*electronics*

*Article*

# Fuzzy Logic-Based Dynamic Difficulty Adjustment for Adaptive Game Environments

**Panagiotis D. Paraschos * and Dimitrios E. Koulouriotis**

School of Mechanical Engineering, National Technical University of Athens, 15772 Athens, Greece; dkoulouriotis@mail.ntua.gr
* Correspondence: pparasc@pme.duth.gr

**Abstract:** Game players frequently seek video games that offer great replayability and unpredictable challenges, aiming to minimize anxiety and maximize fun. One of the most suitable game genres for achieving these objectives is the "shoot 'em up" genre, although it is mainly focused on experienced players. Given the latter, challenges should be balanced and adapted to player characteristics and style in an effort to strengthen their engagement with the game. In the present study, a fuzzy logic-based dynamic difficulty adjustment approach is proposed for dynamically tailoring challenges according to players' behavior and strategies. The testbed of this approach is a shoot 'em up game where the players are faced with a challenging and adaptable opponent. The functionality of the proposed system is examined through game scenarios. The evaluation illustrates that fuzzy logic-based dynamic difficulty adjustment provides balanced challenges to players by easily defining the behavior of the game environment and opponents through a set of rules.

**Keywords:** adaptation; player modeling; non-playing character; named pipes

check for updates

Academic Editors: Dionisios Sotiropoulos and Konstantina Chrysafiadi

Received: 15 November 2024
Revised: 26 December 2024
Accepted: 30 December 2024
Published: 2 January 2025

**Citation:** Paraschos, P.D.; Koulouriotis, D.E. Fuzzy Logic-Based Dynamic Difficulty Adjustment for Adaptive Game Environments. *Electronics* **2025**, *14*, 146. https://doi.org/10.3390/electronics14010146

## 1. Introduction

An essential characteristic of a successful game is player engagement, since games aim to provide the best possible experience via challenging environments and opponents. A well-known problem frequently treated by game programmers is dynamic difficulty adjustment (DDA). DDA dictates that the programmers have to impose a strategy for adjusting difficulty levels "online" without any disruption to the player's experience [1]. Such a system evaluates the player's gaming profile and adjusts the game's features to create proper challenges, neither too hard nor too easy, aiming to reduce both anxiety and boredom. Furthermore, the DDA system should be employed in the background. To evaluate DDA systems, the genre of shoot 'em up games is suggested as a well-suited testbed [2]. In these games, the player controls a ship and shoots at various threats while attempting to avoid incoming shots and obstacles. However, these challenges are often difficult, and thus they are not appropriate for an inexperienced player.

To this end, fuzzy logic is frequently suggested due to its several advantages, as evidenced in academic research. Let us examine some of these advantages. For example, it facilitates nuanced game difficulty adjustment due to its ability to handle uncertainty in complex game environments [3]. A second advantage of fuzzy logic is related to its rule-based structure. That is, due to the linguistic nature of the rules, fuzzy logic ensures flexibility and interpretability for designers, enabling easy modifications to the DDA system [4]. Furthermore, fuzzy logic-based DDA systems are highly adaptable across different game scenarios and have decreased computational overhead, making them appropriate for real-time game difficulty adjustment applications [5]. Finally, due to its versatility, fuzzy logic can be combined with machine learning approaches, facilitating the creation of hybrid approaches [6]. This can further enhance adaptability and scalability in a variety of gaming environments, e.g., educational games. Given the above, fuzzy logic can be considered an effective tool for implementing DDA in games.

In this paper, the shoot ’em up game ShoottheFuzzyShip was built in the Unity game engine. Shoot ’em up games belong to the genre of shooter games. These games integrate simplified mechanics compared to other game types, e.g., action games. This characteristic renders them appropriate for demonstrating DDA systems, as player skill can be easily quantified through actions, including opponent destruction, or player score [7]. The nature of these games allows designers to experiment on DDA-based applications without taking into consideration complicated game mechanics. Furthermore, shoot ’em up games feature increasing challenges, such as enemy waves spawning over the course of a game session [8]. Given that, they are well suited for DDA systems, which endeavor to balance the level of challenge. Finally, shoot ’em up games allow researchers to define metrics that easily quantify the impact of game difficulty on player behavior and engagement [1]. Given the mentioned advantages, the shoot ’em up genre was chosen as a testbed for the proposed approach.

The proposed approach features a dynamic game environment and an adaptable opponent, which are controlled by a DDA system. This system utilizes the fuzzy logic method and is evaluated in the MATLAB R2024a environment. It was developed in an effort to provide answers to the following questions: (a) How does fuzzy logic-based DDA improve player engagement and retention in shoot ’em up games? (b) To what extent does the fuzzy logic DDA system adapt to different player skill levels in real time? (c) What impact do varying levels of player and game characteristics, e.g., player health and shield charges, have on the effectiveness of fuzzy logic-based DDA?

To this end, the goal was to design a game environment that is unpredictable and consists of an opponent whose behavior is tailored according to the skill level of the player. For example, if the player is experienced, the game environment becomes more hostile. It initiates the spawning of waves of asteroids, and the enemy becomes enraged and shoots with rockets. However, Unity restricts communication between MATLAB and ShoottheFuzzyShip. Consequently, MatUn was developed. This application evaluates the fuzzy inference system via a component object model application programming interface (COM API) and communicates with ShoottheFuzzyShip through a named pipe.

The main contributions of this study can be summarized as follows:

- The application of fuzzy logic for constructing an adaptive system that efficiently balances the level of game difficulty faced by the players. To achieve that, the system analyzes in real time the player’s skill and tailors opponent behavior and environmental hostility. This showcases the potential of fuzzy logic for generating unpredictable experiences through human-like reasoning within the context of a specific case study.
- The development of a middleware application, called MatUn, that allows communication between Unity and MATLAB via a COM API and named pipes. This addresses a technical limitation in integrating these platforms into decision making.
- The study of the impact of fuzzy logic-based DDA on players by creating a tailored experience for them. Given that, it could serve as a rule-based interpretable alternative for DDA with the potential of combining it with machine learning. That is, the proposed implementation will serve as the basis for the construction of more sophisticated systems that use intelligent learning mechanisms.

The rest of this paper follows this structure: Section 2 includes a literature review of recent studies in the fields of DDA and fuzzy systems for DDA. Section 3 presents the testbed game, describing gameplay mechanics and features. Section 4 provides the proposed approach description. In Section 5, the proposed approach is evaluated through nine different scenarios. This paper is concluded in Section 6.

## 2. Literature Review

The DDA problem has attracted considerable research attention throughout the years. In this regard, several attempts have been made to tailor game difficulty according to player characteristics, such as skill level [9], during gameplay. These approaches frequently rely on metrics, such as player score. For example, Weber and Notargiacomo [10] employed evolutionary algorithms to balance the game difficulty according to the players' score, aiming to maintain their engagement with the game. By focusing on simplistic metrics, their proposed approach might ignore significant aspects of the player experience, such as strategies. Similarly, Yang and Sun [11] explored the impact of game difficulty fine tuning on player retention in casual games. Their research revealed that DDA increases players' interest in replaying casual games, effectively keeping the players motivated. However, it is safe to assume that this improvement might be short-lived when the DDA system is unable to dynamically respond to player behavior. This could lead to a lack of long-term engagement.

Another prominent approach to DDA is the employment of affective computing devices for collecting physiological data from players, such as heart rate [12], or brain activity [13]. While these devices can offer a more personalized player experience, they also introduce several challenges. For example, in the context of applying DDA to metaverse multiplayer games, Rezapour et al. [14] assessed players' emotional state and skill level by analyzing chat messages using natural language processing models. However, the accuracy and reliability of the emotional assessments based on chat messages are questionable, as this approach was used for the first time in the study. Moreover, as demonstrated in several studies [15–17], the reliance on technologies, such as headsets and eye tracking, excludes players who do not have access to this equipment.

In contrast to the above, fuzzy logic-based systems offer a more accessible solution for adapting game difficulty according to player characteristics, such as skill and performance [5]. For example, Chrysafiadi et al. [3] developed a fuzzy logic-based DDA system for adjusting the level of challenge in an educational game, achieving a near-to-optimal balance between challenge and learning. The study of Soylucicek et al. [4] implemented a fuzzy rule-based system in the enemies of a modified version of Meteor Escape. This system evaluates the x and y distance between the enemy's and player's spaceships to determine in which direction the enemy will fire. Pratama et al. [18] proposed a fuzzy controller as a DDA approach for the MOBA game Defense of the Ancients 2. Their system selects a difficulty level based on players' past game records. Pirovano and Lanzi [19] developed a scripting game named Fuzzy Tactics to demonstrate the use of fuzzy logic as a game mechanic. In this approach, the player issues orders to his troops in the Order Creation System, which creates the fuzzy rules before the gameplay.

