Taking level generation to the next level

Author: Nicky Wild
Institution: Amsterdam University of Applied Sciences
Company: Ludomotion
Advisors: Joris Dormans, Anders Bouwer, Stefan Leijnen
Taking level generation to the next level

Nicky R. D. Wild
500668836
1-11-2017 / 1-4-2018

Amsterdam University of Applied Sciences
Game Development, HBO-ICT
Advisors: Anders Bouwer, Stefan Leijnen

Ludomotion
Jaarbeursplein 6
3521 AL
+31 6 28 79 95 67
Advisor: Joris Dormans
# Table of contents

Abstract ........................................................................................................................................... 4

1. Introduction .................................................................................................................................. 5

2. Context ....................................................................................................................................... 6

2.1 Ludomotion .................................................................................................................................. 6

2.2 Assignment description .............................................................................................................. 6

2.3 Problem statement .................................................................................................................... 6

2.4 Research questions ................................................................................................................... 7

3. Methodology ............................................................................................................................... 8

3.1 Research methods .................................................................................................................... 8

3.1.1 Desk research ..................................................................................................................... 8

3.1.2 Primary research .................................................................................................................. 8

3.1.3 Evaluation through questionnaires ...................................................................................... 8

3.2 Version management ............................................................................................................... 9

3.3 Workspace ................................................................................................................................ 9

3.3.1 Ludoscope .......................................................................................................................... 9

3.3.2 Micro-machinations ............................................................................................................ 9

3.3.3 Neural network ................................................................................................................... 10

4. Related work ............................................................................................................................. 11

4.1 Tools for game design .............................................................................................................. 11

4.1.1 Mixed-initiative design tools ............................................................................................. 11

4.1.2 Ludoscope ....................................................................................................................... 13

4.1.3 Micro-machination diagrams ............................................................................................. 15

4.2 Automated generation for games ............................................................................................ 16

4.2.1 Neural networks ................................................................................................................ 17

4.2.2 Generating music .............................................................................................................. 17

5. Usability of Ludoscope .............................................................................................................. 18

5.1 Current state ........................................................................................................................... 18

5.2 Improving usability .................................................................................................................. 19

5.2.1 Documentation .................................................................................................................. 19

5.2.2 Functionality .................................................................................................................... 23
6. Supporting the creativity of the user with automated generation methods ........................................ 25
   6.1 Generating micro-machination diagrams of game dynamics ......................................................... 25
   6.2 Implementing deep learning ...................................................................................................... 26
7. Discussion ........................................................................................................................................... 29
   7.1 Limitations ................................................................................................................................. 29
      7.1.1 Ludoscope .......................................................................................................................... 29
      7.1.2 Micro-machination diagrams .............................................................................................. 29
   7.2 Future work ................................................................................................................................. 30
8. Conclusion .......................................................................................................................................... 31
9. Epilogue .............................................................................................................................................. 32
10. References ....................................................................................................................................... 33
Appendices ............................................................................................................................................. 34
    I. Questionnaires ............................................................................................................................... 34
    II. Ludoscope to Unity ..................................................................................................................... 50
    III. Ludoscope Android ................................................................................................................... 54
Abstract

The goal of this project was to figure out how to improve tools of game design in terms of usability and creativity. This project was part of the LIVE Game Design project at the Amsterdam University of Applied Sciences in collaboration with Ludomotion. We have mainly focused on two tools of game design in particular: Ludoscope and micro-machinations. Neural networks were also utilized to attempt generating micro-machination diagrams of a higher quality than randomly generated ones.

We start by exploring the context of the project, such as the company and research questions, as well as the research methods used during the project. We then define usability and creativity within the scope of this assignment. Armed with these definitions we assess the state of the tools with the help of interviews and questionnaires. The possible solutions and steps taken are described in detail and evaluated halfway through. New information arising from this evaluation is used to guide the next steps. Another section specifically about the implementation of deep learning on micro-machination diagrams describes the pipeline as currently envisioned. Current limitations and future work provide recommendations on following up on this project. Finally, we conclude to what degree we have reached our goal and why or why not this is the case.
1. Introduction

During my studies I have attended multiple lectures by Joris Dormans about automated game design, and as a game developer who is primarily a programmer this has always been an intriguing subject for me. During workshops I have been introduced to Ludoscope and the concept blew me away. These workshops were quite basic, but the potential was clear, and it bothered me that there was no easy way of integrating this tool into my workspace yet. I would later, with the help of Stefan Leijnen, dig deeper into automated game design and procedural content generation using neural networks at his company, The Asimov Institute. As Stefan did not offer internships this year, he introduced me to the Live Game Design project.

After the first few meetings with Anders Bouwer and Riemer van Rozen the decision was made to make the Live Game Design project my internship. This would be at the Amsterdam University of Applied Sciences in conjunction with another company. There were a couple of options, but as soon as I heard Joris Dormans’ company, Ludomotion was an option, I was sold. An internship where I would have the chance to work on Ludoscope and in combination with an ambiguous plan to experiment with neural networks on procedural content generation is about as good as it gets.

This thesis explores how tools for game design can be improved. The learning curve and lack of documentation for Ludoscope currently prevent wider adoption of the tool. Furthermore, we explore the possibilities of applying neural networks to micro-machination diagrams.

Chapter 2 of this thesis explains the context of the project, this includes information about Joris Dormans’ company, Ludomotion, an assignment description, a problem statement and the research questions. Chapter 3 investigates the research methods used during this project. Chapter 4 discusses related work in the fields of automated game design and neural networks. Chapter 5 explores Ludoscope’s shortcomings before my work at Ludomotion as well as the steps I have taken to improve the tool. In chapter 6 we look at how the creativity of a human designer can be supported with automated generation methods. Chapter 7 discusses the limitations still in place as well as what potential steps can be taken in the future. And finally, we draw our conclusions in chapter 8.
2. Context

This chapter describes the context of the project and research for this thesis, as well as providing background information about Ludomotion. Furthermore, it contains a description of the project, a problem statement and the research questions to be answered.

2.1 Ludomotion

Ludomotion is an indie game company founded by Joris Dormans, located in the Netherlands. They have worked on games such as SUMICO and Unexplored, and a mixed initiative design tool named Ludoscope, which is also being used in the development of Unexplored 2. SUMICO and Unexplored have both been released on Steam, while SUMICO has also been released for the Nintendo 3DS and Unexplored will be released for the Nintendo Switch this year. These games all heavily rely on procedural content generation. Ludomotion operates worldwide as their games have been published globally on platforms such as a Steam.

