

Declarative approaches

Procedural Content Generation, Autumn 2012

Julian Togelius

(Thanks to Adam Smith!)

Declarative procedural content generation?

- Designers state their *intent* (what they want) instead of *method* (how to get it)
- Algorithms are used “under the hood” to deliver whatever was asked for
- The user might be agnostic about the algorithms
- Non-specified aspects may vary

This lecture

- Two very different papers...
- Tutenel et al. use declarative world modelling to integrate PCG algorithms and allow multi-level editing
- Smith and Mateas state properties and constraints for game content declaratively, and create content through solving

A declarative approach to procedural modeling of virtual worlds

R. M. Smelik, T. Tutenel, K. J. de Kraker and R. Bidarra

Computers & Graphics 35 (2011) 352–363

Sketchaworld framework

Goals:

- Increase designers' productivity while retaining creative control
- Provide intuitive way of working with PCG algorithms for non-experts
- Provide framework in which to integrate new PCG research

Declarative modelling

- Procedural sketching: “paint” with PCG tools
- Consistency maintenance through a GIS-inspired system of layers
- Edits can be done at any time in any layer, without messing up the other layers more than necessary

Layers

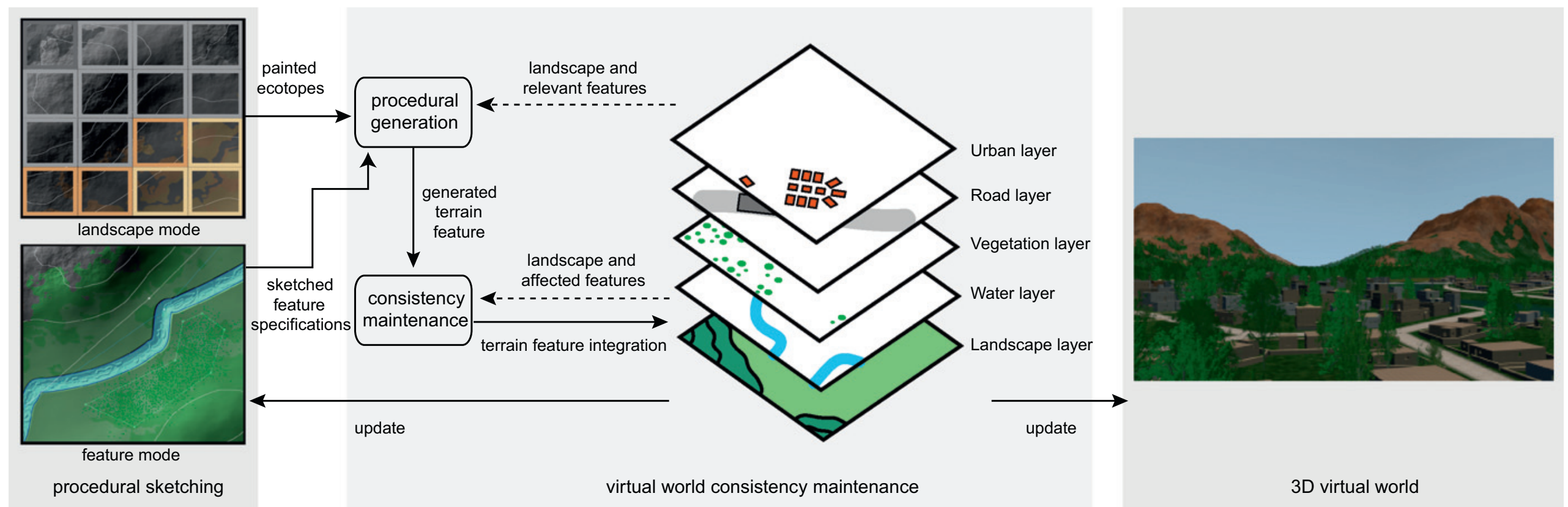


Fig. 1. An overview of the workflow of the declarative modeling approach: using procedural sketching (Section 2), designers interactively create the virtual world, of which each feature is automatically generated, integrated and maintained within the virtual world model (Section 3). From this semantic model of the virtual world, other representations are derived, such as a 3D geometric model.

