**Final Thesis**

**LIVE Game Design for Money Maker Deluxe**

**Author:** Christian Stiehl

**Student Number:** 500669869

**Education:** Information Technology (Game Development)

**Internship Supervisor:** Alexander Bonnee

****

**Final Thesis**

**LIVE Game Design for Money Maker Deluxe**

**Student:**  Christian Stiehl

**Student Number:** 500669869

**Phone:**  0627551183

**Email:**  [christian.stiehl@hva.nl](mailto:christian.stiehl@hva.nl)

**Place:**  Utrecht, Nederland

**Date:** 25-06-2017

**Institution:** Amsterdam University of Applied Sciences

**Education:** Information Technology (Game Development)

**Internship Supervisor:** Alexander Bonnee

**Research Supervisor:**  Riemer van Rozen

**Career Counselor:** Wally de Munk

**Company:**  Firebrush Studios

**Address:** Europalaan 20

3526 KS Utrecht

**Company Supervisor:**  Paul Brinkkemper

**Phone:**  0619027569

**Email:**  [paul@firebrushstudios.com](mailto:paul@firebrushstudios.com)

**Internship:** Second Semester, 2016/2017

# Prologue

This thesis is written to finish my Bachelor of Science at the Amsterdam University of Applied Sciences. It is meant as proof, to demonstrate the work I have done during my internship period and to show my abilities as a professional in my field.

There are many people I would like to thank for helping me along the way, my friends, family and the lecturers that have taught me. I have learned many lessons during this period of my life, some were of an academic nature and some were life lessons, these lessons have helped shape me into the person I am today.

Specifically, I would like to thank the following people; Wally de Munk for helping and guiding me throughout my bachelor. Alexander Bonnee for helping me with my internship and providing great advice and feedback for this thesis. Stefan Leijnen, Riemer van Rozen and Anders Bouwer for working on the LIVE Game Design project with me, helping me with my internship and helping me find this opportunity in the first place.

I would also like to thank Paul Brinkkemper and my co-workers at Firebrush Studios. It has been a fun and informative semester, and I wish them all the best.

**“An investment in knowledge pays the best interest.”**

- Benjamin Franklin

Table of Contents

[Prologue 1](#_Toc485047152)

[Abstract 5](#_Toc485047153)

[Introduction 6](#_Toc485047154)

[1. Context 8](#_Toc485047155)

[1.1 Firebrush Studios 8](#_Toc485047156)

[1.1.1 Money Maker Deluxe 8](#_Toc485047157)

[1.1.2 Money Maker Educative Goals 8](#_Toc485047158)

[1.2 LIVE Game Design 9](#_Toc485047159)

[1.2.1 LIVE Game Design Summary 9](#_Toc485047160)

[1.3 Research Definition 10](#_Toc485047161)

[1.3.1 Problem Definition 10](#_Toc485047162)

[1.3.2 Research Question 10](#_Toc485047163)

[1.3.3 Products 10](#_Toc485047164)

[1.4 Terms and Definitions 11](#_Toc485047165)

[2. Research Methods 12](#_Toc485047166)

[2.1 Preliminary Research 12](#_Toc485047167)

[2.2 Iterative Refinement 12](#_Toc485047168)

[2.3 Informed Design Decisions 13](#_Toc485047169)

[2.4 Using Existing Technology 13](#_Toc485047170)

[3. Requirements 14](#_Toc485047171)

[3.1 User Requirements and Library Requirements 14](#_Toc485047172)

[4. Products 15](#_Toc485047173)

[4.1 Tool 15](#_Toc485047174)

[4.1.1 Machinations 15](#_Toc485047175)

[41.2 Micro-Machinations 16](#_Toc485047176)

[4.1.3 Incorporating New Features 16](#_Toc485047177)

[4.1.4 UI Changes 17](#_Toc485047178)

[4.1.5 Model View Controller 17](#_Toc485047179)

[4.1.6 Back-End Implementation 18](#_Toc485047180)

[4.2 Expressing Educative Goals with Micro-Machinations 19](#_Toc485047181)

[5. Conclusion 21](#_Toc485047182)

[5.1 Existing Technology 21](#_Toc485047183)

[5.2 Functional Requirements 21](#_Toc485047184)

[5.3 Possibilities 21](#_Toc485047185)

[5.4 Usage 21](#_Toc485047186)

[5.5 Conclusion 21](#_Toc485047187)

[Recommendations 22](#_Toc485047188)

[Additional Features 22](#_Toc485047189)

[Importing Old Micro-Machinations 22](#_Toc485047190)

[Diagram Refactoring 22](#_Toc485047191)

[Interactive Tree View / Runtime View 22](#_Toc485047192)

[Prediction, Simulation and Statistics 23](#_Toc485047193)

[Threats To Validity 24](#_Toc485047194)

[Design Decisions 24](#_Toc485047195)

[Micro-Machinations Knowledge 24](#_Toc485047196)

[Epilogue 25](#_Toc485047197)

[Sources & References 26](#_Toc485047198)

[Attachments 27](#_Toc485047199)

[I LIVE Game Design Summary 27](#_Toc485047200)

[II Visual Definition Editor Specification of Micro-Machinations 29](#_Toc485047201)

[Element 29](#_Toc485047202)

[Node 29](#_Toc485047203)

[Pool 29](#_Toc485047204)

[Gate 30](#_Toc485047205)

[Source 30](#_Toc485047206)

[Drain 30](#_Toc485047207)

[Reference 30](#_Toc485047208)

[Converter 31](#_Toc485047209)

[Interface 31](#_Toc485047210)

[Edge 31](#_Toc485047211)

[Flow Edge 31](#_Toc485047212)

[State Edge 32](#_Toc485047213)

[III Micro-Machinations Grammar 33](#_Toc485047214)

[IV Exported Micro-Machinations Diagram 37](#_Toc485047215)

# Abstract

Christian Stiehl, Game Development, Amsterdam University of Applied Sciences

Abstract of Bachelor’s Thesis, Submitted 13 June 2017:

LIVE Game Design for Money Maker Deluxe

The aim of this thesis is to identify the requirements of a Live Intelligent Visual Environments (LIVE) tool for game designers, to document the development of such a tool and to give recommendations for the future of the LIVE Game Design project at the Amsterdam University of Applied Sciences (AUAS).

In the first part the topic and the associated companies are introduced. This research was conducted for the LIVE Game Design project at the AUAS as a pilot study at Firebrush Studios. The research is based on the previous research conducted at Firebrush Studios by Tom Vaessen and numerous papers written about Micro-Machinations by Riemer van Rozen.

