Mini games for educative speech recognition

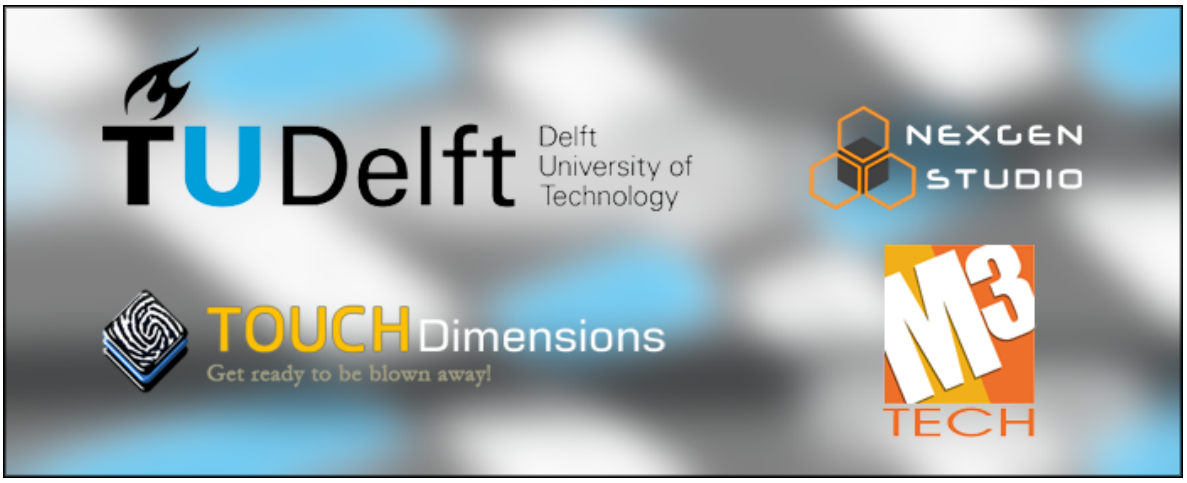
BSc Computer Science thesis, IN3405

Irene Renkens(1534629), Timothy Kol (1509217)

August 29, 2011

M3 Interactive, Nexgen Studio
M3 Technologies, Touch Dimensions

Delft University of Technology
EEMCS Faculty



Committee

Touch Dimensions
Jeffrey Jiang

Delft University of Technology
Peter van Nieuwenhuizen
Gerwin de Haan

i

Preface

In order to graduate their bachelor exam at the Delft University of Technology, students are required to complete a BSc project. For those majoring in Computer Science, this project comes in the form of completing an assignment, usually for an external company during an internship, although it is also possible to do this assignment within the university itself. The project grants the student 15 ECTS and is to be completed with a group of two to four students. The faculty provides students with several assignments to choose from, although it is also possible to arrange for an internship yourself, at a Dutch company, or even abroad.

This thesis is a report of the BSc project of Irene Renkens and Timothy Kol, which was completed as an internship in the country of Singapore. The internship was offered by M3 Interactive, a joint venture of Singaporean gaming company Nexgen Studio, and the Malaysian company M3 Technologies.

We'd like to thank Cabal Entertainment Software's CEO Danien Chee for passing on our application to Nexgen Studio, and Jeffrey Jiang, director and co-founder of Touch Dimensions, another Singaporean gaming company that works closely with Nexgen Studio, for ensuring us a position at Nexgen and guiding us throughout the project. We'd also like to thank Alvin Yap, Nexgen's CEO, who gave us this opportunity to both gain experience in the gaming industry and get to know Singapore. Furthermore, we'd like to thank Yuchao Cheng and William Koh for their graphical support, Adi Wibowo for his programming insight and all other Nexgen employees as well for their kindness and support.

Closer to home, we'd also like to thank Peter van Nieuwenhuizen for being the BSc project supervisor for all Media & Knowledge Engineering students, as well as being our mentor through our project. We also thank international internship officer Jan de Vries, who helped us in arranging a scholarship for our project abroad and gave us useful general information. Finally, we thank Gerwin de Haan for being part of the project's grading committee.

Irene Renkens
Timothy Kol

Delft, August 29, 2011

ii

Summary

When we left for Singapore, the assignment we would be working on during our internship was still rather vague. We knew the company was developing some game that implemented voice recognition, and that we were to be part of the team developing this game. Upon arrival we found out that the game was primarily educative, and to prevent it from becoming too boring, several mini games had to be implemented as well. It was our task to develop these mini games, starting from scratch, except for some vague concepts that were already worked out.

We quickly decided to apply agile programming during this project, having a preference for being able to make changes *after* the design phase the game as well, by means of a new iteration. We worked out a schedule, from which, as is usually the case, we deviated quite a bit. In the end, we did do the amount of work we expected to, it was just distributed differently, due to company deadlines and changing priorities. In the end, we managed to develop three mini games, one of which is completely worked out and ready to be implemented in the final product, while the other two are prototypes.

Although all three mini games are separate from each other, they do use a lot of the same code. For example, they all use a very similar game loop to update variables and re-render the graphics, using the same base game states. The separate voice recognition class is also used in the same fashion. These similarities would make it quite easy to create a new mini game out of the existing code.

During implementation, we did stumble upon several difficulties. At first, we had to create a game loop. We first tried to achieve this using a storyboard, which didn't turn out very well, as the loop would stop after some time. We then decided to implement a game loop using the event that is fired every time a new frame is rendered.

Another issue that came up was the lag that started to appear as we started to use higher resolution images. Noticing a short delay when new images were created, we soon figured that this was the cause of the lag. We solved this problem by pre-loading all needed images, setting them to an invisible state. This caused some loading time when the game was first started, but it made the game itself a lot smoother, which was worth it.

Another problem was ensuring a correct, synchronous 3D animation in a 2D environment for two objects. Tweaking variables and adding in some calculations, we eventually found an animation that seemed correct to the human eye, and was also mathematically correct in most ways. We synchronized both objects by simply using the same transformations, as the animation was too complicated for the objects to start half way the animation, for example.

In the end, the result was satisfactory for both ourselves and the company. We had fulfilled all our own must have and should have requirements, except for one, which was discarded with a good reason. The company was also pleased with the polished mini game we completed, and will implement it in the final game.

Contents

1

Introduction

Although currently a very small industry, Singapore's gaming market is slowly but steadily emerging, benefiting from the government's support[1] as well as the country's unique situation in the same way as many other, further developed Singaporean industries already are. While most game developers are still employed in countries such as the United States, Japan, Canada and the United Kingdom[2], the continuous growth of mobile applications[3] gives smaller companies an opportunity to develop and distribute their own projects, without the need for a big investment, as is usually the case with big console or PC games. Nexgen Studio – the company that employed us – and Touch Dimensions – our Singaporean mentor's company – are two of these small-scale companies, having focused mainly on smaller but innovative games and mobile applications.

Singapore is a unique country; it is highly developed, English is a main language, and it is located in the middle of Asia, making the country a bit of a hybrid between the Asian and Western cultures. This makes it easy for Singaporean companies to maintain relations with both Western and Asian foreign business partners. The project we have been working on is a perfect example of this: it is targeting the Chinese market, and is being developed with the help and support of several Westerners.

The goal of the project was to develop one or more mini games for an educational game, which teaches native Mandarin (the main Chinese language) speakers correct English pronunciation. To accomplish this, the game utilizes Microsoft's built-in speech recognition for Windows 7. Our job was to make fun mini games, which reiterate certain words or phrases the player has already studied in the main game.

The game is officially being developed by M3 Interactive, which is just a joint venture of Nexgen Studio and M3 Technologies, a big listed Malaysian company. M3 Interactive is working closely with Touch Dimensions during this project. In practice, most work is done by Nexgen Studio's and Touch Dimensions' employees, while M3 Technologies focuses on investing and decision making, which gave the whole structure the idea of M3 Technologies simply assigning Nexgen to make a game, basically making M3 Technologies a client of Nexgen. For some more information on the companies, please see Appendix A.

This report will focus on the development of the mini games, and is split up in six chapters. Chapter two will discuss the problem the company presented us with, as well as our analysis. It will contain a more detailed description of the assignment, and a summary of the tools we used in completing our task, including an argumentation, if applicable. Furthermore, this chapter will discuss the software development technique we've used, agile programming, and our initial reasons for this choice. Finally, the project's plan will be discussed, starting with our initial schedule and ending with how we finally distributed our time, discussing the deviations from our original plan and their causes.

Chapter three will mainly focus on the design of the mini games, discussing how the schedules of requirements were constructed and the mini games' class architecture, including several diagrams and

describing design changes that were made after iterations. The chapter will then continue with the similarities between the games, and which code we reused and why, which also shows how new mini games could be developed with a head start, recycling much of the code that we've produced. Finally, the state machines of the mini games will be discussed in the final section of this chapter.

Chapter four will discuss the actual implementation of the mini games, starting off with discussing the problems, changes and trade-offs that were encountered during implementation. This section includes the changes that were made during implementation and an argumentation of these changes, and choices that we've made when faced with certain trade-offs or technical difficulties. The last section will discuss the feedback we received from the Software Improvement Group and how we processed this feedback.

Chapter five will discuss the results and recommendations by reviewing our original schedules of requirements compared to the finished products. Furthermore, it will contain a section about recommendations to improve on our work and use it for future mini games. Finally, this chapter will discuss our looking back upon the project and what we would have done differently.

The final chapter, chapter six, will consist of our conclusion on both the final delivered product, the project in general, and the experience of actually working in the Singaporean gaming industry.

2

Problem and analysis

This chapter will describe the assignment and our analysis of this problem. Furthermore, we will discuss the tools used in developing this project, including an argumentation, while the next section will explain the software development technique that was applied during the project, present the schedule and discuss the necessary changes we made to come to our final time distribution.

2.1 Description of the problem

The game that our project would be part of was still in an early stage when we started working, although the design for a prototype version was already present. This design also included ideas for the mini games that we were to develop, although the company indicated that we were free to come up with our own ideas and the design was very much prone to change, due to the Malaysian company being the main investor having the final say.

Nonetheless, the main idea of the game remained unchanged: an educative game that taught Mandarin speakers to improve their English pronunciation. This is achieved using multiple scenarios, to teach pronunciation that is relevant in different situations. These scenarios provide the user with a specific environment and corresponding words, like being on a plane and having a standard conversation with a stewardess. During these scenarios, speech recognition allows the game to see how well the user can pronounce the English words and sentences, and is also used to give feedback to the players, to allow them to see what they still need to improve on.

The problem that we were presented with, was to design, implement and test some mini games that would be fun and exciting to play, and would recapture the words and sentences that were already discussed in the scenarios in the main game.

2.2 Tools

The mini games were to be integrated in the game itself, which led us to use the same tools that were already being used by the rest of the team. The game was being written in C#, using *Windows Presentation Foundation*, WPF, as a graphical subsystem. WPF is used to easily construct user interfaces using XAML, a mark-up language that is heavily based on XML. The company wanted us to use *Microsoft Visual Studio 2010* - which includes the WPF SDK - to ensure compatibility with the already existing prototype. In addition, we used *Microsoft Expression Blend 4*, a utility with similar functionality to Visual Studio, but with more focus on the graphical aspect. Blend was only used for certain graphical implementations, while most programming and testing work was done in Visual

Studio. Besides the fact that the company presented us with Blend, another advantage was that it is highly compatible with Visual Studio, allowing you to freely edit project files, classes and XAML files in both applications.

Rarely resorting to pair programming, we also needed a good solution for synchronizing the files. The company presented us with *Team Foundation Server*, a source control application by Microsoft that is similar to subversion. Because the server was already running with the game's prototype, it was easier for us to use this as well to connect our source code to each other and, later on, to the game itself.

Class diagrams were made using *astah* community*, a free tool for creating UML diagrams. We chose this application from a list of options and were satisfied with its ease of use, and the option to easily export UML data as an image.

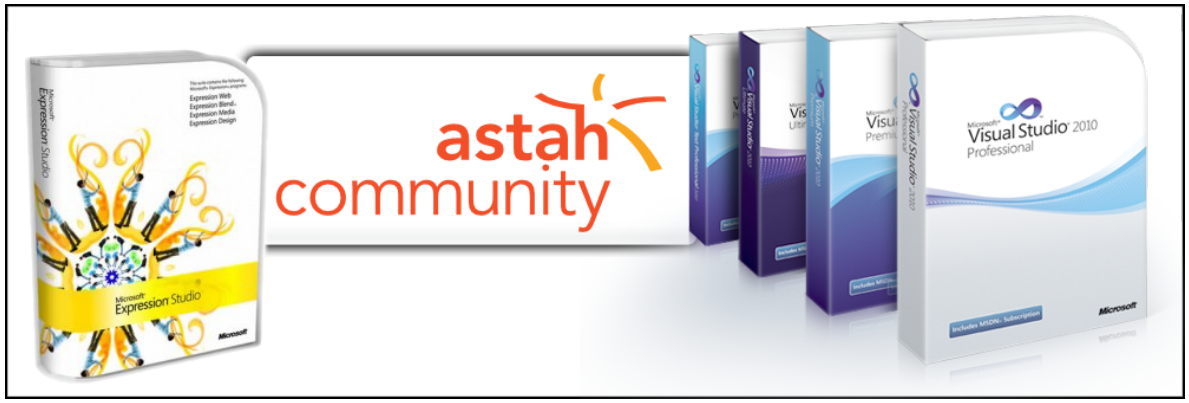


Figure 2.1: The most important tools were Visual Studio, Expression Blend and astah* community

2.3 The approach, the plan and deviations from it

The main voice recognition project is a hybrid of the waterfall and agile methodology, while for the mini games we quickly decided upon implementing it using agile development. This would ensure a working version each iteration, which is desirable given the fact that the assignment consists of multiple mini games. At first, the iterations would focus on constructing working, but basic prototype versions of the mini games, while later on, new features could be built to improve the entertainment aspect.

Every iteration, we decided to work through a full development cycle, taking into account the previous iteration.

Although we did keep up with the agile methodology, we did not go through as much real iterations as originally planned, which will become clear from the original schedule and how we ended up distributing our time. This was due to both the company having other plans, and small improvements and additions that we kept on making, where we would of course update the class diagrams and unit tests, but where we didn't see the need to work through a whole new cycle. In the end, these small updates piled up to create a much more advanced version than was originally determined in the schedule of requirements, although it did cause us to make fewer development cycles.

Table 2.1 shows a copy of our first version of our schedule, which was constructed on May 6.

#	Date	Iteration description
-	25 Apr – 6 May	No iteration, orientation phase, constructing a prototype version of the *Whack a Mole* mini game and making a plan of approach and report on the orientation phase
1	9 May – 13 May	Iteration of implementing *Whack a Mole*, this time using a full software development cycle while also adding new functionality
2	16 May – 20 May	Iteration of developing a prototype of *Falling Words*
3	23 May – 27 May	Iteration of implementing additional functionality to *Whack a Mole*
4	30 May – 10 Jun	Iteration of developing our own mini game
5	13 Jun – 17 Jun	Iteration of implementing additional functionality to *Falling Words*
6	20 Jun – 24 Jun	Iteration of implementing additional functionality to our own mini game
7	27 Jun – 1 Jul	Iteration of implementing additional functionality to *Whack a Mole*
-	4 Jul – 8 Jul	No iteration, final testing of created mini games as well as implementation in voice recognition project

Table 2.1: Schedule on May 6, # denotes the iteration number

We started the project getting familiar with WPF, but because the company was eager for results during the first two weeks, we decided to practice our skills with WPF as we went by developing a prototype for *Whack a Mole*, one of the company's original mini game ideas, in which you must whack moles that pop out of the ground by pronouncing the word they are holding. This did turn out to be a good way to get into the programming, although we did not have enough time left to write a detailed orientation report, which was the first deviation from the plan.

On the long term, we were aiming to create prototypes for a total of three mini games, while also implementing additional functionality to them during a later iteration. These mini games were to be *Whack a Mole*, *Falling Words*, another one of the company's original ideas, and a mini game we would have thought up ourselves. During the last week, we would do some final testing and integrate the mini games in the final project.

The first iteration, where we would implement *Whack a Mole* by going through a full software development cycle, did succeed, although we were not able to add much new functionality. During the second iteration, however, we found out we could not complete the full development cycle in one week, partly because we were also implementing some of the company's feedback into *Whack a Mole*. We deviated from the plan here because there was a deadline for the first prototype, and we wanted to deliver something the company would like. In the end, we decided to revise the plan to only develop two mini games, so we would have enough time to add quite a bit of exciting functionality to both. This led to a new plan, which was constructed around May 25 and can be seen in Table 2.2.

