A Generation System of 2D Video Games Levels Using TOAD-GAN

Jason Ravagli
jason.ravagli@stud.unifi.it

Università degli Studi di Firenze
Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione
Course of Human Computer Interaction

9 April 2021

Outline

Procedural Content Generation

- Creating video games contents (characters, levels, quests, etc.) requires **time and efforts**
- For small teams or huge projects it is not feasible to manually create all the contents
- Sometimes, all the game contents cannot be stored on the installation support of the game (more frequent for old games)
- **Procedural Content Generation (PCG)** is the process of automatically generate game content using algorithms

PCG-ML

- The main limit of PCG is that generation algorithms are too game-specific
- Procedural Content Generation via Machine Learning **(PCG-ML)** uses Machine Learning generative models to create game contents
- The flexibility of machine learning methods allows you not to bind the derived PCG techniques to specific games
- Recent PCG-ML techniques exploits **GANs** to automatically identify and extract patterns from existing contents and generate new ones

PCG-ML

- **Problem:** GANs and deep learning models need many examples to correctly extract and generalize patterns
- **TOAD-GAN**[1] is a deep learning architecture for the generation of 2D tile-based video games levels
- It is inspired by **SinGAN**[2] and can be trained on a single example level

[1] Frederik Schubert Maren Awiszus and Bodo Rosenhahn. *TOAD-GAN: Coherent Style Level Generation from a Single Example*. 2020.
[2] Tomer Michaeli Tamar Rott Shaham Tali Dekel. *SinGAN: Learning a Generative Model From a Single Natural Image*. 2019.

Project Goal

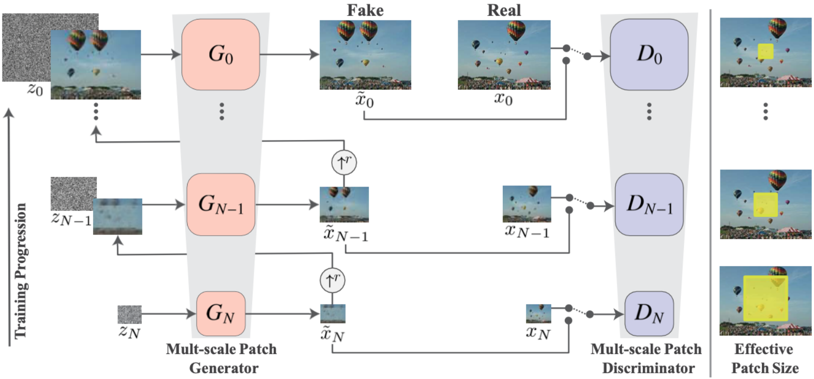
We implemented a **game-independent tool** to automatically generate 2D tile-based video games levels by:

- Investigating the performance of TOAD-GAN applied to different types of levels
- Finding a setting for TOAD-GAN suitable for most tranining levels
- Providing to the user a system to design, train TOAD-GANs and generate new levels with little to no knowledge about its implementation details

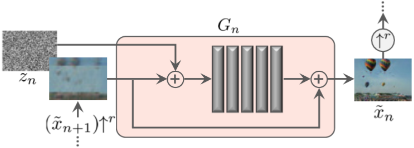
SinGAN

- Deep learning architecture that allows training a generative model of images from a **single example image**
- It uses a cascade of **WGAN-GPs**, each of which is associated with a scaled version of the example image
- Each generator is responsible for mapping a 2D gaussian noise tensor to an image having patches from the same distribution of the assigned scaled image

SinGAN - Architecture



SinGAN - Single Scale Generator



- **5 convolutional blocks** formed by convolutional layer (3x3 filters), batch normalization layer and LeakyReLU (the same as the critic)

SinGAN - Training

- SinGAN is trained sequentially, from the lowest GAN to the highest one in the hierarchy
- **Critic loss**: the same used in WGAN-GPs
- **Generator loss**:

$$L_{adv} + \alpha * L_{rec}$$

  - L_{adv}: adversarial loss (WGAN-GP loss)
  - L_{rec}: reconstruction loss
  - α: constant weight factor (tipically 0.1)

