
Hogeschool van Amsterdam
Amsterdam University of Applied Sciences

UNIVERSITY
OF AMSTERDAM

Measuring Quality of Grammars for Procedural Level Generation

Procedural Content Generation Workshop
August 7th 2018, Malmö, Sweden

Riemer van Rozen[1,2,3] Quinten Heijn[3]

1. Amsterdam University of Applied Sciences – Play & Civic Media
2. Centrum Wiskunde & Informatica – Software Analysis & Transformation
3. University of Amsterdam – Master of Software Engineering

Live Game Design

- **Premise: Live Game Design project**
  - case study with Ludomotion
  - quality assurance of procedurally generated game levels
  - Ludoscope

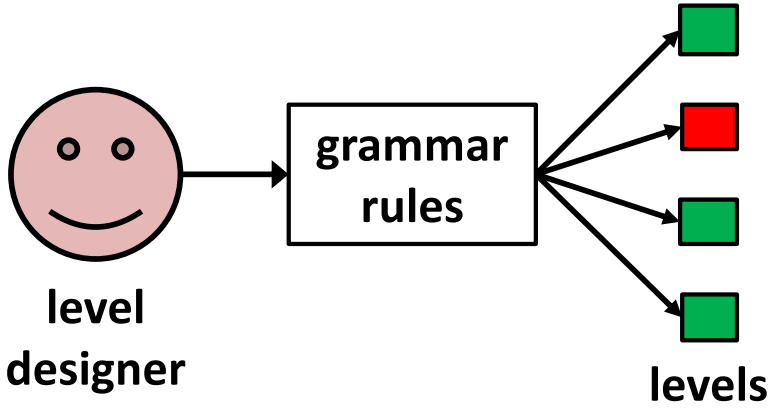
- **Problem.** grammar-based procedural level generation raises productivity of level designers at the cost of quality assurance

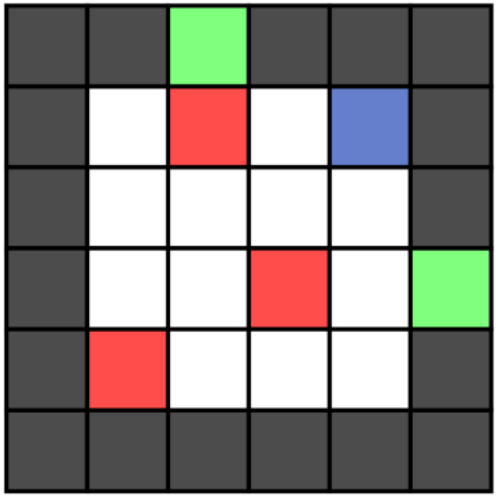
Unexplored: a roguelike dungeon crawl game that embeds a grammar-based dungeon generator

Ludomotion
Advancing Games

Problem Statement

- **Problem.** authoring, improving, maintaining grammars is difficult
  - lack of direct manipulation
  - hard to predict how each grammar rule impacts the overall level quality
- **Challenge.** lack of tools and techniques for debugging and testing
- **Question:** How can the quality of grammars that work on tile maps for procedural level generation be improved?
- **Objectives.** Better tools & techniques

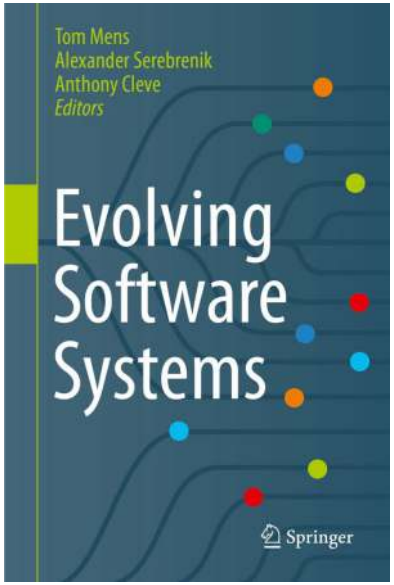




problematic level

Software Evolution

- **Software Evolution**
  - software conforms less and less to the changing expectations of its users
  - it becomes harder to adjust the software and maintain its quality
  - games = changing requirements
- **Two branches of research**
  1. **What evolves.** improve knowledge, plan and predict, e.g., by analyzing source code over time
  2. **How to evolve.** improve techniques for adjusting software to new requirements, e.g., with model transformations



Software Evolution. Editors: Tom Mens and Serge Demeyer. Springer, 2008

Evolving Software Systems. Editors: Tom Mens, Alexander Serebrenik, Anthony Cleve. Springer 2014

Approach and Contributions

- **Level Designer questions**
  1. **Efficiency.** No dead content?
  2. **Effectiveness.** Are the intended parts in the level?
  3. **Root-cause analysis.** Given a level with a problem, by which rules were the affected tiles generated?
  4. **Bug-fixing.** Does changing a rule improve levels, or does it also introduce new problems?
  5. **Bug-free.** How to test?

