Hacking Education with Virtual Microworlds

Pascal Chatterjee

July 9, 2014

KTH Department of Computer Science (CSC)
Thesis supervisor: Linda Kann

Abstract

Contrary to popular belief, scientists working in their laboratories are not the members of society who learn the most in their daily lives. The members of society who do *by far* the most learning in their daily lives are children.

Children tend to walk by the age of 12 months; they can express themselves in the extremely complex system known as natural language by the age of two years; and they have developed a theory of mind after spending a mere four years in the world. And they manage all of this before entering formal education.

The progress of even the hardest working university student pales in comparison. This is not just due to the difference in age between adults and children. It also has to do with the environment in which we learn.

Microworlds are environments in which adult minds can **construct** knowledge in a similar way to children. This paper explains the ideas behind microworlds and describes two implementations of them.

One microworld created for this paper, *Narrative Roulette*, is both an engaging and effective learning environment for teenage students at Kungsholmens Gymnasium. However, it does not seem to provide enough direct value for teachers for it to become a regular part of Kungsholmens Gymnasium's curriculum.

Contents

Chapter 1

Introduction

In 2014, it was reported that Swedish 15-year-olds performed under the OECD-average in an international exam which tests problem-solving abilities[1].

I believe that the low performance of Swedish students was not due to their lack of ability, but due to problems with their learning environment. In this thesis project I am going to try and create my own (virtual) learning environment, in which students can learn according to constructionist learning theory.

This can be formulated as the following question:

> Is it possible to create an engaging virtual environment in which Swedish gymnasium students can learn effectively, according to constructionist learning theory?

I will attempt to answer the question by constructing a virtual environment that I will let Swedish gymnasium students use. By observing how the system is used, and informally evaluating students' discussion of any work produced, together with an expert, I hope to find out if my virtual environment actually is engaging and effective.

My hope is that such an environment can provide inspiration for educators that want to motivate students and enable their learning.

1.1   The scope of this project

I choose to place the following constraints on my project:

- The virtual environment should be suitable for classroom-use by gymnasium students in Sweden.

- The virtual environment should enable students to learn by personally constructing knowledge, according to constructionism.

- The virtual environment should encourage students to create "public entities" - products of their experimentation that can be shown to and evaluated by their peers.

Chapter 2

Background

2.1   What is a microworld?

Imagine this. You're hungry. It's late in the day and you need to decide what you're doing for dinner. It's a familiar problem.

What does the solution space for this problem look like? The usual way to figure this out is by examining a single solution and then seeing how its properties vary.

A single solution to the whats-for-dinner problem is a single meal. So the solution space encompasses everything that can be considered a single meal: everything from a steak to a snack bar. And of course there's the null solution: not to eat anything at all.

Ideas: open-ended goals

We explore this solution space every time we plan a meal. We work under certain constraints, such as the availability of ingredients, our personal preferences, diet plans and even the cultural acceptability of certain meals. The set of things we are likely to eat is far smaller than the set of all things that we *could* eat.

That said, we can also be creative with our solutions. Though our goals must conform to a certain shape (say roughly 1000 calories of nutrition), they are also **open-ended**. There are an infinite number of ways to get 1000 calories of nutrition, even taking into account all of the constraints we just mentioned.

Of course, we rarely think about all of these possibilites. We tend to stick to what we know, reducing an infinite vastness to the comfortably familiar.

I cook pasta dishes a *lot*, and I doubt I'm the only one.

If I decide to combine pasta with meat and pesto, I have come up with a potential solution to the whats-for-dinner problem. This is a creative idea in the domain.

To implement my creative idea, I have to cook.

Implementation

What kind of meat shall I use? Shall I make pesto myself or buy it in a jar? What kind of pasta shall I use, and how should I prepare it?

When I make these decisions, I realise my abstract idea with a concrete implementation. I turn the creative idea in my head into something I can put on my plate.

In this case, let's imagine that the idea of pasta is reified into boiled, fusili pasta; meat is reified as fried bacon; and pesto is reified as the home-made kind.

Evaluation

Once my reified idea is on my plate, the evaluation phase begins. If my solution contains roughly 1000 calories, we can call it a meal. But is it a satisfactory one?

Like the solution space, the evaluation space is infinite. I may have achieved my nutritional goals, but failed my goals of taste. Or maybe my meal was both nutritious and tasty, but it just wasn't *novel* enough.

In the end, my evaluation of my meal is evident in what I do the next time I try to solve the whats-for-dinner problem. If I prepare the pasta and the bacon in the same way, but change how I make the pesto, then evidently my pasta and bacon implementations were good enough, but I felt my pesto implementation could be improved.

Learning is a process

By **implementing** my **creative idea** as a consumable "product", and then **evaluating** it by consuming it, I have *learnt* about what works and what doesn't in the whats-for-dinner domain. I have learnt that I can make tasty pasta and bacon. I have learnt that my pesto needs improvement, and my first attempt should have given me some hints about what I should try next time.

> Learning is what happens when someone executes an **idea-implementation-evaluation loop**, and uses feedback to guide future iterations.

Seymour Papert coined the term "microworld" in 1980, describing them as "incubators for knowledge"[27, p120].

I propose the following, more concrete, definition:

> **A microworld is an environment that enables a person to iterate an idea-implementation-evaluation loop within a certain domain.**

A supermarket, kitchen and dinner table is a microworld for learning how to cook meals - for learning to solve the whats-for-dinner problem, day after day.

2.2 What microworlds exist today?

A lot of people seem to spend a lot of time playing video games[2].

What is it that allows games to command the attention of so many people, for so long?

What is a game?

According to Jane McGonigal, a game designer and writer, the four defining traits of a game are the following:

1. "The **goal** is the specific outcome that players will work to achieve. It focuses their attention and continually orients their participation throughout the game. The goal provides players with a sense of purpose."[3]

2. "The **rules** place limitations on how players can achieve the goal. By removing or limiting the obvious ways of getting to the goal, the rules push players to explore previously uncharted possibility spaces. They unleash creativity and foster strategic thinking."[3]

3. "The **feedback system** tells players how close they are to achieving the goal. It can take the form of points, levels, a score, or a progress bar. Or, in its most basic form, the feedback system can be as simple as the players' knowledge of an objective outcome: "The game is over when..." Real-time feedback serves as a promise to the players that the goal is definitely achievable, and it provides motivation to keep playing."[3]

4. "Finally, **voluntary participation** requires that everyone who is playing the game knowingly and willingly accepts the goal, the rules, and the feedback. Knowingness establishes common ground for multiple people to play together. And the freedom to enter or leave a game at will ensures that intentionally stressful and challenging work is experienced as *safe* and *pleasurable* activity."[3]

To summarise, a game comprises of the **voluntary** attempt to achieve a certain **goal**, according to a set of **rules**, with **feedback** on how close you are to that goal.

Are games microworlds? Or are microworlds games?

Let's compare this with our definition of a microworld, from the previous section. A microworld is an environment that supports a loop comprising of:

1. An **idea** that a student comes up with *themselves*.

2. An **implementation** of that idea, according to the *rules* of the microworld.

3. An **evaluation** of the implementation, that gives the student *feedback* about the quality of their implementation and idea.

4. The evaluation generates further ideas, and the loop continues.

As we can see, there seems to be a correspondence between microworlds and games. An **idea** in a microworld corresponds to a **goal** that a student chooses **voluntarily**. The **implementation** of that idea must conform to certain **rules**, set by the *structure* of the microworld. And the purpose of **evaluating** a student's implementation is to give them **feedback** about it, so they can improve their future ideas.

But the two are not exactly equivalent, as a microworld's **idea** conflates the **voluntary** and **goal** traits of a game. Every microworld is a game (each microworld contains all four traits of a game), but not all games are microworlds.

> Only those games that allow the voluntary choice of goals can be considered microworlds.

Flappy Bird vs Minecraft

Wikipedia says this about the gaming phenomenon known as Flappy Bird:

> "Flappy Bird is a side-scrolling mobile game featuring 2D retro style graphics. The objective is to direct a flying bird, which moves continuously to the right, between each oncoming set of pipes without colliding with them, which otherwise ends the game. The bird briefly flaps upward each time the player taps the screen. If the screen is not tapped, the bird falls due to gravity. The player is scored on the number of pipe sets the bird successfully passes through, with medals awarded for the score."[4]

Though players play Flappy Bird **voluntarily**, they have no say in their **goal**. They just have to keep flapping, or they die and the game is over. The simplicity of Flappy Bird's **rules** (just flap) and the sophistication of its **feedback system** (if I'd flapped *slightly* earlier, I'd be alive!) are what make the game addictive[5].

This means that Flappy Bird qualifies as a game, but not as a microworld.

The videogame called Minecraft, on the other hand, is defined by Wikipedia as:

> "Minecraft allow[s] players to build constructions out of textured cubes in a 3D procedurally generated world. Other activities in the game include exploration, gathering resources, crafting, and combat. Gameplay in its commercial release has two principal modes: survival, which requires players to acquire resources and maintain their health and hunger; and creative, where players have an unlimited supply of resources, the ability to fly, and no health or hunger."[6]

The goal of Minecraft's **survival** mode is, unsurprisingly, to survive. This goal is non-negotiable. However, Minecraft also has another mode, in which players have neither health nor hunger. This means that there is only one reason for

playing this mode of the game: the joy of building things from virtual, textured cubes.

