Towards a Celeste AI Framework: Agent-free Automated 2D Level Generation for Multidirectional Platformers

Anonymous Author(s)

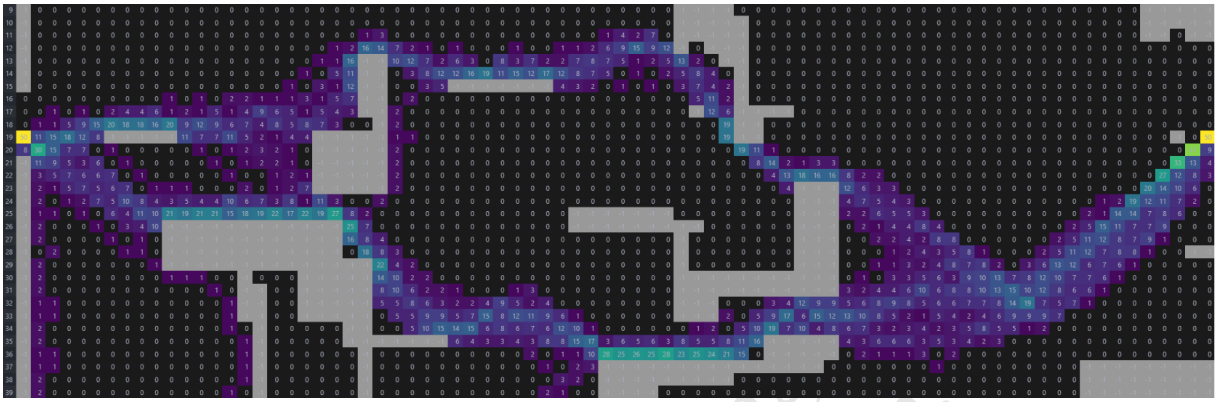


Figure 1: Heatmap fragment of possible paths for clearing a generated Celeste level based on adapted heuristics.

Abstract

We present a procedural content generation (PCG) pipeline for Celeste, a complex 2D platformer (horizontal and vertical movement) with limited prior AI framework development. Our approach utilizes a Markov Chain-based model to capture the game's unique structural and gameplay elements, generating playable levels that adhere to Celeste's design principles. We implemented post-processing steps to enhance playability and strategically place game elements. Our evaluation metrics focused on playability and interestingness, with results indicating success in replicating the desired gameplay experience for beginner players. The evaluation involved 12 players across different skill levels, providing insights into the effectiveness of our generated content. While some limitations were observed, such as occasional lack of creativity and difficulty in controlling challenge levels, our pipeline demonstrates promise as a foundation for a Celeste AI framework. This study contributes to the broader field of PCG for complex platformers and opens avenues for level generation in which agent-based evaluation is not feasible.

CCS Concepts

• **Applied computing** → **Computer games**; • **Theory of computation** → *Random walks and Markov chains*; *Generating random combinatorial structures.*

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

Keywords

Level Generation, Procedural Content Generation, Celeste, Markov Chains, 2D Platformer, AI Framework

ACM Reference Format:

Anonymous Author(s). 2018. Towards a Celeste AI Framework: Agent-free Automated 2D Level Generation for Multidirectional Platformers. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Procedural content generation (PCG) in video games, particularly level generation, has been a significant area of interest for both game development and artificial intelligence research. Approaches for level generation often rely on AI agents modeled after the main character's abilities to evaluate level fitness. Nintendo's *Super Mario* series has become a fertile ground for level generation research. This popularity has spurred the development of Mario AI frameworks [6] and even AI competitions [13, 14] aimed at creating challenging and engaging levels.

While existing frameworks and AI agents, such as A* for *Super Mario*, yield sophisticated results, their applicability to platformers like *Celeste*, with its distinct verticality and gameplay style, remains limited. Unlike *Mario*, *Celeste* lacks a dedicated AI framework capable of navigating its complex levels, highlighting a key gap in current research.

This research explores level generation for *Celeste* and similar 2D platformers that emphasize vertical movement and complex level structures. Our approach extracts structural information from existing *Celeste* levels to generate new, diverse levels.

Celeste is a 2D platformer where players ascend Mount Celeste through intricate levels, employing four core mechanics: moving, jumping, dashing, and grabbing. By combining these mechanics, players execute advanced maneuvers; for example, climbing by moving vertically while grabbing a wall. The game incorporates various interactive entities, both hazardous (such as spikes or monsters) and beneficial (like bouncers or dash refills), which are crucial to each level's identity. In fact, removal of some entities can make a level impossible to complete. Consequently, level design in *Celeste* is not merely about structural elements; it is about striking a balance between structure and entities.

This research project was initiated with the aim of developing a modding tool accessible to the *Celeste* community, enabling players to procedurally generate novel, interesting, and playable levels. The development of a comprehensive workflow capable of generating levels from minimal input parameters presents a significant challenge, particularly given the limited existing resources in this domain, with the exception of a code repository for a visual map editor project [3].

Two major challenges in *Celeste* level generation are:

- Non-linear level progression: Unlike *Mario*, *Celeste* features vertical gameplay, requiring generation of clearable rooms with paths from entrance to exit.
- Lack of AI framework: Without a *Celeste*-playing agent, an agent-free model for level generation and evaluation must be developed.

This study aims to develop a fast and efficient procedural content generation (PCG) pipeline for *Celeste* level generation, using a Markov Chain-based model to capture key structures and gameplay dynamics. The approach prioritizes speed and accessibility, favoring rapid prototyping and potential real-time applications to encourage wider adoption within the modding community.

The central research question is: **How can we generate interesting and playable 2D platforming levels for *Celeste* without an AI agent for evaluation?**

To address this, we developed a configurable PCG pipeline with built-in mechanisms to evaluate levels in the absence of an AI agent. A user study with *Celeste* players of varying experience levels assessed the model's effectiveness, with participants rating generated levels on playability, challenge, and enjoyment. The results showed success in generating functional, playable rooms that offered sufficient challenge, especially for beginner players. However, limitations were noted in creativity, difficulty control, and meeting the expectations of experienced players, highlighting the need for further refinement and more advanced generation techniques.

The following section discusses related work that informed our approach to *Celeste* level generation.

2 Related Work

While no scientific work has been published on procedural level generation for *Celeste*, approaches to Metroidvania-type games offer relevant insights [8]. These games feature interconnected rooms with complex structures, often tied to mission-based gameplay requiring backtracking. Our focus is on methods that emphasize underlying structures in existing levels as foundations for AI-generated rooms. This approach aims to capture the essence of playable levels, facilitating exploration of possible level designs while ensuring clearability.

Procedural Content Generation (PCG) encompasses various techniques for algorithmic game content creation. Our review focuses on grid-based methods, where levels are modeled as matrices of cells or tiles representing terrain or objects, a common approach in 2D platformers. This method allows for efficient capture and replication of level structures, balancing creativity with playability.

The random dungeon generation approach in *Spelunky*, described by [9], utilizes room templates and predefined rules to create grid-based levels. This method ensures playability by generating a solution path and modifying rooms accordingly, providing unique experiences in each playthrough. The concept of creating a high-level theoretical path from start to end inspired our skeleton generation for room assembly. Furthermore, Stammer et al. [18] demonstrated that level generation in *Spelunky* can be tailored to individual playstyles and difficulty preferences, enhancing player engagement.

Markov Chain Based Methods. Markov Chains are useful to model probabilistic transitions between different states and are also used in level generation in 2D video games like *Super Mario* [16]. Levels are then represented as 2D arrays and the probabilities of each tile/entity are computed from a set of existing maps to generate new ones. In another paper from the same authors dealing with Markov Model-based level generation [17], they applied their models to other games such as *Kid Icarus* that present more vertical gameplay and even tackled the playability issue. That method seems very promising for *Celeste* level generation with some caveats: because *Celeste*'s levels gameplay is either horizontal or vertical -and sometimes both-, mixing up all the levels (and therefore vertical and horizontal structures) in the process could harm the performance of the model from a structural point of view. The same thing goes for the difficulty; some rooms are by design more difficult than others, and it might be interesting to split the training rooms by difficulty levels or some other parameter, hence creating several well-performing models rather than one that mixes up everything. This approach definitely inspired the room generator module developed for this study, and will be discussed in 3.3.2.

