Modern Game AI Algorithms
Procedural Content Generation

Mike Preuss | LIACS

Universiteit Leiden
The Netherlands

roadmap

lecture plan (Thursdays 14:15), https://smart.newrow.com/room/nr2/?room_id=wft-364

Feb	10	introduction	Mar	31	player experience modeling
	17	learning and optimization	Apr	7	believable behavior
	24	procedural content generation I		14	team AI and mass behavior
Mar	3	Monte Carlo tree search		21	realtime strategy AI
	10	experimentation / ANN / GAN		28	(game) AI outlook with Zhao Yang
	17	learning from pixels	May	12	free project presentations 1
	24	procedural content generation II		19	free project presentations 2

labs plan (Thursdays 16:15)

Feb	10	no lab	Mar	31	deadline ass. 2 / start free proj
	17	single assignment: Minecraft house	Apr	7	free projects (assignment 3)
	24	single assignment (week 2)		14	free projects (assignment 3)
Mar	3	single assignment (week 3)		21	free projects (assignment 3)
	10	single ass. deadline / start ass 2		28	free projects (assignment 3)
	17	assignment 2	May	12	
	24	assignment 2		26	free project submission deadline

assessment

what is graded:

- single assignment 25%, smaller group assignment 30%,
  free project assignment 45% (including presentation), no written exam

the single assignment (Feb 17):

- simple assignment uses Python based GUI tool for Minecraft add-ons
- task is to procedurally create a house

smaller group project (second week of March):

- group work, new environment for strategy games, will have internal competition
- assumed working groups approx. 3 people, exceptions possible

free group project (free means you choose the topic):

- working in small groups (4-5 people, exceptions possible)
- during the second half of the labs, and in your preparation time

Discover the world at Leiden University

topics of today

- defining Procedural Content Generation
- the search-based PCG paradigm
- SBPCG examples (incl. mixed initiative)
- virtual landscapes/Procedural Modeling
- L-Systems
- Wave Function Collapse (WFC)
- facet orchestration

picture from Michal Jarmoluk on Pixabay

what is procedural content generation?

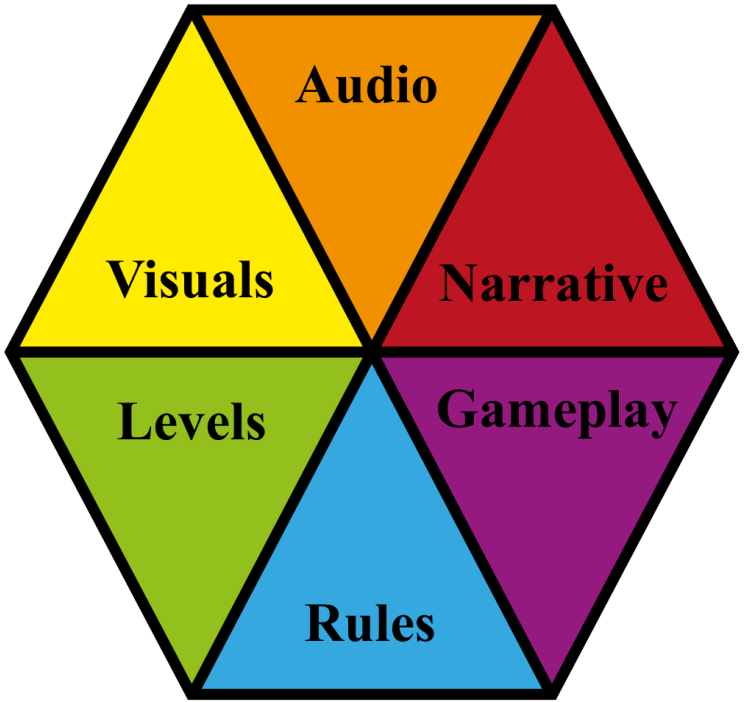
- procedural generation:
  little or no human intervention, algorithmic
- content:
  not NPC behavior, not game engine, everything that relates to gameplay (however, there are new approaches that affect NPC behavior)
- games:
  in principle, all types of games, in a rather limited fashion also for board games (etc. Settlers of Catan, simple map building algorithm is executed by players)

picture from David Mark on Pixabay

what is game content?

- levels, maps, dungeons
- terrains, buildings, plants
- artefacts, weapons, vehicles, clothing
- puzzles, quests, dialogs, characters
- camera movement, parameters

- facets of generation: visuals, audio, rules, levels, gameplay, narrative



Diablo III

Civilization VI

Borderlands 3

https://www.theverge.com/2019/8/14/20803683/borderlands-3-preview-gearbox-interview-social-media-streamers-vlogging

No Man's Sky

why PCG?

- improve efficiency of the creative process
- adapt games to the preferences of players
- new types of games or game contents which are not possible by means of manual content creation

picture from Peggy und Marco Lachmann-Anke on Pixabay

difficulties

- speed
  fast enough for realtime content generation?
- reliability
  catastropic mistakes destroy gameplay
- controllability
  how can we parametrize the process (aims, constraints)?
- diversity
  generated content looks too similar
- creativity
  content looks “computer-generated”

picture from Gerd Altmann on Pixabay

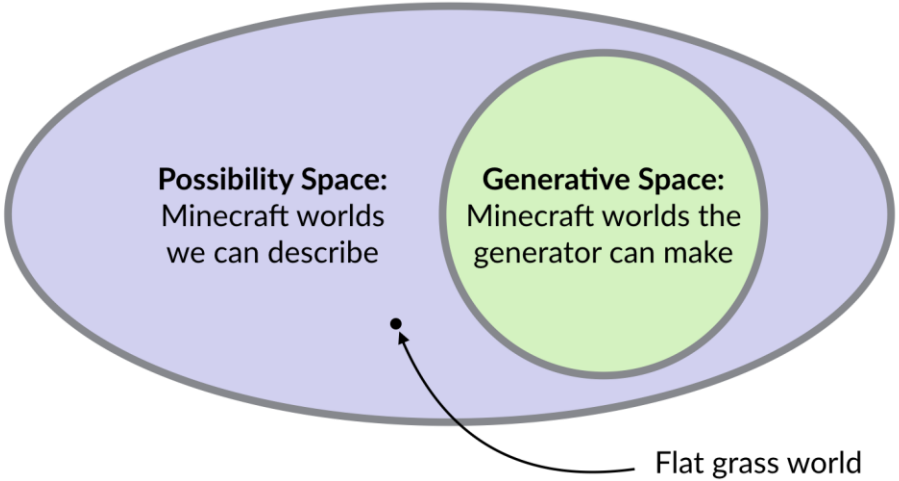
a PCG taxonomy

- online/offline:
  generation during the game?
- necessary/optional:
  does the content has to be “run through” to complete the game?
- random seed/parameters:
  what level of control do we have for producing content?
- generic/adaptive:
  does the content fit for many games or is it adapted to a game/player?
- stochastic/deterministic
  content always looks the same or variates even with the same control parameters
- constructive/generate-and-test
  generate content in one go vs. evolve content according to utility function
- algorithmic/collaborative
  designer adapts parameters of algorithms for obtaining required result vs collaborative (mixed-authorship): designer actively controls process of generation

