

Lecture 8:

Procedural Content Generation

- ECS7002P – Artificial Intelligence in Games
- Raluca D. Gaina – r.d.gaina@qmul.ac.uk
- Office: CS.335



<http://gameai.eecs.qmul.ac.uk>

Queen Mary University of London

Outline

- ✓ Introduction
- ✓ Applications
- ✓ PCG Types
 - ✓ Level Generation
 - ✓ Rules & Game Mechanics
 - ✓ Others
- ❑ Methods
 - ✓ Constructive
 - ❑ Search Based
 - ❑ Evaluation & Testing
- ❑ Examples & Benchmarks

Procedural Content Generation

Search Based Methods

Generative Methods

- Find the right tool for the job!
- **Search-based**
 - Given a possible space of fully created content, find the one that is most suitable.
 - Using search algorithms, i.e. evolutionary algorithms.
 - **Advantages:**
 - Some controllability.
 - Higher creativity possibilities.
 - Higher expressivity and diversity, can surprise with results obtained!
 - Setting high-level goals.
 - Can work online, at runtime.
 - **Disadvantages:**
 - More complex to implement and interpret.
 - Lower controllability and reliability.
 - Can be slower.
 - What is “good” content?



Search Based PCG

- Generating content is seen as a search, among all possible solutions, for a candidate that fulfils certain criteria. For this, we need three components:
 - **Search algorithm:** in charge of finding possible solutions for the problem.
 - **Content representation:** defines what type of content must be generated, and its representation is crucial for PCG to work. Examples are levels, textures, items, etc., which can be represented differently, as strings, array of numbers or graphs.
 - **Evaluation function(s):** A procedure to assess how good is a given solution, giving it a numerical value. This indicates the criteria that the solution must fulfil, and could encode different things such as playability, beauty, fun, etc.

Procedural Content Generation

Search Based Methods



Constraint Solving

- Constraint satisfaction, constraint propagation.
- Use **constraints** to define desired output: facts that must be *true* in the final output, manually designed or learned.
- Search in the space of possible values to find those solutions that meet the constraints.
- **Input:** set of variables, set of possible values for the variables, set of constraints.
- **Output:** all variables are filled with a specific value.
- Steps:
 - All variables associated with a list of all possible values.
 - Pick 1 variable and choose arbitrarily from its remaining values (**collapse**).
 - Based on the constraints, remove values from the surrounding variables to represent that they are no longer possible (**propagation**).
 - End when all variables have exactly only one value.
- Main advantage: constraint solvers readily available.

Constraint Solving Example

- Variables: grid cells

- Possible tiles:

- Blank 
- Wall 
- Enemy 
- Treasure 

- Constraints:

- There must be a path from the entrance to all treasure and all enemies.
- There can only be at most 1 treasure.
- There can only be at most 2 enemies.
- Enemies cannot be next to each other.

- Example by Dr. Stephen Lee-Urban, Georgia Technology Research Institute

1,2 3,4	E	1,2 3,4
1,2 3,4	1,2 3,4	1,2 3,4
1,2 3,4	1,2 3,4	1,2 3,4

Constraint Solving Example

- Variables: grid cells

- Possible tiles:

- Blank 
- Wall 
- Enemy 
- Treasure 

- Constraints:

- There must be a path from the entrance to all treasure and all enemies.
- There can only be at most 1 treasure.
- There can only be at most 2 enemies.
- Enemies cannot be next to each other.**

- Example by Dr. Stephen Lee-Urban, Georgia Technology Research Institute

1,2 3,4	E	1,2 3,4
1,2 3,4	1,2 3,4	1,2 3,4
1,2 3,4	1,2 3,4	

Collapse

1,2 3,4	E	1,2 3,4
1,2 3,4	1,2 3,4	1,2 4
1,2 3,4	1,2 4	

Propagate

Constraint Solving Example

- Variables: grid cells

- Possible tiles:

- Blank 
- Wall 
- Enemy 
- Treasure 


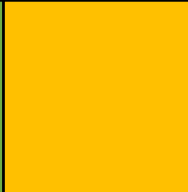
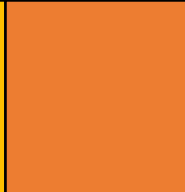
- Constraints:

- There must be a path from the entrance to all treasure and all enemies.**
- There can only be at most 1 treasure.**
- There can only be at most 2 enemies.
- Enemies cannot be next to each other.

- Example by Dr. Stephen Lee-Urban, Georgia Technology Research Institute

1,2 3,4	E	1,2 3,4
1,2 3,4	1,2 3,4	1,2 4
1,2 3,4		

Collapse

1,2 3	E	1,2 3
1,2 3	1,2 3	
1,2 3		

Propagate

Constraint Solving Example

- Variables: grid cells

- Possible tiles:

- Blank 
- Wall 
- Enemy 
- Treasure 

- Constraints:

- There must be a path from the entrance to all treasure and all enemies.
- There can only be at most 1 treasure.
- There can only be at most 2 enemies.**
- Enemies cannot be next to each other.