Layers

- Urban layer
- Road layer
- Vegetation layer
- Water layer
- Landscape layer



Consistency maintenance

- A change in one of the levels will typically affect other levels (e.g. unleashing a river through a city)
- A feature can make a *claim* for an area
 - The claim can be granted or rejected, depending on the priority of the feature
- A feature can request a *landscape*

Feature interaction

- Two features claim the same area...
- If possible, this solved with a *connection* (e.g. bridge, tunnel, road junction)
- Otherwise, the lower-priority feature loses and will have to remodel the part

// solve all interactions between a feature and existing features
// f_x -feature which has made a new claim
// a_x -area of terrain claimed by f_x
// l -level of abstraction

SolveFeatureInteractions(feature f_x , area a_x , level l):

// find all interacting features with granted claims

$F = \{f_y | f_y \in \text{features}, a_x \cap f_y.\text{area} \neq \emptyset\}$

sort F according to highest $\text{priority}_{\text{claim}}(f_y, l)$

// handle all feature interactions

for all feature f_y in F **do**

if $\text{priority}_{\text{claim}}(f_x, l) > \text{priority}_{\text{claim}}(f_y, l)$ **then**

SolveInteraction($f_y, f_x, a_x \cap f_y.\text{area}, l$)

else

SolveInteraction($f_x, f_y, a_x \cap f_y.\text{area}, l$)

end if

end for

// solve an interaction between a pair of terrain features

// f_{lose} -feature for which the claim is rejected

// f_{win} -feature for which the claim is granted

// a -disputed area

// l - level of abstraction

SolveInteraction(feature f_{lose} , feature f_{win} , area a , level l):

// determine whether connection can and should be formed

if connectionDefined($f_{\text{lose}}.\text{type}, f_{\text{win}}.\text{type}$)

$\text{priority}_{\text{connect}}(f_{\text{lose}}, f_{\text{win}}, a, l) > \text{priority}_{\text{conflict}}(f_{\text{lose}}, a, l)$ **then**

// interaction is solved with a connection

$f_{\text{lose}}.\text{connectTo}(f_{\text{win}}, a, l)$

else

// interaction is solved by restructuring the losing feature

$f_{\text{lose}}.\text{restructure}(a, l)$

end if

Landscape layer

- Grid of painted ecotopes is turned into elevation and roughness
- Basically, randomization and interpolation is added to the coarse painted grid

Water layer (rivers)

- Rivers flow from higher to lower elevation
- A sequence of control points are created...
- for each new point, a number of points in a circle around it are scored according to equations (next slide)
- The best point is chosen as next

Vegetation layer

- Type of vegetation at each spot depends on:
 - elevation profile, soil type (different species have different preferences)
 - simulation of competition for resources (space, water)

Road layer

- Similar to the river creation: a sequence of control points is created, and a spline along these points define the road
- Prefers no change in elevation (unlike the rivers!)

Urban layer

- Recursive subdivision approach (clusters, districts, blocks, buildings)
- Land use model determines repulsion and attraction between different city features
- Different land use models for different types of cities (mercantile, feudal...)

