand of in-game inventory to manipulate the difficulty of the game. The system is essentially a group of libraries maintained in the Half Life engine. In the case of adjusting the computer's movements to the player's skills so that they are equal, the DDA is carried out in real time. The goal is primarily to keep the user in a given environment for a longer period of time to keep them entertained and enhance their experience. The adaptive AI in the game must be proficient enough to make unpredictable but rational judgments like human players, but it must not exhibit overtly mindless behaviour. The AI must also be able to correctly assess its opponent at the beginning of the game itself and adapt its playstyle to the opponent's skills. In this study, two adaptive algorithms were proposed: adaptive uni-chromosome controller (AUC) and adaptive duo-chromosome controller (ADC), which use the concepts of evolutionary computation and reinforcement learning [16] for adaptive real-time play. Another example is Upper Confidence Bound for Trees and Artificial Neural Networks. It is a DDA technique that uses artificial neural networks (ANNs) based on data derived from the upper trust limit for trees (UCT). The game "Pacman" was used as a testing ground in this study. Given that UCT is a computational intelligence method, the performance of UCT significantly correlates with the duration of the simulation [17]. This means that the neural network can be trained based on the data created by the UCT. Even though this approach can be implemented as a DDA in games like Pacman, it is not practical for complex online games due to the computational intensity. However, because the performance of the UCT can be adjusted by changing the simulation time, it becomes possible to train the ANN offline by running the data created by the UCT with the changed simulation times. In this way, DDA can be generated from the created data, bypassing the computational intensity. In this study, the 3-Layered Feed Forward Artificial Neural Network model in the WEKA software was used for implementation [18].

A final example was the development of a self-organizing system (SOS), which is a group of actors that demonstrate the global characteristics of a system through local interactions, without centralized control [19]. This method proposes a new technique that attempts to adjust the difficulty by creating an SOS of non-player characters (NPCs) that are not controlled by the player. Neural networks are used to track the player's human characteristics in the system. Neural Networks (ANNs) must adapt to players with different skill levels and characteristics; therefore, an evolutionary algorithm with adaptive skills has been developed that modifies ANN weights.

3. Proposed Method and implementation of adaptive system

As part of the work, the use of a neural network as a mechanism for adjusting the difficulty level in the game was proposed. The solution is based on downloading information about the game m.in the player's life, their progress in the game and adjusting the gameplay parameters to it. The general model is shown in Figure 1.

DOI: https://doi.org/10.53052/9788367652292.09

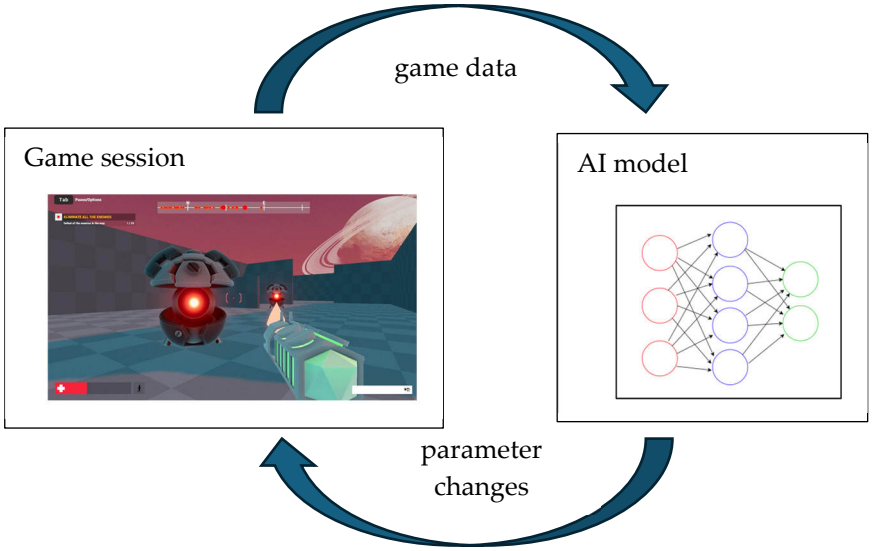

Figure 1. A model of interaction with the game in the form of a feedback loop**.**

