Generating Lode Runner Levels by Learning Player Paths with LSTMs

Kynan Sorochan
University of Alberta
Edmonton, Canada
ksorocha@ualberta.ca

Jerry Chen
University of Alberta
Edmonton, Canada
jerry3@ualberta.ca

Yakun Yu
University of Alberta
Edmonton, Canada
yakun2@ualberta.ca

Matthew Guzdial
University of Alberta
Edmonton, Canada
guzdial@ualberta.ca

ABSTRACT

Machine learning has been a popular tool in many different fields, including procedural content generation. However, procedural content generation via machine learning (PCGML) approaches can struggle with controllability and coherence. In this paper, we attempt to address these problems by learning to generate human-like paths, and then generating levels based on these paths. We extract player path data from gameplay video, train an LSTM to generate new paths based on this data, and then generate game levels based on this path data. We demonstrate that our approach leads to more coherent levels for the game Lode Runner in comparison to an existing PCGML approach.

CCS CONCEPTS

• **Computing methodologies → Machine learning approaches**; **Neural networks**.

KEYWORDS

datasets, neural networks, path learning, path detection

1 INTRODUCTION

Procedural content generation via machine learning (PCGML) is the study and application of machine learning to procedurally generating content, particularly for games [23]. While PCGML has had enjoyed considerable popularity recently, a number of open problems exist. Particularly in comparison to traditional, non-ML PCG, PCGML approaches struggle with controllability and coherence.

We define controllability as the ability for a user to impact particular attributes of the generated content. In traditional PCG, since the system is authored by a human, there are a number of strategies to allow a user to impact the output or enforce particular constraints [7, 9, 14]. There are efforts to make PCGML approaches controllable, but this is still an under-explored problem [2, 3, 11, 13]. In particular, we identify a lack of focus on approaches that allow users to specify high-level, intuitive constraints as the input to an ML generator that then outputs game content that matches those constraints.

By coherence we indicate the problem of game content demonstrating global coherence, global structure that fits human understanding of that game content. Global structure includes a wide range of constraints, and is dependent on the particular game content in question. For example, playability in game levels, the ability to complete said level, is an example of global structure that we expect from human-authored game levels. However, a level being playable is not the only element of global structure that we expect. A completely flat platformer game level would be playable, but would violate other elements of global structure. Modeling global structure is a common problem in machine learning generally [10]. In PCGML, there have been attempts to model global structure, with the most common approach being to model the player's path through a game level [12, 21]. However, this area is also under-explored.

In this paper, we investigate a novel PCGML approach that attempts to address these two open problems: controllability and coherence. Specifically, we introduce an approach to generate Lode Runner levels based on specified paths. Our focus on coherence will be in reference to Lode Runner Levels. We model these paths to ensure that they are human-like with a Long Short-Term Memory Recurrent Neural Network (LSTM RNN or LSTM). For training data we extract real human paths on existing Lode Runner levels from gameplay video. We then use the LSTM to generate novel human-like paths and employ a Markov Chain to generate novel levels based on these generated paths. We employ a Markov Chain for this initial investigation as it represents a simple ML model that typically struggles to capture global structure. Therefore, it's an ideal choice to investigate whether this approach improves level coherence. Our approach is inherently controllable as we can input arbitrary paths, though we focus on our generated, human-like

paths in this paper. We acknowledge that we won't be directly evaluating the controllability of this approach, but still contend that it is controllable.

In this paper, we first introduce related prior work. We then overview our generator, from data extraction to the novel generation of Lode Runner levels. We compare the performance of our generator to an existing Markov Chain generator without path data to evaluate whether we demonstrate improved coherence [18]. We then present a secondary evaluation of our approach in comparison to the original Lode Runner levels. We end with a discussion of our limitations and future work.

2 RELATED WORKS

In this section we overview work in terms of prior PCGML approaches to generate Loderunner levels, controllability in PCGML, and coherence via paths in PCGML.

Many PCGML approaches have been applied to level generation for Lode Runner in recent years. Thakkar et al. proposed the use of a variational autoencoder to generate new levels for Lode Runner based on a binary encoding of each character in the original levels. They attempted to improve the playability, one aspect of global structure, by searching the learned, latent space with an evolutionary algorithm [25]. We also attempt to increase playability, but based on altering the input to the generation pipeline instead of including search-based PCG within the pipeline. Snodgrass and Ontanón made use of Markov models to generate content for many games including Lode Runner [19], we include this approach as a baseline as we also make use of a Markov chain for our generator.

