Dungeon Digger: Apprenticeship Learning for Procedural Dungeon Building Agents

Evan C. Sheffield
College of Computer and Information Science
Northeastern University
Boston, MA 02115, USA
sheffield.e@husky.neu.edu

Michael D. Shah
College of Computer and Information Science
Northeastern University
Boston, MA 02115, USA
mshah.475@gmail.com

CHI PLAY '18 Extended Abstracts, October 28-31, 2018, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5968-9/18/10.
https://doi.org/10.1145/3270316.3271539

Abstract
One of the challenges of generating video game levels procedurally is capturing what in the design of a specific level makes it fun to play. In this paper, we demonstrate our preliminary work on a system which learns from expertly designed game levels to produce new game levels automatically. We developed a platform for designers to create tile-based dungeon levels and a level-generating agent which consumes recordings of design sessions to learn and then create its own levels. We evaluate the output of our agent using metrics gathered from a static analysis and a discount usability study using a digital game prototype that renders the level designs. Our preliminary results suggest that this system is capable of generating content that emulates the style of the human designer and approaches the level of fun of human-designed levels.

Author Keywords
Procedural Content Generation; Machine Learning; Inverse Reinforcement Learning; Apprenticeship Learning

Introduction
One of the primary goals of a game designer is to craft engaging experiences for their players. They may wish to evoke specific emotions or produce content that is fun and interesting. Procedural Content Generation (PCG), the algorithmic generation of game content, provides the poten-

Fun
For the purposes of this study, we define fun as the subjective enjoyment expressed by players. We used player-reported fun ratings to compare the levels evaluated in our usability study.

Level Style
In this paper, we use holistic level evaluations to determine stylistic similarity of levels (Table 2). Through visual inspection of agent-produced levels we also identified some of the same design elements (e.g. rooms) which were combined and arranged in unique ways (Figure 2).

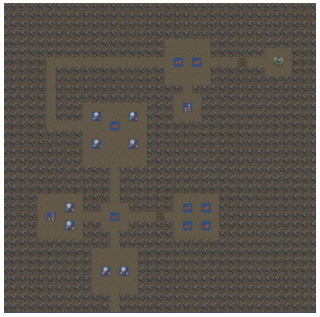

Figure 1: A screenshot of the tile-based level generator used to produce dungeons and expert demonstrations. Art assets courtesy Unity Technologies [15].

tial to lessen the burden on game designers by allowing for the rapid production of content. Game levels are an enticing prospect for PCG because they are core to the experience of games. Although it is well-established that PCG may save time and resources [16], it is difficult to design generators to produce content that embodies human design goals.

Although powerful, many "constructive" approaches to PCG rely on the designers to codify their requirements and constraints in the algorithms and parameters of the generator. Applications of PCG to commercial games are not widespread [4, 18], and this complexity stands as a barrier to adoption. Procedural Content Generation via Machine Learning (PCGML) has emerged as a potential solution by automating content generation based on learned models of existing designs [12], but the problem of how best to capture the creative spark of the designer remains an open question. In this paper we explore the problem of automatically generating fun levels based on expert demonstrations of level creation. We hypothesized that the act of level creation would intrinsically capture some aspect of the designer's goal and style.

We present Dungeon Digger, an automated system for the generation of dungeon game levels. We chose to focus on dungeon levels because of their structural complexity and their ubiquity in roguelike and adventure games like *The Legend of Zelda* series. The Dungeon Digger agent learns to produce levels by observing demonstration data produced from a custom level generator that we have developed for creating tile-based level designs. In a single pass, the agent can produce tilemaps for playable levels that include level topology, enemies, and items. In order to capture the innate fun quality of human designs, we apply Apprenticeship Learning via Inverse Reinforcement Learning (IRL) [1] to extract an overarching design goal without the need to manually encode constraints. We report the results of a discount usability study conducted using a digital game prototype that we developed to allow players to test and evaluate various samples of human and agent-generated levels.

Our main contribution is a fully-automated approach to dungeon level generation that learns from examples of human level creation and produces levels that approach the degree of fun of human-designed content.