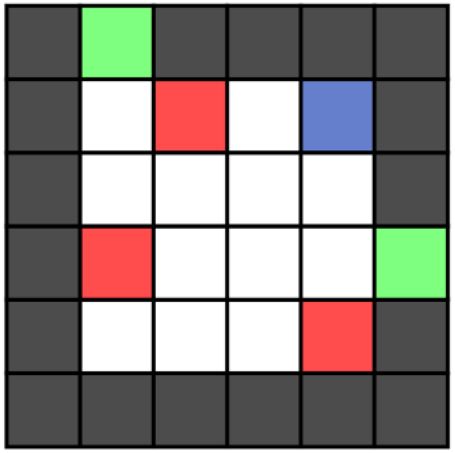
- **Approach**: use SE techniques to answer level designer questions

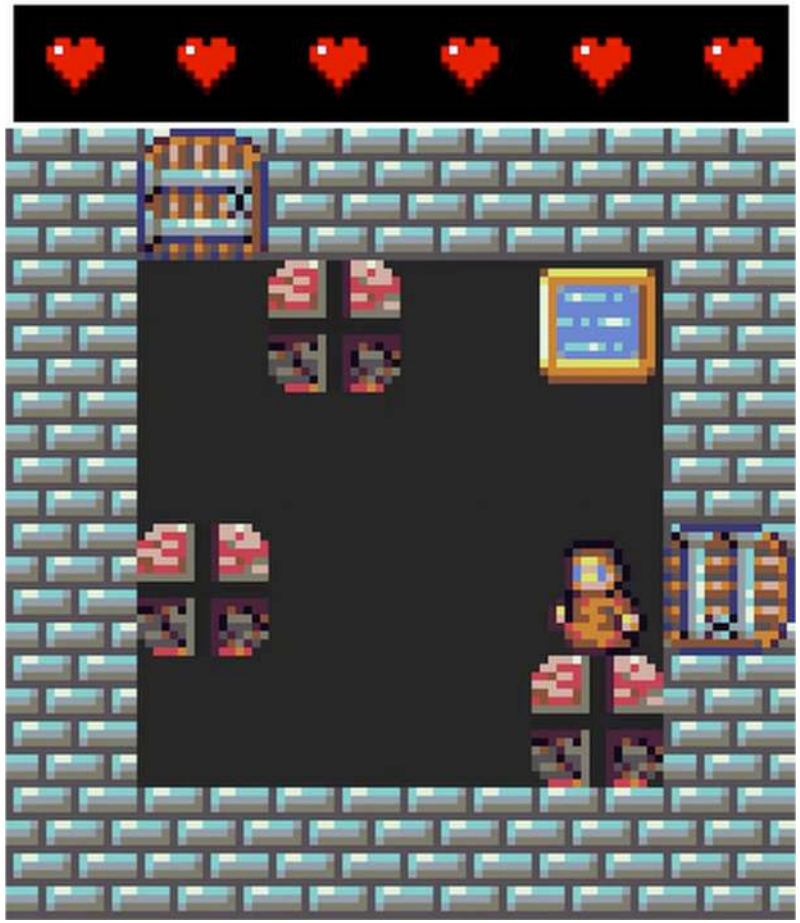
1. **Metric of Added Detail (MAD)**
   - Lines of Code (LOC): used to measure volume (or size)
   - Cyclomatic Complexity (CC): calculates branch points in control flow
   - addresses question 1

2. **Specification Analysis Reporting (SAnR)**
   - grammar rules perform model transformations
   - apply origin tracking, record transformations, analyze history
   - specify level properties
   - addresses questions 2, 3, 4, 5

Dungeon Room Generator

- **Dungeon room generator**
  - toy example
  - simple, representative
- **Goals**
  - Traverse the room
  - Evade fire pillar traps
  - Extinguish the flames
- **Level shown**