Both the game data and the modification of parameters should be adapted to the type of game and the neural network model adopted. DDA is not one of the simplest functionalities that can be implemented in your own environment. Nevertheless, neural networks, thanks to their ability to learn from large data sets, are an effective tool for implementing these mechanisms. One of the main advantages of using neural networks in DDA is the ability to adjust the difficulty level in a more precise and personalized way. Traditional DDA methods often rely on simple rules or sampling that may not account for all the variables that affect the player experience. Neural networks, learning from data collected during gameplay, can take into account a number of complex factors, such as the player's reaction rate, their playing style, and their challenge preferences. This makes it possible to create a more engaging and rewarding experience for the player. Neural networks can continuously analyse gameplay data and adjust DDA algorithms on their own, saving developers time and resources. In addition, thanks to predictive capabilities, neural networks can predict which elements of the game will be most engaging for a given player and adjust the difficulty level accordingly [20]. In addition, training them is a time-consuming process and requires significant computational resources [21]. Then there is the interpretability and transparency of neural networks. These models often act as "black boxes", which means that it is difficult to understand why they made certain decisions about how to adjust the difficulty level [22]. A lack of transparency can be problematic, especially when players feel that the game is unfair or the difficulty is not suited to their skills. There is also a risk that ANNs can lead to unpredictable or undesirable effects [23]. Therefore, it is important to carefully test and monitor the performance of DDA algorithms to ensure an optimal experience for users.

3.1. Design environment

The created environment consists of one project in the Unity engine, which is a first-person shooter (FPS) game. The aim is to compare the artificial intelligence technique used to control the levels adaptively with the classic settings. This was made possible thanks to the game engine, which is Unity, and technologies such as TensorFlow for training a neural network model, or the Barracuda library for implementing and servicing the model in the environment. The player finds himself on one of the distant planets, which is overrun by an advanced civilization and must face opponents in the form of robots. The time it takes the user to go from start to finish for each mode takes no more than ten minutes. It is therefore a short game developed for the needs of the DDA implementation, and thus – the neural network model is configured for this specific type and length of the game, so that the results of the analysis for the presentation part are used as part of the visible manipulation of statistics. While the neural network can be used in further research in both this and other projects, it should then be properly adapted to the environment to achieve the best effect.

3.2. The model and rules of the game

The entire game model is divided into several scenes occurring in a specific order, depending on the setting of the game. The project uses the "FPS microgame" asset made available on the Asset Store or directly on the Unity Hub. This template offers a small world, complete with pre-built in-game objects such as: walls, basic FPS game mechanics (including shooting, taking damage, moving), enemies, weapons, health aid kit, stats, and many more. The player takes on the role of the main character, who has the ability to go in any direction, jump or crouch. It is equipped with a laser patch to attack enemies, so you can fire both in full-auto and single fire. The player selects the difficulty settings and also becomes familiar with the actions assigned to the keys, which is shown in Figure 2.

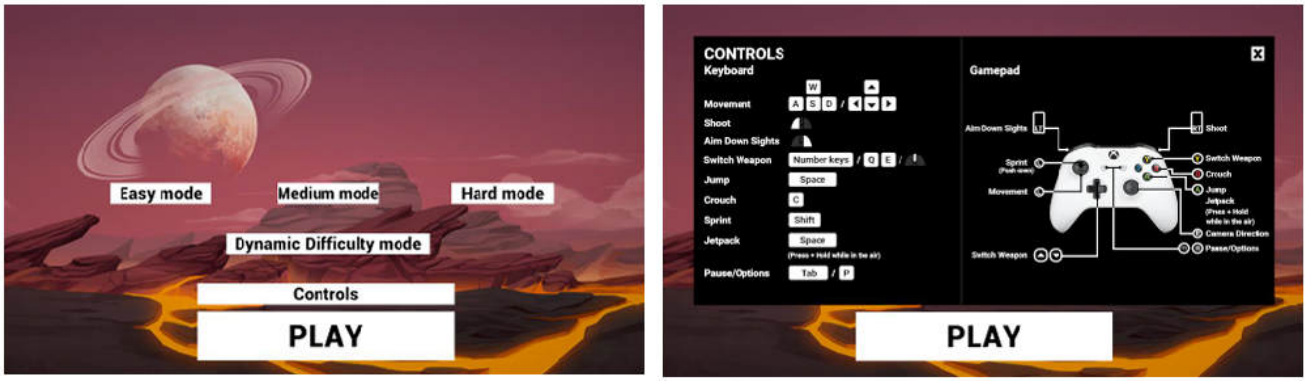

Figure 2. GUI and steering of a game

The player can move in any direction, jump, accelerate, shoot, crouch, and change the difficulty. Later on, the user is transferred to the game world, where the main process takes place. After going through all the obstacles or being defeated by enemies, the end of the game window is displayed. The player is asked to test the selected difficulty levels.