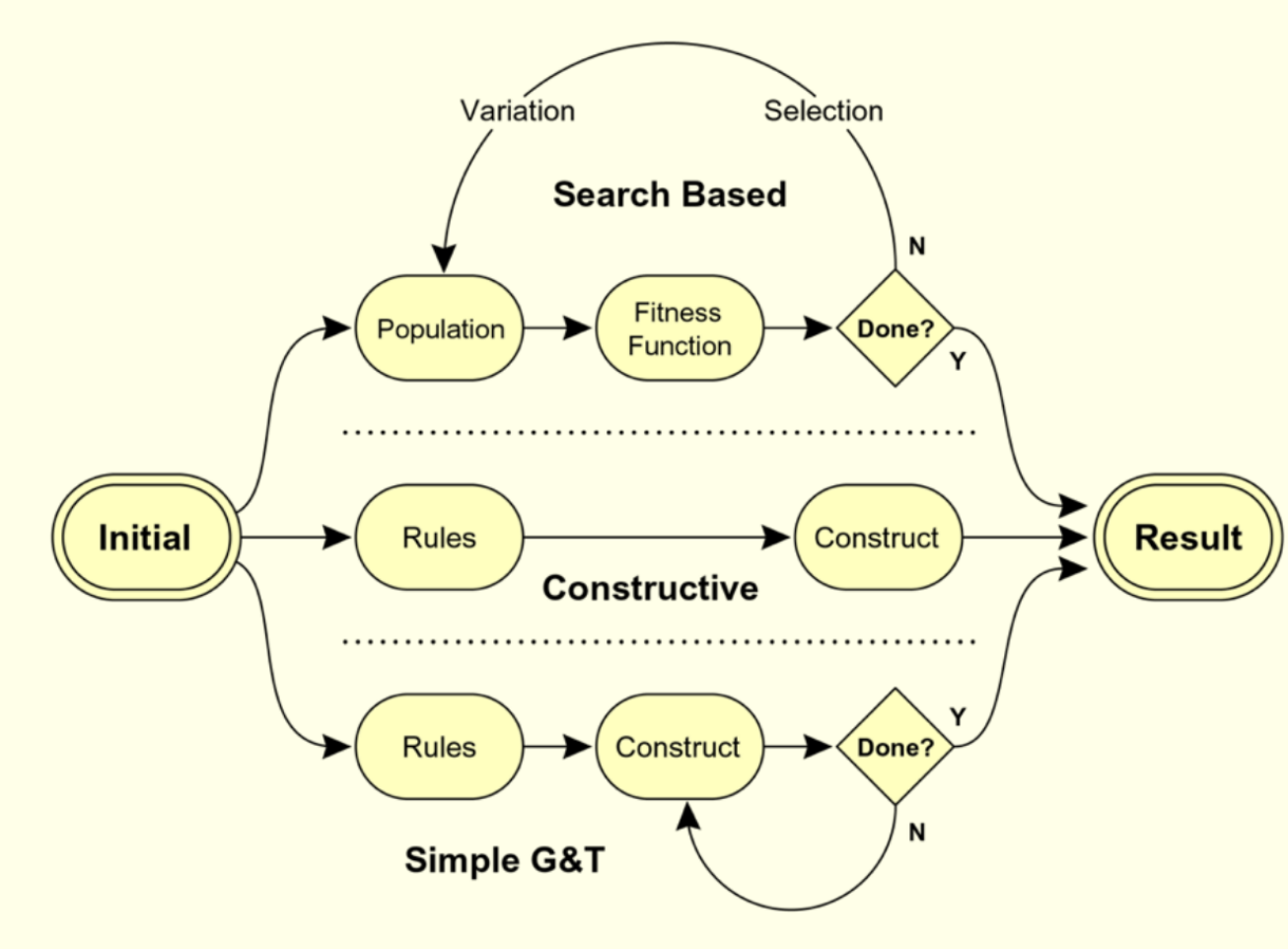
generative space / possibility space

what can we generate?

- there are worlds/levels we can imagine but that cannot be generated with a specific generator
- most often the generative space is much smaller than the possibility space
- how small? that is a tradeoff between controllability and flexibility
- see Michael Cook's tutorial: http://www.possibilityspace.org/tutorial-generative-possibility-space/



search-based vs. generate-and-test



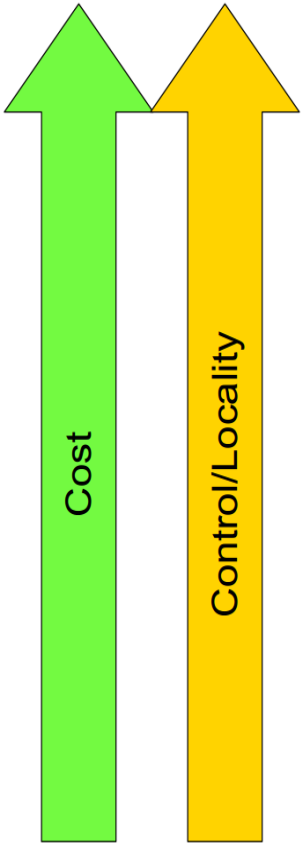
the search based paradigm

a special case of generate-and-test:

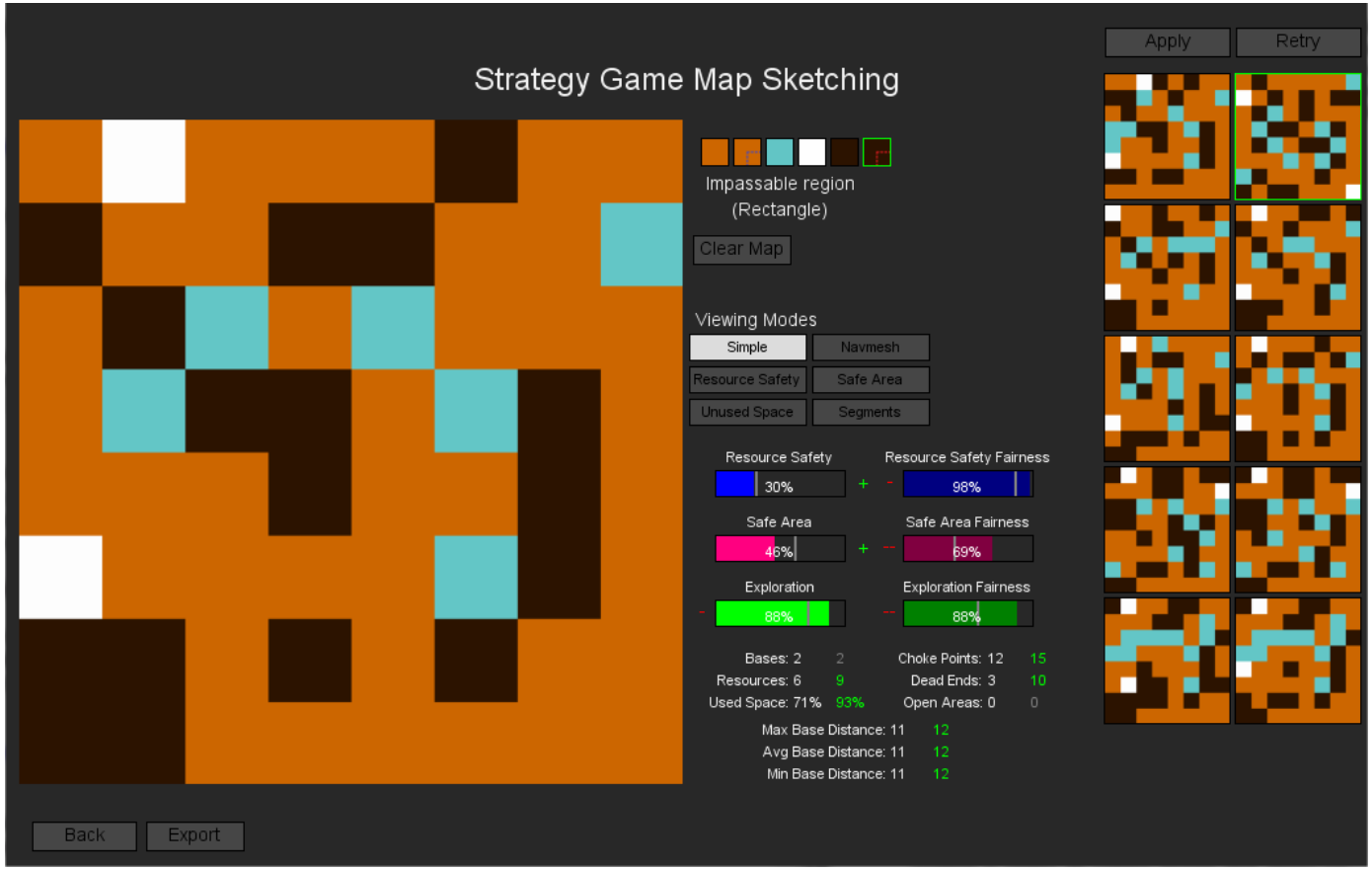
- the utility/target function returns a numerical fitness value instead of accept/reject
- the fitness value is utilized for controlling the generation of new content (-> better)
- realized often via evolutionary algorithms as optimization methods

representation example: dungeon

- direct: grid
- not quite that direct: position and orientation of walls
- more indirectly: partition into repeated patterns of walls and corridors
- very indirectly: number of rooms and doors
- maximally indirect: seed of random number generator



example 1: rts map design with sentient sketchbook

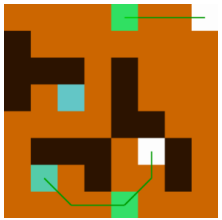


Preuss, Liapis, Togelius. Searching for Good and Diverse Game Levels. IEEE CIG 2014.

utility

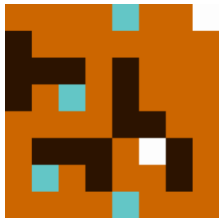
(aggregated from 3*average and 3*variance)

ressource safety:



safe area:

exploration:



diversity

(3 different measures)