Given the above, the present paper attempts to address limitations mentioned in the state-of-the-art research. That is, it implements a fuzzy logic system into a real-time game environment without the employment of a specific player assessment device, such as a biofeedback one [13,17]. This contrasts the related research that integrated fuzzy logic in predefined or fixed game scenarios. To achieve dynamic adaptation, a middleware software, called MatUn 1.0, was developed to address the challenge of bridging Unity with MATLAB's fuzzy inference system. This implementation is not extensively explored in the relevant research. Moreover, the present work extends the work presented in [3,18,19], which merely employed player score metrics, such as score. In this regard, the proposed approach incorporates a wide range of input variables, including player health and shield charges, into the decision-making process. The aim is to provide a game environment that promptly responds and adapts to the player's style and behavior. Such an approach will provide insights regarding the impact of real-time fuzzy logic-based DDA on player engagement and retention.

## 3. The Testbed Game

The testbed of this work is the shoot 'em up game ShoottheFuzzyShip, which was developed in Unity (https://unity3d.com, accessed on 20 October 2024) using the programming language C#. In ShoottheFuzzyShip, the player controls a spaceship with the objective of destroying the enemy's spaceship while avoiding the occasional waves of asteroids. However, this is not an easy task. The game uses a fuzzy inference system as a dynamic difficulty adjustment mechanism, which dynamically scales the difficulty and tailors the enemy's behavior to the player's skill level and style. The game ends when either spaceship is defeated.

As mentioned, ShoottheFuzzyShip is a modified version of Space Shooter (https://unity3d.com/es/learn/tutorials/s/space-shooter-tutorial, accessed on 20 October 2024), which was developed by Unity Technologies as a tutorial. The assets and source code of Space Shooter are provided for free. For the paper's purposes, Space Shooter was significantly modified, with new features, such as health and new weaponry, being implemented.

### *3.1. Differences with Space Shooter*

In Space Shooter, the player controls a spaceship and encounters waves of enemy spaceships and asteroids that spawn randomly across the game environment. The player's objective is to score points by shooting either of the spawning threats. For example, if the player destroys an asteroid, the score levels up by one point. The score is displayed at the top of the screen. However, the movement and behavior of the enemy spaceships are random, and they move toward the player's spaceship in every frame. As they are approaching, the enemy spaceships can either be elusive, or they can shoot at the players. Therefore, the player should carefully navigate their spaceship, as it is vulnerable to both incoming enemy shots and asteroids. The game ends when the player collides with a game object or receives an incoming shot.

In ShoottheFuzzyShip, the player's objective is to defeat the enemy's spaceship. The player's spaceship has shields, limited ammo, and health points (HP). The ammo and the shields are replenished with packs, which can be spawned sporadically at various points in the game environment. The player loses HP when their spaceship either receives an incoming shot or collides with an asteroid. The enemy has the same abilities as the player but with different attributes. It shoots at the player with a variety of weapons and, occasionally, becomes enraged. Its position is fixed at the top of the screen. Additionally, its ammo is unlimited, and it can use unlimited shields with a cooldown period that varies at each difficulty level. The enemy loses health when it receives an incoming shot from the player, while the game difficulty determines the spawn rate of the asteroids. During gameplay, the enemy moves back and forth along the horizontal axis. Thus, it never collides with the player's spaceship.

### *3.2. Gameplay Mechanics and Features*

The Heads-Up Display (HUD) overlays the screen. On the top, it displays the health bar of the enemy. It is the only known information about the enemy. In the case of the player, the HUD displays detailed information about their attributes. The health bar and the number of shields of the player are on the bottom left. In this sense, the player knows how many health points their spaceship has and how many shields they can use to protect themselves. Finally, on the bottom right, the HUD displays the ammo of the player at the current moment.

The control of the player's spaceship is simple. The player moves their spaceship, horizontally and vertically, in the game environment using the keys W, A, S, and D on the keyboard. Using the left and right mouse buttons, they shoot and enable shields, accordingly. The enemy, on the other hand, is a non-player character and moves automatically back and forth horizontally. The behavior of the enemy is scripted and dynamically adjusted during difficulty scaling. For example, if the level is set to Very Hard, the enemy does not shoot, but it uses shields.

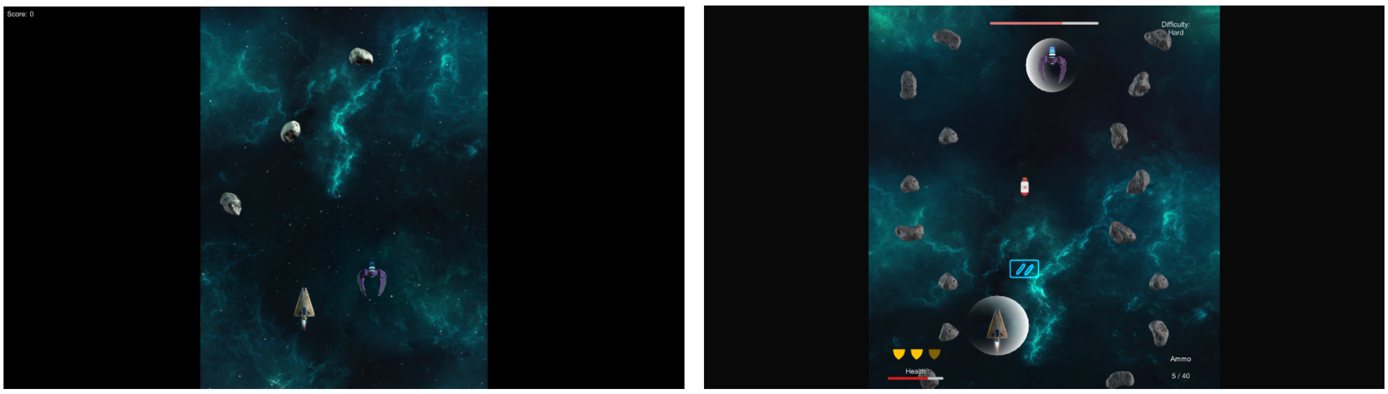
At the start of the game session, both spaceships have 100 HP. To avoid taking any damage, the player has three available shields. In contrast, the enemy can use unlimited shields to protect itself from incoming shots of the player. However, the use of the enemy's shield is restricted by a cooldown period. This cooldown varies at each level, leaving the enemy temporarily vulnerable.

Another feature of the game is the variety of available weapons. There are four weapon types: (a) yellow bolts, (b) green bolts, (c) bombs and (d) rockets. While the player can use only yellow bolts, the enemy can shoot either with green bolts, bombs, or rockets. Therefore, the enemy has an implemented weapon selection system. Furthermore, each weapon deals a different amount of damage. For example, the player loses 1.5 HP when their spaceship receives a rocket. Regarding the number of shots, the player can shoot only 40 yellow bolts, while the enemy has unlimited ammo. When the player has five shots left, the ammo packs start to spawn in the game environment. Hence, the player replenishes both ammo and shields.

Additionally, ShoottheFuzzyShip consists of five difficulty levels. Each level differs in the tactics of the enemy and changes elements of the game environment, such as the spawning waves of asteroids. Consequently, the five difficulty levels are as follows:

- Very Easy: The enemy does not shoot at the player and protects itself with shields. The shields have a cooldown period of 5 s. Moreover, waves of asteroids are spawning in the game environment.
- Easy: The enemy shoots at the player and is vulnerable to incoming shots. Likewise, waves of asteroids are spawning as in the previous level.
- Normal: As in the previous level, the enemy shoots again at the player's spaceship and protects itself with shields, which have a cooldown period of 3 s. Unlike the previous levels, there are no spawning waves of asteroids in the game environment.
- Hard: The enemy has the same properties and behaves the same way as in the normal level. Furthermore, the player additionally faces the spawning waves of asteroids.
- Very Hard: It is the most difficult level of the game. The enemy shoots and uses shields that have a cooldown period of 3 s. In addition, asteroids are spawning in the game environment.