Background and Related Work

PCG of Dungeon Levels
One category of dungeon PCG approaches involves the use of generative grammars in which replacement rules are applied to piece together chunks of levels or other representative units. Dormans et al. [2] utilized a combination of graph grammars to create abstract mission graphs and shape grammars to construct games spaces. Van der Linden et al. [17] used gameplay grammars to encode design constraints as grammar rules. Such techniques are highly customizable and allow encoding of high-level design goals, but designing grammars and non-terminal chunks is complex and not easily transferable between games. Our approach seeks to automate the level generation process to reduce the burden on designers. Liapis et al. [7] used a genetic algorithm to implement a level suggestion feature in their game design tool, Sentient Sketchbook. User studies showed favorable opinion of this feature, but the fitness function is based on hand-coded design assumptions whereas our system attempts to infer them from human demonstrations. Summerville et al. [13] generated dungeon levels by training a Bayesian Network on topology of *The Legend of Zelda* levels. Their approach relied on super-

Reinforcement Learning
Reinforcement learning (RL) is a machine learning technique in which an agent learns behavior by trying various actions and discovering their outcomes in order to maximize a numerical reward [14]. Typically, the learning problem is defined as a Markov Decision Process (MDP). An MDP consists of finite sets of states and actions, a set of transition probabilities between states conditioned on actions, a reward function defining the immediate reward received upon a transition, and a discount factor which measures the importance of future rewards. Given an MDP, the task of RL is to recover an optimal policy π^* that guides the behavior of an agent to produce the maximum total reward from a start state.

vised learning using manually annotated levels whereas our agent learns from unsupervised level creation data.

Inverse Reinforcement Learning (IRL)
IRL, originally described by Ng and Russell [9] inverts the standard Reinforcement Learning (RL) task. In our work we utilize Apprenticeship Learning via Inverse Reinforcement Learning [1], a technique which applies IRL to extract a reward function and policy from observations of an expert's behavior in an MDP environment. Apprenticeship learning has been successfully applied to many domains, including the training of an agent to play *Super Mario Bros.* [6] based on demonstrations of play by human experts. To the best of our knowledge this is the first time it has been applied to PCG.

The Dungeon Digger System

In an effort to create fun dungeon level designs, we developed three primary components for the Dungeon Digger System: 1) A tile-based level generator for creating levels and expert demonstrations, 2) an apprenticeship learning system which learns from the output demonstrations and automatically creates new level content, and 3) an adventure game prototype which renders human and agent-generated levels as a platform for evaluation by players.

Tile-Based Level Generator
Developed with the Unity game engine, our custom level generator (Figure 1) allows a user to produce levels by controlling a "digger" character. A level begins as a 50x50 grid of blocks. As the digger moves, it clears out paths in a manner similar to the constructive ad hoc agent described by Shaker et al. [11]. In addition to movement up, down, left, and right, the digger can also create small, medium, and large rooms centered around itself and place enemy, treasure, key, locked door, and exit tiles. We require placement of the exit to be the last action taken in creating a level. An advantage of this design is that all levels created in the system must contain an exit and a path between the start location and exit tile, making them inherently playable.

As the user controls the digger to produce their level, we record the sequence of states visited and actions taken. Demonstration sequences and tilemaps are written to files at the conclusion of a level creation session for consumption by the other modules.

Apprenticeship Learning System
In order to train an agent from the expert demonstrations produced by the level generator, we modeled our domain in the Brown-UMBC Reinforcement Learning and Planning (BURLAP) Java library [8].

We read in the demonstration data from our level generator to produce "episodes" that act as input to the apprenticeship learning algorithm and construct an MDP. We next run BURLAP's implementation of apprenticeship learning on our expert demonstration episodes to produce a reward function which is a linear combination of the discretized tile counts and "has exit" flag from our state representation. We augmented the learned reward function by applying an additional arbitrarily large reward for reaching a terminal state to discourage the agent from reaching infinite reward loops and failing to complete a level. Applying this reward function to the MDP, we use value iteration to solve for the optimal policy π^*.

We can produce levels by selecting actions in succession according to π^* beginning from the start state until we reach our end state. A BURLAP visualizer (Figure 3) visualizes the agent's actions and output levels. Because the policy obtained from value iteration is deterministic, we introduce stochasticity in the generation process by op-

States
The state representation consists of a parameterizable $N \times N$ field of vision centered around the agent, the agent's distance from the start, and various features that describe the level including counts of enemy, treasure, door, and empty tiles and a boolean for the presence of an exit tile. All continuous values such as tile counts are discretized into a small number of buckets based on maximum observed values according to the formula: $discrete(count) = \left\lfloor \frac{count}{(\frac{maxcount}{numbuckets})} \right\rfloor$

Actions
Up, Down, Left, Right, Room (S,M,L), Enemy, Treasure, Key, Door, Exit. In each state, we limit the set of possible actions to those sampled from the expert demonstrations.

