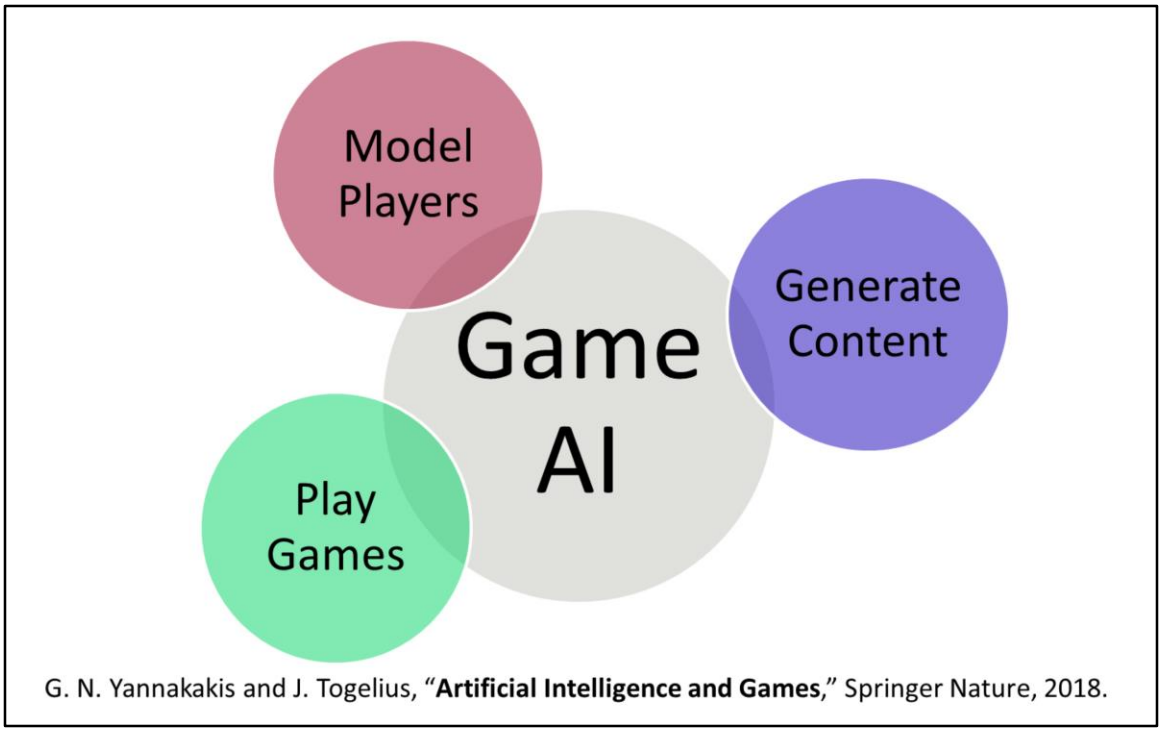
Artificial Intelligence and Games
Generating Content

Georgios N. Yannakakis
@yannakakis

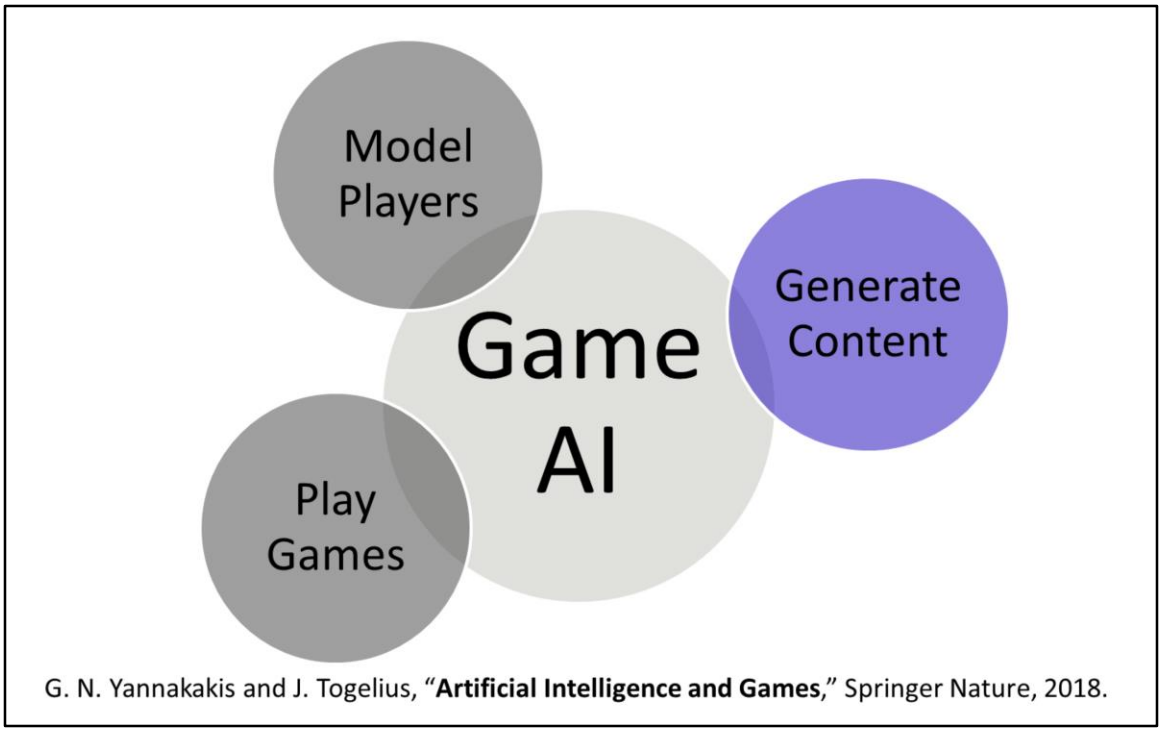
Julian Togelius
@togelius

These are the slides accompanying the book Artificial Intelligence and Games through the gameaibook.org website

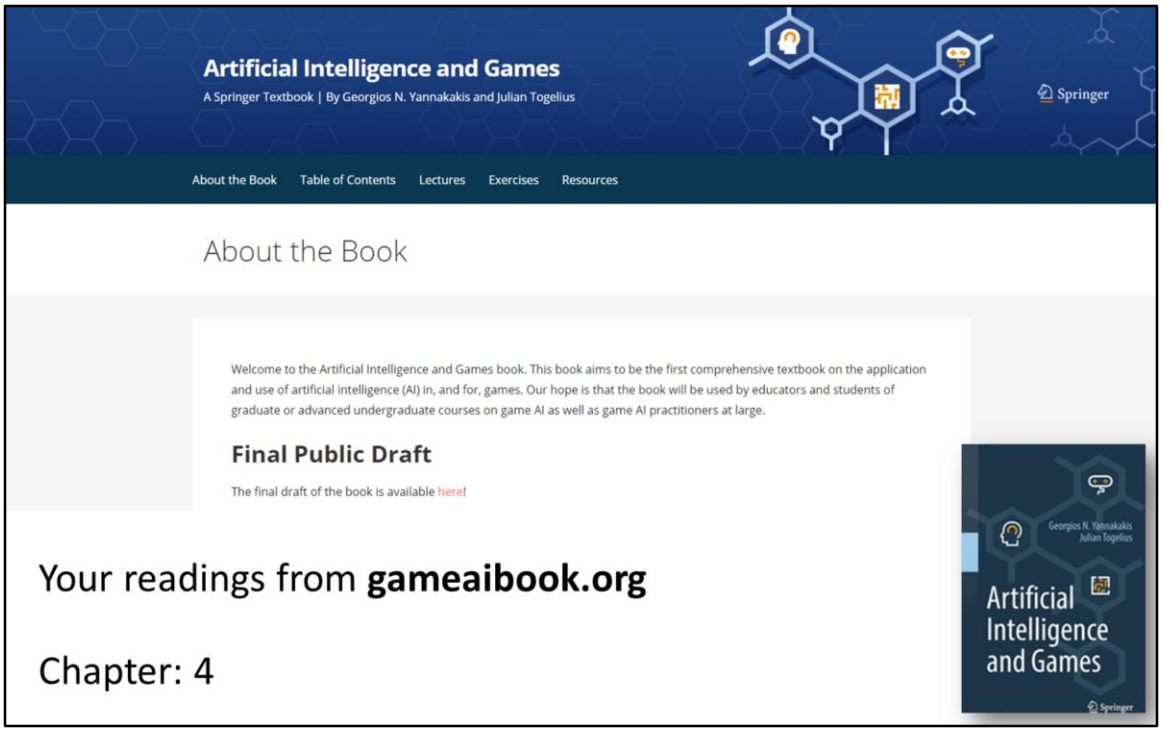


G. N. Yannakakis and J. Togelius, "**Artificial Intelligence and Games**," Springer Nature, 2018.

As a reminder: "Generate Content" is identified as one of the three major roles of AI in games in this book.



The focus of this slide deck is on the role of AI for generating content



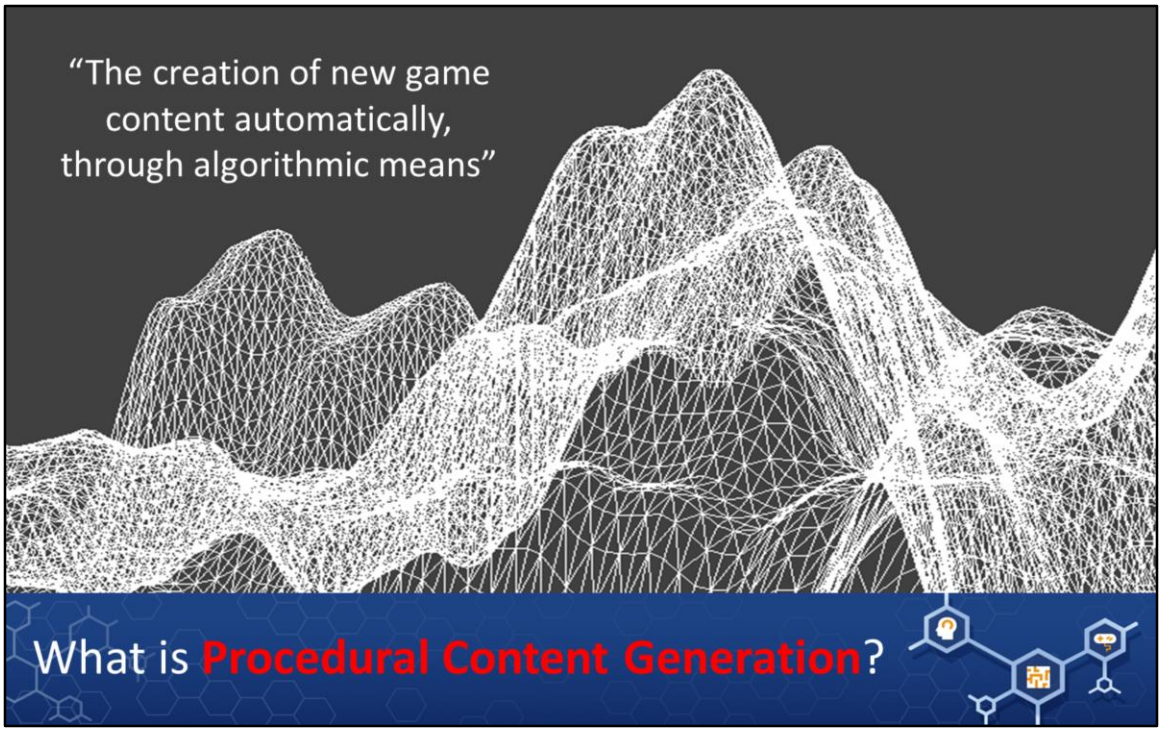
Your readings from **gameaibook.org**

Chapter: 4

Overview

- Procedural Content Generation (PCG)
  - What is it?
  - Why we need it?
- Constructive Approaches
- Search-Based PCG
- Machine Learning PCG
- Mixed-Initiative PCG
- Experience-driven PCG

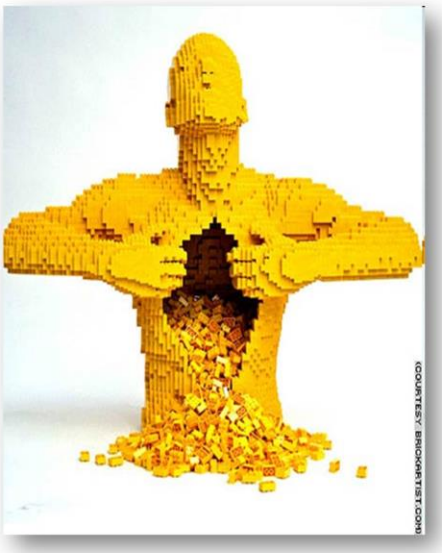
The overview of the slide-deck



Simply put, PCG refers to methods for generating game content either autonomously or with only limited human input.

What is Game **Content**?

- Content can be:
  - NPC behavior (aspects)
  - Quest/story/narrative
  - Camera profiles
  - Audiovisual settings
  - Levels/maps/tracks
  - Items
  - Game mechanics
  - Reward schedules
  - …
  - **Everything** together?
- Content **is** the game context
- Content has differing quality (ability to player experience elicitation)



Game content is that which is contained in a game: levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles, characters, etc.

Content Types

- Maps
- Levels
- Weapons / items
- Game rules
- Stories
- …
- Everything together?

[see introduction of Chapter 4 for more details]

Procedural content generation (PCG) is an area of game AI that has seen an explosive growth of interest.

Games that incorporate some procedurally generated content have existed since the early 1980s. Some Examples:

- Dungeon crawler *Rogue* (1980) – level generation [top right]
- Space trading simulator *Elite* (1984) [top left image]
- *.kkrieger* is a FPS made entirely with 96 kb of memory [top right image]
- *SpeeTree* (bottom right): vegetation and tree generation software/dev tool
- *Minecraft* – pseudo-random level generation [top middle]
- Spelunky - level generation [bottom middle]
- Civilization – level generation [bottom left]
- Borderlands – weapon generation [middle left]

Hundreds of commercial games feature PCG by now… e.g. according to pcg.wikidot

PCG in Academic

- An IEEE Task Force
- A book: pcgbook.com
- A number of PCG dedicated sessions/workshops since 2010: FDG, IEEE CIG, AIIDE, AAAI, IJCAI, ….
- A paradigm shift:
  - ML-based PCG
  - Mixed-Initiative PCG
  - Computational Game Creativity
  - Experience-driven PCG

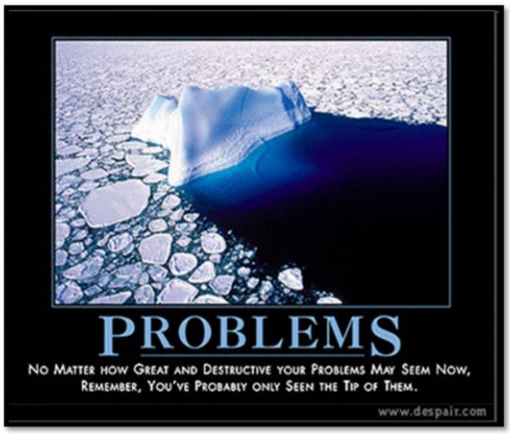
[see introduction of Chapter 4 for more details]

Research interest in academia has really picked up within the second half of the last decade.

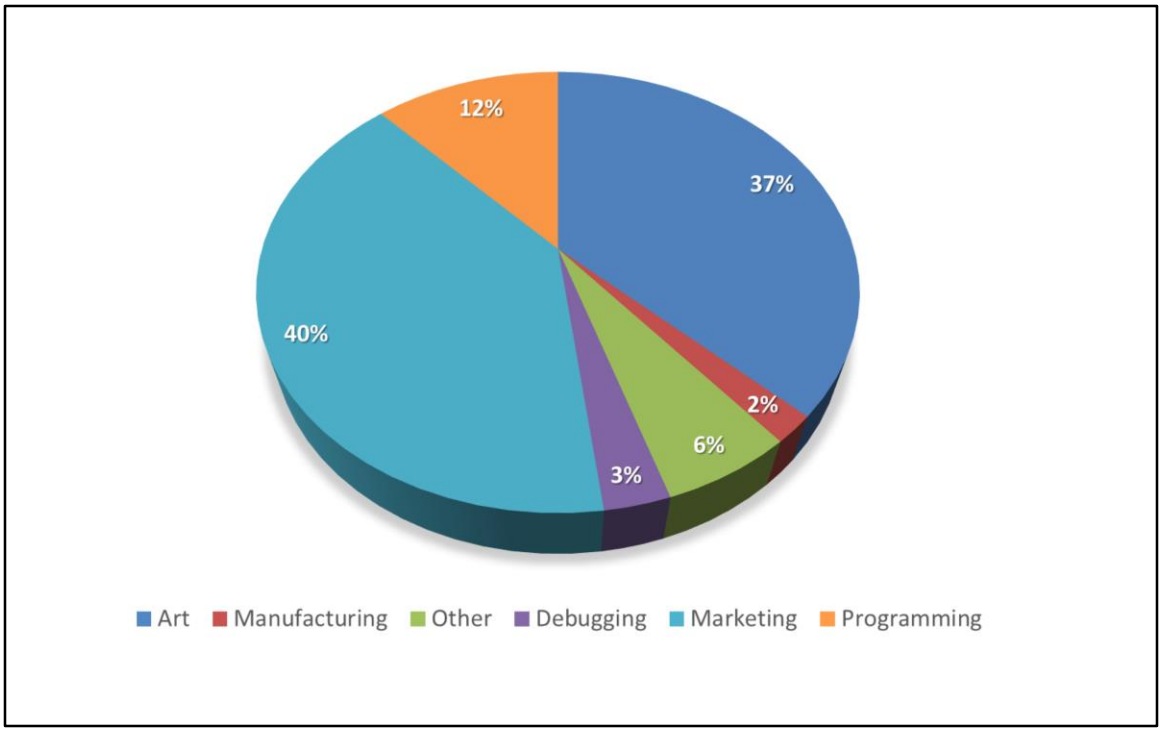
What is the Problem?

- Content costs! (money and time)
- Games end when finished
- Game worlds have bounds
- Human imagination (creativity) is limited
- Designer is not always present

In sum, there is **content shortage**



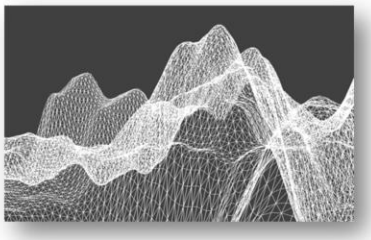
[see Section 4.1 for more details]



[see Section 4.1 for more details]

The Figure illustrates a cost breakdown of an average AAA game and showcases the dominance of artwork and marketing in that process. Art, programming, and debugging constitute around 50% of the cost of an AAA game. Essentially, PCG can assist in the processes of art and content production, thus directly contributing to the reduction of around 40% of a game's cost.

What Can PCG Do?



- Can we drastically cut game development costs by creating content automatically from designers' intentions?
- Can we create games that adapt their game worlds to the preferences of the player?
- Can we create endless games?
- Can the computer circumvent or augment limited human creativity and create new types of games?
- Can we understand game design through formalising the design process?

[see Section 4.1 for more details]

What Are the Trade-offs

- *Speed*
  Real-time? Or design-time?
- *Reliability*
  Catastrophic failures break gameplay
- *Controllability*
  Allow specification of constraints and goals
- *Diversity*
  Content looks like variations on a theme
- *Creativity*
  Content looks “computer-generated”

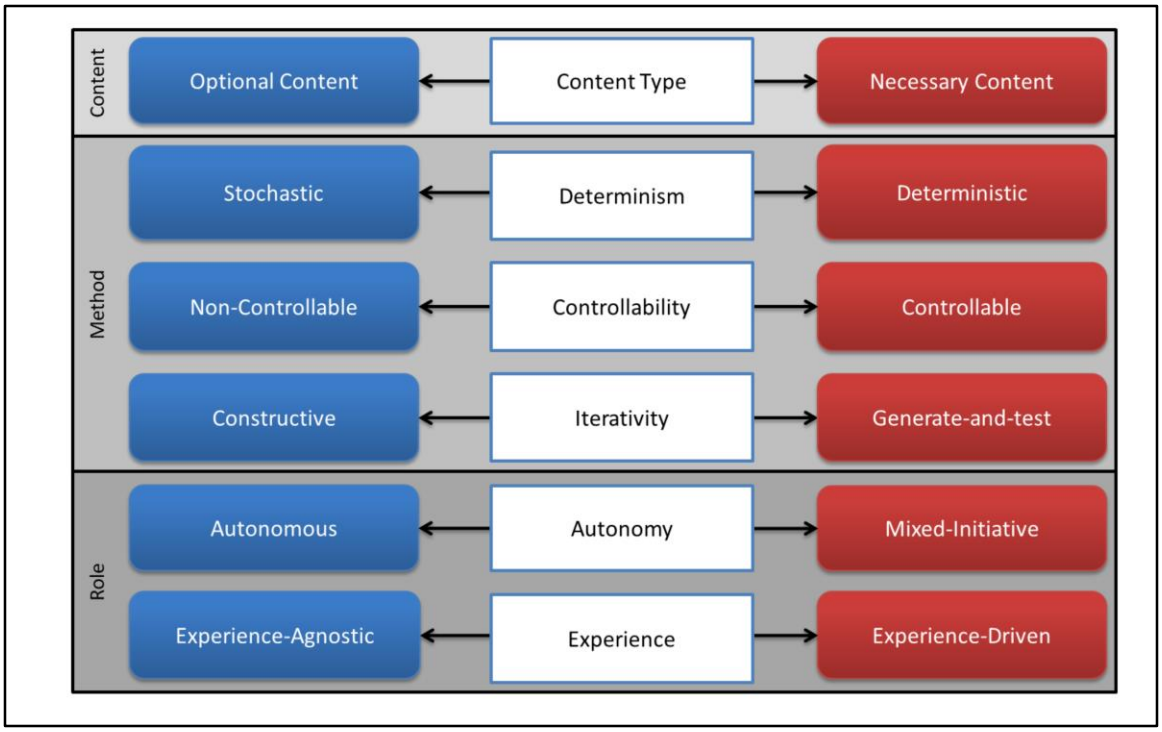
[see Section 4.1 for more details]

The challenges and key questions of PCG. Some questions have been addressed to a good degree by now; some questions generate more questions than answers

A PCG Taxonomy

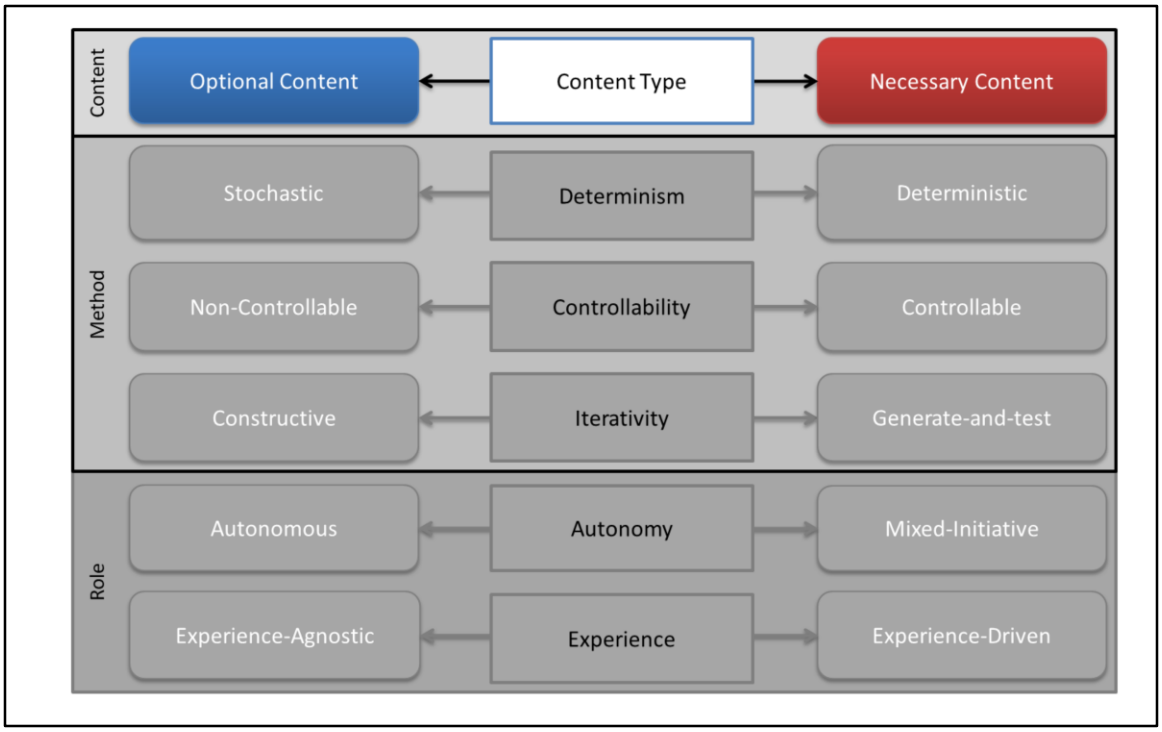
[see Section 4.2 for more details]

With the variety of content generation problems, methods and approaches that are available, it helps to have a structure that can highlight the differences and similarities between them.



[see Section 4.2 for more details]

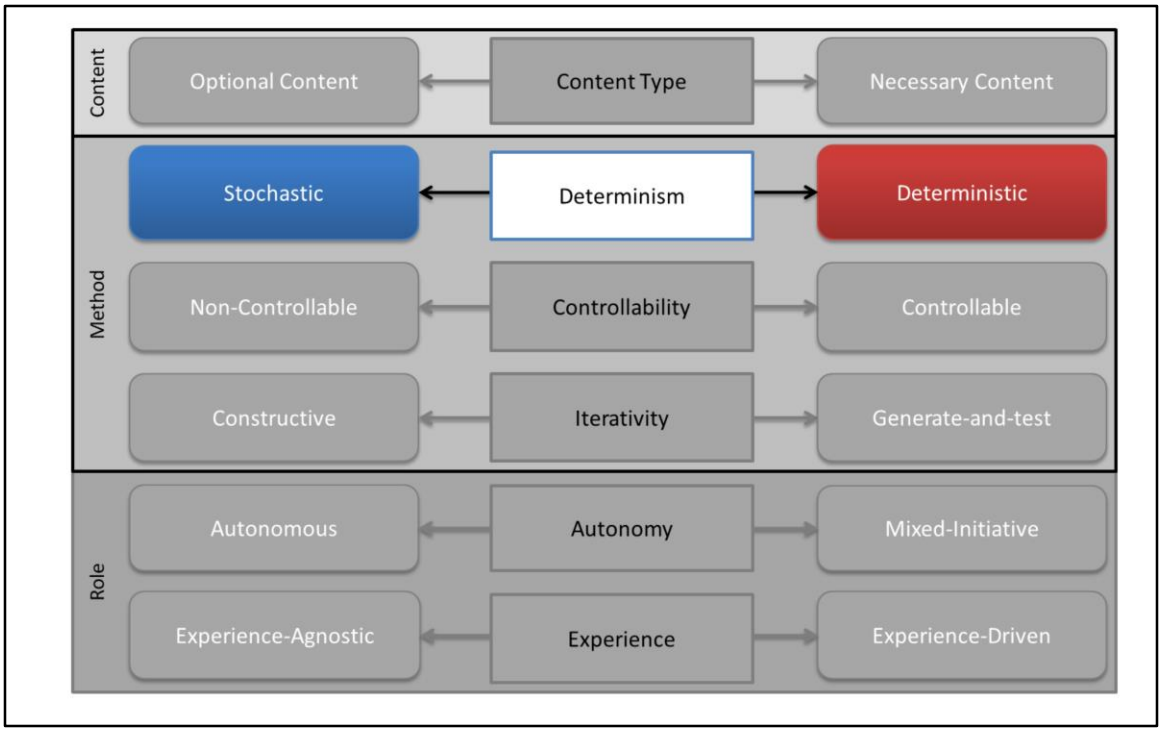
The taxonomy consists of a number of dimensions, where an individual method or solution should usually be thought of as lying somewhere on a continuum between the ends of that dimension. Beyond the taxonomy for the **content types** and the **properties** of the PCG methods we provide an outline of the **roles** a PCG algorithm can take



[see Section 4.2.1.1 for more details]

PCG can be used to generate **necessary** game content that is required for the completion of a level, or it can be used to generate **optional** content that can be discarded or exchanged for other content.

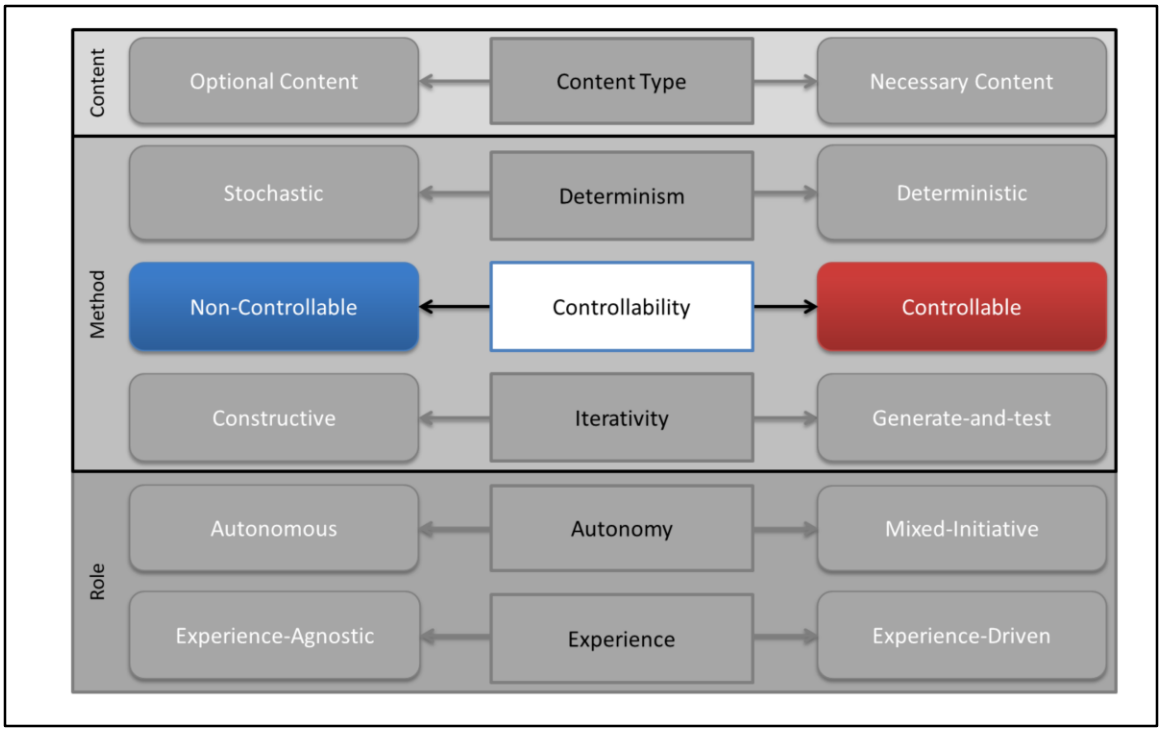
The main distinguishing feature between necessary and optional content is that **necessary content should always be correc**t while this condition does not hold for optional content.



[see Section 4.2.2.1 for more details]

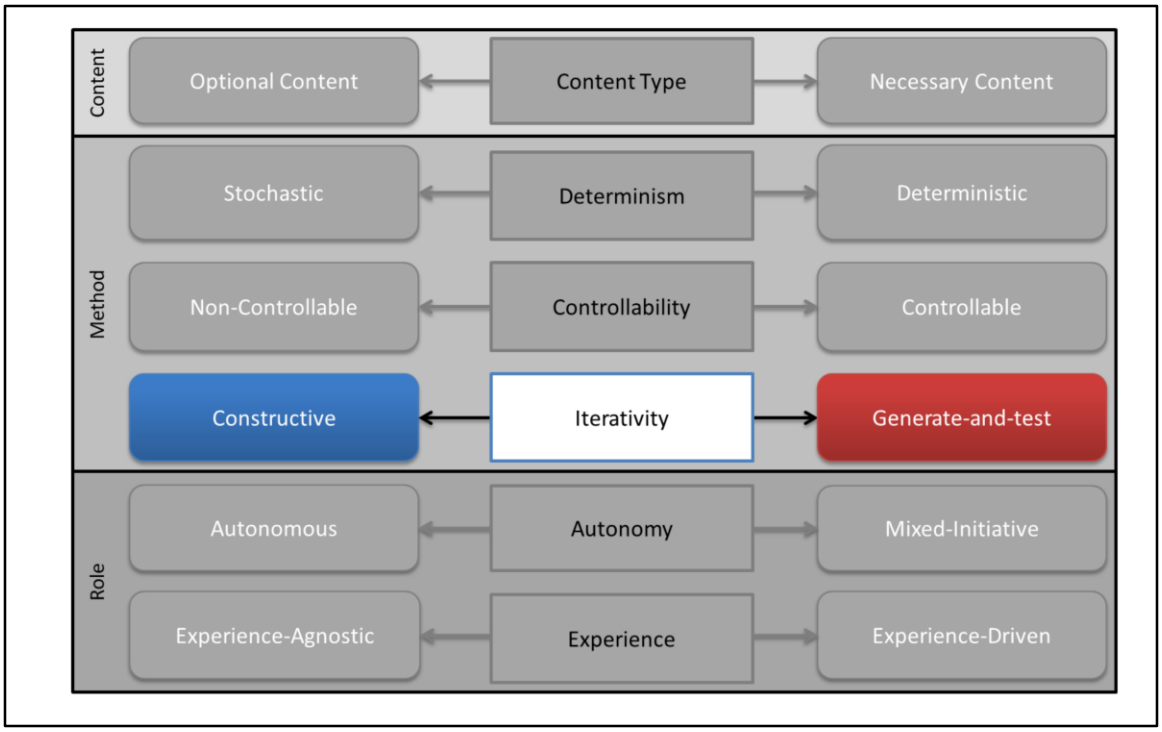
PCG algorithms can be classified according to a number of properties such as their level of controllability, determinism and iterativity.