Grammar-Based Representation. Initially developed for natural language, generative grammars have been adapted to procedural content generation for video games. Graph grammars are adapted to model dungeon levels, where nodes represent rooms and edges represent connections. Adams [1] used this approach to generate FPS levels, focusing on topological control but facing limitations due to hard-coded rules. Dormans [5] extended this by introducing mission grammars to generate adventure game dungeons, adding gameplay-based control. Van der Linden et al. further refined this in [11] with gameplay grammars, allowing designers to create generic graph-based dungeon layouts tied to player actions, demonstrated in *Dwarf Quest*.

Metric-Centered Methods. Some studies focused on trying to generate coherent and playable levels using some smart hand-crafted

metrics [4], where for example *leniency* is illustrating the ratio between platforms and gaps in a *Mario* level, hence a metric related to the difficulty of a level. While this approach alone is incomplete to generate levels, it can be interesting to couple with a level generator model to influence characteristics of the created rooms, as in another study [10], where metrics are used in order to influence gameplay of new levels.

Hybrid Evolutionary Approaches. Recent studies on level generation have explored advanced techniques combining deep learning and evolutionary algorithms. Volz et al. [22] used GANs with evolutionary algorithms to generate *Super Mario* levels, while Thakkar et al. [20] applied autoencoders and evolutionary algorithms to *Lode Runner* levels. Another approach [2] utilized genetic algorithms for 2D platformer level generation, demonstrating the potential of evolutionary methods in creating engaging and varied levels. These studies highlight the effectiveness of integrating machine learning and evolutionary computation for procedural content generation in game design.

Approach Design Based on Prior Research. Our level generation approach for *Celeste* favors supervised techniques using existing game levels as training data. We opt for Markov chain-based methods for their efficiency in learning and replicating grid-based level structures. This choice offers rapid learning and generation, captures essential structural elements and gameplay dynamics, and allows easy fine-tuning for varying difficulty or style. The simplicity of Markov chain implementations aligns with our goal of creating an accessible, modifiable tool, while their compatibility with grid-based representations ensures seamless integration with *Celeste*'s existing systems. This approach enables us to develop a PCG pipeline that generates interesting, playable levels while meeting our requirements for speed, efficiency, and adaptability, positioning us to create a robust and versatile level generation tool for *Celeste*.

3 Method

To address our research question, we implement a Markov chain-based model, adapt existing level manipulation tools, create a two-step PCG pipeline, and conduct a user study to evaluate the generated content. This section outlines our approach, beginning with an overview of the model and workflow, followed by our use of an existing map editor. We then describe the key modules developed, including the Room and Celeskeleton classes for level structure generation and the Markov chain-based model for room content generation. Finally, we discuss the user study conducted to assess the effectiveness of our generated content.

3.1 Model and Workflow

This section outlines our model and workflow, adapting Markov chains to capture and replicate *Celeste*'s room and level structures. Our approach addresses the game's unique challenges while maintaining efficiency and adaptability. We generate playable levels by integrating procedurally created rooms within a level skeleton, using grid representation for efficiency [19].

A key challenge is that level data doesn't explicitly indicate room exits; instead, they're determined by room arrangement. We address this by first generating a level's high-level *skeleton*, inspired by *Spelunky*'s level generation approach [9]. This ensures exit placement at a global level before individual room generation.

Our top-down strategy creates the level skeleton first, then procedurally generates individual rooms to fill it. This method leverages all available information and extracts low-level features, offering advantages over whole-level generation approaches. Structurally, levels consist of interconnected rooms linked by exits, with defined starting and ending points. Level generation is a two-step process:

(1) General Level Structure: Placing empty rooms in 2D space and ensuring a path connects both ends of a level.
(2) Room Generation: Using a Markov chain-based model for generating rooms with concrete, structured content and entities, used to fill the empty rooms created in the previous step.

While this two-step generation doesn't guarantee playability for each room, it produces a set of interconnected rooms filled with foreground tiles and entities, arranged in 2D space without overlap. This output meets the necessary and sufficient conditions for conversion into a playable binary file.

3.2 Leveraging Existing Map Editors

The active *Celeste* modding community has developed a visual map editor based on the Julia repository *Maple* [3]. This wrapper code reads binary files containing original game levels and has been adapted for this project to generate rooms and levels programmatically. In *Celeste*, levels are defined as sets of rooms assembled in 2D space, with each room containing structural details including foreground tiles, entities, background elements, metadata, and decorations:

- Foreground: the set of tiles which composes the ground, ceiling, and walls of a room
- Entities: all additional items that the player can interact with, can be helpful, neutral, or even deadly depending on the entity
- Background: decorative tiles that are in the background, only aesthetic and do not interact with gameplay elements
- Metadata: special settings of a room, such as wind (alters gameplay) or a specific music track (does not impact gameplay but can be chosen accordingly with level difficulty for example)
- Decoration: all elements or *decals* contributing to the visual aesthetics of the game

The scope of this project encompasses the generation of foregrounds and sets of entities that are well-structured and coherent in their interaction regarding playability. These elements constitute the fundamental requirements for room generation in *Celeste*. Backgrounds, metadata, and decorations could be incorporated at a future stage and do not present significant challenges once a functional pipeline is established.

3.3 Modules

3.3.1 Room & Celeskeleton. Our two-step process begins with generating a level's global structure, or skeleton, which must meet specific criteria for viability. While this approach offers diversity, it presents a challenge: How do we assemble generated rooms into a cohesive level that aligns with sound game design principles?

In encoding a level for *Celeste*, each room's coordinates must be specified for correct 2D placement. Rooms are essentially rectangles defined by four key attributes:

- Origin: coordinates of the bottom left corner of a room.
- Size: width and height of a room.
- Data: 2D array with symbols representing all tile and entity positions. A room is said to be *empty* if this array is full of zeros.
- Exits: 4-key dictionary (one key per side) indicating the coordinates of the different exits.

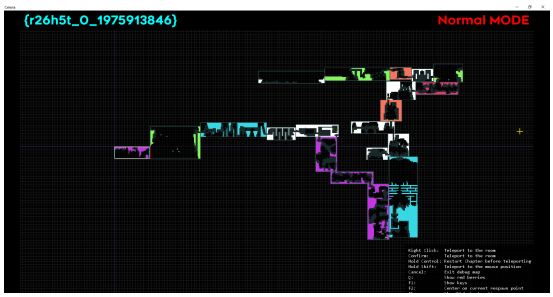


Figure 2: Example of a level seen in debug mode

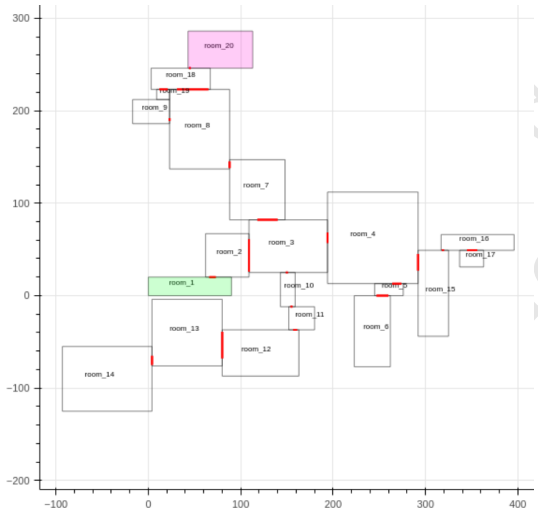


Figure 3: Example of a Celeskeleton object generated with 20 rooms

For the assembly of rooms, some requirements must be met:

(1) Two rooms are *connected* if they have adjacent exits.
(2) Each room needs to be connected to at least one room.
(3) The rooms must not overlap.

The *Celeskeleton* initially consists of a collection of empty *Room* objects with indicated starting and ending rooms. To streamline the process of arranging rooms into playable levels, as described below, we drew inspiration from examining existing levels in debug mode (see Fig. 2 and Fig. 3).

Algorithm 1 Celeskeleton generation algorithm

```
Initialize Celeskeleton object
Set initial room size (from input or random)
Initialize empty Room object and add first room to skeleton
while number of rooms in Celeskeleton < nb_rooms do
    Generate new empty Room
    Choose connection side and choose Room to connect with (with probabilities p1, p2)
    Attempt to connect and place new room
    if no overlap then
        Add Room to Celeskeleton
    end if
end while
Set start and end rooms
Return the skeleton
```

3.3.2 Markov Chains-based Model. As previously mentioned, our model's design draws inspiration from earlier work [16]. Given the need to construct the entire AI framework for *Celeste*, we aimed for a PCG core model that is both straightforward and adaptable.

