Customizable Procedural Content Generation with LLMs

Marcelo Júnior[1], **Mario Adaniya**[1], **Luiz Nunes**[1]

[1] Departamento de Computação - Centro Universitário Filadélfia (UNIFIL)
Londrina – PR – Brasil

fmarcelocarlos@gmail.com, luiz.nunes@unifil.br, mario.adaniya@unifil.br

Abstract. Introduction: *Procedural Content Generation (PCG) faces challenges in creating customizable levels with specific characteristics, such as difficulty patterns and unique layouts. Large Language Models (LLMs) offer a promising solution due to their natural language understanding and generative capabilities.* ***Objective:*** *This study proposes fine-tuning the DeepSeek-R1 LLM to generate customizable 2D game levels (similar to The Legend of Zelda), evaluating its feasibility for PCG.* ***Methodology:*** *Using a dataset with around 1000 playable levels of the Zelda-like game the model (DeepSeek-R1-Distill-Llama-8B-unsloth-bnb-4bit) was fine-tuned with instructions specifying element quantities (e.g., enemies, blocks), and then evaluated using various metrics, including playability (ASTAR agent), novelty (Levenshtein distance), diversity (graph-based analysis), and accuracy (input adherence).* ***Results:*** *The model achieved very high results on novelty, diversity and playability, but falling a little on the accuracy tests. However it still shows impressive adherence for the inputs specifications, demonstrating LLMs' potential for customizable PCG, outperforming earlier models like GPT-2/3.*
Keywords *Procedural Content Generation, Large Language Models, Level Design, Game Development.*

1. Introduction

The development of game levels is a time-consuming and costly process that involves a significant effort from everyone involved, since it's a central part of the player's experience. With the evolution of technology, the demand for games with increasingly more content has increased, overwhelming designers and developers, leading to missed deadlines and budget overruns. Procedural Content Generation (PCG) emerges as a way to address these challenges, using algorithms capable of automatically creating content, reducing costs and development time, while maintaining quality, adding diversity and replayability.[Hendrikx et al. 2013].

Different algorithms have been extensively explored to implement Procedural Content Generation (PCG). Traditionally, genetic algorithms are one of the most common techniques for PCG. Examples are a level generator [Moghadam e Rafsanjani 2017], an endless runner game [de Pontes e Gomes 2020], and a serious game; that uses the algorithm to generate dynamic and adaptable content [Mitsis et al. 2020].

In recent years, the use of Artificial Intelligence (AI) techniques in procedural generation has gained prominence, especially with the emergence of studies involving Machine Learning [Summerville et al. 2018] and Reinforcement Learning [Khalifa et al. 2020], which have shown promising results and opened new possibilities for the field of PCG.

Despite advances in PCG techniques, one of the main limitations of current methods is how complex can be to generate custom content. Adapting the algorithm to create levels with specific characteristics, such as difficulty patterns, visual styles, or unique layouts, is a tough task that often requires intricate changes to the algorithm's logic and generation parameters. Furthermore, research on this topic is limited, yet it is an important issue that can directly impact the player's experience [Sudhakaran et al. 2024].

To create a more flexible and customizable method for Procedural Content Generation, the use of Large Language Models (LLMs) emerges as a promising solution. According to the following study [Zhao et al. 2023], LLMs refer to language models based on the Transformer architecture that possess billions parameters and are trained on vast amounts of textual data. These models are particularly powerful for understanding natural language and solving complex tasks through text generation. Due to their high potential, LLMs have gained significant attention in the technology market. The release of ChatGPT by OpenAI [OpenAI 2024] in 2022, for example, solidified the importance of these tools, attracting the attention of major companies that developed their own LLMs, such as Google with Gemini [Team et al. 2023], Meta with Llama [Touvron et al. 2023] and the chinese DeepSeek [DeepSeek 2024].

Although LLMs already demonstrate remarkable capabilities in their "base" form, they can still be enhanced through the fine-tuning process. In the case of various models such as Deepseek, this process involves the use of three parameters: instruction, input, and output. The instruction provides general guidelines on what the model should generate, while the input adds more specific context, detailing the particular characteristics of the task. The output represents the desired result that the model should learn to generate. By adjusting its predictions based on these pairs, the model refines its parameters so that, when receiving similar inputs in the future, it can generate responses more suited to the specific task [Zhao et al. 2023].

