Problem Description

Title: Game-based Learning
Sub-title: Motion-based Educational Game

This project focuses on the development, testing and experimentation with motion-based educational games. The games will use body movement as input to provide a natural user interface and will provide learning through gesture based movement.

The project consists of a state-of the-art study, prototyping and development, testing and experimentation with real users, and analysis of the experimental data.

The project will utilize technology like Kinect or similar.

Assignment given: 13:01.2015
Supervisor: Alf Inge Wang

Abstract

This master's project is dedicated to investigate the potential for educational games with a motion controller based interface. Through professor Alf Inge Wang at NTNU we were assigned the task of brainstorming and developing a concept that would utilize motion controllers, more specifically the Microsoft Kinect, and investigate if it could be useful outside of a traditional game setting. Additionally it was expressed a desire to have emphasis on reusable software that would benefit future projects investigating the same area or using the same technology.

Throughout our research period we have discovered several interesting things. We found that although the Kinect failed to gain any traction on its intended area of use, it is a very impressive piece of affordable technology that is worth investigating further. Educational games have varying popularity among consumers due to the difficulty of finding a formula that combines the values of education and fun in a good way. A teacher will want a game that can be adapted into a curriculum, which the students can learn from, while the students themselves will want a game than is engaging and entertaining.

Using the Kinect SDK and the MonoGame framework we were able to develop a prototype of a basic grammar game, 4-6 players will stand in front of the kinect and be designated a word from a sentence. Through discussion and cooperation the players will need to exchange words between each other until they agree on how the sentence should be. The prototype was tested in an experiment with a group of third graders who also answered a questionnaire for our data gathering.

We conclude in this report that there is definitely potential in the idea of using motion controllers for educational purposes that is worth exploring further. The children who tried our prototype expressed great enthusiasm over interactive play in a subject that is normally considered dull for pupils of this age group. With this report and the software we have produced we hope that we have accomplished to make fundament for future endeavours in the topic of using motion controllers for educational games.

Sammendrag

Denne masteroppgaven er dedikert til å utforske potensialet for læringsspill som er styrt via bevegelsessensorer. Gjennom professor Alf Inge Wang hos NTNU ble vi gitt i oppgave å tenke ut og utvikle et konsept som skal bruke bevegelsessensorer, mer spesifikt Microsoft Kinect, til å finne ut om den kan være nyttig i bruksområder utenfor et mer tradisjonelt dataspill. I tillegg ble det utrykt et ønske om å gjøre programvaren gjenbrukbar til nytte for framtidige prosjekter som undersøker samme felt eller teknologi.

Gjennom vår forskningsperiode har vi oppdaget flere interessante ting. Vi har funnet at selv om Kinect sensoren ikke ble en suksess på sitt tiltenkte bruksområde så er det like fullt et imponerede og rimelig stykke teknologi som er verdt å undersøke nærmere. Læringsspill har hatt varierende popularitet blant forbrukere grunnet vanskeligheten med å finne en god balanse i kombinasjonen læring og spill. En lærer vil ha et spill som kan tilpasses inn i pensum, og som elevene kan lære av, mens elevene på en annen side vil være mer interesserte i et spill som er engasjerende og morsomt.

Ved å ta i bruk Kinect SDK og MonoGame rammeverket var vi i stand til å utvikle en prototype for et grunnleggende grammatikkspill, 4-6 spillere vil stå foran Kinect sensoren og hver spiller vil bli tildelt et ord fra en setning. Gjennom diskusjon og samarbeid må spillerne utveksle ord seg imellom til de blir enige om hvordan setningen skal utformes. Prototypen ble testet i et eksperiment med en gruppe av tredjeklassinger som også svarte på et spørreskjema i forbindelse med vår datainnsamling.

Vi konkluderer i denne rapporten med at det definitivt finnes et potensiale i ideen med å bruke bevegelsessensorer for læringsspill som er verdt å utforske videre. Barna som prøvde vår prototype uttrykte stor entusiasme over å få delta interaktivt i ett tema som vanligvis blir betraktet som kjedelig for elever på denne aldersgruppen. Fra denne rapporten og programvaren som vi har laget så håper vi å ha oppnådd et fundament for framtidige forsøk basert på å ta i bruk bevegelsessensorer for læringsspill.

Preface

This Master Thesis is the presentation of the work devoted by Per Olav Flaten and Henrik Reitan during the fall semester 2015 and spring semester 2016. This project will conclude our master's degree in Informatics with specialization in Game Technology at the Norwegian University of Science and Technology (NTNU) with the Department of Computer and Information Science (IDI) under the supervision of Alf Inge Wang.

We would like to express our gratitude to Martin Almvik and Mikael Rino Solstad for helping us test our prototype and its functionality, we would like to thank Linda Skistad and the children at the Eberg SFO for helping us with our experiment, and of course we want to thank Alf Inge Wang for providing us with supervision, assistance, and feedback during the course of the thesis.

Trondheim, June 8th, 2016

Per Olav Flaten Henrik Reitan

Contents

List of Figures

List of Tables

Part I

Introduction

An introduction to our thesis, covers thesis structure, our methodology, and background information about the project.

1 Thesis Information

Here we will describe some general information about the thesis, starting with the thesis structure. Afterwards we will go into more details on our project's context, before we have a bit on our personal motivation, followed by the goals we hope to achieve.

1.1 Context

This project covers the work done in conjunction with our master's thesis as part of our master's programme in informatics, with specialisation in game technology, at the Norwegian University of Science and Technology.

The project partially builds upon the findings in the projects presented in Section 7 Related Work, but also stems from an interesting in having more research that can be used in future projects. The field of motion controlled educational games is practically non-existent, this leaves a lot of questions unanswered.

1.2 Personal Motivation

Both of us have a large interest in games and technology in general, being able to apply this into the largest project we will undertake during our education is part of the reason we chose this project in particular. We both believe that there is potential for applying game technology into the regular education curriculum, especially for the earlier parts of primary school.

Many of the subjects that are generally considered as dull could benefit greatly from giving the students more variation in how they can learn about the different topics. If we can figure out a way to also give them a motivational boost, we would be able to show that there are several reasons to start considering the implementation of more game technology in school.

1.3 Goals

The main goal of this project is to research the basis for using motion control in an educational game, and if the addition of motion control has an effect on

how the players interact with the game. Our focus will be on whether using motion control can improve the degree of intrinsic motivation the player's experience while playing the game, as well as to map out some reactions in regard to motion controllers. We want to know if there is any reason to spend resources on developing this type of games aimed at education, and if they can have a positive effect on educational gain.

In addition we will attempt to develop a framework that is suited for future work with similar projects. This is to reduce the amount of time that would be required for implementing the basic functionality of motion controllers, and instead allow more focus on creating the games themselves. Doing this could allow future projects to aim towards an even larger scope.

1.4 Thesis Structure

The thesis is structured according to our process in the project. Starting with an introduction to the project, then our findings from the literature study in research, followed by our own contribution to the subject, and lastly a summary of our project as well as some comments on future work.

1.4.1 Introduction

Covers the context of the project, our motivation for working on this project, and the goals this project hopes to achieve.

1.4.2 Research

The findings from our literature study covering the subjects Motion Capture Technology, Game Development Technology, Serious Games, Designing an Engaging Game, and Related Work.

1.4.3 Own Contribution

Presents our work on the prototype, our framework, and our experiment.

1.4.4 Conclusion

A summary of the project and our recommendations for future work.

1.4.5 Appendices

Contains our subsidiary documents that did not fit into our regular sections due to size or relevance.

2 Methodology

In this chapter we will present the methods we will use during this project. Starting with our goals and research questions, and how we reached them, using the **G**oal **Q**uestion **M**etric (GQM) [13] approach. We then go through our process and the different approaches we have used to gather data and any potential practical issues that may arise.

2.1 Goals and Research Questions

This section details the questions we want to answer with this project, and the method of which we have used to refine them into a more quantifiable form. The questions aim to identify the usefulness of educational games, and the impact of using less traditional input methods, as well as the social aspect. We have used the GQMMetric to reach these questions.

2.1.1 Goal Question Metric

The GQM approach is focused on defining useful questions with measurable metrics. To achieve this GQM works in a top-down fashion resulting in a measurement model with three levels. This model, shown in Figure 2.1, is defined with a goal as the conceptual level, the question being the operational level, and metric on the quantitative level.

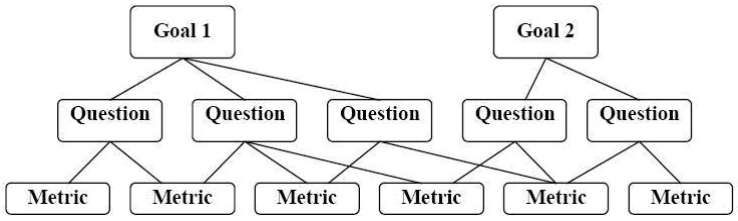


Figure 2.1: Goal Question Metric.

This hierarchical structure starting with the goal specifying the purpose/issue to be measured. The goal is then refined into several sub questions that are easier to measure individually compared to the goal. Each question is then further refined into metrics, which are either objective or somewhat subjective in nature, ie. customer satisfaction is hard to measure completely objectively. Table 2.1 shows an example of the GQM in practice.

Goal	
Purpose	Improve
Issue	the timeliness of
Object (Process)	change request processing
Viewpoint	from the manager's viewpoint
Question	What is the current change request processing speed?
Metric	Average cycle time Standard deviation % case of the upper limit
Question	Is the performance of the process improving?
Metric	$\frac{Current\ average\ cycle\ time}{Baseline\ average\ cycle\ time}100$ Subjective rating of managers satisfaction

Table 2.1: Goal Question Metric

GQM has shown to be practical for creating research goals relating to software development, if the goal is to "make the program better", this is very hard without identifying some more narrowed down questions on the operational level, and then deciding on the quantifiable metrics.

2.1.2 Research Goals

Table 2.2 through 2.4 shows our overarching goals as well as the research questions they have been broken down into, as well as the metrics for the questions. The main purpose of this report is to provide satisfactory answers for these questions.

Goal 1 The study seeks to explore if motion-controlled application can have a positive effect on educational gain.	
RQ 1.1:	Is there a benefit for using motion-control as input compared to traditional control methods? (motivation, engagement, ease of use)
Metric	Questionnaires Observation Literature study
RQ 1.2:	Does a cooperative game increase the amount of discussion/cooperation between people in a classroom compared to traditional teaching?
Metric	Questionnaires Observation Literature study

Table 2.2: Research Goal 1

Goal 2 The study seeks to explore the possibilities and maturity of motion-capture technology.	
RQ 2.1:	How easy is it to start developing a motion-capture application? (SDK's, API's, etc)
Metric	Literature study
RQ 2.2:	How mature is the current technology, is the hardware available for the average developer good enough?
Metric	Literature study Experimentation with the Microsoft Kinect v2
RQ 2.3:	Is it easy for a new user to adapt to using motion controllers as opposed to more traditional input devices?
Metric	Observation Questionnaire

Table 2.3: Research Goal 2

Goal 3 The study seeks to map the most important factors for creating an engaging and fun game.	
RQ 3.1:	How do we keep the player(s) interested in the game over time?
Metric	Questionnaires Literature study
RQ 3.2:	What are important factors to consider when designing a game?
Metric	Questionnaires Literature study

Table 2.4: Research Goal 3

2.1.3 Summary

Many of our research questions require gathering general information about motion capture and educational games, these will mainly be answered by a literature study. While the more practical oriented ones, will require experimentation with our own prototype.

2.2 Process

This section will detail the metrics used to answer our research questions, as well as our reasoning behind them. We will also look at some potential practical issues and how best to avoid them.

2.2.1 Literature Study

A literature study is one of the most used methods for any research, reviewing literature relevant to the subject at hand. The purpose of this is to familiarize oneself with the research topic, and thus increase the credibility of the research. Our sources for this project will be published articles, documentation on the web, and other master's theses. The topics of our literature study are:

Motion Capture Technology will cover basic information about the workings and usage of the technology. We will also cover the Microsoft Kinect sensor in a bit more detail, as this is the one we had available during our project.

Engines and Frameworks showcases the game engines and frameworks most compatible for use with C#. We have focused on this since we will be using the Microsoft Kinect, whose official SDK is .NET based and thus making C# the preferred language for us.

Serious Games defines the term as it is used in our project, as well as listing some of the more common subgenres, and examples.

Designing an Engaging Game is where we go into more general game design theory, touching upon subjects such as flow. The principles covered here play a large role when designing a game, as an oversight will often harm the enjoyment of the game.

Related Work covers two master's theses which also worked with motion technology.

2.2.2 Prototype Development

Our research in this project is based upon gathering information about how well a motion controlled educational game is received, and if it can have a positive impact on learning when compared to more traditional educational methods. To collect data on this we need to develop a prototype that fits these criteria. Considering that this type of technology is relatively new, we are also interested in the maturity of the development tools that are available, especially compared to how they behaved only a few years back as shown from the master's theses in Section 7 Related Work.

As opposed to just creating a simple game prototype for use in this project, we also want to develop a more general usage framework that can be used in future projects. This also helps ensure that our prototype will be as flexible as possible, keeping the intricacies of the motion tracking away from the actual game implementation, so that it can be used to easier create other games in the future.

2.2.3 Questionnaire

We have decided to use a questionnaire to help us gather data on the users of our prototype system. The data we are interested in is simple background information such as age, gaming habits, and experience with similar technology to see if this impacts their perception and educational gain of our game. In addition to this, we want to gather opinions on whether they feel that using games like this could be useful in education. Since our game will be aimed at children, the format of the questionnaire will be mostly yes/no answers with a few questions that have a more open answer. This is to keep the questionnaire easy to answer, so that we can ensure the information we get is as accurate as possible. Experience has shown that scaling questions often end up with results that are either very positive, very negative, or simply just neutral because it is difficult to quantify impressions such as fun on a scale from one to ten. In addition to the questionnaire we will be talking to the players after they are done, to gauge the overall impressions, so we can see if there is anything they want to give us feedback on that goes outside the questionnaire.

2.2.4 Observation

Observing how people play the game can give us information that is difficult to come by using more structured data gathering methods, like interviews. It is particularly useful to discern whether our control schemes and graphical interfaces are intuitive enough. In this project we are also interested in seeing whether solving tasks in a game format makes the players more inclined to discuss solutions compared to when working in a traditional classroom setting, this would however require some assistance from a teacher or similar who works with the kids on a regular basis and knows how they normally behave to compare. Data gathering with observation can be difficult, since things like taking notes while someone is playing could impact their behaviour. As such we will try to write down impressions between sessions, and then compare with the questionnaire result afterwards.

2.2.5 Practical Issues

Considering the nature of this project, there are several possible issues that can occur. This is a given in any software development project when dealing

with unknown technology, but we also have some potential problems regarding the execution of the experiment. One such problem can be finding a suitable location to perform the experiment, we aim to contact a school or an after school programme, so we can gather a useful number of participants of the right age. We might also find some problems with the body tracking when it is applied to the smaller stature of the children compared to how it is calibrated. In regards to the questionnaire there is a possibility that the answers will be skewed towards being positive due to the children not wanting to disappoint us, this is a main reason to keep it anonymous and combine the results with data from observation.

2.2.6 Summary

Of the methods presented here we will attempt to find answers to our research goals with the literature study, and these findings will be tried and tested during the development of our prototype, and the experiment. We believe this will give us enough data to come to a conclusion for our research questions.

Part II

Research

A short examination of the underlying motion capture technology, games, and using them as an educational tool.

3 Motion Capture Technology

Since our project entails using motion capture as the primary input method, this section will give a general background about the subject, as well as more in depth information about the Microsoft Kinect sensor that we use in our prototype. Motion capture, or motion tracking, is defined as the process of recording the movement of people or objects. The most known usage of this is the entertainment industry, where an actor's movements are transferred to a digitally animated avatar. It also has applications for the military, in sports, and even in the medical fields. In these fields the resulting data and animations are used for more in-depth analysis of movement than what is realistically possible to do in real time. Most equipment used for motion capture is based on optical systems, where cameras detect markers placed on the persons or objects being tracked. These markers are either passive or active, with active markers sending out light on their own, while passive markers are either brightly colored or reflecting signals from the camera device. With the development of more advanced computer vision techniques we also have markerless optical tracking, such as the Microsoft Kinect, which comprises a normal color camera, and infrared depth tracking. The markerless versions are more aimed at the consumer market so far, with the tracking being less precise than the marker based systems, but they also cost significantly less.

3.1 Microsoft Kinect

The Kinect, shown in Figure 3.1 is Microsoft's version of motion tracking for the home consoles, an answer to Nintendo's Wii Remote, and Sony's PlayStation Move. Initially sold for the Xbox 360, there are now improved versions for the Xbox One as well as PC. Microsoft have also released a software development kit that allows developers to write applications that utilize the Kinect. The initial version for the Xbox 360 was fairly well received, but not a huge success much due to the fact that the cost was about three quarters of an actual Xbox, as well as somewhat intimidating space requirements. The second version, which came as an integrated part of the Xbox One, gathered a somewhat colder reception. This was much due to the fact that people assumed this was a large cost increase compared to a version of the Xbox without the kinect, as well as some privacy concerns since it was said that the kinect would always be "on" to listen for commands to start the Xbox via voice activation. It is a very capable piece of hardware,

despite its low cost, and have been used for many research projects related to computer vision and motion capture.

Figure 3.1: Microsoft Kinect v1 Sensor

3.2 Hardware Comparison

On the hardware side the two Kinect versions are similar, they both have a color camera similar to a webcam, an infrared camera that functions as a depth sensor, and a microphone array that captures positional audio. Table 3.1 shows the different specifications of the kinect versions, both versions of the Kinect have an effective distance of depth tracking at approximately 0.5 to 4.5 meters. The v1 has a smaller field of view than the v2, but compensates with a tilt motor that can adjust the focus of the sensors. The largest difference between the two is the jump from USB 2.0 to USB 3.0 allowing a lot more data to be transferred, this made it possible to equip the v2 with a 1080p resolution camera while keeping the framerate at 30. Additionally the increased data rate has increased the number of fully trackable skeletons to 6, up from 2 on the v1, as well as 26 instead of 20 joints per skeleton. The v2 can also display a video stream from the infrared camera, which the v1 is unable to.

