# Mixed-Initiative Level Design with RL Brush

Omar Delarosa[1], Hang Dong[1], Mindy Ruan[1],
Ahmed Khalifa[1,2], and Julian Togelius[1,2]

[1] New York University, Brooklyn NY 11201, USA
{omar.delarosa,hd1191,mr4739,ahmed.khalifa}@nyu.edu
[2] modl.ai, Copenhagen, Denmark
julian@togelius.com

**Abstract.** This paper introduces *RL Brush*, a level-editing tool for tile-based games designed for mixed-initiative co-creation. The tool uses reinforcement-learning-based models to augment manual human level-design through the addition of AI-generated suggestions. Here, we apply *RL Brush* to designing levels for the classic puzzle game *Sokoban*. We put the tool online and tested it in 39 different sessions. The results show that users using the AI suggestions stay around longer and their created levels on average are more playable and more complex than without.

**Keywords:** Mixed Initiative Tools · Reinforcement Learning · Procedural Content Generation.

## 1 Introduction

Modern games often rely on procedural content generation (PCG) to create large amounts of content autonomously or with limited human input. PCG methods can achieve many different design goals as well as enable particular aesthetics. Incorporation of PCG methods can streamline time-intensive tasks such as designing thousands of unique tree assets for a forest environment. By off-loading these tasks to AI, the time constraints put on game developers and content creators can be relaxed freeing them up to work on other tasks for which AI may be less-well suited. Through such blending of AI and the human touch a system of human and AI co-creation yields not only unique game content the human designer may not have even considered alone but also enables new creative directions [14].

In Procedural Content Generation via Reinforcement Learning, or *PCGRL* [9], levels are first randomly generated and then incrementally improved. The generated levels are initially good enough that they could–though unlikely good enough that they *would*–be used by a human designer. That reluctance to use a level could arise from a level's misalignment with the human designer's needs and they would likely have to keep generating new levels until they find one that is satisfactory. Generally speaking, the human designer exerts minimal control over the resulting level's features and may end up generating many just to find one that suits their needs.

In order to make this level generation method more compatible with a human designer's workflow, we leverage the incremental nature of PCGRL in building a mixed-initiative level-editing tool. This paper presents *RL Brush*, a human-AI collaborative tool that balances user-intent and AI model suggestions. *RL Brush* allows a human designer to create levels as they please while continuously suggesting incremental modifications to improve the level using an ensemble of AI models. The human designer may choose to accept suggestions as they see fit or reject them. The tool thereby aims to assist and empower human designers to create levels that are good, unique, and suitable to the user's objectives.

## 2 Related Work

Procedurally generated content has been used in games since the early 1980s. Early PCG-based games like *Rogue* (Michael Toy, 1980) used PCG to expand the overall depth of the game by generating dungeons as well as coping with the hardware limitations of the day [14, 21]. This section will lay out more contemporary applications and methods of generating game content procedurally, with a focus on the use of reinforcement-learning-based approaches.

### 2.1 PCG via Reinforcement Learning

Reinforcement Learning (RL) is a Machine Learning technique where, typically, an agent takes action in an environment at each time-step and desirable actions are reinforced, interpreted as state and reward, from the environment [18]. Most of the RL work in games focuses on playing. We suspect it may be due to the direct and easy way of representing the game playing problems as Markov decision processes. On the other hand, representing content generation as a Markov decision process poses challenges and hence could explain the disproportionally smaller number of works using RL in game content generation problems.

Of the existing works describing RL approaches to game content generation, a few approaches stand out and demonstrate the breadth of possibilities. Chen et al. [5] demonstrate using Q-learning to create a card deck for collectable cards games. Guzdail et al. [7] have shown how active learning can be used to adjust an AI agent to adapt to user choices while creating levels for Super Mario Bros. PCGRL[9] introduces reinforcement learning into level generation by seeing the design process as a sequential task. Different types of games provide information on the design task as functions: an evaluation function that assesses the quality of the design and a function that determines whether the goal is reached. RL agents that *play* out the content generation task defines the state space, action space, and transition function. For typical 2D grid based games, the state can be represented as a 2D array or 2D tensor. Agents of varying representation observe and edit the map using different patterns. The work demonstrates how three types of agents, namely *narrow*, *turtle* and *wide*, can respectively edit tiles in a sequential manner, move on the map in a turtle-graphics-like way and modify the passed tiles, or have control to select and edit any tile in the entire map.

