Multimodal Interfaces

JPotter

Report

Department:	Information and Communication Technology
Field:	Computer Sciences
Keywords:	MMI, gesture, speech, sphinx, wiigee, wiimote, multimodal
Date:	25.02.2011 – 11.04.2011
Students:	Christophe Dorthe, Thomas Michel, Maxime Reymond
Supervisors:	O. Abou Khaled, J. Bapst, D. Lalanne, E. Mugellini

EIA-FR
HTA-FR

JPotter

Table of Contents

1 Introduction

This chapter introduces the reader to the project and its context and scope.

1.1 Context

During the "Multimodal interfaces" course, which is an optional course we chose to take, we had not only the chance, but also the obligation to pursue a project. This is in the interest of applying the theoretical elements we learned during the course.

There were essentially two constraints. First, the project had to be a game and second we had to use at least two modalities.

Our game concept will be explained further in this report but in the interest of keeping the reader's attention, here is a brief description of our game. We decided to model a character from the Harry Potter saga. We want to let the user say incantations and perform the gesture corresponding to the spell using a wand.

1.2 Structure of this report

This report is divided into five chapters, including this introduction, that describe the different phases of this project. It is terminated by a few appendices.

The Analysis chapter identifies the different issues we encountered and a few choices we made from the start. The Design chapter describes the game itself and designates the modalities we chose. The Implementation part illustrates the implementation of our prototype and lists the libraries we used.

The report is concluded by a comparison of what was achieved in JPotter with the main goals. The conclusion also unfolds possible enhancements.

2 Analysis

This chapter explains the different directions we took to solve the variety of problems we encountered. Since we did not know much about multimodal interfaces when we started, we did not even know what technology should be used to implement our prototype. Therefore we chose to spend some time to find out what was available.

The next subchapter, Technologies, describes the paths we took to determine the technology to use for this project.

The following subchapters essentially compare Java and .NET on different levels.

2.1 Technologies

Considering the fact that we had to submit a description of our application on the first week, and given the fact that we liked the name JPotter, we inevitably settled with Java. We knew that we would need a speech recognition library and a library to interact with the Wiimote.

We got to the conclusion that the only recent and usable speech recognition library for Java was Sphinx [1]. We also heard during the course that it was not advised to use that library because they had issues during the previous years.

Regarding the wiimote, we found a library called Wiigee [2].

Due to the fact that it was not advised to use Sphinx, we thought we should try another technology. We played a bit with the Windows 7 built-in speech recognition system and honestly we were fairly amazed by how well it was working. Incidentally, we decided to investigate what C# had to offer too. The speech recognition library is built-in Windows and we found a library called WiimoteLib [3] for .NET.

2.2 Sphinx versus Windows Speech Recognition

Let us start with a word about how we thought we would use speech recognition in our application. As we will be able to pronounce incantations, it is essential to define these words since most of them are not part of any common language such as French of English. It is fundamental that the library we use is able to recognize these magic words properly.

2.2.1 Training

It was very easy to train the Windows Speech Recognition programme because it offers all the tools for it and they are very neat and easy to use. That was actually extremely convenient.

Unfortunately we cannot say the same thing about Sphinx. Indeed, although it does offer a tool for training, named SphinxTrain, it is by far much more complex to use than the Windows tool and at this time there is no java version for it. SphinxTrain is written in C and we also had to run some Perl scripts in order to setup the trainer.

2.2.2 Global Usability

We found that it is inconveniently burdensome to get SphinxTrain to run but in the end it is capable of recognizing speech.

2.3 Wiigee versus WiimoteLib

The first thing we would suggest to anyone who wants to use a Wiimote in their application is to watch the Wiigee demonstration video. It looks amazing and in the case of our project it illustrates exactly what we want to do. There is also a very short and absolutely comprehensive code example in the download section of Wiigee's website. Wiigee looks a bit too perfect to be true. There was indeed a problem although it was not directly Wiigee itself but more about the BlueCove [4] library. BlueCove is a java library for Bluetooth and each of us had to install some specific driver in order to have BlueCove working. One of computers still does not even work with it, although in that very case it may as well be more of a computer problem than a BlueCove issue.

On the other hand, our experience with WiimoteLib was a bit more chaotic. There was no issue with the Bluetooth driver though. That library seemed to work well; the main problem was that we could not find how to easily get some gesture recognition going. We could have implemented it by ourselves but that really was not the aim of this project as the time we had was rather limited.

2.3.1 Global Usability

The only big issue with Wiigee was to manage BlueCove and have it recognize our Bluetooth devices. Apart from that, it was really easy and intuitive to use Wiigee.

We found that using WiimoteLib was fairly hard compared to Wiigee. We did not manage to easily find good running examples on the Internet.

2.4 Summary

We are confident that we did try these two approaches quite in depth, because it took us three weeks to test these different technologies and to finally a decision. The decisive factor that made us favour Java over .NET was how amazing the Wiigee library is, and we do not regret it. Sphinx comes with some default configuration that happens to work well for our game.

The next chapter will explain the game concept and describe the multimodal interactions.

3 Design

This chapter describes the game concept in depth as well as the interaction modes.

3.1 The game

Our game can be seen as a model of the wizard world in Harry Potter. We assume the reader is familiar with Harry Potter; if not, then a quick Google query will get you up to speed. Harry Potter is basically a sorcerer.

