Procedural Content Generation, continued

2019-11-06

N-3: PCG intro

1. PCG can be used to p____ or a____ game aspects
2. What are some reasons to use PCG?
3. What are some risks / concerns of PCG?
4. Design-time vs run-time PCG?
5. How does the use of a random seed in PCG effect development and gameplay?
6. What is flow theory? How does it relate to dynamic difficulty adjustment & drama management?
7. How do you know you are generating something interesting?

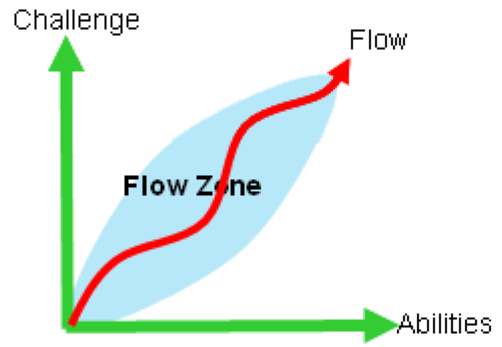


Figure 2 Player in-game Flow experience

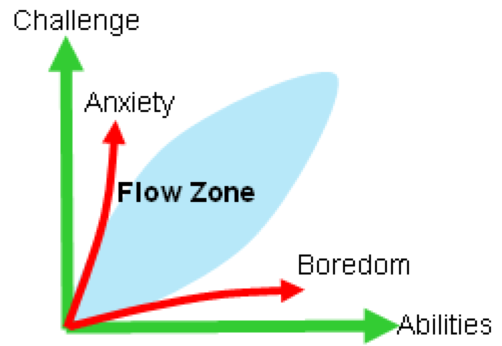


Figure 3 Player encounters psychic entropies

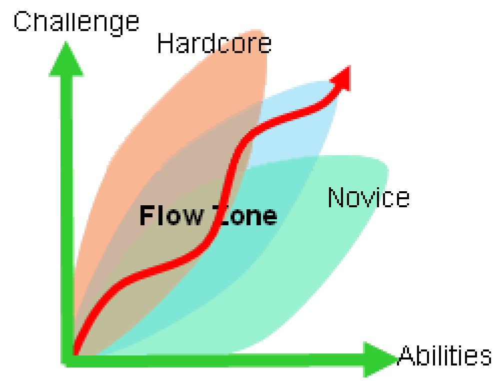


Figure 4 Different players and Flow Zones

http://jenovachen.com/flowingames/designfig.htm

N-2: PCG as Search

1. What “search” is happening? Do we seek a path to goal?
2. What is the state space? How many states do we save?
3. How memory efficient is this search?
4. Hill climbing: (PCG as parameter search part 1)
   1. L____ search
   2. What is the “landscape”?
   3. Need a function that maps p____ to f_____
5. GAs: (PCG as parameter search part 2)
   1. Good in ______ domains, where _D.K.__ is scarce or hard to encode
   2. Can also be used for ____ search
   3. Also needs a f_____ function (maps c____ to f_____)
6. Other local search techniques
   1. Gradient Descent, Simulated annealing, Local beam, Tabu, Ant Colony Optimization, …

N-1: Player modeling

1. What is a player model? What does it allow?
2. What are two high-level categories of modeling?
3. What are a couple major types within the first category?
4. What are some ways to get a player model?
5. What are some differences between Elo and Trueskill?

Model-based versus Model free

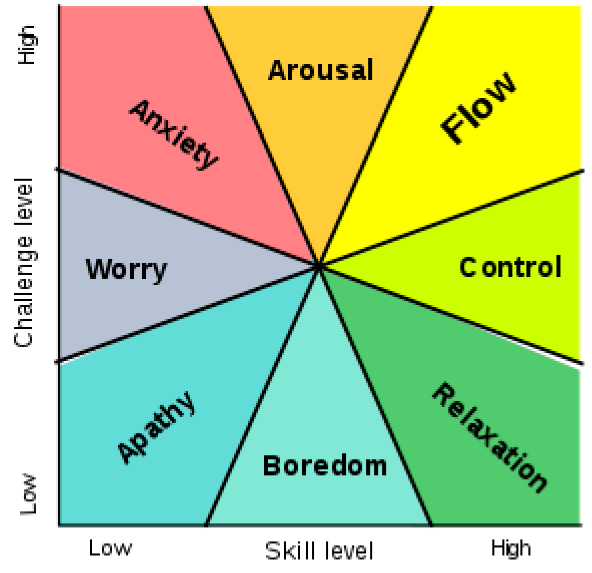
We can separate player modeling approaches into two more general groups:

- **Model-based**: Roughly corresponds to player modeling within games. **It presupposes we have a set of categories or value range to assign to some part of a player's experience**

- **Model free**: Roughly corresponds to player modeling outside of games. **Looks for patterns of behavior without pre-existing hypothesis,** drawing on statistical/machine learning techniques

Model-based: 4 Major Models

- **Note:** This is not to say that there exists only four models, but that vast majority can be understood as derivatives of these four



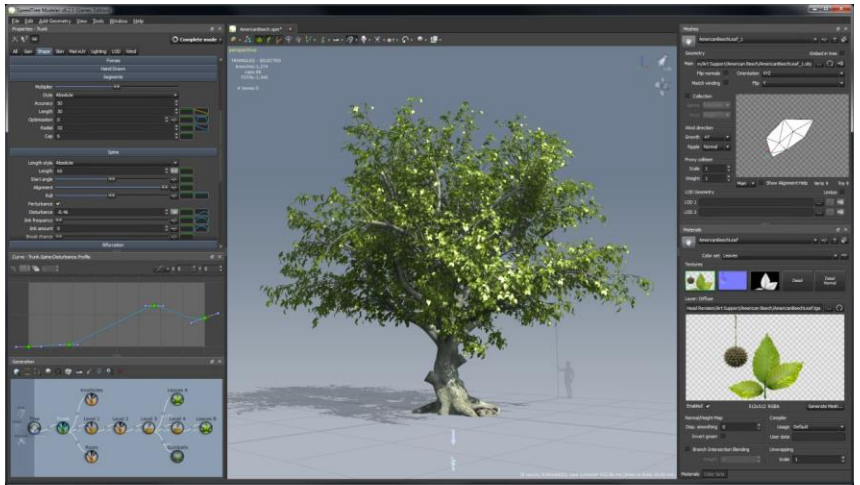
1. Csikszentmihalyi's Flow (Position in space)
2. Bartle’s player types (Categories of play style)
3. Elo’s performance rating (Continuous value)
4. Drama Management (Difference between target value and .)

Game Bits

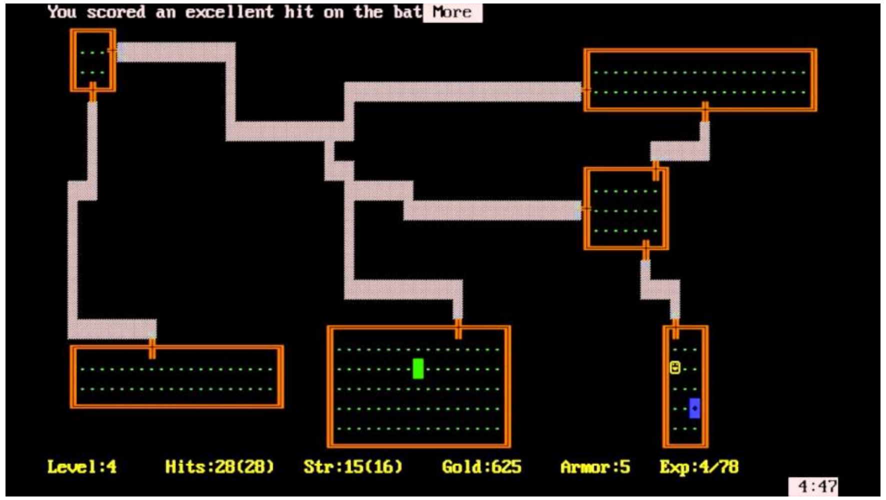
Borderlands series

No Man's Sky

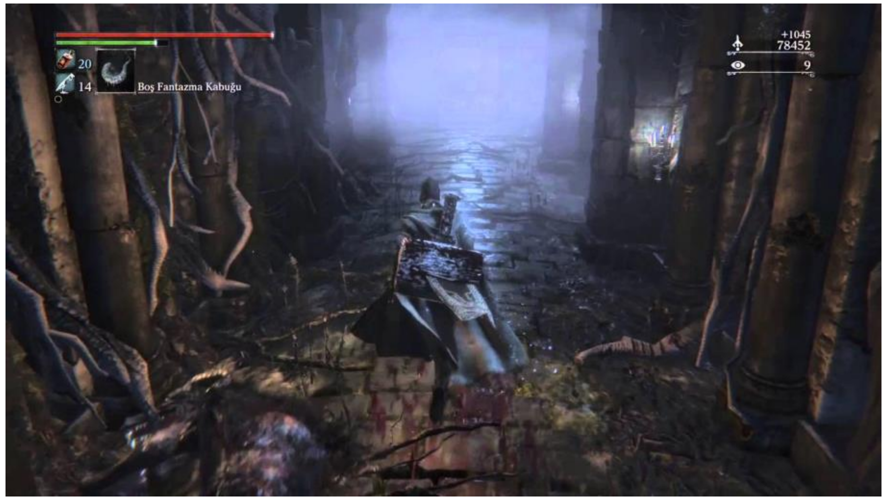
SpeedTree



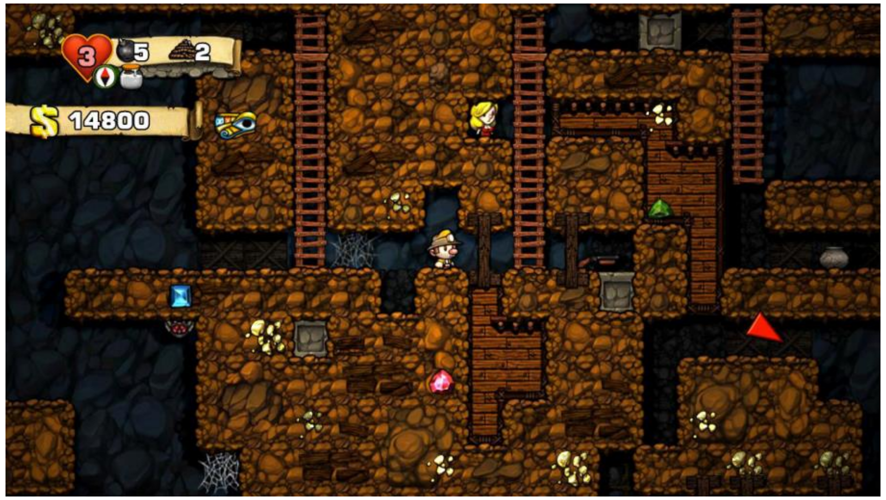
Game Space


You scored an excellent hit on the bat More
Level:4 Hits:28(28) Str:15(16) Gold:625 Armor:5 Exp:4/78
4:47