#	Date	Iteration description
-	25 Apr – 6 May	No iteration, orientation phase, constructing a prototype version of the *Whack a Mole* mini game and making a plan of approach and report on the orientation phase
1	9 May – 13 May	Iteration of implementing *Whack a Mole*, this time using a full software development cycle while also adding new functionality
2	16 May – 03 Jun	Iteration of developing a prototype of *Falling Words*
3	30 May – 10 Jun	Iteration of implementing additional functionality to *Whack a Mole*
4	13 Jun – 17 Jun	Iteration of implementing additional functionality to *Falling Words*
5	20 Jun – 24 Jun	Iteration of implementing additional functionality to *Whack a Mole*
6	27 Jun – 1 Jul	Iteration of implementing additional functionality to *Falling Words*
-	4 Jul – 8 Jul	No iteration, final testing of created mini games as well as implementation in voice recognition project

Table 2.2: Schedule on May 25

However, later that week, during a meeting, Nexgen Studio let us know that M3 Technologies, the company that made the final decisions about the game, thought that *Whack a Mole* was too static, and that they wanted something more exciting to be included in the first prototype version of the full project. After several brainstorm sessions, we decided upon the so-called *Toy Factory*, a new mini game that was partly our idea as well. In this game, the player would look at a conveyor belt from the front, in a 3D fashion. Toys holding signs with words on them would then fall on the belt and start moving forward, towards the screen. The player would then have to pronounce the word before the toys smash into their screen. Although from the outside, the game seemed quite different, we purposely picked an option that wouldn't require big changes to the underlying structure.

Because the company gave *Toy Factory* the highest priority, we abandoned the half-finished *Falling Words* prototype for now, and revised our plan again, which led to a new schedule, which was constructed around May 30, as can be seen in Table 2.3.

#	Date	Iteration description
-	25 Apr – 6 May	No iteration, orientation phase, constructing a prototype version of the *Whack a Mole* mini game and making a plan of approach and report on the orientation phase
1	9 May – 26 May	Iteration of implementing *Whack a Mole*, this time using a full software development cycle while also adding new functionality
2	27 May – 03 Jun	Iteration of developing a prototype of *Falling Words* (incomplete)
3	30 May – 10 Jun	Iteration of developing a prototype of *Toy Factory*
4	13 Jun – 17 Jun	Iteration of implementing additional functionality to *Toy Factory*
-	20 Jun – 24 Jun	Finishing the *Falling Words* prototype as well as writing unit tests for *Toy Factory*
5	27 Jun – 1 Jul	Iteration of implementing additional functionality to *Falling Words*
-	4 Jul – 8 Jul	Writing unit tests for *Falling Words*

Table 2.3: Schedule on May 30

Working on the prototype for *Toy Factory*, we got a little carried away during the implementation phase, adding more and more functionality that wasn't in the original requirements, basically merging iterations 3 and 4. By June 13, we got informed that the first prototype of the full project was due by the end of June, and *Toy Factory* was to be included. We did not worry, as the mini game was already in an advanced stage, but in the end we may have spent too much time on it, adding more functionality, fixing small bugs and adding extensive unit tests. Because of this, *Falling Words* was still not finished by June 24.

During the next week, the company was moving to a new office location, where a delay with the internet connection cost us some time in starting work again. In the end, we did add some additional functionality to *Falling Words*, but being more concerned with *Toy Factory*, we cleaned up its code and implemented some final feedback from SIG, while *Falling Words* remained incomplete. This led to our final time distribution, which is displayed in Table 2.4.

#	Date	Iteration description
-	25 Apr – 6 May	No iteration, orientation phase, constructing a prototype version of the *Whack a Mole* mini game and making a plan of approach and report on the orientation phase
1	9 May – 26 May	Iteration of implementing *Whack a Mole*, this time using a full software development cycle while also adding new functionality
2	27 May – 03 Jun	Iteration of developing a prototype of *Falling Words* (incomplete)
3	30 May – 10 Jun	Iteration of developing a prototype of *Toy Factory*
-	13 Jun – 17 Jun	Implementing additional functionality to *Toy Factory*
-	20 Jun – 24 Jun	Implementing additional functionality to *Toy Factory* as well as writing unit tests for *Toy Factory*
-	27 Jun – 1 Jul	Implementing additional functionality to *Toy Factory* as well as fixing small bugs and writing unit tests for *Toy Factory*
-	4 Jul – 8 Jul	Implementing additional functionality to *Falling Words* as well as cleaning up code for *Toy Factory*

Table 2.4: Final schedule

While the tables show some important deviations from the first schedule, another change in our original plan was the use of unit tests. Originally, we wanted to add these every iteration, making it easier to check the correctness of a new iteration. In the end, the unit tests were implemented at a much later stage, which negated this anticipated advantage. However, the tests did help us detect and solve several bugs, and although we did not implement any full iterations afterwards, we did reap the tests' benefits while making changes to the code later on, with the unit tests ensuring correct behaviour for all methods. The original plan of performing acceptance tests was not applied either, although both us and the company played and tested the mini games to a certain extent, eventually leading to a dismissal of the *Whack a Mole* concept, introducing the more dynamic *Toy Factory*.

From all this, it can be concluded that we did deviate a lot from the original plan, although we did manage to deliver a total of three mini games. However, because we did not finish an iteration in the beginning, one mini game remained incomplete, while another one was very polished and ready to be included in the main game.

3

Design

In this chapter we will provide the design documents and discuss the design decisions we have made during the course of the project. The project involved the development of several mini games, which had to be made in 2D due to the company wishes and WPF's limitations. The first mini game we worked on, *Whack a Mole*, had a simple concept: the game consisted of hitting moles appearing from their hole with a hammer by saying the word they are holding. This concept was already decided upon when we arrived at the company.

During a meeting with the client, M3 Technologies, a prototype of the *Whack a Mole* game was shown. The client decided they did not like the prototype enough, as they found it lacking in animation. After some discussions and brainstorm sessions with our colleagues at the company, it was decided and agreed upon to create a new concept: *Toy Factory*. Right after the decision was made, all development on *Whack a Mole* game was cancelled, and the development of the *Toy Factory* game began. When the prototype was approved by the client, the game was developed further into a complete product that would be integrated in the final game. All of this naturally led to some design changes during the process.

Besides *Whack a Mole* and *Toy Factory*, there had also been made a design and prototype for a mini game called *Falling Words*. Its concept was that the player has to catch items that are dropped from words in a sentence, if these words have been correctly pronounced by the player. As with *Whack a Mole*, the concept already existed prior to our arrival. The design of the game was made and part of the implementation was done while the *Whack a Mole* prototype was waiting for approval. Quite a bit of the actual implementation of the prototype, albeit incomplete as our internship came to an end, was only done after *Toy Factory* was completely finished.

For each of these mini games that have been (partially) developed, several design documents were created such as a schedule of requirements, UML diagrams and state machines.

3.1 The *MoSCoW* model

Prior to creating any UML diagrams, state diagrams or other design tools, a schedule of requirements is usually composed. This schedule is intended to list the functional requirements for the intended implementation as well as the requirements, usability and performance for the system that will be running said implementation.

For the composition of the schedule of requirements, the *MoSCoW* model is used. This model is a prioritization technique used to specify the importance of the requirements. The requirements can be classified under one of four categories. These are, from high to low priority: *must have this*, *should have this if at all possible*, *could have this if it does not affect anything else* and *won't have this but*

would like to have this in the future[4].

During the process of prototyping and polishing the mini games, the requirements have undergone some changes. Most changes were a result of the addition of extra functionality to the mini games. The final schedule of requirements for *Whack a Mole*, *Toy Factory* and *Falling Words* can be found in Appendix D.

3.2 Similarities between the games

Although the games all have a different concept, there are also a few things that they have in common. The primary commonalities are the game loop and the recognizer class they all use.

3.2.1 Game loop

All mini games are designed to make use of a so-called game loop, which is situated in the Main class of the mini game. The game loop serves as an update cycle, and is called multiple times a second. The game loop should contain all logic concerning the updating of game objects. For example, animations in the mini games can be created by changing the x- and y-positions or by applying transformations to the visual object. When this logic is placed inside the game loop, which is called every time a new frame is rendered, these changes are immediately visible. The continuous re-rendering of the visual objects results in the animations.

Non-visible data such as the user input, in this case speech, is also handled in the game loop. For example, the speech recognizer is updated in the game loop, which ensures that after recognition, other words can be detected.

In short, we can state that the game loop will manage and handle the input and data which need to be continuously updated and revised.

3.2.2 The recognizer class

All the games have a class called *Recognizer*. This class contains the recognizer to detect the speech input from the user. In this class, a so-called grammar is created, which is needed in order to tell the recognizer which words he can detect. With use of the grammar, the recognizer can choose from the specified words when detecting sound, and see how well the input matched the standard speech profiles that correspond with the pre-set words. Whenever a word is detected, an event is fired and data related to the recognized word is saved. This data can then be retrieved by other classes.

3.3 Class architecture

Before the actual implementation of the mini games, *Unified Modeling Language*, or UML diagrams were designed. The diagrams serve the purpose of showing a better idea on how the different classes relate to each other, as well as making us think beforehand about how to structure the final implementation. Since the project involved creating small games, hence the name mini games, it was decided to keep all classes in the same package. In our opinion, the number of different classes was small enough to keep them together, while splitting them in different packages would most likely result in packages consisting of only one class.

3.3.1 *Whack a Mole*

Figure 3.1 shows the class diagram of the mini game *Whack a Mole*. The first prototype of the game was created following this design. After the first prototype, the game was extended for a bit by adding

extra functionality to the game play. This led to a revised version of the class diagram. The new diagram can be seen in Figure 3.2. Note that to improve readability, not all attributes are listed.

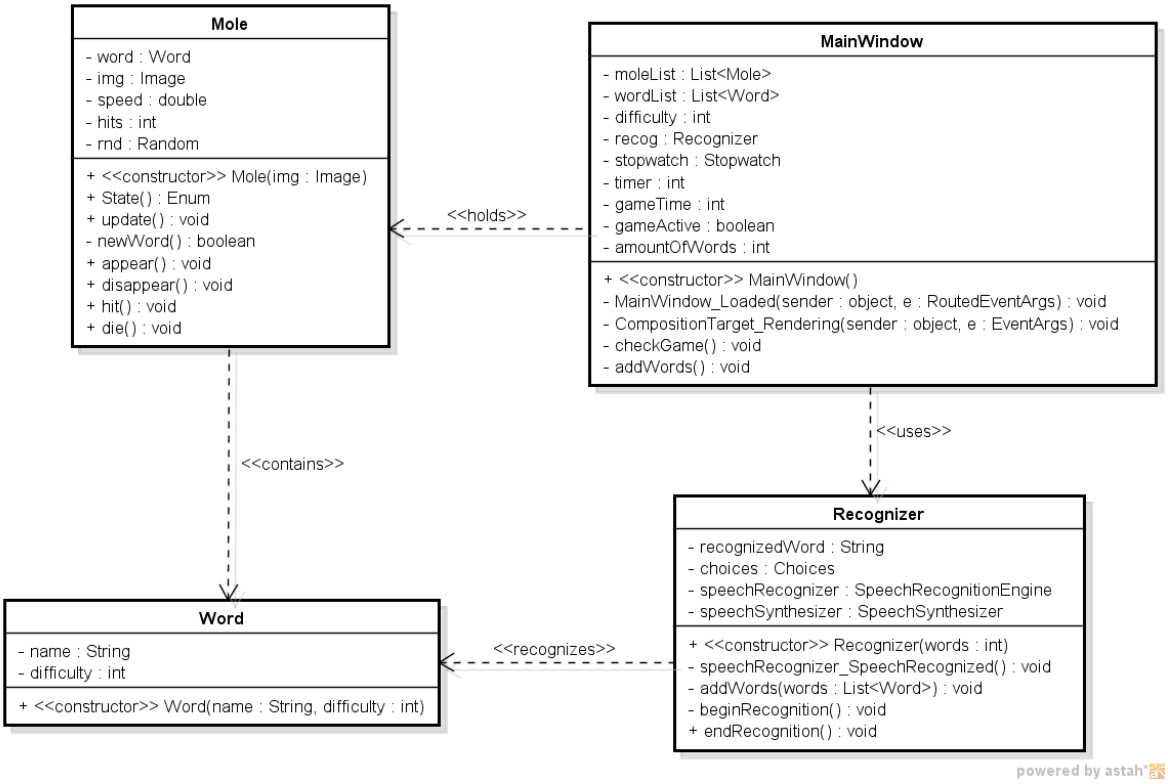


Figure 3.1: Original class diagram of *Whack a Mole*

The diagram in Figure 3.1 consists of four classes: *MainWindow*, *Mole*, *Recognizer* and *Word*. In *MainWindow*, a game loop is implemented which contains logic to update all visual objects in the game. This class also handles all data related to the overall game such as sounds, starting and ending the game. The *MainWindow* class makes use of a speech recognizer, the Recognizer class.

The *Recognizer* class contains a method to create a grammar consisting of words, or *Word* objects. *Word*s are simple objects consisting of a String representing the *Word*'s name. The grammar can then be used to determine which words need to be recognized. The *Recognizer* also contains methods to begin or end voice recognition and most importantly, it is able to recognize speech. When the user's speech input is recognized as one of the words in the grammar, an event is fired and the recognized word and its quality are saved in the class. The saved data can be retrieved and used by *MainWindow* with the help of C# properties.

The *MainWindow* class holds a list of *Mole* objects. These are moving objects in the game and hold a word for the player to pronounce. The *Mole* class contains the visual objects to represent a mole and the word it's holding. It also contains logic to assign a *Word* to it and execute an animation corresponding to its current state. For example, when the state of a mole is *appearing* the mole is animated to come out of his hole and when the state is *hit* there will be an animation of a hammer hitting the mole.

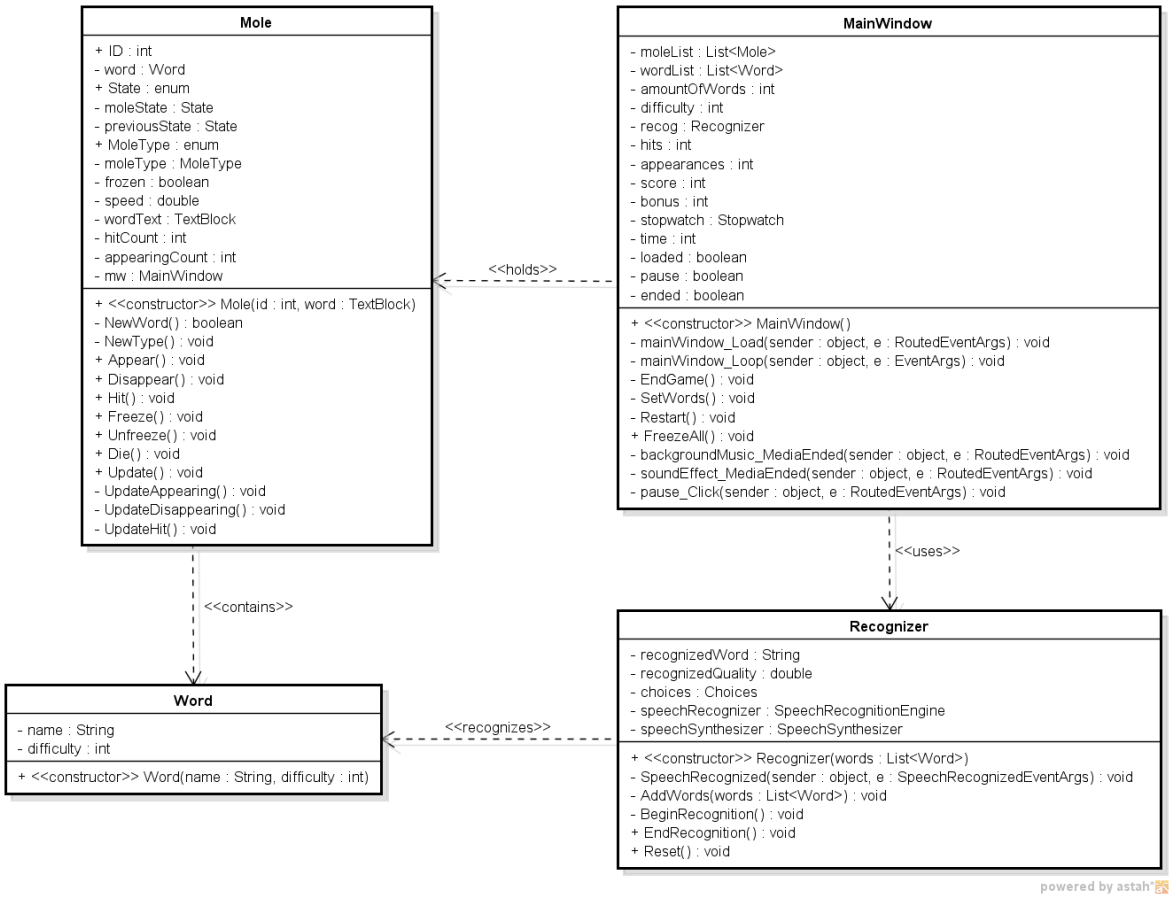


Figure 3.2: Revised class diagram of *Whack a Mole*

3.3.2 *Toy Factory*

The class diagram initially created for the mini game *Toy Factory*, displayed in Figure 3.3, resembles the class diagram of the mini game *Whack a Mole* quite a lot. The *Recognizer* and *Word* classes are exactly the same, while the only difference between the *Main* class in *Toy Factory* and the *MainWindow* class in *Whack a Mole*, is added functionality. Furthermore, the *Mole* class has changed to a class called *Toy*, but the class is still used in the same manner. However, *Toy Factory* had gotten the green light from the client to be used in the final product, whereas *Whack a Mole* had not. This meant that is was to be developed even further, leading to several design changes that had to be made. These deviations have also altered the class diagram.

