Explaining AI Behavior in PCGRL with Saliency Maps

Lunshuo Tian Chenyi Liao

ABSTRACT

Procedural Content Generation(PCG) reduces the workload of the humans in the content generation process. Procedural Content Generation via Reinforcement Learning(PCGRL) enhance this aspect. However, the black-box nature challenge users without reinforcement Learning(RL) expertise to better improve the RL models. Existing RL explanation is most foucus on playing game but not on designing game. Explaining PCGRL can refer previous methods but cannot simply copy them due to difference between agent foucus on playing game and PCGRL. We review the existing PCGRL model and conduct experiment based on it. We use perturbation-based method with saliency maps to explain the PCGRL model and calculate the saliency with two different methods, the action-based method and the reward-based method. Evaluate their explainability with several little imperfect maps to see whether the saliency maps indicate the locations led to imperfections. We demonstrate our current method, the action-based method, is with high explainability. It can help general user to understand the PCGRL model in some degrees.

1 INTRODUCTION

Procedural Content Generation (PCG) is the process of generating game content using algorithms and programs based on input provided by human users[1]. The functionality is to reduce the workload of humans in the content generation process. There are various methods for implementing PCG, including Procedural Content Generation via Reinforcement Learning (PCGRL)[2], which generates content by the Reinforcement Learning(RL) agent. This implemnentation method with the advantage of utilizing the characteristics of RL has rapid generation process and doesn't need to prepare training data[3]. However, it comes with the drawback of the black-box nature of RL agents[4], making it challenging for users without expertise in RL to understand and explain the generation process and RL agent behaviors. The lack of understanding hinders the ability to make improvements and adjustments for models. To address this issue, explainability[5] needs to be introduced to PCGRL, just like explaining other black-box behaviors of artificial intelligence.

In the past few years, some researchers have introduced some methods to explain RL agents. Milani et al.[6] categorized literature exploring XRL based on the nature of different methods. Greydanus et al.[7] conduct a series of investigative explorations into explaining Atari agents. Puri et al.[8] use SARFA to explain the actions taken by agents for board games, and for Atari games. However, the RL agents explained previously are focus on how to play games based on the game rules. Few researches study explaining the RL agents for game level or content design. The RL agents of PCGRL is the latter one.

There are differences between the two types of RL agents. The goal of agents focus on playing games is typically to maximize game scores or achieve certain objectives. Given two different game states, it's relatively easy to determine which one is closer to the goal. On the other hand, game level or content design is generating game levels or contents. While there are basic objectives such as ensuring the level is solvable and requiring a certain number of steps to solve, evaluating which of two levels is better when they both meet these

basic criteria can be challenging. In RL agents focus on playing games, significant features in the environment, such as scoring positions or danger zones, are easy to mark and explain, which is guiding the agent's actions. However, in PCGRL, such features are less straightforward, making explanation more complex. Due to those differences between RL agents focus on how to play games and PCGRL. We could not simply determine the methods of previous explanation to apply in PCGRL.

Our research is to finding a appropriate way to achieve explainability in PCGRL, that is, explain AI Behavior in PCGRL with saliency maps [10], to make common users without expertise in RL easily understand why the RL agent performs specified action in specified states and which elements play a significant role in agent actions then they are able to adjust and improve input of PCGRL with a clear direction. Our research extends explainability to the PCGRL domain, which is seldom studied by others to the best of our knowledge.

We utilize existing PCGRL models[2] to provide explanations for their generation process and outcomes. There are three different 2D-level game environments, Binary, Zelda and Sokoban, with different generation objectives and three different representations, narrow, turtle and wide with different RL action spaces. We just focus on Zelda environment in turtle representation. Input an initial 2D-level map and other relevant settings, the RL agent move its own location or modify the tile over its location at each step. We add explanations at such step with saliency maps, emphasizing which tiles take significant roles on the determination of next action. Users without RL expertise can understand those tiles emphasized is important for the current state. If users doesn't want the RL agent emphasize a specified tile, just modify the settings or parameters relevant to that tile.