3.3. Game environment

The game has been implemented in Unity and allows you to create objects in the hierarchy, which in turn can be used in a given environment. The most important of them, however, is the *GameManager*, which is responsible for the HUD and stores the neural network model, whose statistics are displayed in the upper right corner during gameplay. The player has his health, amount of ammunition, game objective and compass displayed during the game, on which all enemies are marked. Then there is the part labelled Enemies. As in the case of the player, all objects of two types of enemies are grouped here – a smaller moving robot and a larger static turret. These are bots that start shooting at the main character as soon as they get too close. They are also accompanied by appropriate scripts relating to mechanics and behaviour, as well as hitboxes [24] and visual models. In addition, the "Paths" tab stores enemy patrol areas that are located on the map. The game environment in the project consists of two small platforms and one larger one, on which the opponents and the player are placed (Figure 3).

Objects such as walls or obstacles (health packs also belong here) combine to form rooms, and all of them are assigned to the "Level" part of the hierarchy. The player can move around these areas and progressively progress. The platforms consist of appropriately placed walls of various sizes, lengths and widths, creating rooms. The entire map is limited, but the player can return to the previous room at any time.

There are obstacle opponents in the game and their behaviour is the main part of the whole project observed in this work. They are designed to offer a variety of challenges and keep the player on the edge of their seats. Their changing behaviours and skills when choosing different levels of difficulty, force the player to use different strategies and adapt to the conditions on the board.

DOI: https://doi.org/10.53052/9788367652292.09

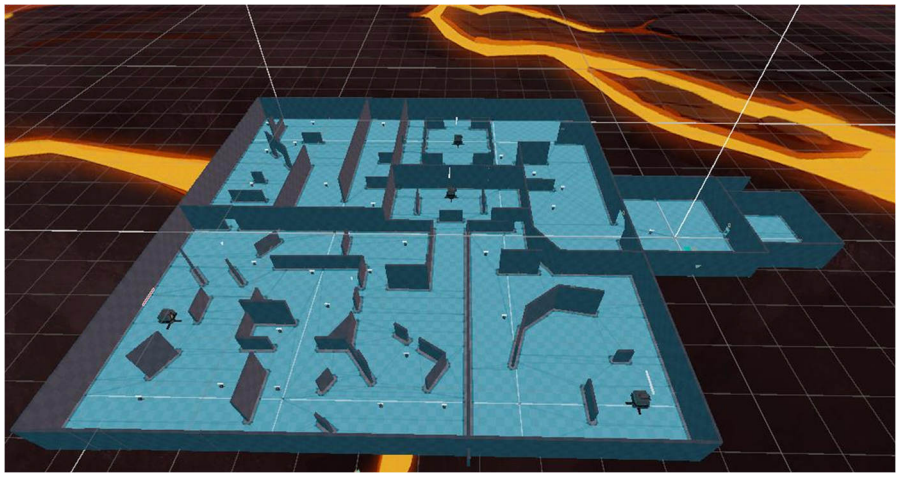

Figure 3. Designed level for testing.

In this project, two types of enemies (Figure 4) are used to attack the player from a distance – a patrolling robot and a static turret. The first is a small robot that attacks the player with a single laser of a given frequency, depending on the difficulty level, set on the initial scene. They appear on the map in limited numbers, thus creating obstacles for the player and a significant threat of not completing the game. Its visual assets were imported along with "FPS Microgame".

Figure 4. A view of enemies in the game.

Its main difficulty characteristics are damage to the player, the speed of shots and maximum health. When you set the modes from easy to hard (and dynamic difficulty), these parameters are either low or high, respectively.