- Example by Dr. Stephen Lee-Urban, Georgia Technology Research Institute

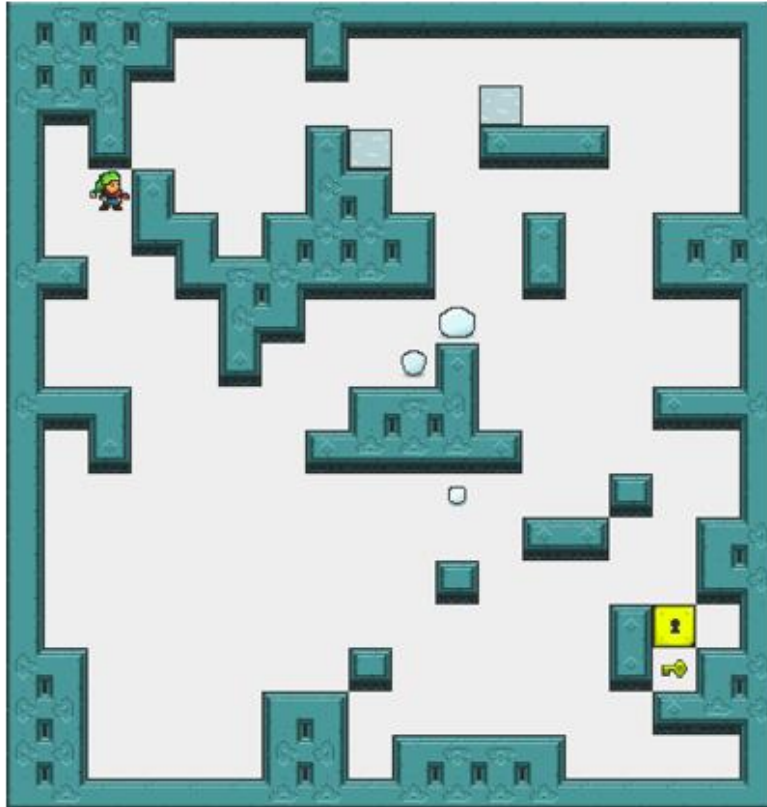
	E	1,2 3
1,2 3	1,2 3	
1,2 3		

Collapse

	E	1,2
1,2		
1,2		

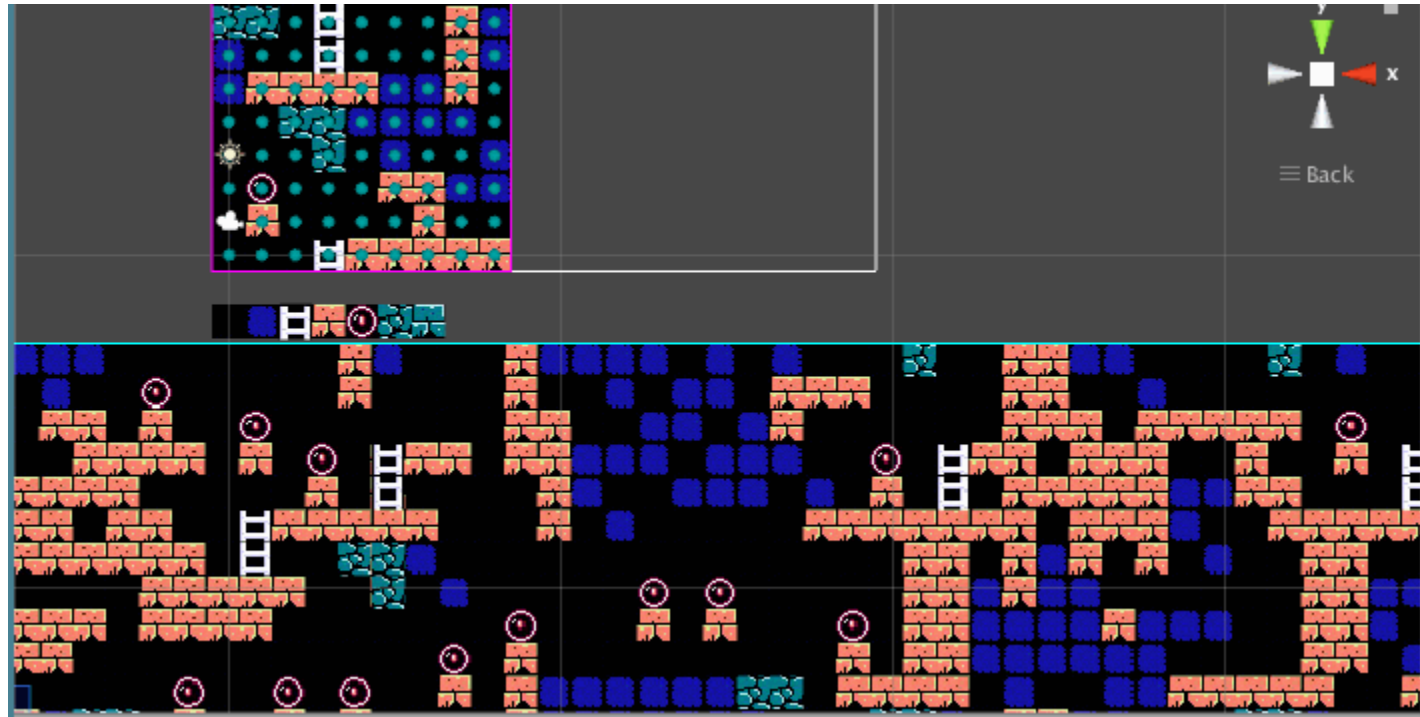
Propagate

Constraint Solving Example



Wave Function Collapse (1)

- Example-driven constraint-solving algorithm.
- Extracts **patterns** from the example(s) given (particular, unique configuration of input tiles).
- Determines valid intersections of patterns to design **constraints**.
- Every local window (or pattern) in the output occurs somewhere in the input.



<http://www.procjam.com/tutorials/wfc/>

Wave Function Collapse (2)

Algorithm steps:

- Extract NxN patterns from sample (optional: rotations and reflections of patterns).
- Determine valid intersection of patterns for constraints.
- Initialize output as empty, with all cells in grid having a list of possible valid patterns.
- **Repeat:**
 - Choose cell with more than 1 valid option.
 - **If** contradiction reached, **return** error.
 - **If** all cells have 1 option, **return** level.
 - Choose pattern from valid options, attempting to match pattern distribution from input.
 - Replace cell with chosen pattern.
- **Return** level.

<https://github.com/mxgmn/WaveFunctionCollapse>

Procedural Content Generation Search Based Methods



Content Representation

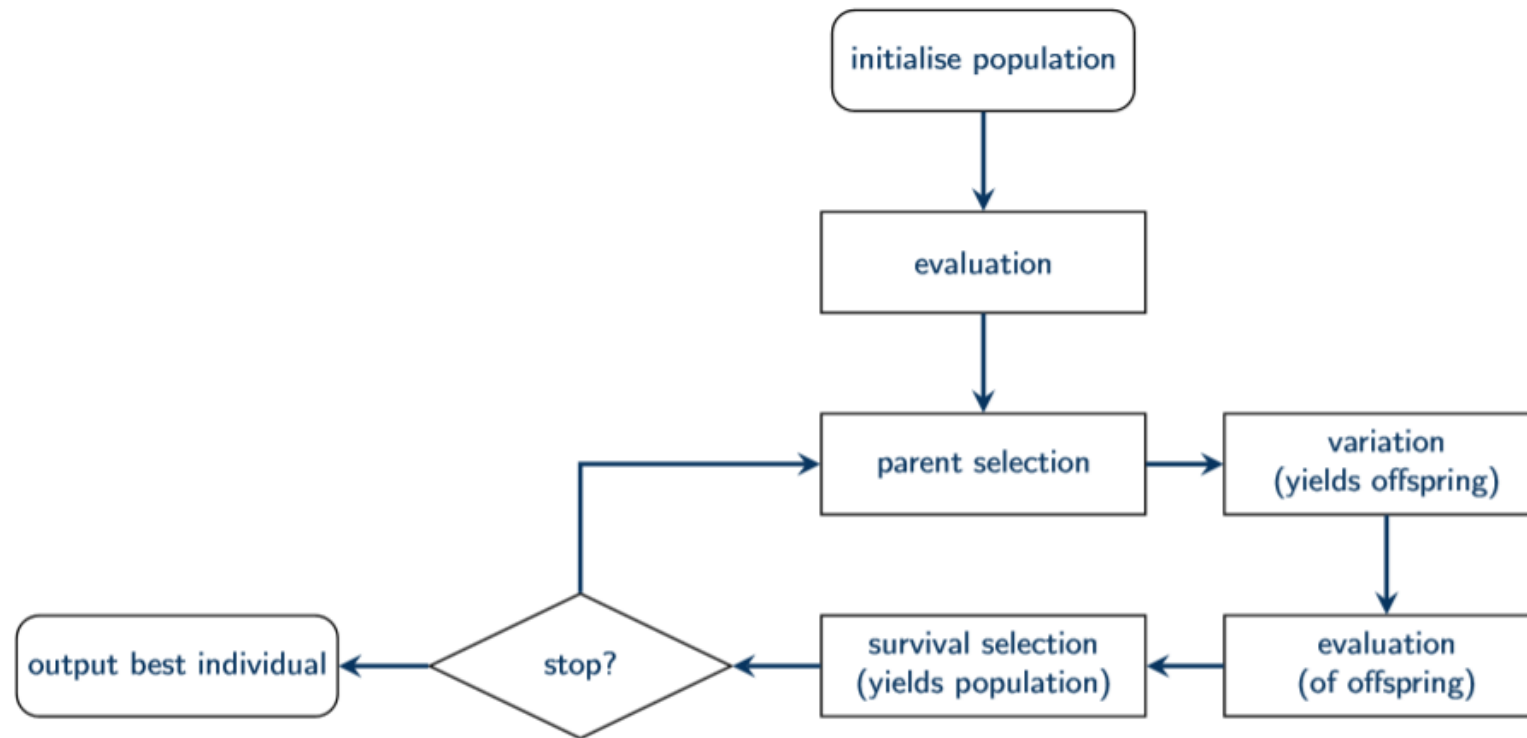
The choice of representation plays an important role in the efficiency of the algorithm. We must distinguish between two concepts:

- **Genotype**: encoding of the solution (i.e, an array of float values, instructions).
- **Phenotype**: expression of the genotype as a solution (i.e., positions of objects in a level, the level itself).

Representation is defined in the genotype, and can be more or less close to the phenotype. For example, a dungeon level could be represented differently:

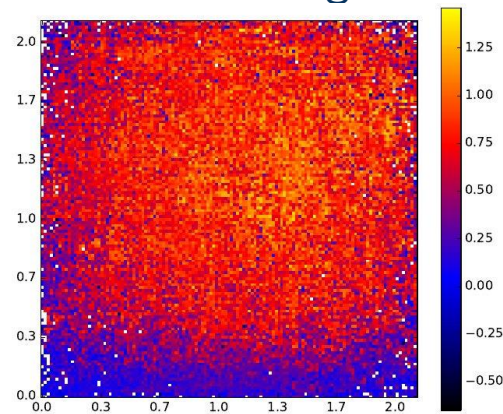
- Grid cell: each element in the game is precisely specified.
- List of positions, orientations, and sizes of areas.
- Repository of segments/chunks of levels to combine.
- Desired properties: number of corridors, rooms, etc.
- Random number seed: the level generator creates a level taking only one number as an input.

