Generating Educational Game Levels with Multistep Deep Convolutional Generative Adversarial Networks

Kyungjin Park
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
kpark8@ncsu.edu

Bradford W. Mott
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
bwmott@ncsu.edu

Wookhee Min
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
wmin@ncsu.edu

Kristy Elizabeth Boyer
Computer & Information Science & Engineering
University of Florida
Gainesville, FL, USA
keboyer@ufl.edu

Eric N. Wiebe
Department of STEM Education
North Carolina State University
Raleigh, NC, USA
wiebe@ncsu.edu

James C. Lester
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
lester@ncsu.edu

***Abstract*—Educational games offer significant potential for supporting personalized learning in engaging virtual worlds. However, many educational games do not provide adaptive gameplay to meet the needs of individual students. To address this issue, educational games should include game levels that can self-adjust to the specific needs of individual students. However, creating a large number of adaptable game levels requires considerable effort by game developers. A promising solution to this problem is to leverage procedural content generation to automatically generate levels for educational games that incorporate the desired learning objectives. In this paper, we propose a multistep deep convolutional generative adversarial network for generating new levels within a game for middle school computer science education. The model operates in two phases: (1) train a generator with a small set of human-authored example levels and generate a much larger set of synthetic levels to augment the training data for a second generator, and (2) train a second generator using the augmented training data and use it to generate novel educational game levels with enhanced solvability. We evaluate the performance of the model by comparing the novelty and solvability of generated levels between the two generators. Results suggest that the proposed multistep model significantly enhances the solvability of the generated levels with only minor degradation in the novelty of the generated levels.**

Keywords—Educational Game, Procedural Content Generation, Generative Adversarial Networks

I. Introduction

Digital games for learning have been recognized as a promising tool for education and training [1]. Educational games have been shown to enable the effective integration of problem solving and adaptive instruction, while promoting engaged learning [2]. They provide rich, virtual worlds for students to develop enhanced problem solving, critical thinking, and other twenty-first century skills [3][4]. A common design approach for educational games is to present students with a series of challenges incorporating progressively advanced learning objectives [5]. However, students who are unable to master prior learning objectives might either give up playing the game or resort to a trial-and-error approach to completing challenges. Likewise, students come to educational games with vastly different prior game-playing experience [6]. Providing students with an adaptive environment that presents a series of tailored challenges that directly build on their demonstrated competency with respect to educational outcomes and gaming skills has the potential to support mastery learning and engagement [7]. Instead of utilizing a static sequence of challenges, by presenting students with challenges that target specific learning objectives for which mastery has not been demonstrated, educational games could provide personalized learning experiences for all students [9]. However, this requires game developers to create vast libraries of challenges tailored to specific learning objectives and gaming skills, which is prohibitive from a development perspective.

Procedural Content Generation (PCG) is automatically generating content (*e.g.*, rules, levels, and stories) for games using algorithms requiring limited human input [10]. Because PCG enables the efficient generation of large volumes of content, it has received significant attention in the game industry as an approach to reducing development costs, while at the same time providing players with immense worlds to explore. Search-based and solver-based approaches to PCG have been widely used to generate new content for games via searching predefined content spaces [11][12]. PCG via Machine Learning (PCGML) is an emerging approach to generating novel game content using machine-learned models trained on existing content [13]. Unlike search-based and solver-based methods that generate content through searching a content space, PCGML methods generate content using a trained model directly. In parallel to these efforts, researchers have started exploring the application of PCG to educational games to reduce content development time as well as to adapt content to specific needs of students [5][14][15].

In this paper, we investigate a machine learning-based approach to generating novel levels in a game-based learning environment that supports middle school students in learning computer science concepts and practices. We present a multistep deep convolutional generative adversarial network (DCGAN) [16] for generating educational game levels with the ultimate goal of creating adaptable levels to meet individual student needs. We first train a DCGAN generator using a small set of solvable human-authored levels. This generator is used to generate a large set of training examples that are then filtered based on their solvability. These solvable levels are then used in combination with the human-authored levels to train another DCGAN generator with the objective of creating solvable levels. With only a small reduction in the novelty of the generated levels, the resulting generator exhibits significantly enhanced performance by generating a higher percentage of solvable levels compared to the generator trained only on human-authored levels.

II. Related work

Procedural content generation has been widely used in digital games. However, relatively little work has explored its use in educational games. In this section, we describe prior PCG work on level generation in games for entertainment as well as initial work exploring the use of PCG in educational games.