The thesis then explains the research methods used. Because of the lack of comparable existing technology most of the development was iterative refinement, trying certain features in a certain way and evaluating if they meet the requirements by the target audience. Work was done in weekly development cycles.

The core of the thesis is the explanation of the requirements set for the tool and the in-depth documentation of the tool. There are two sets of requirements for the tool, usability for the target audience and compliance with the Micro-Machinations library for C#. During the development of the tool, the library was redesigned for use with visual tools by Riemer van Rozen. In collaboration, the library and tool were tweaked to work together in optimal fashion. The tool itself underwent lots of design changes during development. The core design was based on the Machinations tool by Joris Dormans, this design was adapted and extended to allow for functionality introduced in Micro-Machinations.

The thesis then showcases some of the usages of the tool by expressing the educational goals of the game Money Maker Deluxe by Firebrush Studios in Micro-Machination diagrams. These diagrams represent the game economy and can later be used to simulate the game flow and predict certain outcomes.

The thesis concludes with a recap of all sub questions and answers them with the information presented so far and evaluates the research and products as a whole. We conclude that the research was a success even though the tool was not fully completed.

# Introduction

This thesis is written to document the creation of a live game design tool and its uses for the game Money Maker Deluxe developed by Firebrush Studios. This tool will be used by the LIVE Game Design research project to help game designers change game mechanics at runtime and receive valuable feedback in real time. This instant feedback will help shorten the iteration cycle of game development.

This thesis is based on previous research at Firebrush Studios conducted for the LIVE Game Design project by Tom Vaessen and uses Micro-Machinations, a visual programming language developed by Riemer van Rozen.

This thesis is divided into several chapters so it can be easily read and understood. Because of this structure some chapters will include information explained in earlier chapters. It is recommended to read the chapters in order.

The products that would initially be delivered were a Live Game Design Tool based on Micro-Machinations and a prototype of the game Money Maker Deluxe made using the Tool.

However, to ensure this project got completed the prototype was delineated. Because the back-end implementation of the Micro-Machinations Library took longer than expected, a full prototype was not feasible. Instead a series of diagrams simulating the educative goals of Money Maker Deluxe were created using the completed tool.

The first chapter explains the context of the project. This includes a brief history of Firebrush Studios and the game Money Maker Deluxe. Information about the LIVE Game Design research project and the research definition including the problem definition, research question and the products that accompany this thesis.

The second chapter explains what research methods were used during this project. It also summarizes the preliminary research done by the previous intern at Firebrush Studios.

The third chapter describes the requirements that had to be met when building the tool.

The fourth chapter puts the products that accompany this thesis in perspective and explains them. Included in this chapter is an overview of the tool and a rundown of the diagrams created using the tool.

The fifth chapter concludes the thesis by answering the primary research question using the information presented in the previous chapters.

At the end of this document you can find recommendations for the future of this project, sources and references, terms and definitions and some attachments.

**Version History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Status** | **Changes** |
| v0.1 | 06-03-2017 | Template | Added front page, foreword, table of contents. Set up first chapter including research question and problem definition. |
| v0.2 | 13-03-2017 | Template | Added more information about Money Maker and LIVE Game Design to the first chapter. Ready for peer review meeting. |
| v0.3 | 20-03-2017 | Template | Added more content to the introduction and incorporated some feedback from the first peer review meeting. |
| v1.0 | 27-03-2017 | Concept | Added remaining chapters and started filling out all chapters with the progress made so far. |
| v1.1 | 03-04-2017 | Concept | Filled out Research Methods, finished Context and added to Products. |
| v1.2 | 10-04-2017 | Concept | Restructured chapter 4, fleshed out chapter 3.1. Added threats to validity |
| v2.0 | 08-05-2017 | Final Concept | Preparation for Go/No Go meeting in week 15. Explained the delineation of the prototype in the introduction. Added first educative goal diagrams to Chapter 3. Started filling out chapter 4. |
| v2.1 | 11-05-2017 | Final Concept Addendum | Last fixes for preliminary screening before the Go/No Go meeting. Switched chapter 3 and 4. |
| v3.0 | 06-06-2017 | Preparation Final Version | Added conclusion, filled out chapter 3. Started working on the abstract. Need to revise chapter 4.2. |
| v3.1 | 08-06-2017 | Preparation Final Version | Added abstract, revisions to 4.2 still needed. Preparing version for proofreading. |
| v4.0 | 12-06-2017 | Final | Implemented feedback finished chapter 4.2 |

Table 1: Version History

# 1. Context

This chapter describes the background and context of the research. To help illustrate the research it will include a closer look at Firebrush Studios and the LIVE Game Design project. It also contains the definition of the research question this thesis is based on, the problem definition and some terms and abbreviations.

## Firebrush Studios

Firebrush Studios is a SME Game Development studio located in Utrecht, founded by Paul Brinkkemper and Alexander Kappelhoff.

### 1.1.1 Money Maker Deluxe

Firebrush Studios started the development of Money Maker Deluxe in 2011. The game is inspired by the economic crash of 2008. The goal of the game is to educate people about the banking system and the problems within this system. While in development Firebrush Studios had eight educative goals the game should teach the player. Achieving all eight of these goals was a huge challenge because of the slow nature of the iteration cycle of digital games.

The team started using more paper prototypes and eventually switched to developing a board game. After only three months their board game prototype had all eight educative goals. Compared to the six years they had worked on Money Maker Deluxe this was incredibly fast.

It was then that Firebrush Studios got in contact with the Amsterdam University of Applied Sciences and the LIVE Game Design project. They would like a way for them to develop their digital version of the game as fast as their paper prototyping went.

### 1.1.2 Money Maker Educative Goals

Money Maker Deluxe wants to educate high schoolers about the following topics:

1. How banks earn money by lending more money than they have.

2. The difference between money and credit.

3. How credit is created as a promise of money.

4. How credit creation raises prices.

5. How credit creation in large amounts causes a credit bubble.

6. How a credit bubble grows larger and becomes unstable.

7. How this instability leads to a Minsky moment, a collapse of the system.

8. How to use your money wisely so when the system collapses, you will not be affected but can profit instead.

## LIVE Game Design

The LIVE Game Design project is the overarching research project the Amsterdam University of Applied Sciences takes part in. For this research a number of companies were asked to contribute to the project by hosting research projects. One of the companies asked to contribute was Firebrush Studios, which was a perfect candidate for the research as the results could help them develop Money Maker Deluxe a lot easier and faster.

The goal of this research project is to provide software and tooling that can help shorten the design cycle of digital games. By providing real time feedback and feedforward designers can make more informed decisions when designing their game.