	Kinect v1	Kinect v2
Color Camera	640x480 @30 fps	1920x1080 @30 fps
Depth Camera	320x240	512x424
Infrared Camera	None	512x424
Max Depth Distance	~4.5 m	~4.5 m
Min Depth Distance	40 cm	50 cm
Horizontal Field of View	57 degrees	70 degrees
Vertical Field of View	43 degrees	60 degrees
Tilt Motor	Yes	No
Skeleton Joints Defined	20 joints	26
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0

Table 3.1: Kinect Hardware Specification Comparison

3.3 Software

Table 3.2 compares the different available development kits with support for the Kinect. The easiest to use and most comprehensive SDK for the Kinect is the official kit from Microsoft themselves. This SDK includes everything needed to start creating applications for the Kinect written in any .NET compatible language, such as C#, Visual Basic, or C++. The SDK includes several examples and test applications that can be used to troubleshoot the Kinect itself or help in development of new applications. The main drawback of the official SDK is that it is proprietary, and no way to look at the actual source code for the drivers, it also only works on Windows. To use the Kinect on other operating system it requires the use of third party drivers. The most popular third party driver for the Kinect is libfreenect from the OpenKinect group, they provide drivers for both the v1 and v2. Using such drivers however, does not provide the full functionality of the SDK, requiring pairing with other libraries or frameworks for computer vision to provide skeleton tracking and similar features that are built into Microsoft's SDK.

	Kinect SDK 1.8	**Kinect SDK 2.0**	**OpenKinect**
Author	Microsoft	Microsoft	Open Source
Kinect v1.0	Supported	Not supported	Supported
Kinect v2.0	Not supported	Supported	Supported
License	Proprietary	Proprietary	Apache v2 & GPL v2
Language	.NET compatible, Visual C++	.NET compatible, Visual C++	C/C++, C#, Python, Ruby, Actionscript, Java
OS	Windows	Windows	Windows, OS X, Linux

Table 3.2: Kinect SDK Comparison

Additionally there is a framework called OpenNI (Open Natural Interaction) [7] which has been developed to act as an abstraction layer between multiple supported hardware devices, including the Kinect. It has not been included in the comparison because it is believed to not be very relevant anymore. The original OpenNI project was acquired by Apple and shut down in November 2013, since then a new open-source project called OpenNI 2 has surfaced, but any concrete information on the capabilities and features of said project is hard to come by.

3.4 Summary

Since we have a Kinect v2 Sensor for use with our experiments, we want our prototype to build upon its strong features, mainly the advanced skeleton tracking that handles up to six people at once together with the large tracking area. This is very well suited for a multiplayer game, especially in a classroom setting, considering that the only real downside to tracking this many people at once is the requirement for a large room.

4 Game Development Technology

This section is devoted to finding possible candidates for frameworks or engines that meets our requirements. For a simple mock prototype we do not require a high level of functionality of features, however the bare essentials should be covered, which is:

- An implementation of the Game Loop.
- Handling of Graphics.
- Handling of Audio.

Additionally the framework or engine should be compatible with the Kinect SDK.

This section will start with explaining what we mean by the game loop, before we review possible candidates for engine/framework for our prototype.

4.1 The Game Loop

Most computer/video games are made using some variant of the Game Loop [3], which is represented in Figure 4.1. The Game Loop is responsible for handling the game initialization, update and draw mechanics, where update and draw usually runs once for every frame rendered in an endless loop until termination of the game has been requested.

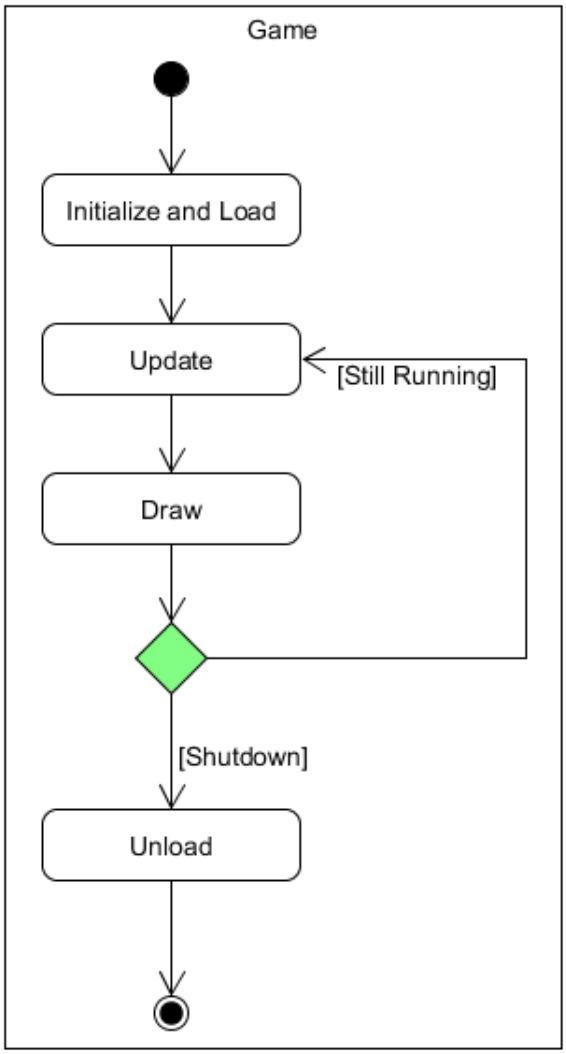


Figure 4.1: The Game Loop

4.1.1 Initialize and Load

The "Initialize and Load" step is where data, graphics, audio, etc. is instantiated and loaded, it is usually executed only once after the game is started

and before the "Update" and "Draw" for the first frame is invoked. If the software supports it, graphics may be loaded onto the GPU.

4.1.2 Update

The "Update" step is invoked before "Draw" of any frame in the game. This is where data is updated and prepared in the event of any changes so that it is ready to be rendered, one example of change being the result of I/O input like mouse and keyboard.

4.1.3 Draw

The "Draw" step is invoked after "Update" of any frame in the game. This is where the actual rendering takes place.

4.1.4 Unload

The "Unload" step is where all game data is unloaded before the game shuts down.

4.2 Engines and Frameworks

One of the first question we would like to answer is "What game engine or framework should we choose for developing a game?". The list of candidates are seemingly endless, to narrow it down we will only look at popular choices in relation with the C# programming language, as C# is the most relevant when developing with the Kinect in mind. C/C++ is also a relevant alternative, however the project members does not have much experience with C/C++, whereas they are quite familiar with C#.

The second question is "What do we need? An engine or a framework?". Keeping in mind that a framework is usually built to solve a specific issue, while an engine may be densely packed with features that go beyond the scope of what we are trying to accomplish.

The differences between a game framework and an engine is not clearly defined, however it is generally understood that a framework is a set of libraries packed together to solve a specific issue, while a game engine is a set of frameworks bundled together, usually with an editor. An engine comes with many tools and eases the burdens of programming on the developer, which makes development much more accessible to those who does not strictly come from a programming background. However a game built on frameworks instead of an engine will usually be more modifiable and easier to upgrade to newer technology as it does not have to follow the rigid "recipe" set from an engine and its editor. For C#, Unity is the most popular game engine, and Microsoft XNA is the most popular framework. All engines and frameworks listed in Section 4.2.1-4.2.6 uses some variant of the "Game Loop", which is described in Section 4.1 The Game Loop.

4.2.1 Unity

Figure 4.2: Unity

Unity [8], Figure 4.2.1, is a commercial game engine which is widely used, and is one of the most popular engines currently on the market. The latest edition supports scripting in the languages C# and JavaScript. Two of the major reasons it has become so established amongst game developers is that it is easy to use, and have wide platform support, spanning from PC, Consoles, Mobile and even platforms that are not common like Smart TV's. It has been used to make games such as Ori and the Blind Forest, Endless Legend, Dungeon of the Endless, Satellite Reign, Hearthstone, Cities Skylines, and many more. It uses a custom made edition of Mono to achieve support for all the platforms. This custom edition is based off of the rules and conventions of .NET 2.0, with some added modifications that allows

developer to script with certain .NET 3.5 features on code files within the Unity environment. This however, means that it is incompatible with the official Kinect SDK which requires .NET 4.0 at minimum. To circumvent this issue Microsoft has made a custom Unity add-on that allows developers to use the Kinect in Unity.

4.2.2 Xenko

Figure 4.3: Xenko

Xenko [10], Figure 4.3, is a open-source game engine for development of games in a C# environment, meaning that it provides its own editor, however it generates a C# project following the standards established by Microsoft, meaning the Xenko editor is optional for those who prefer using an IDE instead. It supports Windows, Android, iOS and Windows Phone, there is also some functional experimental support for Linux, OS X and Consoles. Xenko is fairly new, so unfortunately there are not many examples of games made with it, however it is fully compatible with the official Kinect SDK.

4.2.3 Microsoft XNA

Figure 4.4: XNA

XNA [11], Figure 4.4, is arguably the best known framework for game development in the C# programming language. Developed by Microsoft it supports platforms such as Windows, Xbox 360 and Windows Phone. It has been used in development of titles such as Bastion, Apatheon, Rogue Legacy, Fez, Stardew Valley and more. It is also fully compatible with the official Kinect SDK. Unfortunately it has been discontinued by Microsoft which makes it an unideal candidate for development with reusability in mind.

4.2.4 MonoGame

Figure 4.5: MonoGame

MonoGame [5], Figure 4.5, is an open source game development framework maintained by MonoProject (known for Mono, MonoDevelop, etc.), as a derivative of XNA. Originally started out as a wrapper library to port

games developed in the XNA framework to other platforms, it eventually evolved into a stand-alone framework. Of all the frameworks for C# game development, MonoGame currently has the biggest community and highest development rate. It supports platforms such as Windows, Linux, OS X, iOS, Android, Xbox One, PS4. Bastion, Fez and Stardew Valley are examples of games that started development in XNA, but has been ported to other platforms using MonoGame. Since MonoGame is just a framework that can be implemented into a standard C# project it is fully compatible with the official Kinect SDK.

4.2.5 FNA

Figure 4.6: FNA

FNA [2], Figure 4.6, is an open source game development framework made and maintained by Ethan Lee, and just like MonoGame it is also a derivative of XNA. It is a fairly young and immature project so information about it is sparse. It supports PC platforms exclusively, namely Windows, Linux and OS X, and just like MonoGame it is fully compatible with the official Kinect SDK for the same reasons.

4.2.6 CocosSharp

Figure 4.7: CocosSharp

CocosSharp [1], Figure 4.7, is an open source game development framework maintained by Xamarin(known for the Xamarin Platform, Xamarin Forms, Xamarin Studio, etc.). It is built on top of MonoGame with C# implementations of the API of the already well established open source game development frameworks of Cocos2D and Cocos3D. Like FNA it is a relatively young project so there is not much concrete information on the framework. Since it is built on top of MonoGame it can run any platform that MonoGame supports, and for the same reason it is also compatible with the official Kinect SDK.

4.3 Summary

It is evident that there is no lack of alternatives when it comes to doing game development in C# for the Kinect, all the candidates we have found are solid and feature complete. Some however, like Xenko, FNA, and CocosSharp have the disadvantage of being introduced relatively recently to the market, which makes it hard to favor them in the interest of building a prototype that can be reused, modified, or expanded upon for later research projects, as any of these projects may or may not be relevant in the future. Since XNA is discontinued it is clearly not a candidate, but it was important to mention either way because it has been so important for C# game development, and many of the potential candidates are derivatives of XNA. Unity is usually the crowd favorite for game development in C#, however the lack of support for the official Kinect SDK is worrying.

Xenko or MonoGame seem to be the best and safest choices in the interest of building reusable software. Xenko is still a fairly new engine, and the

project team does not have much experience with it, but it appears to be quite capable. MonoGame is probably the strongest contender because of its popularity and devoted developers.

Conclusively it also is worth noting that choosing any XNA or any derivative will provide the benefit of easy portability, since they are all designed to be easy to convert from one to another.

5 Serious Games

As we are developing an educational game, which is a subcategory of Serious Games, this section will give a definition for this term, and present some general history and examples. There is no formal definition for serious games, but most attempts converge towards "A game that has a purpose other than entertainment". The term serious game predates computer games, but the principles remain the same, and computer technology have greatly increased the possibilities.

> "Reduced to its formal essence, a game is an activity among two or more independent decision-makers seeking to achieve their objectives in some limiting context. A more conventional definition would say that a game is a context with rules among adversaries trying to win objectives. We are concerned with serious games in the sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement." Serious Games, Clark Abt, 1970

Most serious games are meant to be entertaining as this is the main motivation behind calling them games in the first place. These games are supposed to assist in education, training, or other purposes such as marketing. In the marketing case the games are in a large degree just normal games with an elevated degree of product placement, while games that focus on education and training often see substantial differences when compared to regular games only meant for the purpose of entertaining. While a lot of the serious games developed over the years have been focused on regular education such as language and math, these have not proven very successful. The most successful versions of serious games are those focused around situations and skills that are difficult to exercise due to being either too dangerous, time consuming, or expensive.

5.1 Common Types of Serious Games

This chapter will present some of the most popular types of serious games, and try to explain the reasoning behind why they are created and used.

5.1.1 Edutainment

As the name suggests, edutainment is a term used for entertainment that is designed to also educate. Most of these focus primarily on the entertainment part, with the educative components are a small part of the whole. But there are cases which are almost purely educational with little to no focus on the entertainment value. Video games intended for edutainment started out in the 1980's when computers began being available for classroom use, and games including Oregon Trail and the Munchers series gained popularity. With computers becoming more common over the 1990's, the number of edutainment games exploded. However, the interest for these games did not last, as most of them were generally bad.

> "Most existing edutainment products combine the entertainment value of a bad lecture with the educational value of a bad game"
> Squire & Jenkins, 2003, p.8

Much of the reason for the failing of these games was that they usually were just a variation on drilling exercises, repeating the same task over and over. Games that went in a different direction, by promoting creative thinking and unusual problems became popular, one such game was The Incredible Machine (TIM). This type of game have not found their way into classrooms in the same degree as the ones from the early 80', much due to the fact that they do not fit any particular curriculum. Judging by the trends over the years, it is most likely that in the future we will not have games dedicated to teaching specifics subjects in the degree that we have seen. Instead we will probably have more game technology complimenting the traditional teaching, such as the general-purpose quiz game Kahoot! as shown in Figure 5.1.

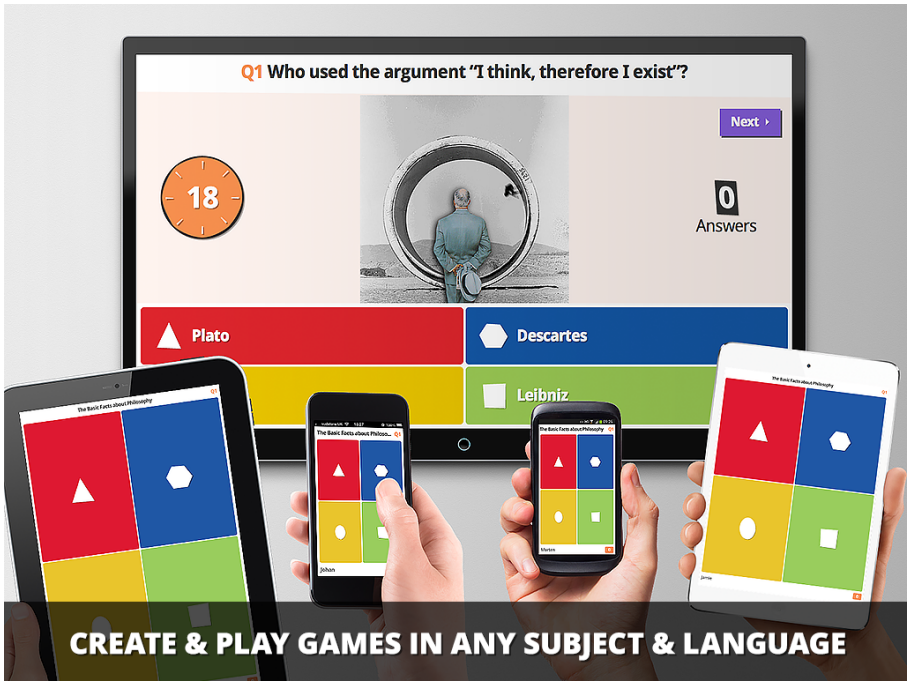


Figure 5.1: Kahoot!

5.1.2 Games for Health

Health focused serious game is one of the fastest growing sub genres, with a wide range of applications. Many of these games utilize the motivational factor of games to lessen the burden of more monotonous tasks for instance in physical therapy. In some cases the act of gaming itself has proven beneficial in terms of health, many stroke patients have shown increased recovery rate from the fine hand-eye coordination required in many games. There have also been shown a correlation between gaming experience and performance in laparoscopic surgery. We also have games like wii fit, that physically tracks the player and help guide workouts, and similar games for rehabilitation. Games have even been used to treat post traumatic stress disorder for war veterans [15].

5.1.3 Simulation

Simulation games is a genre by itself, but combined with the serious games term we have a smaller subset consisting of more realistic simulators. This

is a broad category including management, life, sports, medical, military, vehicular, social, and more. While most of these are very much games, such as SimCity, there are cases that are more or less equal to their real life counterpart, like very advanced driving simulators with specialized input devices. Simulators have gotten a lot of attention over the years and inspired many impressive setups, not only by professional companies, but also enthusiasts building entire plane cockpits in their garage. One of the largest actors, not only in the simulation business, but also in the serious games business as a whole, is the US military. They have been responsible for creating many different simulations of all kinds of situations, meeting different cultures, war scenarios, vehicular training, environment adaption, and more. In addition to this they have also commissioned two games, America's Army (2002) and Full Spectrum Warrior (2004) intended for the general public to aid with recruitment and spreading knowledge about the military.

5.2 Examples

5.2.1 The Oregon Trail

The Oregon Trail, Figure 5.2, is an "edutainment" title made in the 70's for the purpose of teaching students about american history, more specifically the 19th century life of the pioneers travelling from Independence to Oregon. Originally the game started out in the classroom as a concept of combining learning with an extra element of fun to grab the attention of the students. The players found it entertaining and it became popular and successful, to the level where people worldwide still talk about it 30-40 years later.

Figure 5.2: The Oregon Trail

5.2.2 The Incredible Machine (TIM)

TIM, Figure 5.3, is an "edutainment" puzzle game that puts the player in the position of having to solve contraptions similar to a Rube Goldberg machine. TIM challenged players to think outside the box to solve abstract problems and became quite popular, however it was critiqued by teachers for having a game format that was difficult to adapt to a curriculum.

