

---

# Procedural Content Generation of The Legend of Zelda Using Markov Models

---

**Shan Lu\***

Department of Electrical and Computer Engineering  
University of Alberta  
slu1@ualberta.ca

**Ali H. G. Foomani\***

Department of Computer Science  
University of Alberta  
hosseing@ualberta.ca

**Shengyao Lu\***

Department of Electrical and Computer Engineering  
University of Alberta  
shengyao@ualberta.ca

**Xinlei Chen\***

Department of Computer Science  
University of Alberta  
xinlei1@ualberta.ca

## Abstract

Procedural content generation of game maps has become a popular topic in recent years. However, most content generation systems focus on platformer games, like *Super Mario Bros.*, *Loderunner*, and *Kid Icarus*. We are interested in the methods that can be applied specifically to dungeon-like games. In this paper, we explore Markov models for a dungeon-like game, *The Legend of Zelda*. We compare the performance of the models in terms of playability, diversity and style match. We also propose a new multi-layer Markov Chain with Post-Processing Approach (MLMC+PP) which has better performance compared to other Markov models.

## 1 Introduction

Procedural content generation (PCG) has become a popular approach for generating game maps algorithmically, which aims to introduce more game levels and allows the user to experience new and unique content. As most of the generation techniques are intended for platformer games. We are interested in exploring the generation algorithms for dungeon-like games like *The Legend of Zelda*.

In this paper, machine learning techniques like Markov Random Field (MRF), Markov Chains (MdMC and MLMC), and Post-Processing (PP) are implemented and discussed to generate fresh and reasonable room maps in the dungeons of *The Legend of Zelda*. These algorithms are trained by training room maps and then generate new room maps.

We focus on the cross-model comparison by testing the playability of the generated room maps, how far the generated maps differ from each other, and how closely the style of the generated maps is to the testing maps. A good performance is defined by high playability, a diversity that closes to the training maps and a style matches the testing maps as tight as possible.

To summarize, we give a brief background of PCG and Markov models in Section 2. And in Section 3, we show how we represent the room maps, and discuss all the methods we implement. In Section 4, we explain our experimental setup and present the performance comparison between the aforementioned methods. Finally, we conclude our research and give some views of future extensions.

---

\*These authors contributed equally to this work.

Table 1: Tile types with general functionalities














Icon	Grid Functionality	Tile
	Block	<i>B</i>
	Door	<i>D</i>
	Floor	<i>F</i>
	Monstor	<i>M</i>
	Void	-
	Wall	<i>W</i>
	Water	<i>P</i>
	Stair	<i>S</i>

Table 2: Tile types with special functionalities

Icon	Grid Functionality	Tile
<b>Special Walls</b>		
	Single Arrow Going In	<i>N</i>
	Single Arrow Going Out	<i>U</i>
	Double Arrow	<i>E</i>
	Breakable Wall	<i>A</i>
<b>Special Block</b>		
	Movable Block	<i>C</i>

## 2 Background

This section gives background on Procedure Content Generation via Machine Learning (PCGML) and on Markov models.

### 2.1 Procedure Content Generation via Machine Learning

PCGML refers to the use of models that have been trained on existing game content to generate new game content [4]. This section discusses previous works on the content generation of 2-D games via machine learning.

Summerville and Mateas [5] trained Long Short-Term Memory recurrent neural networks (LSTMs) to generate new levels for *Super Mario Bros.*. Since LSTM is not as explainable as probabilistic graphical models and requires more data to train, in this paper, our approaches focus on the level generation via Markov models.

Snodgrass and Ontañón applied the multi-dimensional Markov Chain (MdMC) [2] and Markov Random Field (MRF) [1] to the level generation of *Super Mario Bros.* and *Kid Icarus*. They found MRF beats the basic MdMC in *Kid Icarus*. Also, Snodgrass and Ontañón [6] have extended their MdMC approach with the use of multi-layer representations of levels for *Super Mario Bros.*. They represented the level maps using a set of layers. Since the multi-layer MdMC model can capture the dependencies across multiple layers, the authors found the multi-layer MdMC can model the level maps more accurately. To increase the playability of the generated room maps for *The Legend of Zelda*, we also add the height and width layers to encode the tile position to our basic one-layer MdMC model.