2 RELATED WORK

Guzdial et al. [9] explore the explainablility of PCG with labels understandable for users. These labels are applied to generated content. By labeling various components of existing levels and training the agent on these labeled levels, users can understand the reasons behind the agent generating that content. However, this explaination of PCG is based on PCGML and differs significantly from our goal of achieving explainablility in PCGRL. It just tells us what explanation with agents can be and enhances our understanding of explaining PCG.

Milani et al. [6] categorized literature exploring XRL. Through their work, we understand various methods at different aspects and their corresponding characteristics. To facilitate an intuitive understanding of agent actions for users without RL expertise, we choose a popular state explaining method with the characteristics of High Clarity, Low Need RL Prior [10], known as saliency maps. Saliency maps are a post hoc explanation method that highlights specific parts of scenes to show which part is important for the agent's current action.

Simonyan et al. [11] visualized pixels in input images that have a significant impact on the decision-making of the intelligent agent by treating the gradients of the neural network model's input and output as saliency values. The saliency values forms the saliency maps. Though the method is based on machine learning and classifiers, we gain a deeper insight into what is saliency maps and how it works.

The formation of saliency maps involves saliency computation. One method for saliency computation is perturbation-based[12], where a portion of the current state is perturbed, the same policy is executed, and the outputs results from comparison before and after perturbation. The perturbation method varies across environments. Puri et al. [8] removed each removable piece on the board and calculating the saliency of each piece at each step. Greydanus et al. [7] blurred specific areas in each frame. Our PCGRL model operates on arrays, and for perturbation, we modify the correspond element in the array to simulate the lack of the specified tile to calculate saliency maps. For saliency computation, Greydanus et al. [7] used differences

in the value function and policy vector between the original and perturbed state. Puri et al. [8] improve this method by introducing specificity and relevance, called specific and relevant feature attribution(SARFA). We refer to Greydanus et al. [7] and calculate the diffenences in action distirbution between the original and perturbed state.

Khalifa et al. [2]have provided a tile-based 2D-level generation model that is easily understandable. In their model, there are three representations: narrow, turtle, and wide. Each representation results in different action and state spaces for the agent. Additionally, there are three game environments: Binary, Zelda, and Sokoban, each generating different types of tiles and having distinct generation goals. Each representation can be applied to all provided game environments. The model outputs levels based on a two-dimensional array, and further rendering are applied. Our explainablility implementation is based on explaining the above model. We explain each action of the agent in generating levels, providing explanations for the rationale behind each action.

3 EXPLAIN PCGRL AGENT

In this section, we will fist introduce the workflow of PCGRL model we explain. Then we propose our methods of computing saliency and displaying explainability.

3.1 PCGRL Workflow

Our experiments are based on the system provided by Khalifa et al. [2]. There are three game environments and three representations. We conduct experiments focus on Zelda environment in turtle representation.

Zelda[2] The goal in the game is to move the player to grab a key and then reach a door while avoiding getting killed by the moving enemies. To make this possible, the level has to have exactly 1 player, 1 door, and 1 key (different elements in the game) and the player has to be able to reach the door and the key in at least X steps (X can be set by user) while making sure enemies are not too close.

There are 8 kinds of tile types, including empty, wall, player, key, door, bat, scorpion and spider. The last three are enemy types that are equivalent in generation processes except appearance. They are represented by different numbers(0-7) in integer arrays for 2D-level maps. The default size of the level map is 7×11. The borders(not included in level map size) are walls. Figure 1 is an example of Zelda.

