Procedural generation of rollercoasters

Jonathan Campbell and Clark Verbrugge

***Abstract*—The "RollerCoaster Tycoon" video game involves creating rollercoaster tracks that optimize for various 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 track generation through the use of Markov chains and various deep learning methods. We show that we can achieve relatively good tracks in terms of the game's measurement of success, and that reinforcement learning allows for more control of the generated tracks and for different rider experiences. A focus on multiple measures allows our work to extend to other track properties drawn from real-world research. This paper extends a previous publication by adding a new reward function for our reinforcement learning agent as well as further analyses of the generated tracks, including a metric measuring rider excitement over time, a revised novelty metric and an analysis of controllability.**

***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), an amusement park simulator in which players build rides such as rollercoasters. 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 metrics to determine its popularity with simulated park guests. The generation process is thus complex, needing to satisfy multiple and disparate metric requirements. Successful generation of RCT tracks has not previously been performed.

To generate 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 (Proximal Policy Optimization). The first three approaches are trained on a dataset of pre-built tracks while the fourth is trained from scratch. To analyze the generated tracks we score them on the in-game metrics as well as on other measures including excitement and novelty over time.

Specific contributions of this work include:

- an approach to procedurally generate rollercoaster tracks using different algorithms in an RCT-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 builds on a previous publication wherein we described the basic PCG approaches [1]. Here we significantly extend that work with a new reward function for our reinforcement learning approach which improves the quality of the generated tracks, two new metrics to provide additional insight into track properties and additional analysis of the tracks that show the controllability of our approach.

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

J. Campbell and C. Verbrugge are with the School of Computer Science, McGill University, Montréal, Québec, Canada (e-mail: jonathan.campbell@mail.mcgill.ca, clump@cs.mcgill.ca).

...

II. Related Work

Below we explore prior work done with our selected PCG algorithms. 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 [2]. They took vertical slices of game levels and extracted from them occurrence counts to use in generating new levels. Snodgrass and Ontañón generated Mario levels one tile at a time [3] by conditioning a tile on some of its neighbors, and extended their approach to Lode Runner and Kid Icarus in a further paper [4]. They also used lookahead and fallback strategies to recover from sampling states not present in the level dataset. Our work similarly uses a Markov chain of n-grams of track sequences as one way to generate new tracks, in our case conditioned on the past n track pieces instead of grid neighbours, and also using fallback and lookahead strategies tailored to our domain.

Machine learning has also been used for PCG in platformer games. Summerville and Mateas used LSTMs to generate Mario levels [5] and found them 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 [6].

A game domain more similar to ours is that of racing games and their race tracks. 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 in both. However, race tracks are typically two-dimensional and thus player driving skill is a more significant factor than track physics.

Togelius, Lucas and De Nardi investigated an evolutionary algorithm (EA) to generate race tracks [7]. 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 [8]. 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 track designs rather than targeting a particular player profile, they evaluated tracks based on the entropy of their curvature and speed distribution.

More recently, reinforcement learning has also been applied to 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 level generator for 2D game environments like Sokoban and Zelda. In a further paper, Earle et al. showed how the approach could generate levels aiming for particular characteristics (e.g., number of crates on a Sokoban level) [10]. Level generation is typically a forgiving environment for PCG since any one generated tile may not make or break the quality of the entire level and, similarly, there is typically a small action space to explore (e.g., in Sokoban, a tile can typically only be a boulder, wall or empty space). A rollercoaster track is a much more constrained setting for PCG in which a single piece (out of a large number of possibilities) could have large ramifications either immediately or much later on in the generation process, leading to different challenges.

In terms of RCT specifically, Cerny Green et al. studied ride and shop placement to maximize profit in a simplified subset of the game [11]. Earle et al. used RL in this same environment to create placements that could maximize park income and guest satisfaction [10].

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

Part of our work quantifies the visual novelty seen by a rollercoaster rider. This line of research draws from work in urban planning on landscape visibility analysis, which quantifies perception of a landscape from a particular point or route [14] [15]. Novelty has also been explored in tourism and hospitality research. In particular, Chang, Shu and King study novelty in theme parks and conclude in part that novelty of physical surroundings has a positive effect on guests' perception of the theme park [16]. Novelty was assessed in terms of facility aesthetic (building architecture, size, colour schemes, etc.) and ride layout and placement. However, they did not assess a rider's experience of novelty during a ride.

We also explore in our work a metric for the excitement experienced by a rider over time. Bastiaansen et al. measured the emotional arousal (a possible correlate for excitement) of a virtual reality rollercoaster rider by recording skin conductance responses. They found in part that responses increased during accelerations and end braking [17], but that the data could not predict final rider evaluation.

III. Background

RollerCoaster Tycoon (RCT) is a series of best-selling amusement park simulation games created by Chris Sawyer. Rides, concession stalls, paths and scenery can be constructed. Simulated guests traverse the paths, go on rides and buy from the shops. Figure 1 shows a game screenshot. Many kinds of rollercoasters can be built in the game; we limit this paper to generation of wooden rollercoasters, the most common rollercoaster type. For a more detailed discussion of the game, our original paper may be consulted [1].

Fig. 1: A screenshot of part of a player-created amusement park in RCT2. Five rollercoasters are visible. Guest pathways and trees are placed throughout. Near the bottom-right is a garden area with concession stalls.

RollerCoaster Tycoon 2, released in 2002, remained popular for many years, but its compatibility with modern operating systems dwindled over time. OpenRCT2 [18], 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.

