Procedural generation of rollercoasters

Jonathan Campbell
School of Computer Science
McGill University
Montréal, Québec
jonathan.campbell@mail.mcgill.ca

Clark Verbrugge
School of Computer Science
McGill University
Montréal, Québec
clump@cs.mcgill.ca

***Abstract*—The "RollerCoaster Tycoon" video game involves creating rollercoasters that optimize for various in-game metrics, while also being constrained by the need to ensure a feasible structure in terms of physical and spatial bounds. Creating these procedurally is thus a challenge. In this work, we explore multiple approaches to rollercoaster generation, including Markov chains and machine learning and reinforcement learning algorithms. We show that we can achieve relatively good tracks in terms of the game's measurement of success, and that reinforcement learning may give more control over other factors of potential interest. A focus on multiple measures allows our work to extend to other factors that also mimic actual player constructions.**

***Index Terms*—procedural content generation, machine learning, reinforcement learning, game AI**

I. Introduction

Procedural content generation (PCG) is used to automatically create game content such as level maps, graphics and other constituent pieces of games. In this work, we show the application of PCG to a part of the game RollerCoaster Tycoon (RCT). RCT is an amusement park simulator in which players build rollercoasters, amongst other rides. A rollercoaster track in RCT is built one segment at a time and must satisfy various physical properties (in terms of speed and self-intersection); after being built it is scored by certain game metrics that determine its popularity to simulated park guests. The generation process is thus complex, needing to satisfy multiple, quite disparate metric requirements, and successful generation of RCT tracks has not been previously performed.

To generate the rollercoaster tracks, we explore four different approaches, all of types commonly used in PCG research: Markov chains, two machine learning algorithms (Transformers and CNNs) and a reinforcement learning algorithm (PPO), and combine each with a lookahead and backtracking system for efficiency. The first three approaches are trained on a dataset of pre-built tracks, whereas the fourth is trained from scratch in a Gymnasium environment. To analyze the generated tracks we show their scores on the in-game metrics as well as other potentially interesting measures.

Specific contributions of this work include:

- an approach to procedurally generate rollercoaster tracks using different algorithms in a RollerCoaster Tycoon-like environment,
- a suite of metrics to analyze the generated content, and
- a Gymnasium environment in which reinforcement learning experiments for PCG in RCT can be conducted.

This work was supported by the Fonds de recherche du Québec.

II. Related Work

Below we explore prior work done with the main types of PCG algorithms we use. We also touch on work done in the similar domain of racing games and other work done in RCT.

Markov chains have often been used for platformer games. Dahlskog, Togelius & Nelson used a Markov chain based on n-grams to generate levels for Super Mario Bros [1]. They took vertical slices of game levels and extracted from them unigram, bigram and trigram occurrence counts to use in generating new levels. Snodgrass and Ontañón generated Mario levels one tile at a time [2] by conditioning a tile on some of its neighbors. They also used lookahead and fallback strategies to recover from sampling states not present in the level dataset. Our work similarly uses n-grams of track sequences to generate new tracks, also using fallback and lookahead strategies tailored to our domain. The same authors also extended their approach to Lode Runner and Kid Icarus in a further paper [3].

Machine learning has also been used for PCG in platformer games. Summerville and Mateas used LSTMs to generate Mario levels [4] and found that they are better able to generalize than Markov chains. Sorochan, Chen, Yu and Guzdial used a pipeline of LSTMs and Markov chains to generate Lode Runner levels from training on player path data [5].

A game domain more similar to ours is racing games, for which race tracks can be generated. Race tracks and rollercoaster tracks are both curves on which cars are driven, and speed of a car on the track is an important consideration. However, race tracks are typically two-dimensional and driver skill is a more significant factor than track physics, since racing cars are player controlled.

Togelius, Lucas and De Nardi investigated an evolutionary algorithm (EA) to generate race tracks [6]. They formulated tracks as sequences of Bezier curves, with each segment defined by two control points; mutation was done by changing the position of the control points. Fitness metrics were player-dependent and involved track difficulty and maximum speed.

Loiacono, Cardamone and Lanzi also used EAs to generate race tracks [7]. They introduced additional constraints to the track properties suggested in the above work, including that tracks must be closed circuits with a constrained curvature radius. Aiming to create a diversity of tracks rather than targeting a particular player profile, they evaluated tracks based on the entropy of their curvature and speed distribution.

Prasetya and Maulidevi used EAs to create race tracks [8], representing a track as a sequence of segments, each with a type, length, curve arc and radius. Their fitness function checked that the track was a closed, continuous non-intersecting circuit, and prioritized the proportion between curved and straight segments, the diversity of the curve arcs and distribution of the curves, and diversity of the speeds obtained by optimal driving at each track segment.

Reinforcement learning has also been used for PCG. Khalifa, Bontrager, Earle and Togelius proposed a framework to use RL to generate game levels [9]. Framing content generation as an iterative improvement problem, they trained a model to generate high-quality levels for 2D game environments such as Sokoban and Zelda. The framework was further expanded in a paper by Earle et al. that demonstrated how RL could be used to generate levels that aim for particular target level characteristics (e.g., number of crates on a Sokoban level) [10].

