Large Language Models and Dynamic Difficulty Adjustment: An Integration Perspective

Carlos H. R. Souza[1], **Saulo S. Oliveira**[1], **Luciana O. Berretta**[1,2], **Sérgio T. Carvalho**[1,2]

[1]Institute of Informatics – Federal University of Goiás (UFG)
Goiânia – GO – Brasil
[2]Advanced Knowledge Center for Immersive Technologies (AKCIT/UFG)
Goiânia – GO – Brasil

{carlos_henrique_rorato,saulosoares}@discente.ufg.br
{luciana.berretta,sergiocarvalho}@ufg.br

Abstract. *Dynamic Difficulty Adjustment (DDA) aims to enhance player retention by adjusting the difficulty level of a game according to the player's skills. However, rule-based DDA systems often struggle with scalability, adaptability, and the cognitive burden of defining exhaustive rules. In this paper, we propose the integration of Large Language Models (LLMs) into DDA mechanisms to overcome these limitations, based on prompt creation techniques. As a proof of concept, we apply this approach to the DDA-MAPEKit framework, replacing its static rule-based logic with LLM actions using ChatGPT, and an experiment was conducted in a space shooter game. The results showed promising outcomes, with pertinent adjustments to the game variables, no hallucinations, and values coupled to the current context. Overall, our findings suggest that LLM-based DDA mechanisms hold significant potential for improving adaptivity in digital games.*
Keywords *Dynamic Difficulty Adjustment, Large Language Models.*

1. Introduction

Dynamic Difficulty Adjustment (DDA) is a solution that addresses the need for adaptive gameplay in digital games [Seyderhelm e Blackmore 2021]. Its goal is to improve player retention by adjusting the game's difficulty level based on the player's skills [Seyderhelm e Blackmore 2021, Zohaib 2018]. The DDA-MAPEKit framework was created to support the development of DDA mechanisms [Souza et al. 2024, Souza et al. 2023a, Souza et al. 2023b]. It is based on the MAPE-K (Monitor, Analyze, Plan, Execute - Knowledge) loop, which is a concept from Self-Adaptive Systems renowned for its modular and highly flexible nature [Weyns 2021].

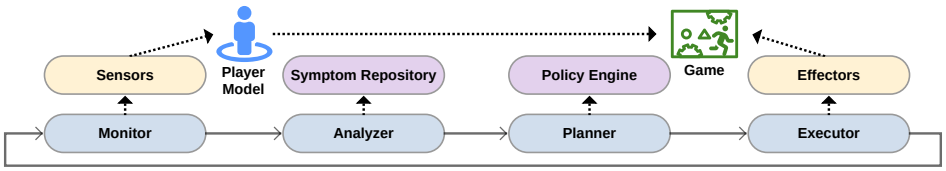

Figure 1. DDA-MAPEKit's main loop.

The Monitor observes the player data (Player Model) obtained through sensors (Figure 1). The Analyzer is then notified to calculate the player's performance and access the Symptom Repository, which gathers symptoms. Each symptom consists of a description and a threshold, that determines whether the performance aligns with the referred symptom. The Planner receives the adaptation request containing the identified symptom and checks for

an associated adjustment rule for it. Each rule consists of a symptom and its corresponding set of values for variables that must be changed to balance the difficulty. When applicable rules consistent with the player's profile are found, the Planner creates a Change Plan that includes these new values. The Executor arranges and sends the changes to the Effectors, that update the game variables. Thus, the framework offers a rule-based approach, where adjustments are made based on predefined rules and symptoms detected during gameplay.

However, a significant challenge in existing DDA frameworks lies in rule-based approaches' static and exhaustive nature [Tagliaro 2022, Segundo et al. 2016]. In this approach, the game designer must define all adaptation rules based on the player's performance classification [Tagliaro 2022, Segundo et al. 2016]. While rules hold promise, they may not effectively adapt or respond to each player's context [Rahimi et al. 2023, Streicher e Smeddinck 2016]. Moreover, configuring these rules can be difficult, requiring a high cognitive load on the game designer's part [Tagliaro 2022].