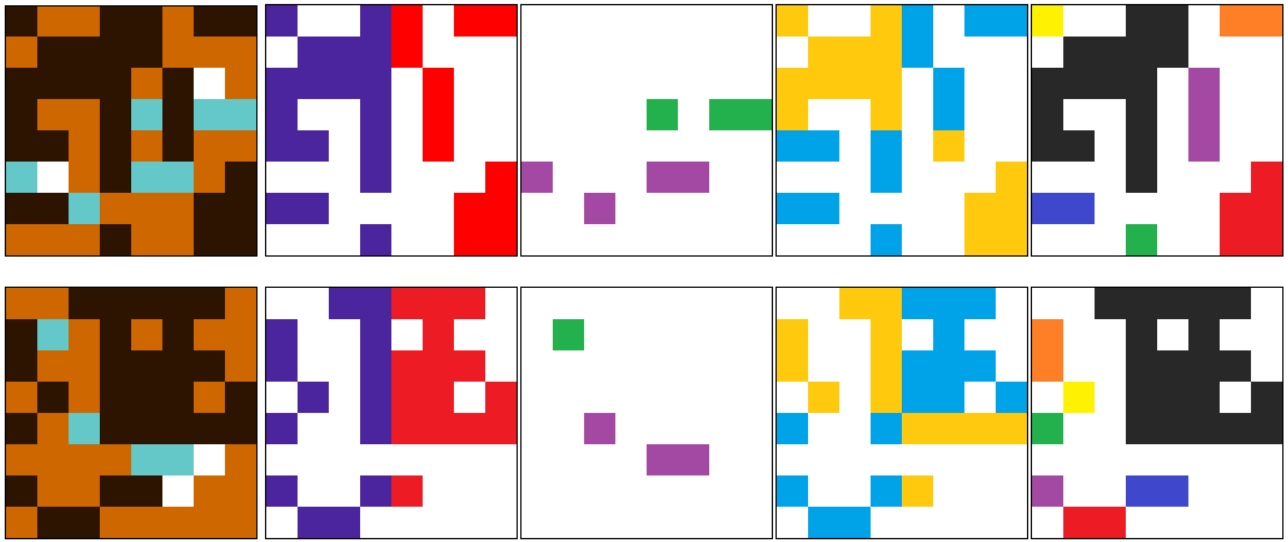
tile-based distance:

utility-based distance:
(original, safe ressources,
safe area)

visual impression distance

formal expression of visual differences

(original, vertical balance of impassable tiles + concentration of impassable tiles in the left half, horizontal balance of resources + concentration of resources in the top half, diagonal concentration of impassable tile, impassable segments + largest segment)



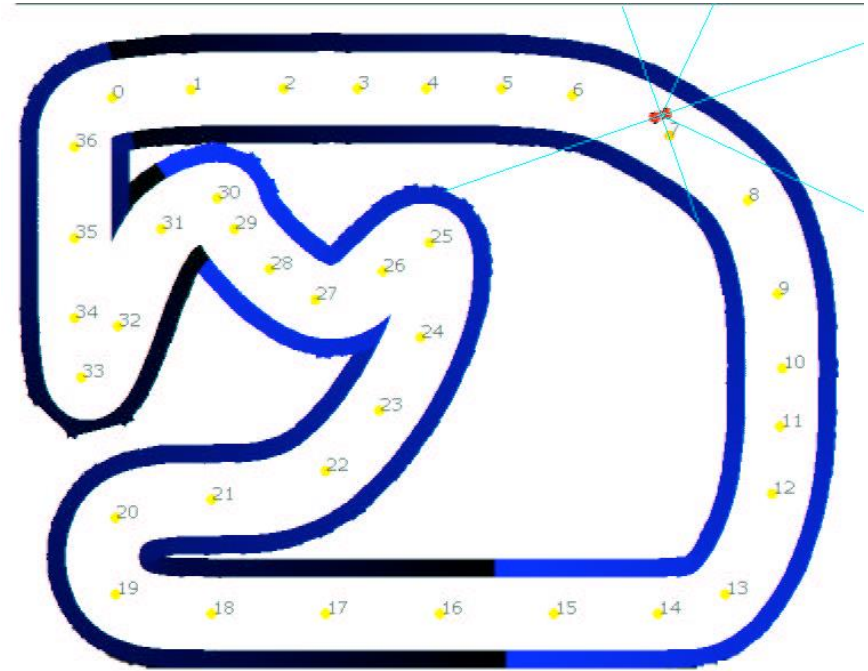
some more details

- map representation: grid, 4 possible values per tile (empty, basis, resource, impassable)
- no recombination (too complex)
- variation (mutation): change a randomly selected tile (may be used several times)
- termination criterion: run length (approx. 1 minute)
- decisions taken via experimental comparison

example 2: personalized car racing tracks

Togelius, de Nardi, Lucas. Towards Automatic Personalized Content Creation for Racing Games. IEEE CIG 2007

- task: create car racing tracks that match individual preferences
- representation: b-Splines
- mutation: random switch of parameters
- utility function based on player model (artificial neural network)

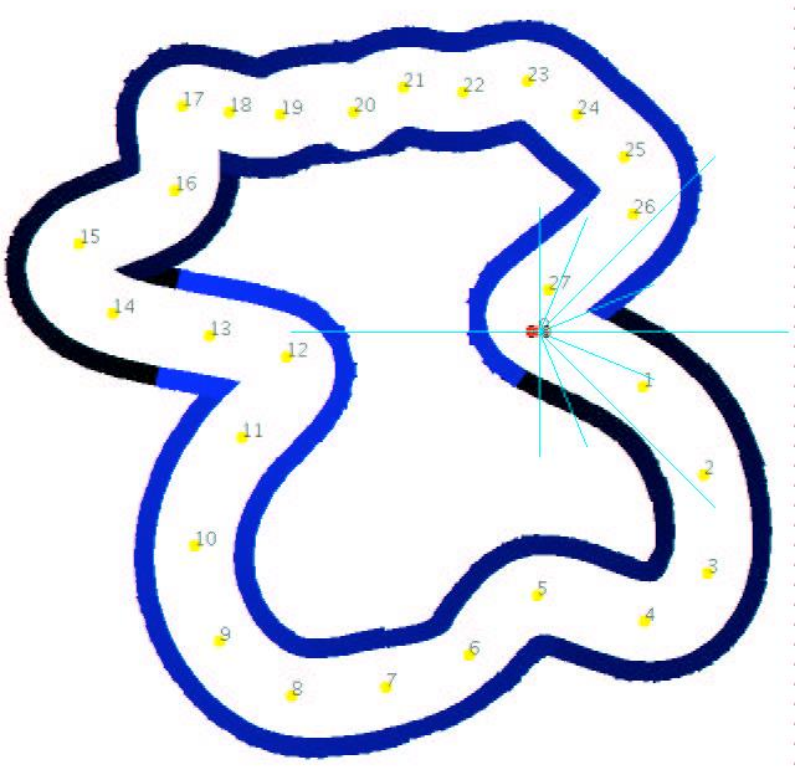


utility function

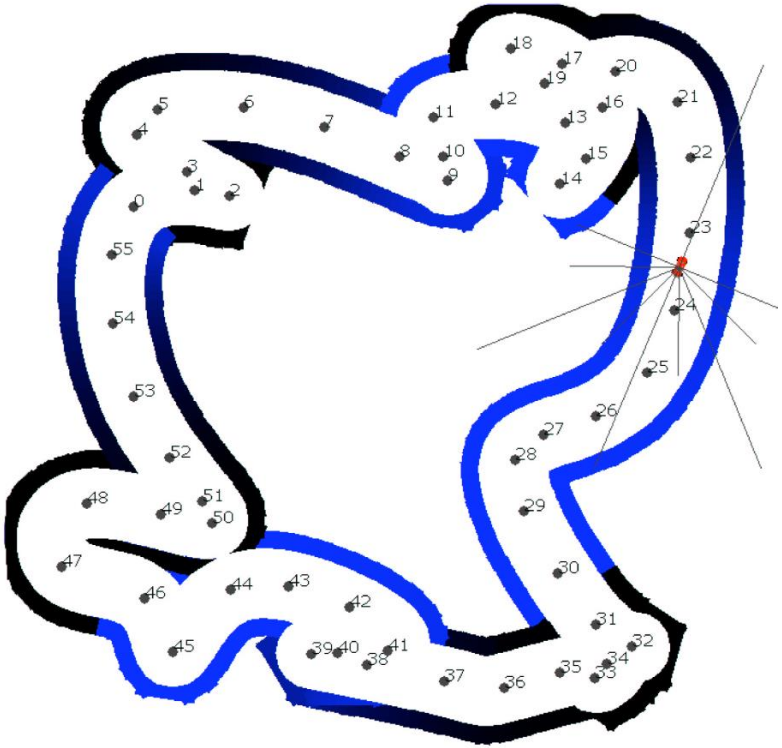
- players drive a test track
- a neural network learns to replicate the driving style
- new tracks are tested by means of the neural network driver

utility function based on:

- difficulty (neither too easy nor to difficult)
- progress (variation between repeats)
- speed (variation within one round)