### 2.2 Markov Models

Markov models model the transition probabilities between states. They can be applied to model and generate level maps for 2-D games. Markov Chain [7] is defined as a set of states  $S = \{s_1, s_2, \dots, s_n\}$  satisfying the Markov Property, which is that the probability of transiting to the next state only depends on its previous state. Multi-dimensional Markov Chains (MdMC) [1] are extended from Markov Chains. For MdMC, the probability of transiting to the next state depends on  $k$  previous surrounding states. Thus, MdMCs can model 2-D game level data easily. Markov Random Fields (MRFs) [9] can model dependencies in all directions, that is, the probability of transiting to the next state depends on surrounding states in all directions.

## 3 Method

### 3.1 Map Representation

Similar to the map representation technique proposed by Snodgrass and Ontañón [1], which is used to represent the platformer maps. We represent all the room maps in *The Legend of Zelda* via tile

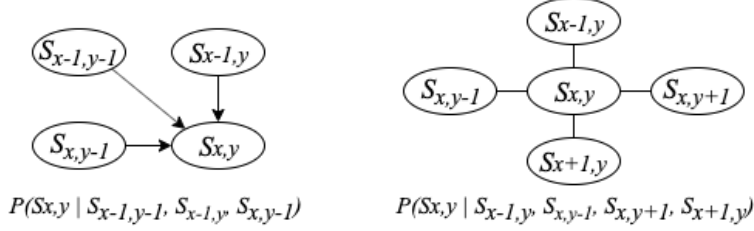


Figure 1: The third-order MdMC (left), The Markov Random Field (right)

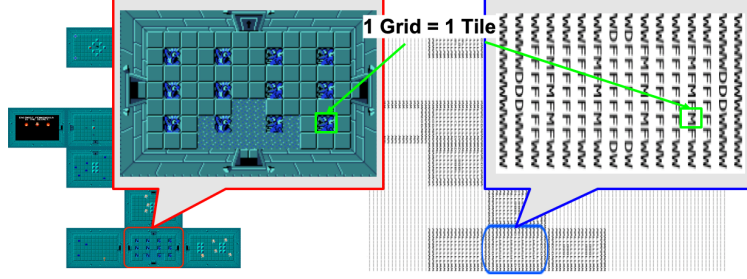


Figure 2: The raw map of the first room in the *Eagle* dungeon in *The Legend of Zelda*; And its corresponding tile representation.

representation, where each grid of the room map takes a value from tile types shown in Table 1 and Table 2. In particular, each room map has an  $h \times w$  array  $m$ , where  $h = 9, w = 16$  in our case. Fig. 2 gives an instance of a room map and its tile representation.

In *The Legend of Zelda*, there are some tiles that have special functionalities, like the ones in Table 2. During the actual game-playing, special walls look the same as the ordinary ones until the player interacts properly with them, and so does the movable blocks. For evaluation purposes, we use different tile types to represent these special tiles to distinguish them from the general tiles. For example, in a generated room map that has all paths between the doors blocked, if one of the blocks along the way is a movable one, and once the block has been moved, the path can become unblocked. In such way, these special tiles can affect the evaluation of playability. In order to visualize the generated room maps in a more intuitive way, we use special icons to represent the special tiles.

### 3.2 Markov Random Fields

Markov Random Field (MRF) proposed by Clifford [9] models probabilistic relations between the surrounding states in a graph. Complicated networks could be applied. However, since each room map in *The Legend of Zelda* has only  $16 \times 9$  tiles, which is relatively small, we simply introduce the CPD that contains the top, the bottom, the left, and right states for each state being investigated (Fig. 1 (right)).

