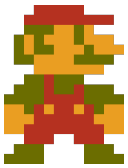

Generating Playable Mario Levels with GANs

Holden Schermer

May 2025

1 Abstract

Creating game levels remains a difficult challenge, particularly when trying to ensure they're both fun to play and actually completable. My project investigates using GANs to generate platformer game levels by using data from the Mario AI Framework. I trained a generative adversarial networks (GAN) to capture design patterns from classic Mario games and create new level segments. To make sure my generated levels completable, I built in playability testing using both rule-based checks and an AI agent that attempts to complete each individual level. I aimed to show that AI can create diverse, enjoyable platformer levels that meet basic gameplay requirements and capture the essence of human-designed levels.

2 Introduction

Game developers have long explored ways to automate level design, particularly through procedural content generation. Platform games are especially challenging due to their need for careful pacing, obstacle placement, and fairness qualities that require time and expertise to craft by hand. Traditional rule-based generators can produce functional layouts, but often lack the creativity and nuance of human-made levels.

Recent advances in machine learning provide a new direction. Instead of relying on hand-coded rules, models like generative adversarial networks (GANs) can learn directly from example levels, capturing the patterns that make them playable and engaging. In this project, I explored whether GANs could be used to generate Super Mario-style levels that not only resemble authentic platformer stages but also offer coherent gameplay experiences.

To ensure generated levels were both diverse and playable, I incorporated conditional generation, structural constraints, and post-processing heuristics. The goal was to balance creative freedom with the functional requirements of a platformer levels that are visually convincing, structurally sound, and offer fair challenges. Ultimately, this work aims to demonstrate how AI can assist in automating game design without sacrificing quality or playability.

2.1 Literature Review

Traditional procedural content generation (PCG) systems largely depended on manually designed algorithms that combine predefined tiles into levels [1]. While these approaches consistently produce playable levels, they rarely generate truly novel designs because their output space is limited to combinations the designer explicitly anticipated. Machine learning approaches, especially generative adversarial networks (GANs), offer a more expansive design space by learning level structure directly from examples, allowing them to create patterns beyond those specifically seen during training.

In their work on 2D platformers, Rodríguez Torrado and colleagues developed the Conditional Embedding Self-Attention GAN (CESAGAN), demonstrating that self-attention mechanisms help generators maintain long-range spatial relationships like keeping pipes vertically aligned throughout a level [2]. CESAGAN featured a bootstrapping process where generated levels passing playability tests were reincorporated into training data—an approach that directly influenced my use of playtesting filters. Similarly, Rajabi's team integrated deep convolutional GANs with reinforcement learning agents that tested each generated level, filtering out unbeatable designs and moving generation toward balanced difficulty [3]. Both studies show how GANs can implicitly learn level design principles while remaining flexible enough to adapt to high-level goals like variety or challenge level.

Training instability and mode collapse remain persistent problems in GAN research. Silva and colleagues reviewed hybrid approaches that combine GANs with rule-based post-processing to enforce constraints when generators produce unrealistic outputs [4]. I adopted this hybrid philosophy through my structural loss components and sky-cleaning heuristics, which act as guardrails layered onto the adversarial training objective. Lack of data presents another challenge. The classic Mario games contain relatively few original levels. Mao's research shows that basic data augmentation techniques like horizontal flipping and subtle repositioning create sufficient variety for stable GAN training without introducing fake artifacts [5]. My preprocessing pipeline implements similar augmentation strategies.

While playability forms a necessary foundation, it doesn't guarantee engaging gameplay. Levels can be technically completable yet boring. Shaker, Yannakakis, and Togelius pioneered experience-driven PCG approaches that incorporated player behavior data into generation systems, allowing them to customize difficulty curves to individual player skills [6]. Though real-time personalization exceeds my current scope, their findings show why my generator must produce diverse level distributions. Providing raw material that future personalization systems could filter or resequence according to player needs.