Multi-dimensional Markov Chains (MdMC). To clarify how the procedural generation works, we start with a short mathematical parenthesis about Markov Chains (MC). A MC is a stochastic process that transitions from one state to another within a finite or countably infinite set of states; for this study, the set of states is the set of possible tiles and entities in the grid representation of *Celeste* rooms and is hence finite. The key property of a first-order MC is that the probability of transitioning to the next state depends only on the current state and not on the sequence of events that preceded it.

Let $\mathcal{S} = \{S_i\}_{i=1}^{N}$ be a finite set of states, and P be conditional probability distribution such that $P(S_t|S_{t-1})$ represents the probability of transitioning to a state S_t given that the previous state was S_{t-1}.

The Markov property for first-order MC can be translated as:

$$P(S_t|S_{t-1}, S_{t-2}, \ldots, S_0) = P(S_t|S_{t-1})$$

However, higher-order MC are definitely essential to the model we use. Let r be a strictly positive integer. An MC of order r therefore takes into consideration r previous states:

$$P(S_t|S_{t-1}, S_{t-2}, \ldots, S_0) = P(S_t|S_{t-1}, \ldots, S_{t-r})$$

To generalize the basic concept of Markov Chains to a 2D-space, let $R = \{R_{i,j}\}_{(i,j)\in\{1,\ldots,N\}\times\{1,\ldots,M\}}$ be a 2D room of shape (N, M), and $\mathcal{S} = \{S_i\}_{i=1}^{T}$ be the set of the possible states, accounting for the T possible types of tiles or entities one can find in *Celeste*. For any pair of coordinates (i, j), the tile $R_{i,j}$ is in a given state S_k, where $k \in \{1, \ldots, T\}$. Simply put, a room is represented by a 2D-array of given states among all the existing tiles and entities within *Celeste*.

Learning with MdMC. Consider a MdMC representing the probability of a tile in a room according to the surrounding tiles. This set of tiles used for learning probabilities will be called a *configuration* and can be represented using a 3×3 configuration matrix C:

$$C = \begin{bmatrix} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & 2 \end{bmatrix}$$

where $C_{22} = 2$ is by default representing a tile $R_{x,y}$ to learn, and for all other (i, j) pairs in C, $C_{ij} = 1$ means that the probability of the tile $R_{x,y}$ depends on the tile $R_{x+i-2,y+j-2}$; else $C_{ij} = 0$. Let's consider an example. The configuration matrix

$$C_{000011012} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

considers the tiles on top, on left, and on top-left of the tile considered. Each configuration will be identified by a unique 9-digits sequence corresponding to the values of its matrix read line-by-line. The example above represents the 000011012 configuration.

For the sake of clarity, we will introduce the concept of *adjacent* tiles. For example, we consider the configuration 001001112 whose matrix is given by:

$$C_{001001112} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

and a example 5×5 room given by:

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & D & D \\ 1 & 0 & 0 & D & D \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

For a given position (x, y) in the room R, the tiles of interest given by $C_{001001112}$, the *adjacent* tiles, are the two tiles above, and the two tiles on the left of $R_{x,y}$. These adjacent tiles, being read from left to right, then from top to bottom, constitute a n-gram that will prove useful for practical training of this MdMC-based model. Therefore, if we look at $R_{4,5} = D$, the adjacent tiles are represented by the n-gram made of $(R_{2,5}, R_{3,5}, R_{4,3}, R_{4,4}) = 0D0D$.

For the learning process: let C be a configuration matrix, and $\mathcal{R}$ a dataset of rooms R that are considered for the training. The MdMC model is learned in two steps:

- The method begins by counting the occurrences of each tile type w.r.t. the adjacent tiles in the whole training dataset, with adjacent tiles being defined by the configuration matrix C. These counts are called *absolute counts*, and are defined for each tile type S_i by
$$N(S_i|S_{i-1}, \ldots, S_{i-t})$$
where t represents the number of 1's in C.
- Using the absolute counts, *transition probabilities* are calculated. These probabilities represent the likelihood of a tile type following a specific arrangement of adjacent tiles, the tiles considered for this arrangement being once again defined by the chosen configuration C. The transition probability of a tile type S_i is computed as:
$$P(S_i|S_{i-1}, \ldots, S_{i-t}) = \frac{N(S_i|S_{i-1}, \ldots, S_{i-t})}{\sum_j N(S_j|S_{j-1}, \ldots, S_{j-t})}$$

In practice, we coded a function which, for a given configuration and a given training dataset, builds a dictionary whose keys are all the possible n-grams of adjacent tiles encountered in the training set, and whose values are also dictionaries, where this time the key/value pairs are given by all the possible tile types and their associated transition probability computed following the above algorithm. Training a model is therefore equivalent to building this dictionary of probability transitions, and depends of course on the choice of both the configuration and the set of rooms considered for the training.

Room generation. Once the model has learned a given MdMC for a pair configuration/training dataset, meaning that a dictionary of probability transitions (DPT) has been built, we create an room array full of zeros R that are updated using an iterative process, starting from the top-left tile $R_{2,2}$ (accounting for the size of the configuration matrix), the following steps are done for every single tile, from left to right, and then top to bottom:

- Extract the adjacent tiles n-gram
- Select the corresponding probability distribution in the DPT
- Pick the next tile randomly among this probability distribution

However, a significant challenge we encountered using this process in [16] was the handling of unseen states, which prevented this method from being used in-game as the PCG model would eventually fail. We adapted this process to ensure room generation would proceed; if the model encounters an unseen state (i.e., a n-gram not encountered during the training step), we introduced a backtracking mechanism. The generation process reverts and attempts to generate another tile until the updated n-gram is recognized. If all tile types have been unsuccessfully tried, the process backtracks further until one of two outcomes is reached:

- A combination of tiles is eventually found that allows the model to overcome the unseen state, enabling the generation process to resume normally.
- If no possible combination of tiles can avoid an unseen state and the model reaches the maximum backtracking depth (which we set to prevent excessively time-consuming room generation), we resolve the issue by assigning a random tile type.

While random generation alone could theoretically resolve unseen states, our tests showed this often leads to cascading failures. Combining random generation with backtracking allows our model to generate complete rooms without producing random, unrealistic patterns.

3.4 User Study

To validate our ad-hoc evaluation against user experience, we conducted a study with 12 players of varying skill levels. Participants played 4 rooms with extreme interestingness scores (defined in section 4.2) and rated their engagement on a scale of 1-10. The sample included both advanced players (intermediate and expert) and beginners who had never played *Celeste* . Results are discussed in section 4.2.2.

4 Evaluation of Generated Rooms

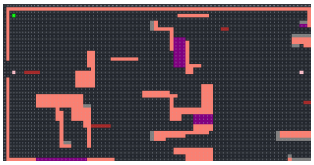
Robust evaluation tools are essential for effective PCG systems in games like *Celeste*. These tools facilitate the fine-tuning of control parameters, model optimization, and understanding of how various factors influence the generated content. Our current PCG

pipeline, while capable of generating parametrizable levels, lacks a comprehensive understanding of how control parameters affect output quality and player engagement. To address this, we propose three key metrics for evaluating generated rooms: playability, interestingness, and difficulty. These metrics guide the fine-tuning process, ensuring that generated rooms are functional, engaging, and appropriately challenging. In the following subsections, we define these metrics mathematically, discuss their importance for player experience, and present experiments evaluating our PCG model's performance in these areas.

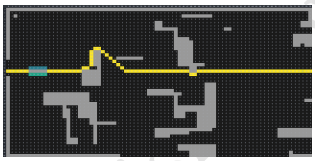
4.1 Playability

Our playability metric focuses on evaluating whether a room's exits are connected by a viable path. We consider rooms as 2D arrays with exactly two exit points. The goal is to determine whether a path exists between these exits, adhering to the physics and movement constraints of the game.

To achieve this, we use an adapted A* pathfinding algorithm, tailored specifically for platformer games. Standard A* algorithms, while efficient for general pathfinding tasks, do not account for the unique physics dynamics of games like *Celeste*, where gravity and player movement significantly influence gameplay. Traditional A* may find paths that are technically valid but impractical due to the game's specific physics constraints, such as gravity affecting jump trajectories and platform interactions as shown in Fig. 4b.



