MAZE GENERATION TECHNIQUES

Author: Petercă Adrian

Coordinator: Lect. Dr. Eugen Croitoru

STATE OF THE ART METHODS

Random methods

- Aldous-Broder Algorithm
- Kruskal Randomized Algorithm
- Hunt and Kill algorithm

Division methods

- Recursive Division

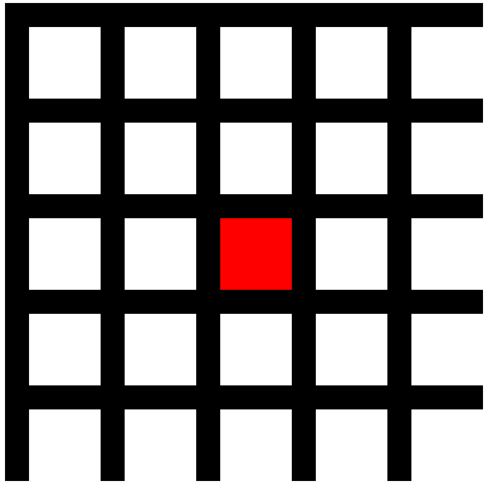
Cell traversal methods

- Depth First Search
- Breath First Search

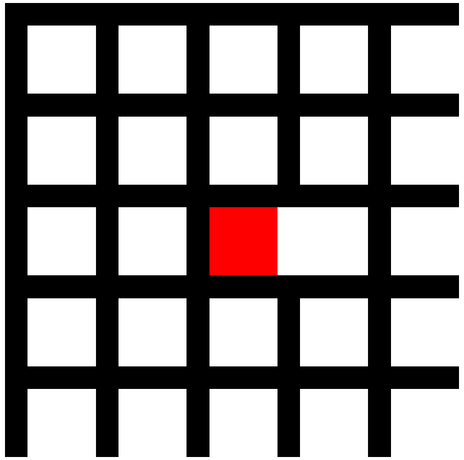
Cellular Automata approaches

- Binary Tree
- Eller’s Algorithm
- Sidewinder Algorithm

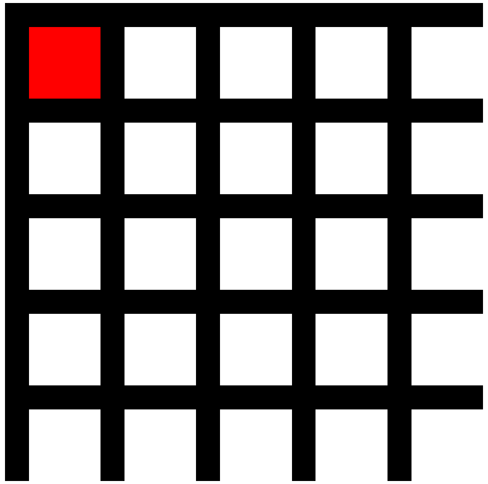
STATE OF THE ART METHODS



Aldous-Broder Algorithm

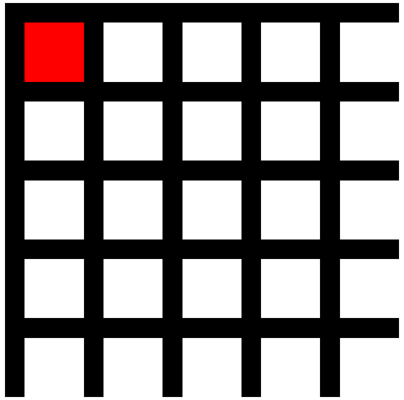


Kruskal Randomized Algorithm

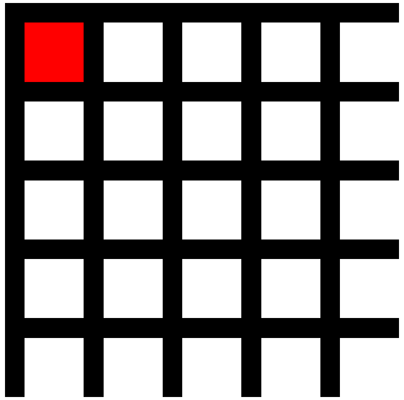


Hunt and Kill algorithm

STATE OF THE ART METHODS



Binary Tree



Depth First Search

CHALLENGES

How is a maze defined?