Markov chains have been a common method for PCGML since its inception [15]. Much of this prior work has focused on *Super Mario Bros.* level generation, a common area of PCGML research [15, 17, 18, 22]. We also employ Markov chains, as they typically struggle with global coherence in comparison to other methods [5].

Many approaches have been made to attempt to improve the coherence of PCGML output [25]. Of particular interest to us are approaches that attempt to do this by invoking some representation of the player path [12]. Summerville et al. trained an LSTM on Super Mario Bros. levels that included representations of potential player paths [21]. Follow up work by Summerville et al. extracted player paths from gameplay video and found that these led to significantly different output levels when they were used to train LSTMs [20]. We use a similar method to extract player paths. The major difference between Summerville et al.'s approach and ours is that they alter the level representation to include the path information. Instead, we generate novel paths and use these as input for a PCGML level generator. The Summerville et al. generator could not take a specified path as input without modification. This and much of the prior work mentioned above is based on the representations from the Video Game Level Corpus (VGLC) [24], which we draw on for our level representation.

3 SYSTEM OVERVIEW

We aim to show the benefits of human-like paths to improve global coherence in PCGML generators. Our approach can be divided into several steps: (1) extracting the paths from gameplay videos, (2) training an LSTM on the path data, (3) training our Markov chain

on the original Lode Runner levels, and (4) generating a new level from a generated path.

3.1 Data extraction

Our goal for this first step is to extract human paths for solving the original Lode Runner levels, which we extract from gameplay video. We focus on human paths instead of paths generated by an automated level playing agent as in prior work [21]. We made this choice as prior work found that human paths led to different output levels than automated paths [21]. Based on this prior work, we make the assumption that automated paths would lead to levels that were more sparse, but more human-like paths will lead to paths closer to the original game. We leave a verification of this assumption for future work.

We download a series of videos from YouTube, based on similar approaches in prior work [6, 21], Once we have the videos, we extract the frames and track the location of the player in each frame for each level, based on the same approach used in the above prior work. We tag each location with the type of movement using OpenCV and pattern matching [1]. We do this by hand tagging a series of images or sprites representing the different actions of the player in Lode Runner. In each frame, we identify the player's action or type of movement based on the image with the highest probability. We map the player's location in the frame to a 32 by 22 grid, which is the size of the VGLC dataset for each level. This gives us a sequence of 32 by 22 grids equal in length to the amount of time the player played a given level. After obtaining this sequence we can identify one out of five possible actions for each grid (moving left as 'l', moving right as 'r', climbing up 'u', climbing down 'c', and falling down as 'f') based on the location change of the player between pairs of grids. This allows us to store a player's path for each level as a one-dimensional sequence.

3.2 Path generation

We could have simply reused the extracted player paths as the input for a Lode Runner level generation process. However, this would have limited the number of levels our system could generate. As such, we need some way to generate new human-like paths.

Our extracted human paths vary significantly both in terms of length and patterns.[4] To address this, we first split the sequences of actions into small chunks of a fixed size (50). We represent each action as a one-hot encoding of length five for the five actions. Every action becomes a vector of length five, with every action type represented as an index in that vector. An action is represented by a 1 at its index if it occurred at this position in the sequence and a 0 otherwise. Thus our data becomes a series of 50x5 matrices.

We employ a Long Short-Term Memory Recurrent Neural Network (LSTM) for our path modeling, as they have been demonstrated to work well on PCGML tasks with sequence-like data [20]. An LSTM is designed to better learn long-term dependencies than standard recurrent neural networks. This is important for our use case as the human paths tended to have many repeated actions in a row, and we didn't want our model to learn to just repeat the same action endlessly.

We employ an LSTM-based Seq2Seq model that is built to generate new paths. Our input is one 50-length path sequence and the

expected output is the next 50-length path sequence. The model is composed of 2 layers with 512 LSTM cells each. We employ dropout and gradient clipping to prevent overfitting and gradient explosion, respectively. The final layer is a fully connected layer of length 50 with softmax activation, to better represent the probability distribution of a single character at each time step. We trained our model for 60 epochs with a 0.001 learning rate and the adam optimizer[8]. We probabilistically sample each output action based on treating the softmax activation as a probability distribution. We note that despite using 50 inputs and 50 outputs this model can be used to create paths of arbitrary lengths by inputting empty (all 0s) actions initially, and then continually generating based on previously generated outputs.