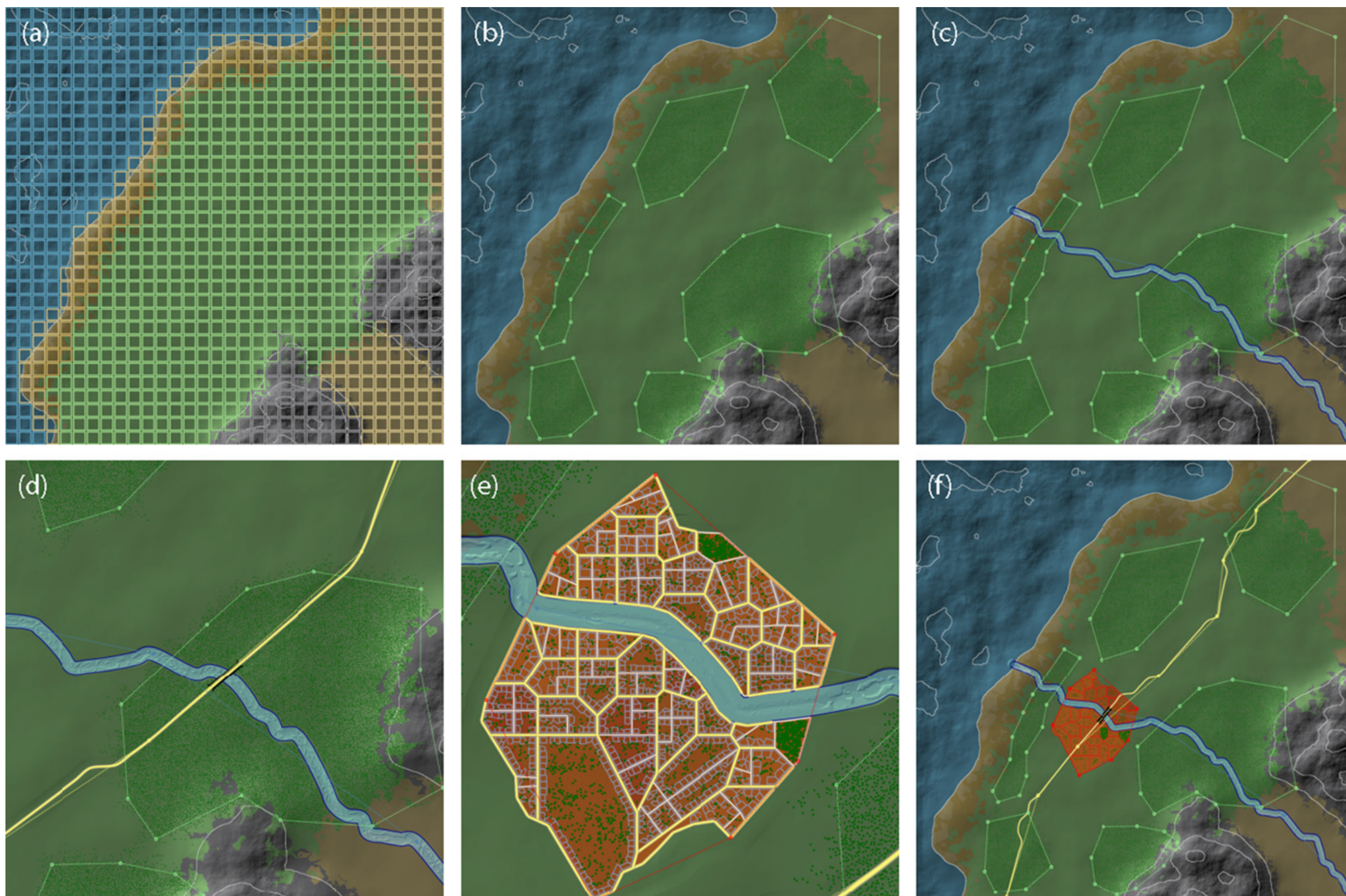


Fig. 4. A procedural sketching session: (a) basic landscape, defined by brushing ecotopes, (b) several forest features, (c) river flowing towards the sea, (d) road feature crossing this river, (e) city created along the river and (f) road rerouted to run through the city.

A taxonomy of PCG

- Online/Offline
- Necessary/Optional
- Random seeds/Parameter vectors
- Stochastic/Deterministic
- Constructive/Generate-and-test

Answer Set Programming for Procedural Content Generation: A Design Space Approach

Adam M. Smith and Michael Mateas

IEEE TCIAIG, 2011

Answer set programming

- 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

AnsProlog

teleportation_disabled.

initial_health(100).

weather_model(springtime).

allies(humans,elves).

damage(sword_of_might,11).

scripted_event(spawn(boss,temple),120).