To explore the potential of LLMs and fine-tuning for Procedural Content Generation, this article proposes fine-tuning a model to generate levels for a 2D game similar to The Legend of Zelda, using the Zelda-Game-Level dataset [Zafar 2023], which contains over 1000 playable levels. The model will be based on distilled version of DeepSeek-R1, specifically DeepSeek-R1-Distill-Llama-8B-unsloth-bnb-4bit. This model will undergo the fine-tuning process and then be evaluated for its ability to generate playable and customized levels, with the aim of assessing the feasibility of using LLMs as a practical tool for PCG.

2. Related Works

The application of Large Language Models (LLMs) to Procedural Content Generation (PCG) is an emerging and promising research area, with recent studies exploring innovative ways to enhance game development. This section examines other examples of LLM-Driven Level Generation, analyzing their contributions and limitations, we can contextualize their relevance to the present work and identify opportunities for further advancement of this topic.

Emerging researches into LLM-driven PCG underscores its transformative potential for game development. Pioneering this area, [Todd et al. 2023] demonstrated GPT-2's capability to generate novel, playable Sokoban levels, emphasizing the critical role of high-quality datasets in achieving robust results. Following this study, [Sudhakaran et al. 2024] introduced MarioGPT, a modified GPT-2 architecture enhanced with cross-attention layers and BART-based natural language processing, enabling text-prompted level generation for Super Mario Bros. By integrating novelty search and data augmentation, the model achieved 88.4% playability and strong adherence to user-specified constraints (e.g., enemy counts, terrain features), even with limited training data.

Taking one step further, subsequent studies explored how this method could be applied in commercial games, using GPT-3 and fine-tuning it on small datasets to produce viable levels at moderate cost, highlighting its scalability for industry use [Todd et al. 2023]. Collectively, these efforts validate LLMs as versatile tools for PCG, balancing creativity with functional rigor. However, the quality of available LLMs has improved rapidly. Models like DeepSeek R1 and GPT-4o are significantly more capable than their predecessors. Also, open-source alternatives such as DeepSeek and Llama can provide cost-effective options compared to GPT.

3. Methodology

3.1. Game Description

The game chosen is a simplified version of the classic The Legend of Zelda, originally released for the Nintendo Entertainment System [Nintendo 1986]. The game is represented on a 13x9 grid, surrounded by walls, where the player (avatar) must navigate to complete the objective. In each level, the avatar needs to collect a key and bring it to a door to win, while avoiding or fighting enemies scattered across the map. These enemies vary in speed and behavior, adding the need of strategy from the player to reach its goal. This specific version of the game is made available by GVGAI (General Video Game Artificial Intelligence), a framework able to read VGLC levels of different games, and use intelligent agents to play and test it [Perez-Liebana et al. 2019].

3.2. Data Collection and Preparation

The Data for training the model were obtained from the Zelda Game Levels dataset available on Kaggle [Zafar 2023]. This dataset contains 2048 image files of the GVGAI Zelda game levels, of which 1024 are considered playable and 1024 are considered unplayable.

The original data set consists of image files, but for the purposes of this study, the levels needed to be converted to the VGLC textual format, as LLMs are designed to work with textual input.

The Figure 1 is an image of a level taken directly from the dataset.

3.2.1. Conversion of Images to Text

To start the conversion, a image file containing all the different elements present in the game was selected and separated into a grid, sorting all the tiles into different spaces. Analyzing each tile separately, It was possible to get the RGB value of them.

With this information, a script written in Python looped all the images in the dataset, separating them into a grid. Then each space of the grid was compared to the colors of each tile, the closest value would determine the token each space received.

