M Systems – simulator architecture and practical examples

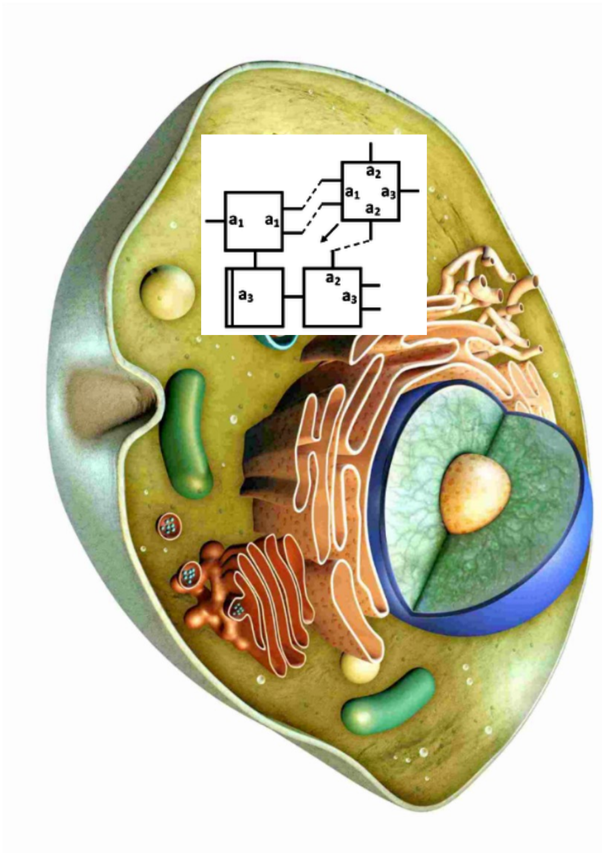
Petr Sosík[1], Vladimír Smolka[1], Jan Drastík[1],Jaroslav Bradík[1], Max Garzon[2]

[1] INSTITUTE OF COMPUTER SCIENCE, FACULTY OF PHILOSOPHY AND SCIENCE, SILESIAN UNIVERSITY IN OPAVA, CZECH REP.

[2] THE UNIVERSITY OF MEMPHIS, TENNESSEE, USA

Motivation

- Create a computational model with focus at basic morphogenetic phenomena such as:
  - Growth
  - Homeostasis
  - Self-reproduction
  - Self-healing
- Simulate morphogenesis from scratch
  - Not to use atomic assembly units (cells)
  - Start from 1D/2D/3D primitives
  - Use self-assembly feature to create 3D cell-like forms



M systems – formal definition

Morphogenetic systems (M systems)

- Based on principles of common membrane computing models, especially with proteins on membranes
- Live in a 3D space (generally *d*D)
- Introduce explicit geometric features and self-assembly capabilities
  - Each elementary object has a fixed shape and position in space at any point in time
- Exhibit emergent behavior from local interactions
- Informed by tile assembly models
  - Polytopes and connectors like tiles and glues
- Use 3 types of objects:
  - Floating objects
  - Tiles
  - Protions (abtraction of biological "proteins")

Basic M system objects

- Floating objects
  - Small shapeless atomic objects floating freely within the environment
  - With a nonzero volume and specific position
- Tiles
  - Have their predefined size and shape (convex bounded polytopes)
  - Can stick together along their edges or at selected points called *connectors*
  - Can self-assembly into interconnected structures
- Protions
  - Are placed on tiles
  - Catalyze reactions of floating objects
  - Serve as „protion channels“ through (d–1)D tiles

```xml
<floatingObjects>
  <floatingObject name="a">
    <shape value="sphere"/>
    <size value="0.05"/>
    <color name="Lime" alpha ="255"/>
    <mobility value="15"/>
    <concentration value="0.1"/>
  </floatingObject>
```

```xml
<tile name="q0">
  <polygon>...</polygon>
  <positions>...</positions>
  <connectingAngle value="2.034443935795703" unit="rad"/>
  <connectors>...</connectors>
  <surfaceGlue name="gx"/>
  <color name="DeepSkyBlue" alpha="64"/>
</tile>
```

```xml
<proteins>
  <protein name="p0"/>
  <protein name="p1"/>
  <protein name="p2"/>
  <protein name="p3"/>
  <protein name="p4"/>
</proteins>
```

Formal definition

- Morphogenetic system $\boldsymbol{M} = (F, P, T, \mu, R, \sigma)$
  - $F = (O, m, \rho, \varepsilon)$, the catalog of floating objects
    - O – the set of floating objects
    - m – mean mobility of each floating object
    - ρ – radius of interaction of each floating object
    - ε – concentration of each object in the environment
  - P – is the set of protions
  - T – is a polytopic tile system
  - μ – maps proteins to positions on M-tiles
  - R – is a set of reaction rules
  - σ – maps glue pairs to a multiset of floating objects produced when the binding is established

Formal definition

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <tiling>
    <tiles>...</tiles>
    <glues>...</glues>
    <glueRelations>...</glueRelations>
    <initialObjects>...</initialObjects>
    <glueRadius value="0.1"/>
  </tiling>
  <Msystem>
    <floatingObjects>...</floatingObjects>
    <proteins>...</proteins>
    <proteinsOnTiles>...</proteinsOnTiles>
    <evoRulesWithPriority>...</evoRulesWithPriority>
    <signalObjects>...</signalObjects>
    <reactionRadius value="14"/>
  </Msystem>
</root>
```

Reaction rules

- Are used for reactions and modifications of the M system during growth
- Four types of reaction rules:
  - Metabolic rules
  - Creation rules
  - Destruction rules
  - Division rules

```
<evoRulesWithPriority>
  <evoRule type="Metabolic">...</evoRule>
  <evoRule type="Metabolic">...</evoRule>
  <evoRule type="Metabolic">...</evoRule>
  <evoRule type="Create">...</evoRule>
  <evoRule type="Destroy">...</evoRule>
  <evoRule type="Create">...</evoRule>
  <evoRule type="Create" priority="1">...</evoRule>
  <evoRule type="Create" priority="1">...</evoRule>
  <evoRule type="Create">...</evoRule>
  <evoRule type="Create">...</evoRule>
  <evoRule type="Create">...</evoRule>
  <evoRule type="Divide">...</evoRule>
  <evoRule type="Divide">...</evoRule>
  <evoRule type="Divide">...</evoRule>
  <evoRule type="Divide">...</evoRule>
  <evoRule type="Metabolic">...</evoRule>
  <evoRule type="Metabolic">...</evoRule>
  <evoRule type="Metabolic">...</evoRule>
  <evoRule type="Metabolic">...</evoRule>
</evoRulesWithPriority>
```

Metabolic rules

- A multiset of floating objects reacts and changes, or it is transported through a (*d*–1)D tile
- (*d*–1)-dimensional tiles have their sides marked “in” and “out”, by convention

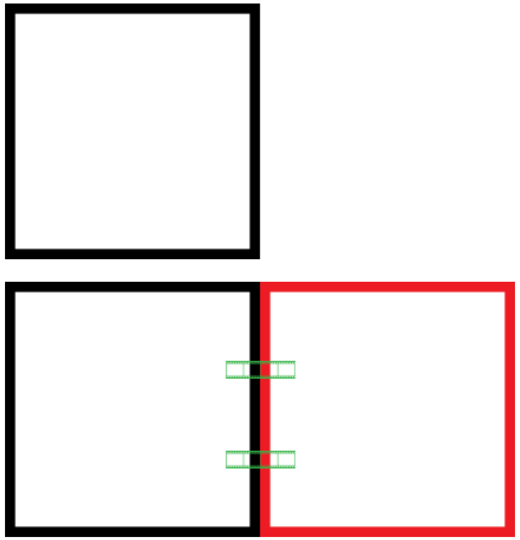
TYPE	RULE	EFFECT
SIMPLE	u → v	objects in multiset *u* react to produce *v*
CATALYTIC	pu → pv u[p → v[p [pu → [pv	objects in *u* react in presence of *p* to produce *v*; this variant requires both *u*, *v* at the side “out” of the tile; this variant requires both *u*, *v* at the side “in” of the tile;
SYMPORT	u[p → [pu [pu → u[p	passing objects in *u* through protion channel *p* to the other side of the tile
ANTIPORT	u[pv → v[pu	interchange of *u* and *v* through protion channel *p*

```
<evoRule type="Metabolic">
  <leftside value ="a,p0"/>
  <rightside value= "p0,a"/>
</evoRule>
```

Creation rules

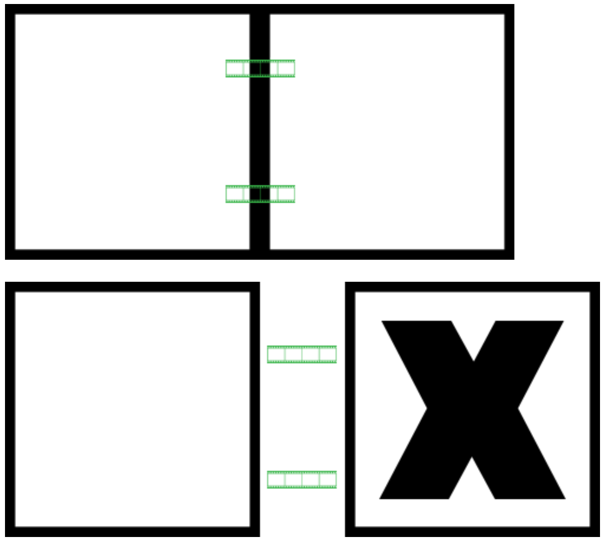
- Creates tile *t* while consuming the floating object in *u*
- Rule format: *u* → *v*