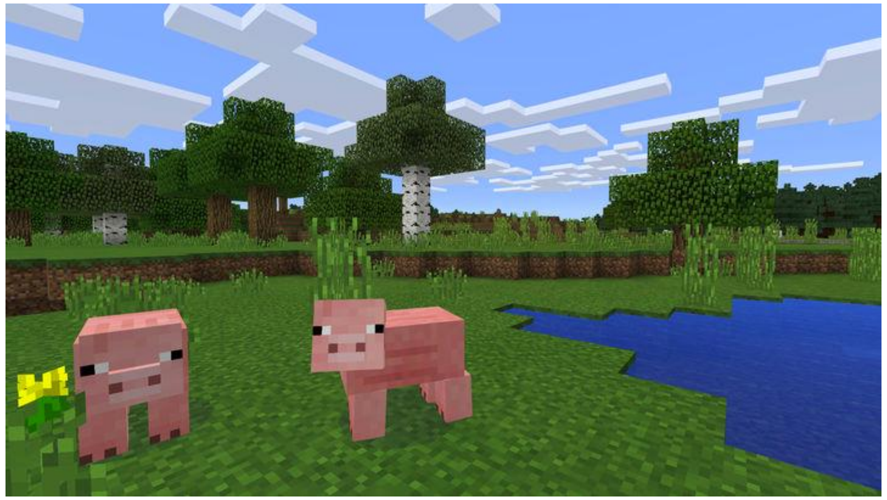
Rogue (1980)



Bloodborne, Chalice Dungeons

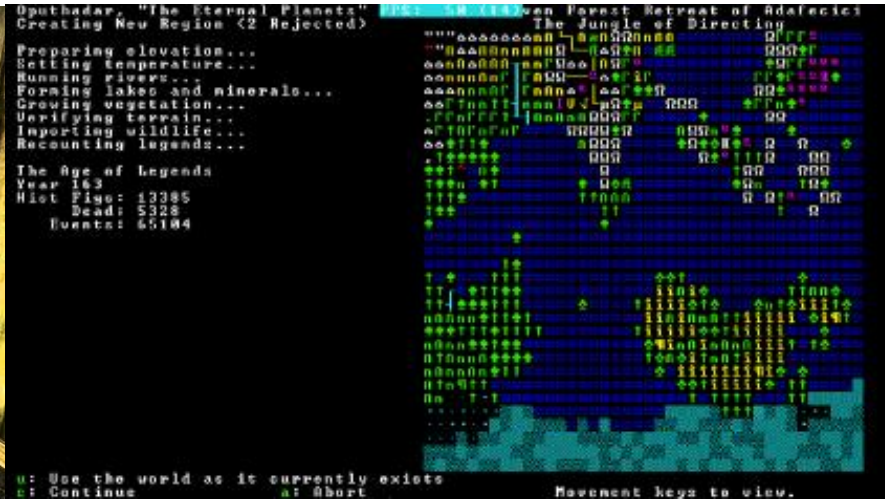


Spelunky



Minecraft

Game Scenarios



```
Oputhadar, "The Eternal Planets"            ven Forest Retreat of Adafecici
Creating New Region (2 Rejected)               The Jungle of Directing

Preparing elevation...
Setting temperature...
Running rivers...
Forming lakes and minerals...
Growing vegetation...
Verifying terrain...
Importing wildlife...
Recounting legends...

The Age of Legends
Year 163
Hist Figs: 13385
     Dead: 5328
   Events: 65104

u: Use the world as it currently exists
c: Continue             a: Abort                 Movement keys to view.
```

Skyrim, radiant quests

Dwarf Fortress

Diablo 2, randomizing situations

Praise you! That jade figurine will complete the set I was collecting.

Here! I've had this statuette of a golden bird for years, but I consider it a fair

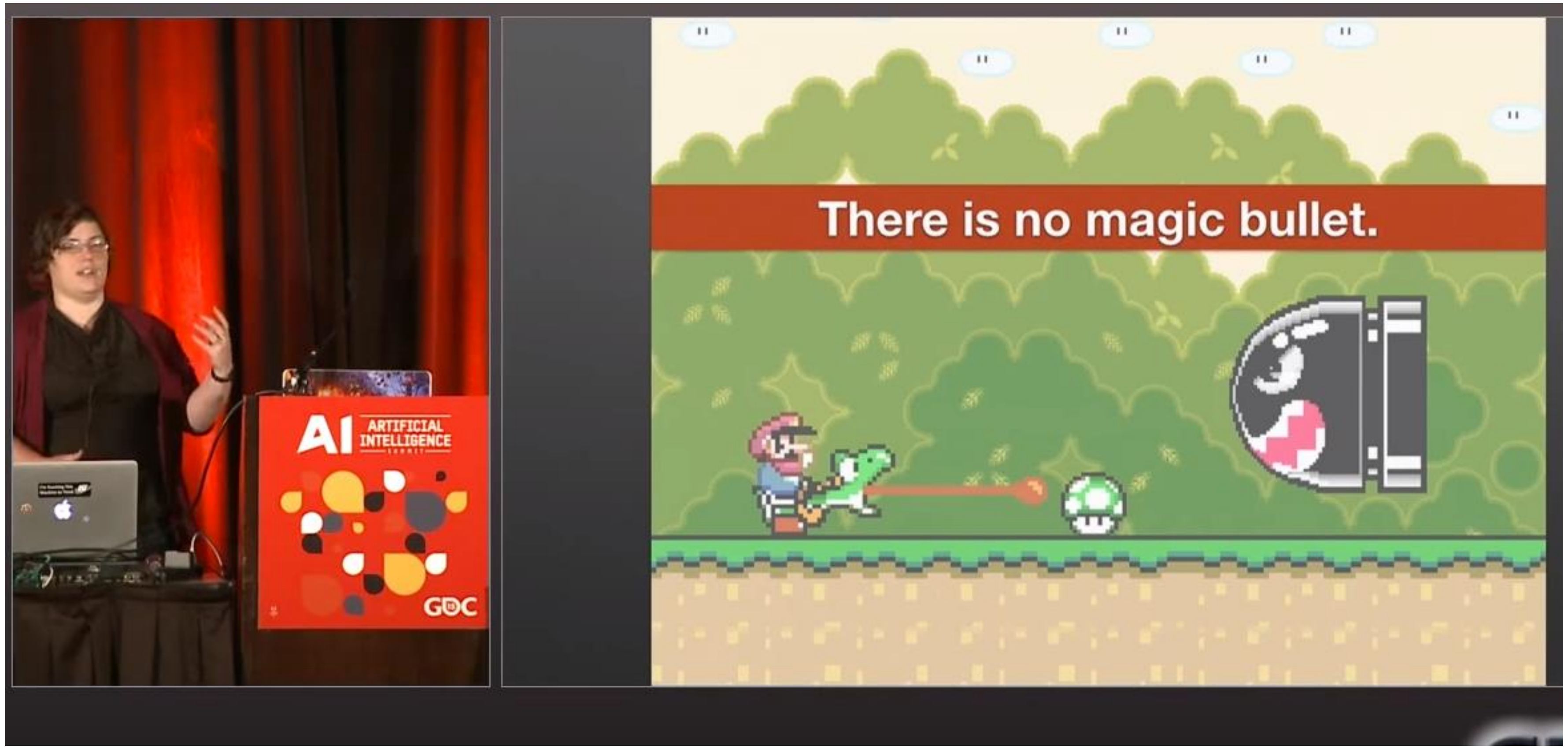
PCG in General

Indie/AAA games folks use as touchstones

- Rogue (1980)
- Dwarf Fortress (2006)
- Minecraft (2009)
- Spelunky (2012)
- No Man’s Sky (2016)
- Horizon Zero Dawn (2017)
  - Japp van Muijden: GPU-based run-time procedural placement
  - https://www.youtube.com/watch?v=_ooDLiU-o6c

Making Things Up: The Power and Peril of PCG

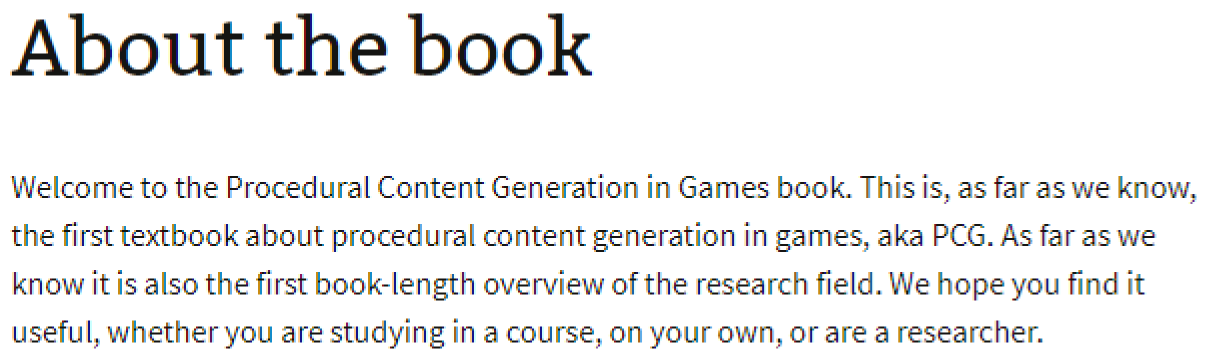
- https://www.youtube.com/watch?v=B11RlHZsmGE



PROCEDURAL CONTENT GENERATION

Content is king!