(a) Example of room used for A* refinement



(b) Heatmap of paths found, base A* - 50 runs

Figure 4: Base version of A* - no directional weighting and no custom heuristic

While functional, this algorithm is suboptimal for playability assessment due to the unrealistic nature of the paths it generates. To address these challenges in our custom A* implementation, we incorporated game-specific adaptations. First, we introduced randomness in neighbor selection by shuffling potential next steps, encouraging diverse pathfinding outcomes and mimicking the exploratory nature of gameplay. Additionally, we assigned varying weights to movement directions to reflect the game's gravity. For example, movements that go against gravity are penalized more heavily, favoring paths that align with the natural gameplay mechanics.

We then added these weights to the heuristic, resulting in significantly more diverse path generation, as illustrated in Fig. 5. However, this approach is not entirely satisfactory; the variance is now too high, and some paths do not align well with the *Celeste* gameplay mechanics.

To better align paths with gameplay mechanics, we refined our approach by incorporating a penalty score into the heuristic. This

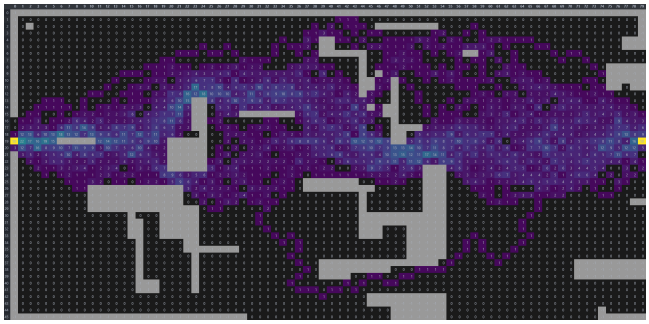


Figure 5: Heatmap of paths found, A* with weighted directions - 50 runs

score is based on the distance to the nearest non-lethal entity (NLE); a tile that a player could feasibly reach. In platformers like *Celeste*, players typically navigate from one platform to another, occasionally using intermediary entities for assistance. A gameplay-accurate path should, therefore, remain close to the available NLEs within a room.

Fig. 6 illustrates the results of this final adaptation. We find these outcomes highly satisfactory. Our adapted A* version strikes an ideal balance, generating paths with appropriate variance while maintaining player feasibility. This blend of diversity and practicality significantly enhances the authenticity of our playability assessment.

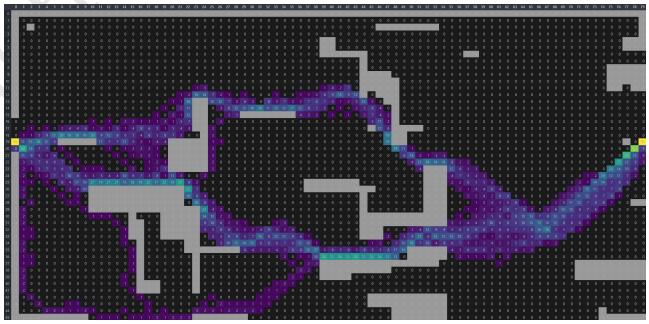


Figure 6: Heatmap of paths found, A* with weighted directions and adapted heuristic - 50 runs

Our adaptation of the A* heuristics aims to generate more realistic paths that are feasible within the game's physics constraints. However, this approach can be quite demanding in terms of both memory and time. We need to strike a balance in the weights and heuristics we choose, as suggested by [7] in their review of pathfinding algorithms for platformers. Since our project is intended to be a real-time module for *Celeste*, we aim to keep the computing time as reasonable as possible. Fig. 7 demonstrates that modifications to the pathfinding algorithm lead to exponential increases in computing time; an effect further amplified by the size of the investigated room.

We compared base A* and our tailored *Celeste*-A* on 100 rooms of sizes 40×23 and 120×69. Despite higher computational intensity

Algorithm 2 A* Pathfinding Algorithm adapted to *Celeste*

```
Initialize start and end nodes
Create open_list (priority queue) and closed_list
Add start node to open_list
Define movement options and costs
while open_list is not empty do
    Remove node with lowest f from open_list (current_node)
    Add current_node to closed_list
    if current_node is the end node then
        Return path from start to end
    end if
    for each adjacent node do
        if node is within bounds then
            Calculate costs and heuristic for this node
            if node is not in closed_list or has a better path then
                Add node to open_list
            end if
        end if
    end for
    if iteration limit exceeded then
        Return None (no path is found)
    end if
end while
Return None (no path is found)
```

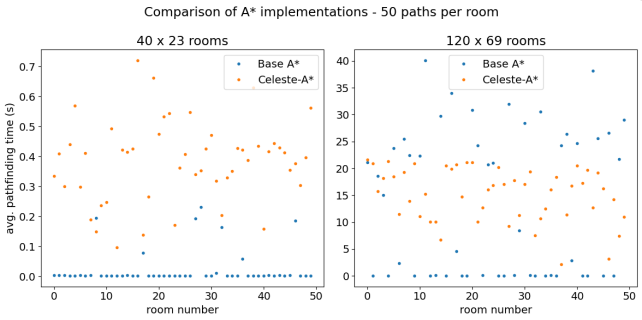


Figure 7: Running time of pathfinding algorithms and effect of the room size

per iteration, *Celeste*-A* often required fewer iterations due to its more sophisticated heuristic. This effect was pronounced in larger rooms, where *Celeste*-A* sometimes outperformed base A*, which frequently reached iteration limits in complex rooms.

Fig. 7 demonstrates this, particularly for 120 × 69 rooms. Room 23 exemplifies this trend, with explicit values shown in Table 1. High generation times for both algorithms typically indicate unplayable rooms. However, *Celeste*-A* occasionally found paths in complex rooms where base A* failed, while base A* remained adequate for simpler geometries, albeit with less realistic paths.

Beyond improving room evaluation, our gameplay-accurate paths demonstrate that the PCG generator effectively captures the global structure of a room, producing coherent layouts from a player's perspective. This is evidenced by the enhanced pathfinding performance when incorporating Celeste gameplay elements into the heuristic.

Room	Algorithm	avg. nb_iter	avg. time (s)
23	A*	30420	329.4
23	*Celeste*-A*	7384	15.3

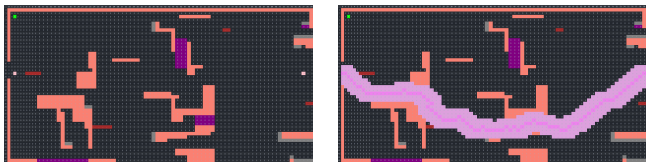
Table 1: Comparison of base A* and *Celeste*-A* performances on a complex room - average on 50 runs

The pathfinding algorithm's output not only finds a path but ensures it aligns with gameplay realism. While the absence of a path strongly indicates an unplayable room, finding a path doesn't guarantee playability. However, paths respecting the game's gravity and movement constraints are more indicative of playable rooms. These paths also serve to compute metrics like interestingness and difficulty, representing areas a player is likely to traverse. To further assess playability, we introduce a path evaluation metric based on proximity to Non-Lethal Entities (NLEs):

$$s_{path} = \frac{1}{N} \sum_{i=1}^{N} Distance_i$$

where $Distance_i$ is the distance from point i on the path to the nearest NLE, and N is the total path length. This metric quantifies how well-supported a path is by nearby NLEs, crucial for player progress. A lower score indicates better playability, while a higher score suggests a less supported, potentially less playable path.

We also use paths to evaluate metrics beyond playability. We define an *area of interest* (AOI) comprising all tiles from the path, plus n tiles above and below each path tile. This AOI_n estimates the area a player will likely visit when crossing the room. It serves as a mask to extract relevant entities and features for further evaluation in subsequent sections.



(a) Example of room used for reference **(b) Same room with path and AOI displayed**

Figure 8: Example of an AOI_n with $n = 2$

In summary, our playability metric seamlessly integrates advanced pathfinding techniques with game-specific adaptations to evaluate room connectivity and environmental support. By combining these methods and assessing paths relative to Non-Lethal Entity (NLE) proximity, we've developed a robust framework. This approach ensures that generated rooms are not only navigable but also consistent with the gameplay experience of *Celeste*; striking a balance between feasibility and authenticity.