```xml
<evoRule type="Create">
  <leftside value="a,a,a,a,a,a,a,a"/>
  <rightside value="q2"/>
</evoRule>
```



Destruction rules

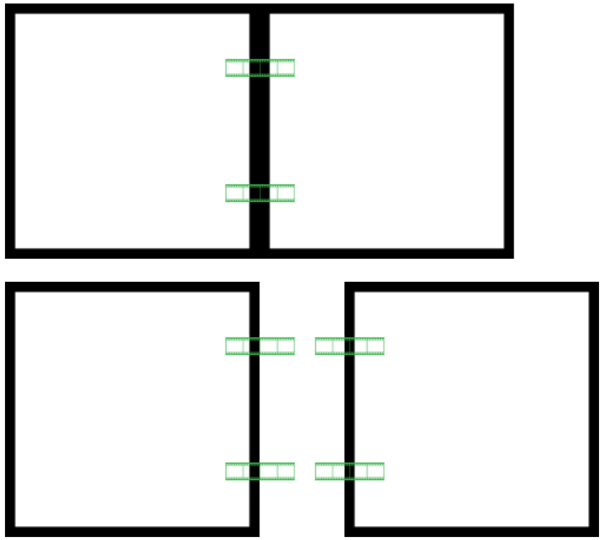
- Tile *t* is destroyed in the presence of multiset of floating objects *u* which is consumed
- All connections from t to other tiles are released
- Rule format: $ut \rightarrow v$



```
<evoRule type="Destroy">
  <leftside value="a,q1"/>
  <rightside value="b"/>
</evoRule>
```

Division rules

- Two connectors with glues g, h get disconnected and the multiset x of floating objects is consumed
- Rule format: $g \overset{x}{\text{——}} h \rightarrow g, h$



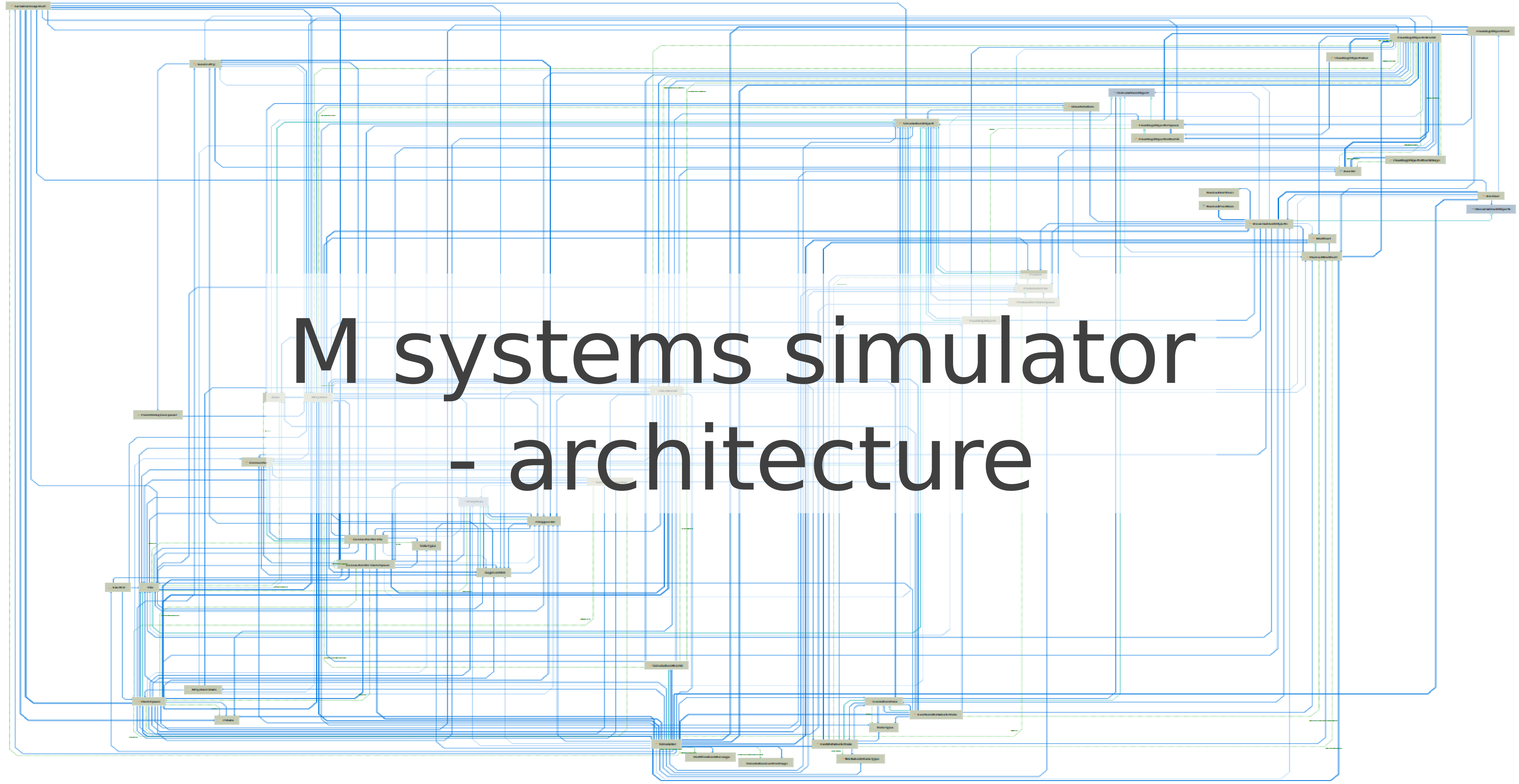
```
<evoRule type="Divide">
  <leftside value="gdd,x,gdd"/>
  <rightside value="gdd,gdd"/>
</evoRule>
```