Other generative models present tradeoffs for level design. Transformers excel with sequential data but struggle with the global coherence needed for two-dimensional layouts. Variational autoencoders often produce visually soft outputs that don't translate well to discrete tile sets. GANs, however, directly optimize for realistic outputs through adversarial training, creating categorical results that map back to level tiles. Furthermore, conditional GANs provide explicit control mechanisms (like difficulty parameters) while leveraging convolutional architectures that capture spatial patterns effectively closing the gap between data-driven generation and designer-controlled constraints. For these reasons, GANs offer the optimal balance of expressiveness, controllability, and output quality for my goal of generating playable, varied Mario-style platformer levels.

3 Methods

The central objective of this study was to explore the capability of generative adversarial networks (GANs) to automatically produce Super Mario Bros-style game levels that are both playable and stylistically coherent with original human-designed levels. The methodology involved four primary stages: data preprocessing, neural network architecture design, adversarial training with a custom structural loss, and thorough evaluation with post-processing steps. All code referenced throughout this section was independently developed and is detailed explicitly in the accompanying Jupyter notebook.

3.1 Data Preprocessing

The initial data preprocessing phase required transforming raw Super Mario Bros level data from ASCII-encoded text into numeric tensors appropriate for neural network training. Original game levels from the Mario AI Framework were represented in ASCII format, with each character corresponding uniquely to specific tiles such as enemies, ground blocks, platforms, or empty space. The critical preprocessing step, shown in the first cell, mapped each ASCII character to a unique integer identifier. This numeric mapping was essential to ensure compatibility with PyTorch's tensor-based computations.

Subsequently, levels were segmented into standardized chunks of 16×16 tiles. This specific chunk size closely mirrors the dimensions of a typical Mario gameplay screen, thus allowing generated levels to realistically reflect actual gameplay scenarios. By segmenting levels into smaller, uniform chunks, the complexity of training data was reduced while at the same time increasing the amount of training data avaliable. This enhanced computational efficiency and enabling more effective model training. Implementation details for this step can be found in cell 2 of the Jupyter notebook.

To ensure basic structural integrity and playability, I implemented a filtering function `clean_chunk`, which excluded chunks lacking sufficient ground tiles. This filtering was crucial, as ground tiles form the basis of fundamental gameplay mechanics like character movement and platform interaction. Including structurally incomplete chunks in training data would have negatively impacted the GAN's ability to generate functional levels.

A custom PyTorch dataset class, `MarioDataset`, was also created to streamline loading, batching, and shuffling data during training. This class leveraged PyTorch's DataLoader functionalities, increasing efficiency and ensuring optimal data presentation during the GAN's training phases.

3.2 GAN Architecture

The GAN architecture was composed of two convolutional neural networks specifically tailored for the task of level generation. The generator network, detailed in notebook cell 3, consisted of convolutional transpose layers complemented by batch normalization and Leaky ReLU activations. This design choice effectively captured and reproduced spatial dependencies inherent to platform game designs.

Similarly, the discriminator network (specified in cell 4) distinguished authentic Mario level chunks from those generated artificially. Its architecture included convolutional layers with

Leaky ReLU activations, culminating in a sigmoid activation for binary classification. Balancing complexity between the generator and discriminator networks ensured stable adversarial training and prevented common GAN issues such as mode collapse.

3.3 Adversarial Training with Custom Structural Loss

Training followed the standard GAN paradigm using adversarial optimization. During each epoch, the generator G and discriminator D were updated alternately: D minimized binary cross-entropy to distinguish real from fake level patches, while G aimed to maximize D's error, effectively generating levels that appeared real to the discriminator.

Optimization was performed using the Adam optimizer with a learning rate of 1e$-$4 and $(\beta_1 = 0.5, \beta_2 = 0.999)$ for both G and D. These values are standard for stabilizing GAN training and reducing mode collapse and worked effectively to generate levels. The lower β_1 encourages the optimizer to adapt faster to gradient changes, which is particularly useful in the adversarial setting.