SinGAN - Reconstruction loss

- Constrains the latent space to contain a specific set of noise tensors (**reconstruction noise** tensors) that generates the original training image
- A reconstruction noise tensor for each of the N GANs in the hierarchy, fixed before training:
  - 0 at scale $n < N$ (all but the lowest)
  - z^* at scale N (the lowest)
- Sum of squared differences between reconstructed image and original image
- The RMSE between reconstructed and original images at scale n multiplied by a γ factor also determines the **noise standard deviation** σ_n

TOAD-GAN

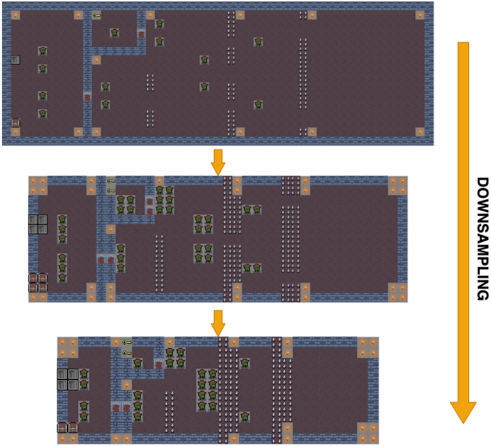
- SinGAN was devised for natural images
- **TOAD-GAN** (Token-based Oneshot Arbitrary Dimension GAN) is an adaptation of SinGAN to the generation of 2D tile-based video game levels
- Each tile in a level is considered as a one-hot encoded vector (a token) and corresponds to a pixel in an image

TOAD-GAN - Downsampling

- **Problem**: applying simple downsampling to calculate the scaled versions of the training image would result in a loss of information
- **Aliasing** would make important tokens disappear
- The authors proposed a **downsamplig procedure** that considers the importance of tokens
- Tokens are organized in a **hierarchy** built a priori with the value of importance of each token
- During downsampling, more important tokens (lower in the hierarchy) will be retained at the expense of less important ones

TOAD-GAN - Downsampling

Value	Tokens
0	
1	
2	
3	
4	



Implemented System

The system is composed of 2 main parts:

- A **training environment** for TOAD-GANs (runnable from a CLI)
- A **GUI application** for manually designing and automatically generating levels

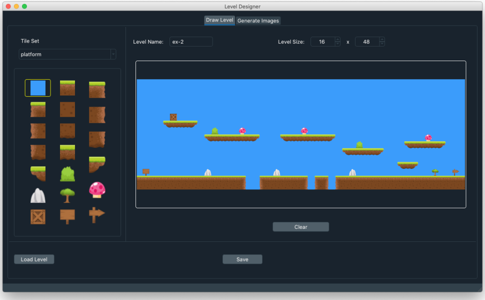
Implemented System - Training Environment

- We implemented a Tensorflow environment to train a TOAD-GAN starting from the original PyTorch code
- We defined the **TOAD-GAN project** file format:
  - It stores and organizes the information to reload trained TOAD-GANs
  - It contains additional information about the training
- We defined a game-independent file format for the **tokens hierarchy**:
  - Hierarchies define also the tile sets used to design levels
  - Users can easily define their own tile sets

Implemented System - Training Environment

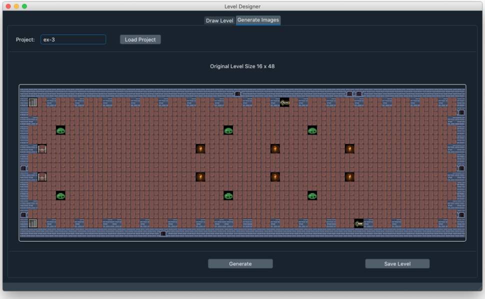
- Training can be run through a script from the command line
- Almost **no settings are required**: the provided default values are good for different types of training examples (see next slides)
  - For advance users, default settings can be overwritten through a *config.yaml* file
- Our training process is about **30% faster** the the original one