- Stolen terms (bits, space, scenarios): Procedural Content Generation for Games: A Survey
  - https://course.ccs.neu.edu/cs5150f13/readings/hendrikx_pcgg.pdf
- Search-Based Procedural Content Generation: A Taxonomy and Survey
  - https://course.ccs.neu.edu/cs5150f13/readings/togelius_sbpcg.pdf
- PCG in Games: A textbook and an overview of current research (2016)
  - http://pcgbook.com
- http://pcg.wikidot.com/



About the book

Welcome to the Procedural Content Generation in Games book. This is, as far as we know, the first textbook about procedural content generation in games, aka PCG. As far as we know it is also the first book-length overview of the research field. We hope you find it useful, whether you are studying in a course, on your own, or are a researcher.

Credit: Matthew Guzdial

PLAYER MODELING: ANALYTICS (MODEL FREE APPROACHES)

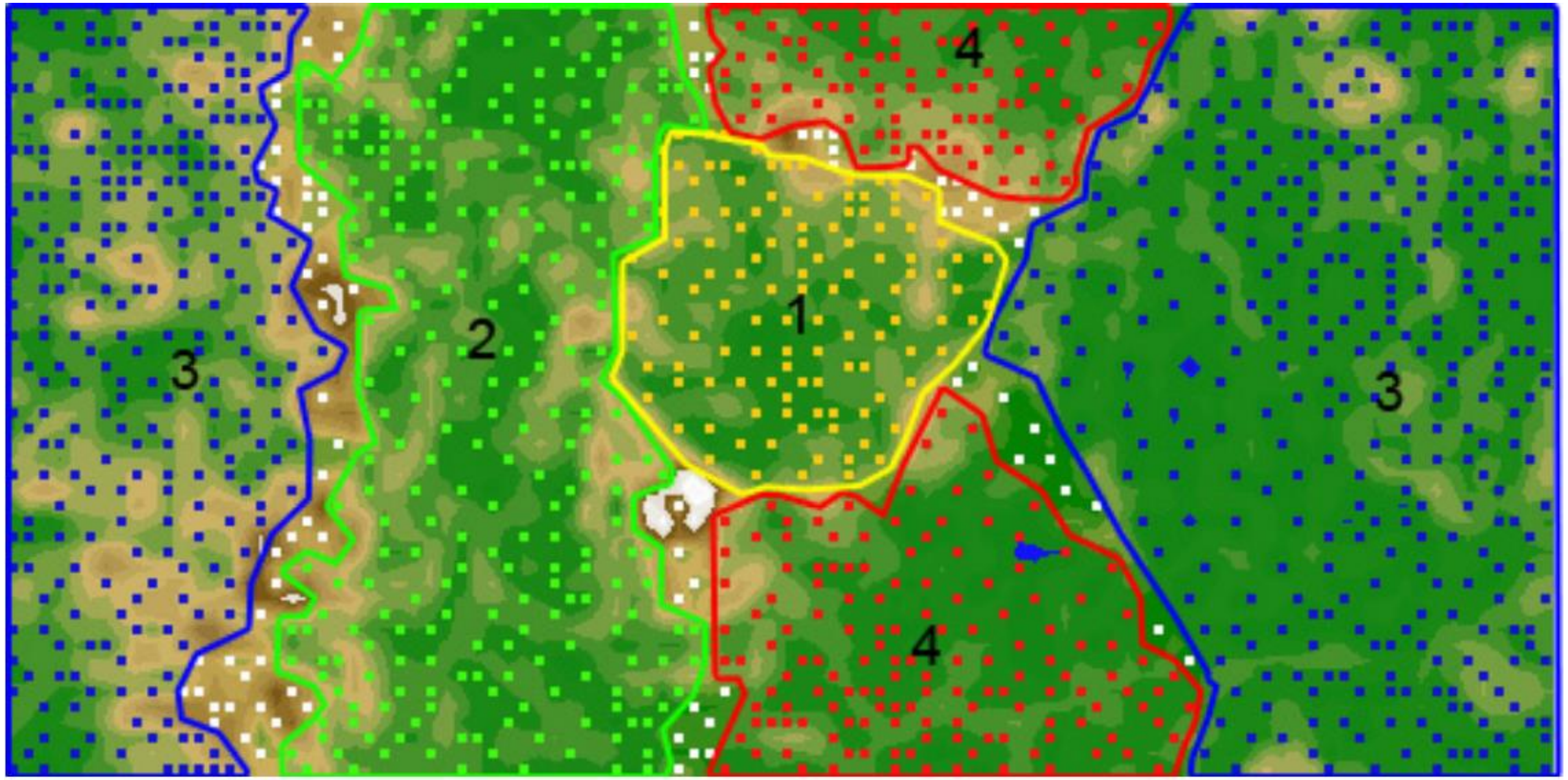
Model free Approaches

- Start with some data about a playerbase and game world

- **Categorize** learn what types exist
- **Regression analysis** how does x relate to y?
  - X: Input about the playerbase/game world
  - Y: Something we want to predict/understand
- **Classify** which y is x a member of?

Quick Example

https://youtu.be/HJS-SxgXAI4?t=6



Clustering

- Uses:
  - Automatically discover categories
  - Group similar entities
  - Reduce complexity/variance
    - E.g. easier to say this player is of 4 types than 1 of 1000
- Common Approaches
  - K means/medoids
  - Hierarchical clustering
- Further info: http://scikit-learn.org/stable/modules/clustering.html

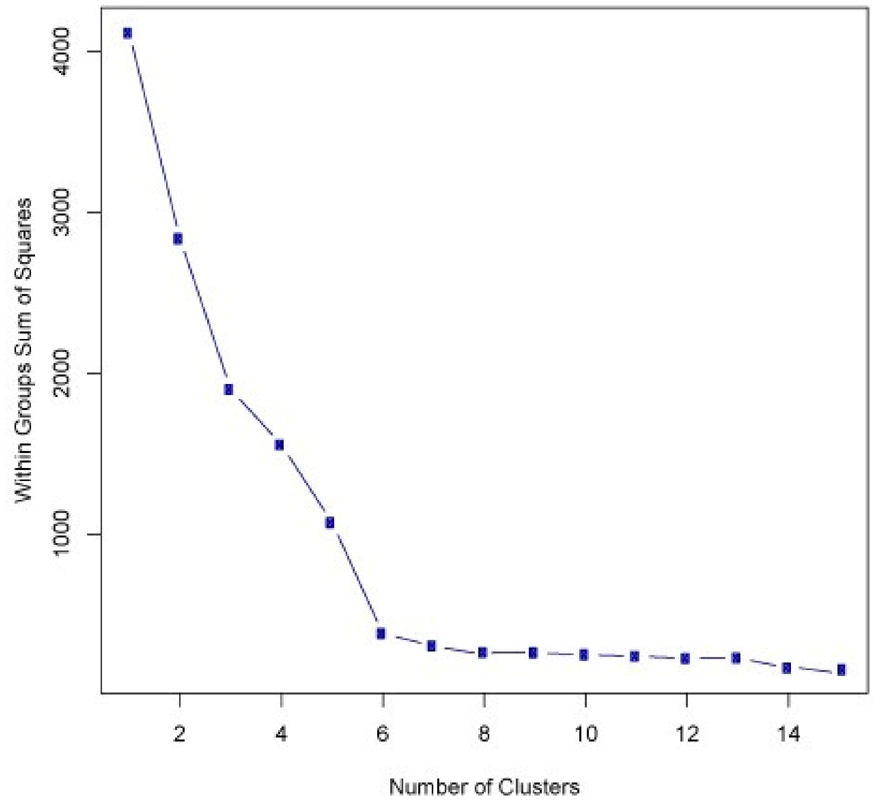
K means

```
initialize centroids randomly
oldcentroids = []
while not centroids==oldcentroids:
    oldcentroids = centroids
    calculateClusters();//cluster each element to closest centroid
    centroids = average of each cluster

return centroids
```

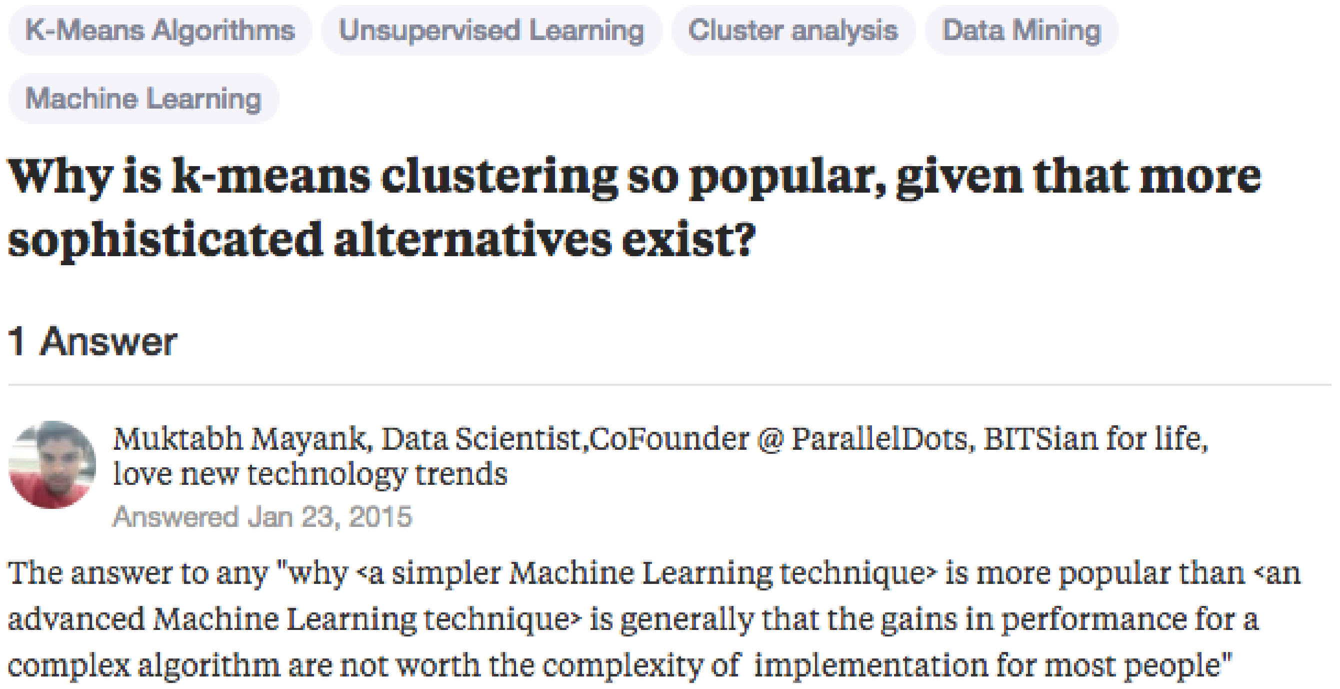
Restrictions and solutions

- Means can give unintuitive clusters
  - K Medoids
- Have to give K
  - Elbow method



Other clustering techniques?