Critically, generator training also incorporated a custom structural loss to encourage gameplay coherence. The structural loss $\mathcal{L}_{\text{struct}}$ included three components:

- `ground_excess`: penalized over or underuse of ground tiles, ensuring reasonable terrain coverage;
- `stretch_pen`: discouraged large flat sections or overly frequent gaps, balancing level pacing;
- `sky_solid_pen`: penalized solid blocks in the upper sky region, which typically should remain free or sparse.

Each component contributed to the total loss as a weighted sum:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{adv}} + \lambda_{\text{ground}} \cdot \texttt{ground\_excess} + \lambda_{\text{stretch}} \cdot \texttt{stretch\_pen} + \lambda_{\text{sky}} \cdot \texttt{sky\_solid\_pen}$$

Empirical tuning showed the best trade-off between level variety and structure at $\lambda_{\text{ground}} = 1.0$, $\lambda_{\text{stretch}} = 1.0$, and $\lambda_{\text{sky}} = 1.5$. The slightly higher weight on the sky penalty reflects the need to more strongly discourage unnatural block placement in visually sensitive areas of the level which was a prominent issue when initially generating levels. These structural penalties helped enforce global coherence and maintain a playable, human-like feel across the generated levels.

3.4 Post-processing and Level Generation

Post-processing played a crucial role in refining generated levels. Functions such as `carve_random_pits` and `prune_sky` were introduced to further enhance playability. `carve_random_pits` inserted randomized gaps according to preset difficulty parameters, thus controlling the challenge level. `prune_sky` removed inappropriate tiles from upper rows based on probabilistic gradient, ensuring that sky regions were not unrealistically obstructed. Additionally, `generate_multiple_levels` systematically produced multiple levels across different difficulty settings (easy, medium, hard) for easy testing. Implementation details are detailed in notebook cell 7.

3.5 Evaluation and Metrics

Evaluation combined both quantitative and qualitative metrics to assess generated levels. Quantitatively, I calculated tile entropy, linearity, and leniency. Tile entropy measured creative diversity over tile frequency distributions. Linearity captured terrain variation by computing the standard deviation of ground heights across columns. Leniency quantified gameplay difficulty by combining enemy density with the number and average width of pits. These metrics were computed across 30 sampled levels and visualized to analyze the expressive range of the generator's output.

Qualitative visual analyses using heatmaps and expressive range scatter plots provided intuitive evaluations of the generated levels' structural and stylistic coherence. Additionally,

the final stage of evaluation involved generating a comprehensive set of 90 levels—30 each for easy, medium, and hard difficulty settings. These levels were tested for playability using the *robinBaumgarten* agent from the Mario AI Framework. Although I did not develop this agent myself, I selected it specifically for its recognized effectiveness in accurately assessing the practical playability of generated levels, providing an external and objective validation of the GAN's outputs.

4 Results

4.1 Quantitative Results

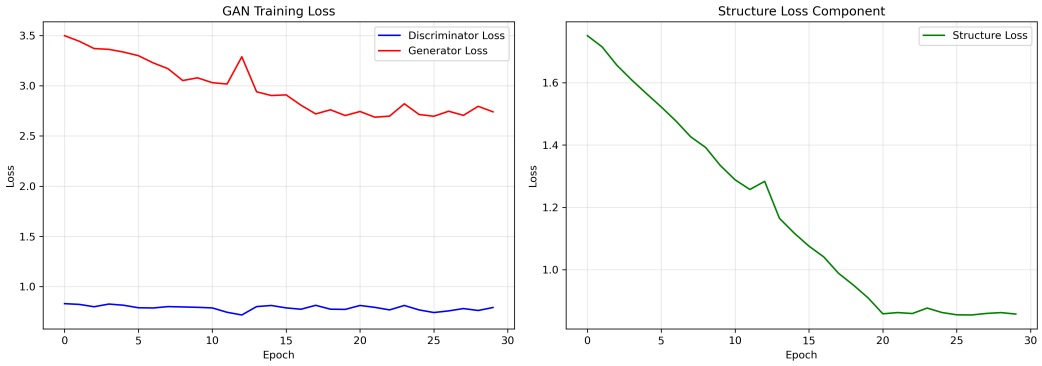

Figure 1: Loss curves for generator, discriminator, and structural losses over 30 epochs.