Determinism: amount of randomness in content generation. Stochasticity allows an algorithm to offer great **variation**, necessary for many PCG tasks, at the cost of **controllability**. While content diversity and expressivity are desired properties of generators, the effect of randomness on the final outcomes can only be observed and controlled after the fact. Completely deterministic PCG algorithms, on the other hand, can be seen as a form of data compression (e.g. .kkrieger)



[see Section 4.2.2.2 for more details]

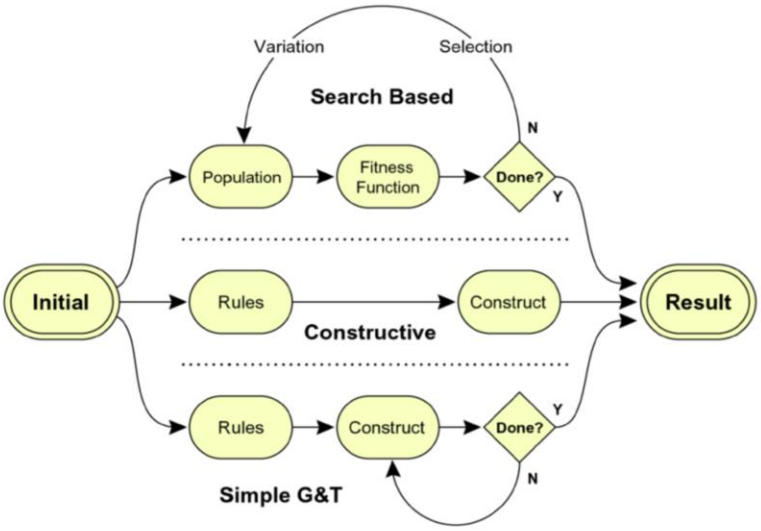
The generation of content can be controlled in different ways. The use of a random seed is one way to gain control over the generation space; another way is to use a set of parameters that control the content generation along a number of dimensions.



[see Section 4.2.2.3 for more details]

A final distinction may be made between algorithms that can be called **constructive** and those that can be described as **generate-and-test**.

Three Main PCG Approaches



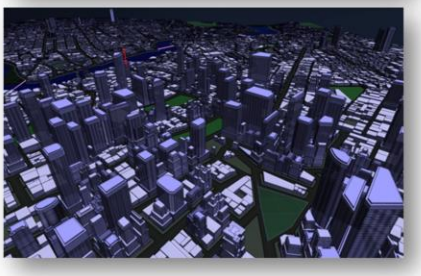
- A constructive algorithm generates the content once, and is done with it; however, it needs to make sure that the content is correct or at least "good enough" as it is being constructed (e.g. fractals or cellular automata).

- A generate-and-test (G&T) algorithm, instead, incorporates both a generate and a test mechanism. After a candidate content instance is generated, it is tested according to some criteria (e.g., is there a path between the entrance and exit of the dungeon). If the test fails, all or some of the candidate content is discarded and regenerated, and this process continues until the content is good enough.

- A popular PCG framework that builds upon the generate-and-test paradigm is the search-based approach.

“Classical” PCG Approaches

- Mostly constructive, stochastic, non-controllable, random seed, online
- Mostly optional (non-vital) content such as decoration and redundant items
- Interesting examples from classic games:
  - Elite (deterministic, offline)
  - Rogue (online, crucial content)
- Techniques: L-trees, Fractals, …



For reference: these are the properties of traditional PCG Algorithms we meet primarily at the game industry; and in game AI academia to a lesser degree.

[see Section 4.3 for more details]

There are many different algorithmic approaches to generating content for games.

PCG Methods

- Cellular automata
- Solver-based
- Grammar-based
- Search-based
- Machine learning
- Noise and fractals
- Ad-hoc constructive methods

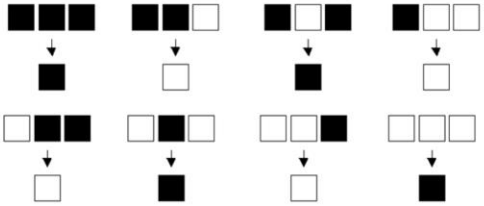
While many of the PCG methods are commonly thought of as AI methods, others are drawn from graphics, theoretical computer science or even biology. The various methods differ also in what **types of content** they are suitable to generate. The slide lists a number of PCG methods that are consider important – let us look at them in more detail starting with cellular automata

N.B. This slide deck does not cover “noise and fractals” and “ad-hoc constructive methods” in detail. Details about such approaches can be found in section 4.3.

Cellular Automata

[see Section 4.3.4 for more details]

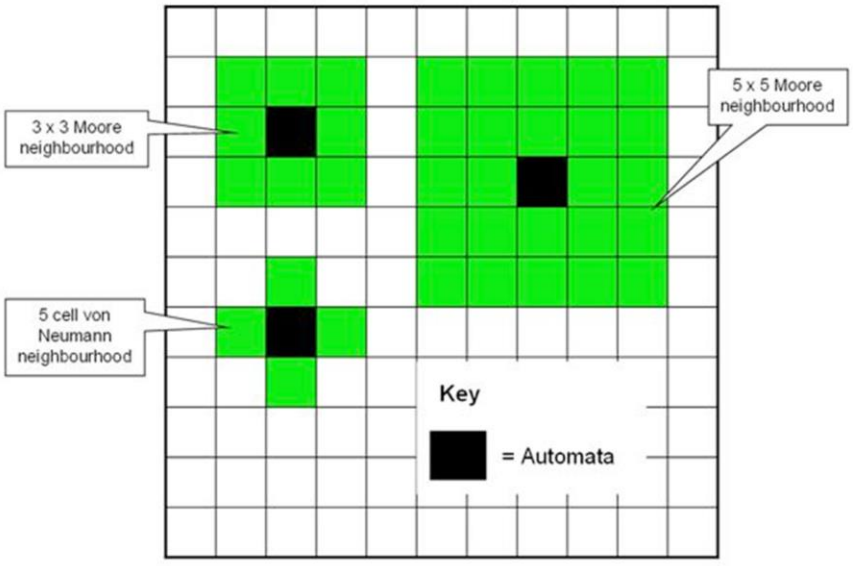
Cellular Automata



- Computational paradigm based on local interaction
- Used in artificial life and complexity studies
- The value of each cell in iteration *n+1* is based on the value of neighboring cells in iteration *n* and some rule

[see Section 4.3.4 for more details]

2D Cellular Automata



An Example

Cellular automata for real-time generation of infinite cave levels

Lawrence Johnson, Georgios N. Yannakakis and Julian Togelius

FDG PCG Workshop 2010

This…

A CA-based algorithm for generating infinite 2D caves

- simple
- real-time
- looks good
- *somewhat* controllable

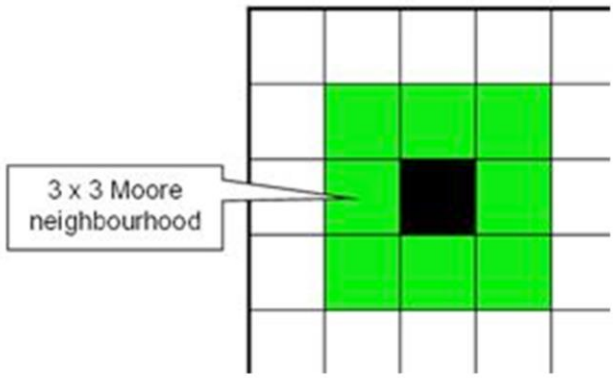
The Motivation

- *Cave Crawler* : a cooperative *abusive* dungeon crawler
- Never ends – therefore needs to produce infinite caves...

CA Cave Generation

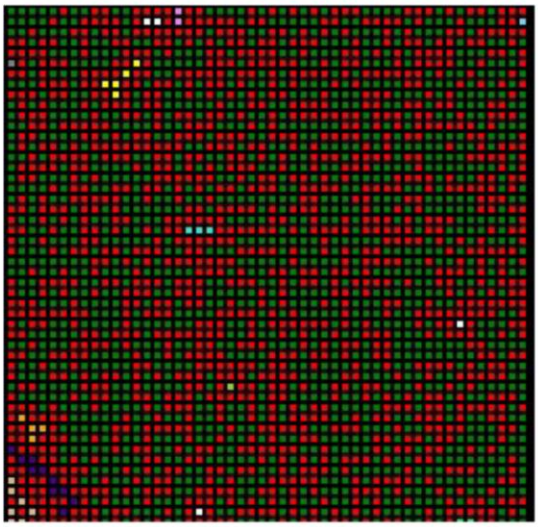
- Start with a square grid (e.g. 50*50) – all floor
- Randomly switch a proportion of cells from floor to rock
- Run a CA *n* times, where each cell is set to:
  - Rock: if at least *T* neighbors are rock
  - Floor: otherwise
- Fill in the interior of rock formations

Core CA Mechanic

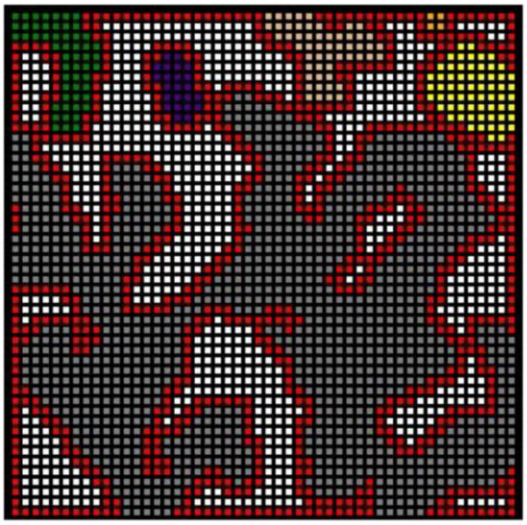


CA Parameters

- *r* : initial proportion of rock cells (0.5)
- *n* : CA iterations (4)
- *T* : neighborhood value threshold that defines a rock (5)
- *M* : Moore neighborhood size (1)



(a) Random map



(b) CA map

Adjacent Rooms

- The infinite cave needs to be contiguous - and you need to be able to turn back! (Visited rooms stored as random seeds)
- Generate all four neighbors of a new room
- Dig tunnels from the central room to the new rooms at the shortest points
- Run the CA *m* times (2) on *all five rooms together* to smooth out edges

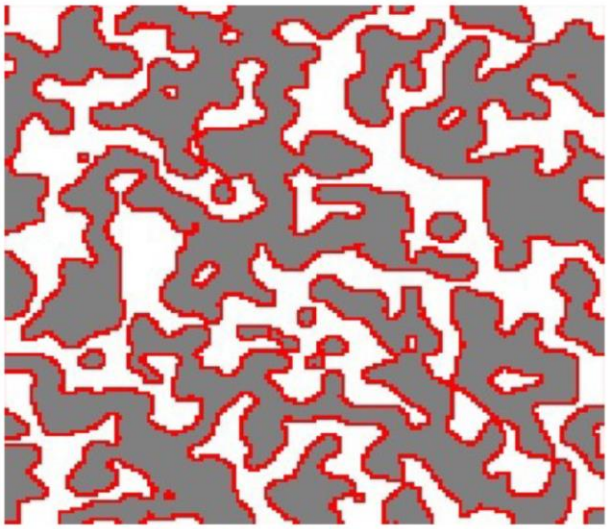
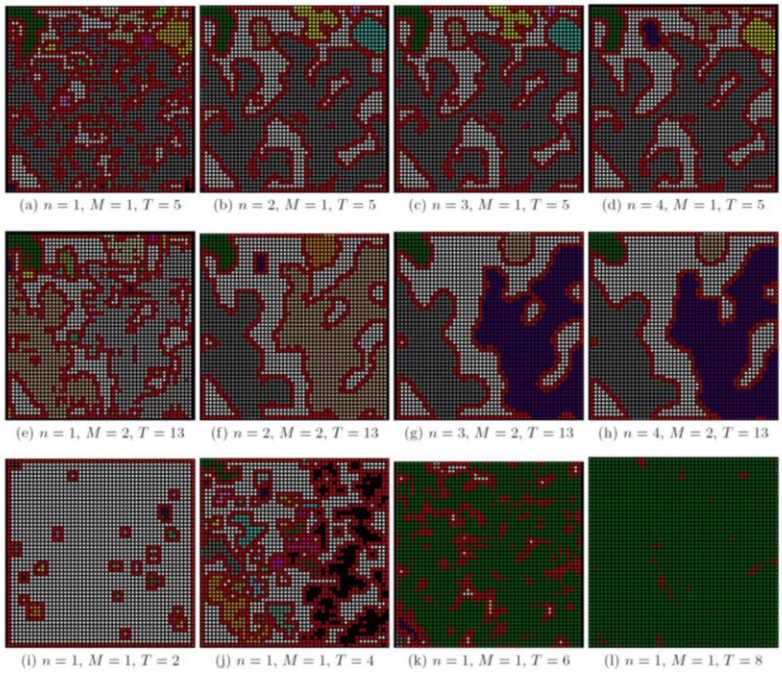


Figure 3: A 3×3 base grid map generated with CA. Rock and wall cells are represented by red and white color respectively. Grey areas represent floor. ($M = 2; T = 13; n = 4; r = 50\%$)

Controllable?

Parameters can be varied…
…but what do they mean?

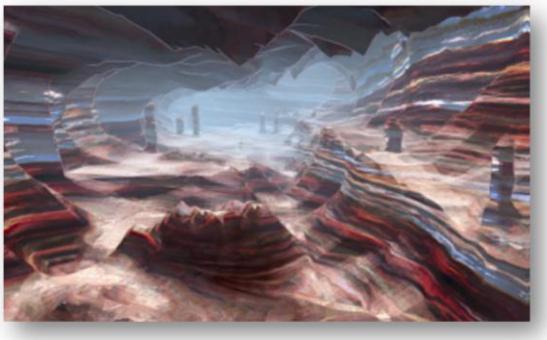


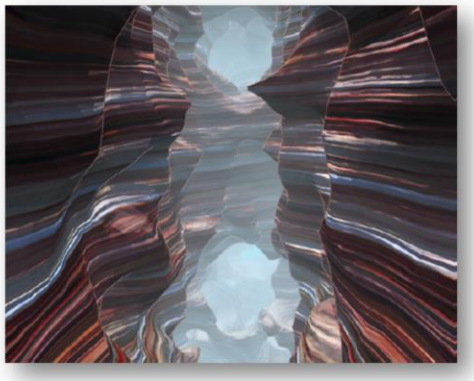
(a) $n = 1$, $M = 1$, $T = 5$ (b) $n = 2$, $M = 1$, $T = 5$ (c) $n = 3$, $M = 1$, $T = 5$ (d) $n = 4$, $M = 1$, $T = 5$

(e) $n = 1$, $M = 2$, $T = 13$ (f) $n = 2$, $M = 2$, $T = 13$ (g) $n = 3$, $M = 2$, $T = 13$ (h) $n = 4$, $M = 2$, $T = 13$

(i) $n = 1$, $M = 1$, $T = 2$ (j) $n = 1$, $M = 1$, $T = 4$ (k) $n = 1$, $M = 1$, $T = 6$ (l) $n = 1$, $M = 1$, $T = 8$

3D L-systems + CA





Solver-based Methods

[see Section 4.3.2 for more details]

Solver-based Methods

- Instead of using evolutionary algorithms, use constraint solvers and specify constraints
- Example: Answer Set Programming (ASP)
  - Declarative programming for search problems
  - Based on logic programming (syntax very similar to Prolog)
  - Finding an answer set is equivalent to solving a satisfiability problem

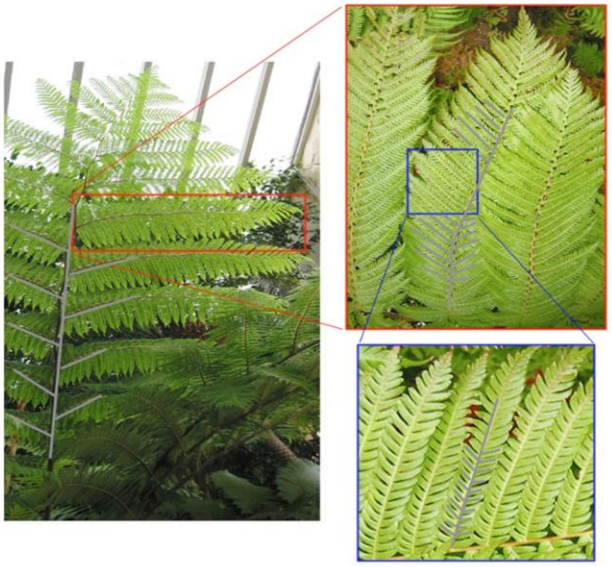
Grammars

[see Section 4.3.3 for more details]

Grammars

- Basic computer science concept, used in theory of computation
- Define a grammar and an alphabet and then watch an axiom unfold into ever more complex strings
- Commonly used for generating e.g. plants

Self-Similarity



Self-Similarity

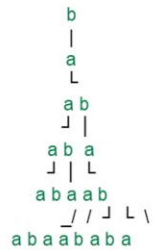
- Nature has obviously thought out some clever way of representing complex organisms using a compact description...
- ...permitting individual variation...
- ...why is this relevant for us?

L-systems

- Introduced by Aristid Lindenmeyer 1968, to model plant development
- Creates strings (text) from an *alphabet* based on a *grammar* and an *axiom*
- Closely related to Chomsky grammars (but productions carried out in parallel, not sequentially)

An Example L-system

- Alphabet: {a, b}
- Production rules (grammar):
  a>ab
  b>a
- Axiom: b



Example of a derivation in a DOL-System

Types of L-systems

Context

- **Context-free**: production rules refer only to an individual symbol
- **Context-sensitive**: productions can depend on the symbol's neighbors

Determinism

- **Deterministic**: there is exactly one production for each symbol
- **Non-deterministic**: several productions for a symbol

A Graphical Interpretation of L-systems

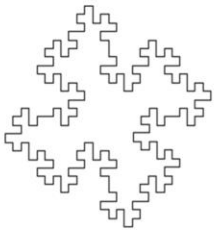
- Invented/popularized by Prusinkiewicz in 1986
- Core idea: interpret generated strings as instructions for a turtle in **turtle graphics**
- Read the string from left to right, changing the state of the turtle (x, y, heading)

Example Graphical Systems

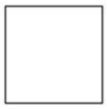
- Alphabet: {F, f, +, -}
- F: move the turtle forward (drawing a line)
- f: move the turtle forward (don't draw)
- +/-: turn right/left (by some angle)

Graphical L-system

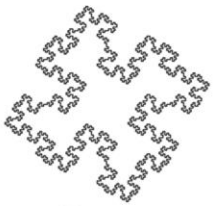
- Axiom: F+F+F+F
- Grammar:
  F>F+F-F-FF+F+F-F
- Turning angle: 90^o



n=1



n=0



n=2

Graphical L-systems

What's the limit of these systems?

Bracketed L-systems

- Alphabet: {F, f, +, -, [,]}
- [: push the current state (x, y, heading of the turtle) onto a pushdown stack
-]: pop the current state of the turtle and *move* the turtle there *without drawing*
- Enables branching structures!

Bracketed L-systems

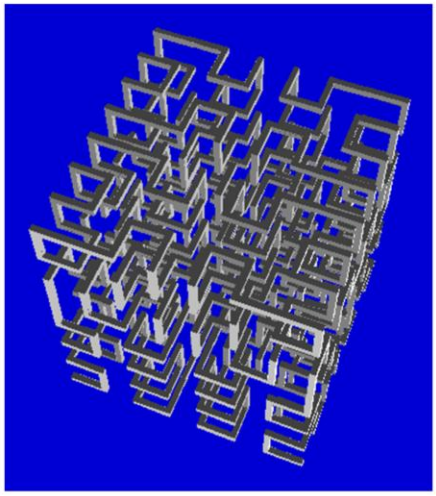
Axiom: F
Grammar: F>F[-F]F[+F][F]
Turning angle: 30^o

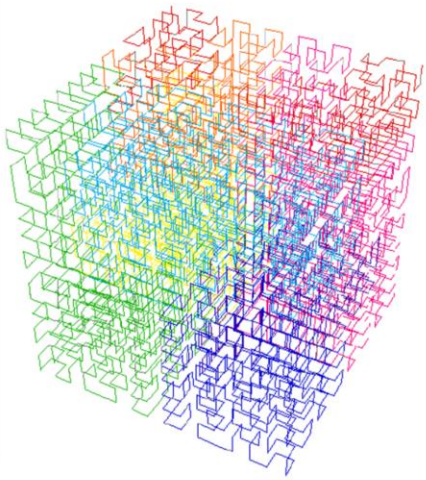
n=1,…,5

3D Graphics

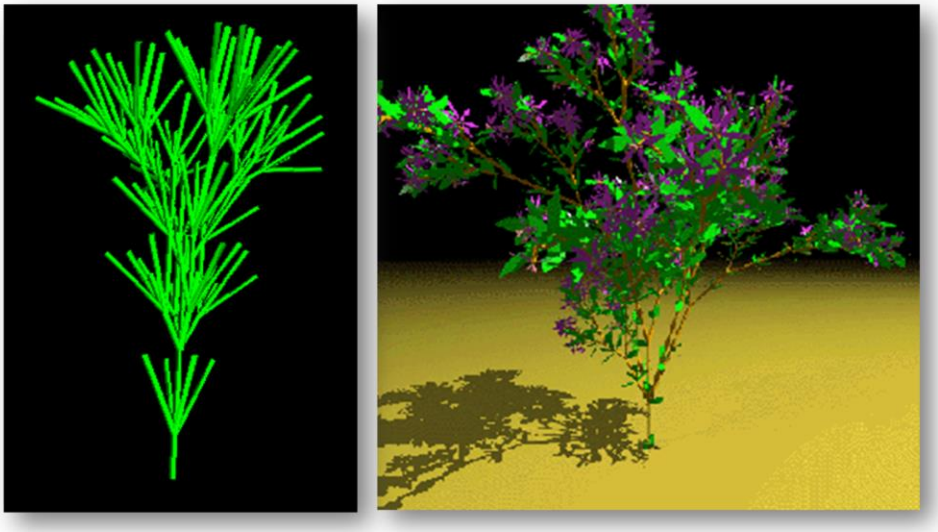
- Turtle graphics L-system interpretation can be extended to 3D space:
- Represent state as x, y, z and pitch, roll, yaw
- +, -: turn (yaw) left/right
- &, ^: pitch down/up
- \, /: roll left/right (counterclockwise/clockwise)

3D Interpretation of L-systems



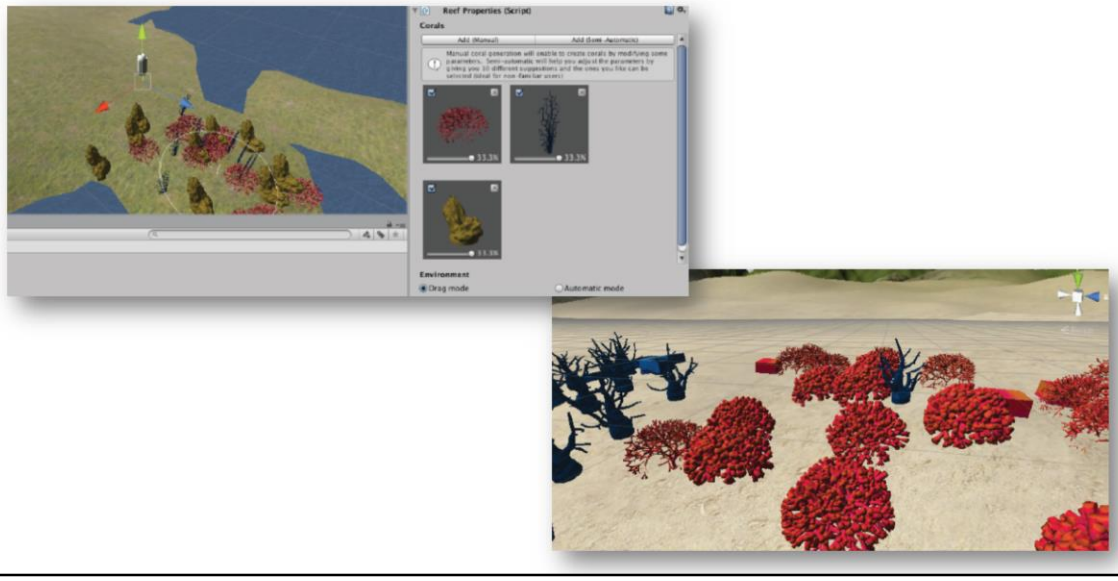


3D Interpretation of Bracketed L-systems

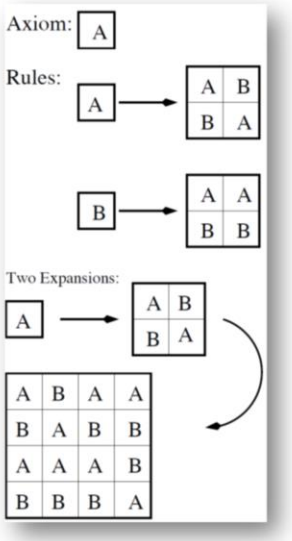


Coralize: 3D Corals in Unity

Abela, R., Liapis, A. and Yannakakis, G.N., 2015. **A constructive approach for the generation of underwater environments**. In Proceedings of the FDG workshop on Procedural Content Generation in Games.

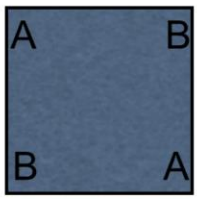


2D L-systems


Axiom:
A
Rules:
A
A B
B A
B
A A
B B
Two Expansions:
A
A B
B A
A B A A
B A B B
A A A B
B B B A

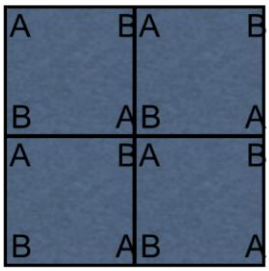
Terrain Interpretation of 2D L-systems

- Each group of four letters is interpreted as instructions for lowering or raising the corners of a square
- E.g. A=+0.5, B=-0.5

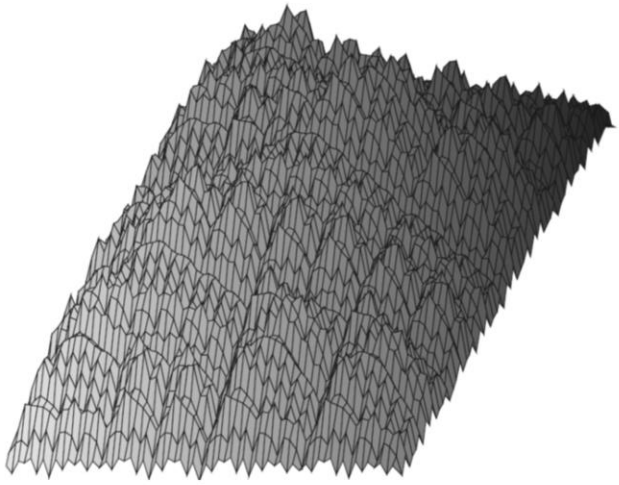


Terrain Interpretation of 2D L-systems

- In next iteration, the 2D L-system is rewritten once, and each square is divided into two
- “Doubling the resolution”



Terrain Interpretation of 2D L-systems



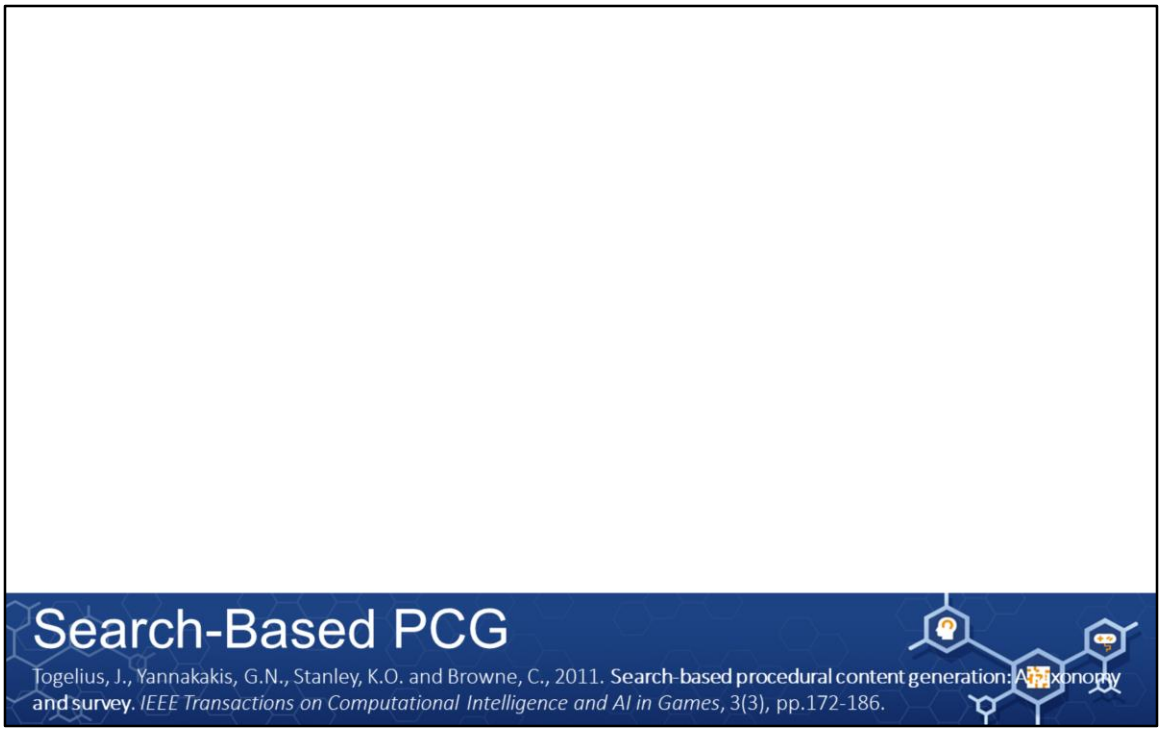
Six rewritings of A>ABBA, B>AABB

Grammars for Adventure Level Design

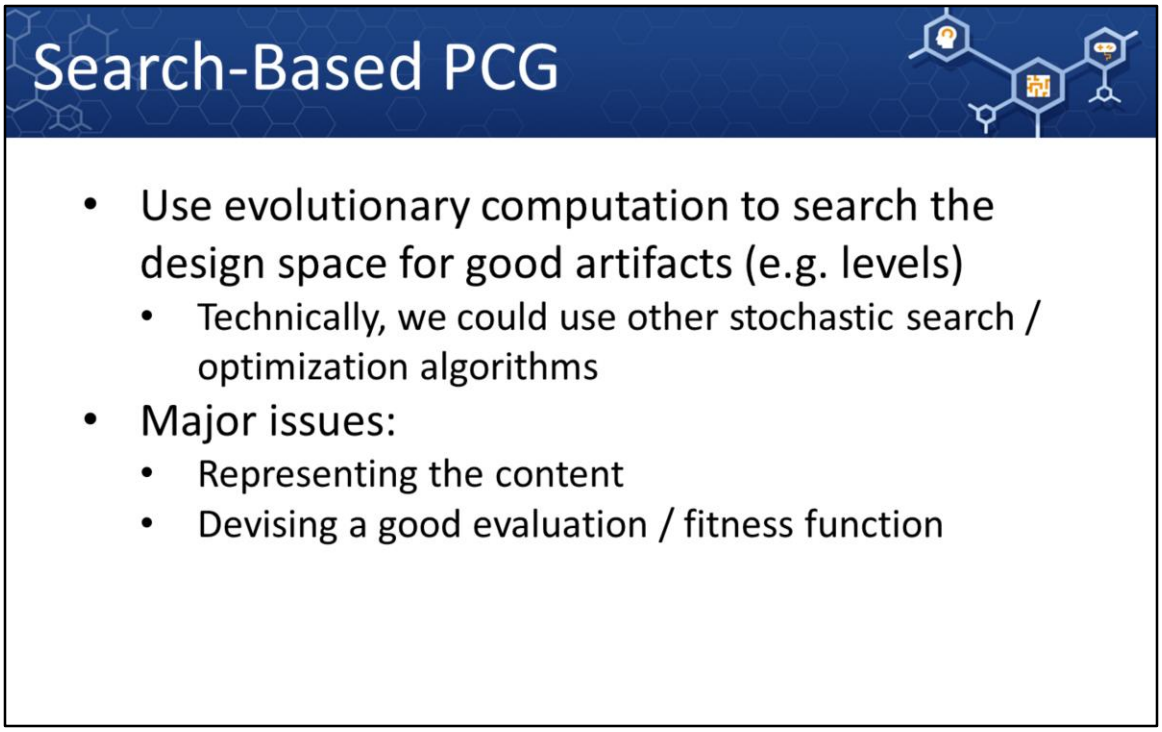
1. Dungeon → Obstacle + treasure
2. Obstacle → key + Obstacle + lock + Obstacle
3. Obstacle → monster + Obstacle
4. Obstacle → room

could give...

key + monster + room + lock + monster + room + treasure
key + monster + key + room + lock + monster + room + lock + room + treasure
room + treasure
monster + monster + monster + monster + room + treasure



[see Section 4.3.1 for more details]



[see Section 4.3.1 for more details]

- The search-based approach to PCG has been intensively investigated in academic PCG research in recent years.

