UNIVERSITÀ DEGLI STUDI
DI MILANO

Procedural Content Generation

Part 1

A.I. for Video Games

Davie Gadia (gadia@di.unimi.it) and Dario Maggiorini (dario@di.unimi.it)

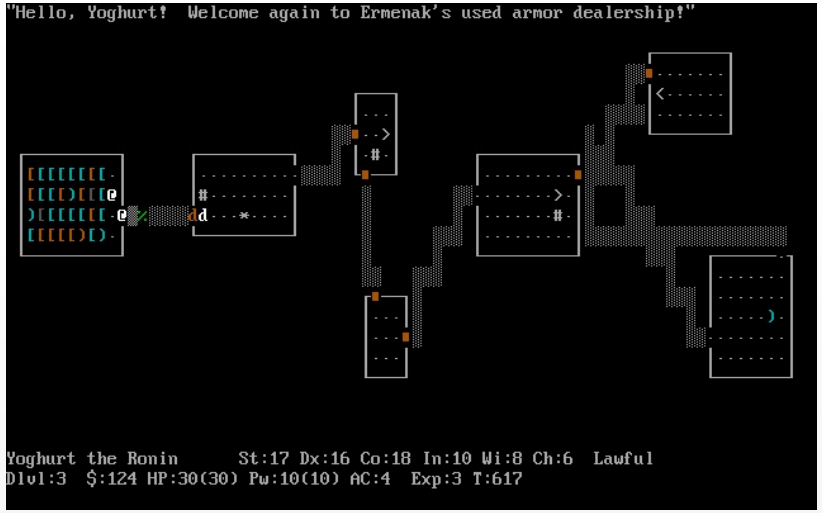
Procedural Content Generation

- PCG is the algorithmic creation of game content with limited or indirect user input

- We want to create a computer software that can provide game content on its own, or together with one or many human players (or designers)
  - Levels generation
  - Evolving players, NPC, and weapon

A Common Misconception

- PCG is NOT “just” about creating content on the fly



- PCG can reference any tool assisting a designer to create better content by taking additional decisions or validating data created by the users
  - Such as an assisted map level creator

Believe it or not, these two games are using the same algorithm.

Original is on the LEFT!

Why PCG ?

- To reduce creation time
  - Less time required
  - Less designers required
  - Games can be more profitable
  - *Developers' revenge over designers?*

So, why are we still paying those "creative guys" in the first place?

*… actually **NOT**!*

Because no algorithm can be 100% sure your content is fun to play!

- To make a game technically "infinite"
  - And you can overcome storage limitations

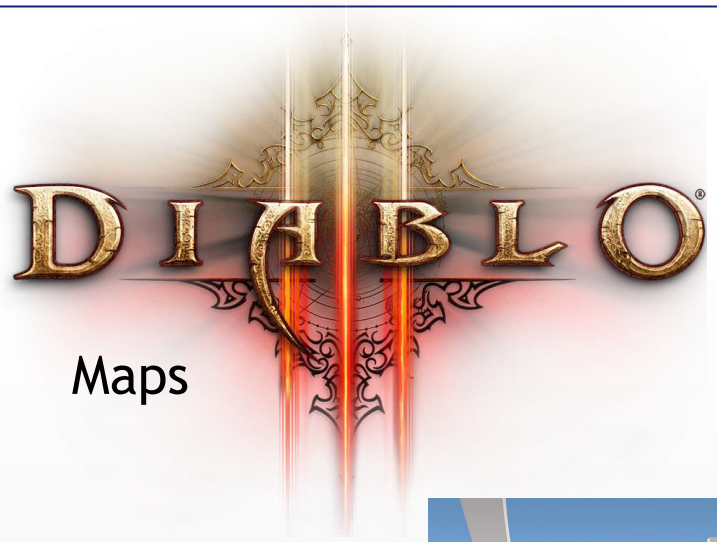
Elite was using a single number to generate the whole (universe) map.

Technically, we had a universe condensed in 32 bits

- To tailor the content to the taste or need of your player
  - And this will make your game more enjoyable

Many games are generating enemies or maps adapting to the player skill, in order to keep he/she in the flow.