Different from platformer games, *The Legend of Zelda* room maps have "borders", which consist of walls and doors. Doors and special walls can only be placed in the middle of the walls in each room. Due to this special map structure, we choose to train and sample the borders in a different approach from the indoor maps (the room map exclude its borders). For the borders, we use a histogram-based nonparametric probability estimation approach to train the doors and special walls. And then we sample the border randomly with the probability trained. Other than the locations that could potentially have a door or a special wall, walls will be simply sampled. The borders are fixed after initial sampling.

We train and sample the indoor maps in a similar manner to the one Snodgrass and Ontaño [1] used. The MRF training algorithm with a histogram-based nonparametric probability estimation approach returns a CPD dictionary, which gives the conditional probability at each combination of the location of the tile being investigated, the tile type of the current tile, and the tile type of surrounding tiles. As for the sampling, since MRFs capture dependencies in all directions, instead of using a sequential

sampler, we sample the room maps using the Metropolis-Hasting algorithm [10]. With the border sampled, the indoors sampling algorithm initializes an indoor map with each tile chosen based on the network structures with no dependencies. And then modifications are made by randomly choosing two positions in the indoor map, and swapping the tile types located correspondingly. The swapping is repeated by  $b$  times, which can be human-defined.

### 3.3 Multi-Dimension Markov Chain

We train and generate new room maps for *The Legend of Zelda* with the MdMC model structure in Fig. 1 (left).

The training procedure for the basic MdMC is similar to Algorithm 2 used in Snodgrass and Ontañón [1]. For every position in the room maps, we count the total number of times that a tile type of the current node  $S_{x,y}$  appearing with the tile types configuration of its previous nodes  $S_{x-1,y-1}, S_{x-1,y}, S_{x,y-1}$ , which is noted as  $T(S_{x,y}, S_{x-1,y-1}, S_{x-1,y}, S_{x,y-1})$ . And we use the frequency counts to estimate the conditional probability distributions of the MdMC model. That is,

$$P(S_{x,y}|S_{x-1,y-1}, S_{x-1,y}, S_{x,y-1}) = \frac{T(S_{x,y}, S_{x-1,y-1}, S_{x-1,y}, S_{x,y-1})}{\sum_{S'_{x,y} \in \text{tile types}} T(S'_{x,y}, S_{x-1,y-1}, S_{x-1,y}, S_{x,y-1})}$$

The sampling procedure for the basic MdMC contains two parts. Firstly, we generate the "borders" of the room map first, and the detailed method for generating the "borders" is discussed in section 3.2. Then, similar to Algorithm 3 used in Snodgrass and Ontañón [1], we generate the indoor map. We sample a tile type for the every position  $S_{x,y}$  from the conditional probability distributions  $P(S_{x,y}|S_{x-1,y-1}, S_{x-1,y}, S_{x,y-1})$ . And we use the look-ahead and fallback strategies [1] to avoid encountering the unseen tile configuration.

### 3.4 Multi-Layer Markov Chain

This section will discuss the multi-layer representation and how this new representation can help the model to perform better. Then the new extra layers, which we added to our model, will be introduced. At the end of this section, how the model works based on multi-layer representation and the procedure of generating new tile will be represented. The main idea of this approach comes from [6].

#### 3.4.1 Multi-layer Representation

In our multi-layer representation, each room is represented using a set of layer,  $L = \{l_1, l_2, \dots, l_n\}$ , where each  $l_i$  is a two-dimensional matrix of tiles with the dimensions of  $h \times w$  (height and width of the room). Each layer is represented using a special set of tiles  $T_i$ , where the meaning of the tiles varies by layer. For example, one layer may have some information about the structure of the room. Therefore, each tile in this layer should represent an object or element in the room while another layer may contain metadata about the room, such as the distance of the element from the borders of the room in each direction.