One paper that deals specifically with RCT is that by Cerny Green et al., who investigated ride and shop placement in a park in order to maximize park profit [11]. To this end, they introduced a simplified subset of the game called MicroRCT. Earle et al. used RL in this environment to create placements that could maximize park income and guest satisfaction [10].

There have also been some prior attempts to generate tracks for RCT, albeit not successful. Burke suggested the idea of using a genetic algorithm [12], but the implementation stalled as the game had not yet been fully reverse engineered at that time. An RNN model combined with human input was tried by Ebert [13], but was trained on only 34 tracks, did not take into account self-intersections or physics and did not consider the quality of the generated tracks.

III. Background

RollerCoaster Tycoon (RCT) is a series of best-selling simulation games created by Chris Sawyer. We briefly describe the game below with a focus on rollercoasters, track construction and in-game track metrics.

RollerCoaster Tycoon & OpenRCT2

In the series, players build and manage an amusement park. Rides, concession stalls, paths and scenery can be constructed. Simulated guests traverse the paths, go on rides and buy from the shops. The goal of a park can be to attain a certain profitability or number of guests, amongst others, and a free-play mode also exists. Figure 1 shows a game screenshot.

RollerCoaster Tycoon 2, released in 2002, remained popular for many years, but its compatibility with modern operating systems dwindled over time. OpenRCT2 [14], an open-source reimplementation of the game maintained by Ted John and other fans, serves as a faithful stand-in, and we use a modified form of it for our experiments, as discussed later.

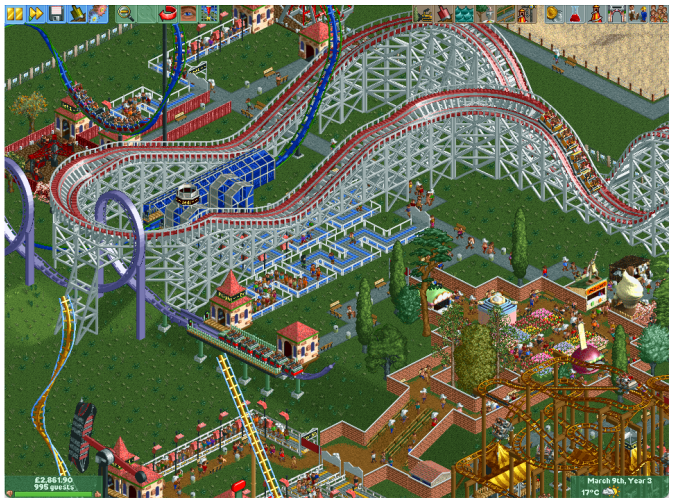

Fig. 1. A screenshot of part of a player-created amusement park in Roller-Coaster Tycoon 2. Five rollercoasters are visible, each in a different colour. Guest pathways and trees are placed throughout. Near the bottom-right is a garden area with concession stalls.

Many different kinds of rides can be built in the game. Some are simple and easily placeable like haunted houses, while others, such as rollercoasters, consist of tracks that are built one segment at a time. Players can choose out of 33 different types of rollercoasters, each with its own appearance and subset of track elements used in construction.

We limit generation to the wooden rollercoaster type in this work in order to limit training time. Extension to other types would be straightforward as our dataset contains tracks of all subtypes and our implementation supports all track elements. We choose wooden rollercoasters in particular since they occur most commonly in our dataset and support a wide variety of track elements. The wooden rollercoaster (or 'woodie') is also a classic from a historical point of view.

Track construction

Building a track is a complex activity in which both physics and entertainment value must be taken into account. A track is built by placing a sequence of track elements.

Each track element is made up of a sequence of individual 1x1x1 unit-sized pieces (or coordinates) on the 3D grid. A track element also has a starting and ending angle, starting and ending bank [1] and change in direction. For example, a regular flat element is made up of one piece and has no inclination, banking or change in direction, while a curved up-angled element may be made up of seven pieces taking up a total size of 5x5x3 units, with a starting and ending inclination of 25 degrees, no banking, and change in direction of 90 degrees to the left. There also exist special elements like

[1]A banked track piece is slightly raised on one side so that the car will tilt down to the other side. Banked tracks are used to reduce lateral G-forces (that cause a sensation of being pushed into the side of a car). They frequently occur when a curve is placed after a long drop.

loops, corkscrews, helixes, water splashes, brakes and pieces that move on the diagonal.

To construct a track, a player must place a sequence of elements that forms a closed, non-intersecting circuit in the three-dimensional grid. The starting angle (flat or 25/60/90 degrees) and banking (flat, left or right) of each element must match the ending angle and banking of the previous. The wooden rollercoaster supports 152 different track elements, though the actual number that can be built is much lower given this constraint. Each element also has a clearance height which must be observed.

Speed is also a factor. After a track is built, the game runs an empty car through it to check that it can make it to the end. If the car does not have enough speed to make it up a hill (or conversely goes too fast and flies off the track) and cannot reach the end, then the track cannot be ridden by guests and must be fixed by the player.

The great variety in the track design space, with different ride types, track elements and physics properties thus allows players to construct diverse coasters that give way to a wide range of rider experience.

Excitement, intensity and nausea

Excitement, intensity and nausea (EIN) ratings are calculated following the car trial run described above. They are used in part to determine a ride's attractiveness to guests, and thus profitability. Since rides cost money to build, a more profitable ride will allow the player to further enhance their park.