### 1.2.1 LIVE Game Design Summary

For a complete summary of the LIVE Game Design project please refer to attachment I.

## Research Definition

The title of this thesis, as approved by the board of examiners, is:

**LIVE Game Design for Money Maker Deluxe**

This thesis is about the creation, implementation and testing of a tool designed to adjust games live (during runtime). And how it can be used to help Firebrush Studios create the game Money Maker Deluxe faster while still achieving the eight educative goals they have set for it.

### Problem Definition

The current iteration cycle of the digital game Money Maker Deluxe is too long, and the code has become too complex to easily make changes to mechanics. The aim of this research is to expedite the iteration cycle by creating a tool that can change the game mechanics at runtime so the designer can test his changes immediately.

### Research Question

To describe the project and measure the results, a research question was set up for this thesis. The research question is as follows:

**How can an interactive visual tool be made to enable designers to change game mechanics at runtime?**

To help answer this research question a number of sub-questions have also been defined:

* **What existing technology could be used for such a tool?**
* **What functionality does a designer require for this task?**
* **What possibilities should the tool offer the designer?**
* **How can a generic tool be used for a specific game?**

### Products

The products that accompany this thesis are a tool that can change game mechanics at run time and a series of Micro-Machination diagrams visualizing the educative goals of Money Maker Deluxe created using this tool.

## Terms and Definitions

A lot of specific terms and abbreviations are used in this thesis. Below are two tables containing the most common abbreviations and terms used and their definition.

|  |  |
| --- | --- |
| **Abbreviation** | **Definition** |
| AUAS | Amsterdam University of Applied Sciences |
| LIVE | Live Intelligent Visual Environments |
| MM | Micro-Machinations |
| MMD | Money Maker Deluxe |
| MVC | Model View Controller |
| UI | User Interface |

Table 2: Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Game Mechanics | The inner workings of a game. |
| LIVE Game Design | Live Intelligent Visual Environments for Game Design is a research project hosted by the Amsterdam University of Applied Sciences. |
| Machinations | A visual game design language designed by Joris Dormans. |
| Micro-Machinations | A concrete version of machinations created by Riemer van Rozen. |
| Runtime | The game is running, the player can interact with it. |
| Tool | A piece of software designed to help people accomplish a goal easier. |
| Unity | The engine used to create the tool. |

Table 3: Terms

# 2. Research Methods

This chapter describes what research methods were used for this thesis. It also includes previous research for the LIVE Game Design project.

## 2.1 Preliminary Research

This thesis is built on the foundation created by Tom Vaessen, who did the preliminary research for the LIVE Game Design project at Firebrush Studios (Vaessen, 2017). In his thesis he explores the potential technologies that can be used to create a prototype of Money Maker Deluxe that can be changed at runtime, comparing them based on which one could change mechanics the fastest.

From this comparison, he concluded that Micro-Machinations would be the fastest, but it would also require a redesigned C# version of the Micro-Machinations library. This version was not available during his internship period.

By the time this thesis started the C# version was well on its way and the development of a visual tool to accompany it could begin. To get a better understanding of Micro-Machinations and the work that needed to be done research papers by Riemer van Rozen about the language were consulted (Rozen, 2013).

## 2.2 Iterative Refinement

From there the process was mostly iterative refinement. The work was done in weekly design cycles, thinking about problems the users would have or what information they would require, creating functionality in the tool to solve these problems and checking with experts at the AUAS to see if the solutions were implemented correctly.

After the tool was in a testable state the weekly design cycles were adjusted to include observation and testing. Game designers were asked to use the tool and were observed to see what information they were lacking or what problems they had when using the tool. From these observations, the design could be adjusted more accurately to the actual needs of the target audience.

## 2.3 Informed Design Decisions

In the early stages of the project, the design was based mostly on the Machinations tool made by Joris Dormans (Dormans, 2012). However Micro-Machinations does not share all of the features that Machinations has, and in turn also has additional features. Therefore, the design started to adapt to these differences by removing deprecated features and designing solutions for the additional features.

Because this tool was to be the first interactive Micro-Machination tool based on the Micro-Machinations library, there was not much previous information to go on. While working on solving these design problems Riemer van Rozen and Stefan Leijnen were often consulted to make sure these solutions were user friendly and in line with the skill set of game designers.

The original design of Machinations is still reflected in the design of the new tool because of game designers who have already learned how to work with the Machinations tool. They will be more familiar with the new tool because of the similarities to the Machinations tool.

## 2.4 Using Existing Technology

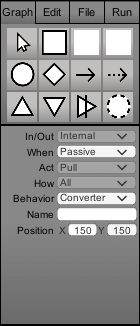
To summarize, this chapter and to answer the first sub-question, what existing technology can be used for this tool, we can see the three main sources of information and inspiration. The Machinations tool by Joris Dormans was used for UI inspiration and basic functionality, the Micro-Machinations Library by Riemer van Rozen was used as a back-end and the preliminary research by Tom Vaessen was used to determine what technologies were worthwhile to begin with.

# 3. Requirements

This chapter explains what requirements had to be kept in mind when developing and designing the tool.

## 3.1 User Requirements and Library Requirements

During the development of the tool many design decisions were made to accommodate future users. The goal of the tool is to help game designers accomplish their tasks in the game development cycle. The tool needs to be a simple way for designers to understand Micro-Machinations and help them create diagrams without needing to know the exact syntax.



The ease of use is not the only requirement the tool had to comply with. The tool needed to be built in a specific way to implement the Micro-Machinations library. Certain features needed to be present to represent functionality added in Micro-Machinations like tabs for different definitions, functionality to import and export textual diagrams and more.

Sometimes these requirements went hand in hand even complementing each other. The attribute editor is a perfect example, it is ordered in the way the attributes are printed in the syntax, giving the designer some insight on how to read textual micro-machinations. The tool also greys out attributes that cannot be changed to prevent user error.

Other times combining these requirements was more difficult. Because Micro-Machinations was formerly a textual language, visualizing all elements was difficult. Riemer van Rozen already had visualizations for a lot of elements in his various papers (Rozen, 2013, 2014, 2017). Moreover, several tool prototypes exist, notably Micro-Machinations Analysis in Rascal (MM AiR) (Klint, 2013) and Mechanics Design Assistant (MeDeA) (Rozen, 2015). However, several interactive visualizations were not analyzed and designed yet or had to be changed for easier understanding.