Figure 3.4 shows the enhanced class diagram. It can be seen that additional classes have been added to the diagram. A class called *ResourceManager* has been added, which reads an XML property file and loads all resources upon starting, in order to improve the game's performance. The *ConveyorLine*s of the conveyor belt have also gotten a separate class, as they serve as a way to animate the belt and require quite a bit of logic. To keep the code clean and structured, this logic has been put in a separate class. Furthermore, the factory design pattern has been implemented. The class *TFactory* will have the responsibility of creating *Toy*s, so the *Main* class won't have to worry about this any more. If any

changes should be made with respect to the creation of *Toy*s, the factory pattern will make it easier to make any changes to the code.

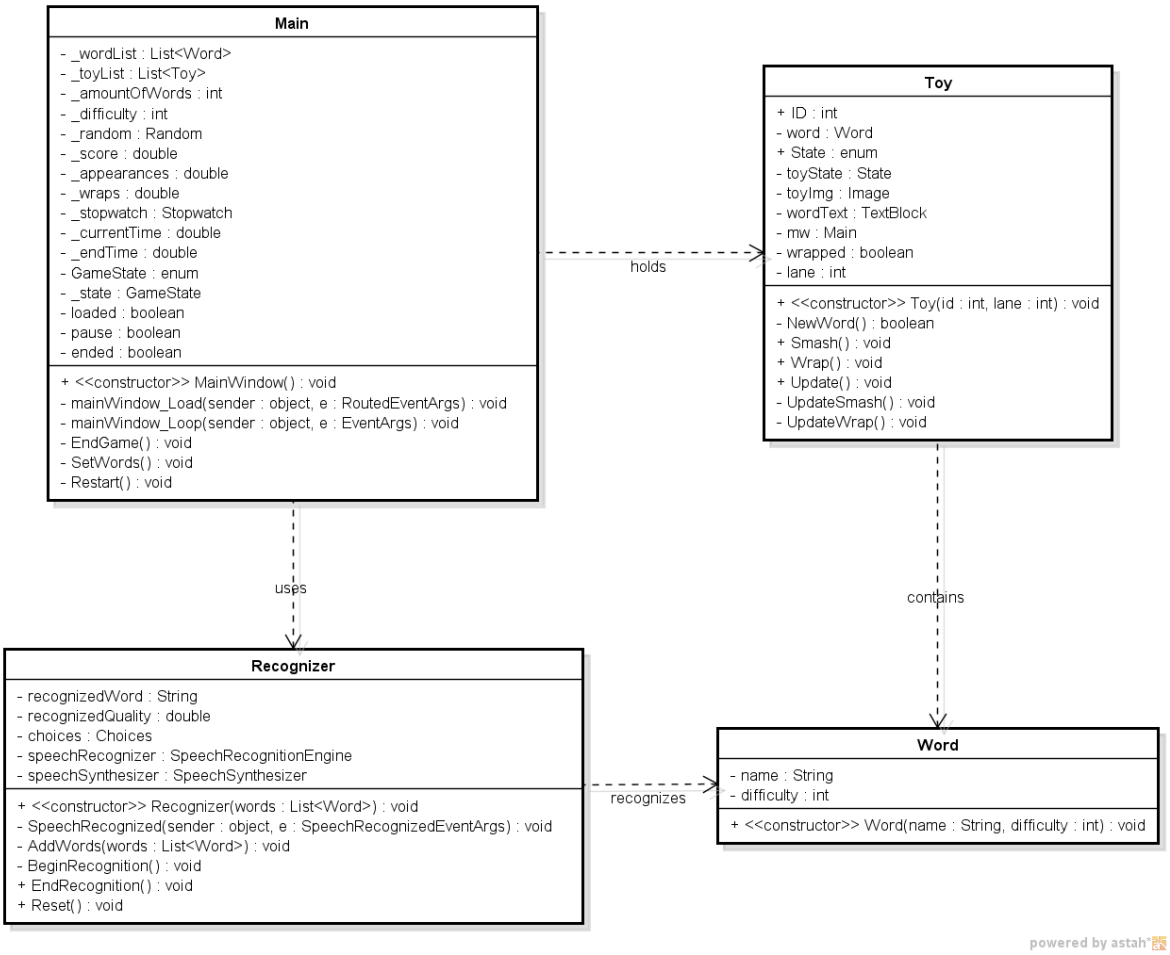


Figure 3.3: Original class diagram of *Toy Factory*

Figure 3.4: Revised class diagram of *Toy Factory*

3.3.3 *Falling Words*

Since the main focus of the project ended up being the *Toy Factory* mini game, *Falling Words* has only been prototyped. This prototype has been made using the class diagram in Figure 3.5. It's seen that the construction of this UML diagram differs a bit from the two other games. This is because this game also uses the user's input from the keyboard. The *Avatar* class represents a movable object in the game that the user is controlling. With the avatar, the user can catch items that have been dropped when words or phrases are correctly pronounced.

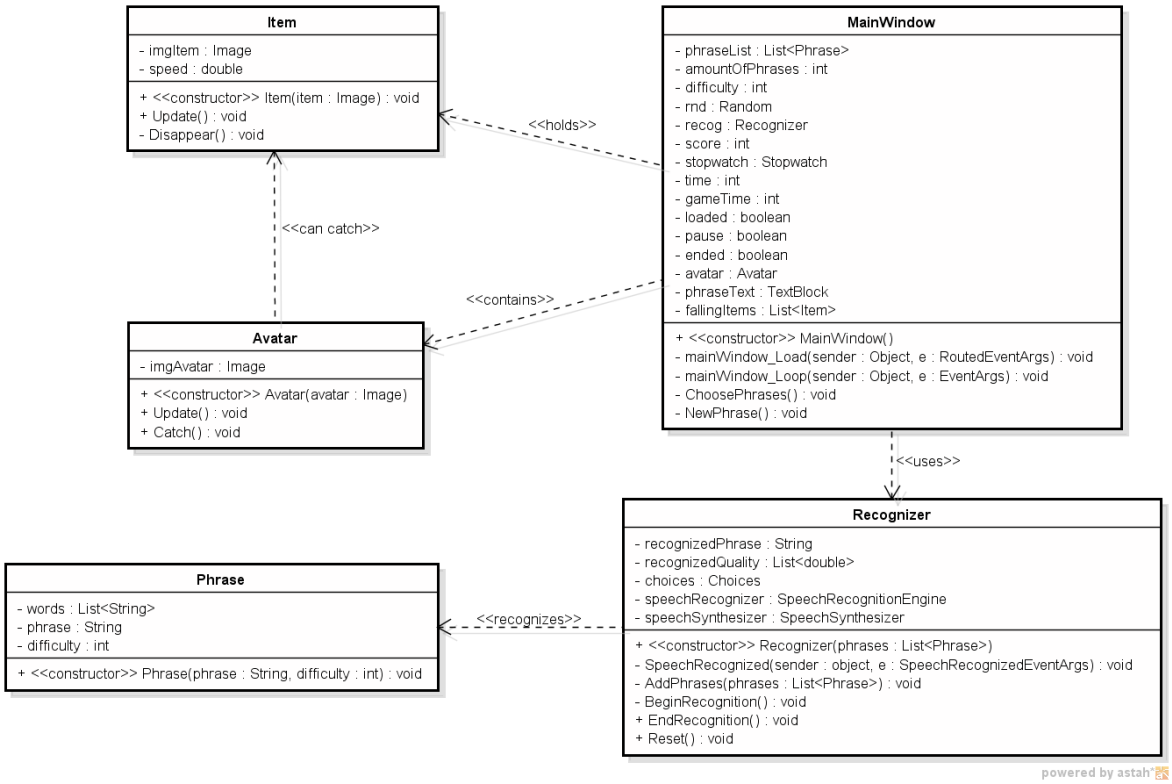


Figure 3.5: Class diagram of *Falling Words*

3.4 State machines

Since the objects involved in the games have to behave in a certain way, it was decided to make use of the state design pattern and apply this to all mini games. With the use of states, it is easier to implement state specific logic. For example, *Mole* classes in the *Whack a Mole* game can have the states *appearing*, *disappearing*, *hit* or *hidden*, and *Toy*s in the *Toy Factory* game can have the states *onBelt*, *smashed*, *frozen*, etc.

Not only for in-game objects these states can be defined, the game as a whole can be easily divided in several states as well, such as a playing state and paused state. At first things were kept simple with a state diagram that is displayed in Figure 3.6. The game consisted of a *Start*, *Play*, *Pause* and an *End* state, the most basic states needed.

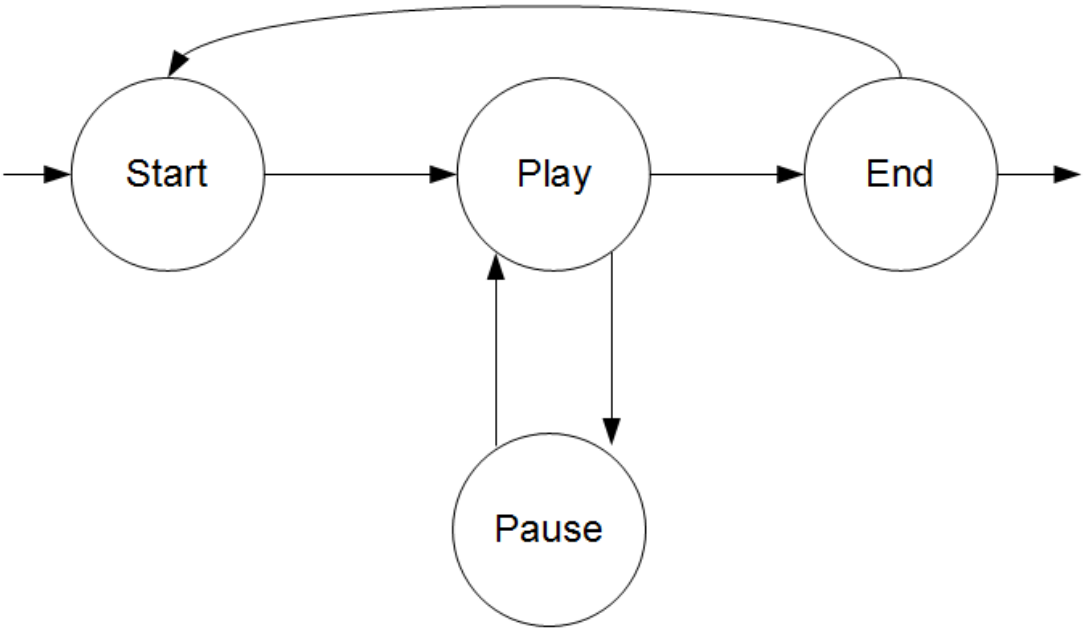


Figure 3.6: Original state machine

As the development of a final mini game progressed, a more expanded version of the state machine was needed. This state machine can be seen in Figure 3.7.

It can be seen that a few extra states have been added. The *Start* state has been replaced by the *Starting* and *Started* states. In the *Starting* state, the game will load all its resources and go to the *Started* state when this has been done. The *Started* state can be regarded as a still state, the game instructions will be shown and nothing happens until the user presses the start button.

To improve the visual aspect of the overall game, the panel that shows the instructions and final score at the end, has been made a sliding panel, which slides out of your screen when the play button is pressed at the beginning, and slides back into the screen when the game is over. Because of this screen's movement, another state was introduced: the *Screen moving* state. When the game is in this state, the sliding panel is still moving. When it's done moving, the state will be changed to the *Play* or *End* state, depending on whether the panel is moving into or out of your screen.

The *Play* state is obviously active when the player is playing. When the player presses the pause button during the game, the state switches to the *Pause* state and the game pauses. The game will only resume when the resume button is hit, which changes the state to *Play* again. When the game has ended, the *End* state will be active. If the player decides he wants to play the game again the state of the game changes to *Resetting*. While in this state, all variables will be reset if necessary, and the state will change to the *Started* state again when resetting is done and the game is ready to be played again.

To recap, the game starts up and loads its resources (*Starting*). It then shows the player an instruction screen (*Started*). When the player clicks the start button, the instruction screen will move up and disappears (*Screen moving*). The game then starts (*Play*). The user has the option to pause the game at any time (*Pause*), only to resume it afterwards. When the game has finished, a game over screen moves down into the screen (*Screen moving*). When the screen is done moving, the player is game over and sees its final results (*End*). The player then has the option to play the game again, resulting in the variables being cleared and reset (*Resetting*), and the game will then be ready to be played again (*Started*).

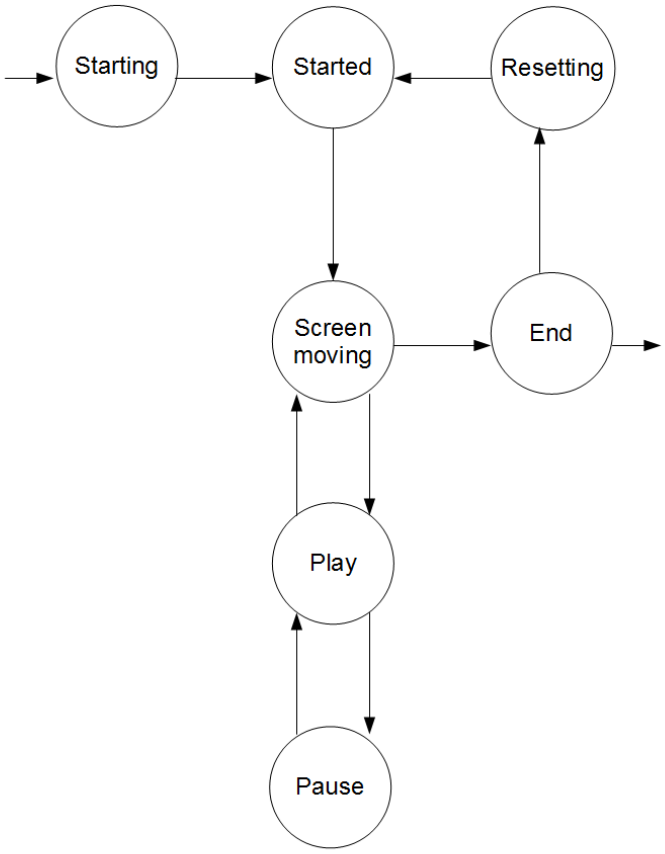


Figure 3.7: Extended state machine for *Toy Factory*

4

Implementation

In this chapter, we will mainly explain and review the implementation of the *Toy Factory* mini game, as this is the only game that was fully developed and finished to be integrated in the end product. This chapter will discuss what problems arose during the process, and how these were solved. It also reflects on changes that were made, trade-offs that were faced and problems that were encountered. Of course, while focusing on the *Toy Factory* mini game, some problems and trade-offs were very general and do not merely apply to *Toy Factory*, but also to the other two mini games or any WPF project in general.