In this particular case, of a creative mode of a sandbox game, we have an environment in which players can *choose their goals with total freedom*. This makes Minecraft's **creative** mode a model example of something that is both a game and a microworld.

What other activities can be microworlds?

We've established that microworlds are a subset of games - those activities that contain all four of McGonigal's traits listed above. Of course, the set of all games is wider than just video games. Let's look at some other activities that could be described as microworlds.

Jamming

Consider a few people gathering in a room, each with their own musical instrument, jamming together. There is a **goal**: to make music that sounds good. There are **rules** (or *constraints*): of timing (4 beat bars), chord sequences that sound good together, the traditions of the genre, the expectations of the potential audience, etc. **Feedback** is instant: the musicians can tell if something sounds good, *while they're playing it.* And of course, jamming is **voluntary** in the majority of cases.

This means that jamming is actually a game, even though we wouldn't usually describe it that way. Is jamming also a microworld?

For a game to also be a microworld, it has to allow voluntary choice of **goals**. Though the overarching goal of jamming is to make good music, the group can work towards that with subgoals.

Perhaps the group decide the first step is to come up with a catchy chorus. The guitarist might have some ideas for chords she wants to try; the singer has some idea of what words he wants to sing; the drummer has some ideas about a beat, and so on.

These **ideas** are **implemented** immediately, and **evaluated** soon after, by the musicians who judge whether what they're doing is working. If it isn't, they update their **ideas**: the drummer tries a different beat, or the guitarist switches chords, and they iterate. Otherwise, they move on to another part of the song.

In this light, jamming can be described as a microworld, as it shares its basic structure with something like Minecraft, even though, on the surface, the two activities look very different.

Creative writing

Writing fiction is a microworld too. A lone writer has an **idea** in her head for a story. It probably isn't the whole story, word for word, fully-formed in her head.

It's probably a rough idea, like: "dinosaurs on the loose in zoo, people try to escape" or "criminal couple rob banks and are then shot by the police".

The smaller ideas that make up this grand vision might be: "the people escaping the dinosaurs are archaeologists", or "wouldn't it be cool to have a scene where a rampaging tyrannosaurus rex destroys a skeleton of its own species?".

The **implementation** of these ideas takes the macro form of scenes, characters and events; and the micro form of the words used to describe them. **Evaluation** occurs when the writer reads back what she's written. Often this evaluation leads to immediate changes in ideas and implementation - further iterations of the loop - a process that the author David Foster Wallace once called "feeding the wastebasket"[7].

Again, this makes writing a game, and the free choice of goals, or ideas, makes it a microworld too.

Programming

The microworld of programming is the one I have most experience with. An **idea** in this world could be something like: "I want to build a system where users can vote for things". Smaller ideas could be: "the system should work on mobile phones" and "the page should update by itself".

An **implementation** of this idea could be a HTML5 front-end, a Python back-end, and HTTP and WebSockets to communicate between them.

The **evaluation** would be testing the system, first in unit and end-to-end automated tests, and finally by real users in a production environment.

There is also another way in which programming can be a microworld. Many programming languages support a REPL environment, which stands for **R**ead **E**val **P**rint **L**oop. In a REPL, users can try out **implementations** of their **ideas**, in code, and have them **evaluated** immediately. This provides an additional microworld at the micro-level, embedded within the macro-level microworld of software engineering.

For example, when working in Python:

```
[1] >>> ",".join([d for d in range(10) if d % 3 == 0])
TypeError: sequence item 0: expected string, int found

[2] >>> ",".join([str(d) for d in range(10) if d % 3 == 0])
'0,3,6,9'
```

When we **evaluate** the **implementation** at line [1], the Python REPL throws a `TypeError`, saying that it found an integer where it expected a string.

This gives us feedback to update our **idea**, which leads to the different **implementation** at line [2]. When this is **evaluated**, we get back '0,3,6,9' which is ostensibly what we wanted, and we learned something about the `string.join` method in the process.

2.3   Microworlds enable learning

John Dewey

99 years ago, John Dewey, a philosopher of education, wrote:

> "[. . .] the school in turn will be a laboratory in which the student of education sees theories and ideas demonstrated, tested, criticized, enforced, and the evolution of new truths."[8, p55]

Dewey conceived of a school consisting of laboratories and studios, where students would be able to construct things and experiment with their own hands, instead of constantly being told what to do and how to do it.

He wanted to see **ideas** demonstrated (or **implemented**); tested and criticized (or **evaluated**); for the evolution of new truths (or the updating of ideas through iteration).

Jean Piaget

According to Edith Ackermann's analysis of the work of Jean Piaget:

> "To Piaget, knowledge is not information to be delivered at one end, and encoded, memorized, retrieved, and applied at the other end. Instead, knowledge is experience that is acquired through interaction with the world, people and things."[9]

This is because, to Piaget (according to Ackermann):

> "Kids don't just take in what's being said. Instead, they interpret what they hear in the light of their own knowledge and experience."[9]

Imagine an adult and child standing near a fire. According to Ackermann's analysis of Piaget, Piaget would say that the child will not refrain from touching the flame just because the adult tells the child it will burn their hand (they "don't just take in what's being said").

Once the adult has gone, the child will reach for the flame regardless. They will only abort their attempt to touch the fire once its heat on their hand becomes uncomfortable (adding "their own knowledge and experience" to what they've been told).

Here is our first hint at *why* microworlds could be useful for learning. It's because microworlds are environments in which rich experiences can be had (experiences such as those described by Dewey). Microworlds are also safe places to have those experiences - hurting yourself in a game is far less painful than doing so in real life.

Seymour Papert

> "Constructionism means "Giving children good things to do so that they can learn by doing much better than they could before." [...] Instructionism is the theory that says, "To get better education, we must improve instruction.""[10]

> "Constructionism [...] shares constructivism's connotation of learning as "building knowledge structures" irrespective of the circumstances of the learning. It then adds the idea that this happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe."[11]

Piaget thought that knowledge is **constructed** by the learner: we build **knowledge structures** in our minds, from concrete experiences - interactions with the world, people and things. This is called "constructivism".

Papert added to this theory, to come up with what he calls "constructionism" - the idea that knowledge structures are best constructed when the learner is building a *public* entity.

I believe the inclusion of the word "public" is important. I think it has to do with the **evaluation** stage of our microworld loop.

If an entity is public, it will be evaluated in public, by more people than its creator. This allows for richer feedback than if the creator was the sole person in charge of evaluating their work.

In practice, this means that products of microworlds (whether they are songs, stories or programs) should be exhibited to maximise the value of the feedback received by the creator of the work.

> "Now one can make two kinds of scientific claim for constructionism. The weak claim is that it suits some people better than other modes of learning currently being used. The strong claim is that it is better for everyone than the prevalent "instructionist" modes practiced in schools. A variant of the strong claim is that this is the only framework that has been proposed that allows the full range of intellectual styles and preferences to each find a point of equilibrium."[11]

Microworlds allow students to construct knowledge *in their own way.* They offer building blocks that each student can use to construct knowledge in their own personal style.

The "instructionist" framework of traditional educational does not allow this. When you are being told how to do things, you need to follow the rules set by the instructor, not those that you come up with yourself.

What do microworlds reward?

Let's look at what microworlds reward in their users. To understand this, we should look at the Papertian public entities that would be constructed within these microworlds.

Consider Minecraft. Fans of the TV and book series *Game of Thrones* have built a version of Westeros, the series' fantasy realm, from 1.2 billion bricks. Compared to the size of the characters, this Minecraft construction is the size of Los Angeles[12].

Creating something the size of Los Angeles, even if it only exists in cyberspace, requires considerable skill. Builders need to be able to place single blocks to form superstructures, and compose those superstructures to form hyperstructures, and so on[13].

When shared on servers, Minecraft artifacts are public entities. How are these evaluated by other members of the public? Like other works of art. It appears that complexity for its own sake is not admired; non-trivial complexity must be twinned with aesthetic beauty for an artifact to be admired[14].

The same can be said of the creative writing microworld. Ralph Ellison, the American novelist, has said: "Good fiction is made of what is real, and reality is difficult to come by," which can be interpreted as saying that writing fiction rewards non-trivial "truth" value, instead of mere complexity.

Playing Flappy Bird, on the other hand, rewards (non-trivial) hand-eye coordination. A public entity in the Flappy Bird world is a player's high score. Using this as a feedback mechanism does increase your hand-eye coordination, in a very narrow domain, but does not improve much else.

How do these rewards lead to learning?

During iterations of the microworld loop in the microworlds mentioned, you do the following:

- You build complex superstructures in Minecraft[15].

- You think creatively and express yourself in natural language when writing fiction[16].

- You time your flaps in Flappy Bird.

There is a learning algorithm in the field of Artificial Intelligence called **Reinforcement Learning**. It can be described as follows:

> "[An agent] must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards."[17]

By carrying out actions in its environment, evaluating rewards and iterating, an agent *learns.* If we accept that this applies to human agents as well as artificial ones, we must conclude that iterating the microworld loop within microworlds enables learning too.

Therefore, the rewards in the environments mentioned lead to learning in the following ways:

- Building in Minecraft makes you better at creating complex superstructures.

- Writing fiction makes you better at thinking creatively and expressing yourself.

- You get better at timing your flaps in Flappy Bird.