Difficulty modes

The level of difficulty plays a key role in shaping the players' experience in this project and is the basis for analyzing the results. The game implements a total of four difficulty levels (Figure 2), including the classic scheme – easy, medium and hard modes. Attached to this is the main mode, i.e. Dynamic Difficulty mode, which is a representation of DDA. These are set for the entire game, with the ability to switch between them after the game is over. The difficulty levels are displayed on the stage where the menu is presented. Enabling one of the difficulty levels means setting the parameters of the enemies in a specific way, which include: damage, shooting speed and maximum enemy health After selecting a level, the game will begin, which can be interrupted at any time to end or try another mode. In the event of death or defeating all enemies, a separate window will appear with a message indicating the result of the game. Then the player will have to choose two options – go to the menu, where you can choose a different level, or play again. If the player chooses the Dynamic Difficulty level, the difficulty will be set in real time based on the implemented neural

network model. So the nature of the game will be strictly dependent on the player's skills, reading data based on his progression and then updating the parameters of enemies every now and then.

3.4. Implementation of DDA

The process of training a network model in TensorFlow involves several key steps. The first step is to define the model architecture, which includes the appropriate connection to the environment. Next, you need to prepare training and test data, often requiring processing and transformation, which TensorFlow makes possible by supporting various data formats and processing operations. Another key element is the definition of the loss function, which is used to assess the model's error during training. TensorFlow provides a rich set of built-in loss functions and allows you to define custom functions tailored to the specifics of the problem. Then it is necessary to configure optimization, i.e. choose the right optimization algorithm for the best results. During training, which is an iterative process of fitting model parameters to training data, TensorFlow provides tools to manage the loop and training and monitor learning progress. Once the model is trained, it is necessary to validate and evaluate its performance on the test data to assess its ability to generalize and deal with new data.
The structure of the neural network designed and trained in the presented code consists of three layers and was created using the TensorFlow library and Keras as follows:

```
model = Sequential([
    Dense(8, activation='relu', input_shape=(3,)),
    Dense(8, activation='relu'),
    Dense(3, activation='linear')
])
```

The main idea of this network is to process inputs about in-game parameters such as accuracy, number of enemies defeated, and damage received to predict specific enemy parameters based on this data. The input data to the network is in the form of three features, and each of the data samples is represented by a vector with dimensions "(3,)". These are, respectively, accuracy, the number of enemies defeated and damage taken. This data is normalized from 0 to the maximum values specified for each characteristic. Then, based on these inputs, the corresponding targets (outputs) are generated, which in this case are in the form of three parameters, with a value dependent on simple rules based on the values of the input features. The neural network defined in the code is a sequential network, which means that the layers are added in a linear fashion, one after the other. The network consists of three layers: Hidden Layer 1: This is the Dense layer with eight neurons and the ReLU (Rectified Linear Unit) activation function. This layer takes input data with dimensions "(3,)", which corresponds to three input features. ReLU is a popular activation function in neural networks that introduces nonlinearity into the model, which allows the network to learn more complex relationships in the data. Hidden Layer 2: Like the first, it is also a dense layer with eight neurons and a ReLU activation function. This layer processes the output from the previous layer, further transforming feature representations in more complex ways. In this case, the "input_shape" parameter is not required because TensorFlow will automatically shape the input based on the previous layer. Output Layer: This is also a dense layer, but with three neurons and a linear activation function. The choice of a linear activation function in the output layer is deliberate, because the network is supposed to predict continuous values that can take arbitrary real values. Each of the three neurons in this layer corresponds to one of three parameters that the network is supposed to predict. The model was built using the Adam optimizer, which is an adaptive version of the simple gradient algorithm. This optimizer is popular for its efficiency and ability to quickly converge to a minimum cost function. The mean squared error (MSE) was chosen as the loss function, which is commonly used in regression tasks because it penalizes large errors in predictions more severely. In addition, the model monitors average absolute error (MAE) and accuracy (acc) as additional metrics used to create charts that monitor training.

3.5 Data collection from players

One of the important elements is to collect the input data required by the designed network. The process begins by adding up the various enemy metrics stored in the base. These metrics include *Accuracy, KillFrequency, BulletShooted, and DamageTaken*. The method then calculates the differences between the current indicator values and their previous

values, which are stored as the *previousAccuracy*, *previousKillingDelay*, *previousBulletShooted*, and *previousDamageTaken* variables. These differences are calculated using the Mathf.Max function to ensure that the results are not negative, which is important for further data processing and analysis. After the differences are calculated, the method updates the *previousAccuracy*, *previousKillingDelay*, *previousBulletShooted*, and *previousDamageTaken* variables to the current values so that they are ready to be used the next time the method is called. Finally, the method returns a float array containing the computed differences as input to the neural network model. This data is crucial for predicting the behaviour of enemies in the game, which allows it to be dynamically adjusted in response to changes in the environment and player interactions. In this way, *CalculateInputData* plays a vital role in the intelligent management of enemy behaviour in computer games, by continuously collecting, processing, and providing up-to-date data for the AI model.