2.2 Assignment description

My task for this assignment is improving the usability and creativity of tools for game design. For this thesis, the focus lies on Ludoscope and micro-machinations. Our initial plan was to work on improving Ludoscope as well as a smaller side-project in which we would experiment with the generation of micro-machination diagrams using neural networks. This side-project quickly grew into a larger portion of the work, as it was very time consuming. This started to cut into the time I could spend working on Ludoscope, but eventually we found a way to combine the work by making a micro-machination diagram generator in Ludoscope.

2.3 Problem statement

Ludoscope is a tool that is tough to grasp conceptually, and the lack of tutorials and documentation make it hard for new users to adopt. The objective of this research is to improve the tool on usability and by supporting the creativity of the user with the help of generated micro-machination diagrams. Neural networks play a part in improving the generation of micro-machination diagrams.
2.4 Research questions

The main research question is as follows: How can tools for Game Design be improved in terms of usability and creativity?

To answer this research question, usability and creativity must be defined within the scope of this assignment. Usability encompasses the accessibility of the tool, the user’s ability to utilize the functions of the tool, and the addition of functions to increase the efficiency of specific tasks within the tool. Creativity encompasses the use of imagination or original ideas to create or invent something.

The following sub-questions are addressed to help answer this research question:

1. What obstacles do new users of Ludoscope currently face in terms of usability?
2. How can the usability of Ludoscope be improved?
3. How can the creativity of the user be supported by automated generation methods?
4. Have the changes to Ludoscope led to a more accessible and usable product?
3. Methodology

This chapter elaborates on the research methods used in this thesis. This includes the work previously done on the micro-machination diagram generator in Rascal and documentation on Ludoscope. Furthermore, the version management and workspaces are explained.

3.1 Research methods

There are a couple of different research methods used throughout the course of my work at Ludomotion.

3.1.1 Desk research

Because of the ambitious nature of this project, I had to get more familiar with everything regarding this space. My advantage was being somewhat familiar with micro-machinations, having very recently done work on neural networks and having a large interest in Ludoscope. Before I could really get to work I needed to set up a workspace for Ludoscope and the micro-machinations diagram generator, repeat a previous experiment regarding micro-machinations and neural networks, and dive into the documentation and code of Ludoscope. This was made possible by working closely with experts on these fields: Riemer van Rozen, Joris Dormans, Anders Bouwer and Stefan Leijnen.

3.1.2 Primary research

After the foundation had been laid, I had set up a couple of interview questions. The aim of these interviews was to identify the gaps in terms of usability regarding Ludoscope, so they could be addressed in an effective manner. After taking a couple of interviews and taking my own experience with Ludoscope in account, I had a good starting point for tackling these issues. Questionnaires were used to measure the gaps in terms of usability regarding Ludoscope with a new, wider audience. Chapter 5 goes deeper into my findings and how they were addressed.

3.1.3 Evaluation through questionnaires

The aim of these questionnaires was not only to measure the gaps in terms of usability regarding Ludoscope, but also whether some had been addressed. These findings led to the creation of more tutorials on demand and allow users to integrate Ludoscope into Unity as well as using the tool on Android. Due to time constraints these new tutorials have not been evaluated.
3.2 Version management

For version management of micro-machinations and Ludoscope, GitHub was used. The tool Ludoscope is dependent on Phantom, an entity component-based game engine for XNA 4.0 and Phantom Grammar, which is the code base for the transformational grammar extension to Phantom. The micro-machinations diagram generator in Rascal did not yet have a GitHub repository when this project started. The setup for the recurrent neural network did not have a version management as the training results were meant as proof of concept rather than a product to work with in the future.

3.3 Workspace

As this project required multiple operating systems for all setups, the workspaces have been divided in the three most important tools used.

3.3.1 Ludoscope

Ludoscope, as of writing this thesis, requires the XNA 4.0 framework to run. This means Ludoscope will only run on a Windows computer and can run into compatibility issues with Windows 10. Despite this, I got Ludoscope to work on a Windows 10 dual-boot setup. The second problem I ran into was compiling Ludoscope, as the XNA 4.0 plugin is not supported on newer versions of Visual Studio. After extracting the installer and making some tweaks to the installation path, I managed to trick the XNA 4.0 plugin to properly install on Visual Studio 2017.

Most of this hassle can be circumvented by running Windows 7 and an older version of Visual Studio, but as this was the workspace I had previously set up and knowing it was possible to get it to work in this workspace, I was adamant on doing so.

3.3.2 Micro-machinations

Even something as seemingly simple as running an existing Rascal project in Eclipse with a plugin turned out to be tougher to set up than expected. The Rascal plugin required a very specific version of Eclipse as well as the Java Runtime Environment. After going through the entire setup, the code still would not compile. As it turned out the Java version I used was a few releases too new, which is something we found out when I followed the same process on Mac, minus the Java installation, and the code did compile. Since I ran both operating systems off the same computer, I decided to not mess with the setups and just run them in different operating systems.
3.3.3 Neural network

Despite the setup pains thus far, getting the neural network running was the biggest challenge. First, Karpathy's recurrent neural network requires Torch\(^1\), a scientific computing framework for machine learning algorithms. Torch runs on both MacOS and Ubuntu and works best with a GPU. As my MacBook did not have a dedicated graphics card and I could not afford to leave it an unusable state for days on end while the network was training, I needed a second computer with a dedicated NVIDIA graphics card. I installed Ubuntu 16.04 LTS, Torch and all other prerequisites on an older laptop with a GTX670M which was doing nothing but collect dust. I also needed the CUDA toolkit to train on the GPU as opposed to the CPU as this can be about a factor of 10x faster \([4]\). After eventually getting the CUDA toolkit and the required Torch packages to work, the neural network would still reject my GPU. As it turned out, the GTX670M is an overclocked GTX570M and the only one in the series that is running on an older architecture, which left me with an incompatible version of CUDA. With no alternative, the setup ultimately consisted of running this laptop on over the weekends while training on the CPU.

\(^1\) “Torch” [http://torch.ch/](http://torch.ch/)
4. Related work

In this chapter we discuss related work in this field with the intention of introducing the concepts required to understand the work presented in this thesis. This chapter is split up in two paragraphs as it covers both the tools for game design and the automated generation aspect.

4.1 Tools for game design

There are a multitude of tools for game design, think of tools that are used to create content such as art, sound effects, levels and game mechanics. Some of these tools are used to assist while others are used to automate the task. A simple tool such as ‘make 8-bit art’ does nothing more than provide the user with an interface to draw pixel art, whereas some other tools such as Bfxr allow the user to generate random sound effects and tweak them afterwards, as well as building a sound from scratch. Tools like these are often used to save time while some are used to stretch or explore the design space.

In this paper we will be looking at two tools; Ludoscope and micro-machinations.