The first two environments, Minecraft and creative writing, seem to lead to *deeper* learning than Flappy Bird. This is because they are microworlds as well as being games. That said, the mechanism by which learning takes place is the same - reinforcement learning.

2.4 Microworlds are engaging

What is the key to motivation?

According to Dan Pink, it's autonomy[18]. Roman Krznaric agrees, and uses the following statistic to prove his point: 47% of self-employed people say they are "very satisfied" with their jobs, compared to only 17% of those in regular employment[19].

Here is the table from which he got his data[20]:

% of respondents	Full-time	Part-time	Self-employed
Very dissatisfied	5.8	3.2	1.6
Dissatisfied	11.2	10.5	4.7
Neutral	18.7	18.9	12.5
Satisfied	46.9	48.4	34.4
Very satisfied	17.3	18.9	46.9

There are also more than twice as many dissatisfied people working full-time compared to those who are self-employed.

Why would this be?

The report has this to say:

> "This could be attributed to the control that the self-employed have over their work: whilst many work very long hours, their ability to determine when, where and how they work may contribute to their high levels of satisfaction"[20].

If you have the "ability to determing when, where and how" you work, that means you have autonomy over your work.

Do microworlds provide autonomy?

The first stage of our microworld-defining loop is coming up with an **idea** that obeys the constraints of the *structure* of that microworld.

Imagine you are a self-employed, freelance web designer. You have been tasked with coming up with a new landing page for a coffee shop brand.

You have autonomy over when you work, where you work, and how you work. You can work only between 22.00 and 04.00. You can work from home. You can use or skeumorphic design, or flat design.

Let's say you try skeumorphic design, but after you **implement** some skeumorphic elements, and **evaluate** them, something seems off. You update your **ideas** and try flat design instead. Much better.

You exercised your autonomy in *how* you work.

But however much you iterate, you still need to deliver something that looks like a web page. You can't deliver a design for a printed book. Or a design for a 15-minute movie.

Those are the constraints of the *structure* of your web design microworld.

What would a microworld look like without autonomy?

Let's imagine our web design microworld without autonomy.

Your task is still the same: creating a landing page for a coffee shop brand.

But this time, you must work between 09.00 and 17.00 on weekdays. You must work from the office, from your cubicle. You're free to use skeumorphic design or flat design. . . but your boss *really* likes skeumorphic design.

After **implementing** your initial skeumorphic design **idea**, you **evaluate** it, and as before, you find it lacking.

Unfortunately, you can't change it. Because your boss likes it, and their opinion overrules yours.

The structural constraints of the microworld still apply. But now some arbitrary constraints *also* apply: over where, when and how you work.

These additional, arbitrary constraints suck the fun right out of the microworld. They make the microworld less engaging. They make it feel like work, not a game.

What does engagement lead to?

When a microworld allows for autonomy, it is a highly motivating place to experiment in.

In Minecraft, this motivation leads to people building their first basic block structure. For a band jamming together, or a novice writer, it might lead to the first work they're comfortable sharing with friends. For the budding programmer, it usually leads to their first non-trivial, mostly-working program.

And that leads to a sense of accomplishment. But that's not where it ends.

After an initial success, the temptation is there to tweak the original, ever so slightly. The first block structure leads to a house; one song, story or program leads to another, similar in structure but differing in detail.

Eventually, we end up with a blockrealm the size of Los Angeles; a band and author with a string of hits; a programmer in charge of a program that serves billions around the world.

As we argued in the previous section, each iteration of our loop led to learning for all involved.

But it took the engagement enabled by autonomy to start the loop spinning in the first place.

The importance of rapid evaluation

Autonomy is useless without rapid evaluation. Unless your autonomous ideas and implementations are quickly evaluated, your motivation will suffer[18]. Evaluation converts ideas and implementation into *value*. Until you have evaluated your work, you don't know *why* you're doing certain things, and if reality (in the microworld) matches your expectations.

The quicker a student can assign a value to their knowledge of ideas and implementation, the more motivation they have to iterate - to add value to their store of constructed knowledge.

The opposite is also true. Remove rapid evaluation and a microworld becomes far less engaging: creative writing with no hope of publication is less engaging than creative writing with the ability to publish online immediately, even though both scenarios have the same autonomy.

2.5   Virtual microworlds

Unlike our jamming and creative writing microworlds, Minecraft is a *virtual* microworld. This means that software is an integral part of the microworld, in a way that is not necessarily true for mostly physical microworlds.

How does software change things? It eliminates routine tasks, by delegating those to the machine. Let's see how that works in practice:

The physical microworld equivalent of Minecraft is a pile of Lego bricks on the floor. Unlike Minecraft, everything is, understandably, manual. The only way to search for certain kinds of bricks is by conducting a linear search, or, if you're very organised, sorting the bricks first.

There is a limit to the number of bricks the average person can handle. This limits the complexity of our Lego creations. There are also constraints of time and space.

Playing with Lego can occupy the entire floor. This means that only those people with access to the floor can play. Even at the most social Lego sessions, it's unlikely more than a handful of people will be able to participate simultaneously.

And before long, the Lego will have to be tidied away. As bricks are physical, this usually entails breaking up any structures and putting single bricks back in the box.

Minecraft is different. As the bricks are virtual, they can be sorted and organised by the machine. They can be practically infinite in number, and the increasing complexity in structures can be handled by the user interface treating a grouping of blocks as one.

The environment is virtual too, so bricks don't have to be tidied away when you're done; their patterns are stored in bits until you're ready to return. This makes it easier to work on larger-scale projects, as it eliminates the need to ever start over from scratch.

It also makes it easier for users to collaborate. Builders no longer have to be friends, or even geographically close, to work together in the same environment. They just need to be logged in to the same server. This encourages large-scale collaborative projects.

But software also creates a barrier to entry. To participate in a virtual microworld, you need specialised equipment - a computer - that can run it. That said, you need specialised equipment, such as musical instruments, to participate in physical microworlds too, and it seems that computers are becoming less and less "specialised" as time goes on.

However, to run a virtual microworld, you also need specialised *software* to go along with your hardware. For Minecraft, you need to install the Minecraft software package. For programming microworlds, you often need to install an interpreter, configure an editor and to set up a build system.

The easier a microworld is to enter, the greater its potential audience. The microworlds that are easiest to enter require the least set up of specialised software on the part of the user.

The web browser is fast becoming less and less specialised. All computers come with them pre-installed, and most users spend a significant part of their computer time using them.

Microworlds in the web browser are arguably the most easily accessible virtual microworlds. Let's take a look at some current examples.

CodePen (http://codepen.io/)

CodePen is a browser-based virtual microworld for front-end web development, i.e. HTML, CSS and JavaScript. The creation view looks like this:

Figure 2.1: CodePen interface

There are four panes: one each for HTML, CSS and JavaScript code, and one that dynamically renders the result.

This microworld leverages the browsers capacity for dynamic rendering: of parsing and showing the results of code on the fly. It is a website that allows you to experiment with creating websites.

Created entities in CodePen - "Pens" - are public entities and can be browsed by others, and even tweaked by them. This creates a rich environment for creativity and experimentation, and as we have argued so far, a rich source of learning, in this case in the domain of front-end web technology.

Pacemaker (http://pacemaker.net/)

Pacemaker is an iPad application that is also a virtual, musical microworld. It enables users to remix and edit songs from Spotify, in real-time, by touching the iPad screen.

Sadly, Pacemaker does not allow sharing of creations, due to copyright issues. But creations can be played back on the iPad that created them so that listeners can evaluate them.

Blogging

There are many blogging systems on the internet, such as Wordpress, Tumblr, Medium and Ghost.
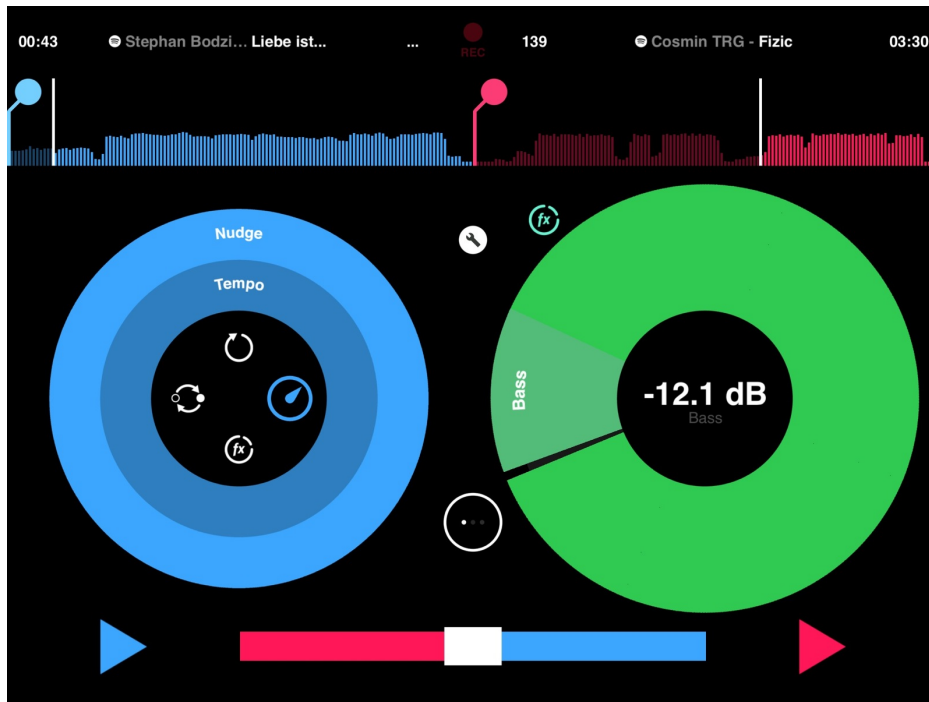
19