Implemented System - GUI Application



- Tile sets can be chosen from the drop-down menu
- Change the level size
- Select the tile, click and drag the mouse over the level area to draw
- Levels can be saved and reloaded for further editing

Implemented System - GUI Application



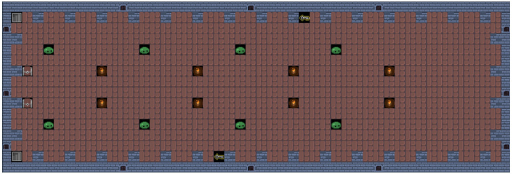
- TOAD-GAN projects can be loaded to generate new levels with trained networks
- Generated levels can be saved and reloaded in the design screen for further editing

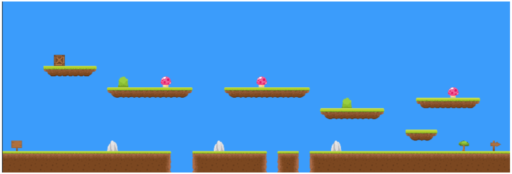
Experiments

- The original paper presented TOAD-GAN applied to Super Mario Bros. levels
- We want to test TOAD-GAN to different types of levels (both in size and style)
  - To investigate its limits
  - To provide to users a good default setup for the hyperparameters

Test Levels

- We used test levels created using **two different tilesets**

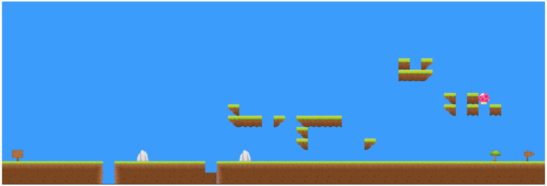




Exploratory Tests

Main features of TOAD-GAN:

- The network can identify and **replicate simple patterns at the edges**
- More complex patterns can be learned if they are sufficiently repeated along an axis
- It **struggles** to learn patterns **in central areas**

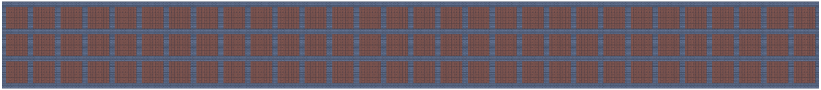


Hyperparameters Tuning

We considered **5 hyperparameters**:

1. Number of training epochs
2. γ factor for the calculation of the noise standard deviation
3. Number of scales in the hierarchy
4. Patch evaluation method used by the critic
5. Number of convolutional blocks in generators and critics

Hyperparameters Tuning - Evaluation



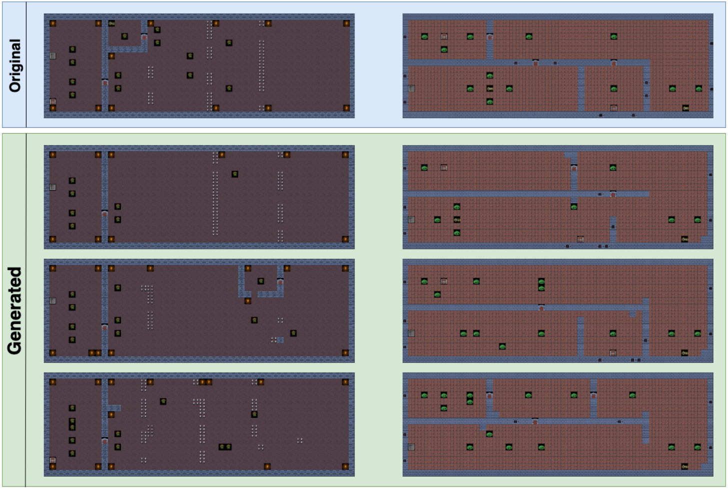
We defined a **test level** and **5 qualitative criteria** for its evaluation:

- Edge patterns replication
- Horizontal patterns replication
- Vertical patterns replication
- Square patterns replication
- Level cleanliness

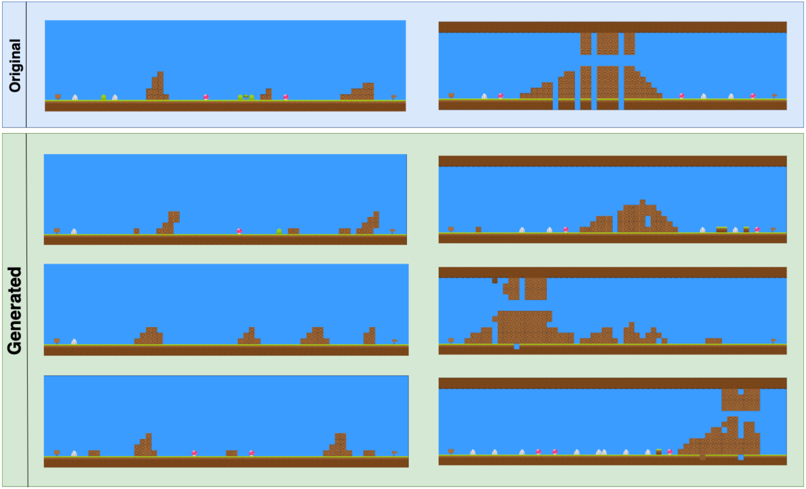
Hyperparameters Tuning - Results

Hyperparameter	Default	Optimal
Epochs	4000	8000
γ	0.1	0.05
N Scales	4	$3 \sim 4$
Critic Evaluation	mean reduction	mean reduction
N Conv. Blocks	3	5

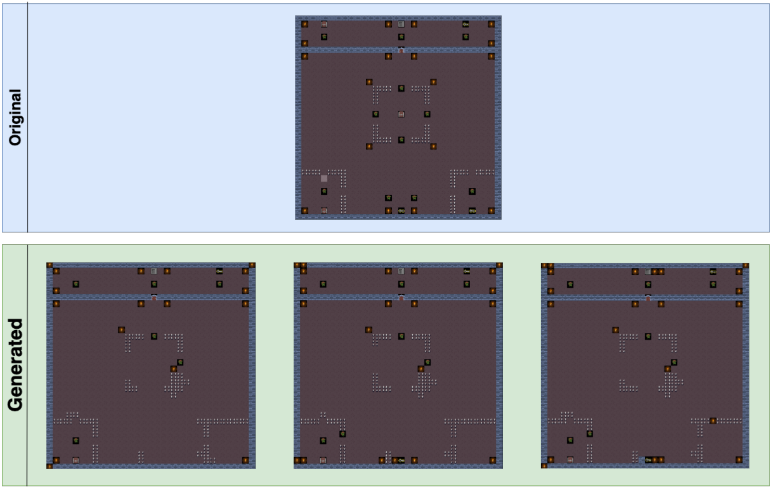
Results - 1



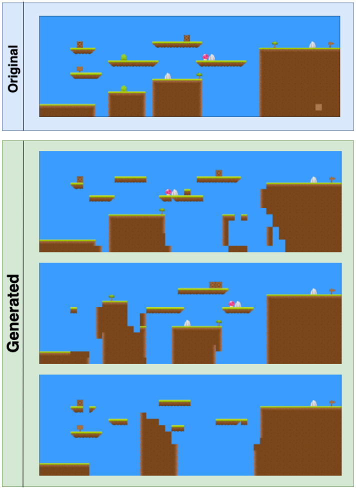
Results - 2



Results - Mode Collapse



Results - When Things Go Wrong



Conclusions and Future Works

- The implemented system is a useful and convenient tool to design 2D game levels through PCG
- TOAD-GAN is a promising tool in the PCG-ML field, but it is fragile and presents some limitations
- Functional requirements are not considered
- We could extend the TOAD-GAN idea to use multiple training examples

Thank you for your attention