4.1.1 Mixed-initiative design tools

While the quality of procedurally generated game levels is generally considered high enough, it is not yet on par with game levels designed by a human designer. This is where mixed-initiative comes in, mixed-initiative is an approach where a designer can manipulate a level generator and choose from different level configurations. This helps speed up the design process while keeping in the human element in designing game levels [3].

In some cases, the human designer takes the lion’s share of the initiative and in other cases it’s the computer. With computer-aided design the human designer is the director of the creative process and with interactive evolution it’s the computer who comes with new ideas. [6] An example of computer-aided design is Sentient Sketchbook [5], a tool which assists human designers in designing levels. Sentient Sketchbook provides feedback and autonomously suggests modifications to be made to the level in real-time. Galactic Arms Race [2], a space shooter with procedurally generated weapon projectiles, is an example of a game using interactive evolution. The game takes into account how much you use a weapon as a measurement of whether you enjoy the weapon or not. The weapons are evolved via neuroevolution of augmenting topologies to adapt to user preference.

2 “Make 8-bit art” https://make8bitart.com/
3 “Bfxr” https://www.bfxr.net/
Figure 1: The user interface of Sentient Sketchbook. A human designer edits their sketch (left) and a generator, acting as the artificial designer, creates map suggestions in response (right). Adapted from [5]

Figure 2: Galactic Arms Race with multiple players using different weapons. Adapted from [2]
4.1.2 Ludoscope

Ludoscope can generate levels in conjunction with a human designer by using a computer to systematically apply handcrafted rules for transformations to given inputs and allowing the designer to make tweaks along the way to solve potential problems. Ludoscope is used for progression design, the structure of missions and the structure of the game world. Ludoscope uses formal grammars and rewrite rules for control and communication between the designer and the generation system as well as to specify and shape the constraints used to generate content.

Below is an example of a simple 2D platformer level. It is generated by going from the input module to an actual level using grammars. In this case we start with a basic tile map which will be expanded and filled in with a couple of predefined level chunks. For more complexity you can use grammar rules within these predefined level chunks, but for this example we will not dive too deeply into this.

Figure 3: Completed output the Unity platformer prototype
For example, this is what the grammar used to generate a level chunk from 'medium difficulty' tiles looks like:

![Figure 4: Grammar used to generate a level chunk from medium difficulty tiles](image)

In the left hand we look for a 10x10 chunk of medium tiles to apply this grammar rule to. The 1x10 row of not medium tiles on the left is used for alignment; there’s always another chunk to the left of difficulty tiles, either another difficulty or the level start chunk. We want to make sure we line up with those tiles to avoid generating the chunk in the wrong place. A less specific left hand can be used to make smaller changes to existing level chunk for variation, combining this with more right hands can quickly generate a nearly endless variety of level chunks. This makes Ludoscope a very powerful tool for level generation.
4.1.3 Micro-machinations

Micro-machinations is a visual programming language for game dynamics and is based on machinations. Machinations is a visual language for creating diagrams and simulating those diagrams. Machinations diagrams are designed to represent game mechanics in an accessible way without compromising the structural features and dynamic behavior of the games they represent but is limited to design itself [1]. Micro-machinations is a more concrete version of machinations and better suited for implementations. Intended gameplay and the gameplay emerging in a game during playtests often differ and adjusting this is difficult because designers lack the means for defining and balancing game mechanics. Micro-machinations allow a designer to do just that, resulting in increased adaptability and reduced time between design iterations [8].

---

![Diagram of micro-machinations elements](image)

**Figure 5:** Visual micro-machinations of basic elements. Adapted from [8]
4.2 Automated generation for games

As creating content takes a lot of time and effort, automated game design can speed things up quite a bit for a designer. One method of automated generation is procedural content generation. Procedural content generation is computer software that can create game content. Content in this context is defined as most of what is contained in a game; think of levels, game rules, textures, items, quests, characters etc. Artificial intelligence is not considered content and the game engine is omitted from this definition as well. There are multiple reasons to use procedural content generation, two of the most obvious reasons tie in well within the scope of this paper. The first is that generating content removes the need for a human designer or in the case of a mixed-initiative at the very least speeds up the process of creating content. The second reason for using procedural content generation is that intelligent and embedded design tools can enhance the creativity of human content creators. Procedural content generation allows a smaller team to create a larger game. Take for example Minecraft, a massively popular game which extensively uses procedural content generation techniques to generate unique worlds with content ranging from mountains to caves [9].

Figure 6: Minecraft; a game of a large scale originally made by a single developer
4.2.1 Neural networks

Another method of automated generation is the usage of neural networks. There are many different types of neural networks such as a Feed Forward Network, which is a very general approach and is guaranteed to solve the problem but cannot handle large data very well. Convolutional Neural Networks now outperform humans at image recognition [7] but require a lot of data and without enough data samples the pattern cannot be confirmed. And Recurrent Neural Networks, which are very effective at learning from and generating short sequential data but are notoriously sensitive to how the parameters of the model that get optimized are chosen. For this assignment the latter has been used, more specifically ‘char-rnn’ by Andrej Karpathy [4].

4.2.2 Generating music

Some tools, like Bfkr, are specifically made to randomly generate audio and does a good job of it. Other tools, such as neural networks, are not made for a specific task, but can be trained to become better at a specific task. Google’s own neural network for creating art and music, Magenta⁴, has come a long way in the past years and has about a dozen demos on their website. Curious about the possibilities of neural networks in the music field, I have trained Karpathy’s char-rnn on Famitracker⁵ files in text format with mixed results. While neural networks may not be replacing humans when it comes to generating music anytime soon, they can provide inspiration and potentially save time with a mixed-initiative approach.

⁴ “Google Magenta” https://magenta.tensorflow.org
⁵ “Famitracker” http://famitracker.com/
5. Usability of Ludoscope

In this chapter the usability of Ludoscope is explored. As previously stated, usability in the scope of this paper encompasses the accessibility of the tool, the user's ability to utilize the functions of the tool, and the addition of functions to increase the efficiency of specific tasks within the tool. This chapter is split up in two parts; first we gauge the current state of the tool, the documentation surrounding the tool, and how people experience the tool. Then we proceed to look at possible solutions for those current issues and limitations, which will be divided into documentation and the added functionality.

5.1 Current state

Ludoscope is a rather unique tool and because of this it can be hard to wrap your head around the fundamentals, most notably grammars. With grammars being a composition of rules that always result in grammatically correct sentences, you can guarantee that the objects generated by these grammars are also always correct. In this case the elements of the objects can be seen as words. In Ludoscope a designer can create different types of grammars such as string grammars, graph grammars and tile grammars. The examples in figure 3 and 4 use tile grammars, while chapter 6 gives examples of graph grammars. In short, a tool such as this requires an introduction of sort, but currently the documentation and tutorials are either outdated or severely lacking. To answer what obstacles users of Ludoscope face, we have interviewed existing users about their first impressions and current experience. Furthermore, we have gathered user data from first time users with a questionnaire during the Advanced Game Design course, and lastly, I have included my own personal experience with the tool, ranging from my first impressions from when the tool was introduced to me to more advanced tasks as I progressed through this project.