Evolutionary Algorithms



Quality Diversity Algorithms

- Explore different behaviours while optimizing performance.
- Keep track of solutions achieving diverse behaviours.
- MAP-Elites:
 - Create an empty, N-dimensional map of elites: {solutions X described by behaviours B and their performances P}
 - Initialize by generating P random solutions.
 - Repeat for several iterations:
 - Select elites x_1 , x_2 and generate offspring x' (mutation/crossover).
 - Simulate the candidate solution x' and record its feature descriptor b' and performance p' .
 - If the appropriate cell in the map is empty or its occupant's performance is $\leq p'$, then store the performance of x' in the map of elites according to its feature descriptor b' .
- More:
 - Novelty search
 - Surprise search



Co-Evolution

- Evolve a solution $p \in P$ relative to another solution $q \in Q$ that is also evolving.
 - P = set of all possible solutions to problem 1.
 - Q = set of all possible solutions to problem 2.
- Example:
 - 2-player games, evolve your best strategy against the opponent's best strategy
 - “good” vs “bad” levels
- $f(p) \rightarrow f(p|q)$ or $f(p \mid \mathbb{E}[q \in Q])$
- Can be expressed by evolving 2 populations simultaneously.
- Advantages:
 - No need to define absolute heuristics
 - More flexible and adaptive
 - More reflective of natural evolution

Procedural Content Generation Search Based Methods



Generative Adversarial Networks (GANs)

Generator



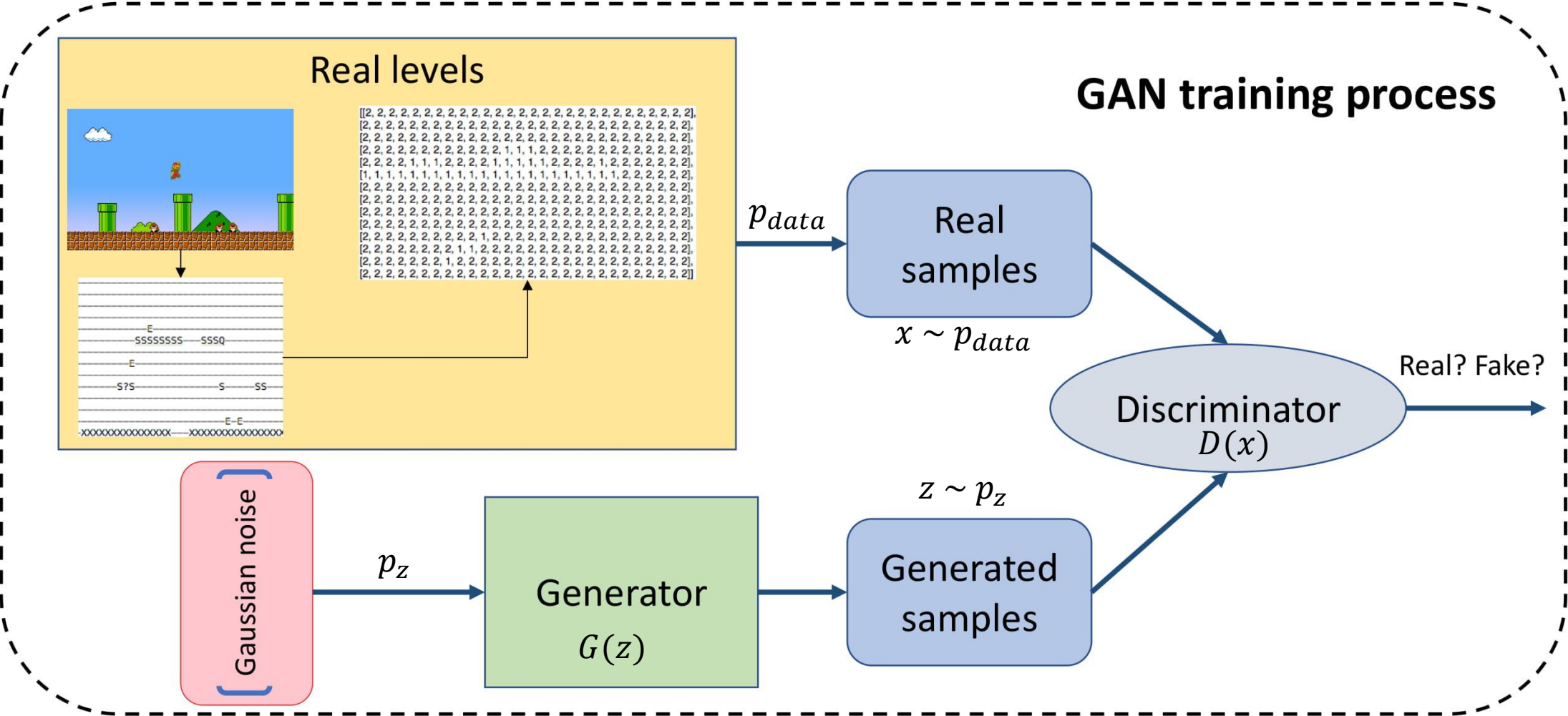
Discriminator



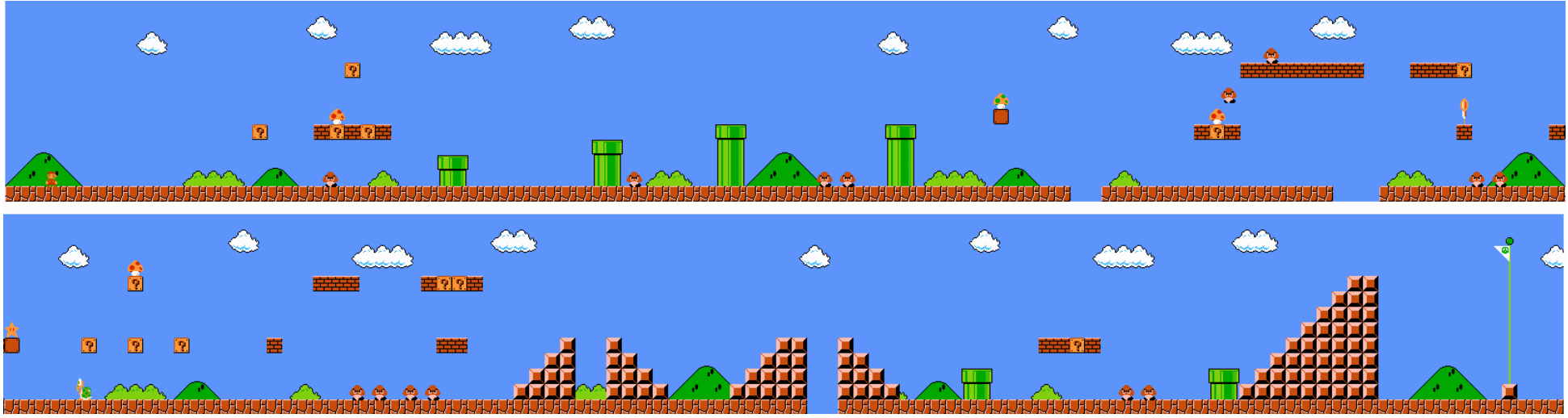
<https://youtu.be/NObqDuPuk7Q>

Volz, V., Schrum, J., Liu, J., Lucas, S.M., Smith, A. and Risi, S., 2018, July. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 221-228). ACM.