Ratings are positive real numbers that usually range from 0 to 13. Generally speaking, guests will prefer the ride with the highest excitement of a subset of rides in their vicinity. Each guest also has a preferred intensity range, the most common being 4 to 6, though some guests may prefer more gentle or extreme rides. An intensity above 10 is considered too high and will automatically penalize the ride's excitement rating. Guests also have a variable nausea tolerance; if the ride's nausea rating is too large, they will refuse to ride it. Other factors that come into a guest's decision include ride age, ticket price and weather, but we do not consider these for this work as they require a full amusement park context.

EIN ratings are determined through a complex weighted sum of many factors which depend on the particular type of rollercoaster. Some of the most important factors for excitement for wooden rollercoasters are average speed, track length, air-time (when riders feel lifted from their seats like they are floating, due to negative G-forces), number of drops and of turns, and adjacency of track elements to other parts of the same track, park pathways, other rides and tracks of other rides. Penalties are applied to the excitement rating if the ride has too few segments or drops, too low a maximum speed or maximum drop height, or excessive G-forces.

Intensity and nausea are likewise calculated using a weighted sum of many of the same factors listed above, though at different ratios. We omit the exact calculation as the factors and weights are too numerous to succinctly reproduce; we refer interested readers to the game's code.

IV. Methodology

We present our approach for generating rollercoasters below. We begin with a description and analysis of our dataset, followed by details on the generation algorithms, the lookahead and backtracking system, the work done after generation and a discussion of track metrics.

Dataset

Some of our models require training data, so we have collected a dataset of user-created rollercoaster track designs for RCT2. Our dataset has a total of 56,278 tracks split amongst 55 different ride types including rollercoasters, mazes, go karts, and mini-golf. The dataset is sourced from a variety of online track archives where users can upload their creations. For our current work, we consider the subset of 6,579 track designs built for the wooden rollercoaster.[2]

The dataset has a large diversity in track designs. Some have extreme drops and speeds, or conversely, long, monotonous flat stretches, while others are fastidiously designed. Since our goal is to generate 'good' tracks, at least by the game's definition, we train our generation algorithms only on those tracks that score well on the in-game metrics. Specifically, we filter out 2,427 tracks that have an intensity larger than 9, excitement lower than 4, or intensity larger than two times excitement. Further, a small number of tracks (22) are removed for being too short (< 50 pieces) or long (> 300) for a typical ride. We thus arrive at a total of 3,230 tracks for our training set. However, although these tracks meet the game's definition of success, it is not known whether they match to positive player opinions in terms of aesthetics or other factors but, having uploaded them, the authors are presumably proud of at least some aspect.

We also remove all scenery (trees, fences, signs and other landscape elements) from the tracks so that it is not a confounding factor in comparisons to our generated ones. Scenery can increase the ride's aesthetics to a player and can also have a limited impact on the excitement rating depending on its placement. We note that it would be trivial to decorate a track with scenery to increase the excitement in this manner (though likely not for player aesthetics).

Generation algorithms

Here we detail our four generation algorithms. The first three algorithms use our dataset and will generate probabilities for the next track element given the previous n; the fourth (reinforcement learning) does not use the dataset. Note that we start each track with a station platform (where guests enter and exit the ride).

Markov chain: To use a Markov chain, we first extract from our track dataset occurrence counts of track element sequences of length n. Then, to generate a next piece given the previous n pieces, we can use as probabilities the normalized occurrence counts of those n pieces. If the previous n pieces occurred

[2]We hope to make this dataset available, but have not done so given that the data is scraped from various sites leading to the need to verify permissions.

less than some threshold (set to 50 in our experiments), we decrease n by one and try again until we reach an amount over the threshold, so that our sample size remains adequate. We set n to 16 in our experiments. A track piece may also have additional properties (such as presence of chain lift, brake speed, and so on), so we encode each combination of a track piece and properties into a unique integer value for the purposes of the occurrence counts.

Transformer: The sequential nature of a track leads us to consider using a transformer. Transformers have had of late success recently in learning tasks involving natural language processing and sequential data [15].

Our network architecture is as follows. We take a vector of the past n track elements and associated flags (presence of chain lift, brake speed, etc.) and pass it through a positional embedding layer of 128 units. We then concatenate the output with a second input vector containing the current car speed, lateral G-force and vertical G-force. The concatenated vector is passed through a transformer block (with two attention heads, embedding dimension of 128 units and fully-connected layer of 128 units) followed by a fully-connected layer. The network output is a vector of probabilities, one per possible track element.

The model is trained on pairs from the dataset of sequences of n elements and the $n+1$'th piece. We set n to 8 in our experiments.

CNN: Although tracks can be represented as a series of integer embeddings (as is typical of text data in natural language processing), here there is also a spatial element to consider. Therefore, we introduce a second track representation: a three-dimensional boolean grid, where a cell is activated if that coordinate is part of the current track.

This grid alone is not sufficient to capture all the information in a track, since it is composed not only of a sequence of coordinates but also of angles, banks and other properties. Therefore, we stack seven more grids on top of the coordinate grid, each for a different track property: starting angle, ending angle, starting bank, ending bank, chain lift, brake speed (discretized to 16 values) and a final grid for rarer properties. We also pass in a flat vector containing the number of track coordinates, coordinates of the final track element, and speed of the car at the final element.