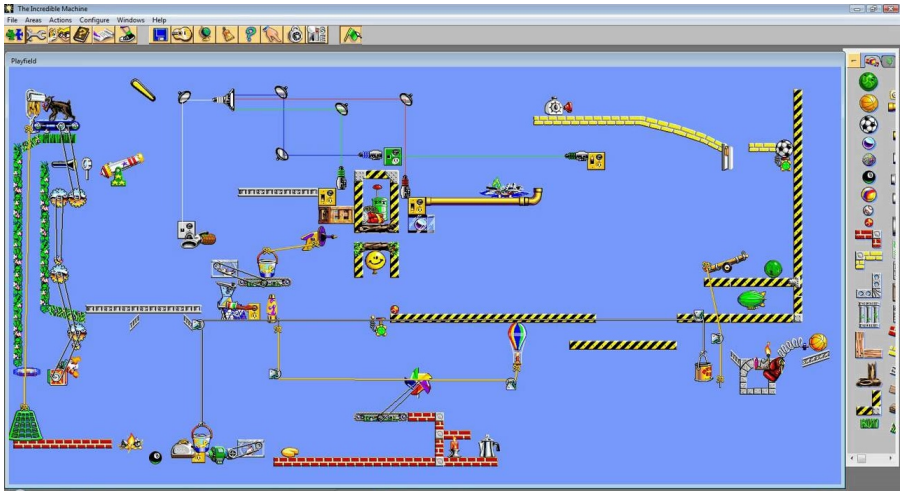


Figure 5.3: The Incredible Machine

5.2.3 America's Army

America's Army, Figure 5.4, is a first person tactical shooter created to provide a virtual experience of being a soldier. The game was published by the US Army in 2002, as a free download. The game has since seen numerous updates and several re-releases on new engines with the latest in 2013. Despite being what could be called a PR stunt, the game was well received with great review scores and several awards. Much of this reception is credited to the realistic portrayal of soldiering, including basic training, instead just the gamified fighting that most similar games portray. It also includes an optional medical training, which has been credited to saving lives in at least two situations, where players used the knowledge from the game in actual emergencies.

Figure 5.4: America's Army

5.2.4 X-Plane

X-Plane, Figure 5.5, is a "Simulation" title, more specifically a flight simulator that aims to deliver a realistic flying experience. It is FAA approved and is used for pilot training [9], advertising such features as Artificially Controlled Air Traffic Controller, an extensive amount of airports from real life, a wide range of aircrafts available, training on autopilot and other things.

Figure 5.5: X-Plane

5.2.5 Wii Fit

Wii Fit, Figure 5.6, is a "Games for Health" title for the Nintendo Wii that by an extra peripheral allowed the players to do exercises and measure BMI amongst other things in their living room. The effectiveness of the Wii Fit has been debated, as the intensity in the exercises is lackluster and will most likely not yield much benefits unless the player was in poor shape to begin with. It has however been reported with successful use in the field of physiotherapy rehabilitation.

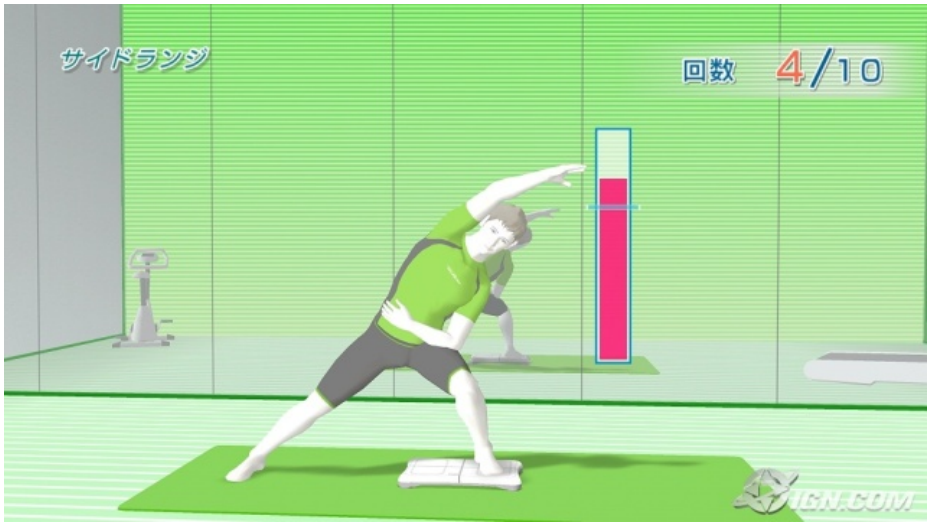


Figure 5.6: Wii Fit

5.3 Summary

What we can gather from this is that the most successful serious games, are indeed still games. All of the most memorable of these provide an experience that, in terms of entertainment, rivals games that are purely aimed for this purpose. Most serious games end up with so much focus being directed at the serious part that there is no room left for any fun. As shown by the trends in the early 90's, the odds that anyone will prefer a mainly educational game over one aimed at entertainment is fairly low. Considering this, we believe that instead of trying to compete with other games, we should compete with the traditional blackboard teaching methods and introduce an alternative to this instead.

6 Designing an Engaging Game

In terms of technology game development have changed drastically over the years, and will continue to do so. However, when it comes to the core aspects of game development, there has not been any real change. The reason people still play games is because they still find them fun, and the reason for this has not changed on a fundamental level. This means that the article by Malone [16] about what makes games intrinsically motivating, as well as the study by Sweetser and Wyeth [18] on flow elements in games, based upon Mihaly Csikszentmihalyis studies on flow, are still relevant for this project. All of these studies have researched the psychology behind things we find interesting, and isolated it to apply to game elements such as the user interface and sounds. Although Malone lists five things that are important for a game to be engaging, and Sweetser and Wyeth identifies eight different flow elements, they can be broken down to three main categories. These categories are challenge, fantasy, and curiosity.

6.1 Intrinsic and Extrinsic Motivation

Intrinsic motivation comes from a sense of accomplishment and enjoyment, something you do just because you enjoy doing it. Whereas extrinsic motivation involves some external factor motivating you to do something, either to gain a reward or avoid some negative consequence. Hobbies are mostly intrinsically motivated as this is something people do for their own enjoyment, while work and most day to day tasks such as cleaning are extrinsically motivated.

There is also a special phenomenon related to this called the overjustification effect. This occurs when you provide someone that is already intrinsically motivated for a task with an additional extrinsic motivation. The result of this is that the degree of intrinsic motivation diminishes because the activity now feels less fulfilling when the external reward is missing. Games are for most people mostly intrinsically motivated, the enjoyment comes from the feelings of accomplishment that often accompanies the effect of flow, or being in the zone. Achieving this is reliant on several factors within the three categories mentioned above.

6.2 Challenge

Creating a suitable level of difficulty is possibly the hardest part about designing a game, and often also the most important part. Suitable in this context does not entail a single level of difficulty, but scaling it in a way that keeps it on a level that changes according to the player's own skill development. If a game is too easy the player will get bored and lose interest, while if it is too difficult frustration will occur and the player might give up. Most games today become more challenging as you progress, unlocking more complex mechanics or harder enemies, in addition to often giving a choice to select a preferred difficulty level before starting the game. Competitive multiplayer games handle this by matching players of similar experience and skill together to even the playing field, often by also giving players a transparent ranking system to track their own and other people's progress.

First of all, to achieve a challenge you must present a goal that needs to be reached. This goal then needs to be clearly defined, so that the player knows what to do, and there needs to be clear feedback on the progress towards said goal. A fact that is especially important in serious/educational games is that the goal itself is not to learn, but learning is a way to achieve the goal. The purpose of the game, and the goals in the game itself, are usually not the same. For instance in a driving simulator a good goal would be getting safely to a destination, not "learning to drive safely" which is the skill required to reach the goal.

To avoid players dismissing the game too soon, it should require little to no knowledge before starting the game, it should be possible to jump straight into the game and get started without reading the manual. In most games this is achieved by having one or more tutorial levels in the start of the game to familiarize the player with the basic mechanics, controls, and graphics interface. This makes the initial learning necessities an integrated part of the game, which avoids the need to look for external sources of information. Introduction of new game mechanics and other rule changes should be introduced gradually so the player is not overwhelmed at the beginning, while also making the game more challenging as it progresses. Doing this ensures that the challenge ramps up as the player's skill increases.

Lastly, the game controls are a very important. The player needs to feel that he is in control of his character and/or units, and be able to issue commands effectively. To attain a large enough level of control for the

player it is important to consider all aspects of the game interface. Input devices need to be responsive and precise. The graphics interface, including sound, need to show feedback immediately when required as a response to commands from the player. If the interfaces are too clunky to use it will not matter how intriguing the story is, or how good the mechanics are, the player will stop playing the game because of undue frustration. It is also important that the player feels that he is discovering and planning strategies on his own, not just uncovering what the designers put in place. If this is done poorly it will feel more like pressing buttons to play a movie instead of a game.

All in all the challenge aspect has a huge impact in how a game is perceived by the player, and is a very important part of the design process. Most aspects of a game relates to challenge in one way or another, even if it is just displaying the current objective or playing a sound when a point is scored.

6.3 Fantasy

Computer games in general involve a fantasy aspect. The different implementations however vary widely, including both extrinsic and intrinsic fantasies. Common for all is that the fantasies are used to invoke curiosity and interest in playing the game. Fantasies that are very emotionally involving including topics like war, competition and destruction are more popular than the ones that invoke very little emotional involvement.

Fantasies are categorized as either extrinsic or intrinsic, shown in Figure 6.1. Extrinsic fantasies depend on the correct usage of a skill, such as right or wrong answers, or the time it takes to answer. In these types of games the fantasy itself has no impact on the skill usage, the actual problem could easily be presented in a myriad of other ways, one example of a game with an extrinsic fantasy is hangman. For intrinsic fantasies the usage of the skill also depends on what happens in the fantasy, the required responses change as input is given to the game. In a racing game for instance, the current speed would impact the amount of time needed to take a turn.

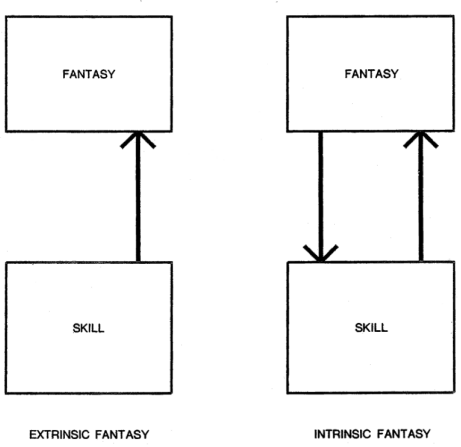


Figure 6.1: Extrinsic and Intrinsic Fantasy

These fantasies have a wide range of realism, often varied to suit the skills needed to complete the given goals. For instance an arcade focused racing game could quickly become boring if the player had to constantly keep track of fuel, oil pressure, and other small tasks, however these things could be very helpful if the game was intended to train professional drivers instead. Depending on whether the fantasy is extrinsic or intrinsic, and the amount of realism involved, the skills acquired from playing the game could in some cases be applicable to a real world counterpart of the fantasy, like in a driving simulator.

6.4 Curiosity

Invoking the player's curiosity is a crucial part in maintaining their interest in the game. This is also highly dependant on the challenge level. The complexity of the required by the game has to be balanced towards the player's current level of knowledge. Optimally, the player should have enough knowledge about the environment that he can predict what might happen, but also still be surprised by being wrong. Although challenge and curiosity are closely related, it is more beneficial from an analytical standpoint to look at them individually. Curiosity is also divided into two categories, sensory and cognitive.

> "Challenge could be explained as curiosity about one's own ability. Curiosity could be explained as a challenge to one's understanding" - Thomas W. Malone

Sensory curiosity is the response to changes in the environment, such as light, sound, or movement. In his book "Four Arguments for the Elimination of Television", Jerry Mander [17] makes an interesting observation regarding how this is manipulated in different types of television programs. He notes the number of "technical events" such as camera angle changes, zoom, and similar changes to the picture per minute. The average number of events for commercials were 20 to 30, while regular programs has about 8 to 10, with public television on 2 to 3. This shows that adding more events can be a simple way to boost interest by engaging the sensory curiosity.

Cognitive curiosity is based upon wanting to expand on currently existing knowledge. The degree of curiosity is largely related to the knowledge one already has about a subject. With no knowledge at all, the curiosity level is normally low to nonexistent, while having a good grasp on a subject more often will lead to wanting to learn even more. You are much more likely to want to finish a story that is ninety percent complete compared to ten percent complete. Cognitive curiosity is also triggered by getting information that at first conflicts with what is currently known. For instance knowing that plants survive by photosynthesis, then hearing about plants that survive in complete darkness.

Utilizing this information in games can be done by graphical or sound effect to arouse sensory curiosity, and by handling the information flow carefully to match the player's current knowledge. The feedback and responses to actions should be clear and concise, with the accompanying information being constructive to aid in the educational aspect where applicable.

6.5 Summary

The main points we can take with us from this is that we need a way to scale the difficulty, as well as having a user interface with clear and concise feedback. We also need to make sure the controls feel intuitive, since this greatly influences the player experience. On top of this we should have a small tutorial that explains how the game is played, to eliminate the need for any instructions before starting.

7 Related Work

Here we will show the work of two earlier master thesis projects that are also using the Kinect sensor. Both of these projects created game prototypes using the Kinect v1 sensor, one with the OpenNI framework, and one using the Microsoft Kinect framework.

7.1 Evaluation of an Interactive Campaign using Motion Sensing Technology

This project, from 2012 by Mari Hansen Asplem and Mia Aasbakken [12], created a motion controlled game to be used in public spaces. The goal of the project is to map out people's reactions to this type of installations and judge if there is a use for them. In order to figure this out, they created a game where the player use their own silhouette to gather balls in a basket. Since the focus of this project was more on people's behaviour and reactions than on the pedagogical aspect, their results give a great deal of insight into how to best approach creating a game aimed at a public setting.

7.2 Learning Recycling From Playing a Kinect Game

José de Jesús Luis González Ibáñez performed a study in 2013, creating a motion controlled educational game with the goal of teaching recycling [14]. This project had a focus on how to best create a game that is fun, while still keeping the learning aspect. There is a multiplayer aspect in this game as well, with both cooperative and competitive modes, although due to the older version of kinect it is limited to two players. The results show an interesting trend in that the multiplayer mode is largely preferred over single player, by over 90%.

7.3 Summary

From these two projects we can gather that there are many unexplored potential uses for the combination of motion tracking and game technology. Combining the findings of Asplem and Aasbakken, as well as the more game related project of Ibáñez, gives a good foundation of results.

Part III

Own Contribution

Our results and findings from working with this project.

8 Microsoft Kinect

In this section we will present our experiences from working with the Kinect v2 sensor combined with the Kinect for Windows SDK 2.0, both from Microsoft.

8.1 Kinect v2 Sensor

This is the latest of the two sensors in the Kinect family, with superior technical specs and tools compared to the first iteration as shown in Section 3.1 Microsoft Kinect. Due to the larger field of view this sensor is easier to place, because it is less restrictive in terms of angling the camera. The improvements in image quality and tracking speed gained from the increased bandwidth of USB 3.0 is noticeable. However, the sensor is limited to USB controllers from Intel and Renesas [4], using other types can result in issues such as losing the connection, corrupted image frames, and even with some of them not detecting the sensor at all. These are not common issues, but it is a good idea to be aware of the possibility and research accordingly. The fact that the sensor does not work at all with a USB 2.0 port also limits the usage of some older computers. The Kinect v1 also required periodical calibration to keep performance at an optimal level, to do this you needed a Kinect Calibration Card which was recognised by the sensor and then aligned with a position on the screen. Calibration is no longer necessary on the Kinect v2 as this is handled automatically.

8.2 The Kinect for Windows SDK 2.0

The Kinect for Windows SDK 2.0 is the latest version of development tools from Microsoft meant to assist the development of applications using the Kinect v2 sensor. The SDK is available from Microsoft's web site free of charge (the Kinect v2 sensor is not). Installing the kit was very straightforward. There is also add-ons for Unity pro and Visual Studio. We installed the NuGet packages from within Visual Studio to add Kinect capabilities to our projects.

8.2.1 What is Included in the Kit

The installation file includes the driver for the Kinect sensor itself, small programs showing the different sensor capabilities, assembly files, Kinect studio, a beta version of Visual Gesture Builder, and Visual Studio (VS) solutions for the example programs. We found the demo programs helpful for understanding the different sensor modes on the Kinect. The VS solutions was a big help for getting started with the project. Although the C# examples are based on Windows Presentation Foundation (WPF), which we were not using, the general logic and approach were the same.

8.2.2 Kinect Studio

Kinect Studio, displayed in Figure 8.1, can be used to monitor the data streams from the Kinect v2 sensor in real time, as well as create eXtended Event File (.xef) files that functions as recordings. It has access to all the data streams, such as color, infrared, depth, audio, and body. This is probably the first application you'd want to launch after installing the SDK, mainly to ensure that everything is working correctly. It is very helpful when placing the Kinect, being able to see the sensor input while adjusting the angle and positioning. It can also play back the .xef files for other application, by sending the data via the Kinect sensor driver. This allows testing applications on a non changing data set, instead of using live input from the Kinect. If Microsoft decides to implement the functionality to allow programmatically controlling Kinect Studio, it would in theory be possible to create automated tests for Kinect based applications without even having a Kinect sensor plugged in.

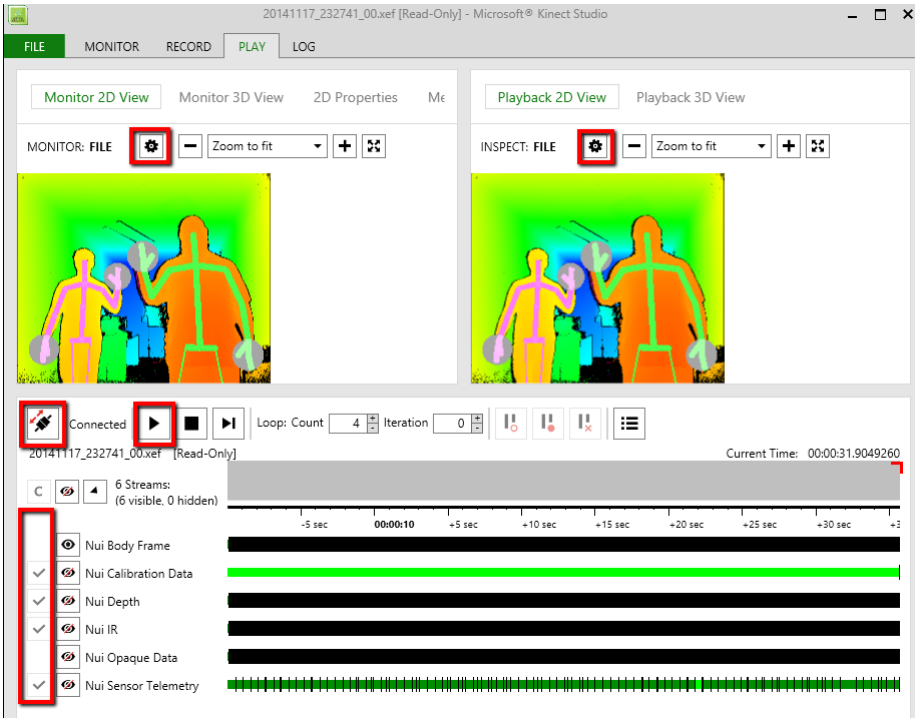


Figure 8.1: Kinect Studio

8.2.3 Visual Gesture Builder

Visual Gesture Builder (VGB), shown in Figure 8.2, is the main tool for creating a Gesture DataBase (.gdb) file, which is used to recognise gestures with the Kinect's skeleton tracking capabilities. VGB uses the recordings from Kinect Studio to create and test gdb files. To create a gesture in VGB you create a new solution, like in Visual Studio, and add a new project to that solution for each gesture. Then one or more clips from the Kinect Studio recordings are added to each project. These clips are manually tagged to indicate whether the gesture is being performed, before they are fed through machine learning algorithms (which ones depend on the type of gesture) which results in a gesture file. This gesture file can then be manually checked by doing a live preview in VGB, as shown in Figure 8.3, showing how confident the system is that the gesture is being performed. If the result is not satisfactory, you can go back and add more clips, or redo

the tags on the previous one. Once all the required gesture files have been created, a gdb file is generated by combining these, this file can then be used to recognise these gestures by another program.