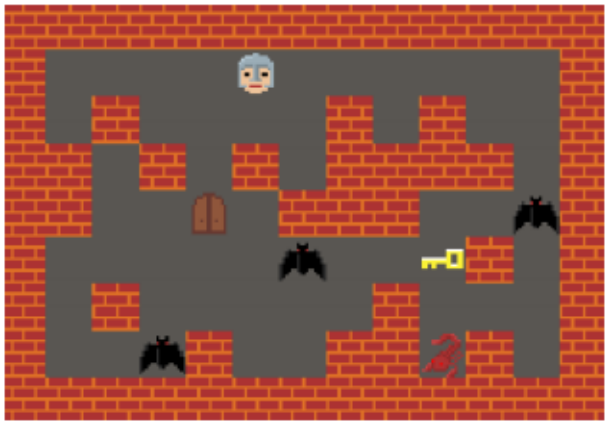

Figure 1: An example of Zelda

We have several objectives for Zelda level map design metioned above. In experiment, they all have concrete quantitative criteria, that is, the objective number ranges or distance ranges for some game elements. They are player, key, door, enemies, regions, nearest enemy and path length. The first 5 are about number ranges. The other are about distance ranges. A region is the set of empty tiles in which any two empty tiles can find an "empty" path to reach each other. Nearest enemy is the minimum of distances between player and each enemy. Path length is the smallest steps in which reach the game goal. Details are as follows:

- **player:** 1
- **key:** 1
- **door:** 1
- **enemies:** 2-5
- **regions:** 1
- **nearest enemy:** >=4

• **path length:** >=16

For each item above, a number marks its importance, called reward weight. They have different reward weights. The more the weight is, the more important the element is, that is, the criteria of the element should be satisfied first. Assume the distance between the objective and the current of an element is the minimum difference between the objective number and the current number. In training, if the current number of an element approaches the objective, agent gains the rewards of reward weight timing the distance that is shorten in this approach; if the current number of an element escapes from the objective, agent loses the rewards of reward weight timing the distance that is elongated in this approach. The detail of reward weight is as follows:

• **player:** 3
• **key:** 3
• **door:** 3
• **enemies:** 1
• **regions:** 5
• **nearest enemy:** 2
• **path length:** 1

turtle[2] At each step, the agent can move around and modify certain tiles along the way. The observation space is represented as the current state (as the 2D integer array) and the current agent location (as an x and y index in the 2D array). The action space is defined as: a movement action (which changes the agent's current location by moving it either up, down, left, or right) or a change tile action (value between 0 to n - 1 where n is the number of different objects provided by the problem module).

The action space in Zelda consists of four direction movement actions and change tile actions for each tile types, totally 12 actions. We use integers to represent actions, 0-3 for direction movement actions and 4-11 for change tile actions. The observation space in Zelda is the 7×11 level map and the current agent location. We use a 2D array to save the level map state, but how about the current agent location? We expand the origin 7×11 level map around that location. The new map is twice as big as the old map and the center of that new map is the location needed. The parts not related to the level state are filled with 1s (where 1 corresponds to wall tiles). Then, we convert the array into a one-hot array and then input the one-hot array to the agent to obtain the corresponding action probability distribution. We select the action with highest action probability distribution as the next action, and modify the level map state accordingly.

Above is the whole workflow of the Zelda-turtle PCGRL model. The training process is similar to it. The training algorithm is Proximal Policy Optimization (PPO) [13] on Stable Baselines. The policy model is a feed forward agent network with 3 convolution layers followed by a fully connected layer. The model is trained for 100 million frames.

3.2 Explanation Method

In computing saliency, we employed the perturbation-based saliency method. Initially, our perturbation-based method is to select a position and change the tile of that position to each of other tiles and restore the level map state respectively to gain the reward of such modification. We choose the max reward as the saliency of that position. We call it reward-based method. Let t be the current time step, (i, j) be the selected position, I be the original level state, A be the set of all change tile actions , and π be the action probability distribution of the agent. The saliency is the max reward.

$$S_\pi(i, j, t) = max_{a \in A}(R_\pi(I, a))$$

Then We improve the perturbation-based method to calculate saliency by perturbing the one-hot array in the workflow after we know the next action and before the agent perform the action to the current level state. We select a position in the level map, set the corresponding one-hot vector in the one-hot array to all zeros, then obtain the new probability distribution of the next action, and compare it with its original probability distribution to gain the saliency of that position. We call it action-based method. Let t be the current time

step, (i, j) be the selected position, I be the original level state, I' be the perturbed level state, a be the next action and π be the action probability distribution of the agent. The saliency is calculated by computing the difference between the two action probability distributions.