- There are a lot!
- But this isn't a machine learning course
- Plus…


K-Means Algorithms Unsupervised Learning Cluster analysis Data Mining Machine Learning

Why is k-means clustering so popular, given that more sophisticated alternatives exist?

1 Answer

Muktabh Mayank, Data Scientist,CoFounder @ ParallelDots, BITSian for life, love new technology trends
Answered Jan 23, 2015

The answer to any "why <a simpler Machine Learning technique> is more popular than <an advanced Machine Learning technique> is generally that the gains in performance for a complex algorithm are not worth the complexity of implementation for most people"

Open Questions

- How do we pick what variables to cluster on?
  - To normalize, or not to normalize?
- How do we pick a distance function?
- How do we use the clusters once we have them?

Question(s) 1

Think of a game and a particular element of that game (player strategy/player deaths) you'd apply clustering to.

What variables would you cluster on? Why?

What distance function would you use? Why?

How could these clusters benefit the game devs?

Clustering

- Pros:
  - Can learn “big picture” information
  - Can cut down on complexity
  - Can still give good answer with errors in training data
- Cons:
  - Individuals can get lost in the noise of groups
  - Those decisions to the open questions can hugely impact results

Quick Tangent: Churn

- Churn rate: The rate at which customers cut ties with a company
- In games this is how quickly a game loses players. After how long do they stop playing?
- Churn is one of the most common problems player analytics teams are tasked with

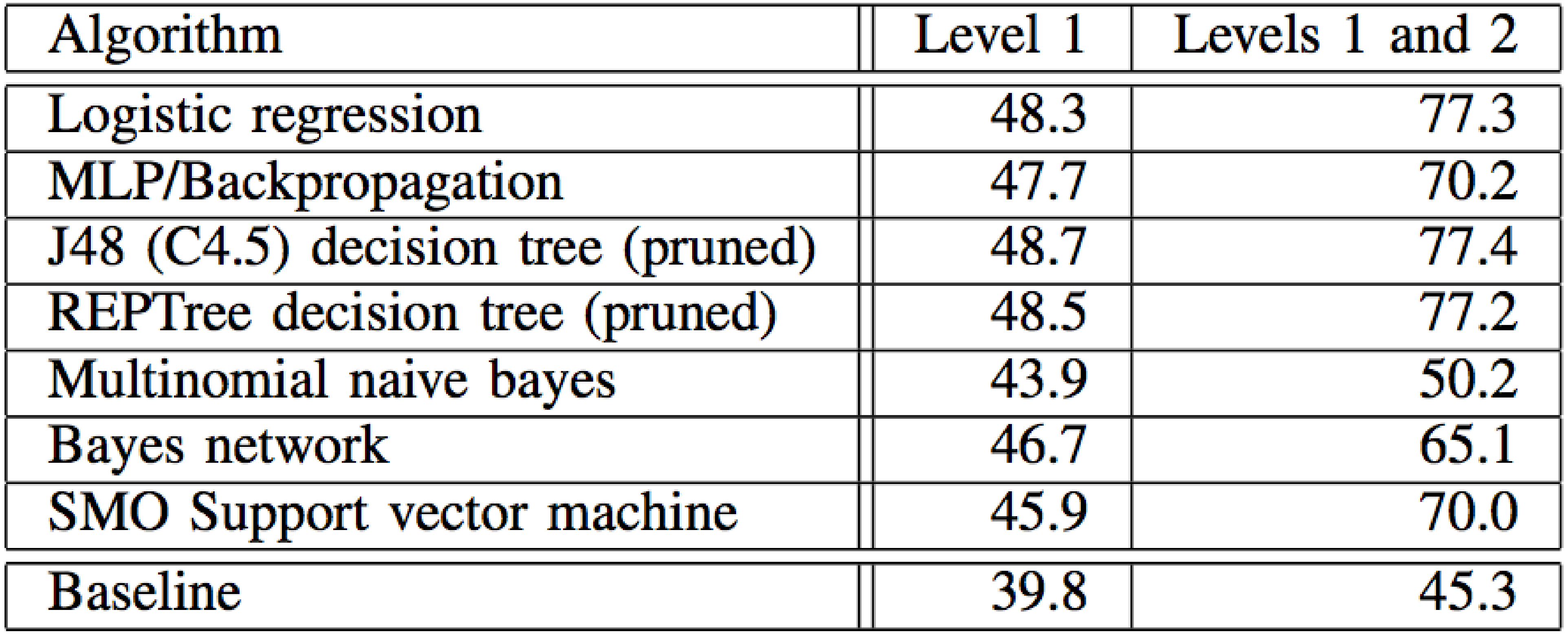
Regression

- If we can map variables onto likelihood of churn (how long the players will play) we can figure out what the designers need to fix!

- Two approaches
  - Linear regression
  - Decision trees

Only two regression techniques?

- No of course not.
- But similar story as with K means…



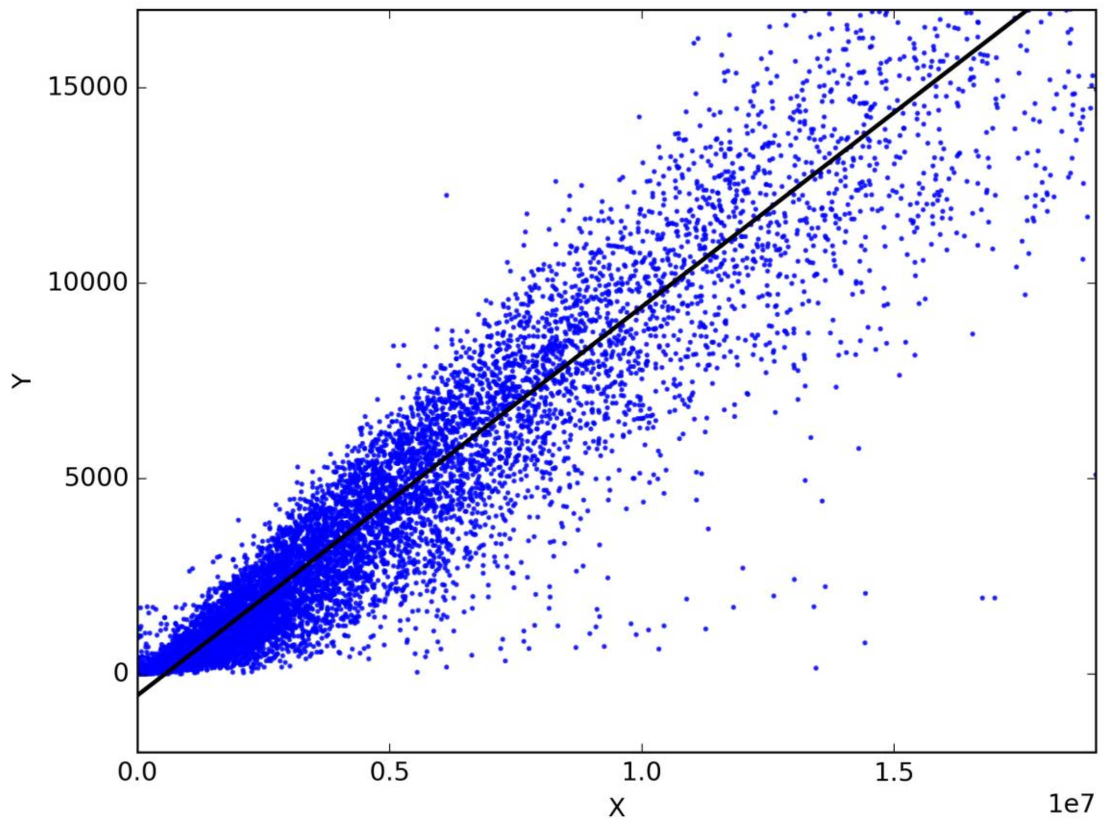
Algorithm	Level 1	Levels 1 and 2
Logistic regression	48.3	77.3
MLP/Backpropagation	47.7	70.2
J48 (C4.5) decision tree (pruned)	48.7	77.4
REPTree decision tree (pruned)	48.5	77.2
Multinomial naive bayes	43.9	50.2
Bayes network	46.7	65.1
SMO Support vector machine	45.9	70.0
Baseline	39.8	45.3

Linear Regression

Given input X and expected output Y, find m and b such that:

$$Y = m*X + b$$

It is, of course, often impossible to find exact values



Linear Regression Algorithm

Find the best fit values for m (slope) and b (y-intercept)