In OpenRCT2, as in RCT2, a rollercoaster is built by placing a sequence of track segments. The sequence must form a closed, non-intersecting circuit on the three-dimensional grid.

Fig. 2: Some common track segments in RCT2. Some segments occupy a 1x1 square while others consist of multiple pieces. Each segment is separated by an empty space for visualization purposes, but in a real track would be connected.

There are 152 wooden rollercoaster track segments. Each is made up of a sequence of individual unit-sized pieces and has some combination of a starting and ending angle (flat or 25/60/90°), starting and ending bank (flat, left or right) and change in direction. Certain segments can also optionally have a chain lift which can pull cars up a hill. Figure 2 shows some common segments.

When placing a segment, its starting angle and banking must match the ending angle and banking of the previous; each segment also has a clearance height which must be observed. Speed is also a factor. The physics of a track must allow a car to make it up every hill, while at the same time not go so fast that it flies off the track. G-forces are also calculated; their importance will be discussed later.

Building a track is a complex process in which both physics and amusement value must be considered. At the same time, the great variety in the track design space with numerous track segments and physics properties allows players to construct coasters that give way to a wide range of rider experience.

IV. Methodology

We present our approach for generating rollercoasters below. We begin with a description and analysis of our dataset, followed by details on how we represent tracks, our generation algorithms, the lookahead and backtracking system, track circuit completion and testing, and a discussion of track metrics.

A. Dataset

Our generation approaches, with the exception of reinforcement learning, require training data, so we have collected a dataset of user-created rollercoaster tracks for RCT2. Our dataset has a total of 160,214 tracks split amongst 87 different ride types including rollercoasters, mazes, go karts, and minigolf. The dataset is sourced from a variety of online track archives where users can upload their creations. Sources for the dataset are available at our GitHub repository[1]. For our current work, we consider the subset of 10,814 tracks built for the wooden rollercoaster.

The dataset has a large diversity in track designs. Some have dire 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 select only those tracks that score well on the in-game metrics. Specifically, we exclude duplicate tracks, tracks which did not run in-game due to construction error, tracks with intensity larger than 9, excitement lower than 4 or intensity larger than two times excitement, and excessively short or long tracks. We thus arrive at a total of 4,824 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.)

An analysis of the training set finds that its tracks contain all possible wooden rollercoaster track segments, but not at the same frequency. As may be expected, segments that are used in common building patterns (e.g., hills and slopes) are much more common than more situational ones. For example, a standard 25-degree upward segment occurs as 16.86% of all segments, while a 3-tile right-quarter turn segment angled down 25 degrees and ending on a right bank occurs as only 0.01% of all segments. It is thus expected that models trained on this dataset will not adequately understand the use cases of this and other low-occurring segments, though such segments have use in very few construction scenarios and may be important more for aesthetics.

Tracks in the training set occupy on average a 40 by 18 rectangle with height 35, and contain an average of 262 total track pieces (with a standard deviation of 88.7).

B. Track representations

We use three different kinds of track representations for our generation algorithms: integer, grid and piece-based.

In a simple representation, each track segment is mapped to two integers: one describing its type and the other for special properties (e.g., presence of a chain lift or brake speed). A sequence of these integer pairs forms a full track. The geometry of the track is thus not represented. Our Markov chain and Transformer approaches use this representation.

For a more spatial representation, we encode tracks into a series of eight three-dimensional grids. The first grid indicates whether a cell is occupied by a track piece; seven further grids are used to capture the properties of a piece: starting angle, ending angle, starting bank, ending bank, chain lift, discretized brake speed and a final grid for rarer properties. An example of one of these grids (the starting bank) can be seen in Figure 7 b). Our CNN and RL approaches use this representation.

The size of the grids must also be considered. Tracks can be of any length up to the size of the park but, for cost and space efficiency, are often built within a certain area around the station platform. To accommodate all sizes while keeping training tractable, for our dataset we limit grids to size 20^3, and centre them around the end of the current track to attempt to capture the most locally important information.

Combined with either the integer or grid-based representation, we can obtain a finer level of detail by using sequences of unit-sized pieces instead of segments. Although the game displays and constructs tracks as sequences of segments, internally the game divides them into individual unit-sized pieces. This representation has the advantage of allowing for more discretized calculations of track ratings and physics for each single piece, required for our linear reward RL approach.

C. Generation algorithms

Here we detail our generation algorithms, which generate probabilities for the next track segment given the previous n. Tracks will start with a pre-built station platform, where guests enter and exit.

Markov chain: For this approach, we extract from our track dataset occurrence counts of track segment sequences of length n encoded in the integer representation. Then, to generate a next segment given the previous n, we can use as probabilities the normalized occurrence counts of those n segments.

With a large n, and thus large sequence of segments, it is likely that we will end up generating a near exact replica of a dataset track. Therefore, to maintain an adequate sample size while capturing as much of the past track as we can, we set n to a high number initially (32, the same as our next approach) and then decrease it until the sequence of n pieces occurs at least a certain number of times (set to 50 in our experiments, a relatively arbitrary value which could be fine-tuned).

Transformer: The sequential nature of a track leads us to use a transformer. Transformers have of late had great success in learning tasks involving sequential data [19].

For our network architecture, we take a vector of the past n track segments encoded in the integer representation 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 and G-forces. The resulting vector is passed through a transformer block (with two attention heads, embedding size 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 for each track segment.