$$S_\pi(t, i, j) = |\pi(I_t, a) - \pi(I_t^{'}, a)|$$

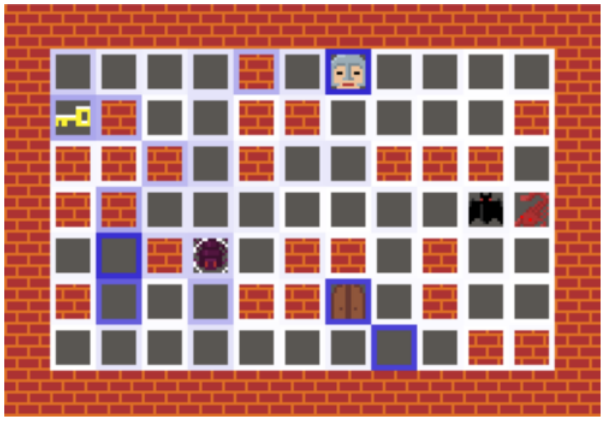

Figure 2: An example of saliency map in PCGRL

The saliency obtained by both methods are just saliency arrays and not intuitive. We convert these saliency arrays into corresponding RGB values [8] and add the corresponding colored boxes to the original level input state(as Figure 2 shows). The darker the color, the higher the saliency at that position(as figuere 3 shows).

For the reward-based method, the saliency map indicates in which locations the agent perform the specified change tile action can gain the max reward. The locations with high saliency may appear the next change tile action for the agent to gain high rewards. For the action-based method, the saliency indicates the impact of the tile at that position on the agent's next action in the current state. Changing the corresponding one-hot vector to all zeros at the position simulates the lack of information at that position. If the lack of information at that position has a significant impact on the agent's next action probability distribution, then the saliency at that position is high.

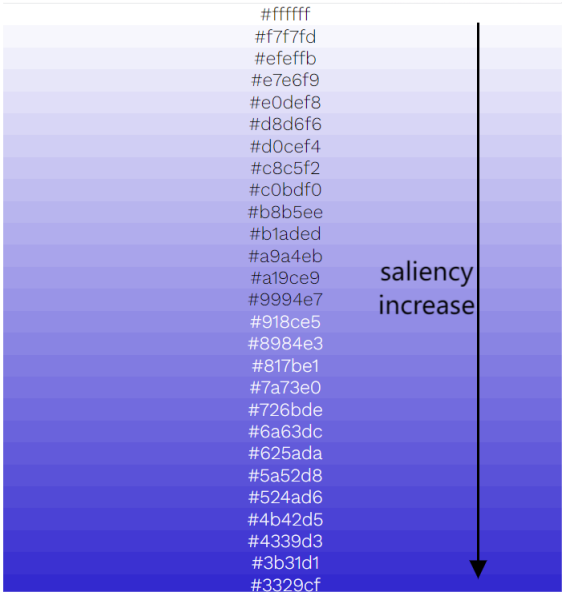

Figure 3: The relation between color and saliency

4 EVALUATION

For existing explanation methods, we need to perform validation and compare their explanatory capabilities with two methods. In last semester's experiment, we roughly evaluated the saliency maps by calculating the frequency of modifying tiles in highly significant areas with the reward-based method as figure 4 shows. It shows that the reward-based can explain the model in some degrees. However, this method is relatively crude and cannot comprehensively assess the performance of saliency maps. This time, we will evaluate the saliency maps from both quantitative and qualitative aspects.

First, we will use the PCGRL model to generate several playable, high-quality Zelda game maps. Based on the reward calculation rules of the Zelda game, we will change perfect maps by violating the 7 criteria that significantly impact the maps. The corresponding violating implementations are multiple regions, multiple doors, multiple keys, multiple players, multiple enemies, short game path and enemies too close to player. We implement it

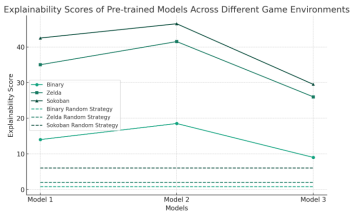

Figure 4: The explainability scores of saliency maps for the decisions of the three models in the Turtle representation across three game environments with reward-based method, as well as the corresponding scores for random policie