Model training process

The process of creating inputs to the model starts with generating data samples. To do this, the number of samples was set at 1000 and random values were generated for each sample. Each sample consists of three characteristics: *accuracy*, *enemiesDefeated*, and *damage taken*. The game downloads and updates data every ten seconds for illustrative purposes. Taking into account the possible reload time of the weapon and the placement of opponents – the player will be able to defeat a maximum of five opponents. As a result, the average number of shots hit on the target, representing the accuracy parameter, will be around 35, as an average of seven shots are enough to defeat an enemy (in the case of neutral difficulty). This information translates into the upper and lower limits of the input data values that will be fed into the network. The value ranges for these traits are 0 to 40 for accuracy, 0 to 10 for the number of enemies defeated, and 0 to 30 for damage taken. These values are generated using the function np. random.randint, which creates an array with dimensions [1000, 3]. The written code is shown bellow.

```
num_samples = 1000
data = np.random.randint([0, 0, 0], [41, 11, 31], size=(num_samples, 3))

def generate_targets(data):
    targets = []
    for sample in data:
        accuracy, enemiesDefeated, damageTaken = sample
        if accuracy > 25 or enemiesDefeated > 3 or damageTaken < 3:
            targets.append([7, -0.05, 10])
        else:
            targets.append([-10, 0.05, -20])
    return np.array(targets)

targets = generate_targets(data)
```

Then, based on this input, the corresponding targets are generated using the generate_targets function. This function processes each sample of inputs using simple rules: if the accuracy is greater than 25, the number of enemies defeated is greater than 3, or the damage taken is less than 3, then the score is set to [7, -0.05, 10]. Otherwise, the result is set to [-10, 0.05, -20]. In this way, each input sample is transformed into an output sample, resulting in an array of results with dimensions [1000, 3]. It is worth noting here that the neural network selects the most appropriate values with an accuracy of several decimal places, which translates into gameplay and the parameters will usually be updated as rational numbers.

The process of training a model (model.fit) is to fit the model to the input data and its corresponding results. The model is trained for 100 epochs (epochs=100), which means that the entire dataset passes through the neural network 100 times. During callback training, TensorBoard saves logs that can be analyzed later to understand how the model learns. After the workout, the model is saved to a file with the .keras extension. This allows the trained model to be loaded and used in the future without having to be retrained. The training result is shown in Fig. 4.