```
public void RunSimulation(object numberOfSteps)
{
    try
    {
        ulong stepsNumber = TypeUtil.Cast<ulong>(numberOfSteps);
        // stepsNumber = 0 => unlimited number of simulation steps.
```

M system computation

- Initial configuration contains only *seed tiles* in *S* and random distribution of floating objects with concentration ε
- Computation takes place in discrete steps
- During each step, rules from *R* are applied in maximally parallel manner
  - Applicable rules are chosen randomly until no further rule is applicable
- Rules are applied in parallel to the actual configuration
- Each floating object changes its position randomly within its mobility perimeter
- A sequence of transitions between configurations is called a computation (nondeterministic)
- A computation ends when there is no longer any applicable rule

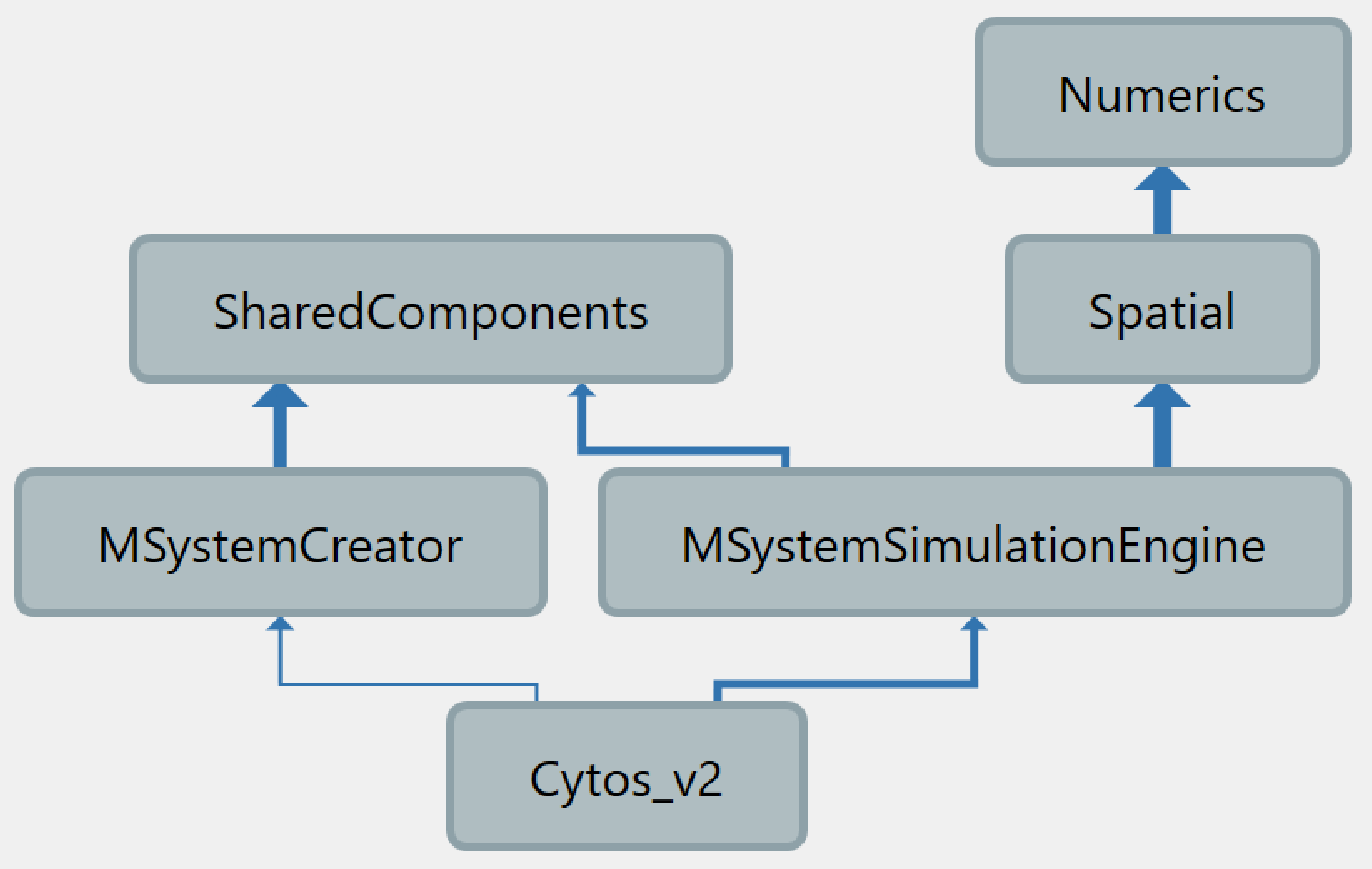
M systems simulator - architecture



M systems simulator - architecture

- Modular architecture with strong OOP approach
- Separated simulation engine (standalone DLL) with user friendy API
- M system is defined as XML file (available validation agains our XSD)
  - XML is also possible to create using our M System creator tool
- Using Unity Game Engine for visualization of simulated data

M systems simulator – dependency tree



M systems simulator – modules

- Cytos_v2
  - Main application for running simulator
  - Contains only UI all computation is done by MSystemSimulationEngine
- MSystemSimulationEngine
  - Main ENGINE of whole project
  - Contains deserialization tool (usage is optional)