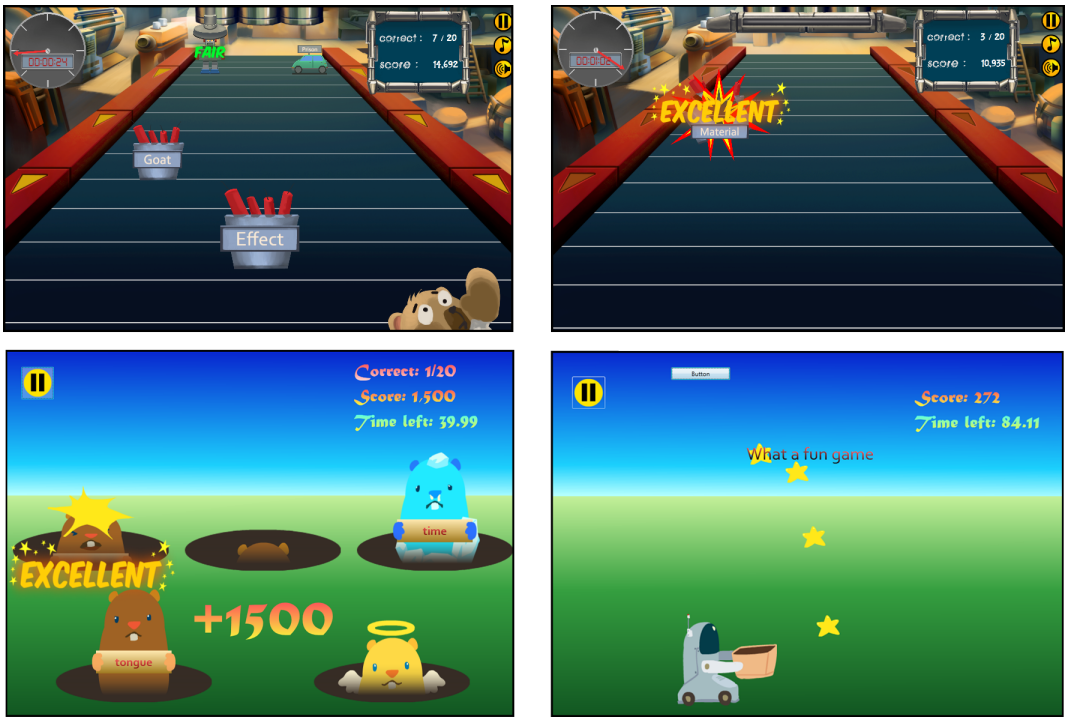


Figure 4.1: Screenshots of implemented mini games: *Toy Factory*, *Whack a Mole* and *Falling Words*

4.1 Problems, changes and trade-offs

As with almost all software implementations, we also experienced some problems during the implementation phase which sometimes led to needed changes or let us face trade-offs.

4.1.1 WPF Related

The mini games were to be created in Microsoft's Windows Presentation Foundation (WPF), as this was a requirement by the company. During development we ran into several issues related to WPF. They will be explained in this section.

Game loop

Because a game loop was needed in the game, a way to implement one had to be found. Our project supervisor sent us a link[5] to a website where a game loop method was described. We tried it out, but discovered that it wasn't working properly in WPF. It turned out this method is meant for Silverlight, but does not function in WPF. Since Silverlight can be considered as a subset of WPF, we found this a little strange, but sadly couldn't do anything about it.

Research then brought us two different methods to consider for the game loop, making use of the *DispatchTimer* or the rendering event. Researching both options, the *DispatchTimer* quickly was dismissed, as it turns out to be less accurate and stable than the rendering event method, which works as follows. Before the rendering of each frame a rendering event is fired, which we modified to make a call to our *mainWindow_Loop()* method, which in turn handles the required logic to create a successful game loop.

Data binding

WPF makes use of a declarative mark-up language called XAML. This language is meant to simplify the creation of an application's user interface or UI. The UI elements can be created in an XAML file, and this file will be joined to a code-behind file, where the run-time logic can be specified. Using XAML, the UI objects' declarations and their underlying logic can be separated[6].

To establish a connection between an object in the XAML file and the logic in the code-behind file, so-called *data binding* can be used. With data binding, one can simply *bind* data in the code-behind file to an object in the XAML file. When the data changes its value, the element(s) bound to this data will immediately change accordingly.

We've considered using data binding to reflect the changes to the visual elements. After some experimenting, we decided not to use it because in most cases the changes we wanted the visual elements to reflect could not be made using this method.

Storyboards

WPF provides a simple way for animating objects by introducing storyboards. A storyboard is a type of timeline that provides information about the objects that are defined in the XAML file and are animated along this timeline. With the use of storyboards, objects can be moved, changed in size or color, etc., which results in an animation[7]. And example storyboard is shown in Figure 4.2.

The storyboard in Figure 4.2 animates an existing rectangle named *MyRectangle*. The size of the rectangle increases from 100 pixels to 200 pixels in a time span of one second.

With the use of a program like Blend, an animation can be very easily created by just changing the object the way you want, and saving the changes to the timeline of a storyboard. Blend will then auto-generate the XAML code. Since a lot of animations are involved in the mini games, storyboards could be very useful and save a lot of time on implementing animations.

```
<DoubleAnimation Storyboard.TargetName="MyRectangle"
     Storyboard.TargetProperty="Width"
     From="100" To="200" Duration="0:0:1" />
```

Figure 4.2: An example on defining a storyboard in XAML

The problem with storyboards, is the fact that the duration of the animations has to be provided. The animation we want in these particular mini games however, don't have a pre-defined duration, because the speed of the objects that need to be animated differ from each other. This means that the duration of the storyboard cannot be set to a standard. Data binding might provide a solution to this, but after experimenting for a bit, there doesn't seem to be a way to bind data to the storyboard duration due to its format. Because the use of storyboards didn't seem possible, all animations were done in the code-behind files, which also had the advantage of having full control over the animations.

Resource manager

A problem that was discovered during the implementation, was WPF lack of speed when it came to resources. When a new image was created, a short lag was experienced. To reduce this lagging, a resource manager was introduced. This resource manager loads all images used in the game before it starts, which decreases the lag drastically. To make this more visually attractive, we introduced a loading screen, which can be seen in Figure 4.3.

Figure 4.3: The game is being loaded

4.1.2 Game related

Besides any issues related to WPF, there were also decisions to be made related to the game and its game play. These decisions will be explained in this section.

Processing speech recognition

Because speech recognition is the most important feature of the game, a suitable approach to process the input was needed. To recognize any form of speech, Microsoft's Speech API (SAPI) has been used. The decision of using this speech recognizer was made by the company. In order for the speech recognizer to detect any words, a set of desirable words to be recognized has to be provided. With this set Microsoft's speech recognizer can try to match the user's speech input to any of the words in the set of possibilities.

When the recognizer recognizes any word that matches an entry in the predefined set, an event is fired. Microsoft's SAPI has been constructed in such a way that the event holds more specific data concerning the recognized word. When a word has been detected while playing the mini game, this word and its confidence rate are retrieved. This confidence rate is a measurement of how well the word was pronounced, on a scale of 0 to 1. In practice, with a varied pre-defined set, this rate will usually be above 0.85 for the recognizer to detect a certain word at all. For the mini game it was decided to use this rate to give the user a rating, which provides them with feedback. When the game has ended, a final rating is given, which is based on the total number of words correctly pronounced and how well they were pronounced.

Cheating

The mini games are all focused on the correct pronunciation of words or phrases. Before being able to correctly pronounce a word, one has to read it. Since the player only has a limited time to read and pronounce a word, he could cheat by pausing the game. Pausing the game would then lead to an advantage, as the player would be able to read the word at ease and pronounce it upon resuming the game. To prevent this from happening, it was decided to not let any words show on the screen when the game is paused, as shown in Figure 4.4.

Figure 4.4: The game is being played on the left and paused on the right

Control buttons

Besides being able to start the game, you could ask yourself whether the player should have any other controlling options. We chose to give the player the ability of pausing the game, and turning the background music and sound effects on or off. Pausing the game has been enabled as sometimes, during playtime, the player is forced to pause the game due to external influences, like a phone or doorbell that's ringing. The ability to enable or disable the music and sound effects has been implemented for

the player's comfort. Some people might prefer to listen to their own music, rather play the game in silence or simply find the game music distracting. Now that the player has the option to enable or disable the music and sounds, everyone can play the game as he or she pleases.

In most games, one of the options the player has is to end the game, but in the mini games we developed, there is no such option. You either have to complete the game by pronouncing all the words correctly or wait for the game to end. The reason we didn't introduce any *quit* button, is because this is a mini game with a short playing duration. Besides this, the mini game will be integrated in the final product, which will have some option to end the game from there.

The mini game also doesn't have a *save* button. Since the duration of the game is so short, there simply isn't a need to save the game. Instead the game should just be played again, or paused instead.

4.2 Technical difficulties

Of course we also encountered some technical difficulties during implementation. Sometimes implementing certain things didn't go as well as we expected, meaning suitable solutions had to be found.

Animating objects

Since we decided not to use a storyboard for animations, we had to manually animate objects with just C# code. Luckily, WPF has some built-in methods that are very useful for translating, scaling and rotating objects. These transformations can be achieved by the use of so called *Transform*s. To combine multiple transformations, a *TransformGroup*[8] can be used, as is shown in Figure 4.5.

To add transformations to the *TransformGroup*, one should include several parameters to tell WPF exactly what the transformation should do. For rotations, for example, an angle and rotation axis should be included. For scaling and translations, parameters related to respectively object size and position should be passed to the transform method. In our game, we make use of animation counters and the screen and object widths and heights, to obtain the right values, which in turn are passed on to the transformation methods. Sometimes, there is also an offset involved, which is obtained by tweaking the value until the desired effect is established.

Animation counters

As can be seen in the previous section, transformations are executed by specifying a certain parameter to a certain transformation method. However, when you want to fluently move an object from the left side of the screen to the right side, for example, you'd need a loop that moves the object just a little bit every time a new frame is rendered. In WPF, this can not be achieved by using a static number as a parameter, because for every transformation, WPF considers the object's original starting point. In our example, giving a parameter of 100, would move the object 100 pixels to the right the first time you loop through the animation. The second time, the object will move 100 pixels to the right again, but, again, from the original starting point, leaving the object in the same position it just was. This process repeats itself, and the object will remain static, 100 pixels to the right of its original location.

To solve this, we introduced animation counters. This time, instead of the number 100, we use the variable *animationCounter*. In the loop, we move the object to the right again, by the value of *animationCounter*. The first time this will be one pixel, but, in the loop, we also increment *animationCounter*. So the second time the program goes through the loop, it moves the object two pixels to the right, from its original position, which is one pixel from its previous position. This goes on and on, with the object moving one pixel to the right every time the loop is completed, causing a fluent animation on screen.

We applied this method to all dynamic animations, both for scaling and translations, and often combining multiple transformations as was explained in the previous section.

```
// Create a Button that will have two transforms applied to it.
Button myButton = new Button();
myButton.Content = "Click";

// Set the center point of the transforms.
myButton.RenderTransformOrigin = new Point(0.5,0.5);

// Create a transform to scale the size of the button.
ScaleTransform myScaleTransform = new ScaleTransform();

// Set the transform to triple the scale in the Y direction.
myScaleTransform.ScaleY = 3;

// Create a transform to rotate the button
RotateTransform myRotateTransform = new RotateTransform();

// Set the rotation of the transform to 45 degrees.
myRotateTransform.Angle = 45;

// Create a TransformGroup to contain the transforms
// and add the transforms to it.
TransformGroup myTransformGroup = new TransformGroup();
myTransformGroup.Children.Add(myScaleTransform);
myTransformGroup.Children.Add(myRotateTransform);

// Associate the transforms to the button.
myButton.RenderTransform = myTransformGroup;
```

Figure 4.5: An example of using a *TransformGroup*[9]

Animating the conveyor belt

A technical difficulty that already arose during the brainstorming session where Toy Factory was even so much as suggested, was how to animate the conveyor belt that would transport the toys towards the screen. After a few seconds of thought, we came up with a seemingly elegant and simple solution: we would simply animate the belt by quickly rendering around five images in quick succession. These images consist of a picture of the belt, with the texture moving slightly down every image, up till the point the image would be similar to the starting image. For this, we needed a so-called tileable texture, with a repeating pattern. Because we were not sure this would be the way to go, at first, we proposed to the graphical artist to use simple lines as a texture to see how it would look. To clear up this concept, Figure 4.6 displays the images that were used.

The lines on the image were soon called conveyor lines, but they turned out to be rather problematic. It was impossible to synchronize the movement of the toys, which was handled in the code, with the simulated animation of the conveyor lines that the graphical artist gave to us. The lines would either move too slowly at the beginning of the belt, or too fast near the end. It would be very difficult for the graphical artist to produce something that could be synchronized, due to the unusual nature of the toys' animation, which combined scaling and translation with an exponentially decreasing help variable to prevent either an explosive increased scale or movement speed. The toys' animation was basically flawed, compared to how an item in the real world would behave, but it was not feasible, if even

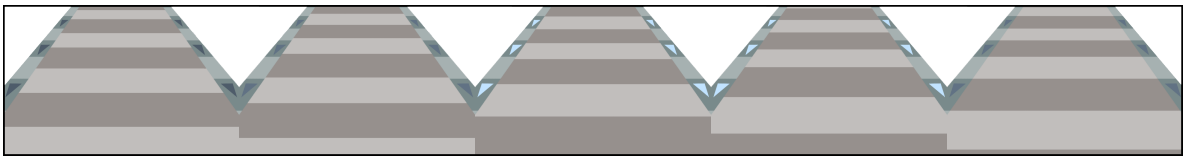


Figure 4.6: The fives images that were rendered in quick succession to simulate an animation

possible, for us to produce a perfect animation. To comprehend why, it should be taken into account that the game only allowed for 2D animation, while what we really needed was a 3D animation, which would subsequently be rendered on the screen. We would need to apply some tough mathematics to the toy's animation, further complicating it, while there was yet another option: handling the conveyor lines' animation in the code as well.

Figure 4.7: Light gray lines are *ConveyorLine* objects, synchronized to the toys' animation

Thus, we decided to choose this option. After tweaking some variables, it turned out even now it was impossible to synchronize the lines, except for when we applied the exact same animation on the both the conveyor lines and the toys. This meant, however, that the lines too would fall from the pipes and would only start moving forward on the belt when they hit the same spot as the toys, which was near, but not at the back end of the belt. The first issue was solved by only making the lines visible once they had reached the belt by tweaking the zIndex of several background objects, while the second issue was discarded as not that important, as the toys would finally be synchronized with the lines. The result can be seen in Figure 4.7, where the lighter gray lines are each separate objects and animated the same way as, in this case, the teddy bear.

Accuracy of the speech recognition

During implementation, it was discovered that the accuracy of the speech recognition didn't always seem to be optimal. When there are two words that are quite similar to each other, the wrong word is sometimes recognized. Since this is more an issue of Microsoft's Speech Recognition, we can't really do anything about this. The only thing that can be done to reduce any mix-ups between different words is to pick words for the game that aren't too similar to each other. By doing this, the risk of a wrongly recognized words will be minimized.

4.3 SIG feedback

During the internship, the code that has been written was sent to the *Software Improvement Group* (SIG). The code has been evaluated and rated by them. In Appendix F, the feedback received can be found. After the internship, the code was sent to SIG another time, to see if the feedback has been used to improve the code. This too can be found in Appendix F.

First feedback

There were some problems receiving the feedback by email, which caused us to read the feedback quite late. Nevertheless, a lot of the feedback concerned issues we were already aware, that, by that time, had already been improved.

According to SIG, the rating of the code was 2 out of 5, which is quite low. The main reason for this score were the *Module Coupling*, the *Unit Size* and the *Unit Complexity*. They recommended us to split up classes, shorten methods by splitting them up as well, and implementing a configuration file. We implemented this last recommendation by constructing a resource manager in a separate class that reads an XML file containing the configuration of the game. We also implemented the Factory Design Pattern to structure the code a bit more. Besides those changes we split up quite some methods to simplify and shorten them, making them easier to understand. We also created JUnit test cases, something which was also recommended by SIG.