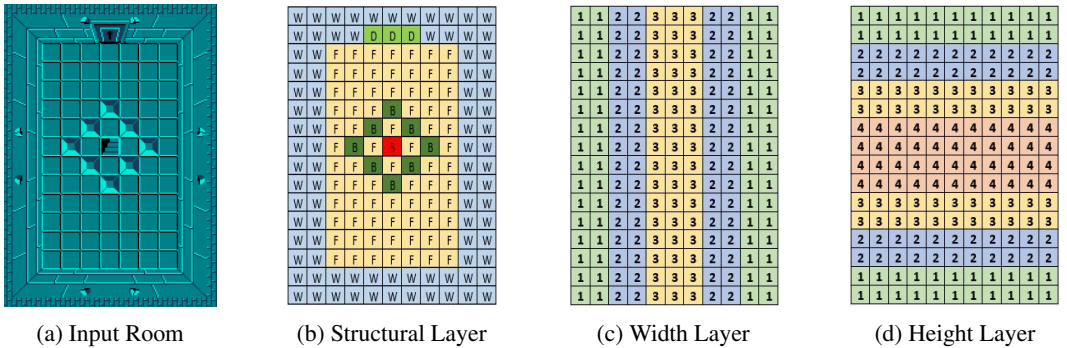


Figure 3: This figure shows (a) a room map of *The Legend of Zelda*, (b) represented using a structural layer, (c) a width layer, and (d) a height layer.

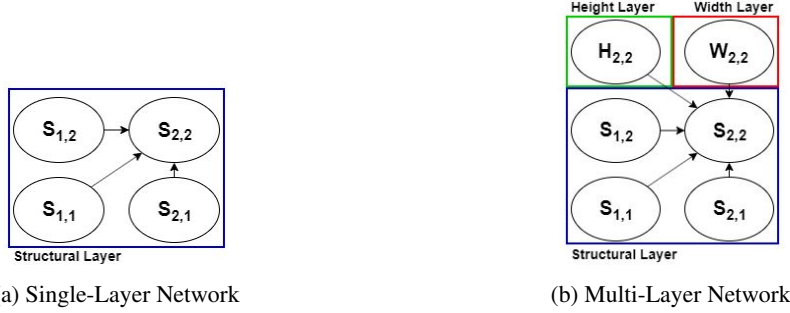


Figure 4: This figure shows a network that is used by the single-layer MdMC method (left) and a network that is used by multi-layer MdMC method (right).



Figure 5: This figure show playable (left) and unplayable (right) rooms generated by multi-layer MdMC method.

Figure 3 shows a room of *The Legend of Zelda* represented by three layers: (b) Structural layer, representing the placement of elements in the room; (c) Width layer, representing the relative distance from left and right walls of the room; (d) Height layer, representing the relative distance from top and bottom walls of the room. The two latter layers will help the model understand that the patterns happening in the different regions of the map are not the same. The structural layer is the most important layer that we have because this is the layer that provides the tile set that can appear in the rooms and defines which tiles can come alongside each other.

Here we only include structure, height and width layer while these are not the only possible layers. We could add a player path layer producing some information regarding the possible optimum path that the player would go through.

### 3.4.2 Multi-layer Sampling

The multi-layer method is an extension of MdMC, so these methods are the same with some little differences. In MdMC, a set of surrounding tiles will be considered as previous states. The difference comes from the fact that the set of tiles, which are being considered to generate a new tile, are all placed in the structural layer in the MdMC method. On the other hand, in the multi-layer approach, which we proposed, in addition to considering tiles from the structural layer we need to consider the tiles from the width and height layer indicating the location of the tile that is going to be generated.

Consider the structure shown in the figure 4b. The conditional probability term defining the next tile to generate can be written as  $P(S_{x,y} | S_{x-1,y}, S_{x,y-1}, S_{x-1,y-1}, W_{x,y}, H_{x,y})$ , where  $S$ ,  $W$ , and  $H$  are states corresponding to structural, width, and height layer. Figure 5 shows the playable and unplayable room, which have been generated using the multi-layer MdMC method.