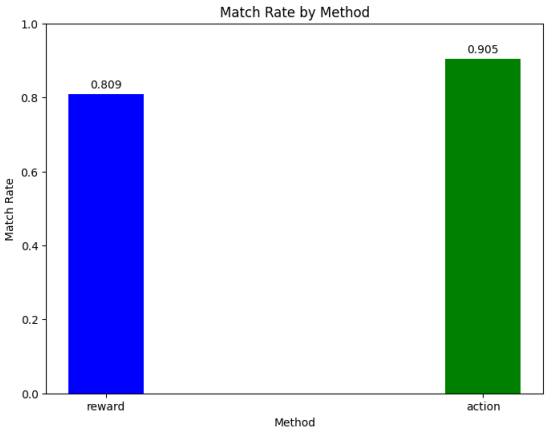

Figure 5: The comparison of match rate

by modifying a important tile to another tile based on some perfect maps.

4.1 Quantitative Evaluation

match rate After obtaining dozens of imperfect sample maps through modification, we will have the agent make decisions to modify the maps on these imperfect maps. We will then analyze and statistically determine whether the saliency maps can effectively highlight the modified locations. During this process, the agent might directly modify the tiles that were modified compared to the perfect map, or it might make other better decisions instead of repairing the modified tiles. The process of determining whether the saliency maps match the agent's decisions is complex and cannot be simply programmatically quantified, thus it relies on human analysis and statistics. If the agent modifys the location modified manually before and the saliency of that loaction is high or the agent doesn't modify the location modified manually before and the saliency of that loaction is low, we call a match of the imperfect sample map. The match rate is the number of match divided by total number of imperfect sample maps. We analyzed and compared the performance of the action-based method and the reward-based method on these dozens of imperfect sample maps, and obtained the following results as figure 5.

It can be observed that the match rate of the action-based method, which calculates saliency maps based on actions, is higher than the reward-based method, which calculates saliency maps based on rewards.

calculation rate Assuming the time required for the model to forward propagate and obtain the prediction results is t_p, and the time required for the game environment to execute the prediction results is t_s, with the map dimensions being m and n, and the action space length being l, the calculation time for the saliency map using the reward-based method is approximately $m \times n \times l \times (t_p + t_s)$, while the calculation time for the saliency map using the action-based method is only $m \times n \times t_p$. In the Zelda game environment, the action space length is 8, so the time required to calculate a map using the reward method is nearly ten times that of the action method. The following diagram shows the computation speeds of the two methods as figure 6. The match rate is total time divided by total steps.

4.2 Qualitative Evaluation

Based on the analysis of the samples and code implementation, the reward method only targets the saliency value of the current state, having only short-term effects, while the action method can

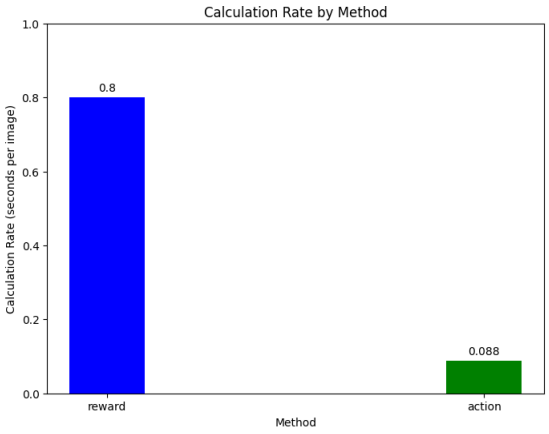

Figure 6: The comparison of calculation rate

explain future behavior through saliency value calculation. According to the reward function settings, the reward method tends to produce many tiles with the same saliency value, whereas the action-based method assigns each tile a unique saliency value.

The reward method only focuses on the map state, with the agent's position having no impact on the saliency calculation. In contrast, the action method considers both the map state and the agent's position. Due to the influence of the discount factor, the saliency of tiles near the agent is generally higher than that of distant tiles. This design better reflects the current actual situation and needs, making the saliency value distribution more reasonable and dynamic.