4.1.1 Optimizing Playability. In evaluating the playability of procedurally generated rooms in *Celeste*, we aim to identify key parameters that significantly influence this aspect. By understanding these factors, we can refine our generation process to ensure rooms are not only aesthetically pleasing but also functional and engaging. Our goal is to determine parameters that consistently produce baseline rooms suitable for further interestingness and complexity evaluation. We assess playability across four critical parameters:

- Configuration: The matrix used to train the MdMC model
- Training Set: The source data for the Markov Chain model, as its diversity and structures affect generated rooms
- Backtracking Depth: To balance stable generation with exploration of unseen states
- Room Size: As observed in Fig. 7, size may impact player navigability due to increased structures and challenges

Configuration. We evaluated various configurations using a 3×3 matrix for playability (previously not assessed in [16]). Our hypothesis was that configurations not considering all three tiles around the tile of interest would lack coherence in generation. For instance, configuration 000000012, which only considers one tile on the left, was expected to produce chaotic, unplayable rooms. Conversely, configurations considering too many tiles were anticipated to be overly restrictive, leading to numerous unseen states and resulting in near-random generation with few playable rooms.

To test this, we generated 100 rooms of two different sizes for each configuration. Given the non-deterministic nature of *Celeste*-A*, we ran the pathfinding algorithm 10 times per room, considering a room playable if at least one attempt succeeded in finding a path. We find that configuration 0000010112 significantly outperformed others, validating our intuition. Consequently, we used this configuration for subsequent experiments. Further examination of generated rooms confirmed our hypothesis: most rooms generated with other configurations contained randomly distributed symbols -degeneration-, supporting our initial assumptions about coherence and playability.

Training Set. As previously mentioned, we grouped data from *Celeste* by levels. Each training set corresponds to a specific level, with all 9 levels of *Celeste* comprising nearly 800 rooms in total. We hypothesized that the model would perform better when learning from subsets of the dataset rather than the entire collection. This assumption stems from the observable continuity in room design within each level, where rooms share similar structures and gameplay elements. While combining certain levels could yield interesting results, we believe training the MdMC model on the entire dataset would be suboptimal for two main reasons:

- Entity dilution: Some entities, like dream blocks, appear only in specific *Celeste* levels. Training on all levels would significantly reduce the probability of generating these rare entities, limiting the diversity of generated rooms. We propose training several models, using a different one for each room to maintain this diversity.
- Design variations: Levels in *Celeste* vary greatly in design, from pathway-like structures reminiscent of *Super Mario Bros.* to vertically-progressing levels, and even labyrinth-like designs combining vertical and horizontal gameplay. These structural differences could potentially confuse the pattern learning process if mixed indiscriminately.

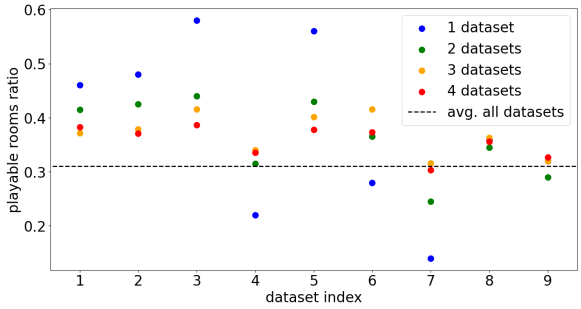


Figure 9: Impact of levels as training datasets in the playable rooms ratio - 10 *Celeste*-A* iterations - 40x23 rooms - playability of model trained on all datasets: 0.31

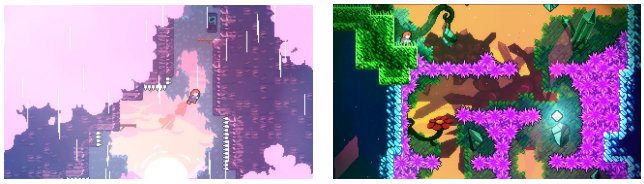
To evaluate training sets, we trained models for various combinations, generating 50 rooms per model. We ran *Celeste*-A* 10 times per room, considering a room *playable* if at least one path was found. We chose a room size of 40×23, as it's the most common format in the game and the largest size displayable at once.

We considered individual levels, as well as 2, 3, and 4-level combinations for training sets. For clarity, a training set using rooms from levels 2, 3, and 8 is denoted as 238. We also trained a model on the global dataset for comparison. In total, we evaluated 256 training sets, resulting in 128,000 *Celeste*-A* runs. Each of the 256 trained models was evaluated on 50 rooms with 10 paths each, yielding 256 playable room ratios. To present these results concisely, we grouped them by levels, as shown in Fig. 9.

In Fig. 9, blue dots represent the performance of models trained on individual levels. For other colors, we use the following methodology: Let $n \in \{2, 3, 4\}$ be the number of datasets in a group. For each level $k \in \{1, \ldots, 9\}$, we compute the average playable room ratio for all n-dataset models including k. For instance, the green dot for level 2 is the average playability of models trained on datasets 12, 23, 24, 25, 26, 27, 28, and 29. This visualization shows each level's average contribution when combined with other datasets.

Models trained on individual levels generally perform well in terms of playability, with some expected exceptions:

- Levels 4 and 7 exhibit poor playability due to their emphasis on vertical gameplay (cf. Fig. 10a) and moving entities, which are incompatible with our grid representation. The learned structures from these levels do not translate well to horizontal gameplay.
- Level 6 presents a unique challenge with its complex room designs as displayed in Fig. 10b. The abundance of precisely placed spikes, often floating rather than wall-attached, biases our model towards generating less playable rooms with an overabundance of walls and spikes.



(a) Level seven verticality (b) Celeste spiky sixth level

Figure 10: Examples of rooms that are expected to harm the generator performance

Barring these single-level exceptions, our findings indicate that the model trained on the entire 9-level set displays a lower overall playability, of approximately 0.31, suggesting that models trained on smaller, coherent subsets perform better. Models 1 and 2 achieve noteworthy results with playability near 0.5, while models 3 and 5 exceed 0.55.

We observed that combining training sets slightly decreases the playability ratio. This effect is particularly pronounced in "undesirable" combinations like model 57, which achieves a low playability rate of 0.28 due to the contrasting designs of the two levels (although model 7 alone achieves only 0.14). However, these combinations also introduce diversity in the generated tiles. For instance, model 25 allows dream blocks (specific to level 2) to be associated with red boosters (specific to level 5), creating novel gameplay elements. We believe that sacrificing a small degree of playability for increased diversity and originality is a worthwhile trade-off in our procedural generation approach.

We also examined the path evaluation metric defined earlier, which measures the average distance of a path to the closest Non-Lethal Entity (NLE). Fig. 11 illustrates these results, using the same grouping method as in Fig. 9.

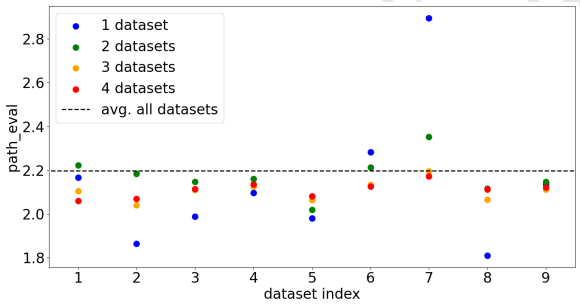


Figure 11: Impact of levels as training datasets in the average path evaluation - 50 rooms, 10 *Celeste*-A* iterations - 40x23 rooms - average on all found paths

Our analysis reveals that training set 2 is particularly promising. Not only does it maintain a satisfactory playability ratio, but it also yields one of the lowest average path scores. This indicates that rooms generated using this model are well-supported by NLEs, providing safe spaces for player progression. Consequently, we selected this training set for our backtracking and room size experiments. Model 2 consistently produces simple yet coherent rooms, making it an ideal baseline for investigating the impact of other parameters.

Backtracking Depth & Room Size. Our investigation of the impact of backtracking on room generation revealed limited influence. However, for larger rooms, the absence of backtracking becomes more significant. Degeneration, i.e., completely random tile generation, occurs more frequently, potentially preventing path creation if it occurs early in the generation process. We concluded that backtracking for generating coherent and playable rooms was definitely suitable, though no optimal value was clearly identified. We recommend maintaining a backtracking depth of 2 to allow for generation variety while maintaining a reasonable computational time.