DOI: https://doi.org/10.53052/9788367652292.09

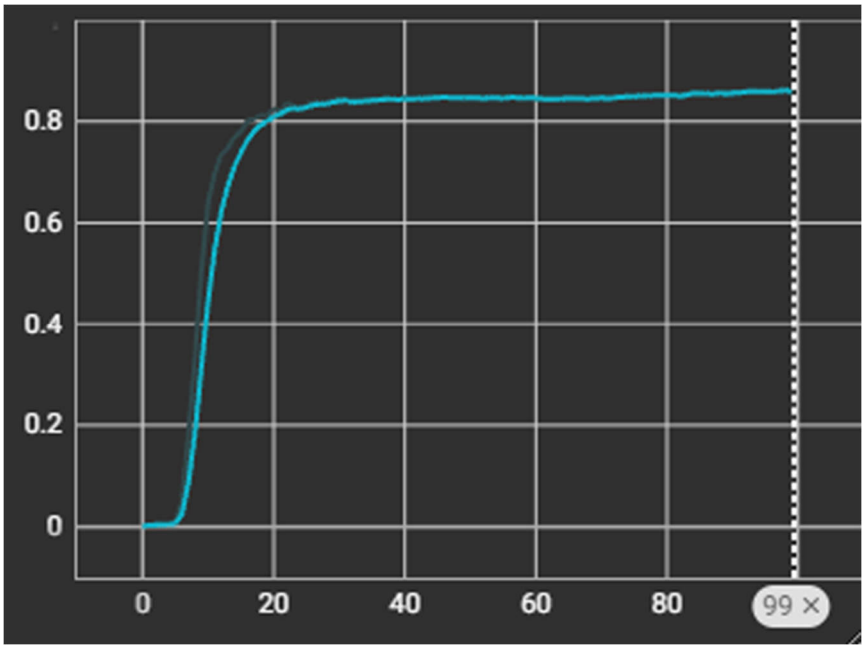

Figure 4. Training process in epochs to accuracy

In the first few epochs (up to about 20), the model quickly increases its accuracy. This is common for many models that quickly learn basic patterns from data. After about 20 epochs, the accuracy of the model stabilizes around 0.85, indicating that the model has reached the saturation point and further training does not bring significant improvements. The final accuracy of the model is at around 85%, suggesting that the model has learned the patterns in the training data well. So, based on the graph, we can conclude that the model quickly learned the patterns in the training data and achieved stable high accuracy after about 20 epochs – gradually increasing it after that time. The training lasted a total of 99 steps, and the final accuracy is about 85%.

The main functionality of the method is to update the parameters based on the output passed as the *outputData* parameter. For the player, the method checks and increases the damage value of the enemy, provided that the addition of output does not exceed a set range (modifiable). Due to the fact that the enemy's health is divided into maximum and current, both properties must be updated. So, for each enemy and their weapons, the method increases or decreases the maximum health (*MaxHealth*) and the current health value (*CurrentHealth*) based on the corresponding output, checking if the new values are within the set ranges. The speed of shots by enemies is modified one by one, implemented analogously to the previous cases. The upper and lower limits are set in such a way as not to exceed the acceptable limits and for the gameplay to be playable. For each parameter, the values will look like this: *Enemy Damage: Average Damage = 15, Minimum Damage = 7, Maximum Damage = 35, Health Level: Average Health = 100, Minimum Health = 70, Maximum Health = 130, Shooting Speed: Average Speed = 0.8, Minimum Speed = 0.5 (Hard Difficult), Maximum Speed = 1.1* (Easy Level) Additionally, when updating each parameter individually, The current value is assigned to global variables in order to continuously display the value on the player's screen. An interpolation is created for the text field assigned to the GameManager object in the project hierarchy. The entire process allows for dynamic adjustment of the game's difficulty and interactions between different elements of the world based on the results of the neural network model.

4. Experiments

The main goal of the analysis was to determine the level of satisfaction with the game mode played. It was also important to determine whether the DDA mechanism reacted appropriately in real time to the nature of the player's gameplay. The questions in the questionnaire are divided into four parts. Below are all the issues grouped using the order used in the survey. The survey included questions concerning: satisfaction with the game at each difficulty level (static and dynamic), assessment of the level of difficulty and its adequacy in the DDA mode, observation of the smoothness of the difficulty level adjustment in the DDA mode, and assessment of commitment and motivation to continue playing in the DDA mode. The first thing that can be noticed after comparing the results is the high consistency of answers between

the respondents. This results in easier interpretation of data and the possibility of effective conclusions. The results are tabulated and discussed below. In all lists, cells containing numeric values symbolize the number of participants in the experiment who selected a given option in the questionnaire.

Table 1. Satisfaction in playing in 5 scale score.

Participant	Easy Mode	Medium Mode	Hard Mode	DDA Mode
1	1	3	2	4
2	2	4	2	5
3	4	2	1	5
4	1	2	4	4
5	1	3	2	3
6	1	3	5	4
7	5	3	1	4
8	3	4	4	5
9	1	3	2	4
10	1	4	2	3

Observing the results from Table 1, concerning the level of satisfaction of players (where the rating scale is 1 – very dissatisfied, 5 – very satisfied), one can clearly notice a clear shift in the distribution of results. All the answers for easy mode were cantered around numbers indicating dissatisfaction. Taking into account the level of proficiency of the testers, where the vast majority are skilled players, the results suggest that this mode was too easy. However, the exception are two people whose skills were not as high as those of the rest – they voted for satisfied and very satisfied. For medium mode, the results are also mixed. The most ratings are 3 or 4, which indicates moderate satisfaction. However, the most mixed results are in the hard mode. For some people, this mode turned out to be too difficult, which may indicate that less experienced players are affected. Nevertheless, for others, it was a rewarding experience, pointing to the opposite conclusions. The DDA mode, which uses a neural network, achieved the greatest compatibility, while having the greatest satisfaction. Only two people rated this mode less than 4, indicating general acceptance and positive evaluation by the majority of participants.