An emerging solution to this challenge is the integration of Artificial Intelligence (AI) into the dynamic difficulty adjustment process [Mi e Gao 2022, Tagliaro 2022, Seyderhelm e Blackmore 2021, Zohaib 2018]. Large Language Models (LLMs), such as ChatGPT[1], are state-of-the-art AI architectures designed to process and generate human-like text based on input prompts [Gallotta et al. 2024], as they employ transformer architectures, a type of deep learning model renowned for its effectiveness in handling sequential data [Radford et al. 2019]. They are increasingly being integrated into various dimensions and stages of game development [Gallotta et al. 2024, Hu et al. 2024], and we advocate that they can be used in DDA contexts.

This study aims to investigate the potential of LLMs as an alternative to traditional rule-based approaches in the context of Dynamic Difficulty Adjustment mechanisms. We advocate that integrating LLMs in the context of DDA in digital games can present another direction for generating adaptive gameplay experiences. As a proof of concept, we propose the integration of ChatGPT, a widely used LLM, into the DDA-MAPEKit framework. This integration involves replacing the Policy Engine of the framework with requests to ChatGPT, leveraging prompt engineering techniques and methods to integrate the LLM with game variables effectively. To the best of our knowledge, prior work has yet to investigate the relationship between LLMs and DDA [Gallotta et al. 2024]. Thus, we seek to contribute to the field by exploring possibilities in this direction.

2. LLM and DDA: An Integration Perspective

We advocate that LLMs can offer an avenue for dynamically generating variable adjustments based on in-game data, thus offering a solution to the challenges posed by traditional rule-based DDA mechanisms. The integration process consists of seven key steps, as follows.

1. **Definition of Game Variables Corresponding to Input:** Identify game variables reflecting player performance changes and potential skill-challenge imbalances. These variables serve as input for the LLM to generate dynamic difficulty adjustments.
2. **Definition of Game Variables Representing Game Difficulty:** Next, game variables representing the game's difficulty must be defined. These variables are the LLM's output, structured in JSON. Hendrix *et al.* [Hendrix et al. 2019] provide a six-step methodology for identifying these variables based on tests and detection of changes in game difficulty, that can be adopted.

[1] `https://chatgpt.com`

3. **Creation of JSON Examples: Input-Output:** Providing examples optimizes the LLM's understanding of input-output correlations. JSON format is employed due to its widespread usage and standard representation in various domains. Through tests, it has been observed that employing JSON format can reduce the hallucinations of LLMs and enhances data interpretation and output formatting.
4. **Creation of the Prompt (Situation, Purpose, Action, Examples, Request):** The prompt format was adapted from the SPAR framework[2] (Situation, Purpose, Action, and Result). The "Action" field contains a description of adaptation guidelines based on the Chain-of-Thought Prompting (CoT)[3] technique, aiming to direct the LLM's reasoning explicitly [Hu et al. 2024, Wei et al. 2022]. The "Examples" section utilizes few-shot prompting, indicated for scenarios with limited data processing capabilities [Hu et al. 2024]. This approach balances the need for quick responses with the model's capacity to learn from a few examples. Finally, the Request field, an integral part of the prompt, contains the game variables' values that are used for input.
5. **Sending the Message to the LLM and Receiving the JSON Response:** The message is sent to the LLM using protocols such as HTTP. The JSON response generated by the LLM is then received.
6. **Applying the Response to Game Variables:** The JSON string's validity is verified, then deserialized if valid. The obtained values are then directed to the corresponding game variables within the source code.
7. **Integrating Input and Output Values into Examples:** Lastly, the input and output data are included in an example buffer, ensuring the persistence of the LLM's learning and the attainment of increasingly accurate results. This buffer may be partially or entirely incorporated into subsequent prompts, depending on the LLM's token and processing limitations.