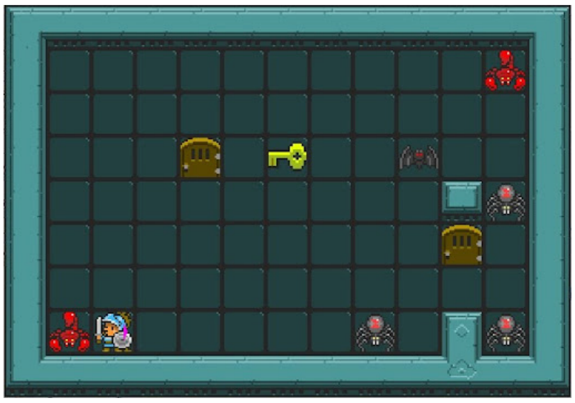


Figure 1. Level example from the dataset Font: [Zafar 2023]

The output for each level would look like Figure 2, this example is a exact representation of the level showed in Figure 1, converted into a tokenized text, following the VGLC convention.

3.2.2. Prompt Engineering

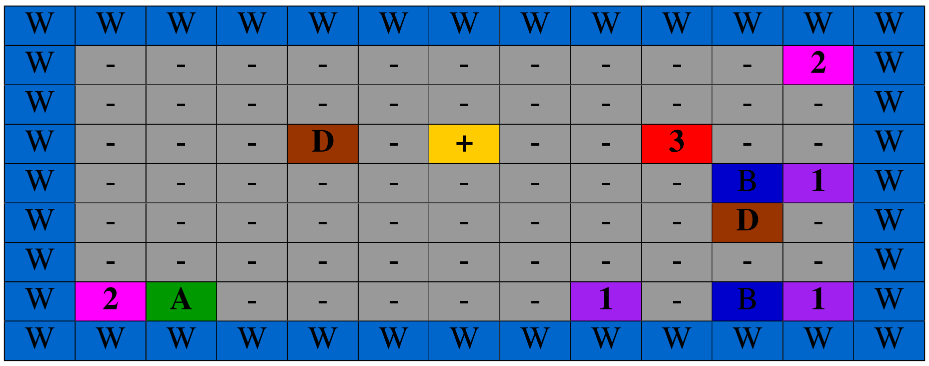
Next step is to create the prompt used to fine-tune the model, the prompt is a very important part, since it is what will be explaining and teaching the model what exactly needs to generate, so it needs to be written clearly and packed with all the information needed. In this case, the Deepseek model usually is trained with the alpaca prompt structure [Taori et al. 2023], that divides the prompt in 3 parts: instruction, input and output.

The instruction should dictate what is the primary objective the model has to achieve and give sufficient information and context to generate a good response, so basically it describes that the model needs to generate a game level and give the guidelines required for the level to be playable. Resulting in the following instruction:

"Generate a level for a 2D game where each level is a 13x9 grid where 'W' represents the wall surrounding the room, 'A' represents the player, 'D' represents the door, '+' represents the key, 'B' represents blocks, '1', '2', '3' represent enemies, each with different speeds (1 = quick, 2 = normal, 3 = slow), '-' represents empty spaces the player can navigate. The goal of the game is to avoid enemies, collect the key, and reach the door to win. The level should have at least one player, one door and one key. Ensure the level is well-structured and allows the player a path to reach the key and door."

This instruction describes everything required by any level on the dataset, in addition, it explains what each token represents and gives a general idea of how the game is played. Since all the information present here is common to all levels, it does not needs to be changed for each particular case.

However, the inputs should be created for each level individually, the inputs are responsible of giving more context and details for the specific level, in this case It will give the number of different elements of the game (number of enemies, blocks, doors and keys), adding a layer of customization for each level generated.



W	W	W	W	W	W	W	W	W	W	W	W	W
W	-	-	-	-	-	-	-	-	-	-	2	W
W	-	-	-	-	-	-	-	-	-	-	-	W
W	-	-	-	D	-	+	-	-	3	-	-	W
W	-	-	-	-	-	-	-	-	-	B	1	W
W	-	-	-	-	-	-	-	-	-	D	-	W
W	-	-	-	-	-	-	-	-	-	-	-	W
W	2	A	-	-	-	-	-	1	-	B	1	W
W	W	W	W	W	W	W	W	W	W	W	W	W

Figure 2. Textual representation of the level Font: Author

Table 1. Legend for the Textual representation Font: Author

Colors and Character	Meaning
-	Empty
W	Wall
B	Block
A	Player
+	Key
D	Door
3	Bat
2	Scorpion
1	Spider

Instead of using the exact number of each element present in the level the numbers were replaced with expressions that represented a range of numbers, for block and enemies, they could be assigned three quantities, "no" meaning 0, "some" for an average amount or "many" meaning more than average. for keys and doors only two possibilities were possible, since they are essential elements for this game, "one" meaning only one or "many" meaning more than one. With this arrangement, the model would need to learn less unique inputs and would have more freedom to develop the level, since it would not be tied to very specific constraints.