The model is trained on pairs from the dataset of sequences of n segments to predict the $n+1$'th segment. We set n to 32 in our experiments so that training time would be tractable. Class weighting is used to offset the imbalance in frequency of track segments present in the dataset.

CNN: We use a convolutional neural network to train on our grid-based track representation. Our network takes two inputs: the stack of eight grids and a flat vector containing the number of track pieces, coordinates of the final track segment, and speed of the car at the final segment.

The eight grids are passed 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 segment. Class weighting is also employed here.

Reinforcement learning: We use Proximal Policy Optimization (PPO) as our RL algorithm and train it in a Gymnasium environment for track generation[1]. Below we discuss the specifics of our environment: state and action space, network architecture, termination condition and reward functions.

Our state space is comprised of a stacked grid, as in the CNN approach, though here the grid fully captures the track so that the state is fully observable. The state space also includes two real-valued vectors. 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 said properties. The properties include number of track pieces, 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 used by Earle et al. [10] to control the content generated from an RL model for PCG. Their experiments were done in games in which target properties could both increase and decrease in an episode. Here, certain properties (such as number of drops) can only increase and not decrease, since our action space does not allow for track segments to be removed; thus, it is likely that learning to attain these targets will be more difficult. However, using these vectors generated more diverse tracks.

The grids are passed through a stack of four 3D convolutional layers (with filters 8/16/16/16 and kernel size dependent on grid size, ranging from 3x3x3 to 5x3x5) and real-valued vectors through a fully-connected layer (of 64 units), before the two are concatenated and passed through further fully-connected layers. The action space is a vector of length equal to the number of possible track segments (152). As we note in the following subsection, we mask the action space to handle invalid actions.

An episode is terminated when the number of current track pieces 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).

We base our reward function on that used 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 vector of track properties.

This reward function will lead to tracks that maximize excitement as early as possible in the ride. Since excitement can saturate after a certain amount, the excitement over time for generated tracks will likely thus stagnate after reaching its maximum, and the rest of the track may be more monotonous. To attempt to generate tracks with a more consistent excitement over time, we present a second reward function which we call the linear reward function. Here, we modify the reward by setting our target properties at each timestep to be proportional to the current number of pieces built. That is, at the beginning, our target excitement rating (amongst others) will be zero, while at the end of the track, the targets will have been scaled up to their full values. Excitement will thus be rewarded as long as not too much is gained at once. We use the unit-sized piece track representation for this reward function since it relies on the amount of excitement gained at each timestep.

D. Lookahead & backtracking

We combine our generation algorithms with a lookahead and backtracking system to handle the large number of invalid track segment combinations. A lookahead search determines which segments can connect to the current end of the track. Using a rudimentary implementation of the game's logic and physics engine, compatibility of the angles and banks, self-intersection and negative speed are all checked. The probabilities for these segments are set to zero. Given the large number of invalid segments at any timestep, this form of action masking, a common practice to increase RL learning efficiency [20], was essential for our RL agent to learn.

If, after the lookahead search, no segments can be placed, then we backtrack by removing segments from the end of the track, with the number based on an exponential backoff starting at n^0 pieces. We set n to 3 in our experiments, since

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

difficulty in placing a piece is often caused not by the last segment but by at least a few preceding it. To ensure that our 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.

E. Track circuit completion and testing

Tracks must form a closed circuit that begins and ends at the station platform, but our generation techniques do not guarantee this property. Therefore, after generating a track of a specified length, an A$*$ search is run to bring the track back to the platform. Brakes are added first to decrease speed followed by a drop to reach the ground; the search then uses flat pieces, standard three-tile turns and 's-bends' (which continue straight four tiles and end offset one tile to the left or right) to find its way back. If pathfinding fails or takes too long, we throw out the track. Otherwise, a final sanity check is done by sending the track to a modified version of OpenRCT2. We note that the number of additional track pieces added by A$*$ was found in our experiments to be very consistent on average, and its contribution to a track's ratings to also be consistent and limited.

F. Metrics

We use several metrics to evaluate the tracks generated by our approaches. We begin with those defined by the game, followed by controllability, plagiarism and two tailored for our domain: novelty and excitement over time.

Excitement, intensity and nausea: Our primary metrics are those defined by the game: the excitement, intensity and nausea (EIN) ratings. These are used in part to determine a ride's popularity with guests (and thus profitability), and are therefore the most salient in terms of gameplay. EIN ratings are positive real numbers that usually range from 0 to 13. They are determined through a weighted sum of different ride physics factors, such as speed and G-forces, though each factor caps at a maximum level so as to not dominate the others. Generally speaking, guests prefer rides with higher excitement and lower nausea. Each guest also has a preferred intensity range, the most common being from 4 to 6. We offer a more detailed discussion on ride ratings in our original paper [1].

Controllability: Controllability is an important property in PCG. We will analyze the controllability of our RL approach by defining certain fixed sets of target properties: one with more gentle values (lower airtime, lower speed, etc.) and one with more intense values. We will then generate tracks given these two sets (instead of varying the targets within their ranges at the start of each episode) and compare the properties of the resulting tracks to the targets.

Plagiarism: We also use plagiarism as a metric of success. Plagiarism is important to quantify in PCG since we do not want our content to exactly reproduce the dataset. 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 [21]. 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 segments.

Visual novelty over time: Although EIN ratings are the most important for gameplay, it is also possible to define the quality of a track based on other factors. For example, even if a track scores high on the in-game metrics, it may not be appealing enough for a real guest to want to ride it. To attempt to capture something beyond the in-game metrics, we propose the use of visual novelty over time (and later, excitement over time).