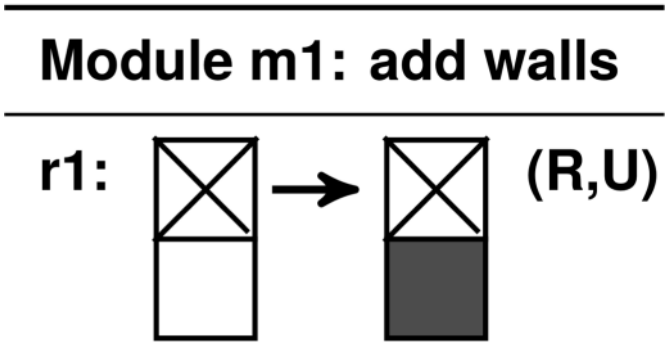




Dungeon room pipeline – stage 1

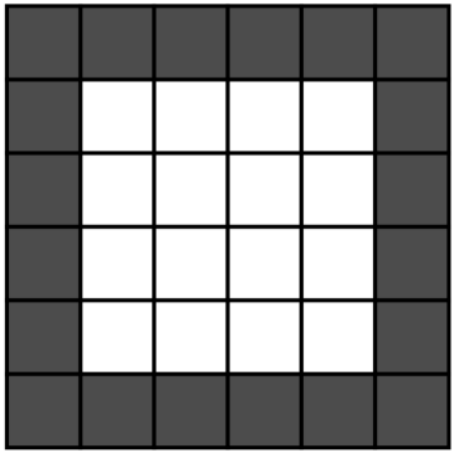
- First, we add walls

☒ =off map ☐ =empty ■ =wall



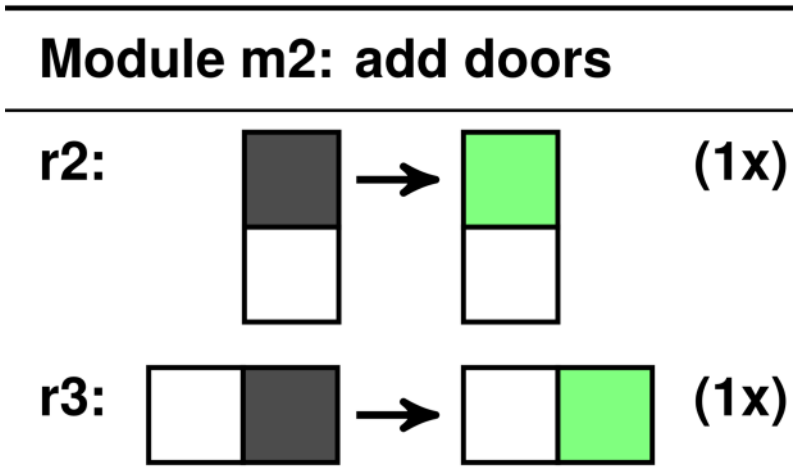
- Rule r1 replaces empty cells on the room borders by walls
  - (R) includes rule rotations
  - (U) repeats until the rule can no longer be applied

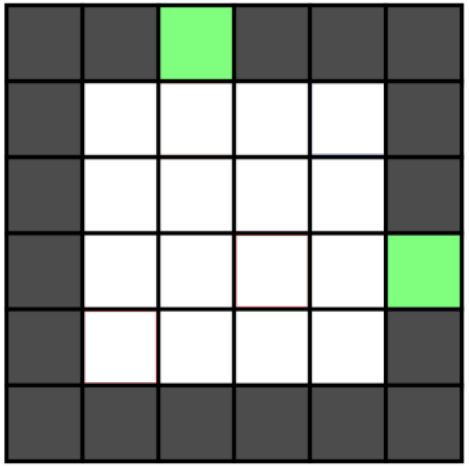
- Running the module results in an empty room with walls



Dungeon room pipeline – stage 2

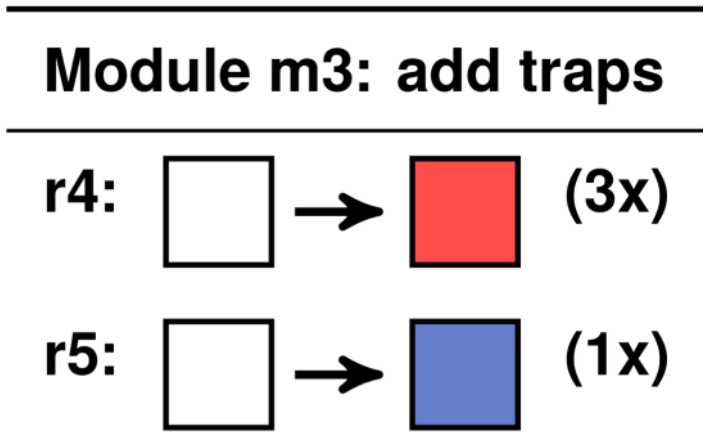
- Next, we add doors
  =door
- North door. Rule r2 replaces a north wall (neighboring an empty cell) by a door
- East door. Rule r3 replaces an east wall (neighboring an empty cell) by a door
- Running module m2 results in two doors in the walls