This is a example of one input used on the dataset: "Generate a level with some blocks, many enemies, one door, one key"

The Figure 3 shows the distribution of each element across the levels on the dataset, making it easier to understand and see the frequency that each element appears.

3.3. Hardware

The fine-tuning of this model was entirely done in the Cloud, using the Google Colab, which is a version of jupyter notebooks that can be accessed on the browser. This tool gives access to use a simple GPU, Nvidia Tesla T4, that is not very powerful but with some optimizations to reduce memory consumption, it can handle the fine-tuning of models with 7 to 8 Billion parameters.

Those optimizations were achieved using Unsloth [Unsloth 2024], an open-source library for Python that can be used with Nvidia GPUs. It can reduce memory usage with

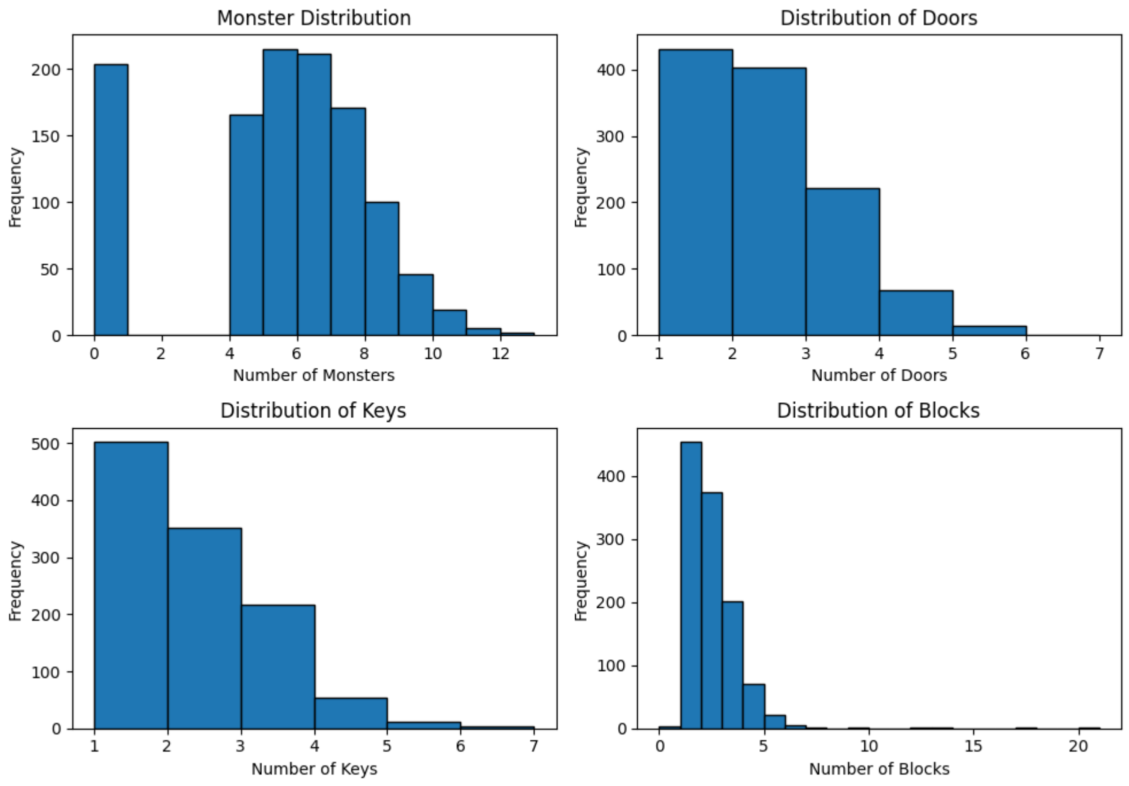


Figure 3. Graphs for the occurrence of each element Font: Author

quantization techniques allied with Low-Rank Adaptation (LoRA).