Figure 1 shows the training losses for both the adversarial and structural components over 30 epochs. The generator loss ($\mathcal{L}_{\text{adv}}$) steadily decreased and plateaued around epoch 20, while the discriminator maintained a relatively stable loss throughout, indicating that neither network collapsed during training. This suggests a healthy adversarial balance, where the generator was learning to produce plausible levels without overpowering or being overpowered by the discriminator.

The structure loss, which enforces gameplay coherence through penalties on terrain and layout irregularities, decreased sharply during the first half of training and began to plateau

around epoch 20. The early and consistent decline in structure loss suggests that the generator quickly learned to follow these constraints, producing levels with reasonable terrain, varied pacing, and realistic spatial structure. Given that both structure and adversarial losses stabilized by epoch 30, I used this as my stopping point to avoid overfitting and reduce unnecessary computation.

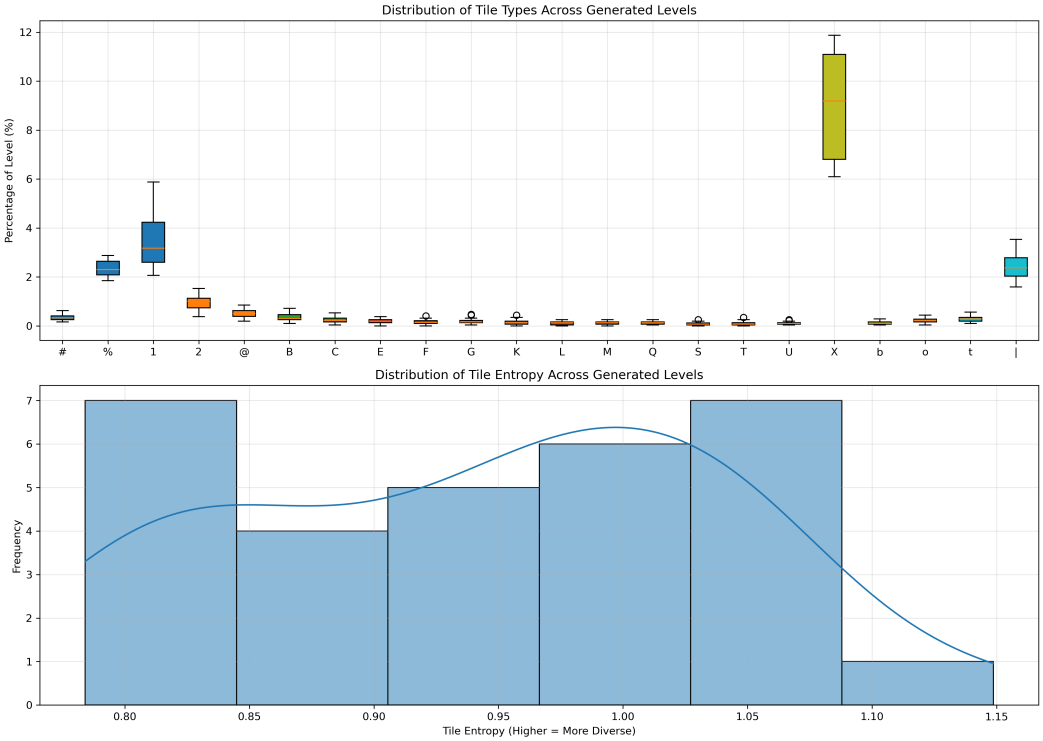

Figure 2: Tile distribution heatmap across generated levels.