3.3 Level Structure Learning

The path information we generate only partially defines a level. The same level path could be associated with a large, but not infinite number of levels. This is particularly true in Lode Runner, where the player can directly "make" paths through their own actions (e.g. digging holes). This means that we still need more information about what kind of tiles can be associated with what action, and how to fill out the rest of the level "away" from the player path. Given our interest in demonstrating the impact of player paths in improving global coherency, we employ a multi-dimensional Markov chain [16]. In our multi-dimensional Markov chain each tile value depends on the tiles to its left and directly below, along with the player action at the current position, to the right, and above. We note that prior examples of platformer Markov chains have made use of nodes with 3 dependencies: the left, below, and to the left and below. We experimented with this model, but found that our 2-tile dependency sufficiently modeled level structure and led to increased diversity in the output. We train our multi-dimensional Markov chain on the Lode Runner levels from the VGLC [24], with the added information from our extracted paths in the grid-based representation discussed above.

We also record a series of statistics in terms of the number of enemies and gold pieces in the training levels. For each original level, we found the ratio of the numbers of enemies and gold pieces to the associated player path lengths. We represent these two distributions as two Gaussian distributions. This allows us to sample from these two distributions and derive an overall number of desired gold pieces and enemies for a new level.

3.4 Complete Level Generation

Our level generation process begins by generating a new path. From this path we can determine the minimum size of a level necessary to contain this path. We instantiate this level as an empty grid of the minimum size. We label every visited tile in this level with the action that the player path indicates, and the remaining tiles with a special token that indicates no actions. We automatically constrain some tile values based on the action type, based on the existing keys in the Markov Chain. For example, if the action taken at a tile is to move up, then we know that this tile must contain a ladder. Or if the action at a tile is to move to the left, then this tile can be empty, have a brick (as the player may break it from above and

fall to it, then move left), or a rope. This gives us some high-level constraints on the tiles and a basic initial structure to work with.

We apply our Markov chain to fill the level out with the final tile types from the partially specified state. We start from the bottom left corner and move along each row from left to right until we've reached the top of the screen. If a tile has been specified we do not need to generate a new tile at this location, and we just move on. If the tile has been partially specified then we remove the tile possibilities that have already been ruled out, and then probabilistically sample from the remaining options based on the learned probability distribution in the Markov chain. If the tile has not been specified at all, then we simply probabilistically sample from the Markov chain as normal [16]. In the event that a key does not exist, we remove all dependencies except for the left and below dependency and then sample from this simplified distribution. We made this choice as this removes all path requirements and only considers the structure of the level.

The final step of our generation process is to place all enemy and gold tiles. We randomly sample from both of the Gaussian distributions we described above. We multiply the sampled values by the generated path length to get the final numbers of enemies and gold pieces. We then randomly place these elements along the player's path, as they are elements that the player would have to avoid or seek out, respectively. We made this choice as the Lode Runner levels in the VGLC only place enemies and gold pieces in place of empty tiles, but both entities can move. Further, the Markov chain struggled to learn to place these elements effectively given how rare they are in comparison to the other tile types.

4 QUANTITATIVE EVALUATION

We extracted 66 player paths for 66 game levels from a 4.5-hour long gameplay video. After training on these sequences, we were able to generate an arbitrary number of levels. Figure 2 gives an example of a generated level based on our approach.

This research seeks to determine whether adding a player path as input to a PCGML generator improves global coherence. We employ Markov chains, a PCGML approach that tends to struggle with global coherence in order to better understand the impact of player paths [5]. As such, the natural choice for an initial evaluation is investigating how the inclusion of a generated path as input changes the generated levels in comparison to a Markov chain approach without player path information. We therefore compare against the work of Snodgrass and Ontanón, who employed a Markov chain without path information to generate Lode Runner levels [19].