Figure 1b shows a screenshot of ShoottheFuzzyShip. In this example of gameplay, the player (bottom) protects their spaceship from the incoming bombs of the enemy (top) while waiting for ammo packs to replenish ammo and shields. The difficulty is set to Hard, and waves of asteroids are spawning in the environment.



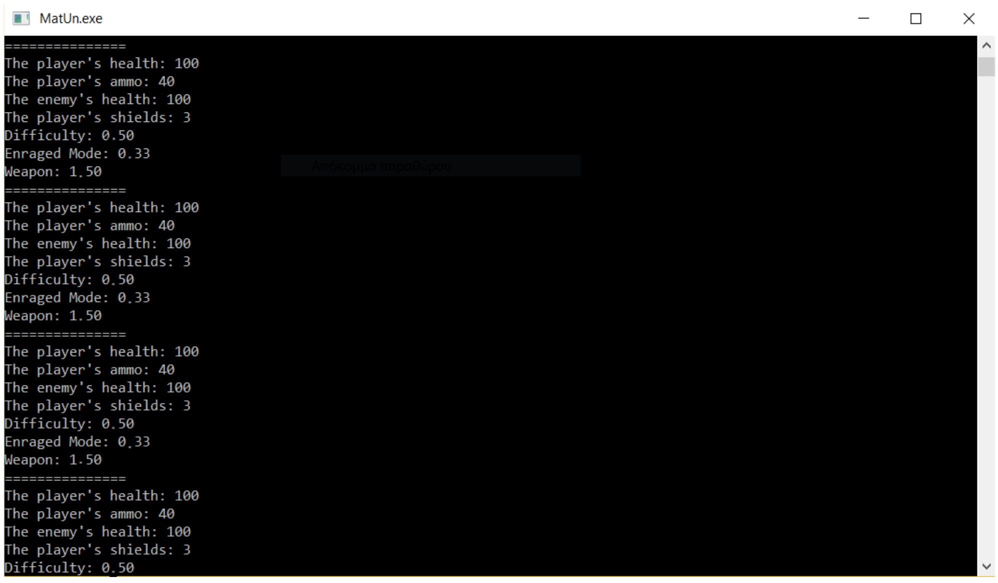
(**a**) (**b**)

**Figure 1.** An example of gameplay in (**a**) Space Shooter and (**b**) ShoottheFuzzyShip.

## 4. The Proposed Approach

### *4.1. MatUn*

According to [20], a major drawback of the Unity game engine is the lack of external library support. Consequently, ShoottheFuzzyShip cannot connect directly with the fuzzy inference system. To this end, the command-line application called MatUn was developed to allow communication between ShootheFuzzyShip and the fuzzy inference system. It is written in C# and runs only on Windows Command Prompt. Figure 2 shows MatUn in action. In this figure, it displays the crisp values of the fuzzy input and output variables of the fuzzy inference system.



```
===============
The player's health: 100
The player's ammo: 40
The enemy's health: 100
The player's shields: 3
Difficulty: 0.50
Enraged Mode: 0.33
Weapon: 1.50
===============
The player's health: 100
The player's ammo: 40
The enemy's health: 100
The player's shields: 3
Difficulty: 0.50
Enraged Mode: 0.33
Weapon: 1.50
===============
The player's health: 100
The player's ammo: 40
The enemy's health: 100
The player's shields: 3
Difficulty: 0.50
Enraged Mode: 0.33
Weapon: 1.50
===============
The player's health: 100
The player's ammo: 40
The enemy's health: 100
The player's shields: 3
Difficulty: 0.50
```

**Figure 2.** A screenshot of MatUn.

MatUn controls objects of MATLAB using MATLAB COM API (https://www.mathworks.com/help/matlab/call-matlab-com-automation-server.html, accessed on 20 October 2024). It creates a MATLAB instance by creating a MLApp.MLApp object. This instance runs the MATLAB Command Window. The pseudocode of the middleware software is provided in Algorithm A1 included in Appendix A. To evaluate the fuzzy inference system, MatUn calls the function Feval to evaluate the MATLAB function fuzzyFunc in the automation server.

The function fuzzyfunc is hardcoded in a script file [21]. The pseudocode of the function is given in Algorithm A2 found in Appendix A. This function is defined by four input variables and three output variables. These variables define the fuzzy inputs and output variables of the fuzzy inference system. Finally, after the evaluation, MatUn sends the resulting crisp values to the ShootheFuzzyShip.

## *4.2. The Named Pipe*

MatUn exchanges byte data with ShoottheFuzzyShip using a Windows named pipe [22]. Thus, it was selected for supporting full duplex communication.

ShoottheFuzzyShip acts as a pipe server. It stores the following values in a double array of four elements: (a) the player's health, (b) the player's ammo, (c) the enemy's health, and (d) the number of shields. Afterward, the elements of the array convert to string and send them to MatUn in byte form. Furthermore, ShootheFuzzyShip receives four string values from MatUn. It stores them in a double array of three elements. Using the stored values, ShootheFuzzyShip adjusts the difficulty level, enables or disables the enraged mode, and selects a weapon for the enemy. The pipe server refreshes every 200 ms. If the client disconnects from the server, ShoottheFuzzyShip closes.

In contrast, MatUn acts as a pipe client, which runs after the MATLAB Command Window is executed. It receives four string values from ShoottheFuzzyShip and stores them in a double array of four elements. Afterward, MatUn inputs these four elements in the MATLAB function fuzzyFunc by using the automation method. Following the evaluation, the output values are stored in a string array of three elements. MatUn sends these elements to the pipe server of ShoottheFuzzyShip. The pipe client of MatUn refreshes every 200 ms along with the pipe server. When the game ends, MatUn and the MATLAB Command Window close. In addition, MatUn displays an appropriate message to the user when it disconnects from the pipe server.

## *4.3. The Fuzzy Inference System*

The fuzzy logic [23] inference system was designed using MATLAB Fuzzy Logic Toolbox (https://www.mathworks.com/products/fuzzy-logic.html, accessed on 20 October 2024). This system uses a Mandani-type [24] inference engine and centroid as the defuzzication method. It is defined by four fuzzy input variables. These variables are related to the health points, the ammo, the number of shields of the player, and the health points of the enemy. They are mapped by three fuzzy sets. Figure 3a–d show the membership functions that define the fuzzy input variables, as thoroughly described in Table 1.

**Table 1.** Fuzzy input variables.

| Fuzzy Input Variables | Definition | Range | Membership Functions |
|---|---|---|---|
| playerHealth | The current HP of the player | [0, 100] | 3 trapezoid functions (Low, Normal, High) |
| playerAmmo | The current ammo of the player | [0, 40] | 3 trapezoid functions (Low, Medium, High) |
| enemyHealth | The current HP of the enemy | [0, 100] | 3 trapezoid functions (Low, Normal, High) |
| shieldCharges | The current number of shields of the player | [0, 3] | 3 triangular functions (Low, Medium, High) |

According to the fuzzy input values, the fuzzy logic inference system scales the difficulty level and decides which weapon the enemy will use and whether it will become enraged at the given moment. They represent the three fuzzy output variables of the system. The fuzzy sets of these variables are two, three, or five, as shown in Figure 4a–c. The fuzzy output variables are described in the Table 2.

**Table 2.** Fuzzy output variables.

| Fuzzy Output Variables | Definition | Range | Membership Functions |
|---|---|---|---|
| difficulty | The selected difficulty level | [0, 1] | 5 triangular functions (VeryEasy, Easy, Normal, Hard, VeryHard) |
| enragedMode | The enraged behavior of the enemy | [0, 1] | 2 triangular functions (Off, On) |
| weaponSelector | The selected weapon of the enemy | [0, 3] | 3 triangular functions (Bolts, Bombs, Rockets) |
| shieldCharges | The current number of shields of the player | [0, 3] | 3 triangular functions (Low, Medium, High) |

The fuzzy sets of the input and output variables are defined by trapezoid or triangular-shaped membership functions. These sets are symmetrical and overlap with each other to achieve a smooth transition between them.

To create a dynamic game environment and an adaptable behavior for the enemy, 81 rules were implemented in the fuzzy inference system. These IF–THEN rules are designed in such a way to provide the right difficulty to the player. They combine the fuzzy input variables with the logical operator AND. As an example, the following rule is:

If (playerHealth is Normal) and (playerAmmo is Normal) and (enemyHealth is Low) and (shieldCharges is Medium) then (difficulty is VeryHard) (enrangedMode is On) (weaponSelector is Rockets).