The size of the grids must also be considered. Tracks in general can be of length up to the size of the park but, for cost and space efficiency, are usually built within a certain area around the station platform. That said, tracks can still take on many different shapes: long and thin rectangles versus compact squares. To accommodate all sizes while keeping training tractable, we limit our grids to size 20^3. We center the grids around the coordinate of the current ending track element to attempt to capture the most locally important part of the track.

Our network architecture takes the eight grids and passes them through a stack of four 3D convolutional layers with 64 filters each, while the flat vector of additional information is passed through a fully-connected layer of 32 units. Outputs from the two layers are then concatenated and passed through two further fully-connected layers of 256 units and 164 units, respectively. The output of the network is a vector of probabilities, one per possible track element.

Reinforcement learning: Reinforcement learning (RL) is useful as a PCG method since it does not rely on pre-existing data. Equally importantly, it can learn to maximize an objective absent of the biases that may be lurking in existing data.

We use Proximal Policy Orientation (PPO) as our RL algorithm and train it in a Gymnasium environment for track generation[3]. Below we discuss the specifics of our environment: state and action space, network architecture, reward function and termination condition.

Our state space is comprised of a stacked grid, as in the above CNN approach, although here the grid size bounds the track size, thus making the track fully observable. The state space also includes two vectors of scalar inputs. The first vector consists of a series of target values (normalized between 0 and 1) for desired properties of the generated track, and the second vector consists of the current values for the properties. The properties include number of track segments, maximum speed, average speed, number of drops, amount of air-time, and excitement and intensity ratings. Target values are each initialized randomly from a specified range at the beginning of each episode. These vectors were suggested by Earle et al. [10] to control the content generated from an RL model for PCG. However, their experiments were done in environments in which target properties could be increased or decreased during an episode. Here, if a particular target value is surpassed, e.g., number of drops, it may not be possible to decrease it (since our action space does not allow for existing track elements to be removed or altered). However, including these vectors in state (and reward) allowed for more diverse output of tracks.

The grids are passed through a stack of 3D convolutional layers (with number of filters 8/16/16/16 and kernel size dependent on grid size, ranging from 3x3x3 to 5x3x5) and scalar vectors through a fully-connected layer (of 64 units), before being concatenated and passed through further fully-connected layers. The network output is a vector of length equal to the number of possible track elements.

For reward, we use the same approach suggested by Earle et al. [10]: the difference between the loss value of the previous and current step. Loss is defined as the sum of the difference between the target and current property value vector.

An episode is terminated if either the loss drops below 0.05, if more than 250 steps have been taken in the current episode, or if the number of current track coordinates plus the Manhattan distance from the current end of the track to the station platform has reached a certain threshold (thus placing a limit on the total length of a track).

Lookahead & backtracking

We combine our generation algorithms with a lookahead and backtracking system to handle the large number of invalid

[3]Our code can be found at https://github.com/campbelljc/trackrl.

track element combinations.

We do a lookahead at the same time we generate probabilities for the next track element. The lookahead checks available track elements and sets the probability to zero of any that are either incompatible with the previous piece (due to angle or bank mis-match), would intersect with a prior piece, fall outside of the track size boundary or would make a car's velocity reach zero. These checks are done using our own rudimentary implementation of the game's logic and physics engine. We then set probabilities of the invalid elements to zero before choosing the next piece to place.

If, after the lookahead search, there is no possible next piece to place, we then backtrack by removing elements from the end of our track, with the number dependent on an exponential backoff starting at n^0 pieces (we set n to 3 in our experiments, as it was often the case that a difficulty in placing a piece was caused not by the previous piece but by at least a few before it).

We note here that the combination of the lookahead and backtracking system was essential for our reinforcement learning approach to learn. Also, to ensure that the RL agent does not learn to backtrack infinitely (to prolong the episode and get more reward), we give a negative reward when a backtrack occurs equal to the sum of rewards of the backtracked steps.

Tidying up

Our generation techniques do not themselves guarantee a closed track. After a track is generated, it must therefore be extended to form a closed circuit that ends at the station platform. To do so, we first add brakes to decrease speed, then add up or down-angled pieces to reach the height of the station platform. An A$*$ search is then run to find a path to the platform. To make the search tractable, we limit track elements to flat pieces, standard three-tile turns and 's-bends' (which continue straight 4 tiles but are offset one tile to the left or right). If pathfinding fails or takes too long ($>$2,000 iterations), we throw out the track. Otherwise, a final sanity check is done by sending the track to a modified version of OpenRCT2 and, if the track passed the testing phase, receiving from it the excitement, intensity and nausea ratings.

Metrics

We use several metrics to evaluate the tracks generated by our approaches. Our primary metrics are those defined by the game: the excitement, intensity and nausea ratings, as discussed earlier. These are the ratings used by the simulated guests to decide whether or not to go on a ride, and are therefore the most salient in terms of gameplay. This situation is perhaps unlike that of other games in which PCG is used, where the judge of content is the actual player who is playing the level, seeing the graphics or racing on the race track.

