Niilo Säämänen

Building a cognitive gaming platform

User centric gaming experiences with organic movement

Helsinki Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

1.03.2013

Helsinki Metropolia University of Applied Sciences

Author(s)	Niilo Säämänen
Title	Building a cognitive gaming platform: User centric gaming experiences with organic movement
Number of Pages	106 pages
Date	1 March 2013
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Media Engineering
Instructor(s)	Matias Palva, CEO Neuroware Group Harri Airaksinen, Head of Line

The increasing focus on efficiency and optimizing the way people think and work has led to a new area of serious gaming – cognitive games. The rise of modern web rendering technologies has enabled the creation of visually interesting cognitive games on browser based technologies. The goal of this study was to assess the applicability of using modern browser technologies to create a user centric cognitive gaming platform and the use of mathematical formulas in organic rendering.

The approach discusses the current market situation and the products and methods of cognitive gaming as well as the technologies involved. The user centric approach is studied through user experience design as well as graphic design and animation aspects. The reference implementation is project CCA; a user centric cognitive gaming platform built on top of Adobe Flash that uses seemingly organic movement rendering.

The technical implementation is discussed from the platform client-server aspect as well as an overview of the structure of the front end architecture. The rendering engine methods go through the 2D –based rendering of mathematical formulas, the use of continuous Bezier curves in organic movement and the creative ways of using Perlin noise to generate textures as well as movement. Optimization of complex rendering and platform building is an essential part of the process.

The results show the viability of using modern browser based technologies in the creation of a cognitive gaming platform. Through the use of optimization and creative mathematical solutions, as well as tending to user experience needs a successful product is built. The project platform is used in medical trials, as well as the Science Changing the World Exhibition shown in science centers around Europe. This study stands as a testament to the possibilities of cognitive end user training and a guide on the aspects of building a successful gaming platform.

Keywords	cognitive gaming, user experience, organic rendering

Tekijä Otsikko Sivumäärä Aika	Niilo Säämänen Kognitiivisen pelialustan rakentaminen: käyttäjäkeskeinen pelikokemus luonnollisella liikkeellä 106 sivua 1.3.2013
Tutkinto	Master of Engineering
Koulutusohjelma	Information Technology
Suuntautumisvaihtoehto	Media Engineering
Ohjaajat	Matias Palva, CEO yliopettaja Harri Airaksinen

Tehokkuuden optimointi on johtanut muutoksiin ihmisten ajattelu- ja työtavoissa. Kognitiiviset pelit kehittyivät vakavan pelaamisen alalajiksi vastaamaan tehokkuuden asettamiin tarpeisiin. Samaan aikaan nykyaikaiset selainteknologiat ja renderöintitekniikat ovat kehittyneet sille tasolle, että niillä voidaan luoda visuaalisesti näyttäviä kognitiivisia pelejä. Opinnäytetyön kohteena oli arvioida nykyaikaisten selainteknologioiden sopivuutta käyttäjäkeskeisen kognitiopelialustan luomiseen ja matemaattisten kaavojen käyttöä luonnollisen liikkeen ja muodon renderöinnissä.

Tutkimuksen lähtökohtana oli nykyinen markkinatilanne kognitiopelaamisen alueella sekä siihen liittyvät tuotteet ja teknologiat. Käyttäjäkeskeistä lähestymistapaa tutkittiin käyttökokemussuunnittelun sekä graafisen suunnittelun ja animaatioiden kautta. Referenssitoteutuksena kehitettiin projekti CCA, käyttäjäkeskeinen kognitiivisen pelaamisen alusta, joka pohjautuu Adobe Flash-teknologiaan. CCA:n renderöintiteknologia perustuu luonnollisen liikkeen optimointiin.

Pelialustan tekninen toteutus on modulaarinen palvelinintegroitu ratkaisu, jonka tarkoituksena on olla mahdollisimman muokattava taustajärjestelmän kautta. Renderöinti perustuu matemaattisten kaavojen 2-ulotteiseen ilmaisuun, jatkuvien Bezier-käyrien sekä erilaisten Perlin-kohinoiden luoviin käyttötapauksiin. Optimointi on elintärkeä osa monimutkaisen renderöintialustan rakentamisessa.

Opinnäytetyön tulosten mukaan nykyaikaisilla selainteknologioilla voidaan rakentaa onnistunut kognitiivinen pelialusta. Optimoinnin ja luovien matemaattisten ratkaisujen sekä käyttäjälähtöisen suunnittelun ja toteutuksen avulla luodaan onnistunut tuote. Projektin tuote on käytössä kliinisissä kokeissa Suomessa ja Virossa, ja siitä eriytetty moninpeliversio kiertää tiedekeskuksissa ympäri Eurooppaa. Opinnäytetyön lopputulos on esimerkki käyttäjäkeskeisten kognitiopelien mahdollisuuksista ja toimii ohjeena selainpohjaisen pelialustan kehittämiselle.

Avainsanat	kognitiopelit, käyttäjäkokemus, orgaaninen renderöinti

Contents

1 Introduction 1

2 Research problem background: Project CCA 3

2.1 Introduction and history 3
2.2 User walkthrough 5
2.3 Target user group 7
2.4 Project lineage 7
2.5 Technology choices 12
2.6 Challenges and key concepts 15

3 Cognitive gaming 18

3.1 History and definition 18
3.2 Current market situation and future 19
3.3 Products and competition 20
3.4 Common test types 26
3.5 Game engines 28
3.5.1 Static game engines 29
3.5.2 Dynamic game engines 29

4 The User Experience approach 31

4.1 Definition 31
4.2 User experience in gaming 32
4.3 Unified visual style 33
4.3.1 Typography 33
4.3.2 Composition and the use of space 37
4.3.3 Elements 40
4.3.4 Colors 42
4.4 Motion and visual narrative 44
4.4.1 Timing and flow 45
4.4.2 Tweening and easing algorithms 46
4.4.3 Movement patterns 48
4.4.4 Overrides in code based animation 49
4.5 Visualization of data 51

5 The CCA project platform 53

5.1 General architecture 53
5.2 Client server communication and data management 56
5.3 Game generation 58
5.4 Game rendering event 60
5.5 User reaction tracking 62

6 Rendering engine 63

6.1 Basic principles 63
6.2 Creature shape rendering 65
6.2.1 Overview of creature rendering 65
6.2.2 Creature formulas 67
6.2.3 Path wrapping 69
6.3 Path creation and path finding 70
6.3.1 Continuous Bezier curves 72
6.3.2 Path interpolation and frame path 74
6.3.3 Path finding and target location generation 75
6.4 Background rendering 77
6.4.1 Perlin noise 77
6.4.2 Luminance in path generation 78
6.4.3 Treshholding Perlin Noise for pattern backgrounds 80

7 Performance optimization 82

7.1 Basic principles 82
7.2 Platform optimization 85
7.2.1 Approach 85
7.2.2 Creature optimization 87
7.3 Flash specific optimization 89
7.3.1 Overview 89
7.3.2 Function calls and language specific libraries 92
7.3.3 Bitwise operators 93
7.3.4 Drawing solutions 94

8 Results and future implications 96

8.1 Current situation 96
8.2 Clients and users 96
8.3 Heureka multiplayer version 98
8.3.1 Introduction 98
8.3.2 Multiplayer 99
8.3.3 AIR platform 101

8.3.4 Testing, long uptime, automatic start up and maintenance 102
8.4 Future 103
9 Conclusions 105
References 107

1 Introduction

Humanity as a whole has seen increased focus on optimizing the way we work, the way we consume, and the way we think. This need for ever improving performance has made us consume smarter, perform more efficiently at work and waste less time.

During recent years, the world of gaming and digital entertainment has seen a growth in a new area, one that was not focused solely in time consuming entertainment; the coming of so called “brain fitness”. Gaming suites and platforms offering rather simple, basic mechanics that provided the user various activities they could do to train their brains. The central tenet in these games was that through exercise powered by game related reward models one could improve the way one thinks, and track that improvement as well.

The brain games became very popular, and spread throughout the entertainment ecosystem from mobile phones to modern console platforms. The games varied from simplistic memory exercises to the ones based on largely theoretical premises, and offered actual data as back up of their effectiveness. For the most part the games have been shown to increase your ability to memorize things, but the actual science proven benefit is still under much debate.

Aside from the more mass market oriented products, the world of science started to take interest in the inherent interest / reward models and game environments and how they could be used in rehabilitation and analysis of different diseases and medical conditions that affect our cognitive functions. Cognitive gaming is based on the concept of neuroplasticity; the ability of the brain to physically adapt to new stimuli.

At the same time, the front end tools for web development have improved to a level closer to full desktop experience. The introduction of RIA (Rich Internet Applications) technologies like Adobe Flex and Microsoft Silverlight, in addition to the rising support for the next W3C standard HTML5 and CSS3 have enabled high production value, fully fledged user experiences across platforms within the browser.

This study focuses on the key issues in building a user oriented gaming platform, and the applicability of organic movement and shapes with mathematic formulas in cogni-

tive gaming. The reference implementation is a consumer oriented platform for cognitive gaming built for Neuroware Group; a browser-ran application to measure, improve and develop cognitive abilities. The goal of this thesis is to study how to develop a rendering system for seemingly organic movement, and a fully-fledged cognitive gaming platform with a user experience focus for improving people's lives using modern web front end technologies.

The present study does not explore the medical theories of cognitive gaming, nor is it a study of software engineering methods themselves. This study focuses mostly on the front end platform implementation, where the game logic and rendering lies, and only has an overview of the backend system of project CCA.

The first part of the study focuses on the basis of the reference project, the setup of the research question and key factors and challenges. The second part provides an overview of brain gaming and introduces the user experience needs and specifics in consumer market oriented platform building, as well as details on CCA platform implementation. The third part introduces the technological core of the platform, rendering engine specifics, optimization approach and implementation and how the challenges were overcome.

After going through the theory and reference implementation, the final part discusses the merits, accomplishments and future of the project CCA. From current user base and use cases to possible future uses, as well as the related branch of the project currently presented at the Science Changing the World exhibit.

2 Research problem background: Project CCA

The reference project is collaboration between the concept owner and CEO of Neuroware Group – Matias Palva, PhD, and Niilo Säämänen, the author of the present thesis. Neuroware Group is an innovative small company focusing on neuro-gaming and cognitive training games that are based on theoretical research on the fundamental workings of the human brain.

2.1 Introduction and history

The project began based on the research done by Satu and Matias Palva at the University of Helsinki, Neuroscience center. Matias Palva had started a prototype of the creature rendering system on the LabView platform to test the possibilities of making a consumer oriented cognitive game based on his research. The project CCA started in 2008, on the 18th of December with a meeting with Matias Palva. Based on the prototype idea a consensus on the viability of creating the platform with modern web technologies was found.

The aim of the project was to create a consumer oriented platform for cognitive gaming based on the research done by Satu and Matias Palva on neuroplasticity and the application of organic shapes and movement in the realm of brain fitness. The target was to make a browser based solution, with a fairly large portion of the platform logic and controls coming from the back-end solution, enabling a modular and extendable solution for brain training.

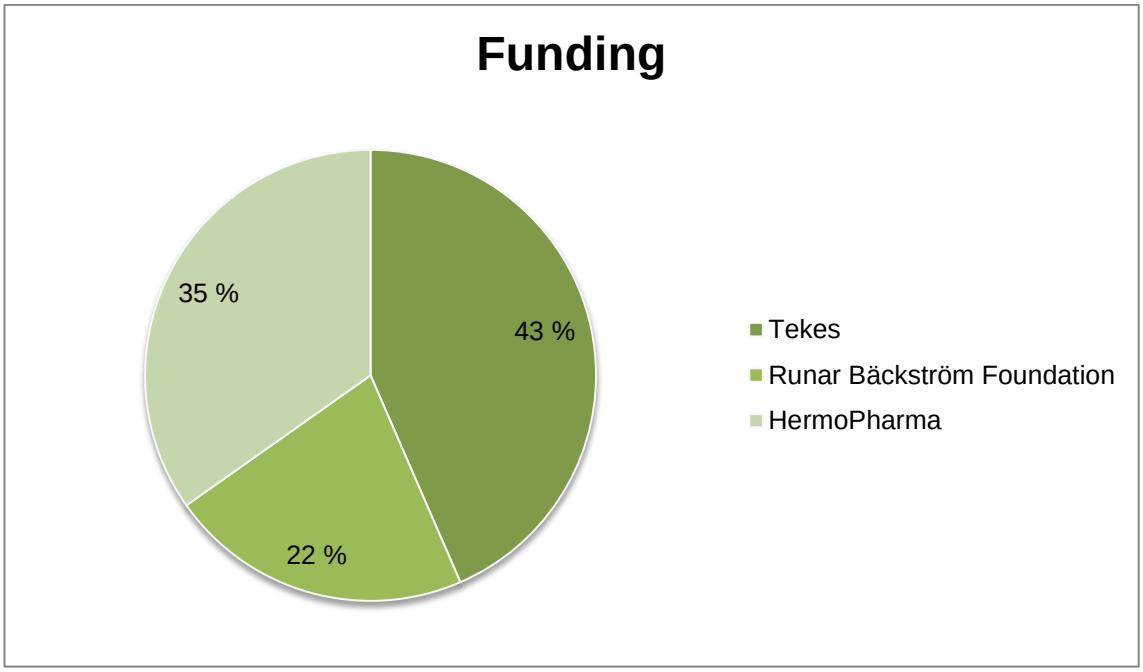


Figure 1. CCA project funding

Figure 1 shows the approximate funding partners and their relative contributions to the project during its lifetime. The project had Tekes funding for the initial prototype and development. Development was started without certainty about future budget options. During the development additional funding was acquired as a grant from the Runar Bäckström Foundation, and closer to the final stages of the project, the trials with HermoPharma funded further development. The approximate total external funding in project CCA was 70 000€.

The projects first release version was finished during the spring of 2011, after 2 years of development. Since the 1st release version, the platform has been further developed and optimized. The main goals of the project have been achieved, and the whole platform had been heavily reworked and evolved to a point of maturity.

The project work was delegated as follows: The front end platform development, rendering engine development and UX-design by Niilo Säämänen, graphic design by Mikko Häkkinen and the game design, mathematical theorems and back-end solution by Matias Palva. Mostly the project was an intense collaboration between Matias Palva and Niilo Säämänen.

2.2 User walkthrough

The reference project is a cognitive gaming platform for consumer use. The tool is meant to be used by end users of all ages and trades, and to be easily approachable and trustworthy tool for measuring and improving cognitive processing efficiency. From a user's perspective, the platform is a web-browser based game system with user authentication and personal, account based training programs.

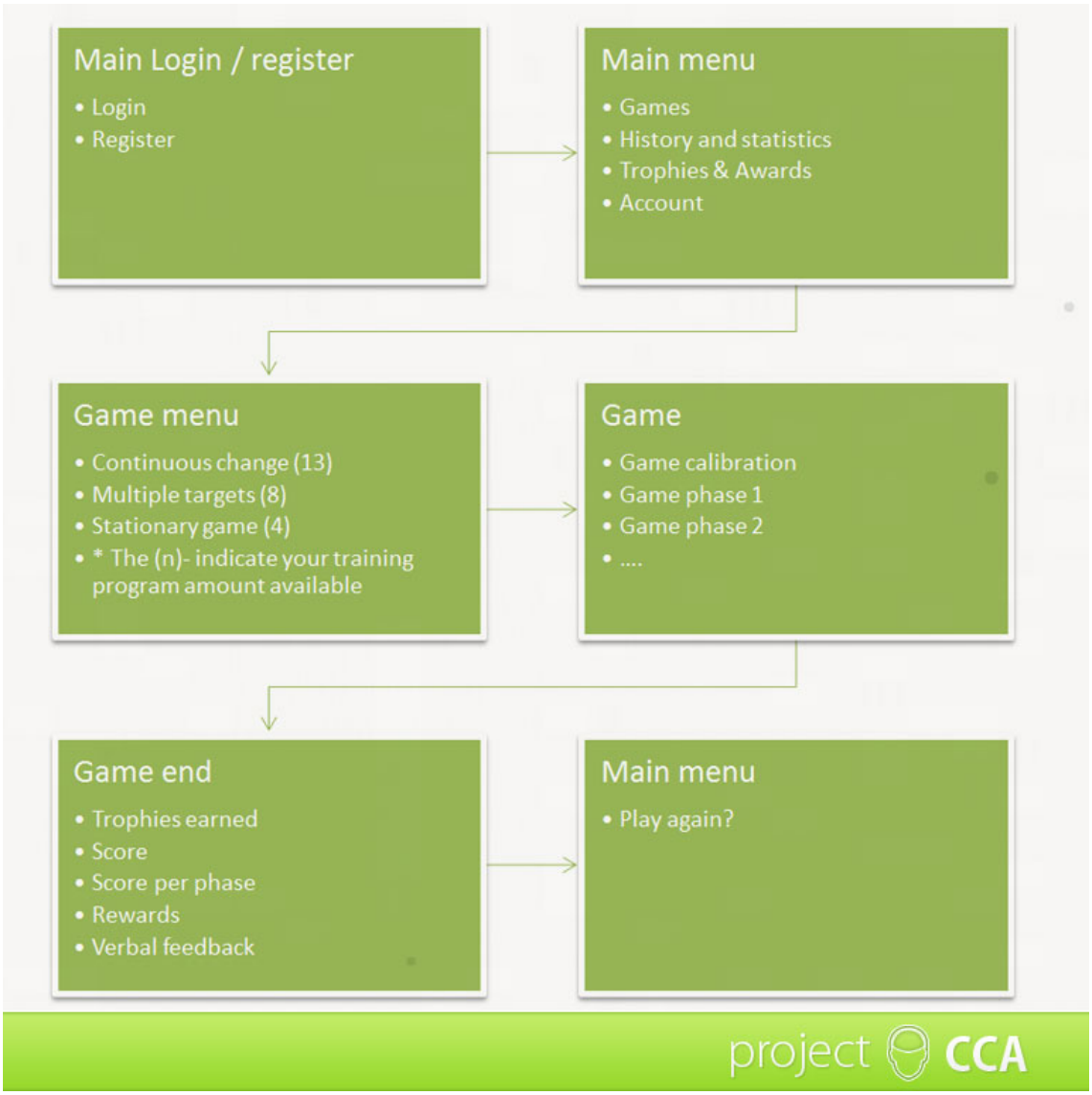


Figure 2. CCA user flow diagram

The general user flow in CCA is straightforward; users either have an account, or register for one, and log in to the game platform. The users have free choice over which order they plays their games in, and can choose from various available games to them

as seen in Figure 2. The different game modes depend on the users account and targets. Each user has a specific user account based training program that allows for them to play a certain amount of games per day.

Most games last around one minute and in each day a different set of games is played. The total duration of a training program is approximately 30 minutes per day. The games vary depending on the type, but contain various amounts of moving or static visual objects, called creatures.

The users' task is to perceive and/or memorize creatures or the target visual states thereof according to the instructions. The target state of a creature is a brief contrast-, color-, and/or shape-change. In some of the game modes, the target state change is relative based on a calibration round played before the actual measurement portion. Calibration changes the size of the target state so that the subject detects around 60-90% of the changes in a two creature game phase. The detected amount of target states is expected to get better with training.

When the target state is perceived, there is a limited window of opportunity for reaction, within which the user must react by pressing a key. (Configurable, but by default it is the space bar) The platform measures the hit rate and reaction time of the user per target state per creature on screen. The hit rate (HR) is the main measurement used in CCA to define user capability.

After a successful play, the users are given a summary of their performance. The performance metrics shown are based on the hit rate and reaction time of the user. The main numerical feedback given is called capacity. Capacity is defined as $C = 1/m \Sigma HRi * Ni$, where N is the number of creatures. Capacity is given as a total value for the playthrough and as a separate value for each phase of the game.

In addition to the numerical metrics, the users are also given trophies, achievements and stars based on their performance. These categories provide user friendly feedback on how well the users are doing without the need for detailed metrics. The detailed metrics are available, but not shown as the default content. The purpose behind the trophies is to empower the user and give them solid, clear and maintainable goals for their training.

After the game end screen, the user returns to the main interface of the CCA platform, where he can see his statistics, review his achievements, change his information and play another game.

2.3 Target user group

The projects target user groups were divided into two sections: The general public, and the clinical trials.

The general public users were divided into two sections: Elderly citizens whose interests would be to both test attention and working memory as well as train to improve them, and school aged children who would benefit from attention disorder testing with an automated platform. The current methods for testing and diagnosing attention disorders is work intensive and expensive, and automating the testing would yield significant savings. These users focus was on daily life, and they suffered from no known disruptions in their cognitive capabilities.

Because of the scientific nature of the platform, and the precise data collected from our users, the platform could be used as a reference in clinical trials, to see how and if people improve their performance with the use of a medicine. This is important in studies of new medicine and helping people with brain trauma. The platform enables a way of measuring and making training programs specific to studying the target groups' differences with medicine and without.

Since a portion of potential users is visually challenged, significant emphasis was put on making the platform visually appealing and simple to use. The focus was on pleasant user experience, instead of pure efficiency. To make the users feel safe with the program, and to trust it, a solid user experience was necessary.

2.4 Project lineage

The work started soon after the first meeting, and the focus was on creating a prototype of the basic rendering mechanics as a proof of concept. This was necessary as the rendering power of browser based solutions is still far behind native language

based compilers, and it was needed to see if it was possible to create the rendering engine on such platforms.

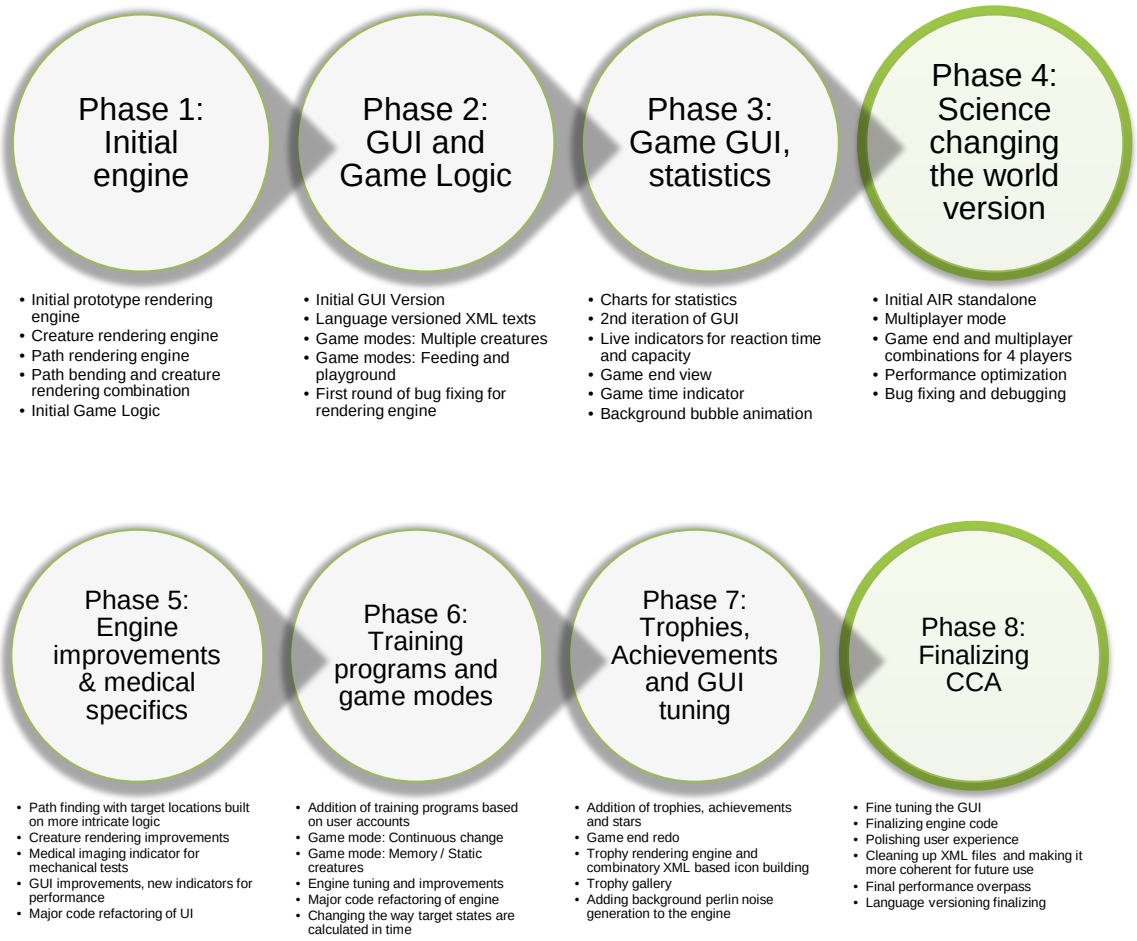


Figure 3. CCA Project phases

As seen from Figure 2, CCA was a comprehensive project spanning many years. The project was divided into separate phases, each phase consisting of a number of sprints. The first priority was in the rendering and game engine construction to make sure it was viable to build a full scale platform with the technologies chosen. After a few weeks of development, a first prototype of the rendering engine was developed and published. After some tweaking of parameters and bottleneck analyzing, it was considered to be a viable solution, and a sufficient base for building the platform.

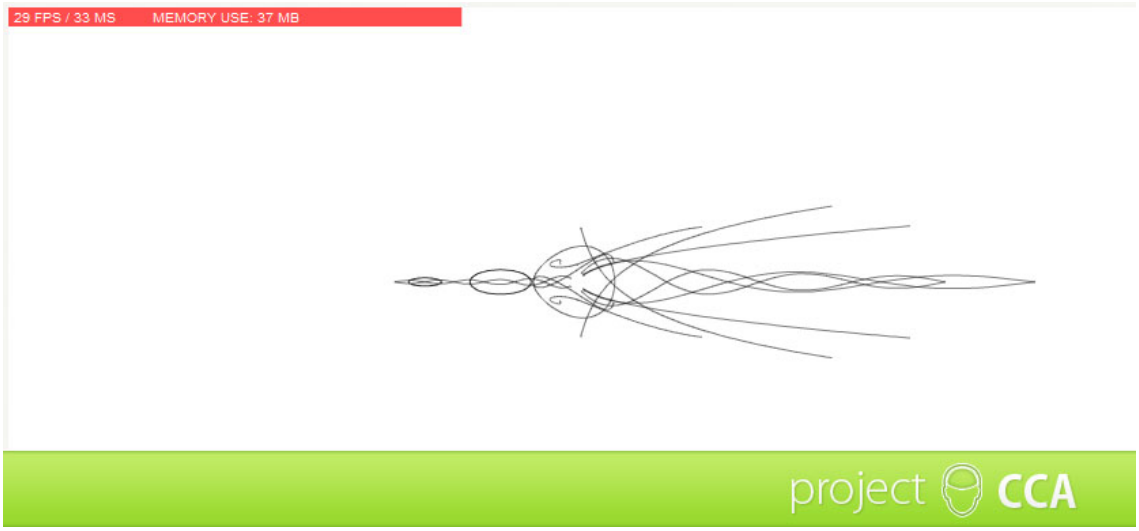


Figure 4. First CCA output

Figure 4 shows the first version of the platform rendering engine. The result contained only one stationary creature, comprised of a nominal amount of points, rendered just so one could see how it would work and what the possible performance bottlenecks were. The initial version rendered the stationary creature with barely 30 FPS, and was quite heavy on the CPU.

After the POC was finished, the full development of the platform began. Development was done in an agile way, with sprints of 2-4 weeks [1], where one aspect of the platform was tackled at a time. Since the most difficult part of the project was the rendering engine itself, the first 4-5 months were spent solely on creating, optimizing and fine tuning the first version of the core rendering engine.

Once the basic engine was built, a fast pass over visual style was done for the project, an initial GUI for test users and funders to see and test the progress. Work was started also on doing reusable UI components for various user prompts and game information needs. Beyond the GUI needs, the first game mode logic was coded, and the first actual play through of CCA was possible.

Figure 5. First version of the CCA UI

As depicted in Figure 5, the first GUI version took the metaphor of the circles quite far, and was oriented around cells that contained smaller cells. The first version of the UI was fully implemented, and contained growth animations for chosen cells and a small amount of fluid dynamics, however it was quite confusing to use, and more of a game in itself.

After the initial GUI pass, the focus turned to the game modes of the system. The final version of CCA supports 4 different game modes for cognitive gaming; from a single creature reaction observing to feeding multiple creatures. In addition to building the initial game modes, the first bug fixing and improvements for the rendering engine were implemented.

Once the game modes were done, a sprint was dedicated for properly introducing a full GUI to the platform, and further separating UI components. After the initial logic and games and UI were done, the focus was on Login and register functionality as well as

user support mechanisms such as account handling and performance graphs and information graphics in general.