State Transitions:
Transitions from each state s to state s' on action a are based on transition frequencies observed in the training data as in [6].

tionally applying one of several "refresh policies". These policies will trigger the MDP to jump to a new state based on a probability that increases as the agent takes actions without refreshing. We have implemented several variations of refresh policies, including ones that refresh to the start state, a random state, or a state with the same field of vision. Another attempts to avoid actions which could lead to overlapping rooms. The result of these policies is the ability to create a variety of levels from a single learned policy (Figure 3).

Adventure Game Prototype
We created a digital game prototype in Unity styled after *The Legend of Zelda* to allow players to test and evaluate generated content (Figure 4). In the game, players navigate their character through the dungeon levels, collecting keys to unlock doors and reach the exit. They can swing their axe to fend off zombie enemies when they draw near. Unlike in *Zelda*, the dungeon is explored as a continuous space rather than moving between discrete rooms.

Experiments

To test the performance of our system on true "expert" designs, we recreated the first eight dungeons from *The Legend of Zelda* [10] using our tile-based level designer and trained an agent on the output demonstration data (e.g. Figure 5). We collected representative samples of output levels from separate runs of apprenticeship learning with various vision parameter assignments and refresh policies.

User Study
To evaluate the level of fun of the generated levels compared to their human inspirations, we performed a discount usability test on a combination of human and agent levels using our digital game prototype ($N = 6$, 1 female, mean age of 30.67). Participants represented a range of experience with video games: 50% of participants reported playing video games multiple times per week, while others reported less frequent play of a few times per month or year. After a brief tutorial introducing them to the game prototype, participants were asked to play a series of six levels (pictured in Figure 2), pausing between levels to record their impressions. Each participant played the same six levels, though the order of levels was randomized for each session. Two of these levels were our reproductions of Zelda levels, while the other four were selected from the aforementioned agent output samples. Participants were only told that they may be playing human-designed levels, agent-designed levels, or some combination thereof. The questionnaire we used for evaluation of levels asked users to rate the level on a Likert scale from 0-4 (0=Not at all, 4=Extremely) for fun, difficulty, exploration, playability, and humanness of design. The rating scale and several questions were adopted from the Game Experience Questionnaire [5].

Static Level Evaluations
To characterize the human and agent-designed levels, we compared them using four static evaluation metrics: linearity, leniency, exploration, and density. *Linearity* characterizes the winding quality of the path from start to finish and is calculated as 1 divided by the number of changes in direction following Dijkstra's shortest path. *Leniency* is a measure of how easy it is to score points. We first derive the level's score: +1 per treasure, -0.25 per enemy, -1 per locked door. Leniency is calculated by applying the sigmoid function $\frac{1}{1+e^{-x}}$ to the score. *Exploration* is estimated by running a flood fill algorithm from the start location until the exit is reached and determining the percent of traversable tiles filled, as in [7]. *Density* is the ratio of enemy, item, and door tiles to all traversable tiles.

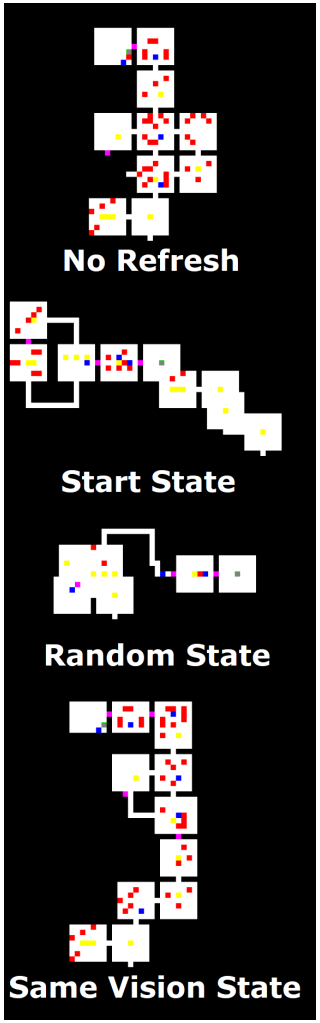

Figure 3: Levels created from the same learned policy using various refresh policies. The policy was trained on demonstrations of creating eight *Zelda* levels.