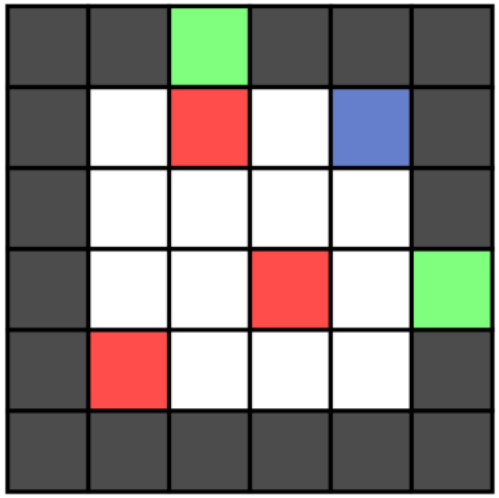




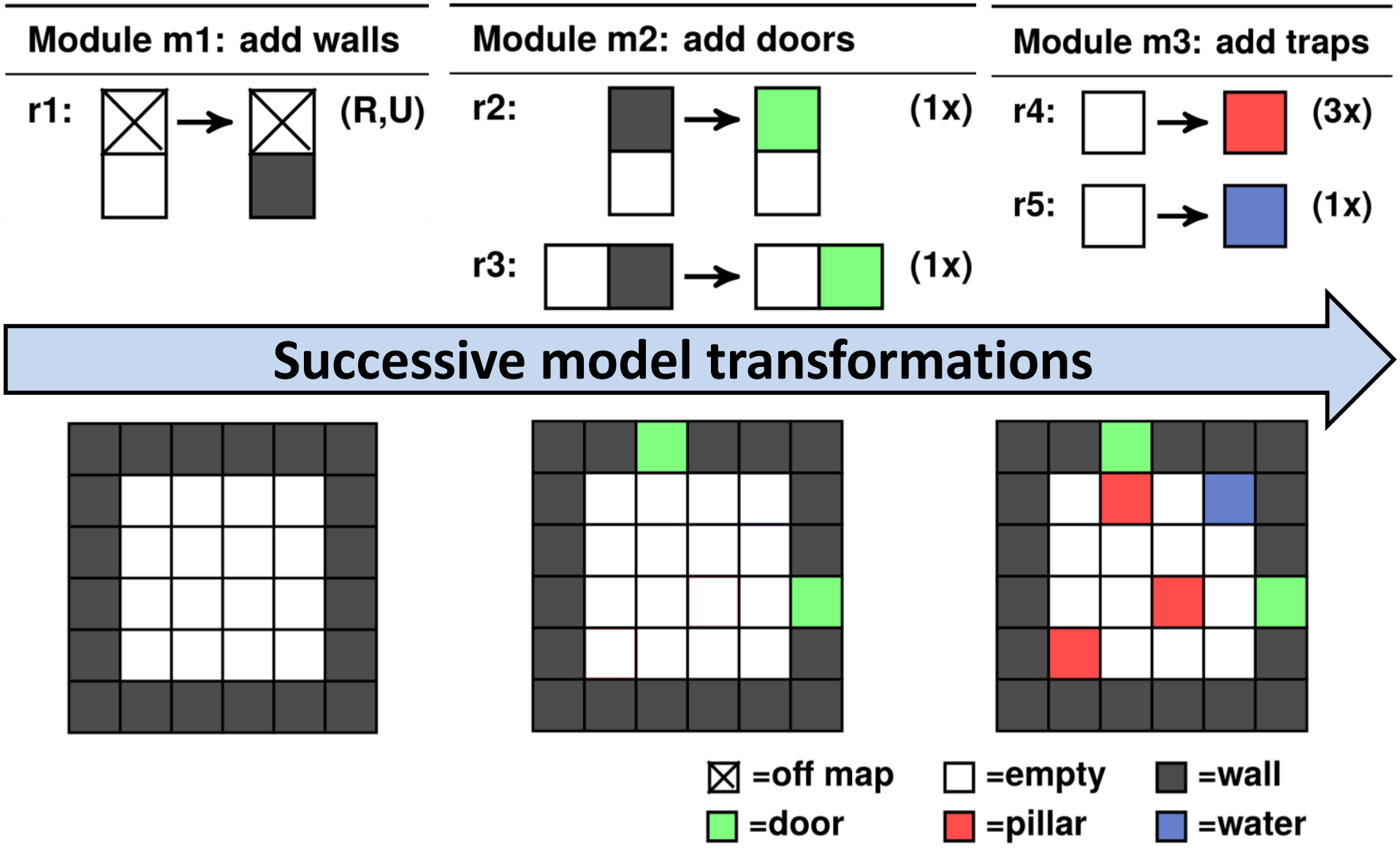
Dungeon room pipeline – stage 3

- Finally, we add a challenge
  =pillar =water
- Fire pillars set players on fire when they remain close for too long
- Water from a pond enables players to extinguish the flames
- Running module m3 results in a room populated with three fire pillars and a pond