- Structure
- Complexity
- Connectivity (very important)
- Solvability

What is the smallest part of a maze? What are its building blocks?

Where does it start/end?

- Single start/end point
- Multiple start/end points

What makes it “hard”?

- How quick can it be solved?
- How many start/end points does it have?
- How many start-end paths between the same start/end points does it have?
- Density of walkable areas over number of walls?

NEW APPROACH: GANS

Multiple tested models, various results to compare

Already used for PCG (Procedural Content Generation)

Can achieve good results even with a smaller size

GANS: CHALLENGES

How to define the discriminator? (again, what defines a maze?)

- Let the GAN decide
- Provide a standard way (an agent perhaps) to determine solvability

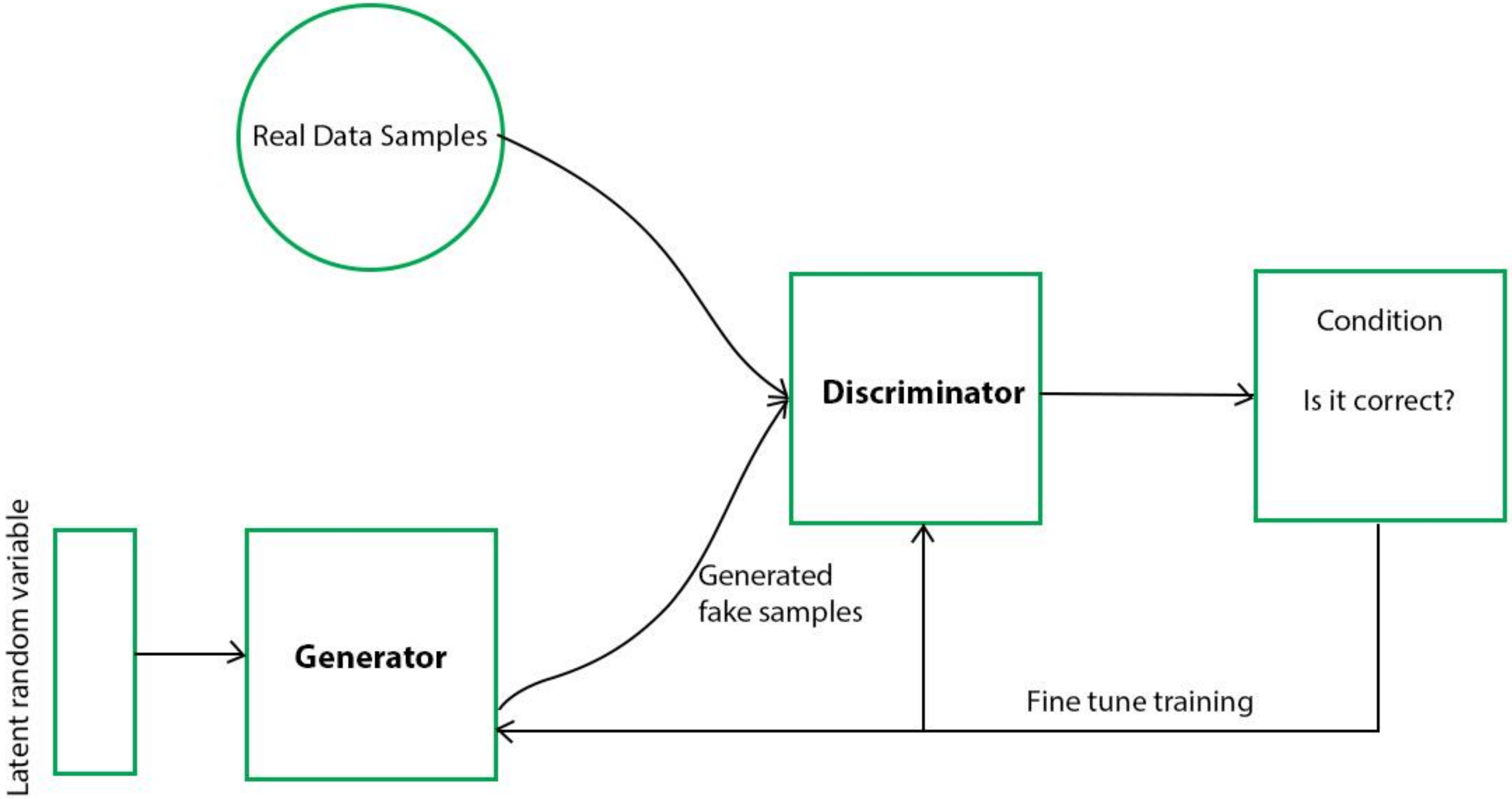
How to prevent mode collapse?