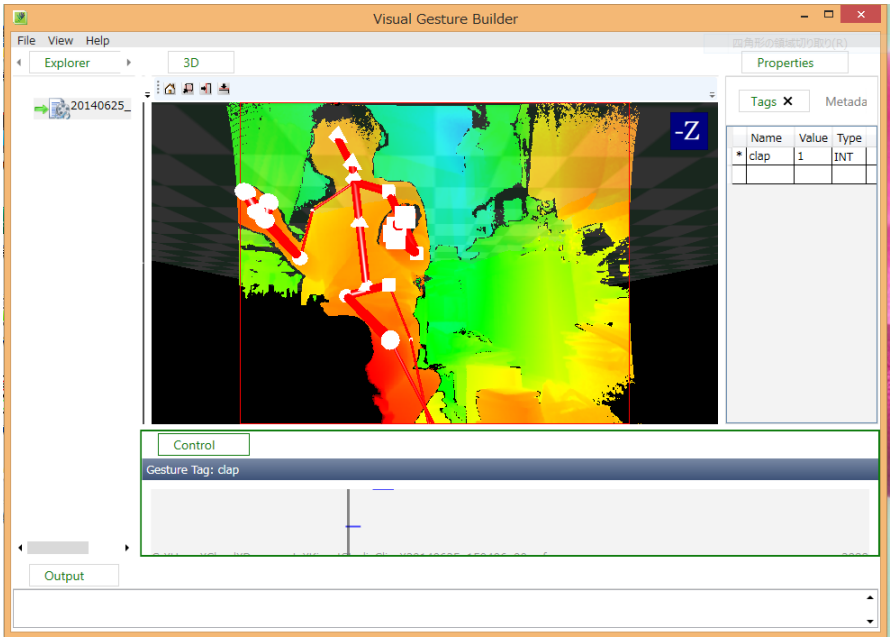


Figure 8.2: Visual Gesture Builder

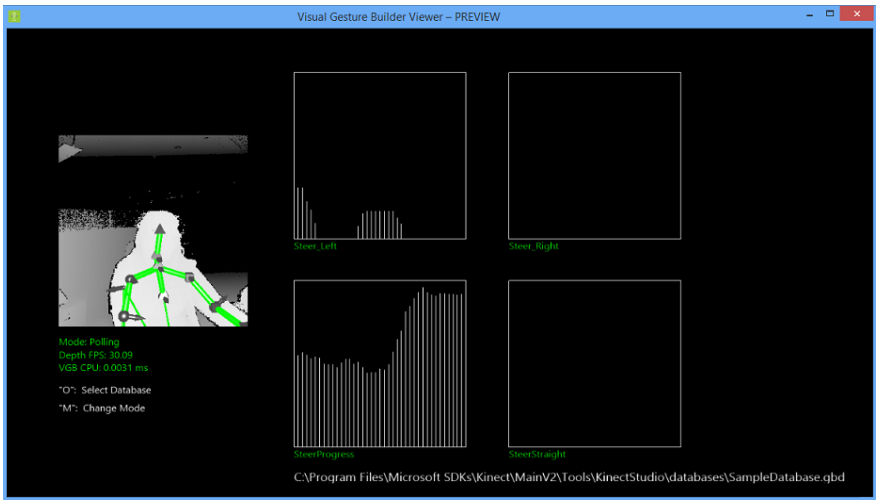


Figure 8.3: Visual Gesture Builder Live Preview

8.3 Body Tracking

Implementing the body tracking itself was fairly easy thanks to the example projects. There are some things about how it behaves though that is worth noting. When deciding on gestures to use, it is useful to use the monitoring functions in Kinect Studio to check if they may be difficult to track. Monitoring the skeleton figure displayed there will show if there is any problems detecting the relevant joint points. Gestures that might obscure the line of sight for the Kinect can cause problems getting detected, we tried with a gesture that required crossed arms, and this did not work well. Looking at the body source with Kinect Studio showed a lot of jittering of the elbow joints during the gesture, due to the occlusion caused by the crossed arms, this made it difficult for the sensor to get good data.

Tracking multiple people at once is fairly simple when using the Kinect for Windows 2.0 SDK, for each person the software returns a body frame. This frame contains all of the data related to body tracking, and some identification functions, such as an id for each registered body. This id however, does not solve everything. The assigned id is an arbitrary integer, and it does happen that a body disappears from the tracking for a frame or two sometimes, for instance if someone walks too close to the sensor. If this happens the same body will get a new random id, this means that the id is not good enough identification by itself. In our case we decided to arrange the bodies according to their x-coordinate instead of relying on the assigned id.

9 Framework

This section will explain our system in detail. We will start with an overview of the system, before we go over each module showing how they are implemented, before finishing with the module that binds it all together.

9.1 Design goals

Our main goal with this framework is aimed at having someone else use it for future projects that also involves creating motion controlled applications. With this in mind we have decided that the main goals of this framework should be modifiability, reusability, and usability.

9.1.1 Modifiability

Modifiability relates to how much of an impact changing one part of a system will have on the rest of the system. Ideally the impact should be small, but the extra cost of preparing for this could outweigh the gain. It is important for this framework since we do not know how it would be used in the future. It has to be possible to make changes without causing large ripple effects throughout the system. We have attempted to minimise the impact future changes would have by splitting the framework into smaller modules, each with their own specialised tasks. In addition we have focused on increasing cohesion and reducing coupling, often via encapsulation.

9.1.2 Reusability

Reusability is defined as the reuse of existing assets, with varying degree of modification. A simple example of reuse is adapting a function from an earlier project. Our solution file also contains two startup projects, enabling building both a DirectX and an OpenGL version. This should be easy to port over to other operating systems.

9.1.3 Usability

Usability describes how quickly a new user can start using a product efficiently. In this case it would mean how quickly they would be able to use the framework to creating a functional application. Usually it is used to describe an application, where the user refers to the end-user and covers items such as user interface and persisting user states. In this case however, we need to tweak it a little so that it covers future users of our framework, who also are developers. The main steps we have taken to make it easier to use is separating it into logical modules with recognisable naming, hoping that anyone using it will find the structure intuitive.

9.2 Engines and Frameworks

Here we will explain our choices in regards to the frameworks and engines we have decided to use for this project. We will cover which factors we felt were most crucial, and our experience with using them.

Note that we will mention NuGet [6] packages several times in this section, NuGet is a standardized way of importing frameworks into a .NET/Mono source code project and is supported by most major IDEs that are designed for these kind of projects, Visual Studio, Xamarin Studio, and MonoDevelop will automatically download the NuGet packages pre-build.

9.2.1 Microsoft Kinect

As we were provided with a Kinect sensor, the project naturally focused on SDK's aimed at this specific sensor. While all the alternatives likely would have had enough functionality for this project, there are a couple reasons why we specifically went for the official SDK. The code module provided with the official Microsoft Kinect for Windows SDK provides the necessary classes for connecting to the sensor, getting data frames from the sensor, and some helpful functions for transforming this data into useful formats for less specialised classes.

Considering the fact that we had no requirements to make this system available for platforms other than Windows, the main downside of using the

code supplied by Microsoft disappeared. If we had multiplatform support as a priority we would have had to go for something else.

The main reason we chose this over OpenNI in the end was mainly the supplied documentation, which also include a lot of useful code samples. These saved us a lot of time, since they showed us how to best utilise the different features of the Kinect. It also allowed us to use the gesture tools provided by the SDK, presented in Section 8.2 The Kinect for Windows SDK 2.0, which greatly simplifies the amount of work required to create gestures that can be used in the system.

All that is needed to start using this module is installing the NuGet package. In addition the Kinect driver included in the SDK is required to get the application to utilise the sensor, but it is possible to start programming without it. After installing the NuGet package, all that is needed is creating a .NET compatible project and including the Kinect based namespaces.

9.2.2 MonoGame

In Section 4.2 Engines and Frameworks we have reviewed several possible candidates with the favorites being Xenko and MonoGame. Ultimately we ended up deciding on MonoGame, primarily because it feels like the safest framework to use in the interest of building software that can be reused in the future. One factor that is not previously mentioned is that the project members were already familiar with Microsoft XNA, which is almost identical with MonoGame from a coder's perspective. The Xenko engine was mentioned as another interesting candidate, but given the previously mentioned reasons, and the fact that our game concept is not complex enough to warrant the introduction of an engine, MonoGame was the more reasonable choice.

Using MonoGame for our project can be best described as "smooth sailing", there has been no apparent bugs or difficulties, aside from one where using OpenGL in Windows would refuse to toggle Full Screen mode, but this did not affect us much as all team members were using Windows and could switch to DirectX. Almost every tutorial or code sample for XNA are directly applicable to MonoGame, so there is plenty material to find that helped us develop using MonoGame. One minor difference is the tool in the SDK provided to organize and build assets (graphics, audio, shaders, etc.), in XNA this was a tool within Visual Studio, in MonoGame it is provided

as a stand alone application. From our experiences with it we would say MonoGame is a robust framework that we can recommend to anyone with programming knowledge who is interested in making simple games.

MonoGame is implemented into our project as a NuGet package, however installing the SDK is still needed as the Content Builder is required to build the project.

9.2.3 Empty Keys UI

Figure 9.1: Empty Keys

On top of MonoGame we also needed a framework for creating a user interface, as this is not included in the MonoGame package. For the sake of building reusable software we wanted a framework that was feature complete, while also being actively maintained and updated. As far as we could tell Empty Keys, Figure 9.1, was the only framework that filled those criterias. Empty Keys is a UI framework compatible with .xaml forms popularized by Microsoft WPF, this also means that by choosing this framework we are given the added bonus of being able to design our UI using the .xaml designer provided in Visual Studio. Empty Keys is compatible with MonoGame, Xenko and FNA, which is another good thing as it means we could with minimal effort replace MonoGame with Xenko or FNA.

An alternative to using Empty Keys would have been to create our own UI using sprites and drawings in MonoGame, which we consider a reasonable alternative in our case, as the UI required for our prototype was very simple. We decided however, that this was best left as a last resort if we could not find any frameworks that simplified creating user interfaces.

Unfortunately we had some issues figuring out how to implement Empty Keys into our application. Documentation or tutorials for Empty Keys is sparse, there is a sample project available however it is not very evident on how you are supposed to set up a project using Empty Keys. We spent

too much time trying to tackle this obstacle, but once it was set up and functional development resumed, our productivity increased greatly as it was easy to further develop and extend the user interface.

There were primarily two things we were struggling with:

1. As it turns out, for Empty Keys to work the developers needs to attach a pre-build step with the "Empty Keys Generator" to their project. The generator takes the .xaml files and generates source code files from them, which will be used in the actual build. We had failed to notice this, much due to the lack of documentation, and spent a long time trying to understand why the user interface in the game did not reflect what we were seeing in the designer.

2. Not everything from WPF is supported. Certain parameters are missing completely, User Controls, which is how WPF splits a user view into separate smaller view components did not seem to be at all functional.

Overall we had a great time working with Empty Keys, it is just a shame that the tutorials and documentation leave something to be desired, for someone already familiar with frameworks using .xaml technology like WPF and Xamarin Forms, Empty Keys is a great framework once it is properly set up.

Empty Keys is included in our project as a NuGet package, where the generator is also included.

9.3 Architecture and Implementation

This section will explain our system in detail. We will start with an overview of the system, before we go over each module showing how they are implemented, before finishing with the module that binds it all together. Most of the following sections contain simplified class diagrams, the full versions of these can be found in Appendix C Class Diagrams.

9.3.1 Modules Overview

A rough overview of our System is presented in Figure 9.2. It consists of four modules that we ourselves have created, namely Motion Control, Game Core, Game Logic, and User Interface. In addition to this we have implemented the third party frameworks MonoGame, Empty Keys, and Kinect SDK. We have decided to give the modules generic names in this report for the sake of simplicity, so here we will give a brief presentation on what each module does and what would be their respective C# project counterparts in our Visual Studio solution as seen in Figure 9.3. Detailed descriptions of each module can be found in Section 9.3.2-9.3.5.

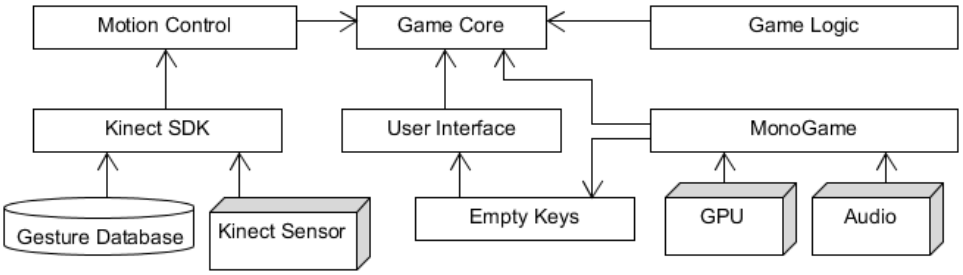


Figure 9.2: Module Overview

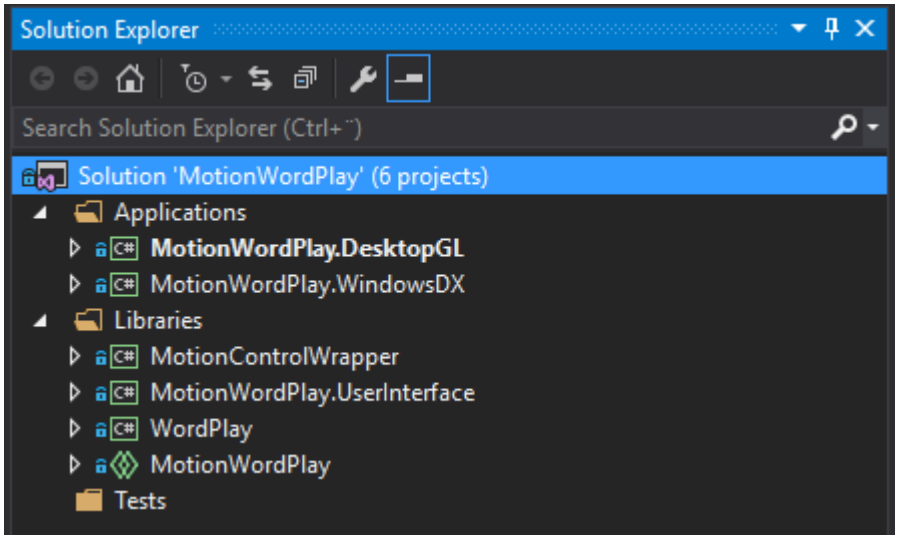


Figure 9.3: Visual Studio Project Screenshot

The **Motion Control Module** is a framework for handling motion controller hardware, more details in Section 9.3.2. The corresponding C# project would be **MotionControlWrapper**.

The **User Interface Module** contains the UI, this is put in a separate project to make the UI more independent, more details in Section 9.3.3. The corresponding C# project would be **MotionWordPlay.UserInterface**.

The **Game Logic Module** contains all the logic for our word play game, more details in Section 9.3.4. The corresponding C# project would be **WordPlay**.

The **Game Core Module** is the implementation module, it imports all other modules, encapsulates them and add game specific functionality like graphics, more details in Section 9.3.5. The corresponding C# project would be **MotionWordPlay**, **MotionWordPlay.WindowsDX** and **MotionWordPlay.DesktopGL**.

9.3.2 Motion Control Module

This module handles everything related to motion control, including sensor communication and skeleton tracking. The main purpose of this module is

to abstract this functionality away from the rest of the system. This is accomplished by using a factory pattern that creates a class which implements our IMotionController interface, as shown in Figure 9.4. In the current state of the system, the only type of motion controller is our implementation for the Kinect v2 sensor, using the official Kinect NuGet package.

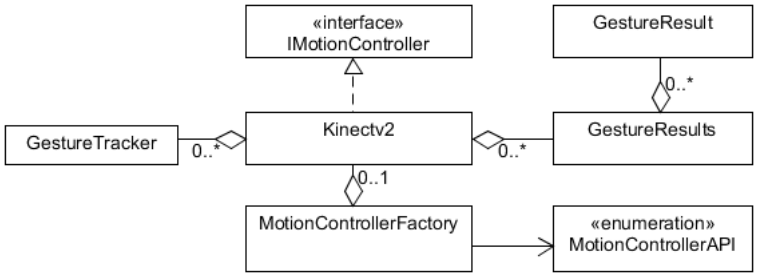


Figure 9.4: Class Diagram for the Motion Control Module

One example of how this interface works is the way we poll and utilise the data frames used for drawing our different display modes. Instead of returning the different frame objects created by the Kinect, the interface specifies that they should be returned as byte arrays, shown in Figure 9.5. Luckily for us the frames themselves have functions implemented to do this conversion.

```
byte[] MostRecentColorFrame { get; }
```

Figure 9.5: Code extract from IMotionController.cs

This byte array is then passed through the interface before it is converted to a Texture2D by MonoGame using the UpdateFrame function, which is then used by the Draw function in MotionController.cs which is part of the Game Core Module, shown in Figure 9.6.

```
private static void UpdateFrame(Texture2D frame, byte
    [] data, Action pollNewFrame)
{
    pollNewFrame();

    if (data != null)
    {
        frame.SetData(data);
    }
}

private static void DrawFrame(Texture2D frame,
    SpriteBatch spriteBatch)
{
    if (frame != null)
    {
        spriteBatch.Draw(frame, Vector2.Zero);
    }
}
```

Figure 9.6: Code extract from MotionController.cs

The intention here is that any future users of this framework can implement their own implementations for other sensors using the same interface. This allows the addition of new sensors, or even just new versions of drivers for existing sensors, without changing any other parts of the system. Every change regarding sensors and their capabilities are contained within the module itself, even if they require a drastically different implementation compared to the Kinect within the module. This means that no matter what may change within the module, as long as the interface itself stays untouched, it will be compatible with any other modules that depend on it.