3.4. Fine-tuning

The model chosen to be fine-tuned was a variation of DeepSeek-R1, called DeepSeek-R1-Distill-Llama-8B-unsloth-bnb-4bit. This specific model was chosen because it is lightweight and able to run in the current hardware, It is already optimized for unsloth and It has a key enhancement compared to the other options.

The enhancement in the Deepseek-R1 architecture is its DeepThink reasoning mechanism [AI 2024], which employs progressive token-by-token deliberation to improve output coherence, helping on the self-verification of the model, preventing mistakes and improving adherence for the details on each input and instruction.

The training itself was done using the Unsloth's parameters recommendations with one epoch of 146 steps. The process went without any problems, the training loss was stable and no sign of overfitting.

3.5. Evaluation

The evaluation of the model was conducted using a set of metrics employed by [Todd et al. 2023]. These metrics cover essential aspects to evaluate LLMs and game levels, those aspects being: playability, novelty, diversity, and accuracy.

3.5.1. Playability

To evaluate playability, an approach based on the ASTAR agent available in the GVGAI framework was used. This agent is capable of playing automatically and completing the vast majority of levels on its own. The goal of this metric is to verify whether the generated levels are playable according to the rules of the game. A level is considered playable if:

- It contains a key and a door;
- It is solvable by the ASTAR agent, which is capable of finding a viable solution within 2000 steps. If the agent fails to find a solution within this limit, the level is classified as unplayable.

This metric assesses the minimum functionality of the levels and is crucial to ensure that the generated content can be used without external configuration.

3.5.2. Novelty

The Novelty metric measures how unique the generated levels are compared to those in the training set and ensure that the model is not copying levels from the training data. This evaluation was based on the Levenshtein distance (or edit distance), which quantifies the minimum number of operations (insertions, deletions, and substitutions) required to transform one text into another.

For the evaluation, generated levels and the original levels from the dataset, were compared against each other, if the edit distance between them is grater than 5 ($k = 5$), they will be considered novel.

3.5.3. Level Diversity

Level Diversity refers to the difference between all levels generated. This metric ensures that generated levels remain distinct and helps verify if the model is overgeneralizing their data.

The evaluation of diversity builds upon the concept of edit distance, similar to the novelty assessment, but extends it with a graph-based approach. Two levels are considered distinct if their edit distance exceeds a predefined threshold ($k = 5$). To quantify diversity, the generated levels are represented as a graph, where each node corresponds to a level, and edges connect nodes only if their edit distance is greater than the threshold. The Diversity Rate is then calculated as the ratio of the size of the largest clique to the total number of generated levels, providing a measure of how different the outputs are within the generated set.

$$\text{Diversity} = \frac{\text{Size of Biggest Clique}}{\text{Total Amount of Levels}}$$

3.5.4. Accuracy

To assess the fidelity of the generated levels to the input instructions, accuracy was measured by comparing the output against the specified design parameters. Each input outlined precise level characteristics, including the quantity of enemies, blocks, keys, and doors.

The evaluation was done with a script, able to read the levels generated and the inputs used, it could check all tokens for each level and count how many of the elements were present, then compare with the input and classify that level as accurate or not.

This metric is essential for determining the model's capability to produce customizable levels that accurately reflect the constraints provided in natural language, ensuring reliable and user-aligned level generation.

4. Results

For the evaluation of the model, 1300 levels were generated, 50 levels for each different input combination.

4.1. Novelty

Using the edit distance metric with a threshold value of $k = 5$, it was possible to measure the models' ability to generate truly new levels. From the total of 1300 levels, 90% were considered novels compared to the original dataset.

4.2. Diversity

The model also demonstrated a high capacity for diversity, with all the levels being considered diverse enough from each other.

4.3. Playability

Playability was evaluated using the ASTAR agent within the GVGAI framework, which used 3 attempts with a maximum of 2000 actions to test each level and determine whether they could be completed. Out of the 1300 generated levels, 1040 were classified as playable, around 80%. However the agent can make mistakes and/or get stuck in some levels, so using a more simple approach and checking all levels that contains a door, a key and a player, 89% have succeeded in this criteria.

4.4. Accuracy

The accuracy of the models was analyzed by comparing the characteristics of the generated levels with the specifications provided in the inputs. Out of the 1300 levels, 62.8% were considered accurate. This result indicates a good customization via inputs but with room for improvement.