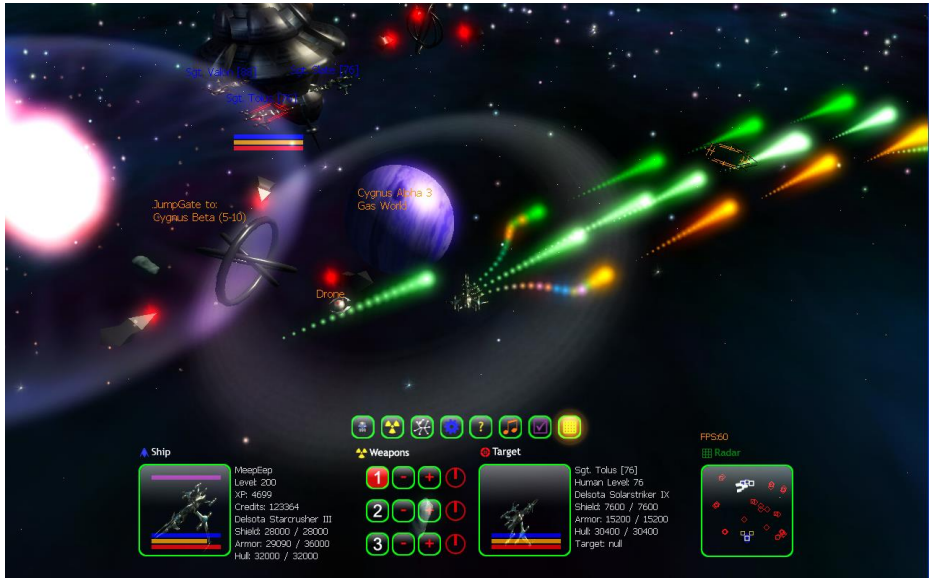
example results



example 3: galactic arms race

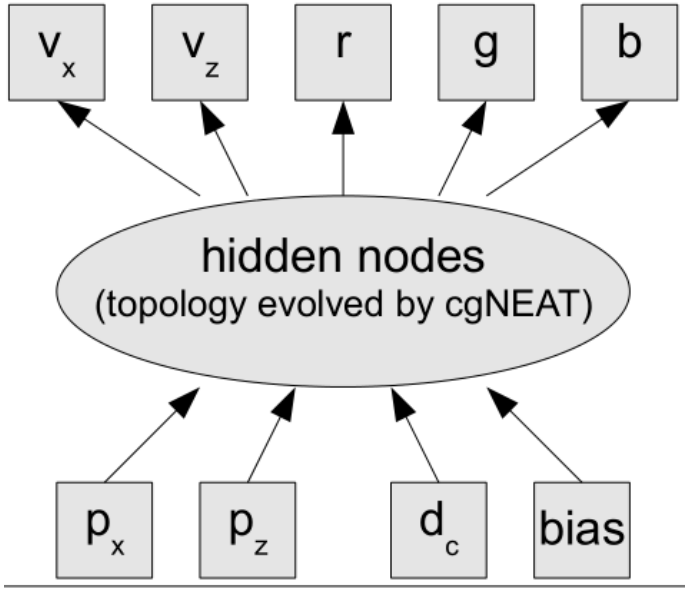
Hastings, Guha, Stanley. Evolving Content in the Galactic Arms Race Video Game. IEEE CIG 2009

- idea: continuously (online) generate new content for a massive multiplayer game (-> replay value)
- generate content matching an individual player
- agency: provide means to the player to enable them to influence content generation

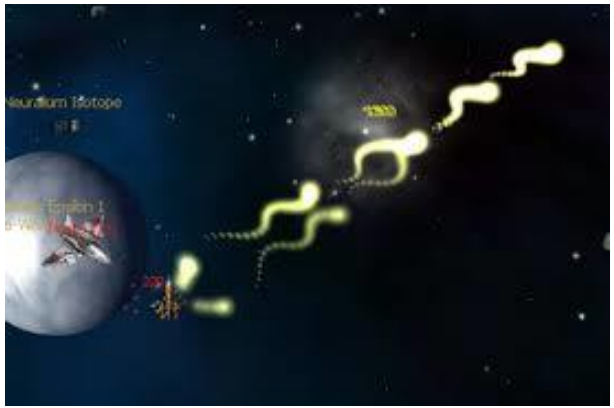


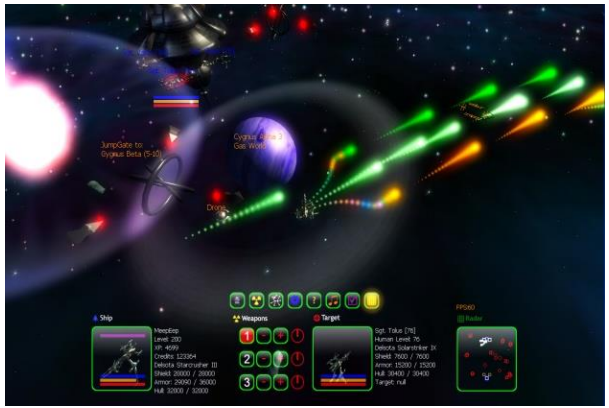
how are the new weapons produced?

- very small population, 3 solutions (candidates, individuals)
- interactive selection: player removes one existing solution when a new one can be generated
- weapons are created via a parametrized particle system by complex neural network
- cgNEAT: neural network topology is optized by means of an evolutionary algorithm



a sample gallery





virtual landscapes

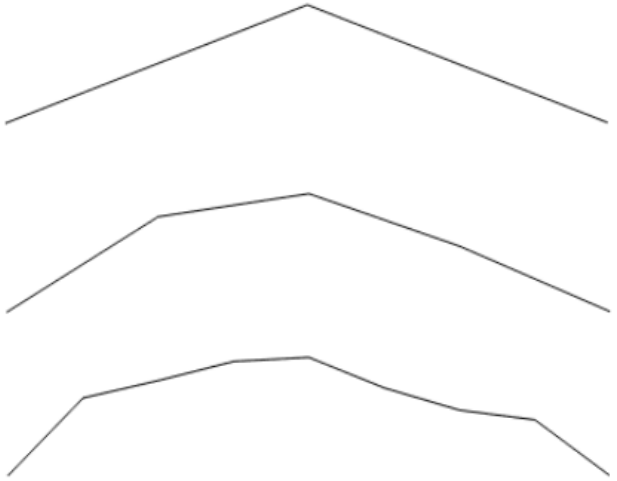
- for strategy games
- for flight simulations
- for all kinds of first person games

properties of landscapes:

- height map (2D grid with height values)
- passable/impassable terrain
- special fields (items, bases, spawn points, choke points)

midpoint displacement

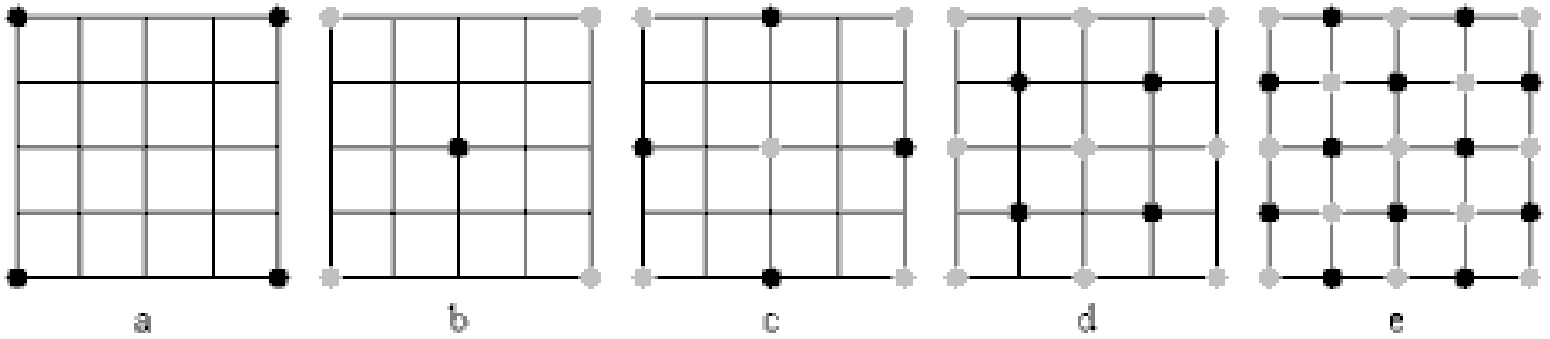
- 2D algorithm generates “silhouettes” by “displacing” (moving) points on a line
- in every iteration, the midpoint of every line is moved up or down randomly
- parameter H: a factor that is employed to reduce the displacement in every iteration
- this results in 2 new lines that can recursively be partitioned further



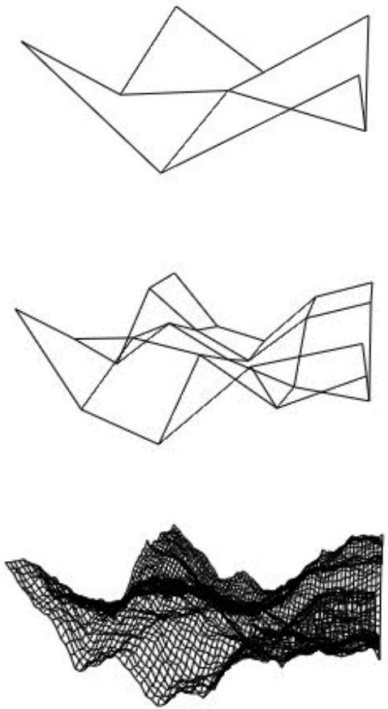
the h-parameter

diamond-square

- extension of the midpoint displacement algorithm to 3 dimension
- instead of lines, we relocate the midpoints of squares
- 2 steps per iteration:
  - squares: horizontal squares
  - diamonds: by 45 degrees rotated squares