- Bigger dataset?
- Longer training time?

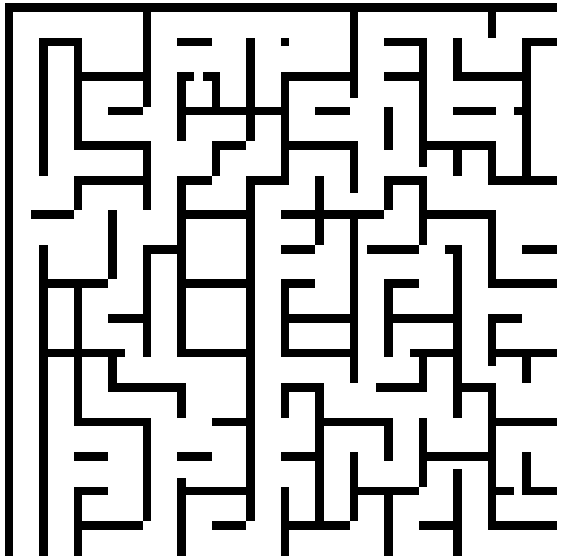
How to prevent overtraining?

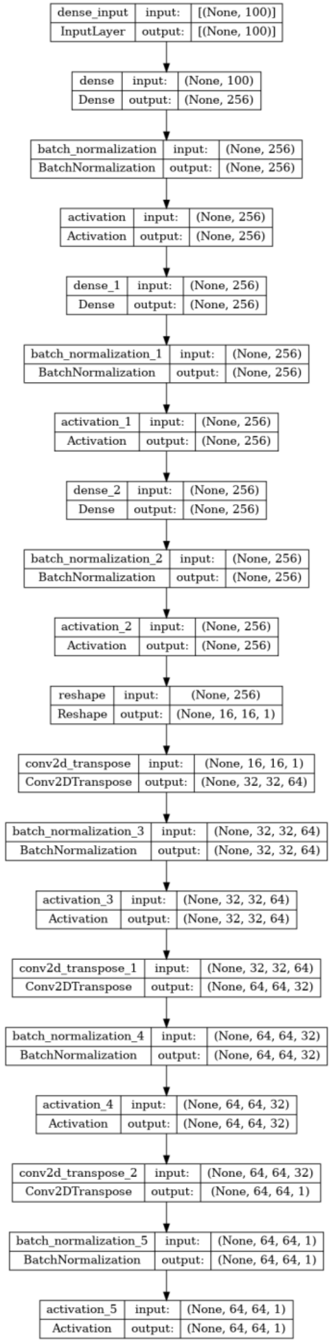
Are bigger models suitable?