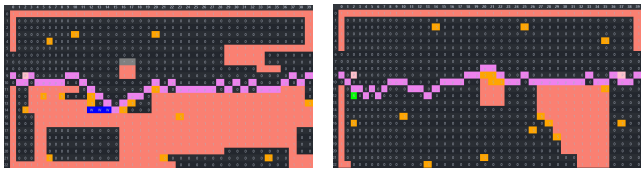
Interestingly, we noticed higher playability for larger rooms. Further investigation indicates that playability increases with room size up to a certain point. Larger rooms offer more potential paths, making it harder to prevent their existence. However, as room size increases, the *Celeste*-A* algorithm's runtime grows significantly (see Fig. 7). What we thought to be a decrease in playability for *very large* rooms (e.g., 120×69) is likely due to algorithmic limitations rather than inherent room properties. With a larger computational budget, many of these rooms would likely be classified as playable.

4.1.2 Discussion on Playability. We found a setup that aligns with our objectives for this section. By training our PCG model using configuration 000011012 and training set 2, we achieved a playability ratio of 0.76 for 80×46 rooms and 0.64 for 40×23 rooms, using a backtracking depth of 2. While these results appear promising, we must clarify some important points.

Although a 76% playability ratio seems excellent, it's somewhat misleading due to our definition of playability. For these rooms, our computed number represents an *upper bound* of the *true playability*. We adapted *Celeste*-A*'s behavior using a hand-crafted heuristic and directional weighting to improve path accuracy. However, paths may still be found in rooms that are unfeasible from a gameplay perspective, as long as the exits are not theoretically disconnected. For very large rooms, the limitations stem from the budget allocated to the pathfinding algorithm, making it impossible to confirm whether the computed playability is truly an upper bound.

Determining True Playability. Accurately assessing true playability remains a challenge without an AI agent capable of test-playing generated rooms. While path evaluation provides a starting point, our current implementation falls short. The path scoring in Fig. 11, which only considers the average distance from the path to the closest NLEs, proved insufficient for establishing a reliable rule of thumb for path viability.

To address this limitation, we introduced an additional metric to our path evaluation: variance. By considering both the mean and variance of the path's distance to NLEs, we gain deeper insights into path safety and support distribution. A small variance paired with a reasonable average distance indicates a consistently well-supported path, while a large variance often reveals problematic zones lacking adequate support. This approach provides a more nuanced assessment of path viability, improving our ability to generate playable rooms.



(a) Path is consistently supported - mean: 1.23 / **var:** 0.22

(b) Path is not consistently supported - mean: 3.0 / **var:** 5.25

Figure 12: Example of extreme variance paths found during the training of single level models - median path eval: 2.00 / **median path var:** 1.65

While extreme variance cases are straightforward to assess, paths with variance near the median pose challenges. Edge cases, such as a floor full of spikes, may yield good path-evaluation metrics despite being unplayable. In our project, aimed at generating levels for actual players, we prioritize avoiding false positives (classifying unplayable rooms as playable) over false negatives. Currently, we exclude rooms with variance above twice the median as a rule of thumb, but this approach emphasizes recall over precision, which is not ideal.

Defining these upper bounds is a promising start, but we aim to extend this work by incorporating advanced AI/ML techniques. We envision developing and training a classifier to assess room playability, with a focus on precision to eliminate false positives. Deep learning approaches could prove more feasible than developing a *Celeste* AI agent capable of room playability characterization. While an AI agent would effectively prevent false positives, it would require significant development to complete interesting and challenging rooms. This avenue represents a separate project in itself.

4.2 Interestingness

In the context of procedural content generation for 2D platformers like *Celeste*, we propose an interestingness metric as a measure for evaluating generated levels. This metric quantifies the engagement and variety within a room, indicating its potential to captivate and challenge players. Our interestingness metric primarily considers two key factors: the density and diversity of Non-Lethal Entities (NLEs). We define the interestingness I of a level as:

$$I = w_1 \times d_{NLE,global} + w_2 \times d_{NLE,local} + w_3 \times s_{diversity}$$

- w_1, w_2, and w_3 are tunable weights.
- $d_{NLE,global}$ is the global NLE density: $\frac{N_{total}}{A}$, with N_{total} being the total number of NLEs and A the room area.
- $d_{NLE,local}$ is the local NLE density: $\frac{N_{AOI}}{AOI}$, where $AOI = AOI_2$ (cf. Fig. 8).
- $s_{diversity}$ is the NLE diversity score, computed using Shannon entropy: $-\sum_i p_i \cdot \log(p_i)$, where p_i is the proportion of NLE i in the room.

The global NLE density forms the foundation of our metric, evaluating the overall concentration of interactive elements in the room. A high global density indicates a rich environment with numerous potential interactions, encouraging exploration and diverse gameplay opportunities. Complementing this, the local NLE density focuses on areas where players are likely to spend more time, ensuring that key gameplay zones are engaging and impactful.

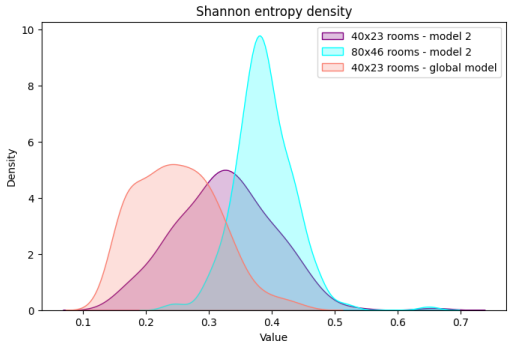


Figure 13: A peak into Shannon entropy to measure entity diversity - 200 rooms per model

We enhance the metric with a diversity score, measuring the variety of NLE types present. This diversity is crucial for maintaining player interest and ensuring dynamic gameplay. These components collectively make our interestingness metric a comprehensive tool for assessing room quality, guiding our PCG system towards generating rooms that are both playable and enjoyable.

Our approach aligns with previous research in PCG for video games, balancing content richness and player engagement. For instance, prior work [15] explored similar metrics in *Super Mario*, assessing the impact of content variety and placement on player satisfaction. Additionally, Togelius et al. [21] emphasized the importance of diversity in PCG systems for sustaining long-term player interest. By grounding our metric in these principles, we ensure that our evaluation framework reflects established best practices while adapting to the specific challenges and opportunities presented by *Celeste*.

4.2.1 Experiments. We began by analyzing the diversity score using Shannon's entropy, a previously utilized technique [12], to assess structural diversity. Our goal was to compare entropy measurements across various models.

Our analysis revealed that the global model, despite its larger pool of potential tiles in the training set, displayed less diversity than model 2. This counterintuitive outcome stems from the low probabilities of special tiles appearing in the global model. In contrast, model 2, with its more limited but realistically distributed tile set, generated more diverse output rooms. Additionally, we noticed that larger rooms tended to exhibit greater diversity, likely due to increased opportunities for random generation and a higher probability of rare tile occurrences.

While Shannon entropy is theoretically unbounded, it is maximized in equiprobable situations. For instance, with 100 uniformly distributed entities in a room, the entropy would be:

$$s_{diversity} = -\sum_i \frac{1}{100} \cdot \log \frac{1}{100} = \sum_i \frac{1}{100} \cdot \log 100 = 100 \times \frac{2}{100} = 2$$

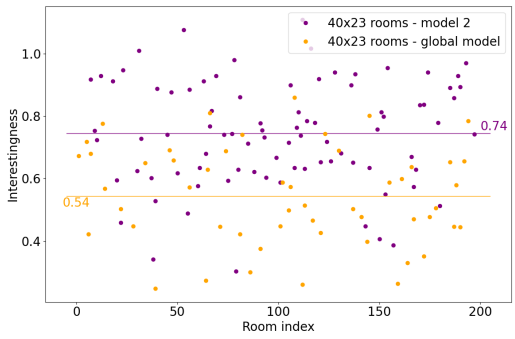
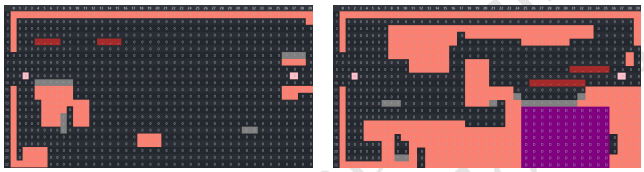


Figure 14: Comparison of the interestingness score between model 2 and global model - 200 rooms, 5 paths

We chose not to normalize this by the maximum achievable entropy, instead incorporating it into the weight w_3. It's worth noting that our interestingness score I is not necessarily normalized to $[0, 1]$.