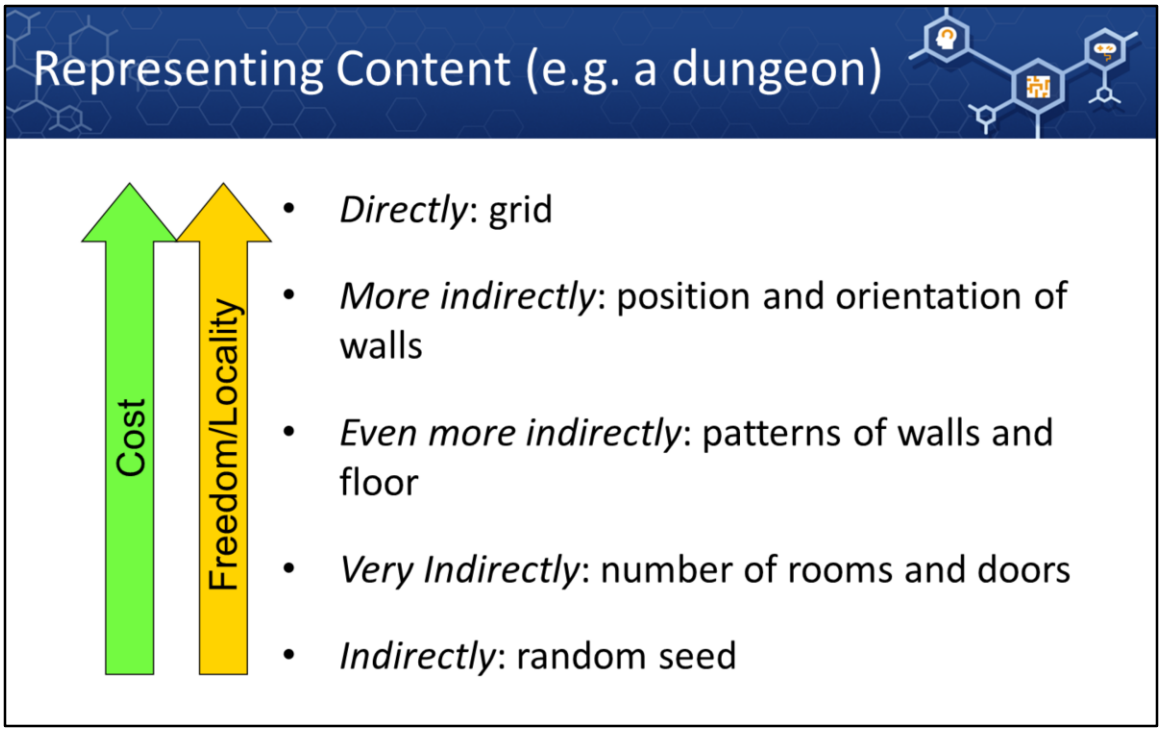
- In search-based procedural content generation, an evolutionary algorithm or some other stochastic search or optimization algorithm is used to search for content with the desired qualities.

- The basic metaphor is that of design as a search process: a good enough solution to the design problem exists within some space of solutions, and if we keep iterating and tweaking one or many possible solutions, keeping those changes which make the solution(s) better and discarding those that are harmful, we will eventually arrive at the desired solution.

The Algorithm

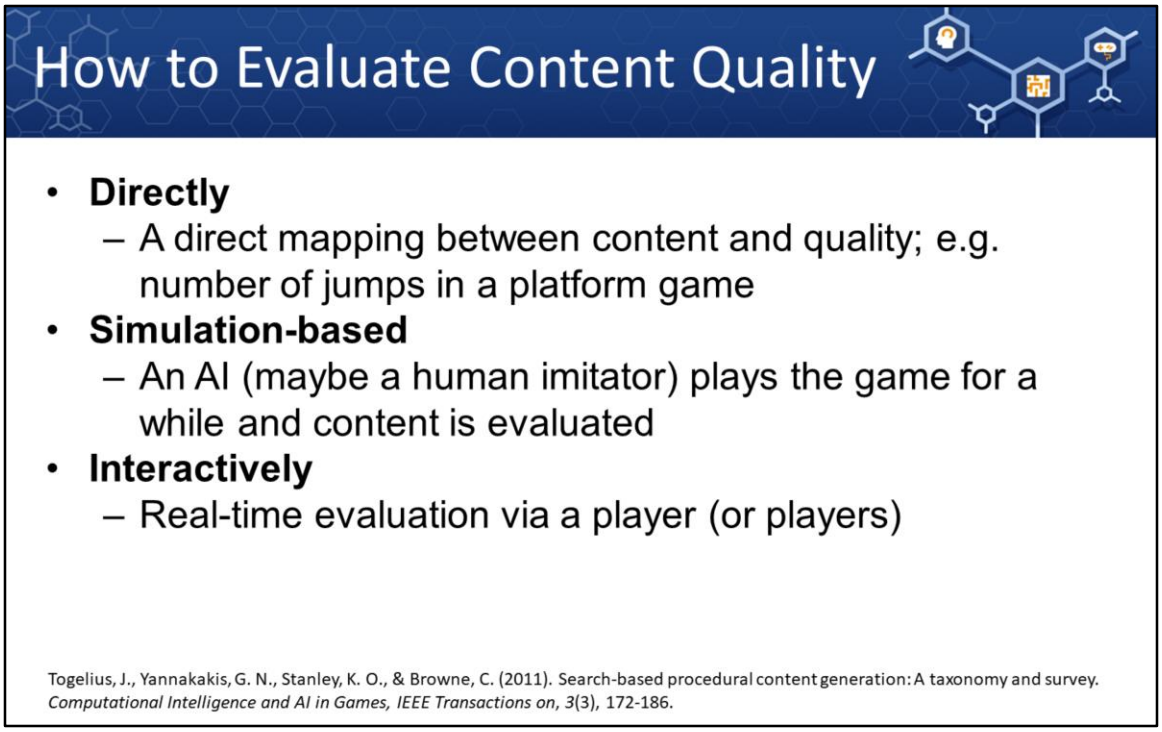
- Lots of different types of evolutionary algorithms: Genetic Algorithms, Evolution Strategies, Evolutionary Programming
- And evolution-like algorithms: Particle Swarm Optimization, Differential Evolution
- Keep It Simple, Stupid!
  - Often, simple μ+λ ES with no crossover and no self-adaptation works well enough

The first core component of the search-based approach is the **search algorithm** per se. This is the "engine" of a search-based method. Often relatively simple evolutionary algorithms work well enough, however sometimes there are substantial benefits to using more sophisticated algorithms that take e.g., constraints into account, or that are specialized for a particular content representation.



The second core component of the search-based approach is **content representation**

This is the representation of the artifacts you want to generate, e.g., levels, quests or winged kittens. The content representation could be anything from an array of real numbers to a graph to a string. The content representation defines (and thus also limits) what content can be generated, and determines whether effective search is possible



The third core component of the search-based approach is the design of one or more **evaluation functions**

I.e. This is a function from an artifact (an individual piece of content) to a number indicating the quality of the artifact.

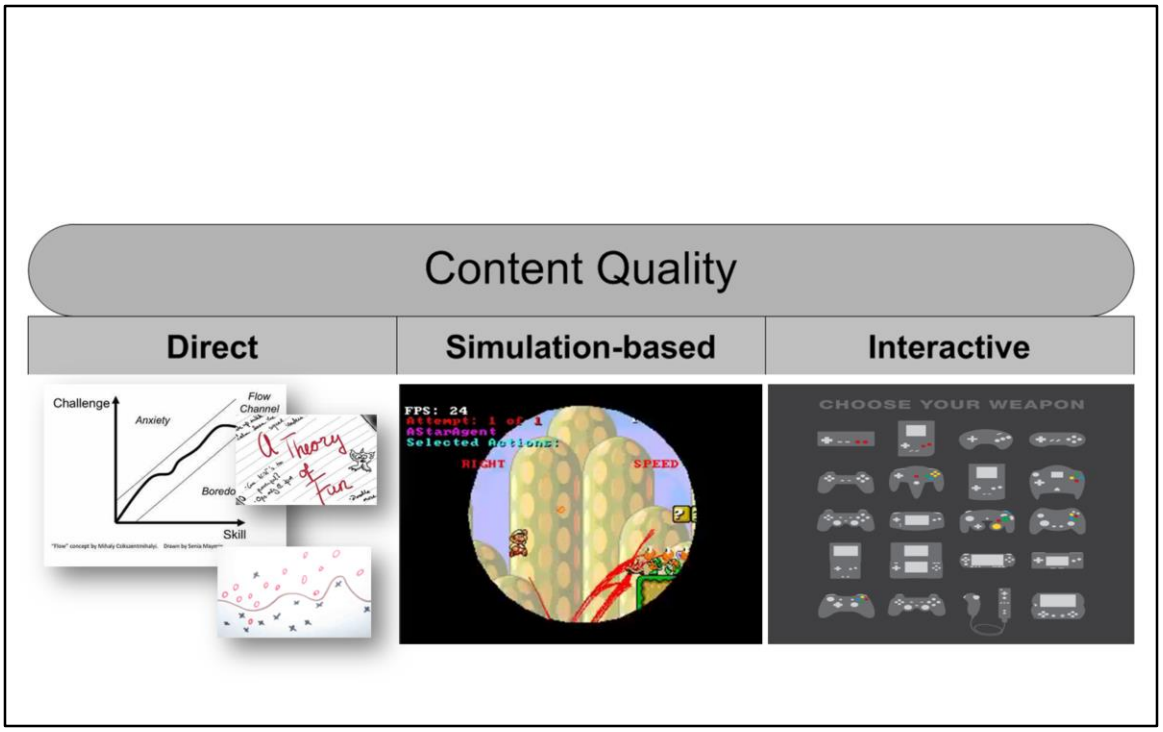
There are three main ways to evaluate content: directly, simulation-based and interactively

How to Evaluate Content Quality

- **Directly**
  - **Theory-driven**: evaluation function is based on a theoretical model – e.g. Koster's theory of fun
  - **Data-driven**: evaluation function is derived via gameplay (or other modalities of) data
- **Simulation-based**
  - **Static**: evaluation function does not change over time
  - **Dynamic**: evaluation function is affected as time goes by
- **Interactively**
  - **Implicit**: game behavior gives value to content (e.g. preference over a weapon)
  - **Explicit**: ask players to score content

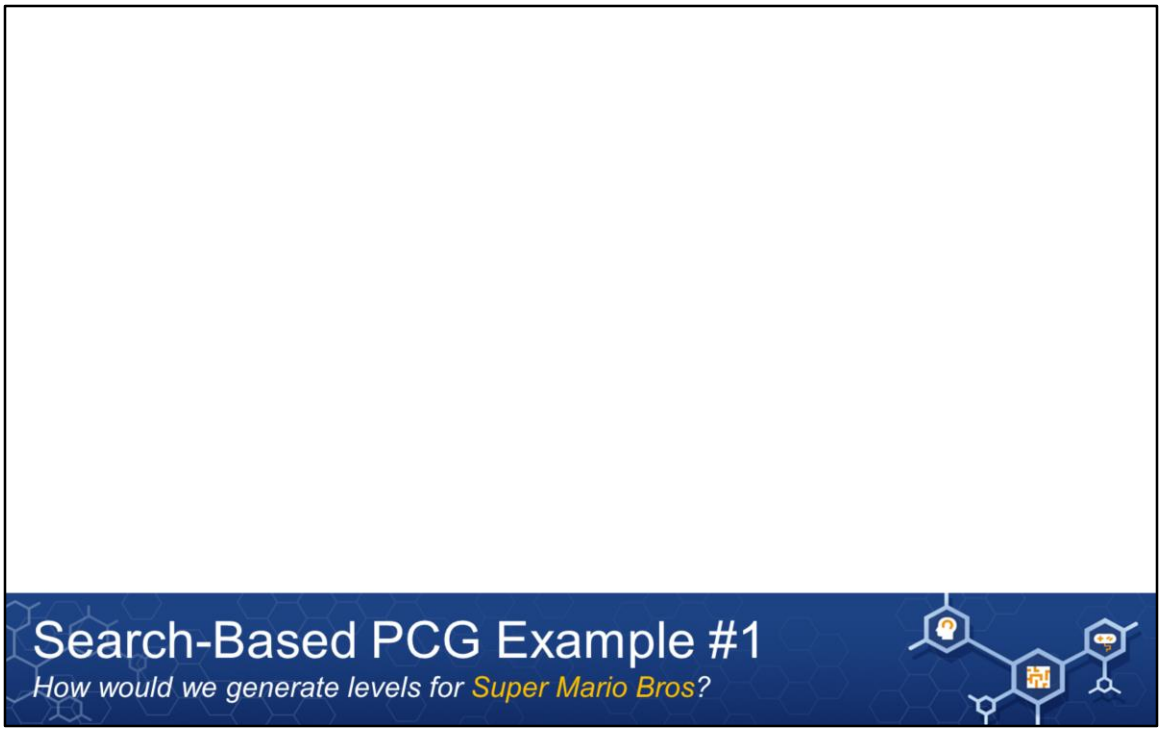
Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *Computational Intelligence and AI in Games, IEEE Transactions on*, *3*(3), 172-186.

The taxonomy of content evaluation expanded. See Section 4.3.1 for more details



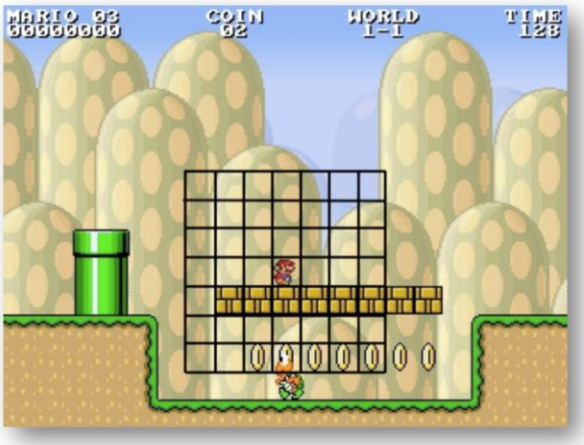
A graphical representation of the different ways an algorithm can evaluate content

- **Directly (**A direct mapping between content and quality; e.g. number of jumps in a platform game)
  - –**Theory-driven**: evaluation function is based on a theoretical model – e.g. Koster's theory of fun)
  - –**Data-driven**: evaluation function is derived via gameplay (or other modalities of) data
- **Simulation-based (**An AI (maybe a human imitator) plays the game for a while and content is evaluated)
  - –**Static**: evaluation function does not change over time
  - –**Dynamic**: evaluation function is affected as time goes by
- **Interactively (**Real-time evaluation via a player (or players))
  - –**Implicit**: game behavior gives value to content (e.g. preference over a weapon)
  - –**Explicit**: ask players to score content



The following example is based on the paper: Steve Dahlskog and Julian Togelius: **Patterns as Objectives for Level Generation**. PCG Workshop 2013

The Mario AI Benchmark

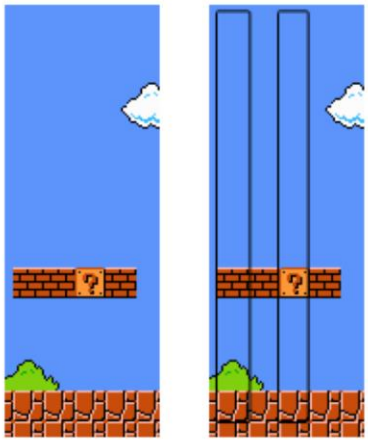


- Reasonably faithful clone of SMB 1/3
- APIs for level generators and AI controllers

Steve Dahlskog and Julian Togelius: **Patterns as Objectives for Level Generation**. PCG Workshop 2013

Representation

- A number of “vertical slices” are identified from the original SMB levels
- Levels are represented as strings, where each character correspond to a pattern

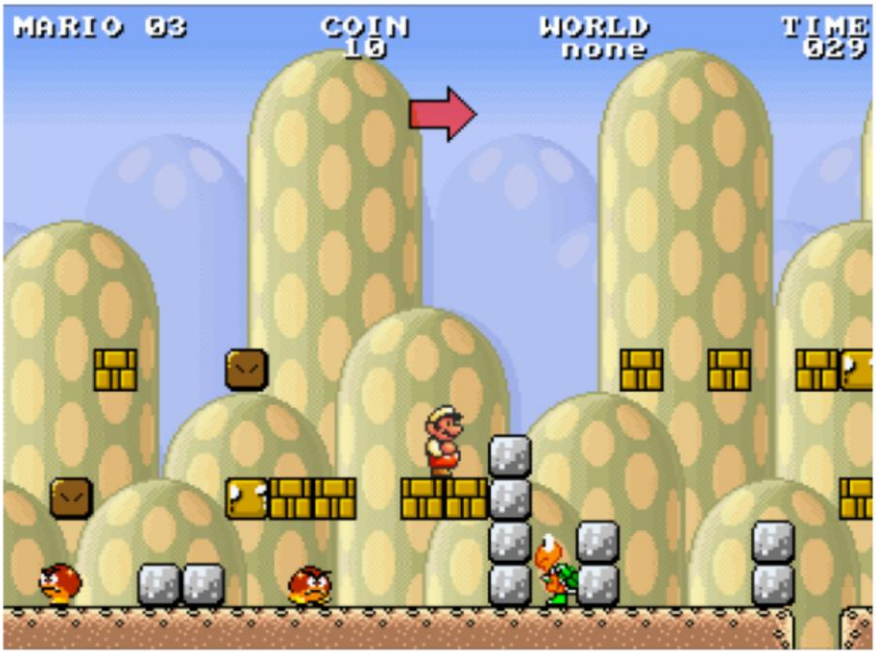


Steve Dahlskog and Julian Togelius: **Patterns as Objectives for Level Generation**. PCG Workshop 2013

Evaluation

- 25 patterns are identified in the original SMB levels
- e.g. enemy hordes, pipe valleys, 3-paths…
- The fitness function counts the number of patterns found in the level

Steve Dahlskog and Julian Togelius: **Patterns as Objectives for Level Generation**. PCG Workshop 2013



Steve Dahlskog and Julian Togelius: **Patterns as Objectives for Level Generation**. PCG Workshop 2013

Fig. 15. FFMacro #98 MC: 6, fitness value: 2332 (highest).

Fig. 16. FFMesoB #6 MC: 0, fitness value: 545.

Fig. 17. FFMesoB #30 MC: 3, fitness value: 409 (lowest).

Fig. 18. FFMesoB #64 MC: 6, fitness value: 2065 (highest).

Steve Dahlskog and Julian Togelius: **Patterns as Objectives for Level Generation**. PCG Workshop 2013

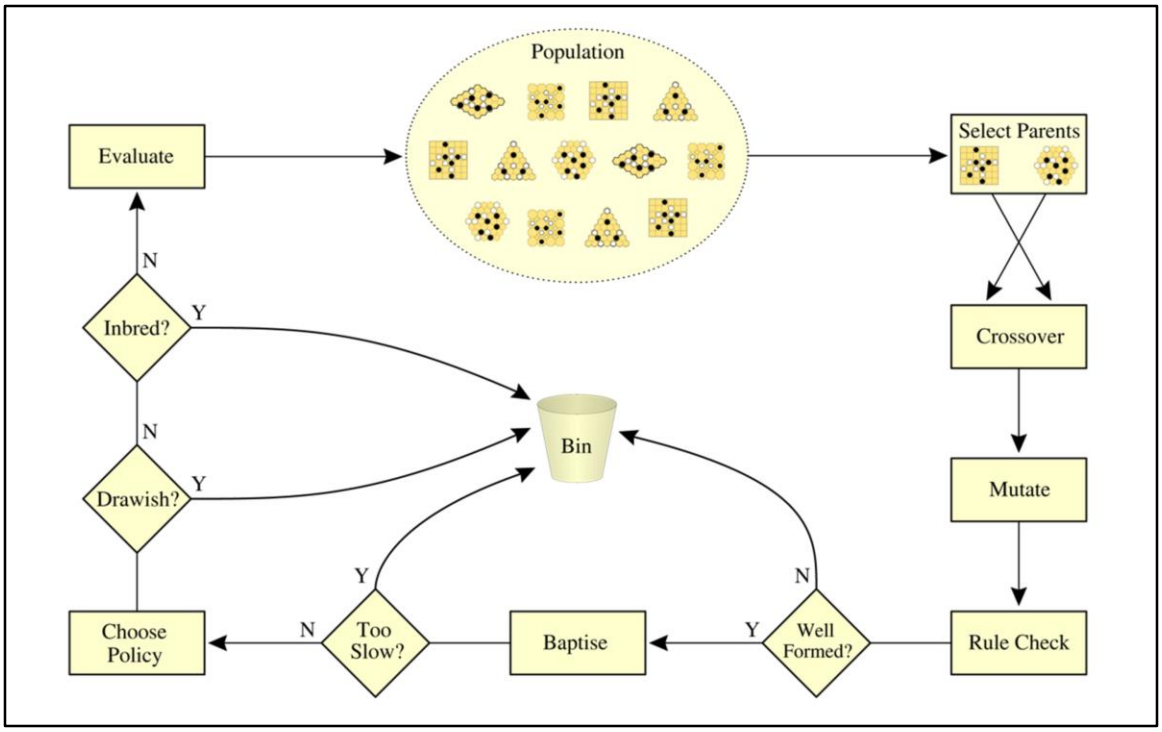
A number of generated levels with their corresponding fitness

Search-Based PCG Example #2

How would we create new game rules?

Creating Game Rules

- Rules are also content…
- Will need simulation-based evaluation - you can only judge game rules by playing the game
- Has been attempted for simple Pac-Man-like games (Togelius 2008), GVGAI games (Nielsen et al 2015)
- Perhaps most convincingly for board games (Browne 2008)



The Ludi game rule generator – see more details in Section 4.5.6

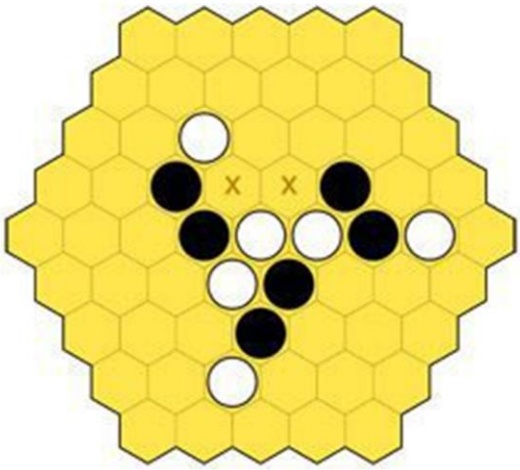
Yavalath

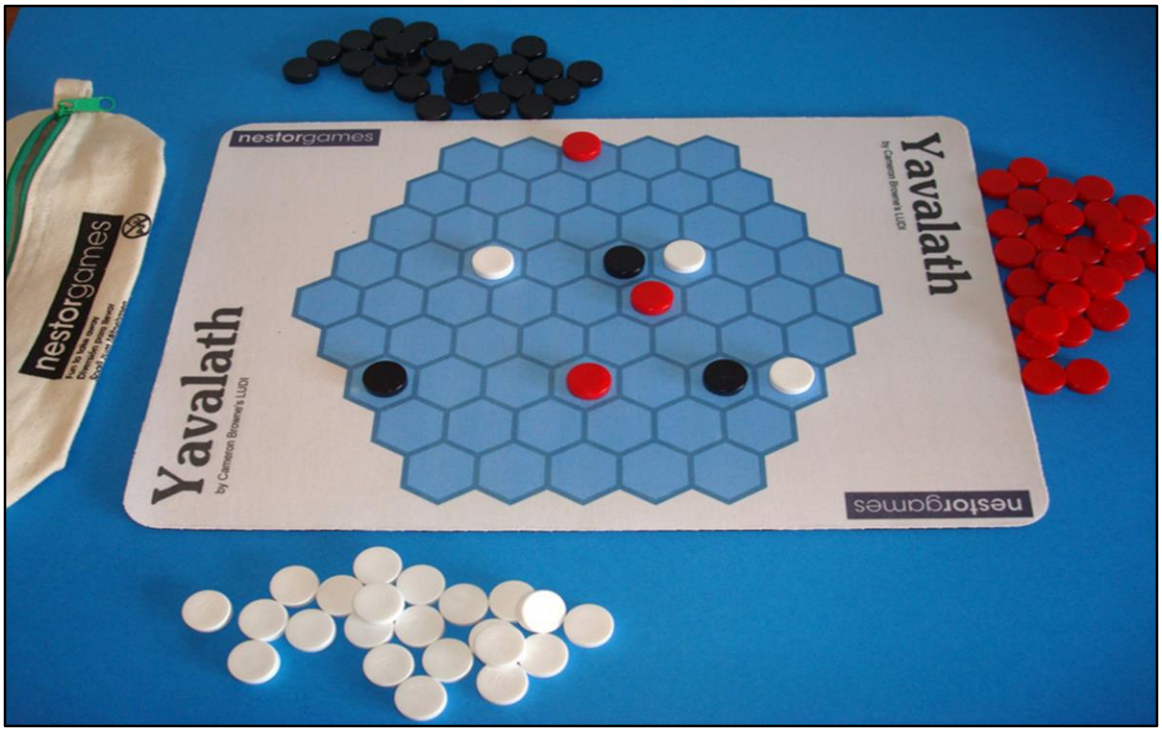
YAVALATH (#2)

```
(game Yavalath
   (players White Black)
   (board (tiling hex) (shape hex) (size 5))
   (end
      (All win (in-a-row 4))
      (All lose (and (in-a-row 3) (not (in-a-row 4))))
   )
)
```

The Yavalath game generated by Ludi (by Cameron Browne) – a grammar-based representation

Yavalath




nestorgames
Yavalath
by Cameron Browne's LUDI
nestorgames
Yavalath
by Cameron Browne's LUDI
nestorgames

Search-Based PCG Example #2

How would we design L-systems?

Evolving L-systems

How can we combine L-systems with evolutionary computation?

Evolving L-systems

- Evolving the axiom
- Evolving the grammar:
  - Change the shape of one or more production rules, or
  - Add/remove/replace productions
- Evolving the interpretation:
  - Evolve production probabilities
  - Evolve other aspects (e.g. turning angles)

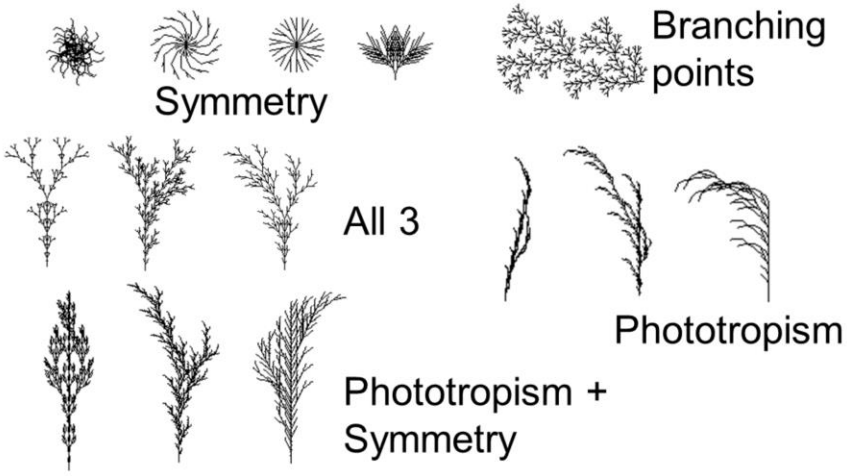
Evolving L-systems

- One example: Ochoa evolved the consequent of a single production rule
  - starting from F>F[-F]F[+F]F
- Mutation: replace single symbols, or blocks of a few symbols
- Crossover: swap complete “sub-trees” (like in genetic programming)

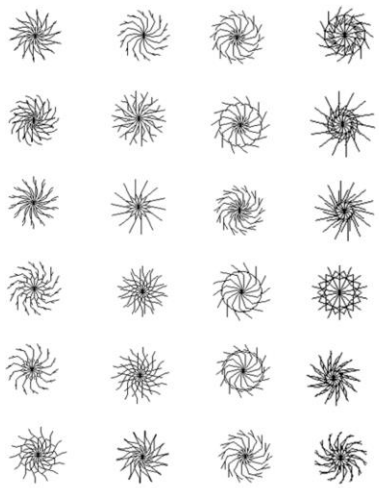
Fitness Functions

- Phototropism
- Bilateral symmetry
- Proportion of branching points

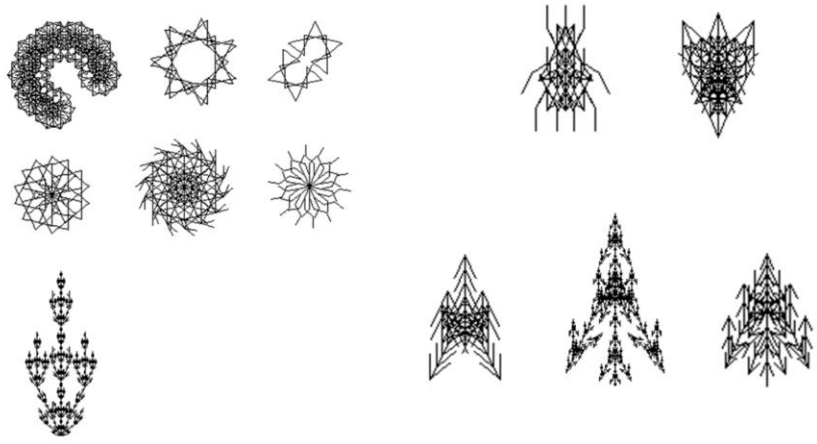
Evolved Systems



Evolved Systems

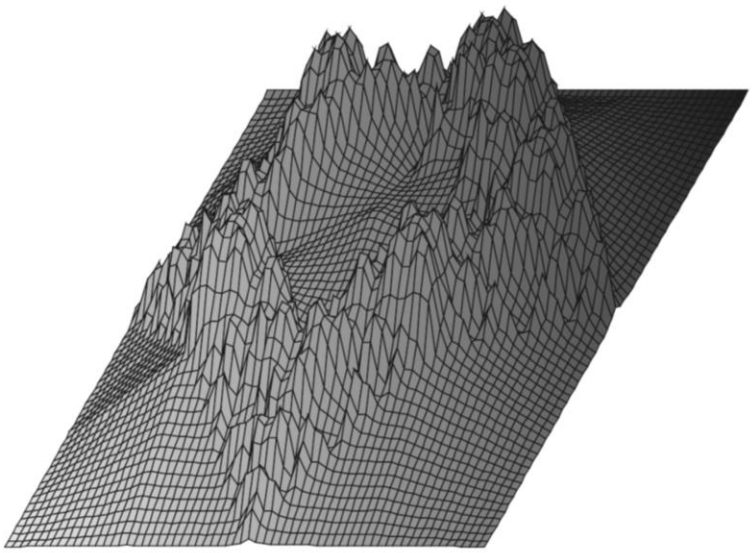


Evolved Systems

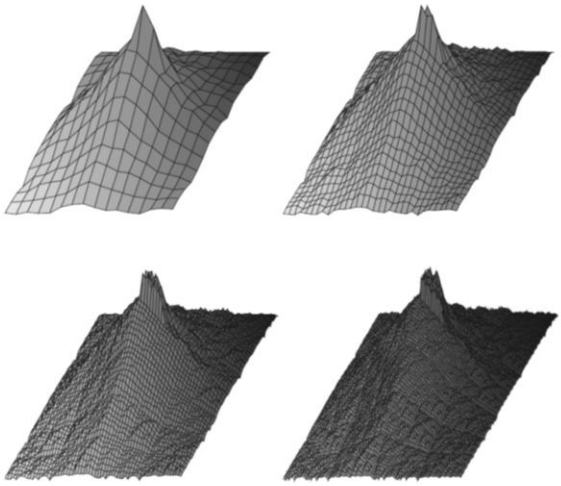


…and this was an extremely simple L-system!

Evolved 2D L-system Terrains



Evolved 2D L-system Terrains



Very short specification, yet infinite resolution!

[see Section 4.3.6 for more details]

An emerging direction in PCG research is to **train generators on existing content**, to be able to produce more content of the same type and style. This is inspired by the recent results in deep neural networks, where network architectures such as generative adversarial networks and variational autoencoders have attained good results in learning to produce images of e.g., bedrooms, cats or faces, and also by earlier results where both simpler learning mechanisms such as Markov chains and more complex architectures such as recurrent neural networks have learned to produce text and music after training on some corpus.

PCG via Machine Learning

- Basic idea of Procedural Content Generation via Machine Learning (PCGML): train machine learning models on corpuses of existing content, then generate new content
- Many methods useful, including n-grams, Markov chains, and… yes, even deep learning
- Reference: Summerville et al. (2018): Procedural Content Generation via Machine Learning (PCGML)

N-Grams

- Capture the statistic co-occurrence of sequential characters
- n: number of previous characters a new character depends on
- Commonly used for predictive text
- Can also be used for linear game content

(a) $n = 1$

(b) $n = 2$

(c) $n = 3$

Figure 6: Mario levels reconstructed by n-grams with n set to 1, 2, and 3 respectively.