Figure 2.2: Pacemaker interface

Blogging is a virtual microworld where **ideas** are **implemented** as text and images, and then (hopefully) **evaluated** by readers who give feedback by sharing and commenting about blog posts on social media.
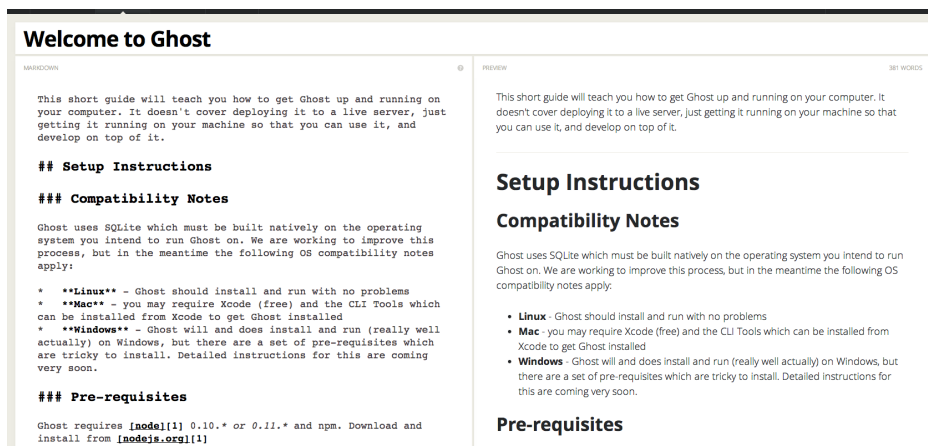

Figure 2.3: Ghost editing interface

Chapter 3

Method

3.1 Choosing a method

To recap, the question behind this thesis is:

> Is it possible to create an engaging virtual environment in which Swedish gymnasium students can learn effectively, according to constructionist learning theory?

Now that we know a little background to microworlds and constructionist learning theory, we have to come up with a method for creating and evaluating the virtual environment (or microworld) named in the question above.

Constraints on my method

My method should produce something that satisfies the following constraints:

- It should allow students to come up with their own **ideas**.
- It should allow students to **implement** their ideas in their own way.
- It should allow students to **evaluate** the implementation of their ideas.

From our original aims, our produced microworld should also. . .

- . . . be suitable for classroom-use by gymnasium students in Sweden.
- . . . encourage students to create "public entities".

Personal choices

I choose to fulfil the constraints listed above by creating **web applications that serve as in-browser microworlds**. I will design these web applications specifically for the classroom. Their purpose is to enable students to create digital public entities. I do this because I have a lot of experience in building web applications.

Planning my evaluation

In the end, I created two microworlds: *Talking to Machines* (TTM) and *Narrative Roulette* (NR). They will be described in detail in the following two sections.

For now, it is sufficient to say that TTM is a microworld in the domain of programming and geometry, and NR is a microworld in the domain of Natural Language and role-play.

Planning how to evaluate *Talking to Machines*

How does one evaluate a programming microworld? The public entities produced in a programming microworld are computer programs. I planned to use peer-evaluation to evaluate the quality of these programs, by having a students' peers give their opinion on the results of a program.

Unfortunately, TTM was too complex a microworld, in terms of syntax and semantics, to be suitable for classroom-use by gymnasium students. I established this first-hand by introducing non-programmers to the microworld, and realising that I had to provide considerable help in order for them to produce even a working program.

This meant that I needed to create another microworld, which would be suitable for classroom-use by gymnasium students: NR.

Planning how to evaluate *Narrative Roulette*

The public entities created in NR were fictional texts in which students' took on certain perspectives. This means that these texts are the products of a simulation exercise. According to Megarry (1978)[25, p187-207], simulations can be evaluated in the following ways:

- "using narrative reports"

- "using checklists gathered from students' recollections of outstanding positive and negative learning experiences"

- "encouraging players to relate ideas and concepts learned in games to other areas of their lives"

- "using the instructional interview, a form of tutorial carried out earlier with an individual learner or small group in which materials and methods are tested by an instructor who is versed not only in the use of the materials, but also in the ways in which pupils learn."

I considered checklists too simplistic and instructional interviews too time-consuming as evaluation methods. I also didn't want to evaluate the texts themselves - treating them as narrative reports - because a guideline for evaluating simulations is that you should be"primarily concerned with the process rather than the product of simulation"[26].

Consequently, I chose the third item above: "encouraging players to relate ideas and concepts learned in games to other areas of their lives". This took the form of oral group discussions of texts, which were informally evaluated by myself and the teacher present during the workshop.

3.2 Talking to Machines

After researching microworlds and developing my theoretical framework, I felt ready to design my own microworld, which I named *Talking to Machines*. You can find it online at http://draw.talkingtomachines.org.

Concretely, this entailed designing an environment in which an **idea-implementation-evaluation** loop could be carried out. For the reasons discussed in the previous section, I decided to host this environment within the web browser.

This decision brought with it some constraints: the interface would have to be built from HTML and CSS, and interactivity would have to either be supplied by client-side JavaScript, or a server-side evaluation environment. But this decision also enabled the increased accessibility discussed earlier.

I decided to focus this microworld on programming, as that is an activity I have a lot of experience with. In a web environment, that would entail using JavaScript as the programming language, as that is available on the client-side, and removes the need to execute a different language on the server and then send back the results.

However, I'm not a great fan of JavaScript when it comes to teaching beginners to program. Instead, I chose to use a dialect of Lisp - Clojure (in its JavaScript-hosted form, ClojureScript) - as Lisp has a history of being a language well-suited for beginners (MIT has used it in an introductory programming course for years [28]).

From ClojureScript to JavaScript

At the time of creating *Talking to Machines*, there was no way to dynamically evaluate ClojureScript in the browser, as the reader and compiler for ClojureScript forms was in fact a Clojure program that could only run on the Java Virtual Machine.

The following makes this rather complicated concept a little easier to understand:

The ClojureScript syntax for printing "Hello World!" to the console looks like this:

```
(.log js/console 'Hello World!')
```

If ClojureScript could be evaluated in the browser, you'd expect to be able to do something like this, in **JavaScript**:

```
ClojureScript.eval("(.log js/console 'Hello World!')")
```

and have "Hello World!" printed to the console.

Unfortunately, in standard ClojureScript, you can't do that yet.

Instead, a **Clojure** program, running on the JVM (i.e. not in the browser), has to **transpile** a ClojureScript file to a JavaScript file. This means that

```
(.log js/console "Hello World!")
```

in the Clojurescript file, `foo.cljs`, is transpiled to

```
console.log("Hello World!")
```

in the output Javascript file, `foo.js`, by a server-side process run during a compilation phase.

It is this file, `foo.js`, that is later evaluated in the web browser.

So what does this mess actually mean, practically?

It means that if a user writes ClojureScript in the browser, we won't be able to evaluate what they wrote without having a backend Clojure process transpiling their code to JavaScript for us.

This isn't actually a massive issue, as we still don't evaluate arbitrary code on the back-end, which would obviously be a Big Deal (we only transpile it), but it would introduce some network lag between the **implementation** and **evaluation** stages of our microworld loop.

We know the tighter the microworld loop, the more engaging the experience, so lag was something I wanted to avoid if at all possible.

Kanaka to the rescue

Luckily for me, there was a fork of ClojureScript[1] that did allow evaluation in the browser, written by Joel Martin ("Kanaka" on Github). At the time of writing it is still a fork, and hasn't been merged with ClojureScript core, probably because it contains "miscellaneous broken things that have not been tracked down yet".

But it was definitely good enough to evaluate ClojureScript for a microworld, and it probably took me less time to modify Martin's code to provide my wished-for `ClojureScript.eval` JavaScript function than it took me to explain why that was necessary.

Why did I go to all this trouble?

This might seem like a very convoluted mess to get into just to avoid writing pure JavaScript, which, after all, is good enough for sites like Codecademy[2]. That's a fair point but I believe that the declarative purity of ClojureScript forms makes up for the chaos going on behind the scenes. As you will soon see.

---

[1] https://github.com/kanaka/clojurescript
[2] http://codecademy.com

A better first impression of programming

Most very-first-introductions to a programming language, or programming in general, involve printing the text "Hello World!" to the screen. Later exercises usually include things like printing the numbers "1 2 3 4 5 6 7 8 9 10", or adding up all integers less than 100.

Though these feats may have been impressive 30 years ago, today these things fail to blow most people's socks off.

Producing plaintext is far from unimportant (most webservers do little else), but in a world where technology means Oculus Rift and 4K Netflix, it's not really sexy.

Is there something simple we can have beginners do, to explain functions and variables and loops, that is more interesting than printing text to the screen?