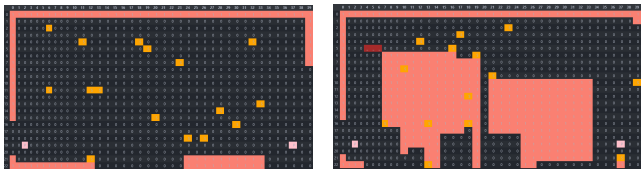
To compare the interestingness of model 2 and the global model, we set $w_1 = w_2 = w_3 = 1$. We generated 200 rooms for each model and computed the mean interestingness over 5 paths (or fewer, depending on successful pathfinding runs). As expected, Fig. 14 shows that model 2 produces more interesting rooms overall, demonstrating better capture of *Celeste*'s underlying structures.

We asked players to play 4 rooms that were at the extremes in terms of the interestingness score, two per model, and asked them to give a grade from 1 to 10 to the room interestingness: their engagement with the room, whether they found it fun or not. The idea is to see if players' opinion align well on the score definition.



(a) Model 2, $I = 0.41$ **(b) Model 2, $I = 1.08$**

Figure 15: Rooms tested - produced by model 2



(a) Model global, $I = 0.33$ **(b) Model global, $I = 0.81$**

Figure 16: Rooms tested - produced by global model

Initially, we were skeptical of the interestingness scores for the model 2 rooms. However, upon closer examination, the room in Fig. 15b appears to be the most well-designed; it offers clear gameplay while allowing for player freedom and showcases a nice diversity of tiles. In contrast, the room in Fig. 15a seems less polished,

	Count	Room a	Room b	Room c	Room d
Advanced players	8	6.75	3.25	5.38	5.25
Beginner players	4	4.25	7.75	2.25	7.50
All players	12	5.92	4.75	4.33	6.00

Table 2: Player statistics across grade rooms

with a limited variety of tile types. For rooms in Fig. 16a and Fig. 16b, we found the scores to be more aligned with our expectations, as they accurately reflect the diversity in both structures and apparent gameplay opportunities.

4.2.2 *Results & Discussion.*

User Study. Our results revealed an unexpected finding: the expert group gave similar ratings to rooms 16a and 16b. While room 16a was nearly impossible for beginners due to its advanced mechanics requirements, we anticipated that the advanced group would find it more engaging. However, the lack of music and background in the brute format of the levels may have influenced the experienced players' focus on gameplay alone.

Interestingly, the beginners' evaluations aligned more closely with our proposed interestingness metric. This suggests that our metric, which primarily considers NLE density and diversity, may better reflect a novice player's perception of a game they've never played before. In contrast, experienced players seem to value more nuanced aspects of level design, such as originality, novelty, and innovation.

Although our sample size is insufficient for statistical significance, we believe the observed trends are meaningful. Our current metric emphasizes high-level considerations but fails to account for advanced gameplay elements such as complex mechanics, trajectory precision, or movement variety. This limitation likely explains the disparity between beginner and advanced player evaluations.

For future research, we suggest recruiting more testers and generating rooms with a broader range of interestingness scores. This approach would allow us to assess the robustness of our metric for novice players and develop a refined version tailored to experienced players. Ultimately, this strategy could enable our PCG model to customize level generation for specific target audiences, introducing an additional layer of personalization before even factoring in difficulty levels.

4.3 Difficulty

The difficulty metric is crucial for evaluating our PCG system's effectiveness in generating *Celeste*levels. We view difficulty not merely as a measure of challenge, but as a comprehensive assessment of how generated rooms test player skills, manage risk, and enhance the overall game experience. Our metric incorporates various factors contributing to room challenge, including hazards (holes and lethal entities), NLE scarcity, and spatial distribution of lethal entities.

We define the difficulty D of a level as:

$$D = z_1 \times H_f + z_2 \times d_{LE,local} + z_3 \times s_{scarcity}$$

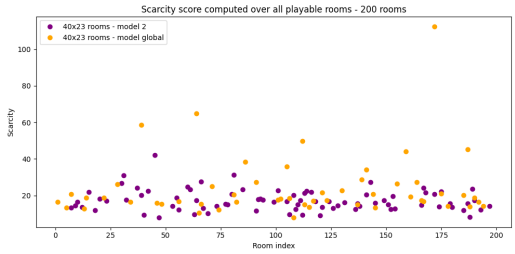


Figure 17: Scarcity score investigation

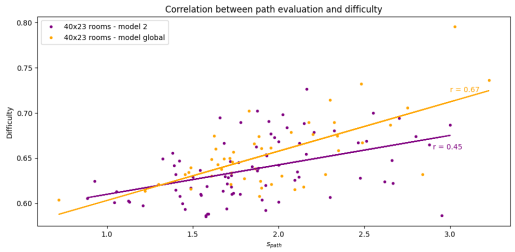


Figure 18: Correlation investigation between difficulty and s_{path} - 200 rooms, 5 paths

- z_1, z_2, and z_3 are adjustable weights
- $H_f = \frac{H}{L}$: hole frequency (ratio of holes H to path length L)
- $d_{LE,local} = \frac{N_{AOI,LE}}{AOI}$: local LE density (LEs in AOI to area of interest)
- $s_{scarcity} = \frac{1}{d_{NLE,local}}$: NLE scarcity (local NLE density inversed)

Our difficulty metric considers key factors: frequency and placement of holes (causing player death and room restart), lethal entities requiring precise navigation, and scarcity of Non-Lethal Entities (NLEs). Fewer safe zones or movement aids increase failure risk. This comprehensive evaluation enables our PCG system to generate appropriately challenging rooms.

We considered including dynamic hazards such as timed obstacles to enhance the metric. While we didn't incorporate these due to representation incompatibilities in training, they present an avenue for future improvement. Additionally, considering the number of required inputs or gameplay complexity could further refine the metric, given an AI agent capable of playing *Celeste*.

The importance of difficulty metrics in Procedural Content Generation (PCG) is well-recognized in literature. Research on dynamic difficulty adjustment in game AI highlights its role in maintaining player engagement. Adaptive difficulty has become increasingly central in video game development, emphasizing the need for accurate difficulty assessment to integrate relevant adaptive features within a PCG model. Drawing from these insights, we designed our difficulty metric to measure challenge levels and align with the intended player experience in Celeste.

4.3.1 Experiments. Our current difficulty metric design showed incompatibility with playability requirements, with limited achievable difficulty and inconsistent results. We generated 200 rooms, computing each component of the difficulty score D and path metrics like s_{path} and path variance. The unstable nature of scarcity caused the difficulty score to escalate rapidly, which we investigated in 17. High scarcity indicated very low local NLE density, resulting in unsupported paths. We displayed scarcity over rooms generated from model 2 and the global model to examine this measurement's behavior. Additionally, we explored potential correlations between difficulty and path characteristics in Fig. 18.

4.3.2 Results & Discussion. Our analysis revealed that scarcity values are significantly higher and more volatile for rooms generated by the global model compared to those from model 2. High scarcity indicates very low local NLE density, corresponding to a high s_{path}. Upon manual inspection, we found that many of these rooms were unplayable. This suggests that scarcity, along with other metrics, could be instrumental in better characterizing true playability. Ultimately, unplayable rooms should be excluded from difficulty and interestingness evaluations, emphasizing the need for a more precise playability assessment algorithm.

We discovered a moderate positive correlation between our current difficulty metric and s_{path}, which we believe indicates a flaw in our difficulty definition. As previously mentioned regarding interestingness, integrating advanced mechanics and precise gameplay elements into the difficulty concept is crucial, as these form the foundation of player progression. To address this, we propose expanding our study by classifying training rooms not only by levels but also by difficulty. This approach could enable us to train models capable of extracting difficulty-related patterns, rather than focusing solely on the structural continuity we initially sought through level-based classification.