A rider on a rollercoaster will see a variety of visuals: upcoming or past parts of the track, other rides in the park and part of the geography in or outside the park. Drawing from work in urban planning on landscape visibility analysis, we propose that part of the excitement of a rollercoaster involves the visual experience of the rider, and in particular, visual novelty, meaning newly-seen aspects of one's surroundings.

To approximate visual novelty of a track, we propose a simplistic metric which counts at each track piece the amount of new track of the current rollercoaster seen by the rider. To calculate the number of newly visible pieces, we find, at each track piece, 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. Then, to compare the visual novelty between tracks of different sizes, we take the curve of cumulative number of new points seen over time and simplify to the highest common minimum number of equidistant points. We then calculate the mean curve of tracks from each approach.

This metric is different from the one suggested in our original paper, which compared the number of times at which at least 10% of the track was newly visible. This new formulation better captures the totality of the novelty of a track by taking the mean curve over all pieces, not just at certain points.

Excitement over time: In order to investigate our linear reward RL approach, we define a new metric to calculate the excitement of a rider over time. Although, like visual novelty, excitement over time has no in-game impact, it represents an aesthetic consideration that a designer may consider in creating a more realistic track: a ride that starts out very exciting but then quickly turns monotonous has less appeal than one that slowly builds in excitement throughout the ride.

Excitement in RCT can plateau, since many of its factors saturate to maximum values; therefore, it may not be the best game analogue for this metric. However, it is arguable that excitement in real life may also plateau or, at least, that diminishing returns may apply (e.g., a rider may find each subsequent drop less exciting than the last). Thus, it can still be instructive to weigh a ride's excitement rating over time.

To compare excitement over time between tracks of different sizes and approaches, we perform the same processing as for our novelty curves above. To further analyze this metric, we cluster the excitement curves. Curves can be clustered using a path distance metric such as the Fréchet distance [22]–[24]; we do so here and use a k-means clustering algorithm.

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.

A. 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 30 minutes. Accuracy on the test set was 0.95 for the Transformer and 0.91 for the CNN model.

The RL model was trained using the RLlib implementation of PPO [25] 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 192,000 timesteps or about 4 hours. Hyperparameter values included 5e-6 for learning rate, 0.4 for clip parameter, 1.2 for value function loss coefficient, 0.16 for KL coefficient and 0.01 for KL target, all determined through a grid search in a smaller-sized environment.

The track property ranges used for RL episode initialization were 30 to 50 for maximum speed, 15 to 25 for average speed, 6 to 10 for drops, 60 to 120 for airtime, 10 (constant) for excitement and 4 to 6 for intensity. These were calculated partly based on the average values for our dataset, then lowered to make less intense (and thus more popular) tracks. For example, there were about 10 drops on average in a dataset track; we chose a range from 6 to 10. Similarly, the average intensity for dataset tracks was 8.1, whereas we chose a range from 4 to 6 (the most common preferred intensity range).

B. 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. To compare our generation approaches to the dataset in a first experiment, we use the average size of a dataset track (40x18 with height 35) as a maximum bound and the average number of track pieces of a dataset track (262) as an exact bound. 1,000 tracks were generated per approach for this and all other experiments.

Further, in testing, most approaches had a tendency to sometimes produce tracks with intensities too extreme for guests (greater than 10). We corrected this issue by adding an additional constraint to the generation phase to prevent segments from being placed that would cause the intensity to increase past 10. Testing confirmed that this fix eliminated the clusters of tracks with these intensities.

Figure 3 shows the excitement, intensity and nausea ratings for our generation approaches and the dataset. We can observe that the generated tracks for all approaches come close in all three ratings to the dataset. Note that the confidence intervals are tighter for the dataset since tracks of low excitement or high intensity were explicitly filtered out. We also include in the figure the ratings for the unfiltered dataset, which confirms that the filtering process reduced the variance.

One outlier amongst the approaches is RL with linear reward, which produces very slightly less exciting tracks than RL with default reward (5.34 vs. 6.15, delta of 0.81) but significantly less intense (6.14 vs. 8.24, delta of 2.10) and nauseating tracks, bringing the intensity closer to the optimal range of 4 to 6 and making it the only approach to improve on the dataset in terms of ride popularity (since the slightly lower excitement is more than offset by the larger number of guests who will consider the ride due to its lower intensity).

This difference might be explained by the RL training process. Since excitement saturates to a maximum and cannot increase beyond that, the agent will often learn to include a certain fixed or semi-fixed set of pieces at the beginning of its track to obtain the maximum excitement, then add pieces that meet the other required target ranges (drops, speed, etc.). However, the easiest way to reach maximum excitement is through sequences that also cause similar increases in intensity. The RL agent with linear reward, by contrast, does not seek to maximize excitement right at the start of a track, and may therefore be better able to explore the state space and add excitement in lower intensity sequences throughout the ride. We also note that the linear reward agent trained at a much slower rate and took about 24 hours to reach a plateau on mean excitement compared to 4 hours for the regular RL agent.

We also run a second experiment to investigate the role of height in ratings. For each of our approaches, we generate tracks of 135 pieces in a 20x20 square of height of 10, 20 or 30. Track height was controlled by preventing upward-sloped pieces from being built if they would exceed the specified height. Figure 4 shows the excitement ratings for each approach and height value and figure 5 shows the intensity ratings. A different RL model was trained for each height.

As can be seen in both charts, height has a significant impact on ratings. In particular, when height is limited to 10 units, excitement and intensity are almost uniformly 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.