Second feedback

After all the changes made to the code, SIG received the updated code to review it again. This new feedback can be found in Appendix F. From the feedback received it appears that the code scored a bit better this time and they did see we used their recommendations to improve the code, but also that the code could be optimized even more. In particlar, they commented on a very long method in the resource manager. However, this method consisted of a lot of repetition, because it was loading a *lot* of images that were to be used in the game. Furthermore, all the method's logic was included in a dispatcher, to be able to access the user interface, which runs on a separate thread, so splitting this up would cause some more complexity as well.

5

Results, recommendations and reflection

This chapter will review the schedules of requirements that we composed prior to every iteration. Furthermore, it will offer some ideas for future work to improve the final delivered products, as well as discuss some recommendations for new mini games. Finally, there will be a reflection, describing what we would have done differently if we would start all over again.

5.1 Results: the schedules of requirements reviewed

Prior to the implementation of each iteration, we would compose a schedule of requirements. These schedules, which can be viewed in Appendix D, can now be used to verify if all core requirements (*must have* and *should have*) are met, and if any additional possibilities (*could have* and *would have*) are made possible when final implementation is done. In this section, we will mostly be focusing on the requirements and implementation for *Toy Factory*.

5.1.1 Functional requirements

Most requirements regarding the gameplay are listed under the functional requirements. For the ease of reviewing, the requirements will be reviewed by the categories belonging to the *MoSCoW* model.

Must have

The most important requirements fall into this category. All of these must be satisfied in order to be able to call the result a finished product. In our opinion, we successfully implemented all these *must haves*, although there might be some requirements that are slightly difficult to judge, since they are a bit subjective. Requirements such as *the game must be fun* and *the game must look aesthetically attractive* are subject to the personal opinions of those who play the game. Also, *having enough time to read a word* and *clearly readable* can depend on the person – just think of people with dyslexia or bad eyesight. However, since these are uncommon cases, and experience learns you cannot always satisfy everyone, we feel confident enough to state we successfully met the requirements considering the average person.

Should have

Requirements falling in the category *should have*, should all be implemented. Looking back at the schedule of requirements, it seems that all but one have been implemented.

The only requirement that did not end up being satisfied, states that the user should see if the microphone is picking up any sound. The reason for this is that the mini game is part of a bigger game, and before the user is actually able to play the mini game, it has already been shown if the microphone is picking up any sound through the main game. Also, we found a sign to show if the microphone is picking up any sounds might be distracting to the user, as the sign may keep on changing during gameplay, because the player will not always be talking.

Could have

The requirements listed under *could have* provide options that do not necessarily need to be implemented for a complete product, although they can add a nice touch. In our case, quite a few of the *could haves* did end up being implemented, while some others were left out.

Background music has been implemented, as it didn't seem to interfere with the speech recognition and made the game more dynamic. Realizing not all users might appreciate the background music and sound effects, the player has been given the ability to turn off the music and sound effects separately.

As for bonus items and modes, a few have been implemented. For *Toy Factory*, when the user correctly pronounces a word that is displayed on a bar of dynamite, the conveyor belt in the game temporarily breaks down, which freezes both the belt and the toys, along with stopping time for a short period, giving the user more time to quickly pronounce the other words that show up on the belt. Furthermore, some bonus moles that offer special effects have been added to *Whack a Mole*, while we also added a special item to *Falling Words*, which reverses the keyboard controls.

Concerning the difficulty level and varying speed of the toys, these can be regarded as implemented in the final product. The difficulty level can be adjusted in an XML file that is read upon starting the application, but cannot be changed during the game. The speed of toys depends on the set difficulty, so like the difficulty, it can be changed. We did however let all toys have the same speed, so the variation is purely between different games, and not between the toys themselves.

5.1.2 Usability

In the requirements, it is stated that the game should be easy to get started with, and the player should intuitively know how to play the game before it actually starts. This was implemented by showing an instruction screen is to the user after the game has loaded. This screen will disappear when the actual game starts. To ensure that the player has plenty of time to read the full instructions, the game will only start when the start button is pressed. This also allows users who have played the game before, and don't want to read the instructions again, to simply click the start button right away.

It's also stated that the game should be fully playable by speech recognition only. Depending on how this statement is interpreted, you can say that the game was both successful and lacking in this area. If you consider the part where the gameplay is active and the user can actually do something as the game, this requirement is met. But if you interpret the game as the full product, including start-up and ending screen, then the requirement is not entirely met, since the user still has to click on buttons for starting, pausing or restarting the game. Considering this ambiguity, the requirement should've been rephrased earlier to avoid confusion. In our personal opinion however, we find the game to have succeeded in this area, as we consider 'the game' to be the active gameplay, and it would be impractical to have the user control the menu with his or her voice as well.

5.1.3 Performance and reliability

Considering that the speech recognition doesn't cause delays to the game, performance doesn't seem to be affected by this. Another performance-related issue that came up, was the creation of high-resolution images. However, this was eventually solved using a resource manager, leading to a good overall performance. As for reliability, the speech recognition detects all pre-defined words, although sometimes it seems to be having difficulties recognizing certain words that are not perfectly pronounced. While this does have some effect on the reliability of the game, it cannot be helped, since the speech recognition is a built-in API. In the end, because the effects are so small, we consider the game reliable enough to be very well playable.

5.2 Future work, improvements and recommendations

There is still plenty of work that needs to be done on the mini games part of the main game. As *Whack a Mole* was discarded, we will not discuss any improvements for that. Instead, this section will be split up in three parts, where we will discuss improvements on *Toy Factory* and *Falling Words*, while making some recommendations on new mini games as well.

5.2.1 *Toy Factory*

Although *Toy Factory* was the most developed mini game, it could still be improved upon. For example, there is still room for several special items that have different effects, similar to the dynamite that now breaks the conveyor belt for a limited time. One option that we considered was reversing the belt's movement, which would result in toys moving backwards. This in itself wouldn't be so hard to implement, but we did not come up with a good solution for toys falling off the back of the belt. If they would just disappear, it would make the reversing effect negative in many cases, and if a toy would be put to a stop at the back end, it would cause a problem when another toy that is moving backwards on the same lane, eventually causing a collision with the stuck toy, hiding the stuck toy's word from sight. There are still many other possibilities, like special items that would cause all toys that are currently on the belt to be wrapped, or items that would slow the belt down.

Another possible improvement would be to include a list of high scores, which would be saved in a separate file, so that top scores are stored for the next time the game is started. Increasing the competitiveness may indeed be a good idea, and could also be achieved by introducing a multi-player mode. Using a split-screen mode, with each player having their own unique words on the belt, this would of course lead to interference, and although you could argue this is problematic, it may also cause hilarious situations where players try to counter their opponents by purposely interfering when they are just about to say a word. If this turns out to be too impractical, there is also the possibility of an online mode, which would work in the same way, except for the interference. However, this would remove the interaction between the two players, so instead of a split screen, the online mode may be more fun using a shared conveyor belt and shared words, having the mini game keep up who pronounces the right word first. Online or not, we think a multi player mode could be very much fun indeed, although the implementation could prove to be tricky, especially for the online mode.

5.2.2 *Falling Words*

Falling Words was far from finished, and still requires a lot of work to be qualified to be included in the main game. Currently, the biggest problem is that it is simply too boring for a mini game that is supposed to be fun. It may be a good idea to include some sort of panic factor, like the toys that smashing against your screen in *Toy Factory*. One way to do this is to maybe reverse the concept, where items fall from words if you do not pronounce the whole sentence correctly. The avatar could

still be used to collect the items that fall down, thus minimizing damage, but the intensity of items falling down could increase over time, putting some pressure and excitement into correctly saying out loud the phrases.

Furthermore, *Falling Words* too has many possibilities for bonus items, and we did already implement one, which reverses the keyboard controls. Other options are items which increase the size of the avatar's basket in which the items must be caught, or bonuses which stop the falling items altogether.

5.2.3 Future mini games

While there were some mini game concepts available when we arrived in Singapore, the first one, *Whack a Mole* got rejected by M3 Technologies for being too static. Following this, we held several brainstorm sessions with our colleagues, during which the group and us came up with several ideas. In the end, most voted for *Toy Factory*, but there were several other interesting concepts that may be considered for implementation. One of these was a game that featured a monster lurking at the bottom of your screen that would eat words or phrases falling from the top. Every time a word or phrase is eaten, the monster would grow, slowly filling the screen. The objective is to quickly and correctly pronounce the right text before the monster can eat it. Once the monster fills your whole screen, the game would be over, like the famous *Tetris* game. Similar to *Tetris*, the game increases in difficulty if the monster grows, creating an excitement factor. We do recommend this game to be implemented, as it certainly has potential to be fun and exciting, while still providing the same educative value as *Toy Factory*.

Another interesting concept was a mini game that featured a plane which needed to be prevented from crashing. Looking at the plane from a side view, it would lose height continuously, but gain a boosting lift every time a phrase or words gets correctly pronounced. These phrases could be located on the scenery the plane is passing by, or even be related to this scenery. Another advantage of using sceneries is that the plane mini game could cover all scenarios from the main game, passing from one scenery that corresponds with a certain scenario to another, and featuring the corresponding words or phrases that have been studied by the player during this scenario. This seems like an ideal way to reiterate the learned material and combined with the thrill of keeping the airplane adrift, we present this concept as a recommendation as well.

5.3 Reflection: what we would have done differently

No matter how meticulously planned, every project has a time of reflection, during which the project members will usually say to themselves: *if I would start all over, I would have done things differently.* As expected, this is also the case with our project. The main thing we would have changed, is to keep more true to the original schedule, doing multiple, true iterations for each mini game, while also adding unit tests during each iteration. This would have made it easier to incorporate new functionality later on. Another crucial change we would make, is to actually finish an iteration before starting a new one. At the time, we cut short development on *Falling Words*, focusing on *Toy Factory* instead. This seemed like the logical thing to do, given the priority that was given to *Toy Factory*, but in the end we were left with an unfinished *Falling Words*, which was just what we wanted to prevent by introducing all these iterations. Looking back, we should have done it in our own way, finishing the prototype for *Falling Words* first.

6

Conclusion

This chapter will formulate our conclusion in three parts. The first section will discuss the final product and our opinion of it, while the second part describes how we experienced the project in general. The final section focuses on the actual experience of working in Singapore's gaming industry.

6.1 The final product

Although we may have underestimated the amount of work that needed to be done, and we did deviate from our original plan quite a bit, we can say we are content with the resulting product. Despite its simple concept, *Toy Factory* is, in our opinion, quite a fun mini game to play. There aren't a lot of games around that work with speech recognition – one reason being the difficult challenge of creating a near-perfect recognizer – and we think this originality will surprise the user and keep them entertained for a while. To keep the user interested, however, we recommend to add some sort of competitiveness, at least by implementing high scores, and maybe even by introducing a multi player mode. In its current state, we don't see *Toy Factory* being played for hours. However, it is only a mini game, and by adding more of these, we do believe that the mini game part of the main game could provide the users with hours of fun.

6.2 The project in general

We started out this project by emailing random Singaporean information technology companies – to no avail – before we made up our minds about wanting to do this internship at a gaming company. Finding a list of game developers in Singapore on the website of the *International Game Developers Association*[10], we emailed all of them a message that explained our situation and included our résumés. We only got a response from Nexgen Studio and Cabal Entertainment Software – which forwarded our application to Touch Dimensions. Nexgen and Touch Dimensions were working together on a newly started project: an educative game that would be played with speech recognition. After making several other arrangements, we eventually arrived in Singapore, not exactly sure what to expect of the project. We soon learned that we had a lot of freedom in determining our own way to implement the mini games, and that we also had a say in developing a concept, as became clear during the brainstorm sessions that led up to *Toy Factory*, which was very much welcomed by us. Despite this freedom, we did definitely communicate with the company a lot – working at their office and being present during most meetings and discussions – so that we had a good idea of the company's wishes.

Of course we also regularly communicated with the university, and although there could be no

face-to-face meetings with our project mentor, we did have a lot of contact through email. This may not be as practical as real meetings, but it did not pose an insuperable problem either. All in all we think that, although it requires some extra work to prepare for such a project, doing your internship abroad is a great experience. We surely enjoyed it, and also very much improved our coding skills during this project, while learning a lot about WPF and its many possibilities.

6.3 Working in Singapore's gaming industry

Because it is a bit different from The Netherlands, we decided to dedicate this section to describing a regular day of working in Singapore, at one of the several small gaming companies. Of course, Nexgen Studio may not be representative for all firms in the industry, but we believe that on many points, the company gave a good indication of how things work in this particular branch of Singapore's corporate life.

At Nexgen, the work day starts at 10:30 AM, although this is not custom to most Singaporean companies. After spending an hour in the well-regulated but busy public transport we would arrive at the office and set up.

Figure 6.1: The Singaporean Mass Rapid Transit (MRT) is a very popular and cheap way to travel

Around 1:45 PM, we would go out with most other employees to have lunch at one of Singapore's many food courts, a gathering of many food stalls where you can order your meal. On occasions, the boss would treat us to lunch in a restaurant, which added to the informal atmosphere. This informality did seem common among small game companies, as we did get acquainted with some other, similar companies. After an hour or so we would return to the office to continue work.

Every few days there was a meeting to get an update on everyone's progress, to discuss additions that were to be implemented during the coming days or to communicate any other news. Besides the meetings, both Jeffrey Jiang, our company mentor, and Alvin Yap, Nexgen's CEO, would check up on

us every once in a while to see if the project was coming along and if everything was still consistent with the company's wishes. Around 6:30 PM, work would be over, although both the starting and ending times were very flexible, as long as you finish everything in time for the deadlines. Afterwards, we'd often stay around to play some games with our colleagues, or we'd go to the cinema with them. Besides the work, which was enjoyable in its own right, it was this friendly and informal interaction with our co-workers that made every day fun, and we can certainly say we made friends there.

Working in Singapore is quite an experience, and we can recommend it to anyone who wants to spend some time abroad. It is clean, well-developed, has great public transport, offers many tourist attractions and other activities, is perfectly situated if you want to explore more of Asia, and, on top of that, it has great people who are easy to befriend.

Figure 6.2: Part of Singapore's skyline at night

We can conclude that we thoroughly enjoyed this project abroad and recommend it to anyone who has the time to be away from home for a few months, as it is an educative but fun experience that you will never forget.

Bibliography

[1] C. Nutt, "**Singapore's Government On Encouraging Game Industry Growth**." *gamasutra.com*. UBM Techweb, 16 July 2010. Web. 27 August 2011. <http://www.gamasutra.com/view/news/28815/QA_Singapores_Government_On_Encouraging_Game_Industry_Growth.php>.

[2] "**Canada: We're Number Three**." *gamepolitics.com*. Entertainment Consumers Association, 5 April 2010. Web. 27 August 2011. <http://gamepolitics.com/2010/04/05/canada-we're-number-three>.

[3] "**IDC Forecasts Worldwide Mobile Applications Revenues to Experience More Than 60% Compound Annual Growth Through 2014**." *idc.com*. International Data Corporation, 13 December 2010. Web. 27 August 2011. <http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22617910>.

[4] "**MoSCoW Prioritisation**." *coleyconsulting.co.uk*. Coley Consulting, no date. Web. 27 August 2011. <http://www.coleyconsulting.co.uk/moscow.htm>.

[5] M. Snow, "**Silverlight Tip of the Day #16 – StoryBoard versus DispatcherTimer for Animation and Game Loops**." *blogs.silverlight.net*. Microsoft, 9 July 2008. Web. 27 August 2011. <http://blogs.silverlight.net/blogs/msnow/archive/2008/07/09/storyboard-versus-dispatchertimer-for-animation-and-game-loops.aspx>.

[6] "**XAML Overview (WPF)**." *msdn.microsoft.com*. Microsoft, no date. Web. 27 August 2011. <http://msdn.microsoft.com/en-us/library/ms752059.aspx>.

[7] "**Storyboards Overview**." *msdn.microsoft.com*. Microsoft, no date. Web. 27 August 2011. <http://msdn.microsoft.com/en-us/library/ms742868.aspx>.

[8] "**How to: Apply Multiple Transforms to an Object**." *msdn.microsoft.com*. Microsoft, no date. Web. 29 August 2011. <http://msdn.microsoft.com/en-us/library/ms750975.aspx>.

[9] "**TransformGroup Class**." *msdn.microsoft.com*. Microsoft, no date. Web. 29 August 2011. <http://msdn.microsoft.com/en-us/library/system.windows.media.transformgroup.aspx>.

[10] "**IGDA Singapore/Game companies**." *wiki.igda.org*. International Game Developers Association, no date. Web. 27 August 2011. <http://wiki.igda.org/IGDA_Singapore/Game_companies>.