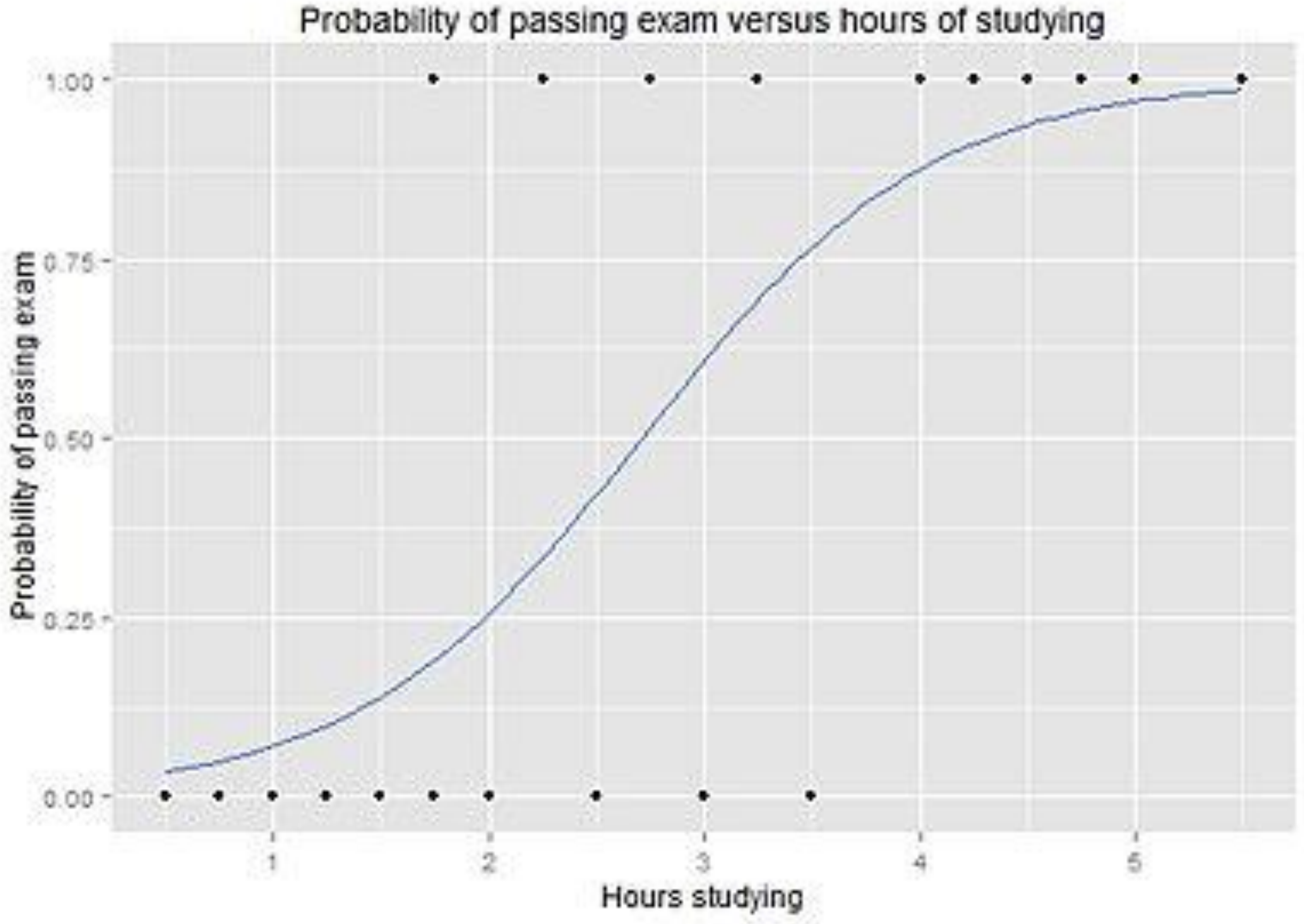
Lots of algorithms to do this

- Minimizing the sum of squared residuals

$$\text{Find } \min_{\alpha,\,\beta} Q(\alpha, \beta), \quad \text{for } Q(\alpha, \beta) = \sum_{i=1}^{n} \varepsilon_i^{\,2} = \sum_{i=1}^{n} (y_i - \alpha - \beta x_i)^2$$

We won't worry about *how* we get the values here

Logistic Regression



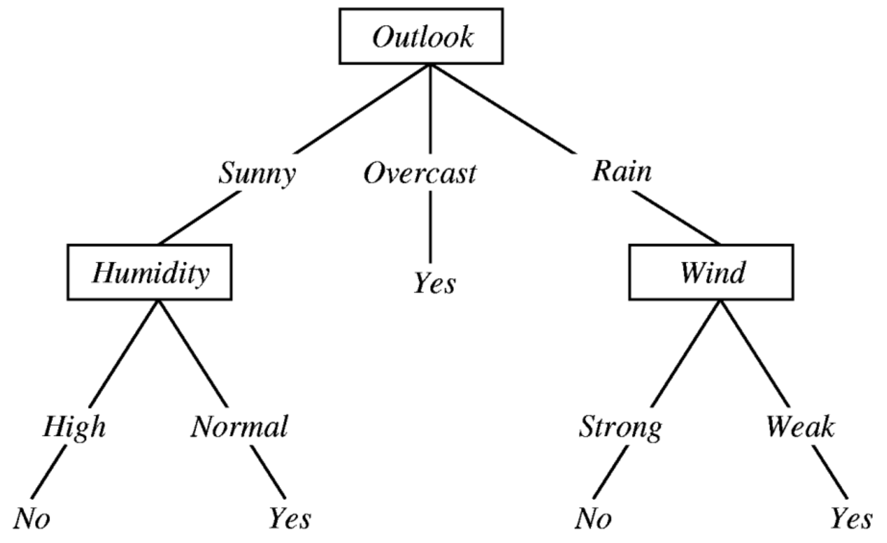
Linear Regression

- Pros:
  - Fast to build
  - Fast to use
  - Simple but powerful
- Cons:
  - Too simple for complex mappings
  - Overly dependent on training data
    - Can’t work with partial information
    - Errors can massively harm its performance

Decision Trees

- Can be used for regression or classification tasks

- Learn a tree to map input to output from training data



Decision Tee Algorithm

```
set currentNode to head
    add all training data to head
openNodes = []
while currentNode.entropy*>threshold:
    currentNode = openNodes.pop()
    children = calculate best split point(s)
    for child in children:
        if child.entropy<=threshold:
            make leaf node
        else:
            openNodes.push(child)
return head
```

*entropy is the measure of how much in agreement the data “in” this node is

Decision Trees are suited to problems where…

- Data is in the form X,Y pairs
- Y has discrete values it can be (yes/no or in a category)
- X->Y is not a simple linear function
- Training data may contain errors/noise
- Training data may contain some X values without all attributes

Decision Trees

- Pros:
  - Fast to use
  - Robust to errors
- Cons:
  - Slow to build
  - Depending on threshold for entropy, can **overfit** to training data

Overfitting

- In which a training method gets *too* good at predicting training data at the risk of making mistakes during testing.

- Child sees three dogs, all of different breeds, and is told that they are each a different breed but all dogs
- Child sees a cat and asks what breed of dog it is

Random Forest

- A random forest makes use of a designer-specified number of decision trees, each trained on a random subset of the training data
- During testing, each tree of the random forest votes, with the majority (or average or median) answer taken.

Question(s) 2

Besides churn, pick some event in general or for a particular game designers might want to be able to predict.

Which technique would you use? Why?

How could the designers use the learned model to improve the game?

PCG high-level Model-based Methods

- Search
- Rule systems
- Generative Grammars
- Constraint Solving

http://pcgbook.com/wp-content/uploads/chapter02.pdf

JUMPED IN TO SEARCH ALREADY…

GA Pseudocode cont

- **Mutate**: Given some probability, randomly replace a member of the population with a neighbor
  - Make a random change
- **Crossover**: Take pairs of the initial population (chosen based on fitness), and combine their features randomly till population grows up to size *Y* (where *X*<*Y*)
- **Reduce**: Reduce the size of the population back down to *X*

Genetic Algorithms

Pros

- Middling authorial burden (more than hill climbing, less than generative grammars/rule system)
- High likelihood of finding global optima-ish

Cons

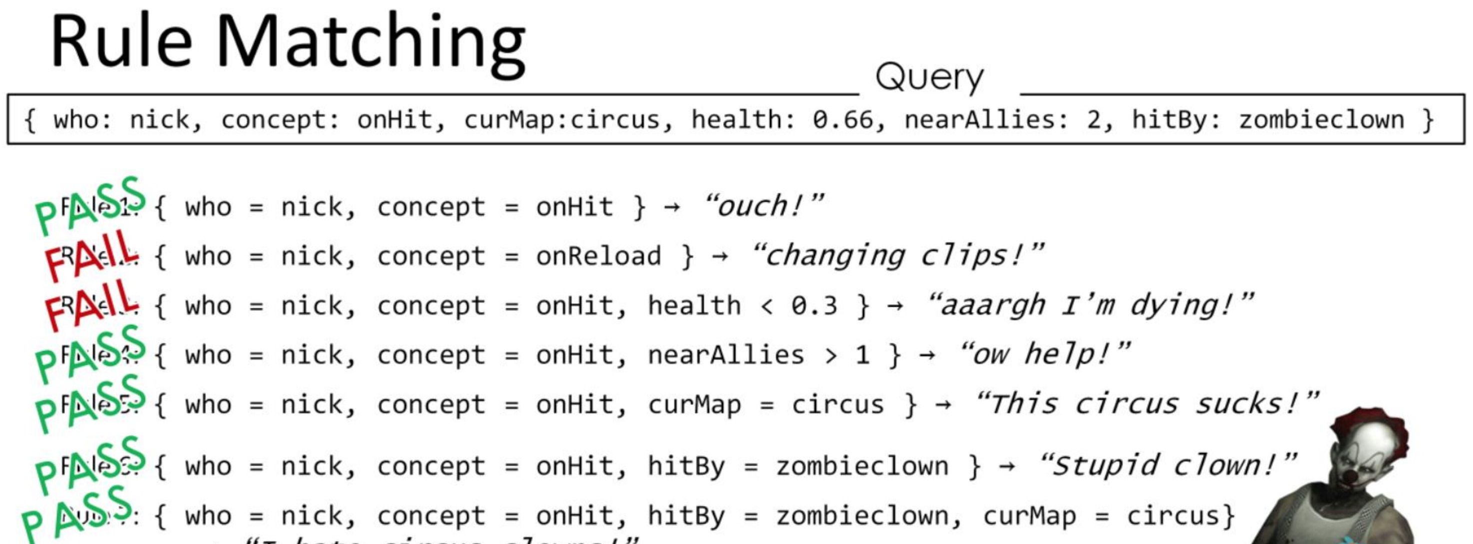
- Takes skill to pick and balance mutation/crossover

PCG: RULE SYSTEMS

Rule Systems

- Similar to the rule systems discussed before
  - If (world has certain conditions)
  - Then (make change to world)

- Now focused on building game bits/spaces, with each rule focusing on one feature
  - Example: *if* y<groundheight *then* set voxel to ground
  - This is how Minecraft world generation works



Rule Matching

Query

```
{ who: nick, concept: onHit, curMap:circus, health: 0.66, nearAllies: 2, hitBy: zombieclown }
```

```
PASS Rule1: { who = nick, concept = onHit } → “ouch!”
FAIL Rule2: { who = nick, concept = onReload } → “changing clips!”
FAIL Rule3: { who = nick, concept = onHit, health < 0.3 } → “aaargh I’m dying!”
PASS Rule4: { who = nick, concept = onHit, nearAllies > 1 } → “ow help!”
PASS Rule5: { who = nick, concept = onHit, curMap = circus } → “This circus sucks!”
PASS Rule6: { who = nick, concept = onHit, hitBy = zombieclown } → “Stupid clown!”
PASS Rule7: { who = nick, concept = onHit, hitBy = zombieclown, curMap = circus}
        → “I hate circus clowns!”
```