Examples of PCG in Games



Maps

Maps and biomes

Textures

Animations

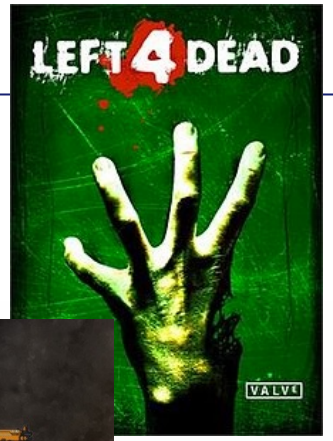
Player “skills”

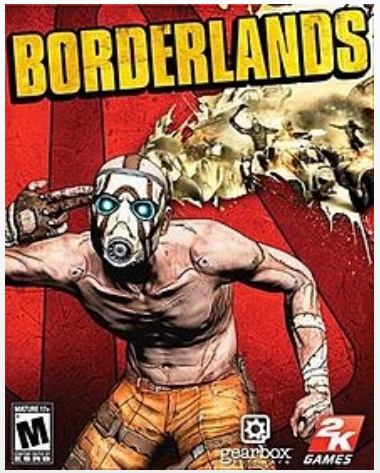
PCG Properties [Wish] List

- Bounded computational time
  - Depending on created content
- Reliability
  - Must satisfy a minimum quality level
- Controllability
  - Generation must be controlled/driven by the human counterpart
- Minimum expressivity and diversity
  - Avoid generating minimum variations around a starting point
- Believability
  - Avoid giving the impression the content has been generated via PCG :)

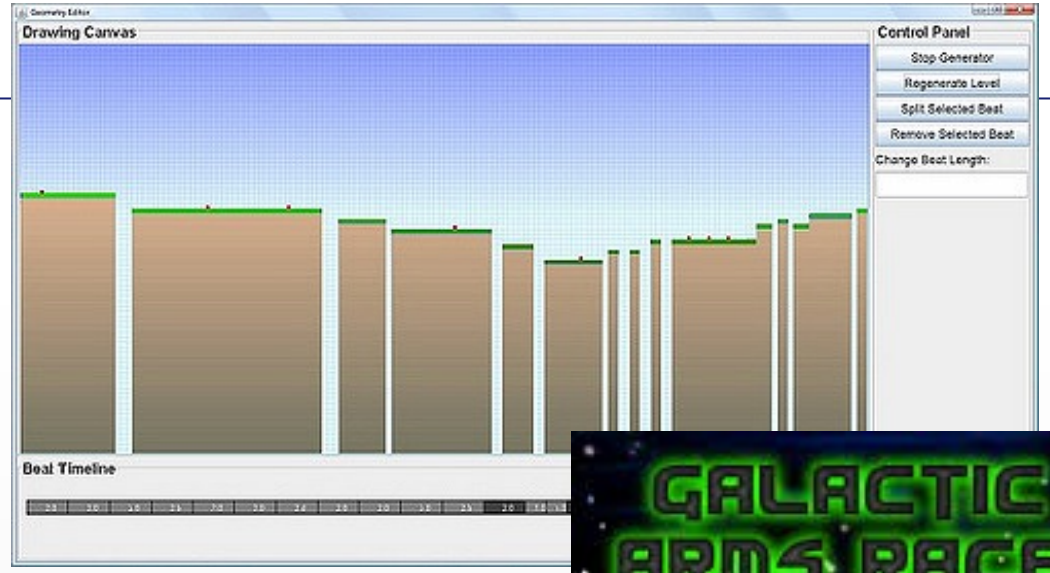
PCG Taxonomy

- Online vs Offline
  - On the fly (in-game) or before building the game
- Necessary vs Optional
  - Generating required content or auxiliary one
- Degree and Dimensional control
  - Use a single random seed or a wide set of parameters