Steve Dahlskog, Julian Togelius, and Mark J. Nelson (2014): **Linear levels through n-grams.** In proceedings of MindTrek

An example of PCGML via n-grams – see reference for further details

Neuroevolution

- Evolving the weights of neural networks, either interactively or according to some automatic fitness function
- Can be used for PCG: predicting some aspect of a level (e.g. enemies in Super Mario Bros) from other aspects (e.g. bricks and question mark blocks)
- Analogically: The different layers of a level are akin to different instruments in a song

Neuroevolution

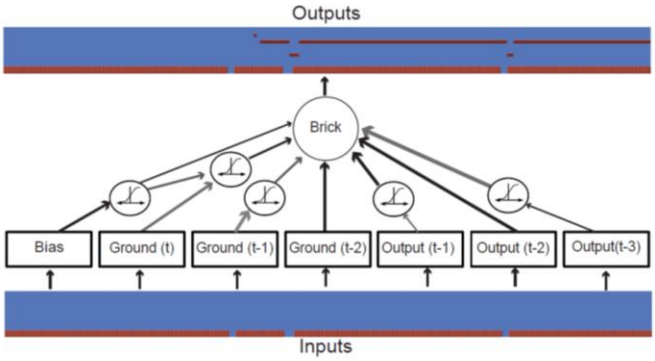
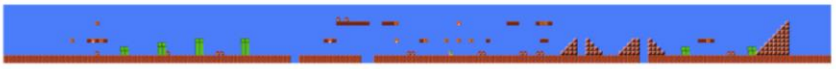
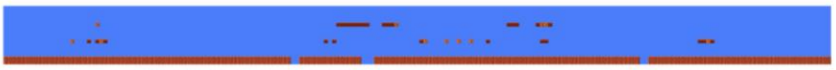


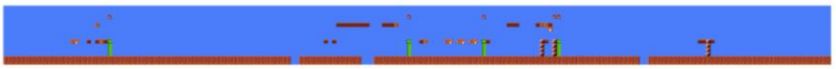
Figure 1: **The ANN Representation**. At each column in a given level, an ANN inputs visual assets (i.e., musical voices) from at least one tile type of *Super Mario Bros.* and outputs a different type of visual asset. This example figure shows ground tiles being input to the ANN while the brick tile placements are output from predictions made through NeuroEvolution of Augmenting Topologies [39]. To best capture the regularities in a level, each column is also provided information about the most recently encountered inputs (i.e., input values for the three previous columns). The inputs and outputs are then fed back into the ANN at each subsequent tick. Once trained, ANNs can potentially suggest reasonable placements for new human composed in-game tiles.



(a) Original World One Level One in Super Mario Bros.



(b) The Ground, Brick, and Question Mark Voices in World One Level One



(c) The remaining voices regenerated from several trained networks, using the three-voice level in (b) as input

(d) Author-created Level with Ground, Brick, and Question Mark Voices

(e) Additional voices generated by trained networks for the author-created level.

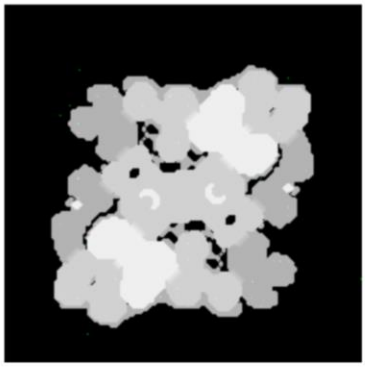
Amy K. Hoover, Julian Togelius and Georgios N. Yannakakis (2015): **Composing Video Game Levels with Music Metaphors through Functional Scaffolding**. ICCC Workshop on Computational Creativity and Games.

An example of PCGML via neuroevolution – see reference for further details

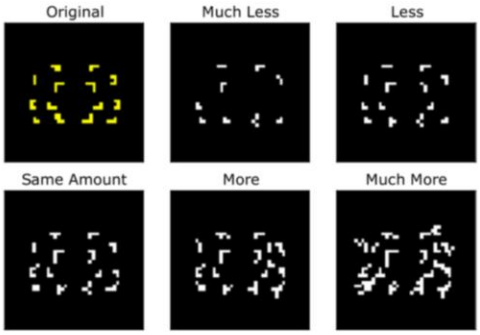
Convolutional Neural Nets

- Commonly used for classification and prediction from high dimensional matrices, such as images
- Makes use of the convolution operation to find recurring features in images while minimizing the number of parameters
- Can be used for PCG: predicting level features from other features (e.g. resources from base locations)

CNNs for *StarCraft* Resources



(a) A *StarCraft II* heightmap



(b) Varying resource amounts.

Figure 4: A heightmap and generated resource locations for *StarCraft II*

Scott Lee, Aaron Isaksen, Christoffer Holmgård and Julian Togelius (2016): **Predicting Resource Locations in Game Maps Using Deep Convolutional Neural Networks**. EXAG.

An example of PCGML via CNNs – see reference for further details

Long Short-Term Memory

- Commonly used for sequence recognition and prediction, for example for audio or text
- Often used for generative text, through predicting the next word or letter based on some training set
- Can be used for PCG: predicting the next element in a sequence, e.g. a level segment or a word in a description

LSTM for Mario

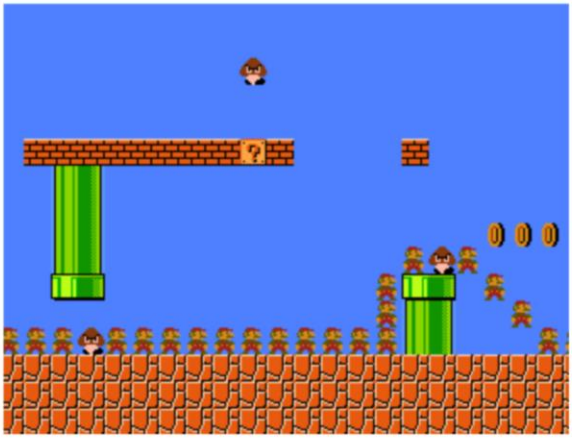


Figure 2: Example output of the LSTM approach, including generated exemplar player path.

An example of PCGML via LSTMs

LSTM for Magic Cards

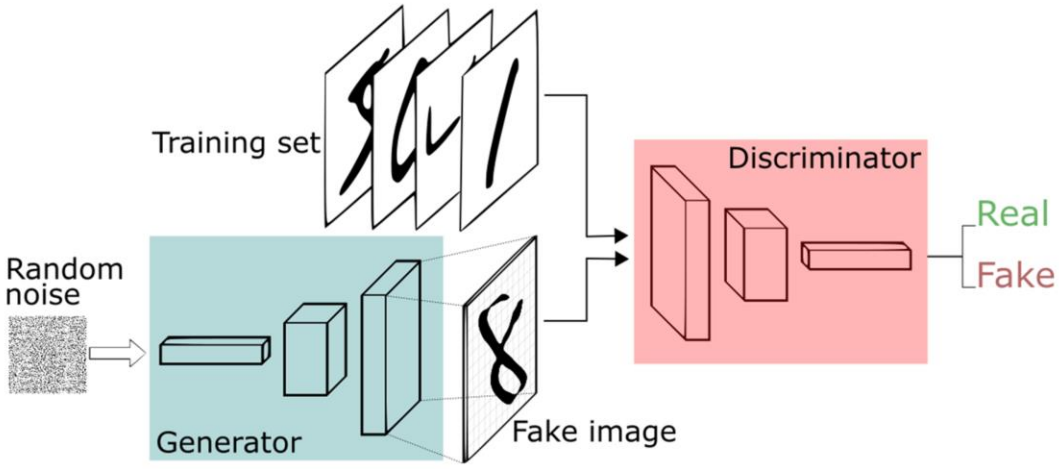
Figure 5: Example partial card specification and its resulting output.

An example of PCGML via LSTMs

Generative Adversarial Networks

- Train two networks intermittently: a generator and a discriminator
- The discriminator is trained to distinguish real artifacts from fake (generated) ones
- The generator is trained to generate artifacts that fool the discriminator
- Essentially the same idea as competitive coevolution

Generative Adversarial Networks



Evolving Mario Levels in the Latent Space of a Deep Convolutional Generative Adversarial Network

Vanessa Volz
TU Dortmund University
Dortmund, Germany
vanessa.volz@tu-dortmund.de

Jacob Schrum
Southwestern University
Georgetown, TX USA
schrum2@southwestern.edu

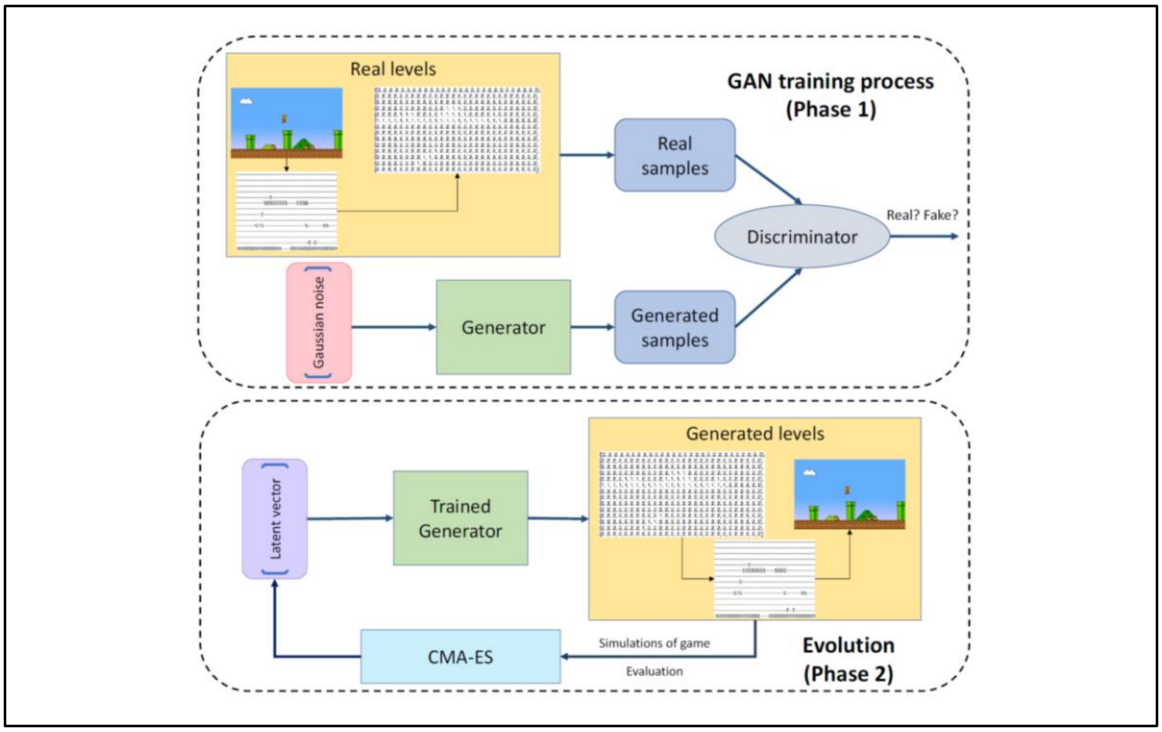
Jialin Liu
Queen Mary University of London
London, UK
jialin.liu@qmul.ac.uk

Simon M. Lucas
Queen Mary University of London
London, UK
simon.lucas@qmul.ac.uk

Adam Smith
University of California
Santa Cruz, CA USA
amsmith@ucsc.edu

Sebastian Risi
IT University of Copenhagen
Copenhagen, Denmark
sebr@itu.dk

- Train a GAN on a Mario level, so that it can produce level segments
- Search the latent space of the trained GAN for level segments with particular properties
  - Approach called *Latent Variable Evolution*


Real levels
GAN training process
(Phase 1)
Real
samples
Discriminator
Real? Fake?
Gaussian noise
Generator
Generated
samples
Generated levels
Latent vector
Trained
Generator
CMA-ES
Simulations of game
Evaluation
Evolution
(Phase 2)

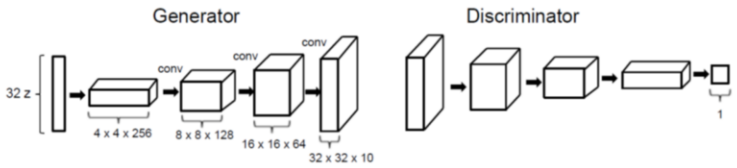
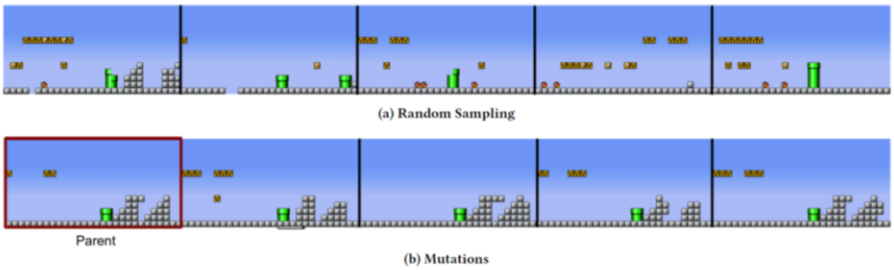


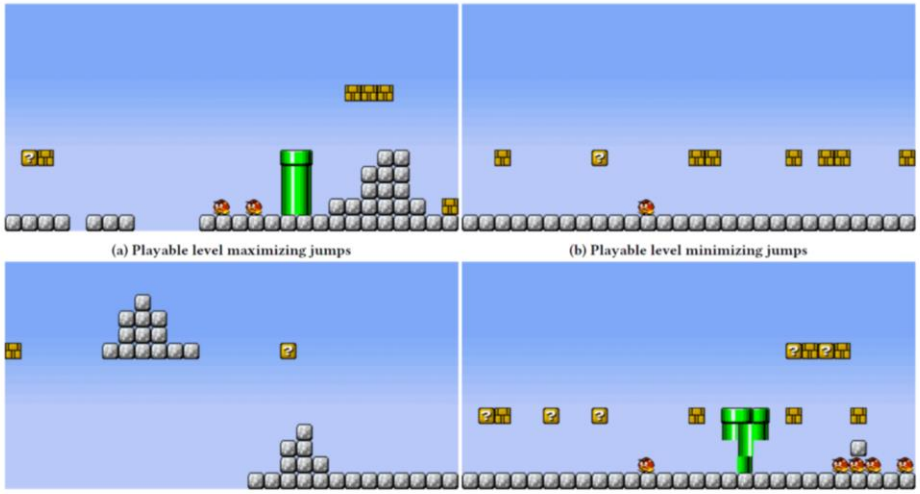
Figure 3: The Mario DCGAN architecture.



(a) Random Sampling

(b) Mutations

Figure 4: Generated Examples. Shown are samples produced by the GAN by (a) sampling random latent vectors, and (b) randomly mutating a specific latent vector. The main result is that the generator is able to produce a wide variety of different level layouts, but varied offspring still resemble their parent.



(a) Playable level maximizing jumps

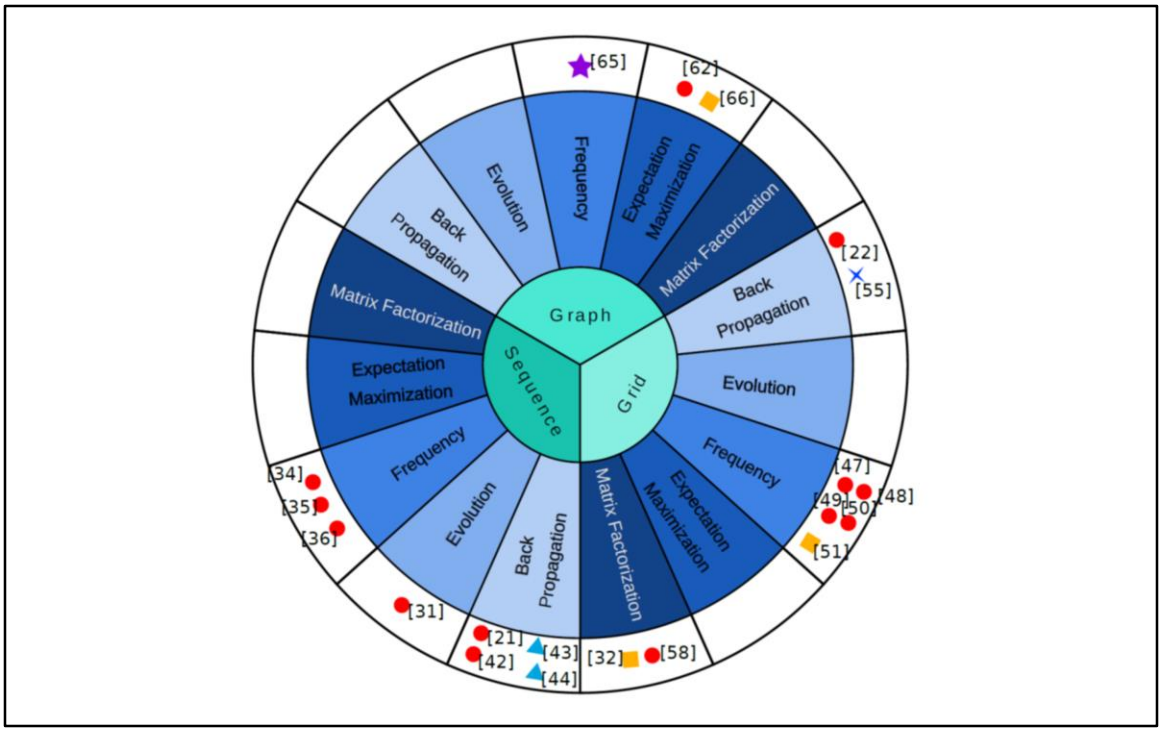
(b) Playable level minimizing jumps

(c) Unplayable level

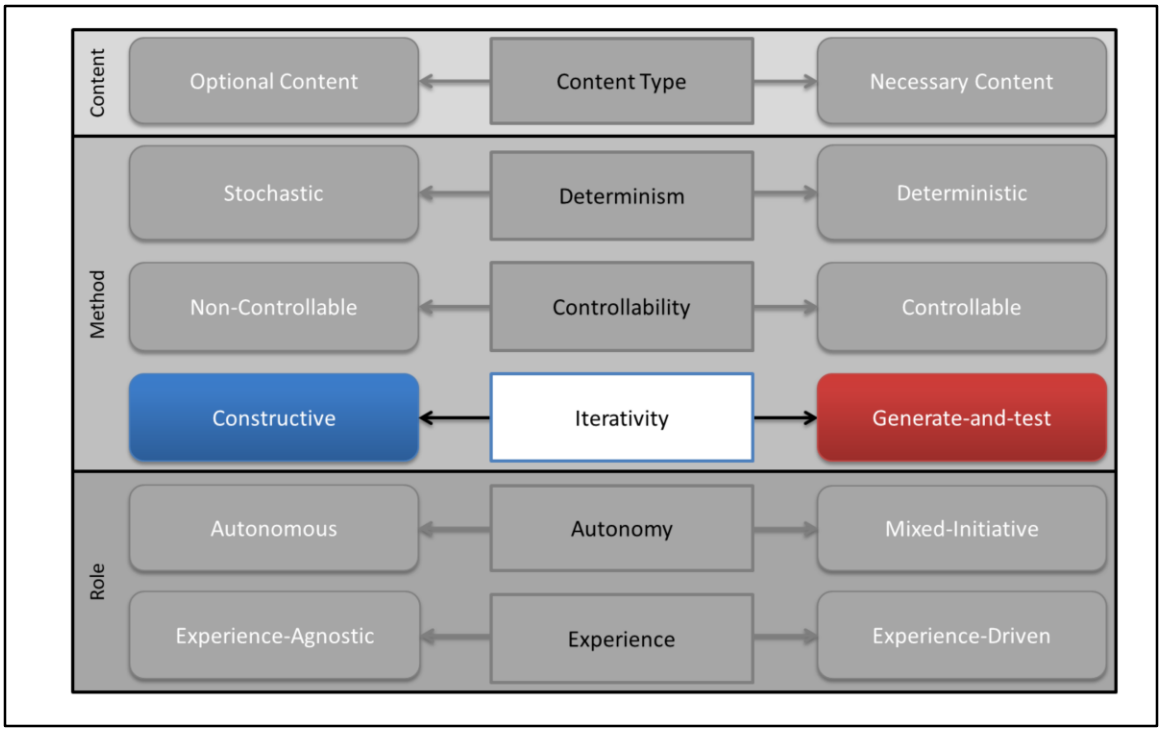
(d) Broken titles

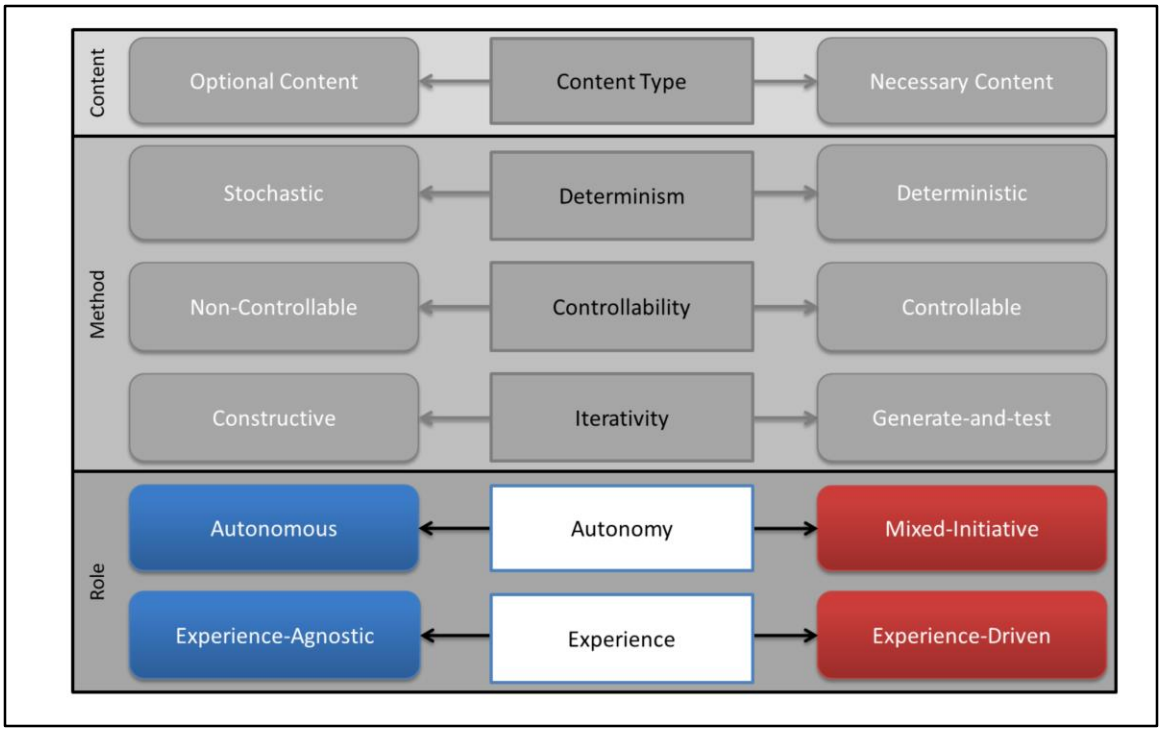
Figure 7: Agent-based optimization examples. (a) and (b) show good examples of levels in which the number of jumps is maximized (F_1), and minimized (F_2), respectively. (c) shows an example of one of the worst individuals found (not playable, F_1). An example of an individual that reaches high fitness (maximizing jumps, F_1) but has broken titles is shown in (d).

Figure 6: Level with increasing difficulty. Our LVE approach can create levels composed of multiple parts that gradually increase in difficulty (less ground tiles, more enemies). In the future this approach could be used to create a level in real-time that is tailored to the particular skill of the player (dynamic difficulty adaptation).



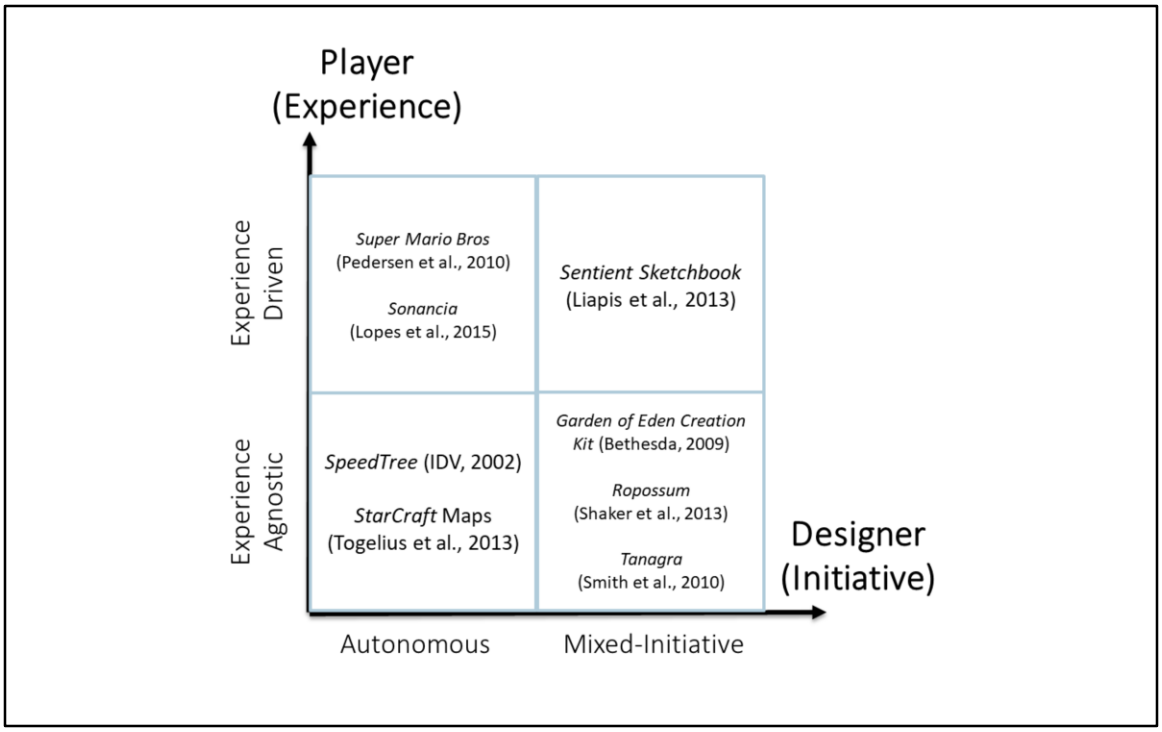
Overview of PCGML work; see more details at *Summerville et al. (2018): Procedural Content Generation via Machine Learning (PCGML)*


Content
Optional Content
Content Type
Necessary Content
Method
Stochastic
Determinism
Deterministic
Non-Controllable
Controllability
Controllable
Constructive
Iterativity
Generate-and-test
Role
Autonomous
Autonomy
Mixed-Initiative
Experience-Agnostic
Experience
Experience-Driven



[see Section 4.4. for more details]

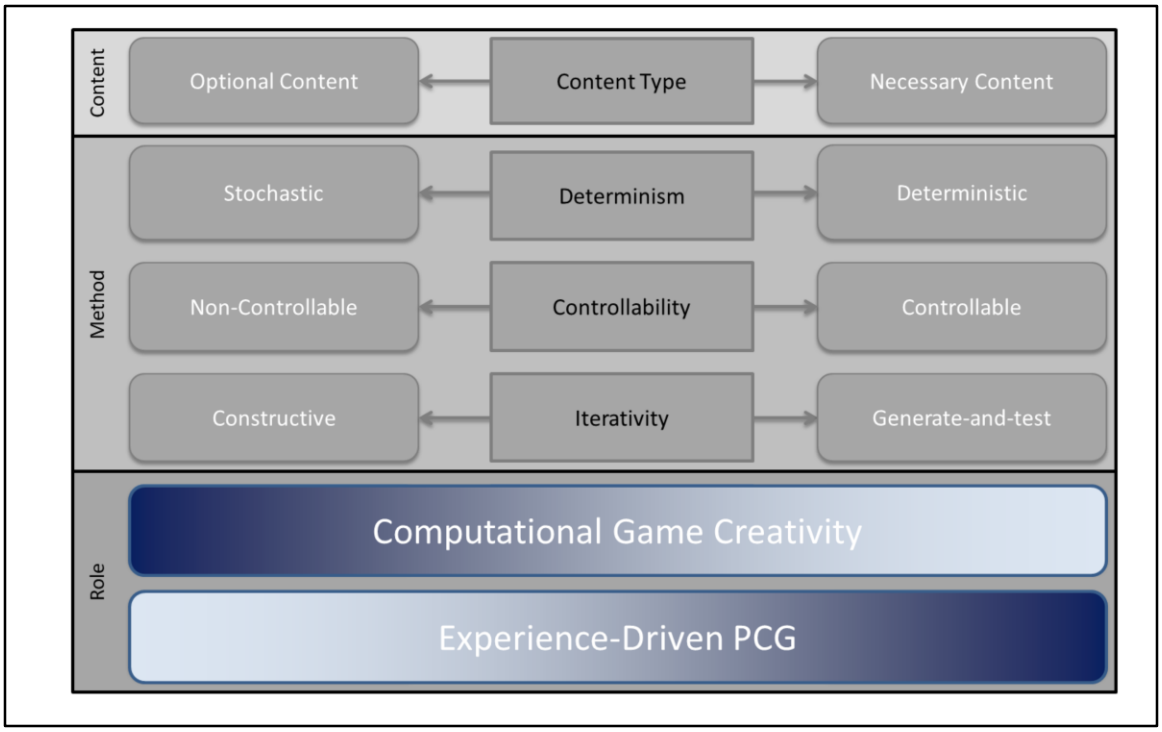
The generation of content algorithmically may take different roles within the domain of games.



- We identify two axes across which PCG roles can be placed: **players** and **designers**

- We envision PCG systems that consider designers while they generate content or they operate interdependently of designers; the same applies for players. The figure visualizes the key roles of PCG in games across the dimensions of designer initiative and player experience.

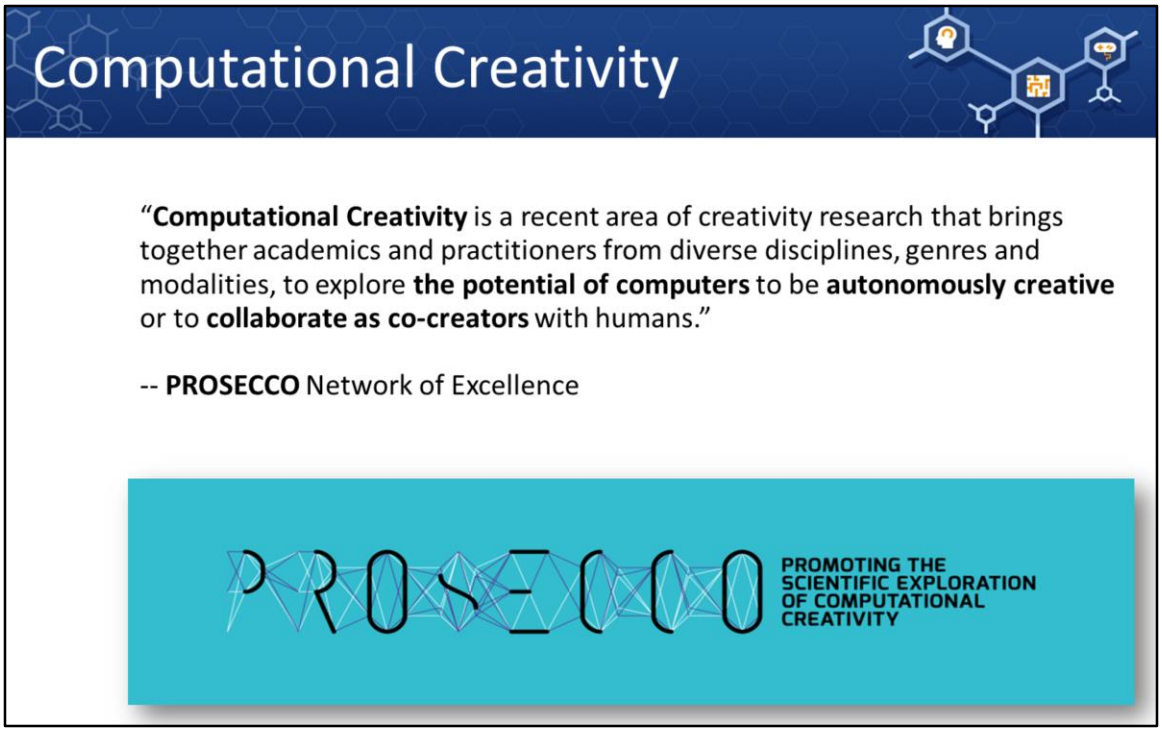
- PCG can act either autonomously or as a collaborator in the design process. We refer to the former role as **autonomous** generation and the latter role as **mixed-initiative** generation. Further, we cover the **experience-driven** PCG role by which PCG algorithms consider the player experience in whatever they try to generate. Finally, if PCG does not consider the player as part of the generation process it becomes **experience-agnostic**.



By viewing the dimensions (roles) of “autonomy” and “experience” from a different perspective we can, respectively, refer to **computational game creativity** and **experience-driven PCG**

In the following part we focus on the notion of computational game creativity: from mixed-initiative generation all the way to completely autonomous generation.