Rule Systems Methods

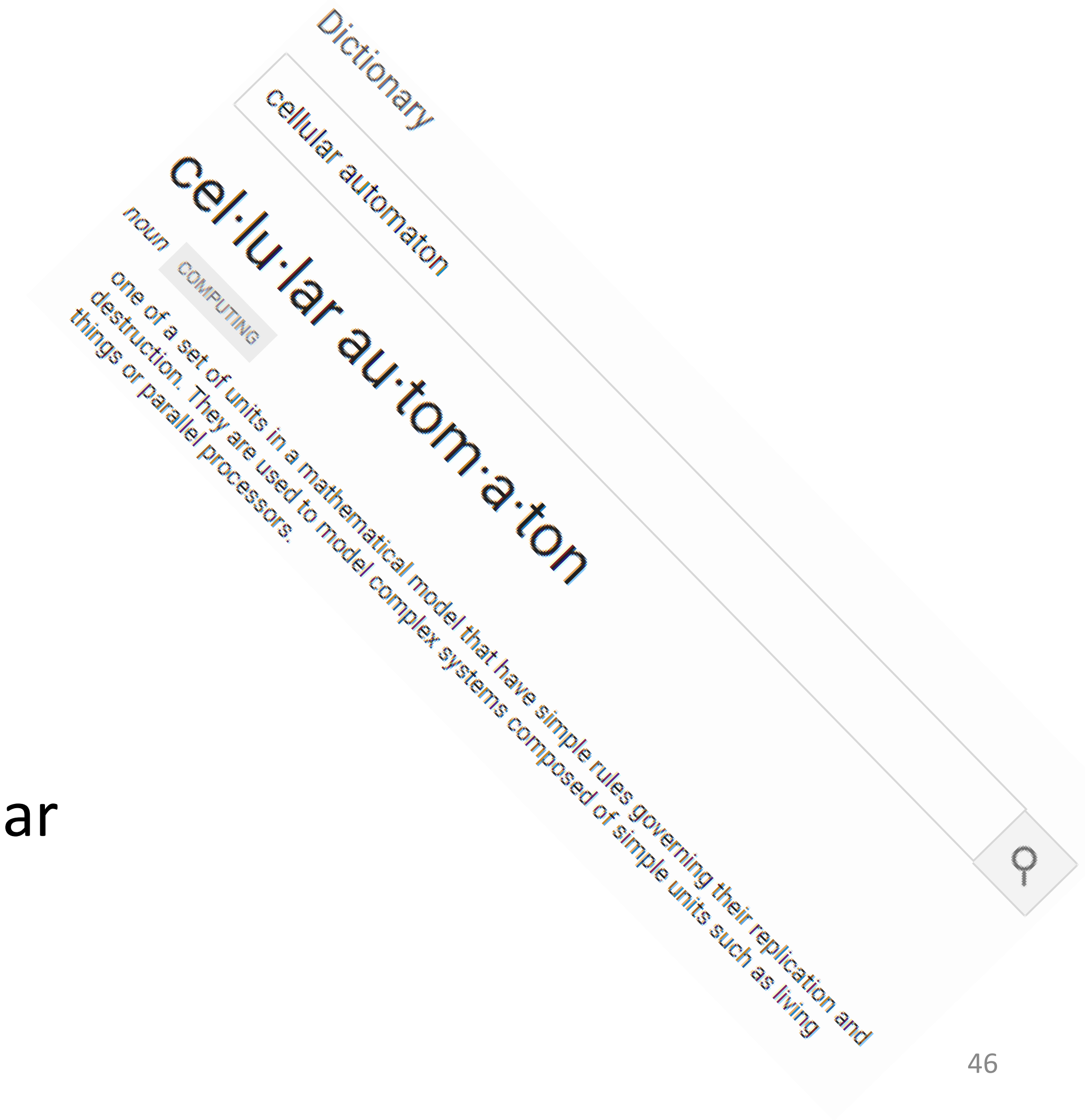
Cellular Automaton

1. Take noise/random values
2. segment bits/space into grids
3. Each grid updates until stable/time limit

Agent-based simulation

1. Take noise/random values
2. Iterate each agent over the world
3. Continue until stable/time limit

Where are the rules? In each grid cell (cellular automaton) or in an agent (agent-based simulation)?



Dictionary

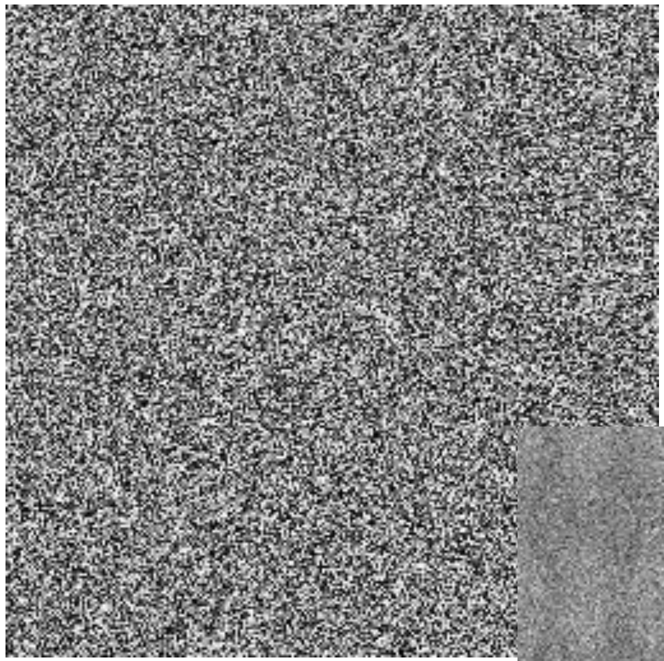
cellular automaton

cel·lu·lar au·tom·a·ton

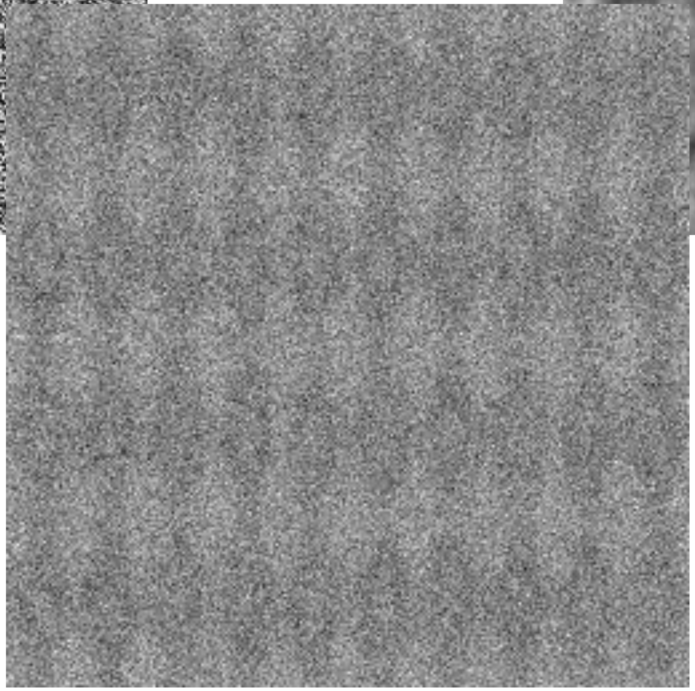
noun COMPUTING

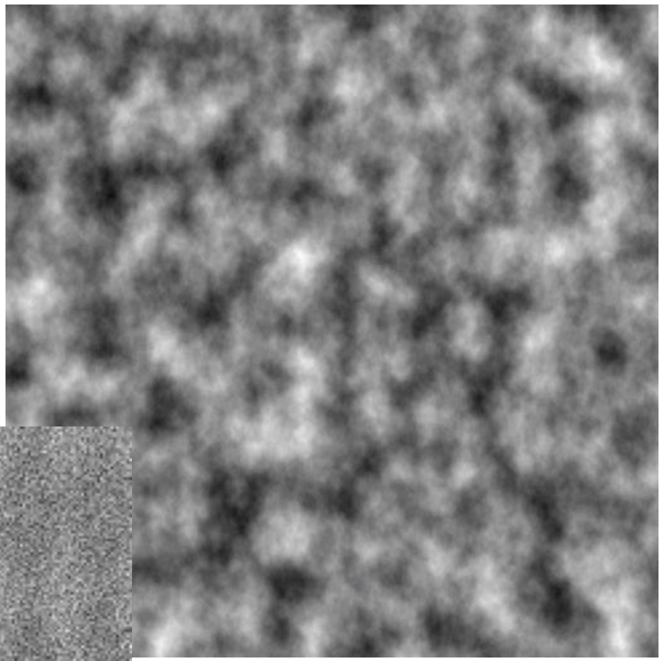
one of a set of units in a mathematical model that have simple rules governing their replication and destruction. They are used to model complex systems composed of simple units such as living things or parallel processors.