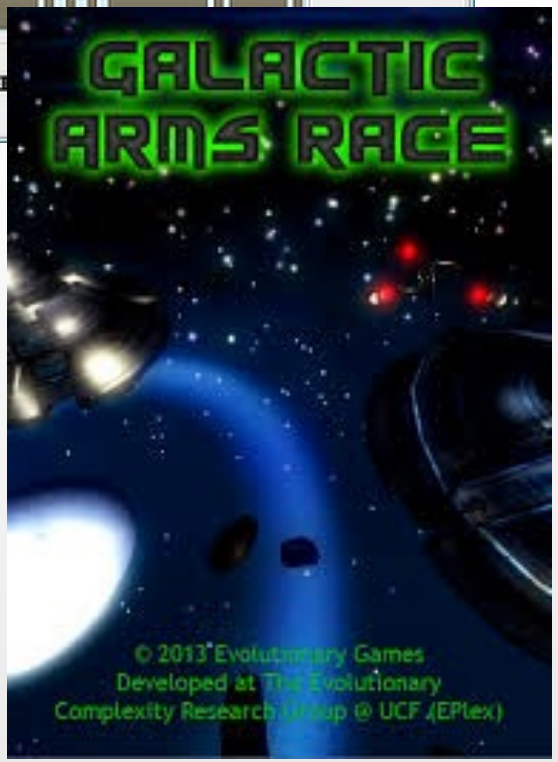




PCG Taxonomy



- Generic vs Adaptive
  - Generate by itself or take players into account
- Stochastic vs Deterministic
  - Completely random or start from a seed
- Constructive vs Generate-and-test
  - Use one pass or iterate generation and test until a certain criteria is met
- Automatic generation vs Mixed authorship
  - Set parameters and generate or have player/designer cooperate with the algorithm



Generating Spaces

- One common usage of PCG is to generate maps
  - Not intended just as “terrain” rather than “where the player is moving and interacting”
- The goal, in this case, is to use an algorithm to layout a set of interconnected areas
  - Interconnection MUST be navigable

Creating a Dungeon

- A dungeon is ….

  “*a labyrinthic environment, consisting mostly of interrelated challenges, rewards and puzzles, tightly paced in time and space to offer highly structured gameplay progressions*”

- Beyond geometry and topology, dungeon must include:
  1. NPC
     - E.g., monsters to slay, princesses to save
  2. Decorations
     - Bare stone walls are not keeping the player there for very long
  3. Objects
     - E.g., treasures to loot

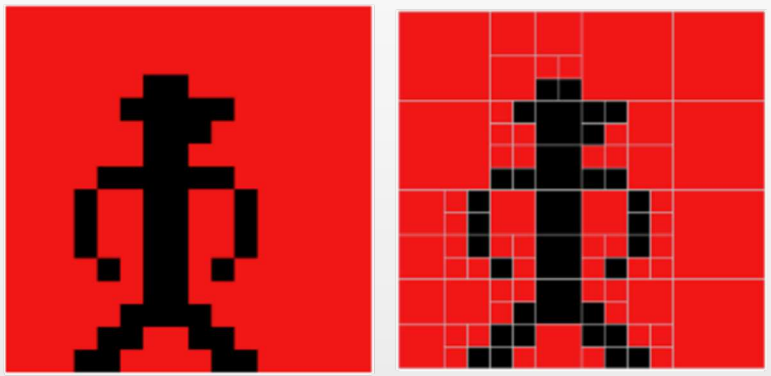
Space Partitioning

- Useful in general to create indoor maps
- The space is recursively divided in disjoint subsets
  - We are used to call such subsets cells
- A space partitioning tree is created during the processing
  - In many cases, it is going to be a binary tree
    - BSP – Binary Space Partitioning
  - The tree data structure can be used to link rooms avoiding unwanted intersections
- This is originally a technique used to represent images

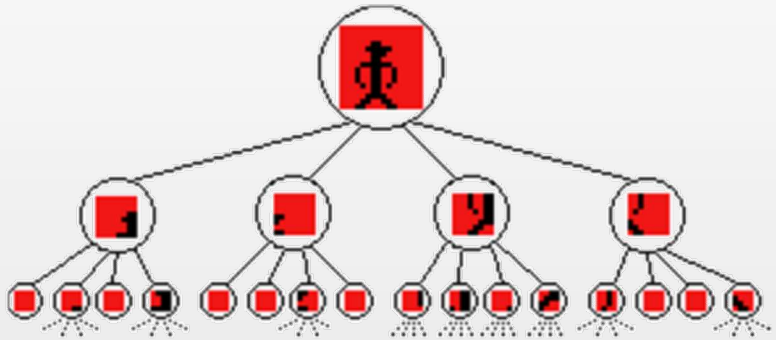
Representing Images Using Space Partitioning