Figure 6. The Beta release of CCA

After the phase three of the project, the official Beta release of CCA was ready. The beta interface can be seen in Figure 6. The platform supported all the basic game modes, user accounts, a full play through experience and statistics to prove it. Once this version was done, the work for the Heureka science exhibition called Science Changing the World started. It was a separate version, completely independent from the back-end and running a local copy of the rendering engine modified for 4 player multiplayer needs.

After the Heureka version the phase 5 concentrated more on improving different parts of the platform at a time. The addition of medical imaging indicator helped automated testing of the platform, the improved target location in path finding enabled smoother movement and the GUI code had a thorough overhaul to separate it more from user logic.

Phase 6 was the 2nd major round of refactoring and improvements for the platform. The introduction of training programs changed the way the platform worked, and the new game modes added a lot of variety to the package. Engine code refactoring helped further development of game modes especially, and the target state calculation change was done in order to make it more manageable for the server to fine tune the user account based training regime.

In Phase 7 the way users were rewarded and progress communicated got a much needed overhaul, changing the direction of the feedback to a much more user friendly result. In Phase 8 the release version of the platform was finalized, the final GUI was put in place, the performance was verified and the user experience perfected.

There were a lot of features and ideas in the beginning, some of which were discarded along the way as unviable for us to implement with our schedule and needs, among those features were evolutionary algorithms in creature generation.

2.5 Technology choices

The technologies enabling browser based heavily mathematic rendering for the front end were evaluated before the project begun, and a choice was made in the beginning on which platform to pursue the solution on. Since much of the projects core engine is based on mathematics that are rather universally supported by the different platforms, the porting of the platform to a different technology is not considered to be an impossible choice in the future.

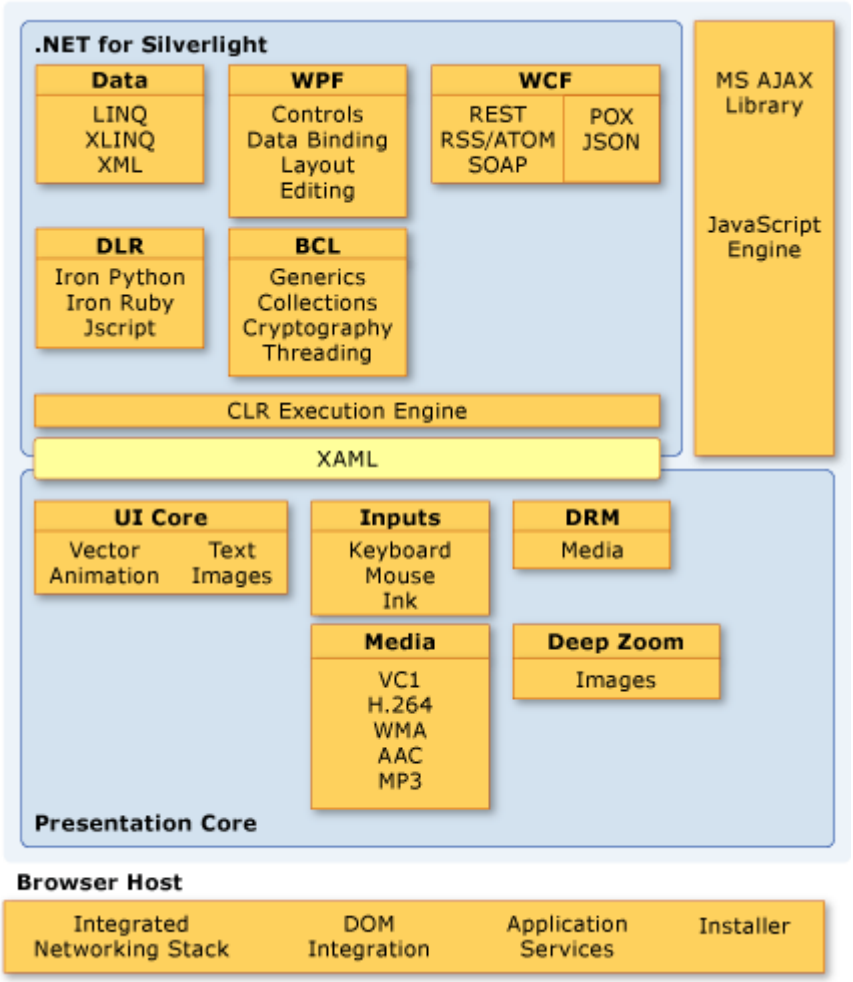


Figure 7. Silverlight technology stack [2]

One of the technologies evaluated was Silverlight - the RIA solution from Microsoft. It is a browser plugin similar to its main competitor Adobe Flash. An overview of the Silverlight technology stack is seen in Figure 7. The runtime is based on the popular C# language, and UI components are marked in the more human readable XAML. (Extensible Apllication Markup Language) XAML is also used in Windows presentation foundation (WPF) and .NET framework of which Silverlight is a sub sect of. Silverlight implements the same version of the Common Language Runtime (CLR) as the .NET framework 3.0.

Silverlight offered similar performance to Adobe Flash, and being a part of the MS solution package, it comes with a wide range of solid development tools and support with Visual Studio family of products. Silverlight supports the integration of multimedia, graphics, animations and interactivity in a single runtime environment. Similar to Flash,

Silverlight also supports vector rendering in addition to the typical Bitmap based rendering.

The main hindrance in Silverlight and one of the reasons it was not used is that it is far behind Flash in adoption rates. Where the Flash Player 10 adoption is around 98.7% in mature markets [3], Silverlight's adoption rate was around 22% [4] at the time of the project start. Whilst being impressive in its abilities, the Silverlight technology was rather young in 2009, and sorely lacking in features.

The technology chosen was the de facto standard for rich content online in 2009; the Adobe Flash platform. As mentioned in relation to Silverlight, it is the most widely spread browser technology in the world, and has a mature and solid development environment and support. Flash supports vector rendering as well as bitmaps, and the newest versions from 11 onwards support using OpenGL based GPU rendering as well. When the project started the player version target was 10.1, but the release version of CCA is targeted for the Flash Player 11 platform. [7]

Whilst requiring the installation of a plugin, the wide spread adoption of Flash made it a clear and easy choice for a consistent user experience across platforms and browsers. There are up–to-date versions of the player for Windows, Linux, OSX, and Android platforms.

The heralded revolution of standards based online development – HTML5 – was not around in a prominent way back in 2008. After studying the possibilities of using HTML5 and Javascript based solution as an open alternative for the plugins, it was found that the performance of Javascript VMs and especially the rendering performance was subpar when compared with the plugin based technologies.

This has changed a lot from the year 2009, and the newest JIT(Just in Time) compiler in Google Chrome is starting to surpass some of the plugin solutions in pure crunching power. However, the rendering capabilities of browsers vary a lot based on the versions, and the adoption rate of new browsers remains one of the larger obstacles in generating modern standards based rich solutions online.

Being a consumer mass market solution, we aimed at all three big players in the market on consumer platforms: Microsoft Windows family, Apple OSX, and Linux based

operating systems needed to be supported. The support for our decided solution platform – Adobe Flash, is wide and envelopes all our target platforms.

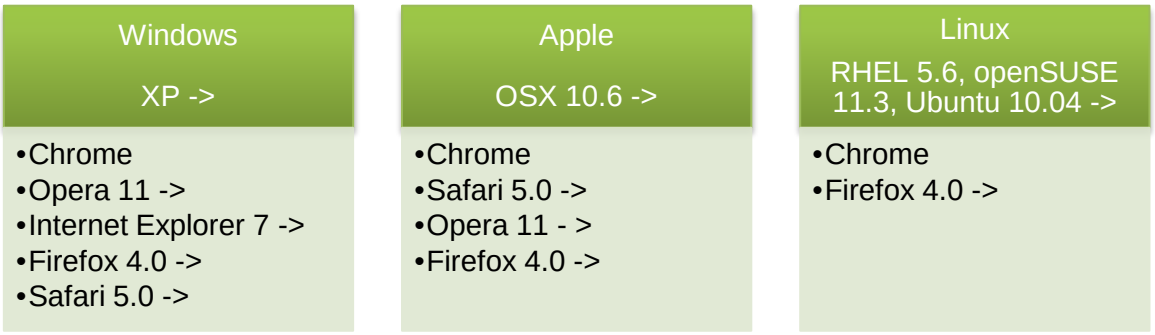


Figure 8. Target platform chart [6, 7]

The benefit of choosing a browser ran online platform was also the generic platform independent nature of the technology. With a single solution, we could reach and deliver a fully functioning solution to the whole user group. As seen in Figure 8, the project supports the main versions of Microsoft Windows from XP onwards, OSX from 10.6 onwards and Linux Red Hat Enterprise 5.6 or later, openSUSE 11.3 or later and Ubuntu 10.04 or later in both 32bit and 64 bit varieties [7].

2.6 Challenges and key concepts

In principle a two-person-project, the CCA was a significant undertaking. The viability of using browser based technology for creating a complex mathematical rendering engine for clinical and consumer use was something not many had done before.

The biggest challenges of the project were on the pure engine building level: The rendering of creatures in real time without pre-rendering or cheating in the rendering pipe line, and the performance of path finding logic and path wrapping. The mathematics used were performance intensive, and the rendering of several creatures with hundreds of rendering points and surfaces in addition to the path creation with continuous Bezier curves was difficult for a CPU based rendering engine.

The goal of building a medical platform for consumer use had its own challenges. The combination of a number based, mathematics oriented solution and attractive user experience provided many obstacles. Keeping the UI code separated from the engine, and ensuring a solid separation of concerns was essential in building a working platform.

On the platform front the separation of as much presentation and user logic to the back end for control and data based optimization of training posed several challenges to overcome. The parameterization of almost all UI elements and processes from game modes and their configurations to even trophy icon generation was a complex issue to solve.

Key concepts

Cognitive Gaming	A type of gaming and exercise that is designed to help and improve cognition. Used in the aid of recovering from brain trauma, also used as recreational activity believed to be beneficial to the mind.
RIA	Rich Internet Applications. Browser based technologies that enable creation of desktop like features in the world of internet. Example technologies include Flash and Silverlight.
Modern Browser technologies	A technology stack that contains all browser ran technologies. HTML5&JS, Flash, Silverlight, Java (?)
Neuroplasticity	The theory that the brain is capable of physical change and improvement based on outside stimuli
Creature	The main target in the project CCA. A collection of mathematical formulas that are rendered based on a path finding engine to represent an organic “creature”. A single unit target of the platform.
Bezier curves	Smooth parametric curves based on 2 control points and a start point and an end point. Essential in project CCA.
User experience	A combination of usability, user interface, interaction design, information architecture and animation to cre-

	ate a complete use experience for a user.
Platform	A combination of technologies that form a coherent, reusable and deployable whole. An extensible combination of modules that works as a basis for building content on top of.

Figure 9. Key concepts of the study

Key concepts used in this study are shown in Figure 9. They cover the areas of cognitive gaming and modern browser based game development as well as the principle technological choices in organic rendering.

3 Cognitive gaming

3.1 History and definition

Cognitive gaming is an exciting new area for consumers and scientists alike. There have been various studies and trial projects about using games as platforms for advanced learning, learning through play and using virtual worlds as class rooms for learning. Cognitive gaming takes the elements of gaming such as repeatable tasks, reward models, user tracking and fun of play and combines them with medical research and recuperative methods for a game experience that also benefits and improves cognition.

The basis for all brain exercise, games and all is the concept of neuroplasticity, or brain plasticity; the ability of the brain to change physically throughout life, in response to stimuli. The times when changes happen in brains are in the beginning of life, when injury hits the brain, and whenever something new is learned and memorized. [8]

Up until recently it was believed that the connections in brains remain fixed with age, and physical changes are impossible. However, recent studies have shown that brain keeps on changing through learning and stimuli throughout our lives [9]. As an example when comparing professional musicians to amateur musicians and non-musicians, the actual physical volume of grey matter in areas involved in music, such as motor regions, anterior superior parietal areas and inferior temporal areas was larger with the professionals who practiced over one hour per day. [10] These changes were also greater when measured over time.

Another example came from a study done on extensive learning with German medical students. They used medical imaging to monitor the brains of the students before their medical exam and after, and compared the results to similar students who were not studying at that time. The students' brains showed anatomical changes in grey matter in different areas of the brain, including the parietal cortex and posterior hippocampus, parts of the brain known to be involved in learning and memory [11].

Despite the recent studies and interest in training programs to be used, there have been very few long term studies in the effects of cognitive games and training. While there are studies that show the short term implications of training [12], especially on

those suffering from early stages of cognitive impairment, there have not been sufficient enough studies to show whether the training can postpone the effects of such impairments as dementia [13].

3.2 Current market situation and future

In 2005, the size of the brain health market globally for software and biometric applications was estimated to be around 210 million dollars. The estimated value of the market in 2012 is over one billion dollars, and by 2020 it is estimated to reach six billion dollars in value. [14] This rapid growth comes in part from recent research, and in part from many medical professionals and researchers trying out the possibilities of using their research in helping people on the consumer market.

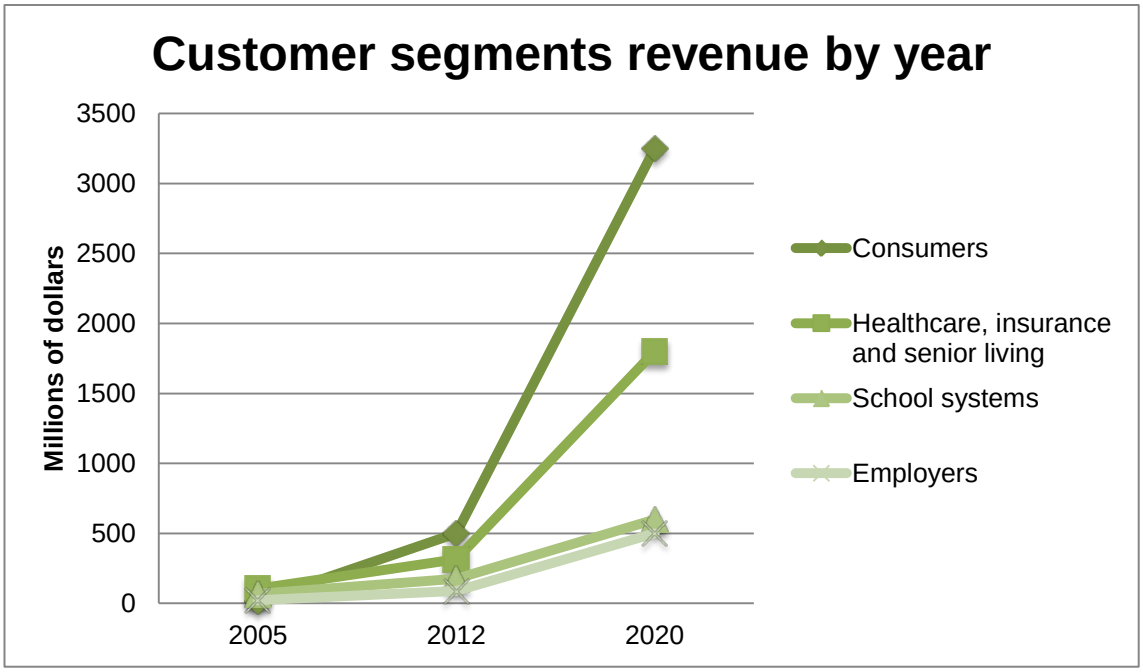


Figure 10. Estimated growth of the market per customer segment [14]

As seen in Figure 10, the biggest growth expected to happen in the growth of cognitive gaming and brain fitness is the consumer market by a large margin. The growth of self service training portals with training regimes directed for home use and self-improvement are already growing fast, and are expected to do so in the future as well. The other big growth area is in the area of insurance and health care as well as elder living. The benefits of preventive training in age related deterioration are substantial, as well as in the rehabilitation of people with brain related injuries. The savings generated

by such actions could be monumental. Aside from the main growth areas, the uses of cognitive games in school systems as well as employee care are expected to grow.

The cognitive game market is divided into 2 different parts, the pure software products and the biometric (applications that require actual hardware to measure physiological responses) applications. Examples of biometric products include products that measure hart rate variability or brain activity through EEG (Electroencephalography), the recording of the brain's electronic activity over a short period of time through the scalp. Our reference project belongs to the software category, and does not need any hardware to function.

3.3 Products and competition

In the area of cognitive games the reference project lies, the focus on perception, attention and working memory, there are various competitors and products on the market. To keep the subject more valid and tied to the subject of this study, the focus is on online, browser based cognitive gaming platforms and their use cases, technology and popularity.

The biggest company in the online cognitive gaming market in 2012 was the Lumosity.com. Lumosity is partnered with researchers at Berkley, Harward and Columbia and works with numerous health care organizations to help create cognitive gaming experiences. The service has over 25 million users, and it provides comprehensive and personalized training programs based on user accounts. [15]

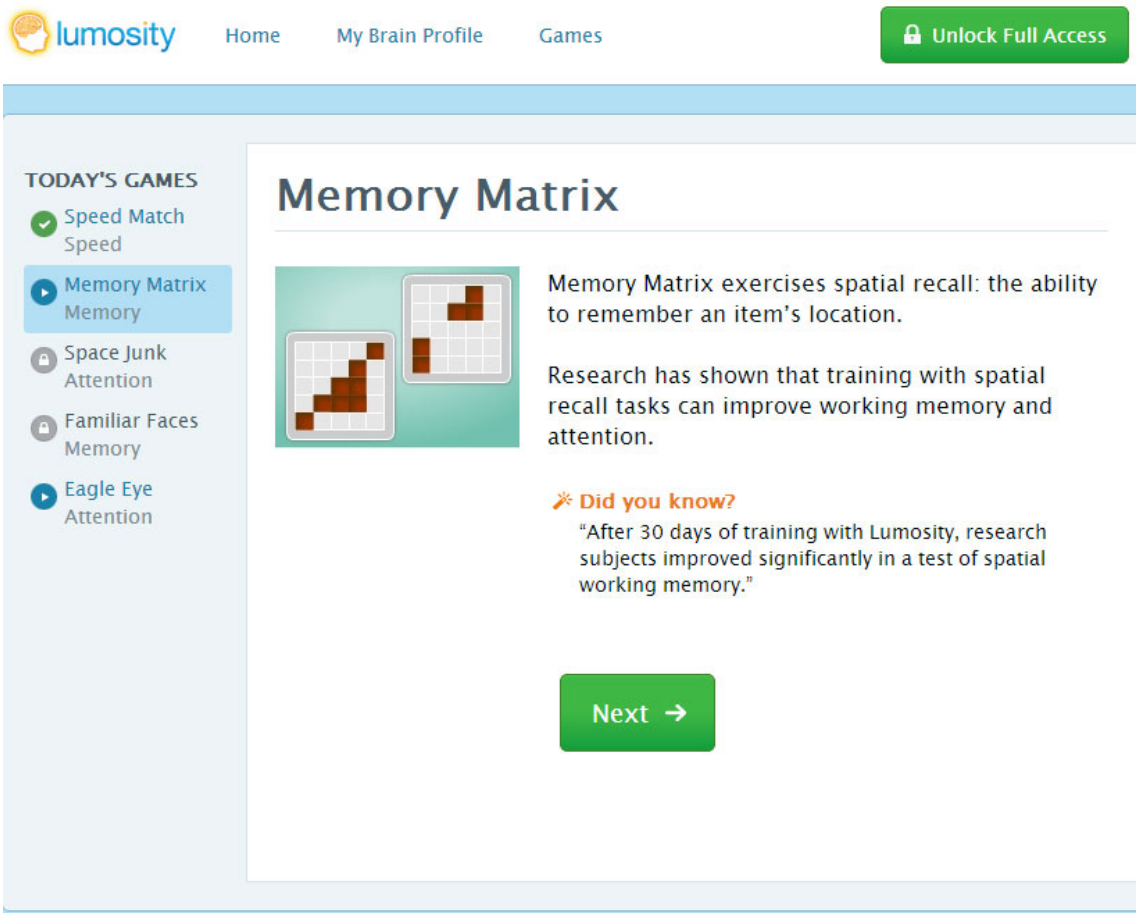


Figure 11. Lumosity gaming platform view [15]

As seen in Figure 11, Lumosity online training program conveys facts and scientific information about what the tasks you are performing at the moment provide, and manages to create a solid user experience with enough information and play to make it interesting. The training program is fitted to your needs based on a simple questionnaire. The first steps in Lumosity are free, but after a few games you get to a point where you cannot benefit from the service without subscribing.

Technology wise the frontend base portal of Lumosity is based on standards based HTML5 and CSS3, providing the general test framework and admin functionality in the portal. The actual games themselves are made with Adobe Flash technology, similar to the reference project. The games are rather simple in function, and are very event based in nature. In addition to the web interface, Lumosity also has a mobile application available for the Apple iOS platform.

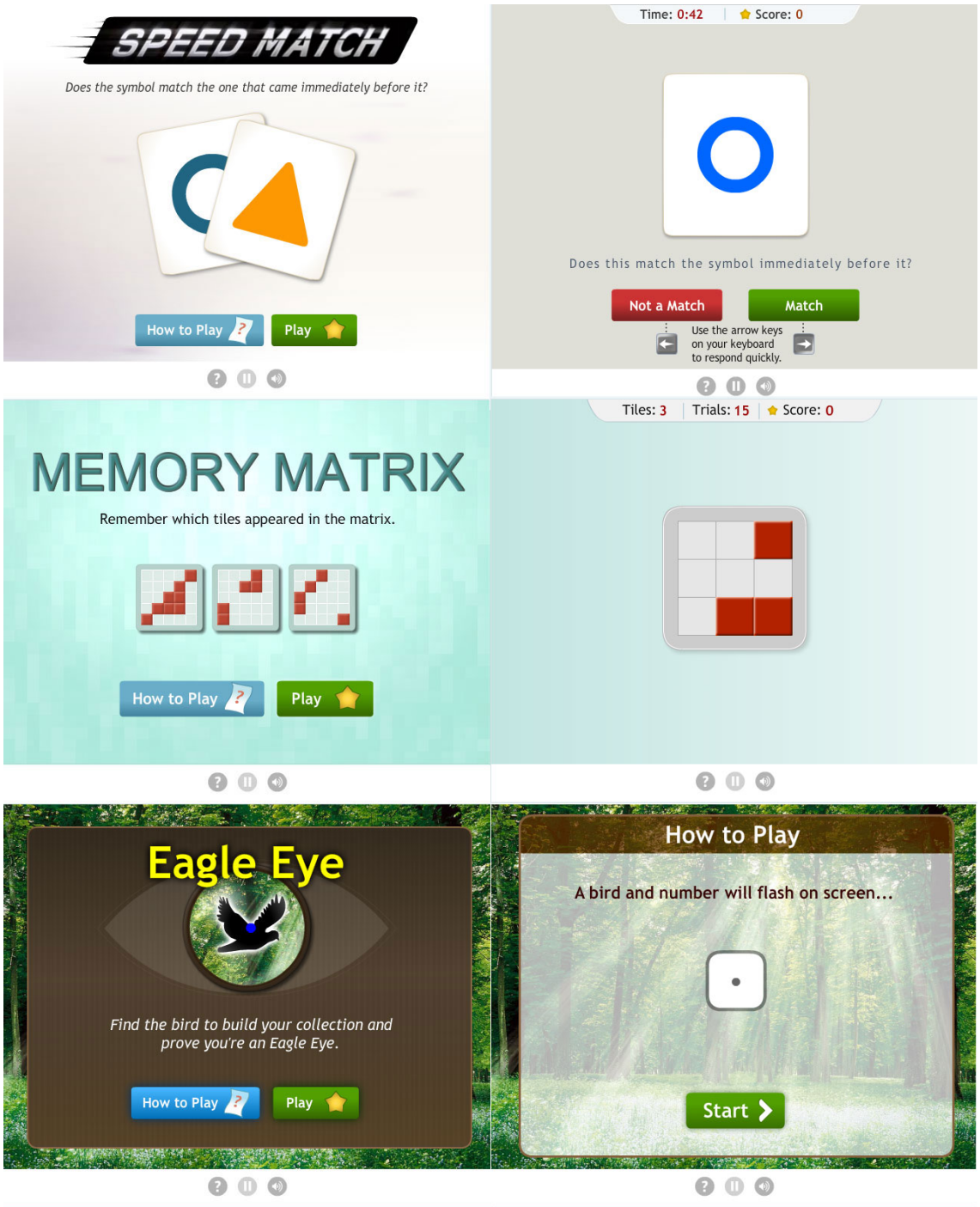


Figure 12. Lumosity game application comparison [15]

From a user experience and game design point of view, the portal comes across more as a collection of different games and a framework that ties them together, as seen from Figure 12. The different games use different visual cues and styles based on the subject matter, and are not uniformly under the same visual design aspect as the main Lumosity portal. However, they do contain repeating elements in the introductory controls, to provide similar functionality across games.

In addition to providing a cognitive gaming service, Luminosity has scientists actively working in the field of research and finding out how to best use cognitive gaming to benefit the human condition. In a recent study, it was shown that there was improvement in the working memory and visual attention of the target group when using a web based training application outside of a clinical trial setting [16].

The other competitive platform taken as an example in this study is the My Brain Solutions portal at www.mybrainsolutions.com. Similar to Lumosity, the portal provides a brain assessment in the beginning, based on which it generates a user profile and a training program for you to follow. Unlike Lumosity, the brain assessment is a quite a comprehensive test of memory, comprehension, emotion and other cognitive functionality, and lasts around 30 minutes up front. [17]

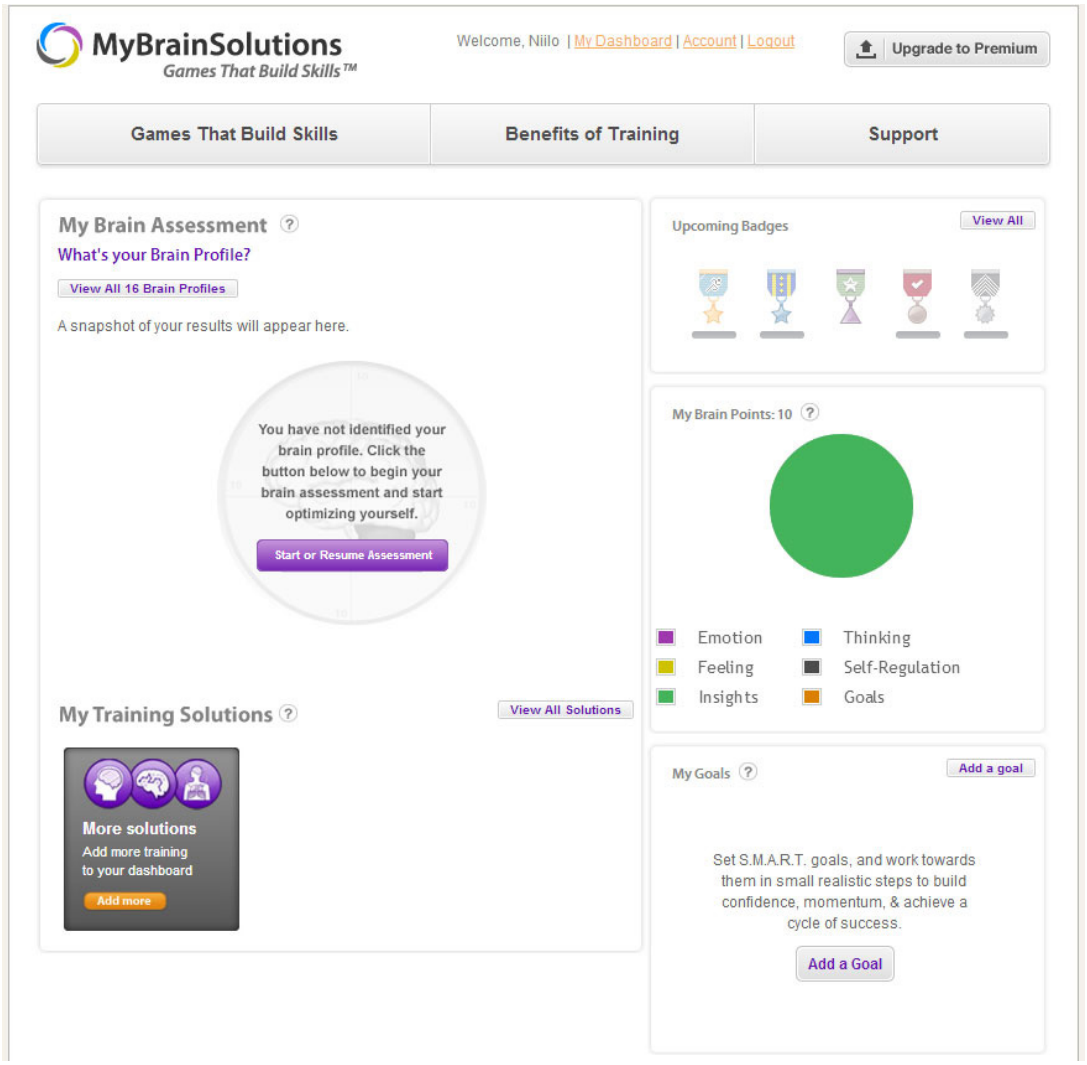


Figure 13. My Brain Solutions portal [17]

As seen from Figure 13, the My Brain Solutions contains a personalized training solution, as well as charts on how the users brain and performance range on the variety of test subjects. In addition to points which Lumosity used, in this platform the user also has badges, a reward mechanism similar to achievements to convert the arbitrary numbers and progression into more human readable terms. The platform allows you to set your own goals and encourages you to set actions for yourself to keep you busy.

