The Automatic Generation of Game Environments for the Purpose of Training Artificially Intelligent Agents

Isaac Dash, Dr. William Edward Hahn (Faculty Advisor)

Abstract

With the rise of self-driving cars and humanoid robots, it has become important to validate the performance of AI agents in simulated environments. In particular, simulated agents need diverse environments to evaluate their skills. This presents an opportunity to use automated methods to generate training data. The purpose of this study is to compare the effects of training AI agents on various mixtures of algorithmically-generated and AI-generated environments under various test conditions. Inside a simulated environment, AI agents were trained using reinforcement learning on different mixtures of artificially generated environments. The results show that the agent trained on a mixture of AI-generated and algorithmically-generated levels performed best, while the AI trained on purely AI generated levels performed worst. These findings show that using data from a mixture of artificial sources may improve the overall performance of trained AI agents when faced with limited data availability.

Introduction and Background

In recent years, machine learning-based Artificial Intelligence (AI) has exploded in popularity as a method of generating new creations and performing complex tasks with limited human involvement. For example, in language processing, OpenAI's ChatGPT can find errors in code and correct them, write essays, and explain concepts (OpenAI, 2022), and computers have been "taught" to play complex games at the level of world champions by exposing them to millions of actual games (Silver et al., 2016). The latter demonstrates that in order to perform complex tasks, AI platforms require extensive training, and such training often requires extremely large amounts of data or simulations of actual events. For example, versions of ChatGPT-3 were trained on over 300 billion tokens (with about 0.7 words per token) worth of data (Brown et al., 2020). The ChatGPT example highlights a major challenge with AI training—the size of the dataset required to teach an AI to perform these

complex tasks is restricted by data availability which can restrict the ability of the technology involved to perform the desired tasks (Brown et al., 2020). Video games, like language processing, have adopted the use of AI in many facets of gameplay and design, in order to create a more immersive and lifelike environment in-game. For example, AI is often used to control non-player characters to act as enemies for a more interactive experience (Skinner & Walmsley, 2019). Generative AI can also be used to create levels for tile or grid-based games such as *Super Mario Brothers* (Awiszus, Schubert, & Rosenhahn, 2020). However, as with ChatGPT, a large number of simulations are required for training which may restrict the use of the AI technology. This data often takes the form of simulation environments, like game levels, which may not be available or may be limited when developing the AI protocol, and can be difficult or expensive to produce. As such, it may not be practical for humans to create these datasets manually. An option would be to use algorithms for developing levels procedurally or using other AI to generate those environments in order to reduce the cost of AI training.

This work compares the effectiveness of an AI agent trained in environments created by other AI and AI agents trained in procedurally generated environments. More specifically, it tests whether there are performance differences between AI trained on AI generated data, AI trained on procedurally generated data, and AI trained on both procedurally- generated data and AI- generated data.

Materials and Methods

In this study, we first generated environments, or levels, using procedural generation. Then, we trained a generative Long Short-Term Memory, or LSTM, neural network to create environmental levels based on those procedurally generated levels. Lastly, we trained reinforcement learning-based agents on various mixtures of the two types of generated data and compare the trained agent's proficiency in completing a task put before them.

Procedural Level Generation

As noted, to train an AI, one needs training data. This research uses two separate methods for creating the dataset: procedural generation and generative AI. The first AI that was trained was a LSTM neural network used to generate the game environments. To create the dataset

for this AI, the Unity Game Engine was used as it features a useful toolkit for machine learning agent training (Juliani et al., 2018). This toolkit has been used in the past to create intelligent agents that perform complex tasks, such as ones used to solve mazes (Hung, Truong, & Hung, 2022). Each level generated was composed of a square grid-based tilemap. In such a tilemap, each of the square grid spaces, called tiles, are set to hold a certain sprite, or image-based game asset. Together, these tiles form the level as a whole. The levels were based on tilemaps that were 70 tiles wide and 70 tiles tall, for a total of 4900 tiles per level. Each level started out with all tiles in the tilemap being set to hold empty space. The tilemaps were then populated with a four-step approach utilizing procedural generation, a form of content generation that uses algorithms to create useful data.

First, the general terrain was created using a smoothed random walk algorithm adapted from Ethan Bruins' work with Unity Technologies (Bruins & Technologies, 2018). In such an algorithm, starting at a randomly chosen height within chosen bounds, the height of the level is changed either up or down by one grid space vertically after a certain number of spaces are passed horizontally, with the exception of if the change in height would send the vertical height of the level outside the chosen bounds for level height. The minimum length was set at the same vertical height to be six spaces horizontally, and the bounds were set to not allow the algorithm to either raise the level beyond three-fourths of the level's total height or below one-third of the total height to prevent camera errors as the level was played. All tiles below or equal to the algorithm's chosen height at any horizontal position were set to land tiles, while those above this position contained empty space.