  - Simulation methods for simulation run
  - Serialize output data into XML structure called snapshot file
    - Contains objects (created/destroyed/moved) with positions and color informations

M systems simulator – modules

- MSystemCreator
  - Easy to use XML creator and editor
  - Creates basic XML structures
  - Contains validation features and help
- SharedComponents
  - Basic library which contains useful and shareable tools across solution
- Spatial and Numerics
  - Open source mathematical libraries (user for eg. object shifts)
- MSystemVisualization
  - Unity game engine used for visualization

M systems simulator – input/output files

- Input
  - Created M system is saved to M system description XML
    - Structure is defined in XSD
  - Deserialized description constructs all expected simulation objects
  - These objects are used for simulation
- Output
  - So-called Snapshot file
    - Also in XML structure
  - Contains visualization objects with its shape, in specified position and with defined color
  - Can be used as input file for MSystemVisualization (Unity engine)

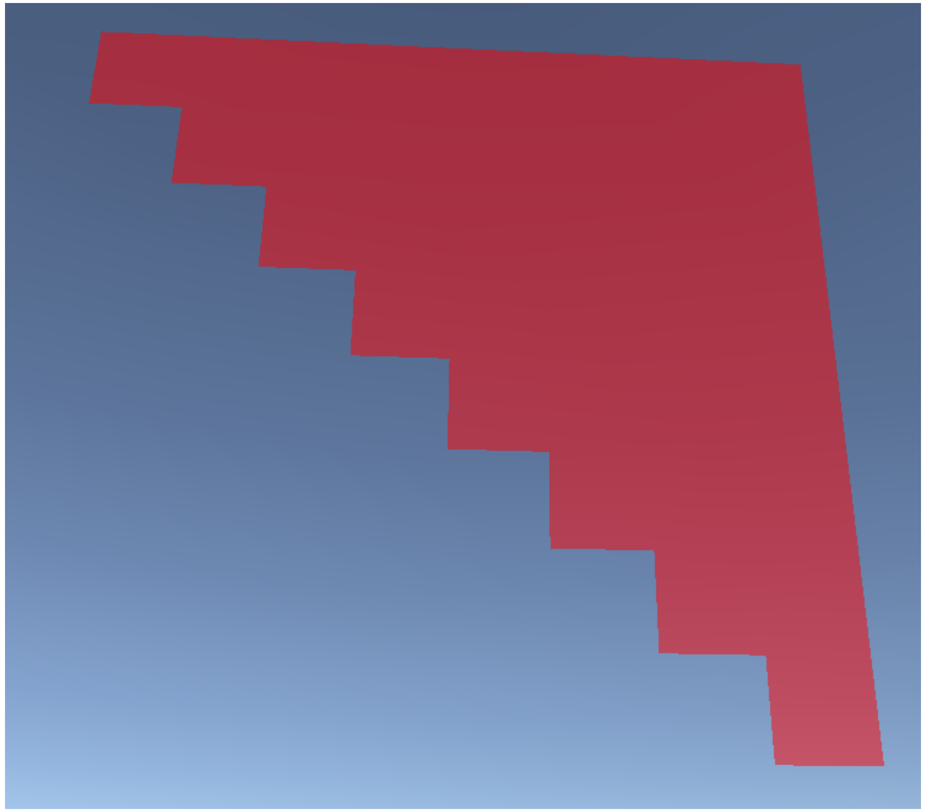
M systems simulator – technical details

- Written in .NET 4.5.2
- Using TFS as version control
- Strong OOP approach
- Development is under SCRUM principles
- Using unit and integration testing across whole solution

M systems simulator - examples

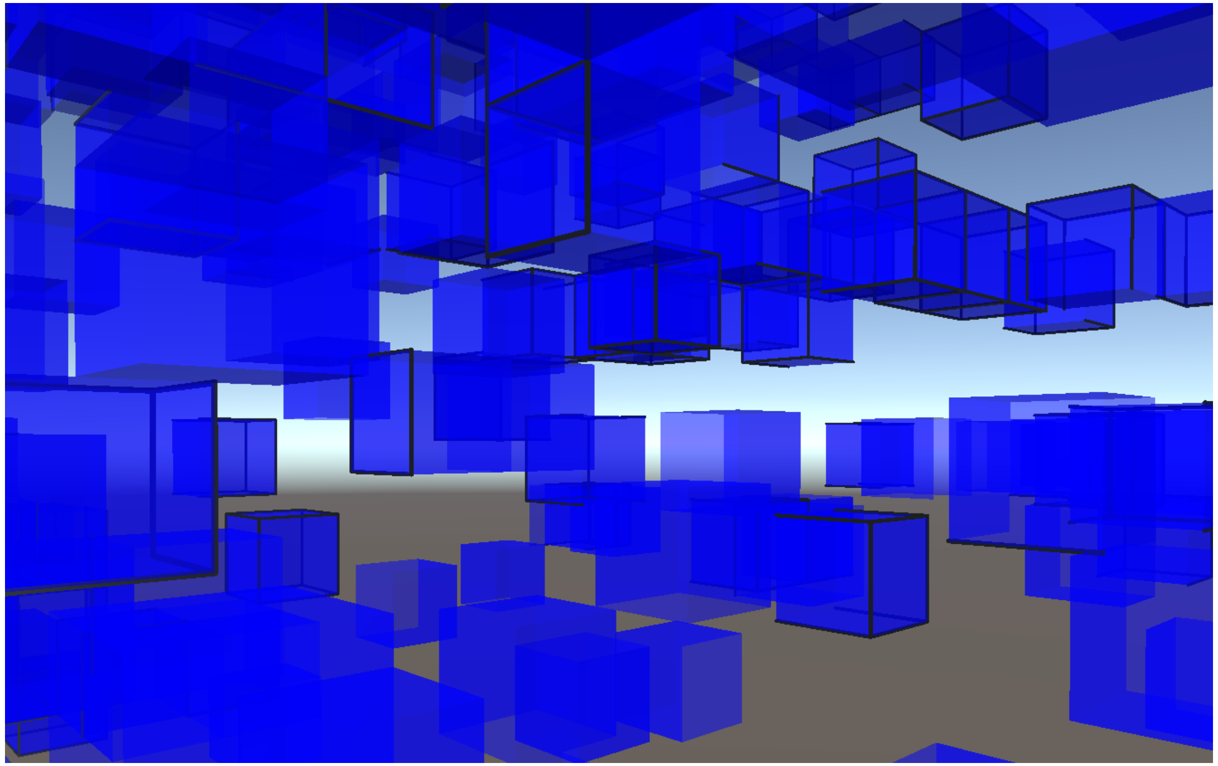
2D square tiling

- Basic square tile (d)
  - 4 sides
  - 4 connectors
    - c1: v1 – v2, glue g1
    - c2: v2 – v3, glue g2
    - c3: v3 – v4, glue g3