Technology of the My Brain Solutions follows that of Lumosity and others, the main site is a web portal built on web standards, while the individual games are based on Adobe Flash technology. In addition to the web portal, My Brain Solution has various applications for different mobile platforms, targeting a specific feature, such as MyCalmBeat that focuses on lessening stress and increasing focus through slow breathing.

Figure 14. My Brain Solutions UX example [17]

In comparison to Lumosity, My Brain Solutions has a more unified gaming platform, with all the games showing similar introduction screens, button layout and statistics / in game information when doing tasks. As seen in Figure 14, the different games all feel as if they are of similar family and go well together visually with the main visual identity

of the portal. From initial testing, the games seem to have more complex interactions in them as well.

From the more entertainment oriented area of gaming recent years have shown increased popularity for titles such as Brain Age (2005) for the Nintendo DS console that sold over 18.96 million copies, as well as its sequel Brain Age 2 (2005,2007) 14.83 million copies to date. [18] While these products were very popular, and studies have been done in order to evaluate their possibilities [19], Nintendo has distanced itself from the use of scientific proof of benefit in the games. [20]

Figure 15. Brain Age 2 from Nintendo

An example of the Brain Age line of games can be seen in Figure 15. In the Brain Age games the user is expected to play a small amount each day, according to a training program. This is a common approach to brain fitness, and is employed in the reference project as well. The tasks a user performs orient around simple calculations and mathematical questions, memory exercises, Stroop tests as well as Sudoku puzzles.

3.4 Common test types

The cognitive gaming platforms focus on a certain set of games, each targeted for training different parts of the cognitive system. These areas are close to the ones used in clinical neuropsychology, but understandably are not as exact or as precise due to the consumer market approach. These target areas can be roughly divided into three categories: memory, attention, and executive function. Some platforms such as My Brain Solutions also contain tests related to emotion and human understanding.

Memory games test and train our ability to memorize items, words, patterns and other objects. There are different variations of memory games; a commonly used one involves working memory or short term memory. These tests are called N-back tests where a user is presented with a sequence of objects, and the task is to react when the current object matches the one shown N objects before [21]. The N-factor can be adjusted to make test more or less difficult. Short term memory tests are well suited for consumer market testing; the test gameplay does not take overly long and the results are parsed live. An example of an N-back game is seen at the top in Figure 12, as well as another type of memory exercise seen in the middle.

Attention games focus on our ability to perceive and react to our perception in a given controlled environment. The idea of attention tests is to train the cognitive processes of focus and visual search as well as long term attention. The tests often focus on pattern recognition as well as visual attention. The way to test attention implemented in the project CCA is to have multiple moving objects on screen that the user has to follow, and react whenever the object achieves target state. Target state is a change in the shape, colour, contrast or form of the object. Numerous examples of attention tests are found in the reference project platform, as well as the example game at the bottom of Figure 12.

Figure 16. Example of an executive function game [17]

Executive function is an umbrella term used for various cognitive processes and sub processes working together.

> Executive functions are those involved in complex cognitions, such as solving novel problems, modifying behaviour in the light of new information, generating strategies or sequencing complex actions. [22]

In the reference platforms for brain gaming, the tests for executive function include games related to arithmetic, quantitative reasoning, planning, verbal fluency and task switching. An example of an executive function test can be seen in Figure 16. This task is about connecting a series of nodes with as much area as possible without ending in a dead node, a task that requires both planning and reasoning.

The reference project is mostly based on various attention games, as well a testing mode for N-back tests to improve your working memory and a continuous change mode that falls in part into the domain of executive function.

3.5 Game engines

The term game engine came to be around mid-1990s in reference to 1st person shooter games such as Doom by ID Software. Game engine is a platform for game development that employs data driven architecture to create reusable software components, such as a three-dimensional graphics rendering system, collision detection system and a physics simulation. [23] The line between an engine and a game is often blurry, and to date there are few engines that can adapt to more than a few genres of games.

The engine defines how the game is rendered on screen, and influences a lot of the design around the core concept of the game. Some game engines include tools for building the actual game content on top of them, such as Unity, Unreal Engine or CryEngine [24,25,26]. This is often called scripting, since it is most often done using a scripting language. Many of the engines contain their own scripting language [26], or use a set of commonly supported high level languages; Unity includes support for C#, Javascript and Python based Boo [24]. Some engines provide only the actual rendering engine for abstracting the hardware level, such as Ogre. [27]

Game building in general has gone through a renaissance of sorts, where the middle ware is increasingly important in creating game experiences in the industry as well as in the education of game development [28]. A large portion of games today use middleware such as Bink video [29], Havok physics engine [30] or the Adobe Flash harnessing Scaleform [31] for game user interface building. These middle ware programs provide a necessary relief from the complexity of building a modern game, each solution doing its part, providing a polished and optimized way of handling one aspect of the game.

In the field of medical and cognitive gaming, some are using the same game engines as entertainment oriented gaming uses [32], while many use their own engines based on a higher level programming environment such as Flash, Silverlight or Java to create their own base engines. Many cognitive games are not yet complex enough to have the need for specific engines, but with the increase in demand, development budget and competition, it is only a matter of time.

When talking from a more conceptual level of how game interaction and gameplay is handled, there are two types of game environments; static and dynamic. In the context

of this study this distinction is defined as the way game events are controlled in the game, and the way the engine handles rendering and interaction.

3.5.1 Static game engines

The traditional game engine is static, as in everything in the engine is defined by an artist or a developer, every move you make is the result of a careful calibration, iteration and concepting. Static engines allow for absolute control for the games designers, and are in many cases more optimal from rendering and calculations point of view. In the comparison products Luminosity, My Brain Solutions and the Brain Age all fall into the static game engine section. They are all also very event based in their approach, everything happening in the games is not based on real time calculations, but on specific events happening at specific times and the reaction to those events.

The downside of static game engines is that in controlling everything they lose the element of surprise in some ways. When everything is designed, there are no happy accidents, nor odd gimmicks that a player can find that cannot be reproduced, and everything in general works as it does, always the same reliable way. Many static engines also employ a basic physics model with hard-body physics. In such an engine the physical values are tweaked and set by the designers in a way it replicates some simple form of physics, but does not really allow for proper surprises.

3.5.2 Dynamic game engines

While being more obscure, and harder to control, dynamic game engines base the world they render on a set of rules. These rules may be physics, different path algorithms and so forth, but they all have in common the lack of direct control over what happens inside. In a dynamic engine the game designer gives the objects targets, creates behavior and sets up boundaries, but how the engine executes these is left for the engines internal system to decide.

Some games use the dynamic game building in ways to create randomized environments, procedural content based on a set of rules. Games such as Diablo [33] and Minecraft [34] have used procedural generation to create whole levels, or in the case of MineCraft, a whole planet. The traditional problems with computer generated content in

games have been the repetition of content and unnatural and uninteresting combinations. Some newer games use this dynamic or procedural generation in the generation of items for the player to use, such as the weapons in Borderlands and Borderlands 2 [35].

In gaming physics, a more solid way of doing physics in a dynamic way is called soft-body physics, where a physical object is a collection of its sub parts physics. This is immensely heavy in calculation, and is only now emerging with the new Cryengine 3 [25] and other new game engine platforms. The project CCA is not in a world of fully fledged physics, nor is the world building in any way overly complex, but the engine and the way the creatures work is entirely dynamic.

4 The User Experience approach

4.1 Definition

User experience (UX) has many different definitions, depending on the subject matter it is related to. A classic example is from 1996, from the first annual ACM Interactions Design Awards:

> “By “experience” we mean all the aspects of how people use an interactive product: the way it feels in their hands, how well they understand how it works, how they feel about it while they’re using it, how well it serves their and how well it fits into the entire context in which they are using it.“ [36]

The Nielsen-Norman group define user experience as:

> "User experience" encompasses all aspects of the end-user's interaction with the company, its services, and its products. The first requirement for an exemplary user experience is to meet the exact needs of the customer, without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use. True user experience goes far beyond giving customers what they say they want, or providing checklist features. In order to achieve high-quality user experience in a company's offerings there must be a seamless merging of the services of multiple disciplines, including engineering, marketing, graphical and industrial design, and interface design. [37]

In principle, UX is everything a user feels, sees, experiences when in contact with a company and/or a product. The point Nielsen-Norman make is to go beyond the needs of the user, the realization that UX is far more than providing the user with what they need, but how they experience it as well. The joy of use, aesthetics and message all combined in a thought out package.

User experience design (UXD) is an umbrella term that covers the different facets of expertise required to create a wholesome UX. The roots of UX design come from Human Centred Design (HCD). HCD can be summarized as:

> - Positioning the user as a central concern in the design process
> - Identifying the aspects of the design that are important to the target user group
> - Developing the design iteratively and inviting users’ participation
> - Collecting evidence of user-specific factors to assess a design [38]

In addition to the methodology of HCD, UXD builds on top of HCD with more complex cultural and business factors. While traditional HCD based usability factors were about performance and smooth operation, UXD brings along aspects of social interaction, the importance of aesthetics, both very culturally complex and context requiring concepts.

Since both UX and UXD are very broad subjects, in the context of this study, the focus of UX is confined to how our project platform uses UXD to create a suitable UX within the platform itself. The areas presented in relation to the reference project are interface design / aesthetics, interaction design / movement, information design / communication, and playability in the cognitive gaming context. There are also various UX concepts such as presence, immersion, flow, fun, involvement and engagement that try to describe the UX in games. [39] These theories, while important, are not discussed within the scope of this study.

4.2 User experience in gaming

What the success of iOS and mobile platforms has taught us; user experience matters. People are willing to pay for better suited and thought out solutions. The wake-up call provided by the mobile markets sudden rise has not gone unnoticed in the world of gaming. Especially in large companies there has been a surge of focus on hiring user experience designers and front end designers to focus on the neglected parts of games, the game UI, and the interactions with it.

Games have been on the forefront of human computer interaction for the past 20 years, as the playability of a game, essential to the overall experience is uniquely a quite complex endeavor from a user experience point of view. But while focusing on playability inside the engines and games themselves, they often forget the first thing a user sees when they enter a game, the user interface of the game itself. The navigation, settings and save/load; these top level controls affect a lot on how one perceives the game experience.

Often these are badly designed, riddled with flaws, inconsistencies and hiding of information. It is not uncommon in games that you are not sure what will happen when you adjust a setting, or a parameter. The inclusion of UXD methods and experts has made games more user-friendly and easy to use. In building a platform for cognitive gaming, the unified UX for the games within the platform as well as the overarching user interface was an important part.

4.3 Unified visual style

As has happened throughout time, the advance of technology and tools for creating visual communication have changed and evolved at a rapid rate in recent decades. Despite this change, the essence of graphic design remains unchanged; to bring order to information, form to ideas, and expression and feeling to artefacts that document human experience. [40]

The international typographic style has been a significant and influential style in graphic design for the past 50 years. Its origins come from Switzerland and Germany in the 1950's and it is also known as the Swiss Design. The visual characteristics of the style thrive for unity though a solid mathematically based grid, objective photography, sans-serif typography and a solid copy that presents information in a clear and factual manner. [40] More than pure style, the importance of the international typographic style lies in the attitude and approach its early pioneers adopted. The role of graphic design was formed more towards shaping information and communication then personal expression and artistic eccentricity.

This clarity of style and information is a basis of much of modern online communication, and it is the basis for the style in CCA as well. In addition to a clear and unified visual communication, in CCA the focus was also on the visual narrative: animations, movement and flow that affect how a user perceives a product.

4.3.1 Typography

Typography is the art of type, the act of arranging to make language visible. In the modern day the term envelops many crafts, from the traditional typesetters and compositors to graphic designers and artists. In the context of this study, the term is used to mean everything we do with type in a digital platform; Font, sizing, characters and legibility.

> *Typography exists to honor content*
> Like oratory, music, dance, calligraphy – like anything that lends its grace to language- typography is an art that can be deliberately misused. It is a craft by which the meanings of a text (or its absence of meaning) can be clarified, honored and shared, or knowingly disguised. [41]

In modern digital communication, the use of solid typography to create a unified and visually attractive, legible message is an essential part. It is used to both communicate efficiently, as well as to add a feel, personality and grace to the communique.

The technological choice in CCA, the Adobe Flash Platform is a good choice for working with typography, as it contains a much more advanced text rendering and font support engine than the traditional browser solutions. With the support of fully embeddable type, the platform could take use of proprietary fonts with full fidelity and control. With the recent advances in CSS type support it is even possible to use proprietary or special fonts on standards based online communication. [42]

In project CCA the main typography is provided by the fairly common Myriad Pro, originally developed by Adobe in 1992 and widely used by companies such as Apple, Walmart and Wells Fargo. [43]

Figure 17. Myriad Pro Regular Font rendering

Myriad Pro is a versatile sans-serif font family designed for mostly digital use. The family contains a wide variety of weights and widths to suit the needs of the CCA platform. It is a simple, elegant font with excellent readability. An overview of Myriad Pro font rendering with examples can be seen in Figure 17.

It is often a problem for digital platforms to provide similar design and typography in different languages, as the Latin based languages have their own typography, and the Chinese, Arabic and various other languages have their own typeset. Rare fonts support even the whole plethora of European language typesets, from Cyrillic to Greek. Myriad Pro is a good choice for a language versioned platform, since it provides a

complete support for Greek and Cyrillic characters, which enables the use of same visual fidelity across languages.

The Flash Platform also provides the tools to embed different character sets for the use of completely different language characters, such as having Myriad Pro for Latin based languages, and a specific font for the Chinese market. Which font to use is decided during runtime when dynamic content is loaded, based on the character sets involved in the UTF-8 encoded content.

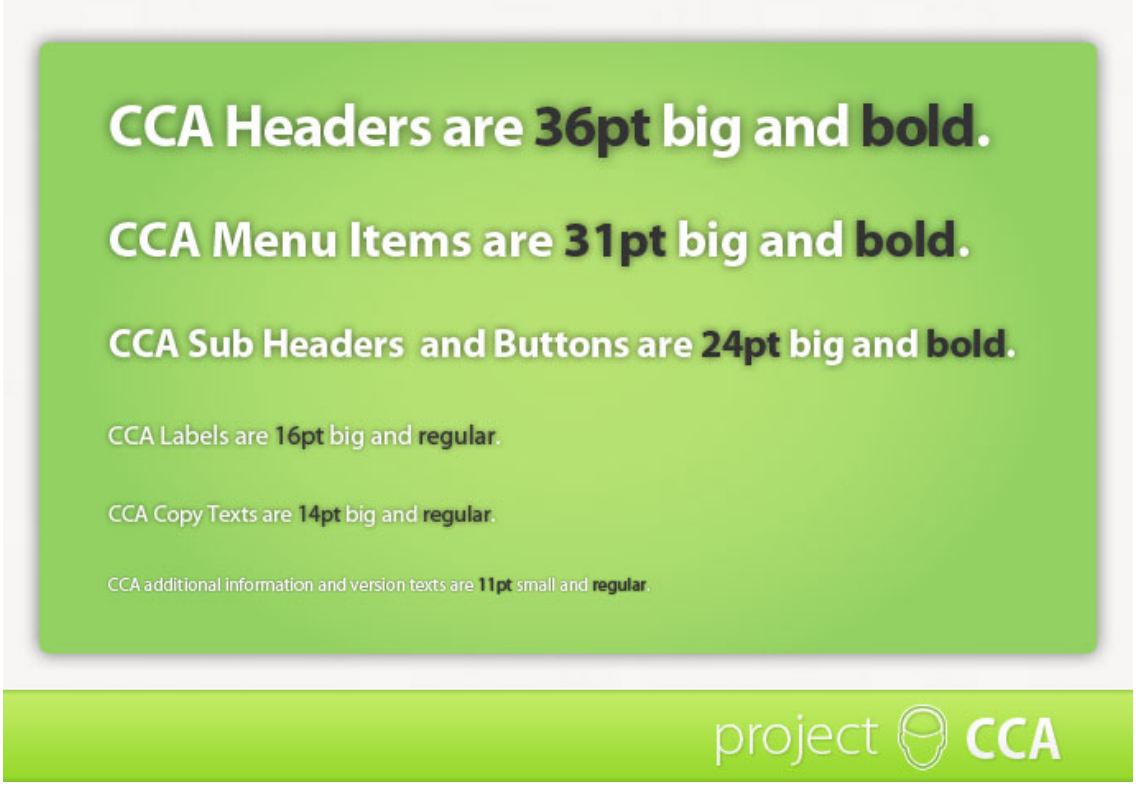


Figure 18. Font sizes in project CCA

As seen in Figure 18, CCA uses Myriad Pro in two different weights, regular and bold. The main headers for each screen are 36 points big, allowing for a clear distinction from the sub headers and regular copy text. As CCA is not a very text heavy platform, the most used texts are the buttons and sub headings. All elements that indicate something that can be activated or form a separate section of information are bolded for effect. All regular text, information notices, explanations and such are in a readable 14 pt size.

4.3.2 Composition and the use of space

In interface design the essential glue that keeps a view, a page, a dialog together is the composition, also known as the page layout. Composition can be thought of as the link between art and mathematics, the use of relations, numerical patterns such as the golden section, and geometric shapes to create a formula that organizes content in a meaningful and clear way. [44] Composition can be based on various styles, from a single visual to perhaps the most used element of composition - grid theory. An effective tool is also to use Gestalt Laws of grouping, the principles of how the mind organizes visual data, in creating coherent compositions. [45]

Figure 19. CCA centred layout example

In CCA the whole composition is based on a centred grid with a focal point always at the horizontal and vertical centre of the display area. All content containers and controls as well as all UI-elements are aligned based on the centre point. As seen in Figure 19, the centre based composition is followed by the main playground, the user menu at the bottom, and the game menu at the top, as well as the main information dialog at the

start of the game. During the actual game, the main visual time indicator is at the centre point as the most obvious visual element for a user.

An important aspect of composition is the use of spatial relationships. The space can be created by content - using images, texts, icons, lists, logos or just plain text – or it can be created by the space between content, called negative space or white space. The space can be actively used to create a point, or it can be passive, there just because the layout process requires it. [44] The use of negative space is essential in giving air and increasing legibility in digital design. It can be divided into two categories, macro white space – the distance between major content elements – and micro white space – the distance between elements within content elements, such as lists.

A challenge in composition in a digital medium is the dynamic nature of the viewer setups. The layout can be viewed with a screen from 1024*768px up to 2560*1440px, with varying pixel density (DPI). This creates quite unique issues for the use of white space, and the arrangement of elements. The traditional approach in digital design is to set a certain target resolution, a compromise that contains reasonable resolutions and provides a good enough result for those not matching the target resolution.

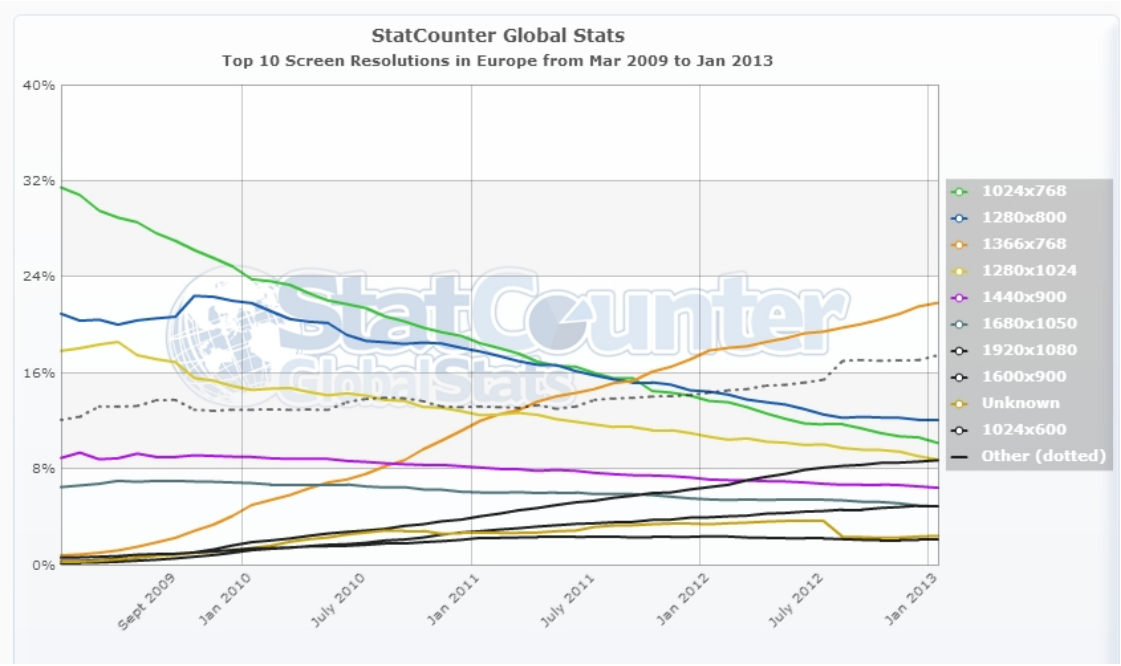


Figure 20. StatCounter screen resolutions in Europe [46]

As Figure 20 shows, the most popular screen resolution of users in Europe has grown from 1024*768 to 1366*768 in the last 4 years. Beyond just the resolution, the size to resolution ratio or DPI (Dots per Inch) has also grown and diversified. The rapid change in resolution, as well as the fragmented size variance has resulted in a situation where designing for just a single base resolution is not a preferable option anymore. This is true even without taking into account the boom in mobile browsing and devices with smaller physical size and screen resolutions.

As a response to the ever growing resolution and varying DPI, the modern way of doing digital layouts is in a state of transition; an ever larger amount of platforms and front-end technologies enable the use of adaptive or responsive layouts. A responsive layout adapts to the viewing area by resizing, reorganizing and resampling the information presented to the user. [47] It is often based on set of steps within which the content scales, and when a step changes, the content arrangement, and / or organization itself changes.

A key challenge in CCA was the requirement of being completely resizable upwards from a minimum size (1024*800px). The elements must fit the screen from the minimum onwards, while maintaining a suitable space and visual feel to them. A part of the solution for this was the center-based grid design. Because the platform can be resized both vertically and horizontally, basing the user controls in the middle of the screen enabled efficient use of both axes.

In CCA the whole viewport scales based on user resolution, but the game area itself has a minimum and a maximum stop for both performance and playability reasons. The size of the game area affects how big an area the user has to visually scan in order to notice changes. This area cannot be too large, otherwise it starts affecting users performance scores.

In addition to the scaling layout size, all the main views in the UI of CCA can be dragged around the screen by the user as needed. This empowers the user to arrange the UI in efficient ways, depending on their own resolution. It also helps solve the problems of using extra space in the GUI. Each element has an active drag area everywhere where there is no control, or content presented. The dragging works with slight simulated physics, calculating a primitive velocity of the object when a user drags and

releases it, and using a static coefficient for friction, decreases the elements velocity until it comes to a stop.

4.3.3 Elements

An important part of understandable GUI-design is the use of repeatable, recognizable visual controls and containers, also known as elements. The use of repeatable controls is also of benefit from the development point of view, as it is substantially less time consuming to create base classes for repeatable elements such as a content container or a button and button sub type. Repeating elements are also essential in creating a coherent user experience, where the user can predict how a certain view behaves, and how to interact with controls.

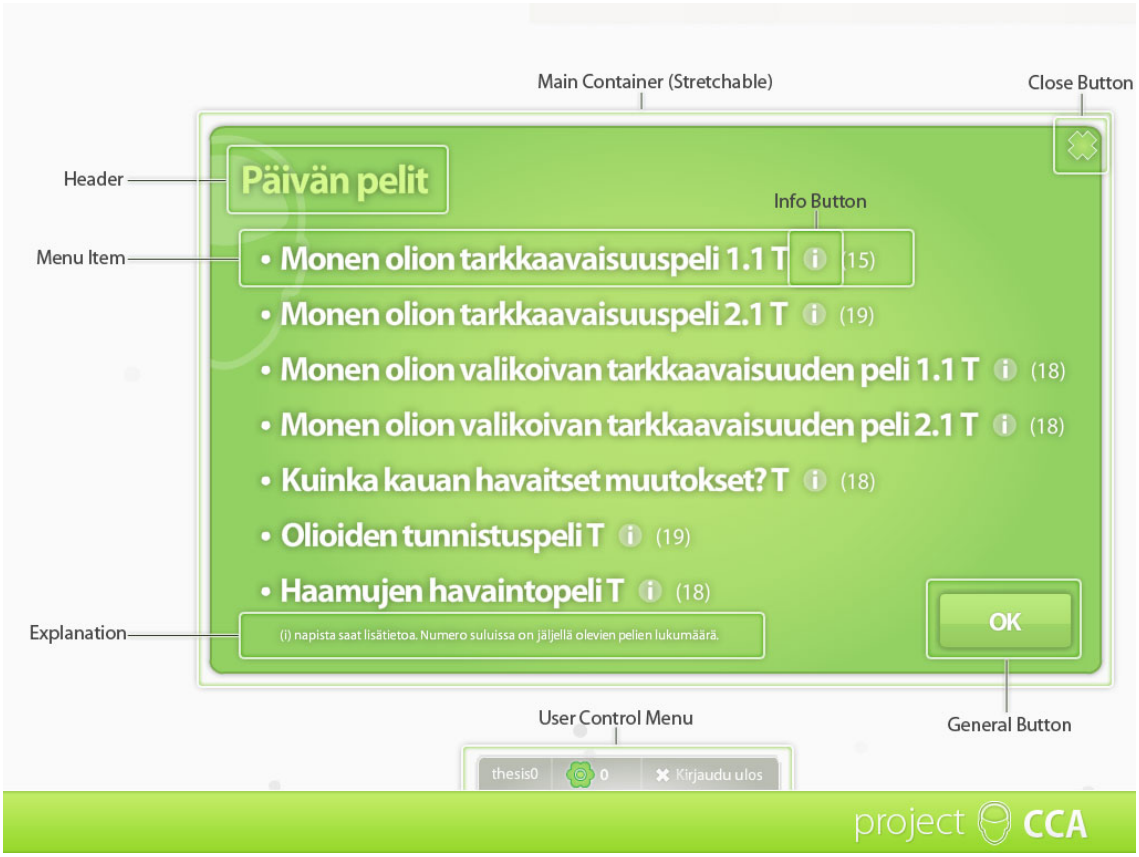


Figure 21. Elements and general containers in CCA

CCA contains a set of containers and controls for the users to use that is largely based on drag able containers that automatically centre on screen when created, and later on allow the user to place them wherever they please. As seen in Figure 21, the main con-

tainer is the base of every content element. It is vector based, stretchable and contains the drag-enhanced functionality.

Other common elements for most of project CCAs layouts are the main header, the general button element and the close button. In prompts, the close and the general button provide the same functionality, with different messages. From these base elements most of project CCAs views have been built.

Aside from the floating elements, a typical CCA UI contains the user control menu. This element attaches itself to the bottom of the game screen and displays user specific information such as the account name and the amount of points the user has, as well as providing logout functionality. During a game play the UI also has a game related menu, with information about the game you are playing, as well as controls to get back to the main UI.

Figure 22. Button elements in project CCA

For proper interaction, every element needs three basic states for a solid user experience; normal, hover / active and clicked / reaction. Examples of project CCA button elements three states can be seen from Figure 22. They all follow the same logic, hav-

ing a subtle highlight specific for the type of the button, and a full fill, recognizable action state that is clearly indicated. Providing these states allows the user to always be in control, to have a predictable response to every interaction.

CCA contains also elements that are not meant for interaction by the user, but for mere show of information. During the game there is a large clock displaying how much time is left in the current game, and how many sections are included in it. Another element that a user can view is the indicator for performance in the current game. The symbols stars and crosses show the user how well he is doing during the exercise, whilst being inconspicuous enough not to distract the exercise. Examples of the clock and the star/cross performance indicator can be found in Figure 53.

4.3.4 Colors

Colors contain a lot of meaning. The selection of colors for a user interface, and especially for branding of a platform is an important step. Colors, while being culturally dependent and highly subjective, offer a lot of meaning and interpretation and can greatly enhance the user experience and aesthetics of a system. Whilst there is a plethora of articles about color in physics, psychology and other disciplines, in the context of this study color is used to discuss the merits of the use of color in the reference project from a user and artistic point of view.