Figure 2 provides insight into how the GAN distributed different tile types across generated levels and how diverse each level was in terms of tile usage. In the top plot, the boxplots show the percentage of each tile type across 30 levels. Ground blocks ('X') were the most dominant tile—aside from sky blocks ('-'), which were excluded for readability—typically accounting for 6–12% of a level's composition. This aligns well with original Super Mario Bros levels, where ground tiles form the structural foundation of platformer gameplay. Invisible 1-Up blocks ('1') and jump-through platforms ('%') were the next most common, both of which are also staples in real level designs.

The bottom plot shows the distribution of tile entropy across generated levels. This is a measure of how many distinct tile types appeared and how evenly they were used. Most levels fell between 0.85 and 1.10 in entropy, indicating that while levels were not uniformly random, they exhibited a sufficient degree of creative diversity. This balance mirrors the structure of real Mario levels used during training, which also blend core terrain elements with occasional special blocks and enemies. Overall, these results demonstrate that the

generator not only learned to prioritize foundational game tiles but also introduced enough stylistic variation to keep levels from feeling repetitive or empty.

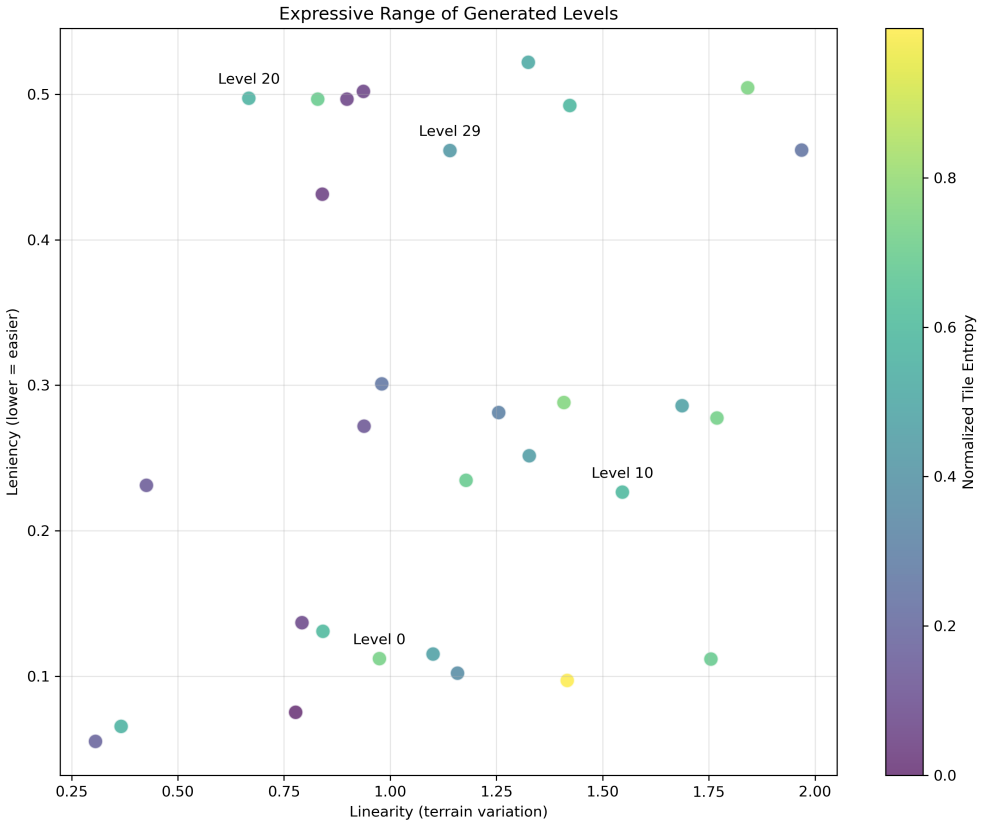

Figure 3: Expressive range scatter plot showing linearity vs. leniency.