GAN Training



Formal Description



- Train alternatively in 2 passes, first discriminator $D(x)$, then generator $G(z)$:
 - $D(x)$: maximize probability of correct label
 - $G(z)$: minimize probability of correct label
- Minimax game: $\max_D \min_G V(D, G)$, where:

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Procedural Content Generation

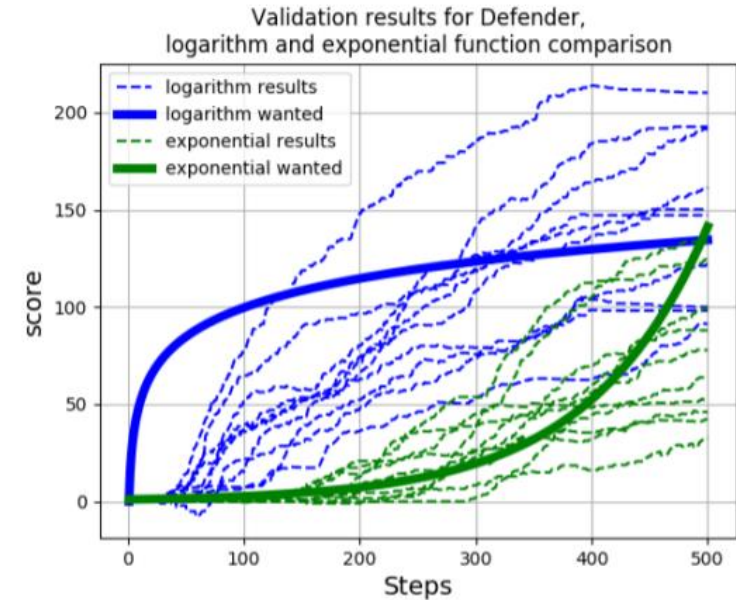
Search Based Methods



Search in the Game Parameter Space

```
ParameterSet
#{Name of the parameter} > {values(min, inc, max)/(boolean)} {descriptive string}

SPHUNT    > values=0.1:0.1:1.5    string=Hunter_Speed    value=1.7
SPGHOST   > values=0.1:0.1:1.0    string=Ghost_Speed
SPE0      > values=0.1:0.1:1.0    string=Enemy0_Speed
SPE1      > values=0.1:0.1:1.0    string=Enemy1_Speed
PE0       > values=0.01:0.01:0.1  string=Enemy0_prob
PE1       > values=0.01:0.01:0.1  string=Enemy1_prob
TTLMIS    > values=1:5:30          string=Missile_TimeToLive value=31
SPMIS     > values=0.1:0.1:1.0    string=Missile_Speed
SPMISA    > values=0.1:0.1:1.0    string=MissileA_Speed
SPMISB    > values=0.1:0.1:1.0    string=MissileB_Speed
```



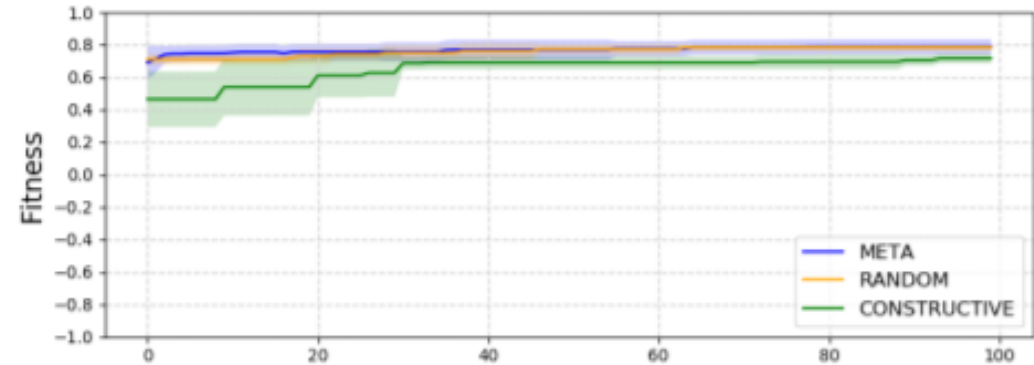
- Search space is all possible combinations of game parameters.
- Find parameter set to:
 - Obtain specific player score progression.
 - Favour long-sighted RHEA vs short-sighted RHEA: https://youtu.be/PWHGM_Bd6Jw
 - Favour short-sighted RHEA vs long-sighted RHEA: https://youtu.be/t6usa_f9jig

Kamolwan Kunanusont, Simon Lucas and Diego Perez Liebana. Modeling Player Experience with the N-Tuple Bandit Evolutionary Algorithm, in Proceedings of Artificial Intelligence in Interactive Digital Entertainment (AIIDE), 2018.

Search in the Generator Parameter Space

TABLE I: Generator parameter search space

Parameter	Type	Search space
Sprite usage _n	Float	[0,1]
<i>x</i> -Gaussian sampling _n	Boolean	true, false
<i>y</i> -Gaussian sampling _n	Boolean	true, false
<i>x</i> -Distribution mean _n	Float	[0,1]
<i>y</i> -Distribution mean _n	Float	[0,1]
<i>x</i> -Distribution st.dev _n	Float	[0,1]
<i>y</i> -Distribution st.dev _n	Float	[0,1]
Level border	Boolean	true, false
Width	Integer	[4,18]
Height	Integer	[4,18]



- Search space is all possible combinations of generator parameters.
- Find parameter set to maximize generated levels fitness.

Drageset, O., Winands, M.H., Gaina, R.D. and Perez-Liebana, D., 2019, August. Optimising Level Generators for General Video Game AI. In *2019 IEEE Conference on Games (CoG)* (pp. 1-8). IEEE.

Outline

- ✓ Introduction
- ✓ Applications
- ✓ PCG Types
 - ✓ Level Generation
 - ✓ Rules & Game Mechanics
 - ✓ Others
- ❑ Methods
 - ✓ Constructive
 - ✓ Search Based
 - ❑ Evaluation & Testing
- ❑ Examples & Benchmarks

Procedural Content Generation

Evaluation Methods and Testing

Evaluation Functions (1)

The evaluation function assigns a score (fitness) to each candidate solution. In principle, this function should evaluate the desired properties of the solution. These functions can be of different forms:

- **Direct evaluation (feature-based) functions:** some objective features are retrieved from the map and a score for the level is provided by an evaluation function.
 - Tile distribution
 - Number of floor gaps
 - Level balance
 - Enemy frequency
 - etc.

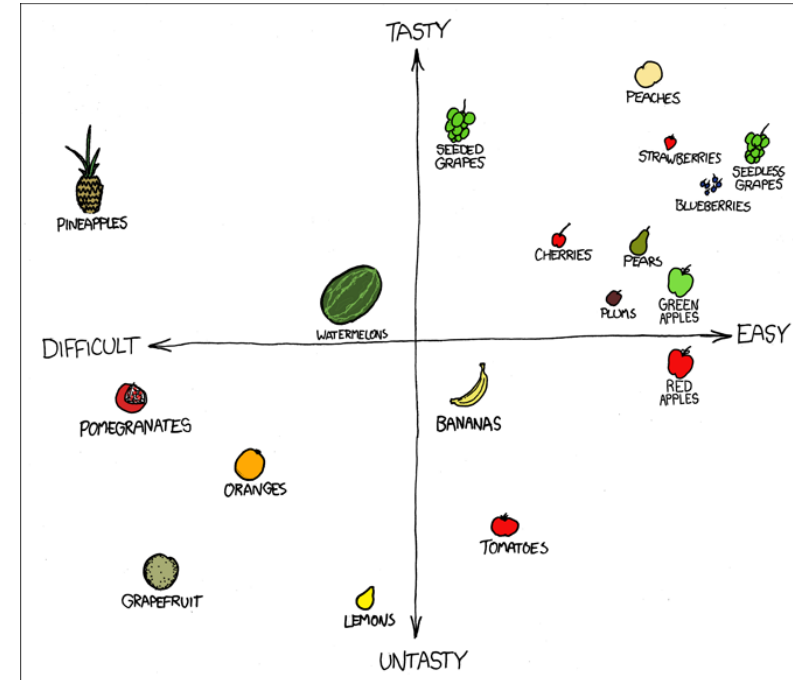
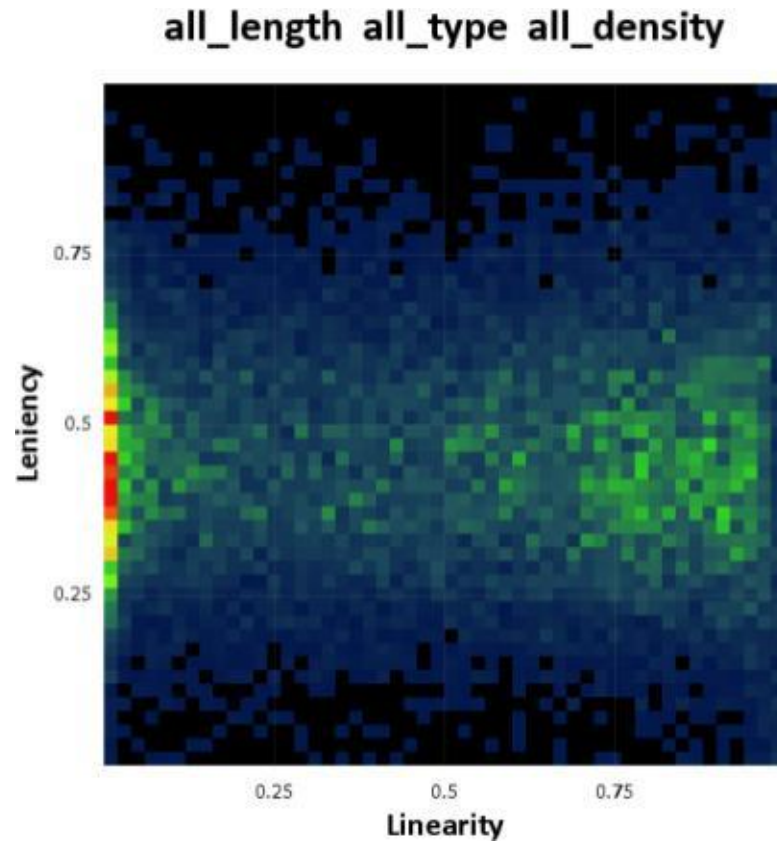