AnsProlog

```
valid_move(rock),  
valid_move(paper),  
valid_move(scissors),  
valid_move(lizard),  
valid_move(spock).
```

AnsProlog

plateau_at(X) :-

height($X-1, H$), height(X, H), height($X+1, H$).

hostile(A, B) :- enemy(A, B).

hostile(A, C) :- enemy(A, B), friend(B, C).

hostile(A, C) :- friend(A, B), enemy(B, C).

Answer set programming: choice rules

{ rain, sprinkler }.

wet :- rain.

wet :- sprinkler.

dry :- **not** wet.

Answer set programming: answer set

dry.

wet, rain.

wet, sprinkler.

wet, rain, sprinkler.

Answer set programming: integrity constraints

- Cannot be true (“implies contradiction”)

`:- sprinkler, rain.`

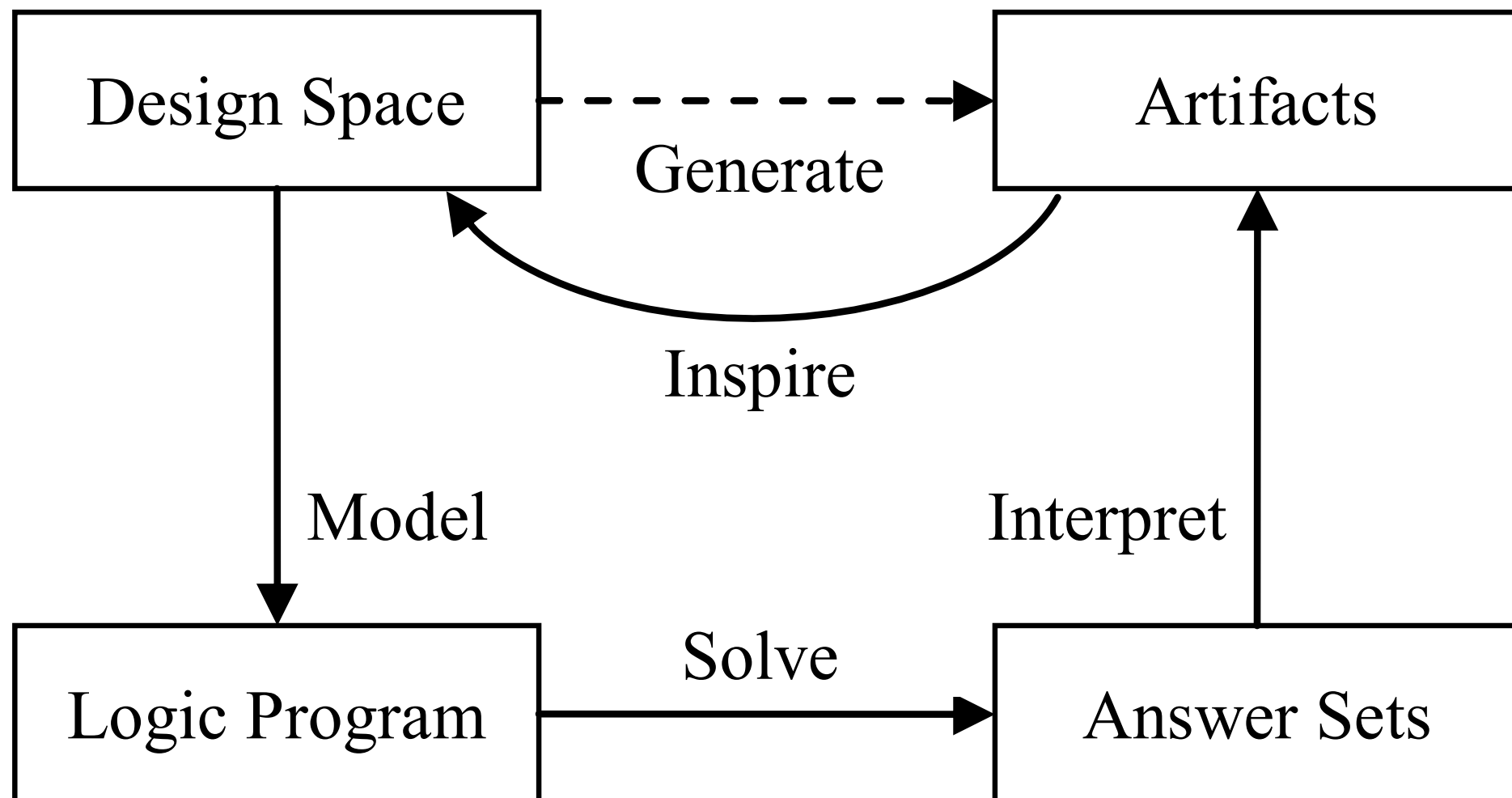
ASP solvers

- Generates answer sets from ASP programs