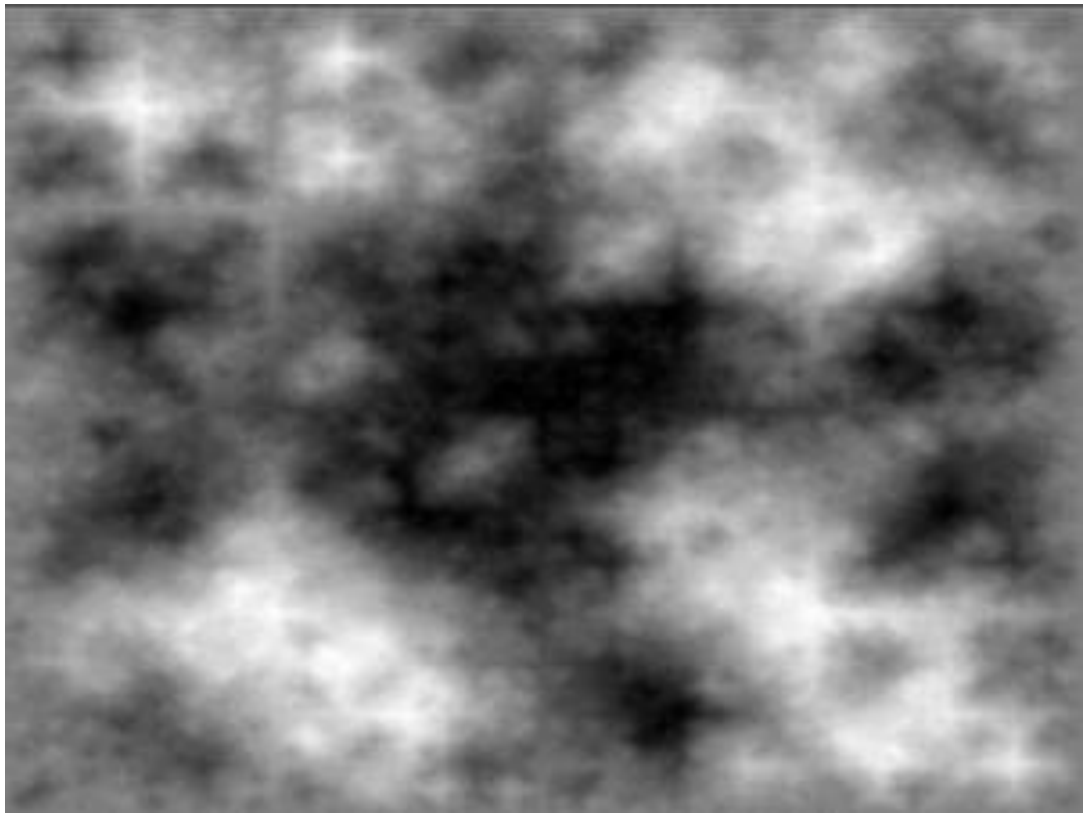


recursive implementation



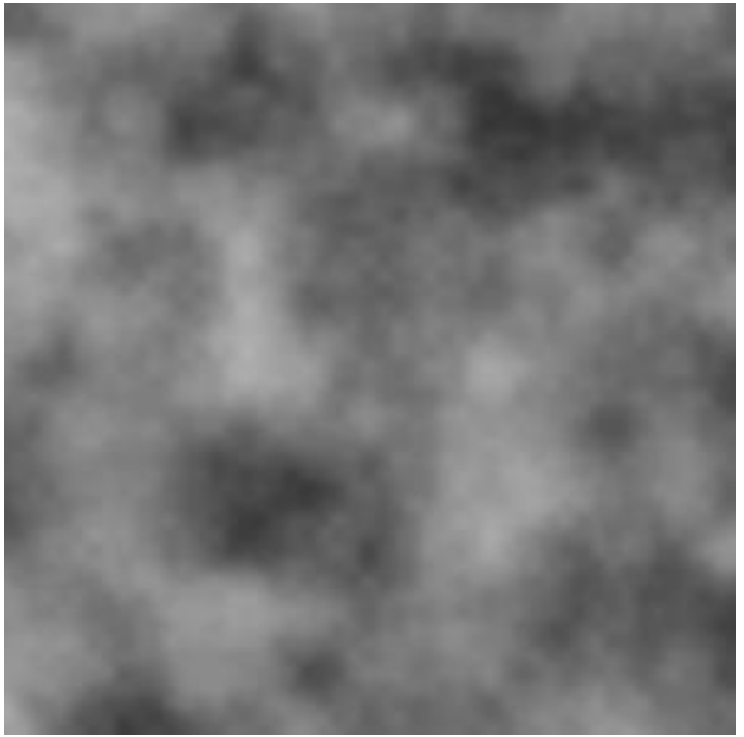
```
diamondsquare (square) {
    midpoint = diamond (square);
    for all 4 edges {
        edge midpoint = square (edge);
    }
    randrange *= H;
    for all 4 resulting squares {
        diamondsquare (new square);
    }
}
```

plasma fractal (diamond-square)



Perlin noise

- method for generating noise in n dimensions
- common in computer graphics, especially movies and games
- suitable for designing clouds, landscapes, water, …
- Ken Perlin got an Oscar for this method in 1997



Perlin noise basic idea

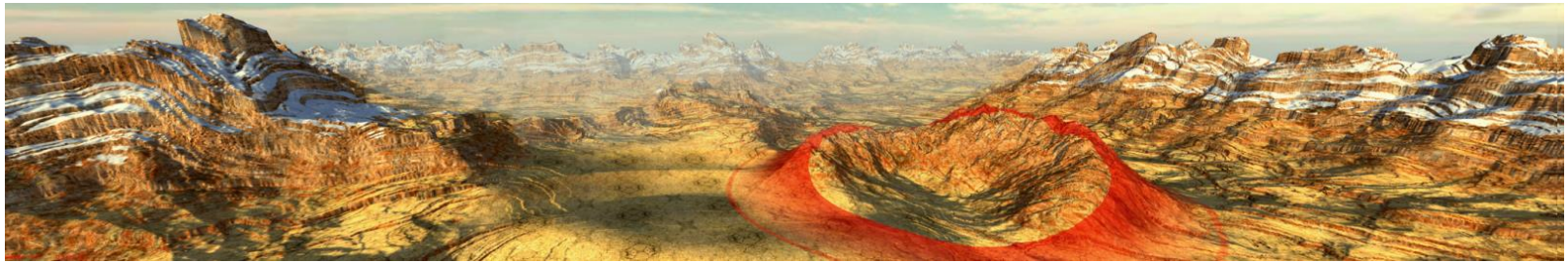
- attach randomized gradient vectors (length: 1 unit) to every grid point
- the gradients express the height
- values between the grid points are interpolated:
  - for every point (4 in 2D), scalar multiply the surrounding grid points with the distance vector to the point
  - the (4 in 2D) values are (maybe weighted) averaged
  - adding several layers with different resolutions leads to more realistic results

try out nice demo at:
https://www.redblobgames.com/maps/terrain-from-noise/
(scroll down to the playground)

procedural modeling

- Procedural Modeling (PM) as the “landscape” side of PCG?
- data amplification/data compression: a small number of parameters represents very complex objects
- “procedural brushes”: improved interactive modelling
- current research: semantic modelling, integration of designer-input

Smelik, Tutenel, Rafael Bidarra, Benes. A Survey on Procedural Modelling for Virtual Worlds. Comput. Graph. Forum 33(6): 31-50 (2014)