    - c4: v4 – v1, glue g4
  - Angle 180°
  - Glue relations: g3 – g1, g4 – g2
  - Contains high concentration of floating objects „a“
  - Initial object (tile): d
  - Only one rule: Create (a->d)



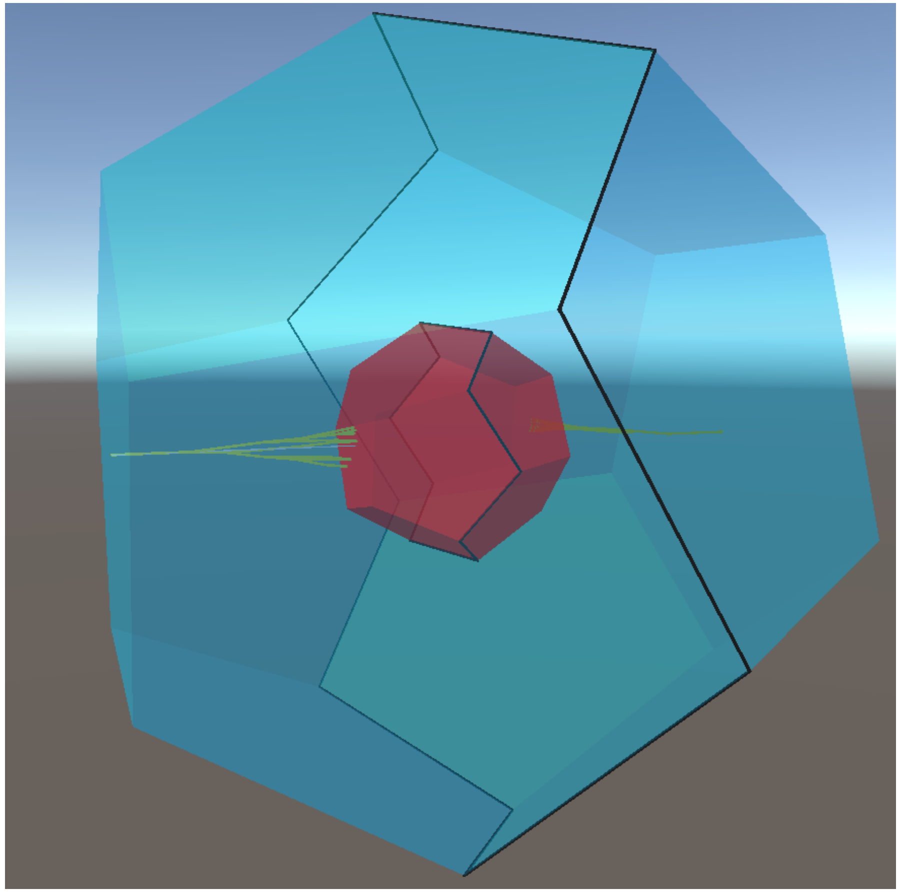
Boxy hallows

- Harry Potter and the Deahtly Hallows part 2 - inside Belatrix's vault
  - Duplication spell
- Basic square tile (d)
  - 4 sides
  - 4 connectors
    - c1: v1 – v2, glue g1
    - c2: v2 – v3, glue g1
    - c3: v3 – v4, glue g1
    - c4: v4 – v1, glue g1
  - Angle 90°
  - Glue relations: g1 – g1
  - Contains high concentration of floating objects „a“
  - Initial object (tile): d
  - Rules: Create (a->d), Divide (g1 g1 $\xrightarrow{a}$ g1, g1)



Cell growth

- Large system of interconnected tiles
- Contains large pentagonal tiles (cytoskeleton), small pentagonal tiles (core) and interconnecting rods
- Cell growth behaviour is simulated by Metabolic/Division/Creation rules
- Initial object is one large pentagonal tile



Circuit Tile Assembly Models (cTAM)

Circuit Theory Background

- Voltage is the eletric potential energy between two nodes in a circuit
- The ground node in the circuit has a potential of zero
- Voltage relative to ground is associated with each node in the circuit
- It is analogous to pressure in a water pipe
  - pressure causes water to flow
  - voltage causes current to flow
- Voltage is necessary to move a positive charge against an eletric field
- Voltage gives rise to current in a circuit
- Electrical resistance is analogous to the resistance produced by different diameter pipes in a water system