It can also be seen that the RL tracks had slightly higher excitement than the others for all three heights. The other approaches were trained on our dataset, which had very few tracks of low height (only 9.1% of its tracks fit within height of 20, and none within height of 10), and therefore likely could not easily adapt to the size bound. By contrast, the RL approach learned on its own and did not use the dataset. However, the increase in excitement comes at a cost of a correspondingly higher intensity than the other approaches, likely since the properties that add to excitement also add to intensity. A sample high excitement, low height track generated from our RL approach is shown in figure 7.

C. Controllability

We generate 1,000 tracks for each of our two sets of target properties (more gentle and more intense) using the linear RL approach. Note that we do not train a new model for each of the two sets of targets, but use the same model trained for the earlier experiment (Figure 3).

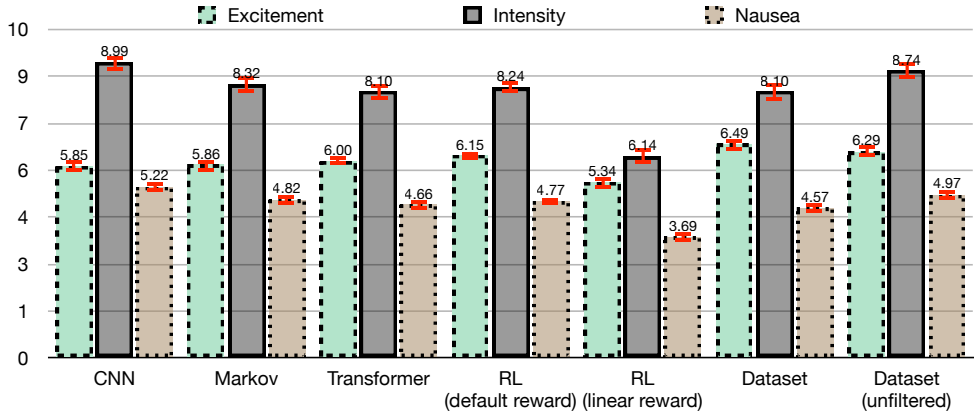


Fig. 3: Excitement, intensity and nausea ratings for the generation approaches with a maximum track size of 40x18x35 and 262 pieces, and for the dataset (both the one used for training, and the unfiltered version including tracks of any EIN value). Error bars show 95% confidence interval. Higher excitement, lower nausea and intensity between 4 and 6 make a ride popular.

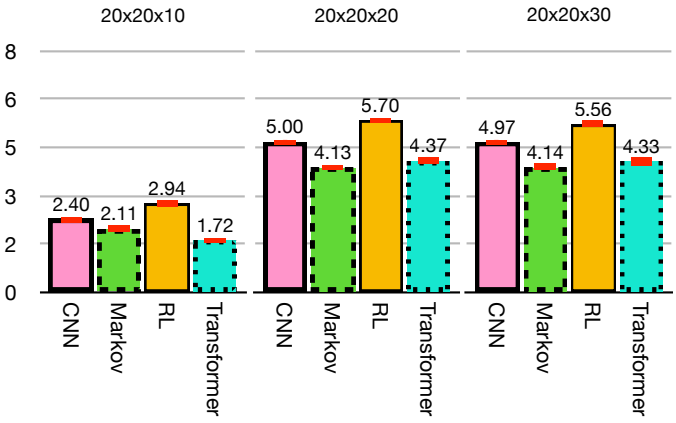


Fig. 4: Excitement ratings for the generation approaches with maximum track size of 20x20x10, 20x20x20 and 20x20x30 and 135 pieces. Error bars show 95% confidence interval. Higher excitement is better.

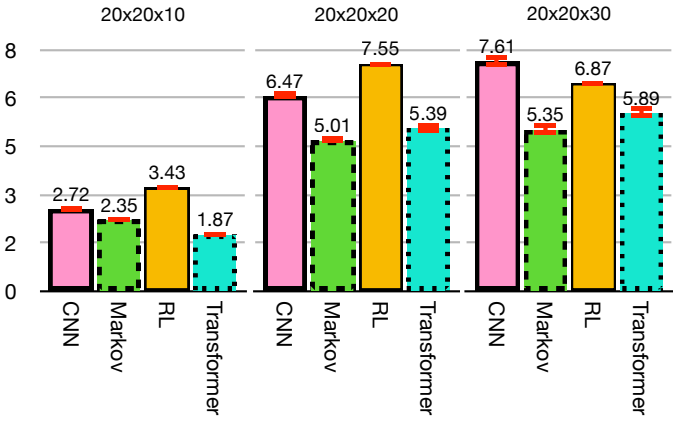


Fig. 5: Intensity ratings for the generation approaches with maximum track size of 20x20x10, 20x20x20 and 20x20x30 and 135 pieces. Error bars show 95% confidence interval. The best range for intensity is between 4 and 6.