We assume that most sensors have their own implementation of body and gesture tracking, hence we have decided to create a general class for providing these results to the rest of the system. This is the GestureResult class, which only contains standard data types, thus ensuring compatibility.

9.3.3 User Interface Module

Our user interface module is based upon .xaml schemas that in turn gets generated into C# classes. The GUI is designed as an view in a .xaml schema, with view model classes for the element types where we have data bindings.

An overview of how this looks in our game is illustrated in Figure 9.7. We have a view which represents the GUI as a whole, and view models for the PuzzleFraction and TextBlock custom elements which has bindable data. TextBlock in this case is a simple textbox with a data binding for the text itself, while PuzzleFraction is an extended version which also allows modifying the positioning. The bindings in these custom elements allow us to modify the GUI during runtime, like changing the visibility, text, or the element positions. This is how we manage to get the text boxes containing our sentence fragments to follow the player silhouettes during play.

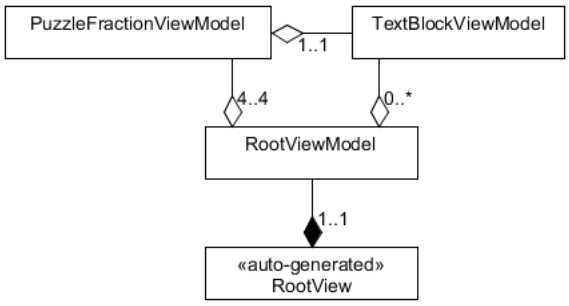


Figure 9.7: Class Diagram for the User Interface Module

The User Interface Module uses the Empty Keys framework which as previously mentioned in Section 9.2.3 Empty Keys UI supports MonoGame, FNA, and Xenko, it achieves this by providing separate assemblies for each of the alternatives. We have explicitly used the assembly for MonoGame in this module, however since it is entirely separate from any MonoGame implementation (this is handled in the Game Core Module, Section 9.3.5 Game Core Module) it can be easily converted to FNA or Xenko by replacing the assembly. The assembly is implemented as a NuGet package which makes the conversion even easier assuming the IDE used by future developers has convenient NuGet support.

9.3.4 Game Logic Module

Here we have gathered the functions pertaining the game itself. The module consists of a single class for this particular game implementation. This class handles everything related to the game logic itself, such as loading tasks from files, the current score, the number of players in the game, and also keeps track of how the current task is progressing.

While the module itself is small, mostly due to the fact that the game concept we have implemented is simple, this only shows the strengths of our other modules. This small class was everything we needed to get a working game that is controlled with the Kinect. And while more advanced game concepts would surely need larger implementations, this shows that it is definitely possible to develop a new game on top of our framework in a relatively short amount of time.

9.3.5 Game Core Module

This is the module that binds everything together, as shown in Figure 9.8. The Game class controls the basic MonoGame functionality, such as setting up a graphics device, loading content, display resolution, and initialising our other modules. The Game Loop, as mentioned in Section 4.1 Game Development Technology, is implemented in the Game class which extends the Game template class from MonoGame.

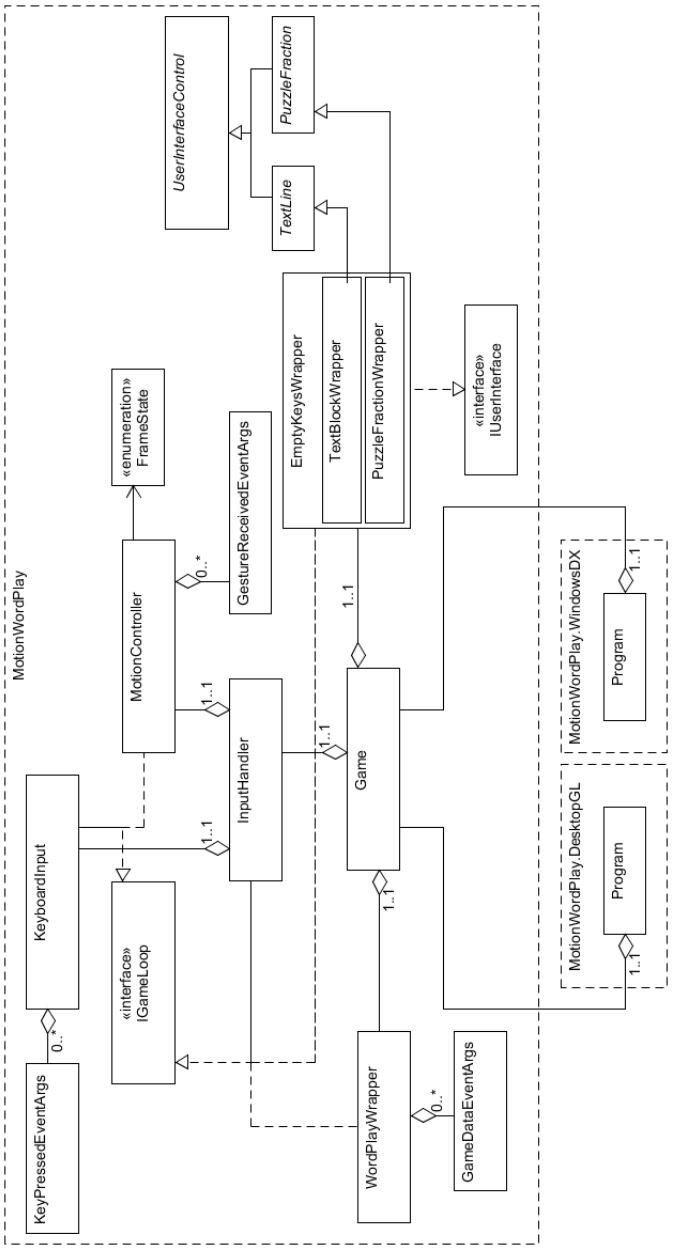


Figure 9.8: Class Diagram for the Game Core Module

MonoGame supports various platforms, for our project platform support is not really relevant as long as we support at least Windows since that

is the native desktop environment for the Kinect. Xbox One is of course also considered native for the Kinect, however in order to develop for that console there is a set of official requirements that needs to be fulfilled so we decided not to. We still support two platforms however, which are both desktop platforms, one for DirectX and one for OpenGL. The OpenGL platform can in theory be executed on Linux and OS X, however without functional Kinect drivers the game would not work properly.

MonoGame handles multiple platform support by providing separate individual assemblies for each platform, we have organized this in Visual Studio by dividing the Game Core Module into three projects, one shared project and two launcher projects. A shared project in Visual Studio is essentially just a collection of code files, nothing more, and this is where we store all of our Game Core code. The launcher projects will then reference the shared project, import assemblies to solve references that might be needed by the code in the shared project, and provide a launcher file that makes sure the game actually start. The launcher project is also where it would be suitable to write platform specific code, but as we have worked mostly on Windows with DirectX there never came an occasion where we had to do that, it is still nice to know that we had the option if we needed it.

There is no communication that runs directly between the other modules, everything is handled through the game core module. Each module is represented by a single class each. This is done to ensure that any future projects building upon this framework can easily reuse or replace modules as they see fit, without breaking any functionality contained in other modules.

Game logic is handled through our WordPlayWrapper, which has events that fires when something important happens in the game, such as getting a correct answer. This cross-module communication is illustrated in Figure 9.9. These events are then used within the Game class to update the GUI or other functionality that is not accessible from the game logic module itself. WordPlayWrapper is a wrapper class which is intended to encapsulate much of the work with initializing the game logic module, as well as providing an easier way to access the information in the game logic module that is relevant for our game core module.

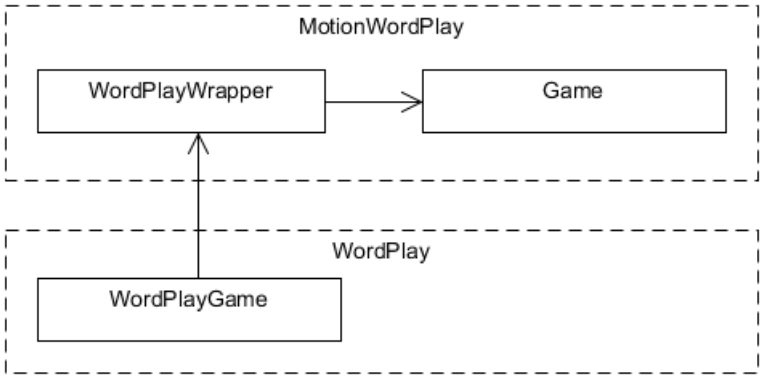


Figure 9.9: Communication between Game and WordPlayGame

The InputHandler, as illustrated in Figure 9.10, takes care of anything related to input, including motion control. Gesture controls via the motion control module is handled similarly, whenever a gesture is registered an event is fired and the Game class can use the passed information to notify other modules, such as the game logic module. This class is created to keep all of the input methods gathered, while also avoiding having functionality for this in the Game class itself, as well as easing the potential work involved with adding new input methods in the future. Motion control within the InputHandler is taken care of by our MotionController class. This works towards objects that implements our IMotionController interface, which are provided through the MotionControllerFactory class. Currently there is only one class that implements this interface, namely the Kinectv2. This is the class that controls communication with the Kinect sensor. To implement another sensor type, it is necessary to create a similar class which also implements the IMotionController interface, and supply this to the MotionController via the MotionControllerFactory.

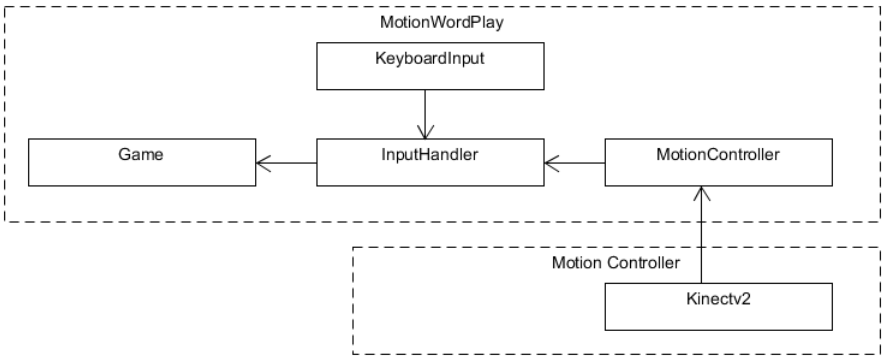


Figure 9.10: Communication between Game and MotionController

The user interface module is accessed through our EmptyKeysWrapper class (shown in Figure 9.11), which fills a purpose similar to the WordPlayWrapper, namely allowing Game access to the different GUI components while abstracting some of the actual implementation. This class is aimed to provide wrapper functionality specifically for the Empty Keys Framework, if any developer wanted to use a different user interface framework they would have to make a new class that extends the IUserInterface and provide their own wrapper functionality. The User Interface Module would also need to be replaced since it is essentially a set Empty Keys specific code.

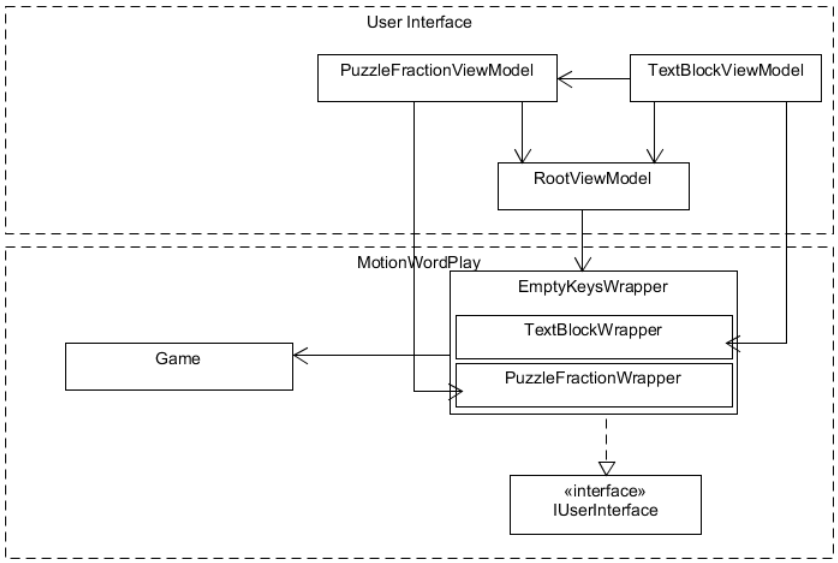


Figure 9.11: Communication between Game and User Interface

10 Game

In addition to developing a framework intended to reduce the workload when creating games for use with motion controllers, we have created a game prototype. This is both to test our own framework, and for gathering data via an experiment. We will present our game concept, show how this has evolved since the original concept which was created before we started working with the Kinect, and detail our final prototype including the results from the playtesting.

10.1 Concept

The game concept is arguably the most important part of any game development process. Although poor execution of a good concept can still make the result fail, even flawless execution will be helpless to save a bad concept. In this section we will show the goals we want our prototype to fulfill, as well as how the consideration of these translated into our final game concept.

10.1.1 Background

Our assignment is very clear on the fact that we have to develop a game intended to encourage learning, this has thus become one of our main goals. In addition to this we also want to do something new, which is very difficult in terms of game development. Since we are developing a game, which should be inherently fun, we want to aim this prototype at the more tedious part of education, namely repetition exercises. These are most prominent in math and language courses, especially at the lower levels of education. Our target audience has therefore become the first half or so of elementary school. Instead of trying to replace the current learning methods, we want to expand upon them. In particular we want to encourage more discussions between the students themselves, and thus make them learn more from each other.

10.1.2 Platform

The concept is intended for use on a Windows PC with a Kinect sensor, but could be adapted to any platform with the ability to do full body tracking of multiple people.

10.1.3 Target Audience

The game is primarily targeted towards elementary school children that needs a motivation boost to take an interest in the written language and the rules behind it. Grammar, wordplay, etc. is something we find that most people are bored with easily. This leads to a lack of practice which in turn leads to a lack of knowledge. We believe that introducing an element of social interaction, both in terms of competitiveness and discussion, will increase the motivation to learn about these topics.

10.1.4 Gameplay

In order to achieve a game that is suitable for an audience with a great degree of variation in preferences, gaming experience, and academic ability, we need to have our core gameplay elements relatively simple. The game needs to be easy to both understand and play, but it also need to be enough of a challenge to keep players interested. From the results based upon "The Recycling Game" in section 7.2 we can conclude that a multiplayer game is preferred. Considering that the Kinect v2 sensor can track up to six persons at once, combined with the space requirements to do this, we believe our game should be targeted at a classroom setting. A typical scenario could then be having a group of students playing the game, while the rest of the class works individually with textbook exercises. In this case the game should be related to the subject at hand and, ideally, tailored to fit the textbooks.

With regard to these points, we have decided to build a prototype focused on language courses, grammar in particular, where the players arrange words to build proper sentences.

Our original concept outlined several different types of tasks that could be suited for this game. We have decided to focus on arranging sentences, displaying a sentences with the word order scrambled, which needs to be put back together in the right order. This type of task is easy to understand, and should be sufficiently difficult for our target audience. The original concept was based upon having the players change position relative to the words. However this caused some issues when they passed by each other, as the Kinect lost track of the person walking behind, thus losing the ID. In order to remedy this we changed it so they instead swap their assigned words with another person, for instance player three and player five can

swap their words by doing a predecided gesture, and thus change the order. This is continued until the players are satisfied that they have created the correct sentence, and then do a gesture indicating that they want to lock their answer. When they do the game checks if it is correct, and if it is the score will get updated and the players are given a new task. The game keeps going like this until a set number of tasks is completed. There is also a bonus for getting several correct in a row to encourage getting it right the first time, instead of going for a more brute-force approach. We believe this can create a better foundation for discussion. There is a timer running in the game, but we do not use it to calculate score. This is because we want to encourage discussion between the players, and time pressure can be a detriment to this. The flow of a game session is shown in Figure 10.1.

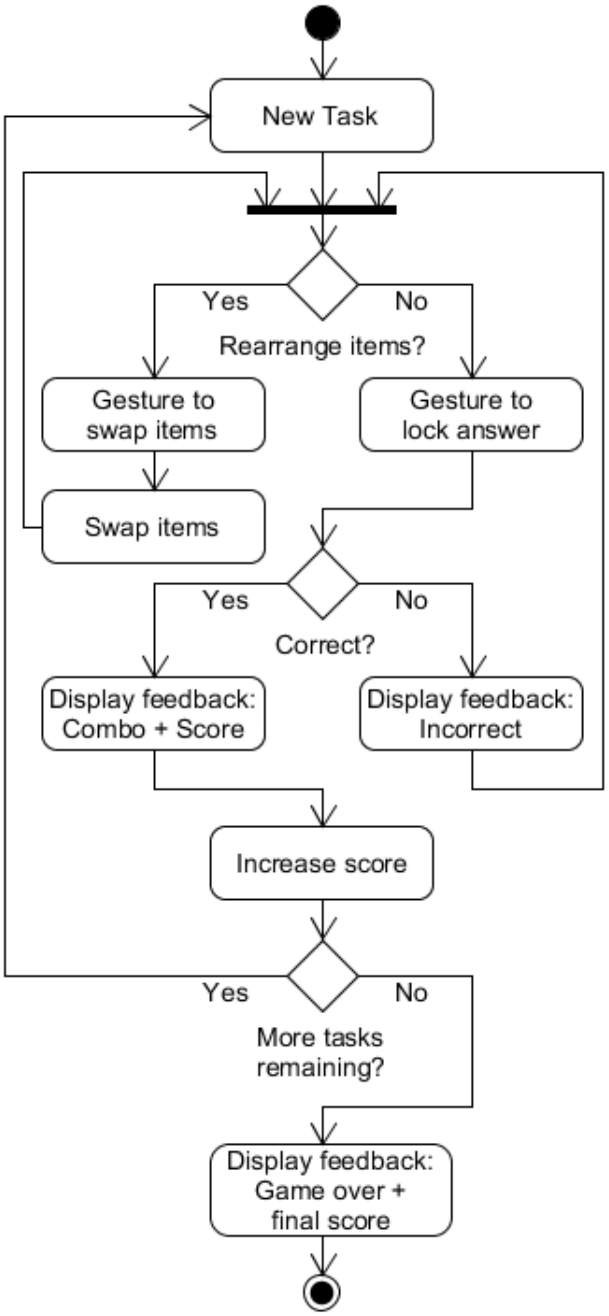


Figure 10.1: Game Flow

10.2 Relation to the Original Concept

First up is the gameplay section of our original concept, in its unaltered state, which we based our prototype on. Our target audience and platform sections are still the same as their originals, and thus need no further discussion in this chapter. Following this we will detail how and why things changed since the first iteration of our concept.