NEW APPROACH: GANS



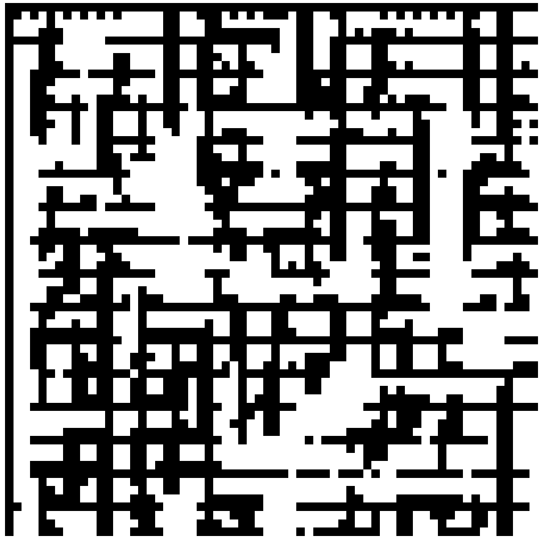
ARCHITECTURE – FIRST MODEL

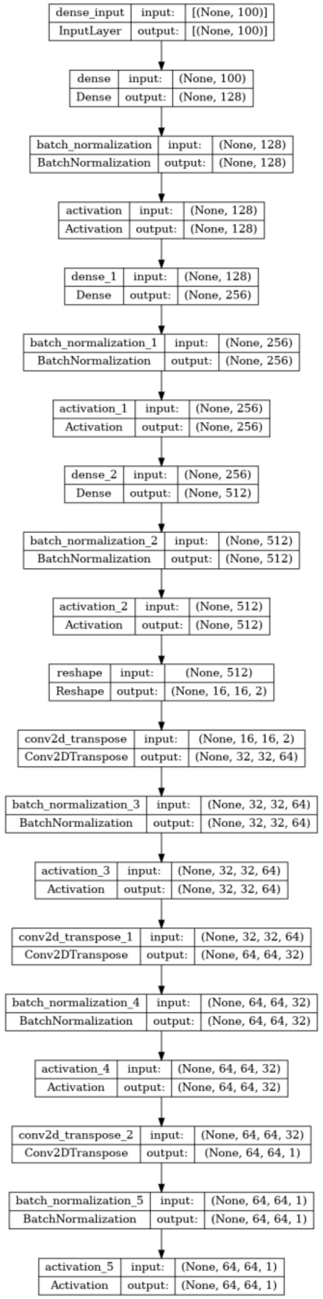




dense_input	InputLayer	input: [(None, 100)]	output: [(None, 100)]
dense	Dense	input: (None, 100)	output: (None, 256)
batch_normalization	BatchNormalization	input: (None, 256)	output: (None, 256)
activation	Activation	input: (None, 256)	output: (None, 256)
dense_1	Dense	input: (None, 256)	output: (None, 256)
batch_normalization_1	BatchNormalization	input: (None, 256)	output: (None, 256)
activation_1	Activation	input: (None, 256)	output: (None, 256)
dense_2	Dense	input: (None, 256)	output: (None, 256)
batch_normalization_2	BatchNormalization	input: (None, 256)	output: (None, 256)
activation_2	Activation	input: (None, 256)	output: (None, 256)
reshape	Reshape	input: (None, 256)	output: (None, 16, 16, 1)
conv2d_transpose	Conv2DTranspose	input: (None, 16, 16, 1)	output: (None, 32, 32, 64)
batch_normalization_3	BatchNormalization	input: (None, 32, 32, 64)	output: (None, 32, 32, 64)
activation_3	Activation	input: (None, 32, 32, 64)	output: (None, 32, 32, 64)
conv2d_transpose_1	Conv2DTranspose	input: (None, 32, 32, 64)	output: (None, 64, 64, 32)
batch_normalization_4	BatchNormalization	input: (None, 64, 64, 32)	output: (None, 64, 64, 32)
activation_4	Activation	input: (None, 64, 64, 32)	output: (None, 64, 64, 32)
conv2d_transpose_2	Conv2DTranspose	input: (None, 64, 64, 32)	output: (None, 64, 64, 1)
batch_normalization_5	BatchNormalization	input: (None, 64, 64, 1)	output: (None, 64, 64, 1)
activation_5	Activation	input: (None, 64, 64, 1)	output: (None, 64, 64, 1)