We employed an A* pathfinding agent to test how a player might play the two different sets of Lode Runner levels. [4] Ideally, we might have used a human subject study to compare the two types of generated levels. However, we lacked the time and resources for a human subject study, and so use this method for an initial comparison. The pathfinder starts from the player tile present in both sets of generated levels and attempts to pathfind to each gold piece in turn. Lode Runner is a complex game, which allows players to momentarily trap enemies, as such, we ignore enemies while pathfinding. This decision was also motivated by the fact that we lacked a simulator to fully simulate enemy movement in the game, meaning that the enemies would just be treated as impassable

obstacles otherwise, which would not be appropriate. We note that prior work did not make this assumption, and so they reported much lower percentages of playable outputs [19]. However, since this assumption is made for both types of levels it's still helpful for comparison purposes. The pathfinder tracks the number of nodes explored on the way to each gold as well as each gold that it was able to reach. It reports the total number of each of those two metrics which we use to compare between the two types of levels.

Ideally, an A* agent should be able to reach each gold piece from the starting location in a playable Lode Runner level. We had the A* agent report the number of nodes explored on the way to each gold piece as a measure of coherence. All of the existing Lode Runner levels have clear paths to access each gold piece, essentially acting as puzzles for the player to solve. Therefore, we take a low number of nodes explore as an indirect measure of global coherence.

We employ the following metrics in this comparative evaluation, reporting the average and standard deviation of each metric across the generated levels:

- *Gold Total Per Level* - This is the average total number of gold pieces the A* agent needed to find. This is used in determining the percentage of gold collected in a level, as well as the potential difficulty and length of a level. The more gold pieces the more potential for difficulty depending on how they are distributed throughout the level. It also coincides with how long a level may take to play. The more gold, the longer it could take for a player to collect each piece.
- *Percentage Collected Per Level* - This metric gives how many of the gold pieces could be collected in a level. Values closer to 100% indicate more playable levels overall. A higher average indicates that more of the levels are playable.
- *Total Nodes Explored* - This metric is the number of nodes the A* agent needed in total to reach all of the reachable gold pieces. We do not include nodes explored when the pathfinder attempted to reach unreachable gold pieces. This number can be taken as an approximation for the minimum amount of time a player would need to complete a level. The higher the number the longer the time needed to complete the level.
- *Nodes Per Gold* - Since levels do not all have the same number of gold pieces, the total nodes explored metric on it's own could potential leave an inaccurate reflection of each level. As such, this metric gives the average number of nodes needed to reach each reachable piece of gold per level. The same rule applies as with the above metric, where the larger the number the longer it potentially could take to reach each gold piece.

These metrics allow us to compare between our two sets of generated levels. We are particularly interested in the second metric as a measure of playability and the fourth metric as a measure of global coherence. We report the other two metrics for context to these two metrics. If the Snodgrass and Ontanón levels outperform our levels in terms of playability, that could indicate that our method for placing gold pieces along the player path is flawed. If the Snodgrass and Ontanón levels outperform our levels or perform similarly in
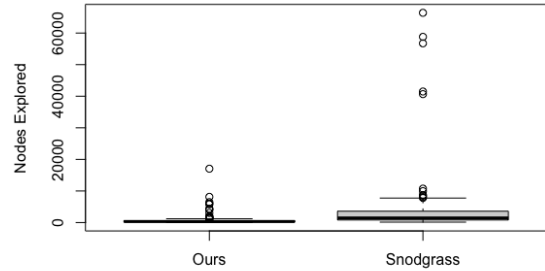

Figure 1: Boxplot Results of Nodes Explored Metric

terms of nodes per gold metric, this would indicate that our inclusion of the player path did not improve global coherence, and led to similarly coherent/incoherent levels as a simpler Markov chain approach.

5 QUALITATIVE RESULTS

Table 1 includes all of our results in terms of our four metrics. The first column gives the average and standard deviation of the total number of gold pieces included in each generated level. This immediately demonstrates the impact of employing a Gaussian distribution to model the total number of gold pieces in a level, as opposed to leaving it up to a Markov chain alone to place gold pieces. The Snodgrass and Ontanón levels have a much higher average and a much larger standard deviation. This indicates that their levels tended to have many more gold pieces on average compared to existing Lode Runner levels, and that this number varied to a great extent with the largest number of gold pieces being nearly three times as many for the Snodgrass and Ontanón levels. This is an initial indication that our approach led to more coherent levels.