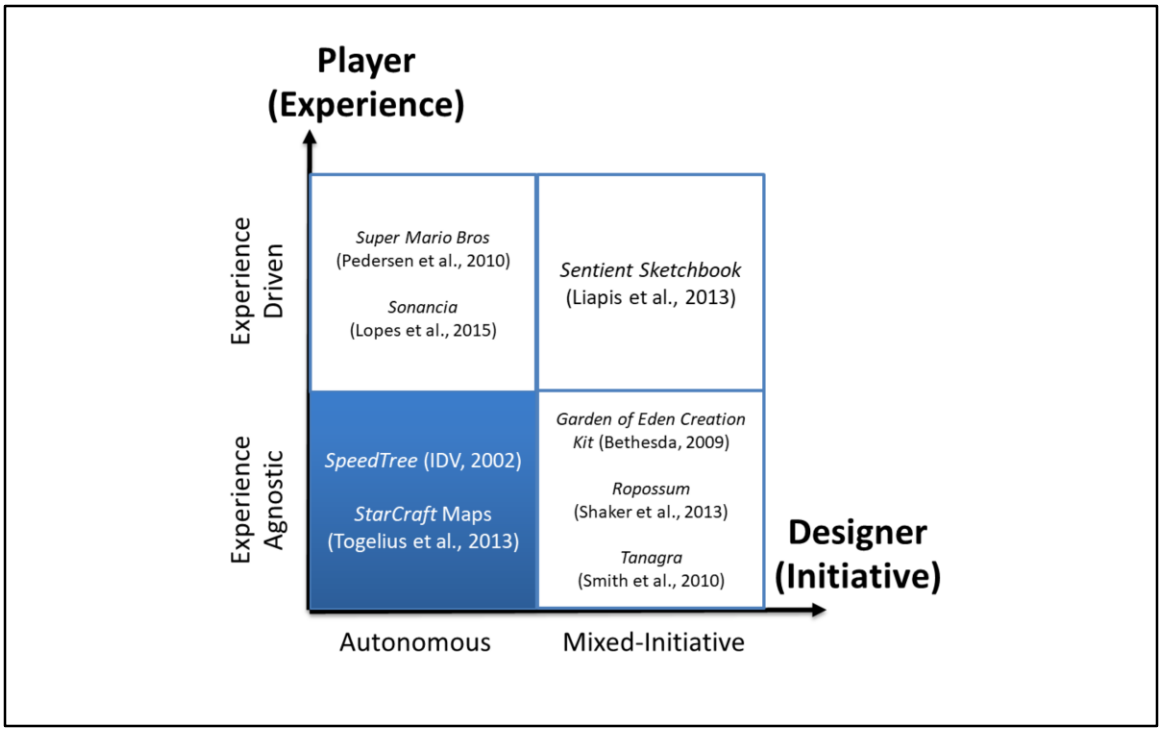
Refer to the cited paper for more details



"Computational Creativity" as defined by the PROSECCO network of excellence

"Computational Game Creativity" as defined by Liapis et al. (2014)

Games challenge and advance computational creativity (and AI at large)

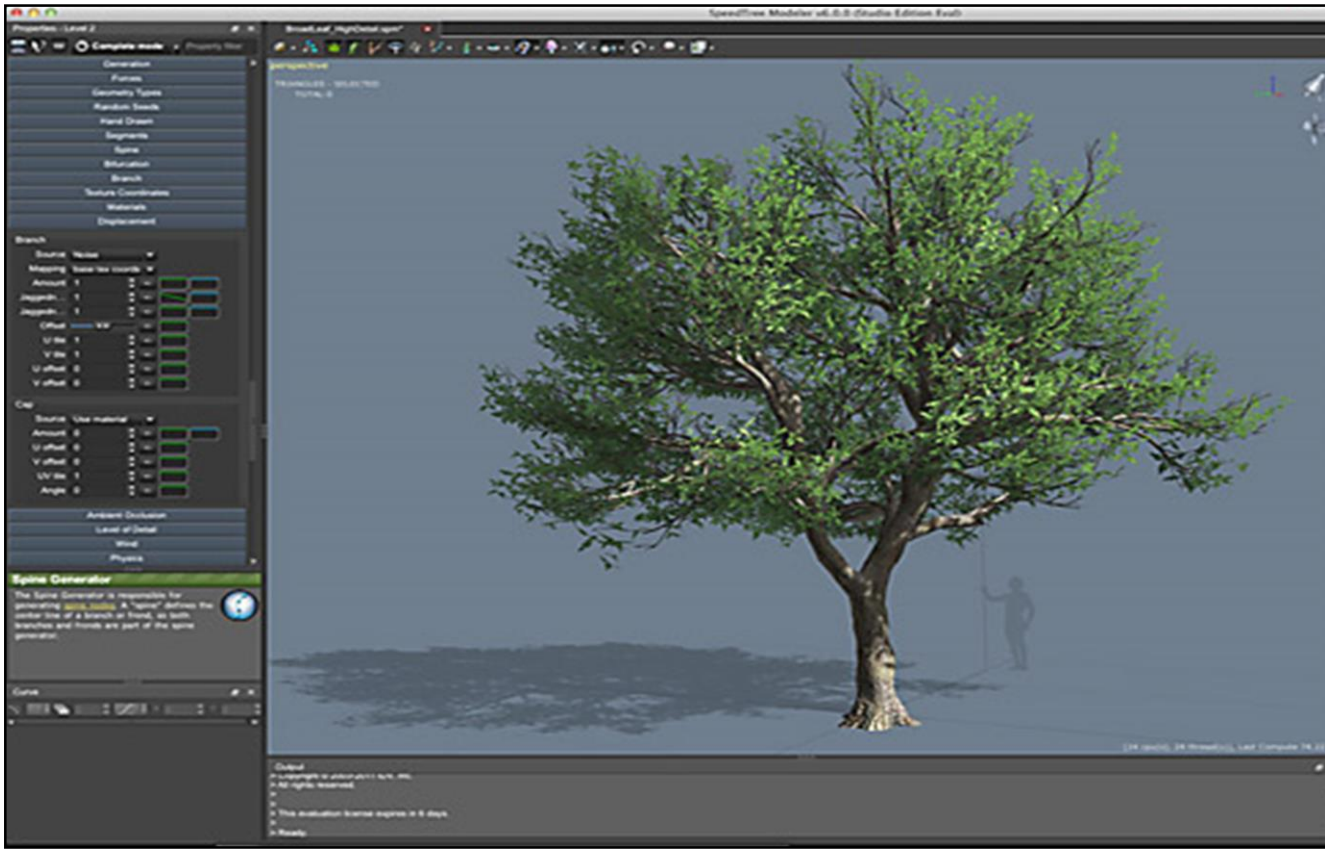


[see Section 4.4.2 for further details]

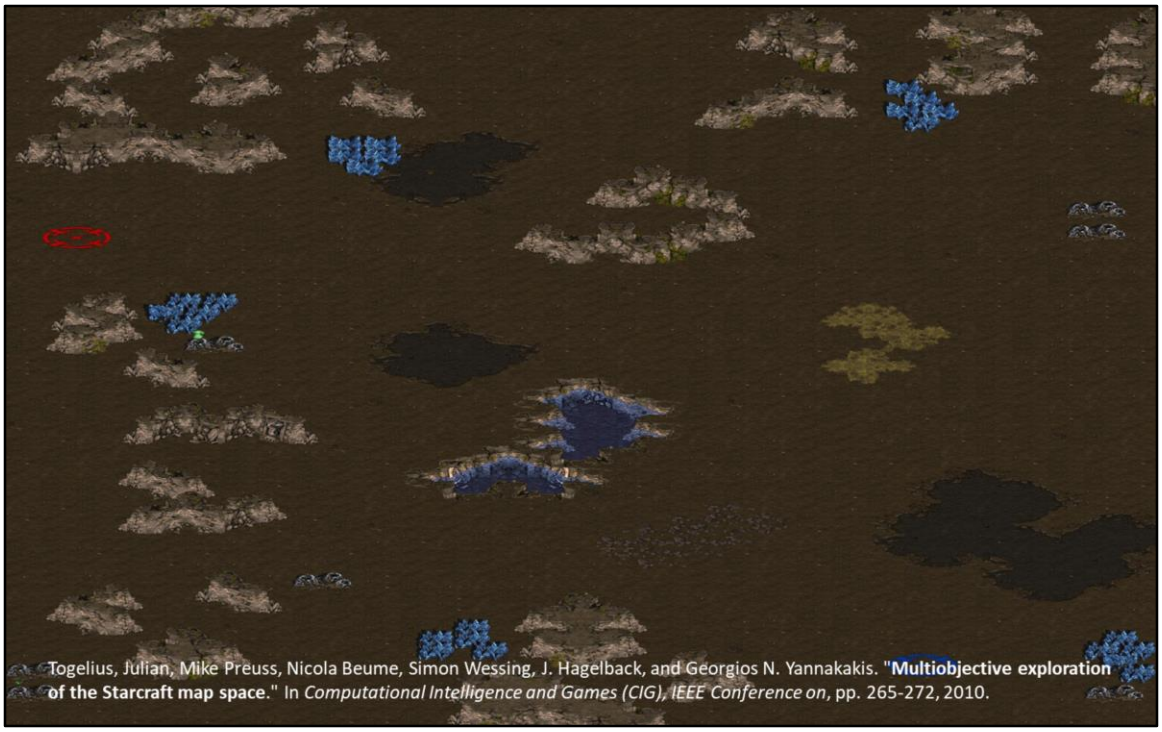
We will first focus on **autonomous**, **experience-agnostic** generation and look at some indicative examples in the area

The role of autonomous generation is arguably the most dominant PCG role in games.

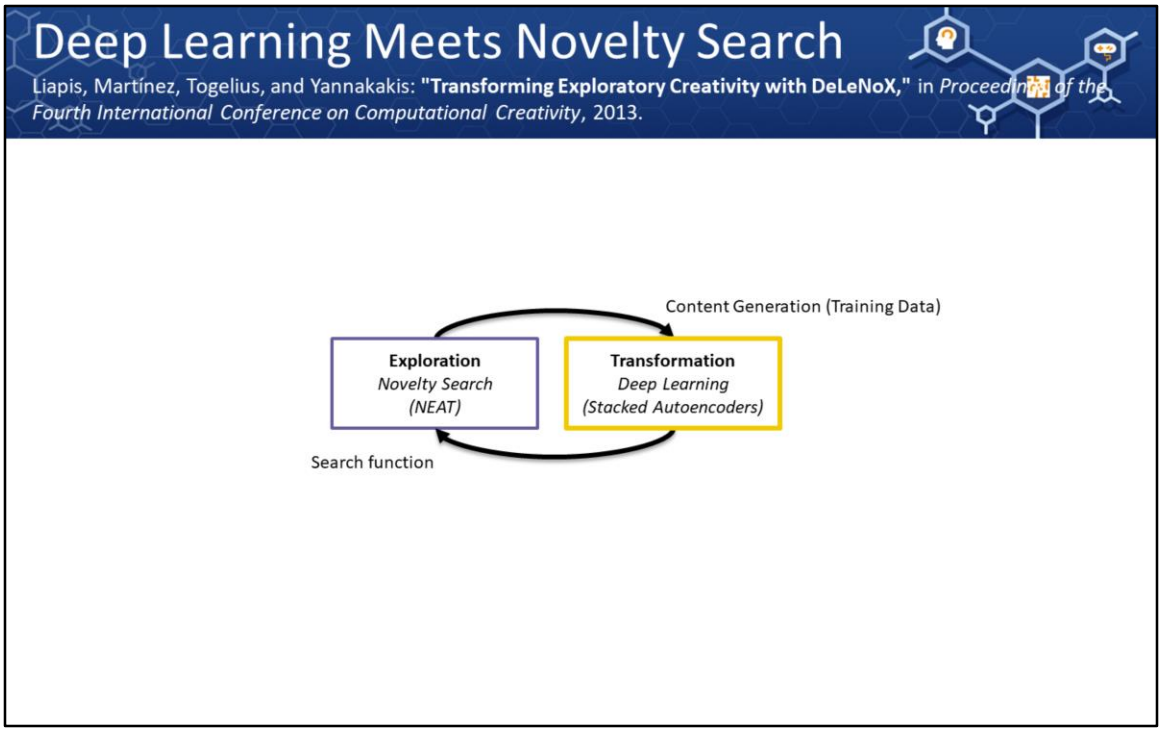
Note the fuzzy borderline between mixed-initiative and autonomous PCG systems. It might be helpful, for instance, to consider autonomous PCG as the process by which the role of the designer starts and ends with an offline setup of the algorithm.



For instance, the designer is only involved in the parameterization of the algorithm as in the case of *SpeedTree* (IDV, 2002).

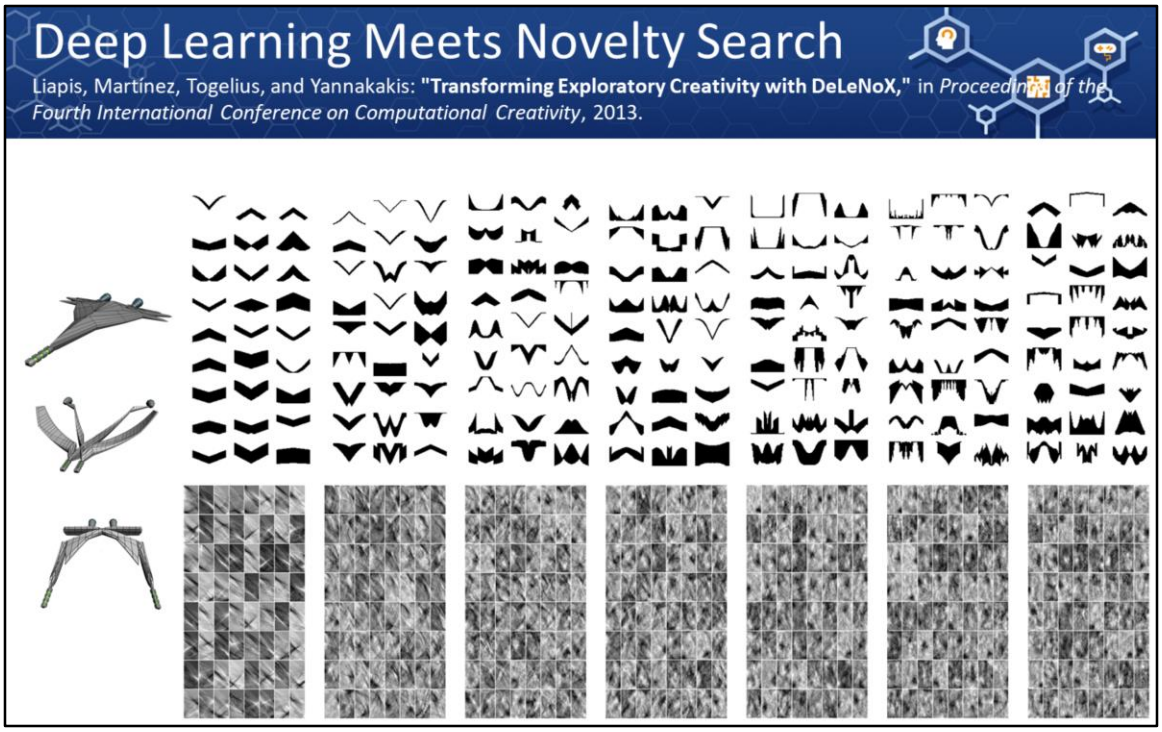


This is a Starcraft map generated autonomously via multiobjective evolution. Objectives included balance and symmetry. Further details can be found in the referenced paper

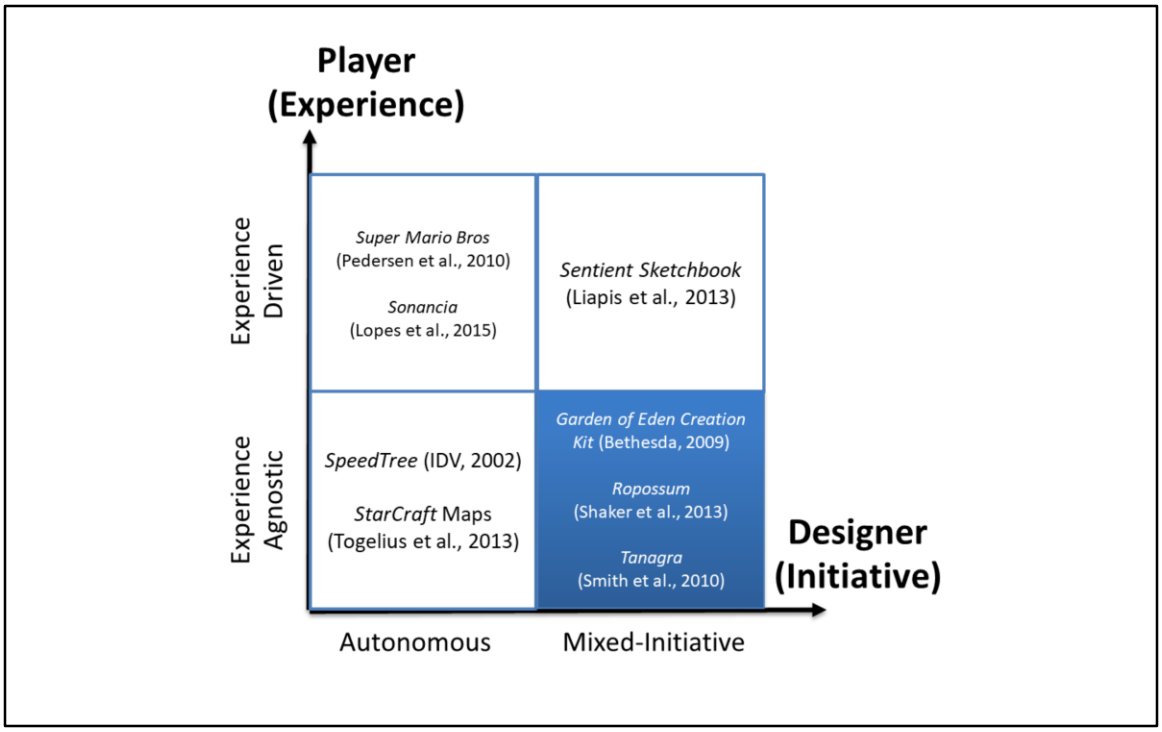


In this work Deep Learning (stacked autoencoders) are used in combination with novelty search (via NEAT) to drive the generation of increasingly complex and novel content (spaceships in this example) in an iterative fashion. The DeLeNoX system realises a process named “transformational exploration”.

The basic steps of DeLeNoX are as follows: 1) Content is generated in massive amounts to provide training data; 2) In the transformation phase, content is deep learned (unsupervised learning) resulting in compact representations of the generated content; 3) Such representations are used as inputs to the final layers of a neural network; 4) During the exploration phase, novelty search shapes Neuroevolution of Augmenting Topologies (NEAT) networks which provide content that is increasingly novel and complex as the network’s architecture grows.

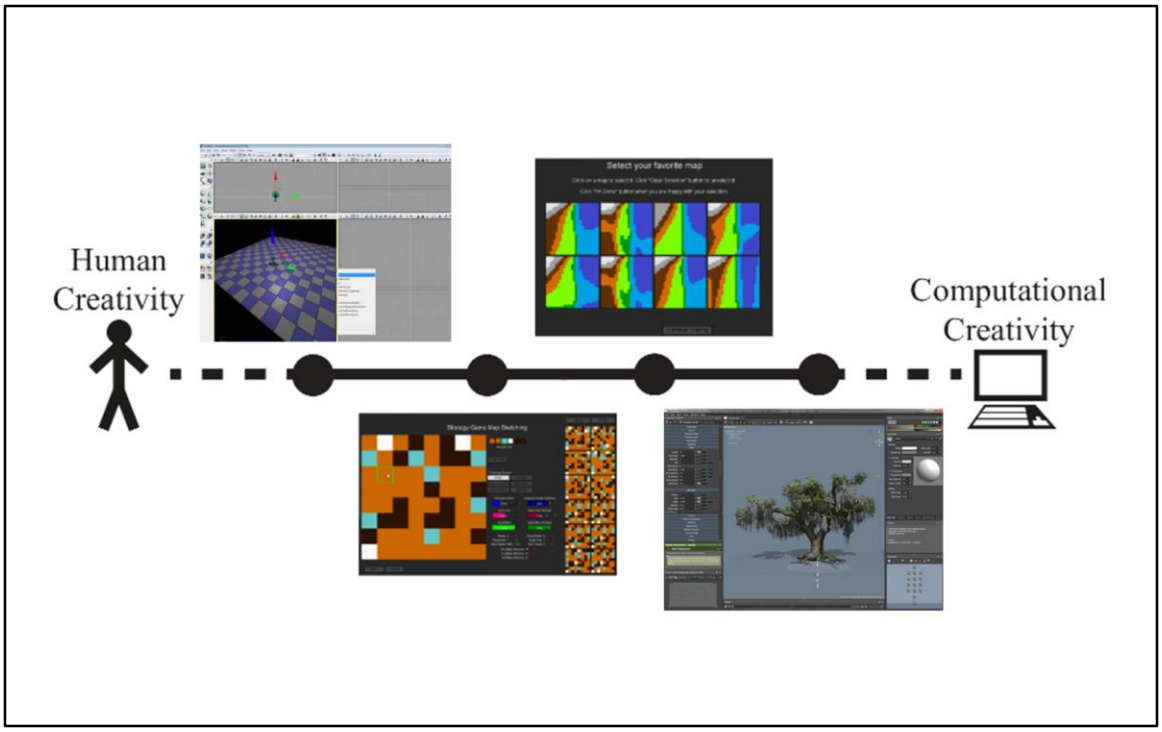


The top images display the evolution of spaceships (from left to right) whereas to bottom images illustrate the autoencoder feature maps constructed at each generation (from left to right). DeLeNoX renders its graphical output in 3D (spaceships on the left)



[see Section 4.4.1 for further details]

Let us now focus on **experience-agnostic, mixed-initiative** generation and look at some indicative examples in the area



[see Section 4.4.1 for further details]

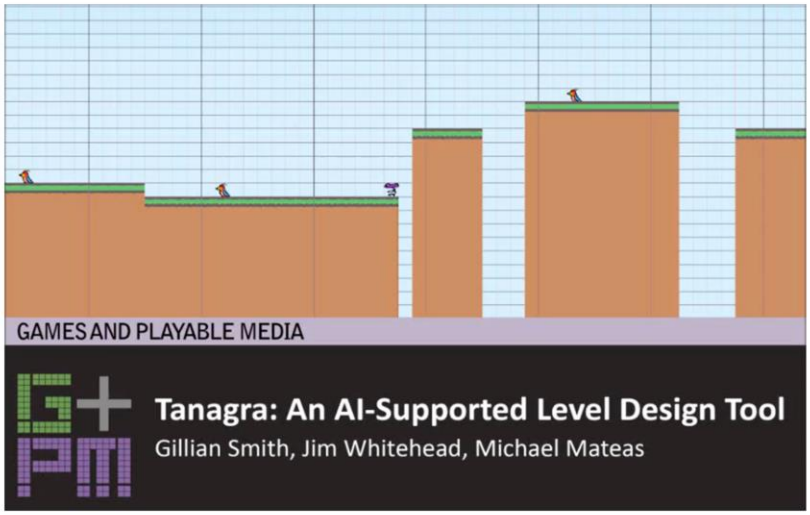
AI-assisted game design refers to the use of AI-powered tools to support the game design and development process. AI can assist in the creation of game content varying from levels and maps to game mechanics and narratives.

The figure illustrates the **mixed-initiative** spectrum between human and computer initiative (or creativity) across a number of mixed-initiative design tools.

From left to right:

- Unreal development kit: mostly designer initiative but boosts creativity through automating interpolations, pathfinding and rendering
- Sentient Sketchbook – active sketching of a map (more on this tool in upcoming slides)
- Sentient World – a few clicks only required by the designer; mixed-intitiave design is realised by interactive evolution (more on this tool in upcoming slides)
- SpeedTree – only a few parameters need to be instantiated by the designer; the tool the generates trees and vegetation autonomously

Tanagra: Constraint Solver for MI-PCG



Tanagra is an early example of mixed-initiative platformer level generation which is based on the design principle of rhythm. Tanagra used constraint solvers to generate levels; the generation process is guided through designer initiatives. Further details about Tanagra can be found in the referenced paper by Smith et al

Ropossum

Physics-based puzzle “cut the rope”

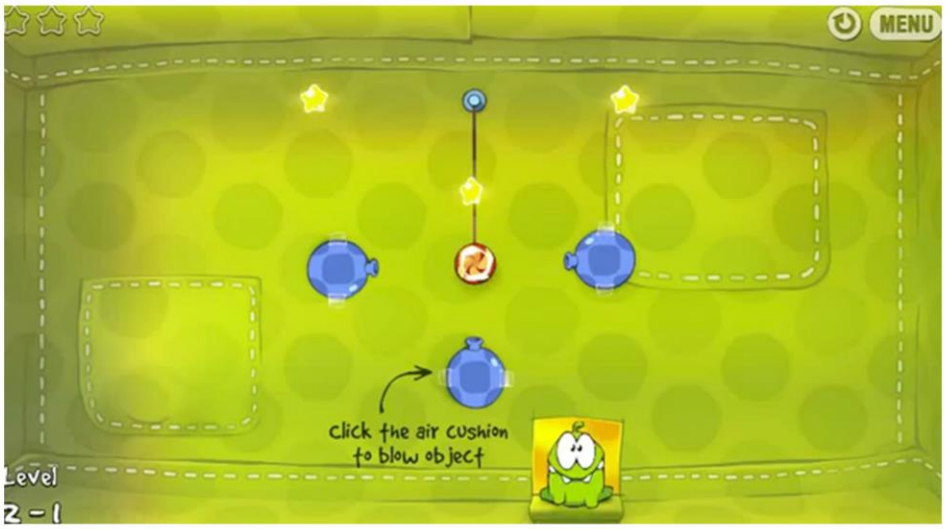
- evolutionary grammars for creating new puzzles
- playability module for testing how (if?) to solve a puzzle
- using the designer’s input in complete or partial designs

M. Shaker, M. H. Sarhan, O. A. Naameh, N. Shaker, and J. Togelius. Automatic generation and analysis of physics-based puzzle games. In Computational Intelligence in Games (CIG), 2013 IEEE Conference on.

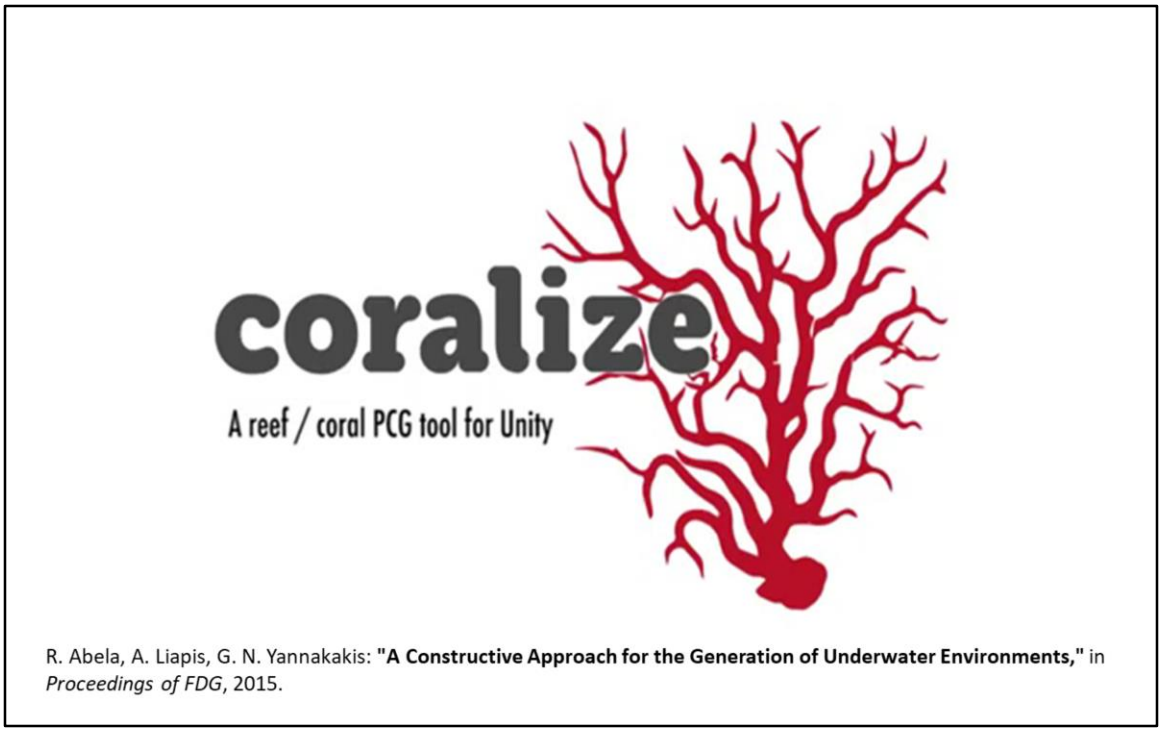
Another mixed-initiative example is Ropossum. In Ropossum the designer may select to design elements of Cut the Rope (Chillingo, 2010) game levels; the generation of the remaining elements are left to evolutionary algorithms to design.

For more details please refer to the cited paper

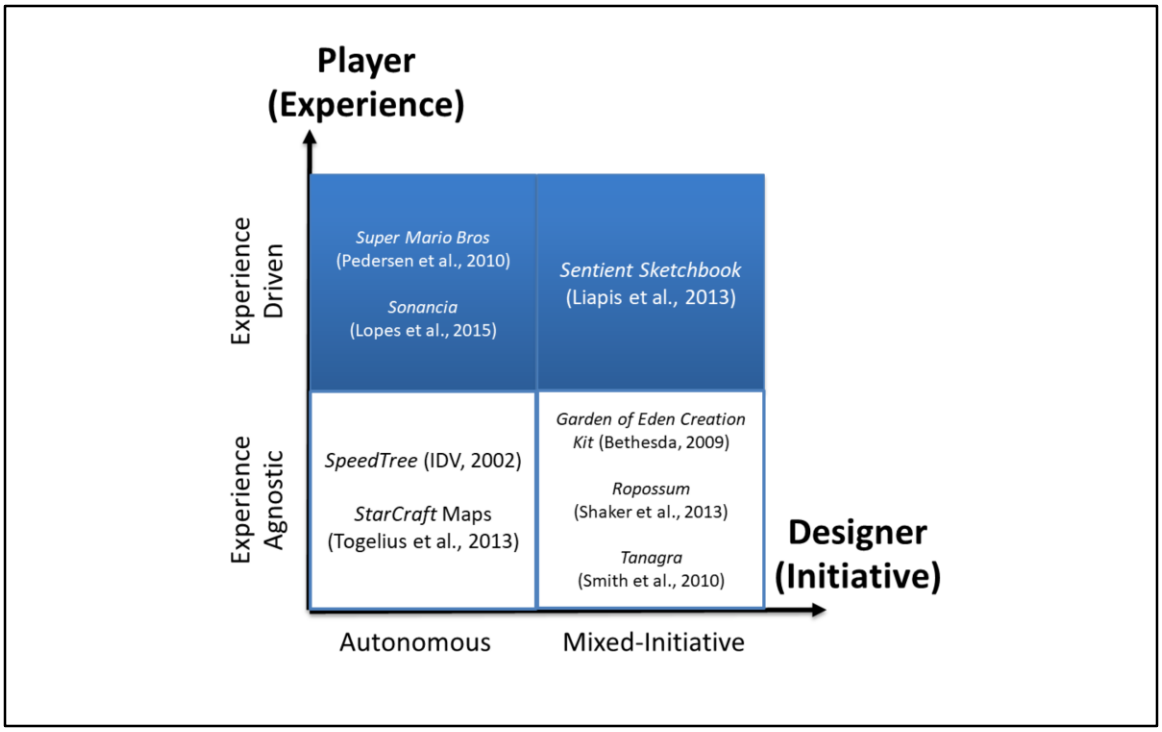
Ropossum



Ropossum

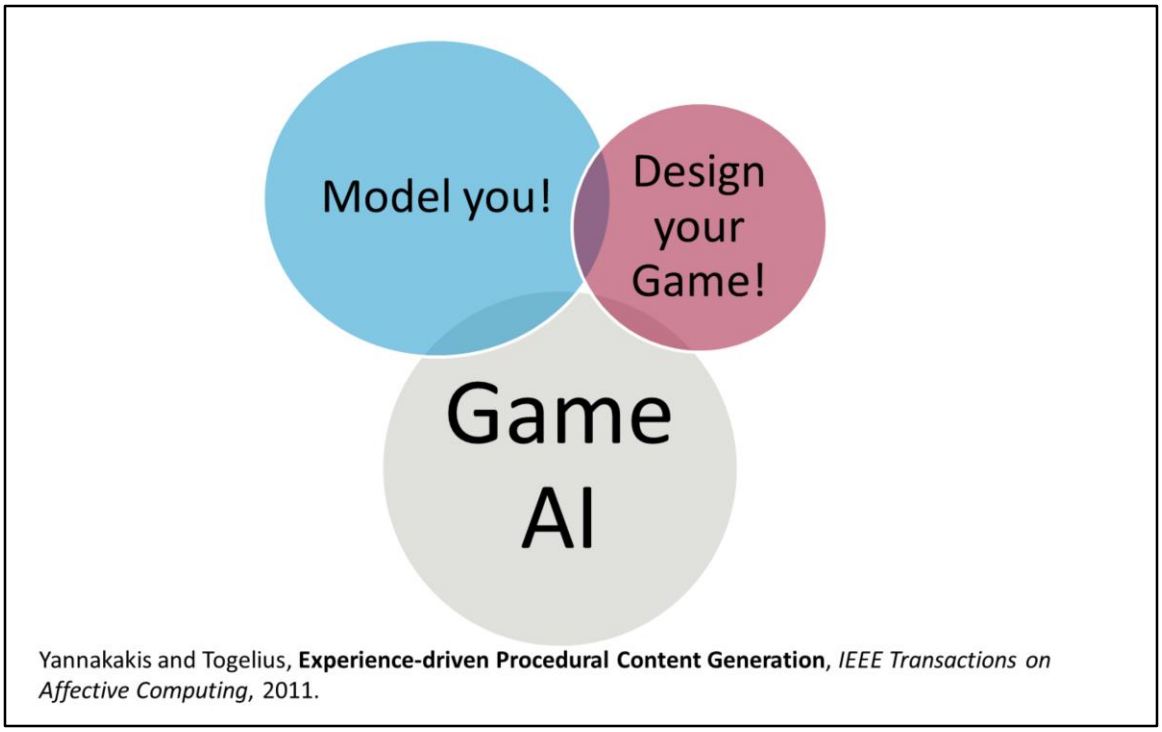


Coralize is another example of constructive-based and search-based PCG approaches for the mixed initiative generation of corals in Unity. See the cited paper for more details



[see Section 4.4.3 for further details]

Now that we have covered the experience-agnostic paradigm (either autonomous or mixed-initiative) we will focus on **experience-driven** generation and look at some indicative examples in the area (either autonomous or mixed-initiative).