- Can be treated as black-box
- Usually do not generate complete candidates, but proceed by excluding whole regions of search space that violate constraints

ASP for PCG

- Crucial properties of game content are represented in ASP form
- Designers ask questions that embody their design intent (“which levels have 15 rooms, no green mushrooms and can be cleared without wall-jumping?”)
- An ASP solver delivers an answer set, where each answer is interpreted as a game content artifact

ASP for PCG



ASP for PCG

Crucial steps:

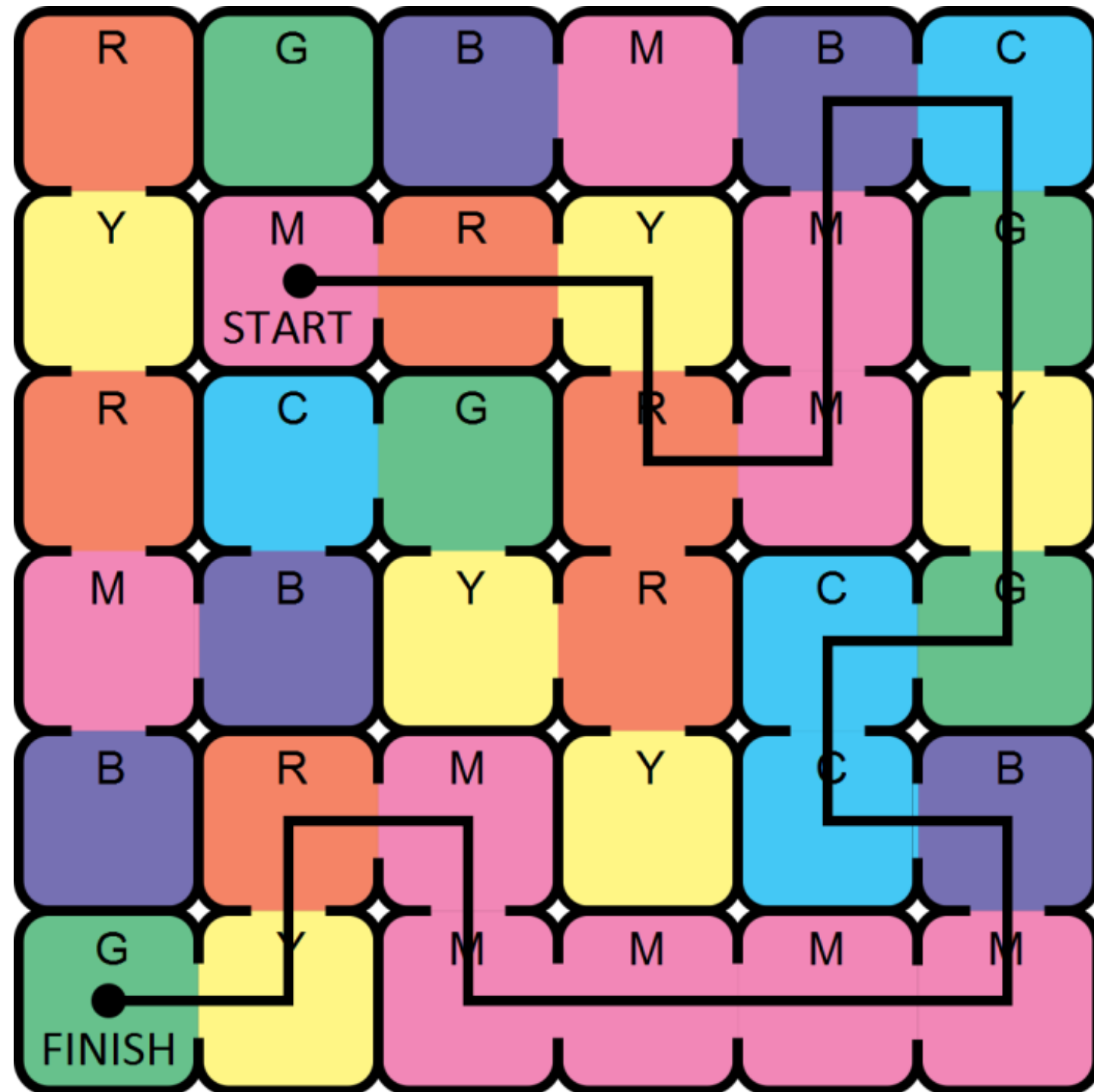
- Modelling the content domain as ASP
- Creating a mechanism that interprets collections of ASP statements as content
- Designing designer-relevant ASP questions