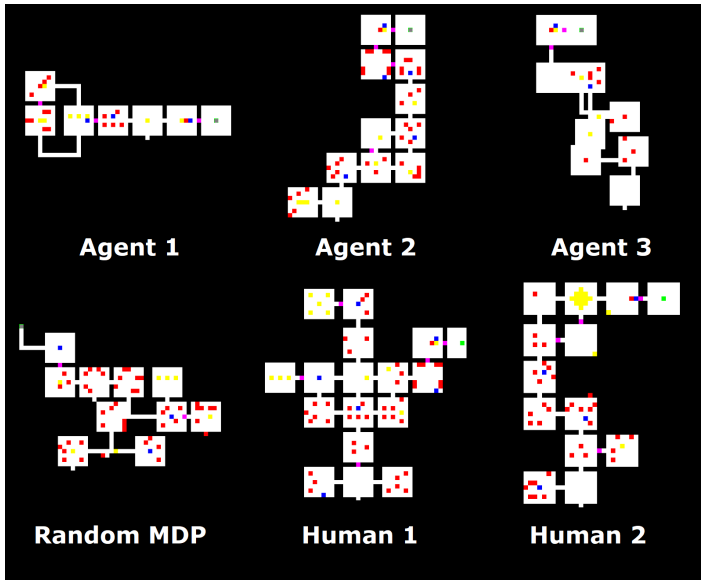

Figure 2: The four agent and two human created levels evaluated in our user study.

Results

In Table 1 we report the mean scores for each level as well as the mean results for agent and human levels. Agent 1 and Agent 2 were produced using a vision of 7×7 using the collision avoidance refresh strategy and no refresh strategy respectively. Agent 3 was produced using vision of 5×5 and no refresh strategy. Random was constructed by selecting random valid actions to walk through the MDP. Human 1 and Human 2 were reproductions of dungeons 1 and 4 from *The Legend of Zelda*. The agent combined result considers Agent 1 and Agent 2 because they reflect our most promising results, outperforming the limited vision agent level in each category. On average, the agent levels tested below the human levels in each metric. Agent 2 however received a higher fun rating than Human 2. Although rated less fun, Agent 1 was convincing enough that on average users believed it to be more human-designed than Human 2. Another observation is that the level produced by random actions in the MDP was evaluated as the most fun level in the group. We believe its high enemy density and chaotic design in tandem with the play style of the game prototype may have contributed to this result.

	User Evaluations				
Level	*Fun*	*Difficulty*	*Exploration*	*Playable*	*Human*
Agent 1	2.5	1	3	3.17	2.33
Agent 2	3	2	2.67	3	0.83
Agent 3	2	1	2.17	3	0.5
Random	3.33	2.33	2.83	3.33	2.5
Human 1	3.17	2	2.83	3.5	2.83
Human 2	2.83	1.5	3.17	3.67	1.83
Agent 1&2	2.75	1.5	2.83	3.083	1.583
Human	3	1.75	3	3.583	2.33

Table 1: Mean level scores based on questionnaire data obtained from our discount usability study.

Table 2 presents the values obtained from our static metric analysis of several levels in the user study. Agent 2 has a similar static analysis to Human 1. The stochasticity of the refresh policy in Agent 1 resulted in the production of a novel design (Figure 2), hence the difference in exploration from some of its training sources. We note that this lower exploration and higher leniency may be correlated with its

Figure 4: A screenshot of the digital game prototype developed in Unity used to conduct our user study. Art assets courtesy Unity Technologies.

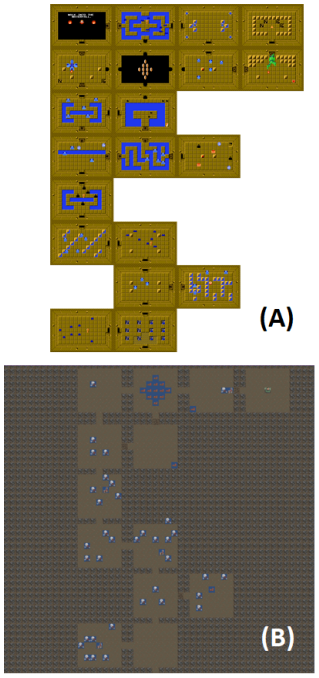

Figure 5: Reproduction of a *Zelda* dungeon (A) (image sourced from [3], © Nintendo) in our tile-based level creator (B).

	Level Metrics			
Level	*Linearity*	*Leniency*	*Exploration*	*Density*
Agent 1	0.25	0.95	0.47	0.08
Agent 2	0.25	0.005	0.95	0.098
Human 1	0.25	0.08	0.98	0.082
Human 2	0.125	0.998	0.95	0.091

Table 2: Static metric evaluations of select user study levels.

lower fun and difficulty evaluations by users (Table 1). In the future, we hope to establish a stronger correlation and use these evaluation metrics to predict player impressions.