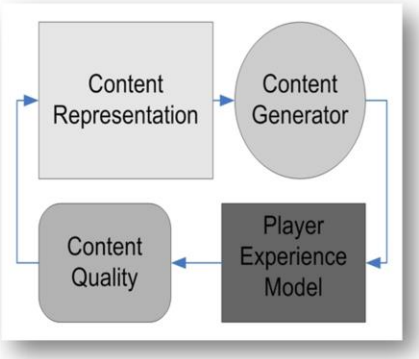


Yannakakis and Togelius, **Experience-driven Procedural Content Generation**, *IEEE Transactions on Affective Computing*, 2011.

In a broad sense Experience-driven PCG can be viewed as the intersection of player modelling (Chapter 5) and PCG.

A General PCG Framework

- Content is the **building block** of player experience
- **Search-based PCG**: use optimization algorithms (such as evolutionary algorithms) to search the space of content
- **Experience-driven PCG**: base evaluation function on player experience models



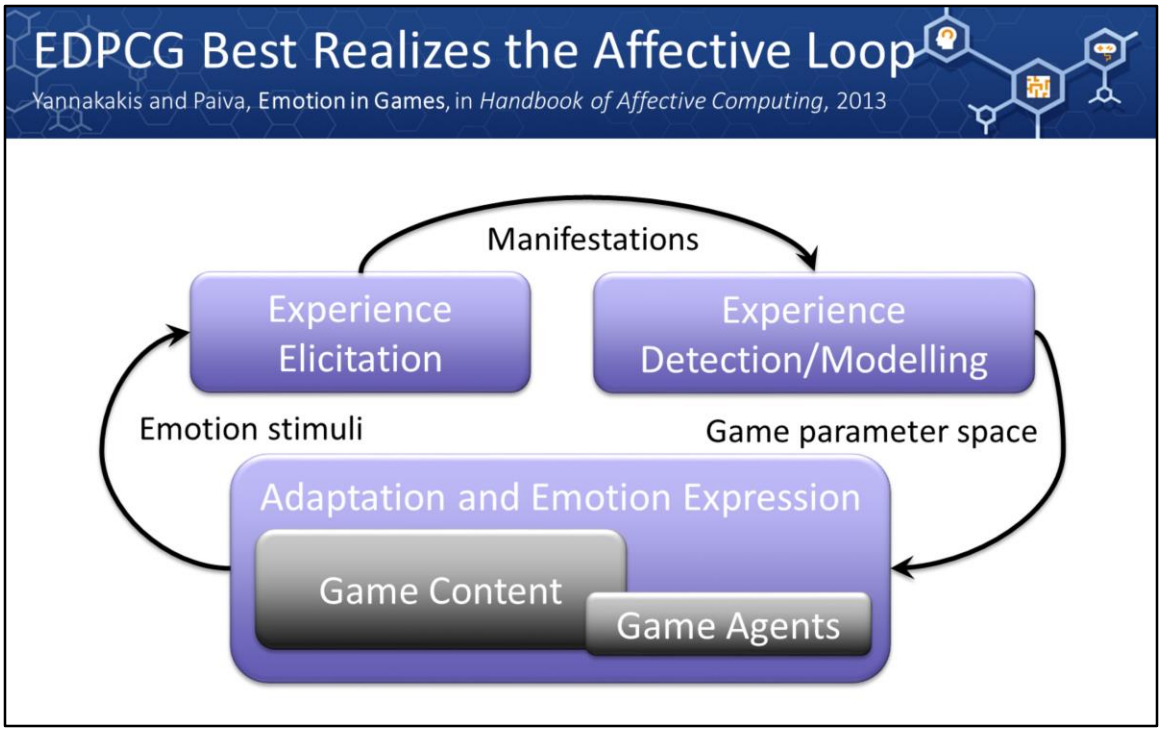
EDPCG: What is it?

"**A framework** for **personalised generation of content** in human computer interaction (in particular in games). It views **(game) content as the *building block* of user (player) experience**"

Yannakakis, G. N., & Togelius, J. (2011). **Experience-driven procedural content generation**. *IEEE Transactions on Affective Computing, 2*(3), 147-161.

[see Section 4.4.3 for further details]

The definition of the experience-driven procedural content generation framework.



This is an illustration of the *affective loop* as applied to games. The 3 core phases include: emotion elicitation, emotion detection and emotion expressions. Here are some reasons why games offer the best possible realization of the affective loop.

- Emotion elicitation: games offer brilliant contextual building blocks for eliciting emotion as stimuli are variant and come from different sources such as images, sounds, stories etc.
- Emotion Detection: users of games (players) are generally more that willing to provide more input of multimodal nature (via sensors) as long as that would lead to enhanced experience. In a sense, players are the best possible users for affective computing and multimodal interaction studies
- Emotion Adaptation:
    - Users voluntarily go through a spectrum of experiences during play: these vary from the very positive to the very negative ones
    - Affective experiences in games are **affected** by players! As a result a player is used to and largely **open** to affect-based adaptation!

EDPCG realises the game affective loop by offering content that is "appropriate" (based on experience heuristics) for players. EDPCG is a content-based formalization of the game affective loop.

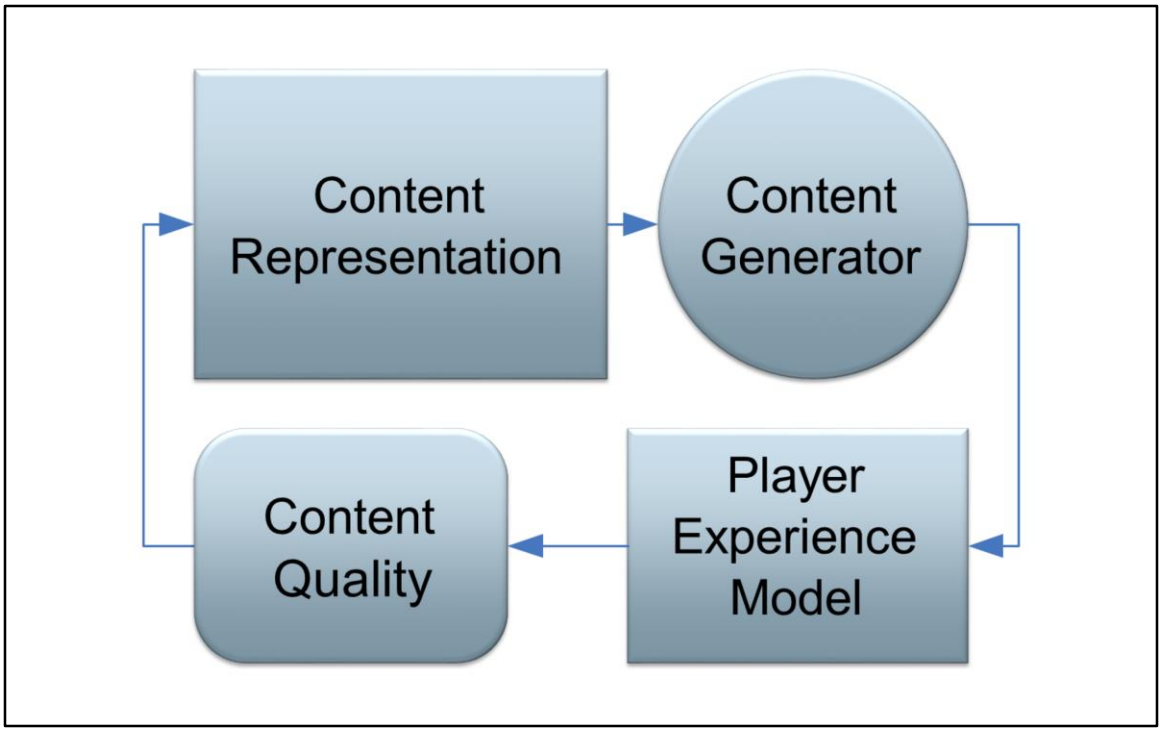
Experience-Driven Procedural Content Generation

"… collection of **affective patterns** elicited, **cognitive processes** emerged and **behavioral traits** observed during interaction (gameplay)"

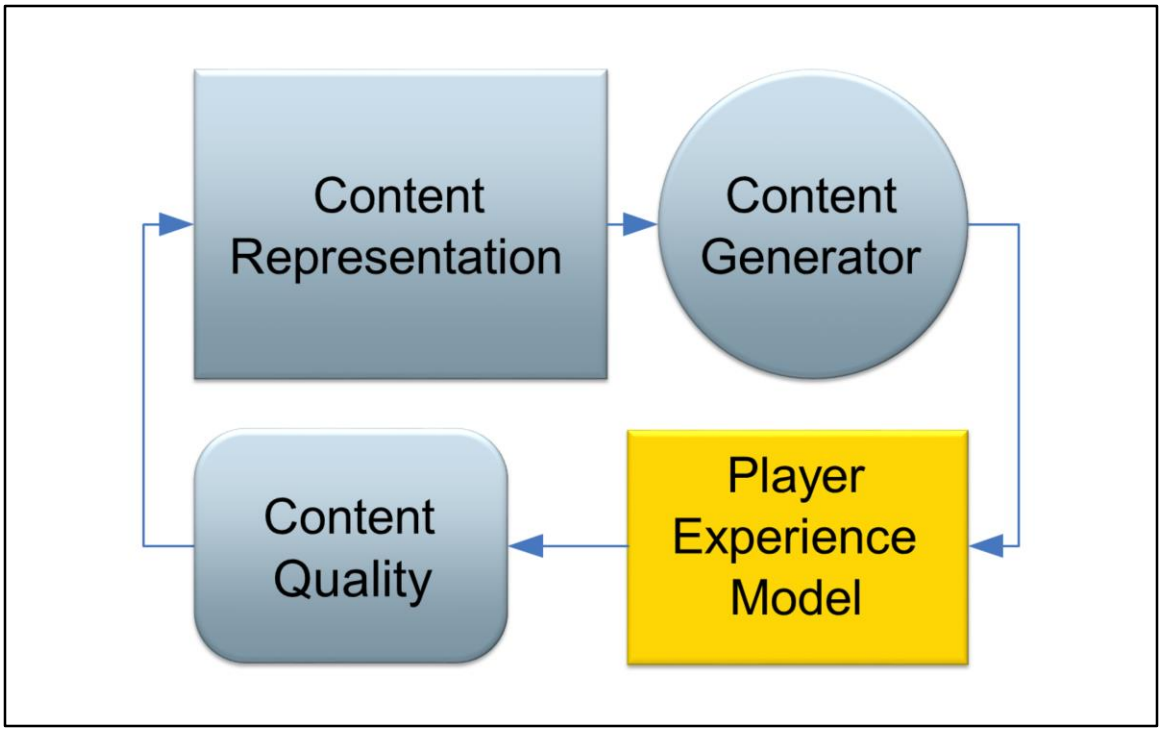
As EDPCG entails and considers experiences of players we first need to define what "player experience" is.

Key Components

The framework comprises of four key components which will look in further detail.



Since games are composed by game content that, when played by a particular player, elicits experience patterns, one needs to assess the quality of the content generated (linked to the experience of the player), search through the available content, and generate content that optimizes the experience for the player.

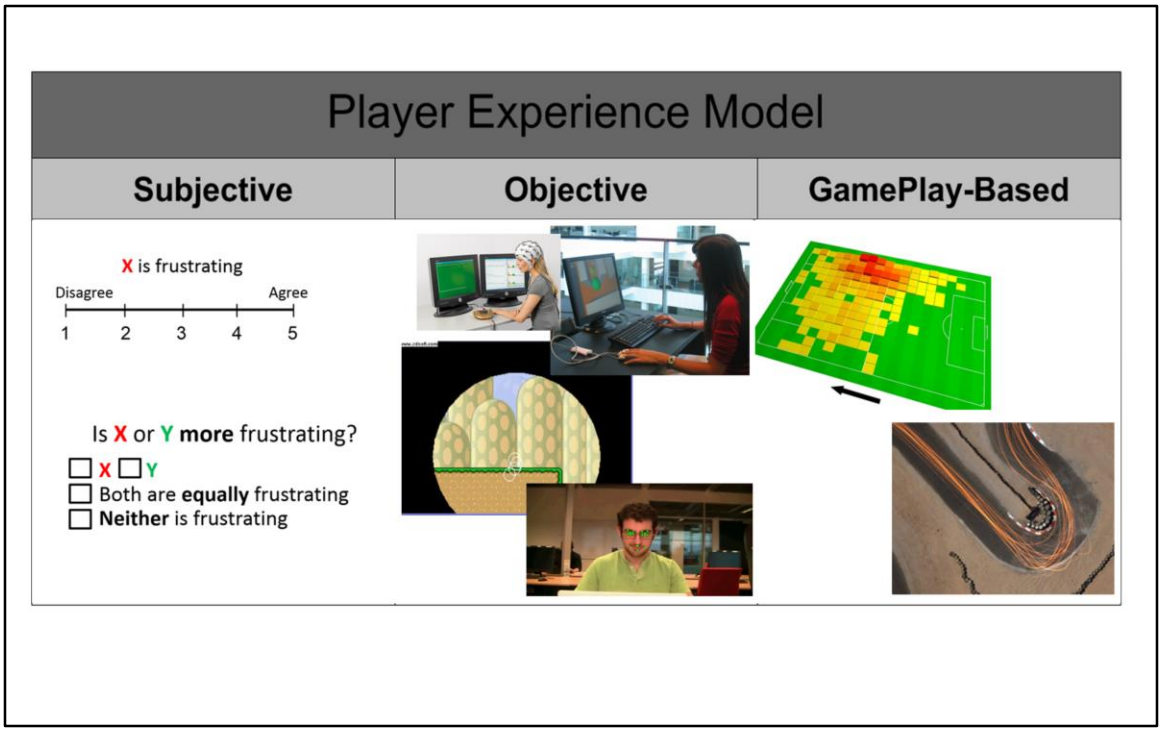


Player experience modeling: player experience is modeled as a function of game content and player.

Player Experience Modeling

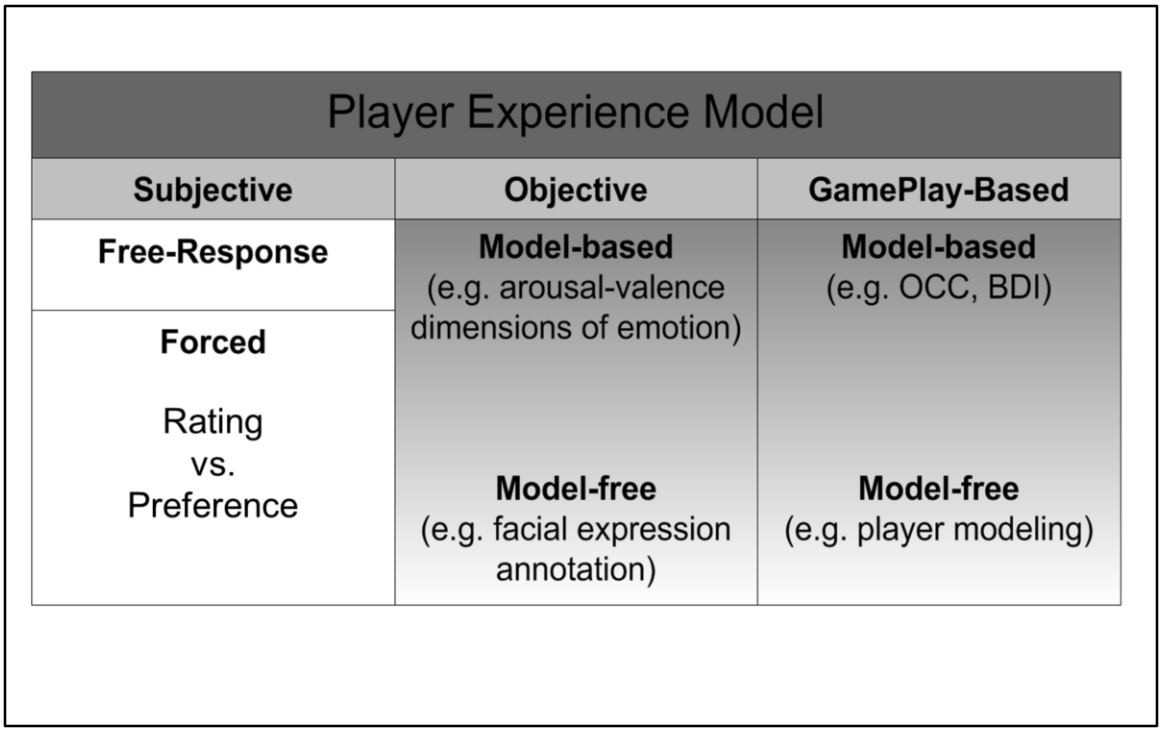
- Content evaluation functions could (should?) be based on player experience models
- Predict player experience from behavior/cognition/affect and/or game content
- Derived from empirical measurements of player experience

Yannakakis, G. N., & Togelius, J. (2011). **Experience-driven procedural content generation**. *Affective Computing, IEEE Transactions on*, *2*(3), 147-161.

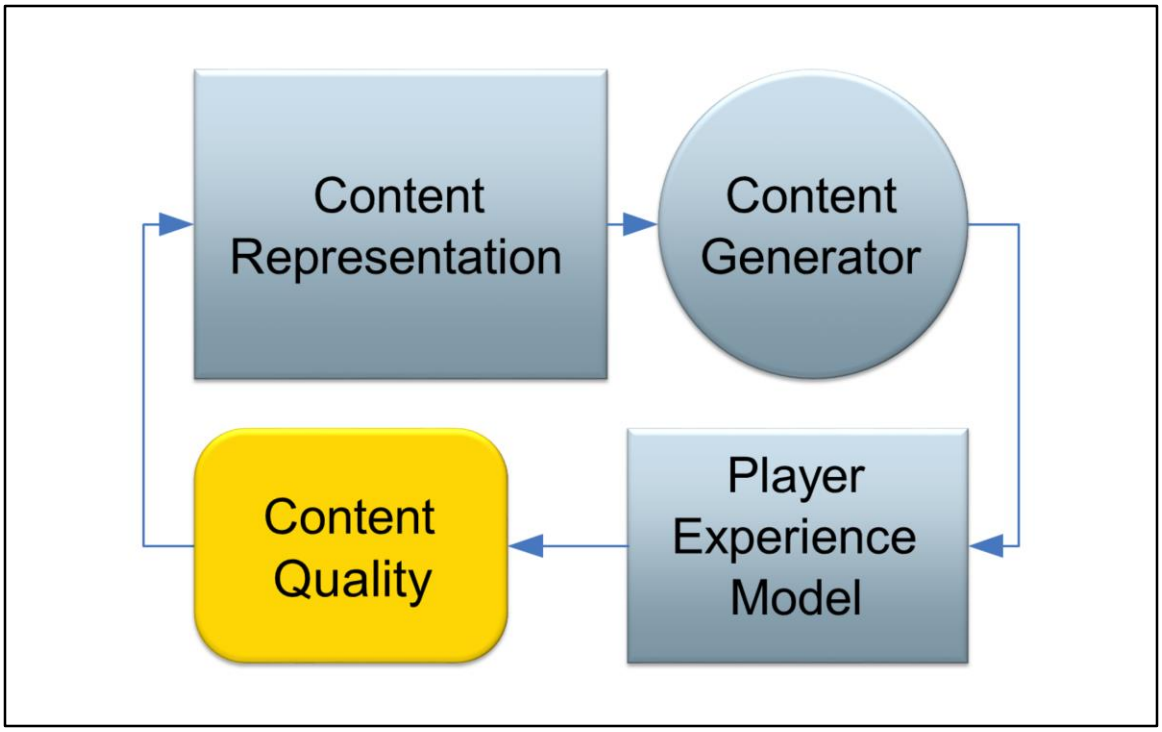


There are three data types considered mainly for the construction of a player experience model

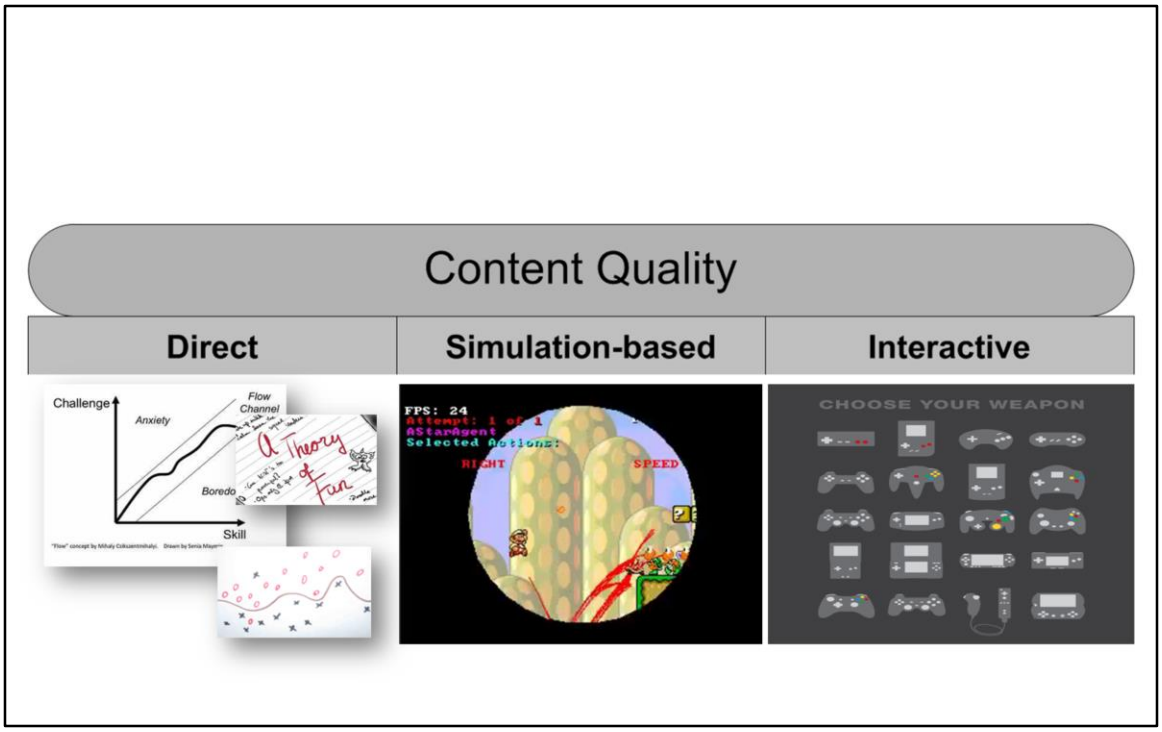
See more details at “Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. Affective Computing, IEEE Transactions on, 2(3), 147-161.”



See more details at "Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. Affective Computing, IEEE Transactions on, 2(3), 147-161."

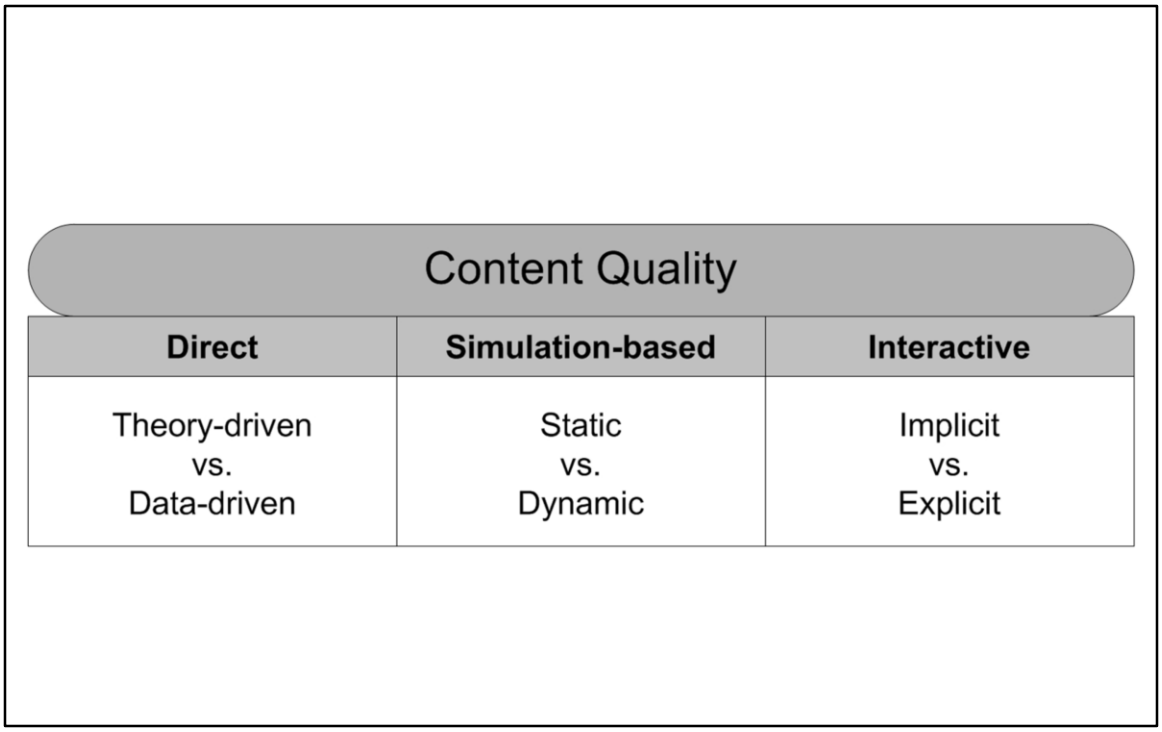


Content quality: the quality of the generated content is assessed and linked to the modeled experience



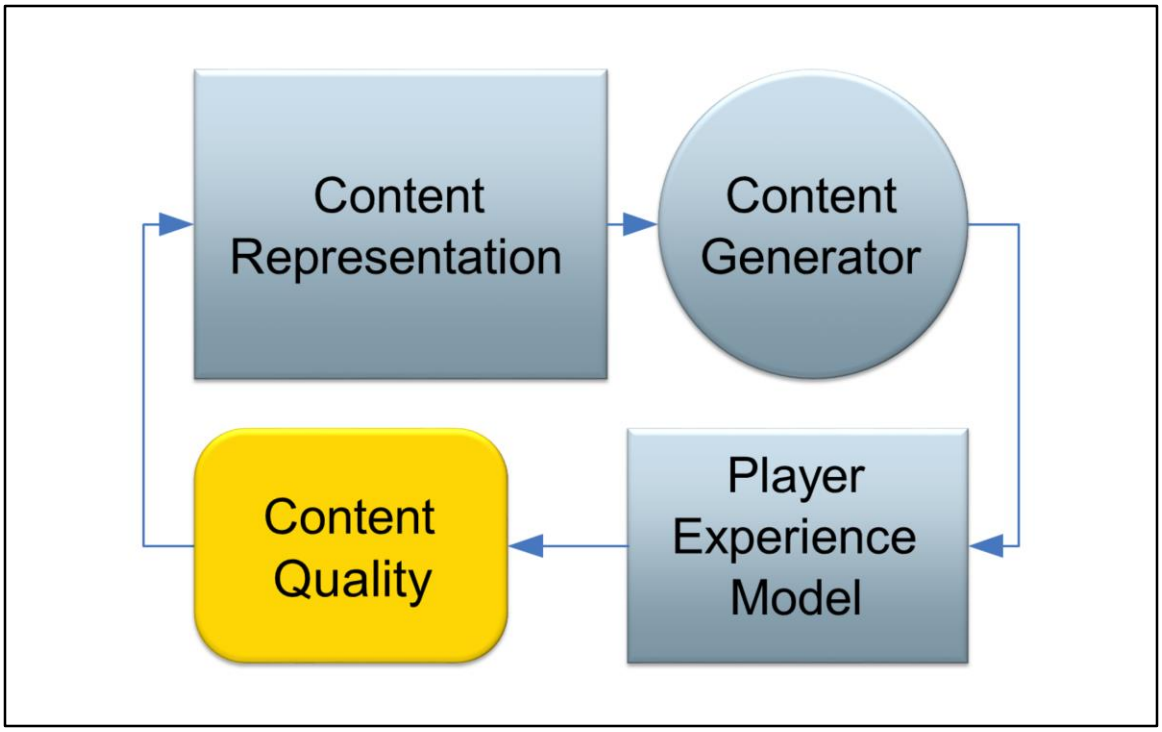
- **Directly (**A direct mapping between content and quality; e.g. number of jumps in a platform game)
  - –**Theory-driven**: evaluation function is based on a theoretical model – e.g. Koster's theory of fun)
  - –**Data-driven**: evaluation function is derived via gameplay (or other modalities of) data
- **Simulation-based (**An AI (maybe a human imitator) plays the game for a while and content is evaluated)
  - –**Static**: evaluation function does not change over time
  - –**Dynamic**: evaluation function is affected as time goes by
- **Interactively (**Real-time evaluation via a player (or players))
  - –**Implicit**: game behavior gives value to content (e.g. preference over a weapon)
  - –**Explicit**: ask players to score content

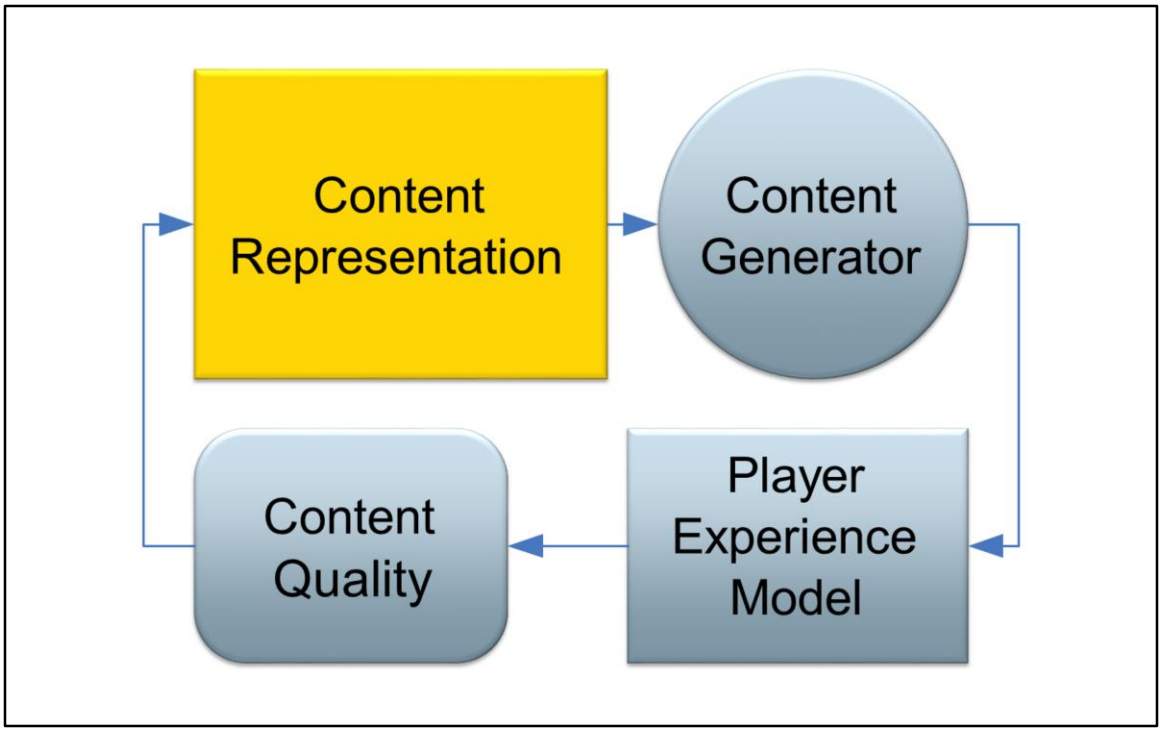
See more details at "Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. Affective Computing, IEEE Transactions on, 2(3), 147-161."