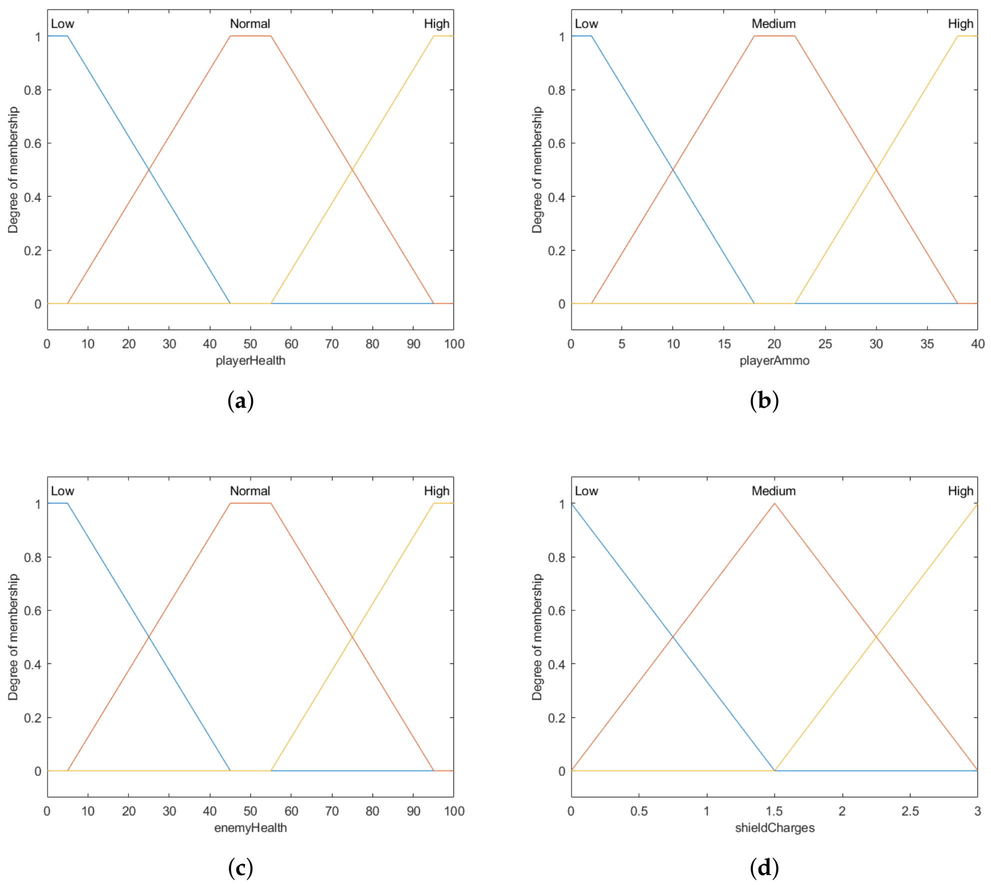


**Figure 3.** The fuzzy input variables of the fuzzy inference system. (**a**) playerHealth, (**b**) playerAmmo, (**c**) enemyHealth, (**d**) shieldCharges.

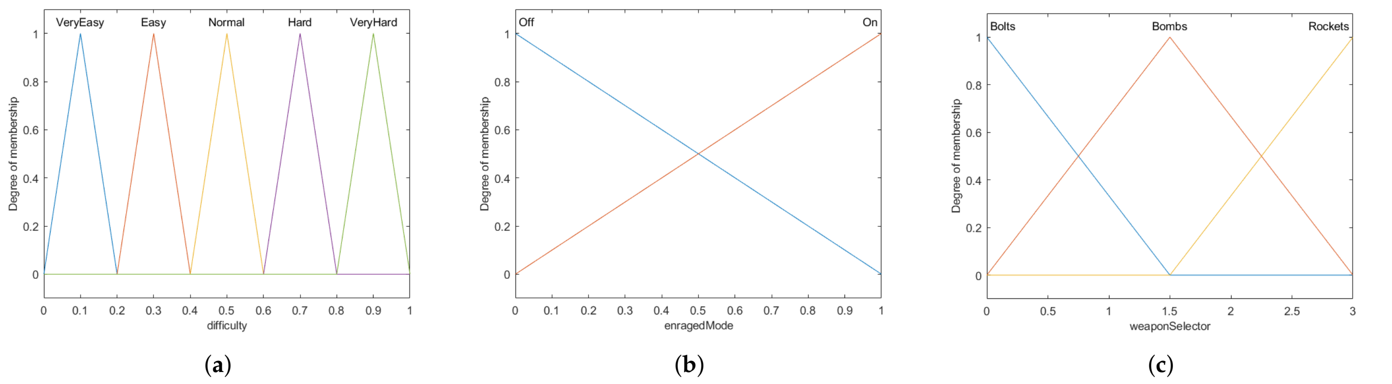


**Figure 4.** The fuzzy output variables of the fuzzy inference system. (**a**) difficulty, (**b**) enrangedMode, (**c**) weaponSelector.

### 4.4. Implementation of the Approach

Figure 5 illustrates the proposed approach and the decision-making process. According to the figure, there is a real-time interaction between the game and MATLAB's fuzzy logic inference engine achieved through MatUn. In this regard, game data, such as player health, are transferred to MATLAB via a communication pipe. These data are then processed by the fuzzy logic inference engine, where the decision making is performed. After the process, the engine's output is returned to the game. According to the fuzzy logic output, the game difficulty level, and the opponent behavior are accordingly adjusted. When the game state is updated, it is sent to MATLAB to perform decision making. This process is iterative and terminates upon the completion of the game session. Therefore, the feedback loop ensures that the game environment is responsive to the players' behavior, improving their experience.

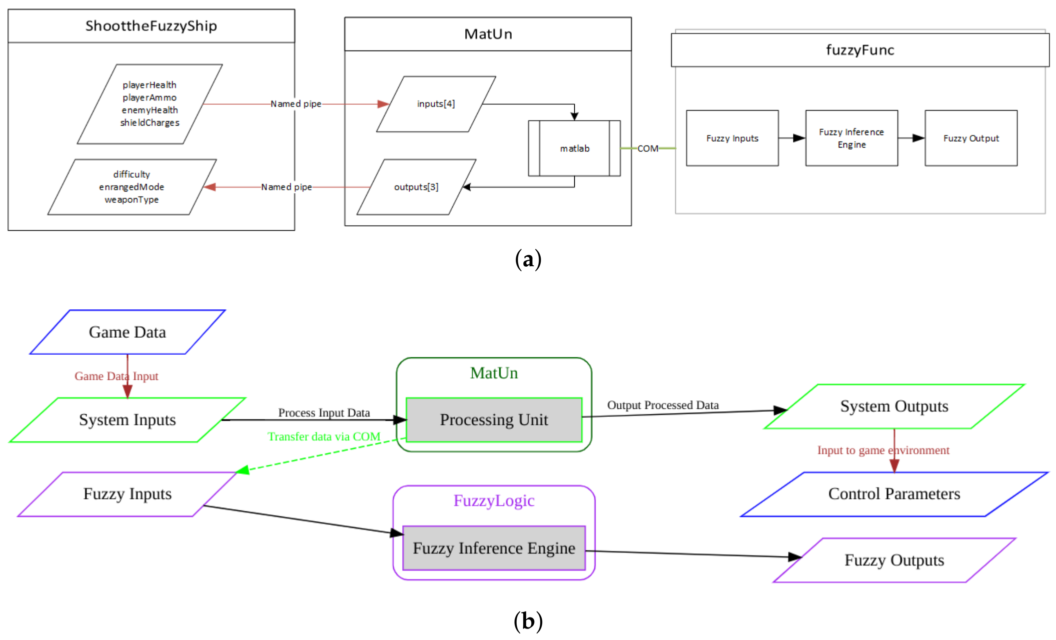


**Figure 5.** The connection of ShoottheFuzzyShip with MatUn. (**a**) Architecture of the proposed approach, (**b**) flow of information.

## 5. Experimental Analysis

The fuzzy logic-based system is implemented to dynamically tailor the behavior of the game environment and the opponent in ShoottheFuzzyShip. For this implementation, Unity communicates in real time with MATLAB's fuzzy logic inference engine to adapt the game according to the player's strategies and behavior. This communication is facilitated by MatUn and is terminated when the player's game session is completed. Given the processing requirement of the application, the fuzzy logic-based system, alongside the game and MatUn, was run on a PC with an i7 CPU processor and GTX 1060 GPU 6GB.