- Resistors are units of dielectric material that impede current flow

Circuit Tile Assembly Models (cTAM)

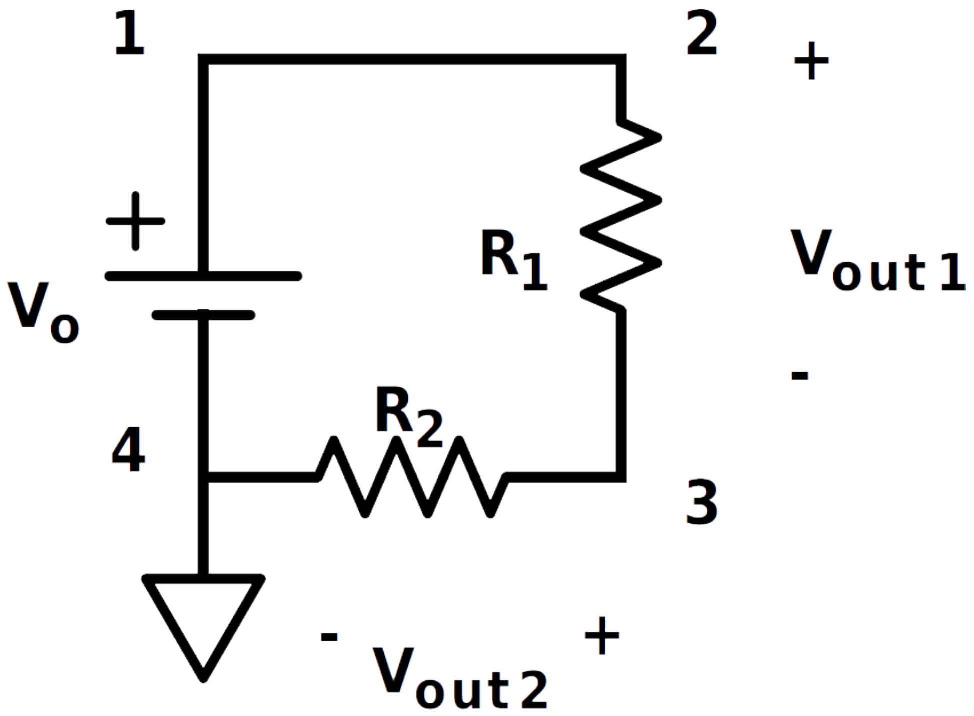
- new self-assembly analog model
- motivated by Tile Assembly Models (aTAM)
- DC resistive circuits self-assemble under voltage control
- This model exhibits both self-assembly and self-control
- attachments are controlled by local voltage differences as the assembly grows
- resource can be thought of as a source of energy for growth reaction

Deaton R., Yasmin R., Moore T., Garzon M. (2017) Self-assembled DC Resistive Circuits with Self-controlled Voltage-Based Growth. In: Patitz M., Stannett M. (eds) Unconventional Computation and Natural Computation. UCNC 2017. Lecture Notes in Computer Science, vol 10240. Springer, Cham

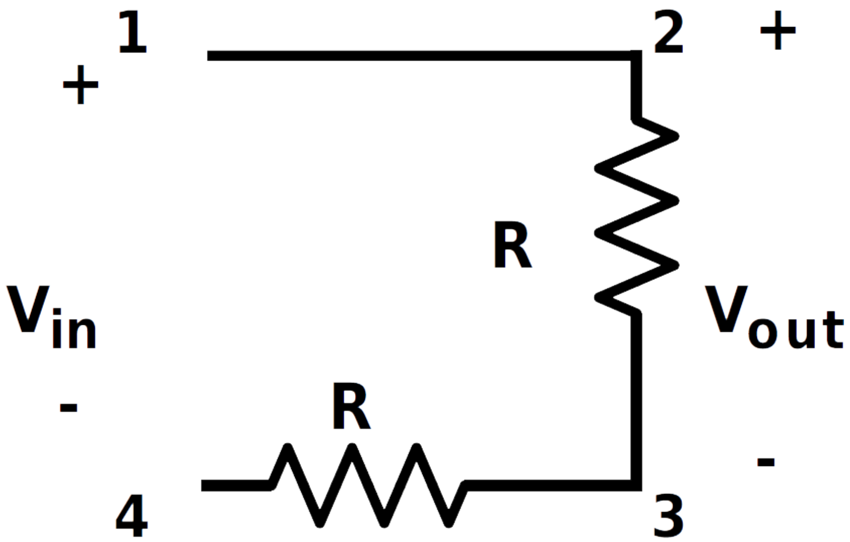
cTAM – formal definition

- A circuit tile assembly system is a tuple $C = (\Gamma, S, \tau, v, \zeta)$
  - $\Gamma - finite\ set\ of\ circuit\ tile\ types$
  - $S - S \subseteq \Gamma\ set\ of\ seed\ circuit\ tiles\ that\ includes\ a\ source\ and\ ground$
  - $\tau - \tau \in \mathbb{R}\ is\ the\ treshold\ voltage\ for\ attachment$
  - $v - N \longrightarrow \mathbb{R}\ is\ the\ eletric\ potential\ energy\ at\ a\ node\ relative\ to\ ground$
  - $\zeta - N_{in} \longrightarrow N_{out}\ maps\ input\ nodes\ to\ output\ nodes$

Circuit Tile Assembly Models (cTAM)



1. Seed circuit tile for all assemblies

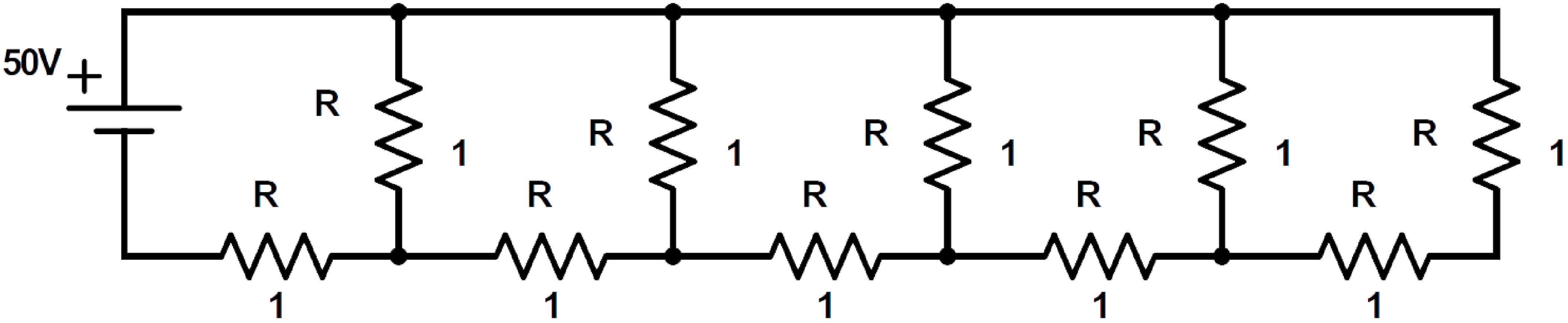


2. Circuit tile for the ladder circuit.

Ladder circuit

- $R^{eq}(m) = \frac{R(R+R^{eq}(m-1))}{2R+R^{eq}(m-1))}$