### 3.5 Post-Processing

We seek to find a way to generate rooms that are consistent and playable. Approaches that are represented in previous sections are based on generating new tiles by considering a limited number of tiles that exist in the past and they do not care about other tiles that exist in the map. These kinds of methods sometimes lead to structures that are not playable. To avoid generating this kind of room

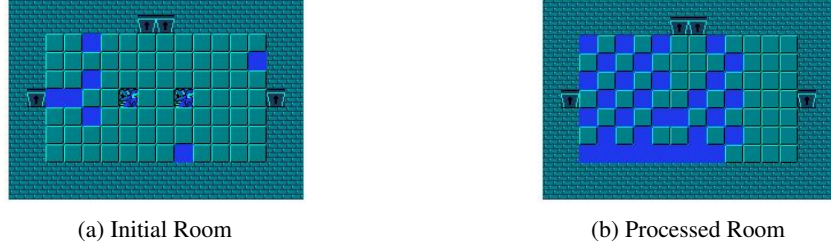


Figure 6: This figure shows an initial room, which is unplayable (left) and the same room after Post-Processing (right).

maps, one approach would be considering more tiles. The number of room maps that are available to train the model is limited, so we cannot use this approach because the number of combinations of different tiles will be too large and the model will overfit. Instead of increasing the number of states to consider for generating new tiles. We propose another method which post-processes the room maps that have been generated using existing methods. This approach works based on searching for sub-maps that did not exist in the set of rooms that have been used to train the model. The method goes through the map and checks each sub-map in the generated room (line 11-22). Whenever it finds an unseen sub-map, it tries to replace the center tile of the sub-map using a tile that has high probability based on the surrounding tiles of the sub-map (line 14-18). Algorithm 1 describes the way that the Post-Processing method works. Figure 6 demonstrates a room which is unplayable initially and the same room after applying Post-Processing method.

---

**Algorithm 1:** Post Processing( $m_o$ , trainingData, numIterations)

---

```

1 // INPUT  $m_o$  = room maps generated using one of previous algorithm
2 SubRoomMapSet = Empty Set;
3 SurroundingSet = Empty Set;
4  $m_n$  = Empty List; // OUTPUT  $m_n$  = post processed room maps
5 for each roomMap  $\in$  trainingData do
6   for each  $3 \times 3$  subRoomMap  $\in$  roomMap do
7     append subRoomMap to SubRoomMapSet;
8     append removeCenter(subRoomMap) to SurroundingSet;
9   end
10 end
11 for numIterations do
12   for each roomMap  $\in$   $m_o$  do
13     for each  $3 \times 3$  subRoomMap  $\in$  roomMap do
14       if subRoomMap  $\in$  SubRoomMapSet then
15         Continue;
16       else
17         surrounding = removeCenter(subRoomMap);
18         selectedSurroundings = items in SurroundingSet having the highest number of
19           same tiles with surrounding;
20         selectedSubRoomMaps = subRoomMaps in SubRoomMapSet having the same
21           surroundings in selectedSurroundings;
22         sampledSubRoomMap = sample from selectedSubRoomMaps;
23         subRoomMap[1][1] = sampledSubRoomMap[1][1]; // replace center
24       end
25     end
26   append roomMap to  $m_n$ ;
27 end
28 Return  $m_n$ ;

```

---

## 4 Experiments

In this section, we test our approach by sampling room maps for dungeon like games, *The legend of Zelda*. Our first goal is to determine whether our multi-layer MdMC with post-processing approach performs better than single-layer MdMC and MRF in terms of three evaluation metrics we describe below. Our second goal is to test how our Post-Processing method alone will increase the playability of single-layer MdMC.

### 4.1 Experimental Set-up

For our experiments, we use 18 whole levels of *The legend of Zelda* from the VGLC corpus [3]. We break the levels into 459 room maps. We split room maps into 80% training data and 20% test data.