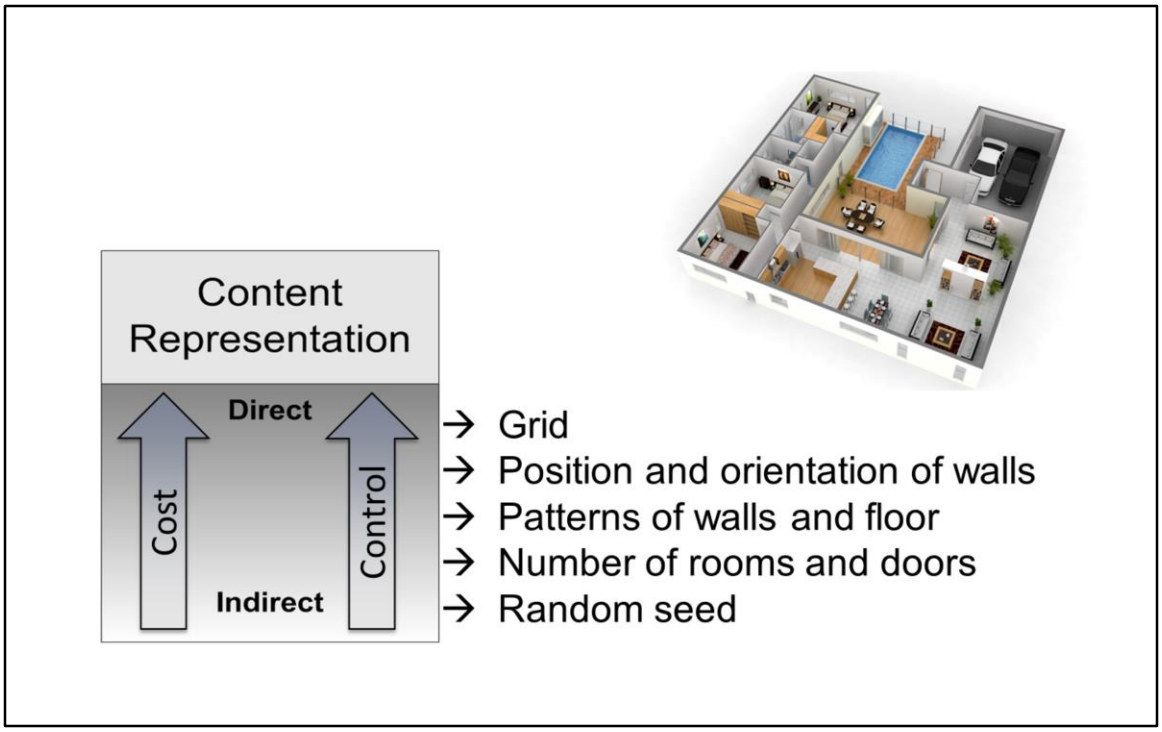
- **Directly (**A direct mapping between content and quality; e.g. number of jumps in a platform game)
  - –**Theory-driven**: evaluation function is based on a theoretical model – e.g. Koster's theory of fun)
  - –**Data-driven**: evaluation function is derived via gameplay (or other modalities of) data
- **Simulation-based (**An AI (maybe a human imitator) plays the game for a while and content is evaluated)
  - –**Static**: evaluation function does not change over time
  - –**Dynamic**: evaluation function is affected as time goes by
- **Interactively (**Real-time evaluation via a player (or players))
  - –**Implicit**: game behavior gives value to content (e.g. preference over a weapon)
  - –**Explicit**: ask players to score content

See more details at "Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. Affective Computing, IEEE Transactions on, 2(3), 147-161."


Content Representation
Content Generator
Player Experience Model
Content Quality



Content representation: content is represented accordingly to maximize search efficacy and robustness

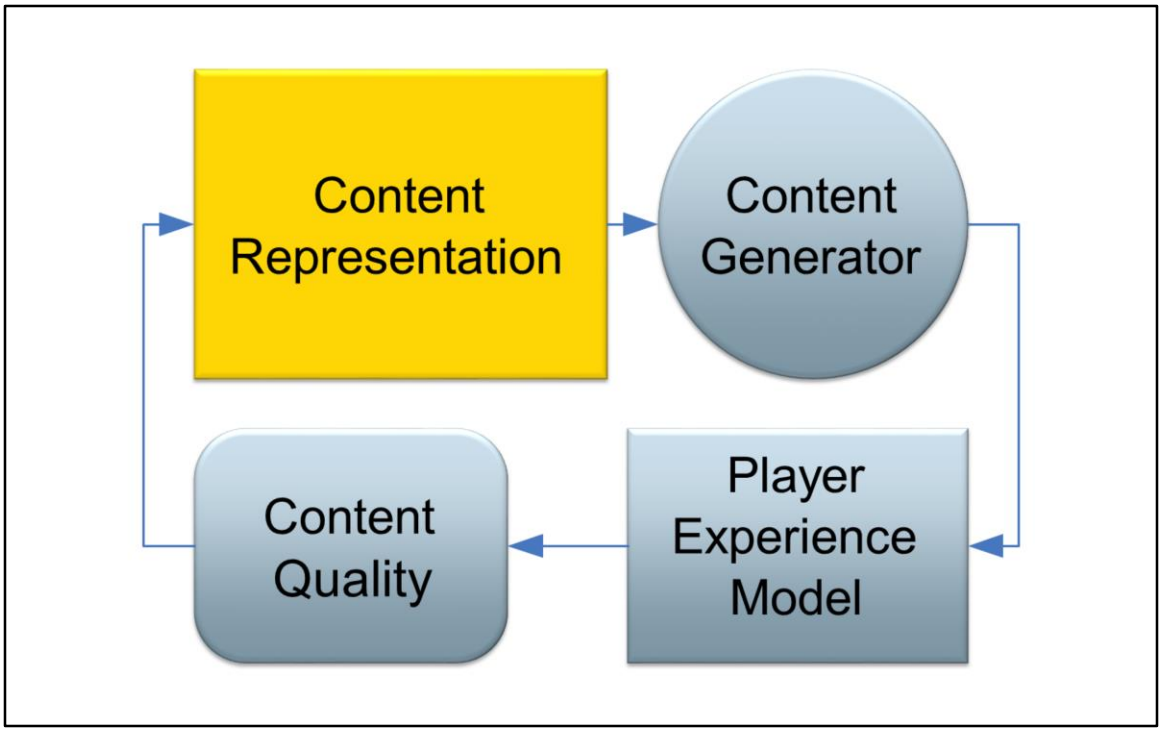


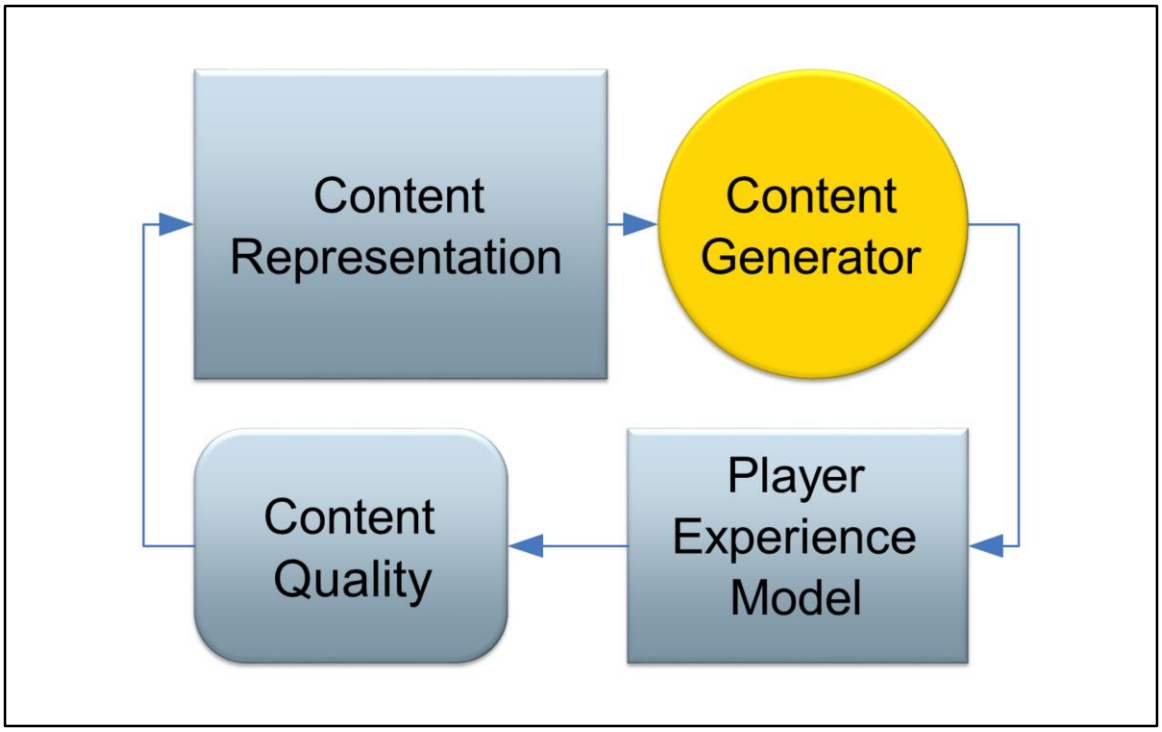
Here is an example of the different ways we could represent an apartment: all the way from very direct to very indirect approaches

In particular, the two ends of the content representation spectrum:

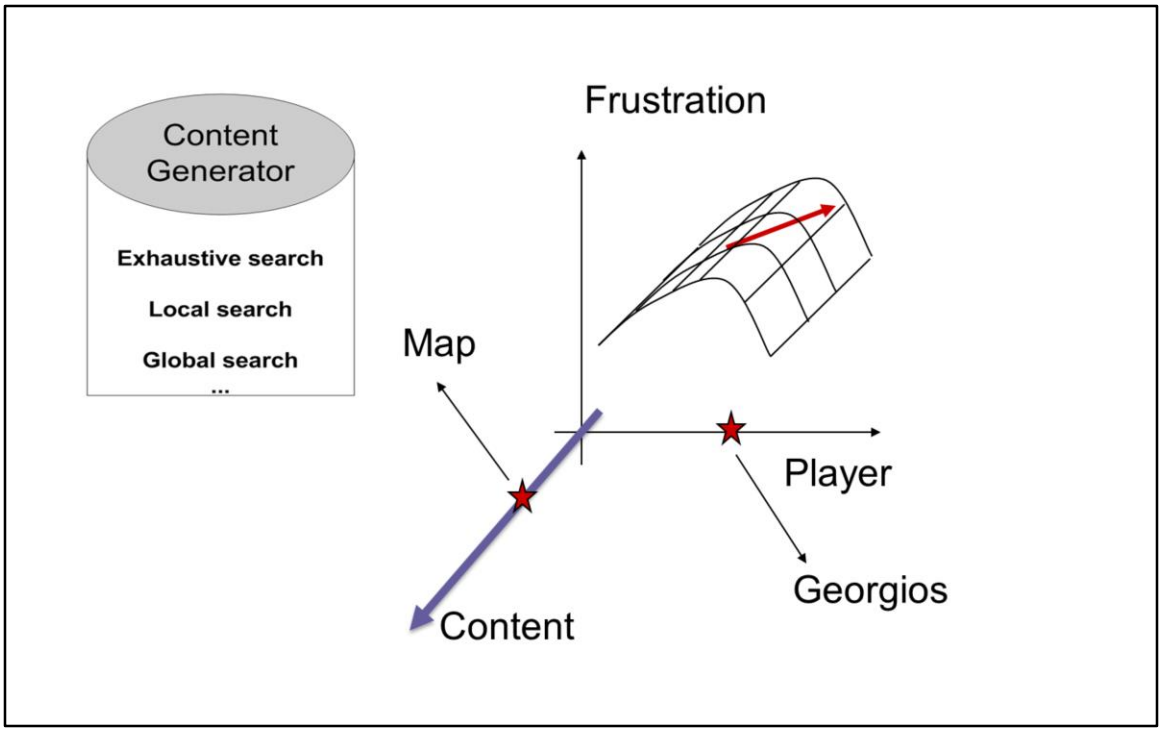
- Grid: costly but ultimate levels of control over the generated content
- Random seed: minimum cost but minimum control too

See more details at “Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. Affective Computing, IEEE Transactions on, 2(3), 147-161.”


Content Representation
Content Generator
Player Experience Model
Content Quality



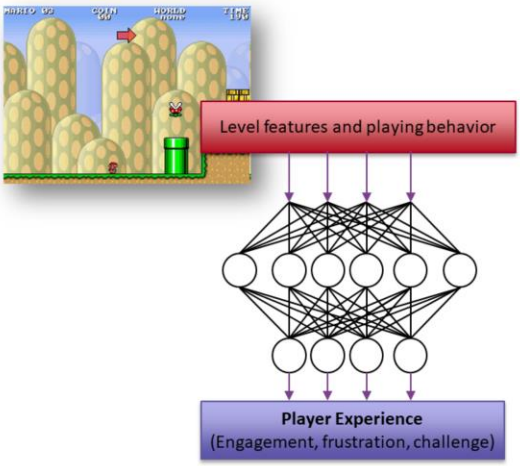
Content generator: the generator searches through the generative space for content that optimizes the experience for the player according to the acquired model.



Theoretically EDPCG maintains a mapping between experience (e.g. frustration in the graph), content (e.g. a map in the example graph) and manifestations of player behaviour (e.g. Georgios in the graph). Given that such a mapping exists any optimization method would be able to alter content parameters in order to optimize particular experiences for particular players, thereby personalising the interaction via content generation.

See more details at “Yannakakis, G. N., & Togelius, J. (2011). Experience-driven procedural content generation. Affective Computing, IEEE Transactions on, 2(3), 147-161.”

A Super Mario Bros Example

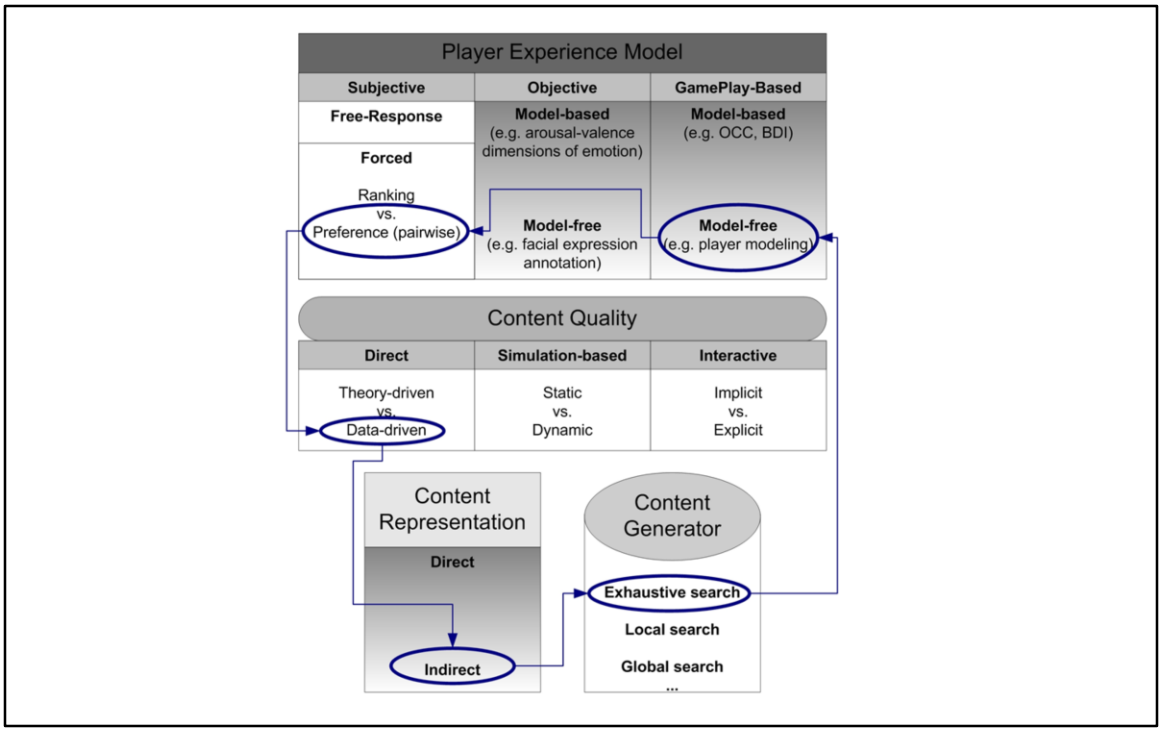


- **Model-free,** gameplay-based PEM (pairwise preferences as outputs)
- **Direct** (data-driven) evaluation function
- **Indirect** content representation (a few parameters)
- Search for good content via **exhaustive search**!

Shaker, N., Yannakakis, G. N., & Togelius, J. (2010, October). Towards Automatic Personalized Content Generation for Platform Games. In *AIIDE*.

For more details see

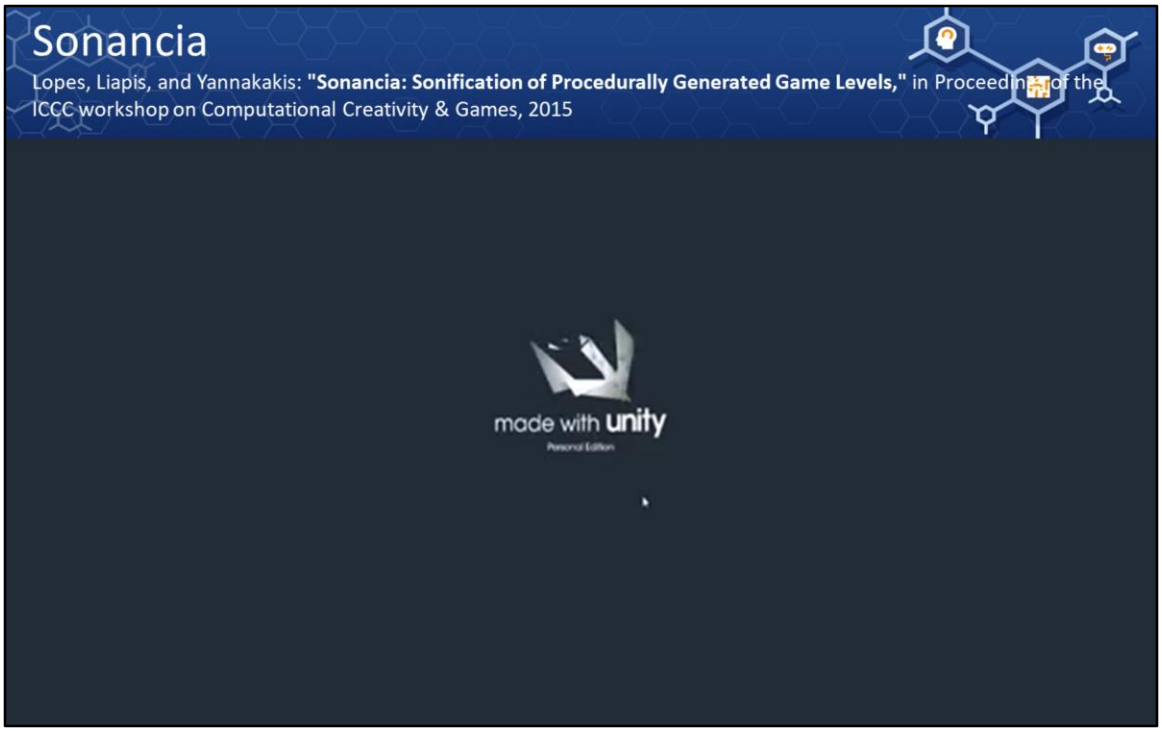
Shaker, N., Yannakakis, G. N., & Togelius, J. (2010, October). Towards Automatic Personalized Content Generation for Platform Games. In *AIIDE*.



For more details see

Shaker, N., Yannakakis, G. N., & Togelius, J. (2010, October). Towards Automatic Personalized Content Generation for Platform Games. In *AIIDE*.

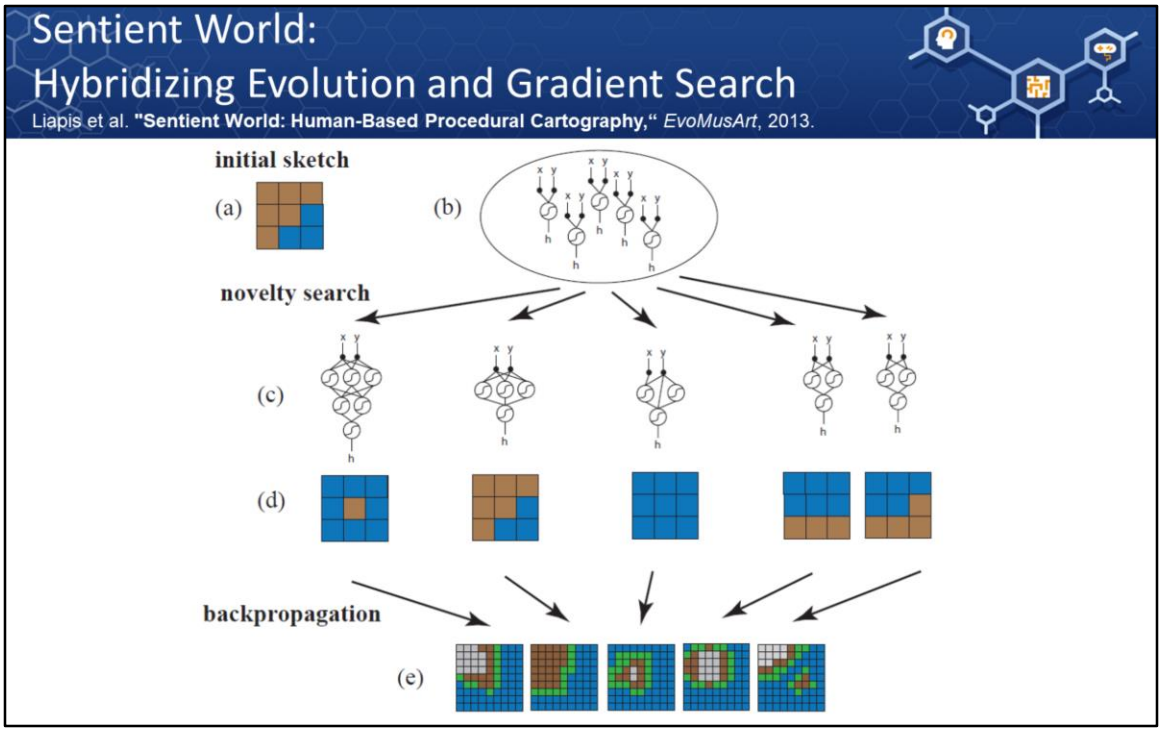
Other EDPCG Examples



Sonancia co-creates levels and sounds based on a tension curve provided by the designer. See referenced paper for more details. See also http://www.autogamedesign.eu/sonancia for videos and more papers.

- *Village Voices* (Best Learning Game 2013, European Serious Games Awards) is a game for teaching conflict resolution. It takes place in multiplayer setting in a medieval village
- The game puts players in conflict quests they have to resolve
- Players learn to resolve conflict collaboratively as that improve their status in the village and the relationship with the other villagers
- The game adapts /generates quests and events with the aim to maintain the conflict of the levels in appropriate levels (by escalating / de-escalating conflict)

Khaled and Yannakakis "Village Voices: An adaptive game for conflict resolution", in Proc. of FDG, pp. 425-426

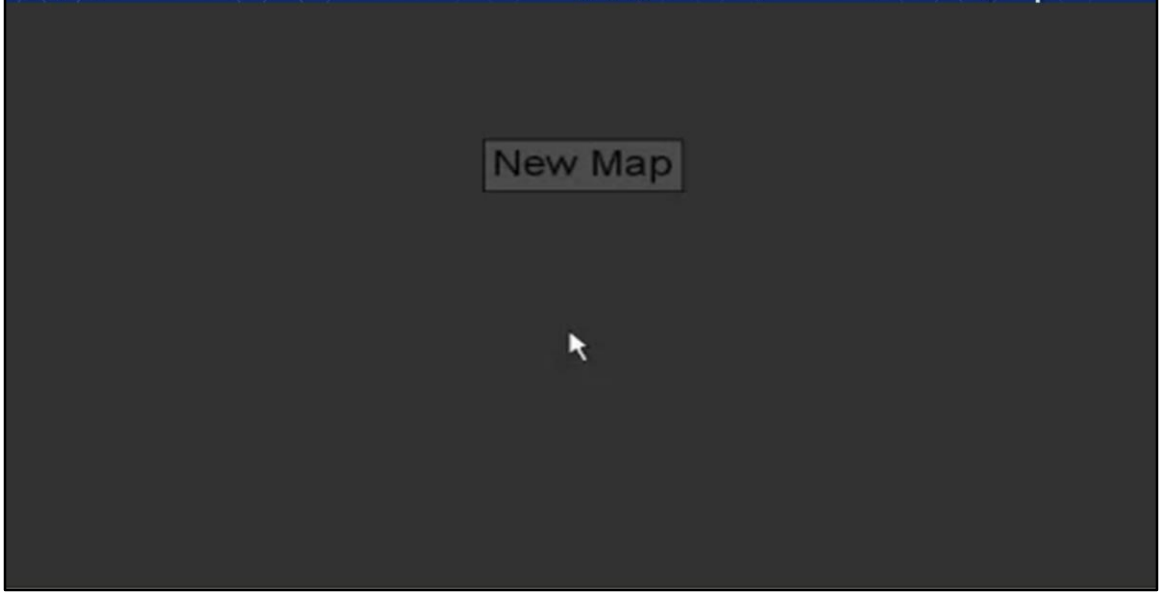


Sentient World hybridizes artificial (interactive) evolution and gradient search (backpropagation) to generate and recommend content during a map design process. The tool allows the designer to draw a rough terrain sketch, adding extra levels of detail through stochastic and gradient search. Novelty search generates a number of dissimilar artificial neural networks that are trained to approximate a designer's sketch and provide maps of higher resolution back to the designer. As the procedurally generated maps are presented to the designer (to accept, reject, or edit) the terrain sketches are iteratively refined into complete high resolution maps which may diverge from initial designer concepts. The tool supports designer creativity while conforming to designer intentions, and maintains constant designer control through the map selection and map editing options. Results obtained on a number of test maps show that novelty search is beneficial for introducing divergent content to the designer without reducing the speed of iterative map refinement.

For more details please refer to the cited paper

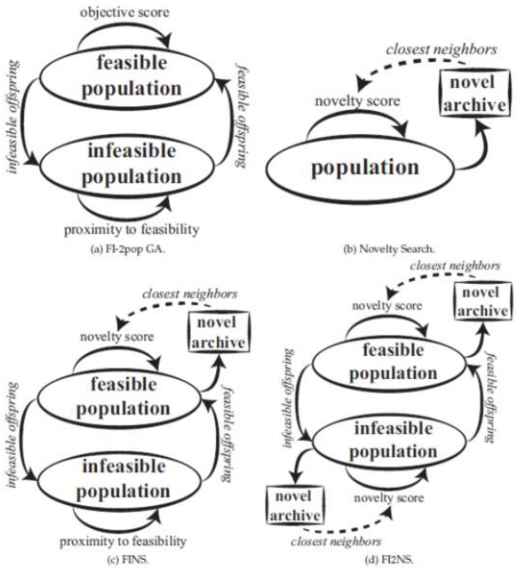
Sentient World:
Hybridizing Evolution and Gradient Search

Liapis et al. **"Sentient World: Human-Based Procedural Cartography,"** *EvoMusArt*, 2013.



Constrained Novelty Search for PCG

Liapis, Yannakakis and Togelius, **Constrained Novelty Search: A Study on Game Content Generation**, Evolutionary Computation, 21(1), 2015, pp. 101-129

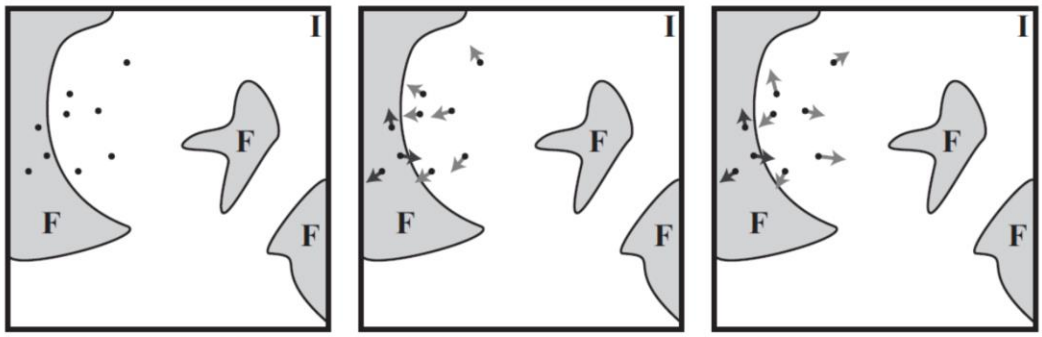


This slide introduces the idea of constraining novelty search for the purpose of generating both novel and valuable (feasible) content in games. One way to constrain novelty search is to integrate it within a feasible-infeasible 2-population genetic algorithms (FI-2pop GA). Further details about the algorithm can be found in

Liapis, Yannakakis and Togelius, **Constrained Novelty Search: A Study on Game Content Generation**, Evolutionary Computation, 21(1), 2015, pp. 101-129

Constrained Novelty Search for PCG

Liapis, Yannakakis and Togelius, **Constrained Novelty Search: A Study on Game Content Generation**, Evolutionary Computation, 21(1), 2015, pp. 101-129



(a) Feasible and Infeasible Spaces. (b) FINS search process. (c) FI2NS search process.

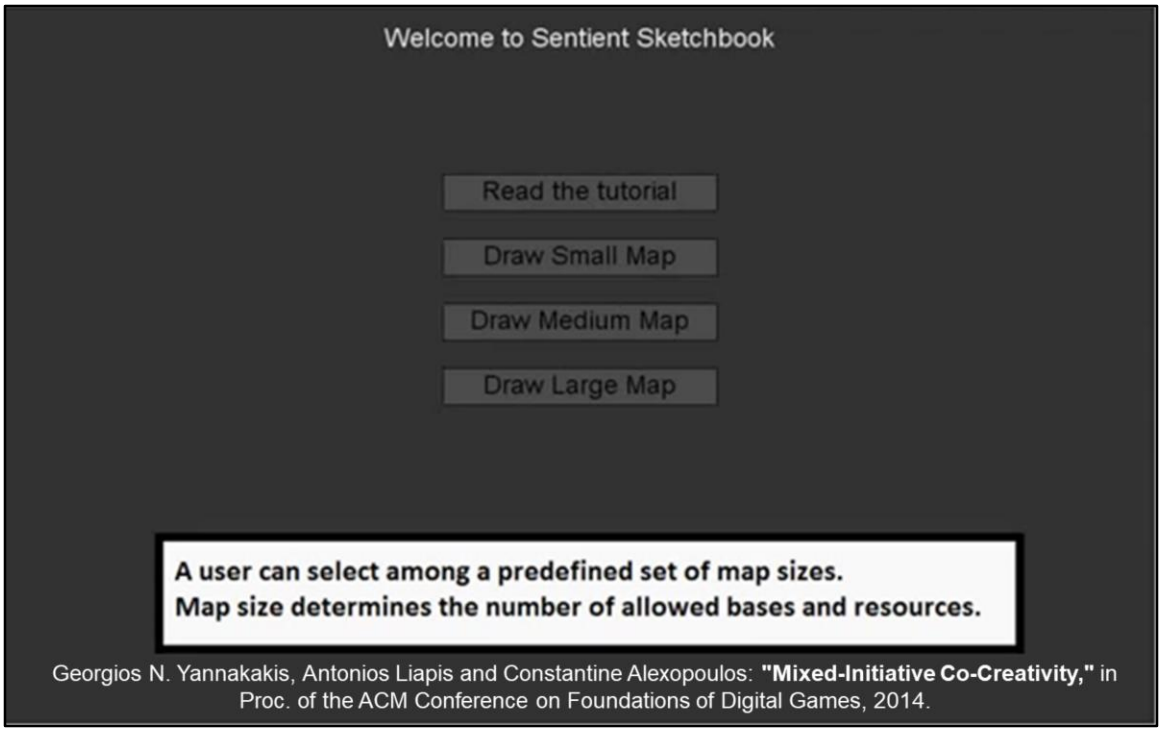
A graphical representation of how the introduced constrained novelty search algorithms (FINS and FI2NS) operate in a hypothetical feasible-infeasible search space. For more details please refer to the cited paper above.

Sentient Sketchbook

Georgios N. Yannakakis, Antonios Liapis and Constantine Alexopoulos:
"Mixed-Initiative Co-Creativity," in Proc. of the ACM Conference on Foundations of Digital Games, 2014.

- Map Sketches (strategy game, dungeon, FPS level)
- Multiple solutions evolved & shown in real-time
- Fitnesses on area influence, exploration and balance… and novelty
- Constraints on playability handled with FI-2pop GA

Constrained Novelty Search is a useful method for mixed-initiative PCG as it generates content that is appropriate for the task (within constraints) and also generates content that is novel thereby pushing the boundaries of the designer's creativity. The method has been used in the Sentient Sketchbook mixed-initiative tool which is presented here. For more details please refer to the cited paper.



This is a video demonstration of the Sentient Sketchbook mixed-initiative tool. For more details about the tool please refer to the cited paper above

What Could be Generated?