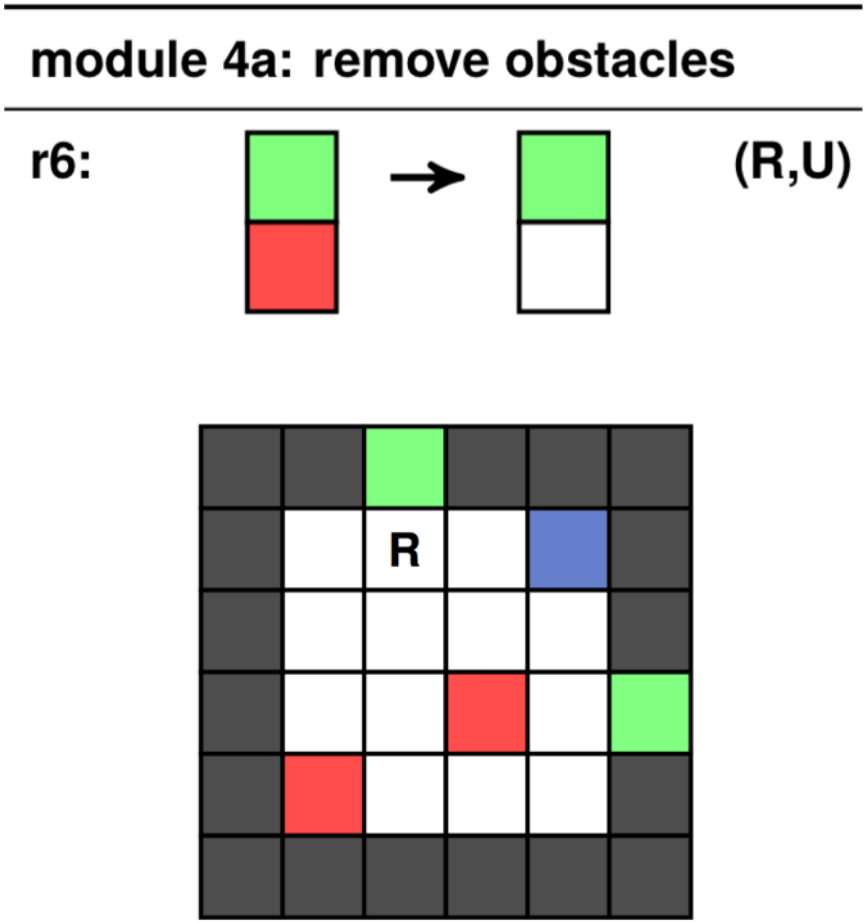


Dungeon room pipeline – complete



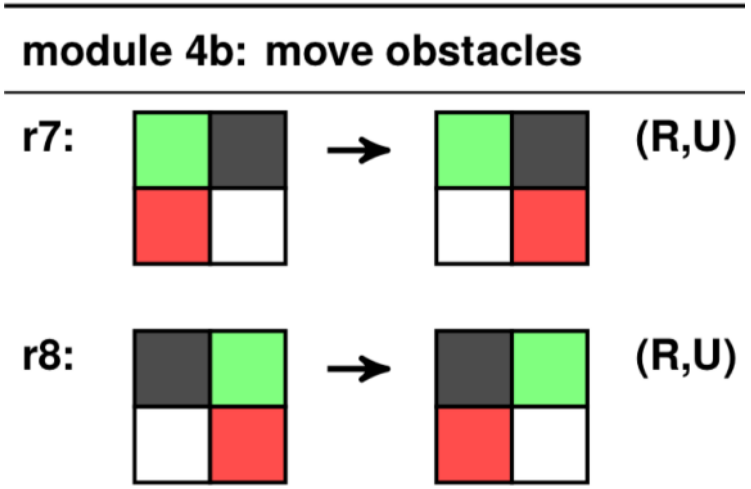
Dungeon room pipeline – problems

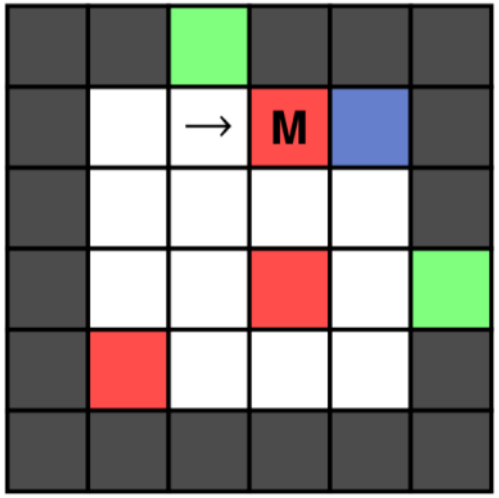
- **Problem:** fire pillars can block access to doors
- **Patch 1**: remove obstacles
  - Rule r6 removes fire pillars blocking doors
- **Result:** fewer traps than intended may reduce the difficulty



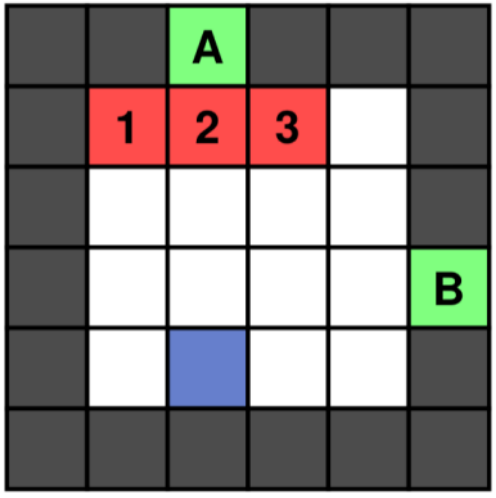
Dungeon room pipeline – problems

- **Patch 2:** move obstacles
  - Rule r7 move a fire pillar blocking a door to the left
  - Rule r8 moves a fire pillar blocking a door to the right