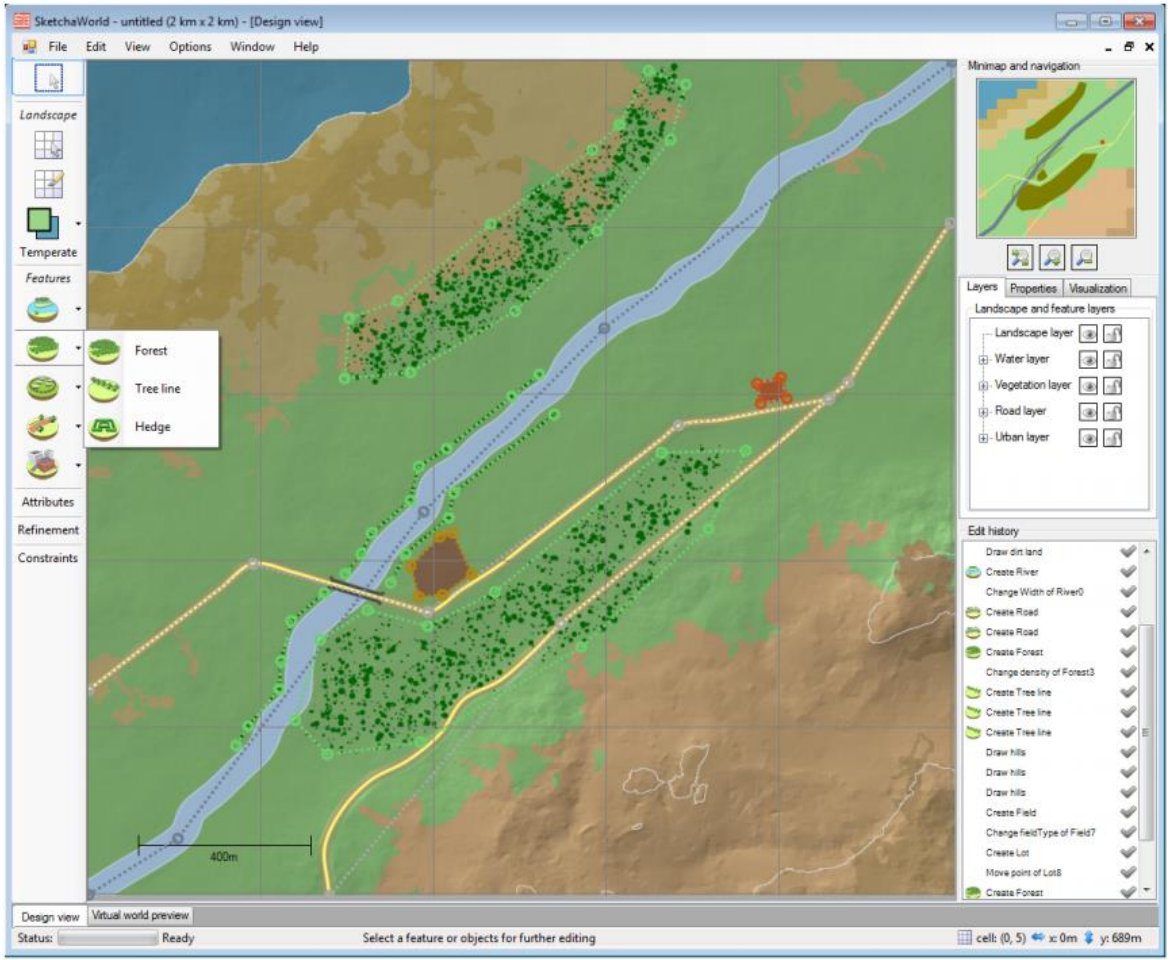


procedural modeling taxonomy

	Stochastic	Artificial Intelligence	Simulations	Grammars	Data-driven	Computational Geometry
Terrain	FFC82, Per85a, Mil86, MKM89, EWM*03, SS05, KU07, dB09, STdB10b	Sau06, DP10	MKM89, Mus93, BTHB06, WCMT07, ASA07, ŠBBK08, KBKv09, PGGM09, VBHv11, GGG*13	PH93	Sau06, ZSTR07	SS05, KU07, Bel07, GMS09, HGA*10, BMV*11
Vegetation		AD05	Hon71, AK88, Gre89, DHL*98, BM02, RFL*05, RLP07, BAv09, PHL*09, PSK*12, BMJ*11	Lin68, PHM93, Pru86, PLH88, MP96, Pru97, LD99, Pru00, IOI06b, IOI06a, LRBP12b	LYO*10	Ham01
Water bodies	PH93, BA05			KMN88		PH93, BA05, HDBB10
Road networks		GPMG10, GPGB11		PM01		AGW86, KM07, MS09
City layout		LWW03, LRW*06	WMWG09, VABW09, EBP*12, VGDA*12	PM01	SYBG02, CEW*08, AVB08	GPSL03, GMB06, dVN06, KM07, GSdB09, LSWW11, VKW*12
Buildings		YK12		KE81, Cag96, Kwo03, WWSR03, YCZY04, CdSF05, LG06, MWH*06, FB08, Fin08, LWW08, Pat12, YK12	MZWG07	
Interiors	MSK10, MSL*11, YYT*11, Cho12			RCMLS96		Cha93, HBW06, Mar06, MM10, TBSd09, TBSd10

Table 1: *Classification of the discussed* PM *methods by virtual world feature and underlying technique family.*

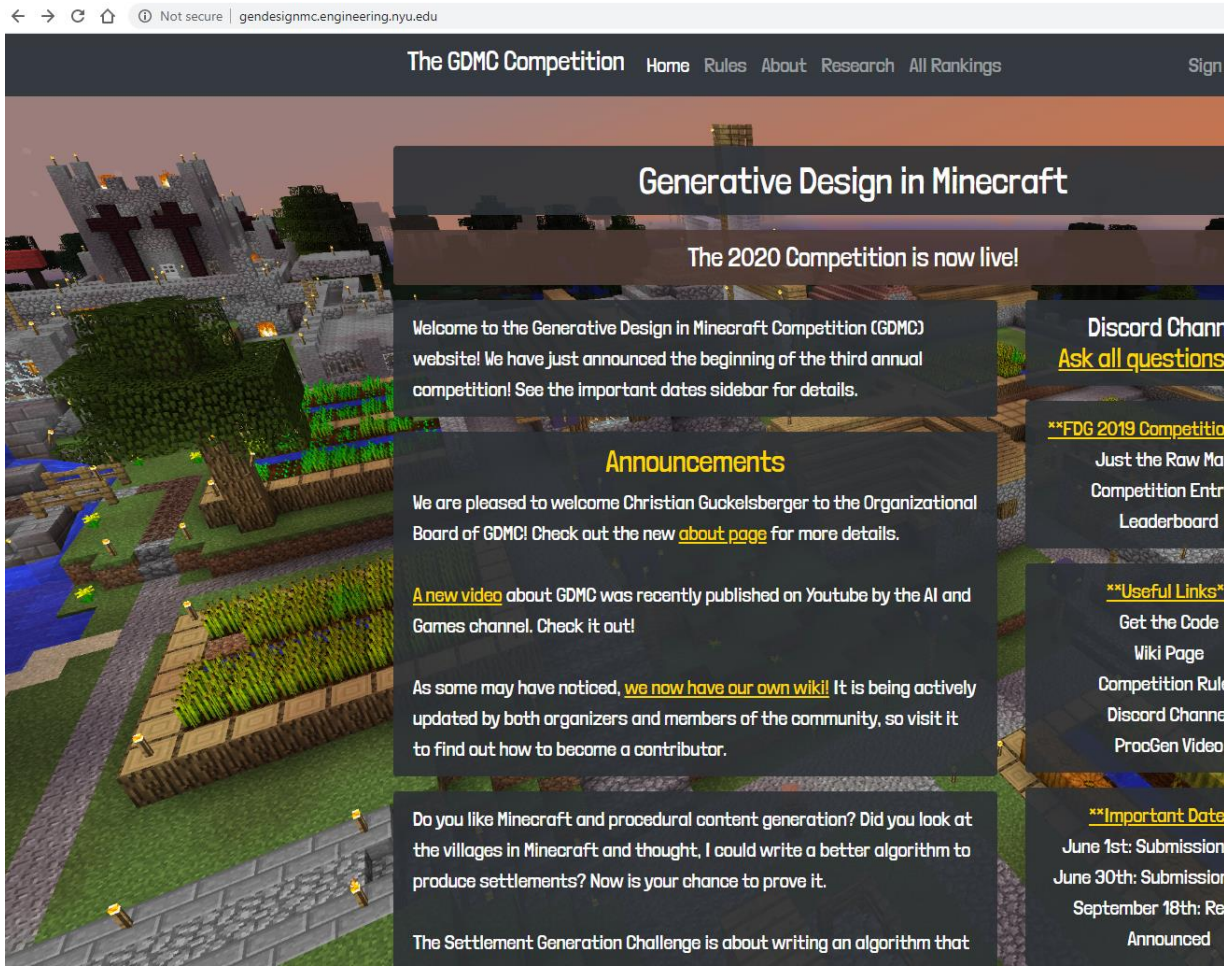
sketchaworld



Minecraft settlement generation competition

- "holistic PCG": make a whole village without human input
- it has to fit the given map
- judging done by humans
- currently open again ☺

