# Handout provided with the workshop Adapting Game Mechanics with Micro-Machinations

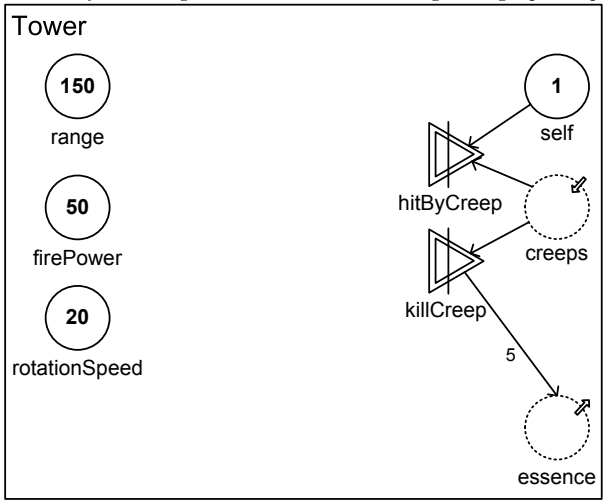
Riemer van Rozen[1,2]

[1] Amsterdam University of Applied Sciences
[2] Centrum Wiskunde & Informatica

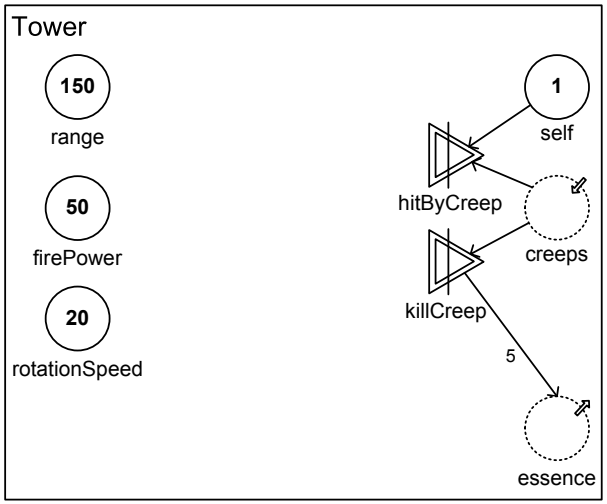
This handout is provided with the workshop Adapting Game Mechanics with Micro-Machinations at the Automated Game Design Symposium. Machinations is a notation for describing and communicating game designs [1] which evolved into the Micro-Machinations (MM) language that also enables formal analysis [2] and game development [3]. During this workshop we invite participants to modify the game mechanics of an example game called AdapTower. We provide a simplified cheat sheet of MM language elements in Appendix A. The workshop slides, the game and the MM software library driving it are available on GitHub[3].

## Sketch Area for Game Mechanics

We provide the vanilla (version zero) Tower mechanics which we modify during the workshop. You are encouraged to sketch ideas, share them, ask questions and modify these diagrams with new nodes and edges using a pen or pencil.




[3] http://vrozen.github.io/agd2014/



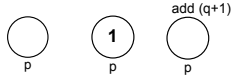

## References

1. Ernest Adams and Joris Dormans. *Game Mechanics: Advanced Game Design*. New Riders Publishing, Thousand Oaks, CA, USA, 1st edition, 2012.
2. Paul Klint and Riemer van Rozen. Micro-Machinations: A DSL for Game Economies. In Martin Erwig, RichardF. Paige, and Eric Van Wyk, editors, *Software Language Engineering*, volume 8225 of *Lecture Notes in Computer Science*, pages 36–55. Springer International Publishing, 2013.
3. Riemer van Rozen and Joris Dormans. Adapting Game Mechanics with Micro-Machinations. In *Proceedings of the 9th International Conference on the Foundations of Digital Games*, 2014.

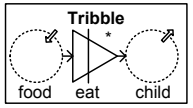
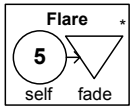
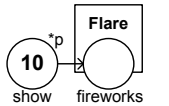
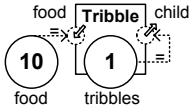
## A Micro-Machinations Cheat Sheet

Here we provide a cheat sheet of Micro-Machinations (MM) language elements that we may use during the workshop. MM models are directed graphs consisting of *nodes* and *edges*, which can be annotated with extra information. They describe the rules of internal game economies and define how resources are redistributed step by step between nodes.