The aim of this application is to explore the performance of an adaptive game and how the responsive game environment affects the player experience during gameplay. To this end, nine scenarios were designed to evaluate the performance of the proposed approach. The scenarios are summarized in Table 3. They are designed according to four in-game variables: player health, opponent health, player ammo, and available shields. These variables are used as inputs for MATLAB's fuzzy logic inference engine.

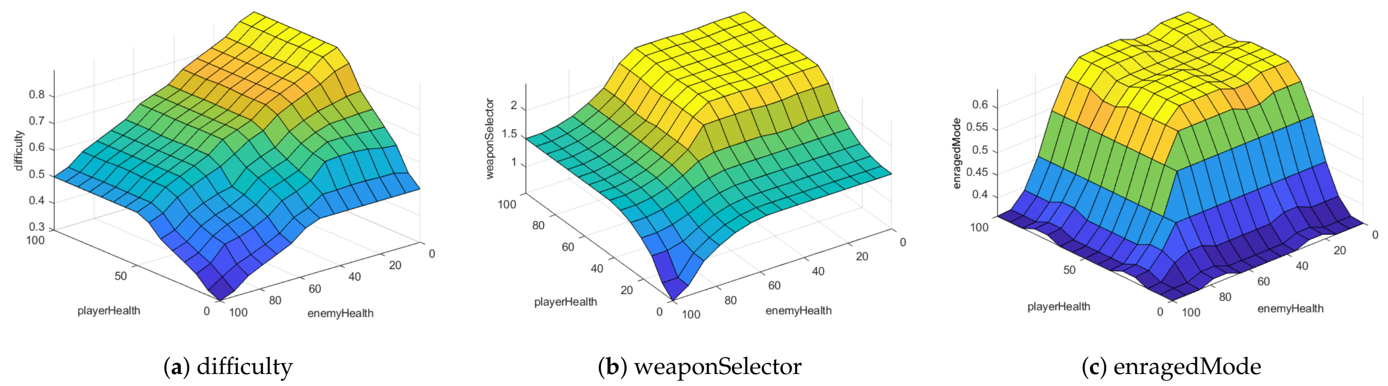
**Table 3.** Game scenarios.

| | **Fuzzy Input Variables** | | | |
|---|---|---|---|---|
| **Scenario** | **playerHealth** | **playerAmmo** | **enemyHealth** | **shieldCharges** |
| 1 | - | 20 | - | 2 |
| 2 | 50 | - | 50 | - |
| 3 | - | - | 70 | 1 |
| 4 | - | 30 | 30 | - |
| 5 | 15 | - | - | 3 |
| 6 | 70 | 10 | - | - |
| 7 | 10 | - | 10 | - |
| 8 | - | 10 | 60 | - |
| 9 | 20 | 10 | - | - |

According to Table 3, the values of each game parameter are specified to minimize variation and create a controlled environment for the player. In this regard, the experimental scenarios ensure repeatability through predefined game conditions that affect gameplay. This standardization facilitates comparison across the experiments as the player faces a variety of challenges. Despite the interaction within a controlled environment, the individual style and behavior of the player still affect the adaptation of the environment and opponent. This is on par with real-world conditions, where the players have to adapt their strategies according to the dynamic of the gameplay and the challenges they face. In this regard, player decision making plays a crucial role in gameplay and is considered unpredictable. Despite the unpredictability, the predefined game scenarios ensure that the proposed approach is effectively evaluated in terms of player behavior.

### *5.1. Scenario 1*

This scenario shows (Figure 6) how the dynamic game environment can be competing and challenging for the player. The values of ammo and shields are 20 and 2, respectively. Initially, the game difficulty is set to Easy. The enemy uses bolts as a weapon and does not become enraged, as the player is at a disadvantage. When the player lowers the health of the enemy, the DDA system increases steadily the game's difficulty to challenge the player. Additionally, the enemy is occasionally enraged and chooses bombs and, mainly, rockets for the enemy.

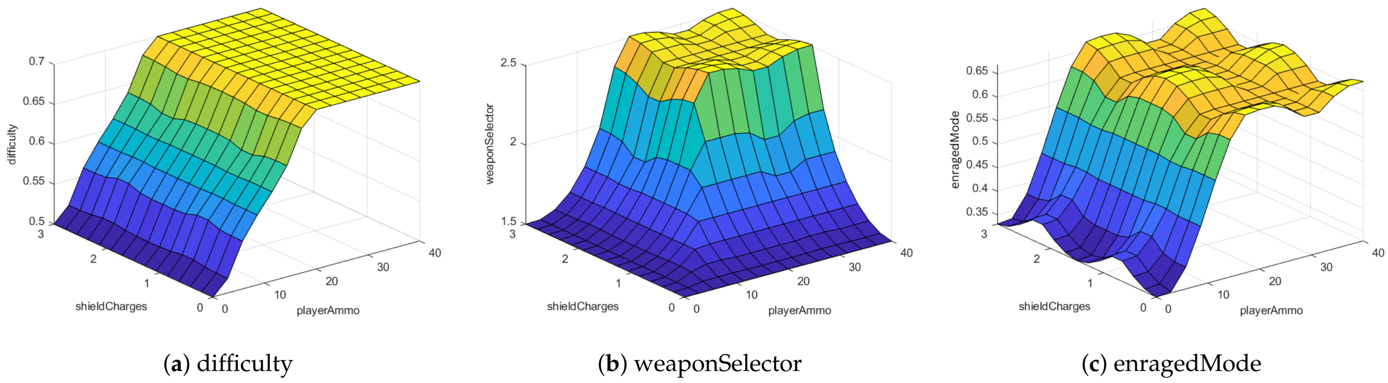


(**a**) difficulty (**b**) weaponSelector (**c**) enragedMode

**Figure 6.** The output surfaces of scenario 1 comparing playerHealth versus enemyHealth in terms of fuzzy output variables.

## 5.2. Scenario 2

The second scenario examines how winning the game is not an easy task for the player (Figure 7). The values of the enemy health and the player health are, respectively, 50. The game difficulty is set to Hard. The enemy shoots at the player with rockets, and it is in enraged mode. In this case, the player is in a difficult position, as the enemy shows aggressive behavior. As the ammo of the player decreases, the DDA system only tends to scale down the game difficulty. Furthermore, the enraged mode is still on, and the enemy still uses rockets.



(**a**) difficulty (**b**) weaponSelector (**c**) enragedMode

**Figure 7.** The output surfaces of scenario 2 comparing shieldCharges versus playerAmmo in terms of fuzzy output variables.

## 5.3. Scenario 3

The third scenario examines how the number of shields and the player's health can affect the gameplay (Figure 8). The enemy's health is 70 and the player has only one shield to use. Initially, the player has the advantage over the enemy, as the enemy's health is lower than in the previous scenarios. Thus, the game difficulty scales to Normal, the enemy is not enraged, and it shoots bombs. If the enemy's health is decreased, the DDA system realizes that the player is at an advantage. In this case, it scales up the difficulty; the enemy is enraged and shoots rockets. When the player has low health and ammo, the game difficulty scales down, the enemy shoots bolts, and it is not enraged.

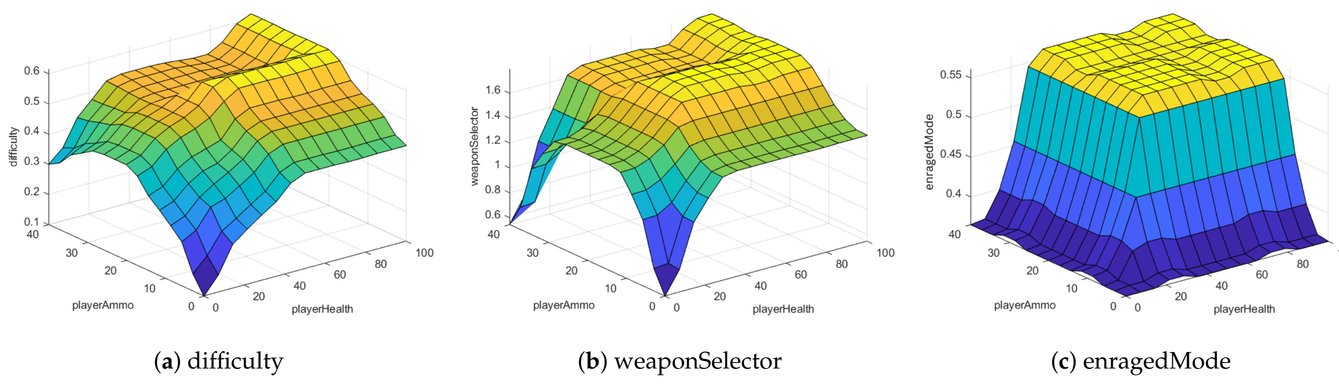


**Figure 8.** The output surfaces of scenario 3 comparing playerAmmo versus playerHealth in terms of fuzzy output variables.