Appendices

The following pages only contain appendices.

Appendix A

Project description

About the company

In total, there are four companies involved in the project. First there is Nexgen Studio, a small game company that primarily focuses on smaller, innovative products for both PC's and mobile phones. We will complete our project at this company's office, and will do our work at this firm's address. More information can be found on the company website: http://www.nexgenstudio.com/ .

Furthermore, there is Touch Dimensions, which is our company mentor's company. This firm specializes in making games and software which is controlled using a touch screen, although they do participate in other projects as well. Their products are primarily aimed towards smartphones, but some also target the PC and tablet market. More information can be found on the company website: http://touchdimensions.com/ .

There's also the listed Malaysian company M3 Technologies, which is a large player on Asia's *Mobile Value Added Services* market.

Finally there is M3 Interactive, our official employer, which is a joint venture of Nexgen Studio and M3 Technologies. This company just got set up and is currently focusing on the project we will be working on as well. The bigger aim is to create a market for language education through speech recognition

About the assignment

The project itself consists of a game that teaches native Mandarin speakers the correct English pronunciation for a great variety of words and phrases. This is done by letting the user do some role playing through several scenarios and play some fun mini games. In this way the player can be playfully educated. The project's deadline is around October.

The assignment that we will focus on, is designing, implementing and testing the mini games for the project. There are already four concepts for mini games available, and we are looking to implement a total of three mini games, one of which we hope is one of ourselves. Despite the existing concepts, we are free to make the games in our own way, while making our own additions and changes.

Voice recognition is handled using Windows Speech Recognition, and the project is being developed in C#, using the graphical subsystem WPF (Windows Presentation Foundation), which is part of the .NET Framework and uses DirectX and XAML, a derivative of XML. WPF is very similar to Silverlight, and our application will be compatible with both.

APPENDIX B

ORIENTATION PHASE

Timothy Kol 1509217
Irene Renkens 1534629

University mentor: Peter van Nieuwenhuizen
Company mentor: Jeffrey Jiang

BACHELOR OF SCIENCE COMPUTER SCIENCE
BACHELOR PROJECT AT
M3 INTERACTIVE, SINGAPORE

PREFACE

This document is a report on all orientation and corresponding research that will be conducted during this project.

This version is a draft, based on the first 10 days of working on our project, in which we mainly focused on orientation by means of creating a prototype and researching the Internet.

TABLE OF CONTENTS

SUMMARY

The main topics that were researched so far, are the differences between Java and C#, general information on WPF and Silverlight, including XAML.

More specific subjects that came forward during the creation of our prototype, were the correct use of a speech recognizer, the different methods for creating a game loop in WPF, and access to elements in an XAML window from another class.

The development approach that will be used is agile software development.

INTRODUCTION

Being assigned the creation of a minigame prototype by the end of the week on the first day, we decided to incorporate our orientation phase with the development of this first prototype. In our opinion, the best way to learn something new is actually doing it, so we decided to do our extensive research as we went along with this first assignment, checking out the Internet when we would stumble upon problems.

The project will consist of several minigames to develop and implement. According to the company's wishes, these games will be written in C# and developed with WPF, but also need to be compatible with Silverlight. The tools we're required to use are Visual Studio 2010 and/or Microsoft Expression Blend 4.

Over the course of the project, we are certain that there will be some additional research needed when problems are encountered or new subjects are introduced. This will also be included by updating this document when necessary.

The remainder of this document will discuss several researches that have been conducted, starting with the programming language C#. Next up is a section on WPF and Silverlight, and the way they differ. This leads to some research on XAML, the markup language WPF utilizes. The following section is a short comparison of Microsoft's Visual Studio and Blend, and how we plan to work with these tools.

The next sections focus on problems already encountered during the creation of our prototype, including the use of a speech recognizer and how we decided to implement a game loop.

Finally, there is a section on research on the software development methodologies, and which one we will use.

THE PROGRAMMING LANGUAGE C#

We are both more experienced in Java than C# since Java has been mainly used during the three years of the Computer Science Bachelor. Both languages have quite some similarities but are not exactly the same. We decided to research some of the similarities and differences between each other that could be useful to us, which are listed below [1][2].

SIMILARITIES WITH JAVA

- Class-based object-oriented
- Uses garbage collection
- So-called 'curly brace' language
- Designed with runtime compilation
- Common terminology and similar syntax features
- Primarily statically, strongly and manifestly typed

DIFFERENCES WITH JAVA

- Pointer usage
- Getters and setters (properties)
- LINQ and Lambda expressions
- the use of keyword *var*
- Naming conventions such as:
    - Private member variables in C# have an underscore "_" to prevent name collision in constructors.
    - C# doesn't distinguish between implementing an interface and extending a class, interface declarations have an "I" prefix, such as IMyInterface.
    - C# uses C style block declarations where open curly brace is on a new line, while Java has the open curly brace on the same line as the expression or declaration, the exception in both cases being inline blocks.
    - Java uses the keywords *implements* and *extends*, C# uses the colon (:) to show that a class implements an interface or extends a class. C# syntax doesn't differentiate between the two whereas Java is strict about which keyword should be used.
    - [2] Provides a full keywords comparison overview between C# and Java

WPF AND SILVERLIGHT

During the course of the project we will be developing several mini-games. These games need to be written in C# with the use of WPF but also need to be able to run in Silverlight. As we have never worked with either WPF or Silverlight, we did some research as to what they are exactly. As the games need to be compatible with both WPF and Silverlight, we also wanted to know if this might lead to restrictions in certain areas.

WPF

Windows Presentation Foundation (WPF) is a graphical interface toolkit developed by Microsoft. It is part of the .NET framework since 2007. With WPF, it is possible to easily design GUI interfaces for Windows Applications. Elements such as 2D/3D rendering, animations, vector graphics, pre-rendered media and various effects are supported. These elements can then be manipulated on events or user interaction [3]. For defining and linking UI elements, XAML is used.

SILVERLIGHT

"Silverlight is a version of WPF intended to run applications in web browsers. To squeeze Silverlight applications into a format that will run reasonably on a browser, Silverlight has a

few restrictions such as fewer controls and more restricted access to the user's system than WPF has, but there are many similarities between the two." [4]

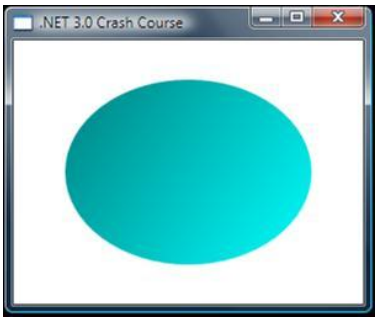
Since Silverlight appears to be a lighter version of WPF, we need to keep this in mind as the mini-games that we will develop need to be able to run in Silverlight as well. This means we need to ensure we will not write any code that will only work in WPF but not in Silverlight.

XAML

Both WPF and Silverlight make extensive use of XAML. XAML is an abbreviation for Extensible Application Markup Language, and is a declarative XML- based language. It is created by Microsoft and used in the .NET Framework 3.0 / 4.0. With XAML one can define and render the graphical user interface.

An example of XAML code and its output:

```
<Ellipse Width="200" Height="150">
   <Ellipse.Fill>
      <LinearGradientBrush>
         <GradientStop Offset="0" Color="Teal"/>
         <GradientStop Offset="1" Color="Aqua"/>
      </LinearGradientBrush>
   </Ellipse.Fill>
</Ellipse>
```



(Source: http://devlicio.us/blogs/rob_eisenberg/archive/2006/10/02/Down-and-Dirty-With-Xaml.aspx)

In WPF, there is a .xaml file in which the graphical elements are defined. Besides this file there is also a .xaml.cs file in which the logic is placed, the so-called code-behind file. As the design and logic are now separated in two different files, the code is much easier to read. When the XAML page is markup-compiled, the code in this file is joined with markup-defined objects.

Since the design and logic are separated, there must be some sort of link between them in order to let them communicate. When the application contains elements such as buttons and checkboxes, it can be easily detected whether the user has performed a mouse click or has pressed a key on the keyboard. According to the action an event can be triggered.

Since we are developing a game, we also want to be able to change elements automatically, for example after a random amount of time. The method which will contain the logic may also not be in the .xaml.cs file but in a different class, meaning we have to figure out a way to access the element we want to change. While playing around with WPF, we found that this is possible when giving the element to the constructor of the class that will contain the logic for the transformation. A better solution we found however, is accessing the required elements by making a public static class Global in which we store the main window, and which we can call from any class to edit elements in this main window.

MICROSOFT VISUAL STUDIO 2010 / MICROSOFT EXPRESSION BLEND 4

We will be working with Visual Studio 2010 and/or Microsoft Expression Blend 4. Visual Studio provides a better environment for creating the C# code-behind but lacks when it comes to creating XAML interfaces, while Blend is the opposite [4]. Blend also provides the possibility to switch between itself and Visual Studio.

Both Visual Studio and Blend provide a Design view in which can be worked visually. When working in Design view the XAML code will be generated automatically. From our looks at it, Blend indeed offers more options concerning the design view such as an overview of the hierarchy between the elements. Though concerning the code-behind, Visual Studio offers much more, such as a debugger and possibilities for testing. As Blend provides the option to switch between itself and Visual Studio, we opt to use both programs. Blend for the design and Visual Studio for the code-behind so we can exploit them both.

SPEECH RECOGNITION

Since our project involves speech recognition, we need to understand how this works. The company wants us to use the built-in Microsoft Speech API (SAPI) [6] for the project. They have also provided us an example project in which the SAPI was used, so we could study this. From this example project we learned that we need to create a grammar (which will contain all the words and phrases we want to be recognized), attach an event handler to the grammar and load the grammar into the recognizer. When the recognizer starts listening and recognizes a word or phrase defined in the grammar, the grammar's event handler will be invoked. The event handler can access the result object which for example can be used to show the word or phrase pronounced.

Because the normal speech recognizer also recognizes Windows commands, we had to use the speech recognition engine class, which only recognizes words from a predefined array, which we can define.

GAME LOOP

Since we are creating a game, we need to find a way in order to keep all elements updated. The usual way to do this is to write a game loop. There are several ways you can do this in WPF and Silverlight. Our project mentor sent us a link [7] in which so called StoryBoards were used to create a game loop. We tried this but found out this only works properly in Silverlight and not in WPF, so we had to find another way to create our game loop.

There were now two other options left. One option was the use of the DispatcherTimer[8]. This however did not seem to be a very good option as the DispatcherTimer does not know when the frames are rendered and is said to be unstable.
The other option involved, makes use of an event, the CompositionTarget.Rendering event

to be precise, which fires once per frame making this a good choice for the game loop[8]. Since this method seemed to work for both WPF and Silverlight, and has no big disadvantages like the DispatchTimer, we use this method for the game loop.

DEVELOPMENT APPROACH

In order to structure the development process of our project, we want to make use of a software development methodology. There are several different methods to choose from. During the three years of our Computer Science bachelor we've already worked with methods as the Waterfall methodology and the Agile methodology. We both aren't fond of the Waterfall method as each phase must be fully completed before moving on to the next one which makes this method very inflexible[5]. For this reason we decided we will not use this method.

Another approach is the Agile methodology. The Agile methodology is based on an incremental approach, meaning that there are multiple phases with each phase containing its own mini-Waterfall phases. Agile software development also makes use of Rapid Application Development (RAD) which means that during the development of the software, prototyping is an important factor. Since we'll have to develop several minigames, the Agile method might be fit. Since we both also have some experience already with this method, we prefer to use this method instead of others.

BIBLIOGRAPHY

[1] Microsoft. (2011) C# and Java: Comparing Programming Languages. [Online]. http://msdn.microsoft.com/en-us/library/ms836794.aspx

[2] Dare Obasanjo. (2007) C# From a Java Developer's Perspective. [Online]. http://www.25hoursaday.com/CsharpVsJava.html

[3] Rod Stephens, *WPF Programmer's Reference*. Indianapolis: Wiley Publishing, Inc., 2010.

[4] Microsoft. (2011) Microsoft. [Online]. http://msdn.microsoft.com/en-us/library/ee125663%28v=VS.85%29.aspx

[5] Mike Snow. (2008, July) Game Programming with Silverlight. [Online]. http://blogs.silverlight.net/blogs/msnow/archive/2008/07/09/storyboard-versus-dispatchertimer-for-animation-and-game-loops.aspx

[6] Mike Snow. (2008, April) Game Programming with Silverlight. [Online]. http://blogs.silverlight.net/blogs/msnow/archive/2008/04/01/timers-and-the-main-game-loop.aspx

[7] Mike Snow. (2008, September) Game Programming with Silverlight. [Online]. http://blogs.silverlight.net/blogs/msnow/archive/2008/09/29/silverlight-tip-of-the-day-50-main-game-loop-revisited.aspx

[8] Microsoft Corporation. (2011) WindowsClient.net. [Online]. http://windowsclient.net/wpf/

[9] Centers for Medicare & Medicaid Services. (2008, March) SELECTING A DEVELOPMENT APPROACH.

APPENDIX C

PLAN OF APPROACH

Timothy Kol 1509217
Irene Renkens 1534629

University mentor: Peter van Nieuwenhuizen
Company mentor: Jeffrey Jiang

BACHELOR OF SCIENCE COMPUTER SCIENCE
BACHELOR PROJECT AT
M3 INTERACTIVE, SINGAPORE

PREFACE

This document serves as a contract that defines the problem to be solved during this project, as well as to capture what is expected of the sponsor (the company, M3 Interactive) and project members (the interns, Timothy Kol and Irene Renkens). On top of that, this plan will offer an approach of the project's phasing, and establish several other agreements.

In general, this document will benefit the process by clearly outlining the project, the approach that is to be taken, and the development process that will be used to solve the problem this assignment offers. This will prevent misconceptions later on, and ensures an efficient course.

TABLE OF CONTENTS

SUMMARY

M3 Interactive is a Singapore-based joint venture of Nexgen Studio, a game developer, and M3 Technologies, a Malaysian provider of mobile value added services.

The project consists of developing minigames for an educational voice recognition game to improve English pronunciation in a playful manner. The objective is to design, implement, test and evaluate these minigames and to make them reinforce lessons learnt during the main game by using voice recognition, while still being fun to play.

The minigames will be developed by using an agile software development approach, each iteration either adding a new minigame or improving an existing one. The plan is to develop two polished minigames, *Toy Factory* and *Falling Words*.

1. INTRODUCTION

This section is a short introduction that includes a profile of the involved companies, as well as a short description of the project and its motivation. The final paragraph will lay out the structure of the remainder of this document.

1.1 COMPANY PROFILES

There are multiple companies involved in this project, although we can define one sponsor

The sponsor is the company that's employing the project members and passing on assignments. This company is M3 Interactive, a Singapore-based joint venture of Nexgen Studio and M3 Technologies. M3 Interactive is a young business that focuses on educational games to learn English through voice recognition.

Nexgen Studio, Pte. Ltd., is a small Singapore-based company that develops creative PC and mobile games for both the education and entertainment industries. This corporation has made numerous games for other companies, as well as releasing some of their own. Touch Dimensions is another game studio based in Singapore, which works closely with Nexgen Studio on several strategic levels. In this particular case, Touch Dimensions assisted in the technical implementation of the project.

M3 Technologies is a listed Malaysian company focusing on mobile value added services, all mobile services beyond voice calls and fax transmissions. They have offices in several Asian countries and offer a wide range of mobile solutions.

1.2 PROJECT DESCRIPTION AND MOTIVATION

As the previous paragraph made clear, our project is part of a larger scheme, which may cause confusion. To be able to clearly describe and distinguish both projects, we will define our task as the minigames project, while the big program will be defined as the voice recognition project.

The voice recognition project is an educational game which teaches Mandarin (the most common Chinese language) speakers to improve their English, mostly focusing on pronunciation. The motivation for this project is based on M3 Technologies' supposition that there will be quite a bit of demand for this kind of game on the Chinese market.

The minigames project is a part of the voice recognition project, and will be incorporated by including several minigames in the final product. This can be motivated by regarding the fun factor this offers to make learning easier. The minigames will be basically testing the player's recently gained knowledge in a playful manner, allowing for a fun way to rehearse the new insight. This rehearsal is expected to improve the user's learning abilities.

1.3 STRUCTURE OF THIS PLAN

The remainder of this document will give a detailed description of the minigames project, touching on the problematic current situation and its solution, the objective and required deliverables.