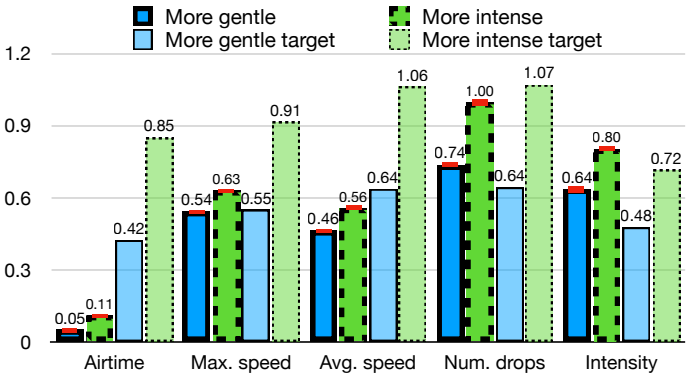


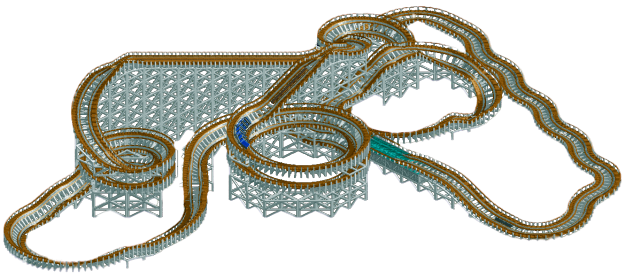
Fig. 6: Selected average target property values for linear reward RL tracks with maximum size of 40x18x35 and 262 pieces. Error bars show 95% confidence interval for the generated tracks. Both the actual average property value and target values are shown. Values are normalized within the range of the average dataset value for each target property.

We show the resulting track property average for the more gentle tracks and more intense tracks, along with the target value for each property, in figure 6. We use the bounds of each target range for our gentle and intense target values; all numbers are normalized to the average dataset values.

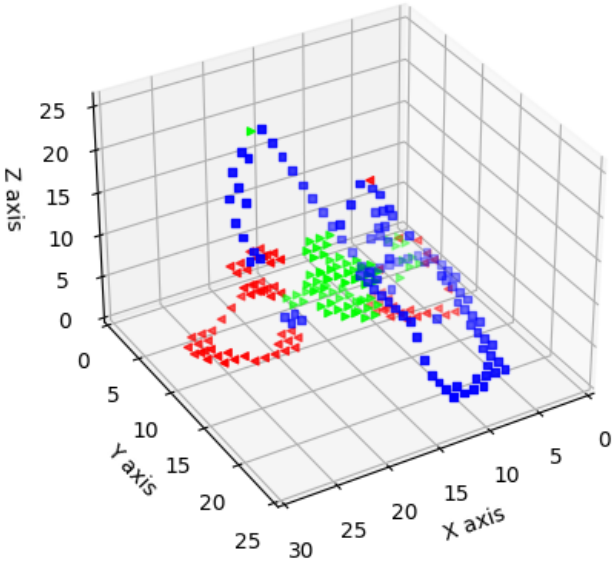
The figure shows that the more intense tracks have a higher (more intense) average value in all properties compared to the more gentle tracks, albeit to varying degrees. For example, the two sets of tracks have a marginal difference in airtime, whereas there is a significant difference in intensity level. Further, for both sets of tracks the average airtime value was much lower than the target value, whereas the intensity values matched the target values much more closely. This result suggests that certain properties like airtime and speed may be difficult to learn, or at least to learn in combination with optimizing for the other properties at the same time.

D. Plagiarism

Figure 9 shows the amount of plagiarism in our generated tracks, i.e., the percentage of track segment sequences of



(a) An RL-generated track.



(b) Part of the grid representation for the above track.

Fig. 7: Shown in a) is an RL-generated track with EIN rating 6.63, 7.72 and 4.85. The curvy section on the right side is the part created by the A∗ algorithm to complete the track circuit. Shown in b) is the grid for the starting bank angle for this track (one of eight grids that comprise the grid representation). Flat banking is shown as blue squares, left banking as red left triangles and right banking as green right triangles.

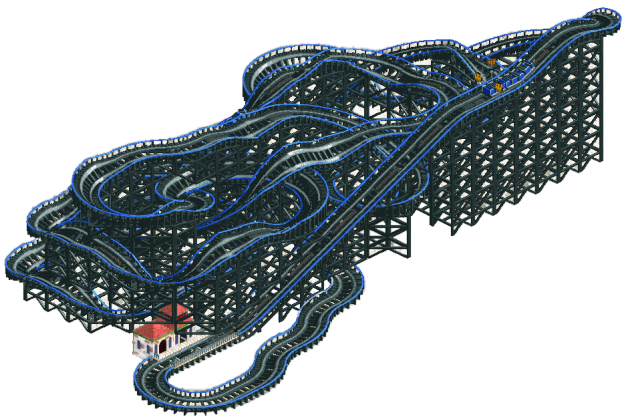


Fig. 8: Another RL-generated track, with EIN rating 6.50, 7.65 and 4.45. The ride begins at the station in the lower left.

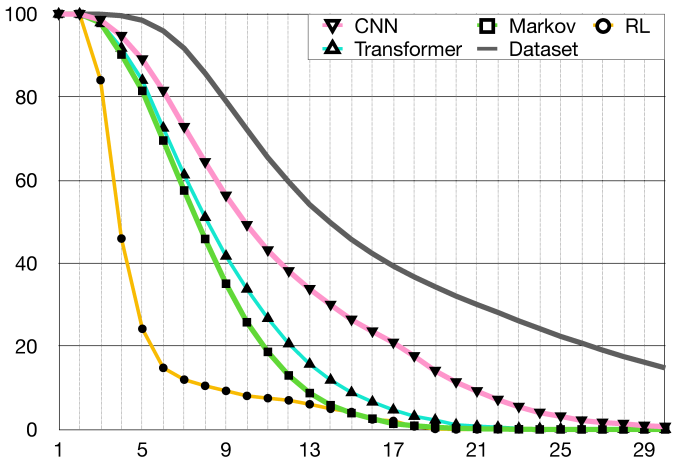


Fig. 9: Percentage of track segment sequences that are present in the dataset, for sequences of length 1 to 30, averaged over all tracks generated from each approach for all experiments.

length n that are also present in the dataset. We omit the pieces generated by the A∗ completion in this calculation. The figure indicates that the RL tracks contain significantly less of the sequences contained in the dataset tracks 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, computed by comparing each individual dataset track to all other dataset tracks, then averaging as usual.