#### 4.1.1 Evaluation Metrics

In order to evaluate different approaches, we generate 400 (16\*11 size) room maps by each one of them, and use three different metrics to assess their quality:

- *% Playability*: in order to test whether the generated room maps are actually playable. In each generated room map, we use A\* search algorithm to find path between each [start,end] pair (the type of start and end tile include DOOR and STAIR). If one of [start,end] pairs can not find path, we will count this room map as unplayable room map.
- *Diversity*: average distance between every two same type room maps in whole set of room maps (Equation (1)). Since we do not want the generated room maps are exactly same or completely different, we use diversity to measure how different is the room map from other room maps. The room maps can be classified into different room types according to the majority tile type in each room map. It is trivial to compute distance of two different type room maps, so we use K-means clustering method (K=5) to classify room maps. We use histogram distance function to measure the distance between two room maps, which performs a simple count for each tile type in both room maps and distance is computed as the Manhattan distance between the resulting vectors.

$$Diversity(rooms) = \frac{\sum_{i=1}^n \sum_{j=1}^n SameType(i, j) * Dist(rooms[i], rooms[j])}{\sum_{i=1}^n \sum_{j=1}^n SameType(i, j)} \quad (1)$$

$$SameType(i, j) = \begin{cases} 1, & \text{type}(rooms[i]) = \text{type}(rooms[j]) \\ 0, & \text{otherwise} \end{cases}$$

- *Style Match*: how well the style of generated room maps by trained model matches the style of test data. In other words, given the trained model, what is the probability that the model would generate test data. In order to avoid probability vanish issue, we use log probability (Equation (2)).

$$\begin{aligned} StyleMatch &= \log P(TestData | Model_{TrainData}) \\ &= \sum_{i=1}^n \log P(TestData[i] | Model_{TrainData}) \end{aligned} \quad (2)$$

In our experiments, we want the diversity of our generated room maps approaches the diversity of training data. Alternatively, We want our generated room maps has high playability and high probability of style match with test data.

#### 4.1.2 Approaches

- *Random*. We use Random Model Approach as baseline. All indoor map tiles in generated room maps sample from uniform distribution during sampling procedure.
- *MRF*. We use the implementation of Markov Random Field Approach described on [1]. It requires one input parameter: b is the number of times each location in room map has been chosen to swap during sampling procedure. We set parameter b to 9999.

- MdMC (single-layer MdMC). We adopt the implementation of MdMC from the paper [1]. The MdMC model structure we used is in Fig. 1 (left).
- MLMC+PP. Our approach is based on the algorithm described in previous sections which includes multi-layer MdMC and Post-Processing.

## 4.2 Results

Table 3: Evaluation Results

Method	Style Match	Diversity Difference	Playability
Random	-15037.99	10.26	0.8660
MRF	<b>-1905.82</b>	-1.84	0.9742
MdMC	-3148.57	3.64	0.9335
MdMC+PP	/	3.03	<b>0.9830</b>
MLMC	/	1.53	0.8835
MLMC+PP	/	<b>-0.11</b>	0.9668

Table 3 shows the evaluation result. The best result of each evaluation metric is shown in bold.

### A. Diversity:

We compute diversity difference by function  $diversity(generated\ room\ maps) - diversity(training\ data)$ . We want the diversity of our generated room maps by each method approaches the diversity of training data, thus we only care about the difference between them. The closer the difference value is to zero, the better performance. Positive value means the average distance among our generated room maps is larger than training data’s, vice versa. One clear observation is that all approaches have better performance than Random Model Approach in terms of diversity. Our approach MLMC+PP get the smallest difference with training data.

### B. Playability:

We can observe that the playability of room maps generated by each methods exceeds 85%. This can be explained by the fact that the player in *The legend of Zelda* can choose some tools which can across block tiles (e.g. LAVA, WATER tiles) under certain restrictions. As we can see from Table 3, after post-processing, the playability is greater than that without post-processing. So our post-processing method alone will increase the playability of single-layer MdMC. Although The room maps generated by MdMC+PP get the largest playability, it is only slightly higher compare to MRF and MLMC+PP.