Figure : Node Inspector

Luckily the development of this tool went hand in hand with the development of a new Micro-Machinations library in C#. Many issues were fixed by collaboration with Riemer van Rozen. For example, the visual locations of elements had to be stored in the textual notation. Since there was no need for these to exist before, they had to be added.

# 4. Products

This chapter documents the creation of the products that accompany this thesis. This includes design decisions made before and during development of the tool.

## 4.1 Tool

In the preliminary research conducted by Tom Vaessen for Firebrush Studios, he concluded that Micro-Machinations would be the best solution for the problem Firebrush Studios had with Money Maker Deluxe. However, he also concluded that it would require new tooling and porting from C++ to C#.

During a meeting with Paul Brinkkemper, Tom Vaessen and Stefan Leijnen it was discussed what the best course of action was for the future of the LIVE Game Design project at Firebrush Studios. From this meeting, it was concluded that a Micro-Machinations tool would be the most useful product, not only for Firebrush Studios but also for other game development studios struggling with long design cycles.

Due to Money Maker Deluxe being developed in Unity and the easy to port nature of Unity it was decided to create the tool in Unity as well. This way it could be used as a standalone application for designers to play around with or be easily incorporated in an existing Unity game.

### 4.1.1 Machinations

The initial design of the tool took a lot of inspiration from the original Machinations tool created by Joris Dormans. Some additional features had to be included in this new tool to accommodate features that are additions to Micro-Machinations.

Machinations is a visual game design language, it can help game designers test economies and even small games by simulating resource flow. It is however very abstract, the nodes and edges within the diagrams do not relate to real game objects easily.

A Machinations diagram consists of nodes and edges. Nodes are objects that have a certain behavior, each node can only have one behavior. Behaviors are the inner working of a node, for example the pool behavior node can hold an amount of resources, the source behavior node can produce resources and the drain behavior node can drain resources.

To connect these nodes a designer can use edges. Edges are lines drawn between nodes to simulate the flow of resources. A pool connected to a source with an edge will slowly fill up with resource and a pool connected to a drain will slowly empty out. Edges also have two types of behavior. State edges and Flow Edges. State edges have a condition attached to them, only transferring resources when the condition is met. Flow edges have an amount attached to them, indicating how many resources the edge tries to move at once.

### 41.2 Micro-Machinations

Micro-Machinations is a programming language created by Riemer van Rozen that formalizes Machinations meaning. This way it can be integrated in actual games and help designers not only simulate mechanics or economies, but actually influence them and make predictions about the quality of game software.

Since the creation of Micro-Machinations in 2013 a couple of games have integrated it, however the process is cumbersome and the library is written in C++.

The biggest additions to Micro-Machinations are references and definitions. Definitions enable the designer to create sub-diagrams. A definition is attached to a pool and translates the number in the pool to instances of the defined diagram. For example, if a designer creates a diagram representing a farm building that produces one food with each cycle, he could then create a definition of this farm diagram. Now whenever the pool of the farm definition gets activated a new farm is created and the production of food increases by one.

References are also involved in these definitions. A reference is a pool that refers to a different pool, in a different diagram. When creating a definition, interface nodes will appear on it based on the references within the defined diagram. These can then be connected to the corresponding pools. Using our earlier example, the food pool could be turned into a reference and connected to a new pool next to the definition of the farms. Now the farms no longer create food just for themselves, but they all send the food to a cumulative stockpile.

### 4.1.3 Incorporating New Features

The first step in designing the tool was creating a way for people to have multiple diagrams open at the same time, navigating them easily and creating new diagrams with ease. For this feature, tabs were added to the diagrams, the design of these tabs was inspired by web browsers like Google Chrome and Mozilla Firefox and programming IDEs like Visual Studio.

With the addition of definitions and references, space was needed to instantiate these node types. Luckily Micro-Machinations no longer supports some of the node types present in Machinations. The following node types were cut to create space for new node types and a larger inspector: Trader, End Condition, Register and Delay.

Because Micro-Machinations is a programming language that is interpreted, chances for user error are increased drastically. Because of this increased chance, a debugging console was added. This could let the user know if an error was found and show him the nodes or edges that were having problems and what the problem was. For this console, a small section at the bottom of the screen was designated, which can be expanded by the user.

The node and edge inspectors were also overhauled. A lot of the node specific features in Machinations were removed and only the grammar of the node remained accessible. This grammar consists of the In/Out, Act, How, When, Behavior and Name. Because some node behaviors can only ever have a certain Act or a certain How, these fields were set to a default value and greyed out in the inspector. For more information about the grammar of Micro-Machinations and an overview of what node behaviors have default values please refer to attachment II.

### 4.1.4 UI Changes

As the tool development progressed, some major changes happened with the design of the UI. Some key elements of the final UI were not part of the original design, looked different and some elements were cut entirely.

One feature that was part of the initial design was the header bar at the top of the tool. This bar was taken from the original Machinations design. It was initially included to store buttons for different functionalities that could not fit in other places of the UI. These buttons were later either removed or replaced. For a while the bar remained at the top of the tool, with the intention of making it easier for future interns to add functions and store them on the bar. However, to create a better tool the bar was removed.

Another example of these features is the aforementioned debugging console. Since the original design of the Machinations tool did need a debugging console, no screen space was dedicated to it. The console was placed at the bottom of the screen, mirroring the usual placement of debugging consoles in other IDEs.

### 4.1.5 Model View Controller

Because the tool is a bridge between visual programming and creating textual Micro-Machination models, the Model View Controller design pattern was chosen. This design pattern has three members, the Model, in the case of this tool this is the Micro-Machination model, an Abstract Syntax Graph (ASG) representing the program. The view, in the case of this tool this is the visual aspect of the tool, seeing nodes and edges on the screen. And the controller, the bridge between the model and the view. Whenever something in the view changes the controller informs the model and updates it accordingly.

### 4.1.6 Back-End Implementation

The back-end to the tool was developed by Riemer van Rozen, it is a redesigned C# version of the Micro Machinations library that borrows much of the original C++ version. The original library was however made for a textual editor, with the new visual editor changes needed to be made.

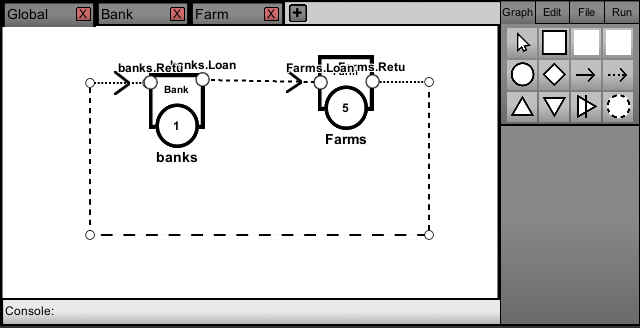
During the implementation of the back-end many issues with the old library became apparent. Nodes did not have positions in the textual notation, nodes could not be added or changed dynamically directly in the ASG, because it parsed the whole textual program again. In the visual editor however, the user would have to place a node and then change its values, so the node needs to be updated with new information whenever the user changes it.