In summary, by replacing conventional rule-based mechanisms with LLM requisitions, we seek to overcome static approaches' limitations and enhance game design adaptability.

3. Proof of Concept: Integrating ChatGPT on DDA-MAPEKit

To integrate an LLM into the DDA-MAPEKit framework, we chose ChatGPT due to its widespread use and extensive application in various experiments. Our strategy focused on the DDA-MAPEKit's Policy Engine, replacing its exhaustively defined static rule logic with the LLM's dynamic capabilities. The Policy Engine is critical as it formulates the actions that most significantly impact the adjustment of difficulty variables. Configuring the Policy Engine is one of the most resource-intensive tasks for game designers using the framework. Thus, this integration aims to provide substantial benefits to the game designer by simplifying this process.

Following the proposed steps in our study, we first defined the input values as the data already used in the Policy Engine, including player profiles and performance symptoms, along with the current values of the variables to be adjusted. These inputs are structured in a JSON syntax that includes thresholds for each variable to prevent hallucinations and regulate adjustments. Game designers define and input the JSON examples into the framework through a JSON file. They must also insert their OpenAI API key for ChatGPT to process requests and responses.

[2] `https://beeazt.com/knowledge-base/prompt-frameworks/the-spar-framework`
[3] `https://www.promptingguide.ai/pt/techniques/cot`

During the framework's execution loop, input data and configurations made by the game designer are received, and a prompt is constructed in the previously discussed format (Figure 2 shows an example). We created a buffer that stores the most recent input-output cases to maintain some historical context. These cases are injected into the Examples section of the following prompt. The prompt is then formed by concatenating the instructions (Situation, Purpose, Action), the Examples (provided by the game designer and recent cases from the buffer), and the Request (comprising the current values of the game variables: profile, performance symptom, and variables to be adjusted). An HTTP request with the defined prompt is created and sent to the LLM. Upon receiving the response, it is verified to ensure compatibility and deserialized. Then, the resulting object is sent back to the Planner to continue the framework loop.

```
# Situation: You are a Dynamic difficulty adjustment mechanism for digital games. You Receive player performance symptom and profile, along with game variable identifiers and threshold.
# Purpose: Generate JSON with float values for game variable adjustments based on player performance symptom and profile. Adjust game variables within threshold. Increase values for player-helping mechanics if symptom is low, decrease if high.
# Action: If low symptom, decrease values for player-harming mechanics, increase for player-helping. If high symptom, increase values for player-harming, decrease for player-helping. For challenge-seekers, more pronounced changes; for explorers/socializers, smoother adjustments.
# Examples:
// Input
{"profile":"killer","symptom":"high","game_variables":
[{"description":"meteorsCount","threshold":[1,10],"value":5},...
// Output
{"game_variables": [{"description": "meteorsCount", "value": 4},
{"description": "meteorsSeconds", "value": 2}, ...
# Request:
// Input
{"profile":"explorer","symptom":"slightly.high","game_variables":
[{"description":"meteorsCount","threshold":[1,10],"value":4},...
// Output
```

Figure 2. Prompt format.

3.1. Experiment

Endless Space Shooter[4] is a game in the space shooter genre, which has been the genre used on several studies [Segundo et al. 2016, Figueira et al. 2018, Bicalho et al. 2023]. In this game, the player controls a spaceship, and encounter enemies, which he can attack. The DDA mechanism considered six game variables (scoring system): the number of special objects, asteroids, and enemy ships present at any given time, as well as the time gap between the appearance of each group of objects (*mC – meteorsCount; mS – meteorsSeconds; eC – enemiesCount; eS – enemiesSeconds; pC – pointsCount; pS – pointsSeconds*). The thresholds of the first five variables were defined as [1,10] and the sixth (*pS*) as [5,25].