10.2.1 Original Gameplay

Gameplay is mainly built upon having the players move words between themselves to arrange them in the correct order. A sketch of how this might look is shown in Figure 10.2, with figures 10.3 and 10.4 showing how it might look when an attempted answer is either correct or incorrect. Players will be able to move words around by swapping their positions relative to each other, making this a shuffle puzzle that needs solving in addition to finding the correct word order.

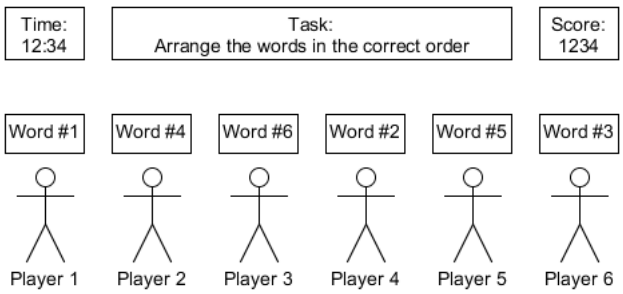


Figure 10.2: Game concept sketch

Main game flow:

1. The game presents the problem along with a short countdown.
2. The game then reveals the initial ordering of the words, and starts deducting points for time spent.

3. Players arrange the words into what they believe is the correct order.
4. Players lock in their answer.
5. The game shows feedback on whether the answer was correct or not. If the answer is incorrect: moves back to 3. and deducts points for every word in the wrong position. Additional reductions if there are several misses in a row to discourage a try/fail approach.
6. Correct answer adds points based on time spent, and a bonus if correct on first try.
7. If the time or number of problem limit is reached the game proceeds to the score screen, else it goes back to 1.

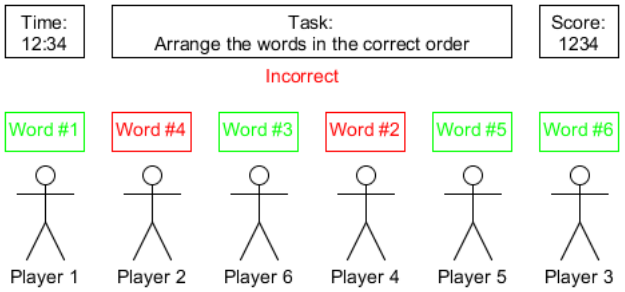


Figure 10.3: Example situation - Incorrect attempt

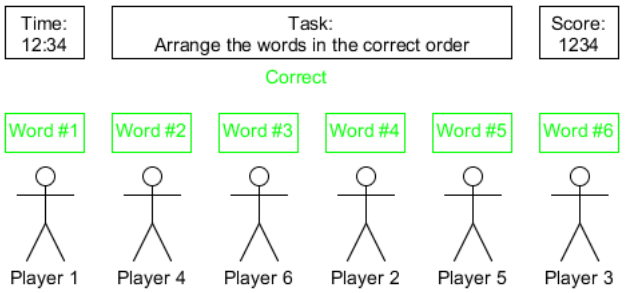


Figure 10.4: Example situation - Correct attempt

Example problem types:

- Arranging sentences: Scrambled sentences that needs rearranging to become grammatically correct.
- Word classes: Identifying and placing words in the right categories such as Nouns, Verbs or Adjectives.
- Verb tenses: Ordering a verbs tenses in the correct order.

10.2.2 Changes from Original Concept

The most fundamental change since our original concept is the fact that we no longer require the players themselves to move around. We thought this would make the game more involving, as opposed to standing on one spot all the time, but we discovered that there were some technical limitations to this, more on this in Section 8.3 Body Tracking. Remedying this would require us to develop our of system for skeleton tracking, a task that would be far too large of an undertaking for us to complete as a part of this project. Instead we adjusted the concept to have the players swap the words.

In addition to this, we have only implemented one type of task. This is mostly due to time constraint, but also the fact that although having several types would be beneficial for a complete game experience, we do not need them in order to create a prototype intended to test if the concept can work.

10.3 Prototype

Since section 9 Framework covers the technical aspect of our implementation, this section will instead cover the game functionality and behaviour during our experiment. The prototype is simple, but does what we have outlined in our concept. Our main goals with this prototype is confirming the functionality of our framework, mapping the Kinect behaviour in practice, and lastly having something playable to use in our experiment.

We have not implemented any menu system, the game starts in a waiting state expecting a set number of players when launched. To start the game from here all of the players need to do a gesture. There are two gestures implemented, one is “Raised Hands”, and the other is “Hands Forward”. These are fairly self-explanatory, “Raised Hands” require the player the place both hands straight up, while “Hands Forward” is both hands straight ahead. The reason we have only two gestures is simply that we did not need more for our concept to work. Our “Raised Hands” gesture is used for two purposes. Firstly we use it to start the game, and second the players perform this gesture again when they want to lock their answer. To swap the words assigned to players during play, any two players perform the “Hands Forward” gesture. This will swap the words they have been assigned. If more than two players perform “Hands Forward” at once, only two will be used.

Our graphical style is simple, the main mode for displaying players is showing their tracked silhouette on a black background, displayed in Figure 10.5, using the silhouette frames provided by the Kinect sensor, with each silhouette having a random predefined color. We have also implemented other display modes supported by the Kinect, namely color, shown in Figure 10.6, and infrared, shown in Figure 10.7. Color acts as a regular web camera, showing the feed from the camera implemented on the Kinect. Infrared does the same, only that it shows data from the infrared sensor as a black and white image, as opposed to a color image. The prototype defaults to the silhouette mode on startup, but the display mode can be changed using the number keys on the keyboard. We decided to use the silhouette frame as default because it is the one that best shows how the Kinect is interpreting the input.

Figure 10.5: Our Prototype in Silhouette Mode

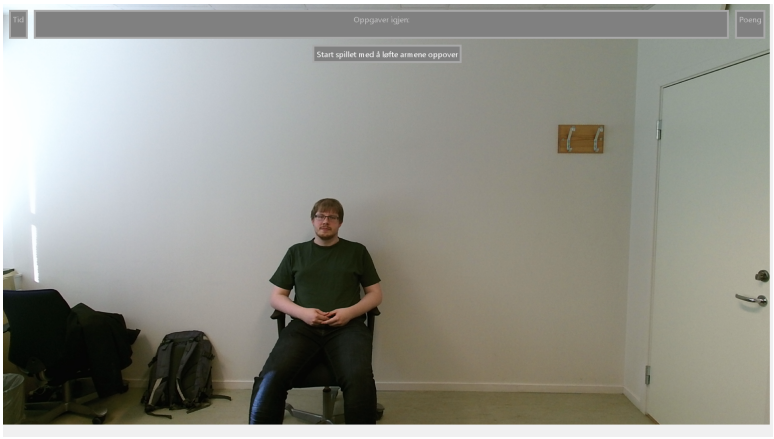


Figure 10.6: Our Prototype in Color Mode

Figure 10.7: Our Prototype in Infrared Mode

10.3.1 Technical Issues

During our experiment, which also doubled a bit as a playtesting session since this was the only time we had been able to get anyone from our target audience to test the prototype, we discovered that we had some technical issues. At first this seemed like it could potentially stop the whole experiment, but we managed to remedy the problem enough to get the prototype playable.

The biggest issue was getting our gestures to register, our "Raised Hands" gesture in particular. Using the diagnostic tools in the Kinect SDK showed that it was registering with a hundred percent confidence level, but we had no luck getting it to work in our prototype. We managed to narrow down the issue to being connected to tracking several people at once, one player showed full confidence levels on the tracking, while the rest hovered around thirty to forty percent. We then tried lowering the threshold value for allowing the gesture to register. This helped the issue, but did not fix it. Ultimately we did not get a working solution for this problem, so we instead used the keyboard shortcuts to bypass the problem altogether. Whenever the players performed the gesture as they should, we pressed the key for the corresponding function. This allowed us to test the game as if it was working as intended, even though it was not.

Luckily the "Hands Forward" gesture was working as it should, allowing us to test the game as we wanted, even if the answer locking was being done by us instead of the players themselves. Having this gesture was much more critical, as we did not have any way to bypass this functionality using the keyboard. It was reassuring to see that the word swapping was working as intended up to six players, as we had previously only managed to test it with up to four. We never had time to find the exact reason for the gesture issues, but we do have a number of theories.

It is possible that the light could have been part of the issue, we ended up being situated alongside some large windows giving us a fair amount of sunlight. We did cover them up as best we could but there was no way to block the light completely. This was probably not the main issue, but it could have had an adverse effect on the sensor input which relies on infrared data. We tried adjusting the angle and position of the Kinect itself, in case the difference when compared to how we had it in our lab had any impact on the readings. After moving it up fairly high giving it a good view of all the players, and making sure it could easily track the joints related to

the gestures, we still did not see any improvements. This indicates that the sensor position does not matter to a large degree, as long as the sensor has a clear line of sight to every player.

Our most likely contender as to why we struggled with the gestures is that we had not been thorough enough when creating our gesture database. This database is generated based upon machine learning using potentially several video clips per gesture, we only used one clip for each gesture because this was working well when we were testing it. While it is not possible for us to actually test if this is the reason, we strongly recommend anyone working on something similar to use several recordings of the same gesture, preferably with different people.

11 Experiment

Here we will present our experiment, which is where we obtained most of our results regarding the potential of using a game such as this for educational purposes..

11.1 Experiment Context

This section will describe the practicalities of our experiment, such as the research group, our participants, location, as well as our results from the questionnaire and observations. The experiment was performed on Monday 30.05.2016 at Eberg SFO.

11.1.1 Research Group

Per Olav Flaten & Henrik Reitan: Handled the main parts of the experiment, setting up the equipment, sorting out the technical difficulties, and explaining the game as well as the questionnaire to the participants. We took turns monitoring the game while the students played, giving the other more room to make observations.

Alf Inge Wang: Our supervisor on this project, he tipped us about contacting Eberg SFO to see if they had opportunity to help with the experiment. He also helped the students fill out the questionnaire, aided us with observations, and even jumped in to play the game when one of the students have to go before they finished playing.

11.1.2 Participants

We were lucky enough to get a group of third graders in the after school programme at Eberg SFO. This fit perfectly with our intended audience. However, since this was an after school programme, we were not able to get a lot of students.

11.1.3 Location

The experiment was performed at Eberg SFO, pictured in Figure 11.1, part of Eberg elementary school, which is relatively new, opened in 1997. The staff was helpful and enthusiastic about the experiment from the start. We were placed in a corner part of a larger room, where there was a projector for us to connect a laptop. Although we were in a multi-use room, we ended up being the only ones there because the students were outside this time of day. Thanks to this we could do the experiment without any neighbouring activities impacting our work. We had some issues getting the participants to maintain their positions due to their energy levels, this was remedied by placing them on chairs instead of standing.

Figure 11.1: Eberg SFO

11.1.4 Experiment Procedure

Once we had our equipment set up, and chairs in place, we started having technical issues. We discuss the actual issues in Section 10.3.1 Technical

Issues. These issues did not slow us down for long, but we did have to send back the first groups of students for a while until we got things sorted out. The students were sorted into groups of five beforehand by the staff, and we had no knowledge about who was in what group, and because we had to delay the first group this caused some mix ups and minor confusion between the groups. This delay sparked some conflicts among the students, as they believed that some of the other groups had skipped their turn.

As soon as we got things working we had no more problems during the experiment. Even having the players fill out the questionnaire after playing was no problem, we had somewhat expected them to complain and make a fuzz about this since filling out forms is not generally known to be very exciting. Most of them had no problem reading the questions, but we did make sure to read them out loud so we avoided any potential misunderstandings. With our participant group being smaller than we had prepared for, we gave them several turns instead. This allowed us to test the modes for different amounts of players with the same group, giving us a better read on the change in difficulty.

One good thing about the fact that we had this few participants was that we could skip a lot of explaining on how to play the game, and instead focus on having them actually play it. The second group of students appeared while the first was playing, so we could also show them by example.

11.1.5 Questionnaire

Our questionnaire was designed to fit our target audience of young children, this includes having mostly yes or no style questions, and relatively simple language. We had hoped to be able to gather more responses to get a dataset that could be used on its own, but instead we have used it as a complement to our observations. There were some responses here that went against what the students said out loud, which is good, this is what we wanted to use the questionnaire for. It is much easier, especially for kids this age, to give negative feedback if they can do it anonymously, though some were definitely not afraid to speak their mind.

Ideally, having a questionnaire like this should be performed with the students separated. It was obvious that the answers given by one student, often also said out loud, swayed the other participants. This is something we would advise any other projects to do, but we did not focus much on this due to our small number of participants.

11.1.6 Interviews

We did not conduct any formal interviews in the traditional sense, but we did get some comments during the playtesting. In addition to this, since our questionnaire got a bit compromised, we tried to get the participants to discuss the questions a bit between themselves. This allowed us to catch up on things that were not necessarily covered by the questionnaire itself.

11.2 Questionnaire results

Here we will cover our most interesting findings from our questionnaire, the full set of questions can be found in Appendix A Questionnaire, and the full set of results in Appendix B Questionnaire Results. We had a group of nine participants, with six girls and three boys, all third graders. Considering the small number of participants we did not distinguish between genders to see if there was any patterns to the answers, we believe this would only lead to potentially false claims. All except one claimed to have tried educational games before. There was also a surprisingly even distribution in terms of gaming habits. This is shown in Figure 11.2, we had three participants each claiming to play games "almost every day", "a few times a week", and "rarely".

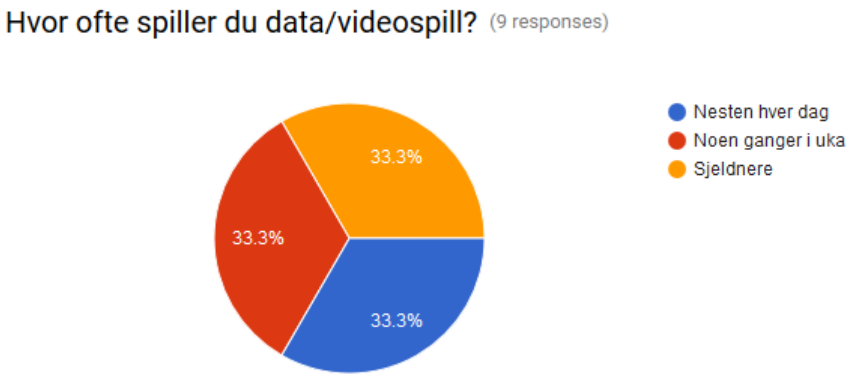


Figure 11.2: How Often Do You Play Games?

All but one of our participants reported having played educational games before, reflected in Figure 11.3, although they were initially a bit unsure what the term meant.When we further explained that it meant games that focused on teaching a subject like for instance math, most of of them recalled having tried something of the sort.

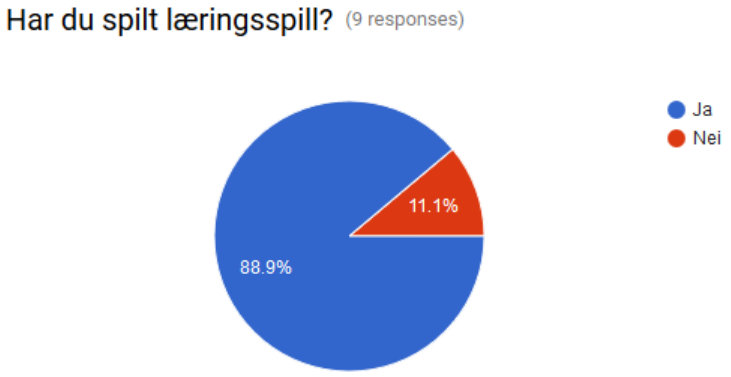


Figure 11.3: Have You Ever Played Educational Games?

We also wanted to see if any of the participants had any experience using motion controllers from before, as this could have an impact on both how quickly they learned the control the game and their impressions. The response, shown in Figure 11.4, was somewhat as we expected. A majority of the participants have tried the Nintendo Wii, and some have tried other alternatives, including the Kinect.

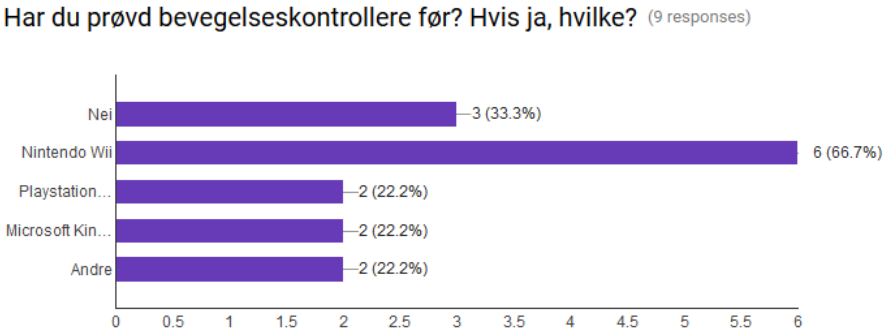


Figure 11.4: Have You Tried Motion Controllers Before?

Once our questions started about our game prototype itself, the answers we got became less varied. Two thirds of our questions, such as "Did you find the game fun?" and "Did you feel that you learned something from the game?" all received 100% yes answers. This was more or less expected, especially considering the age group, but we did get some interesting results nonetheless. Even though everyone claimed that they learned something from playing our game, became more motivated to learn, and said they would have played the game at home if given the chance, a third of them said they would not want to incorporate games alongside traditional education, as shown in Figure 11.5.

Figure 11.5: Would You Want Games to be Used in Addition to Ordinary Education?

When we asked about whether they felt it was easier to focus on the tasks when compared to traditional teaching we got some interesting results, shown in Figure 11.6. Four participants said no, with the other five saying yes. This is interesting because what we observed somewhat indicates that they all worked fairly well on the tasks. One potential cause for this is that they did not feel that they really had to focus on it because it was "just a game", and did not realise themselves the level of concentration they were displaying.

Figure 11.6: Did you feel that it was easier to focus on the tasks than in normal classes?