- A quadtree can easily represent squared images
  - A 2^nx2^n image will fit in a tree with a maximum deep of n

- Branches are split in other branches or leaves iff multiple colors are present

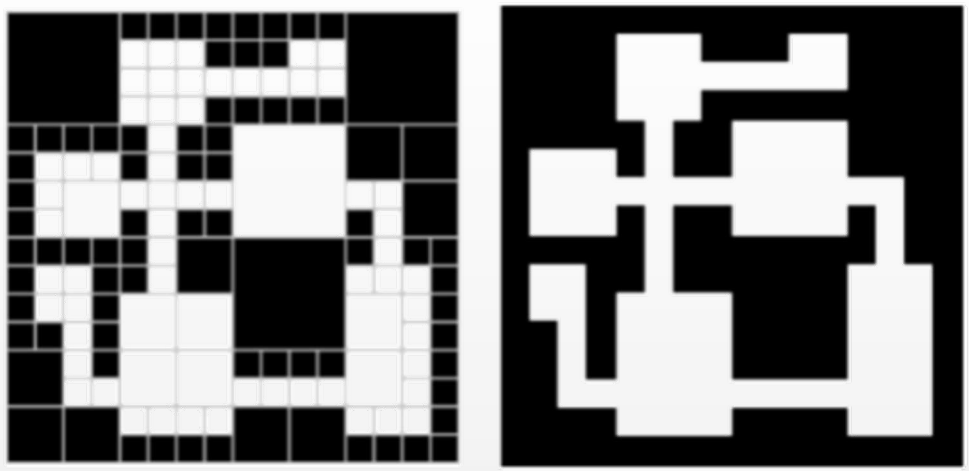


(a) Image and partition



(b) Quadtree

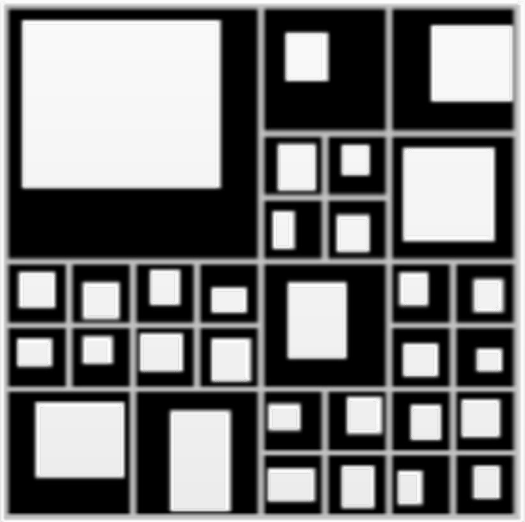
In the Same Vein …



- We can keep splitting the map and associate one color to free space
- Unfortunately, with this approach we end up creating very fragmented and small rooms
  - This is because it is difficult to coordinate different branches associated to nearby locations
- Moreover, in this case it is not easy to guarantee corridors or avoid a partitioned labyrinth