The expressive range plot (Figure 3) further supports the diversity of generated levels using two key gameplay metrics: linearity and leniency. Linearity measures the standard deviation of ground height across the level, serving as a metric for terrain variation. Low values imply flat terrain, while higher values indicate more vertical complexity. Leniency estimates difficulty based on a combination of enemy density and the number and size of pits, with lower values corresponding to easier levels.

The scatterplot shows a wide spread of levels across both axes, demonstrating that the generator produced levels with varied terrain structures and gameplay difficulty. Levels with low leniency and high linearity represent harder, more erratic layouts, while levels clustered toward the bottom-left are flatter and easier. The color gradient encodes normalized tile entropy, with brighter points indicating more diverse tile use. Notably, high-entropy levels are scattered throughout the plot, suggesting that stylistic variety does not necessarily correlate with difficulty. Some easy levels are just as visually diverse as harder ones. Together, this indicates the GAN learned to produce structurally distinct levels without collapsing.

4.2 Playability Testing

Playability—defined as the ability to complete a level from start to finish—was evaluated using the *robinBaumgarten* agent from the Mario AI Framework, a well-established benchmark in procedural content generation research. Out of 90 generated levels (30 from each difficulty tier), the agent successfully completed 81, yielding a 90% completion rate. This is a critical, if not the most important, metric in assessing the viability of level generation systems. Regardless of visual diversity or structural coherence, a level that cannot be completed fails as a playable experience.

Importantly, this 90% benchmark likely underestimates true playability. The *robinBaumgarten* agent, while effective, relies on heuristic planning and is known to occasionally fail on levels that a human player would easily complete. This is known to happen in cases with unconventional layouts, minor timing requirements, or sparse cues. As such, the true completability of the generated levels is almost certainly higher. Still, achieving such a strong result from a generator demonstrates that the GAN not only produced visually coherent layouts but also learned fundamental platformer mechanics like traversability, jump timing, and the placement of reachable objectives.

4.3 Qualitative Analysis

The qualitative results, shown in the tile heatmaps (Figure 4), help contextualize the quantitative diversity and structure metrics. The top heatmap shows where non-empty tiles (anything other than air) tend to appear across the generated levels. As expected, the bottom rows are densely populated, reflecting solid ground placement typical of 2D platformer design. The middle and upper areas are comparatively sparse, with occasional block, platform, or enemy placements, indicating a basic grasp of vertical structuring but limited variation in higher elevations.

The ground tile heatmap reinforces this pattern. Most ground tiles cluster along the bottom two rows, forming a mostly continuous base, although occasional breaks show the inclusion pits. While this confirms that levels are playable from a terrain perspective, the shapes and slopes remain somewhat rigid. There's little evidence of the kind of organic hills, ramps, or multi-level structures seen in human-designed Mario levels, implying the generator favors simple, flat terrain with minimal elevation changes.

The enemy heatmap shows sparse and widely distributed enemy placement across the generated levels. This aligns with Super Mario Bro level design, where enemies are introduced gradually and appear intermittently rather than in large clusters. While this sparse placement makes the levels more approachable and helps satisfy leniency-based diversity metrics, it also reveals a limitation in intentional design. The generator includes enemies, but their locations seem to feel random rather than purposeful. There is little evidence of coordinated enemy setups—such as guarding pits, patrolling platforms, or increasing in frequency over time—which are common in human-designed levels to shape difficulty and pacing. So while the GAN avoids overpopulation of enemies, it still struggles to generate encounters that feel strategically placed.