E. Visual novelty

Results for our revised visual novelty metric are shown in Figure 10. The figure shows the mean curve of novelty over normalized time for each approach and the dataset. We can see that the tracks in the dataset and those generated by approaches that train on it gain most novelty early on; after about halfway through the ride, riders see very little new parts of a track. By contrast, the RL tracks exhibit a very different profile, gaining novelty at a significantly more consistent rate throughout the duration of the ride.

This result is interesting because the RL agent was not rewarded to produce incremental increases in novelty. The linear RL agent was trained to produce such for excitement, but it fares the same here as the regular RL agent, which is not rewarded for incremental increases at all. A further inspection of the kinds of track segments favoured by each approach found that the RL tracks contained 1.5-2x more flat turns and about 1-1.5x more sloped turns than tracks of other approaches. If a track employs a lot of quick turns, and is also relatively dense (which RL tracks tend to be), then there are not as many instances where a rider can see a huge amount of new track, but only perhaps a few new pieces at each turn or drop, which may explain this result.

F. Excitement over time

To analyze excitement over time, we plot the mean curve for each approach and the dataset in Figure 11. For easy

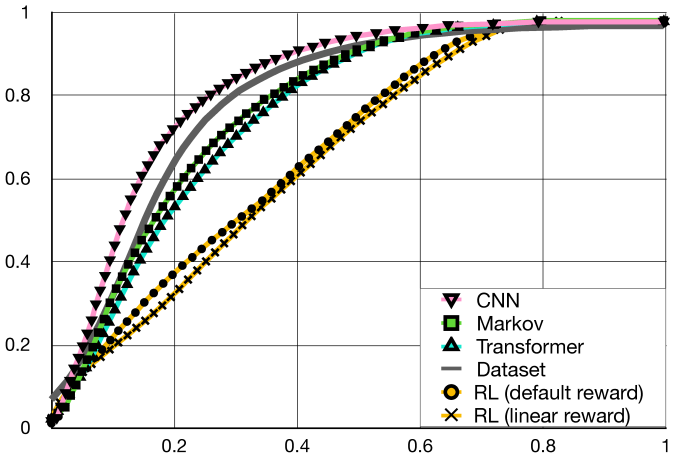


Fig. 10: The mean curve of novelty over normalized time for each approach and the dataset. Tracks were generated with a maximum size of 40x18x35 and 262 pieces.

comparison with the dataset, we again limit our generated tracks to size 40x18x35 with 262 track pieces.

We can first observe that the dataset tracks tend to gain the majority of their excitement by the midpoint of the ride or, more specifically, between the 30% and 50% mark. The Markov and Transformer tracks closely follow this curve, while the CNN and regular RL tracks gain most excitement a bit earlier. The linear RL tracks, by contrast, gain the majority of excitement later on in the ride, and over a longer period of time, from the 45% to 80% mark. This difference is expected since the goal of the linear reward approach is to approximate a linear curve; it does not fully accomplish this, but does do better than the others to this effect. This result shows that we can have control over the excitement over time profile for a track through the choice of a generation approach. We do not claim that one is more preferable to the other, but that it enables a larger diversity in rider experience.

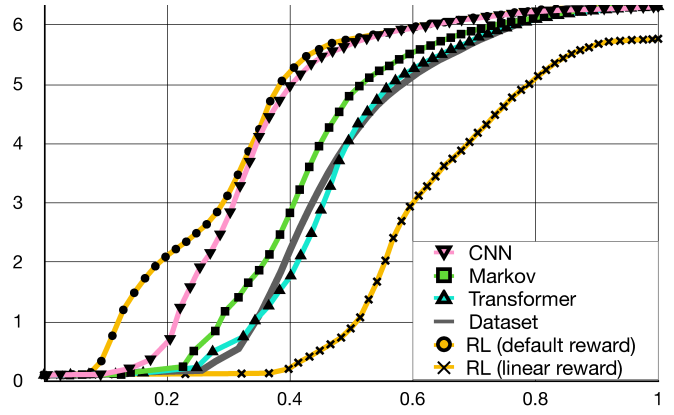


Fig. 11: The mean curve of excitement over normalized time for each approach and the dataset. Tracks were generated with a maximum size of 40x18x35 and 262 pieces.

To further analyze the diversity of tracks with respect to excitement over time, we cluster the tracks for selected

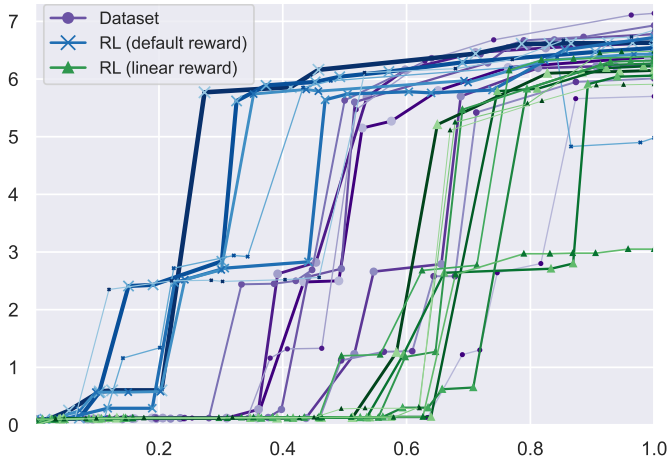


Fig. 12: The path centroids of all clusters of paths generated by the regular RL approach, linear reward RL approach and from the dataset. Each line represents a centroid of a different cluster. The width of a line is proportional to the number of paths in that cluster.

approaches (RL with default reward, with linear reward and the dataset) and show the cluster centroids in figure 12.