While implementing the back-end a lot of changes were still being made to it. During this process contact with Riemer van Rozen was key to successfully creating and implementing the back-end simultaneously. In collaboration, the Micro-Machinations library, specifically its APIs, and the tool were iteratively improved.

One such improvement is the greatly simplified binding of references. We decided to disallow the internal binding of references because an instance is responsible for the amount of resources that flow in or out. It can only be bound to the outside world via its interfaces externally. By extension this also disallows the use of IO modifiers on pools.

## 4.2 Expressing Educative Goals with Micro-Machinations

To demonstrate the tool and to showcase what can be made with it, a small diagram was created. This diagram showcases a game economy representing the first educative goal of Money Maker Deluxe.

The diagram consists of three definitions. It showcases the first educative goal, ‘How banks earn money by lending more money than they have’.

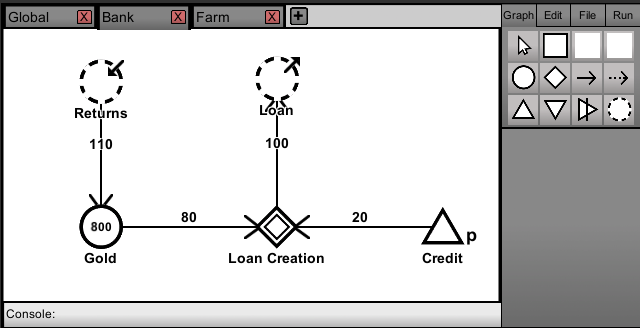
Depicted in Figure 2 is the Global definition. This definition contains the core of the diagram. There are two pools, banks and farms. There are currently 5 farms and 1 bank, indicated by the numbers inside the pools. The loans the banks provide (banks.Loan) is connected to the loan interface of the farms and the returns the farms pay (farms.Returns) is connected to the return interface of the banks.

Figure 2: Global Definition first educative goal

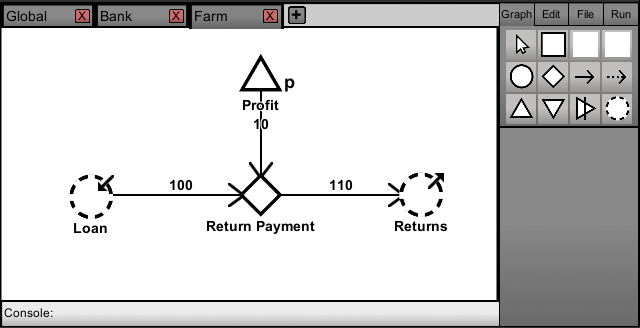
Depicted in Figure 3 is the Bank definition. This definition represents the banks in the diagram. To represent the learning goal the bank currently gives out set loans of 100, these loans consist of 80 gold and 20 credit, where gold is a finite pool and credit is an infinite source.

Figure 4: Farm Definition first educative goal

Figure 3: Bank Definition first educative goal

If the player would create 10 loans he would use 800 gold and 200 credit. However, the returns bring 110 gold back to the bank. Therefore, these 10 loans would bring the bank 1100 gold back. The bank earned money by lending more than it had by creating credit.

For the purpose of simplicity, in this diagram the Farms take their loans and generate a profit of 10 to instantly pay back their loans, including interest.

This diagram can then be exported as a text file, this text file could potentially be read by a game to adjust game mechanics and variables. The text file created from this diagram is available in Attachment IV.

# 5. Conclusion

This chapter documents the findings and concludes the paper using the information presented so far.

## 5.1 Existing Technology

Creating a tool to enable designers to change game mechanics at runtime would take a very long time without the help of any existing technology. Thanks to the preliminary research by Tom Vaessen a direction was chosen right from the start, to use the Micro-Machinations library by Riemer van Rozen. The usage of such technology is critical because it sets the precedent for the entire tool and its functionality.

## 5.2 Functional Requirements

The requirements for a designer to change game mechanics at runtime are an easy to understand UI that helps them understand or avoid the complicated textual syntax, and the semantics underneath. The tool also needs to be able to be opened during gameplay, so designers can playtest, change some mechanics or variables and continue testing in the same game they had before. This instant feedback will help designers shorten the design cycle of game development significantly.

## 5.3 Possibilities

The possibilities offered by the tool should range from simulation to actually changing the mechanics. Being able to simulate a game economy or a small part of the game can already lead to significant insights about the game. This should always be the first step when designing the rough concept of the game. Once the game needs to be fine-tuned the tool should offer the possibility of changing the mechanics and provide statistics about the changes.

## 5.4 Usage

At this stage of development the tool can only be used to create the diagrams that can later be adapted to game mechanics. However, these diagrams can already represent parts of the game, or entire game economies. With some more work the tool could certainly achieve the goal of enabling game designers to change game mechanics at runtime.

## 5.5 Conclusion

The development of an entire tool capable of changing game mechanics at runtime and applying this tool to a prototype version of an existing game was too big of a goal. With more time the tool can certainly achieve this goal, but in a period of 4 months this is just not feasible. Where this research did succeed was breaking the ground on the development of the tool and provide information and recommendations on the future of this tool or any other tool like it.

# Recommendations

This chapter contains recommendations for the future of the Micro-Machinations Tool and the LIVE Game Design project at Firebrush Studios. It lists features of the tool that have been considered and designed but have not been implemented yet, so future interns can easily improve upon the tool by implementing these features.

## Additional Features

During this internship period the focus was on establishing a working version of the tool, so certain features were considered but deemed not required. Some of these features increase efficiency of the tool, some are just in the interest of keeping the tool easy to understand for designers.

### Importing Old Micro-Machinations

One of the features of the tool is the ability to import textual Micro-Machinations of earlier versions and provide a visual for them. However, the tool operates in a new version of Micro-Machinations that contains a position variable, older versions of Micro-Machination files lack this variable, so the tool cannot place the nodes on the screen.

A short research in placement algorithms should be done and the best choice should be used to place nodes in a logical manner when importing nodes that have no position. A function has already been made that is called when a Micro-Machinations file is imported that does not have positions, currently it sends an error message to the console. This function could contain new logic to assign new positions to all imported nodes.

