# Procedural Generation of Rush Hour Levels

Gaspard de Batz de Trenquelleon[1,2], Ahmed Choukarah[1,3], Milo Roucairol[1], Maël Addoum[4], and Tristan Cazenave[1]

[1] LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France
[2] École Polytechnique, Palaiseau, France
[3] École Normale Supérieure - PSL, Paris, France
[4] ISART Digital, Paris, France

**Abstract.** Procedural generation of puzzle games allows for more varied and diverse levels, which provides a better gaming experience. However, the generation based on deep-learning approaches is very challenging with this genre of games due to their inherent complexity relative to strict design rules and discrete components. In this work, we propose a framework that is composed of three main modules: a solver and two classifiers. Instances of *Rush Hour* are generated randomly and the solver aims to assess the playability of the generated levels by classifying solvable and unsolvable levels and by evaluating the difficulty of solvable levels. Two neural networks are used to improve the generation. One network is trained to differentiate between solvable and unsolvable levels and the other is trained to classify the difficulty of levels. The robustness of the framework is examined on 6×6 and 7×7 grid dimensions. The results obtained showed the effectiveness of our modules in generating interesting and various levels with four degrees of difficulty for the *Rush Hour* puzzle, where the classifier was able to quickly identify unsolvable levels. Our framework could be easily adapted and extended for procedurally generating other game genres, such as Platformer or RogueLike.

**Keywords:** Procedural level generation, Deep Learning, Puzzle games.

## 1 Introduction

*Rush Hour* is a puzzle game, initially not a video game, in which the player has to unblock a red car from a crowded parking lot. In the initial version, it takes place on a 6x6 grid with only movable cars, but variants exist on larger grids or with unmovable blocks.

We explored different methods to generate levels of *Rush Hour* by trying to maximize the interest of a level according to the number of moves needed to complete it, as well as to detect unsolvable levels.

Using Generative Adversarial Network (GAN) architecture proved to be an inadequate solution. Hence, we attempted to create a discriminator that would quickly classify solvable and unsolvable levels to generate then random levels and ranking them according to their interesting degree.

In order to provide unsolvable levels for our training set, we used a solver and a random generator despite that is not the most efficient way. A neural network was trained to recognize solvable from unsolvable levels. Since the preliminary results are encouraging, the network is being examined to generate more complex and larger grids for which generation could be computationally expensive with the current solver.

The rest of the paper is structured as follows : Section 2 presents related work and Section 3 describes the application of a solver algorithm to classify solvable or unsolvable levels. Section 4 gives a more detailed insight into the neural network architecture used for the random generator. Section 5 concludes with our findings and proposes possible future work.

## 2 Related Work

PCG via Machine Learning (PCGML) is an emerging technology area that generates new game content using machine-learned models on existing content. Recently, GAN has produced significant results as a new approach for PCG in the computer vision field, such as video games.

### 2.1 PCG for Puzzles Games

Puzzle games are a specific type of game and there are several categories for puzzles such as Sokoban-type, Sliding, Tile-Matching, Mazes, Path-building, Physics-based and Narrative Puzzles [4]. PCGML techniques have mainly focused on Roguelike or Platformer game genres, because the generation of puzzles requires significant knowledge of specific design rules and constraints. Hence, most PCG approaches to generating puzzle levels have mostly focused on search-based methods or constructive algorithms such as the Markov chain. Sturtevant et al. used a large-scale best-first search-based approach for analysis and content generation for Fling! (2013), a puzzle of Sokoban-type and tile-matching [16]. Collette et al. generated hard configurations of the *Rush hour* puzzle using a constructive method based on symbolic technique with binary decision diagrams that allows to iteratively compute reachable configuration from a set of solvable initial levels for *Rush Hour* [2]. The constructive methods have shown their robustness for Narrative puzzle generation [3][5][6]. Recently, Monte Carlo Tree Search (MCTS) approach has been widely used to procedurally generate different game contents as an optimization problem. Kartal et al. used the MCTS approach to automatically generate *Sokoban* levels that are guaranteed to be solvable with varying sizes and difficulty [11].