In the Same Vein …

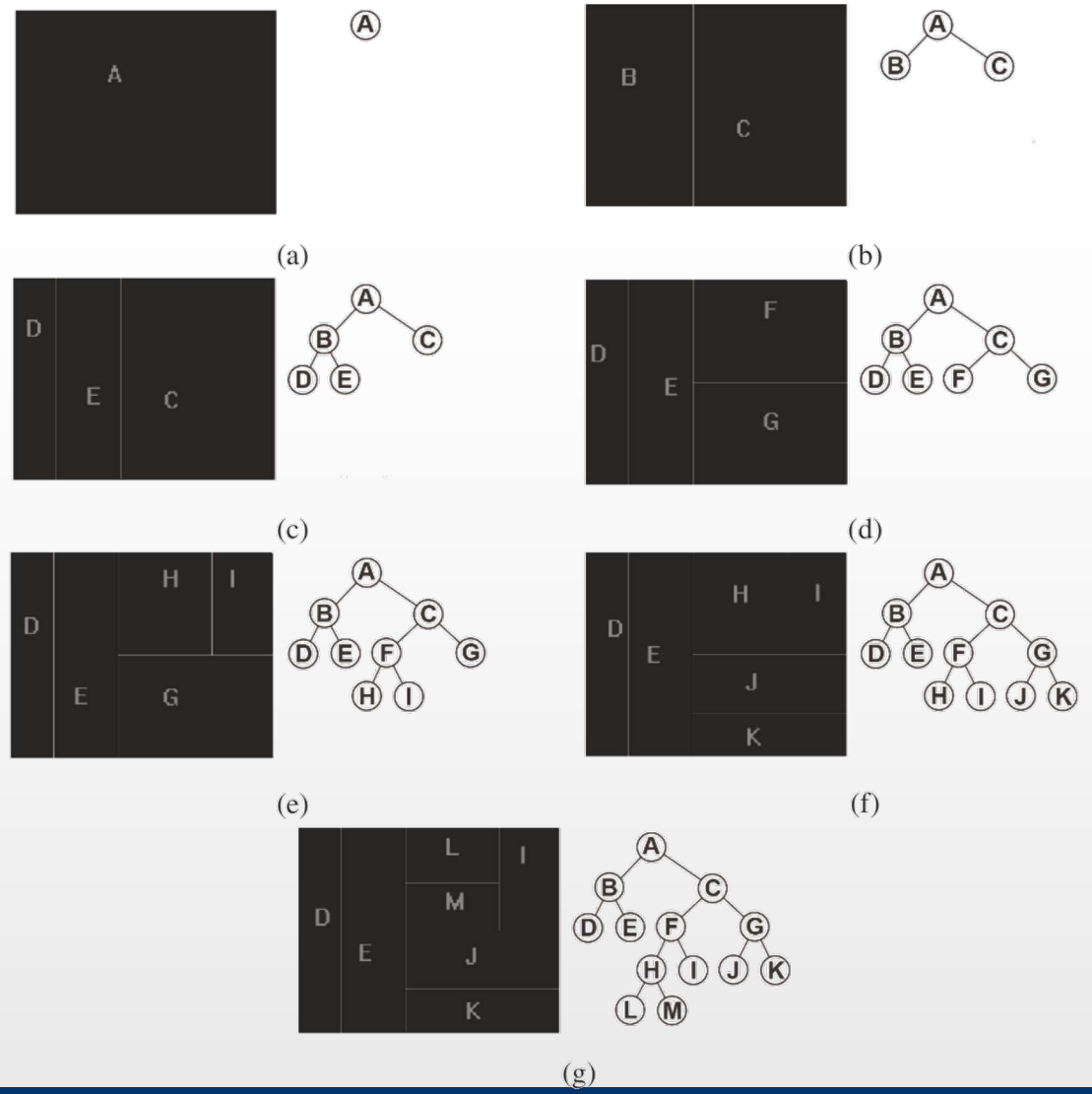
- A much better approach is to:
  1. Partition the space in a geometric way
  2. Put rooms in each space
  3. Create corridors leveraging on the tree data structure