We could try the activity that captivates children (and artists) all over the world: drawing things. With colours!
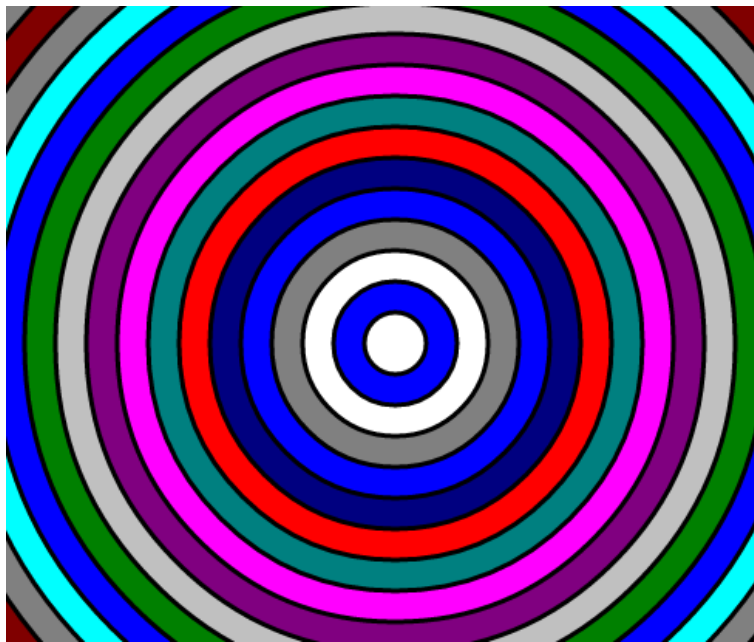

Figure 3.1: The kind of thing that blows socks off.

Declarative shapes

The most basic form of computation is a single function call. It is the simplest action that *does* something (variable assignment does something too, but indirectly, setting things up for later function calls).

It makes sense that drawing a shape to a screen should be the result of just one function call, instead of requiring a novice to construct a class, call methods

on that class, and then draw that class to the screen after having acquired a graphics context.

The more declarative this function call, the easier it is for a novice to deduce its meaning from what happens when its evaluated.

Consider:

```
(circle
 (position "50%" "50%")
 (radius 100)
 (fill "red"))
```

To a non-programmer (and perhaps even a programmer unused to S-expressions), this line of text looks alien. It conforms to a very different kind of grammar than what we're used to reading.

This is why, when I ask non-programmers what they think the `100` in the line above represents, they often have no idea. The text as a whole is just too strange for them to parse and guess that the proximity of `radius` to `100`, and the brackets that enclose them, means that the two tokens have something to do with each other.

If we were in a static environment, the only way to learn about what the above text does would be to read about the syntax and grammar of Lisp, and the semantics of *Talking to Machines.* This is often how programming is approached: from a mathematical perspective. The only way to find out if you had a correct understanding of syntax, grammar and semantics would be to ask a teacher. The gap between **implementation** and **evaluation** would be so large that it would kill a lot of your motivation.

Fortunately, *Talking to Machines* is a microworld. This means that a student can **evaluate** the above text, right in the browser, to render:
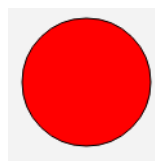

Figure 3.2: The meaning of the S-expression

Then, after processing this visual feedback, the student might have an idea about what the `(radius 100)` form does, and decide to implement her idea by changing that form to `(radius 200)`.

```
(circle
 (position "50%" "50%")
 (radius 200)
 (fill "red"))
```
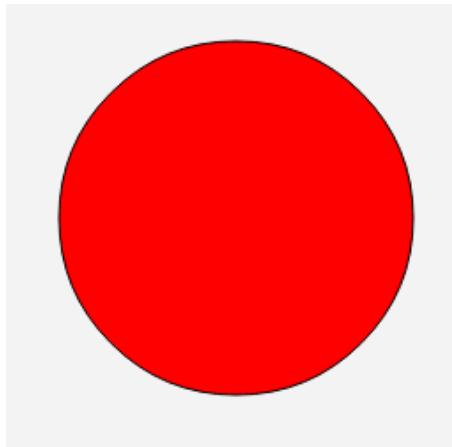
Figure 3.3: Doubling the radius to 200

As soon as this modified text is **evaluated**, a *bigger* circle is drawn on the screen. Immediately, the student realises that the number that was `100` but is now `200` somehow affects the *size* of the drawn circle.

Through their interactions, the student has **constructed** this knowledge for themselves. According to our constructionist learning theory, this knowledge should be far more effective than being *told* that the `(radius 100)` form controls the radius of the drawn circle.

Looping in *Talking to Machines*

Let's see how *Talking to Machines* supports our microworld loop.

Once a student is somewhat familiar with how the microworld works, they can come up with the following **ideas**:

- I want to draw a red circle.
- Can I make it blue?
- Can I make it bigger?
- Can I move it around?
- Can I draw more than one?
- Can I draw other shapes?
- Can I animate them?
- etc.

Their **implementations** of these ideas take the form of composed function calls. These function calls are immediately **evaluated**, and the student sees visually how their change of code affected the change in what was drawn to the screen.

TALKING TO MACHINES

```
1 (circle
2   (position "50%" "20%")
3   (radius 100)
4   (fill "blue"))
```
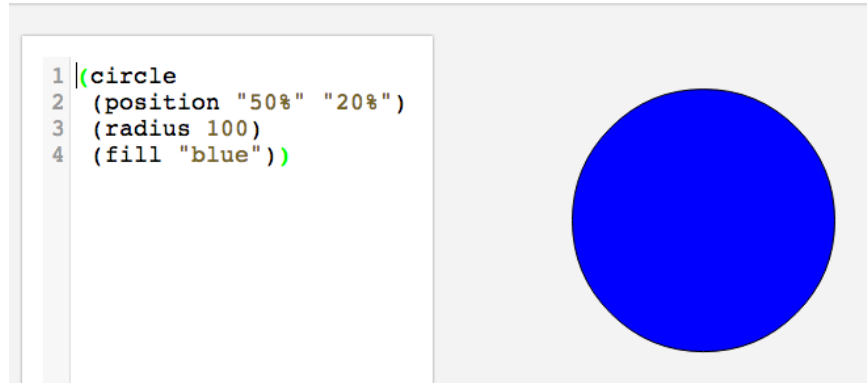
Figure 3.4: The Talking to Machines microworld interface

For example, the first idea above, of drawing a red circle, looks like this:

```
(circle (position 50% 50%) (radius 100) (fill "red"))
```

Making it blue looks like this:

```
(circle (position 50% 50%) (radius 100) (fill "blue"))
```

Making it bigger:

```
(circle (position 50% 50%) (radius 200) (fill "blue"))
```

Moving it around:

```
(circle (position 10% 50%) (radius 200) (fill "blue"))
```

As we can see, the student iterates the following loop *extremely fast*:

1. Hmm, what does this do? (idea)
2. I'll tweak it! (implementation)
3. Ah, that's what that does! (evaluation)
4. Hmm, but what about *this*? (GOTO 1)

This keeps engagement high.

The student learns something new on every iteration, which feels rewarding, which pushes them to iterate again.

29

Limitations of this microworld

Talking to Machines was a success on a *structural* level, but a failure on a *practical* level, at least given the time constraints of this thesis.

It successfully enabled the learning iterations of a microworld, but it required students to have too much prior knowledge for me to be able to conduct workshops with many students at once.

It is possible for students of *Talking to Machines* to learn Lisp syntax through trial and error, by chopping and changing parentheses until the code evaluates. But this is not the most efficient way to learn syntax, so engagement suffers as a result.

Instead, students could be guided through their first attempts to produce syntactically correct code, for example, being told:

> A Lisp function call has the form `(f x)` where `f` is a function and `x` is the argument. Try calling the `circle` function with the argument `50`!

This kind of guidance takes time to do well, either by employing many human assistants with the skill to teach students about syntax, or by programming a tutorial in which the computer itself guides the student.

I didn't have this kind of time. Ideally, I wanted to explore the knowledge-constructing effects of microworlds on students, without having to teach them a new syntax beforehand.

This is what led me to create another microworld, which will be the focus of the next section.
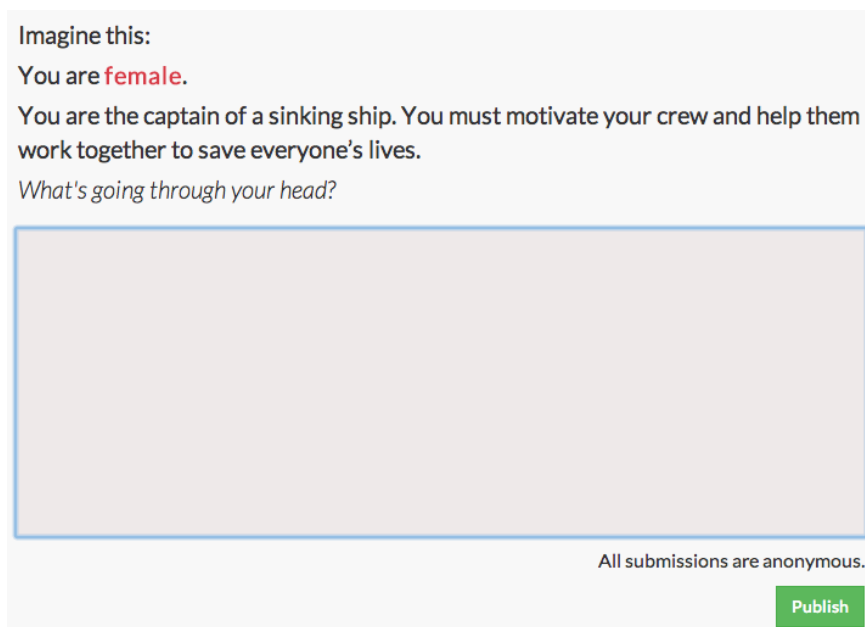
3.3   Narrative Roulette

Narrative Roulette (NR) is a microworld that does not require students to learn a new language before being able to fully explore it. Instead of being built around a programming language like Lisp, *Narrative Roulette* is built around a natural language: English.