Furthermore, there will be a section on the approach that is taken, by defining the software development methodology, activities and providing a draft of a schedule.

The next section will focus on the project design, and will describe the way in which the project is to be set up according to the approach proposed in the previous section, with respect to the involved parties and facilities.

Finally, there is a section on how we plan to control quality assurance.

2. PROJECT DESCRIPTION

This section will contain a detailed description of the project, including the existing problem, the objective, and a specification of the required deliverables.

2.1 CONTACTS AND SPONSOR

Below is a table of all relevant contacts.

Name	Function	E-mail	Phone
Timothy Kol	Intern / student	kollie88@gmail.com t.r.kol@student.tudelft.nl	+31 6 5345 5625 +65 8460 9342
Irene Renkens	Intern / student	irenerenkens@hotmail.com i.m.renkens@student.tudelft.nl	+31 6 1966 6060 +65 9039 2938
Peter van Nieuwenhuizen	University mentor	p.r.vannieuwenhuizen@tudelft.nl	+31 15 278 80 36
Jeffrey Jiang	Company mentor	jeffreyjiang@touchdimensions.com	+65 9354 1264
Alvin Yap	Nexgen CEO	alvinyap@nexgenstudio.com	+65 9685 1941

Table 1: Contacts

To reiterate, the sponsor and employing company is M3 Interactive, a joint venture of Nexgen Studio and M3 Technologies.

2.2 PROBLEM

The voice recognition project is supposed to be a fun way to learn English pronunciation, and the current situation lacks in this area. There needs to be a solution to make this game more enjoyable, while still reinforcing the educational aspect.

2.3 OBJECTIVE

The solution to this is introducing minigames to the project, which will provide a fun way to continue learning, by adapting these minigames to corresponding lessons the game offers. The objective of this minigames project is to design, document, implement and test an array of minigames that incorporate some educational value while they should be fun to play.

2.4 DETAILED DESCRIPTION

The game requires existing knowledge of the English language, and will mostly serve as a tool to playfully improve the player's pronunciation, grammar and vocabulary. This will be achieved by going through different scenarios that simulate common, real life situations. These scenarios will have multiple scenes, varying in difficulty, from simply reading words and listening to the correct pronunciation, to participating in role playing with the computer by correctly choosing and pronouncing sensible responses.

The gaming aspect is enforced by having several scenarios include minigames, to ensure the player is having fun while learning. Each minigame will reinforce the lessons learned during the corresponding scenario, by using words and sentences relevant to the covered situations.

Although several ideas for minigames have been defined by the internal sponsor already, the project members are free to implement the minigames as they see fit, starting from scratch writing their own documentation and tests. The remaining, undefined minigames are to be designed by the project members as well if there is enough time left.

Programming the minigames project will be done in Microsoft Visual Studio 2010, using the C# programming language and the WPF (Windows Presentation Foundation) graphical subsystem to incorporate graphics. The final solution will have to be compatible with the browser-based Microsoft Silverlight. All of these are the sponsor's wishes and are agreed upon by the project members.

The problem with these minigames is making them enjoyable, while still offering some educational value. The games should spice up the recognition project, by offering the user some relaxing fun, but there should be a balance with regard to the learning process as well. We plan on solving this by gathering feedback from our co-workers, the sponsor and possibly Chinese people and an English teacher.

Furthermore, the company is aiming to make the minigames exciting to play. We will be trying to achieve this by animating images and giving unique bonus rewards for correctly pronouncing words. When a user fails at this, a fitting penalty will be given, which they will want to avoid at all costs.

2.5 DELIVERABLES

The project members are required to deliver several minigames (including documentation) which are to be implemented in the voice recognition project. To achieve this, the source code needs to be compatible and readable, and the game play needs to make sense with respect to the scenarios that were dealt with. In the end, the minigames are delivered as a .dll file to be implemented.

Any report that is sent to the university will also be delivered to the sponsor.

3. APPROACH

This section will contain an overview of the proposed approach of the project by giving a description of the software development methodology to be used, an overview of the activities, and a draft of the project schedule.

3.1 DEVELOPMENT METHODOLOGY

The voice recognition project is a hybrid of the waterfall and agile methodology, while for the minigames project we have decided to implement it using agile software development. This will ensure a working version every iteration, which is desirable given the fact that the assignment consists of multiple minigames. At first, the iterations will focus on constructing working, but basic versions of the minigames, while later on new features could be built in to improve the entertainment aspect.

Every iteration, we will work through a full development cycle, taking into account the previous iteration.

3.2 ACTIVITIES

The project member's activities consist of working through a full software development cycle for every iteration to improve the minigames or create new ones. Each cycle includes planning, analyzing requirements, designing, implementing, unit testing and performing an acceptance test on the iteration, as well as ensuring compatibility with Silverlight. Also, two-weekly progress reports needs to be kept by the project members and viewed by both the sponsor and the university. The resulting minigames will be exported in a .dll format to be implemented in the voice recognition project. To ensure correct exporting, we will deliver a .dll every iteration, so that there will be no unexpected problems regarding the integration in the final week.

3.3 SCHEDULE

We will try to define a schedule for the iterations in the following table.

#	Date	Iteration description
-	25 Apr – 06 May	No iteration, orientation phase, constructing a basic version of *Whack a Mole* minigame and making a plan of approach and report on the orientation phase
1	09 May – 13 May	Iteration of implementing *Whack a Mole*, this time using a full software development cycle while also adding new functionality
2	16 May – 03 Jun	Iteration of developing a prototype of *Falling Words*
3	30 May – 10 Jun	Iteration of implementing additional functionality to *Toy Factory*
4	13 Jun – 17 Jun	Iteration of implementing additional functionality to *Toy Factory*
5	20 Jun – 24 Jun	Iteration of implementing additional functionality to *Falling Words, as well as thoroughly testing Toy Factory*
6	27 Jun – 1 Jul	Iteration of implementing additional functionality to *Falling Words*
7	4 Jul – 8 Jul	Thoroughly testing Falling Words.

Table 2: Schedule

The main idea is to make *a* fully playable and polished version of Toy Factory, and a good prototype of Falling Words, that can be implemented in the voice recognition project.

4. PROJECT DESIGN

This section will cover some agreements that are made beforehand concerning involved parties, facilities, and some additional information.

4.1 INVOLVED PARTIES

The involved parties include the project members, Timothy Kol and Irene Renkens, and the sponsor or company assigning the project, M3 Interactive. The project members are also part of the voice recognition project team, as the minigames project is part of this. There is close communication with this team to ensure compatibility and a good accordance.

4.2 FACILITIES

The project members provide their own laptop, while the sponsor, M3 interactive, will provide all other necessary facilities, such as software and working space.

4.3 ADDITIONAL INFORMATION

The project members will work together as a team, dividing only the smallest tasks between them, or even resorting to pair programming on occasions. All this is the project members' responsibility, as well as delivering products at set milestones.

The project members are required to be at the office for 40 hours a week, resulting in a total of 880 man-hours spent on the project. Furthermore, the project members are required to carry out the project to the best of their abilities.

Communication with the sponsor will occur during meetings or just through office conversations.

5. QUALITY ASSURANCE

We will control quality assurance by demonstrating every iteration to the sponsor and communicating what's next on the schedule. All feedback will be carefully evaluated and implemented or countered by a contradictory argument.

Testing will be done by unit testing all methods through assertions, using Visual Studio's standard testing environment. At the end of each iteration, we will also perform acceptance tests, to see which of the specified requirements are fulfilled.

Appendix D-1

Whack a Mole Schedule of Requirements 23 May 2011

The proposed application is a minigame called *Whack a Mole*. The game is inspired on the original *Whack a Mole* game, but with a change in the controls. Instead of using the mouse to hit the moles, speech recognition will be used, which means that the mole is whacked by pronouncing out loud the word it holds. The purpose of the game is to improve the English speech pronunciation of the user.

In order to have an idea of what functionality the game should have, this document is written. It provides an overview of the requirements *Whack a Mole* must satisfy. With the completion of this document, it's time to start thinking on how to realize these requirements.

i. System Requirements

The application will be tested and should run correctly on the following systems with minimum requirements (please keep in mind that the listing below does not exclude other systems).

a. Hardware
- Desktop PC or laptop
- Microphone, either internal or external

b. Platform
- Windows Vista / Windows 7
- Operating on an English language pack

c. Software
- Microsoft speech recognition, must be supported and set up through Control Panel

ii. Functional Requirements

The requirements listed below should be included in the application to be developed. The requirements are written with the *MoSCoW* model in mind. The *MoSCoW* model is a prioritization technique used to specify the importance of the requirements. There are four categories the requirements can be specified to. These are (from high to low priority): 'must have this', 'should have this if at all possible', 'could have this if it does not affect anything else' and 'won't have this but would like to have this in the future'.

a. The game

- The game must be able to recognize the user's speech with confidence rate of 80%, assuming a perfect human pronunciation
- The game must end when a time limit has exceeded or when all possible words are correctly pronounced by the user
- The game must be fun
- The game must offer some educational value in the form of feedback during playing
- The game must provide bonuses in some way
- The game must show instructions on how to play the game
- The game should have the possibility of setting a difficulty level, by varying the Mole's speed of (dis)appearing
- The game could have background music if this does not influence the speech recognition

b. The user

- The user must have enough time to read and pronounce a word
- The user must be able to see what their current game score is
- The user must be able to see how much time is left before the game ends
- The user should get feedback when a word is correctly recognized and how well it is pronounced
- The user should see how many words he has pronounced correctly and how many words are left
- The user should see their final score when the game has ended

c. The moles

- Moles must be popping out of their holes
- Moles must hold a word each time they appear
- Moles must automatically disappear into their hole again after a certain amount of time
- Moles must get hit when the word it's holding is correctly pronounced by the user
- Moles must go down and disappear into their hole when hit
- Moles must add to the score when they are hit, score added should be based on the length of the word it's holding
- Bonus moles must exist
- Moles should randomly appear
- Moles should hold randomly chosen words
- Moles should have varying speed of appearing, based on length of the word it's holding
- Moles could have a partly randomly assigned speed
- Moles could provide sound effects (e.g. when popping up)

d. Bonus moles

- Different types of bonus moles must exist
- Different types of bonus moles must provide different bonuses
- Hitting 'freeze moles' should freeze all active moles
- Hitting a frozen 'freeze mole' should not freeze active moles
- Moles that have been frozen must unfreeze automatically after a certain amount of time

- Hitting ‘angel moles’ should deduct points
- Hitting ‘devil moles’ should add points

e. The words
- Words must be clearly shown to the user
- Words should not be shown again later in the game when they’ve been correctly pronounced by the user already

iii. Usability

When the game starts, it should be clear to the user what they’re required to do. Furthermore, the game should be fully playable with the use of speech recognition only.

iv. Performance & Reliability

The game must run smoothly, meaning that the speech recognition may cause none or very little delay to the game itself. The game must be able to recognize all pre-specified words by speech recognition. When a word is incorrectly pronounced, it should not be mistaken for one of the other pre-specified words.

Appendix D-2

Toy Factory Schedule of Requirements

The proposed application is a minigame called *Toy Factory*. In the game, English words are shown, which should be accurately pronounced by the player. The words are held by toys which move toward the user on a conveyor belt in a 3D fashion, and the user should pronounce the word accurately before the toy smashes against the screen. The purpose of the game is to improve the English speech pronunciation of the user while it should also be fun to play.

In order to get an idea of the functionality the game should have, this document is written. It provides an overview of the requirements *Toy Factory* must satisfy. With the completion of this document, it's time to start thinking on how to realize these requirements.

i. System Requirements

The application will be tested and should run correctly on the following systems with minimum requirements (please keep in mind that the listing below does not exclude other systems).

a. Hardware
- Desktop PC or laptop
- Microphone, either internal or external

b. Platform
- Windows Vista / Windows 7
- Operating on an English language pack

c. Software
- Microsoft speech recognition, must be supported and set up through Control Panel

ii. Functional Requirements

The requirements listed below should be included in the application to be developed. The requirements are written with the *MoSCoW* model in mind. The *MoSCoW* model is a prioritization technique used to specify the importance of the requirements. There are four categories the requirements can be specified to. These are (from high to low priority): 'must have this', 'should have this if at all possible', 'could have this if it does not affect anything else' and 'won't have this but would like to have this in the future'.

a. The game

- The game must be able to recognize the player's speech with confidence rate of 80%, assuming a perfect human pronunciation
- The game must end when a time limit has exceeded or when all possible words are accurately pronounced by the player
- The game must be fun
- The game must be esthetically attractive (e.g. conveyor belt should be animated, different looking toys, etc.)
- The game must offer some educational value in the form of feedback during playing, giving a rating on how well a word was pronounced
- The game must be resizable
- The game should have an instruction screen prior to starting the game
- The game should provide sound effects when a word is accurately pronounced
- The game should show whether the microphone is picking up sound
- The game should have a 'game over' screen showing the final results
- The game could have the possibility of setting a difficulty level
- The game could provide bonus items/modes/etc.
- The game could have background music if this doesn't interfere with the voice recognition

b. The Toys

- Each toy must hold a word for the player to pronounce
- Toys must automatically appear on the conveyor belt by falling out of a chute positioned above the conveyor belt
- Toys must move toward the player through the movement of the conveyor belt and scale accordingly (i.e. get bigger when closer to the player)
- Toys must disappear off the conveyor belt after a certain amount of time
- Each toy should hold a randomly assigned word
- A toy should be wrapped up when the player pronounces the word it's holding accurately
- Toys should smash against the screen and disappear when the player doesn't accurately pronounce the word in time (in time means before the word has fallen off the screen because of the toy getting too close)
- Toys should appear on the conveyor belt at a random time interval
- Multiple toys should be allowed on the belt at the same time
- Toys should visually look different from each other as to improve the game's esthetical look
- Toys could provide bonuses for the player
- Toys could have varying speed concerning moving toward the player

c. The player

- The player must have enough time to pronounce the word accurately before it disappears of the screen
- The player must be able to clearly see the word a toy is holding
- The player must be able to see what their current game score is

- The player must be able to see how much time is left before the game ends
- The player should get feedback when a word is correctly recognized and how well it's been recognized
- The player should see their final score when the game has ended
- The player could be able to make use of bonus items

d. The words
- Words must be clearly (i.e. readable) shown to the user
- Words should not appear again if already pronounced accurately by the player

iii. Usability

When the game starts, it should be clear to the player what he's required to do. In order to accomplish this, clear instructions should be shown before the game starts. Furthermore, the game should be fully playable by the use of speech recognition only.

iv. Performance & Reliability

The game must run smoothly, meaning that the speech recognition may cause none or very little delay to the game itself. The game must be able to recognize all pre-specified words by speech recognition.

Appendix D-3

Falling Words Schedule of Requirements

The proposed application is a minigame called *Falling Words*. In the game English sentences and/or phrases are shown, which should be accurately pronounced by the player. Words that are accurately pronounced will drop items, which the player can catch by directing their avatar right or left.
The purpose of the game is to improve the English speech pronunciation of the user.