Looking through the raw level outputs confirms this impression. While the levels are undoubtedly playable and visually varied, they often lack the design coherence and purposeful layout of human-made content. Platforms float without clear purpose, enemy density feels

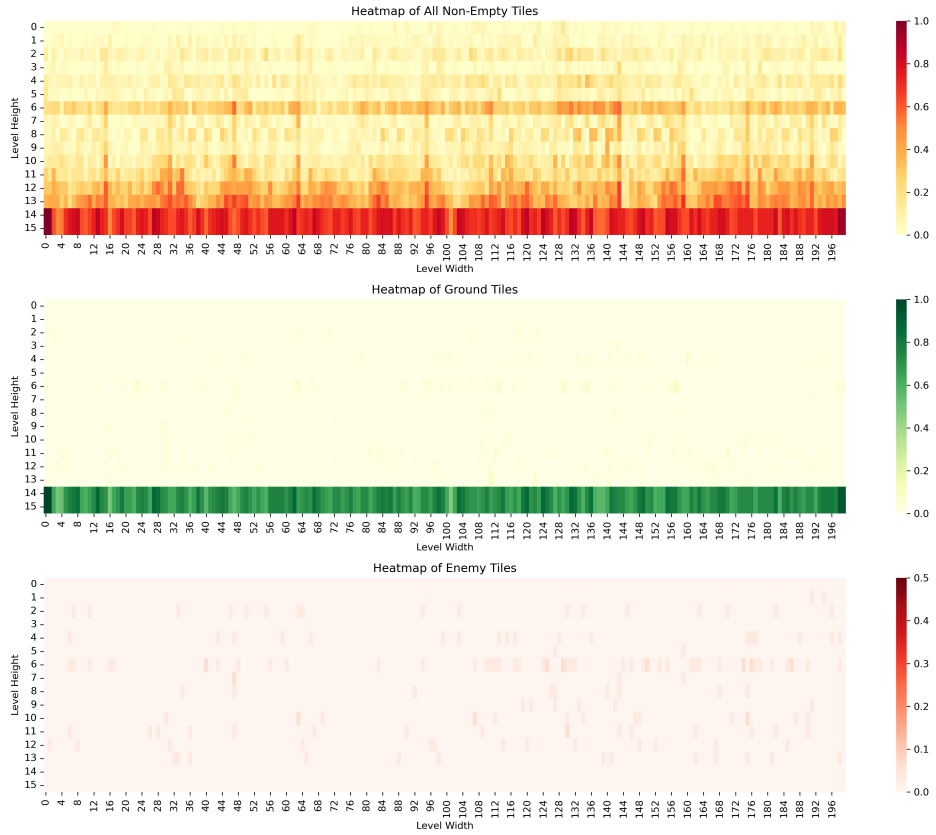

Figure 4: Tile usage heatmaps differentiated by difficulty (easy, medium, hard).

randomized and spatial rhythm, which helps pace in platformers, is somewhat missing. These results point to a core limitation of GAN-based generation, as well as of the current implementation: while quantitative metrics suggest structural success, achieving human-like level design likely requires more nuanced architectural control, refined training strategies, or post-processing techniques beyond what was achieved in this project.

5 Discussion and Reflection

The overall results reveal that it's possible to generate playable Super Mario Bros-style levels using a GAN, especially when combined with structural losses that push the generator toward coherent outputs. The levels weren't just technically functional; they varied in terrain, enemy use, and tile composition, and covered a fairly wide range of difficulty levels. A 90% completion rate by the robinBaumgarten agent supports the idea that most of these levels were playable and respected core gameplay rules.

At the same time, the levels didn't always feel human-designed. While enemies, blocks, and platforms appeared in reasonable places, the arrangement often felt arbitrary. Some sections contained floating blocks with no clear purpose, or enemies placed in isolation.

Other times, chunks with vastly different structures were stitched together, which made the level feel less coherent overall. So while the generator met the structural checklist—ground coverage, some variation, not too many enemies—it often missed the subtler design qualities like pacing, challenge escalation, or surprise. These are harder to quantify but are key to making levels feel engaging and intentional.