However, it is also possible to define track quality outside of gameplay and instead still based on player experience. For example, it may be the case that certain tracks score high on the in-game ratings, but are not appealing or desirable enough for a player to want to include them in their park. That is, although such a track may be profitable and lead to success in-game, part of the game is about designing a good-looking amusement park, and therefore other, more aesthetic factors come into play. As a first attempt at a metric to capture something about this idea, we propose the use of visual novelty.

Visual novelty: When riding a rollercoaster, one can see a variety of different visual scenes: upcoming or previous parts of the track, other rides in the park and even geographical landmarks outside the park, if the rider is high enough. Drawing from research in urban planning on landscape visibility analysis, which quantifies perception of a landscape from a particular point or route [16] [17], we propose that part of the excitement of a rollercoaster ride involves the visual experience of the rider, and in particular, visual novelty, meaning newly-seen aspects of one's surroundings during the ride. We suggest that players of RCT may take this into account when building their tracks, even though it is not a factor for in-game metric values.

To approximate visual novelty, we propose a simplistic metric which measures at each track coordinate the amount of new track of the current rollercoaster seen by the rider. We consider only the current rollercoaster since we generate each track individually and not in the context of a park.

Although this metric is simple, it still allows for substantial differentiation of views. For example, for a track that starts by climbing up a hill, at the beginning perhaps only the uphill track in front of the rider will be visible. But when the rider reaches the top, they may suddenly be able to see a large amount of the coming track. We are interested in the number of times that the rider will experience a large visual novelty. To calculate the metric, we run a car through the track, and at each track piece, find all the track pieces visible to the rider within a line-of-sight cone of 120 degrees (extending in front of the rider in two dimensions) that have not been seen previously during the ride. (Bresenham's line algorithm is used to find the visible points within the cone.) We then take this vector of counts, normalize it, and count the number of pieces at which at least 10% of the track was newly visible.

Plagiarism: We also use plagiarism as a metric of success for our generation approaches. Plagiarism is important to quantify in PCG since we may want our content to be similar to existing content, but not in large part exactly the same. Snodgrass, Summerville & Ontañón, in a paper discussing the effect of training data on generation of Mario levels, introduced a plagiarism metric to analyze their output [18]. They calculated how many sequences of n vertical slices of a generated level were also present in the training dataset. For our domain, we report on co-occurring number of n sequential track elements.

V. Experimental Results

To test our generation approaches, we conduct various experiments. We discuss the parameters used in training and then present and discuss the results of our different approaches.

Model training

We split our dataset into training (70%), validation (15%) and testing (15%) sets and use it to train a model for our Transformer and CNN. Training was done for 10 epochs with a batch size of 1024. Adam was used as optimizer with a learning rate of 0.001. Training time for each model was about 10 minutes. Accuracy on the test set was 0.95 for the Transformer and 0.92 for the CNN model.

The RL model was trained using the RLlib implementation of PPO [19] on an M1 Max chip with TensorFlow 2 and eight rollout workers. Training was ended when the mean excitement of generated tracks reached a plateau, which came at about 230,000 timesteps or about 12 hours. The following track property ranges were used for episode initialization: 100 to 175 for track elements, 40 to 60 for maximum speed, 20 to 30 for average speed, 4 to 10 for drops, 75 to 150 for airtime, 8 (constant) for excitement and 4 to 9 for intensity.

Excitement, intensity and nausea

Tracks can in general be of a wide variety of sizes, bounded only by the amusement park size, but practically are often constrained for cost and space efficiency. An analysis of our dataset found that wooden rollercoasters occupied on average a 40 by 16 rectangle, with height 35, and 252 total track pieces. To compare our generation approaches to the dataset in a first experiment, we use the average size as a maximum bound and the number of track pieces as an exact bound. 1,000 tracks were generated per approach for this and all other experiments.

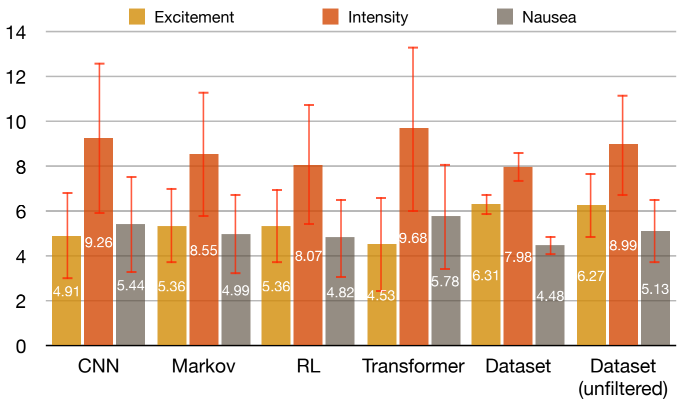

Fig. 2. Excitement, intensity and nausea ratings for the four generation approaches with a maximum track size of 40x16x35 and 232 pieces, and for the dataset (both the dataset used for training, and the unfiltered dataset including tracks of any EIN value). Error bars show the standard deviation. Note that higher excitement and lower nausea is better while the best range for intensity is between 4 and 6.

Figure 2 shows the excitement, intensity and nausea ratings as determined by the game for our four approaches and for the dataset. We can observe that the generated tracks for all approaches come close in all three ratings to the dataset's. Note that the standard deviation is lower for the dataset since tracks of low excitement or high intensity were explicitly filtered out so that our models could train on higher-quality data. We also include in the figure the ratings for the unfiltered dataset, which confirms that the filtering process reduced the variance.