You can find NR online at http://kg.narrativeroulette.com.

Rounds of *Narrative Roulette* go like this:

1. The teacher selects an interesting perspective for students to take. For example: "You are the captain of a sinking ship".

2. Students have 15 minutes to write a few fictional, anonymous paragraphs from that perspective.

3. After those 15 minutes are up, students read each other's submissions and discuss them.

4. Then the teacher selects a new perspective and the next round begins.

Figure 3.5: The Narrative Roulette microworld interface

Implementing *Narrative Roulette*

Narrative Roulette can be described as a distributed, real-time system for collaborative schoolwork. Students all run local instances of the client, which is built

using the AngularJS[3] **Javascript** MVC framework. They run these instances in parallel, and use them to write their texts using a cutting-edge **HTML** editor.

When they are done, they post their texts to the backend, which runs on **Python**, and uses **Flask**[4] for web routing, Flask-Restless[5] to create a **REST API**, and SQLAlchemy[6] and MySQL[7] for **data persistence**.

Students can then fetch their peers' work from the backend, to read it and discuss it orally between themselves. Later, the students' work is fetched by a client connected to a projector, and their texts are displayed for and discussed by the entire class.

When I carried out workshops in Kungsholmens Gymnasium, teachers said it was the first time they had ever seen a system like this being used in a classroom[22].
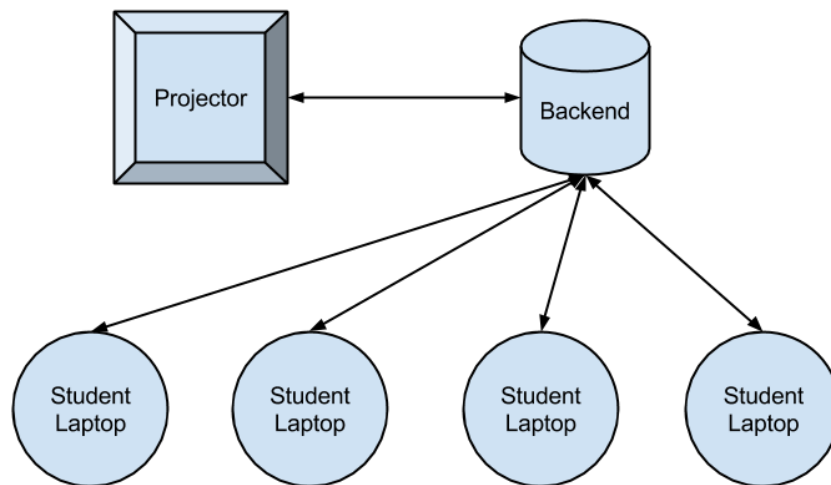
Figure 3.6: Topology of a Narrative Roulette system in a classroom

The benefits of being virtual

As we can see, NR is a creative writing microworld. However, it is also virtual and browser-based. Students read and write their texts in the web browser, and discussions are had while reading texts from screens.

This allows the software to do all of the clerical work. Submissions are edited by students using a web frontend, and then accepted and filed by the backend. As the frontend does not send any identifying information to the backend, anonymity is guaranteed. When it comes to letting students read each other's work, the backend takes care of sending a copy of each text to each student that requests one.

---

[3]http://angularjs.org

[4]http://flask.pocoo.org/

[5]https://flask-restless.readthedocs.org/en/latest/

[6]http://www.sqlalchemy.org/

[7]http://www.mysql.com/

Without this custom-built software, students would have to write by hand, and hope their handwriting is not recognisable by their peers or their teacher. The teacher would have to accept submissions on paper, photocopy them, and then hand them out to the class. This would waste both paper and time, and result in lower motivation for the students.

Narrative Roulette as a microworld

Let's look at this round definition compared to an iteration of the microworld loop:

1. Students come up with **ideas** of what it would like to be the person in the situation defined by the perspective.

2. Students **implement** their ideas in fictional texts.

3. Students **evaluate** those texts by hearing what their peers think of them.

Narrative Roulette is more directed than *Talking to Machines* (TTM). TTM was completely open-ended, allowing the student to draw whatever they wished, whereas NR asks you to write from a particular perspective.

This does not seem to have been a problem; in fact, it seems to have helped focus the imaginations of the students taking part. A totally blank canvas can be intimidating. A frame, such as "you are the captain of a sinking ship", can help you start by providing a trigger for your imagination.

How do you explore this microworld?

Let's see how this works in practice.

A narrative in *Narrative Roulette* contains characters, actions and reactions. Narratives are written from the first-person perspective, so a major character is the narrator, who is the character defined by the round.

Texts usually take the form of internal monologue. This gives the reader first-hand insight into the mental processes of the main character. From personal experience, I know that it also allows the writer to identify, examine and play around with the mental processes they believe the main character could be having.

These mental processes lead to actions, or actually the narrator's description of their actions. For example, the captain of a sinking ship can decide to tell her crew that they need to get into a lifeboat. From our vantage point, inside the narrator's mind, we can see *why* she does this: is she trying to help everyone on board, or is she getting herself and the crew off the ship and leaving the passengers to die? The writer gets to decide exactly how the narrator's thoughts manifest in actions.

Consequences in the narrative are also chosen by the writer. Characters react to the actions of the main character. The writer has to decide what those reactions

are; how they manifest in the actions of those other characters; how the main character *infers* these reactions from the other characters' actions; and finally how the main character reacts to all of these consequences.

And then there are the writer-defined constraints of the storyworld. When we imagine a sinking ship, most of us would imagine enough lifeboats to evacuate all the passengers and the crew. We are in control of the storyworld, so why wouldn't we dream up circumstances in which everyone gets saved?

Because in the real world, things don't always work out so nicely. It's quite possible that there will be fewer seats on the lifeboats than passengers and crew on the boat. Writers who want to challenge themselves will realise this, and in their storyworlds, some people will be left behind. The ensuing moral dilemma will be interesting to both read and write about.

In essence, this is the difference between reading and writing text. When you write, you have to make choices. You choose who your characters are, how they behave and what consequences they have to deal with. Not only this, but you also have to choose how you want to convey all of this to the reader. When time and space is limited, as in *Narrative Roulette*, the best strategy is to say as much as possible using the fewest words.

To recap, writers explore the *Narrative Roulette* microworld by **implementing** their **ideas** about characters, situation and behaviour in effective English prose.

Why is this worth doing?

Narrative theorists have said that the reason we read fiction is to gain a better understanding of the psychologies of the main characters, from their description in the text[21]. We read to learn about why people act the way they do, by analysing fictitious characters and situations. This exercises our theory of mind - our capacity to intuit what other people are thinking and how they will behave.

Reading fiction is a worthwhile activity because the more accurate our theory of mind, the better we are at understanding and predicting the thoughts, feelings and behaviour of others and even ourselves.

When we *write* fiction, we have to **construct** characters, situations, actions, reactions, dialogue and description. In each of these domains, we come up with **ideas**, **implement** them in words, and then **evaluate** them by reading back our text as we are writing it.

According to the constructionist learning theory, this means that we **learn** about characters, situations, actions, reactions, dialogue and description, by **constructing personal knowledge** about them.

Learning how to write rounded characters means learning about human psychology. Learning how to write characters' behaviour means learning about human behaviour. Learning about how to construct interesting situations means understanding the interplay between people and their social, physical, cultural and emotional environment, and being able to manipulate that interplay in the most interesting way.

Writing fiction is an exercise in constructing psychologies, situating them in a storyworld, and then putting them through a chain of events that make up a narrative. This means that a creative writing microworld rewards (and thus improves) theory of mind as well as intuitive theories of causation.

The very best fiction involves realistic characters dealing with realistic consequences. Creative writing microworlds help students construct knowledge about human nature and the world we live in.

How anonymity helps

As mentioned earlier, the quality of texts were evaluated by the quality of the discussion they generated. But if the behaviour of peers dictates the evaluation of a work, won't evaluation descend into a popularity contest? Won't students only read, discuss and share their friends' work, regardless of its actual quality?

They can only do that if they know who wrote a work. Texts in *Narrative Roulette* are anonymous, as the front-end of the software sends no author-identifying information to the back-end.

This forces students to evaluate a text on the strength of only the text itself. It removes bias, unconscious or otherwise. Readers cannot discriminate based on their relationship to the writer, or by the writer's age or gender. They can only discriminate by the writer's abilities, which is how things should be.

This kind of anonymity is rare if not entirely unseen in a classroom setting. As explained above, it brings with it great benefits. Fiction already has the capacity for enabling meaningful discussion about difficult subject matter, with Vladimir Nabokov's fictional portrayal of a paedophile in his novel *Lolita* being a shining example.

Authorial anonymity increases this capacity, and allows students to write and discuss things without fear of being censored by their teachers or their peers.