### 2.2 GAN for PCG

The GANs have shown their ability for platformer and Roguelike game genres. Schubert et al. proposed TOAD-GAN (Token-based One-shot Arbitrary Dimension GAN) to generate *Super Mario Bros* 2D platform levels using a single

original level for training [14]. This approach is inspired by SinGan architecture that allows learning from a single image [15]. Torrado et al. used a new GAN called Conditional Embedding Self-attention GAN architecture to condition their generation process and obtain the targeted token distributions in the generated levels [13]. Gutierrez et al. coupled GAN models to generate individual rooms of *The Legend of Zelda* with a graph grammar that combines these rooms into a dungeon [7]. GANs have also been applied to generate content for serious games, Park et al. used Deep Convolutional GAN (DCGAN) approach to automatically generate levels for educational games that incorporate the desired learning objectives with an adaptive gameplay [12]. A similar approach was used to generate game scenarios and context images while taking specific features into account to provide only plausible and enjoyable images for the game design [10]. Concerning board games, there is little research documenting the ability of the GAN approach to produce playable levels for puzzles. Hald et al. applied GAN to generate puzzle levels for Lily's Garden. While their GANs allow a good map-shape to be generated, they were not able to produce the targeted token distribution [8].

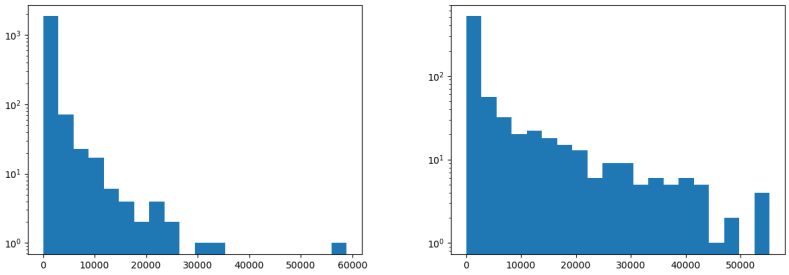
## 3 *Rush Hour* Solvers

Solving a *Rush Hour* grid is a non trivial problem as a single grid often has over 10 000 possible combinations, Figure. 1. The problem can be considered as a graph where the nodes are the states of the grid. Each two nodes are related if we can get from one to the other by moving a car one block away. We naively started with a Breadth-First search algorithm that solved the puzzle. This is a slow and inefficient algorithm as it explores irrelevant positions. It should be noted that the entire graph was explored because no appropriate criteria were identified to find the unsolvable levels. To improve our search, we use a heuristic that determines whether a grid is close to the solution or not, see Fig. 2. It is based on a score calculated by the algorithm 1.

As shown in Figure 2, the lower the score, the less the red car is blocked in the parking lot and therefore the better the grid is (a zero score represents a solved grid).

Using this score, the initial algorithm is modified into a Greedy-Best First Search algorithm 2. This algorithm significantly improves the time required to solve the solvable puzzles. However, it did not considerably improve the complexity of the unsolvable puzzles, as the entire graph still to be explored.

Figures 3(a) and 3(b) show the times for solvable and unsolvable levels, respectively. Expect for one outlier, the GBFS algorithm is much faster than the breadth-first search algorithm, see Fig. 3(a). The time is directly correlated and similar since all states are explored, and are sufficiently large to expect a significant improvement using an AI discriminator, see Fig. 3(b).




(a) 6x6 board (2000 entries) (b) 7x7 board (750 entries)

Fig. 1: Distribution of state graph size

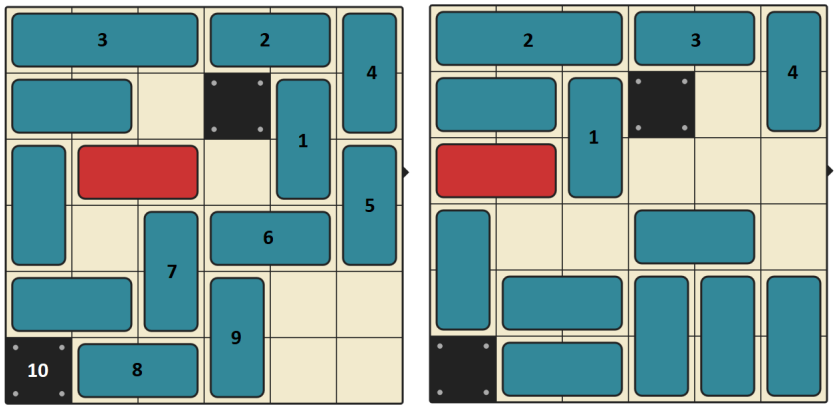
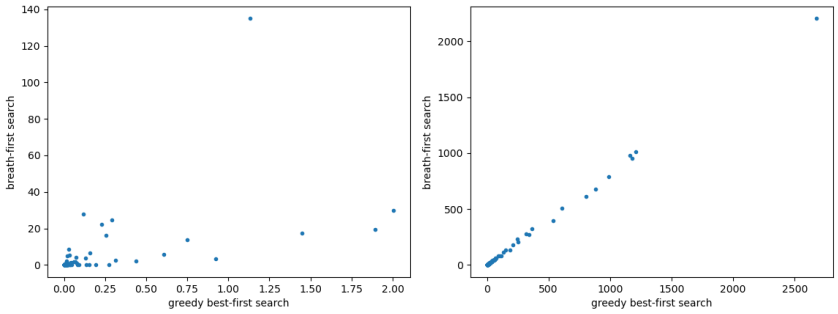