### Diagram Refactoring

A feature could be added that refactors the diagram to an optimal lay out, keeping in mind readability but reducing white space as much as possible. This feature could go hand in hand with the placement algorithm for the importing of old textual Micro-Machination diagrams.

### Interactive Tree View / Runtime View

Riemer van Rozen describes an interactive tree view in his unpublished work on Micro-Machinations (Rozen, 2017). This view should represent the Micro-Machinations diagram in a tree view and keep user interactable nodes clickable. A way to display higher numbers of instantiated definition needs to be considered. If there is a pool with 50 Farms for example, the user should be able to enter the number of the exact farm he wants to inspect.

This view could also double as the runtime view, a view available to the designer while the game is running, where changes instantly appear in the game.

### Prediction, Simulation and Statistics

Once the interactive tree view is implemented the tool should also try to recognize patterns within the diagrams and predict outcomes, simulate the diagrams multiple times and provide statistics of these predictions and simulations to the designer. This could help the designer create game economies that are stable or balanced without having to test them, or let the designer know that the economy is instable.

# Threats To Validity

This chapter is about the threats to the validity of this thesis. Some aspects of the research or the product have not been tested or are not based on scientific research.

## Design Decisions

Because this tool is the first visual interactable tool for Micro-Machinations that is based ob the Micro-Machinations library there was not a lot of previous research available to make informed design decisions. Most of the decisions were based on the Machinations tool by Joris Dormans and the MeDeA tool by Riemer van Rozen (Rozen, 2015). For functionality that could not be found in these examples, inspiration was found in other high-end software products like browsers and programming IDEs.

## Micro-Machinations Knowledge

This tool is designed to be an IDE for Micro-Machinations. It is not meant to teach the user how to use it. The user needs to have experience in using Machinations and Micro-Machinations to reflect game design. Because of this needed knowledge the tool might not expedite the design cycle for all game designers. Future improvements to the tool could make it more user friendly and help the user learn the Micro-Machinations language, or Micro-Machinations could be taught as a course to game development studios seeking to use this tool.

# Epilogue

This internship has been very insightful for me and it has tested the skills I have learned during my bachelor at the Amsterdam University of Applied Sciences. Creating a tool, a piece of software, instead of a game has been a new and unique experience for me. Keeping the user in mind instead of the player, making actions obvious and easy instead of gating them behind puzzles.

Not only has this internship been insightful in an academic sense but it has also made me realize that creating games is what I really want to do. While software development has its own challenges, I find it less fun and less rewarding.

Nevertheless, I think my tool has turned out great. There is a lot of potential in the LIVE Game Design project and I hope my tool can be used to help game studios stay competitive in the market.

I wish to thank everyone once more, and thank you for reading my thesis, I hope it was insightful and has provided the information you had hoped to acquire.

# Sources & References

Dormans, J. (2012). Engineering Emergence: Applied Theory for Game Design.

Klint, P., & Van Rozen, R. (2013). “Micro-Machinations: A DSL for Game Economies.” International Conference on Software Language Engineering. Springer International Publishing.

Van Rozen, R., & Dormans, J. (2014). “Adapting Game Mechanics with Micro-Machinations." Foundations of Digital Games. Society for the Advancement of the Science of Digital Games.

Van Rozen, R. (2015). "A Pattern-Based Game Mechanics Design Assistant." Foundations of Digital Games. Society for the Advancement of the Science of Digital Games.

Vaessen, T. (2017). “Entity Behavior Editor Money Maker Deluxe.” Firebrush Studios.

Van Rozen, R. (2017). “Live Game Design”. Unpublished draft.

# Attachments

## I LIVE Game Design Summary

For many Dutch game development studios the game design process is becoming a big problem during game development. Game design processes are systematically falling behind the hardware- and software technologies being used. For this reason, producing high quality games is becoming increasingly complex. As a result development costs increase. Game studios such as Game Oven, PlayLogic and Lunagames, which were producing successful titles, had to close down. If this trend continues the Netherlands will lose her innovative game studios one by one, and the long term plans the government has for this creative industry will vanish into thin air.

Game Design defines the content, inner workings and play experience of the game through the designing of game rules, levels, stories and mechanics. The cycles required to design, evaluate and adjust games take too long to achieve the expected high quality. Due to a lack of quick and clear feedback about the quality of the game design, designers often work uninformed causing game studios to work slower than required to stay competitive in the market. There is a giant gap between game design and the program code of the finished product. We call this the representation gap of game design.

Several game studios have asked the research group Play & Civic Media of the Amsterdam University of Applied Sciences (AUAS) to use its knowledge and knowhow to develop methods that can accelerate their game design processes. The research group proposes to work in the RAAK-MKB project “Live Intelligent Visual Environments for Game Design” together with twelve game studios, National Research Institute for Mathematics and Computer Science (CWI) and the Delft University of Technology (TU Delft) on powerful live game design tooling that can bridge the representation gap of game design and reduce the time required for design cycles by continually providing feedback about the quality of the final product. Over a period of two years powerful design notations, languages and tools will be developed that will enable designers to design game elements faster, more autonomous, more focused and improve the quality of the player experience. This will allow game studios to remain competitive and create opportunities for thematically specific and smaller projects, which are currently unavailable to them.

First, we will analyze what notations and types of feedback about designs currently missing for mechanics and stories, missions and training. Along with the CWI we will develop the required language technology that enables continuous changes. Then, the chosen visual notations, feedback and language technology will be developed into live game design tools. The AUAS will work together with the TU Delft, which has a lot of experience with the creation of efficient production tools for advanced and realistic (training-)simulations. Additional expertise on the field of design tooling is present at the University of Santa Cruz. Researchers from this institution will monitor the research. After that the tooling will be tested, validated, evaluated and improved during three case studies with game development studios. Dissemination and knowledge circulation are supported by the Dutch Game Association and Dutch Game Garden. The network and project results will be consolidated at the Amsterdam University of Applied Sciences by imbedding Create-IT and the Information Technology education.

## II Visual Definition Editor Specification of Micro-Machinations

Version: 0.01

Status: Draft

Author: Riemer van Rozen

### Element

Every element has a name. This means all nodes and edges have a name that can be modified in the element inspector. The ordering and naming of attributes coincides with the attribute names inside the meta-model and textual Micro-Machinations.

Most attributes are enum types, some are integers and others are strings.