The most common obstacles for new users is understanding the tool on a conceptual level and for some it was getting the tool to run at all. Most users did find it relatively easy to generate some content with the platformer tutorial, but the lack of more advanced tutorials and an easy way of integrating these techniques into their own projects results in users quickly dropping the tool. The fact Ludoscope still depends on the XNA framework to run, which is no longer supported and does not officially support Windows 10, hurts the accessibility of the tool as well. The results of the questionnaire can be found in the appendices.
5.2 Improving usability

Usability has been tackled in two different ways. The first was to add more documentation such as tutorials, as well as updating older ones, to introduce the user to new concepts and allow them the freedom to play around with them. The second way was by adding functionality to make the tool more efficient to use. Joris had requested a couple of basic functionalities to be added to the tool as a way for me to get to know the tool better. This knowledge would have been required for potential future tasks of a higher difficulty, such as optimizing the tool for faster execution of grammar rules and getting rid of the XNA dependencies. As it turned out, the work on the micromachination diagram generator and neural networks proved to be more demanding than we had anticipated. So, the depth of this second approach was limited due to time constraints and has been discussed in the future work section.

5.2.1 Documentation

When a user is first introduced to Ludoscope the biggest obstacle is understanding the tool conceptually, this can be partly attributed to the lack of up-to-date documentation. Only an outdated tutorial on using Ludoscope to design levels for a platformer existed alongside an outdated manual. For this reason, we updated platformer tutorial and Ludoscope project to the latest version of Ludoscope. The main obstacles of this was that in Ludoscope version 0.5 every module had an alphabet, while Ludoscope version 0.6 stores the alphabet in a separate module, and that input modules are handled completely differently now. Once the differences between the versions of Ludoscope were clear, we had to extract the alphabets from the modules and combine them in a standalone module and find a way to handle the inputs. We ended up using a randomized input with the ability of making modifications to the input before moving on to the next module.

Figure 7: Ludoscope input module with a start, an end and randomize difficulty level chunks
In this basic example a tile map with a start, end and obstacles of random difficulty in between them are transformed into chunks of levels, making up a randomly generated level based on chunks designed by a human. The designer needs to be sure that either all chunks are compatible with each other or that there are fallback procedures in place to ensure incompatible level chunks are either not used together or made compatible on the spot. If such a situation still occurs, the designer has the ability to manually edit the incompatible parts or run the generator again and still save plenty of time by designing levels this way.

But how did we get from a few colored tiles to a playable level in Unity? The first step is to resize the input to the dimensions used in the game. In this case, every chunk is 10x10 tiles.

Figure 8: A module using a recipe to split the tiles

Now we have the proper size we want to work with, we can start looking for the chunks of 10x10 “start, end, easy, medium and hard” tiles and apply grammar rules to them. This is what the result looks like in Ludoscope after the generation process before loading it into Unity:

Figure 9: Completed output of figure 7 in Ludoscope
After users get used to the basic idea, they might want to integrate Ludoscope into their own Unity projects. We have set up an example project on how to achieve this so users can dive right into applying their newly learned techniques into their own projects, along with a step by step tutorial. In this tutorial we use a tile map just like in the platformer, so the user should feel comfortable with this already. Furthermore, we have gone deeper into the possibilities Ludoscope offers by looking at some more advanced techniques such as containers and optimizing the generation process. In this example, we have gone with a 2D top-down perspective, as the user already knows how to make 2D platformer levels. They are quite similar but different enough to keep the user engaged with the tutorials. The goal of these tutorials is to provide a better conceptual understanding of the tool and to familiarize the user with its interface.

![Figure 10: Ludoscope grammar example with a container on the enemy tiles](image)

In this case, hard difficulty tiles have enemies in them. However, if the enemy were to move it would leave a void at its original location, because it is an enemy tile without ground underneath it. This is where containers come in handy; containers show up as a tile with a black triangle in the top left corner to indicate there is more content than the one visible in the top layer. By double clicking the tile you will open the 'Edit Symbol' menu where you can see all the objects in the container. As the name implies, you can also use this menu to make changes to the current symbol you are editing. An example of this could be altering the contents of the tile or cleaning the container.
Since this enemy tile contains a ground tile, this enemy would not leave a void in its place if it were to move from its current location. This technique can also be used to decorate tiles with for example foliage or sound effects, as well as to add interactable objects such as a treasure chest. Interesting to note is that containers can also have a container, so you can add a much depth as you deem necessary. For this you need to recursively read out in the containers in Unity and instantiate the game objects in the right place. This is a code snippet of the recursion from the Ludoscope to Unity tutorial, the full tutorial can be found in the appendices:

```csharp
internal void buildContainer(Expression expression, Symbol symbol, Vector3 position) {
    switch (symbol.label) {
    case "ground":
        Instantiate(ground, position, Quaternion.identity);
        break;
    case "wall":
        Instantiate(wall, position, Quaternion.identity);
        break;
    case "spike":
        Instantiate(spike, position, Quaternion.identity);
        break;
    case "enemy":
        Instantiate(enemy, position, Quaternion.identity);
        break;
    case "pit":
        Instantiate(pit, position, Quaternion.identity);
        break;
    }
    if (symbol.container != null) {
        symbol = symbol.container;
        buildContainer(expression, symbol, position);
    }
}
```

Figure 12: Recursively building the content of the containers

With the Ludoscope to Unity tutorial you can use Ludoscope for Unity projects that run on Windows. We have written a third tutorial on porting the Ludoscope Unity projects to Android. The process for getting Ludoscope to work on Android is not too complex once you understand how streaming assets are handled and this process is not limited to Ludoscope itself, but it is a critical step to keep newer users of Ludoscope engaged. The full tutorial about streaming Ludoscope assets to Android can be found in the appendices.
5.2.2 Functionality

One of the basic functionalities Joris had requested to be added to Ludoscope as a way for me to get to know the tool better was the option to import and export parameters. The parameters contain all the registers, as well as the seed used in generation. The purpose of this was to enable the designer to generate the exact same level as before as a way of troubleshooting incorrectly generated levels, as well as saving a desired configuration. For example; if a user really liked the level that was generated for them, they could press a “favorite” button to export the parameters of that level. The user could then load that same exact level again using the same parameters. This functionality can be used to both benefit the player as well as the designer.

```
version: 0.6f
random: testseed
register: width 9
register: height 5
register: requests ["siteOnRoad", "restingPlace"]
register: setRequests true
register: terrainA "forest"
register: terrainB "forest"
register: terrainC "bushes"
register: terrainD "bushes"
register: terrainE "tallForest"
register: terrainF "bushes"
```