When evaluating projects involving LLMs, prompt engineering, and games, assessments may involve observing the assertiveness and success of the generated prompts [Hu et al. 2024]. Considering this scenario, our experiment involved comparing static rules in the previous framework version with five outputs generated using the LLM-integrated version, each considering different possible input situations (various symptoms, player profiles, and variable values obtained during simulated matches).

3.2. Results

Table 1 presents the results of the simulations performed with different player profiles and performance symptoms, compared with the rule-based output values. Each row in the table corresponds to a specific simulation, showing both the input values for game parameters before the application of DDA, the rule-based defined values and the output values suggested by the LLM.

First of all, it was possible to observe that the LLM led to adjustments that were very close to those proposed by the rules of the previous framework version. In Simulation 1, for example, according to the system of rules, the values of the variables, which were in

[4]Open-source game available at `https://github.com/softwarengineer2/Endless-Space-Shooter-Unity-2D-Game/`.

{4,2,4,2,3,18}, would be modified to {3,2,3,2,2,20}. LLM moves in the same direction, offering {3,1,3,2,2,20} as output. Therefore, these values are within the thresholds and relatively close to what was defined by the rules' system. The same was observed in Simulation 2.

Table 1. Input and output values suggested by the LLM.

#	Profile	Symptom	Input Values						Output Values (rule-based approach)						Output Values (LLM-based approach)					
			mC	*mS*	*eC*	*eS*	*pC*	*pS*	*mC*	*mS*	*eC*	*eS*	*pC*	*pS*	*mC*	*mS*	*eC*	*eS*	*pC*	*pS*
1	Achiever	*sharply.high*	4	2	4	2	3	18	3	2	3	2	2	20	3	1	3	2	2	20
2	Achiever	*high*	4	3	2	3	4	12	3	2.5	2	2.5	2	15	3	2	3	3	2	14
3	Explorer	*low*	5	3	5	2	4	20	2	4	1	4	3	6	4	3	4	2	6	16
4	Explorer	*very.low*	8	2	6	2	3	25	2	5	1	5	3	6	7	4	4	3	5	22
5	Killer	*slightly.high*	5	3	6	3	6	14	3	2.5	2	2.5	2	12	6	2	7	2	5	16

Otherwise, using LLMs led to more fine-grained adjustments because it took into account the current values of the variables to determine the changes, which made the adjustment smoother and more relevant to the context. Simulations 3 and 5 are examples of finer adjustments because the given outputs were very far from what would be expected by the rules, which would possibly result, in this context, in a sudden adjustment, perhaps noticed by the player. Most likely, this distance from what was predicted by the rules-based system is due to the game designer being unable to predict scenarios in which the game variables have values that differ from what was expected.

Additionally, it was possible to observe that LLM followed the strategy requested through the prompt. For example, in Simulation 4, it can be seen that the proposed adjustment is quite different from the defined rules, most likely because it takes the context into account, but it still makes correct decisions: in the case of a player with an exploratory profile (i.e., who does not want a high level of challenge) and is underperforming (characterized by a *very.low* symptom), it should have fewer enemies, with a longer instantiation time between them, a fact that was verified in the proposed adjustments. On the other hand, the number of available special objects slightly increased, subtly decreasing the time between spawnings.

Moreover, we observed no hallucinations in the LLM outputs during the experiment, both in format and data. That fact points to a certain consistency of this approach and the engineered prompt, as the generated adjustments adhered to the expected format and data thresholds.

4. Final Remarks

This paper presented an integration perspective of Large Language Models (LLMs) into the Dynamic Difficulty Adjustment (DDA) mechanism using the DDA-MAPEKit framework, aiming to address the limitations of traditional rule-based DDA methods. Through a proof of concept experiment, we could collect evidence indicating that the LLM-integrated DDA mechanism presented pertinent and faithful adjustments to its proposal, without hallucinations and with values coupled to the current context of the game variables.