A. *Procedural Content Generation for Level Generation*

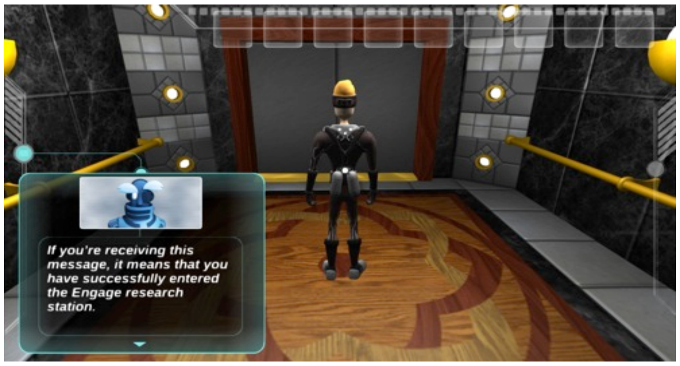
PCG is an emerging area of technology for level generation within the platform game genre. Search-based PCG uses evolutionary or stochastic optimization algorithms to search for content within a predefined content space that has certain desirable properties. These methods generally follow a generate-and-test approach that applies domain-specific evaluation functions to the generated content and tests if it exhibits the desired properties [17]. Togelius et al. used search-based PCG to automatically generate maps for a real-time strategy game, StarCraft, using a fitness function that evaluated characteristics of the map (i.e., playability, fairness, skill differentiation, interestingness) [18]. Smith and Mateas proposed Answer Set Programming as a domain-independent PCG approach that explicitly represents the design space with logical representations of the rules used for generating content [12]. Although these search-based approaches have had success in generating novel content for games, authoring the representation of the content space often requires significant expertise [19]. To address this issue, machine learning approaches that generate content directly from trained models have been an active area of PCG research [13]. Dahlskog et al. [20] examined *n*-gram models, which were further extended by Summerville et al. using Monte Carlo Tree Search [21]. Snodgrass and Ontañón suggested various types of Markov models, multidimensional Markov chains, hierarchical multidimensional Markov chains, and Markov random fields, to generate tile-based Super Mario Bros. levels by learning patterns from a training corpus [22][23]. Summerville and Mateas investigated using long-short term memory recurrent neural networks that leveraged information about player paths through levels to generate better tile-based Super Mario Bros. levels [24]. Most recently, variations of generative adversarial networks (GANs) [25], a generative method that has produced significant results in the computer vision field [16], have started to be applied to level generation [26][27]. An important aspect of this method is its ability to learn the implicit structure of levels through multiple levels of abstraction supported by deep neural networks. Giacomello et al. used GANs to generated DOOM levels [26]. Volz et al. proposed a deep convolutional generative adversarial network (DCGAN) using Wasserstein distance [28] to generate tile-based Super Mario Bros. levels and evolve a latent vector space to select better input noise for the generator using an evolutionary method (CMA-ES) [27]. Our approach builds on this DCGAN work [27], and introduces a multistep DCGAN approach to generating high-quality levels with enhanced solvability.

B. *Procedural Content Generation in Educational Games*

Applying PCG for level generation within an educational game is challenging because it requires the generated content to not only be creative and solvable, but also to exercise the intended learning objectives. As a result, there has been limited work to date on using PCG in educational games. Most of the prior work has focused on generating content from human-authored content spaces or following a set of human-authored rules [5][15]. Hooshyar et al. introduced a data-driven PCG approach using SVMs to construct a genetic algorithm fitness function for generating content adaptable to individual students in an English-learning educational game [8]. Dong and Barnes proposed a template-based puzzle generator for an educational puzzle programming game designed to teach loops and functions [5]. The generator helps reduce the time required to create new puzzles, while producing more creative content. However, the input template used by the generator must be human authored with respect to the desired learning objectives. Valls-Valgas et al. proposed a PCG approach utilizing a rule-based graph grammar for generating new levels in a parallel programming educational game [29]. The system is designed to work with a player model that targets individual player's needs. It is designed to generate a complete solution based on the desired complexity and concepts, and it then removes some elements based on the desired difficulty level. In contrast to this work, we aim to generate levels for an educational game using a multistep machine-learned model trained on a small corpus of human-authored levels that implicitly incorporate the desired learning objectives and level design characteristics.

III. Game-based Learning Environment

Over the past few years, our lab has been developing ENGAGE (Fig. 1), an educational game for middle school students (ages 11-13) focused on helping them learn computer science concepts and practices [30]. ENGAGE is built with the


If you're receiving this message, it means that you have successfully entered the Engage research station.

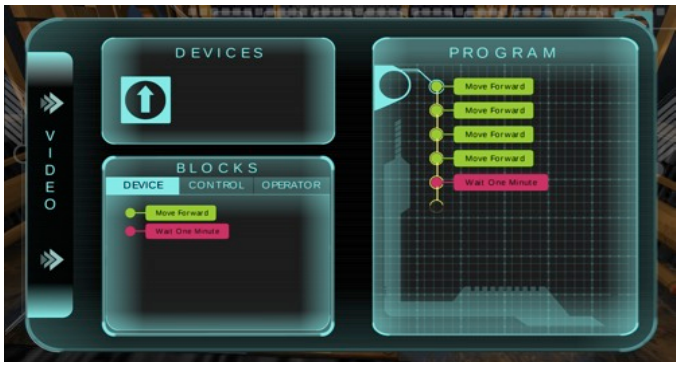


Fig. 1. The ENGAGE game-based learning environment (left) and its block-based programming language interface (right).

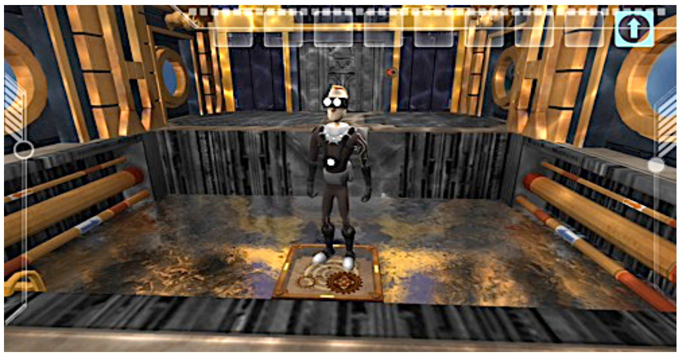
Unity game engine. The curriculum underlying ENGAGE's narrative is derived from the AP Computer Science Principles course developed in the United States by the College Board with support from the National Science Foundation [31].