Next, the tiles at the surface level of the previously generated tilemap were changed to hold surface tiles, such as grass, and inclines were added at the positions in which the height of the level changed to make the level look more cohesive. After that, obstacles were added to the level to make it more difficult. A random value between zero and three, inclusive, was chosen to hold how many obstacles would be placed, and then valid positions among the flat sections of the level would be chosen to add these obstacles. These obstacles would be raised parts of the terrain for the player to jump over. Lastly, the level was completed by adding stretches of flat land on the left and right sides of the level, and

placing a flag that completed the level when touched by the player. Once these steps were followed, a level for the dataset was complete. An image of such a procedurally generated level is in Figure 1. The art for each tile was adapted from the "Platformer Art Deluxe" package by Kenney (Kenney, 2014). Gameplay starts on the flat part of the left side of the level, and the level is completed when the player reaches the flag at the right side of the level.

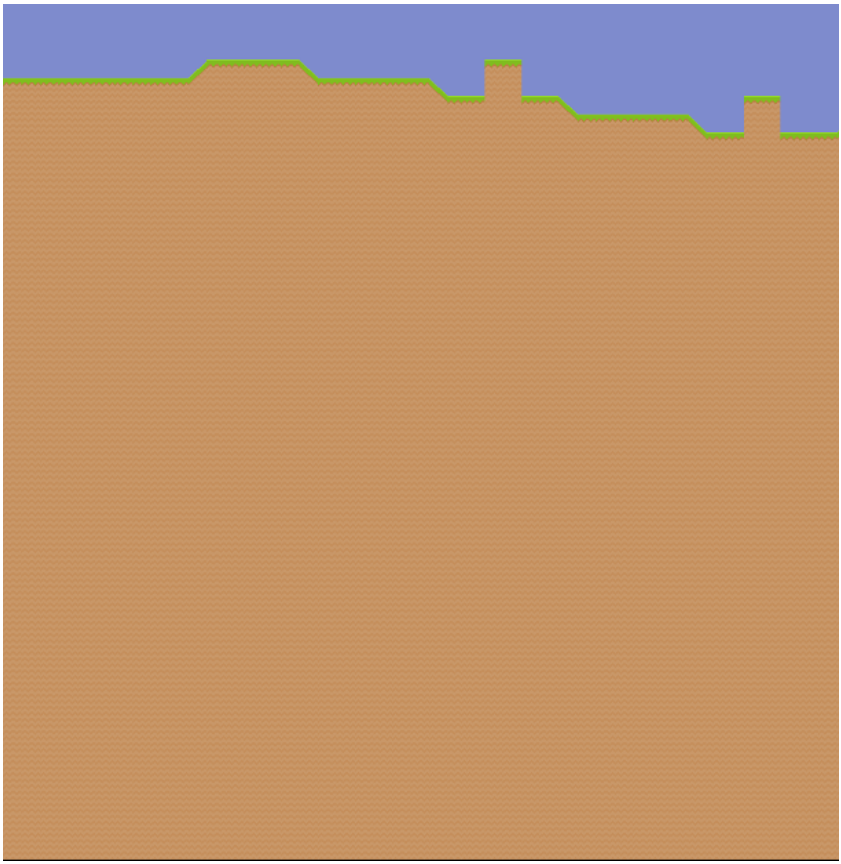

Figure 1. Procedurally generated level. This is an example of a procedurally generated level using a smoothed random walk algorithm with raised sections added

Character	Tile
-	Empty Space
X	Ground
o	Grass Tile
<	Rightward Facing Slope Top
>	Leftward Facing Slope Top
]	Rightward Facing Slope Base
[	Leftward Facing Slope Base

Table 1. Tile Encoding as Characters

Figure 2. Encoded level as text. This is an example of one of the levels generated encoded as text that the LSTM was trained on

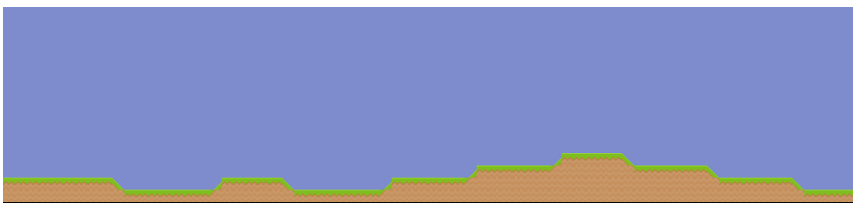

Figure 3. Encoded level as an image. This is an image of the level from a text file in Unity.