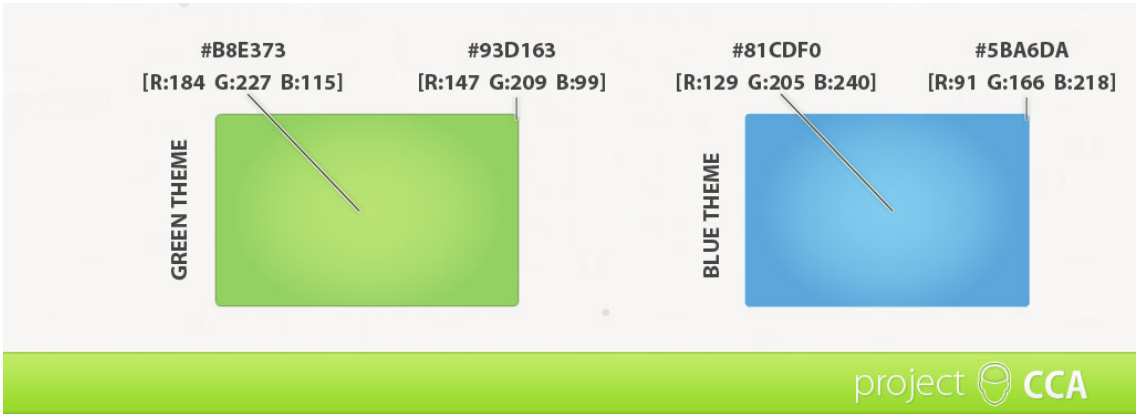


Figure 23. Color themes and main colors in CCA

Figure 23 shows the main color themes in CCA. The main color is a green shade (R: 184, G: 227, B: 115), normally used in a radial gradient with multiple stops between the end and start points. The gradient form is to create a feeling of enlightenment, a subtle

focus on the content in the middle. The green background and color scheme is used throughout the buttons and modal dialogs wherever something related to the main game is encountered. The blue (R: 91, G: 166, B: 218) theme is used in information and statistics, where there is a need for more contrast with the elements and a distinct visual appearance.

Other than the dominant background and highlight colors, the main color for text in CCA is always white, with a subtle drop shadow to bring it forward from low contrast backgrounds. In buttons and other interact able controls, the active color is always dark grey or black, to highlight the change and alert the user to an action.

In addition to the use of color in the platform interface, the games themselves have a specific use of color. Each creature in the game has its own color scheme, containing various hues and shades to create an organic “creature”.

The use of shapes and colors in the game relies on feature integration theory; a theory of attention suggesting that perceiving stimuli can be divided into two separate tasks: features and objects. Features can be registered fast and in parallel, whereas objects are slower and separately identified. Visual searches regarding these two tasks are called feature search and conjunction search. Feature searches are fast sweeps targeting only one feature, such as color or shape, while conjunction searches work with a combination of features. [48]

In CCA, there are game modes targeting both types of visual search. The game types focusing on multiple creatures on a neutral light grey background focus on the use of feature search where the target mode of the creature is a change in contrast, color and / or shape. The game platform also has a mode for rendering a specific Perlin noise background matching the color set of the active creature, aimed at the use of conjunction search. The use of a background where the creature easily blends in forces the user to focus on finding the shape by combining color and shape information. The task of separating the object from the background layer is called figure-ground separation. [49]

4.4 Motion and visual narrative

Motion is a powerful tool in the world of modern digital communication when creating a unified user experience, especially for the consumer market. By creating a specific visual narrative, a repeating pattern of interactions and movement that builds upon the visual style and reinforces it with suitable movement one can build a memorable and effective user experience.

In the realm of traditional animation, there are 12 principles for believable movement [50], of which the following are relevant to transitions in digital media and the CCA platform:

- Anticipation
    - To prepare the audience for the action and to make the action feel more realistic
- Slow in and slow out
    - To simulate most movement in the real world. Most of human movement and gravity based movement have an in-out easing curve, it builds up and it builds down.
- Arcs
    - Trajectories are followed by most natural motion, most living creatures have structures that enable them to move in certain ways, follow certain paths. Emulating these trajectories creates organic looking movement.
- Secondary action
    - A complementary animation that adds to the main animation enhances its effect.
- Timing
    - Correct timing makes objects more real, like they were following the laws of physics.
- Appeal
    - Anything the user sees and finds likeable, pleasant design, a quality of charm, simplicity and communication. Originally about drawn characters but applies to graphics as well.

The sections of movement and animation discussed in this study can be divided into three categories: Timing and flow, movement patterns and easing and nonlinear motion.

4.4.1 Timing and flow

Timing is the part of animation that gives meaning to movement. [51]

The key in efficient visual narrative is twofold; timing and delays. Timing is the essence of transactions, interactions and transitions within the platform. How long does an element transition, how it appears, how it behaves. Delay is essential in combining different views and states. With the combination of these two, the base structure of visual narrative is achieved.

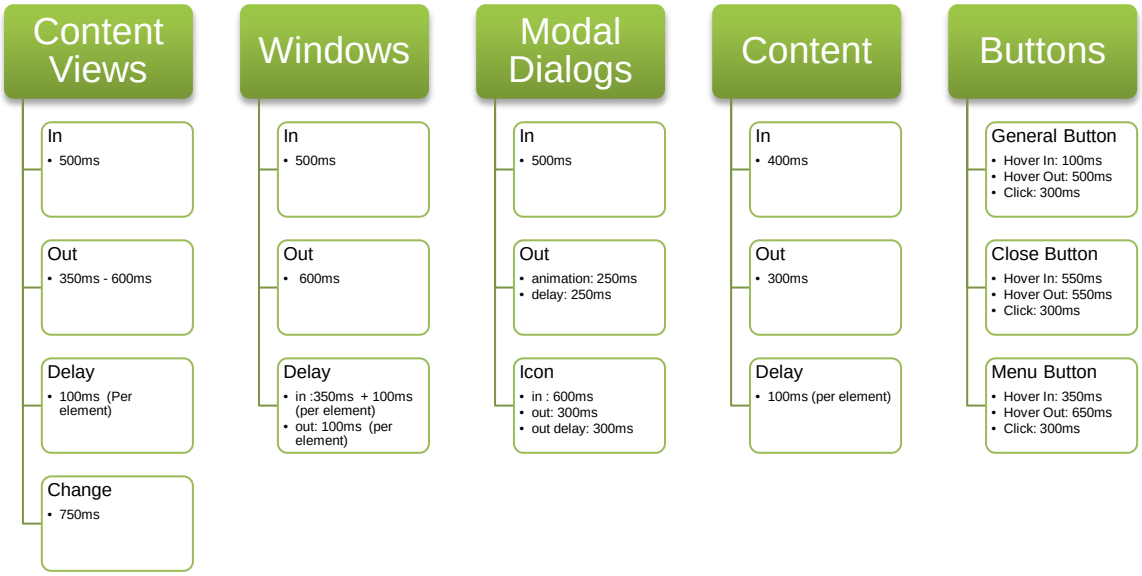


Figure 24. Timing and delays in common CCA objects

As Figure 24 shows, most transitions in the project platform last between 300ms and 600ms. In views, the time in is always 500ms, and the time out varies based on view, but the change trigger is always 750ms; when a transition from a content state is initiated the out animation call stack is started, and the 750ms delay timer is initiated at the same time. Regardless of the time the view takes to animate out, in 750ms, the animate in of the new state is called and started. This overlap makes it possible for a user

to interact during transition, and the loose coupling between view animations makes the user always be in control.

On average, for a transition to be within acceptable limits it has to be between 100 and 1000ms. Immediate response for actions like clicking requires some indication of reaction within 100ms of the action. If you go over the 1000ms limit, people will start to think the system is sluggish and unresponsive. [37] In CCA, the times are based on iterations of visual aesthetics, feeling of motion, and professional opinion. Motion is a rather tender art, and the feel of a transition is based on the shapes, colours, graphics, elements, and the surrounding elements of the target. There is no unified rule that can be quantified for every solution; it depends greatly on the context and surrounding platform.

The buttons have the same click animation time always, to make the reaction based on a user intent unified. The difference in animation in and out time is essential since the shapes and means are different, in a menu button, the line, colour and shadow are animated, in the close button the shape and colour are animated and in the general button, the gradient, shadow and text colour are animated. All the buttons have different hover state animations; however they are visually coherent and unified to represent the same indication for the user.

4.4.2 Tweening and easing algorithms

Tweening, or inbetweening is a method of interpolating the change in value between points A and B. A basic tween is linear, consisting of a predictable amount of points between A and B. Dynamic tweening is adding acceleration and / or deceleration effects to animation by using easing algorithms [52]. The problem with static / linear tweening is that the movement looks fabricated and clunky. Dynamic tweening solves this problem by adding natural feeling movement patterns such as the slow in slow out principle of animation.

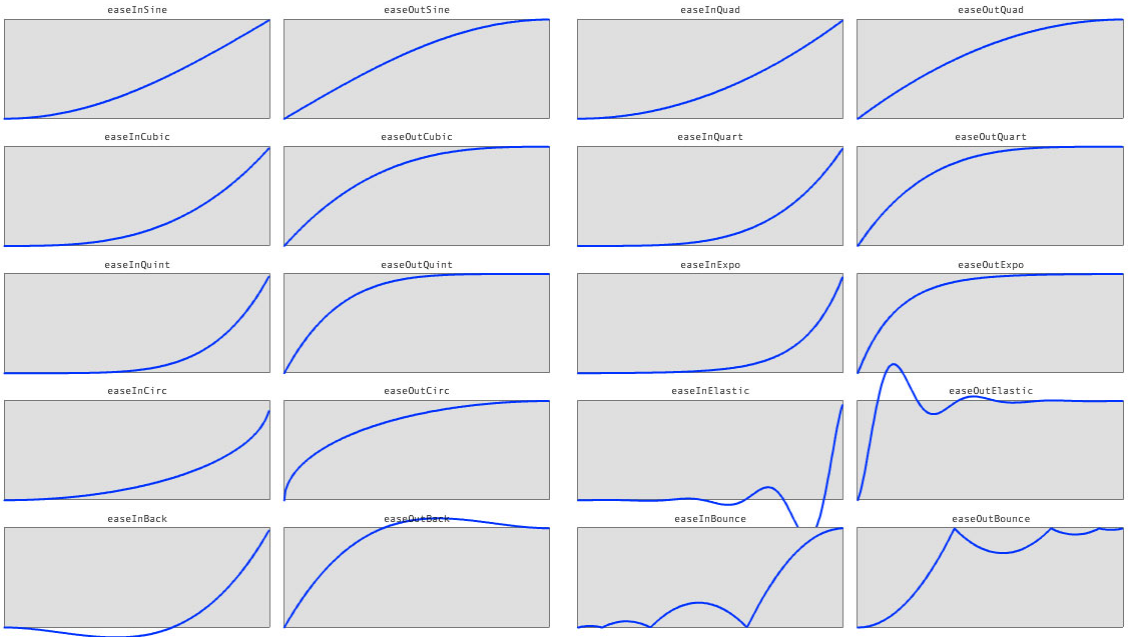


Figure 25. Examples of most common easing algorithms [53]

In programmatic animation, easing algorithms enable the feeling people perceive in physical movement. This effect is done by applying an algorithm to the interpolation of movement of the object. In principle the ease affects how the steps of movement are shown, where there are more steps and where less along the path of the animation. Figure 25 shows the most common easing algorithms used in tweening libraries on various platforms. Most are based on Robert Penners' work on easing and tweening [52].

In CCA, the use of easing algorithms is subtle, and a mix of the following:

- Content boxes
    - Back.easeOut for background
    - Quad.easeOut for fades
- Menu elements
    - Quad.easeOut for fades and color
    - Strong.easeOut for the line
- Countdowns
    - Strong.easeOut for scaling
- Drag
    - Based on inertia, a custom easeOut calculated based on mouse movement and velocity

Most of the transitions are elegant, simple, based on the subtle curve of Quad based movement. The parts where there is a need for highlighting movement, as in dialog boxes and the pop out of elements, the Back based movement is used. For fast movement, and to give the feeling of control, of pace, the Strong movement is used.

For user reactions, the easing type of easeOut is almost exclusively used in CCA. The easeOut means the movement starts fast, and slows down as it decelerates in the end. It works well for user reactions as the start of the movement is the user initiated action, the response needs to be fast and clear, and easeIn algorithms tend to make the transition seem sluggish. EaseIn is only used when animating a background out with scaling of the height of the element. Since the Back easing algorithm has the bounce effect, and the background is closing, it is visually appealing and logical to have it do the bounce in the beginning.

4.4.3 Movement patterns

After timing and easing are taken into account, what is left to create a solid visual narrative is the actual movement of objects. There needs to be a recognizable pattern of movement that strengthens the visual branding and solidifies the feeling of the user experience. In CCA different containers have different movement, but the general movement patterns are very similar, and create a solid feeling of a unified platform.

The button animations all contain the same click effect animation and timing. When a user clicks a button, the text or main shape of the button turn black in colour over 300ms. The menu button is a good example of the secondary action principle of animation. The main action is the colour animation to black; the secondary supporting animation is the line growing from the left to right underneath the text. The line animation does not dominate the visual, but builds on the effect, and supports the feeling of highlighting. Examples of button visual states can be seen in Figure 23.

Dialogs and content views all appear in a similar fashion, the background animates in first by scaling the height to 100% from 0, while animating the opacity of the element to a full 1.0 form 0. These two combined make the background appear smoothly from thin air. The content inside dialogs is animated in with a sequenced animation, fading in elements and in the case of lists, also animating the x coordinates of the element from

–N to 0. (Where N is subjective to the size of the element) The sequence timings can be found from the Figure 24.

The basic pattern of animation in user interface elements for project CCA is the fade in / fade out; the simple vanishing and appearance of objects. It is supported by secondary actions such as colour animations, movement, scaling of horizontal and vertical size, depending on the object animated.

4.4.4 Overrides in code based animation

From development point of view, creating a visual narrative requires a specific way of creating your user interface classes. When transitioning between states, many platforms and systems make the mistake of queuing the transitions in a way that makes the user wait for interactions between states. This breaks the narrative flow and lengthens the response time of the system to the users' frustration.

When creating a solid platform with UX in mind, it is necessary to plan in advance, and to create methods for handling view states concurrently. In project CCA when states are transitioned, every control taking part in the transition executes their hide / show mechanic. There is a set delay between calling the next state when current state is transitioning out and there is no queue. As soon as a control starts to form on the screen, a user can interact with it. The user can stop the current transition and use another control while transitions are in place as well.

Creating modern user experiences requires the developer to take this into account when creating their solution. It saves time and effort to solve the issue of state handling and transitioning between states already in the beginning or planning phase of the project. When building individual controls it is beneficial to have a set template to start from that contains all standard functionality needed for state handling. Whether to implement this via inheritance or other methods is up to the developer.

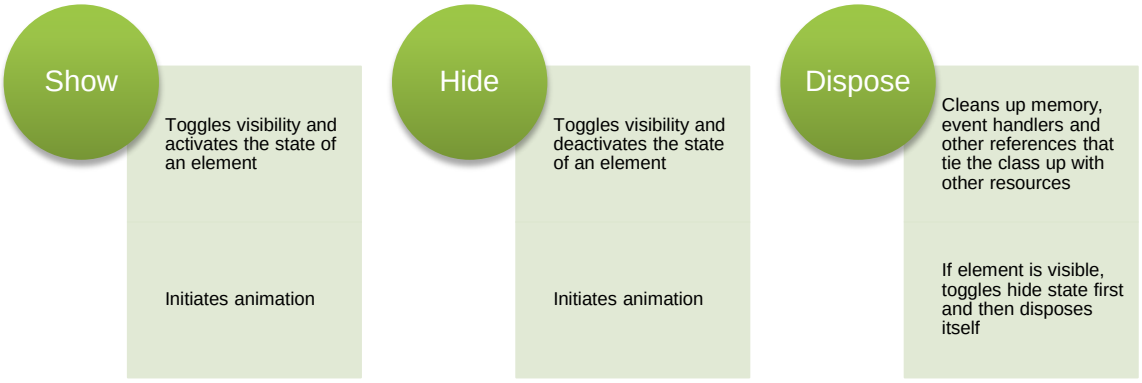


Figure 26. View controls for all visual elements in CCA

In CCA every visual element from single controls such as a button or content views such as the main menu to modal dialogs and user menus, all of them contain basic functionality for handling their own view state. The repeated functionality in all visual controls of CCA are show, hide and dispose, as seen in Figure 26.

A necessary requirement and in creating modern interfaces with programmatic animation is the ability to override on-going animations. Overriding enables the stopping transitions and interacting with controls mid transition to keep the user in control. Modern UI technologies either contain suitable Tweening libraries in-built or one can use one of the many open source alternatives out there. In CCA the library in use is an extremely robust and extensive tweening platform called Greensock Tweening Platform [54] by Jack Doyle (www.greensock.com). GSAP is available for Flash and HTML / Javascript based platforms.

GSAP abstracts many of the verbose methods needed in creating programmatic animation, such as the setting of filters, timelines and sequencing by offering a simple, but powerful syntax that allows for the tweening of any numeric property, alongside platform specific properties such as CSS transformations. It is also quite robust, efficient and contains a full open documentation. The platform is used by many of the top sites and digital experiences in the world, such as big movie productions, big brands and art projects. [54]

4.5 Visualization of data

One of the main concerns in good UX is to show meaningful data in a meaningful way.

Figure 27. Performance chart in CCA

In CCA there are two different systems for showing performance and ability. The user has access to pure numbers and graphical visualizations of performance and numeric data. The platform provides statistics on all game modes, as well as training program based data for the user to analyze. The idea is to enable the user to transparently go through how well he has done and how he has improved and draw his own correlations about the program. Example of the histogram approach is shown in Figure 27.

The second way the user is communicated the importance and performance of the data is via a very UX oriented method – achievements. The user is provided different rewards in three different categories of achievements: achievements, stars and trophies. The different rewards are all catering to a certain part of the measured data and

user response. Every achievement has their own icon and a badge the user can see once they log in.

Figure 28. Achievements, trophies and stars in CCA

The achievements provide a very human understandable way of communicating complex data oriented events. The user can for example get a trophy from doing better than previous runs, or for having a perfect run where there were no missed reactions. The rewards vary between the different game modes. In comparison to the raw data shown by the histograms that do provide a valuable way of seeing visually where you improve, the achievements allow the user to gain specific feedback for specific parts of their game performance. Example view of achievements in CCA is seen in Figure 28.

5 The CCA project platform

CCA is a user experience driven cognitive gaming platform harnessing the power of the Adobe Flash Platform in the front end implementation, and the LabVIEW system design software as the server side solution.

5.1 General architecture

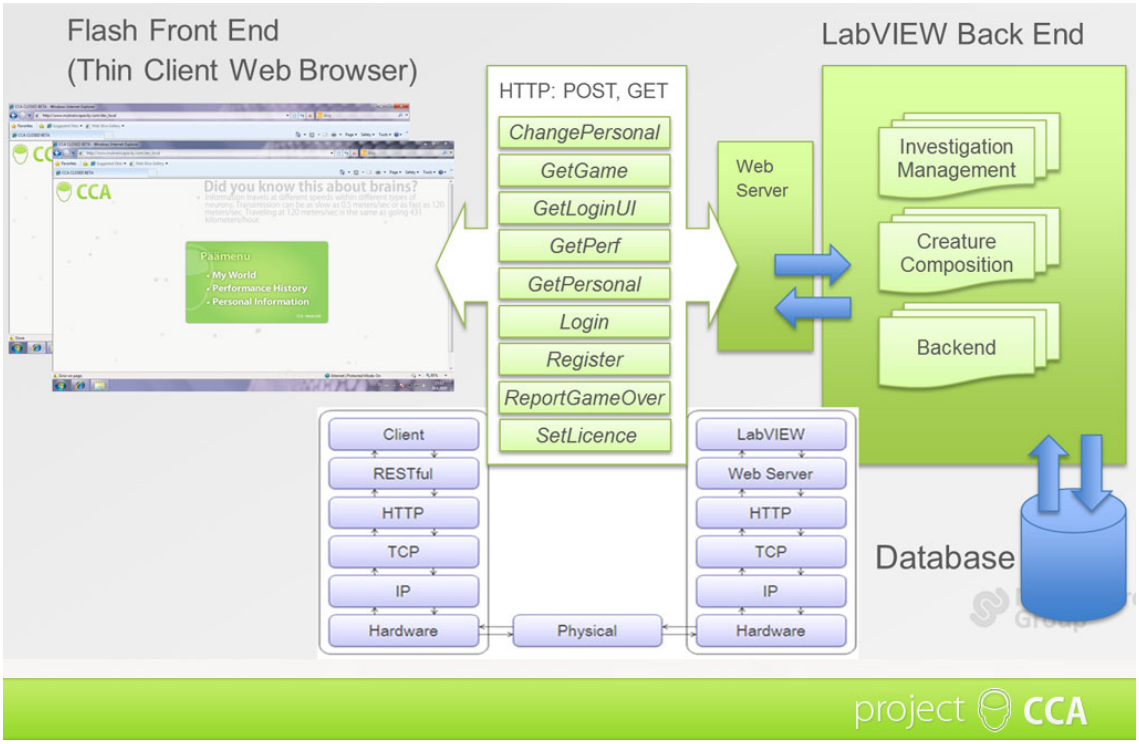


Figure 29. Server-client software architecture

The general architecture of the platform is based on typical client-server architecture. Overview of the architecture is seen in Figure 29. The client connects to the server and authenticates the user account, after which the server sends a packet of data that determines what options are available for the user with the account. The user interacts with the options available, and a new request is sent to the server, to which the server responds by providing wanted data, be it user account details or a list of games.

As is common for Flash based web applications, there are no page loads or page requests during the application run. The state of the page is managed by the application platform itself, and all interactions and reactions by the user are parsed, processed and

handled without the browser. As such the platform itself can be ran with a flash player outside of browser environments as well.

Labview (short for Laboratory Virtual Instrumentation Engineering Workbench) is a programming and system design platform built by National Instruments. It uses a data-flow-based programming language, which enables users to program logic with the use of graphical block diagrams. [55] The platform provides a very visual way of creating functional systems.

The platform front end is based on Actionscript 3.0 (AS3). The Ecmascript based 3rd iteration of the Actionscript language is an object oriented programming language running on the Actionscript virtual machine. The version of the virtual machine that supports AS3 was built from ground up to support the new OOP nature of the language. In combination with the Flash development tools like Flash Builder and Flash CS5, the AS3 is a powerful tool for interactive media.

The front end is built to be modular, and much of the control comes from the backend. This data-driven architecture helps keep the platform modifiable for different purposes and user programs without making changes in the front end implementation. [23] The server side provides lists of menu items and games available, as well as statistics and user account data. The front end has different tracks of interaction and progression through the solution that are triggered by the server messages.

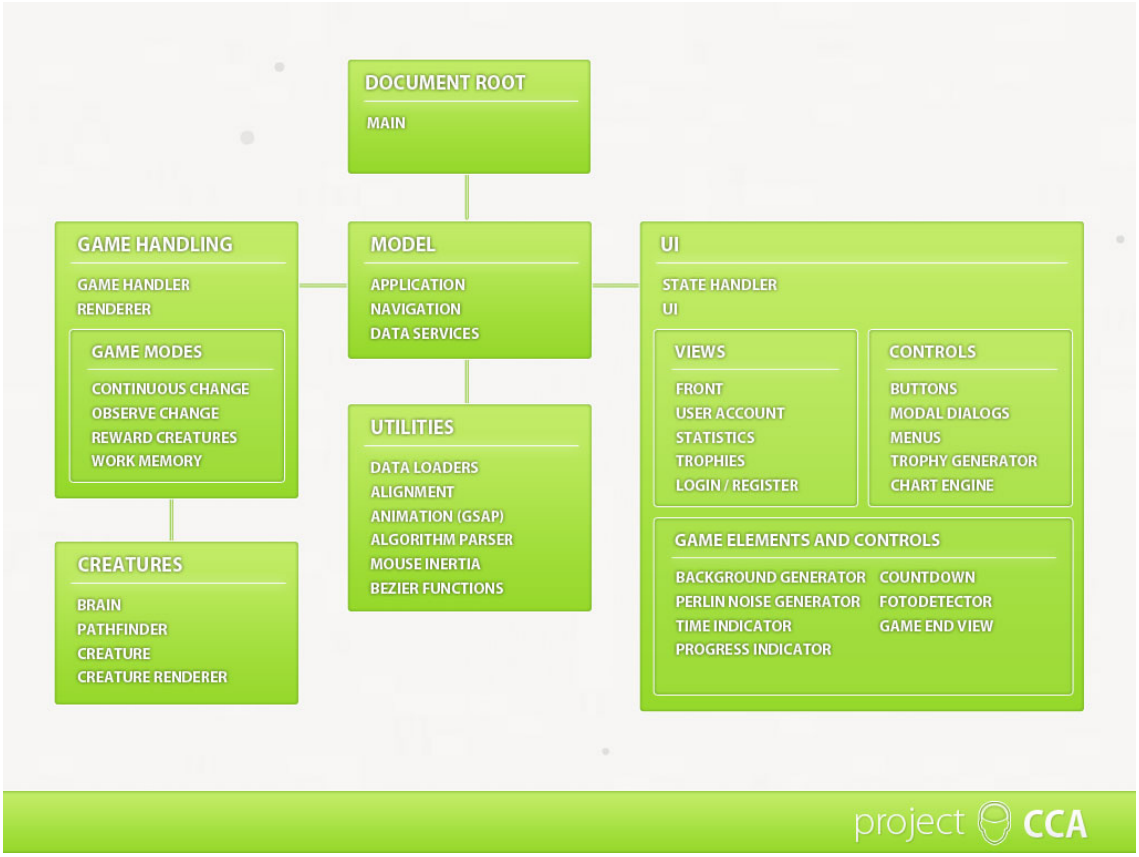


Figure 30. CCA front end platform packages

Figure 30 gives an overview of what packages, views and controls the CCA front end platform contains. In AS3 projects, there is always a root stub that initiates the solution when started. The main solution initiates the application model and main UI elements and queries the server for language versioned UI definition XML. After that the navigation service handles user interactions and along with the UI state handler keeps the visual and logical state of the application in sync.

The model package contains main application logic and data related functionality. The application model handles application states with the navigation service, and holds references to the different parts of the application. The navigation is integrated tightly with the UI state handler that handles the UI state changes and controls what the user sees. Data services handle all server communication and data parsing.

The creatures package is the home of the main rendering engine. All the heavy lifting mathematics with path generation and creature drawing is done by the package. The pathfinder handles the generation of new path points and general route handling. The creature handles the algorithms, creature specific calibration and state information. The

brain is responsible for combining the input from path finder and the creature and to pass off the necessary values to the renderer, which then does the actual plotting and drawing of the creatures. A detailed description of how the rendering core works is in Chapter 6 of this study.

The game handling package handles all game related logic. Once the game data has loaded and parsed, the game handler initiates the correct game mode. The game mode handles the states within the game; everything from info prompts to countdown to the different phases of the game, data tracking, user performance and the end and fail conditions. Each game mode has their own conditions for game complete and their own goals.

The UI package contains all UI elements, views and controls. Essentially the GUI is constructed via the UI package. The main UI class acts as an interface for the UI elements: initiating them based on UI states and disposing them as necessary. The game related UI elements are contained in their own package, and are used and called from the game handling classes. All controls are separated as usable entities, any view can use any control as needed.

The utilities package contains all extra functionality and helper classes that the views, game handling and data services need. It has sections for animation, data loaders and handlers, algorithm parsing, various physics and inertia handlers as well as the core Bezier classes. Utilities are used through the application model by any section of the solution that needs them. Most utilities are initiated only once, and kept in memory to be used as a static instance.

5.2 Client server communication and data management

The CCA platform was built to be highly manageable from the back end solution. Even though the rendering engine and game logic resides in the rendering and game engine in the front end platform, as much as possible of the configuration of the games and rendering options were separated from the engine and interaction logic.

The communication between the front and the back end services is done using a fairly standard REST API. Every interaction is based on HTTP POST calls with structured XML Data. XML is an application profile or restricted form of SGML, the Standard Gen-

eralized Markup Language [56]. XML is a standards-based and well documented human readable format for communication and configuration, and a suitable choice for the platform. Due to the nature of the game rendering engine, XML was also suitable for conveying the mathematical formulas and sets of functions needed to create the creature renders. The Flash platform also contains an in built support for E4X [57], the extension for AS3 that makes XML a native primitive in the programming environment, making the parsing and use of XML efficient.