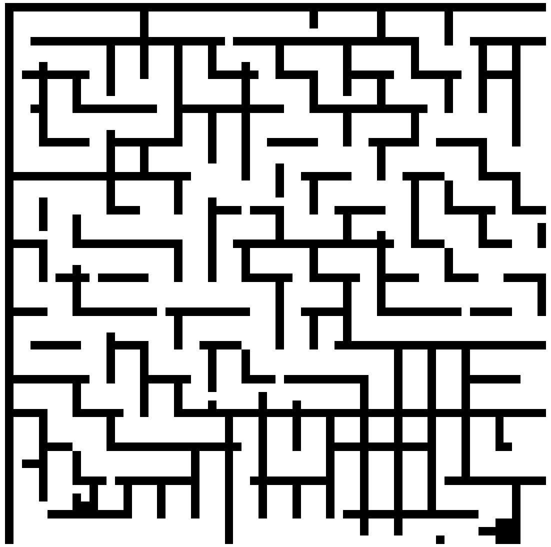
ARCHITECTURE – SECOND MODEL

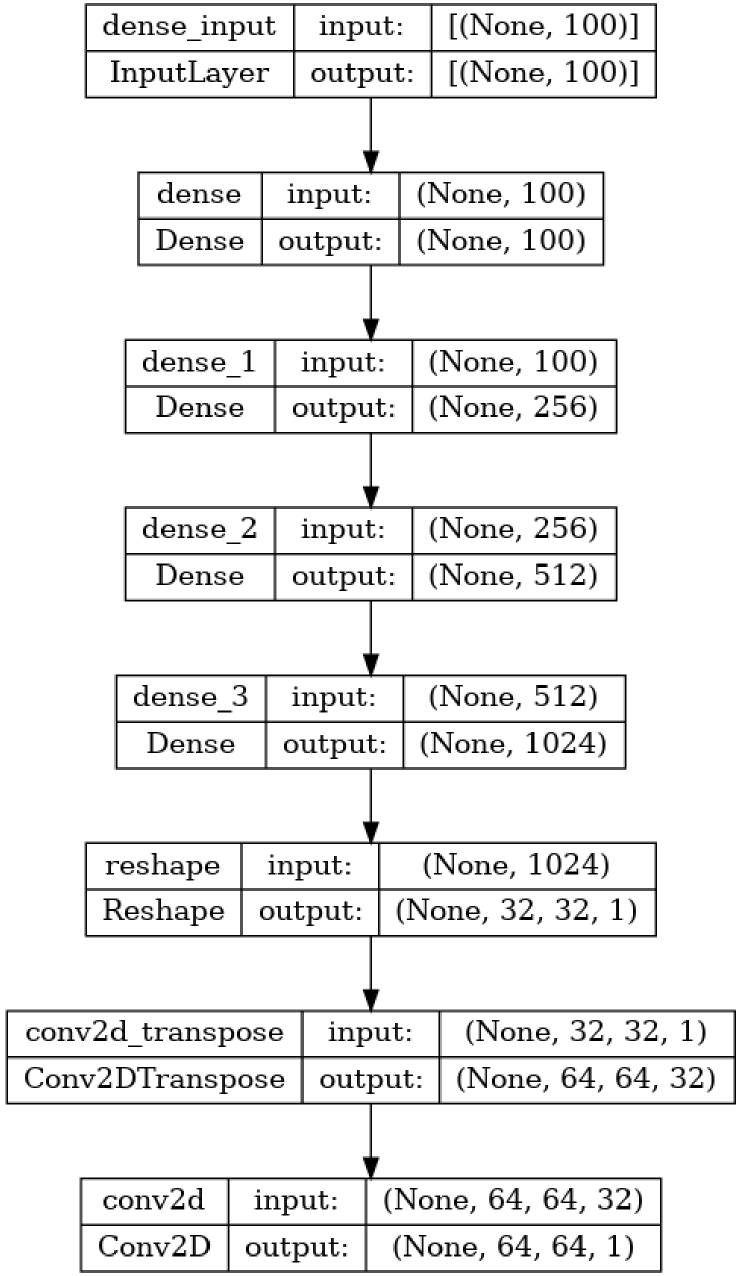




```
dense_input  input:  [(None, 100)]
InputLayer   output: [(None, 100)]

dense  input:  (None, 100)
Dense  output: (None, 128)

batch_normalization  input:  (None, 128)
BatchNormalization   output: (None, 128)

activation  input:  (None, 128)
Activation  output: (None, 128)

dense_1  input:  (None, 128)
Dense    output: (None, 256)

batch_normalization_1  input:  (None, 256)
BatchNormalization     output: (None, 256)

activation_1  input:  (None, 256)
Activation    output: (None, 256)

dense_2  input:  (None, 256)
Dense    output: (None, 512)

batch_normalization_2  input:  (None, 512)
BatchNormalization     output: (None, 512)

activation_2  input:  (None, 512)
Activation    output: (None, 512)

reshape  input:  (None, 512)
Reshape  output: (None, 16, 16, 2)

conv2d_transpose  input:  (None, 16, 16, 2)
Conv2DTranspose   output: (None, 32, 32, 64)

batch_normalization_3  input:  (None, 32, 32, 64)
BatchNormalization     output: (None, 32, 32, 64)

activation_3  input:  (None, 32, 32, 64)
Activation    output: (None, 32, 32, 64)

conv2d_transpose_1  input:  (None, 32, 32, 64)
Conv2DTranspose     output: (None, 64, 64, 32)

batch_normalization_4  input:  (None, 64, 64, 32)
BatchNormalization     output: (None, 64, 64, 32)

activation_4  input:  (None, 64, 64, 32)
Activation    output: (None, 64, 64, 32)

conv2d_transpose_2  input:  (None, 64, 64, 32)
Conv2DTranspose     output: (None, 64, 64, 1)

batch_normalization_5  input:  (None, 64, 64, 1)
BatchNormalization     output: (None, 64, 64, 1)

activation_5  input:  (None, 64, 64, 1)
Activation    output: (None, 64, 64, 1)
```