Figure 13: Export of parameters of an example project, formatted for readability

Another basic functionality Joris had requested was the ability to find words in your project files and to replace them. This could be used to for example to quickly and globally rename objects in your alphabet and grammars, to fix a spelling error, or to find where a specific word is used. This search can be narrowed down to grammars, recipes and the alphabet or any combination of them. There is also an option for case sensitive search as well as an option to make a backup of your changes in case you wish to undo the renaming.
Figure 14: Search output for 'terrainA' in all file types
6. Supporting the creativity of the user with automated generation methods

In this chapter methods for supporting the creativity of human designers have been explored. As previously stated, creativity per definition encompasses the use of imagination or original ideas to create or invent something. To support creativity in this regard, we would either need to enhance the imagination of the user or present the user with different original ideas, for this paper we focus on the latter. The automated generation methods we will discuss are micro-machinations and the implementation of deep learning.

6.1 Generating micro-machination diagrams of game dynamics

For this part of the assignment, some previous work had already been done. A micro-machination diagram generator written in Rascal randomly generated a variable amount of micro-machination diagrams in text format with a variable number of nodes and edges. These diagrams were then tested for a property and sorted, in this case it was whether the diagrams were connected or not. This way we could feed the correct diagrams to a recurrent neural network and train the network on them in the hope of it learning to be a better generator than the original one written in Rascal. Below is one of the micro-machination diagrams by the generator in Rascal. The diagram has two nodes, two edges, and randomly assigned properties:

```ruby
diagram{
  e_node(t_drain(),when_auto(),act_pull(),how_all(),id("n0"),cat_var()),
  e_edge(t_state(),id("e0"),id("n1"),e_die(9),id("n9")),
  e_node(t_converter(),when_user(),act_push(),how_all(),id("n1"),cat_var()),
  e_edge(t_condition(),id("e1"),id("n0"),e_die(2),id("n1"))
}
```

*Figure 15: Micro-machination diagram by the generator in Rascal, formatted for readability*

After repeating the experiment, we decided to start by refactoring the generator written in Rascal with the intention of expanding it to check for a more interesting property than connectedness, but we opted for creating a new micro-machination diagram generator from scratch in Ludoscope instead. Ludoscope offers more control over the generation process using grammars which could contribute to a higher percentage of correct diagrams and overall higher quality diagrams. Furthermore, Ludoscope offers a visual representation of the diagram which can offer the designer more insight on a surface level.
To keep the new generator compatible with the current pipeline, we have written a small parser to turn the micro-machination diagrams generated by Ludoscope into the same text format as the original generator written in Rascal, see figure 15. The parser simply reads out the nodes and edges, and fills in hardcoded strings. A more elegant solution could have been made, but as it is yet unknown whether we will keep the current layout, we have decided that it would do as a proof of concept. These text-based micro-machination diagrams can be fed to a neural network to see if deep learning could improve on the generator made in Ludoscope, just as with the generator written in Rascal. As both generators use the same format, the neural network could use diagrams from either one interchangeably.

6.2 Implementing deep learning

The recurrent neural network of our choice uses Ubuntu and the Torch framework, an open-source machine learning library. Training the network on a GPU, which is roughly about a factor of 10x faster than using the CPU, requires an NVIDIA GPU, the CUDA toolkit and some additional Torch packages. Then the neural network is ready to be fed a text file for it to train on. The time it takes the network to train depends on the size of your dataset and the speed of your GPU. After the network has finished training, it can sample a new text file for us to evaluate.

The idea is to train the network on micro-machination diagrams with interesting properties in the hope that the network will generate new, higher quality diagrams with the desired interesting properties. The end goal is to improve on the current generators written in Rascal and made in Ludoscope. These generators currently solely exist to create the initial dataset for the network to train on, as manually constructing a dataset of this size would not be feasible. A system to evaluate the performance of the network is not yet in place, which currently prevents us from drawing conclusions on the success of this approach.
Figure 17: Current pipeline as envisioned

The micro-machinations interpreter, aptly named the Oracle, is meant to evaluate the quality of the micro-machination diagrams generated by the neural network. Currently the diagrams only have the property connectedness, which is not interesting or complex enough to require the Oracle's intervention.
As can be seen in figure 18, the neural network starts from nothing and makes a lot of mistakes at first. After a couple of lines of seemingly random characters and non-existent words, the network starts to generate seemingly correct diagrams. Priming the network by feeding it a word or a phrase before generating can help it warm up faster. This is especially helpful with smaller outputs but does not seem to be as important for larger outputs as it will correct itself quickly enough.
7. Discussion

This chapter will discuss the current limitations as well as what steps could be taken in the future in more depth. These are limitations we were either unable to solve in our time-frame or fell out of our scope but make a good starting point for the continuation of this subject.

7.1 Limitations

The limitations have been split up over the two tools, as they both could be continued as stand-alone work.

7.1.1 Ludoscope

Ludoscope currently has a couple of limitations which make it a tough tool to use, which is quite disappointing to users as it is a very powerful and useful tool. The fact Ludoscope still depends on XNA libraries makes it a little complicated to install on Windows 10 since XNA is no longer supported and only officially supports up to Windows 7. While the tool does generally run on Windows 10, a couple of new users in the Advanced Game Design class reported that they were unable to run the tool even after installing the required dependencies. Furthermore, XNA prevents users from running the tool on a Mac or Linux based system.

A large portion of users who did get Ludoscope to run reported being repelled by the interface appearing overwhelming, confusing and outdated. While the tool does not need to look good to get the job done, the overwhelming and confusion part are problems that need to be addressed for new users to properly adopt the tool. As new users were able to get basic tasks done with the aid of an extensive tutorial with images, the confusion may also be due to a lack of proper documentation.

7.1.2 Micro-machination diagrams

We are currently unable to evaluate the quality of the micro-machination diagrams, which is the biggest obstacle for the work done so far. Furthermore, the micro-machination diagram generator in Ludoscope does not yet generate anything other than random straight diagrams, which do not yet have an interesting property to check for. The foundation has been laid for the continuation of this approach.
7.2 Future work

For Ludoscope we suggest the following tasks: Writing out the XNA dependencies in favor for something that would also run on Mac and Linux based systems to make the tool a lot more accessible and easier to integrate into a workspace. As for the usability, we suggest updating the manual to the latest version and writing a couple more tutorials on different kinds of grammars to tackle Ludoscope on a more conceptual level to provide new users with a broader understanding of the tool and its features. This should also eliminate some of the confusion new users face using Ludoscope and allow them to generate more complex content.