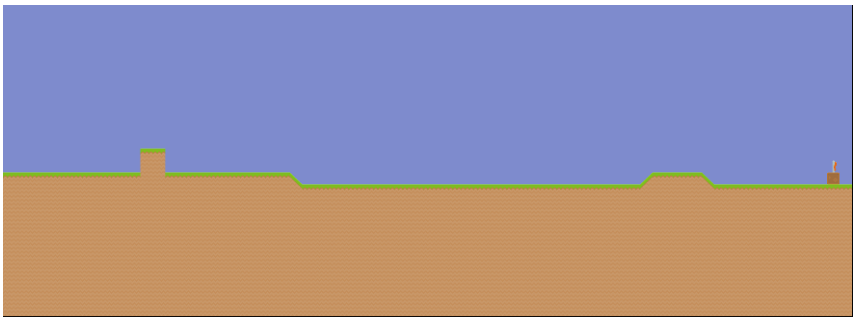

Figure 4. LSTM-generated level in Unity. This is a level from the Long-Short Term Memory Network in Unity.

AI Agent Training

The last step in this research work was the training and testing of AI Agents on these levels. According to Bansall (2023), an AI agent is a "...computer program or system that is designed to perceive its environment, make decisions and take actions to achieve a specific goal or set of goals." In the case of this project, the AI agents were the AI used to play the levels. To make these AI agents functional, they first had to be trained. To test the effectiveness of different methods of generating data, three separate AI agents were trained using Unity's ML-Agents toolkit (Juliani et al., 2018). The first was trained only on procedurally generated levels. The second was trained only on LSTM- generated levels. The last was trained on a mix of both. Each AI was trained using the same parameters, including the same number of maximum training steps: 500,000. During training, the AI agents received an 86-pixel by 64-pixel camera feed as input. As output, during gameplay the agents would choose if they would jump, if they would move, and which direction they would move in if they chose to move. All three agents were trained with a reinforcement-learning technique, where rewards and punishments guided the behavior of agents who work to maximize their reward value. The agents were rewarded for heading towards the goal and reaching the goal and punished for moving away from the goal and taking up time. After training, each of the AI agents were tested on 100 different levels in three different categories: 100 levels made purely through procedural generation, 100 levels made purely through LSTM, and 100 levels from both. In the category where levels from both the LSTM and the procedural

generation script were used, the level was randomly selected to be from either the LSTM-generated levels or the procedurally-generated levels with equal probability. In all cases, the levels the AI agents trained on were not the same as the ones they were evaluated on, as all evaluation levels were newly generated. The time the agents took to complete a level in each test was recorded, and the performance in these tests was then compared.

Results

The results show that the AI agents trained only on LSTM-generated levels performed materially worse in each of the three categories tested than the other AI agents. Not only was it slower, but it also had a much higher standard deviation of time taken to complete levels than all of the other agents tested across all three categories. The best-performing agent across all three categories was the one trained both on levels generated by the LSTM and levels generated by procedural generation, and it also had the lowest standard deviation of time taken to complete a level across all three categories. In all three categories, the second-best performing AI agent was the one trained only on procedurally generated levels, and this agent also had the second-lowest standard deviation of time taken to complete a level across all three categories. However, the difference between the first- and second-best agent was marginal compared to the difference between the second- and third-best. The results are summarized in Table 2 and Table 3. In sumary, the best-performing agent was the one trained on both procedurally generated levels and LSTM-generated levels, while the second-best was the one trained on just procedurally generated levels, with the worst agent being the one trained on only LSTM-generated levels.

	Procedurally Generated Levels Completion Time (seconds)	LSTM-Generated Levels Completion Time (seconds)	Mixed Levels Completion Time (seconds)
LSTM AI	16.0	15.16	15.20
Procedural Generation AI	12.74	13.11	12.64
Mixed Dataset AI	12.67	12.80	12.48

Table 2. Average time per level in seconds.

	Procedurally Generated Levels Completion Time (seconds)	LSTM-Generated Levels Completion Time (seconds)	Mixed Levels Completion Time (seconds)
LSTM AI	4.19	3.58	3.58
Procedural Generation AI	1.27	1.43	1.11
Mixed Dataset AI	0.93	1.23	0.99

Table 3. Standard deviation in average time per level in seconds.