In ENGAGE, students play the role of the protagonist who has been sent to an undersea research facility to determine why all communication with the station has been lost. Unbeknownst to them, the facility has been taken over by a rogue scientist. As students explore the station, they must progress through multiple areas consisting of a series of connected rooms. Each room presents players with a set of computational challenges they must solve using a block-based programming interface to control devices within the room in order to advance to the next area. The overarching narrative is advanced through cinematics and character dialogue (Fig. 1, Left) while student learning is scaffolded with onscreen hints and short animated vignettes.

ENGAGE's visual programming interface (Fig. 1, Right) is a tool for constructing programs for devices in the game using a block-based programming language similar to Scratch and other visual programming languages [32]. The interface consists of three sub-panels: Devices, Blocks, and Program. The Devices panel displays the currently paired devices, the Blocks panel displays the available programming blocks for the selected device, and the Program panel provides a space for writing a program for the selected device.

To illustrate, in a sample program for a platform device (Fig. 1, Right), the player has dragged four *Move Forward* blocks and a *Wait One Minute* block onto the Program panel. When the program executes, this will instruct the platform to move forward four times and wait until one minute has passed. When constructing the program, if a block is placed near another block, they will snap together, and the circular connectors between the blocks will display a locking animation. These animations help students easily manipulate the blocks and also provide visualizations indicating which blocks are connected together.

The first part of ENGAGE introduces players to the game's controls, block-based programming environment, and initial programming tasks. One room in this part of the game requires students to pair with a platform device to program it to move across a water-filled obstacle (Fig. 2, Left). In this room, students can solve the computational challenge by creating a program that instructs the platform to move forward four times and wait one minute to give the player time to walk off the platform on the other side of the room before the platform resets to its original position. For the work presented in this paper, we focus on generating levels in this type of room, and we vary the gaming skill and computational concepts required by the challenge. We authored a set of 132 training levels by making variations of this original room that require the use of loops and conditions as well as various degrees of navigation and platforming skills. We use a two-dimensional tile-based representation of the room to investigate our proposed framework. The room presented from a top-down view in Fig. 2, Right is the basis for all of the authored training data used in this work. Details of the 2D tile-based representation of the levels are given in Section 4.



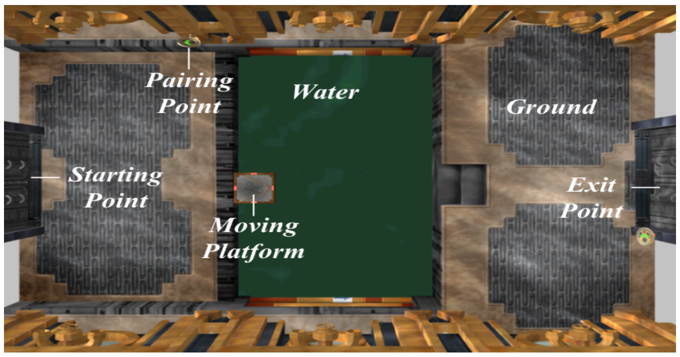


Fig. 2. In game 3D view of platform device room (left) and top-down view of platform device room (right).

IV. Method

In this section, we describe our approach to representing levels and level categories, our proposed multistep DCGAN framework, and metrics to evaluate the solvability and novelty of levels generated by the trained models.

A. Level Representation

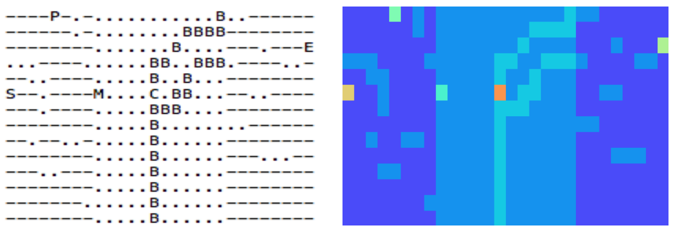
A key preliminary step to train generative models for PCG is devising a method to represent levels in a trainable format. Our approach is inspired by the 2D tile-based representation used for Super Mario Bros. level generation [23][27]. In our work, each level consists of 392 tiles in a 14 by 28 grid, where each tile is characterized by a specific type. We use eight types of tiles (Table I), which were designed to construct platform navigation levels in ENGAGE. Among the eight tile types, four tile types represent significant locations within the level: *Starting Point*, *Exit Point*, *Moving Platform*, and *Pairing Point*. Before programming the moving platform, players must navigate to the pairing point to register the device with their in-game portable computer. Two of the other tile types, *Barrier* and *Conditional Barrier*, encourage the use of loops and conditions in the programs developed to solve the challenge to facilitate students' learning of these concepts. The final two types of tiles are the *Ground* and *Water* tiles, which indicate where the player can walk and where the moving platform can move. With this specification of tile types, we created the 132 training levels as illustrated in Fig. 3, Left. These levels serve as our training data. To accommodate DCGAN model training for these categorical tile types, each tile type is converted using one-hot encoding, which is an encoding process that produces a bit vector whose length is the number of possible tile types (*i.e.*, 8), where only the associated bit is 1 while all other bits are 0.