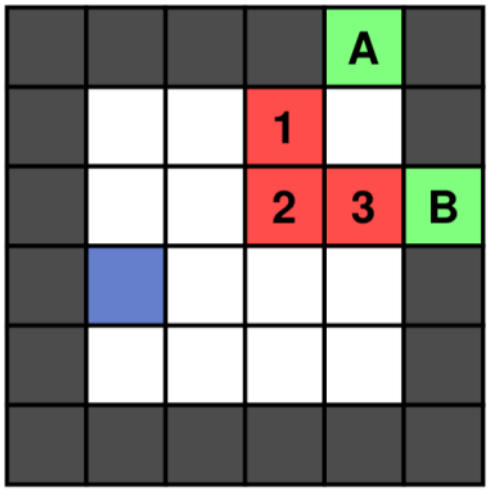




Rule r7 moved pillar M



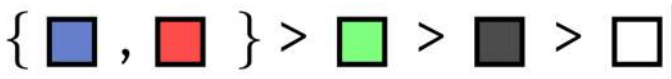
Cannot move pillar 2



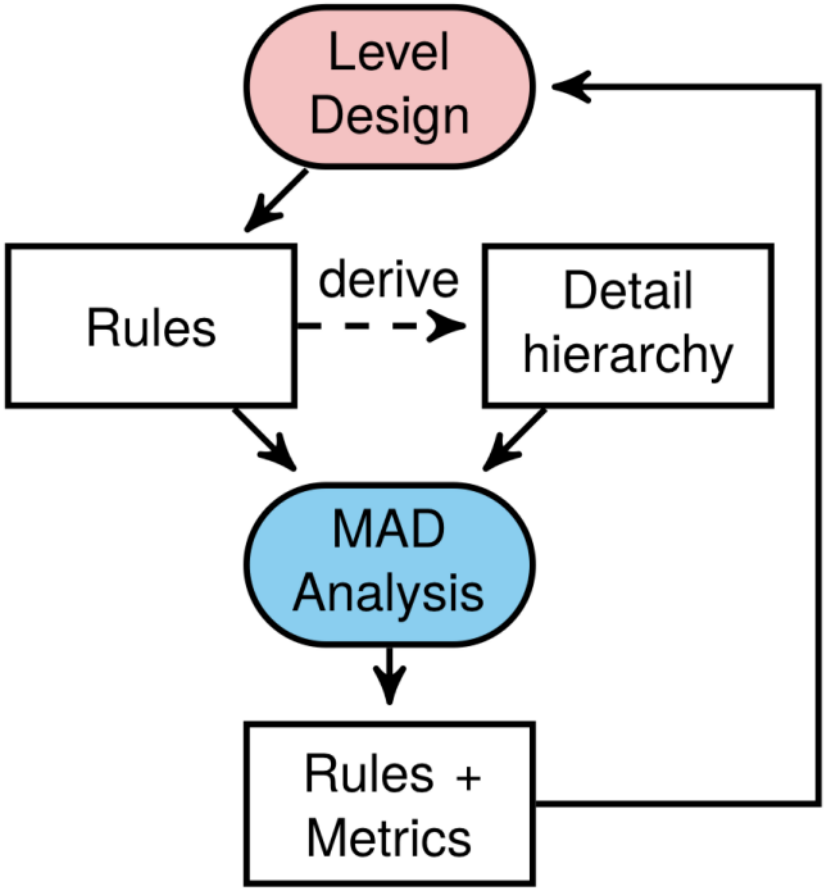
Problems moving pillar 3

MAD Level Design

- Analyzing rule effect with the Metric of Added Detail (MAD)
  - Design a rule and calculate the MAD score with respect to a symbol hierarchy

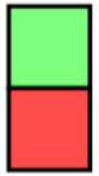
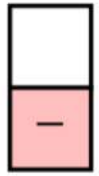


- Calculating the metric
  - Neutral effect (0): cell remains the same
  - Add detail (+1): cell is rewritten to a new symbol
  - Remove detail (-1): cell is rewritten to an old symbol

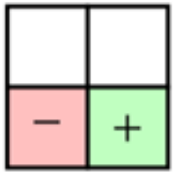
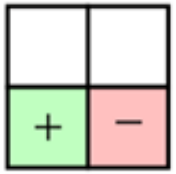


MAD Level Design

Symbol hierarchy { , } > > >

module m4a: remove obstacles				MAD score	heat map
r6:		→		(R,U)	-1 (+0-1) 

Rule r6 has a negative MAD score, since it removes detail.

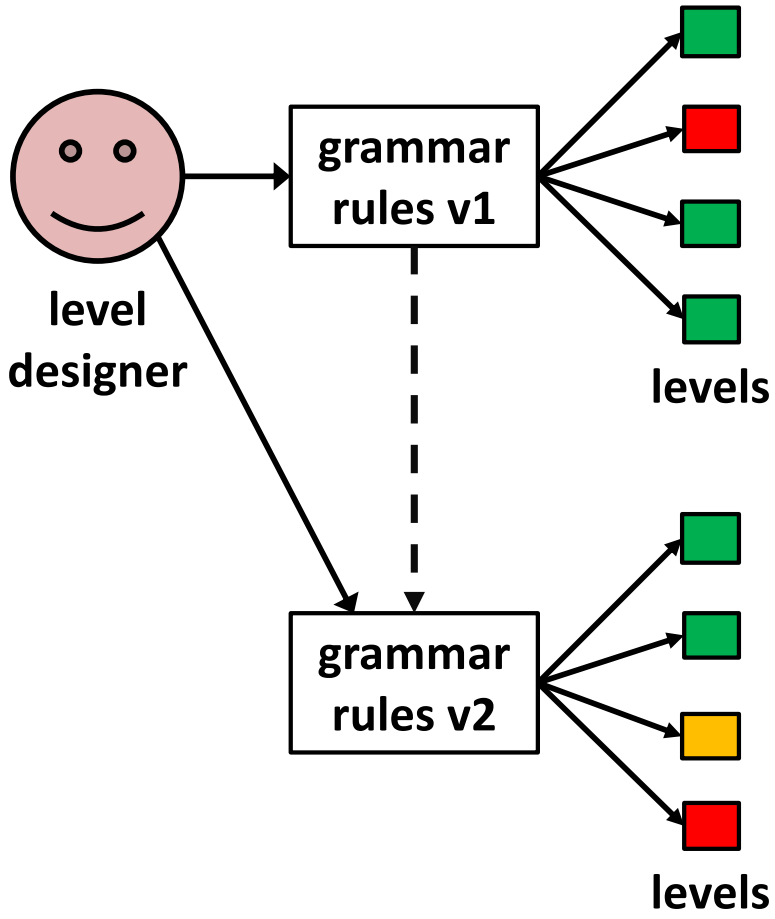
module m4b: move obstacles				MAD score	heat map
r7:		→		(R,U)	0 (+1-1) 
r8:		→		(R,U)	0 (+1-1) 

Rules r7 and r8 have a neutral MAD score, since they preserve detail.