The second column of Table 1 shows the average amount of gold that can be collected per level, indicating how many of the generated levels were playable. These values indicate that the Snodgrass and Ontanón levels are on average more likely to be playable. However, there is more complexity to this value than might first appear. Since on average our levels have less gold than the Snodgrass and Ontanón levels, the impact of being unable to reach a single gold piece is much higher. Collecting 6 of 7 available gold pieces will result in a lower percentage than 17 of 18 possible gold pieces. We also performed a Mann-Whitney U test to determine if these two distributions differed significantly. The test was unable to reject the null hypothesis ($p = 0.05629$) that these two distributions arose from the same underlying distribution. Thus, we take this to mean that the difference in terms of playability was insignificant. As mentioned above, this runs counter to prior reported playability values [19], which is due to our choice not to model enemy locations. Thus, this playability metric should be considered an upper bound.

The third column of Table 1 gives the Total Nodes Explored metric, which we also visualize in Figure 1 for clarity. It is immediately clear that the Snodgrass and Ontanón generator leads to massively

	Gold Total	Percentage Collected	Total Nodes Explored	Nodes per Gold
Snodgrass and Ontanón	18.76±15.07	98.94±6.98	4638.93±10837.32	220.62±477.89
Ours	7.65±2.98	94.68±19.07	910.33±2047.28	116.44±264.82

Table 1: Quantitative Evaluation Results

more explored nodes than our approach. However, this is not a fair comparison due to the higher average number of gold pieces among these levels and the larger variance of the number of gold pieces. Thus, we turn to the fourth column and the average number of nodes explored for each gold piece. This comparison is closer, only indicating that the Snodgrass and Ontanón levels require twice as many nodes to be explored to reach each gold piece. However, this difference is still substantial. This indicates that, for each gold piece, it would take a player roughly twice the time to collect it for an average Snodgrass and Ontanón level. This indicates that there is no clear path between the gold pieces in the Snodgrass and Ontanón levels. The standard deviation also signifies that our levels are more consistent in terms of this metric. We take this as an indication of the greater global coherency of our generated levels, that these levels include clear paths for the player to take to reach most gold pieces, even if the placement of some of these gold pieces makes them unreachable.

Our findings suggest that both our method and the method used by Snodgrass and Ontanón produce levels that are roughly equally playable using our A* agent that ignored enemy positions. Looking at the metric values it would have been very difficult to improve on this metric without achieving 100 percent playability. Our analysis did show that we improved the consistency and reliability of the solutions to levels.

6 QUALITATIVE EVALUATION

In this section, we asses the quality of our output levels in comparison to the original levels. We did this in order to get a more nuanced look as to whether we have actually achieved greater global coherence. Given the results of our first set of evaluations, it's clear that our approach was able to produce levels with clearer and more consistent solutions. However, it's possible that our output levels no longer resemble the original Lode Runner levels. For example, they may have become too simple or have lost other aspects of global coherence. We use the following metrics to investigate this possibility:

- *s* - The minimum size of the level to fit the path. Ideally, we'd like the levels to match the size of the original Lode Runner levels: a 32x22 grid. Since we do not explicitly enforce this, our hope is that our approach will have led to an implicit bias towards levels of this size.
- *e* - The proportion of the level taken up by empty space. The original Lode Runner levels have a fair amount of variance when it comes to empty space, and empty space is often used strategically to create shapes or to indicate potential solutions. Thus, if our distribution of empty tiles matches those from the original levels, this would indicate a positive signal in terms of similar global structure.

- *i* - The proportion of the room taken up by "interesting" tiles that are not simply solid or empty. It goes without saying that the way that the Lode Runner levels employ ladders, ropes, enemies, and gold pieces is very important to the overall design of the level.
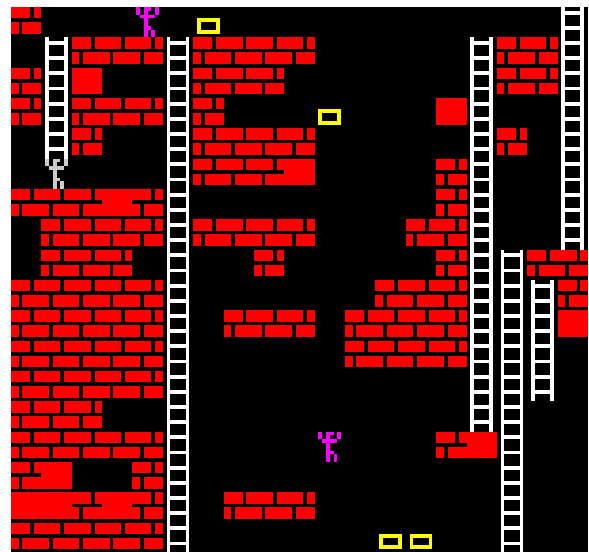
7 RESULTS

Figure 2: A good generated level.