### *5.4. Scenario 4*

In the fourth scenario (Figure 9), the player's ammo is 30 and the enemy's health is 30. In this scenario, the player is again at an advantage, as the ammo is high, and the enemy's health is low. Accordingly, the game difficulty is set to Hard; the enemy is enraged and shoots rockets. As the player avoids any damage, the DDA system attempts to lower the player's health. Thus, the game difficulty is still set to Hard; the enemy is still enraged and shoots rockets. Nonetheless, the player manages to win the game.

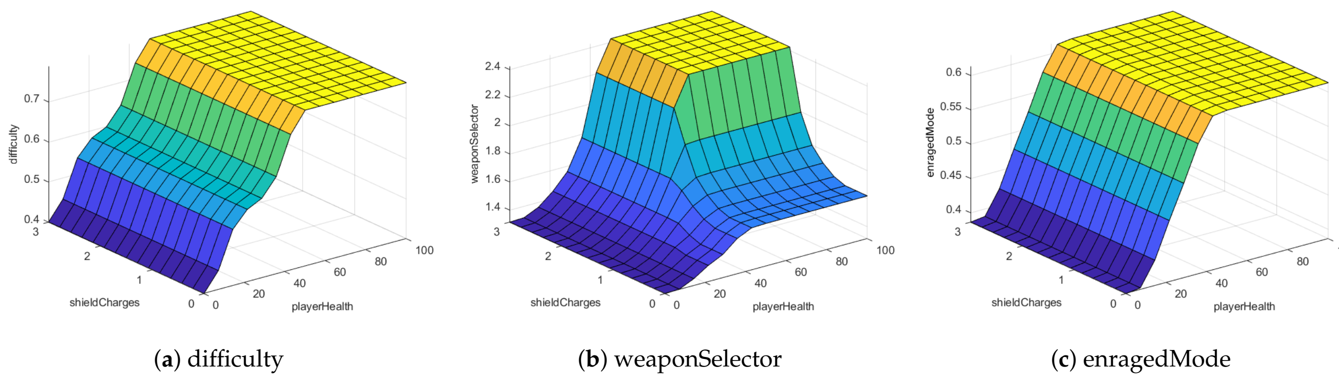
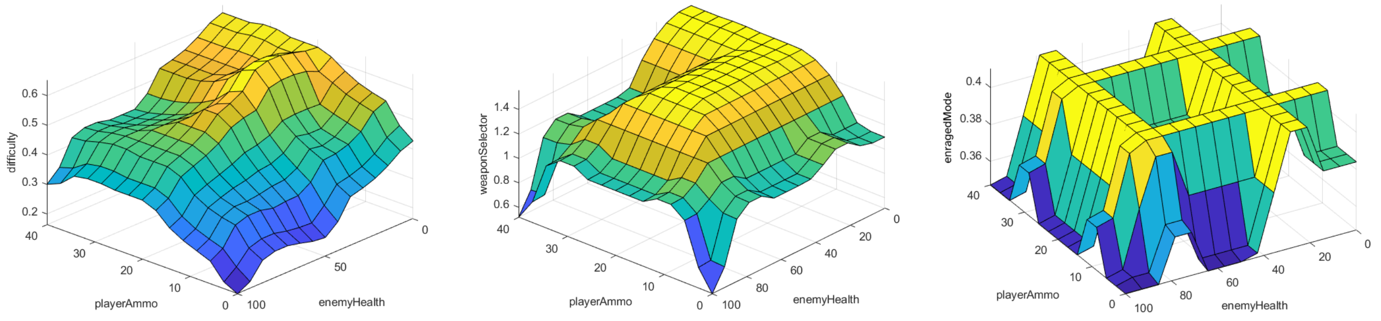


**Figure 9.** The output surfaces of scenario 4 comparing shieldCharges versus playerHealth in terms of fuzzy output variables.

### *5.5. Scenario 5*

The fifth scenario examines how the player can win the game even with low health (Figure 10). The player has 15 health points and three shields. Because the player's health is low, the DDA system favors the player. Thus, it sets the game difficulty to Easy and rarely to Normal. The enemy shoots either bolts or bombs, depending on the values of the ammo and the enemy's health. However, if the player manages to avoid any damage and uses ammo and shields in moderation, they can still win the game.

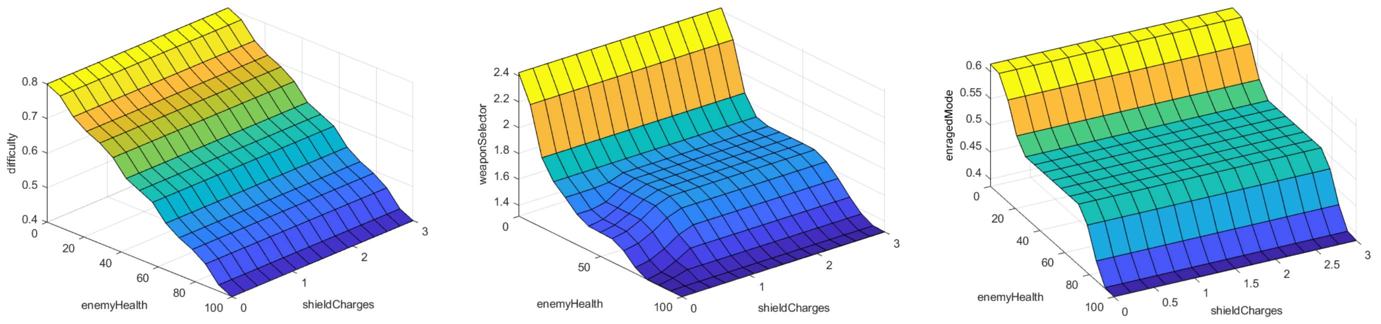


(**a**) difficulty (**b**) weaponSelector (**c**) enragedMode

**Figure 10.** The output surfaces of scenario 5 comparing playerAmmo versus enemyHealth in terms of fuzzy output variables.

## 5.6. Scenario 6

In the sixth scenario, the player's health is 70 and the ammo is 10 (Figure 11). With these values, the game difficulty is set to Easy; the enemy is not enraged and shoots bombs. While the ammo is low, the player easily overcomes the initial disadvantage by using shields to protect themselves. Furthermore, when the enemy's health is decreasing, the DDA system decides to create a hostile game environment; i.e., the enemy becomes enraged and shoots rockets.

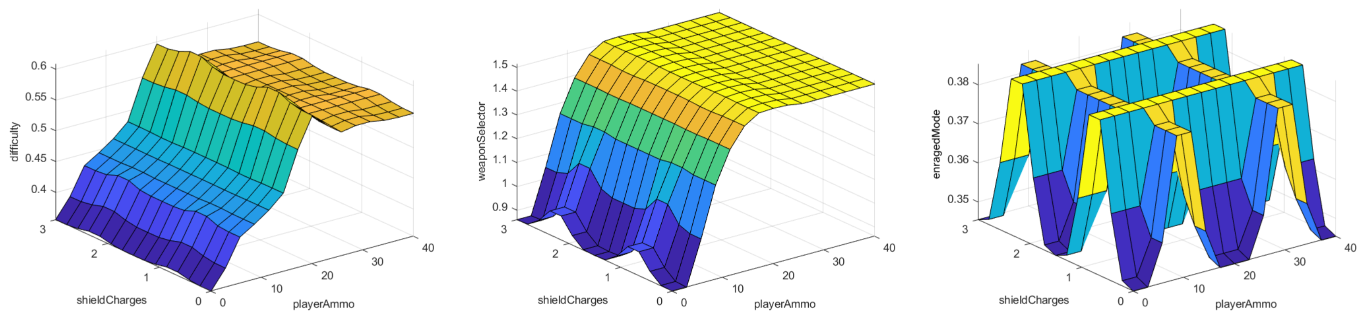


(**a**) difficulty (**b**) weaponSelector (**c**) enragedMode

**Figure 11.** The output surfaces of scenario 6 comparing enemyHealth versus shieldCharges in terms of fuzzy output variables.