All four approaches also have a tendency to sometimes produce tracks with intensities greater than 10 which are too extreme for guests. We corrected this issue in a further experiment by adding an additional constraint to the generation phase. The resulting trends for EIN ratings (not shown) were very similar except for a lower intensity and smaller variance.

We also run a second experiment to investigate the role of height in ratings. For each of our four approaches, we generate tracks in a square of maximum size 20x20 with 135 track pieces and a height of 10, 20 or 30. Figure 3 shows the excitement ratings for each approach and height value and figure 4 shows the intensity ratings. A different RL model was trained for each height.

As can be seen in both graphs, height has a significant impact on the ratings. In particular, when height is limited to 10 units, excitement and intensity are lower than their base ratings (3.2 and 2.6, respectively). Ratings are penalized by the game when a track has a low number of drops, low maximum drop height, or low maximum speed, all of which can be impacted by maximum track height.

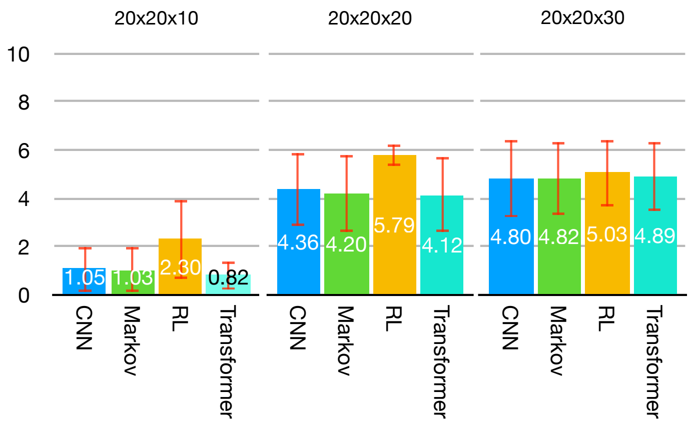

Fig. 3. Excitement ratings for the generation approaches with maximum track size of 20x20x10, 20x20x20 and 20x20x30 and 135 pieces. Error bars show the standard deviation. Higher excitement is better.

It can also be seen that the Markov, Transformer and CNN approaches fared worse than the RL approach. The former three were trained on our dataset which had very few tracks of low height (only 5% of its tracks had below 20 height, and none below 10 height), and therefore could not easily adapt to the size bound. By contrast, the RL approach did not use the dataset and instead sought to maximize its reward via (in part) the target values of excitement and intensity. The RL approach in particular did better with height of 20 for excitement than approaches for any height. As can be seen by the standard deviations for the height of 30, intensities with height of 30 can sometimes go over 10 (which leads to a large decrease in excitement), whereas height of 20 has less possibility to do the same, which may explain why the RL did best at that height. A sample high excitement and low height track generated from our RL approach is shown in figure 5.

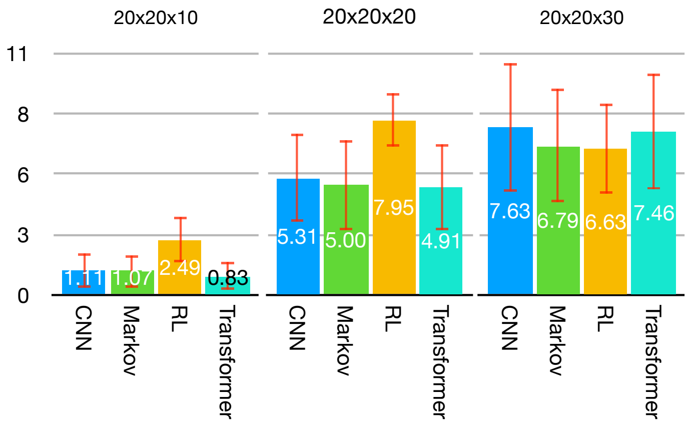

Fig. 4. Intensity ratings for the generation approaches with maximum track size of 20x20x10, 20x20x20 and 20x20x30 and 135 pieces. Error bars show the standard deviation. The best range for intensity is between 4 and 6.

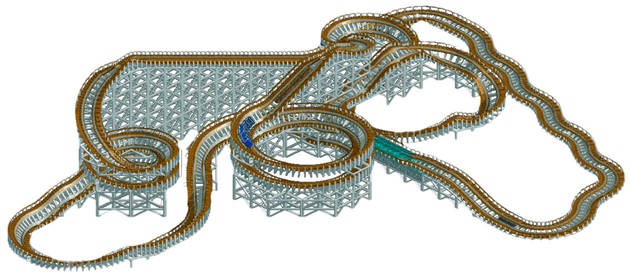

Fig. 5. An RL-generated track with EIN rating 6.63, 7.72 and 4.85. Note that the curvy section on the right side is the part created by the A$*$ algorithm to complete the track circuit.