It also allows readers to be more critical. During the workshops, I was able to honestly critique a student's work without fear of humiliating the author in front of their peers, as the author themselves was the only person who knew that they had written the narrative being discussed.

This kind of anonymity would not be possible (or at least practical) without the help of technology.

Limitations of Narrative Roulette

Narrative Roulette, with its freedom of **ideas**, ease of **implementation** (in natural language), and robust, technology-aided, anonymous **evaluation** mechanisms, is an extremely effective microworld.

However, it suffers from some problems. The biggest of these problems is that it requires a classroom full of students, and that isn't the easiest thing in the world to arrange.

I wondered if there was a way to keep the structure of *Narrative Roulette*, but lose the dependency on a physical classroom. That is the focus of the next section.

3.4 Narrative Roulette on the Web

A problem with *Narrative Roulette* was that it required a classroom and lesson-time for students to take part.

I managed to obtain a classroom without taking time from students on one occasion, by holding a voluntary *Narrative Roulette* workshop on 21 January 2014 at Kungsholmens Gymnasium. Turnout was better than expected (15 students and friends attended), and the session proved that the workshop would work. But it wasn't a true proof of the concept until it was tried on students that *had* to be there as part of their ordinary school day.

Finding lesson-time for *Narrative Roulette* workshops was extremely difficult. Teachers, understandably, seemed to prefer to spend their lesson time achieving the goals of their curricula directly, with their own lesson plans, rather than taking a risk on the indirect value provided by *Narrative Roulette*.

A potential solution to this problem was to remove the dependency on the physical classroom by moving to a purely virtual environment, and trying to take a share of students' leisure time instead of their school-time.

NarrativeRoulette.com

I registered the domain `http://narrativeroulette.com` and modified *Narrative Roulette* so that rounds could take place online.

The modifications were the following:

- Now, rounds would last for one week instead of 15 minutes, with submissions accepted at any time that week.

- All submissions were anonymously posted to Facebook, by the Narrative Roulette Facebook Page (https://www.facebook.com/narrativeroulette).

- Discussion took place in Facebook comments.

- Stories could be shared via Facebook.

- Communication between myself and Narrative Roulette participants took place via posts on the Narrative Roulette Facebook page.

Is this still a microworld?

An iteration of the microworld loop in NarrativeRoulette.com looks like this:

1. The user reads the perspective of the current round. For example: "You are female. You are more scared than you've ever been in your life."

2. The user writes a short narrative, from that perspective, using the editor at NarrativeRoulette.com. This is the **idea** stage of the microworld loop.

3. The user submits their narrative. This is the final **implementation** stage of the microworld loop.

4. People who have liked Narrative Roulette on Facebook see the user's text published by Narrative Roulette. They **evaluate** the text by liking, sharing and commenting on it, all in Facebook.

As we can see, in the NarrativeRoulette.com microworld, the **idea** and **implementation** stages of the microworld loop stayed the same as in the classroom version. The changes were to the **evaluation** stage, where texts were read, discussed and evaluated on Facebook - a virtual space instead of a physical one.

How did this work in practice?

The NarrativeRoulette.com microworld was far less engaging than the classroom version. A round on NarrativeRoulette.com received an average of 7 submissions, while a round in the classroom version received as many submissions as students in the classroom: around 30. This was despite the reach of Narrative Roulette's Facebook page being 38 people.

I believe that there was also an indirect change to the **idea** stage, and this is what caused the difference between the engaging power of the two versions.

When you're in a classroom, attending a *Narrative Roulette* workshop, you have a clear goal: to write a narrative from a certain perspective within a short timespan (15 minutes). You are motivated to do this, even if no teacher keeps an eye on your screen, because you know your peers are all doing this at the same time, and you want to add to the work being created.

Online however, the communal aspect of writing is lost, leading to a loss of motivation. Now you're not writing because you have to, or for your classmates; you're writing for the internet, because you *want* to. And you don't have to submit in the next fifteen minutes, you can submit any time until the end of the week (if at all). This results in a huge lack of *urgency.*

The fun of evaluating texts seemed to disappear too. In the classroom, readers didn't know exactly who had written a text, but they knew it was someone sitting in that room. I believe that led them to be more forgiving, knowing that the writer was one of their peers.

When anonymous submissions appeared on Facebook, readers had no idea who had written them. I believe this led people to be less forgiving (they were just another text on the internet, written by someone they didn't know), which meant that they were reluctant to share, like or comment on the texts, even though they did read them.

NarrativeRoulette.com takeaways

Moving *Narrative Roulette* online removed the need for acquiring classroom space and lesson time from students. However, it failed to be as engaging - there

was less work produced and less discussion generated than *Narrative Roulette*'s physical version.

This means that I consider NarrativeRoulette.com not worth pursuing as a project in its current form; the physical version of *Narrative Roulette* is the superior microworld.

Chapter 4

Results & Discussion

4.1 Was Narrative Roulette successful?

Talking to Machines was a microworld. We saw from its structure how it could enable learning, but we decided it was impractical to test in a classroom setting. *Narrative Roulette* kept the same structure, but changed domain from programming languages to natural language, to make the microworld more accessible.

Narrative Roulette was engaging: there were 140 submissions by 60 students in three obligatory sessions at Kungsholmens Gymnasium. This means that 60 teenagers iterated the microworld loop *at least* 140 times in total.

Why? While writing, students must iterate the loop at a micro-level in their own heads (or on their own screens): they have **ideas**; **implement** them; read them back (**evaluating** them); and then update their ideas.

The lower bound for a student's number of iterations of the loop is one per submission. This occurs if they do *everything right the first time*. In that case, they have an idea, implement it, and evaluate it to be perfect; impossible to improve. Then they send in their submission. Given that this seems unlikely, it is safe to assume that most students iterate the loop multiple times per submission.

Did iterating the loop lead to learning?

We chose to evaluate Narrative Roulette by evaluating the quality of the discussion of student's texts. These discussions took place orally, after each round.

Our discussions turned out to be non-trivial: we talked about moral dilemmas, the use of language, characterisation, gender roles, intertextuality and more[22].

That the work created by students, in the form of 140 narratives, generated reads and intelligent discussion (as evaluated by myself and Eileen Ingulfson[22], a teacher with decades of experience), proves that the work was valuable, and the process that created that work was an effective learning experience.

Arguments against constructionism

There have been many counter arguments made against Piaget's constructivism, and a few (perhaps due to its recency) against Papert's constructionism.

Here are three, taken from Thirteen.org[29]:

1. "It's elitist. Critics say that constructivism and other "progressive" educational theories have been most successful with children from privileged backgrounds who are fortunate in having outstanding teachers, committed parents, and rich home environments. They argue that disadvantaged children, lacking such resources, benefit more from more explicit instruction."

2. "Social constructivism leads to "group think." Critics say the collaborative aspects of constructivist classrooms tend to produce a "tyranny of the majority," in which a few students' voices or interpretations dominate the

group's conclusions, and dissenting students are forced to conform to the emerging consensus."

3. "There is little hard evidence that constructivist methods work. Critics say that constructivists, by rejecting evaluation through testing and other external criteria, have made themselves unaccountable for their students' progress. Critics also say that studies of various kinds of instruction – in particular Project Follow Through, a long-term government initiative – have found that students in constructivist classrooms lag behind those in more traditional classrooms in basic skills."

These are my thoughts on each of the above arguments, in the same order:

1. I cannot refute the argument about elitism. I conducted my workshops at Kungsholmens Gymnasium, one of the hardest schools to get into in the country. This suggests that many of the students that took my workshops came from privileged backgrounds. I was not able to try out *Narrative Roulette* on a more diverse student population due to time constraints.

2. I tried to get around the "tyranny of the majority" by making students' texts anonymous. This ensured that students would discuss each others' texts in an unbiased way, but could not prevent oral group discussions potentially suffering from a "tyranny of the majority".

3. I have tried to account for the value of constructivist work in this very thesis. It is up to the reader (and more pertinently, my examiner) to say how well I have succeeded.

4.2   What do microworlds mean for education?

Let's compare the structure of traditional teaching methods with microworlds. Traditional teaching is often split into three activities: lectures, exercise sessions, and exams. Lectures provide students with theory, exercise sessions (or seminars) allow students to put that theory into practice by solving problems, and exams allow students to demonstrate both their theoretical and practical skills, in isolation - test conditions, generally with no help from textbooks, the internet or their peers.

There are rough parallels between these methods and microworlds. Lectures correspond to **ideas** in a certain domain. A major difference between ideas in lectures and ideas in microworlds is that ideas in lectures are *external*. They are other people's ideas, such as "Recursive loops never terminate if you leave out a base-case". In a microworld, the ideas you have are *internal*, your own: "If I write this base-case, my loop will terminate.".

Exercises correspond to **implementation**. However, unlike microworlds, traditional exercise sessions have students implementing external ideas. For example: "This is Pythagoras' theorem, now apply it to this scenario!". In a microworld, it is a student's internal ideas that are implemented: "I wonder if I can use Pythagoras' theorem to help me draw a right-angled triangle."

Exams are the **evaluation** stage. Unlike microworlds, exams do not give instant feedback. Failing at problem-solving in an exam is not a learning experience, it results in a period of uncertainty, which ends when you receive a grade lower than you were expecting. If having your exam marked by a teacher taught you something, perhaps how to solve a problem correctly, after the course is over, you have no way to apply that knowledge, or to prove that you now have it.