5.1 Challenges and Next Steps

A key challenge in this project was the limited variety in the original Mario dataset. With only a small number of human-designed levels to learn from, the generator had a narrow view of what makes a level engaging or well-paced. This likely contributed to repetitive patterns and a lack of higher-level structure in some outputs. Another challenge was getting levels to feel coherent. While structural losses helped with local constraints—like ground coverage and avoiding block spam in the sky—the overall layout often lacked flow. Sections were sometimes stitched together awkwardly, and enemies or platforms appeared without a clear purpose. These issues reflect both the limits of GANs and the difficulty of capturing human design intent using only low-level constraints.

Another large limitation was the inability to integrate agent-based feedback into the training process. Due to constraints with the Harvard-provided Jupyter GPU platform, I wasn't able to run the Mario AI Framework during training, which meant the AI agent could only be used for post-generation evaluation. As a result, the model had no direct signal from gameplay outcomes during learning. In future work, reinforcement learning could be combined with GAN generation, allowing an agent to influence the training process by rewarding levels that are not only playable but strategically interesting or challenging. This could help the model learn beyond surface-level patterns and adapt based on how a level actually plays.

To make the levels feel more realistic and thoughtfully designed, future work could also expand the training data. Adding more diverse or procedurally tweaked levels would give the model more patterns and ideas to learn from. It might also help to try a more advanced model, like a transformer, which could better handle long-range structure and make levels feel more cohesive. Another approach is to let designers guide the process by giving the model goals like "easy," "lots of pits," or "more enemies." Lastly, even further simple rule-based cleanup steps after generation—like fixing awkward transitions or moving enemies into better spots—could make the levels feel less random and more polished without taking away the creativity of the GAN.

6 Reflection

This project was definitely one of the most challenging but rewarding things I've worked on. I'm not the strongest coder, so getting everything to work, from the GAN architecture to the evaluation and visualization, took a lot of trial and error, and more time than I expected. But working with a game I grew up playing made the process feel meaningful and fun, even when things were frustrating. I'm proud of how much effort and time I put in and that I was able to build something that actually works, even if it's far from perfect. It feels good to have pushed through the hard parts and come out with a result I can be proud of.

References

[1] G. Smith, J. Whitehead, and M. Mateas. Tanagra: A Mixed-Initiative Level Design Tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG)*, pages 209–216, 2010. https://dl.acm.org/doi/10.1145/1822348.1822376

[2] R. R. Torrado, A. Khalifa, M. C. Green, N. Justesen, S. Risi, and J. Togelius. Bootstrapping Conditional GANs for Video Game Level Generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019. https://arxiv.org/abs/1910.01603

[3] M. Rajabi, M. Ashtiani, B. Minaei-Bidgoli, and O. Davoodi. A Dynamic Balanced Level Generator for Video Games Based on Deep Convolutional GANs. *Scientia Iranica*, 28(3):1497–1514, 2021. https://scientiairanica.sharif.edu/article_22082.html

[4] D. F. Silva, R. P. Torchelsen, and M. S. Aguiar. Procedural Game Level Generation with GANs: Potential, Weaknesses, and Unresolved Challenges. *Multimedia Tools and Applications*, 2025. https://www.researchgate.net/publication/388162276_Procedural_game_level_generation_with_GANs_potential_weaknesses_and_unresolved_challenges_in_the_literature

[5] X. Mao, W. Yu, K. D. Yamada, and M. R. Zielewski. Procedural Content Generation via Generative Artificial Intelligence. *arXiv preprint arXiv:2407.09013*, 2024. https://arxiv.org/abs/2407.09013

[6] N. Shaker, G. N. Yannakakis, and J. Togelius. Towards Automatic Personalized Content Generation for Platform Games. In *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, pages 63–68, 2010. https://ojs.aaai.org/index.php/AIIDE/article/view/12399