Quick note: Noise > Random


Pure “random”

Sin wave

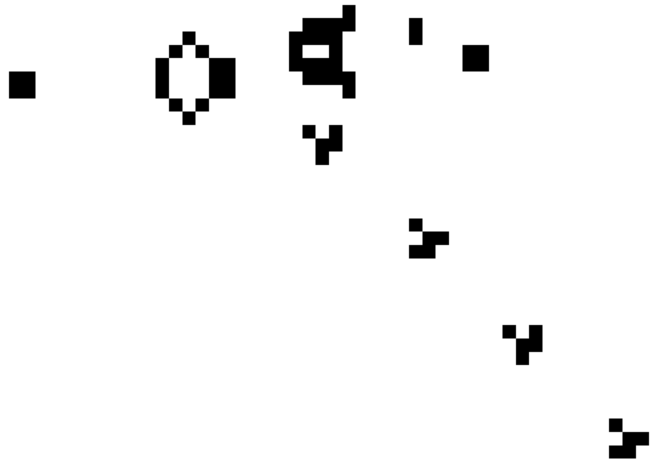



Perlin noise

Cellular Automaton

Stanislaw Ulam and John von Neumann (1940)

Conway's Game of Life made famous (1970s)



Rules:

1. Any live cell with fewer than two live neighbors dies, as if caused by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Cellular Automaton Example

https://www.youtube.com/watch?v=OumDj9PNWjE

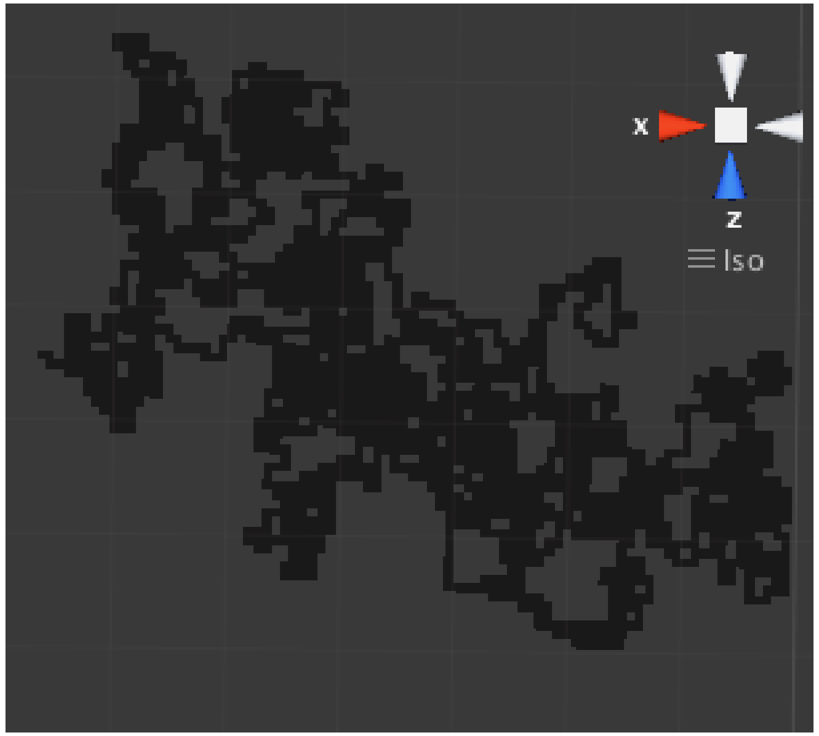
Cellular Automaton

Pros

- Easy to implement
- Complex behavior from small set of rules
- Intuitive mapping to game space generation
- Local coherence

Cons

- Reliant on guess and check for modifying
- Hard to spot edge cases
- No way to forbid undesired output
- Upper bound is slow
- Global incoherence



Question

Cellular Automata are good at local coherence and bad at global coherence, which limits its usage cases (caves, mazes, etc).

What could we add or change to this approach to allow for a designer to ensure global coherence?

What if…

- What if we had another set of rules, not for a single grid cell, but for the whole grid?
- Well, in that case we don’t have to use a grid at all. All the rules could be on this layer
- And we could even have rules that contradict one another, so we’d get…

Agents

- Instead of individual grid rules, have multiple agents that iteratively update a map.
  - Example:
    - Mountain agent: make high points higher
    - Valley agent: flatten out sections
    - River agent: make low points lower and add water
    - Manager: make parts of the map non-editable when they reach thresholds

Comparisons: CA vs Agents

- Similarities (rule system commonalities):
  - Rely on “emergent” output of simple rules
  - Hard to make strong designer restrictions
  - Requires a lot of guess and check
  - Can take a long time to converge
- Differences
  - Agents allows for some global assurances
  - CA more emergent in its output

Major Drawback of Rule Systems

- What cellular automaton could you construct to build a house? A sword? An NPC?
  - Local rules don’t adapt well to generating bits that require strong global coherence
  - Said another way, no one wants an NPC that only looks locally like a human

http://pcgbook.com/wp-content/uploads/chapter05.pdf

PCG: GENERATIVE GRAMMARS

PCG high-level Methods

- Search
- Rule systems
- **Generative Grammars**
- Constraint Solving

Solution: Generative Grammars

- Noam Chomsky’s study of languages in the 1950s and 60s
  - Rich taxonomy of grammars
  - Widespread application
- Key questions: determinism & order of expansion
- Components with (grammar/production) rules about how they are allowed to be put together
  - Words that can be placed together in a sentence according to grammar rules
  - Limbs that can be combined to form creatures according to designer rules

http://pcgbook.com/wp-content/uploads/chapter05.pdf

Formal Grammars

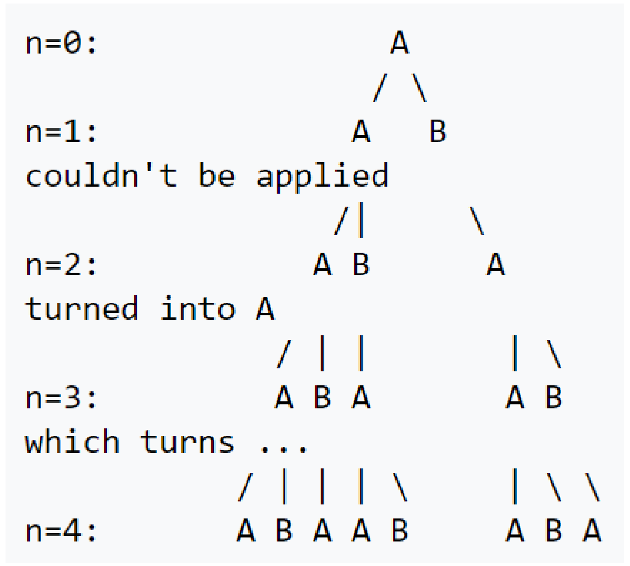
- Grammar: set of **production rules** for rewriting strings
  - Rule: <symbols> → <other symbols>
- E.g. (left hand side vs right hand side; terminal vs nonterminal)
  - A → AB
  - B → b
  - Given ‘A’: [1] AB [2] ABb [3] ABbb
- Applying: for each sequence of LHS symbols in string, replace with RHS of rule

Determinism & Order

- Determinism
  - Deterministic grammar: one rule that applies to each symbol or sequence
  - Nondeterministic grammar: several rules may apply
    - Random
    - Probabilities & Priors
- Rewriting order
  - Sequential: left to right, rewrite string while reading it
  - Parallel: all rewriting done at same time

L-systems

- Aristid Lindenmayer, 1968: model growth of organic systems
- Class of grammars w/ parallel rewriting
  - No preference on left/right expansion
  - Grammar rules lead to self-similarity / fractal-like forms
- EG (Given A): A → AB B → A
  - A
  - AB
  - ABA
  - ABAAB
  - ABAABABA
  - ABAABABAABAAB



```
n=0:                 A
                    / \
n=1:               A   B
couldn't be applied
                  /|     \
n=2:             A B      A
turned into A
               / | |       | \
n=3:           A B A       A B
which turns ...
             / | | | \     | \ \
n=4:         A B A A B     A B A
```

https://en.wikipedia.org/wiki/L-system

Application and Interpretation

- One approach: interpret strings as drawing instructions
  - F: move forward 10 pixels
  - L: turn left 90
  - R: turn right 90
  - EG: F → F L F R F R F L F

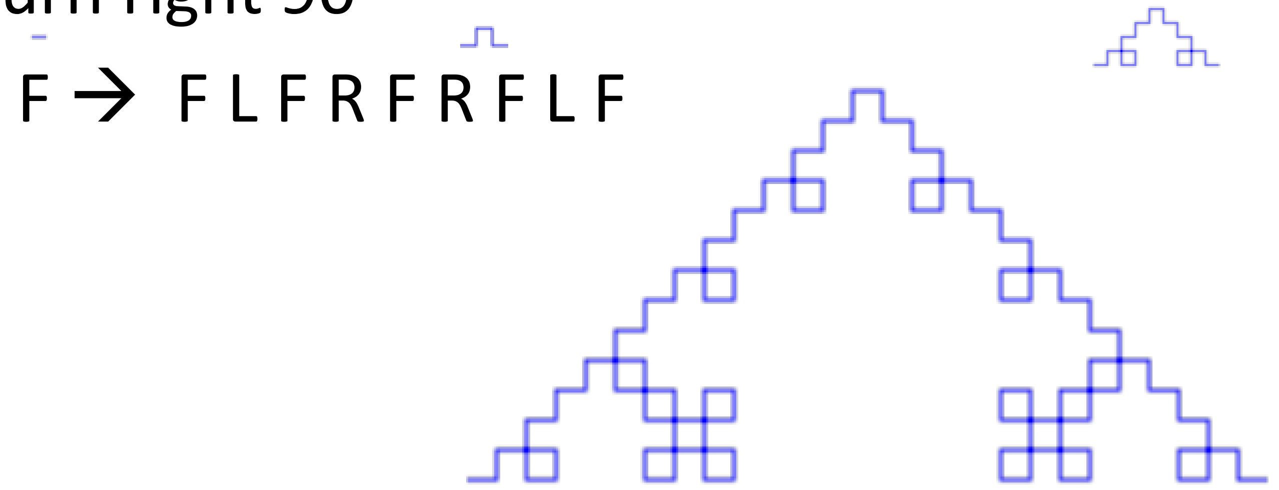
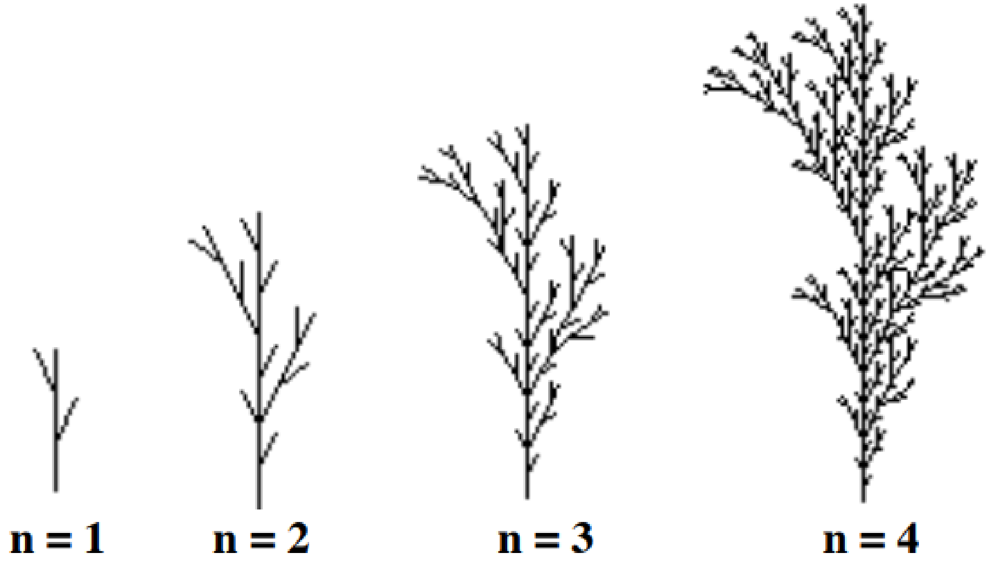