### 2.2 PCGRL Agents

The three RL-based level-design agents introduced in *PCGRL* [9] [3] as *narrow*, *turtle* and *wide* have origins in search-based approaches to level-generation, however the primary focus of the subsequent sections will be on their RL-based implementations. This section describes these three canonical agent types.

**Narrow** The *narrow* agent observes the state of the game and a *location* $(x, y)$ on the 2D-array grid representation of the game level. Its action space consists of a *tile-change* action: whether to make a change or not at location $(x, y)$ and what that change would be.

**Turtle** Inspired by turtle graphics languages such as *Logo* [6] [9], *turtle* agent also observes the state of the grid as a 2D array and a *location* $(x, y)$ on that grid. Like *narrow* agent, one part of its action-space is defined as a *tile-change* action. Unlike *narrow*, its action space also includes a *movement-action* in which the agent changes the agent's current position on the grid to $(x', y')$ by applying a 4-directional translation on its *location* moving it either *up*, *down*, *left* or *right*.

**Wide** The *wide* agent also observes the state of the grid as a 2D array. However, it does not take a *location* parameter. Instead, its action space selects a *location* on the grid $(x, y)$ as the *affected location* and a *tile-change* action.

### 2.3 PCG via Other Machine Learning Methods

In addition to RL, other machine learning (ML) approaches have also been applied to procedural content generation and mostly based on supervised or unsupervised learning; the generic term for this is Procedural Content Generation via Machine Learning (PCGML) [16]. *Mystical Tutor* [17], an iteration on the Twitter bot `@RoboRosewater`, generates never-before-seen *Magic: The Gathering* cards using an Long short-term memory (LSTM) neural network architecture. While Torrado et al. [19] demonstrate that *Legend of Zelda* (Nintendo, 1986) levels can be generated using generative adversarial networks (GAN). However, one distinction that arises when comparing PCGRL with other types of PCGML: PCGRL does not strictly require ahead-of-time training data. RL-based models utilize a system of reward functions instead, which can be either manually designed or themselves learned. PCGRL's system of incremental approach to level-generation also distinguishes it from more holistic ML approaches such as many GAN-based PCGML approaches. At each time step, the agent takes an action such as moving to or selecting a certain position (for example, in 2D grid space) or changing the tile at the current position. This characteristic of PCGRL makes it well-suited for mixed-initiative design.

### 2.4 PCG via Mixed-initiative Level Design

In mixed-initiative design, the human and an AI system work together to produce the final content [20, 22]. Multiple mixed-initiative tools for game content creation have been developed in recent years. *Tanagra*[15] is a prototype mixed-initiative tool for platformer level design in which AI can either generate the entire level or fill in the gaps left by human designers. *Sentient Sketchbook*[10] is a tool for designing a *Starcraft*-like (Blizzard, 1998) strategy game. Users can sketch in low-resolution and create an abstraction of the map in terms of player bases, resources, passable and impassable tiles. It uses feasible-infeasible two population GA (FI-2pop GA) for novelty search and generates several map suggestions as users are sketching. An example of a mixed-initiative PCG tool that generates levels for a specific game is *Ropossum*, which creates levels for the physics-based puzzle game *Cut the Rope*, based on a combination of grammatical genetic programming and logic-constrained tree search [13, 12]. Another such example is the mixed-initiative design tool for the game *Refraction*, which teaches fractions; that tool is built around a constraint-solver which can create puzzles of specific difficulty [4].

More recently, Alvarez et al.[2] introduced Interactive Constrained MAP-Elites for dungeon design, which offers similar suggestion-based interaction supported by MAP-Elites algorithm and FI-2pop evolution. Guzdial et al.[8] proposed a framework for co-creative level design with PCGML agents. This framework uses a level editor for *Super Mario Bros* (Nintendo, 1985), which allows the user to draw with a palette of level components or sprites. After finishing one turn of drawing, the user clicks the button to allow the previously trained agent to make additions sprite-by-sprite. This tool is also useful for collecting training data and for evaluating PCGML models. In a similar vein, Machado et al. used a recommender system trained on databases of existing games to recommend game elements including sprites and rules across games [11].