## Visual Micro-Machinations of Basic Elements

| | |
|---|---|
|  **Empty pool** **Pool and resource** **Pool with add expression** | A *pool* is a named node, that abstracts from an in-game entity, and can contain *resources*, such as coins, crystals, health, experience, etc. Additionally, pools may have define a "bonus" expression whose evaluated value is added to the resource amount. Visually, a pool is a circle with an integer in it representing the current amount of resources, and the initial amount at which a pool starts when first modeled. |
|  **Resource connection with flow rate of one** **Resource connection with flow expression** | A *resource connection* is an edge with an associated expression that defines the rate at which resources can flow between source and target nodes. During each transition or *step*, nodes can *act* once by redistributing resources along the resource connections of the model. The *inputs* of a node are resource connections whose arrowhead points to that node, and its *outputs* are those pointing away. |
|  **Passive pool** **Interactive pool** **Automatic pool** | The *activation modifier* determines if a node can act. By default, nodes are *passive* (no symbol) and do not act unless activated by another node. *Interactive* (double line) nodes signify user actions that during a step can activate a node to act in the next state. *Automatic* (*) nodes act automatically, once every step. |
|  **Pool with pull act and all modifier** **Pool with push act and any modifier** **Pool with push act and all modifier** | Nodes act either by pulling (default, no symbol) resources along their inputs or pushing (p) resources along their outputs. Nodes that have the *any modifier* (default, no symbol), interpret the flow rate expressions of their resource connections as upper bounds, and move as many resources as possible. Additionally, these nodes may process their resource connections independently and in any order. Nodes that instead have the *all modifier* (&) interpret them as strict requirements, and the associated flows all happen or none do. |
|  **Source** | A *source* node, appearing as a triangle pointing up, is the only element that can generate resources. A source can be thought of as a pool with an infinite amount of resources. |
|  **Drain with any modifier** **Drain with all modifier** | A *drain* node, appearing as a triangle pointing down, is the only element that can delete resources. Drains can be thought of as pools with an infinite negative amount of resources, and have capacity to pull whatever resources are available, or whatever resources are pushed into them. |
|  **Condition edge equals one expression** **Condition edge greater equals expression** | A node can only be active if all of its *conditions* are true. A *condition* is an edge appearing as a dashed arrow with an associated Boolean expression. Its source node is a pool that forms an implicit argument in the expression, and the condition applies to the target node. |
|  **Trigger edge** | A *trigger* is an edge that appears as a dashed arrow with a multiply sign. The origin node of a trigger activates the target node when for each resource connection the source works on, there is a flow in the transition that is greater or equal to that of the associated flow rate expression. |
|  **converter** | A *converter* node consumes one kind of resource and produces another when activated, but never holds any resources. Converters appear as a triangle pointing to the right with a vertical line through the middle. |

## Visual Micro-Machinations of Modularity Elements

| | |
|---|---|
|  **Definition of Tribble contains references with in and out modifiers**<br>  **Definition of Flare contains a self pool** | A *type definition* is a named diagram that functions as parameterized module for encapsulating elements. Type definitions define internal elements and how they can be used externally.<br>A *reference*, represented by a circle with a dashed line, is an alias that refers to a node that defines it.<br>Internal nodes annotated with an *interface modifier input*, *output* or *input/output* become interfaces on the instances of the type. The input modifier denotes that an interface accepts inputs, output implies it accepts outputs and input/output accepts both. Interface modifiers appear as an arrow in the top right corner of a node, where an input modifier point into the node, an output modifier points out of the node, and an in-/output modifier accepts both.<br>When a type definition contains a pool named *self*, an *instance* of this type ends when the pool is empty. |
|  **Instance pool fireworks of type Flare** | An *instance pool* is a pool whose resource type is a definition. It represents a set of instances with individual instance data, whose shared interfaces are defined by its type and can be bound to other models, acting as formal parameters. Additionally, the size of the instance set is the amount of resources in the pool. Visually, an instance pool appears as a circle and a rectangle containing the type name. Instances are local to a pool and cannot flow out through resource edges. Resources flowing in create new instances, and those flowing out delete them. |
|  **Instance pool tribbles of type Tribble with bound interfaces** | An *interface* makes internal elements of instances available to the outside, and can be used by connecting resource connections. Visually, an interface is a small circle at the border of an instance with its name under it. Input interfaces have an arrow pointing into the circle, outputs have an arrow pointing outward, and in-/outputs have a bidirectional arrow. The direction of the arrow implies the validity of the direction of the resource edges that connect to it. Only reference interfaces appear with a dashed line. References must be bound to definitions using edges called *bindings*, represented by dashed arrows annotated with an equal sign, that originate from a defining node and target a reference. |