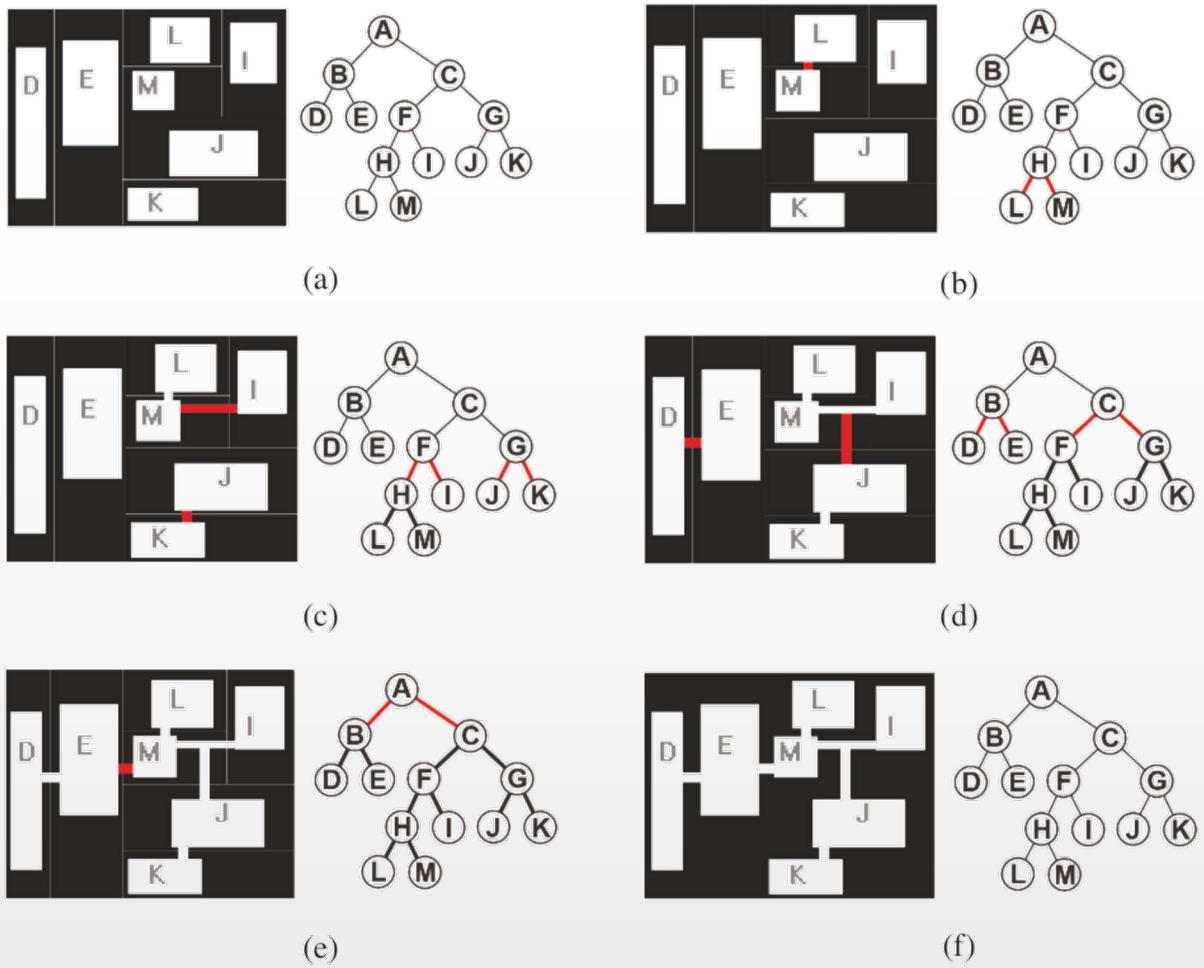
A Simple Space Partitioning Algorithm

1. Start with the whole area (root node)
2. Select a random cell whose size is greater than the minimum room size
3. Split the cell area horizontally or vertically
   - Random split, NOT in halves
4. If there are still eligible cells go to step 2
5. For every cell, create a random room
   - Pick two points inside its areas as corners
6. Starting from the lowest level create corridors connecting room belonging to children of the same parent

Space Partitioning



Space Partitioning



Possible Variations

- Set a target in term of number of rooms
- Set a maximum cells size
- Set constraints to aspect ratio
- Your idea here: _____

Agent-Based Dungeon Growing

- The problem with space partitioning is that the layout is very regular, and the player cannot get lost
  - This is fine if it is the goal of your level design, but will not provide great variety for a professional dungeon crawler

- Think about having an agent dig tunnels and creating rooms in sequence
  - This is a m*icromanagement approach*
  - The result will be more “organic” and less organized
  - Difficult to predict results without extensive trials and errors