Fig. 2: Example of heuristic score evaluation

**Algorithm 1** Heuristic score

```
BlockingCars = []
for each Car between RedCar and exit do
    BlockingCars.add(Car)
end for
while BlockingCars changes do
    for Car in BlockingCars do
        for OtherCar in Board do
            if OtherCar blocks Car and OtherCar not in BlockingCars then
                BlockingCars.add(OtherCar)
            end if
        end for
    end for
end while
Score = length(BlockingCars)
return Score
```




(a) Computation time for solvable levels

(b) Computation time for unsolvable levels

Fig. 3: comparison of the computational time for the two solving methods

**Algorithm 2** Greedy best first search

```
    ToExplore = [InitialState]
 2: Explored = []
    function GBFS(ToExplore, Explored)
 4:     if ToExplore = [] then
            Return False
 6:     end if
        ToExplore.sort(HeuristicScore)
 8:     State = ToExplore.pop()
        Explored.add(State)
10:     if State is Solution then
            return True
12:     end if
        for Board in State.neighboors do
14:         if Board not in ToExplore and Board not in Explored then
                ToExplore.add(Board)
16:         end if
        end for
18:     return GBFS(ToExplore, Explored)
    end function
```

## 4 Accelerating the Generation Using Neural Networks

While, with sufficient time and computing power, the most solvable levels of the 6x6 *Rush Hour* grid can be found[1], any attempt to do so for bigger grids requires excessive amounts of computations.

Our approach was to use transfer learning to obtain an estimator on the solvability of random levels. To do so, we first used a database comprised of solvable levels (see Fig. 4) and unsolvable levels (see Fig. 5) generated randomly with a 50:50 distribution. We then use a Dense-Net [9] to learn to predict for the 6x6 grids if a setting is solvable or not.

(a) A solvable level in 17 moves (b) A solvable level in 38 moves

Fig. 4: Examples of solvable levels

(a) An example of an unsolvable level (b) Another example of an unsolvable level

Fig. 5: Examples of unsolvable levels

This specific network can have two different uses. It is known that the search algorithms become very computationally expensive as the grid sizes increase, especially for unsolvable levels. Our goal is to use transfer learning to avoid these scenarios and adapt the network to learn the solvability of higher dimension grids with few examples, as these tend to be very costly to generate. By using a neural network to dismiss all the unsolvable levels, we can speed up the generation process.

The other use for this network is to generate levels through a GAN's architecture, where our network would be the discriminator, but this still has to be explored.

### 4.1 Architecture Choice

After testing multiple architectures, it seemed that most deep networks had trouble with the task. We deduced that the added padding would dilute the information too much, as our grid size of 6x6 was too small. However, the Densenet model allows that each layer receives a concatenation of every input each previous layer has received, thus leading for a constant flow of information and avoiding information loss between each layer.

### 4.2 Database

The number of solvable and unsolvable levels must be in the same order of magnitude. In our database, there are 2,577,412 puzzle obtained from the Fogleman database. Figures 4(a) and 4(b) represent two examples of solvable levels with 17 and 38 moves, respectively. However, no source for unsolvable levels was available. To address this issue, we synthesized a number of non solvable puzzles, both easy (Fig. 5(a)) and hard (Fig. 5(b)). We then randomly select from the solvable database to construct our final database.