We generated 34 generated levels and compared these to the 34 original levels that we did not use to train our model. We found that 20 of our generated levels matched the expected size of 32x22, with 14 of our levels having a smaller size. This means that these levels could be expanded to fit the expected size while retaining their same generated structure. This is a positive sign, as our approach was able to implicitly lead to levels with the same or similar sizes to the original levels without explicitly modeling this constraint. Notably, none of the generated levels were larger than the original levels, though this may be due to the fact that we employed a constant generated path size of 103 (which was the average of the paths we extracted from the gameplay video). However, a generated path of 103 steps could still have led to a level larger than 32x22.

Figure 3 and Figure 4 show the distribution of "interesting" and empty space tiles in the generated and original levels respectively. The distribution of "interesting" tiles does suggest that our levels tended to lead to fewer interesting tiles compared to the original levels. However, the overall distributions are fairly similar. Further, it's possible that due to the 14 smaller levels the generated levels look more conservative than they truly are since this distribution

does not take into account level size. The empty space distribution seems to differ more with the original levels employing much more empty space. This is not an unusual problem for Markov Chain models: filling in too much content. However, we again note that the 14 smaller levels may be part of the problem here. By filling the remaining space of these 14 smaller levels with empty space, the two distributions would look much more similar.
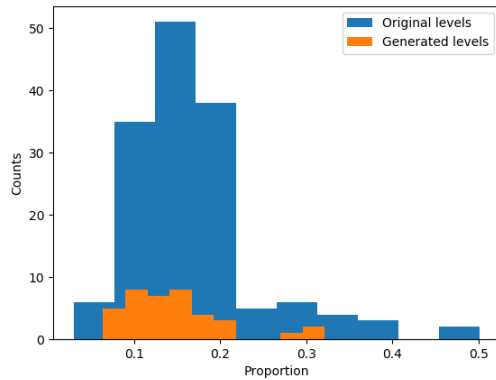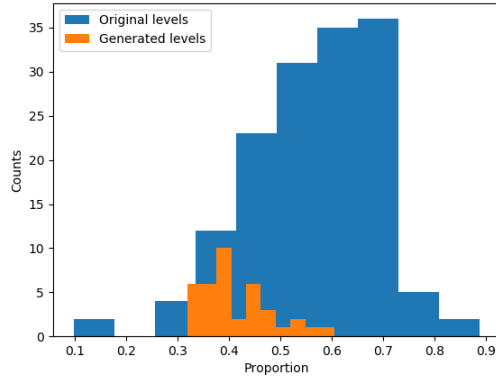

Figure 3: The distribution of "interesting" tiles.

Figure 4: The distribution of empty space.

8 DISCUSSION

There are 150 levels completed in the video we used for this paper. However, trimming and cropping each level from the video was time-consuming and repetitive. Therefore we just dealt with 66 levels to extract player paths. If all levels were processed, the results might be improved.

Extracting path information from videos has a couple of challenges. Our method works well for most levels, but the levels with lots of stairs and levels with fake bricks (where there appears to be a brick, but it is actually an empty space as the player steps on it) will not work very well. Due to the low image quality, when

the player walks passed the stairs, the combined figure becomes very difficult to recognize. This may sometimes lead to losing track of the player. The problem with the fake brick has a similar effect. When the player falls through a fake brick, the program fails to detect the player, hence losing track of the player.