Fig. 5.2: Koch curve generated by the L-system $F \rightarrow F + F - F - F + F$ after 0, 1, 2 and 3 expansions

http://pcgbook.com/wp-content/uploads/chapter05.pdf

Bracketed L-systems

- How do we "lift the pen"?
  - Plants are branching, and branches end
- Introduce two extra symbols: '[' and ']'. A FILO stack
  - [push current position and orientation
  -] pop & use last saved position
- EG: F → F[RF]F[LF][F]



http://pcgbook.com/wp-content/uploads/chapter05.pdf

Barnsley fern

Barnsley's 1988 book *Fractals Everywhere* is based on the course which he taught for undergraduate and graduate students in the School of Mathematics, Georgia Institute of Technology, called Fractal Geometry.

constants : + − [ ]

start : X

Production rules :

X → F+[[X]-X]-F[-FX]+X,

F → FF

https://en.wikipedia.org/wiki/Barnsley_fern

Beyond strings

- Generative grammars are not restricted to representation as strings
  - Graphs, tile maps, 2D/3D shapes, etc.
- Graph grammar:
  - Find subgraph in target that matches LHS; mark subgraph w/ IDs
  - Remove all edges between marked nodes
  - Transform marked nodes into corresponding RHS
    - Add a node for each node on RHS not present in target
    - Remove any nodes that have no corresponding node on RHS
  - Copy edges as specified by RHS
  - Remove all marks

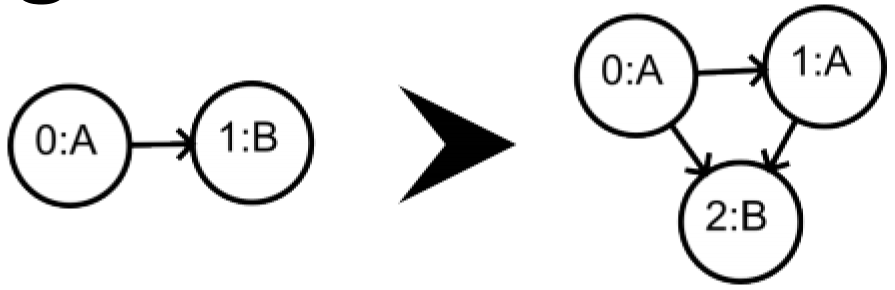
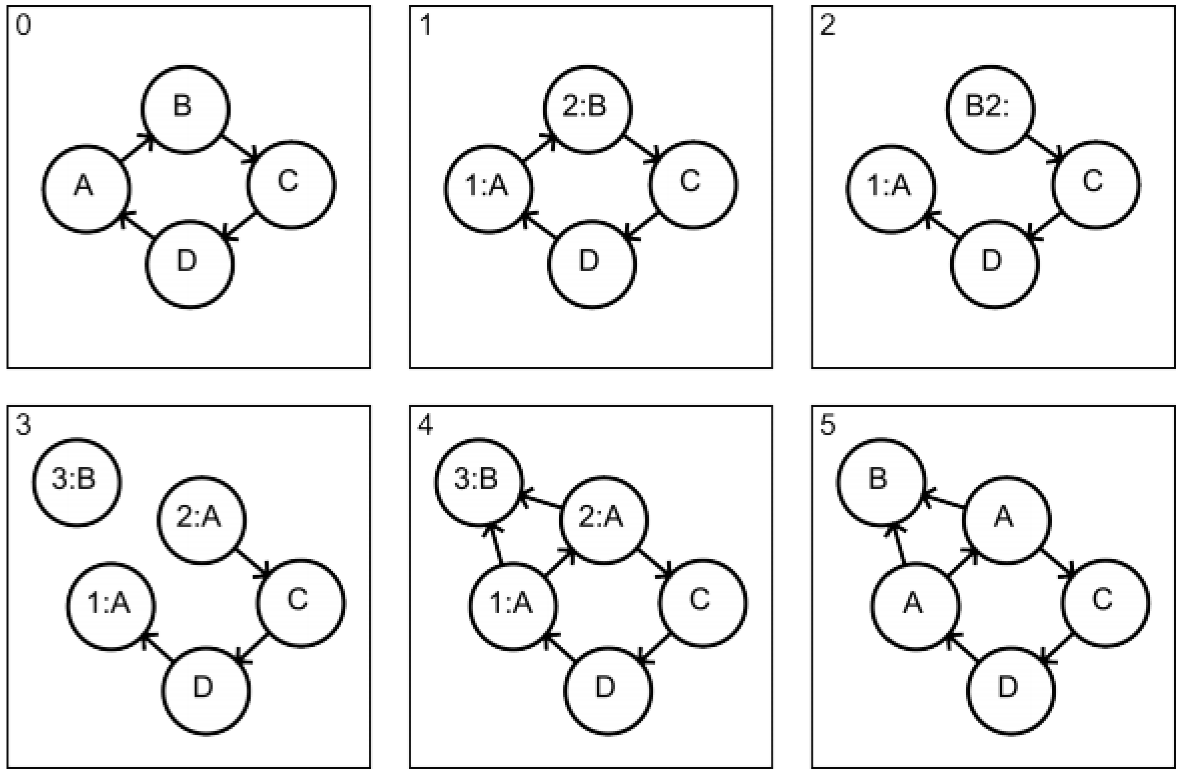


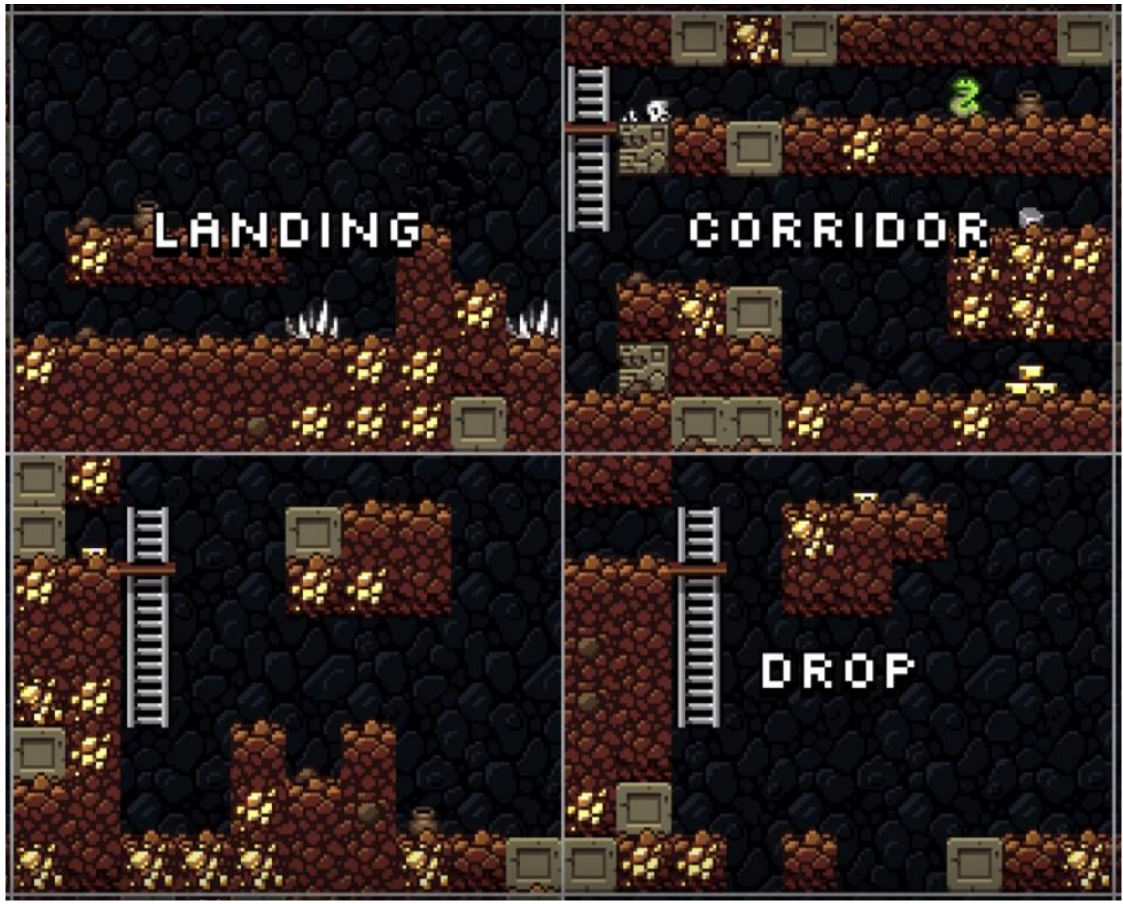
Fig. 5.7: A graph grammar rule



http://pcgbook.com/wp-content/uploads/chapter05.pdf

Example: Spelunky Levels

- Each Spelunky level starts life as a grid.
- Each grid can be filled in with potential templates designed by Derek Yu (designer)
- Templates are filled in one by one
- Later templates chosen according to rules about how templates can neighbor



Game Maker's Toolkit: https://www.youtube.com/watch?v=Uqk5Zf0tw3o

Generative Grammars

- Generative grammars are well-suited to both game bits and game spaces
  - Borderlands’ guns
  - Bloodborne’s chalice dungeons
- …but they are only as good as the components and rules the designer creates
  - No man’s sky’s ship and creature generation

Generative Grammars

Pros:

- General usage
- High-quality, “feels designed” quality
- Accessible
- Fast

Cons:

- Large burden on design
- Hard to “debug”
  - How do you know if a fix actually fixed anything unless you generate infinite output?

Comparing the Approaches

- **Rule Systems** allow for emergent output from simple rules. But hard to control output.

- **Generative grammars** can create high-quality output, but depend upon expert authoring of components and production rules

- **Search** cuts back on authoring burden to just a heuristic and searchable representation, but these are unintuitive for many designers