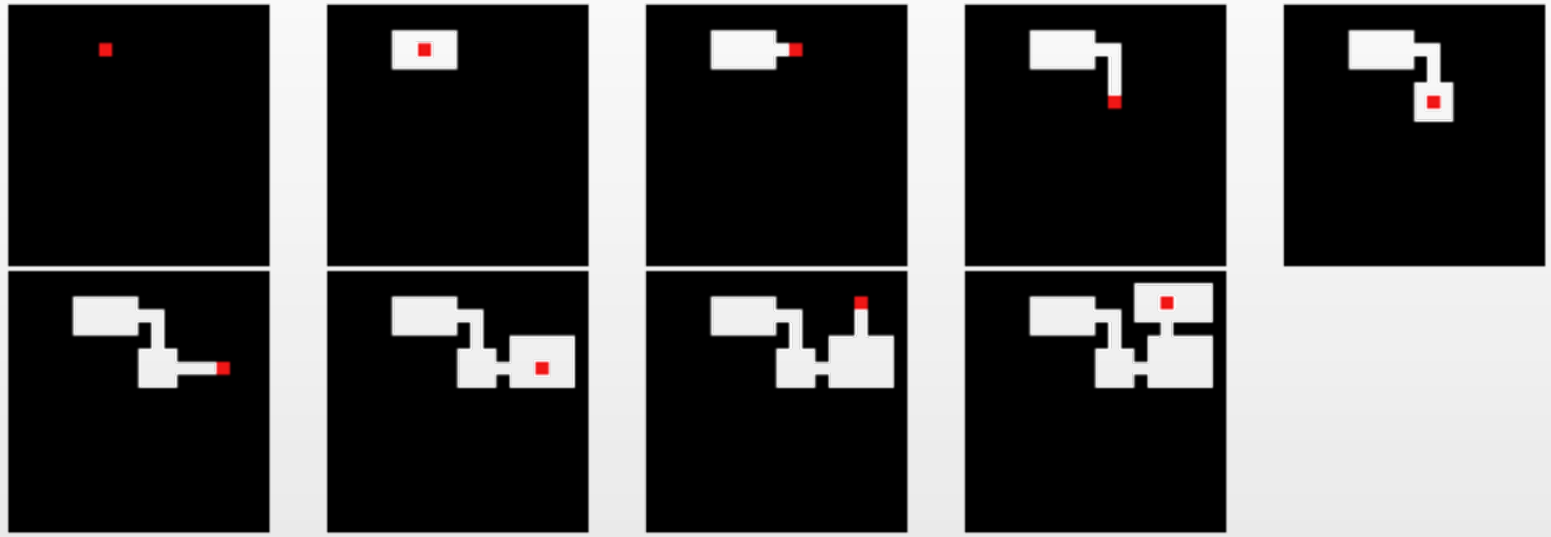
- We try to create an A.I. to generate dungeons
  - An automated miner?

A Completely Random Algorithm

1. The agent is set in a random place of the map digging in a random direction
2. Dig a tile in the facing direction
3. Randomly decide to:
   - Place a room (random size)
   - Change direction
4. If an action is taken, the probability to do it again is set to 0
5. If an action is NOT taken, the probability to do it is increased by a fixed amount
6. If the dungeon is not big enough go to step 2

A Better Digging Algorithm

- If the agent is “informed” about the surrounding, better results might be achieved
  - A room can be placed iff it will not overlap a corridor or another room
  - When changing direction, a random digging distance is also selected



Cellular Automata

- Can be used to simulate more natural digging with a lifelike feeling
  - Such as the digging of a group of miners trying to reach a vein

- Suitable for large areas
- Avoid square rooms
- Increases diversity and believability