  - $R^{eq}(0) = R$
  - The circuit has in series with R.
  - The circuit has $R^{eq}(m-1)$ in series with R.
  - This series combination is then paraller with R resulting in
  - This series combination is then paraller with R resulting in $R^{eq}(m)$
  - Solving the quadratic equation and knowing that resistence cannot be negative
  - Solving the quadratic equation and knowing that resistence cannot be negative
- $V(i) = \frac{R^{eq}}{R+R^{eq}} V(i-1)$
  - Recurrent relationshit calculate voltage at the cTAM
  - Recurrent relationshit calculate voltage at the cTAM
  - Connecting new tiles as long as the output voltage exceeds the treshold limit
  - Connecting new tiles as long as the output voltage exceeds the treshold limit
  - $V(n) = (\frac{1}{\phi+1})^n V_0 < \tau$

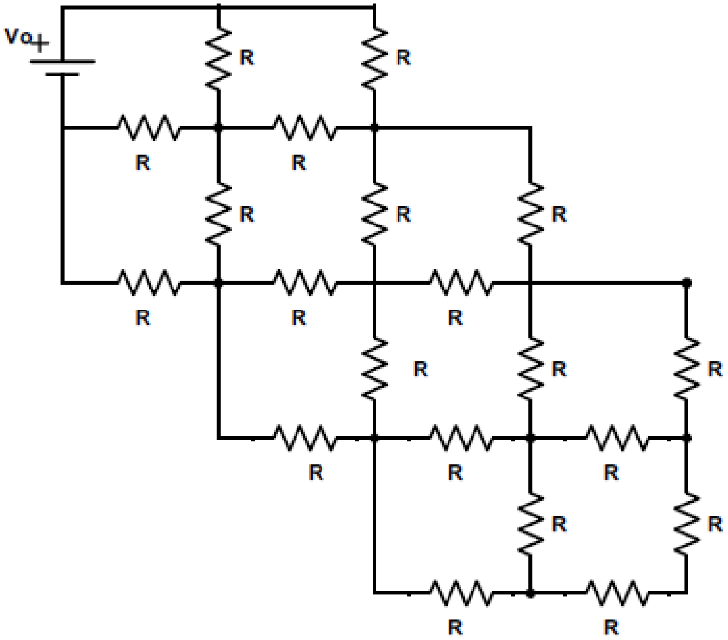
Ladder circuit



3. Ladder circuit assembly with V_0 = 50 and τ = 1. The maximum size is n = 5

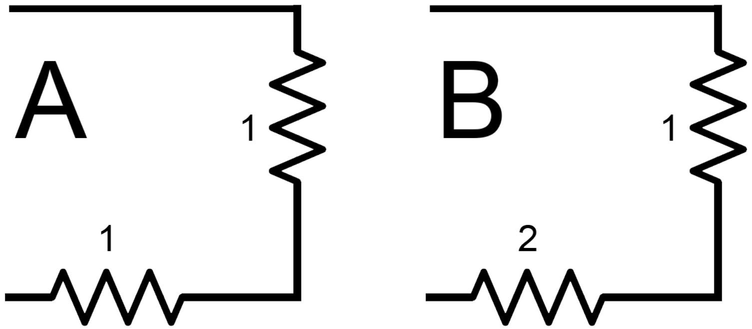
cTAM and M system

- M system simulator is extended with electric cTAM mode
- cTAM can be seen as a special case of M system with dynamical glues
- Now 1D simulate only. 2D simulate in the Spring 2019

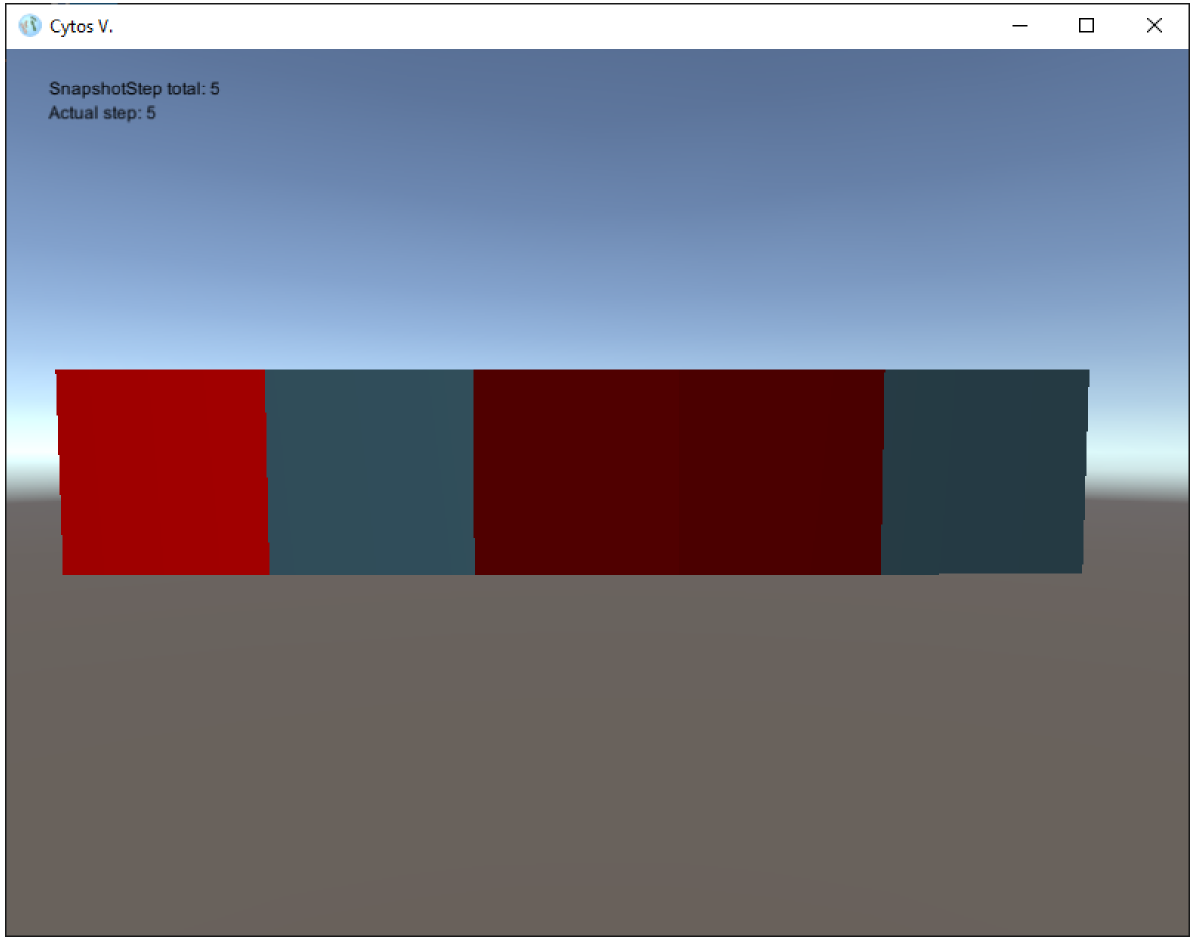


cTAM computation

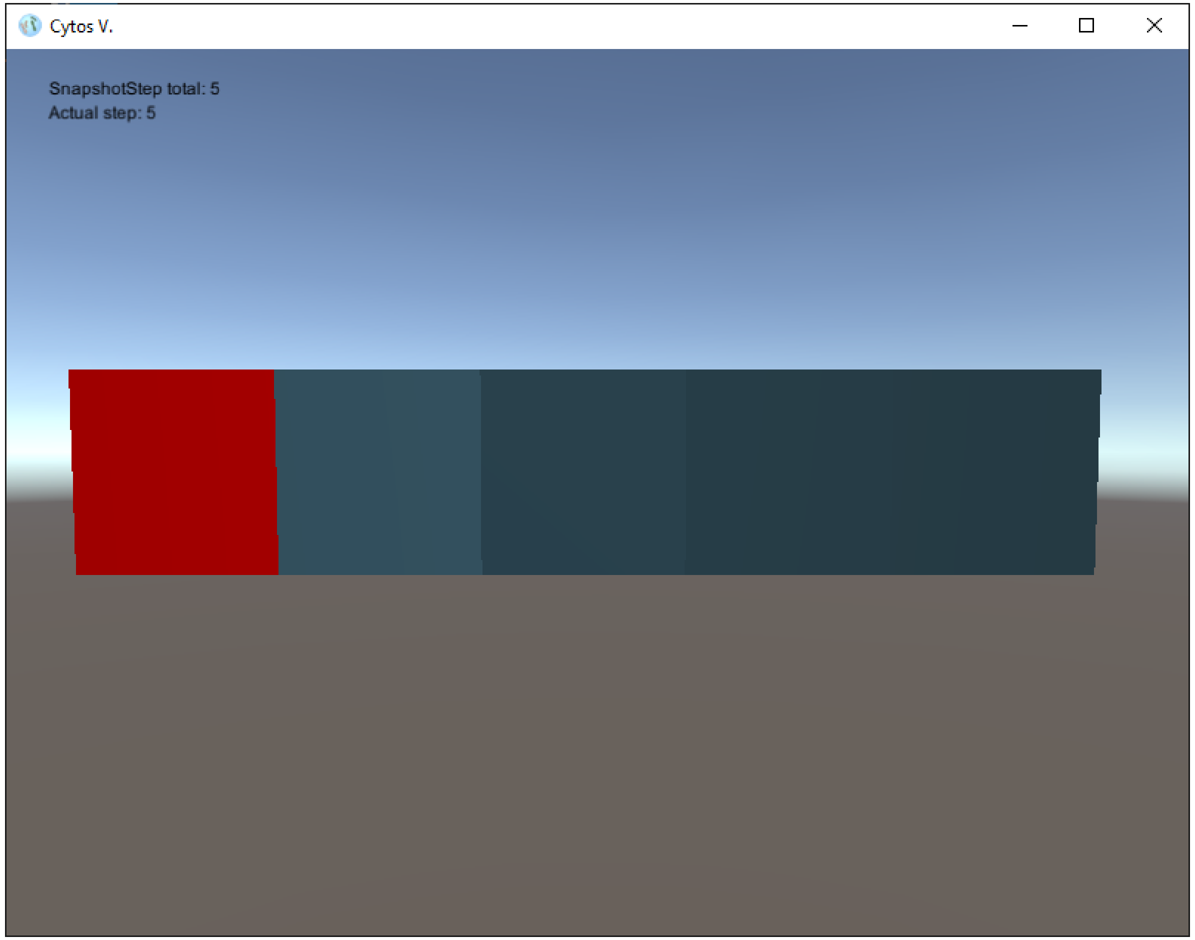
- During each step we take random tile
- Calculate threshold, for connection of a new tile
- A computation ends when output energy of all end tiles is lower than the threshold
- <u>Example</u>
  - Two different types of tiles (A, B)
  - Voltage: 100
  - Alpha ratio: A – 1, B - 2
  - Treshold: 1



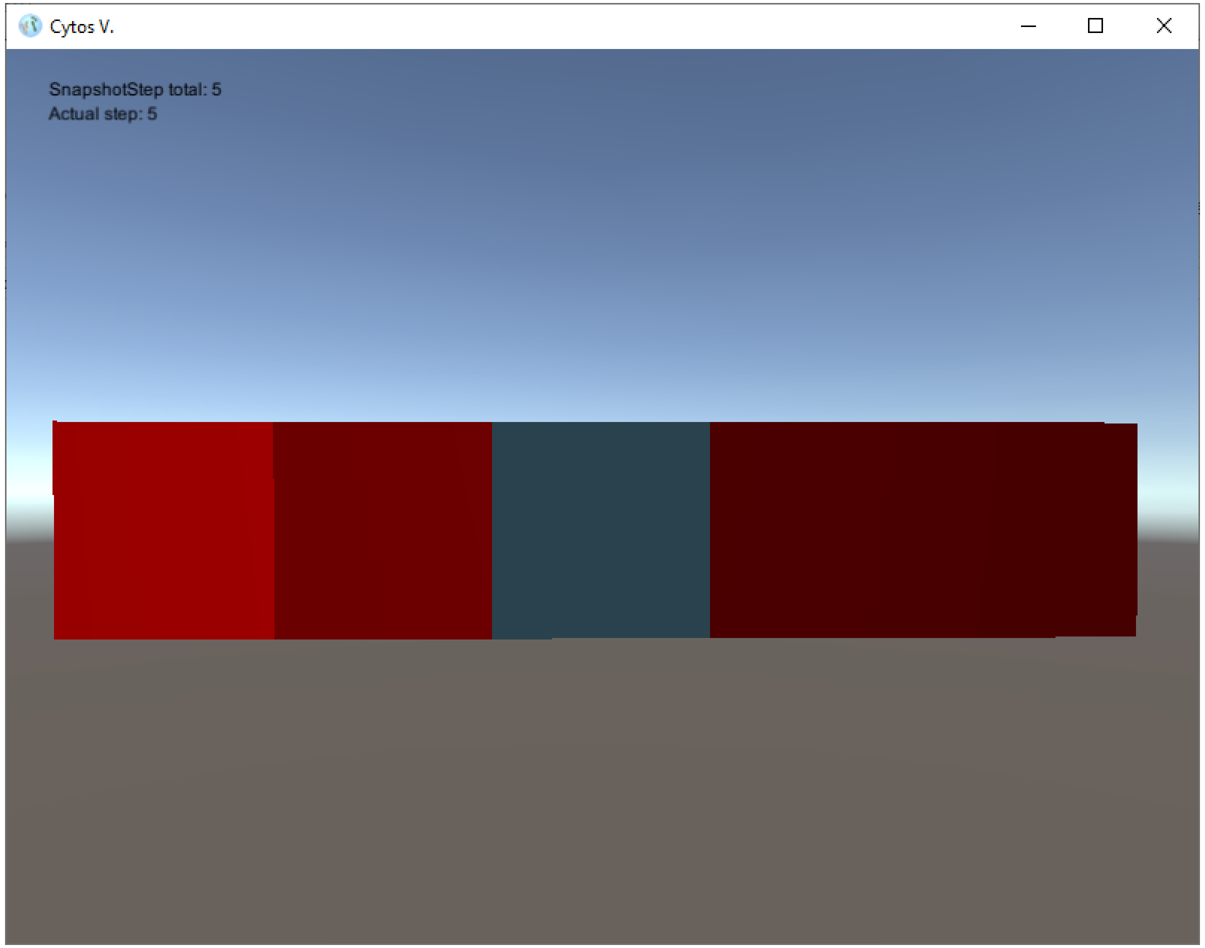