Evaluation Functions (2)

- **Simulation-based evaluation functions:** one or more programmed bots play the game. The bots can be fully hand coded or they can imitate a human player using machine learning techniques. A measure of their performance is then taken, which is used to score the maps:
 - Win rate
 - Distance travelled
 - Action distribution
 - Death locations
 - etc.
- **Interactive (human-in-the-loop) evaluation:** a human plays the game and feedback is obtained from them. This can be done either explicitly, with a questionnaire, or implicitly, measuring gameplay features (as above).

Vanessa Volz, Dan Ashlock, Simon Colton, Steve Dahlskog, Jialin Liu, Simon M. Lucas, Diego Perez Liebana, and Tommy Thompson. **Gameplay Evaluation Measures**, in *Artificial and Computational Intelligence in Games: AI-Driven Game Design (Dagstuhl Seminar 17471)* edited by Pieter Spronck and Elisabeth Andre, and Michael Cook and Mike Preuss, Vol 7:11 (86-129) (2018)

Testing Generators



Playtests



Outline

- ✓ Introduction
- ✓ Applications
- ✓ PCG Types
 - ✓ Level Generation
 - ✓ Rules & Game Mechanics
 - ✓ Others
- ✓ Methods
 - ✓ Constructive
 - ✓ Search Based
 - ✓ Evaluation & Testing
- ❑ Examples & Benchmarks

Procedural Content Generation Examples & Benchmarks

Procedural Content Generation Examples & Benchmarks

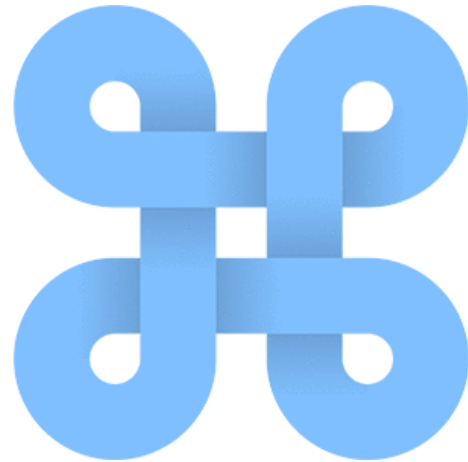


LUDI

Example 1: LUDII (<https://ludii.games/>)

LUDI is a system originally developed by Cameron Browne (Maastricht University) that designs and evaluates board games to guide automated search for new games.

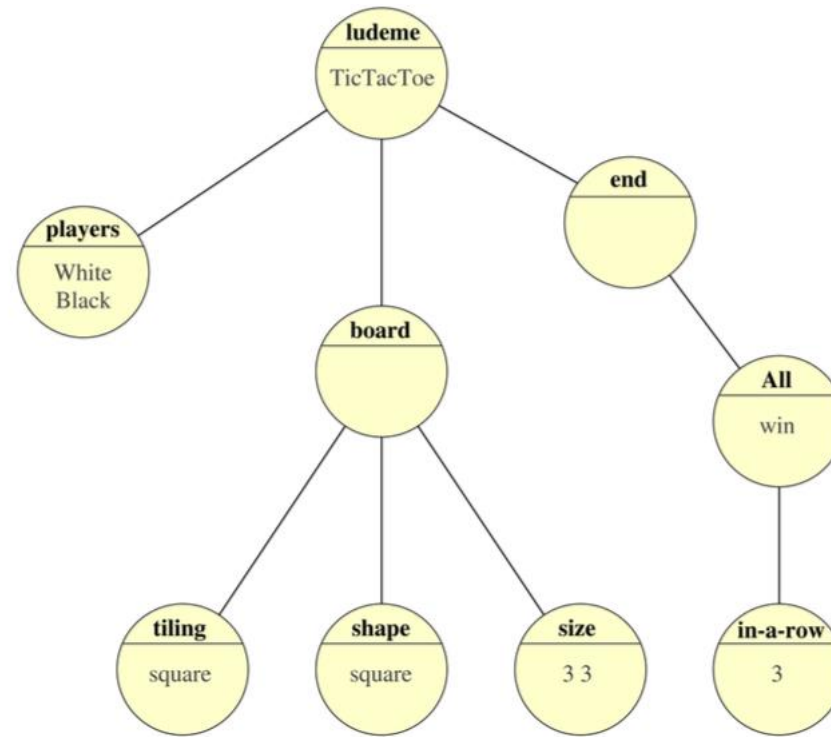
The framework was extended over the last year (LUDII) to include card games, dice games, mathematical games and simple video games as well, together with a set of sample general-purpose agents able to play any of the games (with different abilities).



Browne, Cameron. Automatic generation and evaluation of recombination games. PhD Thesis. Queensland University of Technology, 2008.

LUDI: A Language to Describe Games

```
(ludeme TicTacToe
  (players White Black
    (board
      (tiling square i-nbors)
      (shape square)
      (size 3 3)
    )
  )
  (end (All win (in-a-row 3) ) )
)
```

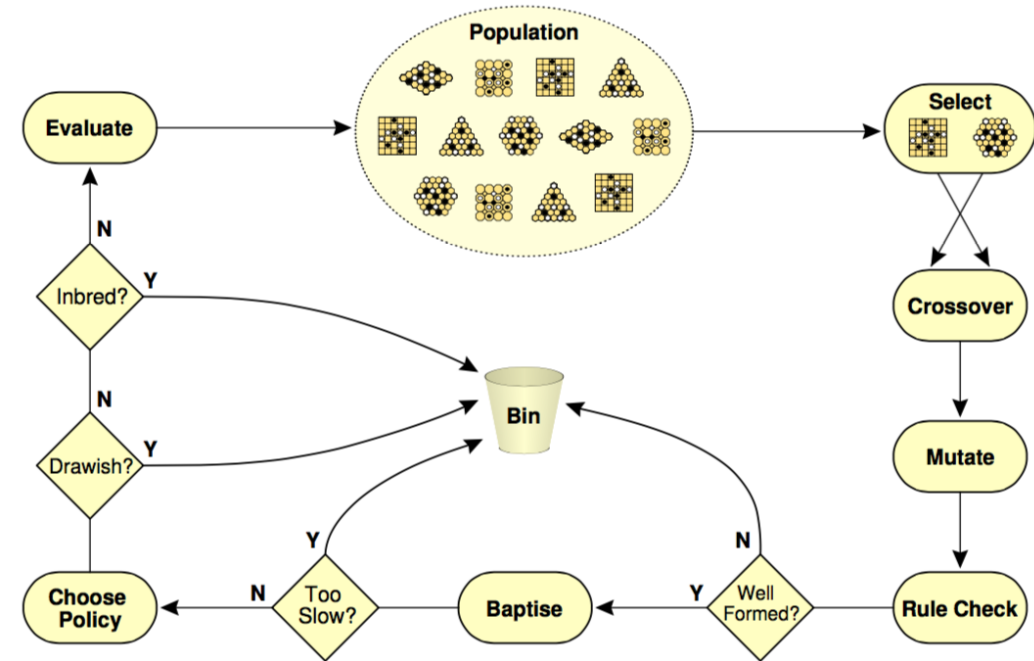


- **Ludeme**: unit of game information.
- Adjacency assumed between orthogonal neighbours, but i-nbors flag adds diagonal neighbours.