### Node

All nodes show the following attributes. This means the node inspector has a uniform look.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **attribute** | **default** | **choices** | | | | | |
| io | internal | internal | in | out | inout |  |  |
| when | passive | passive | auto | user | start |  |  |
| act | pull | pull | push |  |  |  |  |
| how | any | any | all |  |  |  |  |
| name |  | String value name is unique inside its definition | | | | | |
| behavior | pool | pool | source | drain | gate | converter | reference |

### Pool

A pool is a node with Pool behavior.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **attribute** | **default** | **choices** | | | | | |
| io | internal | internal | ~~in~~ | ~~out~~ | ~~inout~~ |  |  |
| when | passive | passive | auto | user | start |  |  |
| act | pull | pull | push |  |  |  |  |
| how | any | any | all |  |  |  |  |
| name |  | String value name is unique inside its definition | | | | | |
| behavior | pool | pool | source | drain | gate | converter | reference |
| at | 0 | integer value indicates how many resources will be inside a pool instance when it is first created | | | | | |
| add |  | string value is an integer expression | | | | | |
| max | 0 | integer value is the maximum a pool can contain when instantiated. default zero means no maximum | | | | | |
| of type |  | string value is one of the definition type names. For each resource in the pool instance an instance of this definition type is created. empty string means no type. | | | | | |

### Gate

A gate is a node with gate behavior.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **attribute** | **default** | **choices** | | | | | |
| io | internal | internal | ~~in~~ | ~~out~~ | ~~inout~~ |  |  |
| when | passive | passive | auto | user | start |  |  |
| act | pull | pull | ~~push~~ | gates cannot push | | | |
| how | any | any | all |  |  |  |  |
| name |  | String value name is unique inside its definition | | | | | |
| behavior | gate | pool | source | drain | gate | converter | reference |

### Source

A source is a node with source behavior.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **attribute** | **default** | **choices** | | | | | |
| io | internal | internal | ~~in~~ | ~~out~~ | ~~inout~~ |  |  |
| when | passive | passive | auto | user | start |  |  |
| act | push | ~~pull~~ | push | sources cannot pull | | | |
| how | all | ~~any~~ | all | sources provide abundant resources | | | |
| name |  | String value name is unique inside its definition | | | | | |
| behavior | source | pool | source | drain | gate | converter | reference |

### Drain

A drain is a node with drain behavior

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **attribute** | **default** | **choices** | | | | | |
| io | internal | internal | ~~in~~ | ~~out~~ | ~~inout~~ |  |  |
| when | passive | passive | auto | user | start |  |  |
| act | pull | pull | ~~push~~ | drains cannot push | | | |
| how | any | any | all | drains pull any or all amounts | | | |
| name |  | String value name is unique inside its definition | | | | | |
| behavior | drain | pool | source | drain | gate | converter | reference |

### Reference

A reference is a node with reference behavior. Its does not behave itself, but instead is defined elsewhere. For now, only reference have in/out/inout modifiers. This may change!

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **attribute** | **default** | **choices** | | | | | |
| io | internal | internal | in | out | inout |  |  |
| when | passive | passive | ~~auto~~ | ~~user~~ | ~~start~~ |  |  |
| act | ~~pull~~ | ~~pull~~ | ~~push~~ | references neither push nor pull | | | |
| how | ~~any~~ | ~~any~~ | ~~all~~ | references neither push nor pull | | | |
| name |  | String value name is unique inside its definition | | | | | |
| behavior | reference | pool | source | drain | gate | converter | reference |

### Converter

A converter is a node with converter behavior. Converters always convert all resources on the incoming edges and produce all resources on the outgoing edges, otherwise they do not function.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **attribute** | **default** | **choices** | | | | | |
| io | internal | ~~internal~~ | ~~in~~ | ~~out~~ | ~~inout~~ |  |  |
| when | passive | passive | auto | user | start |  |  |
| act | pull | pull | ~~push~~ | converters only pull | | | |
| how | all | ~~any~~ | all | converters only pull all resources | | | |
| name |  | String value name is unique inside its definition | | | | | |
| behavior | converter | pool | source | drain | gate | converter | reference |

### Interface

An interface node is a special kind of node that cannot be edited directly. Instead it is created automatically when a (reference) node is exposed using its IO modifier.

An interface appears as a small circle on the edge of a definition type of a pool node with its name appearing near it being the same as that of the exposed node.

**Note:** I suggest starting populating the edge of the definition type on the left and putting new interfaces on the border going up and to the right. Alternatively, show the interface on the side where it was defined inside the definition, that may be more intuitive.

### Edge

There are two kinds of edges, flow and state edges. Unlike nodes, they cannot be replaced without explicitly adding and removing them.

Source and target names may be interface nodes. In that case the name is a qualified name, i.e. a name with a dot separating the two parts: pool\_name.interface\_name

|  |  |  |
| --- | --- | --- |
| **attribute** | **default** | **choices** |
| name |  | String value name is unique inside its definition |
| source |  | String value is the name of a node inside this definition |
| exp |  | String value expresses is an expression |
| target |  | String value is the name of a node inside this definition |

### Flow Edge

A flow edge appears as an uninterrupted arrow.

|  |  |  |
| --- | --- | --- |
| **attribute** | **default** | **choices** |
| name |  | String value name is unique inside its definition |
| source |  | String value is the name of a node inside this definition |
| exp | 1 | String value is an integer expression that indicates how many resources will be pulled or pushed by connected nodes. A missing value (empty string) indicates 1. |
| target |  | String value is the name of a node inside this definition |

### State Edge

A state edge appears as a dashed arrow. The meaning of a state edge depends on the value of ‘exp’.

|  |  |  |
| --- | --- | --- |
| **attribute** | **default** | **choices** |
| name |  | String value name is unique inside its definition |
| source |  | String value is the name of a node inside this definition |
| exp |  | A state edge is a **trigger edge** when its exp is \* or ! Here \* activates the target when a source node succeeds and ! activates the target otherwise.  A state edge is a **condition edge** has a boolean expression. If it evaluates to false the target node is deactivated during a step. Here an empty string value is not allowed. |
| target |  | String value is the name of a node inside this definition |