[See Section 4.5 for more details]

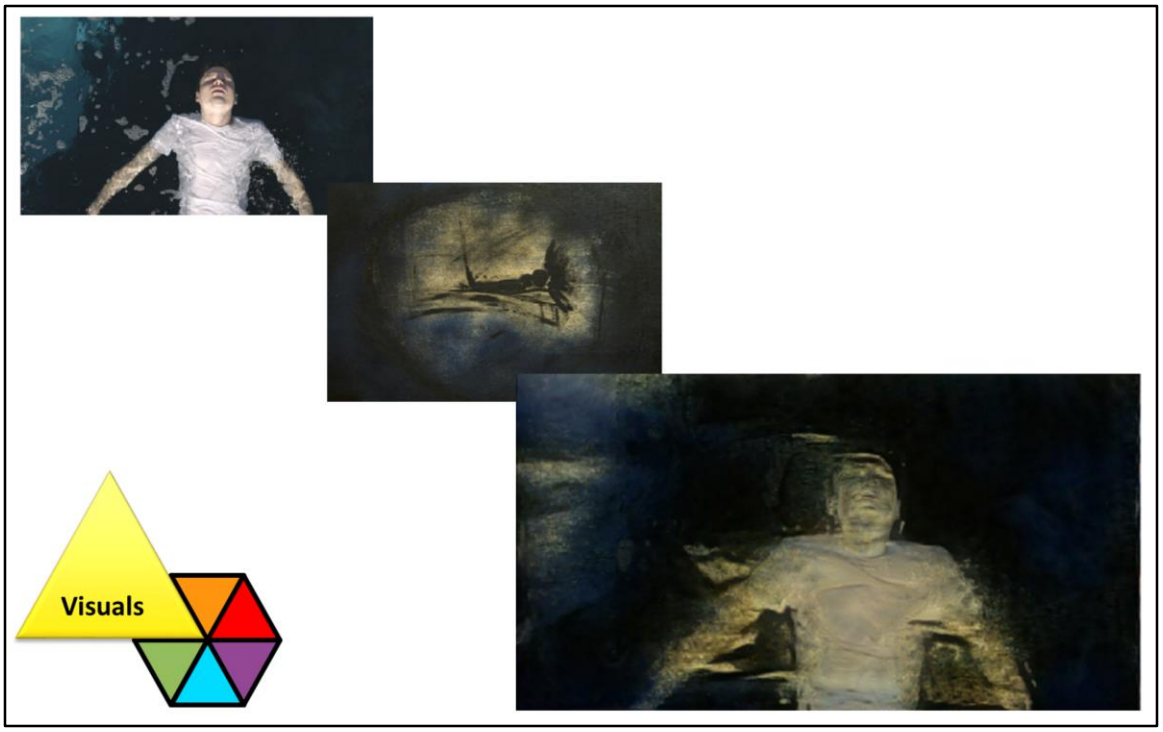
In the last part of this presentation we briefly outline the possible content types that a PCG algorithm can generate in a game.

[Reminder slide: see Section 1.3.5.2 for more details]

Generally speaking Liapis et al. identified six creative domains (or else facets) within games that we will follow for our discussion in this section. These include level architecture (design), audio, visuals, rules (game design), narrative, and gameplay. In this chapter we will cover the first five facets and we purposely exclude the gameplay facet. Creative gameplay is directly associated

with play and as such is covered in Chapter 3.

In the next slides we will see some indicative examples on each of the creative facets within games.



[See Section 4.5.2 for more details]

Games are, by definition, visual media unless the game is designed explicitly to not have visuals—e.g., the Real Sound: Kaze no Regret (Sega, 1997) adventure audio game. The visual information displayed on the screen conveys messages to the player which are dependent on the graphical style, color palette and visual texture. Visuals in games can vary from simple abstract and pixelized representations as the 8-bit art of early arcade games, to caricaturized visuals as in Limbo (Playdead, 2010), to photorealistic graphics as in the FIFA series (EA Sports, 1993).

Image: Impressionism Neural Style Transfer that redraws key scenes in the short-movie *Come Swim* (dir. Kristen Stewart)

[See Section 4.5.3 for more details]

Even though audio can be seen as optional content it can affect the player directly and its impact on player experience is apparent in most games . Audio in games has reached a great level of maturity as demonstrated by BAFTA Game Awards and MTV video music awards for best video game soundtrack. The audio types one meets in games may vary from fully orchestrated soundtrack (background) music, as in Skyrim (Bethesda, 2011), to sound effects, as the dying or pellet-eating sounds of Pac-Man (Namco, 1980), to the voice-acted sounds of Fallout 3 (Bethesda, 2008).

A notable example of adaptive music creation in games is the work of Brown (see video and refer to the cited article for more details)

[See Section 4.5.4 for more details]

Many successful games are relying heavily on their narratives; the clear distinction however, between such narratives and traditional stories is the interactivity element that is offered by games. The study of computational (or procedural) narrative focuses on the representational and generational aspects of stories as those can be told via a game. Stories can play an essential part in creating the aesthetics of a game which, in turn, can impact affective and cognitive aspects of the playing experience. In terms of procedurally generated narrative, planning approaches target typicality as in the case of Facade (see screenshot) which tries to tell a story of a couple having marriage problems.

Another game narrative example: Orkin and Roy (2007) use a lexicon of actions and utterances from data of over 5000 players in a simple restaurant game to train virtual agents' verbal responses based on N-grams;

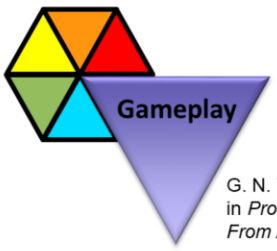
[see Chapter 3 for more details]

Game exploits: Creative gameplay can also be found when the players discover bugs in the game and exploit them to their advantage, such as placing a bucket on the head of an NPC in *Skyrim* and robbing them blind.

There is also **freeform/subversive** gameplay, which is about playing the game completely disregarding the game's intentions, such as proposing in Minecraft. Imagine computational gamers doing that? Wouldn't that be considered creative?

There is obviously a lot of work on AI agent control in games. EC is particularly interesting as it is prone to find bugs in the game and exploit them.

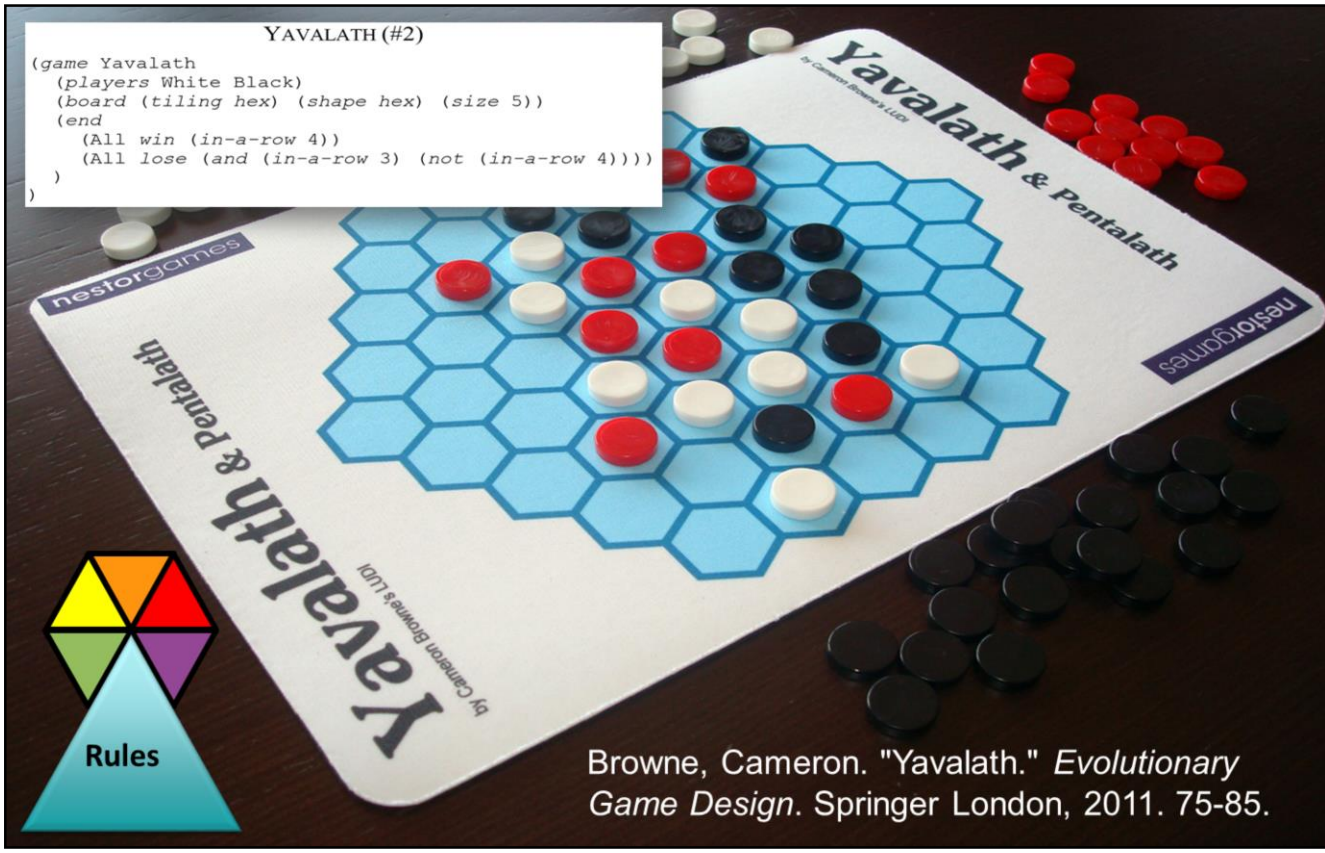
Creative gameplay example: In FIFA the evolutionary algorithm found that the best way to score a goal was by **forcing a penalty kick**. For more details please refer to the cited paper



G. N. Yannakakis, and J. Hallam, "**Evolving Opponents for Interesting Interactive Computer Games**," in *Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04); From Animals to Animats 8*, pp. 499-508, Los Angeles, CA, USA, July 13-17, 2004. The MIT Press.

Another creative gameplay example: Pac-man game interestingness: Ghosts are interesting if they are 1) Challenging, 2) Diverse and 3) Spatially Curious

See cited paper from more details



[see Section 4.5.5 for more details; See also earlier slides of this deck - this is a reminder slide about Yavalath and Ludi]

The game rules frame the playing experience by providing the conditions of play—for instance, winning and losing conditions—and the actions available to the player (game mechanics). Rules constitute necessary content as they are in a sense the core of any game, and a game's rules pervade it.

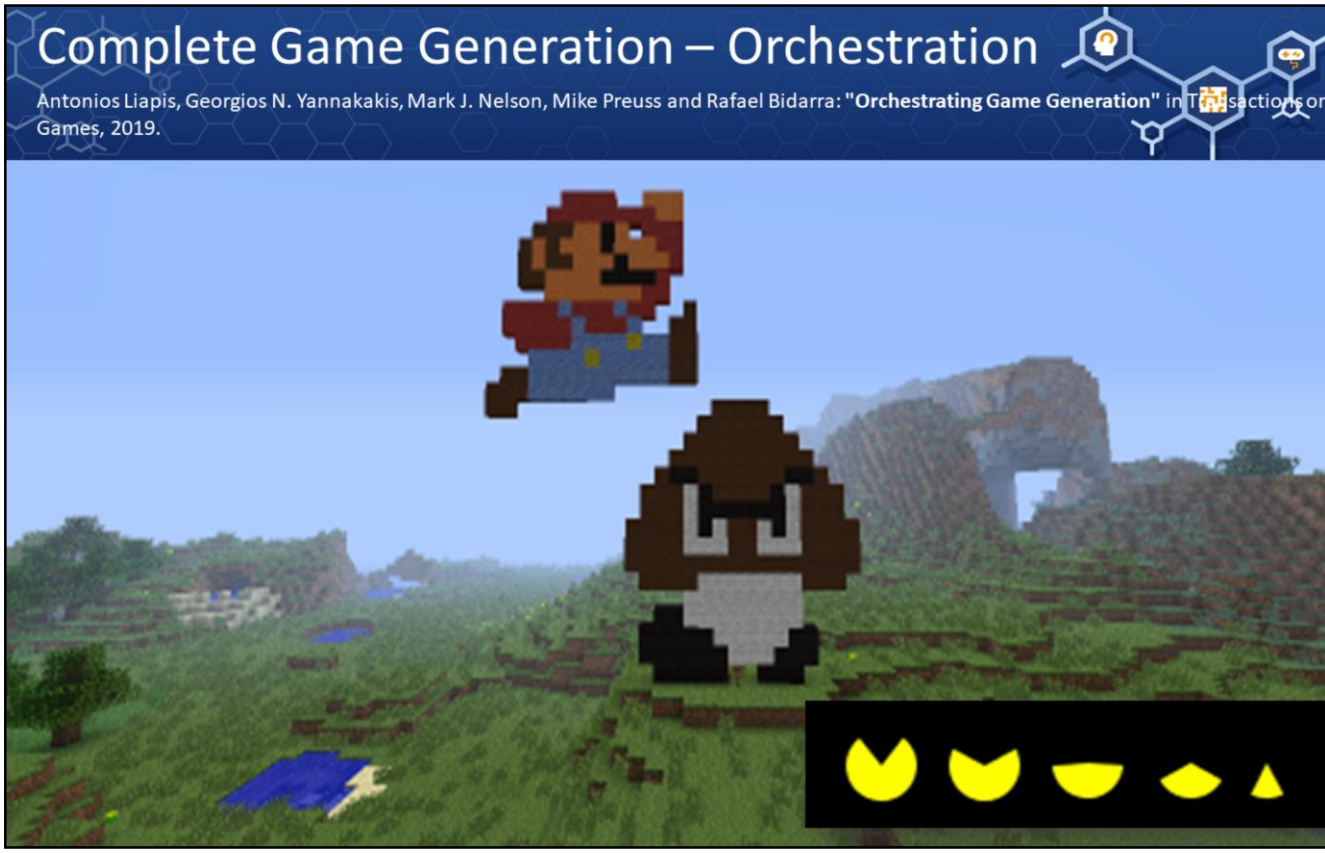
In terms of Procedural Generation of Rules, perhaps the most successful work is that of Cameron Browne where the computer generates the layout and rules of complete board games. Here we see Yavalath, a commercially available game generated entirely by a computer (including its name). Yavalath has been ranked within the top #100 abstract board games ever!

Top figure: the evolved grammar that represents the Yavalath game (hex tiling of size 5) and its simple rules:
- Lose: 3 tokens in a row
- Win: 4 tokens in a row

[See section 4.5.1 for more details]

The generation of levels is by far the most popular use of PCG in games. Levels can be viewed as necessary content since every game has some form of spatial representation or virtual world within which the player can perform a set of actions. The properties of the game level, in conjunction with the game rules, frame the ways a player can interact with the world and determine how the player can progress from one point in the game to another. The game's level design contributes to the challenges a player faces during the game. While games would often have a fixed set of mechanics throughout, the way a level is designed can influence the gameplay and the degree of game challenge. For that reason, a number of researchers have argued that levels coupled with game rules define the absolutely necessary building blocks of any game; in that regard the remaining facets covered below are optional. The variations of possible level designs are endless: a level representation can vary from simple two-dimensional illustrations of platforms and coins—as in the Super Mario Bros (Nintendo, 1985) series—to the constrained 2D space of Candy Crush Saga (King, 2012), to the three-dimensional and large urban spaces of Assassin's Creed (Ubisoft, 2007) and Call of Duty (Infinity Ward, 2003), to the 2D elaborated structures of Angry Birds (Rovio, 2009), to the voxel-based open gameworld of Minecraft (Mojang 2011).

Image: PCG for levels to generate dungeons as in Diablo



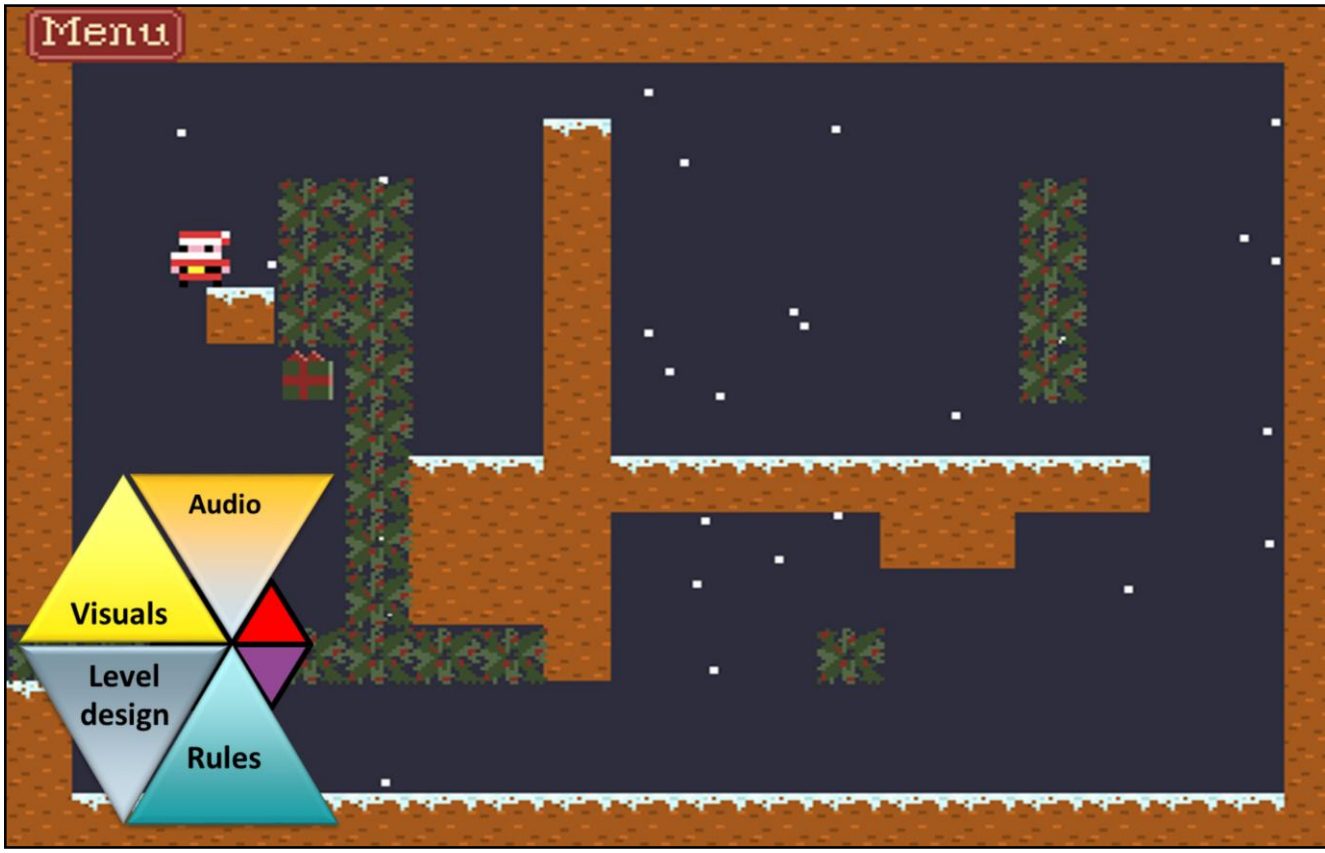
[see Section 4.5.6 for more details]

Game generation refers to the use of PCG algorithms for computationally designing new complete games. The vast majority of PCG studies so far, however, have been very specific to a particular game facet or domain. It is, for instance, either a level that is generated or the audio for some level but rarely both. Meanwhile it is surprising to think that the relationship between the different facets is naturally interwoven. A characteristic example of the interwoven nature among game facets: player actions—viewed as a manifestation of game rules—are

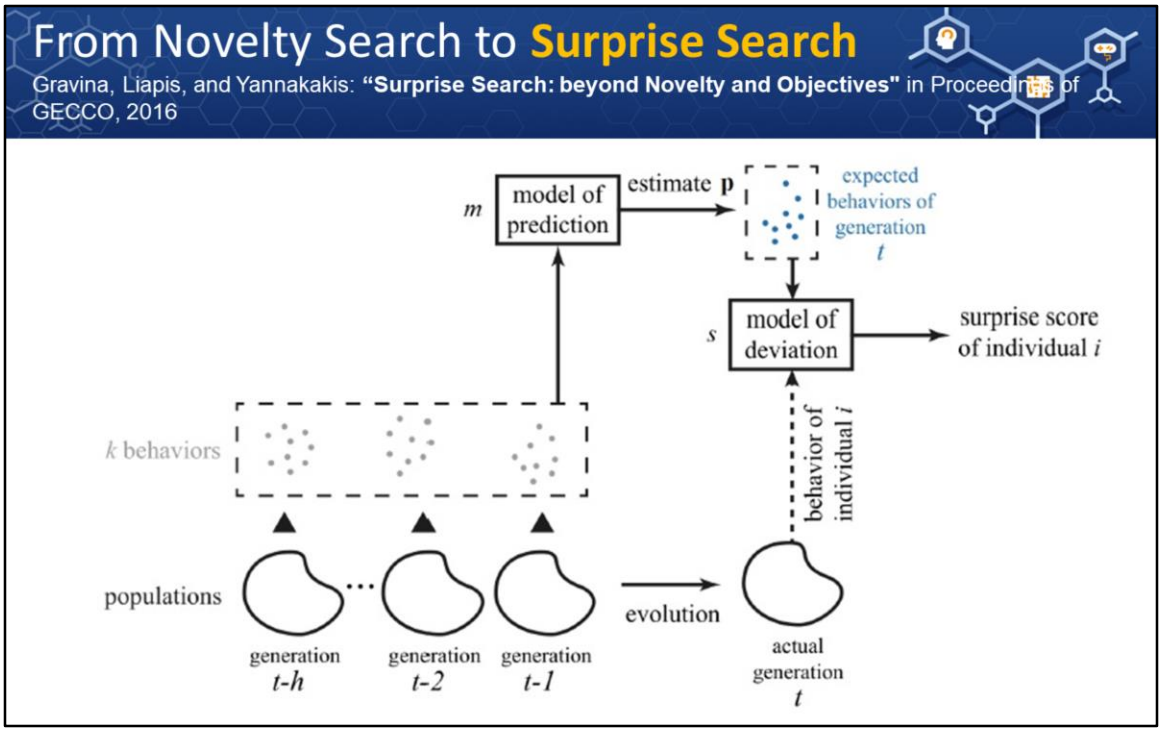
usually accompanied by corresponding sound effects such as the sound of Mario jumping in Super Mario Bros (Nintendo, 1985). Now let us think of a PCG algorithm that introduces a new rule to the game—hence a new player action. The algorithm automatically constrains the sound effects that can be associated to this new action based on a number of factors such as the action's duration, purpose and overall contribution to the game plot. Actions and sounds appear to have a cause and effect (or hierarchical) relationship and a PCG algorithm would naturally prioritize the creation of the action before it generates its sound. Most relationships between facets, however, are not strictly hierarchical or unidirectional. For example, a game

level can be successful because of a memorable landmark as much as the gameplay it affords. Similarly, the narrative of a game relies on a multitude of factors including the camera placement as well as the visuals and the sounds.

Orchestration of creative domains is the process of meaningfully generating content together



An example of an attempt towards complete game generation: ANGELINA's Puzzling Present game. The game features an invert gravity mechanic that allows a player to overcome the high obstacle on the left and complete the level. Image obtained with permission from http://www.gamesbyangelina.org/



Beyond novelty search as a means of generating novel content Gravina et al were inspired by the self-surprise mechanism of humans during creative problem solving. They created a computational model of surprise which can drive evolutionary search with the aim to both solve problems unconventionally and generate unexpected content in games.

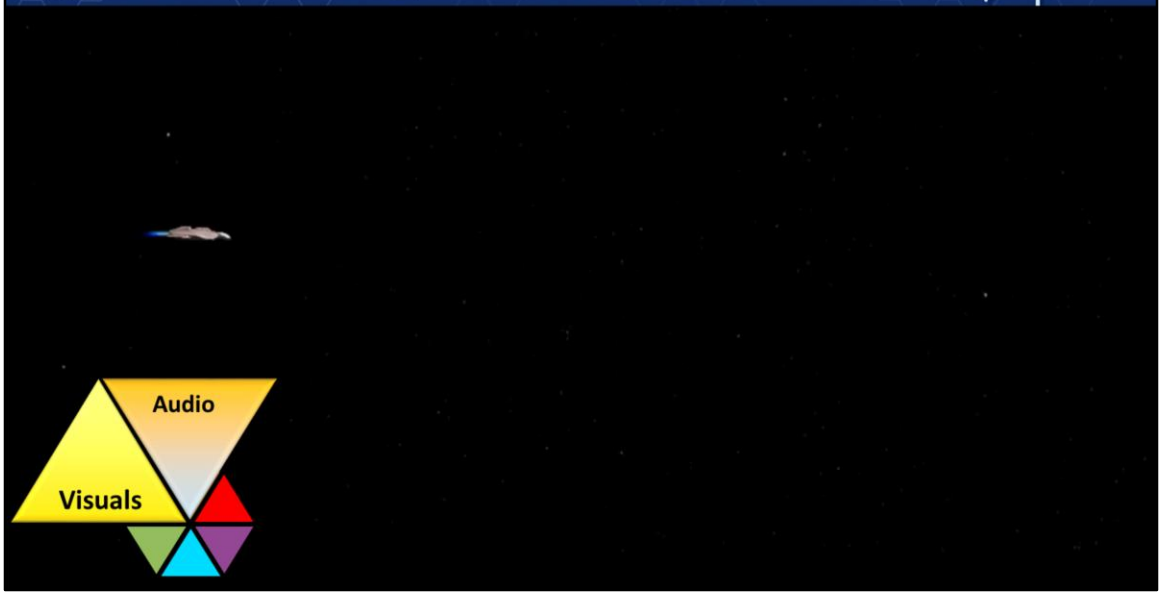
Surprise search maintains a predictive model (m) of where search (iin the behavioural space) is expected to be and rewards behaviors that deviate from this prediction (surprise score *s*). To predict future behaviours surprise search considers *h* generations in the past.

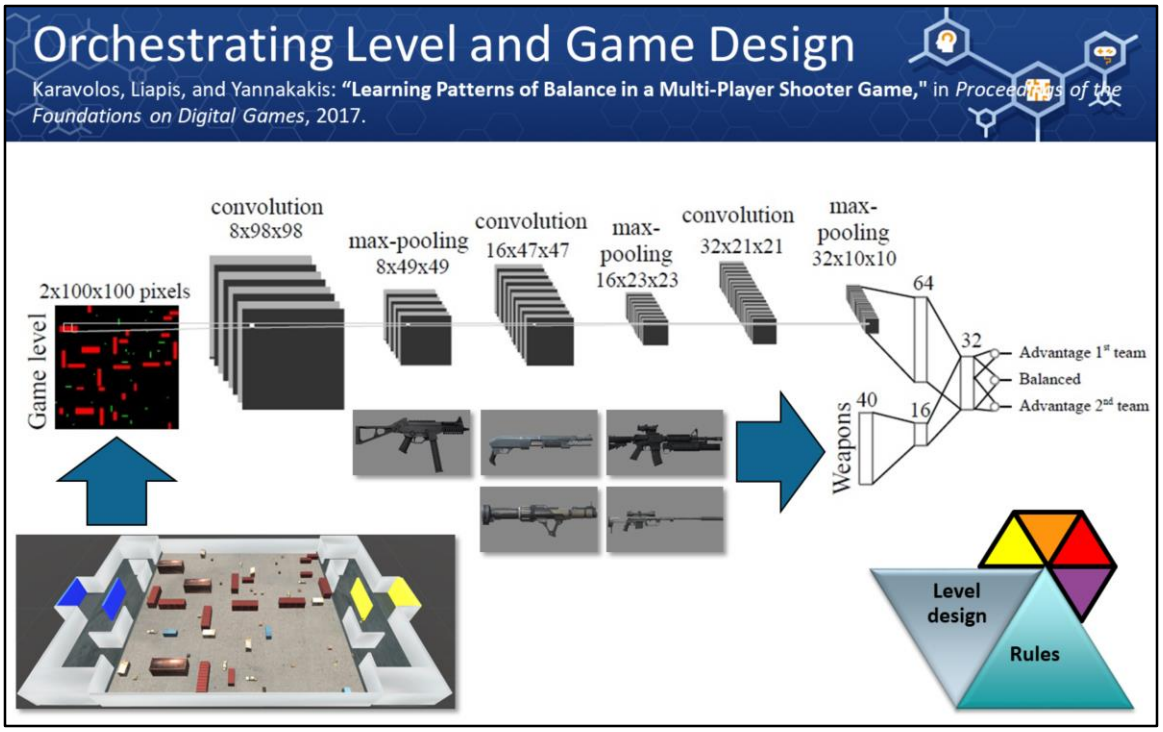
Further details on surprise search can be found in the referenced paper.

An application of constrained surprise search is on weapon generation in Unreal Tournament 2004. As in constrained novelty search the FI-2pop algorithm is used to constrain surprise search. A number of weapon parameters are generated so as to satisfy particular constraints (e.g. balance) while maximizing the surprise score. The outcomes are both useful for designers (i.e. balanced weapons) and surprising at the same time. For more details refer to the cited paper.

AudioInSpace: From Music to Weapons!

Hoover, Cachia, Liapis, Yannakakis, **"AudioInSpace: A Proof-of-Concept Exploring the Creative Fusion of Generative Audio, Visuals and Gameplay,"** in EvoMusArt, 2015





A convolutional neural network (CNN) architecture used for fusing levels and weapons in first-person shooters. The network is trained to predict whether a combination of a level and a weapon would yield a balanced game or not. The CNN can be used to orchestrate the generation of a balanced level given a particular weapon and vice versa. Please refer to the cited paper for more details.

Evaluating Content Generators

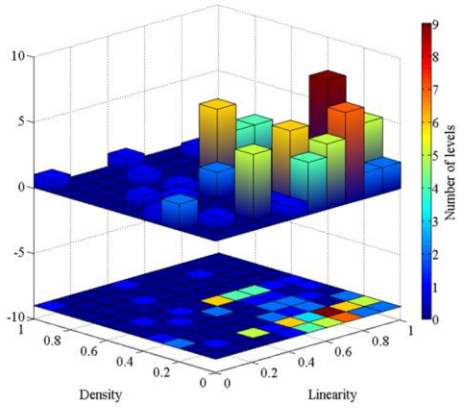
[See Section 4.6 for more details]

Creating a generator is one thing; evaluating it is another. Regardless of the method followed all generators shall be evaluated on their ability to achieve the desired goals of the designer. Arguably, the generation of any content is trivial; the generation of valuable content for the task at hand, on the other hand, is a rather challenging procedure. (One may claim, however, that the very process of generating valuable content is also, by itself, trivial as one can design a generator that returns a random sample of hand-crafted masterpieces.) Further, it is more challenging to generate content that is not only valuable but is also novel or even inspiring.

How Can we Evaluate a Content Generator?

Generally speaking there are three ways:

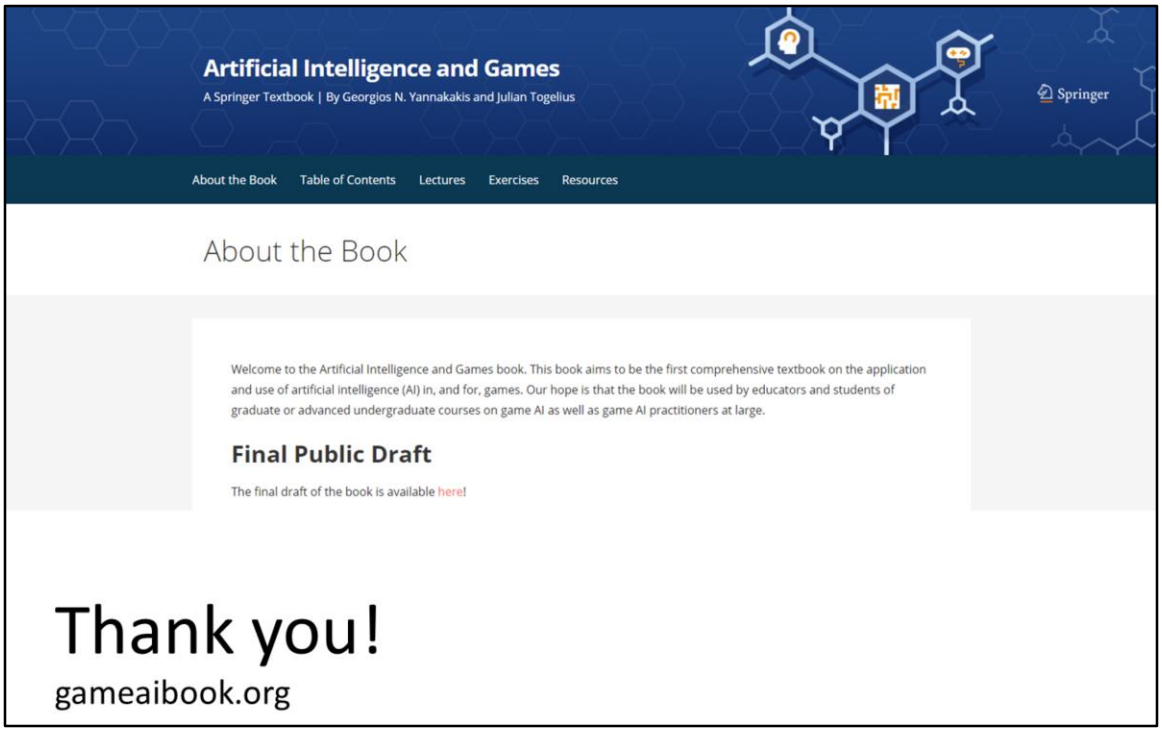
- Visualization (e.g. expressive range)
- AI (playtesting / personas)
- Human players (testing, QA, annotations)



[See Section 4.6.3 for more details]

Figure: The expressive range of the Ropossum level generator for the metrics of linearity and density.

For more details see: Mohammad Shaker, Mhd Hasan Sarhan, Ola Al Naameh, Noor Shaker, and Julian Togelius. Automatic generation and analysis of physics-based puzzle games. In Computational Intelligence in Games (CIG), 2013 IEEE Conference on. IEEE, 2013.



Thank you!

gameaibook.org