It is a limitation of our approach that we always assume a fixed path length. While we made this choice for simplicity, the original levels did not all have the same path length. As such, it would be better to model this path length value separately. Alternatively, we could generate the path until we hit the desired level size and then stop.
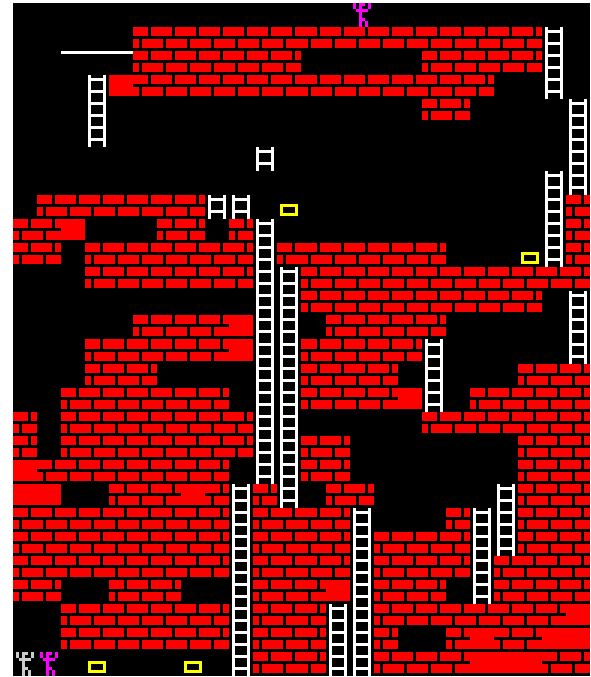

Figure 5: A bad generated level: badly placed enemies.

Figure 5 and Figure 6 show two typical issues that prevent the player from completing the generated levels. In Figure 5, an enemy is directly placed beside the player, and there is no other way to go around or trap the enemy. This shows that randomly placing enemies along the player's path is not ideal, we will also need to take consideration that the player needs to have a way to deal with enemies on the path. We ignored this problem for our pathfinding-based evaluation, but we will need to confront it for future work. Figure 6 had a different problem. The generated structure led to the player getting stuck. This happens when the row where the player is at and the row above it both have path information. Then when filling the tiles, the system would make a mistake that the bottom row can be bricks, and the player can dig along the upper row to create a path to the lower row. But this will not work if the player started from the lower row. This situation would trip up our A* pathfinder, and was one of the factors that led to our lower average playability score.

In this initial investigation we assumed controllability due to our input of player paths. While we can alter these paths and produce
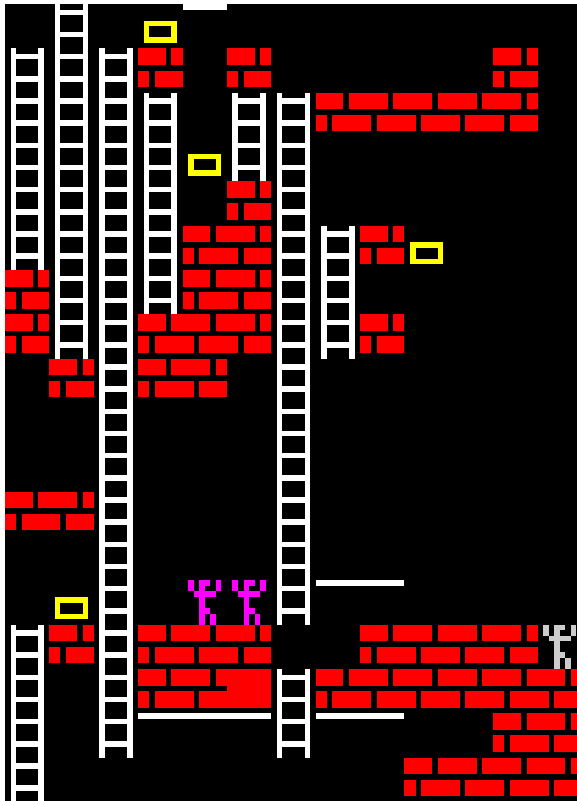
Figure 6: A bad generated level: badly designed structure.

new levels based on them, we did not include any evaluation of this aspect of our research. This would be a difficult thing to evaluate without a human subject study, and so we leave it for future work.

9 CONCLUSIONS

In this research project, we developed a player path-based method for the generation of Lode Runner levels. We extracted player paths from gameplay video to serve as training data, then used an LSTM Seq2Seq model to generate new player paths, and applied Markov Chains to produce new levels based on these paths. Our experimental results show that this approach can lead to improved global coherence of the generated levels while still leading to levels that share a resemblance to the original levels. For future work, we hope to improve the proposed method to ensure playability, to use a more sophisticated pathfinding agent that takes into account things like enemy placement, and to test this approach on other games.