## 3 Methods

This section introduces *RL Brush*, a mixed-initiative level-editing tool for tile-based games that uses an ensemble of trained level-design agents to offer level-editing suggestions to a human user. Figure 1 shows a screenshot of the tool [3]. The present version of *RL Brush* is tailored for building levels for the classic puzzle game *Sokoban* (Thinking Rabbit, 1982) and generating suggestions interactively.

### 3.1 Sokoban

*Sokoban*, or "warehouse keeper" in Japanese, is a classic 2-D puzzle game in which the player's goal is to push boxes to their designated locations within an enclosed space (called goals). The player can only push boxes horizontally or

[3] https://rlbrush.app/

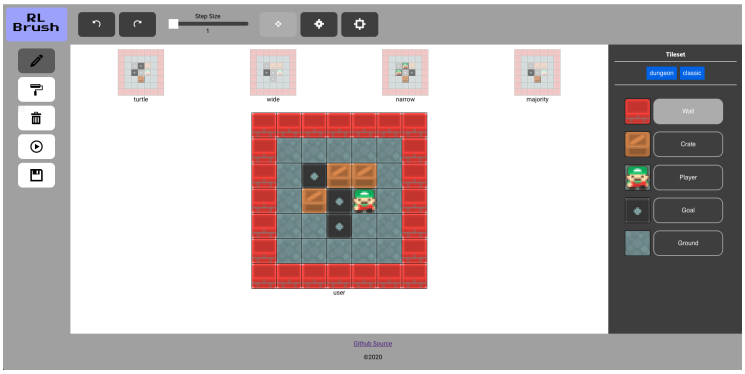



Fig. 1: RL Brush screenshot of the Sokoban level editor.

vertically. The number of boxes is equal to the number of designated locations. The player wins when all boxes are in the correct locations.

### 3.2 RL Brush

In the spirit of human-AI co-creation of tools like *Evolutionary Dungeon Designer* [1] and *Sentient Sketchbook* [10], *RL Brush* interactively presents suggested edits in to a human level creator, 4 suggestions at a time. Instead of using search-based approaches to generate the suggestions *RL Brush* utilizes the reinforcement-learning-based level-design agents presented by [9]. *RL Brush* builds on the work introduced by *PCGRL* [9] by combining user-interactions with the level-designing *narrow*-, *turtle*- and *wide*-agents and an additional *majority*, meta-agent into a human-in-the-loop[4], interactive co-creation system.

### 3.3 Architecture Overview

Figure 2 shows the system architecture for our tool RL Brush. The system consists of 4 main components:

- **GridView** is responsible for rendering and modifying the current level state.
- **TileEditorView** allows the user to select tools to edit the current level viewed in the GridView.
- **SuggestionView** shows the different AI suggestions from the current level in the GridView.
- **ModelManager** updates all the suggestions viewed in SuggestionView if the current level changed in the GridView.

The user can edit the current level ($G$) either by selecting a suggestion from the $SuggestionView$ or by using a tool from the $TileEditorView$ and

[4] These are a subclass of AI-based systems that are designed around human interaction being one of their components

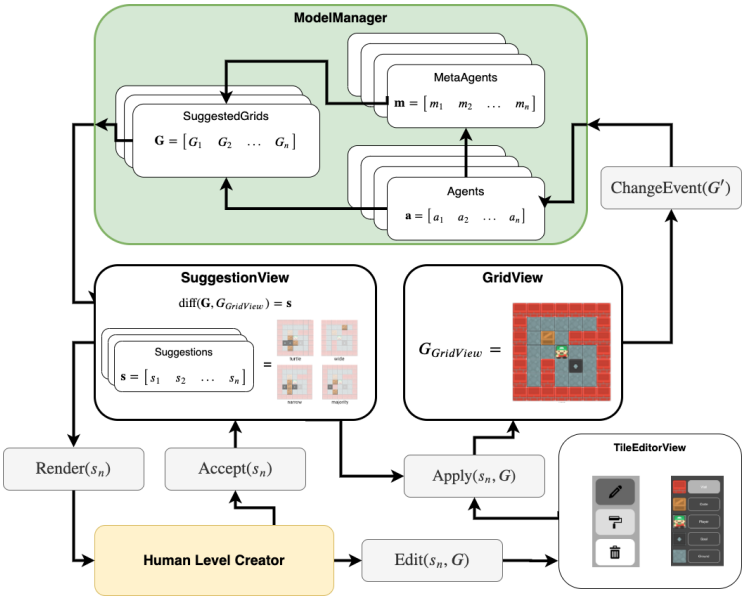



Fig. 2: RL Brush System Arcihtecture.

modifying directly the map. This change emits a *ChangeEvent* signal to the *ModelManager* component with the new grid ($G'$). The ModelManager runs all the AI models and collects their results and send the results back to the *SuggestionView*. The *ModelManager* will be described in more details in subsequent section.