With the multiplayer aspect in mind, we asked if the participants felt it was easier to get a discussion going than it would be in a normal class. Response was mostly towards yes again, depicted in Figure 11.7, but we did have three participants that said no. This matches well with what we observed, with some players being eager to take charge, leaving less room for the more quiet students.

Figure 11.7: Is it easier to start a discussion about the tasks when they are in a game than when they occur in normal classes?

We also asked whether the participants found it easy to know if the answers they gave were correct or not, shown in Figure 11.8. Three participants said no, but we believe this could also be due to the fact that our first group of testers never got any wrong answers, and thus never experienced the actual feedback related to this question. Still, it is an indication that we should have even clearer feedback on correct answers as well.

Figure 11.8: Was it easy to know if the given answer was correct or not?

After this we asked whether they felt time passed by fast while they were playing, as shown in Figure 11.9. Only two out of the nine participants said no, this indicates that most of the players found the game engaging enough that they to some degree forgot about everything else. This is a good result, showing that our prototype might have real potential.

Figure 11.9: Did you feel that time passed by fast while playing?

At the end of the questionnaire we had a field where we asked for the participants input on what we should do to improve the game. The most common comment to this was that they wanted to control which color they were assigned, this was not something we had even considered before this point. Following this, the second most common, was that the game was too easy. Also, they wanted it to last longer. The game length is something we can change on the fly, but we decided not to as it was already a challenge to keep a group of six third graders sitting still for the sessions we were running at six tasks per playthrough. In a more structured setting than what we had, for instance a normal class in a classroom with their regular teacher, a larger set of tasks could be more suitable.

All in all, we feel that the answers we got on our questionnaire were overall good, if a little small to be an ideal dataset. The answers given reflect what we observed, and for the most part, what we expected.

11.3 Observation results

The kids at the SFO had not been informed beforehand that we would be coming to test a game. We did get some reactions once we arrived however, all kinds of variations on questions like who we were, and what we were doing there. This did not stop once we had our equipment up and running, once the Kinect was up and silhouettes started showing up on the canvas

displaying people in the vicinity, more and more kids started coming up to us asking if this was a game and whether they could play. Obviously we had at least succeeded in creating something capable of generating some curiosity.

Once we got everything set up and our technical issues sorted out, we started testing the game with a group of 4 participants. As mentioned before, getting the players to understand the game was easy enough. Having them stand in one place so they avoided overlapping their silhouettes, and thus making the Kinect struggle, was nearly impossible. This was much due to the fact that they found it interesting to see how their silhouettes interacted. Especially the fact that if you got close enough they merged into one color, and when separated again one player would get a new color. After a bit of struggle getting them started with the actual game instead of just playing with the silhouettes, we ended up setting down chairs with a suitable distance in between so we had the players situated as ideal as possible for the body tracking. Everyone was also noticeably calmer when sitting down, compared to when they were jumping around and running in between each other to see what would happen to the silhouettes.

The first round with four players went by smoothly after getting everyone placed on their chairs. There were no problems with understanding how to play the game, despite several of the players having never tried a motion controlled game before. The first round went by quickly, and ended up with a perfect score, the players never locked a wrong answer. One main reason these tasks went by as quickly as they did was that to a large degree a single person, usually the one that spotted a solution first, just took charge and everyone agreed. This combined with the fact that the first word had a capital letter, and the last word having punctuation, made the four player tasks too easy. Most likely, we should have had separate types of tasks for less players than five.

When we upped the number of players to five, each of the tasks took a little bit longer, and there was a noticeable increase in the amount of word swaps they were doing before locking in the answer. In addition we started seeing situations where three or four of the players wanted to lock the answer, with one or two disagreeing with what they had. This was exactly what we wanted to see, as this lead to the players having to discuss their answer until everyone agreed on a solution. We still did not see any real problems getting to the correct solution, with only a few wrong answers actually being locked.

Running the game with six players, we started to see some struggles appearing. With just over half of the answers being locked correctly. This was much more interesting to watch, when the tasks started getting more challenging, more of the participants began speaking up as well. We also noticed that the ones that had been struggling a bit more with actually reading the words joined the discussions, probably because they got the time to finish reading before someone else had already found the solution.

All of the participants were eager to try the game, and most played three or four sessions, with different amounts of players. This allowed us to observe differences between the different sets of tasks more clearly since we could see how their behaviour changed when things became more difficult, instead of having a separate group of people for each set. There was one thing we would have liked, that we did not get the chance to try. Having an observer that knew how the kids normally behave during class, like a teacher, that could tell us if there was any noticeable differences in how they approached the tasks and the amount of participation in the discussions.

From these observations we think the ideal number of players for the current state of the game is five or six, at least for third graders. Four players ended up making the tasks too easy, and thus negate the educational gain. Fourth graders would probably find the tasks to easy altogether, but it might be suitable for second graders with above average reading skills. Overall we were really happy about the results we got from the third graders.

12 Discussion

The main goal of this thesis is to figure out whether or not there is a basis for focusing on developing motion controlled educational games. We have split this into three smaller subcategories, with one focusing on the feasibility of developing these without already having extensive knowledge about the technology, another on more general game design, and lastly if there is any signs that this kind of games could in fact have an effect on learning. This section will cover both of these, and discuss our work and findings in relation to our research questions. We will start with our findings and observations related to using motion controllers, as well as a bit on educational games in general. Following this we shall cover our results related to the technology itself.

12.1 Using a Motion Controlled Multiplayer Game for Education

The aim of this thesis was mainly to research if using motion controllers in educational games could have any benefits. Our research questions relevant for this are:

- **RQ 1.1:** Is there a benefit for using motion-control as input compared to traditional control methods?
- **RQ 1.2:** Does a cooperative game increase the amount of discussion/-cooperation between people in a classroom compared to traditional teaching?
- **RQ 2.3:** Is it easy for a new user to adapt to using motion controllers as opposed to more traditional input devices?

Our experiment involving our prototype game and the third grade students at Eberg SFO have given us a clear indication that there is a great potential for increasing educational gain by using motion controlled games alongside regular classes. The combination of motion control and multiplayer seemed to be especially well suited for engaging players in this age group.

Any concerns we had about the students having difficulty adjusting to using the motion controls, as opposed to traditional input like keyboard and

mouse or controllers, were quickly taken away. Although the majority had tried motion controllers before, only a few had experience with a type that did not involve physical controllers. This turned out to be a non-issue as the students were quick to grasp the concept, some even before we had explained how the game worked. Turns out using silhouette as the display mode made things more intuitive, as it shows how the scene is interpreted by the Kinect. The students started experimenting with how the silhouettes followed their movements and interacted with each other as soon as they were able to get in front of the screen. They even asked if they could dance around in front of the sensor after they were finished playing a round. Obviously the current generation of kids are quick to pick up on new types of technology and controller schemes. Even though this would require some more testing focus on this very subject, we believe that there should not be any issues introducing a motion controlled game for a full class of students and have them playing within minutes.

Based on the response we got from the students, using motion controllers as a replacement for the more traditional input methods did not matter much for their ability to play the game itself. But they did find it more interesting than they probably would have done if they had been sitting on individual screens using a keyboard. Our experiment did not show any real difference to how long it took for the players to get comfortable with the Kinect in regards to their gaming habits. This indicates that the intuitive nature of using your own body as a controller as opposed to any intermediary, such as a keyboard, is more easily picked up by non-gamers. In addition, using the player's own bodies as the controllers makes them feel more like a part of the game, instead of just controlling an avatar on the screen. However, this means that the responsiveness of the controls are more important than ever, it easily feels clunky if the game is too slow to register the player movements.

Adding the multiplayer aspect into the mix with education and motion controllers seemed to be a good idea. First of all this allows us to create games that has the ability to let every student in a class have a turn during a single period, by letting larger groups play at once. This could help with adding variety to the classes where a lot of the time is spent working on individual or group tasks, while at the same time encouraging more discussion and learning with the motivation being getting the game's high score within the class. Our observations pointed out that the students were quick to start discussing the answers to the tasks between themselves, showing that the threshold for speaking up was low. Assuming this is a general trend when

playing games, we believe that using games to present the tasks can create more discussions within the related topics, thus increasing the amount the students are learning from each other. With a well structured game, the students also would not need much supervision (although it is interesting to watch them discuss solutions), making it entirely possible to have a game session going alongside the regular class.

Much of the feedback we received from the experiment was based upon the difficulty of the tasks, as well as the overall length of a round. For the sake of the experiment we did not want to have each group spend too much time on each round, we instead had several players come back and try again. This did have the unfortunate effect that the game ended when the players had really gotten into it and were not quite ready for it to be over yet. We consider this a good sign, as it means they most likely would not have lost interest for a while still. In our prototype the difficulty of the game was directly proportional to the amount of players, since the length of our sentences was one word per player. This lead to the four player groups having an easier time than the six player group. The ideal fix for this would be to create tasks type where the difficulty is not related to the number of players, and ideally having several types.

Overall we have some positive results, and believe they could warrant further investigation. More conclusive evidence gathered by having a much larger scale on the experiment, and ideally including several subjects of education, would further the insight on this topic immensely.

12.2 Important Factors for Designing a Game

When creating an educational game it is imperative that we do not forget to include aspects that make the game fun and engaging. There is no point in creating an educational game that no one will want to play. To circumvent this issue we have tried to answer the following research questions:

- **RQ 3.1:** How do we keep the player(s) interested in the game over time?
- **RQ 3.2:** What are important factors to consider when designing a game?

Since we had every student participating in our experiment coming back wanting to play more after their first turn was done, we must have been doing something right when creating our prototype. As our goal was to be able to create something that could be used alongside textbook tasks during classes, having the students come back to play more of their own volition was much better than we had hoped for.

Because our focus have been on creating a game prototype that was to be used in the lower end of elementary school, it was important that the game concept was not overly complex. We have focused on the findings in Section 6 Designing an Engaging Game. The research covered here lists some interesting facts about how to best design a game to keep the players interest. In particular it lists the need to provide the player with a continuous stream of choices to be made. This does not necessarily entail answering questions constantly, but rather subtle things they do not really think about. In our case this is covered by the fact that they players need to shuffle the words back into the correct order. And although there was no extra reward for doing it in the correct order, they became inherently focused on doing in the most efficient fashion.

Giving the players ample feedback about both how they are doing, and that the game is responding to their actions is also important. Our way of telling them if their answer was wrong is simply showing a dialogue box with the words "Incorrect answer, try again!" with bright red letters. This is a simple, but effective, way of conveying their progress during the game. If their answer is correct we show them a similar dialogue box with green colored text telling them their new score, and bonus points if they have managed several correct answers in a row. We believe the positive reinforcement of a combo system is more effective than subtracting points for giving the wrong answer.

There is also a timer in our prototype, but this is not used for impacting the score in any way currently, because the added pressure of having to hurry the tasks could impact the likelihood of meaningful discussions in a negative way. It is however useful when trying to gauging the most suitable number of tasks per round.

Any time when new types of tasks are implemented to the game, it is important to consider the impact they have on the gameplay. The tasks have to be of a suitable difficulty, preferably scaleable to fit the players preferences. Adding variety to the types of tasks is beneficial to keep the

player's interest, but could also be hazardous for their concentration if the pacing gets broken up too much.

12.3 Technology

Working with motion controller technology has been interesting, and as this is a relatively new field of study there is not a whole lot of previous work to build upon. Our research questions related to the motion technology are:

- **RQ 2.1:** How easy is it to start developing a motion-capture application?
- **RQ 2.2:** How mature is the current technology, is the hardware available for the average developer good enough?

In regards to the hardware, at least in the case of the Kinect, it is plenty powerful. When it comes to deciding which API and drivers to use with such hardware however, it becomes more complicated. There are several possible combinations of drivers and frameworks to use when creating a motion controlled application, and more often than not, the documentation is lacking. It is difficult to decide on which one suits your needs best. In some cases, like linux support, it is easy to for instance discard the Microsoft SDK since it only works on Windows, but this still does not cover all the alternatives. Some of the best sources of information is blogs run by developers who have had the same issue before, but most of the time these refer to older versions of the available software, thus often failing to mention that one tiny detail that would have finalised your decision due to a change in a newer version.

Our impression is that most of the alternatives can do the job, and the final decision should instead be based on what you want to combine it with. If the project is based upon WPF for instance, the Microsoft SDK for the Kinect would be a prime candidate, not only because of the inherent compatibility, but also the included code samples targeting the same platform. Likewise if the project is focused in supporting several types of sensors, and running on any platform, OpenNI is likely to be the better choice. This is a major part of our motivation to develop our framework, as explained in Section 9Framework, is to assist in the starting phase of a new project by providing a foundation to build upon.

While there are several alternatives to the Kinect, all of the hardware options are essentially the same thing, an infrared camera. The important work is done on the software side, this consists of using computer vision techniques to translate the infrared input into a depth image, and from there extrapolating to other features like skeleton tracking. The skeleton tracking implementations vary in function between the different alternatives, but the core of them remain the same; keeping track of where the users are, and how they move. These differences become apparent in things such as what types of gestures are supported, and the number of possible users that can be tracked at once. The skeleton tracking is the most complex part of these frameworks, and the one that helps developers the most. Implementing your own skeleton tracking system would be a huge amount of work, assuming you have the skill to do it. Not needing to do all this allows more time to focus on game design and developing features, instead of doing all the groundwork.

Getting a hold of a sensor for use with a motion controlled application is for the most part affordable, there are several alternatives that most hobbyists should be able to afford. When it comes to the software part, most of the alternatives are open source, and free. While some others, such as the Microsoft Kinect SDK, are proprietary, they are freely available but only supports their specified hardware counterpart, and do not allow any changing of the drivers themselves.

Part IV

Conclusion

A summary of our project, and suggestions for further work.

13 Summary

Over the course of this master's thesis we have designed, developed, and evaluated a motion controlled educational game. We started the project with a literature study to get familiar with the subject at hand, as well as some research into the technology we would base the project on. The literature study helped us decide on how to progress with our project, especially in terms of how we wanted to design our prototype. Our focus quickly turned towards making a multiplayer game, especially targeted towards a younger audience. This decision came from both personal dislike for how much of the earlier years of education are structured, as well as a belief that we could have an actual impact on how it could be shaped for the future.

During the project we performed an experiment involving third year elementary school students, where we had them try out our newly developed prototype. One of the main goals of our design was to be able to allow a whole class to play the game during one period, something we achieved. We were able to have six students play at once, which makes it feasible to allow a whole class to play during a relatively short amount of time by assigning them to rotating groups. The observations we noted during the experiment were largely positive, the students enjoyed the game (especially well noticed by the fact that several came back multiple times), they were actively discussing the tasks they were given, and everyone was eager to participate from the start. Engagement levels during the experiment were way higher than we had anticipated beforehand, especially when considering the simplicity of our game. This only shows how much of an impact adding some less used elements into the mix can have, not only the motion controls but also the multiplayer focus. Neither of these things have been prevalent in educational games targeting this age group, with motion controls largely non-existent in general. They also showed much interest for the Kinect itself, many of the students had experience with motion controls from before, but only a couple had tried a system based on camera tracking. Some of the interest in the game could be explained by the novelty factor of motion tracking, and might be argued that it gets old quick, but we still believe that this is an area of usage that has potential.

All of our student participants claimed that they preferred games like this to be multiplayer. Adding a social aspect to the games often lead to an increase in playtime. Even when the game itself is not intended to be directly competitive, players often create competitions among their friends and social circle to compare how well they do, just look at the Flappy Bird

craze from a short time back. This competitive aspect is a good motivator, and if harnessed correctly it can lead to an increase in educational gain as the students want to spend more time playing the game.

The technology itself has also gotten a thorough review during our project. As far as we can tell the main drawback to using this kind of motion tracking for the average home user is the amount of open space required. In terms of cost the technology is starting to get affordable by most, in the case of the Kinect it even came bundled with the Xbox console for a while. However, there is a distinct lack of good games that utilise it. Taking into account the low interest for having such things at home also affected our decision to aim our prototype at a classroom setting. All of our findings suggest that this is something that should be taken further, with more feature complete games, and a larger pool of test participants.

14 Future Work

The combination of motion capture and multiplayer educational games is an area that still warrants further exploration. Both the observations we noted during the experiment and the feedback we received on our questionnaire show that there is a great deal of potential for this kind of games. The only problem is how to best tap into this potential, not to mention getting through the bureaucracy needed to get it implemented in any kind of approved curriculum. Throughout the project we have gathered some pointers on which areas need more work before we believe this topic can be closed, as well as which technical aspects of our prototype that needs addressing.

First and foremost we would like to see a larger study than what we were able to perform with our experiment, namely a larger amount of participants with more varying age, varied types of tasks, and different subjects.

The game prototype itself is far from a finished product. A menu system would benefit it nicely by working as a sort of tutorial for getting used to controlling the game using gestures, there should also be a small tutorial available that could assist new users on how to play. The user interface needs to be made more clear, and scale better to different screen sizes. We also saw the the horizontal alignment on the textboxes containing the single words did not quite match the players positions.

There should be a highscore list, this would increase the level of competitiveness among the players and thus increase motivation to keep playing. We also had several requests on our questionnaire to add more customisation options, such as nicknames and choosing their own silhouette color for each individual player. It should be possible to modify the game length by setting the amount of tasks for a round before starting.

Support for different topics should be implemented, ideally allowing for a mix of several subjects to the chosen at the start. Examples include math or different languages. In addition to this there should be more variation in the tasks within one subject. There should be an accompanying tool to configure the tasks that can appear in the game, this would allow for instance teachers to modify the subjects covered in the game to suit their curriculum.

For the game itself we have identified two more issues that needs to be fixed. One is the game over screen, currently this is only a short message

showing the words "Game Over" and the score acquired from this round. The other issue is somewhat related to this, namely sound. We currently have no sound implementation as this was not important for a prototype, we are even more sure about this after experiencing the noise level six third graders can produce during a gaming session. However it is something that greatly impacts how finished a product feels and should be implemented.

Lastly, we experienced issues having our gestures register during the experiment. This was most likely caused by us not being thorough enough when creating our gesture database file. Ideally this should be done by working with several recordings of different people performing the same gesture. Since we had everything working out just fine in our lab we assumed this would be the case when we were to test the prototype, which it was not. We also recommend using someone from the target audience when creating this gesture, as a possible reason for the problems is the difference is physical size between us university students which did the recordings in this project, and the third graders who were testing it.

References

[1] CocosSsharp. https://github.com/mono/CocosSharp.

[2] FNA. http://www.monogame.net/.

[3] Game Loop. http://gameprogrammingpatterns.com/game-loop.html.