None of the surveyed people answered affirmatively to the question whether the level of difficulty of the DDA did not adjust appropriately or rather did not adapt. On the other hand, two survey participants were not able to state unequivocally. Such a large predominance of votes for "rather yes" and "yes" indicate the generally positive experience of the participants with adjusting the level of difficulty by the DDA system. The majority of players, eight in total, felt that the system worked adequately, suggesting that DDA was effective in adjusting the difficulty level to match the skill of the players.

The difficulty adjustment liquidity assessment table shows that none of the participants considered the adjustment liquidity to be inadequate or rather inadequate. The lack of votes for "No" and "Rather not" suggests that the DDA mechanism worked correctly in terms of liquidity. Two people had difficulties in clearly assessing the fluency of the difficulty adjustment. This may indicate some doubts or a lack of clear feelings about the smoothness of the system in some cases. This is an area that may require further clarification or a more detailed discussion of how the system works. The rest of the respondents assessed the smoothness of the adjustment positively. These results show that the majority of participants, eight in total, found the ease of difficulty adjustment to be efficient. This indicates an overall positive experience of participants with this mode.

So, based on the above surveys, the DDA mode was generally well evaluated by the participants. The static levels were notable because participants expressed mixed but more negative feelings about these modes. The lack of extremely negative ratings indicates the effectiveness of the DDA system in avoiding problems related to playing too easily or too difficult. However, some areas require further attention, such as clarifying the operation of the DDA system

and further optimization to ensure unequivocally positive evaluations from all participants. In addition, a few comments were collected regarding the game mechanics, the size of enemies, or the arrangement of objects in the environment.

5. Results

This paper on the use of a neural network for Dynamic Difficulty Adjustment (DDA) in an FPS computer game indicates that this is a very promising concept that can significantly improve the quality and satisfaction of the game. The research and tests have confirmed that the implementation of adaptive difficulty control using a trained neural network is more effective compared to traditional, static difficulty levels.

Players who used DDA reported that the difficulty level was better suited to their skills, which resulted in a more satisfying experience. The results of the tests showed that participants playing in the DDA mode showed greater satisfaction with the game. Surveys completed after the game sessions indicated that dynamic adjustment of difficulty has a positive effect on commitment and motivation to continue playing. The neural network effectively analysed input parameters such as player accuracy, the number of enemies defeated, and the damage received, and dynamically adjusted the game's output parameters based on this. This led to smoother gameplay and a more personalized experience that better suited the player's skills. Traditional static difficulty levels, while widely used, are often unable to adequately adapt to the player's changing skills in real-time. As a result, players may feel either too overwhelmed or bored, which negatively affects their feelings. In DDA mode, the difficulty level is constantly adjusted based on the player's current performance data, resulting in a more balanced experience.

Another important aspect is the personalized approach that DDA enables. The neural network, by analysing the input parameters, is able to effectively modify the statistics of enemies, which means that each player receives a unique character of the game, with each pass. This, in turn, leads to greater immersion and meeting expectations from the title being played, because players – even when playing through it again, feel that the environment is actively responding to the user's different techniques and approaches. The research also indicates the huge development potential of this technology. A properly improved and developed DDA model can become a breakthrough solution in the computer games industry.

In the future, with the advancement of technology, further research and publicity of the topic, dynamic adjustment of the difficulty level may become a standard in computer game design. Improved neural network models and integration with advanced player data analysis techniques can allow for even more precise difficulty adjustments. DDA can contribute to the creation of more engaging, balanced, and rewarding games for a wide range of audiences. It is also worth noting the potential challenges and directions of future research. Although the results of this work are promising, there are many aspects that require further research and improvement. For example, choosing a different method of artificial intelligence, or more advanced neural network models that could take into account a wider range of input parameters, including the player's playing style, preferences, and even their emotional state. The analysis of the results additionally suggests that there is still a lot of room for improvement in terms of optimization in terms of model training or its configuration in the environment. Further research may focus on optimizing DDA algorithms for different game platforms and genres.