Discussion

Although our preliminary results show that automated levels are not yet able to consistently achieve the same level of fun as human-generated levels, we believe this approach shows promise as an automation tool. Our state representation and retrieved reward function are based solely on observations of the level and make no assumptions of design goals or constraints, yet we have been able to generate levels that reflect the style of the human designer and test positively with players as evidenced by the similar static evaluations of Agent 2 and Human 1.

In future work we would like to optimize for the metrics tested in our user study, but we recognize that levels that do not score as highly are not necessarily any less valuable. For example, although Agent 1 received a lower average fun rating in our user study than the human designed levels, one user suggested it would make an ideal tutorial level because of its lower difficulty. Another praised the novelty of its design: *"The treasure/key placement to the right seemed designed to draw you directly to the exit, which had the opposite effect of making me want to explore the rest of the level..."*. Resolving a common complaint about odd tile-placements (e.g. locked doors with openings directly next to them) may also improve players' perception of computer generated levels.

Although a useful tool for early results, our discount user study is limited in the level of analysis it can provide. To perform a more rigorous analysis we plan to conduct a larger study to test a greater quantity of level designs. The comparatively high performance of the randomly sampled level suggests that our current game prototype may not be well-suited to the type of levels under study. Additionally, we have not yet validated it with professional game designers to get their impressions on the degree of automation and input style. Our tile-based level creator may be too simplistic for designers to produce serious work with.

Conclusion

We presented Dungeon Digger, an automated system for dungeon level generation that utilizes apprenticeship learning to model the innate design goals of the human designer that lead to generating fun levels. Through a discount user study, we discovered that our agent can not only create levels that mimic the style of the expert levels, but can also produce novel designs. Our analysis showed that our technique is capable of producing levels that approach the degree of fun of human-designed levels.

As a next step we plan to seek feedback from expert game designers in order to adjust the level automation and tile-level creator to fit their workflow and creative goals. This will help align future development with a long-term goal of building Dungeon Digger into a plug-and-play tool to assist designers by automatically providing additional level content from observing their past creations.

REFERENCES

1. Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 1.
2. Joris Dormans. 2010. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 workshop on procedural content generation in games*. ACM, 1.
3. Mases Hagopian. 2018. The Legend of Zelda Walkthrough. (2018). `https://www.zeldadungeon.net/the-legend-of-zelda-walkthrough/`
4. Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9, 1 (2013), 1.
5. WA IJsselsteijn, YAW De Kort, and Karolien Poels. 2013. The game experience questionnaire. (2013).
6. Geoffrey Lee, Min Luo, Fabio Zambetta, and Xiaodong Li. 2014. Learning a super mario controller from examples of human play. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 1–8.
7. Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2013. Sentient Sketchbook: Computer-aided game level authoring.. In *FDG*. 213–220.
8. James MacGlashan. 2016. Burlap: Brown-UMBC reinforcement learning and planning. (2016). `http://burlap.cs.brown.edu/`.
9. Andrew Y Ng, Stuart J Russell, and others. 2000. Algorithms for inverse reinforcement learning.. In *Icml*. 663–670.
10. Nintendo R&D1. 1986. *The Legend of Zelda*. Game [NES]. (21 February 1986). Nintendo, Kyoto, Japan.
11. Noor Shaker, Antonios Liapis, Julian Togeluis, Ricardo Lopes, and Rafael Bidarra. 2016. Constructive generation methods for dungeons and levels. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, Noor Shaker, Julian Togelius, and Mark J. Nelson (Eds.). Springer, 31–55.
12. Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgard, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* (2018).
13. Adam J Summerville, Morteza Behrooz, Michael Mateas, and Arnav Jhala. 2015. The learning of zelda: Data-driven learning of level topology. In *Proceedings of the FDG workshop on Procedural Content Generation in Games*.
14. Richard S Sutton, Andrew G Barto, and others. 1998. *Reinforcement learning: An introduction*. MIT press.
15. Unity Technologies. 2018. 2D Roguelike. (2018). `https://assetstore.unity.com/packages/essentials/tutorial-projects/2d-roguelike-29825`
16. Julian Togelius, Noor Shaker, and Mark J. Nelson. 2016. Introduction. In *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, Noor Shaker, Julian Togelius, and Mark J. Nelson (Eds.). Springer, 1–15.

17. Roland Van der Linden, Ricardo Lopes, and Rafael Bidarra. 2013. Designing procedurally generated levels. In *Proceedings of the the second workshop on Artificial Intelligence in the Game Design Process*.

18. Roland van der Linden, Ricardo Lopes, and Rafael Bidarra. 2014. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 1 (2014), 78–89.