[4] Kinect Issues. https://support.xbox.com/en-US/xbox-on-windows/accessories/kinect-for-windows-v2-known-issues.

[5] MonoGame. http://www.monogame.net/.

[6] NuGet. https://en.wikipedia.org/wiki/NuGet.

[7] OpenNI. https://en.wikipedia.org/wiki/OpenNI.

[8] Unity. https://unity3d.com/.

[9] X-Plane certified. http://www.x-plane.com/pro/certified/.

[10] Xenko. http://xenko.com/.

[11] XNA. https://en.wikipedia.org/wiki/Microsoft_XNA.

[12] Mari Hansen Asplem and Mia Aasbakken. Evaluation of, an Interactive Campaign. Exploring the use of a motion-controlled game in a public space. Master's thesis, Norwegian University of Science and Technology, June 2012.

[13] Victor R. Basili and Gianluigi Caldiera. Goal Question Metric paradigm. *Encyclopedia of Software Engineering*, 1994.

[14] Jose de Jesus Luis Gonzales Ibanez. Motion Capture in Educational Games. Master's thesis, Norwegian University of Science and Technology, June 2013.

[15] Sue Halpern. Virtual Iraq, Using simulation to treat a new generation of traumatized veterans. *The New Yorker*, 2008.

[16] Thomas W. Malone. What Makes Things Fun to Learn? A Study of Intrinsically Motivating Computer Games, 1980.

[17] Jerry Mander. *Four Arguments for the Elimination of Television.* HarperCollins, 1978.

[18] Penelope Sweetster and Peta Wyeth. GameFlow: A Model for Evaluating Player Enjoyment in Games. *ACM Computers in Entertainment, Vol 3*, 2005.

Part V

Appendices

Anything and everything that did not have a place earlier in the thesis.

A Questionnaire

Spørreundersøkelse Eksperiment

Bakgrunnsinformasjon

1. **Er du gutt eller jente?**
   Mark only one oval.
   - ◯ Gutt
   - ◯ Jente

2. **Hvilken klasse går du i?**
   Mark only one oval.
   - ◯ 1
   - ◯ 2
   - ◯ 3
   - ◯ 4

3. **Hvor ofte spiller du data/videospill?**
   Mark only one oval.
   - ◯ Nesten hver dag
   - ◯ Noen ganger i uka
   - ◯ Sjeldnere

4. **Har du spilt læringsspill?**
   Mark only one oval.
   - ◯ Ja
   - ◯ Nei

5. **Har du prøvd bevegelseskontrollere før? Hvis ja, hvilke?**
   Check all that apply.
   - ☐ Nei
   - ☐ Nintendo Wii
   - ☐ Playstation Move
   - ☐ Microsoft Kinect (Xbox)
   - ☐ Andre

Om Spillet

6. **Synes du spillet var morsomt?**
   Mark only one oval.

   - Ja
   - Nei

7. **Var det enkelt nok å forstå hva dere skulle gjøre i spillet?**
   Mark only one oval.

   - Ja
   - Nei

8. **Var det bedre å spille et læringsspill med Kinect enn med mus/tastatur?**
   Mark only one oval.

   - Ja
   - Nei

9. **Foretrekker du et læringsspill der mange kan spille samtidig?**
   Mark only one oval.

   - Ja
   - Nei

10. **Lærte du noe av spillet?**
    Mark only one oval.

    - Ja
    - Nei

11. **Kunne du tenkt deg at spill hadde blitt brukt i tillegg til vanlig undervisning?**
    Mark only one oval.

    - Ja
    - Nei

12. **Ble du mer motivert av å lære om å lage setninger med spill enn av vanlig undervisning?**
    Mark only one oval.

    - Ja
    - Nei

13. **Følte du at det var lettere å konsentrere seg om oppgavene enn i vanlig undervisning?**
    Mark only one oval.

    - Ja
    - Nei

14. **Er det enklere å starte en diskusjon om oppgavene når det er i et spill i forhold til vanlig undervisning?**
Mark only one oval.

- ◯ Ja
- ◯ Nei

15. **Synes du at det er en fordel å kunne samarbeide med andre?**
Mark only one oval.

- ◯ Ja
- ◯ Nei

16. **Var det enkelt å vite om svaret som ble gitt var riktig eller galt?**
Mark only one oval.

- ◯ Ja
- ◯ Nei

17. **Synes du tiden gikk fort mens du spilte?**
Mark only one oval.

- ◯ Ja
- ◯ Nei

18. **Var det viktig å samarbeide for å gjøre det bra i spillet?**
Mark only one oval.

- ◯ Ja
- ◯ Nei

19. **Synes du dere samarbeidet godt?**
Mark only one oval.

- ◯ Ja
- ◯ Nei

20. **Ville du spilt dette hjemme om du hadde hatt muligheten?**
Mark only one oval.

- ◯ Ja
- ◯ Nei

21. **Hva tror du vi bør gjøre for å få spillet bedre?**

.....

.....

.....

.....

.....

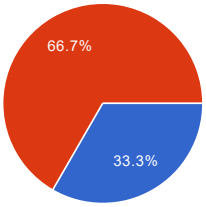
B Questionnaire Results

9 responses

Summary

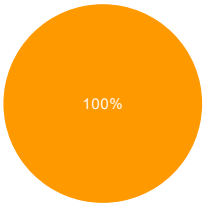
Bakgrunnsinformasjon

Er du gutt eller jente?



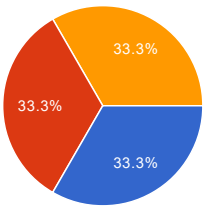
Gutt	**3**	33.3%
Jente	**6**	66.7%

Hvilken klasse går du i?



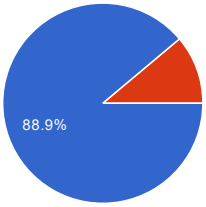
1	**0**	0%
2	**0**	0%
3	**9**	100%
4	**0**	0%

Hvor ofte spiller du data/videospill?



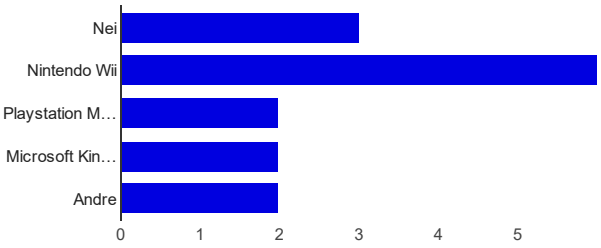
Nesten hver dag	**3**	33.3%
Noen ganger i uka	**3**	33.3%
Sjeldnere	**3**	33.3%

Har du spilt læringsspill?



Ja	**8**	88.9%
Nei	**1**	11.1%

Har du prøvd bevegelseskontrollere før? Hvis ja, hvilke?



Nei	**3**	33.3%
Nintendo Wii	**6**	66.7%
Playstation Move	**2**	22.2%
Microsoft Kinect (Xbox)	**2**	22.2%
Andre	**2**	22.2%

Om Spillet

Synes du spillet var morsomt?

Ja	**9**	100%
Nei	**0**	0%

Var det enkelt nok å forstå hva dere skulle gjøre i spillet?

Ja	**9**	100%
Nei	**0**	0%

nn med mus/tastatur?

Ja	**9**	100%
Nei	**0**	0%

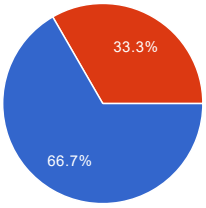
Foretrekker du et læringsspill der mange kan spille samtidig?

Ja	**9**	100%
Nei	**0**	0%

Lærte du noe av spillet?

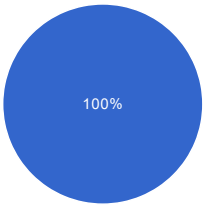
Ja	**9**	100%
Nei	**0**	0%

Kunne du tenkt deg at spill hadde blitt brukt i tillegg til vanlig undervisning?



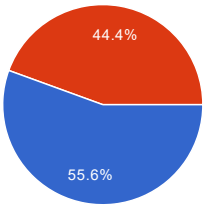
Ja	**6**	66.7%
Nei	**3**	33.3%

Ble du mer motivert av å lære om å lage setninger med spill enn av vanlig undervisning?



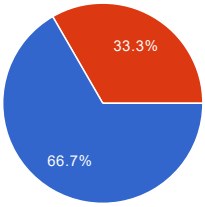
Ja	**9**	100%
Nei	**0**	0%

Følte du at det var lettere å konsentrere seg om oppgavene enn i vanlig undervisning?



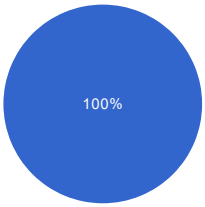
Ja	**5**	55.6%
Nei	**4**	44.4%

Er det enklere å starte en diskusjon om oppgavene når det er i et spill i forhold til vanlig undervisning?



Ja	**6**	66.7%
Nei	**3**	33.3%

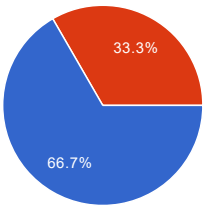
Synes du at det er en fordel å kunne samarbeide med andre?



Ja	**9**	100%
Nei	**0**	0%

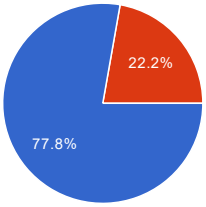
Var det enkelt å vite om svaret som ble gitt var riktig eller galt?

Ja	**6**	66.7%



Nei	**3**	33.3%

Synes du tiden gikk fort mens du spilte?



Ja	**7**	77.8%
Nei	**2**	22.2%

Var det viktig å samarbeide for å gjøre det bra i spillet?

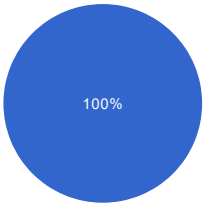
Ja	**9**	100%
Nei	**0**	0%

Synes du dere samarbeidet godt?

Ja	**9**	100%
Nei	**0**	0%

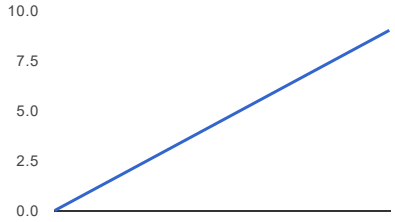
Ville du spilt dette hjemme om du hadde hatt muligheten?

Ja	**9**	100%
Nei	**0**	0%



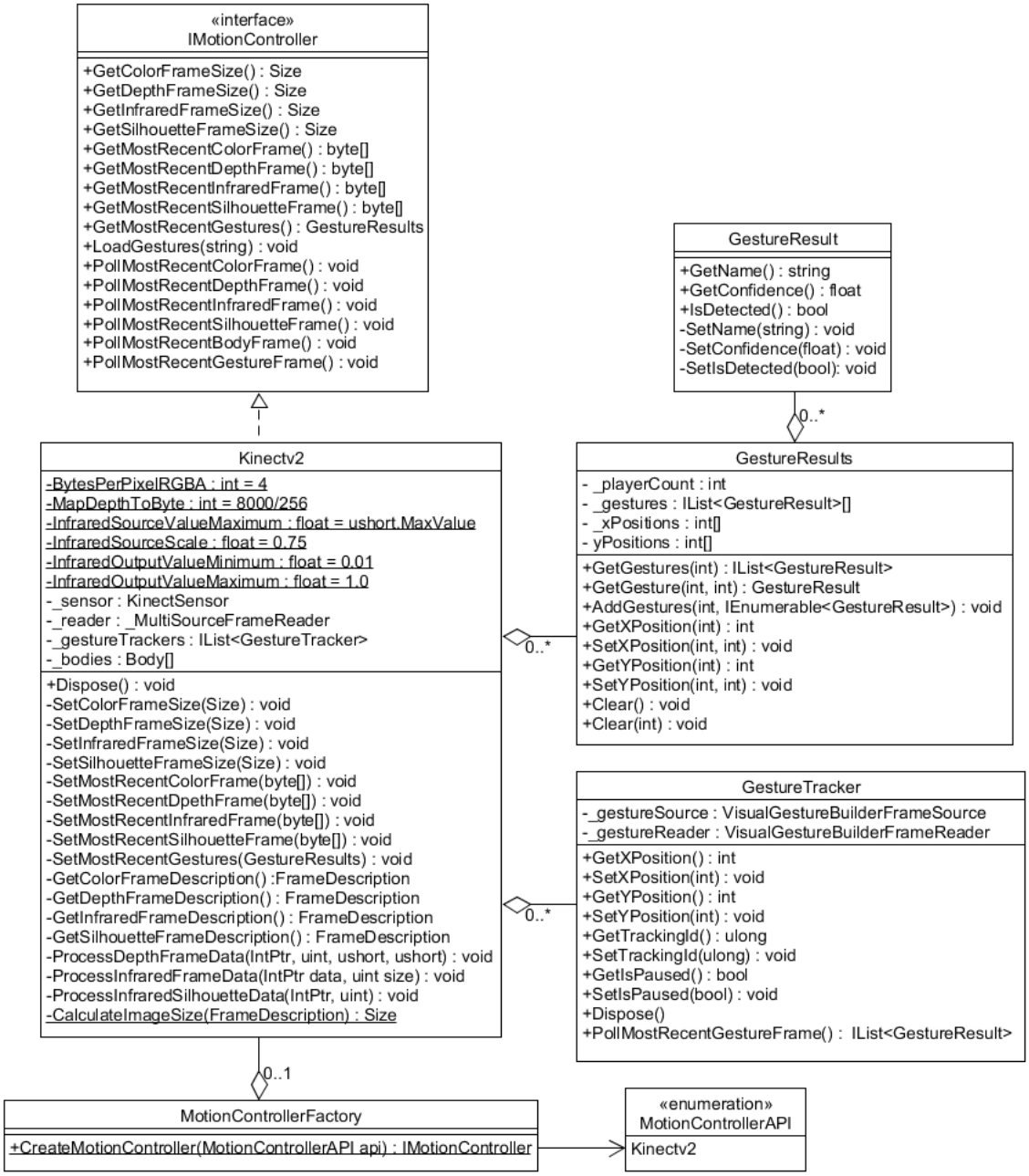
Kontrollere hvilken farge du får Mer spennende bakgrunn

Number of daily responses

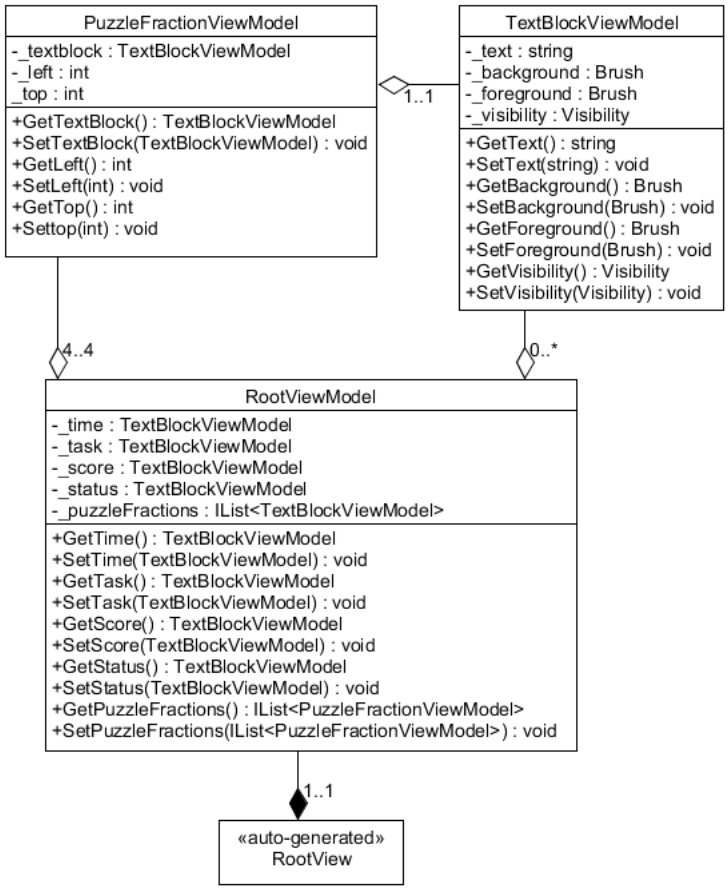


C Class Diagrams

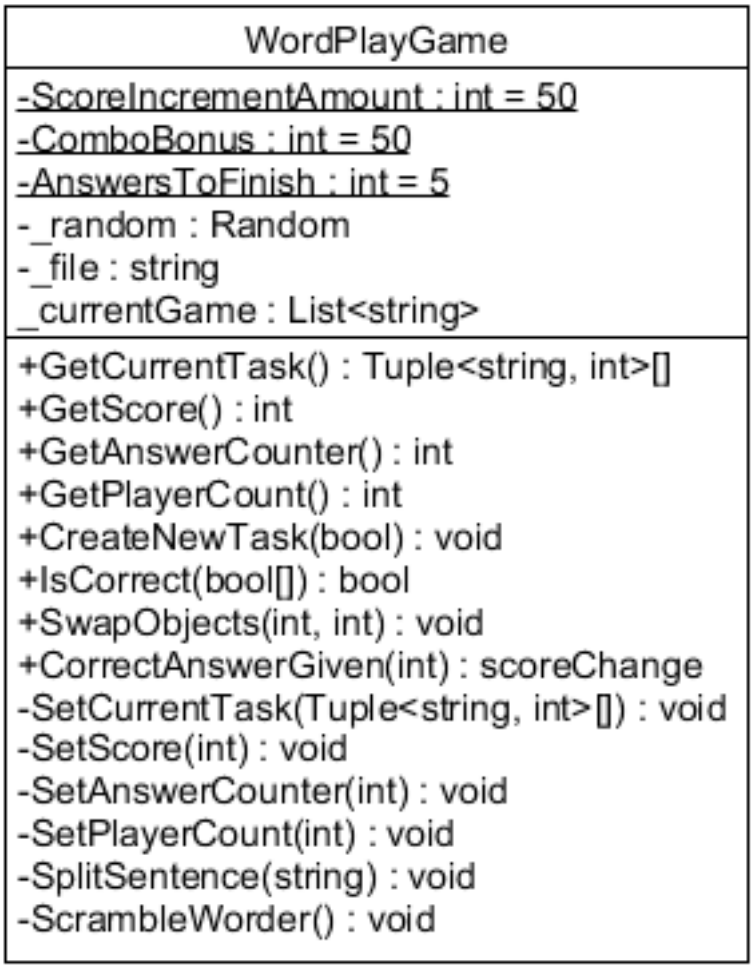
C.1 Motion Controller Module



C.2 User Interface Module



C.3 Game Logic Module



C.4 Game Core Module