Acknowledgments

The work was fulfilled within the framework of Erasmus+ project "The transferable training model - the best choice for training IT business leaders" (project no. 2023-2-PL01-KA220-HED-000179445). Namely, the work contributes to the project results concerning studying use cases of AI and IoT good practices (work packages 2 and 3).

Reference

1. Kent, S. L. The Ultimate History of Video Games. Three Rivers Press, 2001
2. Zackariasson, P., Wilson, T. L. (Eds.). The Video Game Industry: Formation, Present State, and Future. 2012.
3. Salen, K., Zimmerman, E. Rules of Play: Game Design Fundamentals, 2004,
4. Juul, J. Half-Real: Video Games between Real Rules and Fictional Worlds, 2005,

5. Andrade, G., Ramalho, G., Santana, H., & Corruble, V. Dynamic Difficulty Adjustment in Games Using Evolutionary Algorithms, 2005,
6. Adams, E. Fundamentals of Game Design. New Riders, 2010
7. Russell, S. J., Norvig, P. Artificial Intelligence: A Modern Approach. Pearson, 2021
8. Millington, I., Funge, J. Artificial Intelligence for Games, 2009
9. Sweetser, P., Wyeth, P. Game Flow: A Model for Evaluating Player Enjoyment in Games, 2005,
10. Lomas, D., Patel, K., Forlizzi, J., Koedinger, K. R. Optimizing Challenge in an Educational Game Using Large-Scale Design Experiments. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2013,
11. Zohaib M. Dynamic Difficulty Adjustment (DDA) in Computer Games: A review, 2018,
12. S. Xue, M. Wu, J. Kolen, N. Aghdaie, and K. A. Zaman, „Dynamic Difficulty Adjustment for Maximized Engagementin Digital Games," in Proceedings of the the 26th International Conference, pp. 465–471, Perth, Australia, April 2017,
13. C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modelling player experience in Super Mario Bros," in Proceedings of the2009 IEEE Symposium on Computational Intelligence and Games(CIG), pp. 132–139, Milano, Italy, September 2009,
14. P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Online adaptation of game opponent AI with dynamic scripting, „International Journal of Intelligent Games & Simulation, vol. 3,no. 1, pp. 45–53, 2004,
15. R. Hunicke and V. Chapman, "AI for Dynamic Difficulty Adjustment in Games," in Proceedings of the Challenges in Game Artificial Intelligence AAAI Workshop, pp. 91 96, San Jose, Calif,USA, 2004,
16. C. H. Tan, K. C. Tan, and A. Tay, "Dynamic game difficulty scaling using adaptive behaviour-based AI," IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, no. 4, pp.289–301, 2011,
17. J. Yang, Y. Gao, S. He et al., "To Create Intelligent Adaptive Game Opponent by Using Monte-Carlo for Tree Search,"in Proceedings of the 2009 Fifth International Conference on Natural Computation, pp. 603–607, Tianjian, China, August 2009
18. Weka library https://en.wikipedia.org/wiki/Weka_(software) (access date 12.06.2024),
19. A. Ebrahimi and M.-R. Akbarzadeh-T, "Dynamic difficulty adjustment in games by using an interactive self-organizing architecture," in Proceedings of the 2014 Iranian Conference on Intelligent Systems, ICIS 2014, Iran, February 2014,
20. Anderson, E. F., & Engel, S. Dynamic Difficulty Adjustment in Video Games. Game Studies, 2011,
21. Dvir Ben Or, Kolomenkin M., Gil Shabat Deep Learning-Based Adaptive Dynamic Difficulty Adjustment with UX and Gameplay constraints. 2021,
22. Mnih, V., Kavukcuoglu, K., Silver, D., et al. Human-level control through deep reinforcement learning. Nature, 2015,
23. Silver, D., Schrittwieser, J., Simonyan, K., et al. Mastering the game of Go without human knowledge. Nature, 2017,
24. https://ascentoptics.com/blog/pl/the-power-of-server-clustering-how-it-works-and-key-benefits/ (access date, 14.06.2024.),
25. https://onnx.ai/about.html (access date 14.06.2024),
26. https://www.cyberskill.pl/gry-platformowe-w-scratch-poradnik-tworzenia-czesc-2 (access date, 14.06.2024),