ACKNOWLEDGMENTS

REFERENCES

[1] Gary Bradski and Adrian Kaehler. 2000. OpenCV. *Dr. Dobb's journal of software tools* 3 (2000).
[2] Eugene Chen, Christoph Sydora, Brad Burega, Anmol Mahajan, Abdullah Abdullah, Matthew Gallivan, and Matthew Guzdial. 2020. Image-to-Level: Generation and Repair. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 16. 189–195.
[3] Darui Cheng, Honglei Han, and Guangzheng Fei. 2020. Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm. In *International Conference on Entertainment Computing*. Springer, 37–50.
[4] Kynan Sorochan et al. [n.d.]. *Reference Code*. https://github.com/KynanS/ Generating-Lode-Runner-Levels-by-Learning-Player-Pathswith-LSTMs
[5] Matthew Guzdial, Nicholas Liao, and Mark Riedl. 2018. Co-creative level design via machine learning. *arXiv preprint arXiv:1809.09420* (2018).
[6] Matthew Guzdial and Mark Riedl. 2016. Toward game level generation from gameplay videos. *arXiv preprint arXiv:1602.07721* (2016).
[7] Ian Horswill. 2019. Imaginarium: A Tool for Casual Constraint-Based PCG. In *Proceedings of the AIIDE Workshop on Experimental AI and Games (EXAG)*.
[8] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.
[9] Antonios Liapis, Gillian Smith, and Noor Shaker. 2016. Mixed-initiative content creation. In *Procedural content generation in games*. Springer, 195–214.
[10] Han Cheol Moon, Tasnim Mohiuddin, Shafiq Joty, and Xu Chi. 2019. A unified neural coherence model. *arXiv preprint arXiv:1909.00349* (2019).
[11] Justin Mott, Saujas Nandi, and Luke Zeller. 2019. Controllable and coherent level generation: A two-pronged approach. In *6th Experimental AI in Games Workshop at AIIDE 2019*.
[12] Anurag Sarkar, Adam Summerville, Sam Snodgrass, Gerard Bentley, and Joseph Osborn. 2020. Exploring level blending across platformers via paths and affordances. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 16. 280–286.
[13] Anurag Sarkar, Zhihan Yang, and Seth Cooper. 2020. Controllable level blending between games using variational autoencoders. *arXiv preprint arXiv:2002.11869* (2020).
[14] Adam M Smith, Erik Andersen, Michael Mateas, and Zoran Popović. 2012. A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games*. 156–163.
[15] Sam Snodgrass and Santiago Ontañón. 2013. Generating maps using markov chains. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 9.
[16] Sam Snodgrass and Santiago Ontañón. 2014. Experiments in map generation using Markov chains.. In *FDG*.
[17] Sam Snodgrass and Santiago Ontañón. 2014. A hierarchical approach to generating maps using markov chains. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 10.
[18] Sam Snodgrass and Santiago Ontanon. 2015. A hierarchical mdmc approach to 2d video game map generation. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 11.
[19] Sam Snodgrass and Santiago Ontanón. 2016. Learning to generate video game maps using markov models. *IEEE transactions on computational intelligence and AI in games* 9, 4 (2016), 410–422.
[20] Adam Summerville, Matthew Guzdial, Michael Mateas, and Mark Riedl. 2016. Learning player tailored content from observation: Platformer level generation from video traces using lstms. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 12.
[21] Adam Summerville and Michael Mateas. 2016. Super mario as a string: Platformer level generation via lstms. *arXiv preprint arXiv:1603.00930* (2016).
[22] A. Summerville, Shweta Philip, and M. Mateas. 2015. MCMCTS PCG 4 SMB : Monte Carlo Tree Search to Guide Platformer Level Generation.
[23] 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.
[24] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontan'on Villar. 2016. The VGLC: The Video Game Level Corpus. *Proceedings of the 7th Workshop on Procedural Content Generation* (2016).
[25] Sarjak Thakkar, Changxing Cao, Lifan Wang, Tae Jong Choi, and Julian Togelius. 2019. Autoencoder and evolutionary algorithm for level generation in lode runner. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–4.