As for the micro-machinations and deep learning aspect of this assignment we suggest further fleshing out the pipeline to generate more interesting micro-machination diagrams and evaluating dynamic properties. Using deep learning to predict these dynamic properties is arguably the most interesting and challenging for future work.
8. Conclusion

The goal of this project was to figure out how to improve tools of game design in terms of usability and creativity. For Ludoscope this meant researching what obstacles users face by interviewing a couple of them. A follow-up questionnaire was filled out by new users after rewriting a tutorial to gauge the effectiveness of this, as well as to find other potential problems users may face. For micro-machinations this meant repeating a previous experiment: randomly generating micro-machination diagrams, feeding them to a recurrent neural network and evaluating its performance. This with the goal of training the neural network to generate micro-machination diagrams of higher quality than random.

When looking at Ludoscope specifically, I have established the obstacles new users face and the usability has been improved by my addition of the basic features as well as by my rewritten and newly written documentation and tutorials. As for supporting the creativity of the user by automated generation methods, I have laid a foundation in the form of a micro-machination diagram generator and exporter integrated in Ludoscope. To truly attain this goal for Ludoscope, as well as for micro-machinations, the pipeline as described in future work needs to be further developed.

When looking at micro-machinations specifically, I have made strides improving the usability by making a micro-machination diagram generator in Ludoscope. This generator gives the user a visual representation of the diagrams, but it is not a fully-fledged generator yet. As for supporting the creativity of the user, the proof of concept I ran with the recurrent neural network looks promising, but without the full pipeline in place, these results cannot yet be properly evaluated.
9. Epilogue

This internship turned out to be more ambitious than we had previously anticipated and has challenged my skills as a researcher, programmer and designer, as well as my perseverance and flexibility. Working with and for Joris Dormans on Ludoscope is something I would never have imagined possible when I first attended his lectures in class. Documenting everything I have learned about Ludoscope in the form of tutorials hopefully means that the tool will be more accessible to students, which is something I would have been very happy with when I was first introduced to the Ludoscope. Despite my previous work with neural networks and Karpathy’s recurrent neural network in particular, combining it with micro-machinations proved to be a huge challenge and I hope to have contributed to the leviathan of a project that it turned into.

As much fun as I had learning all these new things, my passion remains developing games and I will likely leave the development of these tools up to other passionate creative minds. As a developer I am very excited to see what kind of tools the future brings and how this will impact the development of games.
10. References


Appendices

I. Questionnaires

---

Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?

Seems to have potential

How much were you able to do with Ludoscope?

Grasp the general concept
Follow along example in class

What would you like to do with Ludoscope that you don’t know how to do yet?

Use in Unity

What were the biggest obstacles for you to effectively use Ludoscope?

Interface not really intuitive

Would you continue to use Ludoscope in its current state? Why / Why not?

Perhaps

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?

- It looks complicated

How much were you able to do with Ludoscope?

- Use the tutorials in the class

What would you like to do with Ludoscope that you don’t know how to do yet?

- Use it for our assignment

What were the biggest obstacles for you to effectively use Ludoscope?

- The UI isn’t very intuitive

Would you continue to use Ludoscope in its current state? Why / Why not?

- I will have to compare it to other programs first

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?

Interesting, but outdated

How much were you able to do with Ludoscope?

For now I only did what the teacher showed us

What would you like to do with Ludoscope that you don’t know how to do yet?

I hope I can implement it in my own game but I only experienced Ludoscope for a few minutes

What were the biggest obstacles for you to effectively use Ludoscope?

The UI, it is pretty outdated and not very clear

Would you continue to use Ludoscope in its current state? Why / Why not?

No, the UI is very annoying and that really kills it for me

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?
It looks confusing if you have no experience in how to use it.

How much were you able to do with Ludoscope?
Tweak/Change a little of an existing grammar example

What would you like to do with Ludoscope that you don’t know how to do yet?
Export it to Unity so I can use custom predicates to create what was designed in Ludoscope

What were the biggest obstacles for you to effectively use Ludoscope?
It is a bit unclear in what it does and where to find specific screens/interfaces.

Would you continue to use Ludoscope in its current state? Why/Why not?
Probably not because the current UI makes it difficult to learn and use.

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?
Looked old-fashioned. Maybe
needs some time to get into

How much were you able to do with Ludoscope?
Not much. First things were just
clicking buttons and seeing how things happen or not

What would you like to do with Ludoscope that you don't know how to do yet?
Since I don't know much about possibilities then maybe everything

What were the biggest obstacles for you to effectively use Ludoscope?
I don't know

Would you continue to use Ludoscope in its current state? Why / Why not?
Maybe, need some digging on possibilities?

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?
- The software looks a bit old
- The UI is a bit out-dated

How much were you able to do with Ludoscope?
- Tweak some grammar parts

What would you like to do with Ludoscope that you don't know how to do yet?
- Implement a file in Unity

What were the biggest obstacles for you to effectively use Ludoscope?
- The UI

Would you continue to use Ludoscope in its current state? Why / Why not?
- I don't know. I have to overcome the UI

Thank you for your time.
Questionnaire Ludoscope

What were your first impressions of Ludoscope?
Interesting work, clunky UI

How much were you able to do with Ludoscope?
Little (without studying)

What would you like to do with Ludoscope that you don't know how to do yet?
Build some binding to an existing engine to plug into

What were the biggest obstacles for you to effectively use Ludoscope?
Dependencies, dependencies

Would you continue to use Ludoscope in its current state? Why / Why not?
No, too clunky. Also, don't see a need unless lazy/uninspired

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?

Forced

What were you able to do with Ludoscope?

None

What would you like to do with Ludoscope that you don’t know how to do yet?

None

What were the biggest obstacles for you to effectively use Ludoscope?

Not able to open it

Would you continue to use Ludoscope in its current state? Why / Why not?

Yes

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?
It's good for prototyping

How much were you able to do with Ludoscope?
I had trouble installing XNA

What would you like to do with Ludoscope that you don't know how to do yet?
Nothing

What were the biggest obstacles for you to effectively use Ludoscope?
Installing XNA

Would you continue to use Ludoscope in its current state? Why / Why not?
Maybe, if it utilized a provider-based framework

Thank you for your time.
Questionnaire Ludoscope

Pre-Tutorial

What were your first impressions of Ludoscope?

_not visually, not very good, hard to navigate_

How much were you able to do with Ludoscope?

_not much_

What would you like to do with Ludoscope that you don't know how to do yet?

_generate quests/dialogue_

What were the biggest obstacles for you to effectively use Ludoscope?