B. Categories of Levels

Levels are classified into 12 categories based on the combination of three degrees of gaming skills required to solve the level (i.e., High, Medium, Low) and four variations of two learning objectives (*i.e.,* Loop, Conditional), which are key concepts for Algorithms and Programming in the K-12 Computer Science Framework [31]. The four variations consist of (1) *Basic* level, which only require basic programming blocks to operate the moving platform (e.g., *Move Forward* and *Rotate*), (2) *Loop* level, in which students should write a loop statement (e.g., *Repeat*) in addition to basic blocks, (3) *Conditional* level, in which students should write a conditional statement (e.g.,. *If-else*) in addition to basic blocks,

TABLE I. Tile Types in Tile-based Representation of Engage

Tile Type	Symbol	Visualization
Ground	-	🟦
Water	.	🟦
Barrier	B	🟦
Moving Platform	M	🟩
Pairing Point	P	🟩
Exit Point	E	🟩
Starting Point	S	🟨
Conditional Barrier	C	🟧



```
----P-.-...........B..------
------.-........BBBB--------
--------.......B....---.---E
...----......BB..BBB.----..-
--..----.....B..B...--------
S--.----M....C.BB...--..----
---.----.....BBB....--------
--------.....B........------
--.--..-.....B......--------
--------.....B......---...--
---..---.....B......--------
--------.....B......--------
-------......B......--------
--------.....B......--------
```

Fig. 3. Example authored level in text (left) and visual representation (right).

and (4) *Loop & Conditional* level, which requires implementation of both the Loop and Conditional concepts to solve the level.

Gaming skills refer to the degree of navigation and platforming skill needed by the player to solve the level. While some students who are familiar with playing games may feel more engaged and motivated as they encounter levels requiring more complex character controls and navigation, others with less game experience may be more likely to struggle in such a level. This could lead to negative learning experiences for less skilled students. Thus, based on students' pre-survey of their game experience, producing varied gaming skill levels that are adaptive to individual players is an important functionality for PCG to provide effective, personalized game-based learning experiences.

To introduce gaming skill variations in levels in the training data, we adjust tile types in the ground area. The ground area consists of ground and water tiles. Players can freely navigate on the ground tiles, while they need to jump to get across small holes filled with water. This area is distinguished from the larger area of water where the player must use the moving platform to navigate to the other side. *Low* game skill levels do not require the player to jump (i.e., no holes in the ground area), while *Medium* and *High* game skill levels contain progressively more holes requiring more sophisticated character controlling to successfully navigate.

To encourage the use of certain programming constructs in solving a level, we configure tiles in the area where the player must use the moving platform to navigate. Within the game, the learning objective focus of a level for a specific student is determined by the student's competency that can be measured by knowledge assessment models (*e.g.*, [30]). For example, while some students might struggle with the conditional concept (e.g., *If-else*), others may have mastered the conditional concept but struggle with the iteration concept (e.g., *Repeat*), as observed in their programming pattern of using only basic programming blocks only even when an iterative pattern could be used. It is important for PCG to provide content tailored to individual students' competency level for each programming concept thereby helping remediate their knowledge.

To introduce training data that included each of the programming concepts, we authored levels that included additional obstacles the moving platform had to navigate around within the large water area (shown in blue in the middle of Fig. 3, Right). The water area consists of Water tiles (.), the

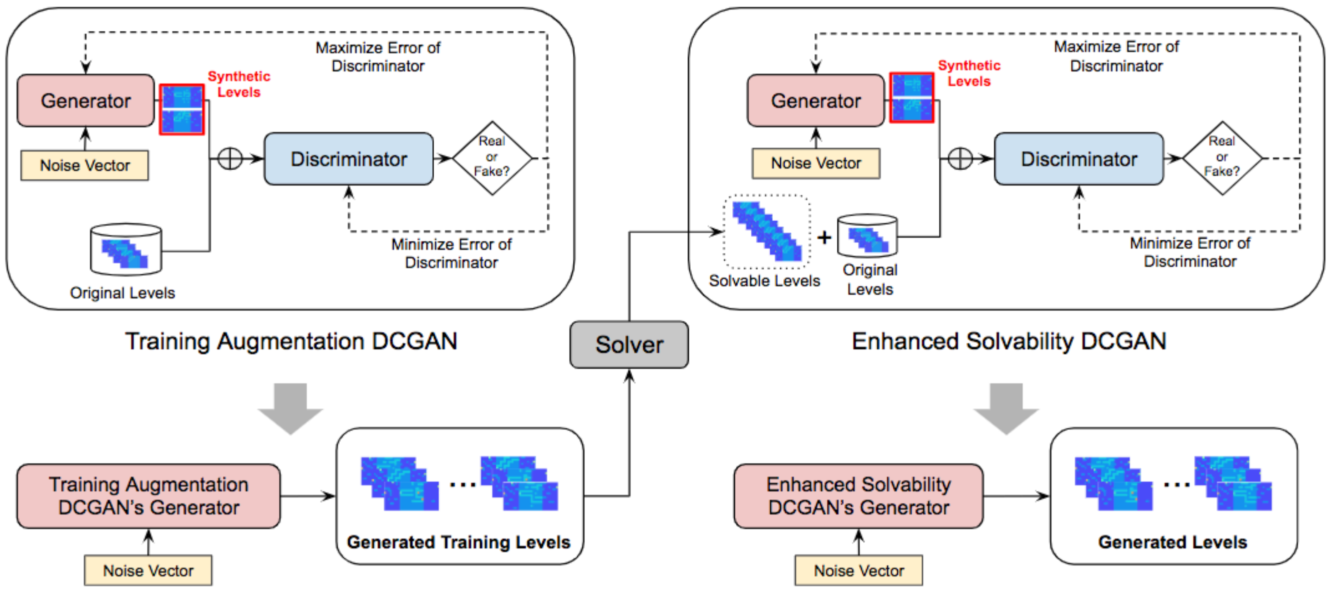

Maximize Error of Discriminator
Generator
Synthetic Levels
Noise Vector
Discriminator
Real or Fake?
Original Levels
Minimize Error of Discriminator
Training Augmentation DCGAN
Solver
Solvable Levels
Enhanced Solvability DCGAN
Training Augmentation DCGAN's Generator
Generated Training Levels
Enhanced Solvability DCGAN's Generator
Generated Levels