For future work, additional experiments can be conducted using simulations to further evaluate our proposed approach across different game genres and scenarios. Additionally, we envision conducting experiments with game developers to evaluate the usability and effectiveness of the framework in a practical game development context. In this way, we seek to advance the understanding and application of LLMs in game design and contribute to developing more adaptive gaming experiences.

Acknowledgments

The authors would thank the AKCIT (Advanced Knowledge Center for Immersive Technologies – UFG) for funding this research.

References

Bicalho, L. F., Baffa, A., e Feijó, B. (2023). A dynamic difficulty adjustment algorithm with generic player behavior classification unity plugin in single player games. In *Proceedings of the 22nd Brazilian Symposium on Games and Digital Entertainment*, SBGames 2023, pages 1–10, Rio Grande. ACM.

Figueira, F., Nascimento, L., Junior, J., Kohwalter, T., Murta, L., e Clua, E. (2018). BinG: A framework for dynamic game balancing using provenance. In *2018 17th SBGames*, pages 57 – 66, Brazil. IEEE.

Gallotta, R., Todd, G., Zammit, M., Earle, S., Liapis, A., Togelius, J., e Yannakakis, G. N. (2024). Large language models and games: A survey and roadmap.

Hendrix, M., Bellamy-Wood, T., McKay, S., Bloom, V., e Dunwell, I. (2019). Implementing adaptive game difficulty balancing in serious games. *IEEE Transactions on Games*, 11(4):320–327.

Hu, S., Huang, T., Ilhan, F., Tekin, S., Liu, G., Kompella, R., e Liu, L. (2024). A survey on large language model-based game agents.

Mi, Q. e Gao, T. (2022). Improved belgian AI algorithm for dynamic management in action role-playing games. *Applied Sciences*, 12(22):11860.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., e Sutskever, I. (2019). Language models are unsupervised multitask learners.

Rahimi, M., Moradi, H., Vahabie, A.-h., e Kebriaei, H. (2023). Continuous reinforcement learning-based dynamic difficulty adjustment in a visual working memory game.

Segundo, C., Calixto, K., e Gusmão, R. (2016). Dynamic difficulty adjustment through parameter manipulation for space shooter game. In *2016 15th SBGames*, pages 234–237, Brazil. SBC.

Seyderhelm, A. J. A. e Blackmore, K. (2021). Systematic review of dynamic difficulty adaption for serious games: The importance of diverse approaches. *SSRN Electronic Journal*, 1(1):1–45.

Souza, C., Berreta, L., e de Carvalho, S. (2023a). Performance evaluation of a framework for dynamic difficulty adjustment in single player games (in potuguese). In *Proceedings of XI Escola Regional de Informática de Goiás*, pages 1–10, Porto Alegre, RS, Brasil. SBC.

Souza, C., Oliveira, S., Berretta, L., e Carvalho, S. (2023b). The use of health data for dynamic difficulty adjustment in serious games (in portuguese). In *Proceedings of SBCAS 2023*, pages 479–484, Porto Alegre, RS, Brasil. SBC.

Souza, C. H. R., De Oliveira, S. S., Berretta, L. O., e de Carvalho, S. T. (2024). DDA-MAPEKit: A framework for dynamic difficulty adjustment based on MAPE-K loop. In *Proceedings of the 22nd Brazilian Symposium on Games and Digital Entertainment*, SBGames'23, page 1–10, New York, NY, USA. Association for Computing Machinery.

Streicher, A. e Smeddinck, J. D. (2016). Personalized and adaptive serious games. In *Entertainment Computing and Serious Games*, pages 332–377. Springer International Publishing, USA.

Tagliaro, L. R. G. (2022). An implementation of adaptive difficulty systems for challenging video games. Undergraduate Thesis.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., e Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models.

Weyns, D. (2021). *An Introduction to Self-Adaptive Systems*. Wiley, USA.

Zohaib, M. (2018). Dynamic difficulty adjustment (DDA) in computer games: A review. *Advances in Human-Computer Interaction*, 2018:1–12.