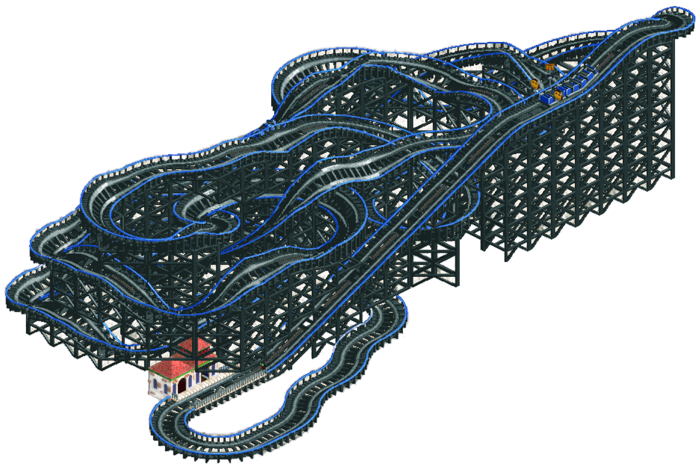

Fig. 6. Another RL-generated track, with EIN rating 6.50, 7.65 and 4.45. The ride begins at the station platform in the lower left.

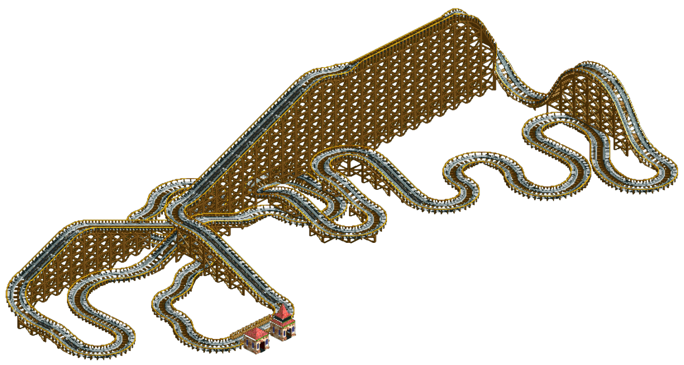

Fig. 7. A track generated by the Transformer model, with EIN rating 1.55, 15.14 and 9.39.

Beyond the metric scores, a manual inspection of generated tracks revealed certain issues. Some tracks were found to contain long stretches of flat segments at high elevation, which may be monotonous for riders and (at least in the real world) unnecessarily costly. One track, shown in figure 6, exhibited a very unusual property. The track begins with the car going up a hill with a chain lift, followed by a camera that takes photos of the riders and then a further hill. However, the car does not have the speed to get past the first piece of the latter hill, as it is missing a chain lift. The car thus starts to fall back until it reaches the chain lift of the previous hill, at which point it regains speed and begins to zig-zag between the two hills (and camera) for several minutes, until finally building up enough speed due to a quirk in the game's physics engine to continue up the second hill.

Figure 7 shows an example of a track with an intensity rating too extreme for guests. The intensity derives from the series of curves after the final drop on the right-hand side. The curves are not banked and the car takes them at a high speed, leading to a high lateral G-force experienced by the rider. However, if banking is added to the curves, the ride's intensity falls below 10, removing the excitement penalty. We note that a post-processing step for automatic curve banking may help temper these kinds of tracks.

Other metrics

Figure 8 shows the amount of plagiarism in our generated approaches, i.e., the percentage of track element sequences of length n that are also present in the dataset. We can note that the RL approach has significantly less of the same kinds of sequences contained in the dataset than the other approaches. This result is expected since the RL approach learns from scratch whereas the others were trained on the dataset.

The figure also shows the plagiarism for the dataset itself. It is computed by taking each individual dataset level and comparing it to all other dataset levels, then averaging as usual.

Results for our visual novelty metric are shown in Figure 9. As stated previously, tracks in our dataset may or may not map to 'good' rollercoasters in terms of rider experience or player opinion. The figure shows that the dataset scores highest on the metric, followed by the CNN approach, with the other three about tied, though the standard deviations are large. This metric is not a factor in the game, so it is not taken into account by guests when choosing a ride to go on. However, the results suggest that the novelty metric may capture something inherent

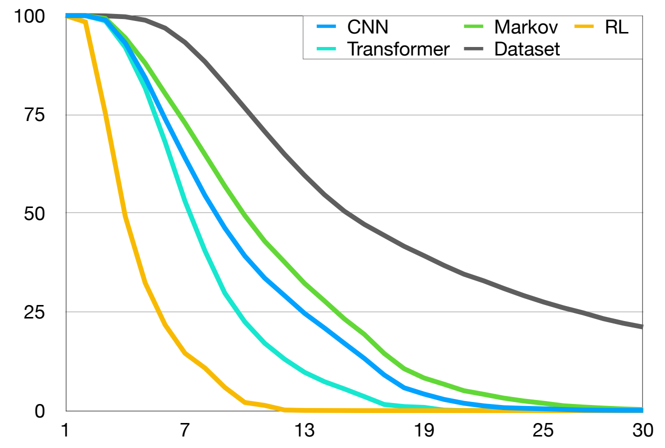

Fig. 8. Percentage of track element sequences that are present in the dataset, for sequences of length 1 to 30, averaged over all tracks generated from each approach for all experiments (4,000 each).