Fig. 4. Overview of the proposed Multistep DCGAN Framework

Moving Platform (M), Barrier tiles (B), and Conditional Barrier tiles (C). We introduce iteration and conditional concepts by using variations of the Barrier (B) tile, where the moving platform (M) cannot navigate, and the Conditional Barrier (C), which encourages students to master the conditional concept, by forcing them to check if the Conditional Barrier is active prior to moving to the corresponding location. Fig. 3 shows an example of a level containing both iteration and conditional concepts. After going through the Conditional Barrier (C) (shown in orange), students encountered a repetitive pattern that encourages the use of iteration.

C. *Multistep Deep Convolutional Generative Adversarial Network*

The PCG models in this work are based on generative adversarial networks (GANs) [25]. A GAN is a type of zero-sum game in which a *generator* and a *discriminator* are trained to outdo the other. For each training phase, the *generator* takes a random noise vector as input and generates synthetic data that looks similar to real data with the goal of deceiving the discriminator about the source of the data. The *discriminator* takes as input either real data or fake data and determines if the given input is real (i.e., the source of the data is from the real dataset) or fake (i.e., the source of the data is from the generator). The goal of training a GAN is to improve the generator's performance to the level that even a well-trained discriminator cannot distinguish the generator-created synthetic data from real data.

DCGANs (deep convolutional generative adversarial networks) are a variant of GANs [16]. DCGANs have been extensively used in the generation of synthetic images. DCGANs use (1) deep convolutional neural networks with feature transposed convolutions for the generator, where the generator takes a noise vector as input and converts it into synthetic data, and (2) deep convolutional neural networks with normal convolutions for the discriminator, where it takes an image as an input and predicts whether the input is real or fake.

In our work, a *multistep* DCGAN approach is utilized for level generation. We train two DCGANs: The Training Augmentation DCGAN performs a data augmentation function (i.e., creating more training data via the generator) and the Enhanced Solvability DCGAN supports generation of levels with higher solvability compared to ones generated by the Training Augmentation DCGAN. Our multistep DCGAN model is adapted from [27], the generators use ReLU activation for all transposed convolutional layers and the discriminators use Leaky ReLU activation for all convolutional layers. Both the generator and discriminators are trained with *RMSprop* [33] for 1,000 iterations with a batch size of 32 and a learning rate 0.00005. The dimension of noise vectors used for the generators is set to 32, while each feature in the noise vectors are initialized following a normal distribution range from 0 to 1.

Fig. 4 presents the multistep DCGAN-based PCG framework. The Training Augmentation DCGAN is used to generate levels for augmenting the training dataset using a limited set of hand-crafted training data. The augmentation process is performed as follows: 1) we train the generator through the Training Augmentation DCGAN model, and save the model every 50 epochs between 50 epochs and 1,000 epochs; 2) each candidate model generates 1,000 levels whose novelty and the solvability scores are computed to choose the best performing model 3) each of the generated levels from the chosen model is examined with a *Solver*, described below, which checks whether each level is solvable or not; and 4) only solvable levels are combined with the original training levels as a process of dataset augmentation.

Then, we train another generator (i.e., Enhanced Solvability DCGAN) using the augmented dataset. Although only a small portion of the training data is human-authored data and the remaining data was created by the Training Augmentation DCGAN, it should be noted that all levels in this augmented training set are solvable. Our multistep DCGAN-based PCG approach enables the Enhanced Solvability DCGAN's generator to produce more solvable levels than the Training Augmentation DCGAN's generator that is more likely to generate levels that are unsolvable. The multistep DCGAN-based PCG framework was implemented with the PyTorch deep learning toolkit [34], building upon the DCGAN-based PCG library written by Volz and colleagues [27].

D. Evaluation Metrics

The performance of the two DCGANs' trained generators are evaluated with respect to their ability to generate levels that are both solvable and novel. We adopt these two metrics, since generating valid, solvable levels is one of the primary goals for PCG and also generating novel levels will help promote user engagement. First, we developed a *Solver* module, which determines if a given level is solvable. The solver checks the following five constraints: 1) the starting point, pairing point, moving platform, and exit point should be unique within the level, 2) there should be a path from the starting point to the pairing point, 3) there should be a path from the pairing device to the moving platform, 4) there should be a path for the moving platform to follow to reach the other side of the level, 5) there should be a path from the moving platform to the exit point of the level. The solver is implemented with *Dijkstra's* algorithm [35] to find the shortest path between two tiles, checking whether the generated levels satisfy the five described constraints.

To measure the novelty of the generated levels, we used the novelty measure from novelty search [36]. The novelty ρ for level x is given by:

$$\rho(x) = \frac{1}{k}\sum_{i=1}^{k} dist(x, \mu_i) \qquad (1)$$

where μ_i is the ith nearest neighbor drawn from the real dataset of level x with respect to the domain-specific distance metric *dist*. The *dist* measure evaluates how novel a generated level is, compared to one of the original levels. We set the number of nearest neighbors to consider (k) to 20 and adopt the heuristic function introduced in Liapis et al. [37], in which the distance between two tile-based levels is defined as the number of mismatched tiles at the same coordinates between the generated level and the original level over the total number of tiles in the original level.

$$dist(x, y) = \frac{\sum_{i=1}^{r}\sum_{j=1}^{c} f(x_{i,j}, y_{i,j})}{r * c} \qquad (2)$$

The function f is defined as follows: $f(x_{i,j}\,,\ y_{i,j}) = 0\ (if\ x_{i,j} = y_{i,j})\,, 1\ (if\ x_{i,j} \neq y_{i,j})$, where i and j are the row and column indices in the level, respectively. Finally, r and c denote the total number of rows and the total number of columns in the level, respectively.