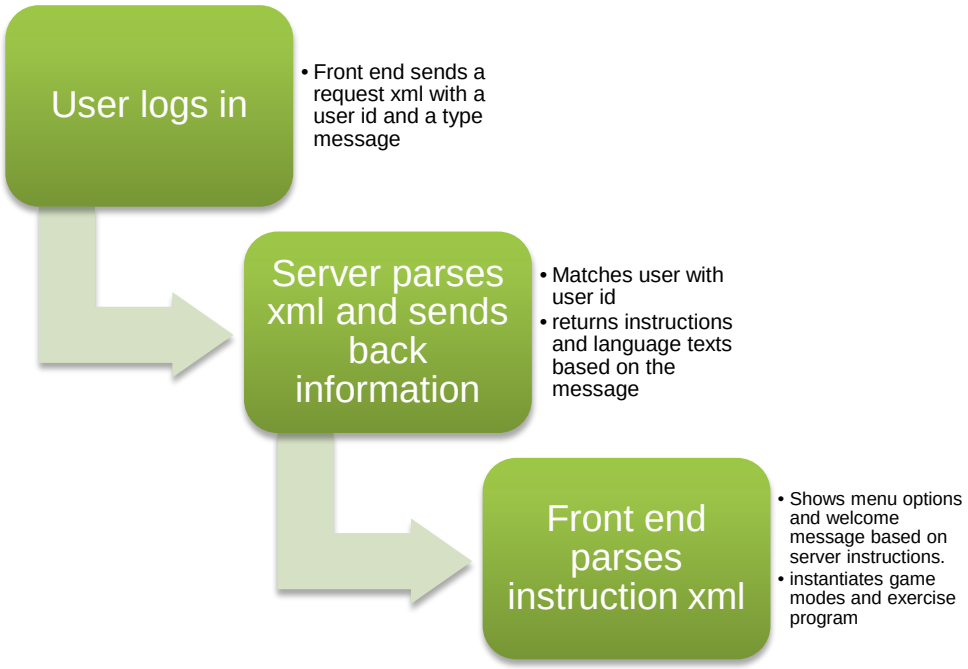


Figure 31. Example process of logging in to CCA

When the user enters the platform, the front end queries the server for the latest language information. After the user inputs his credentials, the server is queried with the user information and a request for the main user interface details for said user. As seen in Figure 31, the server parses the request, matches the user login information and returns information regarding the user account. The front end platform constructs the personalized user interface of the user based on said instructions.

The platform supports a user based exercise program that monitors how well the user is doing and calibrates it to match the performance and improve it. When a user logs in, they are greeted by a personalized message based on their previous performance.

The user has specific games based on his / her exercise program and a set amount of exercised per game per day or a set time period he can perform. The program can be managed and calibrated to each user's needs, based on progression and scores from the game.

Most of the data is served to front end only when needed. For example when the user wants to see a certain performance graph it is loaded only on users request and rendered with the latest data. Each view of the chart is requested per interaction to limit the amount of data sent and to keep response times low. In principle everything in the front end is based on the data transmitted from the back end services, except for the rendering and game logic, and user interaction.

5.3 Game generation

The front end game logic is highly parameterized and the games are generated based on the instructions sent by the server in a game XML.

Figure 32. Game initialization and play through process

Figure 32 provides an overview of a single game. In the initialization phase, the front end fetches the game- and object parameters and initializes the correct game logic engine, instantiates the creature renderers per creature in the configuration file and sets the initial failure, success and timing of the game. During the game the front end game logic handles all user interaction, game events and data gathering. In the event of a game over, be it via failure or success, the game termination is triggered. The performance data is sent to the server, and in return an analysis of the performance of the

users in relation to their performance program is dispatched and shown in the front end platform.

The game is initialized by using a set of configuration flags for the game logic and the rendering engine via the game XML. The game information is parsed from the game XML to a game value object. This VO is essentially the core of what a single game contains, and it is used in the rendering and game logic engine.

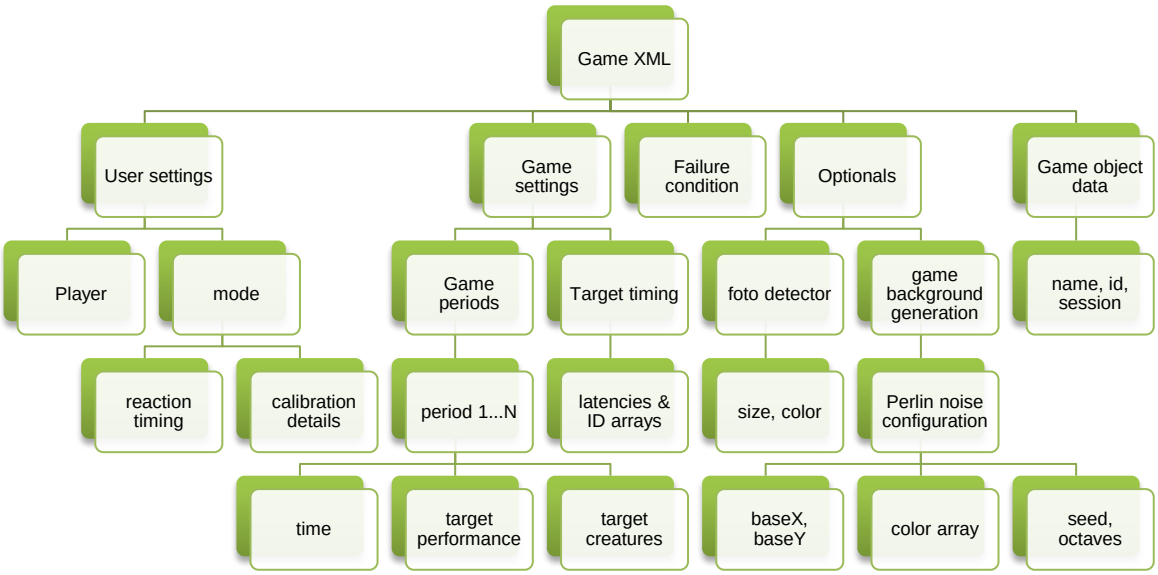


Figure 33. Game XML configuration (without creature rendering details)

The game XML has 2 main parts: the game configuration, and the creature rendering. The creature rendering is covered in detail in chapter 6. A simplified hierarchy of configuration details in the game XML is shown in Figure 33. In principle the configuration construes from 4 main sections: The user settings, game settings, optional configuration and game object data. The game object is a simple set of variables that define the user id, game name, game id and login id for the user session and game management with the server.

User settings are related to the user account of the logged in user. The player information is for UI notifications; mode contains game related data for the user. The mode determines what kind of a game logic instance is created, and also contains information about the delay and timing of the users abilities in reacting to stimuli. The mode

related user information also contains an optional calibration property that determines the time and requirements of a calibration round before the actual game begins. Calibration is done to ensure proper user difficulty in training.

Game settings define (alongside the failure condition) the structure of the game. The periods contain, depending on the game mode, the different sections of the game. Each section has a target time, performance and creatures that are used during that period. The target timing contains an array of timestamps on when the rendering engine displays a target mode effect that prompts the user to react. The target timing also contains information on which creature said target timing affects.

The target timing was originally done in the game logic engine with only minimum and maximum times for the frequency of the target event given by the server. But during development and testing there arose a need for a more fine grained control over target states, especially with the integration of the user training programs. With the server in control of the target times, various iterations of the same game can be made without affecting the front end platform codebase.

The optional settings include modules that are used in different game modes, and a medical imaging and measurement helper. The foto detector is an indicator that shows whenever a creature reaches a target state. Its size and color can be configured depending on the use case. It is used when recording user activity via medical imaging and other instruments as a synchronizing signal. The background generation defines the needed values for the background Perlin noise generation, from the actual noise to the color treshholding. More information about the background Perlin noise generation is found in chapter 6.4.3.

5.4 Game rendering event

The system in game engines that controls the ongoing simulation is the main loop. It is the representation of time in your engine, and the layer responsible for the game life cycle [58]. In CCA the front end game engine is based on asynchronous events between different modules. The core of the rendering engine is the renderer class. Any visual, interaction or module that requires a time based rendering pass - such as the path, creature and target rendering and calculations - subscribe to the main rendering class render event and base their rendering passes on it. The event is a custom event

with a unique rendering time key that is then employed in the various time based calculations in the rendering. Figure 34 shows the most important renderer event related dependencies.

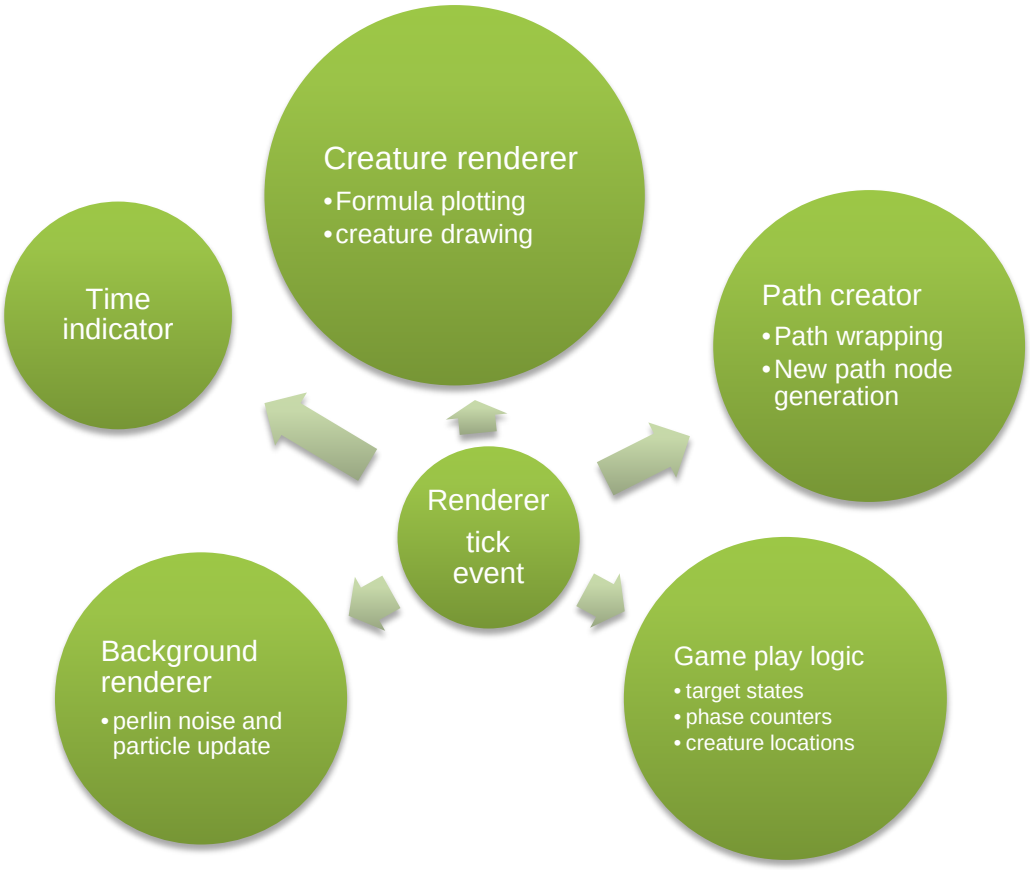


Figure 34. Renderer event relationship

The separation of direct calls gives the platform a manageable and simple way to add new requirements on the render pipeline. During the development of the platform the rendering core started from a simple mode of having creatures running free on the screen, and slowly evolved to support dynamically moving bubbles for the background, time rendering visualizations and calibration events all subscribing to the main rendering thread.

The separation also allows for an important feature for any gaming platform, the complete control over the flow of time. With this control all forms of prompts, countdowns, user feedback and result screens can be efficiently and smoothly incorporated to the game designs at any time.

5.5 User reaction tracking

In order to get a complete account of how a user performs during a game in the CCA platform, all user reactions are tracked, even those that don't end up showing any visual signs in the user interface. Every reaction, be it valid or invalid is time stamped and logged, alongside all creature target renders and game phase changes. The data is held in memory during the game period, and after the game is over, it is sent to the server for analysis.

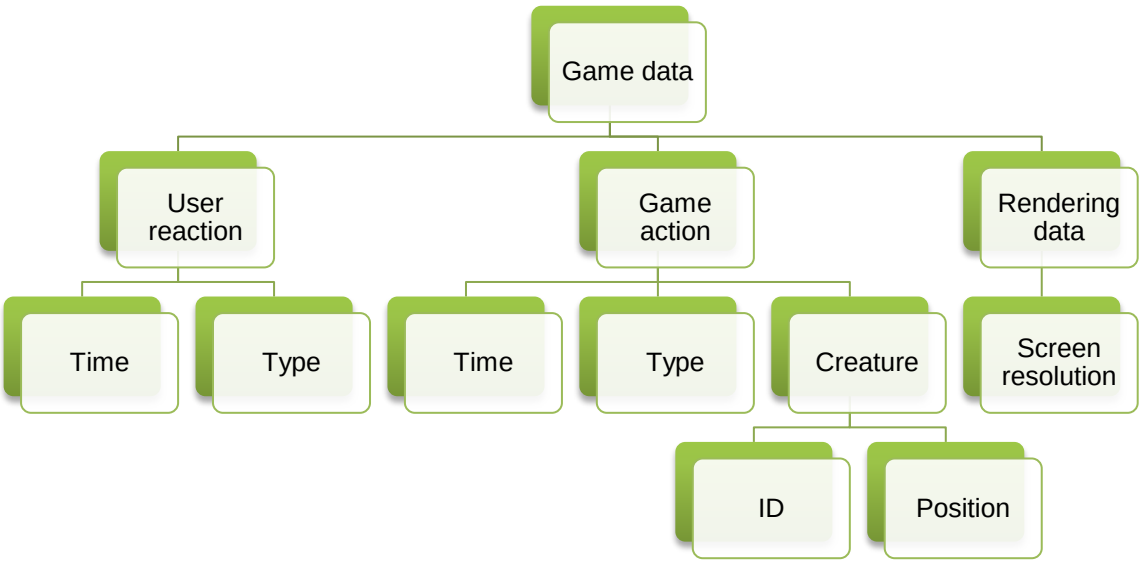


Figure 35. Game data tracking in CCA

The game data tracked during a play is shown in Figure 35. The data contains events divided into the game periods they belong to for precise analysis. Each event contains the timestamp in the relative time of the game rendering process, type of the event handled and possible tags related to the event. In addition, every reaction or target state handled contains a list of all visible creature positions for further analysis. The data also contains information about the actual rendering resolution of the game area.

Tracking enables the use of more detailed information about how people play, and the strategies they employ in reacting optimally within the platform games. For example the difference between a user trying to optimize hit percentage by reacting at a regular interval, and a honest try at seeing the reactions can be taken into consideration with the analysis of the reaction data, especially with comparisons to previous user account performance.

6 Rendering engine

6.1 Basic principles

The core of project CCA is the rendering engine. It is a 2-dimensional engine, based on a standard XY-coordinate system. The coordinate system starts from the top left corner, and expands to the bottom right corner. The whole rendering area is flexible and user scalable, and the platform adapts to the user resolution by adjusting the speed and size of the rendered creatures for optimal playability. For the sake of rendering performance and playability, there are set minimum and maximum width and height values for the scaling.

The engine is a so called black box system, where the game is generated based on set rules and the engines own logic. This means that while being very controllable, the engine does not mindlessly repeat control orders, nor does it follow a linear human made path. The engine draws and moves the creatures based on their paths and targets, trying to find an optimum route, but does so within its own limits and algorithms.

Games often are made based on strict artistic control and very man made worlds, as it allows for a more fine grained control and precisely deterministic outcomes, but it lacks the surprising elegance of simulation based engines. A good example of some simulation engines are the modern physics engines in games, especially the ones based on soft body physics, instead of rigid body physics.

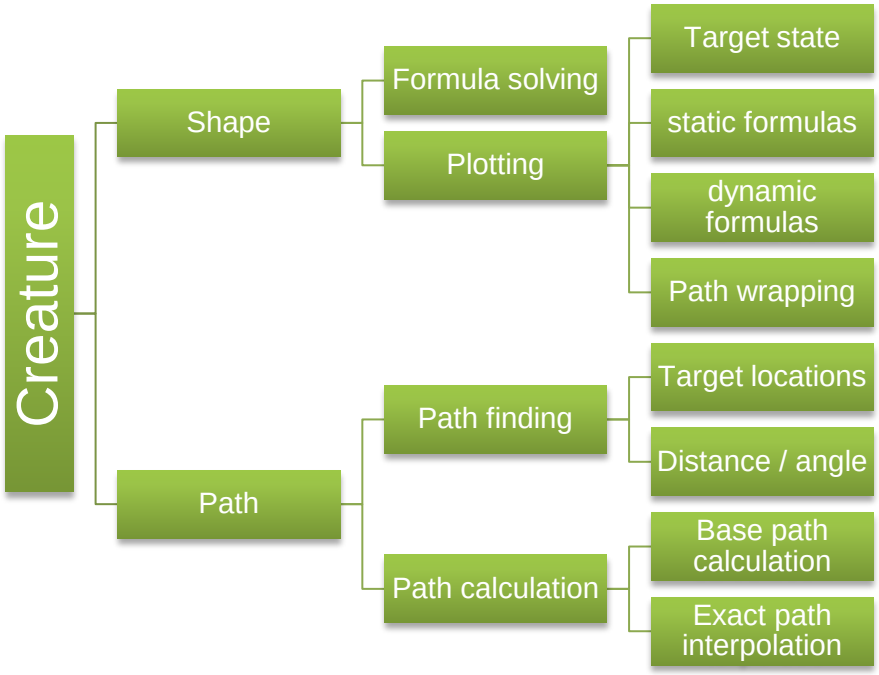


Figure 36. Basic diagram of a creature in CCA

The basic principles of the rendering engine revolve around the concept of a creature. Each creature is an entity with its own path finding and rendering logic, and all game types work with changes in the creatures. The core separation of the creature object is the shape and path, as seen in Figure 36. The shape is the actual form of the creature, the visual end result that the user sees on screen, calculated based on a series of mathematical formulas and a set of parameters. The shape consists of the actual calculations for solving the formulas as well as the rendering of said formulas on screen with a set of colors and transparencies and possible dynamic movement. The shape also takes care of wrapping the shape on the actual path, generated by the path part.

The path is the core movement of the creature itself. Each creature has its own path generation, the role of which is to find out target locations for the creature and calculate optimum movement paths between them. The path is based on continuous cubic beziers, and recalculates itself every time the creature reaches the end of an interpolated target path and embarks on the next segment.

6.2 Creature shape rendering

6.2.1 Overview of creature rendering

The rendering of the creatures on screen is based on two main parts; the plotting of the stationary creature on its own and the path wrapping calculation and plot. The creatures stationary calculation defines how the creature looks, is colored and how it performs its target state. The path wrapping takes the end result of the stationary phase, and integrates the creatures form into the shape of the path it is travelling on. The creature can also be rendered without the path as a stationary object, in its prime state.

During the project lifecycle, there were a couple of different ideas on how to render the actual creatures. With the path generation separated, it was possible to use hand drawn pieces of creatures and animate them on top of the path points to make them look more artistic and use less rendering mathematics as well. But for pure organic feeling and keeping full creative control in the hands of back end generation, it was deemed better to go with the option of drawing the creatures completely with mathematical formulas for full customizability.

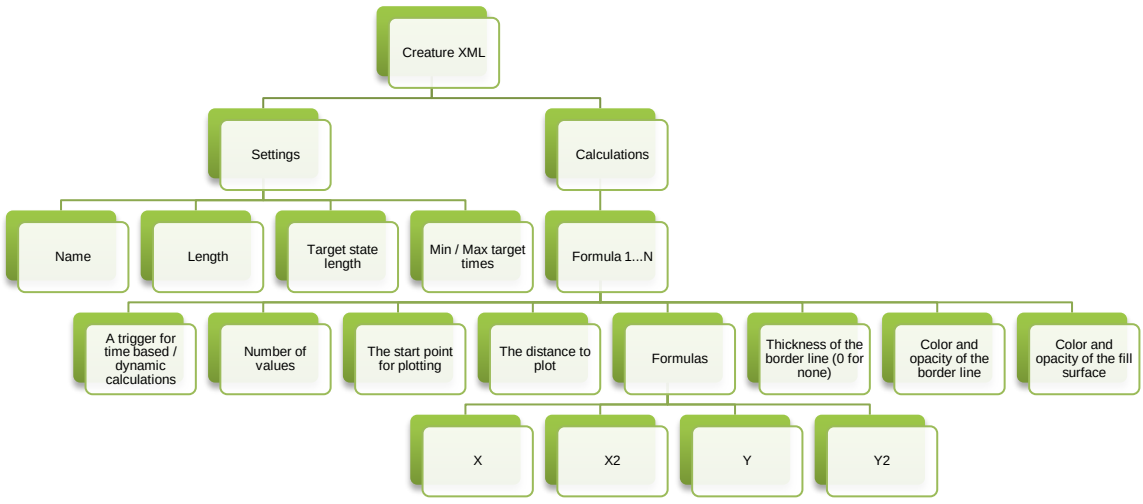


Figure 37. Creature XML schema

The creatures are rendered based on a server served creature XML that contains settings and configuration as well as the actual formulas for creature generation. Figure 37

shows an overview of the creature generation. The settings define features that the creature has as a whole, such as the target states and length of the creature. The calculations contain the essence of the creature. Each formula has a set of configuration attributes that explain how the particular set of formulas is rendered.

The formula settings define the look and feel of both the creature surfaces as well as the creature main lines. In addition the values related to actual plotting of the formulas are introduced; the amount of points to be rendered, the start value of the plot and the distance the plotting is done on. Each formula is a set of two X and Y coordinate plots. The main plot is the first XY pair, and the return plot is the XY2 pair; when these are combined they form a surface with a fill that is a part of the creature.

The actual creature rendering is a stacked plotting approach to rendering mathematical expressions, spreading them over a uniform scale, wrapping them up with return functions to create surfaces and mirroring the end result to optimize rendering. The creature form is unified on both sides, so only one half of the creature form is calculated, and then mirrored to create the actual creature.

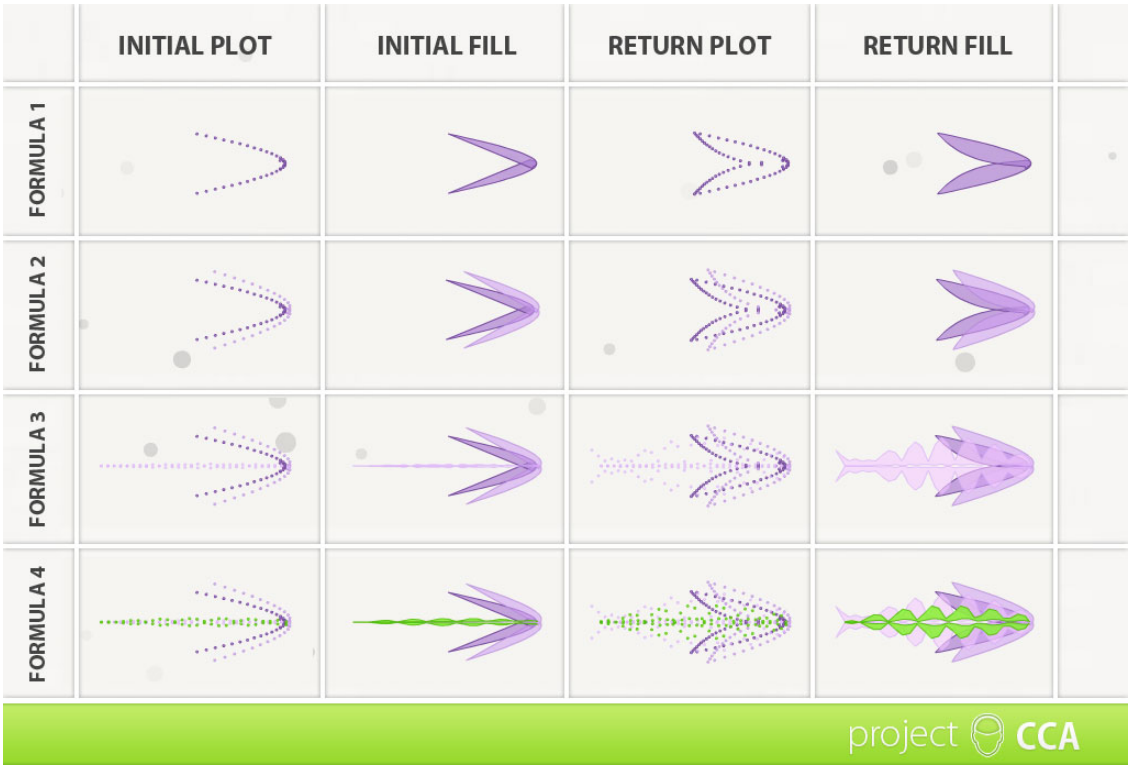


Figure 38. A single creature is rendered from multiple algorithms

Figure 38 shows an overview of how a creature is rendered from the set of algorithms provided by the creature XML. The first line shows the first formula being plotted, first the initial plot of points without lines drawn is shown, then the fill with lines and shape. Then the return formula (XY2) is plotted, and it forms a complex shape. Then the return and the first formula are filled to create the end result. This is done for each formula and drawn on top of each other on the creatures' canvas, and the formula 4 line contains the end result creature, built from the 4 mathematical expressions.

6.2.2 Creature formulas

The creatures are based on two types of formulas. The various formulas are plotted on the screen and the 2d coordinate system with the time seed value from the rendering event time loop.

The base of the creature and most of the formulas are static formulas. Static formulas are plotted once per creature per formula. After the initial plotting of the formula, the plotted points are stored in vectors (typed arrays of AS3) and used for path wrapping and physics calculations, before finally being turned into visual renderings.

The amount of plot points is defined per formula, as are the values the formula is plotted on. The definition of the target state is given as a separate formula to be rendered when triggered. All the times of trigger and densities and colors and opacity levels are configurable per formula.

```
<cal id="2" timeCal="0" valNum="14" valStart="0.0100" valDist="10.0000" lineThickness="1.000000" line
Alpha="0.7000" lineColor="0xDAA9A6" fillColor="0xEEBDBA" fillAlpha="0.6000">
    <x><![CDATA[(t*0.7377)*log(t*0.2000)*-0.6098-0.9492]]></x>
    <y><![CDATA[(t)*0.2262-0.2697]]> </y>
    <x2><![CDATA[log(t*0.1148)*-1.3574-4.2246]]> </x2>
    <y2><![CDATA[(t)*0.2262-0.1713]]> </y2>
</cal>
```

Figure 39. Example of a single static expression

As seen in in Figure 39, a single expression of a creature consists of a set of parameters, followed by the initial and the return formula of the single shape to be rendered. Each static formula has a variable that is plotted with the given attributes. This variable is named (t) and it is parsed in the rendering engine into AS3 native value, along with the text representations of the math functions.

In the example expression, the (t) is plotted from valStart to the distance provided by valDist, over the course of the amount of points defined in valNum. After these have been plotted, the points are stretched with the length factor of the creature XML. The length factor is the length of the creature in ideal circumstances, and is adjusted as needed based on the resolution of the game area during play.

```
<cal id="1" timeCal="1" valNum="9" valStart="0.0100" valDist="10.0000" lineThickness="1.000000" line-
Alpha="1.0000" lineColor="0x9E7EB9" fillColor="0xC2A2DD" fillAlpha="0.7000">
    <x><![CDATA[(t*0.7377)*log(t*0.2000)*-0.7140-1.3426]]></x>
    <y><![CDATA[(t)*0.1869-0.3188 + abs(sin(c * 0.2800 + 1.85))]]></y>
    <x2><![CDATA[log(t*0.2000)*-1.6033-4.1754]]></x2>
    <y2><![CDATA[(t)*0.1869-0.2697 + abs(sin(c * 0.2800 + 1.85))]]></y2>
</cal>
```

Figure 40. Dynamic expression

Some creatures based on rendering type and game mode also contain dynamic formulas. Dynamic formulas are plotted per render loop every time anew. This creates a relatively significant overhead for calculations for the processor. Dynamic formulas are used sparsely and mostly to add some flair or to make it harder to recognize the target states of the creatures. Figure 40 is an example of a dynamic expression, the definition in CCA for a dynamic variable is (c). Typically dynamic formulas are circular, so they create repeating smooth movement.

```
<statecal id="4" timeCal="0" valNum="13" valStart="0.0000" valDist="10.0000" lineThickness="1.000000"
lineAlpha="0.8000" lineColor="0x7ACB38" fillColor="0xA3F461" fillAlpha="0.8000">
    <x><![CDATA[(t)*-1.0000]]></x>
    <y><![CDATA[abs(sin(t*2.0000+0.4098))*abs(cos(t*0.2800+1.8850))*0.2000]]></y>
    <x2><![CDATA[(t)*-0.9800-0.2000]]></x2>
    <y2><![CDATA[abs(sin(t*2.0000+0.4098))*abs(cos(t*0.2800+1.8852))*((abs(c-1)+1))*0.8852]]></y2>
</statecal>
```

Figure 41. Target state calculation expression

The third type of algorithm every creature contains is the target state calculation formula (Figure 41). This formula decides the look of the creature when it reaches target state, the state when a user is supposed to react to the change in the creature. The target state is parameterized in size and effect based on the user accounts previous score and calibration results. It is pre-calculated at the start of a game, and contains only static formulas. The difference between a normal drawing function and the state one is that the state is drawn for a set amount of time, as defined in the creature head part of the creature XML.