### C. Style Match:

Since both MdMC and MLMC use the Markov Chain as base model, their style results will be same if their Markov Chain model use same model structure and be fed by the same training data. As we can see, the style of generated room maps by MRF best matches the style of test data. Although MRF get best performance in style match, its generated room maps contain more randomness compared to MLMC+PP after we manually check the generated room maps.

## 5 Conclusion and Future Work

This paper explore over several Markov models, namely MRF, MdMC, MLMC, to perform content generation of *The Legend of Zelda*. We found that MRF gives relatively good results on the playability and style match. Whereas it has large diversity, which indicates that it has too much randomness. MdMC that samples the maps sequentially gives much randomness as well with even a lower playability. MLMC samples less playable maps, but it gives pretty low diversity, which means less randomness. Less randomness means more player-friendly. We also propose a Post-Processing method. Post-Processing technique is applied on MdMC and MLMC, and improves the playability and diversity for these two algorithms. Overall, MLMC+PP presents the best content generation results.

Due to the time limitation, MRF+PP has not been implemented and is left for future extension. Additionally, we only analyze the room maps separately, and the whole dungeon structure can be also considered [11] for better performance.



## References

- [1] S. Snodgrass and S. Ontañón, “Learning to generate video game maps using Markov models,” *IEEE Trans. Computat. Intell. AI Games*, vol. 9, no. 4, pp. 410-422, Dec. 2017.
- [2] S. Snodgrass and S. Ontañón, “Experiments in map generation using Markov chains” in *Proc. Found. Digit. Games Conf.*, 2014, vol. 14.
- [3] A.J.Summerville, S. Snodgrass, M. Mateas, and Ontañón, “The VGLC: The video game level corpus,” in *Proc. 7th Workshop Procedural Content Gener.*, 2016.
- [4] A.J.Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen, and J. Togelius, “Procedural Content Generation via Machine Learning (PCGML)”. *arXiv preprint arXiv:1702.00539*.
- [5] A. Summerville and M. Mateas, “Super Mario as a String: Platformer Level Generation Via LSTMs,” *DiGRA/FDG*, 2016.
- [6] S. Snodgrass and S. Ontañón, “Procedural Level Generation using Multi-layer Level Representations with MdMCs,” *IEEE Conference on Computational Intelligence and Games 2017*.
- [7] A. A. Markov, “Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a chain” Reprinted in Appendix B of: R. Howard. *Dynamic Probabilistic Systems, Volume 1: Markov Chains*. John Wiley and Sons, 1971.
- [8] W.-K. Ching, X. Huang, M. K. Ng and T.-K. Siu, “Higher-Order Markov Chains” In *Markov Chains* pp 141-176, Springer, 2013.
- [9] P. Clifford, “Markov Random Fields in Statistics” In *Disorder in Physical Systems: A Volume in Honour of John M. Hammersley*. Oxford, U.K.: Oxford Univ. Press, 1990, pp. 1932.
- [10] G. R. Cross and A. K. Jain, Markov random field texture models, *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-5, no. 1, pp. 2539, Jan. 1983.
- [11] A. J. Summerville, M. Behrooz, M. Mateas, and A. Jhala, “The Learning of Zelda: Data-driven learning of level topology, in *Proc. 10th Int. Conf. Found. Digit. Games*, 2015.

## Project Summarization

Table 4: Project Breakdown

Task	Name
Data Re-preprocessing	Shan Lu
Image Map Generation Tool	Shengyao Lu
MRFs	Shengyao Lu
MdMC	Shan Lu
MLMC	Ali H. G. Foomani
Post-Processing	Ali H. G. Foomani
MdMC+PP	Xinlei Chen
Evaluation Metrics	Xinlei Chen
Evaluation Analysis	Xinlei Chen
Final Report	ALL