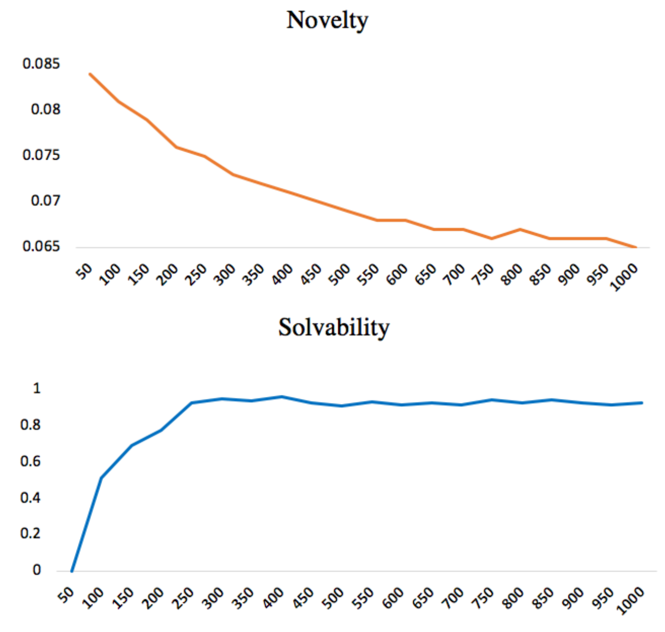


Fig. 5. Change in *Novelty* and *Solvability* by the number of epochs

V. Evaluation

We evaluate the multistep DCGAN PCG framework on the novelty and solvability of the levels it generates. The top graph in Fig. 5 shows changes in the novelty score of the Training Augmentation DCGAN's generator, as the number of epochs increases. Generated samples that are more distinct from real data have higher novelty scores. As shown in Fig. 5, Top, the novelty scores trend downwards and starts decreasing slowly after 450 epochs, indicating that the Training Augmentation DCGAN's generator starts generating samples very similar to the original dataset. The bottom graph in Fig. 5 shows changes in the percentage of solvable levels out of 1,000 generated levels produced by the trained Training Augmentation DCGAN's generator as the number of epochs increases. The solvability rises steeply until 250 epochs and converges.

We test three Training Augmentation models that are saved at 250, 350, and 450 epochs for the Training Augmentation DCGAN to augment the training dataset, with which the Enhanced Solvability DCGAN model is trained. The rationale behind this decision is the Training Augmentation DCGAN models are stable enough to generate solvable levels after 250 epochs, while the novelty score converges after 450 epochs (i.e., models at 450 epochs and any models using a higher number of epochs are not differ significantly with respect to the novelty).

The number of epochs for the Enhanced Solvability DCGAN model matches with the number of epochs used to train each Training Augmentation DCGAN model. After training the corresponding Enhanced Solvability DCGAN, we compare the enhancement in a pairwise manner (e.g., comparing novelty and solvability of the Training Augmentation DCGAN trained using 250 epochs and the Enhanced Solvability DCGAN trained using 250 epochs). Table II shows the comparison of the performance between the Training Augmentation DCGAN's

TABLE II. COMPARISION OF NOVELTY AND SOLVABILITY.

# of Epochs	Novelty (ρ)		Solvability	
	Training Augmentation	*Enhanced Solvability*	*Training Augmentation*	*Enhanced Solvability*
250	**0.075**	0.069	0.930	**0.999**
350	0.072	0.063	0.946	0.973
450	0.071	0.620	0.919	0.986

generator and the Enhanced Solvability DCGAN's generator based on 5,000 levels created by each model across the different number of epochs. The results show a significant enhancement is achieved by the Enhanced Solvability DCGAN models with respect to the solvability, while sacrificing only a small degree of novelty. In particular, the Enhanced Solvability DCGAN model trained using 250 epochs generated only 7 unsolvable levels out of 5,000 generated levels (a solvability of 99.86%).

VI. DISCUSSION AND LIMITATIONS

The results show that the multistep DCGAN-based PCG framework enables the Enhanced Solvability DCGAN's generator to create a high percentage of solvable levels to significantly outperform the Training Augmentation DCGAN's generator with respect to solvability. This is achieved by having the multistep DCGAN model use the *Dijkstra's* algorithm-based solver to select synthetic solvable levels. The Training Augmentation DCGAN's generator generates synthetic levels that are a mixture of solvable and unsolvable levels, the solver checks if each generated level is solvable based on the five constraints, only solvable synthetic levels are integrated into the original dataset (i.e., dataset augmentation), and the Enhanced Solvability DCGAN's generator generates more solvable synthetic levels without significantly sacrificing novelty.

Examples of solvable (Fig. 6b) and unsolvable (Fig. 6c) levels were generated by the Enhanced Solvability DCGAN model using 250 epochs. The water area, the middle blue-colored regions of all levels are complex and creative compared to the original levels shown in Fig. 6a. However, the generated levels also contain undesirable elements. The level in Fig. 6b, Left has many barriers but only require the platform to move forward across the water area. In Fig. 6c, Right, the left-side ground area (shown in purple) requires challenging moves with holes while the right-side ground does not require any challenging moves.