### 3.4 Human-Driven, AI-Augmented Design

Both the `TileEditorView` and the `SuggestionView` respond only to user-interactions in order to ultimately provide the human in the loop the final say on whether to accept the AI suggestions or override them through manual edits. The goal is to provide a *best-of-both-worlds* approach to human and AI co-creation in which the controls of a conventional level-editor can be augmented by AI suggestions without replacing the functionality a user would have expected from a manual tile editor. Instead, the human drives the entire level design process while taking on a more collaborative role with the ensemble of AI level-design agents.

### 3.5 `ModelManager` Data Flow

The `ModelManager` in figure 2 handles the interactions with the PCGRL agents $a = [a_0\ a_1\ ...\ a_x]$ (where $x$ is the number of used PCGRL agents) and meta-agents $m = [m_0\ m_1\ ...\ m_y]$ (where $y$ is the number of used meta-agents). The `ModelManager` gets the current level state and sent to these agents where they

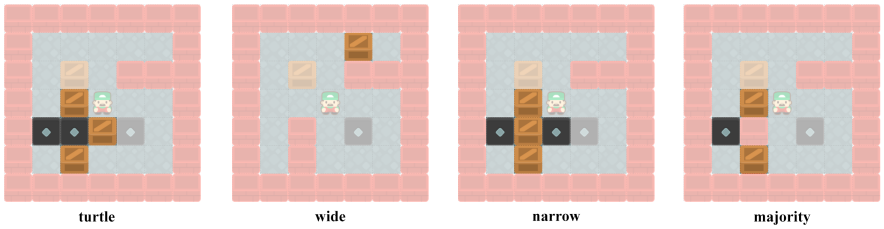



Fig. 3: Each suggestion in the UI is generated by a different agent whose name appears below its diff rendering. Clicking on the suggestion applies it to the grid $G$.

edit it then it emits a stream of `SuggestedGrid` objects $G = G_0\ G_1\ ...\ G_{x+y}$. The `SuggestionView` in turn observes the stream of $\mathbf{G}$ lists and uses them to generates suggestions $\mathbf{s}$ from $\mathbf{G}$ by diffing them against the current level state $G_{\texttt{GridView}}$ to generate a list of suggestions $s = \left[s_0\ s_1\ ...\ s_{x+y}\right]$ for rendering and presenting the user in the UI's suggestion box (figure 3).

Meta-agents in $\mathbf{m}$ consist of agents that combine or aggregate the results of $\mathbf{a}$ in some way to generate their results. In *RL Brush*, the *majority* agent is an example of a meta-agent that aggregates one or more of the agents suggestions ($\left[G_{a_i}\ G_{a_{i+1}}\ ...\ G_{a_j}\right]$) to a new suggestion ($G_{m_i}$). The *majority* meta-agent is powered by a pure, rule-based model that only makes a suggestion of a tile mutation if the majority of the agents have the same tile mutation in their suggestions. In our case, we are using 3 different PCGRL agents (narrow, turtle, and wide) which means at least 2 agents have to agree on the same tile mutation.

### 3.6 `ModelManager`'s Hyper-Parameters


(a) (b)

Fig. 4: These two UI elements *a* and *b* control the *step* and *tile radius* parameters respectively.

Two primary hyper-parameters exist in *RL Brush* for tuning the performance of `ModelManager`. One is the number of *steps* and the other is the *tool radius*. These are each controlled from the UI using the components in figure 4.

The *step* parameter controls how many times the `ModelManager` will call itself recursively (figure 5). For each *step* the `ModelManager` will call itself recursively

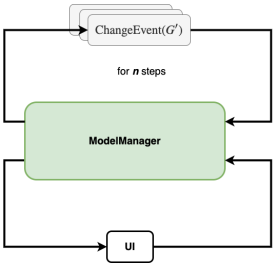



Fig. 5: The changes in the *step* parameter control the number of iterations $n$ in the loop of recursive `ChangeEvent` objects that feed back into the `ModelManager`

$n$ times on a self-generated stream of `ChangeEvent` (G') objects. Having a higher step value allows agents to make more than one modification to the map. This is an important hyper-parameter because most of these agents are trained to not be greedy and try to do modification that requires long term edits. Limiting these agents to only see one step ahead will suffocate them and their suggestions might not be very interesting for the users.