We can see that the centroids for each approach are offset in time, as shown in Figure 11, but that there are differences in the exact increases over time for each approach. Thus, we can conclude that the tracks for the two RL approaches and for the dataset offer varied excitement experiences over time within a certain time window. We can also see that most centroids gain excitement in a stepwise fashion, achieving big gains in short bursts. This behaviour likely occurs due to the presence of certain track segments such as sharp drops, since a drop increases excitement through several factors at once (number of drops, speed, G-forces, and others).

VI. Discussion

Our results have shown that rollercoaster tracks can be generated with common PCG algorithms. In particular, techniques that train on a dataset, like Markov, Transformer and CNN, can replicate certain properties of the tracks in the dataset, such as excitement and novelty over time. Meanwhile, reinforcement learning can be used to obtain tracks with different excitement and novelty over time profiles, and also less intense tracks than the dataset; it can be said that due to this latter property, reinforcement learning-generated tracks are in some sense 'better' than the average dataset track, since they will appeal to a larger number of guests.

RL can also be used to control certain track properties to varying extents, such as the number of drops and intensity rating, giving the player or designer a way to choose what tracks they prefer to generate. They also perform better in environments not present in the dataset (such as constrained height) and favour using track sequences not present in the dataset more than the data-driven approaches.

One limitation with the current work is the RL agent's difficulty in learning to control certain track properties such as

airtime and speed. This result may suggest a longer training time is needed or that the particular combination of targets was infeasible. Further experimentation with target ranges may allow for better attainment of targets and track ratings.

Beyond the metrics, a manual inspection of tracks was informative. Some tracks were found to contain long stretches of flat segments at high elevation, which may be monotonous for riders and (at least in real life) costly. One track, shown in figure 8, exhibited a very unusual property. The track, nicknamed 'Smile', begins with the car going up a hill with a chain lift, followed by a camera that takes photos of the riders, then a further hill. However, the car does not have the speed to climb up the latter, as it is missing a chain lift. The car thus starts to fall backwards 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.

VII. Conclusions and Future Work

In this paper we investigated four PCG methods (Markov chain, Transformer, CNN and reinforcement learning), coupled with A* circuit completion and lookahead and backtracking systems, to generate rollercoaster tracks for the RollerCoaster Tycoon video game. The Markov chain, Transformer and CNN worked well in imitating the filtered dataset of user tracks, while the RL method worked better in producing rides that were more popular with guests, especially in a more constrained, low-height environment, and also allowed for controllability of various track properties. The RL method also allowed for different novelty over time and excitement over time profiles, and used sequences of segments less present in the dataset compared to other approaches.

A next step for this work could include generation using rewriting methods, 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 and excitement over time metrics. It may also be useful to consider an extension to our Gymnasium environment to allow for further experiments in RCT beyond track construction, as well as to make our game code changes more easily portable to updated versions of OpenRCT2. 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] J. Campbell and C. Verbrugge, "Procedural generation of rollercoasters," in *Proceedings of the 2023 IEEE Conference on Games*, ser. COG'23. IEEE Press, 2023.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] J. Togelius, S. M. Lucas, and R. D. Nardi, "Computational intelligence in racing games," in *Advanced Intelligent Paradigms in Computer Games*. Berlin: Springer Berlin Heidelberg, 2007, pp. 39–69.

[8] 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.

[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*, ser. COG'21. IEEE Press, 2021.

[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] 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.

[15] 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.

[16] C.-H. Chang, S. Shu, and B. King, "Novelty in theme park physical surroundings: An application of the stimulus–organism–response paradigm," *Asia Pacific Journal of Tourism Research*, vol. 19, no. 6, pp. 680–699, 2014.

[17] M. Bastiaansen, M. Oosterholt, O. Mitas, D. Han, and X. Lub, "An emotional roller coaster: Electrophysiological evidence of emotional engagement during a roller-coaster ride with VR add-on," *Journal of Hospitality & Tourism Research*, vol. 46, no. 1, pp. 29–54, 2022.

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

[19] 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.

[20] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," in *Proceedings of the 35th International Florida Artificial Intelligence Research Society Conference*, ser. FLAIRS'22, May 2022.

[21] 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.

[22] J. Campbell, J. Tremblay, and C. Verbrugge, "Clustering player paths," in *Proceedings of the 10th International Conference on Foundations of Digital Games*, ser. FDG'15, June 2015.

[23] K. Buchin, A. Driemel, J. Gudmundsson, M. Horton, I. Kostitsyna, M. Löffler, and M. Struijs, "Approximating (k, l)-center clustering for curves," in *Proc. 30th ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2019, p. 2922–2938.

[24] M. Frechét, "Sur la distance de deux surfaces," *Annales de la Société Polonaise de Mathématique*, 1925.

[25] 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'18, vol. 80, 2018, pp. 3053–3062.