### 4.3 Experimental Results

**Solvability Tests - 6x6** Using the 6x6 database limited to 100 000 levels randomly selected from the initial database. This database is divided into 80% train and 20% test. For our hyperparameters, we have a learning rate of $5e^{-3}$ , cosine scheduling and weight decay with a factor of $1e^{-3}$. The algorithm quickly converge towards a 100% accuracy on both train and test set. Moreover, our approach was tested with levels including black boxes, and the accuracy rate reaches 99.9% using the same parameters. Figure 6 shows three examples of generated levels.

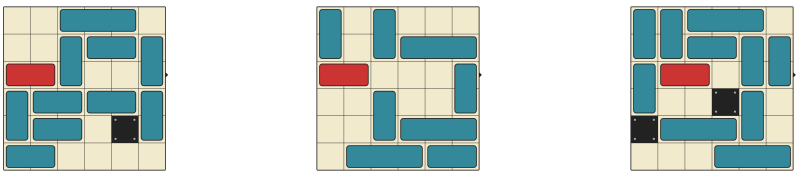


Fig. 6: Three examples of generated levels.

**Solvability Tests - 7x7 - Transfer Learning** We do not have a real database of 7x7 levels, so we opted to generate a database of levels with the Heuristic methods. With that, we generated a total of about 2300 examples, with 1800 solvable and 500 unsolvable levels. Unfortunately, this database is unable to

provide efficient training. Nevertheless, we can use our results from the 6x6 results using transfer learning as a support for the 7x7. Using this method with hyperparameters similar to the ones above, the results are obtained with 88.6% of accuracy rate.

**Learning of difficulty levels 6x6** The created database of *Rush Hour* also included the minimum number of moves necessary to solve each level. This number is used as an indicator of difficulty for each level and train a model on 5 different categories of levels : Unsolvable - Easy - Medium - Hard - Very Hard.

The idea being to have better control over our generation, as most of the levels generated can be quite simplistic.

For similar hyperparameters, the obtained accuracy reaches 87.5%. This can be explained with the confusion matrix depicted in Figure 7 where most of the misclassifications are edge cases, as difficulty can be a complicated concept to reduce to only four categories.

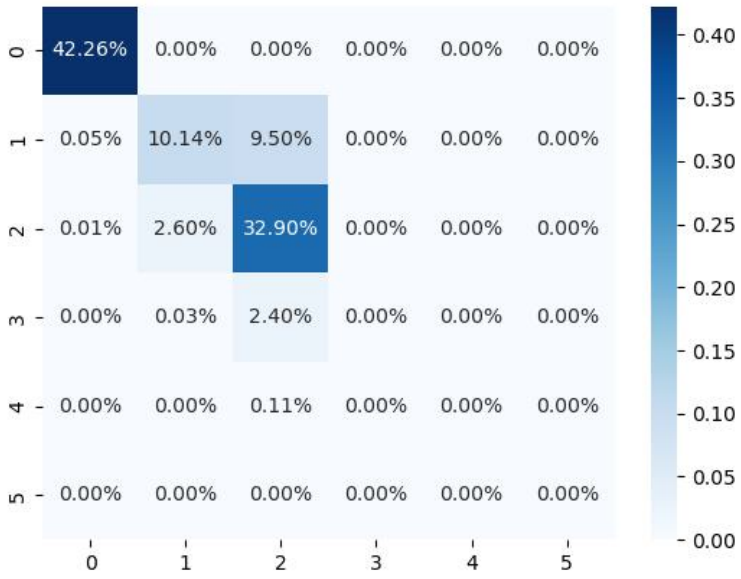



Fig. 7: Confusion Matrix of the results

### 4.4 Discussion

While we first tried approaches such as GAN, the network had difficulty grasping the importance of certain concepts in puzzles, such as puzzle complexity and form. Using our approach that is a more restricted concept gave us a better control over the generator, thus avoiding such issues. This approach also allows these results to be adapted to other games such as Sokoban, or other video games such as Angry Birds or Mario with a more restricted generation protocol. However, it is important to note that the availability of a complete database for the 6x6 and the ability to generate a small but sufficient one for 7x7 grid are what gave convincing results. Similarly, as grids become larger, computational power becomes a limiting factor for generating learning databases.