4.5. Results Analysis

The results presented on this section shows that the majority of generated levels exhibited diversity, originality, playability, and functional accuracy, demonstrating the robust generalization capabilities of the model.

While accuracy metrics show room for improvement, Table 2 reveals that error frequencies remain within acceptable thresholds. On the top issues, two of them are related to missing Blocks, which is a problem that could be solved in future studies by dataset augmentation, something similar to what was done for levels with the "no enemies" input.

In previous iterations of this study, the model suffered to generate levels without any enemies, this error came from the dataset lack of examples for this input. By augmenting 200 levels to enemy-free examples, the model successfully learned this pattern, with current evaluations showing no recurrence of this issue. This shows how proactive changes on the training material can solve many issues around the adherence of certain patterns.

Table 2. Frequency of errors in generated levels Font: Author

Error	**Frequency (%)**
Keys: Expected 2+, found 1	16.4
Blocks: Expected 2+, found 1	11.3
Blocks: Expected 1-2, found 0	8.2
Keys: Found 0	5.4
Doors: Expected 2+, found 1	5.2
Player: Expected 1, found 0	5.0

However, even if the block-related issues were completely solved, the model still would not be completely accurate, leaving many options for future improvement of this idea, a different approach to prompt engineering maybe can result in a better accuracy, or different parameters for the fine-tuning can help the model learn the patterns on the dataset or the implementation of post-processing checks for each level generated can help fix some mistakes made by the LLM. All these results can provide guidance for future improvements over this concept.

5. Conclusion

The results demonstrate the feasibility of developing customizable PCG systems using LLMs, despite certain model limitations, It shows that natural language alone can be effectively utilized to create varied game levels featuring distinct layouts, variable elements and different difficulty settings. This approach significantly lowers the technical barrier for content generation, as it eliminates the need for specialized programming knowledge or complex algorithms.

Furthermore, the entirely free development pipeline presented in this study makes PCG tools more accessible to indie game developers operating with limited budgets. Unlike expensive proprietary systems used by major studios, this solution provides a cost-effective alternative that maintains creative flexibility while reducing production costs. This accessibility could potentially democratize game development tools and foster innovation within the indie game community.

References

AI, D. (2024). Deepseek-r1 model architecture. `https://deepseek.com/docs/r1-model`. Accessed: 2024-07-20.

de Pontes, R. G. e Gomes, H. M. (2020). Evolutionary procedural content generation for an endless platform game. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 80–89. IEEE.

DeepSeek (2024). Deepseek chat. `https://www.deepseek.com`. Accessed: [Insert Access Date].

Hendrikx, M., Meijer, S., Van Der Velden, J., e Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 9(1):1–22.

Khalifa, A., Bontrager, P., Earle, S., e Togelius, J. (2020). Pcgrl: Procedural content generation via reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 95–101.

Mitsis, K., Kalafatis, E., Zarkogianni, K., Mourkousis, G., e Nikita, K. S. (2020). Procedural content generation based on a genetic algorithm in a serious game for obstructive sleep apnea. In *2020 IEEE Conference on Games (CoG)*, pages 694–697. IEEE.

Moghadam, A. B. e Rafsanjani, M. K. (2017). A genetic approach in procedural content generation for platformer games level creation. In *2017 2nd Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, pages 141–146. IEEE.

Nintendo (1986). The Legend of Zelda. [Nintendo Entertainment System].

OpenAI (2024). Página oficial da openai. Accessed: 27 de outubro de 2024.

Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J., e Lucas, S. M. (2019). General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games*, 11(3):195–214.

Sudhakaran, S., González-Duque, M., Freiberger, M., Glanois, C., Najarro, E., e Risi, S. (2024). Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems*, 36.

Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A. K., Isaksen, A., Nealen, A., e Togelius, J. (2018). Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, 10(3):257–270.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., e Hashimoto, T. B. (2023). Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. (2023). Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Todd, G., Earle, S., Nasir, M. U., Green, M. C., e Togelius, J. (2023). Level generation through large language models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*, pages 1–8.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Unsloth (2024). Unsloth. Accessed: 2024.

Zafar, A. (2023). Zelda game levels. Accessed: 10/2024.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223*.