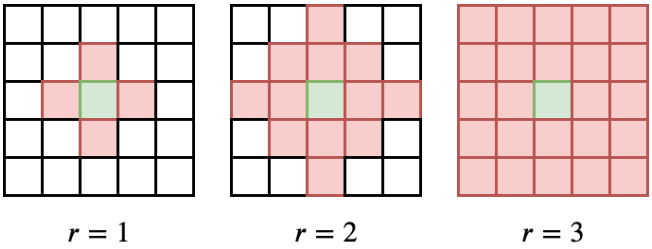



Fig. 6: The changes in the *tool radius* parameter control the size of the slice of grid *G'* that is visible to the agents as input.

The *tool radius* parameter controls how big the window of tiles are visible to the agent as input. Agents can't provide suggestions outside of this window. It focuses the suggestion to be around the area the user is modifing at the current step. In figure 6 the white tiles are padded as empty or as walls, depending on the agent. The red tiles represent the integer values of each tile on the grid $G$. The green tile represents the *pivot tile* or position on the grid $G$ that the user last clicked on if a tile was added manually. In cases where no tile was clicked[5], the center of the grid $G$ is used as the *pivot tile*. The radius $r$ refers to the Von-Neuman neighborhood's radius with respect to the *pivot tile*. However, note that

[5] Such as the case in which the user accepted an AI suggestion

for all grids $G$ where $r \geq \left\lceil \frac{gridRadius}{2} \right\rceil$, the entire grid is used such as in cases of $r = 3$ on *microbans* of size $5 \times 5$.

## 4 Experiments

In this section we demonstrate through a user study conducted to study the interactions between users and the AI suggestions. We are primarily interested in answering the following four questions:

- Q1: Do users prefer to use the AI suggestions or not?
- Q2: Does the AI guide users to designing more playable levels?
- Q3: Which AI suggestions yield higher engagement from users?
- Q4: What is the effect of the AI suggestions on the playable levels?

| Total Event Counts | |
|---|---|
| Total User Sessions | 75 |
| Total Interaction Events | 3165 |
| Total Ghost Suggestions Accepted | 308 |
| Aggregations | |
| Level Versions Per Session | 10.6 |
| Ghost Suggestions Accepted Per User Session | 4.11 |
| Total Interactions Per Session | 42.2 |

Table 1: Interaction Event Summary

For the experiment, we published the *RL Brush* app [6] to the web, and shared the link with university students and faculty on a shared Slack channel as well as on social media platforms Twitter and Facebook. We then recorded anonymized user-interaction events to the application web server. During the course of about 2 weeks, 75 unique user sessions were logged in total. Figure 7 shows the final states of a few levels created using a combination of human and edits and AI suggestions in *RL Brush*. Table 1 shows the counts of key metrics that we used to measure the interactions of users and the *RL Brush* UI. For instance, each session resulted in an average of 10.6 level versions, defined as unique levels, throughout each user's total average of 42.2 interactions with the UI (i.e. button presses or clicks) during the course of the session. For example, a user may have generated 2 level versions and 100 interactions in a session by making a single edit and just clicking "Undo" and "Redo" over and over again. From these 10.6 level versions 4.11 were generated using the AI suggested edits or *ghost suggestions*.

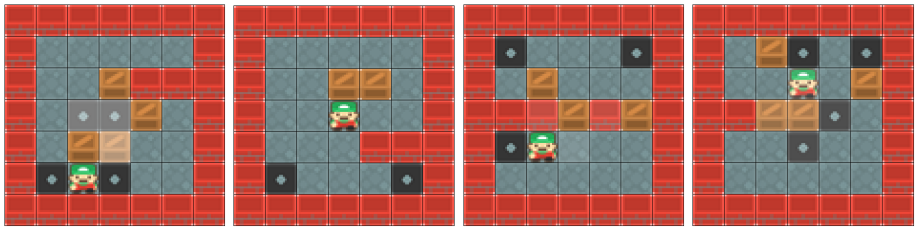


Fig. 7: User-generated levels

| | Used AI | Didn't Use AI | Total |
|---|---|---|---|
| Playable | 9 | 2 | 11 |
| UnPlayable | 8 | 20 | 28 |
| Total | 17 | 22 | 39 |