_using the interface_

Would you continue to use Ludoscope in its current state? Why / Why not?

[Signature] 120

Thank you for your time.
Questionnaire Ludoscope

Pre-Tutorial

What were your first impressions of Ludoscope?

Visually not stunning

How much were you able to do with Ludoscope?

Put some words after each other

What would you like to do with Ludoscope that you don’t know how to do yet?

Generate quests

What were the biggest obstacles for you to effectively use Ludoscope?

Knowing what buttons do

Where to begin

Would you continue to use Ludoscope in its current state? Why / Why not?

No, it seems like it is not finished yet and sometimes gave an error.

Thank you for your time.
Questionnaire Ludoscope

Pre-Tutorial

What were your first impressions of Ludoscope?
outdated and hectic with all those pop-ups

How much were you able to do with Ludoscope?

None, I couldn't install XNA

What would you like to do with Ludoscope that you don't know how to do yet?

export to Unity

What were the biggest obstacles for you to effectively use Ludoscope?

Installation

Would you continue to use Ludoscope in its current state? Why / Why not?

No

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?

Interesting tool, impossible interface

How much were you able to do with Ludoscope?

edit properties

What would you like to do with Ludoscope that you don't know how to do yet?

generate interesting patterns

What were the biggest obstacles for you to effectively use Ludoscope?

time, user interface
MacOS compatible

Would you continue to use Ludoscope in its current state? Why / Why not?

Would, but it would need to be improved for more long-term usage

Thank you for your time.
Questionnaire Ludoscope

Pre-Tutorial

What were your first impressions of Ludoscope?

How much were you able to do with Ludoscope?

How much were you able to do with Ludoscope?

What would you like to do with Ludoscope that you don’t know how to do yet?

What were the biggest obstacles for you to effectively use Ludoscope?

Can you supply all libraries to make it run on Win10

Would you continue to use Ludoscope in its current state? Why? Why not?

Yes, promising stuff :)
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?
I was impressed and... it was user-friendly.

How much were you able to do with Ludoscope?
Not a lot. It's very unintuitive and doesn't have any tooltips.

What would you like to do with Ludoscope that you don't know how to do yet?

Anything.

What were the biggest obstacles for you to effectively use Ludoscope?
Unhandled Exception Errors.

Would you continue to use Ludoscope in its current state? Why / Why not?
No, it's very user-unfriendly and keeps crashing.

Thank you for your time.
Questionnaire Ludoscope
Pre-Tutorial

What were your first impressions of Ludoscope?
Flexible and powerful tool to get better insight and develop automated projects

How much were you able to do with Ludoscope?
Application didn’t work

What would you like to do with Ludoscope that you don’t know how to do yet?
Create a level in Ludoscope and implement it in Unity

What were the biggest obstacles for you to effectively use Ludoscope?

Would you continue to use Ludoscope in its current state? Why / Why not?
I would like to try it

Thank you for your time.
II. Ludoscope to Unity

Ludoscope To Unity DLL Tutorial – Top-down levels
by Nicky Wild

Welcome!
In this tutorial we will continue using Ludoscope to make a level for a simple top-down game in Unity. This tutorial does not show you everything you can do with Ludoscope in Unity and assumes that you have either followed the platformer tutorial or have a basic understanding of how Ludoscope works. The goal of this tutorial is for you to understand how to use Ludoscope in your own Unity project. This is not bound to any genre and top-down is just another example since we have already tackled platformers.

Designing levels
Before we can worry about importing our Ludoscope project into Unity, we need to design a level. While a very basic interpretation of a level has been designed, I urge you to play around with the grammars and make the level chunks more interesting once everything is set up. For now this is what the level generation process looks like:

![Level Generation Process]

The white ‘r’ tiles are turned into random difficulty tiles, with harder difficulties being more uncommon than easier ones, whereas the undefined tiles are turned into one start tile, one end tile and the rest into ground tiles.
The next step is to split these tiles into our desired dimensions. In this case we have gone with 8x8, which gives us 64 tiles per chunk to work with. For your project this may or may not be enough, so tweak to your heart’s content, it will make no difference for the rest of the tutorial. Finally, we fill in the tiles and our very basic level in Ludoscope is complete.

Each difficulty tile, as well as the start tile and end tile, only have one possibly outcome in this example and no further modifications done to them. This is obviously very boring and the only variation here is allowing the level chunks to be mirrored both horizontally as well as vertically.

Importing Ludoscope into Unity
For this step we require the Ludoscope DLL file and a Unity project. This DLL allows you to call Ludoscope methods in your C# scripts to run the generator in your game. Hooking up the Ludoscope DLL into Unity is as simple as dragging and dropping it into your Assets folder. Make sure it is in the root of the Assets folder as it may otherwise not work properly.

<table>
<thead>
<tr>
<th>Unity Projects ➤ LudoscopeTest ➤ Assets ➤</th>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>_Scenes</td>
<td>20-Apr-18 4:32 PM</td>
<td>File folder</td>
</tr>
<tr>
<td>(</td>
<td>Animations</td>
<td>20-Apr-18 1:38 PM</td>
<td>File folder</td>
</tr>
<tr>
<td>(</td>
<td>Prefabs</td>
<td>20-Apr-18 4:52 PM</td>
<td>File folder</td>
</tr>
<tr>
<td>(</td>
<td>Scripts</td>
<td>11-May-18 11:21</td>
<td>File folder</td>
</tr>
<tr>
<td>(</td>
<td>Sprites</td>
<td>20-Apr-18 1:58 PM</td>
<td>File folder</td>
</tr>
<tr>
<td>(</td>
<td>StreamingAssets</td>
<td>20-Apr-18 2:39 PM</td>
<td>File folder</td>
</tr>
<tr>
<td>PhantomGrammarIndependent.dll</td>
<td></td>
<td>20-Apr-18 10:52 AM</td>
<td>Application extens</td>
</tr>
</tbody>
</table>
Now we need to add the entire Ludoscope project into Unity so we can actually generate something. Multiple Ludoscope projects can be added to your project, but in this tutorial we will simply be using the one we used just now. Make a folder named StreamingAssets in the Assets folder and place your Ludoscope project folder in here.

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DifficultySketch.grm</td>
<td>20-Apr-18 1:56 PM</td>
<td>GRM File</td>
</tr>
<tr>
<td>Expand.grm</td>
<td>20-Apr-18 2:39 PM</td>
<td>GRM File</td>
</tr>
<tr>
<td>Expand.rcp</td>
<td>20-Apr-18 2:10 PM</td>
<td>RCP File</td>
</tr>
<tr>
<td>SplitTiles.rcp</td>
<td>20-Apr-18 11:42 AM</td>
<td>RCP File</td>
</tr>
<tr>
<td>TileAlphabet.alp</td>
<td>20-Apr-18 11:26 AM</td>
<td>ALP File</td>
</tr>
<tr>
<td>TopdownTutorial.lsp</td>
<td>20-Apr-18 2:39 PM</td>
<td>LSP File</td>
</tr>
</tbody>
</table>

Your project is now ready for the next step; calling Ludoscope methods in your C# scripts.