ARCHITECTURE – THIRD MODEL





dense_input	input:	[(None, 100)]
InputLayer	output:	[(None, 100)]

dense	input:	(None, 100)
Dense	output:	(None, 100)

dense_1	input:	(None, 100)
Dense	output:	(None, 256)

dense_2	input:	(None, 256)
Dense	output:	(None, 512)

dense_3	input:	(None, 512)
Dense	output:	(None, 1024)

reshape	input:	(None, 1024)
Reshape	output:	(None, 32, 32, 1)

conv2d_transpose	input:	(None, 32, 32, 1)
Conv2DTranspose	output:	(None, 64, 64, 32)

conv2d	input:	(None, 64, 64, 32)
Conv2D	output:	(None, 64, 64, 1)

RESULTS

For a sample size of 10000 mazes with a 64x64 size:

Approach	Average speed (in seconds)
Depth First Search	0.01074
Hunt and Kill	0.03418
Binary Tree	0.00283
Kruskal Randomized	1.13519
Aldous Broder	0.20148

RESULTS

For a sample size of 10000 mazes with a 64x64 size:

Model	Input size	Average speed (in seconds)
First model	(100, 1)	0.00003
First model	(100, 1000)	0.00080
Second model	(100, 1)	0.00004
Second model	(100, 1000)	0.00010
Third model	(100, 1)	0.00003
Third model	(100, 1000)	0.00040

FUTURE IDEAS

Already working on a public accessible API for maze generation using the provided techniques and models

Using Autoencoders

Using Genetic Algorithms to evolve mazes

Aiming for 4 variations for models: small D, small G, small D big G, big D, small G, big D big G over a set of predefined architectures (inspired by the cross-validation approach)

The aim is to build a fast method for creating both 2D and 3D models. For the 3D models I plan to use the API in a separate application

Other ideas:

- infinite mazes generated by patches instead of a whole grid at once
- Using NN designed for graph generation
- Using GANs aimed not at image creation, but actual “maze” data

FUTURE IDEAS

Using Autoencoders (mainly the decoder part) to create mazes from random noise

Using Genetic Algorithms to evolve mazes starting from semi-random noise

- Instead of relying only on random noise, preprocess it

Further testing with GANs using inspiration from cross-validation techniques

- Big/small structure for both discriminator and generator

Adjust GANs for maze “data” instead of maze “pictures”

- Only deep connected layers for the generator, then a post processing step to create the actual image

Using Neural Networks designed to work directly with graphs (Graph Neural Networks)

FUTURE IDEAS

Providing a public API for maze generation using AI (currently in active development)

The final goal is to build a reliable API for both 2D and 3D maze generation

- I plan to implement a simple application that uses the 3D API to generate content for the user

Other research directions:

- Infinite mazes and their applications
- Analysis over whole maze generation versus patch maze generation

Q & A