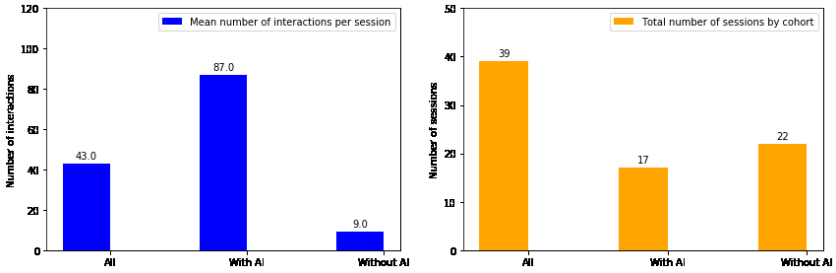
Table 2: Statistics on the 39 full session

## 5 Results

From these 75 user sessions, 39 sessions were fully logged interaction events during the session from start to finish. We analyzed these sessions on an event-by-event basis and found a few trends. Table 2 shows the statistics about all these 39 fully-logged, sessions. The amount of people that didn't use the AI (22) is slightly higher than the ones used the AI (17). There might be a lot of different reasons that users never engaged with the system, but we suspect the absence of a formal tutorial could have impacted the results here. On the other hand, users that interacted with at least one AI suggestion yielded at more playable levels (9 out of 17) than users did not interact with AI suggestions at all (2 out of 22). There is multiple different factors that could reflect this higher percentage. We think that the main factor is that the AI suggestions nudge/inspire users toward building playable levels.

One such trend, as described in figure 8a, users working alongside AI (i.e. users that accepted at least one AI suggestion in their workflow) generate a higher average number of interactions when compared with the other cohort of users working without AI and the combined average of all users. The higher rate of interaction could be attributed to user-engagement, if interpreted positively, or perhaps, if interpreted negatively, that AI increases the complexity of the workflow and requires more clicks to fix any unwanted AI actions. Since the overall average number of ghost suggestions accepted per session is 4.11, as shown on table 2, we interpret the increase in the interactions to positive engagement than an overwrought system, which we assume would have a higher number of accepted suggestions overall compared to average level versions of 10.6 as frustrated users might fight with the system and produce many more interactions

[6] https://rlbrush.app/




(a) Mean interactions by user cohort. (b) Total sessions by user cohort.

Fig. 8: The number of interactions on average is greater in the cohort of users that accepted at least one AI suggestion during their session.

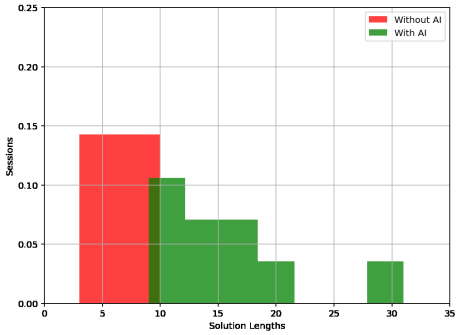



Fig. 9: Users that used AI suggestions seemed to create levels that required more steps in their solutions.

per level version. Of course, other external factors could explain the trend. The small number of users that interacted with the AI at all, as seen in figure 8b, could point to a majority of users new to level editing and without having a formal tutorial may have quit before discovering the AI features at all.

Another trend seems to be a relationship between level complexity and and using AI suggestions, as seen in figure 9. There the solution length is calculated using a BFS (Breadth-First Search) solver. Each level created with the assistance of AI is, on average, longer than levels without AI. This could indicate that AI suggestions yield direct users toward creating more complex levels and therefore higher quality levels. However, further studies on a larger set of users would need to be done to further explore this trend more definitively.

Since *RL Brush* provides different models to choose from, we were also curious to check which suggestions were most useful for the users. Figure 10 shows a histogram about which model saw the most usage, as measured by number of interactions. We found out that the suggestions generated by the majority-voting meta-agent received the most interactions. One interpretation for its unexpected

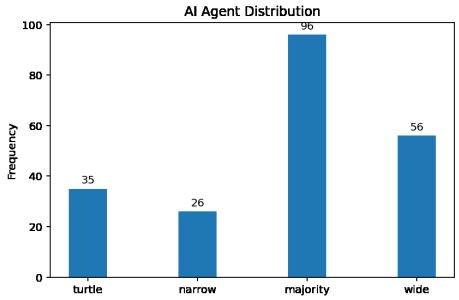



Fig. 10: The *majority* agent seems to be the most popular across sessions and received the most total interactions or clicks.

popularity could be that agents, in isolation, may have divergent lower-quality suggestions but collectively tend toward higher suggestion quality. In the *Discussion* section, we discuss plans for further investigations aggregated suggestions.