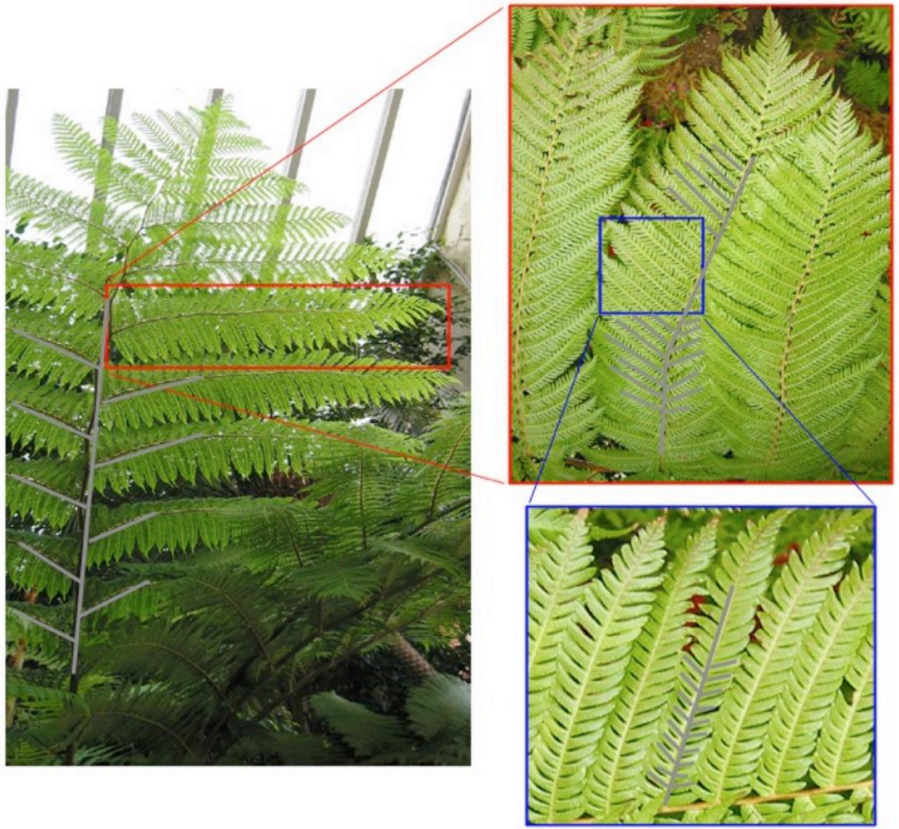
http://gendesignmc.engineering.nyu.edu/



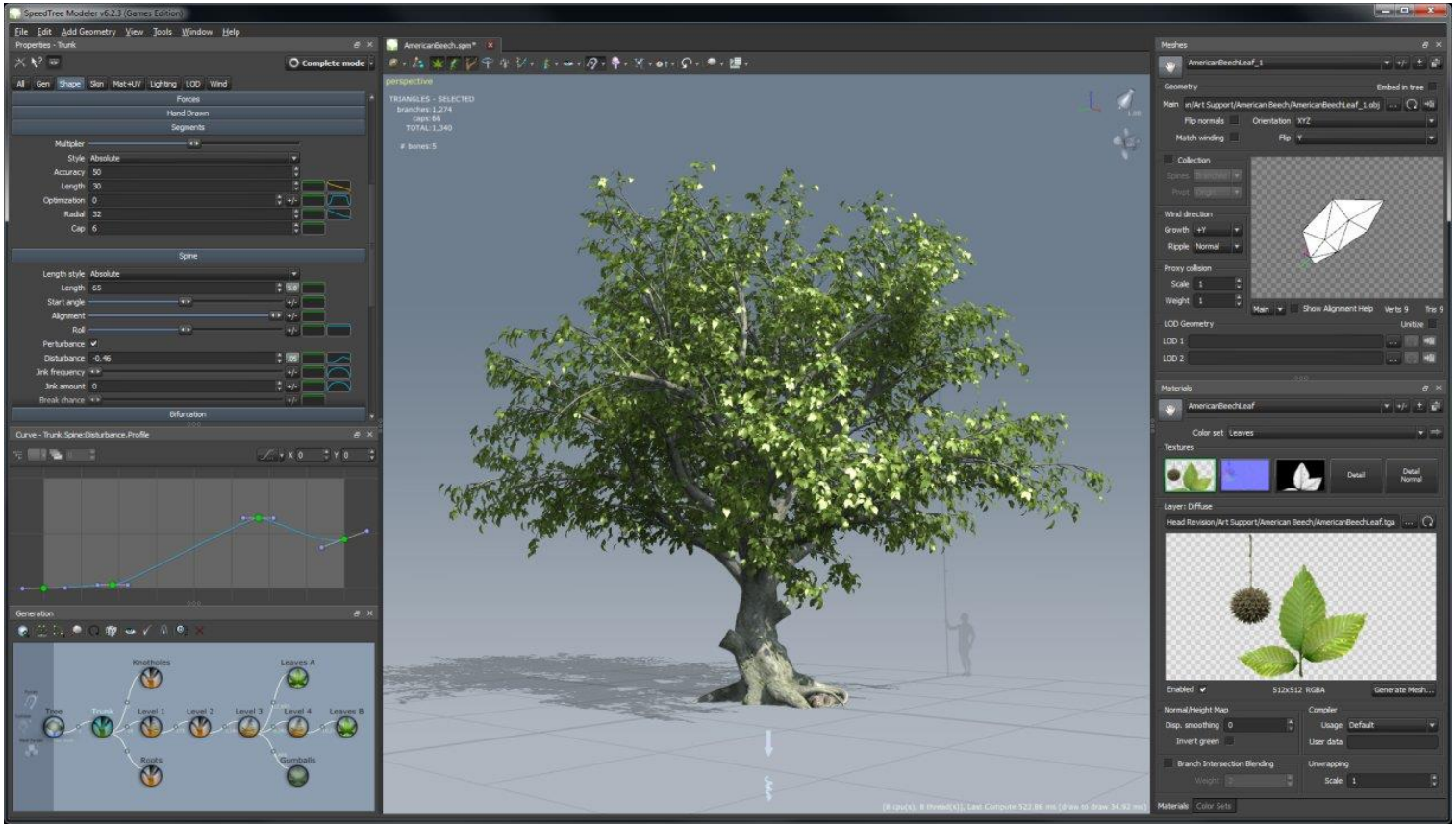
Lindenmayer (L-) systems

- Aristid Lindenmayer, 1968

self similarity



speedtree



demo reels 2020 version

cinema: https://www.youtube.com/watch?v=O3tu926ykvM games: https://www.youtube.com/watch?v=vXFQPixRQOI

L-systems: how it works

- grammar, for which all rules are applied simultaneously
- alphabet (e.g. A B) + axiom (initial term, e.g. A) + production rules (e.g. (A -> AB), (B -> A))
- iterative repetition, until detailed enough
  - $n = 0$: A
  - $n = 1$: AB
  - $n = 2$: ABA
  - $n = 3$: ABAAB
  - $n = 4$: ABAABABA
  - $n = 5$: ABAABABAABAAB
  - $n = 6$: ABAABABAABAABABAABABA
  - $n = 7$: ABAABABAABAABABAABABAABAABABAABAAB

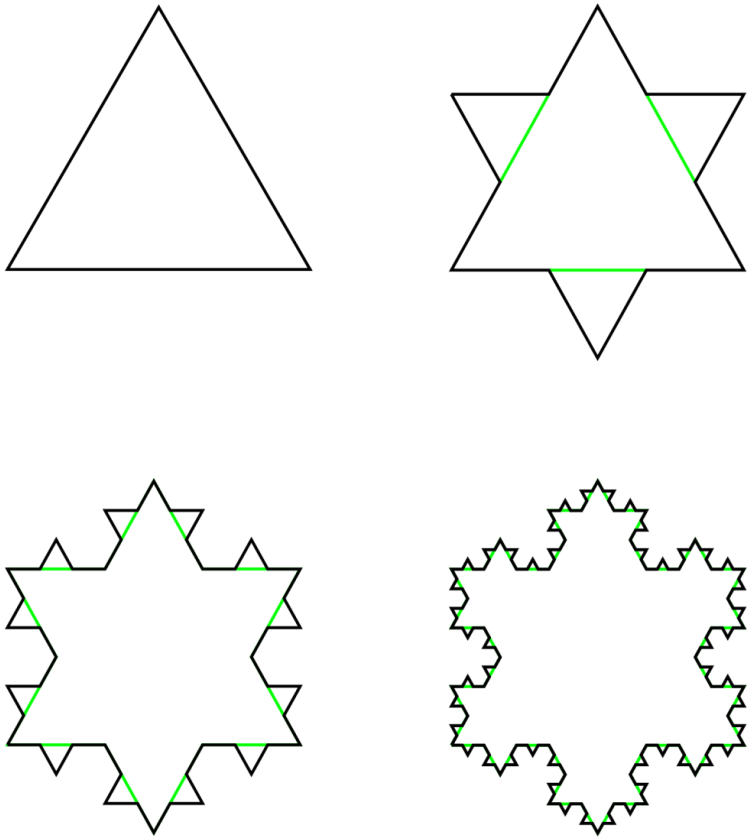
graphical interpretation

Prusinkiewicz 1986:

- idea: interpret string as sequence of instructions in Turtle Graphics
- example: alphabet: {F, f, +, -}
  - F: move forward (and draw line)
  - f: move forward (and do not draw line)
  - +/-: rotate right/left (fixed angle)
  - axiom: F++F++F
  - production rule: (F -> F-F++F-F)
  - rotation angle: 60 degrees

  - what geometrical shape does result?