Discussion

The results show that while training an AI agent only on AI-generated levels may reduce performance significantly, training an AI agent on a dataset of mixed AI-generated and procedurally generated levels may increase performance. The results are somewhat surprising, as one could reasonably expect that the agent trained on procedurally generated levels would perform best on procedurally-generated levels, and the one trained on AI-generated levels would perform best on AI-generated levels. The fact that the agent trained on the dataset including both AI-generated and procedurally generated levels performed the best in all categories was unexpected. Perhaps training on both procedurally generated and AI-generated data trained the agent in more diverse environments, which could lead to better model performance. Alternatively, perhaps the addition of the new data prevented the agent from overfitting, where an AI becomes overly-trained on the training data to the point of losing its ability to generalize to new situations. Future research may help to isolate the actual cause of this improvement and find the best mixture of data sources to maximize agent performance.

Conclusion

In this study, AI-generated data showed promise in improving the effectiveness of AI agents. While exclusive use of AI-generated data was not shown to be optimal, mixed use of AI-generated data and other methods of generating data was shown to improve the performance of the trained agent. This research indicates that it may well become commonplace in the future to train AI agents in such mixed environments. Future research may indicate the precise reasons why this improvement occurs with

mixed training data, combine additional data sources (including human-generated data), and find the optimal ratio of data from different sources to maximize agent performance. These findings may apply more broadly than just in game agent training, as "...it is immensely cheaper to develop and test AI in a created environment with thousands of instances, than to build robots and have them do thousands of tests," (Skinner & Walmsley, 2019). For example, with self-driving cars, "…it is much easier to just train the AI through a driving computer game rather than risk damaging the hardware and injuring people," (Skinner & Walmsley, 2019). Training the AI agent controlling self-driving cars may be done in AI-generated or procedurally generated environments, and thus these findings may apply. Similarly, for surgical robots, the agents controlling the robots may be trained in virtual environments to prevent the harming of patients during real application, and so these findings may apply once more (Bourdillon et al., 2022). As AI permeates human life, training AI agents will become an increasingly large issue. As such, understanding the consequences of training AI agents in AI-generated environments and procedurally generated environments will only increase in importance in the future.

References

Awiszus, M., Schubert, F., & Rosenhahn, B. (2020, October). TOAD-GAN: Coherent Style Level Generation from a Single Example. In *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 16(1), 10-16. Association for the Advancement of Artificial Intelligence. https://aaai.org/papers/00010-7401-toad-gan-coherent-style-level-generation-from-a-single-example/

Bansall, S. (2023, March 10). *Agents in Artificial Intelligence*. GeeksforGeeks. https://www.geeksforgeeks.org/agents-artificial-intelligence/

Bourdillon, A., Garg, A., Wang, H., Woo, Y., Pavone, M., & Boyd, J. (2022). Integration of Reinforcement Learning in a Virtual Robotic Surgical Simulation. *Surgical Innovation*, *30*(1), 94-102. https://doi.org/10.1177/15533506221095298

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Heinighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan & H. Lin (Eds.), *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 159, 1877-1901. Curran Associates, Inc. https://dl.acm.org/doi/abs/10.5555/3495724.3495883

Unity Technologies. (2018). Procedural Patterns to use with Tilemaps. E. Bruins. Github, June 12, 2018. ProceduralPatterns2D. https://github.com/UnityTechnologies/ProceduralPatterns2D

Hung, P. T., Truong, M. D. D., & Hung, P. D. (2022). Tuning Proximal Policy Optimization Algorithm in Maze Solving with ML-Agents. In Singh, M., Tyagi, V., Gupta, P.K., Flusser, J., & Ören, T. (Eds.), *Advances in Computing and Data Sciences*, 1614, 248–262. https://doi.org/10.1007/978-3-031-12641-3_21

Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018). *Unity: A General Platform for Intelligent Agents*. ArXiv. https://arxiv.org/pdf/1809.02627v1.pdf

Kenney. (2014, November 1). Platformer Art Deluxe · Kenney. Retrieved May 12, 2023, from https://www.kenney.nl/assets/platformer-art-deluxe

OpenAI. (2022, November 30). Introducing ChatGPT. OpenAI. https://openai.com/blog/chatgpt

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961

Skinner, G. & Walmsley, T. (2019). Artificial Intelligence and Deep Learning in Video Games A Brief Review. In *Proceedings of the 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, 404–408. https://doi.org/:10.1109/CCOMS.2019.8821783