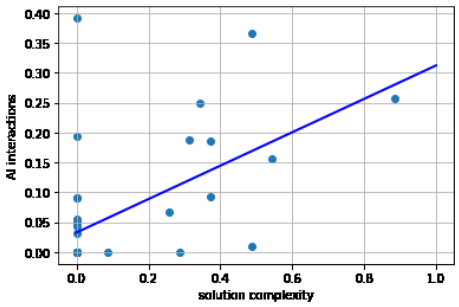



Fig. 11: Correlation between number of AI suggestions accepted and overall solution complexity.

Finally, we compute the correlation between the number of AI-accepted suggestions in a session and the solution length of the created level. Figure 11 shows a correlation (with coefficient equal to 0.279) between the number of AI suggestions used during level creation and the maximum level difficulty achieved during that session, in terms of solution length. This correlation could further make the case that AI suggestions nudge users towards creating more complex levels with longer solutions.

## 6 Discussion

The system described here can be seen as a proof of concept for the idea of building mixed-initiative interaction on PCG methods based on sequential de-

cisions. Most search-based PCG methods, as well as most PCGML methods, outputs the whole level (or other type of content) as a unit. PCGRL, where the generator has learned to design one step at a time, might afford a mode of interaction more suited to how a human designs a level. It would be interesting to investigate whether tree search approaches to level generation could be harnessed similarly [3].

Looking back at the results, we can say that *RL Brush* introduces a few areas for further exploration surrounding the relationship between user engagement and perhaps the complexity of playable levels. We also noticed that more users seem to interact more with meta-agent compared to other models. Comparing these results with our questions introduced in the Experiments section:

- Q1: Based on the data we have, we can't clearly say if the users preferred to use the system with AI or without. However, we suspect that a further study could answer this question.
- Q2: From the collected statistics the amount of playable levels generated by users that incorporated AI into their workflow exceeds the number of those that did not interact with the AI suggestions at all.
- Q3: The majority agent received the most interactions when compared to all the rest. Further studies could explore new ideas for additional types of meta-agents in future work.
- Q4: The results lean towards AI suggestions yielding higher quality levels, as defined using complexity of levels and with longer solution lengths, but more data would be needed to verify that.

In addition to the results described in the previous section, a broader test of human users could further explore the quality of the levels generated beyond the scope of automated solvers and through the use of human play-testing. Additional metrics can be gathered to support this and more targeted, supervised user research can be done here. A more supervised user research will help us understand the different factor affecting our results. We would know if external factor such as game literacy and familiarity with games and level editor affects the user their engagement with AI. We believe that users with higher game literacy may find the tool less intimidating than ones with lower game literacy. Another important study could be to understand the influence of the AI suggestion on the final created levels: were the AI suggestions pivotal for the final created levels or merely inspiration for heavily hand-made levels?

Once the broader user studies have been conducted, additional client-side models can be added to *RL Brush* that learn the weights of meta-agents and continuously optimize them through online-model training. In this way, we could better leverage the `ModelManager`'s ensemble architecture's capabilities. Furthermore, the existing PCGRL models could be extended to continuously train online using reward functions incorporating parameters based on user actions. Similarly, novel client-side models specifically tailored to improve the UX (user experience) could be incorporated into future versions that better leverage the capabilities of TensorFlow.js, which *RL Brush* utilizes in its code already.

Subsequent versions would also add support for additional games, level-design agent types and $N \times M$ grids in order to increase the overall utility of *RL Brush* as a functional design tool.

## 7 Conclusion

In the previous sections we have introduced how *RL Brush* provides a way to seamlessly integrate human level editing with AI suggestions with an opt-in paradigm. The results of the user study suggest that using the AI suggestions in the context of level editing could impact the quality of the resulting levels. In general, using AI suggestions seemed to result in more highly playable levels per session and higher overall level quality, as measured by solution length.