## III Micro-Machinations Grammar

@license{

Copyright (c) 2009-2015 CWI / HvA

All rights reserved. This program and the accompanying materials

are made available under the terms of the Eclipse Public License v1.0

which accompanies this distribution, and is available at

http://www.eclipse.org/legal/epl-v10.html

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

\* Micro-Machinations lang::mm::Syntax

\* @package lang::mm

\* @file Syntax.rsc

\* @brief Defines the syntax of Micro-Machiations

\* @contributor Riemer van Rozen - rozen@cwi.nl - HvA, CREATE-IT / CWI

\* @date February 5th 2015

\* @note Language: Rascal

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module lang::mm::Syntax

start syntax MM

= mach: Element\*;

syntax Element

= ctype: TID "{" Element\* "}"

| constr: "exp" "(" /\*{\*/NID /\*","}\*\*/ ")" "=\>" NID Monotonic

| absNode: When Act How "node" NID Pos

| source: When Act How "source" NID Pos //ACT --> always push

| drain: When Act How "drain" NID Pos //ACT --> always pull

| pool: When Act How "pool" NID At Add Max Category Pos

| ref: When Act How "ref" NID Pos

| converter: When Act How "converter" NID Pos

| gate: When Act How "gate" NID Pos

| flow: NID ":" NID "-" Exp "-\>" NID Pos

| state: NID ":" NID "." Exp ".\>" NID Pos;

syntax Monotonic

= nNone:

| mInc: ":" "monotonic" "+"

| mDec: ":" "monotonic" "-";

syntax Pos

= pos\_none:

| pos: "@(" VALUE "," VALUE ")";

syntax At //initial value of a pool

= at\_none: //default is zero

| at\_val: "at" VALUE;

syntax Category

= cat\_var:

| cat: "is" String;

syntax Add //modification expression of a pool

= add\_none: //default is no modification

| add\_exp: "add" Exp;

syntax Max //maximum value of a pool

= max\_none: //default is no upper bound

| max\_val: "max" VALUE;

syntax When //when does a node act?

= when\_passive: //default

| when\_passive: "passive"

| when\_user: "user"

| when\_auto: "auto"

| when\_start: "start"

| when\_extern: "extern";

syntax Act //what does a node do?

= act\_pull: //default

| act\_pull: "pull"

| act\_push: "push";

syntax How //how does a pool perform an act?

= how\_any: //default

| how\_any: "any"

| how\_all: "all";

syntax Exp

= e\_one:

| e\_val: VALUE //Value with optional unit of measurement

| @category="Value" e\_true: "true"

| @category="Value" e\_false: "false"

| e\_name: NID //{NID "."}+ //Name space query

| e\_override: "(" Exp ")" //Override priorities

| e\_active: "active" {NID "."}+

| e\_all: "all"

| e\_ref: "="

| e\_die: VALUE "dice" //Random number in {1..VALUE+1}

| e\_range: VALUE ".." VALUE //Random number in {VALUE..VALUE+1}

> e\_per: Exp "|" VALUE !>> Exp //amount that will flow after N iterations --> desugar to buffer

> e\_percent: Exp "%" //state (trigger from gate to node) percentage adds up to 100%

//flow percentage refers to the source

> e\_unm: "~" Exp //Arithmetic Negation Unary Expression

| e\_not: "!" Exp

> left

( left e\_mul: Exp "\*" Exp //Arithmetic Multiply Binary Expression

| left e\_div: Exp "/" Exp //Arithmetic Divide Binary Expression

)

> left

( left e\_add: Exp "+" Exp //Arithmetic Plus Binary Expression

| left e\_sub: Exp "-" Exp //Arithmetic Minus Binary Expression

)

> left

( left e\_lt: Exp "\<" Exp //Relational Less Than Binary Expression

| left e\_gt: Exp "\>" Exp //Relational Greater Than Binary Expression

| left e\_le: Exp "\<=" Exp //Relational Less-Equals Binary Expression

| left e\_ge: Exp "\>=" Exp //Relational Greater-Equals Binary Expression

| left e\_neq: Exp "!=" Exp //Relational Not-Equals Binary Expression

| left e\_eq: Exp "==" Exp //Relational Equals Binary Expression

)

> left e\_and: Exp "&&" Exp

> left e\_or: Exp "||" Exp

;

syntax String

= @category="String" "\"" STRING "\"";

syntax NID

= @category="Name" ID;

syntax TID

= @category="TypeName" ID;

syntax UID

= @category="UnitName" ID;

syntax ID

= id: NAME;

lexical VALUE

= @category="Value" ([0-9]+([.][0-9]+?)?);

lexical NAME

= ([a-zA-Z\_$] [a-zA-Z0-9\_$]\* !>> [a-zA-Z0-9\_$]) \ Keyword;

lexical STRING

= ![\"]\*;

layout LAYOUTLIST

= LAYOUT\* !>> [\t-\n \r \ ] !>> "//" !>> "/\*";

lexical LAYOUT

= Comment

| [\t-\n \r \ ];

lexical Comment

= @category="Comment" "/\*" (![\*] | [\*] !>> [/])\* "\*/"

| @category="Comment" "//" ![\n]\* [\n];

keyword Keyword

= "node" | "source" | "pool" | "drain" | "gate" | "converter" | "delay" | "assert" | "delete" |

| "of" | "from" | "to" | "add" | "at" | "min" | "max" | "is"

| "passive" | "user" | "auto" | "start"

| "push" | "pull" | "any" | "all"

| "true" | "false" | "dice" | "active"

| "ref" | "in" | "out" | "inout" |

| "step" | "violate"

| "pattern" | "monotonic" | "exp" | "intent" | "useWhen";

public Exp mm\_parse\_exp(str src) =

parse(#Exp, src);

public start[MM] mm\_parse(str src, loc file) =

parse(#start[MM], src, file);

public start[MM] mm\_parse(loc file) =

parse(#start[MM], file);

## IV Exported Micro-Machinations Diagram

Bank

{

in ref Returns @Bank[(119,250)]

out ref Loan @Bank[(300,250)]

pool Gold at 800 @Bank[(119,100)]

user gate LoanCreation @Bank[(300,100)]

push source Credit @Bank[(450,100)]

Credit -20-> LoanCreation @Bank[(459,102),(334,104)]

Gold -80-> LoanCreation @Bank[(129,101),(326,105)]

LoanCreation -100-> Loan @Bank[(332,108),(329,228)]

Returns -110-> Gold @Bank[(115,232),(118,110)]

}

Farm

{

in ref Loan @Farm[(80,100)]

auto gate ReturnPayment @Farm[(244,100)]

out ref Returns @Farm[(400,100)]

push source Profit @Farm[(244,270)]

Profit -10-> ReturnPayment @Farm[(233,256),(241,170)]

Loan -100-> ReturnPayment @Farm[(79,99),(243,97)]

ReturnPayment -110-> Returns @Farm[(249,103),(420,97)]

}

pool Banks of Bank at 1 @Global[(150,150)]

pool Farms of Farm at 5 @Global[(350,150)]

Banks.Loans.=.>Farms.Loans @Global[(176,195),(323,165)]

Farms.Returns.=.>Banks.Returns @Global[(176,195),(323,165)]