## 5.7. Scenario 7

In this scenario, the player's health is 10 and the enemy's health is 10 (Figure 12). As neither of the two spaceships is in favor, the game's difficulty is set to Normal; the enemy is not enraged and shoots bombs. The game, therefore, is in equilibrium, as the initial state of the game remains the same. The player wins the game effortlessly.

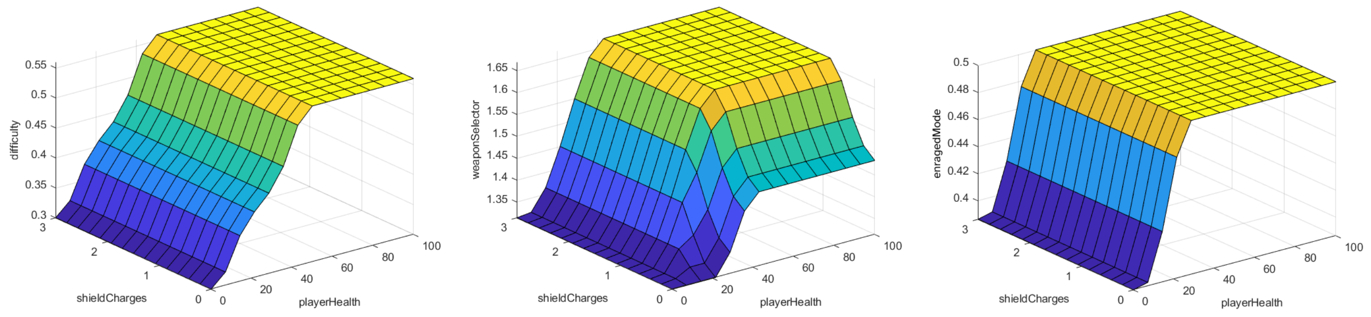


(**a**) difficulty (**b**) weaponSelector (**c**) enragedMode

**Figure 12.** The output surfaces of scenario 7 comparing shieldCharges versus playerAmmo in terms of fuzzy output variables.

### *5.8. Scenario 8*

The eighth scenario examines the game state, wherein the enemy's health is 60 and the player's ammo is 10 (Figure 13). The DDA system realizes that the player cannot become an immediate threat to the enemy, as they have low ammo. It decides to scale the game difficulty to Normal, and the enemy is not enraged and shoots bombs. As soon as the player replenishes ammo with an ammo pack, the game environment becomes more challenging and threatening. In this case, the difficulty scales to Hard; the enemy enrages and shoots either rockets or bombs. Therefore, the player is in a difficult position.

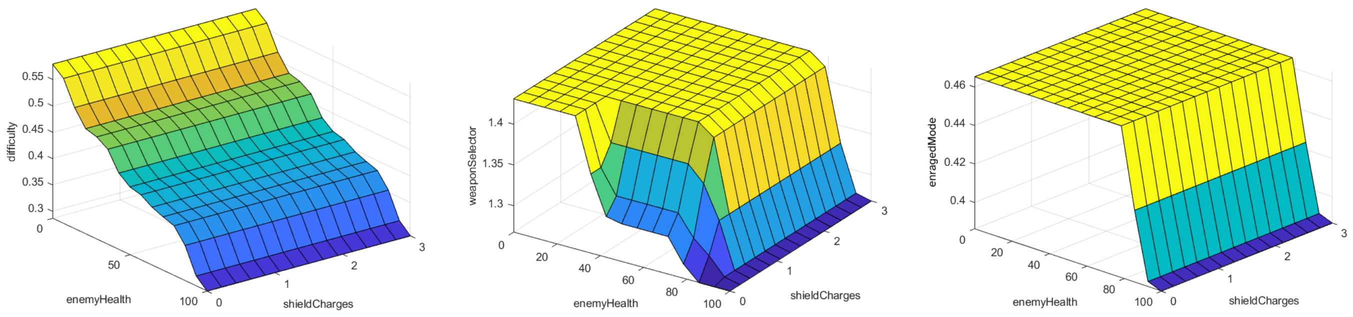


(**a**) difficulty (**b**) weaponSelector (**c**) enragedMode

**Figure 13.** The output surfaces of scenario 8 comparing shieldCharges versus playerHealth in terms of fuzzy output variables.

### *5.9. Scenario 9*

In this scenario, the player's health is 20 and the player's ammo is 10 (Figure 14). The player is very vulnerable, as their health is decreasing. As mentioned, the DDA system is designed in such a way to help the player. Therefore, during gameplay, the game difficulty ranges from Very Easy to Easy, the enemy shoots either bombs or bolts, and it is not enraged. In this case, the player has a chance to threaten the enemy by replenishing ammo and enabling shields. However, it is impossible to win the game, no matter how experienced they are.



(**a**) difficulty (**b**) weaponSelector (**c**) enragedMode

**Figure 14.** The output surfaces of scenario 9 comparing enemyHealth versus shieldCharges in terms of fuzzy output variables.

### *5.10. Discussion*

The analysis of the experiments in Section 5 highlights the strengths and weaknesses of the proposed fuzzy logic-based DDA system in terms of game adaptation. More specifically, the proposed system presents several strengths, including the dynamic adaptation of the game environment and opponent in response to the player's style and behavior. The adjustments are performed by the fuzzy logic in a subtle way without hindering the player's engagement with the game. Through the nine scenarios, the system exhibited high adaptability and responsiveness in extreme and normal game conditions. In this regard, it provided balanced challenges to the player, improving their experience and maintaining their retention. Finally, due to the nature of fuzzy logic, the DDA system could be easily applied and tested within the context of game environments, which may be more complex than shoot 'em up ones. This is achieved by refining aspects of the systems, including rulesets and variables.

On the other hand, the proposed approach presents some weaknesses regarding its capability of applying appropriate adjustments to the game environment. To highlight them, let us examine the behavior of the DDA mechanism under the implemented scenarios. For example, in Scenario 1, the DDA system opts to initiate the game at the easy level and increases the difficulty level as the opponent receives damage from the player. However, as the game starts at a predefined setting (20 ammo-2 shields), it is questionable whether the proposed DDA system can identify unpredictable player behavior or strategies. Similar behavior is observed in the case of Scenario 6. Under this scenario, the fuzzy logic-based system opts to initiate the game at the easy level, as the player has increased health and low ammo. Given the case, it is safe to assume that the proposed may fail in performing appropriate adjustments to the game environment considering long-term variability, unpredictability, and challenges involved in scaling difficulty. Furthermore, there is the risk of reducing the level of challenge for skillful players, as the DDA mechanism reduces the difficulty level when the player finds themselves in a disadvantageous position. This behavior is observed in the case of Scenario 5, where the player has low health.

Finally, the proposed approach does not straightforwardly introduce a winning strategy for players, as it dynamically tailors the game difficulty according to their playing style. In this regard, it adapts and maintains the level of challenge without making the game easy or difficult for the players. For example, when the player is in a disadvantageous position, e.g., low health, the system scales down the game difficulty and the behavior of the opponent. On the other hand, the system scales up the level of challenge when the player shows competence. Given that, the proposed DDA systems create a dynamic environment to keep the player engaged without securing their success. Thus, the presented application provides the appropriate level of challenge to avoid boredom or frustration, offering an adaptive experience to players.

## 6. Conclusions and Future Work

The present study presented an approach to the dynamic difficulty adjustment problem which incorporates a fuzzy logic inference system designed in MATLAB. As a testbed of this approach, the shoot 'em up game ShoottheFuzzyShip was developed in Unity. Due to the limitations of the game engine, MatUn was additionally developed to allow communication between ShoottheFuzzyShip and MATLAB. Using MATLAB's fuzzy inference engine and a named pipe, the proposed DDA system managed to create a dynamic game environment, which adapts dynamically in real time according to the player's skills and style.

To examine the adaptive game behavior, nine different gameplay scenarios were tested. In these experiments, the DDA system performed as intended and provided a satisfactory challenge to the player. When the player was in a difficult position, i.e., low health and ammo, the system scaled down the game difficulty. On the other hand, the enemy was more aggressive, and the game environment became more hostile when the player was in favor.