LUDI: Evolution

Games are evolved by Genetic Programming (GP):

- Evolves tree structures.
- Crossover allows changes of entire branches.
- Mutation changes values of particular nodes.



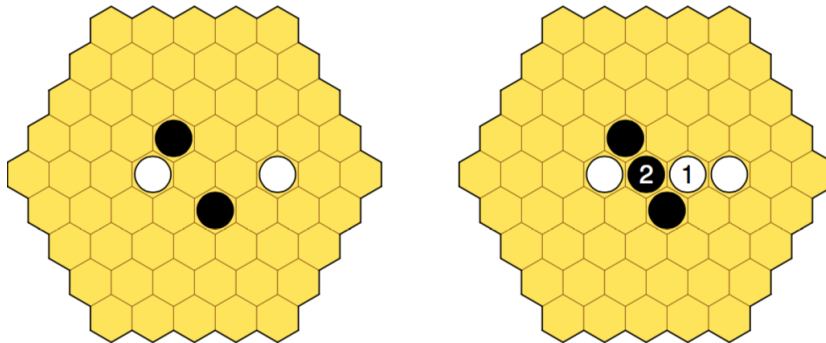
Each game generated is evaluated following 57 aesthetic measurements, divided into:

- **Intrinsic** criteria based directly on the rule set (complexity, move type, goal type...).
- **Playability** criteria based on the outcomes of the self-play trials: penalizes draws, unbalanced games towards one of the players, too strong first/second moves, too short or too long on average.
- **Quality** criteria based on trends in play (i.e., drama).

LUDI: Evolved Games

- LUDI evolved 1389 new games over a week when first released, of which 19 were deemed “playable”.
- The best of these is Yavalath:

```
(game Yavalath  
  (players White Black)  
  (board (tiling hex) (shape hex) (size 5))  
  (end (All win (in-a-row 4)) (All lose (in-a-row 3))))  
)
```



Digital Ludeme Project



- **Focus:** historical games.
- Increasingly larger game library including historical and geographical information, as well as other known evidence of the game's existence.
- Humans have enjoyed *play* for a long time, yet the rules of ancient games are often forgotten and records of such games include limited information:
 - What the game board might have looked like.
 - What the game was about.
 - How the players won.
- These records rarely include full information to be able to implement the game: how do you automatically generate possible games, given some information for guidance?

Procedural Content Generation Examples & Benchmarks

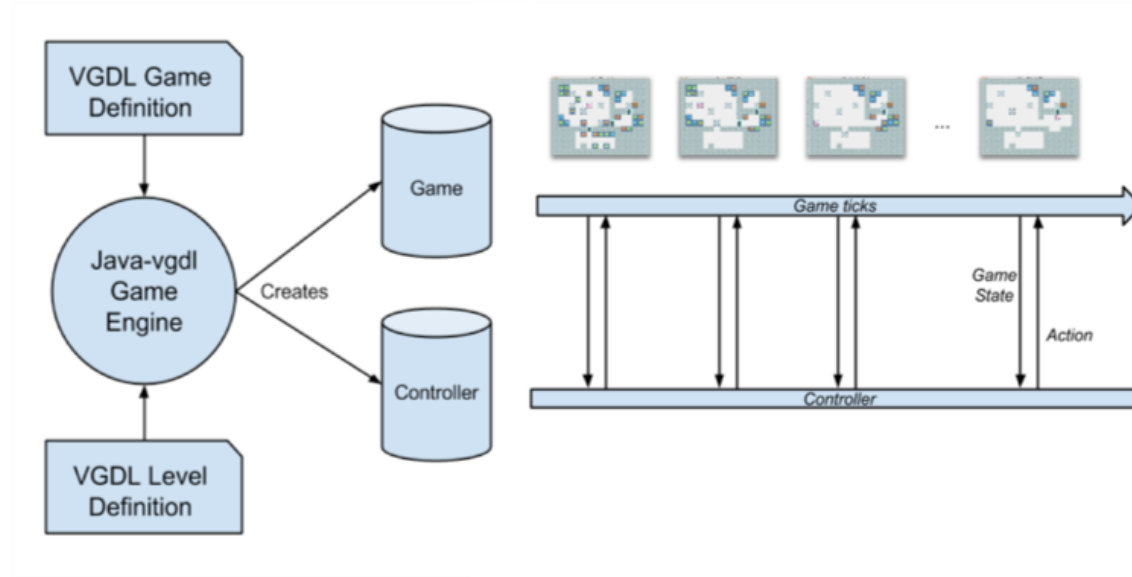


GVGAI

Example 2: GVGAI (<http://www.gvgai.net/>)

General Video Game AI (GVGAI) is a framework and competition aimed at promoting Game AI without relying on too much domain knowledge.

Over 160 real-time (40ms) games described in the Video Game Description Language (VGDL).



Perez-Liebana D, Liu J, Khalifa A, Gaina RD, Togelius J, Lucas SM. General video game AI: a multi-track framework for evaluating agents, games and content generation algorithms. arXiv preprint arXiv:1802.10363. 2018 Feb 28.

The GVGAI Level Generation Competition

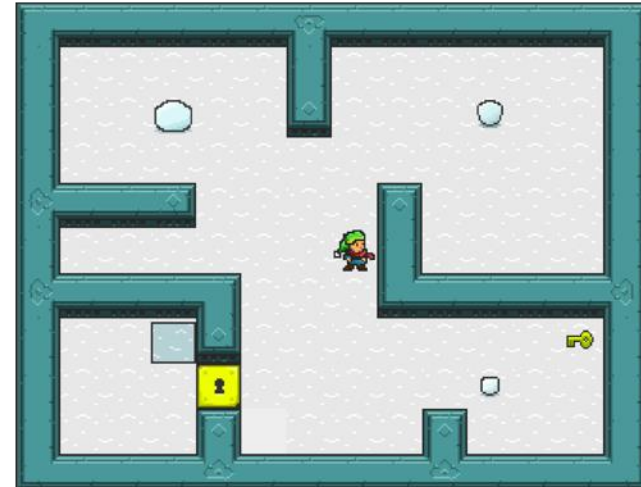
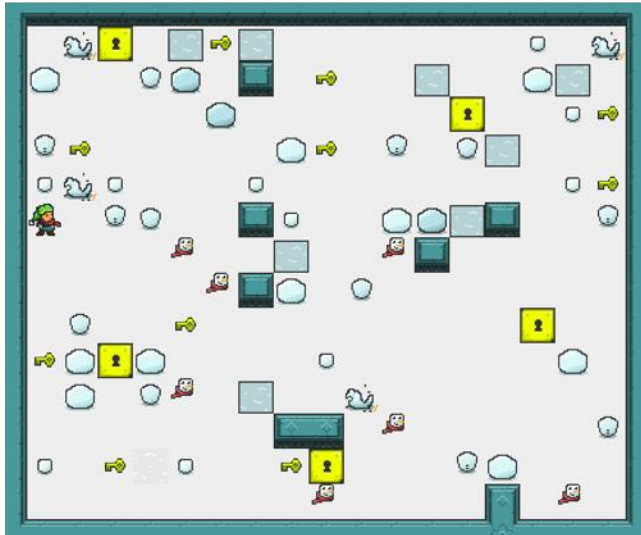
Challenge: generating levels for any game rules given:

- A **game description**, via Java objects, is supplied to the generator.
- It gives access to the main **game objects**: avatar, NPCs, resources, portals, static and moving sprites. The SpriteData structure holds information about the sprite (name, type, etc.)
- Similarly, **Interaction** and **Termination** Data are given, as well as the **level mapping** required (how sprites and sprite combinations are represented as tiles in the level by human designers).
- The generators can create levels and play them with agents for **5 hours**.
- **Samples** provided: random, constructive, search-based (Genetic Algorithm).