**Calling Ludoscope methods**

Open the LevelGenerator.cs script from your Scripts folder. Here you will find this line: `using PhantomGrammar.GrammarCore;`

If this line throws an error, the DLL file is not in the right place. Moving on we have a `private GenerationSystem system;` which will we use to call the Ludoscope methods. In `Start()` you will find how a string with the path to the Ludoscope project, as well as a `GenerationSystem` call to open said project. Ludoscope is now ready to start generating content, which is exactly what we do right away. You can also call the `Build()` later, like after walking through a door or by a button press, but for now we will just generate one level as soon as this script loads.

**Building the level**

As you can see, `Build(Generate());` `Build()` takes another method, `Generate()`, in its parameters. This code may look a little daunting at first, but we will walk you through it. You will not have to touch the majority of it in most cases, so don’t worry too much if it doesn’t click immediately.

`Build` basically consists of two for loops to loop through every tile of our 2D tile map. We then read each tile, one by one, and use the indexes of our for loops to know where the tile should be placed. To keep things simple, we kept every tile equal to one unit in Unity. You may need to apply a scale to the position vector if this is not the case in your project. The reason we use a negative y axis is so we draw from the top left, rather than the bottom left, which would otherwise mirror the tile placement in our scene horizontally.
Next, we call yet another method in which we use a switch statement to instantiate the proper GameObject. The cases are based on the names given in the alphabet of the Ludoscope project, which we read as the label of the symbol. When we find the matching string, we can instantiate a GameObject in its place. We need to assign these GameObjects beforehand as prefabs in Unity. We do this for every single tile in our expression to instantiate the entirety of the generation process. This can be a little taxing, but Unity now has a thing called ECS’s (Entity Component Systems), which should be a lot faster than the current GameObject approach and could be worth looking into if you notice a performance hit.

Back to the generation process, there is still one problem with what we have so far and that is: what if an enemy moves? The enemy tile would leave a void underneath its original location once it moves away from it. The same issue is with generated loot like chests. The solution for this is the usage of containers, which allows us to store more than one symbol per tile.

**Building the containers**

Before moving on the next tile, we are going to look if the current symbol’s container is empty. If it is not empty, we move into that container and instantiate the GameObject in that container as well. We will then continue to look into the container’s container recursively until everything in the current tile has been built. Because a container can contain a container, we can have as much depth in a tile as we want, think of a ground tile with grass and a treasure chest and so on.

This is everything we need to read out a tilemap from Ludoscope in its entirety and turn it into a level in Unity. In this example project, none of the prefabs actually do anything other than have a different sprite. It is up to you to make the collision, damage, etc. Or if you feel confident enough, you can build everything from scratch in your very own project!
Ludoscope Android Tutorial – Top-down levels
by Nicky Wild

Welcome!
In this tutorial we will use Ludoscope in Unity on Android. This tutorial assumes that you have either followed the top-down tutorial or have a decent understanding of how Ludoscope works. The goal of this tutorial is for you to understand how to use Ludoscope on Android in your Unity project. This is not bound to any genre and we stick with the top-down example since we already have that project working in Unity.

Platform dependent compilation
As the Unity editor and the Android environment have different setups for calling Ludoscope, we only start the generator immediately if we are in the Unity editor. On Android we need to make a small detour before we can call the Ludoscope methods. For this we have set up a coroutine which handles all of these steps.

```csharp
void Start()
{
    #if UNITY_EDITOR
    string file = "TopdownTutorial/TopdownTutorial.lsp";
    string fullPath = System.IO.Path.Combine(Application.streamingAssetsPath, file);
    system = new GenerationSystem();
    system.OpenFromFile(fullPath);
    Build(Generate());
    #endif
    #if UNITY_ANDROID
    StartCoroutine("PrepareStreamingAssetsCoroutine");
    #endif
}
```

Streaming Assets on Android
On Android, Streaming Assets are contained in a compressed jar file. To retrieve these files, we need to use Unity’s WWW class. So first of all, we need to specify the location of the files we want to retrieve. Note that .dll files do not participate in the compilation. This means that the only files we need to retrieve are all the Ludoscope related project files:

```csharp
string projectLSP = "TopdownTutorial/TopdownTutorial.lsp";
string tileALP = "TopdownTutorial/TileAlphabet.alp";
string difficultyGRM = "TopdownTutorial/DifficultySketch.grm";
string splitRCP = "TopdownTutorial/SplitTiles.rcp";
string expandGRM = "TopdownTutorial/Expand.grm"
string expandRCP = "TopdownTutorial/Expand.rcp";
```
Note that these steps are the same for every file you need to recover from Streaming Assets on Android. Think of data packages of 3D models or textures that are too big to upload to the Play Store. If you are already familiar with Streaming Assets in Android, you should be able to see where this is going.

From this point on I will only explain the steps for one of the six files, as they are all the same.

The following line creates and starts a web-call to the download location:
```java
new URL("jar:file://" + Application.dataPath + "/assets/" + projectLSP);
```

Next we wait for the download to be done, in your finished game you can write a little animation using WWW.progress, which returns a float between 0 and 1.
```java
while (!wwwProjectLSP.isDone)
    yield return null;
```

Then we set a directory on Android to write our files to:
```java
string directoryPath = Path.Combine(Application.persistentDataPath, "TopdownTutorial");
```

And check whether the directory exists, if not, we create the directory:
```java
if (!Directory.Exists(directoryPath))
    Directory.CreateDirectory(directoryPath);
```

We give every file a unique name in the previously set directory:
```java
string projectLSPPath = Path.Combine(directoryPath, "TopdownTutorial.lsp");
```

To force always having the latest version of the file downloaded, we can delete the files previously in place. Note that if you download files from the web, you will want a more elegant solution than force re-downloading every file. You may want that either way as every file slows down the initial boot time of your app. For this short example, it is a cheap safety measure.
```java
if (File.Exists(projectLSPPath))
    File.Delete(projectLSPPath);
```

Finally, we write the files and we can finally ready our generation system and start generating our level.
```java
File.WriteAllBytes(projectLSPPath, wwwProjectLSP.bytes);
```

That's it!
This tutorial is for Unity on Windows and Android, but it may or may not be possible to get the DLL to work on MacOS and iOS using similar methods.