## 5 Conclusion

We defined a general approach to level generation that works well for the *Rush Hour* puzzle. We have designed a neural network able to discriminate very efficiently between solvable and unsolvable levels of *Rush Hour*. We have shown that a solver with Greedy Best First Search is considerably more efficient for solvable levels than a naive Breadth First Search, but that both algorithms take a lot of time to detect that a level is unsolvable. This property led us to design a neural network that detects unsolvable levels faster. Using a dense network trained on a dataset including unsolvable levels randomly generated and detected by our solver, we reached 100% accuracy on the detection of unsolvable levels for size $6 \times 6$. We also tested on size $7 \times 7$, and experimented with transfer learning and the automatic classification of the difficulty of generated levels.

The general intent of this research is to improve the gameplay by increasing the variability of the games using levels of varying difficulties. It is also to improve the reliability of the generating algorithm detecting unsolvable levels. It is important for the game industry since it provides a richer universe to the players.

Future work may include the use of our framework for other puzzles such as Sokoban, Angry Birds and Professor Layton and for video games such as platformers and roguelike.

## References

1. https://www.michaelfogleman.com/rush/
2. Collette, S., Raskin, J.F., Servais, F.: On the symbolic computation of the hardest configurations of the rush hour game. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) Computers and Games. pp. 220–233. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
3. Dart, I., Nelson, M.J.: Smart terrain causality chains for adventure-game puzzle generation. In: 2012 IEEE Conference on Computational Intelligence and Games (CIG). pp. 328–334 (2012). https://doi.org/10.1109/CIG.2012.6374173

4. De Kegel, B., Haahr, M.: Procedural puzzle generation: A survey. IEEE Transactions on Games **12**(1), 21–40 (2020). https://doi.org/10.1109/TG.2019.2917792
5. Doran, J., Parberry, I.: A prototype quest generator based on a structural analysis of quests from four mmorpgs. In: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games. PCGames '11, Association for Computing Machinery, New York, NY, USA (2011). https://doi.org/10.1145/2000919.2000920
6. Fernández-Vara, C., Thomson, A.: Procedural generation of narrative puzzles in adventure games: The puzzle-dice system. In: Proceedings of the The Third Workshop on Procedural Content Generation in Games. p. 1–6. PCG'12, Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2538528.2538538
7. Gutierrez, J., Schrum, J.: Generative adversarial network rooms in generative graph grammar dungeons for the legend of zelda. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8 (2020). https://doi.org/10.1109/CEC48606.2020.9185631
8. Hald, A., Hansen, J.S., Kristensen, J., Burelli, P.: Procedural content generation of puzzle games using conditional generative adversarial networks. In: International Conference on the Foundations of Digital Games. FDG '20, Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3402942.3409601
9. Huang, G., Liu, Z., Weinberger, K.Q.: Densely connected convolutional networks. CoRR **abs/1608.06993** (2016), http://arxiv.org/abs/1608.06993
10. Jiang, M., Zhang, L.: An interactive evolution strategy based deep convolutional generative adversarial network for 2d video game level procedural content generation. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–6 (2021). https://doi.org/10.1109/IJCNN52387.2021.9533847
11. Kartal, B., Sohre, N., Guy, S.J.: Generating sokoban puzzle game levels with monte carlo tree search. In: The IJCAI-16 Workshop on General Game Playing. p. 47 (2016)
12. Park, K., Mott, B.W., Min, W., Boyer, K.E., Wiebe, E.N., Lester, J.C.: Generating educational game levels with multistep deep convolutional generative adversarial networks. In: 2019 IEEE Conference on Games (CoG). pp. 1–8 (2019). https://doi.org/10.1109/CIG.2019.8848085
13. Rodriguez Torrado, R., Khalifa, A., Cerny Green, M., Justesen, N., Risi, S., Togelius, J.: Bootstrapping conditional gans for video game level generation. In: 2020 IEEE Conference on Games (CoG). pp. 41–48 (2020). https://doi.org/10.1109/CoG47356.2020.9231576
14. Schubert, F., Awiszus, M., Rosenhahn, B.: Toad-gan: A flexible framework for few-shot level generation in token-based games. IEEE Transactions on Games **14**(2), 284–293 (2022). https://doi.org/10.1109/TG.2021.3069833
15. Shaham, T.R., Dekel, T., Michaeli, T.: Singan: Learning a generative model from a single natural image. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 4569–4579 (2019). https://doi.org/10.1109/ICCV.2019.00467
16. Sturtevant, N.: An argument for large-scale breadth-first search for game design and content generation via a case study of fling! Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment **9**(3), 28–33 (Jun 2021), https://ojs.aaai.org/index.php/AIIDE/article/view/12594