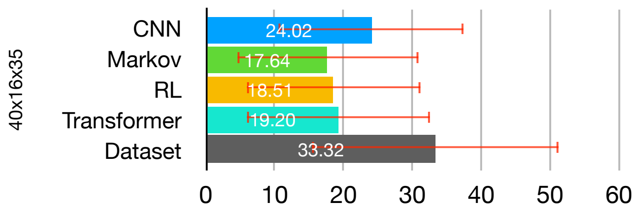

Fig. 9. Visual novelty metric scores for the four generation approaches with a maximum track size of 40x16x35 and 232 pieces and the dataset. Error bars show the standard deviation.

in the user-created tracks that is missing in our generated approaches.

VI. Conclusions and Future Work

In this paper we described an approach to procedural content generation of rollercoaster tracks in the video game RollerCoaster Tycoon. We proposed four generation algorithms (Markov chain, Transformer, CNN and reinforcement learning) combined with a lookahead and backtracking system. All algorithms worked well to produce tracks similar in terms of in-game metrics to those in a collected dataset of user tracks, while only reinforcement learning was able to adapt to the imposition of a low maximum height. The latter algorithm also tended to produce tracks containing sequences more different to those in the dataset than the other algorithms. Finally, we suggested a metric for visual novelty to aid in analysis of tracks.

A next step for this work could include generation using evolutionary algorithms (possibly by formulating the search space as a graph) or vision transformers. Other possibilities include extension of the RL reward function to complete the track circuit automatically, exploring further metrics to differentiate between tracks and conducting a user study to evaluate the proposed visual novelty metric. Finally, our methodology, including the use of reinforcement learning to generate constrained track sequences as well as the concept of visual novelty, could be repurposed for other sequential domains such as racing games.

References

[1] S. Dahlskog, J. Togelius, and M. J. Nelson, “Linear levels through n-grams,” in *Proceedings of the 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*. New York, NY: Association for Computing Machinery, 2014, p. 200–206.

[2] S. Snodgrass and S. Ontañón, “Experiments in map generation using Markov chains,” in *Proceedings of the 9th International Conference on the Foundations of Digital Games*, ser. FDG’14, 2014.

[3] S. Snodgrass and S. Ontañón, “Learning to generate video game maps using Markov models,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 4, pp. 410–422, 2017.

[4] A. Summerville and M. Mateas, “Super Mario as a string: Platformer level generation via LSTMs,” in *Proceedings of the 1st Joint International Conference of DIGRA and FDG*, ser. DIGRA/FDG’16, 2016.

[5] K. Sorochan, J. Chen, Y. Yu, and M. Guzdial, “Generating Lode Runner levels by learning player paths with LSTMs,” in *Proceedings of the 16th International Conference on the Foundations of Digital Games*, ser. FDG ’21. New York, NY, USA: Association for Computing Machinery, 2021.

[6] J. Togelius, S. M. Lucas, and R. D. Nardi, *Computational Intelligence in Racing Games*. Berlin: Springer Berlin Heidelberg, 2007, pp. 39–69.

[7] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Automatic track generation for high-end racing games using evolutionary computation,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, pp. 245 – 259, 10 2011.

[8] H. Adi Prasetya and N. Maulidevi, “Search-based procedural content generation for race tracks in video games,” *International Journal on Electrical Engineering and Informatics*, vol. 8, 12 2016.

[9] A. Khalifa, P. Bontrager, S. Earle, and J. Togelius, “PCGRL: Procedural content generation via reinforcement learning,” in *Proceedings of the 16th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE’20. AAAI Press, 2020.

[10] S. Earle, M. Edwards, A. Khalifa, P. Bontrager, and J. Togelius, “Learning controllable content generators,” in *Proceedings of the 2021 IEEE Conference on Games (CoG)*. IEEE Press, 2021, p. 1–9.

[11] M. Cerny Green, V. Yen, S. Earle, D. Rajesh, M. Edwards, and L. B. Soros, “Exploring open-ended gameplay features with Micro RollerCoaster Tycoon,” *arXiv e-prints*, May 2021.

[12] K. Burke. (2014) Hacking RollerCoaster Tycoon with genetic algorithms. Accessed on March 1, 2023. [Online]. Available: https://kevin.burke.dev/kevin/roller-coaster-tycoon-genetic-algorithms/

[13] D. Ebert. (2017) Neural RCT: Using recurrent neural networks to generate tracks for RollerCoaster Tycoon 2. Accessed on March 1, 2023. [Online]. Available: https://dylanebert.com/neural_rct/

[14] T. John. (2014) OpenRCT2. GitHub repo. Accessed on March 1, 2023. [Online]. Available: https://github.com/openrct2/openrct2

[15] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, “Transformers: State-of-the-art natural language processing,” in *Proc. of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[16] B. C. Chamberlain and M. J. Meitner, “A route-based visibility analysis for landscape management,” *Landscape and Urban Planning*, vol. 111, pp. 13–24, 2013.

[17] S. Ervin and C. Steinitz, “Landscape visibility computation: Necessary, but not sufficient,” *Environment and Planning B: Planning and Design*, vol. 30, no. 5, pp. 757–766, 2003.

[18] S. Snodgrass, A. Summerville, and S. Ontañón, “Studying the effects of training data on machine learning-based procedural content generation,” *Proceedings of the 17th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 13, no. 1, pp. 122–128, Jun. 2021.

[19] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, “RLlib: Abstractions for distributed reinforcement learning,” in *Proceedings of the 35th International Conference on Machine Learning*, ser. PMLR, vol. 80, 2018, pp. 3053–3062.