Given the obtained results, one can deduce that the functionality of the proposed DDA system outperforms that of other approaches [3,19], which relied on simple metrics, such as score. In this sense, the fuzzy logic-based DDA system is more responsive and flexible to the player's style, creating a dynamic game environment. This significantly improves player retention by offering balanced challenges. Furthermore, the fuzzy logic-based DDA system is more accessible compared to the ones that utilized expensive specialized devices [13,17]. Thus, fuzzy logic-based DDA is a widely applicable and cost-effective solution for creating adaptive gaming environments that offer unique experiences to players.

In the future, the proposed fuzzy logic inference system will be implemented directly to the enemy, as the current approach suffers from high computational cost and constraints of Unity. Given its performance, the proposed DDA mechanism will be applied to other game environments to test its scalability and adaptability to complex game situations. Similarly, a thorough experimental analysis involving a diverse group of players will be conducted to examine the generability of the proposed approach. Finally, it would be interesting to examine the current DDA system in the case of an adaptive agent. Thus, an adaptive agent will be created to control the player's spaceship and will incorporate machine learning techniques, such as reinforcement learning.

**Author Contributions:** Conceptualization, P.D.P. and D.E.K.; Methodology, P.D.P. and D.E.K.; Software, P.D.P.; Validation, P.D.P. and D.E.K.; Investigation, P.D.P. and D.E.K.; Formal Analysis, P.D.P.; Data Curation, P.D.P.; Writing—Original Draft Preparation, P.D.P.; Writing—Review and Editing, P.D.P. and D.E.K.; Visualization, P.D.P.; Supervision, D.E.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Appendix A. Pseudocodes

**Algorithm A1** Pseudocode for MatUn

```
1: Start Program
2: Initialize connection to Unity
3: Initialize MATLAB interface
4: Minimize MATLAB window for background operation
5: Print "Waiting for connection..."
6: Wait for successful connection with Unity
7: Print "Connected"
8: Try:
9: while connection is active do
10:     Declare input variables
11:     Read input data from Unity
12:     Process data in MATLAB
13:     Execute necessary MATLAB function according to input data
14:     Retrieve results from MATLAB
15:     Send results back to Unity
16: end while
17: Close connection
18: Print "Press any key to exit"
19: Wait for user input
20: Catch Exception:
21: Handle connection loss
22: Print "Connection lost, press any key to exit"
23: Wait for user input
24: End Try
25: End Program
```

**Algorithm A2** Pseudocode for `fuzzyFunc`

```
1: function FUZZYFUNC($a, b, c, d$)
2:     Load FIS model 'FuzzyModel.fis' into $fismat$
3:     Evaluate FIS using input vector $[a, b, c, d]$
4:     $result \leftarrow$ evalfis($[a, b, c, d], fismat$)
5:     Extract outputs:
6:     $e \leftarrow result[1]$                          ▷ Output for difficulty
7:     $f \leftarrow result[2]$                          ▷ Output for enrangedMode
8:     $g \leftarrow result[3]$                          ▷ Output for weaponSelector
9:     Return $(e, f, g)$
10: end function
```

## References

1. Paraschos, P.D.; Koulouriotis, D.E. Game Difficulty Adaptation and Experience Personalization: A Literature Review. *Int. J. Hum. Comput. Interact.* **2023**, *39*, 1–22. [CrossRef]
2. de Araujo, B.B.P.L.; Feijó, B. Evaluating dynamic difficulty adaptivity in shoot'em up games. In Proceedings of the Proc. XII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2013), São Paulo, Brazil, 16–18 October 2013; pp. 229–238.
3. Chrysafiadi, K.; Kamitsios, M.; Virvou, M. Fuzzy-based dynamic difficulty adjustment of an educational 3D-game. *Multimed. Tools Appl.* **2023**, *82*, 27525–27549. [CrossRef]
4. Soylucicek, A.E.; Bostanci, E.; Safak, A.B. A Fuzzy Logic Based Attack Strategy Design for Enemy Drones in Meteor Escape Game. *Int. J. Comput. Theory Eng.* **2017**, *9*, 167–171. [CrossRef]
5. Lara-Alvarez, C.; Mitre-Hernandez, H.; Flores, J.J.; Perez-Espinosa, H. Induction of Emotional States in Educational Video Games through a Fuzzy Control System. *IEEE Trans. Affect. Comput.* **2021**, *12*, 66–77. [CrossRef]
6. Sutanto, K.; Suharjito, S. Dynamic difficulty adjustment in game based on type of player with ANFIS method. *J. Theor. Appl. Inf. Technol.* **2014**, *65*, 254–260.

7. Hunicke, R. The case for dynamic difficulty adjustment in games. In Proceedings of the ACE'05: ACM International Conference on Advances in Computer Entertainment Technology, Valencia, Spain, 15–17 June 2005; Volume 265, pp. 429–433. [CrossRef]
8. Lopes, R.; Bidarra, R. Adaptivity Challenges in Games and Simulations: A Survey. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 85–99. [CrossRef]
9. Fisher, N.; Kulshreshth, A.K. Exploring Dynamic Difficulty Adjustment Methods for Video Games. *Virtual Worlds* **2024**, *3*, 230–255. [CrossRef]
10. Weber, M.; Notargiacomo, P. Dynamic Difficulty Adjustment in Digital Games Using Genetic Algorithms. In Proceedings of the 2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Recife, Brazil, 7–10 November 2020; pp. 62–70. [CrossRef]
11. Yang, Z.; Sun, B. Hyper-Casual Endless Game Based Dynamic Difficulty Adjustment System For Players Replay Ability. In Proceedings of the 2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), Exeter, UK, 17–19 December 2020; pp. 860–866. [CrossRef]
12. Campo-Prieto, P.; Cancela-Carral, J.M.; Rodríguez-Fuentes, G. Feasibility and Effects of an Immersive Virtual Reality Exergame Program on Physical Functions in Institutionalized Older Adults: A Randomized Clinical Trial. *Sensors* **2022**, *22*, 6742. [CrossRef] [PubMed]
13. Amprimo, G.; Rechichi, I.; Ferraris, C.; Olmo, G. Measuring Brain Activation Patterns from Raw Single-Channel EEG during Exergaming: A Pilot Study. *Electronics* **2023**, *12*, 623. [CrossRef]
14. Rezapour, M.M.; Fatemi, A.; Nematbakhsh, M.A. A methodology for using players' chat content for dynamic difficulty adjustment in metaverse multiplayer games. *Appl. Soft Comput.* **2024**, *156*, 111497. [CrossRef]
15. Bontchev, B.; Georgieva, O. Playing style recognition through an adaptive video game. *Comput. Human Behav.* **2018**, *82*, 136–147. [CrossRef]
16. Stein, A.; Yotam, Y.; Puzis, R.; Shani, G.; Taieb-Maimon, M. EEG-triggered dynamic difficulty adjustment for multiplayer games. *Entertain. Comput.* **2018**, *25*, 14–25. [CrossRef]
17. Szegletes, L.; Koles, M.; Forstner, B. Socio-cognitive gamification: general framework for educational games. *J. Multimodal User Interfaces* **2015**, *9*, 395–401. [CrossRef]
18. Pratama, N.P.H.; Nugroho, S.M.S.; Yuniarno, E.M. Fuzzy controller based AI for dynamic difficulty adjustment for defense of the Ancient 2 (DotA2). In Proceedings of the 2016 International Seminar on Intelligent Technology and Its Applications (ISITIA), Lombok, Indonesia, 28–30 July 2017; pp. 95–100. [CrossRef]
19. Pirovano, M.; Lanzi, P.L. Fuzzy tactics: A scripting game that leverages fuzzy logic as an engaging game mechanic. *Expert Syst. Appl.* **2014**, *41*, 6029–6038. [CrossRef]
20. Foreword, J.H.; Schell, J. *Unity in Action, Third Edition Multiplatform Game Development IN C#*; Manning Publications Co.: New York, NY, USA, 2022.
21. Attaway, S. *Matlab: A Practical Introduction to Programming and Problem Solving*, 4th ed.; Elsevier Inc.: Amsterdam, The Netherlands, 2016; pp. 1–574. [CrossRef]
22. Silberschatz, A.; Galvin, P.B.; Gagne, G. *Operating System Concepts*, 8th ed.; Wiley Publishing: Hoboken, NJ, USA, 2008.
23. Zadeh, L.A. Fuzzy sets. *Inf. Control* **1965**, *8*, 338–353. [CrossRef]
24. Mamdani, E.H.; Assilian, S. An experiment in linguistic synthesis with a fuzzy logic controller. *Int. J. Man. Mach. Stud.* **1975**, *7*, 1–13. [CrossRef]