As discussed, we utilize solvability and novelty to evaluate the generated levels. Each of the measures has limitations. First, the novelty heuristic function (Eq. 1) performs a tile-level comparison between a generated level and *k*-nearest neighbors in the original dataset. Although this heuristic function measures the difference between two levels, similar patterns observed in slightly different areas within the two levels might not be fully captured by this metric. Second, the solvability function uses an algorithm to find the shortest path between two tile positions. However, other levels designed in ENGAGE or more complex levels in other games will likely require different approaches for measuring solvability. Lastly, this work examines one data augmentation step, which achieves enhanced

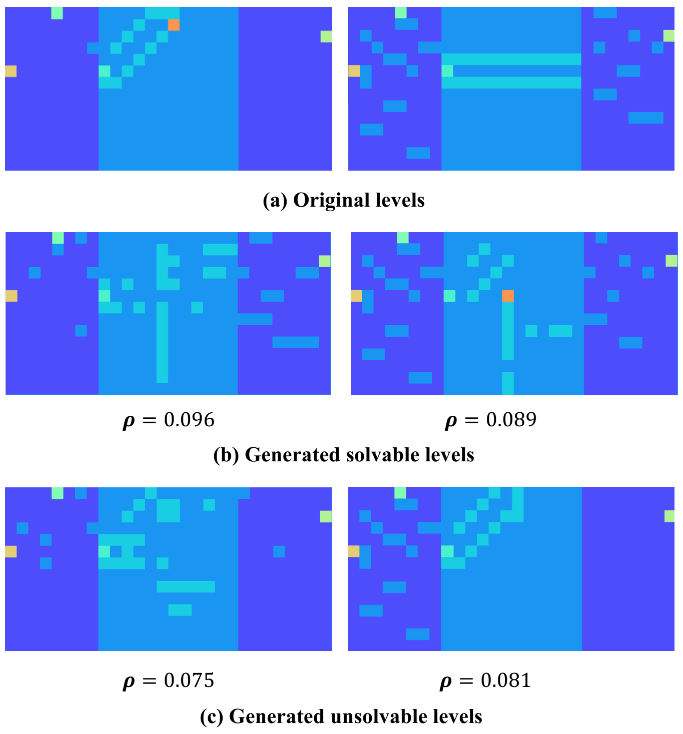


Fig. 6. Examples of original and generated levels. (a) Original levels (b) Generated solvable levels from the Epoch 250 model with *Novelty* (ρ) > 0.08. (c) Unsolvable generated levels from the Epoch 250 model with *Novelty* (ρ) > 0.07 (Left) The moving platform can only move one step forward and cannot escape from the barrier. (Right) There is no way to go forward after the moving platform navigates the iterative path.

solvability, while more augmentation steps have potential to further enhance the performance. While these limitations call for further research to design more generalizable functions to measure solvability and novelty of the levels, the multistep DCGAN PCG framework paves the way toward approaches that significantly improve solvability relative to a single-step DCGAN.

VII. CONCLUSION

Because procedural content generation has proven effective for entertainment games, it holds considerable potential for dramatically increasing adaptivity and replayability for educational games, while simultaneously significantly reducing development effort. The multistep DCGAN model presented here shows promise for achieving the goal of generating solvable levels by augmenting human-authored training data with synthetic training data. Evaluation of the multistep DCGAN model shows that it significantly enhances the solvability of the generated levels with minor degradation in the novelty of the generated levels. In this work, we investigated a multistep DCGAN model, and the model is extendible to additional steps. These results suggest promising directions for future work, including exploring if introducing additional steps yield further improvements to the generated levels. It will also be important to investigate the impact of incorporating additional constraint checkers into the solver to explore how the generated levels address specific learning objectives. One approach is to use conditional GANs [38] to adaptively create levels requiring a specific set of gaming skills and learning objectives, as conditioned on the input of the generator. Using this conditional GAN approach, the set of conditions can be extended to cover other objectives of interest in educational games, in addition to the gaming skills and learning objectives. Another direction for future work is to explore the benefits of incorporating a self-attention GAN model [39] that has the ability to better track patterns with long-range dependencies.

Acknowledgment

This research was supported by the National Science Foundation under Grants CNS-1138497 and DRL-1640141. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] D. B. Clark, E. E. Tanner-Smith, and S. S. Killingsworth, “Digital games, design, and learning: A systematic review and meta-analysis,” *Rev. Educ. Res.*, vol. 86, no. 1, pp. 79–122, 2016.

[2] H. A. Spires, J. P. Rowe, B. W. Mott, and J. C. Lester, “Problem solving and game-based learning: Effects of middle grade students’ hypothesis testing strategies on learning outcomes,” *J. Educ. Comput. Res.*, vol. 44, no. 4, pp. 453–472, 2011.

[3] J. P. Gee, *What video games have to teach us about learning and literacy*. New York, NY: Palgrave Macmillan, 2003.

[4] D. W. Shaffer, *How computer games help children learn*. New York, NY: Palgrave Macmillan, 2006.

[5] Y. Dong and T. Barnes, “Evaluation of a template-based puzzle generator for an educational programming game,” in *Proc. 12th International Conference on the Foundations of Digital Games*, 2017, p. 40

[6] J. P. Rowe, L. R. Shores, B. W. Mott, and J. C. Lester, “Integrating learning and engagement in narrative-centered learning environments,” *Int. J. Artif. Intell. Educ.*, vol. 21, pp. 115–133, 2011.