There is clearly more work to do in this general discussion. We don't know yet to what types of levels and other content this method can be applied, and there are certainly other types of interaction possible with an RL-trained incremental PCG algorithm. *RL Brush* will hopefully serve as a nexus of discovery in the space of using PCGRL in game-level design.

## References

1. Alvarez, A., Dahlskog, S., Font, J., Holmberg, J., Johansson, S.: Assessing aesthetic criteria in the evolutionary dungeon designer. In: Proceedings of the 13th International Conference on the Foundations of Digital Games. pp. 1–4 (2018)
2. Alvarez, A., Dahlskog, S., Font, J., Togelius, J.: Empowering quality diversity in dungeon design with interactive constrained map-elites. In: 2019 IEEE Conference on Games (CoG). pp. 1–8. IEEE (2019)
3. Bhaumik, D., Khalifa, A., Green, M.C., Togelius, J.: Tree search vs optimization approaches for map generation. arXiv preprint arXiv:1903.11678 (2019)
4. Butler, E., Smith, A.M., Liu, Y.E., Popovic, Z.: A mixed-initiative tool for designing level progressions in games. In: Proceedings of the 26th annual ACM symposium on User interface software and technology. pp. 377–386 (2013)
5. Chen, Z., Amato, C., Nguyen, T.H.D., Cooper, S., Sun, Y., El-Nasr, M.S.: Q-deckrec: A fast deck recommendation system for collectible card games. In: Computational Intelligence and Games. IEEE (2018)
6. Goldman, R., Schaefer, S., Ju, T.: Turtle geometry in computer graphics and computer-aided design. Computer-Aided Design **36**(14), 1471–1482 (2004)
7. Guzdial, M., Liao, N., Chen, J., Chen, S.Y., Shah, S., Shah, V., Reno, J., Smith, G., Riedl, M.O.: Friend, collaborator, student, manager: How design of an ai-driven game level editor affects creators. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. pp. 1–13 (2019)
8. Guzdial, M., Liao, N., Riedl, M.: Co-creative level design via machine learning. arXiv preprint arXiv:1809.09420 (2018)
9. Khalifa, A., Bontrager, P., Earle, S., Togelius, J.: Pcgrl: Procedural content generation via reinforcement learning. arXiv preprint arXiv:2001.09212 (2020)
10. Liapis, A., Yannakakis, G.N., Togelius, J.: Sentient sketchbook: computer-assisted game level authoring (2013)

11. Machado, T., Gopstein, D., Nealen, A., Togelius, J.: Pitako-recommending game design elements in cicero. In: 2019 IEEE Conference on Games (CoG). pp. 1–8. IEEE (2019)
12. Shaker, N., Shaker, M., Togelius, J.: Evolving playable content for cut the rope through a simulation-based approach. In: Ninth Artificial Intelligence and Interactive Digital Entertainment Conference (2013)
13. Shaker, N., Shaker, M., Togelius, J.: Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In: Ninth Artificial Intelligence and Interactive Digital Entertainment Conference (2013)
14. Shaker, N., Togelius, J., Nelson, M.J.: Procedural content generation in games. Springer (2016)
15. Smith, G., Whitehead, J., Mateas, M.: Tanagra: A mixed-initiative level design tool pp. 209–216 (2010)
16. Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A.K., Isaksen, A., Nealen, A., Togelius, J.: Procedural content generation via machine learning (pcgml). IEEE Transactions on Games **10**(3), 257–270 (2018)
17. Summerville, A.J., Mateas, M.: Mystical tutor: A magic: The gathering design assistant via denoising sequence-to-sequence learning. In: Twelfth artificial intelligence and interactive digital entertainment conference (2016)
18. Sutton, R.S., Barto, A.G., et al.: Introduction to reinforcement learning, vol. 135. MIT press Cambridge (1998)
19. Torrado, R.R., Khalifa, A., Green, M.C., Justesen, N., Risi, S., Togelius, J.: Bootstrapping conditional gans for video game level generation (2019)
20. Yannakakis, G.N., Liapis, A., Alexopoulos, C.: Mixed-initiative co-creativity (2014)
21. Yannakakis, G.N., Togelius, J.: Artificial intelligence and games, vol. 2. Springer (2018)
22. Zhu, J., Liapis, A., Risi, S., Bidarra, R., Youngblood, G.M.: Explainable ai for designers: A human-centered perspective on mixed-initiative co-creation. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG). pp. 1–8. IEEE (2018)