Khalifa A, Perez-Liebana D, Lucas SM, Togelius J. General video game level generation. In Proceedings of the Genetic and Evolutionary Computation Conference 2016 2016 Jul 20 (pp. 253-259). ACM.

The GVGAI Level Generation Competition

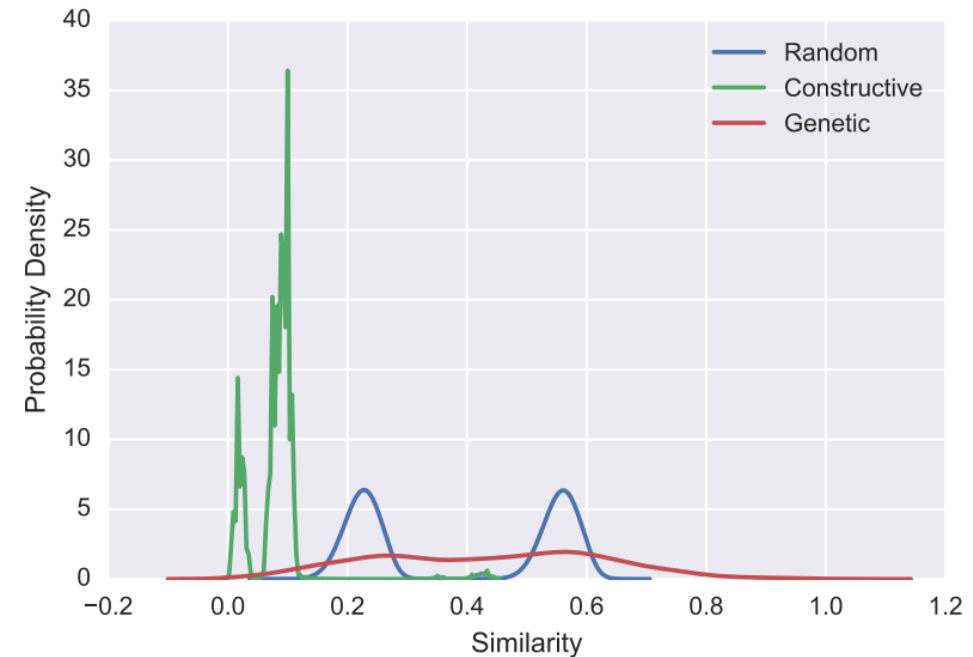
Rank	Username	Country	Percentage Preferred (%)	Description	Generator	Screenshots
1	easablade	New Zealand 🇳🇵	36	Description	Download	easablade
2	bhorn82287	Afghanistan 🇦🇫	12	Description	Download	bhorn82287
2	Jnicho	United Kingdom 🇬🇧	12	Description	Download	Jnicho
4	Number13	Germany 🇩🇪	8	Description	Download	Number13



The GVGAI Rule Generation Competition

GVGAI Rule Generation Competition: given a GVGAI **level** as input, the problem is to generate rules for the game that fit that level. Participants generate the interaction set and the termination conditions.

```
TerminationSet
  SpriteCounter stype=avatar limit=0 win=False
  Timeout limit=800 win=True
InteractionSet
  top EOS > wrapAround
  avatar downshot > killSprite
  upshot EOS > wrapAround
  top avatar > killSprite scoreChange=1
  avatar EOS > stepBack
  diamond avatar > killSprite scoreChange=1
  bottom EOS > wrapAround
  diamond EOS > wrapAround
  avatar upshot > killSprite
  bottom avatar > killSprite scoreChange=1
  downshot EOS > wrapAround
```



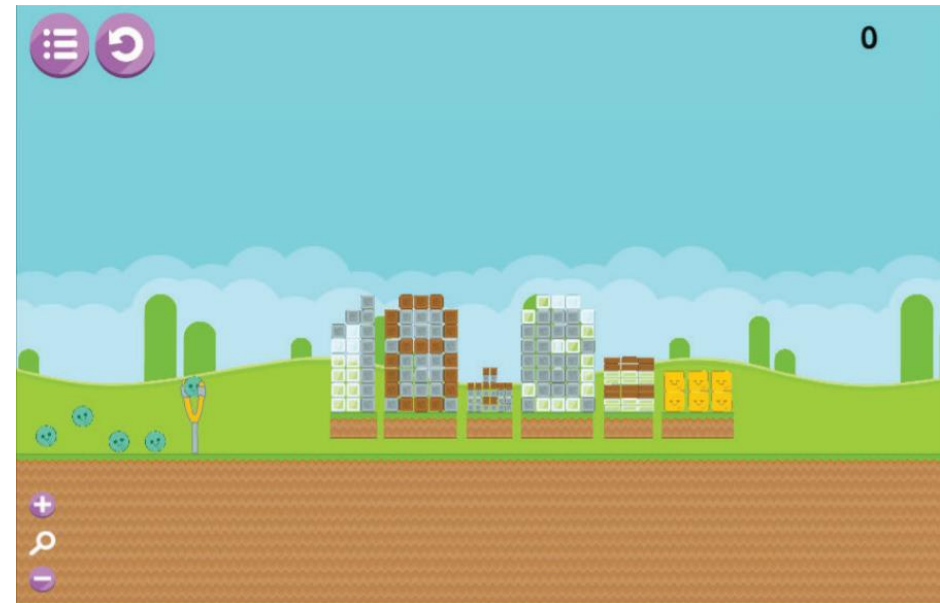
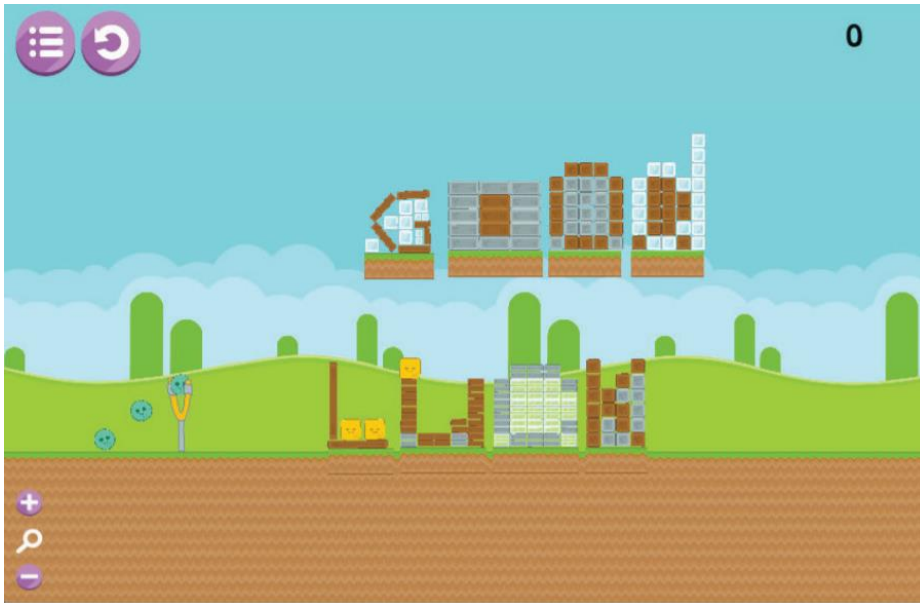
Khalifa A, Green MC, Perez-Liebana D, Togelius J. General video game rule generation. In Computational Intelligence and Games (CIG), 2017 IEEE Conference on 2017 Aug 22 (pp. 170-177). IEEE.

Procedural Content Generation Examples & Benchmarks

└─→ Angry Birds

Example 3: Angry Birds (https://aibirds.org/)

- Levels defined in XML format
 - Identifying all different blocks, their types and their position.
 - Placing pigs.
 - Defining the number, type and order of birds.
- Judged by humans for fun, aesthetics, difficulty.
- Should produce interesting and stable structures, as well as placing birds such that there is strategy involved in solving the level.



Angry Birds Level Generation

- Techniques are mostly rule-based:
 - Split the level area into several sections where structures will be built.
 - Build structures top-down, ensuring support for all elements placed at the top.
 - Choose materials randomly and birds that will be able to break the chosen materials.
 - Place pigs in strategic positions (some inside structures, some on top of structures, some on the ground).
- No simulation of AI or human gameplay.
- Some use machine learning to learn what human-designed levels look like and attempt to replicate that.
 - See next lecture on ML!