Every formula the creature is based on was originally a one way formula, forming a line but not a shape. After some development time there arose a need to make shapes, fills and surfaces in the creatures to add visual fidelity and concretize the creatures some more. To solve this need the ability to render return formulas for the various algorithms was introduced.

```
<x2><![CDATA[(t)*-0.9800-0.2000]]></x2>
<y2><![CDATA[abs(sin(t*2.0000+0.4098))*abs(cos(t*0.2800+1.8852))*((abs(c-
1)+1))*0.8852]]></y2>
```

Figure 42. Return formula

In principle a return formula is plotted on the same time values as the initial formula, and in the same scale. With the main formula it creates a complete shape or surface, which then can be filled with color and opacity to make it more vivid. Adding the return formula to the shapes enabled the creation of complex shapes, interloping with opacity and colors for unique creature designs. An example of how the return formula forms the body of the creature can be seen in Figure 38, and an example of the XML can be seen in Figure 42.

6.2.3 Path wrapping

The creature formulas plotted on their various paths form the natural state of the creature. To create life and movement it needs to be integrated with the path provided by the path creator. The integration of the creature to the path is called path wrapping.

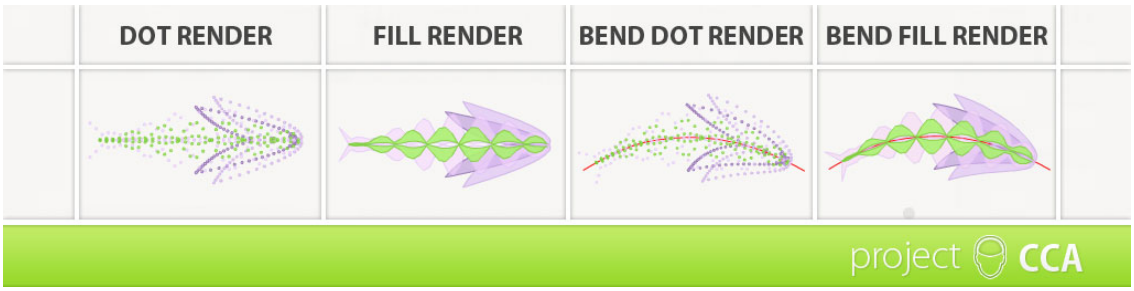


Figure 43. Path bending example

Figure 43 shows a visual representation of the difference between the creature in its natural state, and the path wrapped bended version of the creature. The red line at the

bottom is the interpolated target path of the path calculation for the creature. The target path generation is explained in detail in chapter 6.3.2

When wrapping the formulas on the path, the formulas are plotted as usual to create the natural stationary creature, but after that the whole plot is bent around the segment path so that length-wise the center of the creature is on the current point of the path rendering progression. Then both halves are calculated from the center onwards, up to the point where the length of the creature ends. The interpolated target path is the size of the creature, and contains a point for each plot point in the stationary creature plot.

After the target path is generated, the creature is wrapped around the path point by point, calculating the angle at each step. This calculation is done per formula per creature; each formula is bent individually.

Figure 44. Principle of creature bending on the path

A simplified principle of the creature bending can be seen in Figure 44. First the integral of the segment, as well as the angle of the segment at any given point is solved, while making sure that the angle doesn't go over – or + pi. Then the position of the creature is interpolated on the path, and the point is rotated based on the angle on the frame path at the same point. The creature is wrapped, point by point on the ongoing path, allowing it to react to every small change in the angle of the path individually, enabling the organic feel.

6.3 Path creation and path finding

The movement of the creature is based on the path generation created by the path creator class. The path creator calculates the targets where the creature needs to go and the path itself and then attaches the creature on the path. The path is based on

continuous cubic Bezier curves, to create a harmonic organic movement of the creatures while maintaining randomized movement paths.

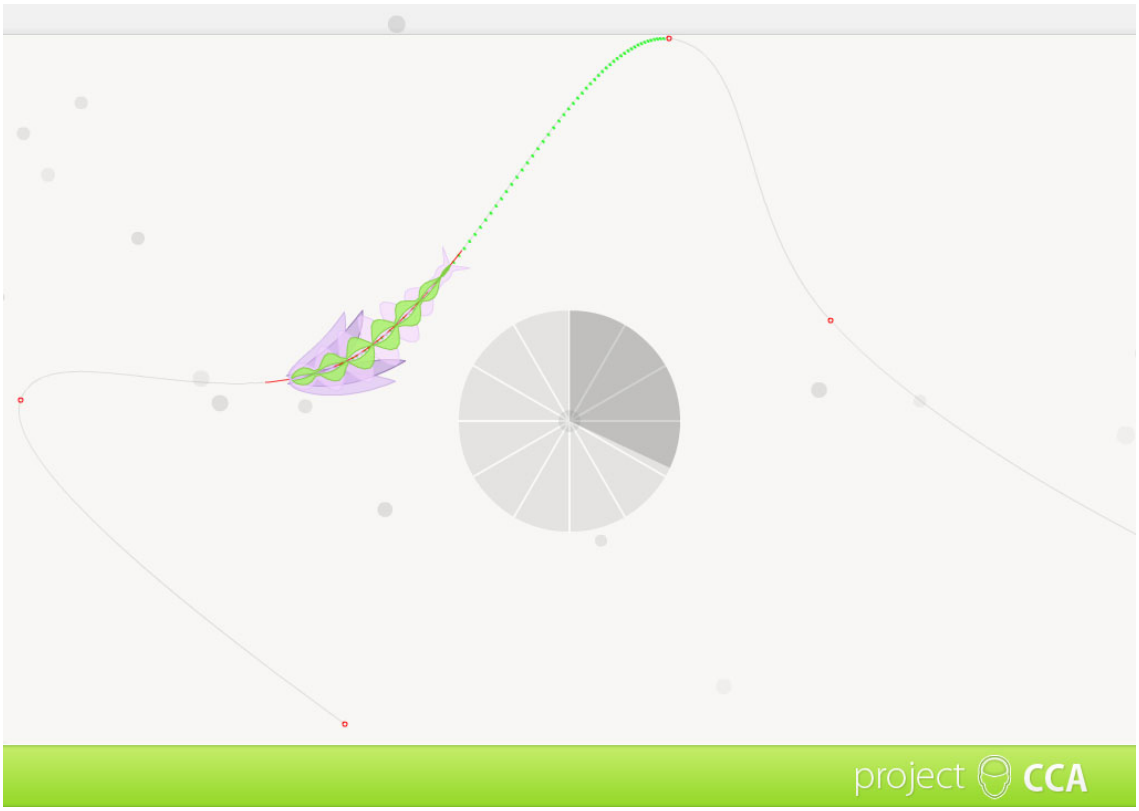


Figure 45. Path creation rendered

A rendering of path creation in CCA can be seen in Figure 45. When the path is created it consists of 5 sections of points, each section has a minimum amount of 100 points between control points. A creature moves between the middle control points 3-4 and whenever it hits the 4rd control point, a new control point is generated on the path and the last control point is removed. When the game begins, first 2 points are generated by random, and after that based on the same rules as the new points in the game.

Because the middle and ongoing segment of the path needs to be unchanged, every decision reaction for the creature takes the time of going through the next 2 segment paths. By making sure the length of the paths per segment is never longer than 0.5 seconds, preferably around 0.3-0.4s, the creatures reaction speed is between 0.6-0.8s which is around the same as the reaction speed of an ordinary human being in a non-trivial task.

6.3.1 Continuous Bezier curves

The paths are based on cubic Bezier curves. Every cubic Bezier curve has two control points and a start and an end point. When making continuous Bezier curves the start point of the next curve is always the end point of the previous curve. Making Bezier curves that continue from each other is relatively simple and requires no complex calculations. But when making smooth movement one needs to make continuous Bezier curves that are smooth, and that means each curve affects the curve before it and after it. An example curve can be seen in Figure 46.

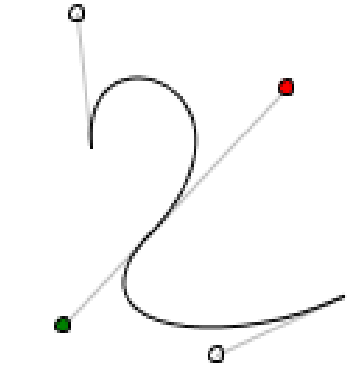


Figure 46. Bezier curve in 2 segments with smooth movement

For the path finding and rendering system to work (And because calculating continuous Bezier curves with the path wrapping isn't cheap), the active creature path at any given moment must be a static Bezier curve. The path will change only when the creature reaches the end of the current path; then the next point on the path is calculated, and rendering continues. Because of this requirement, the path needs to have the five segments, or six points at all times.

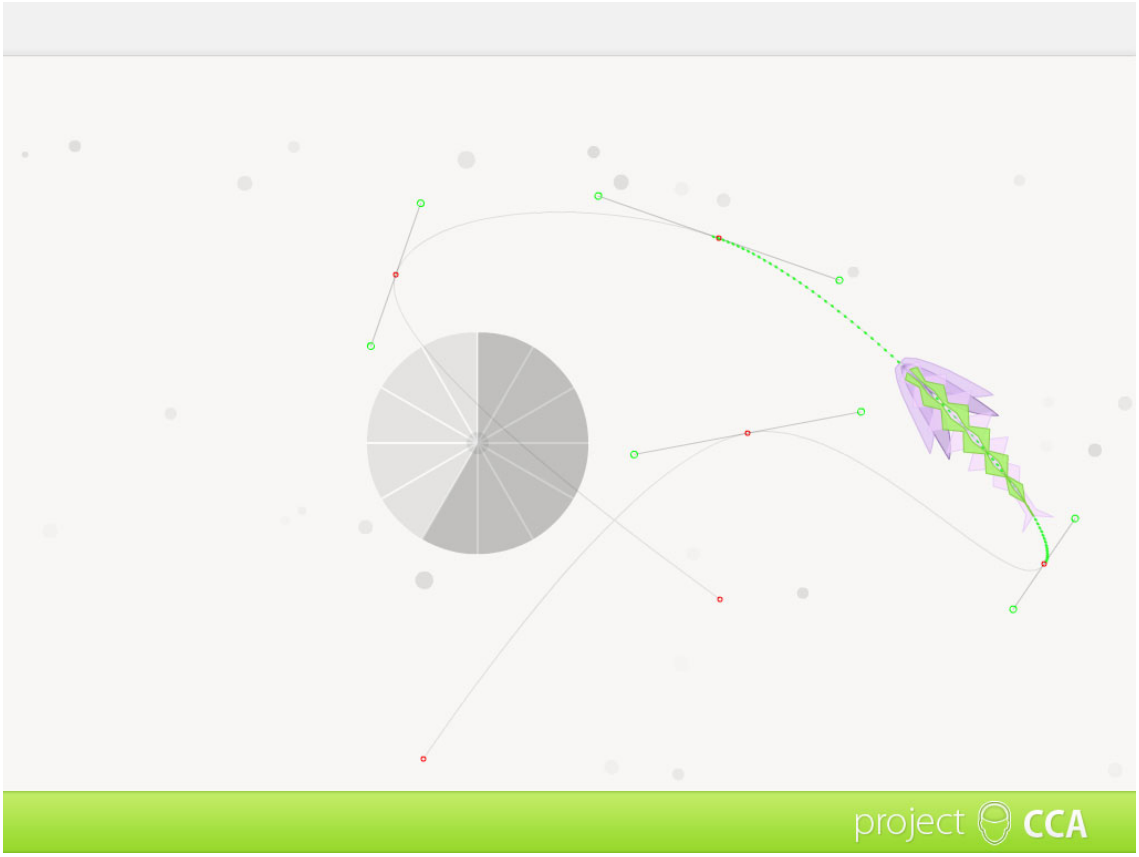


Figure 47. Smooth continuous Bezier curve with control points shown in CCA

Once the point locations are generated, the path needs to be smoothed. Since the motion is continuous through the points, the movement cannot have jumpy or unsmooth sections. To achieve this, the last control point of the previous curve needs to be on a straight line to the first control point of the next curve as seen in Figure 47. (The line goes through the control point and the path point)

For creating cubic Bezier curves, Flash contains a native library called BezierSegment. It is a collection of four point objects that define a single cubic Bezier curve, and has methods for getting the value of the curve at any point on the curve itself. [59] The solution in CCA is based on the BezierSegment class, and more specifically on the work done by Andy Woodruff on the implementation of continuous cubic beziers with AS3 [60].

One issue when using continuous Bezier curves as paths for movement is that when the angles of the previous and next point collide in a straight line (on the x or y axis), the line drawn is a straight line, and by default is calculated without any steps. To overcome this in CCA the straight line situation is checked after the Bezier generation, and

a slight variation is added to the target point to get a proper curve between the points for calculations.

6.3.2 Path interpolation and frame path

In CCA the continuous Bezier curve forms only the foundation for path generation, on which the actual movement is built. Once the base path is generated, the active segment is resampled for smoother movement and simple physics.

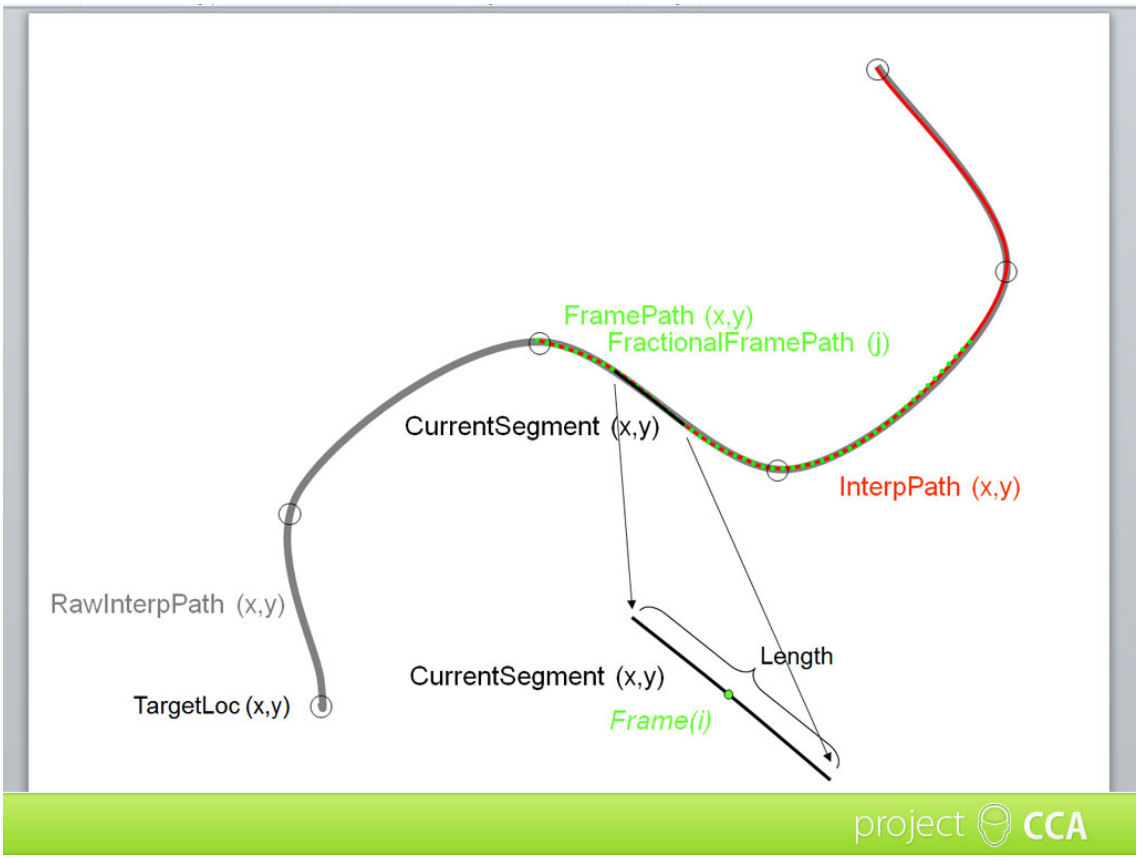


Figure 48. Path creation explained

In Figure 48, the different parts of the path generation are shown. The RawInterpPath is the path created by the smooth Bezier curve generation. After the RawInterpPath is calculated, the active segment of the RawInterpPath is separated as its own entity, called InterpPath. The actual path used by the creature for movement is made by calculating the velocity and displacement of the InterpPath. This path is called a FramePath.

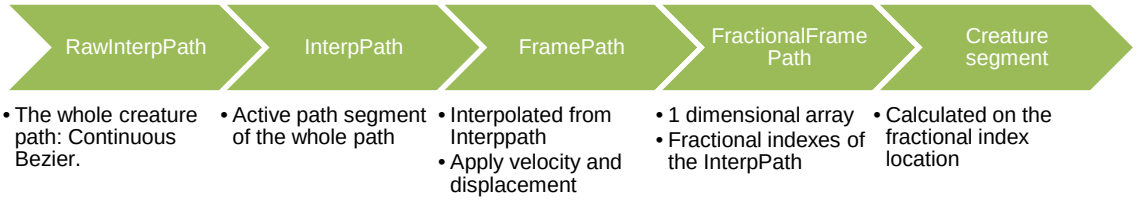


Figure 49. Path creation flow

Figure 49 opens up the complex logic of creating the path. The actual FramePath is not needed for the current segment of the creature for movement. For that a FractionalFramePath is generated from the InterpPath, which contains a fractional index that is used to find the actual movement point in the end. The FractionalFramePath index is used by interpolating a point on the actual curve based on said index. In principle, the FractionalFramePath is the creature location on the InterpPath.

The physics used in the game are fairly simple. The main use for physics as such is to enable the creature to have different speeds based on its mood and the curvature of the path it is on. This creates the feel of organic movement, as the creature slows down to curves and speeds up when going long ways along a slightly curving path.

The path segment where the creature is currently moving on during the rendering is never changed. Because of this, the physics and velocity of the creature are calculated when the path is calculated. In general the path is calculated and physics applied to the movement only at points where the path is recalculated – when the creature reaches a new target location.

6.3.3 Path finding and target location generation

Aside from the mathematics of actual path generation, CCA also contains methods to create path finding for the creatures and boundary methods to keep the creatures rendered inside the actual game area.

There are two modes to the path finding; hunting and foraging. When foraging, the next path point is generated without a specific goal, by using pseudo random logic. In foraging mode, the creature finds its own way around the game area without specific incentives. When in hunting mode, the creature has a target array of points where it aims to

move to, the closest target point is evaluated and an optimal path is generated towards said point. With the closest one calculated, the path finder finds out if it is close enough to nab it in this path, if not it'll create one path segment towards it first and then redo the calculation. If the target is gone, the creature falls back to normal foraging mode.

Figure 50. Path finding in CCA

Figure 50 shows a rough overview of how path finding logic works in CCA. The next target location is generated by combining the orientation (between previous two target points) with a randomized target angle and a speed based on creature length and the angle of movement. The core idea is to have a similar direction of movement to prevent too big, drastic changes in the creature direction.

If the next natural point in a creature's path generation is outside the boundaries of the game area, a bounce method is applied. The bounce calculates how far outside the boundaries of the stage the creature was going for, and changes the angle of movement so that the new point is inside the stage. This is done to ensure the creature is always within the visible target area, and does not wander off the sides of the screen.

After the path generation and bounce factor have been taken into account, the distance between the previous and the new point is calculated to make sure the points are not too close to each other. In case of them being too close, a small speed fix is added to the point's location in order to keep the length of the path relatively uniform. The length of the active path segment is set to be 1…2x the length of the creature to keep the reaction times in check.

6.4 Background rendering

6.4.1 Perlin noise

Perlin noise is a procedural texture, or a pseudo random gradient that is used abundantly in modern computer generated art and graphics for its unique attribute of seeming organic and natural. The algorithm interpolates and combines random noise functions into a single function that generates more natural-seeming random noise. Perlin noise has been described as a “fractal sum of noise”. Developed by Ken Perlin back in 1980s for the movie Tron, Perlin noise has been used in CGI and had a huge impact on computer generated graphics ever since. [61]

Aside from using Perlin noise to draw patterns or textures, it is also very useful for creating organic, smooth movement. The movement is achieved by using the luminance values of a black and white Perlin noise and attaching it to the velocity of a particle engine. Due to the nature of the noise, one can create the noise field once and then change the offset of the x and y coordinate space of a single octave of the noise. With this the noise moves without recalculating the whole noise while maintaining the same visual similarity. Perlin noise is also quite optimized and suitable for performance use.

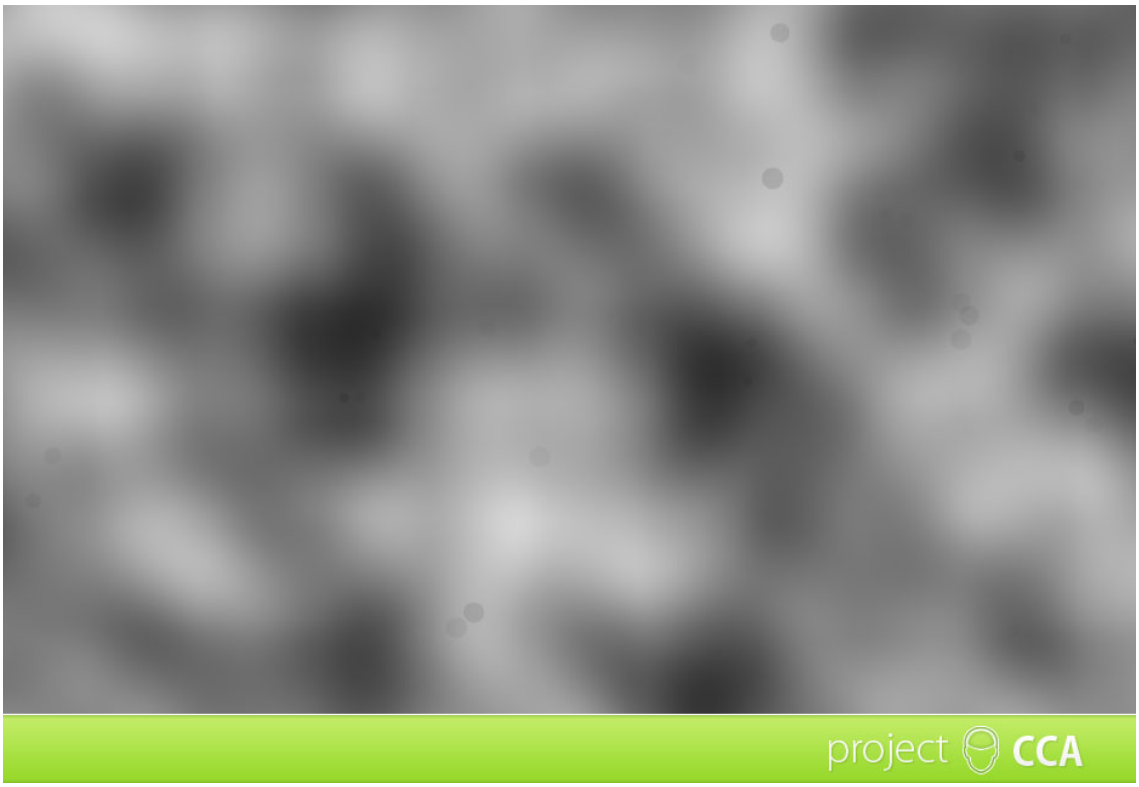


Figure 51. Perlin noise generated in black and white with random seed in CCA

Figure 51 shows a Perlin noise field used in project CCA to move the background particles around. This noise field is randomized on every application run, and resize event, so the movement of the background pattern seems organic and never repeats itself. Perlin noise is rarely used to just create visuals, but rather to blend other visuals together, or to generate objects, landscape and other items that benefit from natural seeming patterns. [61]

6.4.2 Luminance in path generation

The way Perlin noise is used in the background rendering of the CCA system is quite simple. Every resize and init event, a a 2-dimensional Perlin noise is generated to fill the game background layer (no need to make it whole screen sized). It is restricted by the same resolution restriction stops as the main game area, thus simplifying the sizing issue to a single place. This Perlin noise is not visible to a user, and is used only to generate movement.

The noise field is approximately the size of the gaming area, and it is populated with N amount of particles. The amount of particles varies based on the user's resolution, to

create an optimum look versus performance. The particles are simple, round shapes with opacity, randomized to make them look like abstractions of bubbles. In the start the particles are positioned randomly along the noise field.

As previously mentioned, the game has a single renderer that all time based actions are performed on. On every render tick, the position of each particle is changed according to a set offset that is unique to each instance of the particle.

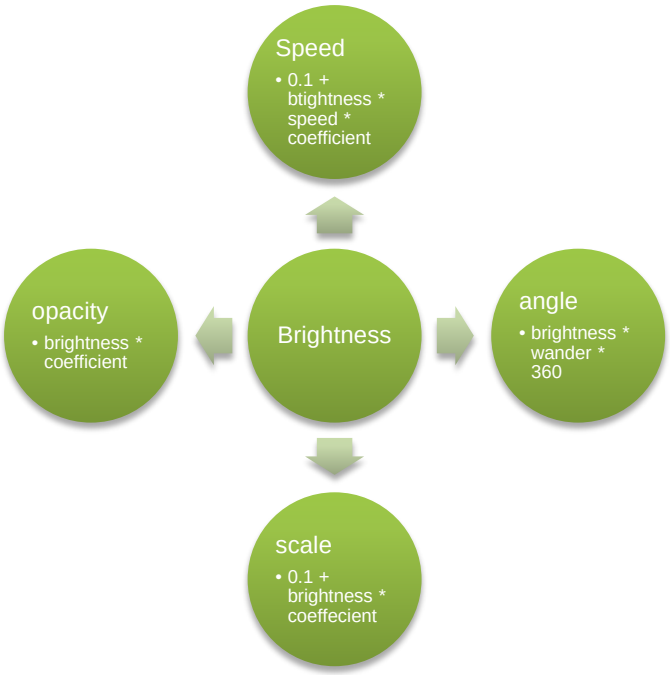


Figure 52. Brightness of Perlin noise and its effect

The opacity, angle, speed and scale of the particle are calculated from the brightness of the pixel at the new position of the particle (Figure 52). The brighter the noise at the coordinates, the faster the particle moves, and the larger it becomes, and vice versa. In addition to this noise based movement, each particle has a unique wander property that affects the navigation of the particle to make the movement a little bit unique. With the smooth, coherent noise we get from Perlin noise, it provides a unique and performance efficient way to generate organic movement.

In addition to working as background organic movement and mood setting, the particles served another purpose. During gameplay, correct reactions triggered a coloring effect on the background particles closest to the creature whose reaction was targeted. There was a set radius around the active creature within which the particles had to be.

The coloring was gradual; a smooth green shade with opacity stops per reaction was introduced. An opposite effect was also introduced; when a user repeatedly triggered reaction without a cause, outside of the reaction window of the target state, the particles were gradually colored with light shades of red. One could get an overview of a game situation at a glance.

6.4.3 Treshholding Perlin Noise for pattern backgrounds

In addition to making movement with Perlin noise, Perlin noise is used in the reference project as a creator of coloured, randomly generated backgrounds that fit the colour scheme of the creatures. The colour match is important as it makes the spotting of creature target states harder when there is less contrast between the background and the creature itself.