[7] K. VanLehn, “The behavior of tutoring systems,” *Int. J. Artifical Intell. Educ.*, vol. 16, no. 3, pp. 227–265, 2006.

[8] D. Hooshyar, M. Yousefi, M. Wang, and H. Lim, “A data-driven procedural-content-generation approach for educational games,” *J. Comput. Assist. Learn.*, vol. 34, no. 1, 2018.

[9] K. Kiili, “Digital game-based learning: Towards an experiential gaming model,” *Internet High. Educ.*, vol. 8, no. 1, pp. 13–24, 2005.

[10] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, “What is procedural content generation? Mario on the borderline,” in *Proc. 2nd International Workshop on Procedural Content Generation in Games. ACM*, 2011, p. 3.

[11] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, “Search-based procedural content generation : A taxonomy and survey,” *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 172–186, 2011.

[12] A. M. Smith and M. Mateas, “Answer set programming for procedural content generation: A design space approach,” *IEEE Trans. Comput. Intell. AI Games*, vol. 3, no. 3, pp. 187–200, 2011.

[13] A. Summerville *et al.*, “Procedural content generation via machine learning (PCGML),” *IEEE Trans. Games*, vol. 10, no. 3, pp. 257–270, 2018

[14] P. Kantharaju, K. Alderfer, J. Zhu, B. Char, B. Smith, and S. Onta, “Tracing player knowledge in a parallel programming educational game,” in *Proc. 14th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment,* 2018, pp. 173–179.

[15] L. Rodrigues, R. P. Bonidia, and J. D. Brancher, “A math educational computer game using procedural content generation,” in *Brazilian Symposium on Computers in Education*, 2017, pp. 756–765.

[16] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” in *arXiv preprint arXiv:1511.06434*, 2015.

[17] N. Shaker, J. Togelius, and M. J. Nelson, *Procedural content generation in games*. Springer, 2016.

[18] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis, “Multiobjective exploration of the StarCraft map space,” in *Proc. 2010 IEEE Conference on Computational Intelligence and Games*, 2010, pp. 265–272.

[19] M. Guzdial, J. Reno, J. Chen, G. Smith, and M. Riedl, “Explainable PCGML via game design patterns,” in *arXiv:1809.09419v1*, 2018.

[20] S. Dahlskog, J. Togelius, and M. J. Nelson, “Linear levels through n-grams,” in *Proc. 18th International Academic MindTrek Conference: Media Business, Management, Content & Services*, 2014, pp. 200–206.

[21] A. Summerville, S. Philip, and M. Mateas, “MCMCTS PCG 4 SMB: Monte carlo tree search to guide platformer level generation,” in *Proc. 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2015, pp. 68–74.

[22] S. Snodgrass and S. Ontañón, “Experiments in map generation using markov chains,” in *Proc. 9th International Conference on Foundations of Digital Games*, 2014.

[23] S. Snodgrass and S. Ontañón, “Learning to generate video game maps using markov models,” *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 4, pp. 410–422, 2016.

[24] A. Summerville and M. Mateas, “Super mario as a string: Platformer level generation via LSTMs,” in *arXiv:1603.00930v2*, 2016.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” in *Advances in Neural Information Processing Systems, 2014, pp. 2672-2680*

[26] E. Giacomello, P. L. Lanzi, and D. Loiacono, “DOOM level generation using generative adversarial networks,” in 2018 *IEEE Games, Entertainment, Media Conference*, 2018, pp. 316–323.

[27] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, “Evolving mario levels in the latent space of a deep convolutional generative adversarial network,” in *Proc. Genetic and Evolutionary Computation Conference*, 2018, pp. 221–228.

[28] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *Proc. International Conference on Machine Learning*, 2017, pp. 214–223.

[29] J. Valls-Vargas, J. Zhu, and S. Ontañón, “Graph grammar-based controllable generation of puzzles for a learning game about parallel programming,” in *Proc. 12th International Conference on the Foundations of Digital Games*, 2017, p. 7.

[30] W. Min, M. H. Frankosky, B. W. Mott, E. N. Wiebe, K. E. Boyer, and J. C. Lester, “Inducing stealth assessors from game interaction data,” in *Proc. 9th International Conference on Artificial Intelligence in Education*, 2017, pp. 212–223.

[31] CollegeBoard, *AP Computer Science Principles*. New York, NY: CollegeBoard, 2017.

[32] Y. K. Mitchel Resnick et al., “Scratch: Programing for all,” *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[33] T. Tieleman and G. Hinton, Lecture 6.5-RmsProp in COURSERA: "Neural network for machine learning," unpublished.

[34] A. Paszke *et al.*, “Automatic differentiation in PyTorch,” in *Neural Information Processing Systems Autordiff Workshop*, 2017.

[35] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.

[36] J. Lehman and K. O. Stanley, “Revising the evolutionary computation abstraction: Minimal criteria novelty search,” in *Proc. 12th Annual Conference on Genetic and Evolutionary Computation*, 2010, pp. 103–110.

[37] A. Liapis, G. N. Yannakakis, J. Togelius, “Enhancements to constrained novelty search: two-population novelty search for generating game content,” in *Proc. 15th Annual Conference on Genetic and Evolutionary Computation*, 2013, pp. 343–350.

[38] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” in *arXiv:1411.1784v1*, 2014.

[39] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” in *Proc. 36th International Conference on Machine Learning, 2019,* pp. 7354–7363.