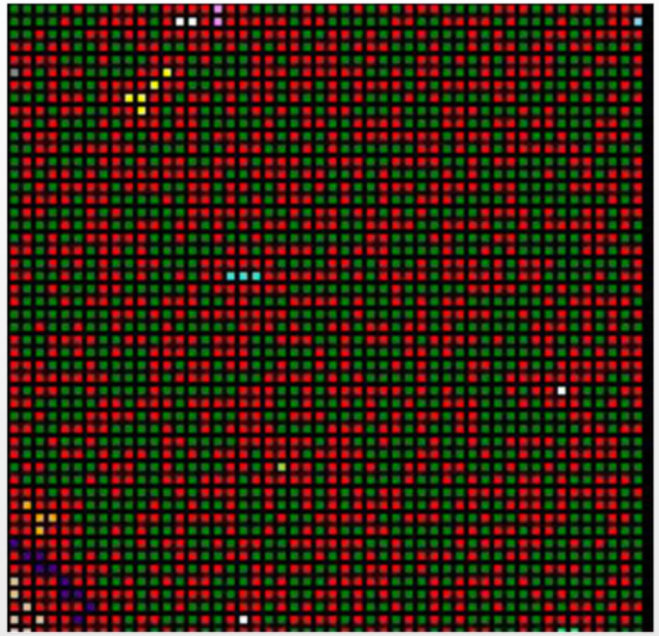
Cellular Automata

- A cellular automaton is a discrete computational model composed by:
  - An n-dimensional grid
  - A set of transition rules
- Each element of the grid is called a cell
  - Each cell can assume one of several states
    - In the simplest configuration, the states can be *on* and *off*
    - The configuration of all cells in the grid is defined to be the configuration of the automaton
- At each step *t*, every cell determinates its new states based on its own current state and the states of its neighborhood at step *t-1*
  - The neighborhood can span more than one cell in distance
- The application of cellular automata is not computationally intensive and can be performed online during a screen change

Recipe for a Cave with a Cellular Automaton

- We need to define the following:
  1. A grid where, for each cell, there can be empty space or rock
     - So, a binary state is enough for this
  2. A probability for each cell to be rock or empty at first

- And we will start with something like this:



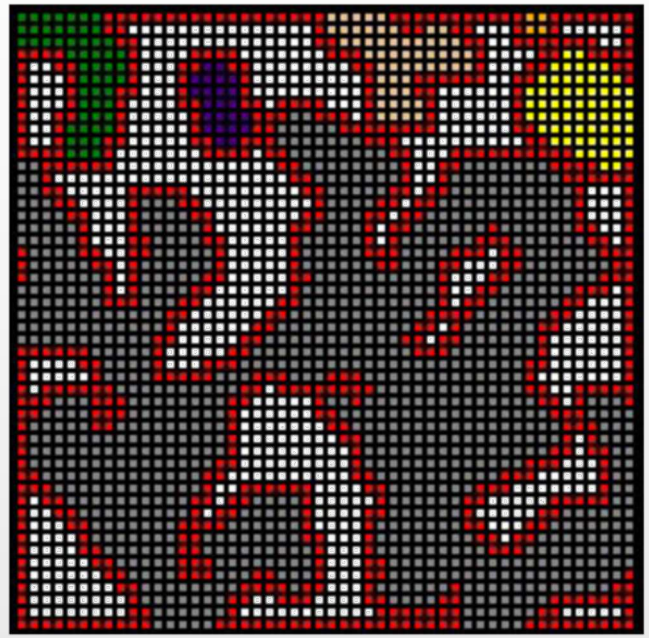
In this example, a cave of 50x50 has been used and the probability to have a rock in each cell is set to 0.5

Recipe for a Cave with a Cellular Automaton

- Then, we need to define how the automaton is evolving
  1. The number of neighborhood cells to consider for evolution
  2. A threshold of rock neighborhoods to be confirmed (or change into) a rock
     - Otherwise, the cell will become empty space
     - This threshold must be compatible with the number of cells in the neighborhood
     - This will make rocks cluster together
  3. The maximum number of iterations to run

- And, with just 4 iterations, we can achieve this:

Threshold has been set to 5 and the neighborhood size to 1



Study Material

- Procedural Content Generation in Games
  ISBN 978-3-319-42716-4
  by N. Shaker, J. Togelius, M. J. Nelson
  - you can download it for free from Unimi IP addresses from:
    https://link.springer.com/book/10.1007/978-3-319-42716-4
  - an open access version of the book is available at:
    http://pcgbook.com/

- Chapter 1
  Chapter 3 up to § 3.4 included