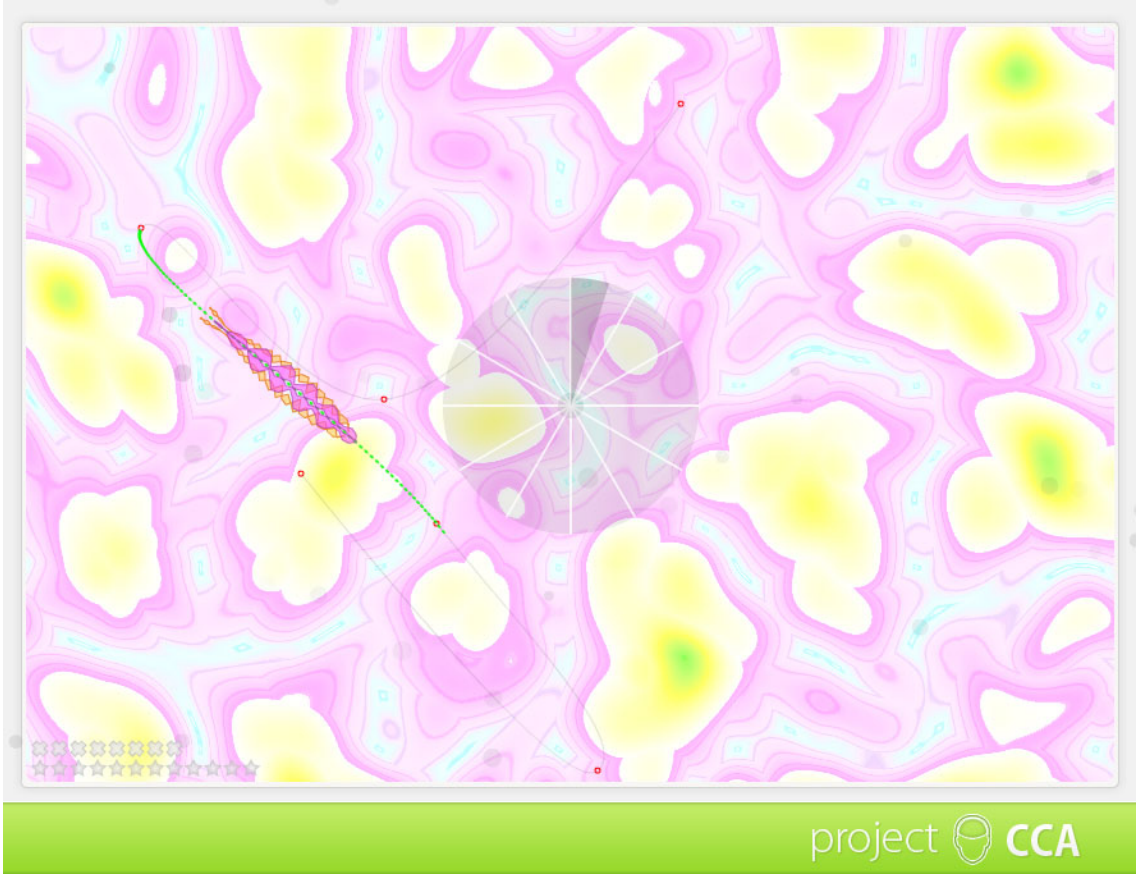


Figure 53. Perlin noise pattern background in project CCA

Figure 53 shows an example of how Perlin noise is used in the backgrounds of certain game modes. It was implemented by partially replicating the Perlin noise in use in the

background animator. The actual effect was developed in true to platform fashion in a parameterized way, where the server provides a possible array of colours to be mapped into the background noise with the creature XML. These colour values are then mapped to the noise based on the brightness values of the noise.

The colours are mapped to the noise by first taking the highest and lowest brightness values in the black and white noise, then mapping the colours to the noise by assigning a brightness index per colour. In CCA a vector of values was used, since it is faster to iterate through an array of bytes than to manipulate an actual bitmap object.

A small trick was used in the resizing of Perlin noise fields since they are quite heavy to instantiate and generate, and there was a need to generate one anew every time the stage was resized. In calculation or rendering heavy tasks, it is not viable to run them every time a user or a program resizes the window, but rather set a small delayed call to the refresh function, and overwrite that delay every time the resize gets called. With this small optimization, one most of the time gets only one re-rendering round, even when the user holistically drags the window around.

7 Performance optimization

> Optimization should always be done holistically. Look at the big picture first and then drill down until you find the specific problem that is slowing your application. When you don't optimize holistically, you risk fruitless optimizations. [62]

The amount of computing resources available to a game is finite. Optimizing code increases the amount of work the engine can achieve per unit of computational power, and maximizes the efficiency of the system. [62] In optimization there are many ways to speed up the code, but it is essential to identify and solve the actual performance bottlenecks within the laws of diminishing returns. The main trade-off to think about when optimizing is the development time versus complexity.

In a project with strict performance requirements such as the CCA, one has to be quite strict and diligent in handling performance bottlenecks. However, untimely optimization leads to chaos and low quality code, it is imperative to find out exactly what needs to be optimized before jumping in. Performance optimization is a balancing act between readable solution code and pure performance gain.

7.1 Basic principles

There are many ways to approach optimization, but the fundamental basis of the optimization lifecycle is testing, or benchmarking.

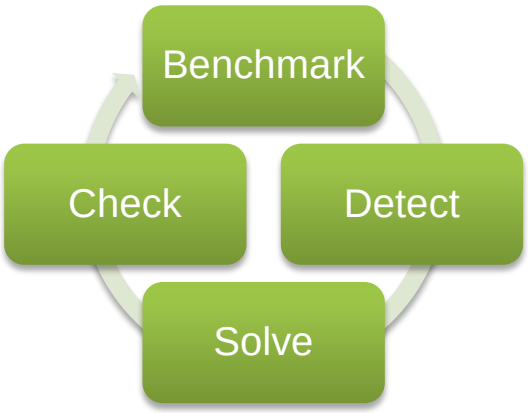


Figure 54. Optimization lifecycle [62]

Figure 54 shows the basic idea of the optimization lifecycle. When optimizing, the essential point is to measure the gains. A benchmark is a point of reference in the game

that serves as a comparison against future implementations. Benchmark should be reliable, quick and it should represent an actual gaming situation. The main use of benchmarks is that they enable relative comparisons.

After the comparison points are gathered, the detection step starts. The point of detection is to find the biggest return on investment for the optimization. In detection phase it is important to start from the big picture, and analyse layer by layer to the finer problem points until a suitable issue is found. After the detection, it is a matter of solving said issue. Solving can be about fixing a bug, toggling a flag, rewriting the algorithm involved or changing the data structure. [62]

Once the solving is done, the check phase begins. When checking, the benchmark is ran again and measured to see if the solution changed anything in the performance. After checking, it is all about repeating the same progress again until the biggest performance gains are figured. The idea of this cycle is to find optimization hotspots and bottlenecks. Hotspots are the points in your program that consume a lot of processing power. Typically hotspots are small amounts of code with a big hit on performance. Optimizing hotspots leads to significant performance benefits. Bottlenecks are particular points in the system execution that clog down the performance of the whole system.

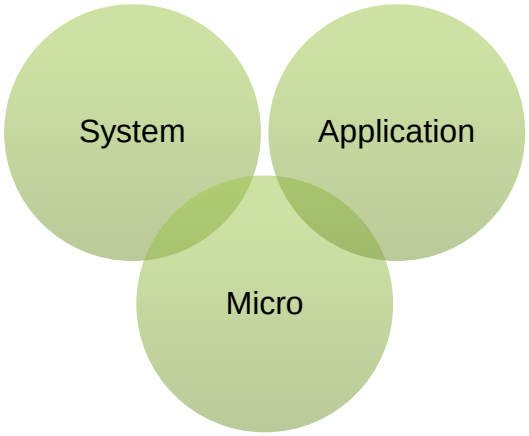


Figure 55. Three levels of optimization process [62]

Optimization can also be approached on a broader level by dividing it into three categories (Figure 55). On system level optimization the focus is on the use of resources of the target platform. The point is to find an implementation that utilises system resources with balance and efficiency. System level optimization is about planning a platform that utilizes the available resources without overusing in a sustainable way.

Application level optimization can also be called algorithm level optimization. It is the choices made in the data structures and algorithms that make up the platform. The idea is to use a good profiler tool to find out call hierarchies and time and iteration amounts of systems most used parts. Algorithm level optimisations are the crucial backbone of optimizing code. The identifying of a key part of code that uses a lot of processing time and optimizing it, or the node that calls it with a more efficient solution is of the best return on development time.

Micro level optimizations are the most common stereotypical types of optimisation. The small hacks on as low level as possible to get a bit of a performance boost out of a specific thing. The optimization of inner loops or pixel rendering routines that are called extremely often and that benefit from the most miniscule of an improvement because of the sheer amount of times they are called during an application execution.

In project CCA the concentration on optimization was made on all three levels. A structural, planning oriented performance review was conducted in the initial phases of the engine building. The algorithmic level was benchmarked and reiterated throughout the application development cycle, and the low level micro approach with rendering and mathematics was made in the engine building phase to smooth out the heaviest of operations in the engine.

Beyond the processes of optimization one must be wary of the pitfalls along the way. It is easy to get stuck on assumptions about performance problems and optimizing prematurely without knowing if it is a big performance sink. Optimizing based on your own machine, or with debug builds gives misleading results, as debug builds are often slower and riddled with processes not found in the final release build. In CCA during development there was a point where the focus was on micro optimizations on the engine rendering loop for too long, and without the aid of a profiler the real optimization pitfalls would have remained hidden.

A common trait in game rendering engines is to render only what is needed and reduce the level of detail dynamically whenever possible. In fact many games could not function without such optimizations due to the sheer size of the game worlds and the amount of rendering. In CCA the amount of rendering is relatively small compared to many of the 3-dimensional engines, but due to the measurements and the nature of the

medical side of the platform it was necessary to render everything on screen without any reduction of rendering quality or level of detail. Because of this, in the creation of the rendering engine it was necessary to try and use all the tricks possible with a managed language to make it as fast as possible.

7.2 Platform optimization

7.2.1 Approach

A good approach to optimizing holistically is to focus on detecting and solving issues on the system level whenever possible, then the algorithmic level, and only if no solution is found resort to micro level optimization.

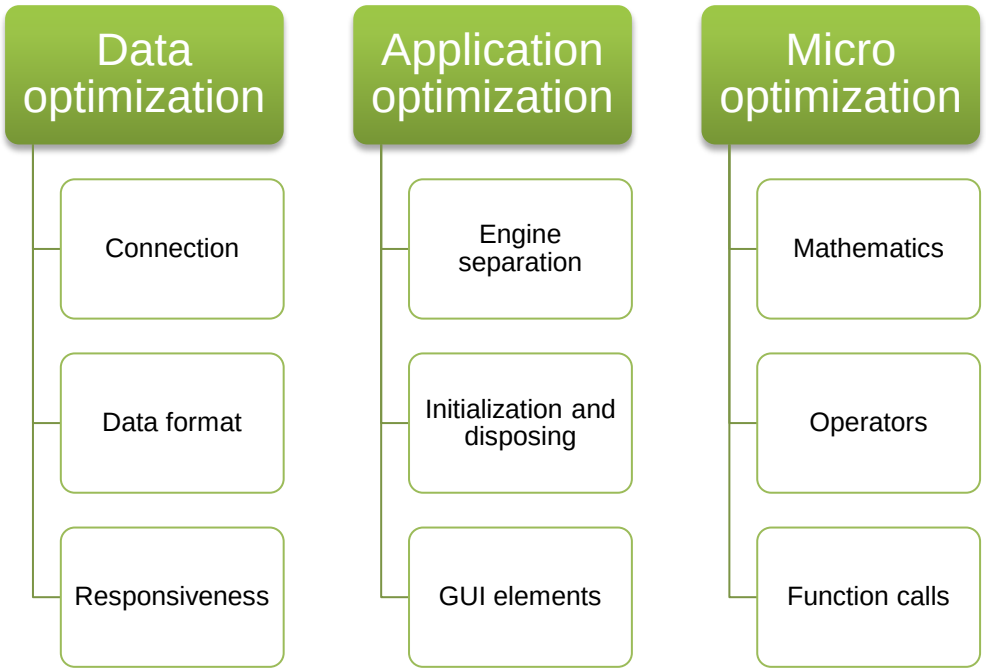


Figure 56. Optimization approach in CCA

The optimization of the platform can be divided into 3 main sections seen in Figure 56. In data optimization, the focus was on the issues of latency and communications with the client-server architecture. The point of data optimization was to minimize the latency of backend calls and limit the amount of data sent between the back and front end platforms. The front end requests data for a view, be it the main user interface, a data

visualization component or a single game, only when initializing the view for the first time.

When non changing data such as the main user interface localization and configuration is loaded, it is cached on the front end and not re-fetched until the user next logs in again. When a response from the system takes between 200ms and 1000ms a user is prone to lose the feeling of flow and the user experience of the system is hampered. [37] For the sake of responsiveness and continuous feedback, the user interface shows a loading indicator whenever a data request takes longer than 400ms. The indicator is not shown on shorter loading times to keep the user flow uninterrupted.

When optimizing the application level, the focus was on making the rendering engine and game logic separate, and to manage garbage collection and memory handling with specific initialization and dispose functionality. The separation of the engine and the game logic optimized not only the development time of the platform, but the way responsibilities were divided between the two.

The initialization of objects in managed languages is costly, and can easily cause variation in the FPS of the system, as well as random hangs in program execution. Also the disposing of objects for garbage collection (GC) can cause the GC to run at times when the user is performing an important part of the platform. The ways of observing smoothness in a system are somewhat immaterial, and cannot be found purely from profilers. Extensive testing and visual assessing is needed to make sure the user experience flow remains unaltered. More information about garbage collection and GUI elements can be found in the Flash specific section 7.3.

The deepest iteration of optimization was done on the micro level; thinking about the mathematics used and testing different operators and in lining function calls and other hacks that produce less readable code but provide a performance boost in a time critical section of the platform. Mostly micro optimizations were done in the core rendering engine, on the calculation and drawing of the creatures and their movement.

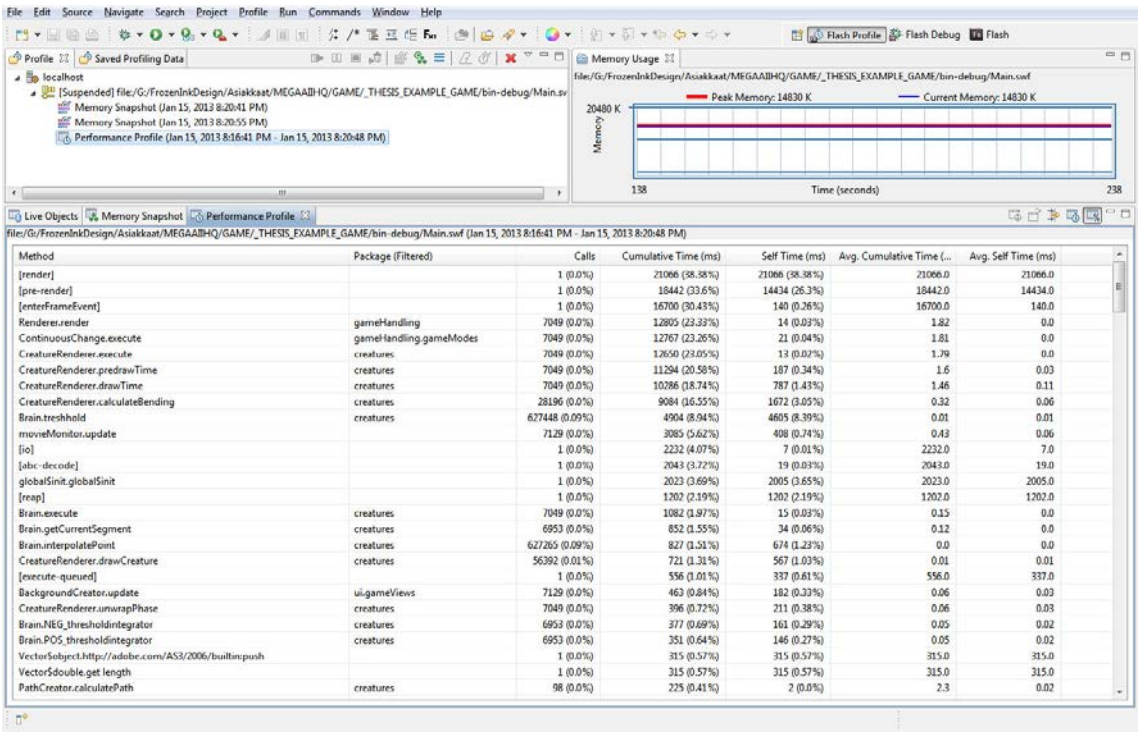


Figure 57. Flash Builder profiler with project CCA

The way to find important performance issues in project CCA was done with using the Flash Builder profiling tool for performance profiling (Figure 57). Profilers are the most common tools for optimization, gathering information about resource usage, most often the CPU load during a session. Profilers typically provide information about the amount of calls, self-time and total time of function calls as well as memory used in them. [62]. In CCA the profiler helped in generating memory footprints and locating high resource hogging sections of the rendering code, as well as finding bugs in disposing of objects.

7.2.2 Creature optimization

In CCA the creature rendering model is a very versatile implementation of a data driven architecture. The creature parameters define how they each formula in them is plotted on screen and the detail level of each creature can be adjusted per formula as well. Each mathematical formula has an amount of points that are calculated and plotted to render out its true form, and these can be adjusted and manipulated with relative ease. The lower the amount points rendered on screen, the less fidelity in the creature, but on low end machines, or situations where there are many creatures on screen, it increases performance smoothly.

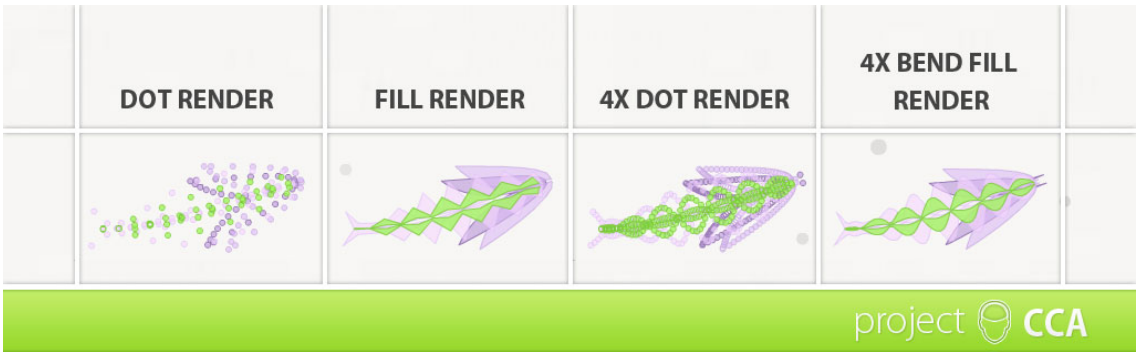
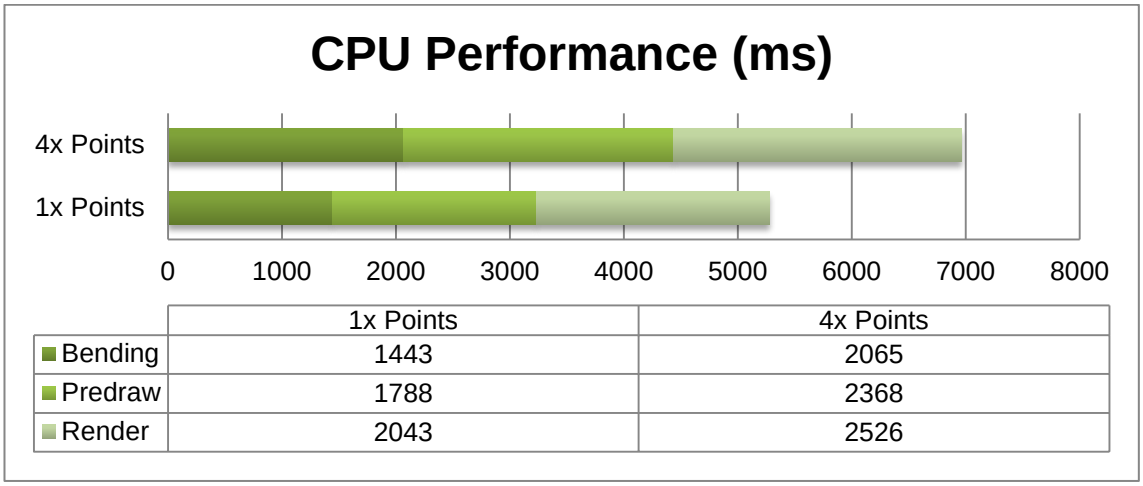


Figure 58. Variance in fidelity by dot rendering change

Figure 58 shows the clear difference in rendering quality based on the amount of control points or dots used in the creature generation. The overall shape and visual style of the creature is the same between the fill render and the 4x fill render, but the subtle details of the creature are more smooth and round in the 4x fill render than the normal fill render. The dot render shows the inflated dots generated by the plotting of the formulas, the dots themselves are used in the rendering only as the control points for the lines that shape the creature itself.



	1x Points	4x Points
Bending	1443	2065
Predraw	1788	2368
Render	2043	2526

Figure 59. Performance change in dot rendering

The difference in cumulative CPU calculation time in milliseconds between the 1x and 4x dot renders is shown in Figure 59. The values benchmarked by using the Flash Builder profiling tool set show that the rendering of 4x the amount of points on the same creature increase the calculation time approximately 32%. The difference is to be expected, as the fidelity increases, so does all the major computation in the creature

rendering and path bending. More difference could be obtained by adding a formula for the creature generation itself instead of adjusting the points of the formulas calculated.

Different creatures can have different dot amounts for visual fidelity. Some formulas work better with low amounts of dots, while others require much more to look and behave properly. When the creatures are moving, the tighter turns they take, the more clear the complexity of the creature; if the creature consists of a low amount of dots, the lines between start to show when doing above 90 degree turns quite visibly. There is a small implementation in the path creator that tries to optimize creature movement in a way that it never does such high degree turns.

Due to the data centered design of the rendering, the creatures can be optimized based on user accounts, and user machine performance to create the best possible outcome. For most formulas in CCA, the conversion of the formulas to native code could be done in the initialization and the calculations for the necessary values out of them only once, and then reuse the plotted model of the creature in all the path bending operations without recalculating the plot. The performance gains were significant. In addition to the fidelity of the creatures, additional features can be optimized, such as the background Perlin noise particle animation can also be toggled as needed.

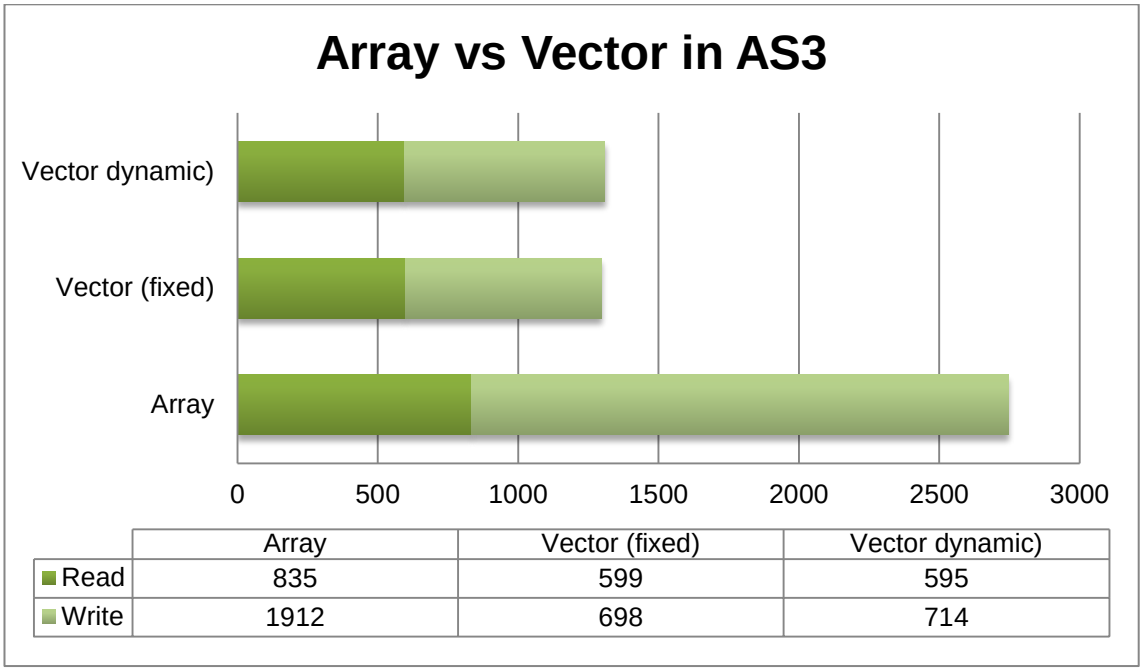
7.3 Flash specific optimization

7.3.1 Overview

The Flash platform, or more specifically the Actionscript 3.0 (AS3) execution model, is single threaded. Because of the lack of threading, all sufficiently heavy actions on the UI thread slow down or stop the rendering of the whole UI layer. While being robust for a web technology, it is a severe limitation of processor use. AS3 is also a managed language, interpreted by a runtime, which leads to system managed memory and garbage collection.

There are various issues when profiling with managed languages, the nature of the byte code leads to more native level calls than pure native languages. The way AS3 handles array actions is more involved than native languages; every array is implemented via a generic data structure, so every array load means hitting that data struc-

ture and querying it for type information. [62] Because of the cost of even basic operations can be much more varied than in native code, it is beneficial to set a good baseline for performance data to know what to avoid.



	Array	Vector (fixed)	Vector dynamic)
Read	835	599	595
Write	1912	698	714

Figure 60. Array versus Vector operations in AS3 [63]

The basic approach in optimizing for AS3 is to always use strictly typed operators everywhere in your code; it provides a significant runtime performance boost. Another way of optimizing what is needed in AS3 came with the Flash Player 10 release. Flash had no history of supporting typed arrays before the introduction of Vectors in FP10. Vectors have a significant overhead in declaring and wiping them, but they are extremely robust when iterated over significant amounts of values. [63] The difference between array and vector use is significant, as seen from Figure 60. The reading speed of vectors is roughly 40% faster than with traditional arrays. The writing speed difference is even greater, roughly 170%.

```
package
{
    import flash.display.Sprite;

    public final class SpritePool
    {
        private static var MAX_VALUE:uint;
        private static var GROWTH_VALUE:uint;
        private static var counter:uint;
        private static var pool:Vector.<Sprite>;
        private static var currentSprite:Sprite;

        public static function initialize( maxPoolSize:uint, growthValue:uint ):void
        {
            MAX_VALUE = maxPoolSize;
            GROWTH_VALUE = growthValue;
            counter = maxPoolSize;

            var i:uint = maxPoolSize;

            pool = new Vector.<Sprite>(MAX_VALUE);
            while( --i > -1 )
                pool[i] = new Sprite();
        }

        public static function getSprite():Sprite
        {
            if ( counter > 0 )
                return currentSprite = pool[--counter];

            var i:uint = GROWTH_VALUE;
            while( --i > -1 )
                    pool.unshift ( new Sprite() );
            counter = GROWTH_VALUE;
            return getSprite();

        }

        public static function disposeSprite(disposedSprite:Sprite):void
        {
            pool[counter++] = disposedSprite;
        }
    }
}
```

Figure 61. Object pooling in AS3 [64]

In AS3 the construction of new objects is very costly. When creating the creatures and parsing the formulas from XML to AS3 native math functions, it was useful to use a technique called object pooling. In object pooling you declare your objects only once and store them in a list. Instead of creating new objects when you need them, you use the objects already declared and stored from the list (Figure 61). The benefits of object pooling are greater when using more complex primitives; the increase in performance is 5x when using a basic type such as a Point, and over 80x when using a complex Sprite type. [62] In CCA each creature is object pooled, but so is each set of formulas that form a part of the creature.

To prevent the UI from slowing down in CCA, the initialization of objects into the object pool was also optimized. In the game handling initialization the creature classes belonging to the game are instantiated; The Perlin noise backgrounds are created, as well as all the creatures. When creatures are not on screen or do not belong in the current play mode, they are hidden from view and removed from the display render loop. Since

the game core is about constant perception of the target moving, it is imperative not to have high changes in frame rate, on average even if the rendering frame rate is not the full required 30fps, the users aren't as phased by the difference as long as it maintains a steady pace.

Alongside the optimizations mentioned here, there are great many smaller point solution optimizations used in CCA and available for the Flash platform. Links to further reading on the subject can be found from the associated references in the end of this thesis.

7.3.2 Function calls and language specific libraries

One of the basic optimizations to do is to remove loop invariant code. When doing large quantities of iterations, the cost of declaring iterators within loops and inner loops can be quite high, especially in managed languages such as AS3. It is useful to try and minimize the amount of variables declared inside loops when it is not necessary, even pre-declaring the loop iterators, let alone variables used in the loops saves performance. [62, 64]

Another optimization that tends to be bad for code readability, but that can help optimize code that is heavily used during rendering is the inlining of functions in your code. When moving functions inline, one must be wary of the cost for code readability, it significantly hampers the future development of the code base. Inlining calculation heavy functions can amount to a performance increase of 400% in AS3. [64]

Other uses for micro optimization are the manual redo of language specific mathematical libraries. There are many ways to solve basic things; such as the absolute of a number, via casting to int or using a ternary operator that can be faster than using the platform inbuilt Math.abs(). Small optimizations in AS3 can also be gained by using the right iterators for the right loops, for example when doing while loops it is faster to loop through in reverse order than forward looping. [64]

In CCA the micro level optimizations were used solely in the core rendering and path finding engine. There was an effort to keep the unmanageable only in the areas identified by profiling to require significant tuning and optimization. Function inlining is used in some processor heavy mathematical operations and reimplementation's of things

like Point.distance() and Math.abs() to improve low level efficiency. For most of the platform code algorithm and system level optimizations were preferred.