5 Discussion & Conclusion

In this study, we set out to develop a fast and efficient procedural content generation (PCG) pipeline aimed at creating playable, interesting, and challenging levels for the 2D platformer *Celeste*, without relying on an AI agent for evaluation. Our approach used a Markov Chain-based model to capture the underlying structures and gameplay dynamics of existing levels, allowing us to generate new rooms that maintained both the aesthetic and functional qualities of the original game. Our primary goals were to enable the generation of levels from minimal input parameters and to develop a modding tool accessible to the *Celeste* community.

We successfully addressed the first major challenge identified earlier in the paper: generating clearable rooms in a game with nonlinear, vertical progression. The Markov Chain-based model proved effective in handling this complexity by modeling probabilistic transitions between tiles and gameplay elements, allowing for coherent room structures with clear paths from entrance to exit. Post-processing steps, such as refining exits and strategically placing respawn points, ensured that the levels were not only functional but also enjoyable and challenging for players.

The second challenge, the lack of a *Celeste*-specific AI framework, was addressed by developing agent-free evaluation mechanisms within our PCG pipeline. Through user testing, we evaluated the playability, challenge, and engagement of the generated levels,

which were rated favorably by beginner players. While the absence of an AI agent did not prevent us from generating functional and engaging levels, we observed limitations in the creative diversity of the generated rooms and in controlling difficulty for more experienced players. These findings underscore the importance of refining the generation process, potentially through integrating more advanced techniques like Generative Adversarial Networks (GANs) or hybrid approaches.

We answered the central research question **How can we generate interesting and playable 2D platforming levels for *Celeste* without an AI agent for evaluation?**

Our Markov Chain-based PCG pipeline provided a solid foundation for this, successfully generating playable levels, albeit with room for improvement in creativity and difficulty control.

Looking forward, future work could focus on enhancing the creative potential of the generation process and better adapting to the expectations of experienced players. This could involve integrating adaptive difficulty mechanisms or expanding the user feedback framework to further tailor the gameplay experience. Additionally, our PCG pipeline could be used to generate a training set for a potential *Celeste* AI agent, advancing research in this area.

In conclusion, our work demonstrates the viability of using a Markov Chain-based approach to generate *Celeste* levels, addressing the unique challenges of the game's vertical progression and lack of an AI framework. While there is room for refinement, this pipeline serves as a promising proof of concept for procedural level generation in *Celeste* and lays the groundwork for future advancements, including the development of a dedicated AI framework for platformers with complex level design.

References

[1] David Adams. 2002. Automatic Generation of Dungeons for Computer Games. (01 2002).

[2] Arman Balali Moghadam and Marjan Kuchaki Rafsanjani. 2017. A Genetic Approach in Procedural Content Generation for Platformer Games Level Creation. https://doi.org/10.1109/CSIEC.2017.7940160

[3] Cruor, Vexatos, Jade Macho, and DemoJameson. 2018. Maple: a thin wrapper for generating map files for the game Celeste. https://github.com/CelestialCartographers/Maple.

[4] Steve Dahlskog, Britton Horn, Noor Shaker, Gillian Smith, and Julian Togelius. 2014. A Comparative Evaluation of Procedural Level Generators in the Mario AI Framework.

[5] Joris Dormans. 2010. Adventures in level design: Generating missions and spaces for action adventure games. (06 2010). https://doi.org/10.1145/1814256.1814257

[6] Britton Horn, Steve Dahlskog, Noor Shaker, Gillian Smith, and Julian Togelius. 2014. A comparative evaluation of procedural level generators in the mario ai framework. In *Foundations of Digital Games 2014*. 1–8.

[7] Umar Iskandar, Norizan Diah, and Marina Ismail. 2020. Identifying Artificial Intelligence Pathfinding Algorithms for Platformer Games. 74–80. https://doi.org/10.1109/I2CACIS49202.2020.9140177

[8] Daniël Karavolos, Anders Bouwer, and Rafael Bidarra. 2015. Mixed-Initiative Design of Game Levels: Integrating Mission and Space into Level Generation.. In *Foundations of Digital Games 2015*. 1–8.

[9] Darius Kazemi. 2008. Spelunky Generator Lessons. http://tinysubversions.com/spelunkyGen/.

[10] Ahmed Khalifa, Michael Cerny Green, Gabriella A. B. Barros, and Julian Togelius. 2019. Intentional Computational Level Design. *CoRR* abs/1904.08972 (2019). arXiv:1904.08972 http://arxiv.org/abs/1904.08972

[11] Roland Linden, R. Lopes, and Rafael Bidarra. 2013. Designing procedurally generated levels. *AAAI Workshop - Technical Report*, 41–47.

[12] Lesedi Masisi, Fulufhelo Nelwamondo, and Tshilidzi Marwala. 2008. The use of entropy to measure structural diversity. *CoRR* abs/0810.3525 (12 2008). https://doi.org/10.1109/ICCCYB.2008.4721376

[13] Noor Shaker, Julian Togelius, Georgios N Yannakakis, Likith Poovanna, Vinay S Ethiraj, Stefan J Johansson, Robert G Reynolds, Leonard K Heether, Tom Schumann, and Marcus Gallagher. 2013. The turing test track of the 2012 mario ai championship: entries and evaluation. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, 1–8.

[14] Noor Shaker, Julian Togelius, Georgios N Yannakakis, Ben Weber, Tomoyuki Shimizu, Tomonori Hashiyama, Nathan Sorenson, Philippe Pasquier, Peter Mawhorter, Glen Takahashi, et al. 2011. The 2010 Mario AI championship: Level generation track. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 4 (2011), 332–347.

[15] Noor Shaker, Georgios N. Yannakakis, and Julian Togelius. 2010. Towards Automatic Personalized Content Generation for Platform Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2010). https://api.semanticscholar.org/CorpusID:3159079

[16] Sam Snodgrass and Santiago Ontanon. 2013. Generating maps using Markov chains. (01 2013), 25–28.

[17] Sam Snodgrass and Santiago Ontañón. 2017. Learning to Generate Video Game Maps Using Markov Models. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 4 (2017), 410–422. https://doi.org/10.1109/TCIAIG.2016.2623560

[18] David Stammer, Tobias Günther, and Mike Preuss. 2015. Player-adaptive Spelunky level generation. In *2015 IEEE Conference on Computational Intelligence and Games, CIG 2015, Tainan, Taiwan, August 31 - September 2, 2015*. IEEE, 130–137. https://doi.org/10.1109/CIG.2015.7317948

[19] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270.

[20] Sarjak Thakkar, Changxing Cao, Lifan Wang, Tae Jong Choi, and Julian Togelius. 2019. Autoencoder and Evolutionary Algorithm for Level Generation in Lode Runner. *2019 IEEE Conference on Games (CoG)* (2019), 1–4.

[21] Julian Togelius, Georgios Yannakakis, Kenneth Stanley, and Cameron Browne. 2011. Search-Based Procedural Content Generation: A Taxonomy and Survey. *Computational Intelligence and AI in Games, IEEE Transactions on* 3 (10 2011), 172 – 186. https://doi.org/10.1109/TCIAIG.2011.2148116

[22] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M. Lucas, Adam M. Smith, and Sebastian Risi. 2018. Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network. *CoRR* abs/1805.00728 (2018). arXiv:1805.00728 http://arxiv.org/abs/1805.00728

A CLI interface for PCG pipeline

Below is the list of input parameters of the end-to-end PCG pipeline used to generate complete levels:

- `-config`, `-c` (`str`): Configuration matrix for training
- `-training-dataset`, `-td` (`str`): Room dataset for training
- `-nb-rooms`, `-nr` (`int`): Number of rooms to generate
- `-proba`, `-p` (`float`): Probability-p_2 for generating labyrinth-style levels. 0 =pathway, 1=random room order
- `-room-size`, `-rs` (`list of two ints`): Room dimensions (width and height)
- `-bt-depth`, `-btd` (`int`): Max. backtracking for room generation
- `-tries-limit`, `-tl` (`int`): Max. attempts of room generation
- `-reset-skeleton`, `-r` (`bool`): Whether to regenerate the skeleton if room generation fails

This command initializes the PCG pipeline by loading the training dataset and creating a Dictionary of Probability Transitions (DPT). The pipeline then generates a level skeleton, populates rooms using a Multidimensional Markov Chain (MdMC) model, and applies post-processing for playability. To prevent generation loops, the process can reset problematic skeletons. Finally, the Room Encoder converts the level into a playable binary file for *Celeste*.