Level Property Language

- **Problems with patching**
  - difficult to get right
  - side-effects
  - do not express level properties

- **Solution: Level Property Language**
  domain-specific language for expressing level properties
  - declarative instead of transformative
  - uses names of tiles and grammar rules
  - add new properties when needed
  - Iterative testing



Level Property Language

- **Amounts.** counts locations of specific tile types and verifies expected amount
  - **1x** water. size {w} == 1: true
  - **3x** pillar. size {a,b,c} == 3: true
- **Adjacency.** filters tile locations
  - **no** pillar **adjacent to** doors
    size {a} == 0: false
- **Topographical inclusion.** filters level generation history using rule names to obtain tile locations
  - **2x** doors **in** walls. collects tiles affected by the walls rule. size {x, y} == 2: true

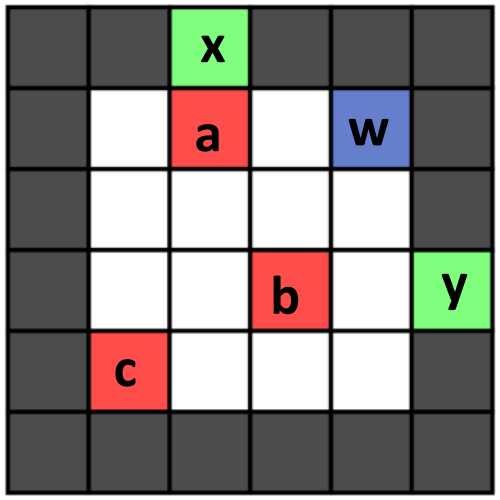
1x water

3x pillar

no pillar **adjacent to** door

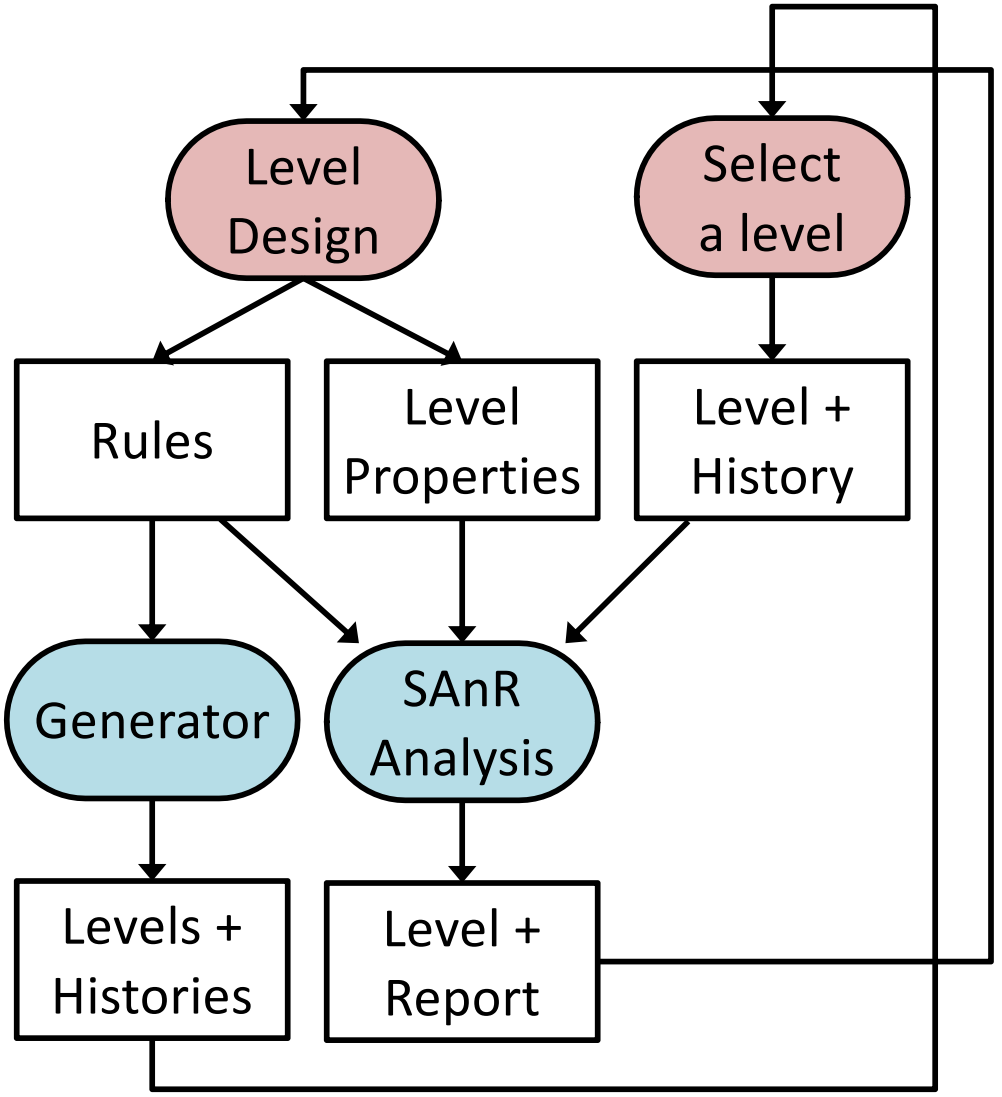
no water **adjacent to** pillar

2x door **in** walls



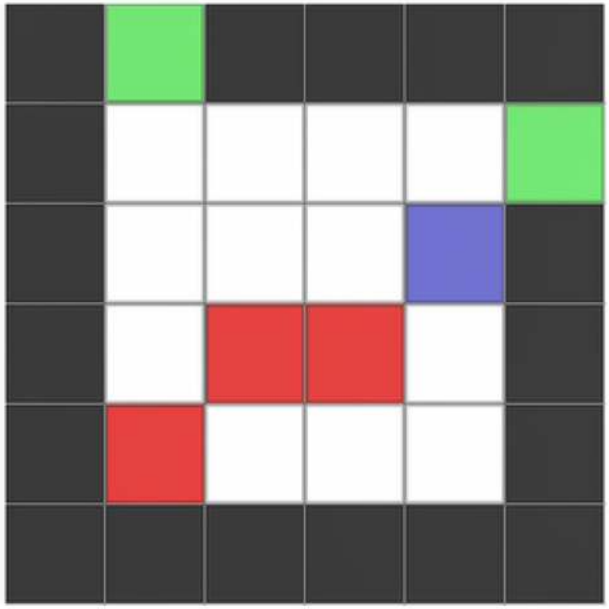
SAnR Level Design

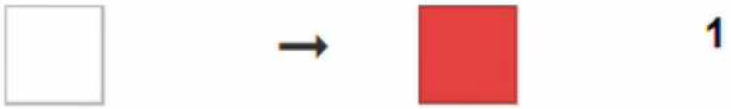
- Mixed-initiative level design using Specification Analysis Reporting (SAnR)
  1. Design grammar rules and level properties
  2. Generate levels
  3. Select a level to debug
  4. Analyze how properties evolved in its level generation history



Ludoscope Lite Project File Execute

#	Module	Rule
0	m1	addWalls
1	m2	addNorthDoo
2	m2	addEastDoor
3	m3	addWater
4	m3	addPillar
5	m3	addPillar
6	m3	addPillar





Properties

- 2x door in addWalls
- 1x water
- 3x pillar