7.3.3 Bitwise operators

Another way of optimizing performance heavy bottlenecks in calculations and iterations is to use bitwise operators instead of verbose solutions. Since computers operate in binary, using bitwise operators can be a powerful programming tool. Often a few quick operations can easily replace what would otherwise be complex and heavy code. In certain situations, using bitwise operators can even amount to clearer code. [62]

```
x = x * 2;
x = x * 64;

//equals:
x = x << 1;
x = x << 6;
```

```
x = x / 2;
x = x / 64;

//equals:
x = x >> 1;
x = x >> 6;
```

```
x = int(1.232)

//equals:
x = 1.232 >> 0;
```

```
eqSign = a * b > 0;

//equals:
eqSign = a ^ b >= 0;
```

```
i = -i;

//equals
i = ~i + 1;

//or
i = (i ^ -1) + 1;
```

```
//version 1
i = x < 0 ? -x : x;

//version 2
i = (x ^ (x >> 31)) - (x >> 31);
```

```
x = 131 % 4;

//equals:
x = 131 & (4 - 1);
```

```
isEven = (i % 2) == 0;

//equals:
isEven = (i & 1) == 0;
```

Figure 62. Common bitwise operations in AS3 [65]

Common bitwise techniques in AS3 revolve around using the inherent quality of binary operations; multiplying and dividing by the power of two, integer conversions, sign conversions, modulo, absolute value, minimum and maximum seeking and color conversions. Figure 62 shows common alternatives to operations in AS3 by using bitwise operators for efficiency.

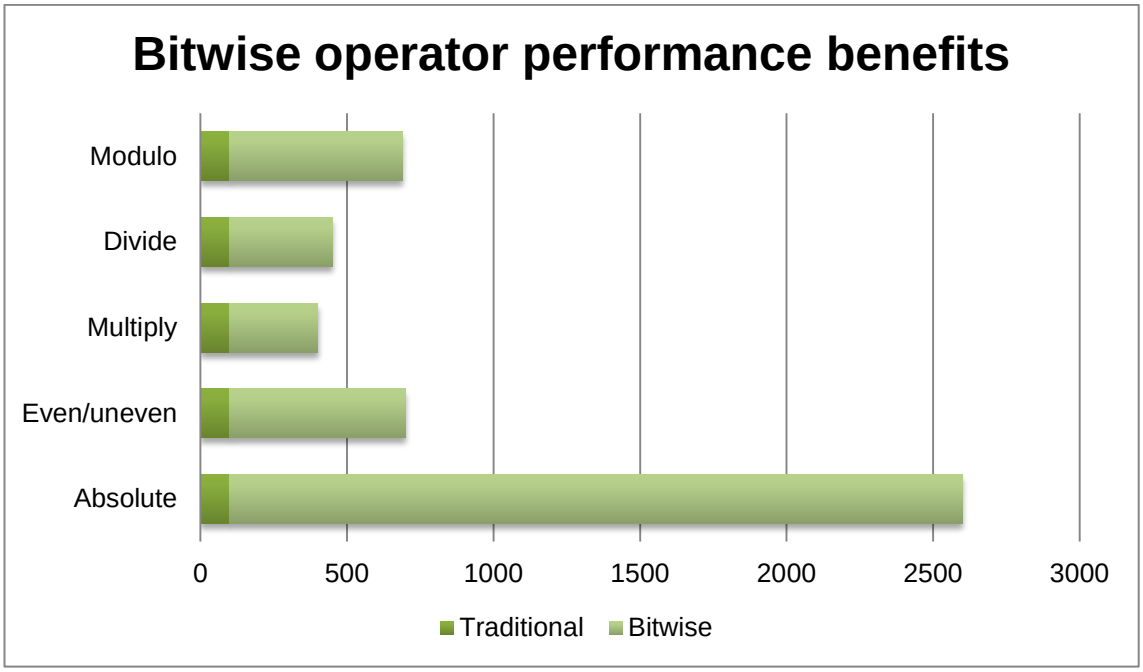


Figure 63. Performance benefit estimations for using bitwise operations in AS3 [65]

Performance benefits for using bitwise operators in AS3 can be immense. Figure 63 shows the benefits of using bitwise instead of natural logic in common logical operations. The benefit is especially clear when using the bitwise method instead of the Math.abs(), an increase of roughly 2500% can be achieved. Basic functions such as multiplication and dividing run around 300-350% faster than the basic solution. When testing the modulus of a number the performance is around 600%, similar to testing if a number is even or uneven.

When optimizing the CCA rendering engine the approach was to benchmark key rendering situations, and find out the most performance intensive mathematical operations and optimize them. The most used and heavy functions were optimized with various micro optimizations, including removing loop invariant code, declaring variables together and bitwise operations.

7.3.4 Drawing solutions

Flash supports both Vector (Not to be confused with the Vector of typed arrays in Flash) and bitmap rendering. Both of them have their own benefits; drawing with vectors enables crisp, smooth and scalable rendering, as well as quite handy tools for handling the graphics. Whereas drawing with bitmaps allows for easy manipulation and

bitmap specific filters such as PixelBender, and especially with complex and large graphical content, performance benefits. [64]

There are many ways to optimize the drawing of content in Flash. One good tool is to use the redraw regions debug option to visibly observe what regions of the screen are “dirty” and redrawn on every frame. Even though content is opaque, if it is on the display list it can still trigger hit tests and other runtime performance hogging tests as long as its visibility is set to true. One can also set the platform level toggle on movie quality to a lower amount; this affects antialiasing, smoothing of scaled bitmaps, fonts and animations.

Other performance heavy operations in native Flash drawing are the use of filters and blend modes. Filters such as drop shadow and blur need to be applied on the whole object they affect during render time, unless specifically set to cached. Efficient use of cached effects and minimizing the amount of alpha blending can also help rendering performance, especially on lower end hardware.

Most vector content in Flash can be cached as bitmaps and reused, even changing the objects x and y coordinates does not affect the caching benefit. However caching cannot be used for any content that needs to be redrawn every frame, such as the creatures in CCA.

In CCA both bitmap and vector based drawing methods were studied. The target was to find which would be most ideal for rendering the CCA creatures. Vector based rendering gives one smooth controlled geometrics in a fully scalable way, enabling graphical fidelity and an ease of creation. As opposed to vectors, bitmap rendering is purely pixel based, and as rendering goes, it is typically significantly faster than vector based rendering. Handling bitmap rendering is somewhat trickier than doing it in vectors, especially if you go with just pixel painting, since you need to manually take into account antialiasing and corners of the creatures

Since the creatures are plotted from mathematical formulas into vector shapes in CCA as the basis of rendering, using pure vector based rendering on the Flash platform level was an obvious choice. In the initial engine tests, there were no significant increases in performance from using bitmap based rendering over vector based rendering, as most of the CPU power went into path finding and creature calculations.

8 Results and future implications

8.1 Current situation

After the past two and a half years of development project, CCA is a mature, bug free platform and currently deployed in clinical trials as well as touring the Science Changing the World Exhibit. It has proven to be a useful tool for cognitive gaming; in preclinical alpha version testing, users were generally pleased with the tool. There were no harmful effects or discomfort reported, and 20 users out of 21 found the game appealing.

Despite the fact that the project got off to a good start, it was a large piece of software for two people to handle. There were many ideas, presentations and funding rounds to try and ramp up a company built around it to make it a viable consumer market platform for cognitive gaming, but none of them have succeeded so far.

Regardless of it not being as big as originally hoped it would be at this point, it is a solid work of engineering, user experience, science and hard work. CCA has performed well under tests and clinical trials. It is currently being translated to other languages and has been developed to be fully multilingual, based on different language resource XML files that are easily configured via the server interaction. There is a single simple API for language versioning that can be triggered on any back end call, so users can change the language at any point during platform use.

After the current clinical trials, there have been rough plans and ideas about the possibilities of reworking the core engine for mobile platforms, as well as ventures into the world of 3D, but the future of the platform remains open. The online version of CCA can be found at www.mybraincapacity.com.

8.2 Clients and users

The current version of project CCA is being used mainly in ongoing clinical trials in conjunction with HermoPharma (www.hermopharma.com), in their amblyopia studies. There have been plans for a mass market version and an open account creation, but for now the plans for large scale deployment are on hold.

Possible uses for the project CCA platform range from helping aged people hone their cognitive abilities to helping the rehabilitation of people after experiencing brain trauma. It has proven its capability as a cognitive gaming platform, and can be used for accurate user statistics and ongoing analysis of user's progress, as well as measuring the effectiveness of drugs, by providing detailed data about gameplay and improvement.

The on-going clinical trial at the moment of writing this Thesis is related to ambylopia, also known as the lazy eye. A fairly common disorder characterized as vision deficiency in an eye that is physically normal and able. Ambylopia is estimated to affect the lives of between 1-5% of the population [66]. Hermopharma are in the process of developing a drug based on antidepressants to prevent or heal damage caused by ambylopia (www.ambylopia.fi).

Figure 64. Amblyopia.fi campaign for HermoPharma study [67]

In the trials, project CCA is being used as a measurement and a training program for the test users to gain accurate information about how the medicine works, and to see if the training of the affected is efficient in improving results. Figure 64 shows the campaign for the clinical trial. In addition to being used as a measurement device for the actual clinical trials, a standalone implementation of a single game mode was made to

support testing for one eye at a time. The purpose for this version was to act as a campaign game for people interested in the possibility of amblyopia affecting their lives, giving them a preview of what the trial was about and to raise interest and awareness for it.

8.3 Heureka multiplayer version

8.3.1 Introduction

Figure 65. Science Changing The World Exhibition [68]

On top of the main game platform build, we were approached by Heureka (www.heureka.fi) to do a multiplayer version of the game for a new exhibition called Science changing the world they were doing in conjunction with 3 other leading science centers. The exhibition is a testament to science; a playful series of exhibits to interact with, and learn about new things in science and the ways health, life quality, even the planet can be improved with scientific methods. The show started at Heureka from 15.4.2010 (Figure 65) up until 15.1.2011. After that it has toured the Museon, the Hague in 2011, the Cité des sciences, Paris in 2011-2012, and it is in the Pavillion of Knowledge in Lisbon until 15.7.2013. After Lisbon, the exhibition is up for rent.

What Heureka wanted was an offline version of the rendering engine, with a specific game type and about 10 different game configurations and creature combinations. It differed from the main game in a few important aspects; It was not a training schedule

based system, having no server interaction or user accounts, but a pure front end implementation, and it was to be played by up to four people at the same time, competing in cognition around who achieves the highest score. The game was also in 3 different languages, and the whole user interface of the game was redone to fit the resolution and language requirements.

Due to the modular nature of the platform and engine code, it was possible to separate the rendering engine with a game logic part and implement the new multiplayer requirements without too much trouble on top of the custom version. It was mutually beneficial, as some of the improvements made in the Heureka version ended up being ported back to the main game platform.

An additional other challenge of the Heureka version was that it was run on a 46” touch screen on a full high definition television (1920*1080px). That meant that the whole game area that the game runs on was relatively bigger than the typical CCA platform game area (around 1280*800px). Because of this it required extensive optimization and calibration of the points rendering and creature styles to make it perform well enough for show use.

8.3.2 Multiplayer

The multiplayer mode was a single mode of the N-object capacity test; the game featured 1 to 5 creatures at the same time on screen, and the users had to react to each target state of any of the creatures within a set time frame. Each successful reaction per player was colored with its unique shade of color glow around the creature the reaction affected to keep it clear on who did the reaction.

Figure 66. Heureka multiplayer version

Example gameplay of the Heureka CCA version can be seen in Figure 66. The game was played in a customized stand, with the 46” screen facing upwards. All controls for users were abstracted from the traditional keyboard, and each player had an arcade style control button to use to react. The custom controls were connected to a keyboard input, and could be mapped into the CCA Flash application via normal keyboard events.

In addition to the changes in the game mechanics, the multiplayer mode required its own user interface. In the multiplayer version, there were between 1 to 4 players, and the game needed be playable with any number of people between that range. The UI had to work in four directions, matching the four player slots around the stand.

To enable the users to play at the same time, a separate UI based on the same logic and graphical design of project CCA was implemented. The UI was built so a separate instance of for each direction was created and shown based on user participation. Each side of the game had an initial information screen informing the users about what they are supposed to do, and the game started by anyone pressing the start button. The game allowed hot seat style multiplayer, where anyone could jump in during the game and start reacting to the creatures, albeit coming in later would affect your score in the end of the game.

Only the players who were active during the game run were shown the game end screen with detailed information about how they did. In addition to the statistics, a crown reward system was implemented; the best user was rewarded with a gold crown, the second player a silver one, 3rd player got a bronze crown and the 4th one had to settle for nothing.

8.3.3 AIR platform

As opposed to the online version of the main platform, the Heureka version had to be a standalone installed package, with everything needed inside a single installer file. To do this with traditional Flash was impossible, and third party extensions were not reliable enough. Luckily Adobe itself had come up with a solution to the problem. Adobe AIR or the Adobe Integrated Runtime is a wrapper for the Flash player that allows it to handle native functionality, from file system access to running its own instance. [69]

Adobe AIR enables the use of web standard HTML and Javascript, as well as Actionscript 3.0/Flash to create native applications for all major desktop environments, as well as Android, iOS and certain smart TVs. [69] It is widely used in creating casual games for mobile platforms, as well as a lot of entertainment and ad campaign applications. Air has full support for Stage 3D OpenGL API for accelerated graphics, as well as a lot of native integration to iOS and Android services.

Developing for AIR brought about some small changes to the CCA codebase because it handles certain things like stage events and window events as well as keyboard interaction a bit differently, but all in all around 90% of the code could be ported to AIR within days. The in-built installer, application visuals such as icons and automatic up-

dating functionality made the AIR version completely stand-alone, and enabled a solid experience for the people keeping up the show at the various science centers.

8.3.4 Testing, long uptime, automatic start up and maintenance

The biggest issue to solve in the Heureka version was the requirement for the game to run on its own for a minimum of 16 hours in a row. The game was automatically started at the beginning of a day, and the whole machine was shut down at the end of a show day. The application needed to be maintenance free during uptime, and it should be trustworthy and robust throughout the whole life time of the show.

The approach to the Heureka version required a lot of profiling and optimization. Before deploying the build in the actual environment, a series of test scripts were implemented. The scripts made the game play itself for a long time, changing the game parameters and reactions to simulate a real user situation. A logging of memory footprint and performance details was also implemented. These features enabled the testing of the game before actual deployment.

But as optimizations go, it is rarely beneficial to test only in development environments, as was the case in Heureka. Using the scripts the game seemed to work fine for the required 16 hours without memory leaks or slowing down, but during the actual show run, it started to show problems. We kept getting reports that the game randomly crashes after 2+ hours, depending on how many users have been using it and the rate of use during the day.

Al lot of time was spent trying to track down what caused this elaborate bug; by the Flash logs it showed to be running fine, and that memory use was not stacking up and rendering times were normal. But from the OS point of view, the app slowly ate away at the memory space allocated and crashed in the end when it had consumed too much of the system memory.

The reason this fatal bug was not noticed in the internal tests was because the tests always ran the game to completion, to the end screen, and then began a new round. It became apparent that the crash happened when time the game was started, but abandoned mid-way through without playing it to the end numerous times. (The game had a

timeout function, if it was not played for a bit, it started showing instructions on what it does and encouraging people to play).

In the end after some serious time spent with the profiler and taking memory snapshots in different benchmark situations, a memory leak was found. A creature rendering class in the core engine rendering system was being reset for garbage collection with a call to not the correct level of inheritance, but the super class of it. This left the game with a small amount of assets each game creation that weren't being cleaned up and slowly started to eat away at the memory available.

This problem had not been found during the main CCA engine development, due to the fact that the main game was never supposed to be used for more than a maximum of about an hour at a time. A lesson was learnt here; profiling and being rigorous about testing the memory footprint is imperative and extremely valuable when creating continuous running software.

Despite the problems during the development, and the rework needed to create the Heureka version, it is up and running around Europe at the moment. The branching was a success, and even if nothing else big will come of the project CCA, at least it got to educate people around Europe about science and cognition.

8.4 Future

At the writing of this thesis, Matias is doing the clinical trials with HermoPharma, and the language versioning system enables fast and easy porting to other language areas. The Heureka version is still circling Europe. The current version is complete, but there are a few routes that have been pondered about for where to take CCA next.

One obvious platform for brain gaming and cognitive use that appeared during the development of the CCA platform and that offers lucrative amount of users and more importantly users that might benefit from the platform is the mobile space, especially the rise of the tablets. Tablets fit perfectly with the way the project CCA core works, and could be easily a good place to start spreading out to.

Due to the fact that Flash platform is not available for mobile except for Android, it is not a viable option for creating the project CCA for mobile. Even though the Air platform

would support building to mobile environments, the computing power required on top of the managed language would most likely be too high for mobile CPUs. The heavy rendering and calculations required would need a native application to be built for CCA platform. On the other hand the core of the CCA engine is reasonably easily converted since it is very mathematics based.

The other way aside from mobile platforms is to evolve the CCA forwards towards the 3D space. The same engine would not work in a similar fashion in 3 dimensions, as it is currently a very much 2 dimensional solution, but the same ideas and approach could be viable in 3D. The serious gaming around platforms such as Unity (www.unity.com) is growing at a significant pace, enabling efficient solutions for beautiful yet useful gaming.

There are some initial steps towards looking into the possibility of building a next version of CCA with the Unity engine toolkit. Unity would provide a unified platform for all native computers, as well as a way into consoles and mobile gaming. Then again the advances in web standards and the rise of WebGL could enable a remake of CCA with Javascript and HTML5 Canvas based approach. The future remains open.

9 Conclusions

Cognitive gaming is a rising area of serious gaming. The approach that a brain can be trained and moulded to perform better at tasks we perceive to be meaningful is a exciting thought. In this day and age efficiency is one of the top most factors working life focuses on, and the ability to improve not only working methods or tools, but people themselves provides valuable and genuine possibilities.

With the rising popularity of mobile platforms and the improvements in traditional web, the focus has shifted from creating features to providing good user experience and servicing user needs. Going beyond usability, the importance of brand and aesthetics are also gaining ground in realizing products and experiences in the online world. User experience matters.

The approach to cognitive game platform building in this thesis is generable to serious game platforms as a whole. The issues solved relate to common, real world problems when creating performance heavy online implementations, and provide ways to overcome obstacles in the creation of good user experience as well as system performance. The project approach emphasizes a focus on user experience and solid software engineering.

The reference project is a complete cognitive gaming product; a feature rich user experience oriented platform for measuring attention and working memory. It is based on a data driven architecture with proper separation of concerns for rendering, game logic and user interface elements. The modular approach allows good controllability and modifiability without changing the front end platform code.

The rendering engine provides organic movement and creatures with a natural feel in endless combinations based on mathematical formulas. It is optimized on the algorithmic and micro level to provide pleasing visuals and fast enough rendering for attention games. Due to the separation of game logic and rendering, as well as the heavy emphasis on mathematics in the rendering, the engine is easy to export to different platforms in the future.

The project was a success, and is being used in clinical trials in Finland and Estonia as well as in the Science Changing the World Exhibit around Europe. User feedback has

been encouraging during the alpha and the current medical trials. The future remains open, but the direction is towards mobile and tablet use, as well as employing 3D as a means for more immersed gameplay.

This study and the reference project stand as an example of the viability of modern web technologies in creating complex serious gaming platforms. It is a testament to the possibilities of cognitive end user training and serves as a guide on the approaches necessary to accomplish a successful cognitive gaming project.

References

1 Agile software development with scrum. Ken Schwaber, Mike Beedle, Prentice Hall, 2002.

2 Silverlight technology stack
http://msdn.microsoft.com/en-us/library/bb404713.aspx

Accessed 27.12.2012

3 Flash Platform statistics
http://www.adobe.com/products/flashplatformruntimes/statistics.displayTab3.html

Accessed 27.5.2012

4 Silverlight statistics
http://www.statowl.com/silverlight.php?&timeframe=custom|2009-01|2011-01

Accessed 27.5.2012

5 Flash Platform overview
http://www.adobe.com/devnet/flashplatform/articles/flashplatform_overview.html

Accessed 28.12.2012

6 Apple OSX versions and release dates
http://support.apple.com/kb/HT1159

Accessed 28.12.2012

7 Flash Platform technical details
http://www.adobe.com/fi/products/flashplayer/tech-specs.html

Accessed 28.12.2012

8 The Sharp Brains Guide to Brain Fitness: 18 Interviews with Scientists, Practical Advice, and Product Reviews, to Keep Your Brain Sharp. Alvaro Fernandez, SharpBrains (April 30, 2009)

9 The Brain That Changes Itself: Stories of Personal Triumph from the Frontiers of Brain Science. Norman Doidge (Penguin, 2007).

10 The musician's brain as a model of neuroplasticity.Munte TF, Altenmuller E & Jannke L Nature Reviews, Neuroscience (2002)

11 Neuroplasticity: Changes in grey matter induced by training. Draganski B, Gaser C, Busch V, Schuierer G, Bogdahn U & May A Nature (2004)

12 A Cognitive Training Program Designed Based on Principles of Brain Plasticity: Results from the Improvement in Memory with Plasticity-based Adaptive Cognitive Training Study. Smith et al. Journal of the American Geriatrics Society, April 2009

13 Immediate and delayed effects of cognitive interventions in healthy elderly: A review of current literature and future directions. Papp, Walsh, & Sny-der. Alz-heimer's & Dementia (2009).

14 Sharp Brains 2012 Market Report
http://www.sharpbrains.com/executive-summary/

Accessed 28.12.2012

15 Lumosity
http://www.lumosity.com/

Accessed 16.12.2012

16 Enhancing visual attention and working memory with a web-based cognitive train-ing program. Hardy, J. L., Drescher, D., Sarkar, K., Kellett, G., & Scanlon, M. Mensa Research Journal, 42(2), 13–20 (2011).

http://static.sl.lumosity.com/pdf/hardy_drescher_sarkar_kellet_scanlon_2011.pdf

Accessed 29.12.2012

17 My Brain Solutions
https://www.mybrainsolutions.com/

Accessed 29.12.2012

18 Nintendo Annual Report 2011
http://www.nintendo.co.jp/ir/pdf/2011/annual1103e.pdf

Accessed 29.12.2012

19 "Nintendo brain-trainer 'no better than pencil and paper'. Adam Sage London: Times Online (January 2009)
http://www.thetimes.co.uk/tto/technology/article1862089.ece

Accessed 29.12.2012

20 Is it worth going to the mind gym? Lawton, Graham. New Scientist (01-12-2008)

21 Cognitive Neuroscience: The Biology of the Mind Gazzaniga, Michael S.; Ivry, Richard B.; Mangun, George R. 2nd edition (2009)

22 Executive functions and their disorders. Elliot R. British Medical Bulletin (2003).
http://bmb.oxfordjournals.org/content/65/1/49.full.pdf+html

Accessed 30.12.2012

23 Game Engine Architecture. Jason Gregory & Jeff Lander. A K Peters (July 10, 2009)

24 Unity 3D
http://unity3d.com/

Accessed 30.12.2012

25 CryEngine Version III
http://mycryengine.com/

Accessed 30.12.2012

26 Unreal Engine
http://www.unrealengine.com/en/features/

Accessed 30.12.2012

27 Ogre 3D
http://www.ogre3d.org/

Accessed 30.12.2012

28 No More Reinventing the Virtual Wheel: Middleware for Use in Computer Games and Interactive Computer Graphics Education. Anderson, Eike. Eurographics 2010 - Education Papers. Pages 33 - 40.
http://www.academia.edu/237502/No_More_Reinventin_g_the_Virtual_Wheel_Middle-
ware_for_Use_in_Computer_Games_and_Interactiv_e_Computer_Graphics_Education

Accessed 30.12.2012

29 Bink Video
http://www.radgametools.com/bnkmain.htm

Accessed 30.12.2012

30 Havoc physics
http://www.havok.com/products/physics

Accessed 30.12.2012

31 Scaleform
http://gameware.autodesk.com/scaleform

Accessed 30.12.2012

32 Use of Unity in serious games and virtual reality
http://blogs.unity3d.com/2012/04/26/unity-authored-projects-win-top-honors-at-serious-games-and-virtual-reality-conferences-2/

Accessed 30.12.2012

33 Diablo II
http://us.blizzard.com/en-us/games/d2/

Accessed 30.12.2012

34 Minecraft
https://minecraft.net/

Accessed 30.12.2012

35 Borderlands 2
http://www.borderlands2.com/us/

Accessed 30.12.2012

36 Quality of experience, Alben (1996)
http://mx1.albenfaris.com/downloads/pdf/quality.pdf

Accessed 31.12.2012

37 The Nielsen-Norman group
http://www.nngroup.com/

User Experience definition by NN group
https://www.nngroup.com/about/user-experience-definition

NN group response times article
http://www.nngroup.com/articles/response-times-3-important-limits/

Accessed 31.12.2012

38 User Experience Whitepaper
http://www.allaboutux.org/files/UX-WhitePaper.pdf

Accessed 31.12.2012

39 Evaluating User Experience in Games: Concepts and Methods. Chapter 3: Pre-cence, Involment and Flow in Digital Games. Regina Bernhaupt.(ed.) (2010)

40 A History of Graphic Design. Philih B. Meggs. (1983)

41 The Elements of Typographic Style. Robert Bringhurst. 3rd edition (2004).

42 The W3C Working Draft for CSS fonts module level 3
http://www.w3.org/TR/css3-fonts/

Accessed 02.01.2013

43 Adobe Myriad Pro
http://www.adobe.com/type/browser/pdfs/MyriadPro.pdf

Accessed 22.09.2012

44 A Practical Guide to Designing for the Web. Mark Boulton (2009)

45 Gestalt Principles Applied in Design
http://sixrevisions.com/web_design/gestalt-principles-applied-in-design/

Accessed 02.01.2013

46 StatCounter Global Online Statistics
http://gs.statcounter.com/#resolution-eu-monthly-200903-201301

Accessed 14.01.2013

47 A List Apart – Responsive Web Design. Ethan Marcotte (May 25, 2010).
http://www.alistapart.com/articles/responsive-web-design/

Accessed 02.01.2013

48 A feature-integration theory of attention. Anne Treisman and Garry Gelade. Cognitive Psychology, Vol. 12, No. 1, pp. 97–136. (1980)

49 Figure-ground perception
http://www.scholarpedia.org/article/Figure-ground_perception

Accessed 17.1.2013

50 The Illusion of Life: Disney Animation. Thomas, Frank; Ollie Johnston Hyperion (1981, reprint 1997).

51 Timing for Animation. Whitaker, Harold; John Halas. Focal Press (2002).

52 Robert Penner's Programming Macromedia Flash MX, chapter 7. Robert Penner. (October 24, 2002)
http://www.robertpenner.com/easing/penner_chapter7_tweening.pdf

Accessed 02.01.2013

53 Tweener easing transitions plot
http://hosted.zeh.com.br/tweener/docs/en-us/misc/transitions.html
‘

Accessed 02.01.2013

54 Greensock animation platform V 12
http://www.greensock.com/v12/

Accessed 03.01.2013

55 LabVIEW product site
http://sine.ni.com/np/app/main/p/docid/nav-104/lang/fi/

Accessed 03.01.2013

56 XML Specification
http://www.w3.org/TR/REC-xml/

Accessed 04.01.2013

57 Ecmascript for XML (E4X) specification
http://www.ecma-international.org/publications/standards/Ecma-357.htm

Accessed 04.01.2013

58 Game Coding Complete, 4th edition. Mike McShaffry & David Graham Course Technology PTR; 4 edition (March 5, 2012)

59 Adobe Actionscript 3 BezierSegment definition
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/fl/motion/BezierSegment.html

Accessed 13.01.2013

60 Continuous Bezier path with AS3
http://www.cartogrammar.com/blog/actionscript-curves-update/

Accessed 13.01.2013

61 Ken Perlin on Perlin Noise
http://www.noisemachine.com/talk1/

Accessed 13.01.2013

62 Video Game Optimization. Eric Preisz. Course Technology PTR. (1st edition March 1, 2010)

63 AS3 Vector vs Array
http://jacksondunstan.com/articles/636

Accessed 16.1.2013

64 Adobe Flash Platform Optimization
http://help.adobe.com/en_US/as3/mobile/flashplatform_optimizing_content.pdf

Accessed 16.1.2013

65 Bitwise operators in AS3, speed and examples
http://lab.polygonal.de/?p=81

Accessed 17.1.2013

66 Amblyopia: Prevalence, Natural History, Functional Effects and Treatment. . Webber, JL; Wood, Joanne. Clinical and Experimental Optometry 88 (6): 365–375 (2005).

67 Amblyopia campaign
http://www.amblyopia.fi/

Accessed 18.1.2013

68 Science changing the world exhibit information
http://www.sciencechangingtheworld.net/index.php?option=com_content&view=article&id=347%3Apay-attention&catid=53%3Alm&Itemid=212&lang=en

Accessed 27.5.2012

69 Adobe Air
http://www.adobe.com/fi/products/air.html

Accessed 19.1.2013