When you fail at problem solving in a microworld, you know immediately, enabling you to update your understanding immediately. Evaluation is *productive* in a microworld, in a way that exams generally aren't.

In short, traditional learning methods do provide **ideas**, **implementation** and **evaluation**, but in components that are spread out in time. A microworld places these components in a tight loop that is iterated many times at speed. This suggests microworlds *might* lead to more learning than traditional teaching methods. Further investigation is required!

Agile methods vs The Waterfall Method

There are echoes of this comparison in Software Engineering ideology. Traditional teaching methods arguably correspond with The Waterfall Method. The Waterfall Method consists of [23]:

1. Requirements and Design (Microworld **idea**s).

2. Implementation (Microworld **implementation**, obviously).

3. Verification (Microworld **evaluation**).

As in traditional teaching, these steps are separated in time. The requirements and software design are done first, before any implementation. Then the implementation is carried out. Finally, the implementation is verified by the customer.

According to Dean Leffingwell and many others, "the Waterfall Method doesn't work"[24]. Instead, most modern software engineers prefer agile methods.

These agile methods move the requirements, design, implementation and verification steps closer in time, and prefer fast iterations. Agile methods make software engineering into a microworld.

4.3   What problems are there with microworlds?

Let's imagine that microworlds - virtual, physical and hybrid - are adopted in the mainstream curricula of schools and universities worldwide. What consequences would this have for the structure of education?

Fully embracing microworlds

As discussed in the previous section, lectures could remain relatively unchanged, with the difference being that they are no longer mandatory, but can be voluntarily attended by students undertaking self-directed research. These lectures could still be labelled by subject in the same way as today, for example: the biology of psychopathy is Natural Science; the ethics of violence is a Social Science; and the literature of crime is Art. The reasoning behind this is that the label simply dictates the *angle* the subject will be approached from. As lectures are voluntary, students can mix and match these angles as they wish.

Instead of producing work within a subject area such as Biology, Philosophy or English Literature, students would then apply their knowledge by exploring microworlds that span multiple subject areas.

Their aim would be to create public entities that can be peer-evaluated, by the consumption of the work by a student's peers, and the sharing and discussion it generates. A possible work could be creating an interactive virtual environment, like a game, where players explore the concept of psychopathy, from a biological, ethical and literary perspective.

This could be something like a point-and-click adventure where you play a psychopath, and perceive the world as a student believes a psychopath would. The student who built the game would have to study biological, psychological and literary work on psychopathy to create a realistic world. The student could then explore ethical questions by offering players choices about how to behave. The resulting narrative in the game world would demonstrate students' literary abilities.

For variety, students could work on multiple projects in multiple microworlds, all with different themes, at the same time during a term. Every so often, a project would come to an end, and students' work would be presented in an exhibition and peer-evaluated.

After one set of projects have been evaluated, a new set of projects, in their own specifically-designed microworlds, would begin.

From lesson plans to microworld construction

If lessons are replaced with microworlds, those microworlds will have to be constructed. Physical microworlds, like jamming with instruments or creative writing on paper can be constructed by teachers today.

Virtual or hybrid microworlds are harder to construct without technical skills. *Talking to Machines* and *Narrative Roulette* had to be programmed specifically

- they had too many custom requirements to be put together from a standard system.

I believe I gained the majority of the skills required to program those microworlds from working in the software industry, not from my Computer Science education. This suggests that, today at least, software industry experience might be necessary to construct microworlds. That is something which, today, the vast majority of teachers do not currently have.

That said, computer programming will be a compulsory part of primary and high school education in the UK from September 2014[30]. Estonia already has a compulsory programming curriculum in place, for children as young as those in the first grade[31].

Maybe learning-to-code initiatives like these will lead to programming microworlds being a greater part of future curricula, and the programming skills conveyed by them will empower teachers and students alike to create virtual microworlds of their own.

4.4   Conclusions

Microworlds seem to enable learning in an engaging and effective way, by allowing students to construct knowledge through exploring rich environments. Unfortunately, there does not seem to be infrastructure in place in educational institutions today to fully embrace virtual microworlds, as teachers do not have the technical skills required to create or maintain them.

Teachers at Kungsholmens Gymnasium thought *Narrative Roulette* was a great idea, but could not find lesson-time for me to hold workshops. I only managed to hold my three workshops when I worked together with a teacher to have them during substitute lessons, and only then when those substitute lessons came up too fast for the substitute teacher to be able to produce a lesson plan for them. It was only in this very particular scenario, when lessons appeared in which the actual curriculum could not be followed, that *Narrative Roulette* was given a chance.

The students loved *Narrative Roulette*[22]. As I have argued, they also learned a lot by participating.

But only I could maintain the software during the workshops, and even if I improved the interface so that others could run it, the workshops did not provide clear, direct value for the current curriculum. This means that *Narrative Roulette*, in its current form, is only a prototype, not something that can be deployed at scale.

Bibliography

[1] Skolverket, Press Release
http://www.skolverket.se/press/pressmeddelanden/2014/
svaga-resultat-i-ny-pisa-rapport-1.217275
April 2014

[2] Jane McGonigal,
TED Conversation
http://www.ted.com/conversations/44/we_spend_3_billion_hours_
a_wee.html
2011.
Retrieved June 2014

[3] Jane McGonigal,
*Reality is Broken: Why Games Make Us Better and How They Can Change
the World*
Vintage Digital
2011.

[4] Wikipedia,
http://en.wikipedia.org/wiki/Flappy_bird
Retrieved June 2014

[5] Ewan Spence,
http://www.forbes.com/sites/ewanspence/2014/02/18/
the-vital-and-depressing-lessons-flappy-bird-can-teach-indie-developers/
Forbes.com
2014.
Retrieved June 2014

[6] Wikipedia,
http://en.wikipedia.org/wiki/Minecraft
Retrieved June 2014

[7] David Foster Wallace,
Both Flesh And Not
Penguin
2012.

[8] John Dewey,
The School and Society and The Child and the Curriculum (Centennial

Publications of the U of C Pr)
University of Chicago Press
1915.

 [9] Edith Ackermann,
Piaget's Constructivism, Papert's Constructionism: What's the difference?
http://learning.media.mit.edu/content/publications/EA.Piaget%20_%20Papert.pdf MIT
Retrieved June 2014.

[10] Seymour Papert,
http://papert.org/articles/const_inst/const_inst1.html
Papert.org
Retrieved June 2014.

[11] Seymour Papert, Idit Harel
Constructionism
Ablex Publishing Corporation
1991.

[12] Laura Hudson,
http://www.wired.com/2013/03/westeroscraft-game-thrones-minecraft/all/ Wired
Retrieved June 2014.

[13] Ikarus3426
http://www.reddit.com/r/Minecraft/comments/dibqq
Reddit.com self.Minecraft
Retrieved June 2014.

[14] Matt Silverman
http://mashable.com/2013/02/13/amazing-minecraft-creations//#_
Mashable
Retrieved June 2014.

[15] Oliver Gee
http://www.thelocal.se/20130109/45514
The Local
Retrieved June 2014.

[16] Shah C, Erhard K, Ortheil HJ, Kaza E, Kessler C, Lotze M.
Neural correlates of creative writing: an fMRI study.
University of Greifswald
2011.

[17] Richard S. Sutton, Andrew G. Barto
Reinforcement Learning: An Introduction
The MIT Press

[18] Daniel H. Pink
Drive: The Surprising Truth About What Motivates Us

Canongate Books
2010.

[19] Roman Krznaric
How to Find Fulfilling Work
Macmillan
2012.

[20] Nick Isles
http://twfold.theworkfoundation.com/assets/docs/publications/145_Joy_of_Work.pdf
The Work Foundation
PDF, retrieved June 2014.

[21] Lisa Zunshine
Why We Read Fiction: Theory of Mind and the Novel (Theory and Interpretation of Narrative)
Ohio State University Press
2012.

[22] Eileen Ingulfson
Personal e-mail correspondence
March 2014.

[23] Wikipedia,
http://en.wikipedia.org/wiki/Waterfall_model
Retrieved June 2014

[24] Dean Leffingwell
Scaling Software Agility: Best Practices for Large Enterprises
Addison-Wesley Professional
2007.

[25] Megarry J.
Perspectives on Academic Gaming and Simulation 3: Training and Professional Education.
Kogan Page
1978.

[26] Louis Cohen, Lawrence Manion, Keith Morrison
Research Methods in Education
Routledge
2007.

[27] Seymour Papert
Mindstorms: children, computers, and powerful ideas
Basic Books
1980.

[28] *Computing as Engineering Design, Course Description*
http://mitpress.mit.edu/sicp/course.html
MIT
Retrieved June 2014.

[29] *Constructivism as a paradigm for Teaching and Learning*
http://www.thirteen.org/edonline/concept2class/
constructivism/index_sub5.html
Thirteen.org
Retrieved June 2014.

[30] Rory Cellan-Jones
Are teachers ready for the coding revolution?
http://www.bbc.com/news/technology-25857276
BBC News
Retrieved June 2014.

[31] Parmy Olson
Why Estonia Has Started Teaching Its First-Graders To Code
http://www.forbes.com/sites/parmyolson/2012/09/06/
why-estonia-has-started-teaching-its-first-graders-to-code/
Forbes
Retrieved June 2014.