An example:

Chromatic mazes

- Colour wheel: red-yellow-green-cyan-blue-magenta
- No *explicit* walls
- Movement is permitted between cells of the same or adjacent colour

Chromatic maze



Chromatic maze

- There are six colours.
- The maze has dimensions six by six.
- There is exactly one cell at each $\{x, y\}$ position, and it has one of the six colours.
- There is exactly one start position and one finish position.

Chromatic maze generator

cell(Color, X, Y), start(X, Y), finish(X, Y).

color(red; yellow; green; cyan; blue; magenta).

dim(1..6).

| { cell(C, X, Y) : color(C) } :- dim(X), dim(Y).

| { start(X, Y) : dim(X) : dim(Y) } |.

| { finish(X, Y) : dim(X) : dim(Y) } |.

Chromatic maze generator

- Actually a complete generator!
- Except that it might generate unsolvable mazes...

Solution

\therefore not victory.

Solution: solvability constraint

player_at(0,X,Y) :- start(X,Y).

player_at(T,X,Y) :-
 player_at(T-1,SX,SY),
 passable(SX,SY,X,Y),
 0 {player_at(0..T-1,X,Y)} 0.

victory_at(T) :- player_at(T,X,Y), finish(X,Y).

victory :- victory_at(T).

Asking for mazes with long shortest paths

`:- victory_at(T), T < 22.`

Diorama



Diorama

- Map generator for open-sourced RTS Warzone 2100
- Maps: heightmaps + cliffs + features
- Predicates available for e.g. raised bases, defendable oil wells, smooth plains etc.

Runtime of ASP solvers

- In general, the problem of finding an answer set is NP-complete
 - Yes, that means exponential time
- But in practice, it seems to work very fast
- So far...

Other ASP issues

- Combining with optimisation - how can we get “good enough” content (anytime) even if optimal content is not available?
- Diversity of generated content
- Multiple objectives
- How can we create adaptive games (based on player modelling) using

A taxonomy of PCG

- Online/Offline
- Necessary/Optional
- Random seeds/Parameter vectors
- Stochastic/Deterministic
- Constructive/Generate-and-test

Today's lab:
Answer set
programming