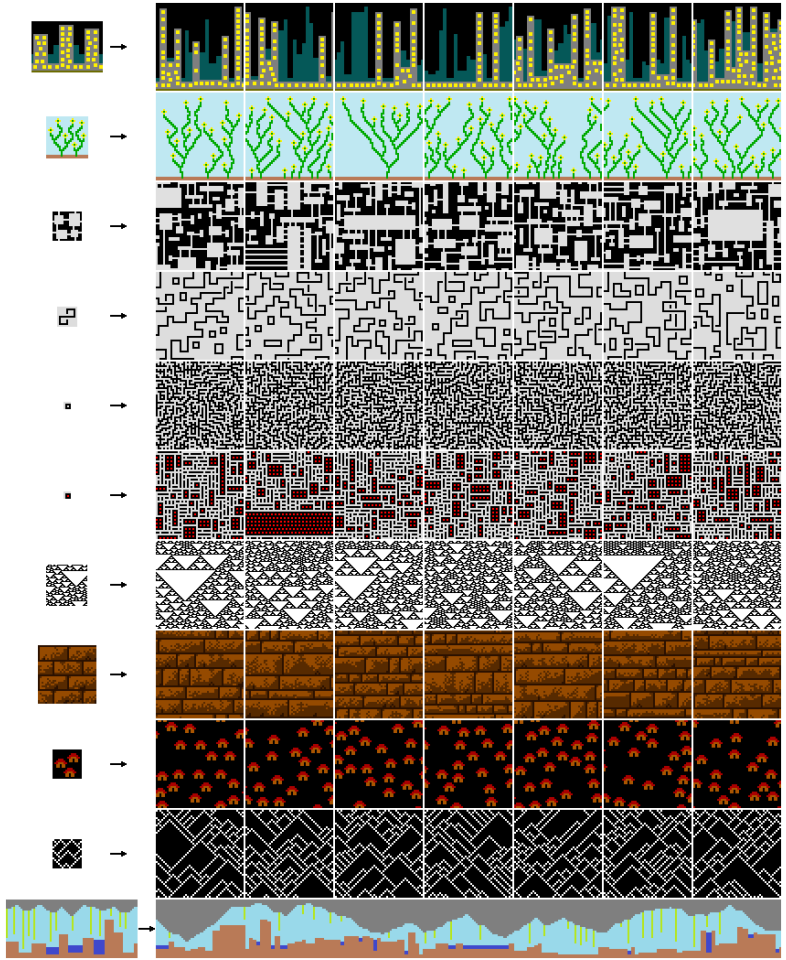
Koch's snowflake



Wave Function Collapse (WFC)

- inspired by Quantum Mechanics
- tiles do have "all states" but then collapse to a definite state (if possible) incrementally
- demo (infinitive city) at https://marian42.itch.io/wfc
- aim is to obtain "local similarity"
- defines N (typically 3) as region size
- chooses region with lowest Shannon entropy and collapses it to definitve state (that is possible due to neighbourhood constraints)
- propagation: new information from collapse is propagated through the output
- can be combined with human presets :-)

- try Unity-based web demo: http://oskarstalberg.com/game/house/index.html



picture from https://github.com/mxgmn/WaveFunctionCollapse

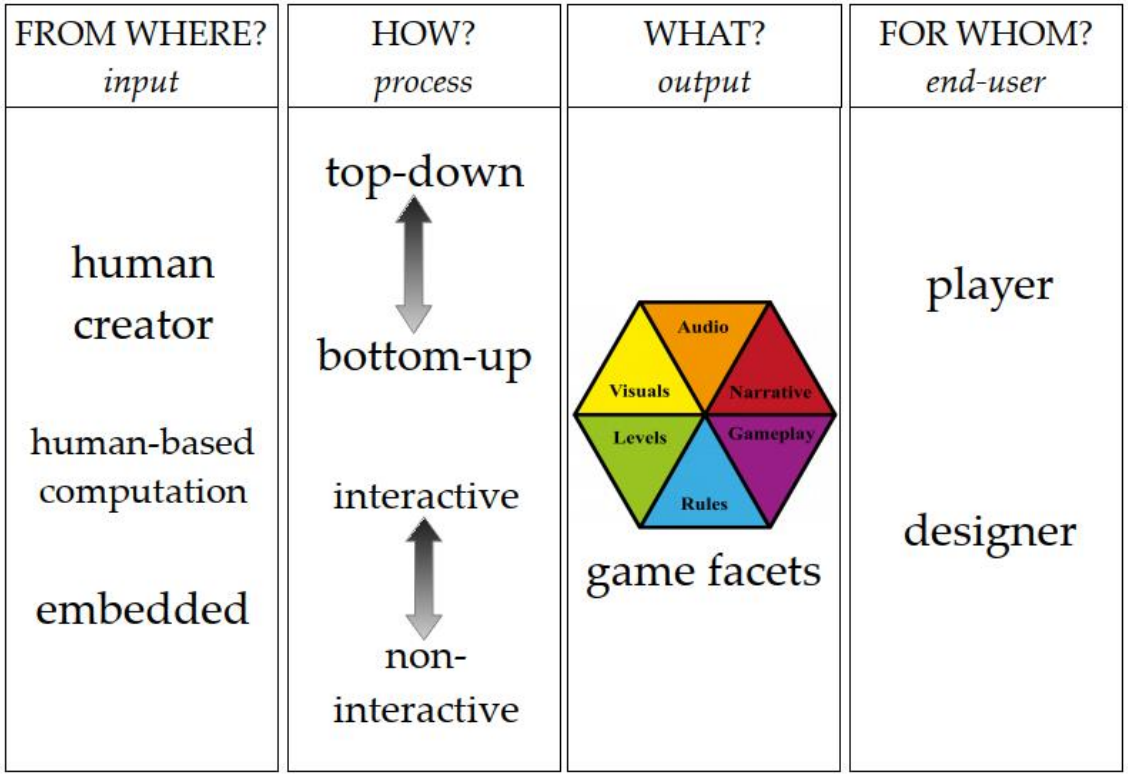
the algorithm

1. read the input bitmap and count NxN patterns
2. create an array with dimensions of the output (called "wave" in the source). each element of this array represents a state of an NxN region in the output. a state of an NxN region is a superposition of NxN patterns of the input with boolean coefficients (possible/forbidden)
3. initialize the wave in the completely unobserved state, i.e. with all the boolean coefficients being true
4. repeat:
   i. observation: find a wave element with the minimal nonzero entropy. If there is no such elements (if all elements have zero or undefined entropy) then go to step (5)
   ii. collapse this element into a definite state according to its coefficients and the distribution of NxN patterns in the input
   iii. propagation: propagate information gained on the previous observation step
5. either all wave elements are in a completely observed state (all the coefficients except one being zero, then return result) or in the contradictory state (all the coefficients being zero, then finish without result)

https://twitter.com/i/status/1215642481162887168

facet orchestration

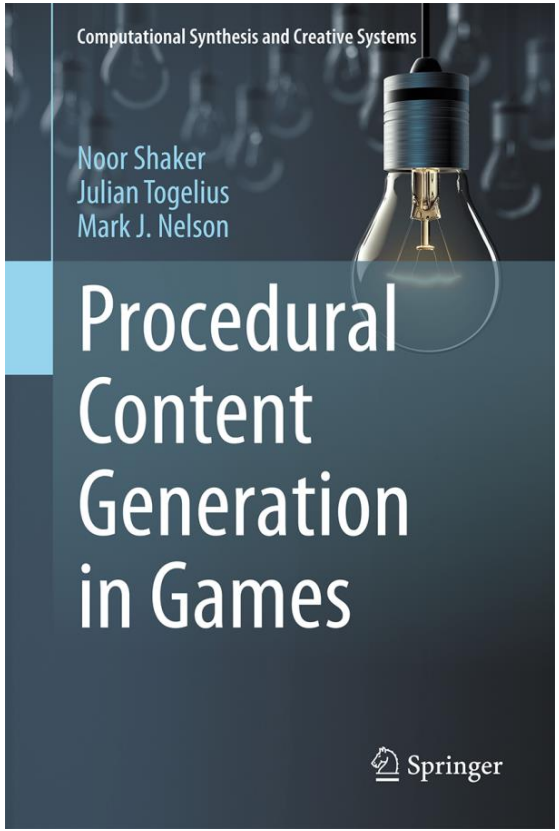
- combining generators for different facets is difficult
- no "overall method" exists, currently manual work
- there are some examples, they focus on a small set of facets



Orchestrating Game Generation: Antonios Liapis, Georgios N. Yannakakis, Mark J. Nelson, Mike Preuss, Rafael Bidarra. IEEE Transactions on Games, 2019

more

- we leave out many interesting developments that come later:
  - GAN for PCG
  - Experience driven PCG
  - difficulty-adaptive PCG
  - PCGRL (PCG by Reinforcement Learning)



- concerning all topics: PCG book http://pcgbook.com
- search-based PCG:
  Togelius, Yannakakis, Stanley, Browne, Search-Based Procedural Content Generation: A Taxonomy and Survey, *Computational Intelligence and AI in Games, IEEE Transactions on* , vol.3, no.3, pp.172,186, Sept. 2011

take home

- procedural content generation is used for more and more game components
- early pcg mostly used for compression purposes, today we want to explore much interesting new stuff
- search-based pcg makes the power of meta-heuristics available for pcg (e.g. evol.algorithms)
- difficulties usually lie in the formulation of “goodness”
- virtual landscapes are produced by many different (old and new) generation algorithms
- wave function collapse (WFC) is a new, interesting algorithm
- facet orchestration is largely unsolved

picture from OpenClipart-Vectors on Pixabay