- 0x pillar adjacent to door
- 0x water adjacent to pillar

Ludoscope Lite Project File Execute

Number of executions: 1000

Start analysis

Executions: 1000

Unique results: 988

Broken results: 727

Property	Broken by	Occurences
0x water adjacent to pillar	addPillar	570 (57.000%)
0x pillar adjacent to door	addPillar	318 (31.8000%)

Ludoscope Lite Project File Execute

```
1 version: 0.6f
2 start: TILEMAP 1 1 0:undefined
3 rule: clearDoors(width=2, height=2,
  gt=7) = TILEMAP 2 2 0:wall 1:door
  2:floor 3:pillar > {0 = TILEMAP 2 2
  0:wall 1:door 2:pillar 3:floor}
4
```

Save changes

m4

Executed as grammar

clearDoors

Ludoscope Lite Project File Execute

Number of executions: 1000

Start analysis

Executions: 1000

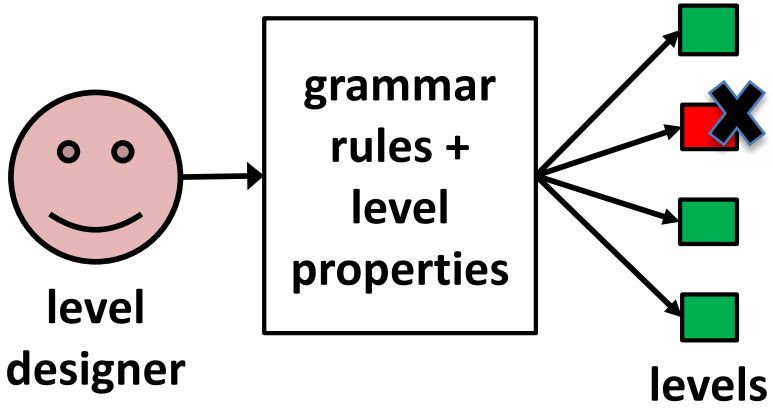
Unique results: 988

Broken results: 585

Property	Broken by	Occurences
0x water adjacent to pillar	addPillar	554 (55.4000%)
0x water adjacent to pillar	clearDoors	22 (2.2000%)
0x pillar adjacent to door	addPillar	38 (3.8000%)
↳ Execution #56		
↳ Execution #82		
↳ Execution #95		
↳ Execution #102		
↳ Execution #158		
↳ Execution #186		
↳ Execution #205		
↳ Execution #220		
↳ Execution #227		

Conclusions

- **Software Evolution perspective**
  - opportunities for tool improvements
  - Two novel techniques for grammar-based level generation, specifically for tile maps



1. **Metric of Added Detail (MAD)**
   - raises flags for rules that remove details (which may be problematic)
2. **Specification Analysis Reporting (SAnR)**
   - adds declarative level properties
   - provides insight into level generation history

- **Evaluation and validation**
  - Case study on Boulder Dash