5 CONCLUSION

In this research, we evaluated the existing explanation method from both quantitative and qualitative aspects and compared it with the reward-based method. The experimental results and analysis indicate that the saliency map calculation method based on actions is superior to the reward-based method in several aspects. The main conclusions are as follows:

- **Match Rate**: By executing agent decisions on imperfect Zelda maps and analyzing whether the saliency maps can effectively highlight the repaired locations, we found that the match rate of the current action-based method is significantly higher than that of the previous reward-based method. This indicates that the action method is more accurate in explaining agent decisions and better matches the modifications made by the agent.
- **Calculation Rate**: The time required to calculate saliency maps based on actions is much lower than that for the reward-based method. In the Zelda game environment, the calculation time for the reward method is nearly ten times that of the action method. This result demonstrates the significant advantage of the action method in computational efficiency, especially when dealing with large-scale maps.
- **Future Behavior Explanation**: The reward method only calculates saliency values for the current state, primarily affecting short-term decisions. In contrast, the action method's saliency value calculation can explain future behavior, taking long-term impacts into account, making decisions more comprehensive and reasonable. This is also the main reason why the action method's match rate is higher than that of the reward method.
- **Saliency Distribution**: The reward method tends to produce many tiles with the same saliency, which may not explain the agent's decisions in some cases and can be confusing. In comparison, the action method assigns a unique saliency value to each tile, providing richer and more diverse decision information.
- **Impact of Agent Position**: The reward method only focuses on the map state, ignoring the impact of the agent's position on saliency calculation. The action method, however, considers both the map state and the agent's position, resulting in higher saliency for tiles near the agent compared to distant tiles. This design better reflects the current situation, making the saliency distribution more reasonable and dynamic, aiding in understanding the agent's priority decisions in the local environment.

In summary, the action-based saliency map calculation method demonstrates significant superiority in both explainability and computational efficiency. It not only better explains and guides the agent's decisions but also provides higher efficiency in large-scale computations. Future research can further optimize this method and explore its feasibility and effectiveness in other game environments and applications.

REFERENCES

[1] Cheong, Y. G., Bae, B. C. (2013). Procedural Content Generation for Games. Communications of the Korean Institute of Information Scientists and Engineers, 31(7), 16-25.

[2] Khalifa, A., Bontrager, P., Earle, S., Togelius, J. (2020, October). Pcgrl: Procedural content generation via reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 16, No. 1, pp. 95-101).

[3] Sutton, R. S., Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[4] Khadivpour, F., Guzdial, M. (2020). Explainability via responsibility. arxiv preprint arXiv:2010.01676.

[5] Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., ... Herrera, F. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information fusion, 58, 82-115.

[6] Milani, S., Topin, N., Veloso, M., Fang, F. (2022). A survey of explainable reinforcement learning. arxiv preprint arxiv:2202.08434.

[7] Greydanus, S., Koul, A., Dodge, J., Fern, A. (2018, July). Visualizing and understanding atari agents. In International conference on machine learning(pp. 1792-1801). PMLR.

[8] Puri, N., Verma, S., Gupta, P., Kayastha, D., Deshmukh, S., Krishnamurthy, B., Singh, S. (2019). Explain your move: Understanding agent actions using specific and relevant feature attribution. arXiv preprint arXiv:1912.12191.

[9] Guzdial, M., Reno, J., Chen, J., Smith, G., Riedl, M. (2018). Explainable PCGML via game design patterns. arxiv preprint arxiv:1809.09419.

[10] Qing, Y., Liu, S., Song, J., Wang, H., Song, M. (2022). A survey on explainable reinforcement learning: Concepts, algorithms, challenges. arXiv preprint arxiv:2211.06665.

[11] Simonyan, K., Vedaldi, A., Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. arxiv preprint arxiv:1312.6034.

[12] Dhurandhar, A., Chen, P. Y., Luss, R., Tu, C. C., Ting, P., Shanmugam, K., Das, P. (2018). Explanations based on the missing: Towards contrastive explanations with pertinent negatives. Advances in neural information processing systems, 31.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arxiv:1707.06347, 2017.