In order to have an idea of what functionality the game should have, this document is written. It provides an overview of the requirements *Falling Words* must satisfy. With the completion of this document, it's time to start thinking on how to realize these requirements.

i. System Requirements

The application will be tested and should run correctly on the following systems with minimum requirements (please keep in mind that the listing below does not exclude other systems).

a. Hardware
- Desktop PC or laptop
- Microphone, either internal or external
- Keyboard, mouse or track pad

b. Platform
- Windows Vista / Windows 7
- Operating on an English language pack

c. Software
- Microsoft speech recognition, must be supported and set up through Control Panel

ii. Functional Requirements

The requirements listed below should be included in the application to be developed. The requirements are written with the *MoSCoW* model in mind. The *MoSCoW* model is a prioritization technique used to specify the importance of the requirements. There are four categories the requirements can be specified to. These are (from high to low priority): 'must have this', 'should have this if at all possible', 'could have this if it does not affect anything else' and 'won't have this but would like to have this in the future'.

a. The game

- The game must be able to recognize the user's speech with confidence rate of 80%, assuming a perfect human pronunciation
- The game must end when a time limit has exceeded or when all possible sentences and phrases are accurately pronounced by the user
- The game must be fun
- The game must offer some educational value in the form of feedback during playing, giving a rating on how well a word was pronounced
- The game should provide sound effects when a word is accurately pronounced
- The game should show whether the microphone is picking up sound
- The game could have the possibility of setting a difficulty level
- The game could have background music
- The game could provide bonus items

b. The words, sentences/phrases

- Sentences/phrases consisting of words must be shown clearly to the user
- Words accurately pronounced must drop items for the player to catch
- Sentences/phrases should not be shown again later in the game when they've been accurately pronounced by the player already
- Words accurately pronounced could drop bonus items for the player
- Words inaccurately pronounced could drop 'bad' items for the player which should be evaded
- Sentences/phrases fully accurately pronounced could provide a bonus to the player

c. The player

- The player must be able to move their avatar to the right and left with the keyboard
- The player must have enough time to catch items dropped by words
- The player must be able to see what their current game score is
- The player must be able to see how much time is left before the game ends
- The player should also be able to move their avatar with the mouse
- The player should get feedback when a word is correctly recognized
- The player should see the next sentence/phrase coming up
- The player should see their final score when the game has ended
- The player could be able to let its avatar jump and/or make other moves than left and right only
- The player could be able to make use of bonus items

d. Items

- Must fall down from correctly pronounced words
- Must be possible to catch for the player
- Must disappear when they reach the bottom of the screen or are caught by the player

iii. Usability

When the game starts, it should be clear to the player what he's required to do. In order to accomplish this, clear instructions should be shown before the game starts. Furthermore, the game should be fully playable with the combination of speech recognition and keyboard, mouse or track pad.

iv. Performance & Reliability

The game must run smoothly, meaning that the speech recognition may cause none or very little delay to the game itself. The game must be able to recognize all pre-specified sentences/phrases by speech recognition. The user must be able to control its avatar by the use of a keyboard, mouse or track pad. These movements of the avatar should not cause any delays to the game.

Faculty Electrical Engineering, Mathematics and Computer Science

TU Delft

IN3405 BSc project - Two weekly progress report

Period 25 Apr 2011 to 9 May 2011

Student name 1	Timothy Kol	Student ID	1,509,217
Student name 2	Irene Renkens	Student ID	1,534,629
Student name 3		Student ID	
Student name 4		Student ID	

Company	M3 Interactive
Company supervisor	Jeffrey Jiang
TU Delft supervisor	Peter van Nieuwenhuizen
Title of assignment	Minigames for voice recognition project

Status of	seen by company	seen by TU Delft	% ready
Clear problem description	×		100
Orientation report			50
Planning	×		100
Requirements specification			14
Design specification			10
Testplan / test results			0
Coding			10
Documentation (including final report)			10

Difficulties encountered / problems solved since previous report:

Difficulties were encountered while implementing the Whack a Mole prototype: creating a game loop required some tweaking as well as accessing XAML elements from other classes. Both problems were solved, although it is not yet certain if found solutions are the best options. A problem that has been solved is Data Binding, which links XAML elements to other elements or even variables in the code. We succeeded in binding images to other images, which is very useful for synchronous animation, as well as binding elements to variables in the code-behind.

Action points next two weeks:

Two iterations are planned: adding more functionality to the Whack a Mole prototype during the week of May the 9th, and creating a prototype of a new minigame, Falling Words, during the week of May the 16th.

For both iterations we will work through a full software development cycle, which includes:
- Making a plan for the iteration
- Specifying requirements
- Designing a new or updated class diagram
- Actually implementing the game or additional functionality
- Testing the new software by performing unit tests and an acceptance test
- Evaluating the final result by discussing it with the sponsor

Faculty Electrical Engineering, Mathematics and Computer Science

TU Delft

IN3405 BSc project - Two weekly progress report

Period 9 May 2011 to 23 May 2011

Student name 1	Timothy Kol	Student ID	1,509,217
Student name 2	Irene Renkens	Student ID	1,534,629
Student name 3		Student ID	
Student name 4		Student ID	

Company	M3 Interactive
Company supervisor	Jeffrey Jiang
TU Delft supervisor	Peter van Nieuwenhuizen
Title of assignment	Minigames for voice recognition project

Status of	seen by company	seen by TU Delft	% ready
Clear problem description	×	×	100
Orientation report	×	×	50
Planning	×	×	100
Requirements specification	×		28
Design specification			20
Testplan / test results			10
Coding			15
Documentation (including final report)			10

Difficulties encountered / problems solved since previous report:

We introduced special moles for Whack a Mole. Easy ones were devil and angel moles, which just give a bonus score when hit, but implementing the freezing mole proved to be more difficult, as it had to freeze all visible moles for some time. The problem with this was that the freezing mole has no access to other moles. We solved this by introducing a new method in the MainWindow (the main class) that the freezing mole would call. This method subsequently tries to freeze all moles, as MainWindow does have access to them. This meant that every Mole needed a method that would stop animation if the Mole is visible and is not the freezing mole itself.

There were also some difficulties with setting up a testing environment for Whack a Mole in Visual Studio. We'd created a Unit test environment that allows us to perform assertions on all program units. Initially we had some problems concerning initialization of the test cases but these problems have been solved later on.

A problem that arose while starting on Falling Words, was that we now needed the speech recognition to recognize sentences instead of separate words. A list of words wouldn't work, as the recognizer can't process them fast enough. We solved this problem by using the recognizer's GrammarBuilder, which allowed for a good sentence recognition, although a limitation is that it is impossible to give the player any feedback if only part of the pronounced sentence was recognized (note: recognizing doesn't mean a word or phrase was correctly pronounced, just that the engine recognized it). Later on, we determine how well it was pronounced based on the recognizer's confidence level.

Action points next two weeks:

The original plan was to complete the Falling Words prototype before today, the 23rd of May. However, since Tuesday was a public holiday and, because our visa expired, we had to leave the country for a trip to Malaysia during Thursday and Friday, we only just got started. Therefore, it will be difficult to stick to the original plan of also completing another iteration of Whack a Mole and starting on a prototype of our own minigame while also completing the Falling Words prototype.

Also, the company now requires an updated version that is based on Whack a Mole, but has a slightly different concept. The deadline for this version is on June the 3rd, so we will be focusing on another Whack a Mole iteration during the next two weeks. The concept for this updated version is not yet decided upon, so while we wait for the company to do this, we can try to do some work on the Falling Words iteration, hoping to get a very basic working version of this by June the 3rd as well.

All this means that we have to revise our schedule for the duration of the project, which will be reflected in an updated plan of approach.

Faculty Electrical Engineering, Mathematics and Computer Science

TU Delft

IN3405 BSc project - Two weekly progress report

Period 23 May 2011 to 6 Jun 2011

Student name 1	Timothy Kol	Student ID	1,509,217
Student name 2	Irene Renkens	Student ID	1,534,629
Student name 3		Student ID	
Student name 4		Student ID	

Company	M3 Interactive
Company supervisor	Jeffrey Jiang
TU Delft supervisor	Peter van Nieuwenhuizen
Title of assignment	Minigames for voice recognition project

Status of	seen by company	seen by TU Delft	% ready
Clear problem description	×	×	100
Orientation report	×	×	50
Planning	×	×	100
Requirements specification	×		50
Design specification			50
Testplan / test results			10
Coding			30
Documentation (including final report)			10

Difficulties encountered / problems solved since previous report:

In the week of 23 May we finished a prototype of the minigame Falling Words. We encountered some difficulties regarding the phrases, since items need to fall out of every word, and you cannot simply obtain the absolute position of a UI element in WPF. We ended up using a grid for every word to position the items that fall out of it in this grid as well. We used the grid's rendered width to calculate the margin needed for the next word's grid.

In the week of 30 May we've created a new version of Whack a Mole called Toy Factory. Whack a Mole involved moles popping out of holes and the player needed to hit them with hammers by correctly pronouncing words. In Toy Factory there are toys moving towards you on a conveyor belt that need to be wrapped up (by correctly pronouncing words) before they smash on your screen. Part of the code of the previous version could be re-used but because of more complicated animations involved, we had to figure out a way to do those too. For the conveyor belt we decided to have five images and loop through them constantly, which causes the illusion of the belt moving. For the toys, we used RenderTransformGroups containing translation and scaling transformations.

The final minigames should be playable in resized windows, so we had to make the Toy Factory resizable. The most challenging part in doing this was the resizing of the toys because their position and size continually changes. We ended up setting all the variables related to the transformations of the toys based on the screen width/height . This means that when the screen size changes while the game is running, the transformations on the toy will change accordingly.

Action points next two weeks:

- Think of more functionality to add to the Toy Factory and Falling words mini games
- Implement more functionality to Toy Factory and Falling Words
- Implement frame independent movement, new graphics and a new animation for the conveyor belt for Toy Factory
- Refactoring code to the factory design pattern in order to improve the code's structure

Faculty Electrical Engineering, Mathematics and Computer Science

TU Delft

IN3405 BSc project - Two weekly progress report

Period 6 Jun 2011 to 20 Jun 2011

Student name 1	Timothy Kol	Student ID	1,509,217
Student name 2	Irene Renkens	Student ID	1,534,629
Student name 3		Student ID	
Student name 4		Student ID	

Company	M3 Interactive
Company supervisor	Jeffrey Jiang
TU Delft supervisor	Peter van Nieuwenhuizen
Title of assignment	Minigames for voice recognition project

Status of	seen by company	seen by TU Delft	% ready
Clear problem description	×	×	100
Orientation report	×	×	50
Planning	×	×	100
Requirements specification	×	×	67
Design specification			67
Testplan / test results			20
Coding			67
Documentation (including final report)			10

Difficulties encountered / problems solved since previous report:

These two weeks, we spent our time on improving the Toy Factory minigame, as the company required a working, polished version by the end of June. We did encounter problems.

The first problem we faced was the animation of the toys compared to the conveyor belt. The belt was animated by rendering five frames of the belt's image after eachother, which simulated the movement of lines coming down the belt. However, it proved to be extremely difficulty to synchronize the belt lines with the toy's movement, which caused the whole animation to look wrong. We solved this by implementing the belt's animation in the code, actually moving lines down the screen in the same fashion the toys were animated.

To make the game more varied, we also implemented a bonus feature, where a toy appeared on the belt in the form of some dynamite. When the word on the dynamite is correctly pronounced, the belt breaks and all animation should freeze. This was done by simply not updating the belt's lines and toys, and halting all toy generation.

Because of new, detailed toy images, another big problem came up regarding lag when new toys were spawned. Originally, when a toy was spawned, it had to recreate the images it used, which usually took some time. We noticed that when a toy with the same image was already on the belt, there was no such lag. From this, we concluded that in WPF, when images are already created, even if they are hidden, it requires much less performance to create another image from the same source. Therefore, we decided to create a resourcemanager, which would pre-load all required images at the start of the game. This proved to be a great idea, as the spawning no longer had an effect on the game's smoothness.

Finally, we also implemented a factory design pattern for creating new toys, to improve the code's structure.

Action points next two weeks:

During the week of 20 June, we will be thoroughly testing and cleaning up the Toy Factory minigame, as we've been kind of lacking in this area. Furthermore, we hope to be able to do some work on Falling Words as well, but it depends a bit on the company's priorities.

The week of 27 June, we will focus on Falling Words, making it a decent prototype, as we expect to have all work done on Toy Factory by then.

We did a lot more work on Toy Factory than originally planned, partly due to the company's requirements to deliver a nicely polished version by the end of this month, but we still hope to make a proper version of Falling Words as well.

If may be possible that Falling Words will be changed, like Whack a Mole changed to Toy Factory, but we should be able to re-use a lot of our code if that happens.

Faculty Electrical Engineering, Mathematics and Computer Science

TU Delft

IN3405 BSc project - Two weekly progress report

Period 20 Jun 2011 to 4 Jul 2011

Student name 1	Timothy Kol	Student ID	1,509,217
Student name 2	Irene Renkens	Student ID	1,534,629
Student name 3		Student ID	
Student name 4		Student ID	

Company	M3 Interactive
Company supervisor	Jeffrey Jiang
TU Delft supervisor	Peter van Nieuwenhuizen
Title of assignment	Minigames for voice recognition project

Status of	seen by company	seen by TU Delft	% ready
Clear problem description	×	×	100
Orientation report	×	×	50
Planning	×	×	100
Requirements specification	×	×	90
Design specification			90
Testplan / test results			80
Coding			90
Documentation (including final report)			20

Difficulties encountered / problems solved since previous report:

During the last two weeks, we implemented unit tests for all relevant methods in the Toy Factory minigame. This allowed us to filter out some bugs.

Furthermore, we have picked up working on Falling Words again, but have run into a problem. The game turns out to be rather boring in the current concept, and this is one thing a game like this shouldn't be. We will discuss the game concept during a meeting to come up with a better idea.

We haven't been able to do a lot of work during this last week as the company was moving to a different location, so the workplaces in the new office still had to be set up. Furthermore there was no connection to the company's server and internet available yet which made it difficult for us to do some useful work.

Action points next two weeks:

There is only one week left, during which we will focus on making a new prototype for a minigame, which should be based on sentences like Falling Words, but shouldn't be as boring. We could also make some sort of improvement on the current concept to still retain the original idea of words falling down.

We will also do some final testing on Toy Factory to ensure a quality product.

Appendix F

SIG Feedback

First Feedback (Friday, June 17, 2011)

"[Aanbevelingen]
De code van het systeem scoort 2 sterren op ons onderhoudbaarheidsmodel[1], wat betekent dat de onderhoudbaarheid van de code onder het gemiddelde ligt. De score wordt naar beneden gehaald door de de Module Coupling, de Unit Size en de Unit Complexity.

Voor Module Coupling wordt er gekeken naar het percentage van de code wat relatief vaak wordt aangeroepen. Normaal gesproken zorgt code die vaak aangeroepen wordt voor een minder stabiel systeem omdat veranderingen binnen dit type code kan leiden tot aanpassingen op veel verschillende plaatsen. Omdat er in dit systeem maar een beperkt aantal bestanden zijn, duidt de lage score voor deze meting op een of meerdere bestanden welke verantwoordelijk zijn voor een groot deel van de functionaliteit (het zogenaamde 'God'-classen anti-pattern[1]). Het opsplitsen van dit soort bestanden in classen met een specifieke functie zorgt ervoor dat de aparte delen van de functionaliteit makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden worden. In dit geval bevat de file 'Main.xaml.cs' ruim 70 procent van de code van het systeem en is deze ook nog eens gekoppeld met bijvoorbeeld de 'ToyFactory'. Het opsplitsen van deze class in aparte classes voor bijvoorbeeld de interface code en het data-model zou toekomstige wijzigingen makkelijker maken.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt er ook weer voor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. In dit systeem is de langste methode 'SetWords', hierin worden onder andere een lijst van woorden geïnitialiseerd. Deze methode kan drastisch worden verkleind door de daadwerkelijke woorden op te nemen in een apart configuratie bestand. Door dit bestand in te lezen wordt dezelfde functionaliteit geleverd, maar is de code makkelijker te lezen en te begrijpen. Daarnaast is ook de lijst van woorden makkelijker aan te passen zonder dat het systeem opnieuw gebouwd hoeft te worden. In andere langere methodes binnen dit systeem, zoals bijvoorbeeld 'main_Loop', zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. In dit systeem geven regions binnen de methodes al aan waar dit soort autonome stukken te vinden zijn. Het is sterk aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, makkelijker te testen en daardoor eenvoudiger te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

[1] Twee uit vijf sterren.

Als laatste nog de opmerkingen dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen."

Second Feedback (Tuesday, August 2, 2011)

"[Hermeting]
In de tweede upload zien we dat de omvang van het systeem is gedaald en dat de score voor onderhoudbaarheid is gestegen. Deze stijging in onderhoudbaarheid is deels toe te schrijven aan het introduceren van een 'ResourceManager' welke een aantal taken van de 'Main.xaml.cs' overneemt. Daarnaast kunnen we zien dat op verschillende plaatsen langere methodes zijn opgesplitst, hierdoor is er een lichte stijging in de scores voor zowel Unit Size als Unit Complexity te zien. Omdat er voor de opgesplitste methodes ook weer langere methodes zijn geïntroduceerd (bijvoorbeeld de 'backgroundWorker_DoWork'-methode in ResourceManager.cs) is er uiteindelijk maar een relatief kleine stijging in de uiteindelijke score te zien.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject. Wat nog opvalt is dat er nog steeds veel commentaren zoals '// Animate stamp if necessary' te vinden zijn binnen de langere methodes, het extraheren van deze aparte stukken functionaliteit zal de onderhoudbaarheid nog verder verbeteren."