cTAM example



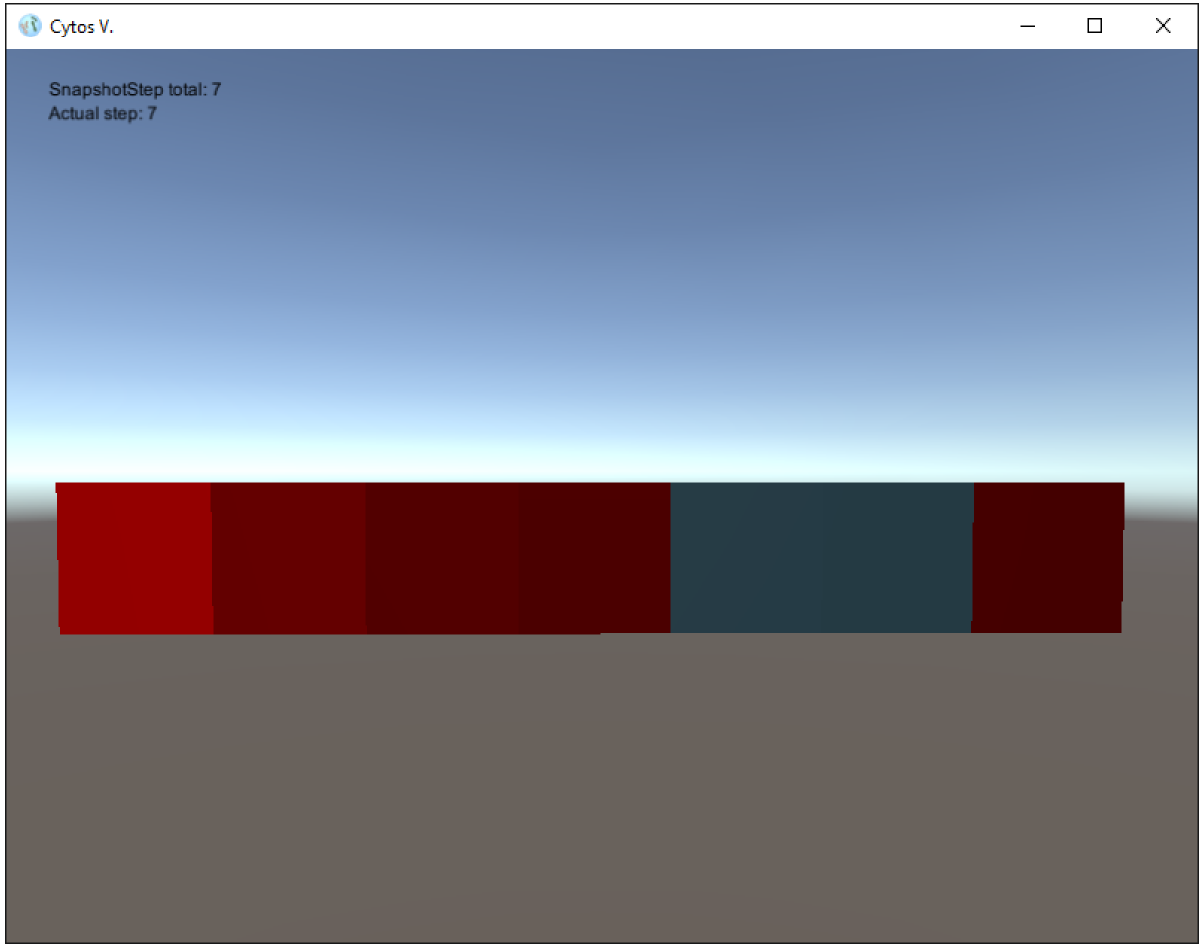
cTAM example



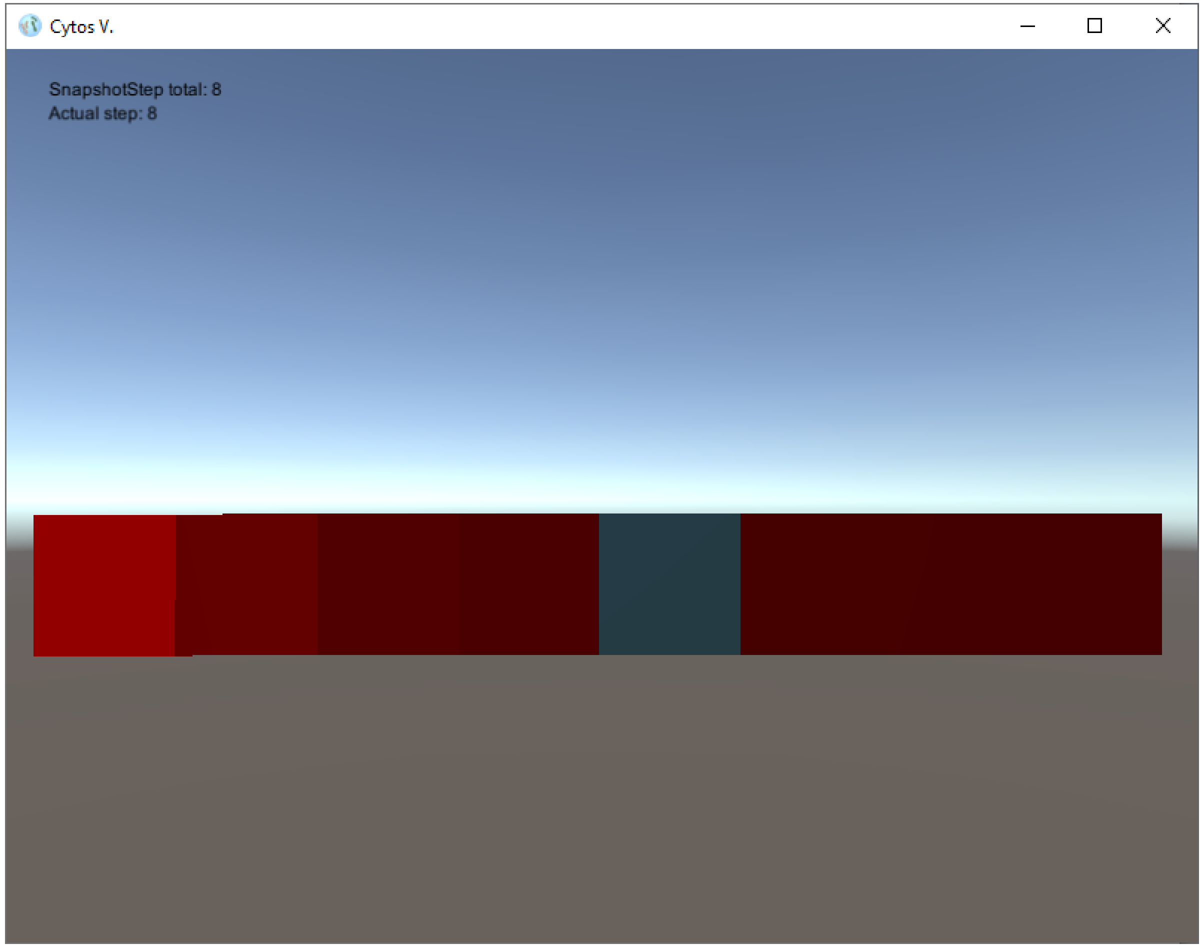
cTAM example



cTAM example (Vo=100, thP = 0.1)



cTAM example (Vo=100, thP = 0.1)

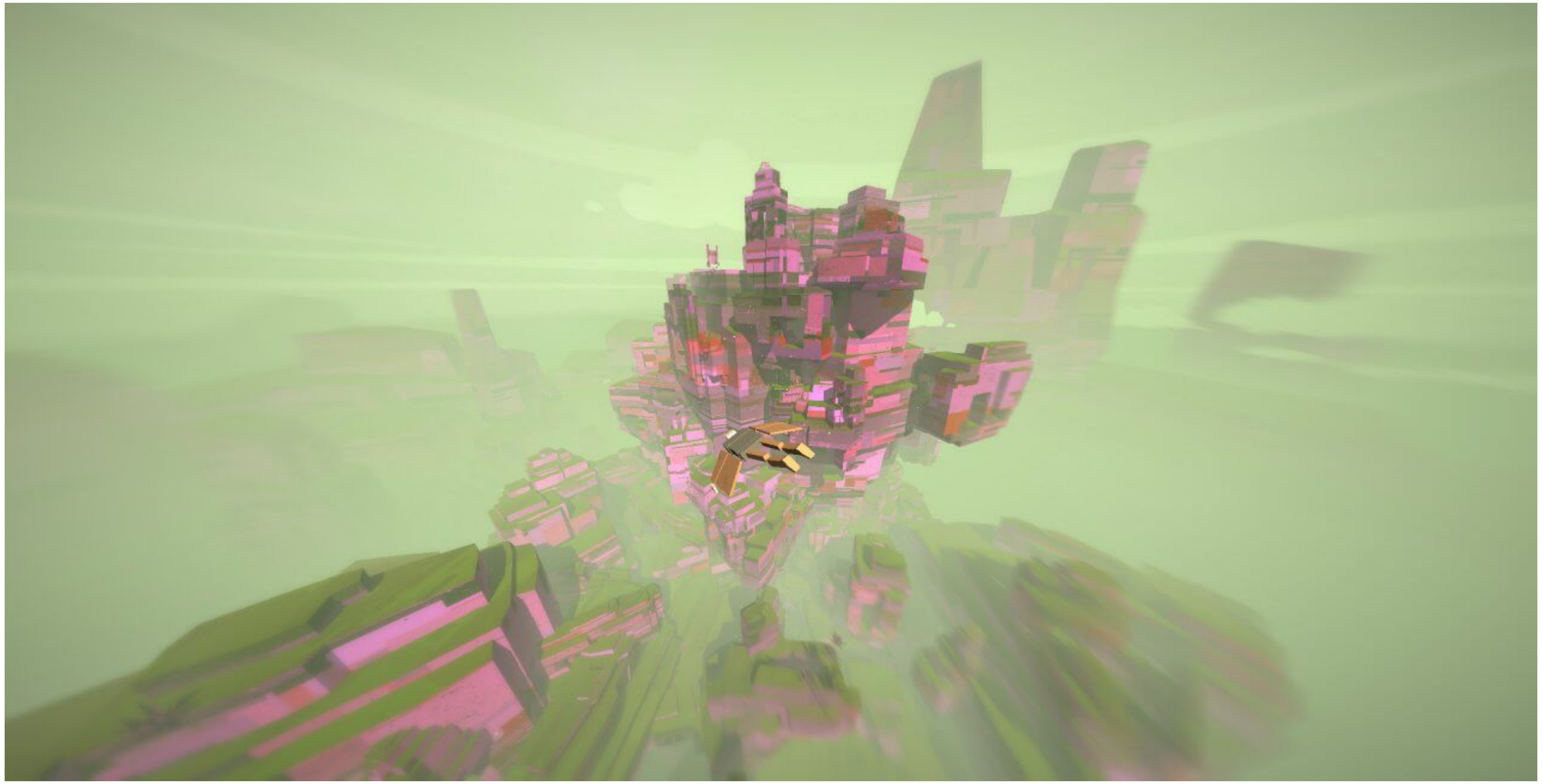


unity

Unity Game Engine

- Developer is the Unity Technologies
- is a multi-platform game engine (Windows, Mac, …)
- unity supports two and three-dimensional environment
- supports natively C# and UnityScript
- supports building to 27 different platforms

Unity Game Engine



Superflight

Unity Game Engine

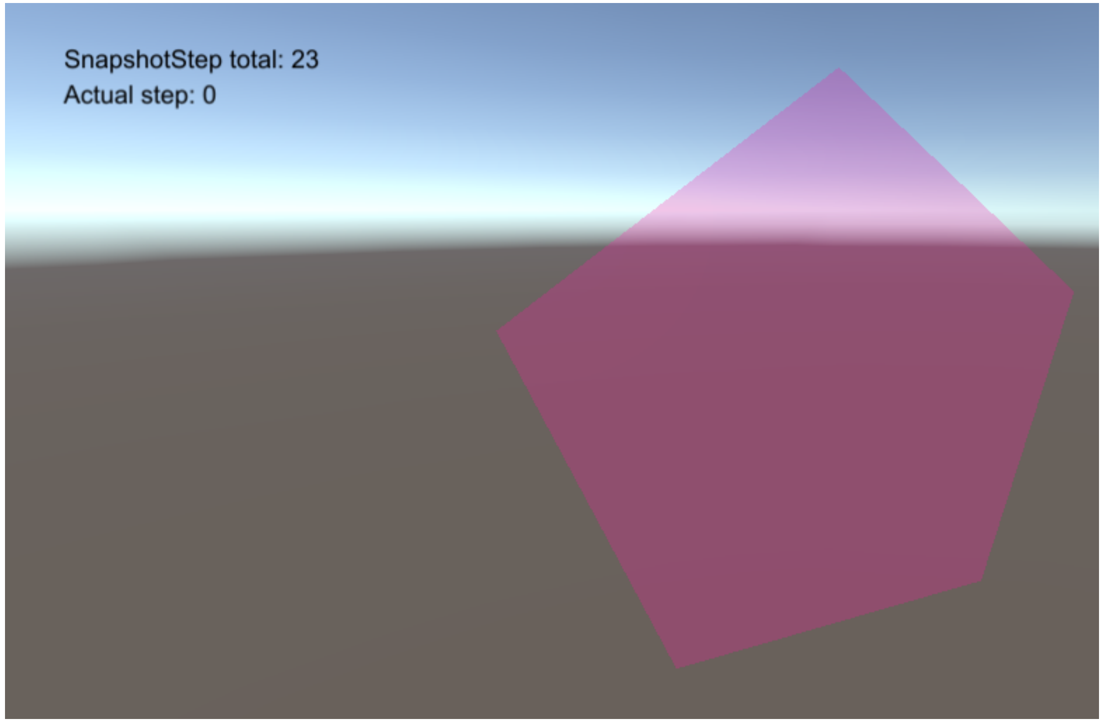
Monument Valley 2

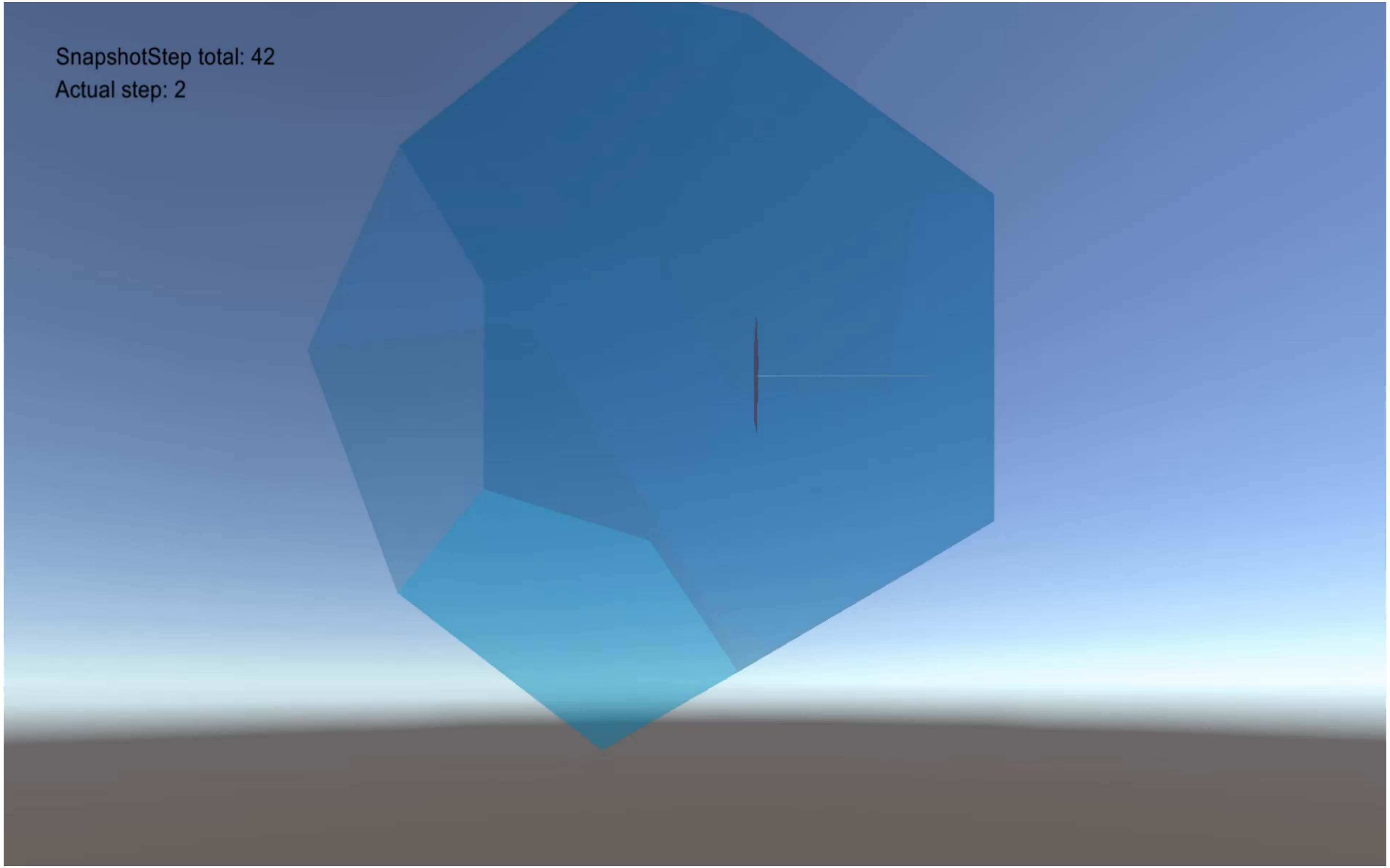
Visualization

- Input data, we get from our simulator (Snapshot file – XML),
- input file deserialize to new steps
- for tile, we use method Mesh drawing
- in visualization, we do not show floating objects
- for easy control in environment, user can use spectator mode, forward and backwards stepping
- Output data – complete visualization (Example M_0)

Example: Tile drawing

```
<tile name="s" objectID="2340" type="tile" state="Create">
    <vertices>
        <vertex>
            <posX value="0" />
            <posY value="10" />
            <posZ value="0" />
        </vertex>
        ….
    </vertices>
    <color name="40ff1493" />
</tile>
```




SnapshotStep total: 42
Actual step: 2

Thank you, any questions?

For more information and free download of the M system simulator and the visualization engine please consult Morphogenetic systems download page:

http://sosik.zam.slu.cz/msystem/

or use QR code