Angry Birds Level Generation

Some generate structures from templates (quotes generator).



Procedural Content Generation Examples & Benchmarks

└─→ Generative Design in Minecraft

Example 4: Generative Minecraft Competition (GMDC)

(<http://gendesignmc.engineering.nyu.edu/>)

- Generate a Minecraft settlement, given different maps (with different tile types or terrain features).
- Evaluated on 3 unseen maps and judged by humans on different criteria:
 - Adaptability
 - Functionality
 - Narrative
 - Aesthetics



<https://youtu.be/7eVK8UAWOaw>

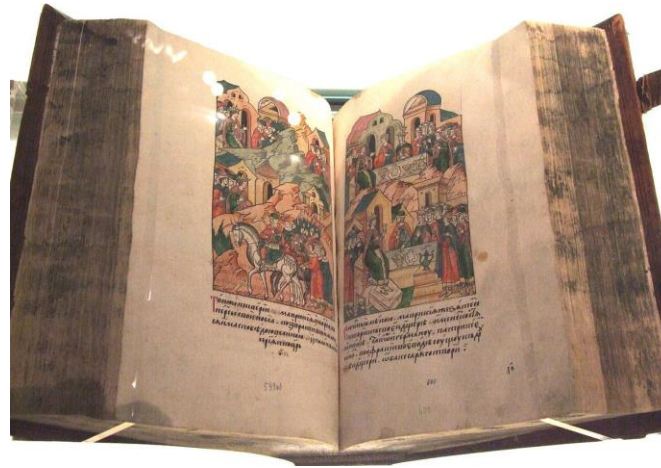
- General techniques used:

- Cellular automata
- Road building with A*
- Templated buildings, some with furnishings (beds, carpets, lighting)
- Biome and material adaptation, placement on suitable terrain
- Intelligent tree removal
- Illumination placement
- Bridges over water to connect different sections of a settlement
- Large signature buildings or structures (e.g. watch towers, windmills, market squares) with unique placement rules



2019 winner

GMDC Chronicle Challenge



- Produce a written book in Minecraft and place it inside a settlement, ideally easily found.
 - The book should contain the chronicle of the settlement (in English) and should contain a narrative on how this particular settlement came about.
 - Evaluated on overall quality and on how well it fits the settlement.
 - No entries yet...
-
- Salge, C., Guckelsberger, C., Green, M.C., Canaan, R. and Togelius, J., 2019. Generative Design in Minecraft: Chronicle Challenge. *arXiv preprint arXiv:1905.05888*.

Procedural Content Generation Examples & Benchmarks

└─→ Angelina

Example 5: ANGELINA (<http://www.gamesbyangelina.org/>)

- Games by Angelina is a project developed by Michael Cook (Queen Mary University of London).
 - Ruleset
 - Map
 - Initial placement of game objects
- Able to create commentary about the games it made
- Data mining (e.g. newsarticles)
- 2D and 3D

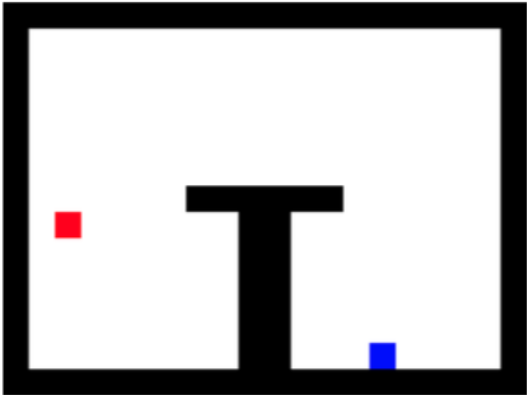
} co-evolution



https://youtu.be/yzWCJiNC_Lg

Key Features

- L-Systems Grammar
- Metroidvania-style platform games, where players incrementally gain powers that allow them to explore new areas of the world.
- Incremental addition of rules:
 - A game that can't be solved.
 - Added rules (via evolution) may make it solvable.
 - Reward these rules as they add utility to the player.

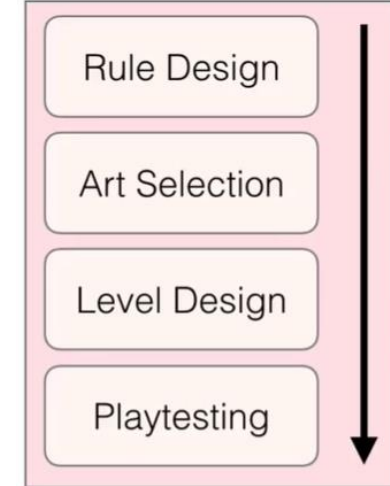


<https://youtu.be/sz0hn3FXTwc>
(AI in Games: Games by Angelina)



Redesigning Angelina

- Linear system, do one task at a time (no more co-evolution)
- Visualisation and interaction with the different tasks
- Continuous creation (creativity = lifetime process)
 - Database of projects (games, ideas etc.) with to-do lists
 - Memory of what it's done, what it's doing
 - Database of design modules or tasks (design rules, levels, play games etc.)
 - Move from task to task
 - Start games
 - Work on games
 - Publish finished games
 - Trash games
- Presence
 - Streaming on Twitch, user interaction through chat
 - Twitter interaction
 - Collaboration with people



Live design activity from the system

Level Design

Mouse Of The Dead

Mice will chase Cats. Mice push Mice. Mice push Mice. If Mice touch Mice, they eat them and you gain a point. If you score two points, you win!

Use the arrow keys to move Cats. Use the arrow keys to move Cats. If all Cats are on Mice, you lose!

Playout

Best Levels Played So Far

Say Hello!

#whats X Y - Ask me what the thing is at those co-ordinates!
#whatgame - Ask me what game I'm working on right now
#whatsnext - Ask me what things I'm working on this week

Status

I've been working for 2 hours and 1 minutes!

I'm currently designing levels for a game I'm making. This session, I've played 614 levels, and made over 818961 moves!

If you're interested in my games, you can play and download them online:
gamebyangelina.itch.io

Help ANGELINA Make Games!
Click Here

In-process framing of algorithmic judgement

Outline

- ✓ Introduction
- ✓ Applications
- ✓ PCG Types
 - ✓ Level Generation
 - ✓ Rules & Game Mechanics
 - ✓ Others
- ✓ Methods
 - ✓ Constructive
 - ✓ Search Based
 - ✓ Evaluation & Testing
- ✓ Examples & Benchmarks

Procjam (<https://itch.io/jam/procjam>)

Learn From The Experts!

Get inspired by amazing talks about generative systems in art, games, music and more!

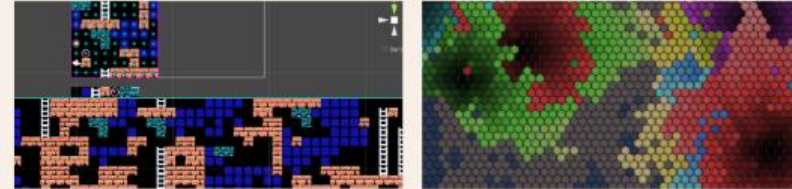
PROCJAM 2018 ▶ PLAY ALL



Click here to visit our YouTube channel!

Follow Our Tutorials

Learn about music generation, colour theory, and more from our tutorials (also in Spanish + Arabic)



Click here to see our tutorials!

Download Free Art

We have thousands of 3D and 2D art assets, made ready for generative projects, all for free!



Click here to get our art packs!

Read Seeds - Our Zine!

Our jam's zine, written by our community about their projects, ideas and inspiration!



Click here to download or read online!

<https://itch.io/jam/procjam/topic/602413/late-submissions>

EECS Research Showcase

- **December 5th 2019 @ The Octagon, QMUL**

- 13:00 - 15:30

- Posters
- Demos
- Industry visitors
- Coffee & cake!



- PhD Opportunities
 - QMUL Game AI Group: <http://gameai.eecs.qmul.ac.uk>
 - Intelligent Games and Games Intelligence (IGGI): <http://iggi.org.uk>