In JPotter, you are Harry Potter the wizard. You will fight different enemies by casting spells at them. The game is turn-based and enemies will also cast spells at you, although in that case you only see the effect on the game's window.

The player has a Wiimote to use as a wand. They can target an enemy using the Wiimote and then start casting the spell by pronouncing a valid incantation and by performing the corresponding gesture.

3.2 Modalities

In the world of magic, the most common way to accomplish something is by casting a spell. It is inadequate to use words such as "see" and "usually" when referring to the magic world since it is not real. We cannot really say we have seen anything like this happen, but for the sake of this report we assume it is not a problem.

What we usually see in the magic world (it sounds odd, right?) is a magician moving his wand and uttering peculiar words. More technically, they use speech and gesture to produce a spell.

3.2.1 Speech

Having described our game, it sounds logical to use speech as part of our game since it is part of a spell. Speech will be used for incantations and so we will have to define what spells are available and how to pronounce them.

3.2.2 Gesture

The magician has to perform a valid gesture that has to match an incantation. There will be a time window during which both the incantation and the gesture have to be performed. It does not matter which one is performed first. Ideally they would have to be done at the same time to better reflect the magical word, but we are not magicians, are we?

3.3 Architecture

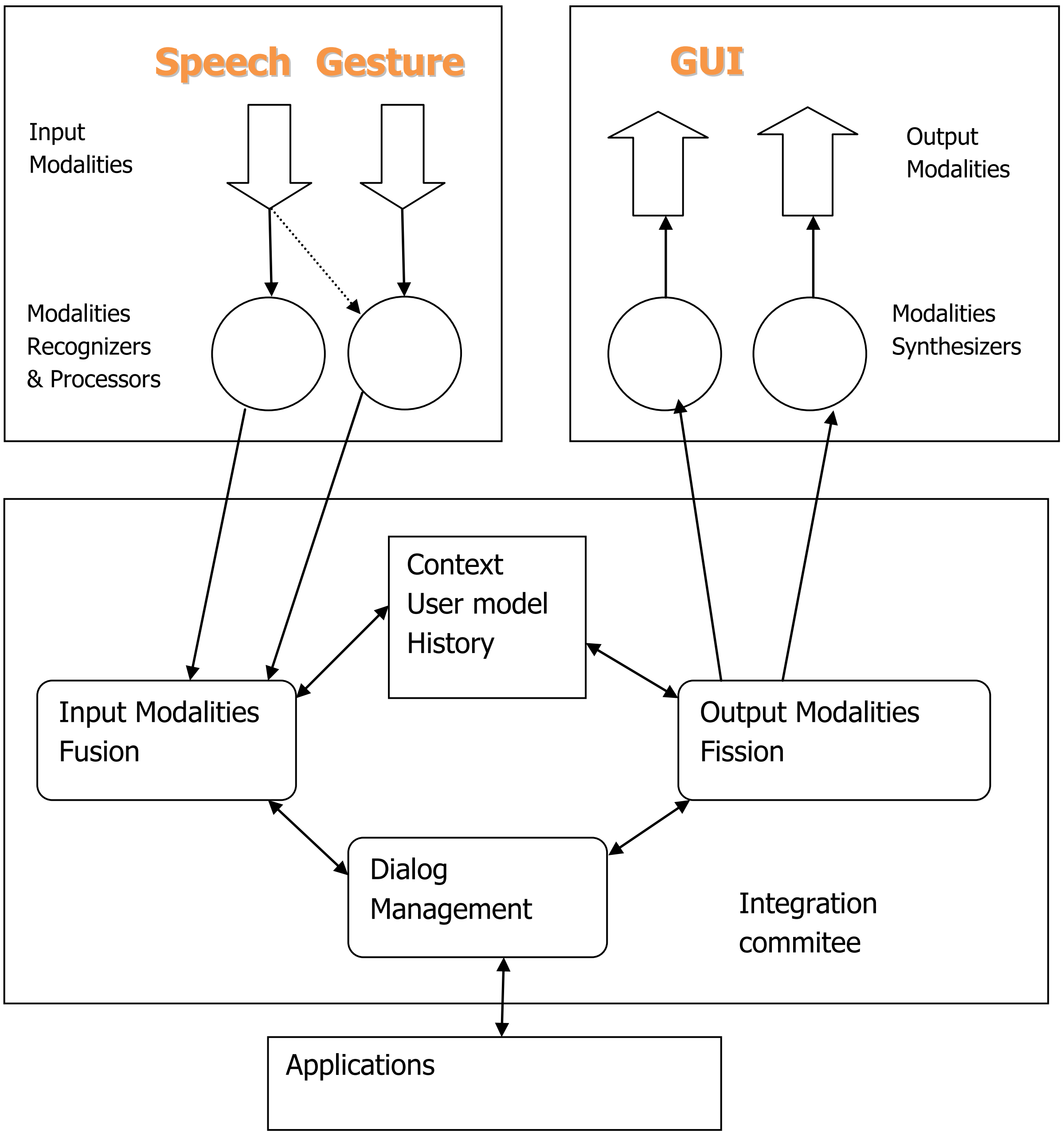

Figure 1: multimodal architecture

Figure 1 above shows the architecture of our application. In the game, the gesture and speech input modalities are handled by the GestureManager and IncantationManager classes respectively. Fusion of input modalities is handled by FusionManager. Fission and dialog management are handled by the IHM and Game classes. Figure 2 shows the class diagram of our application, although it is pretty small. The image is available on the CD.

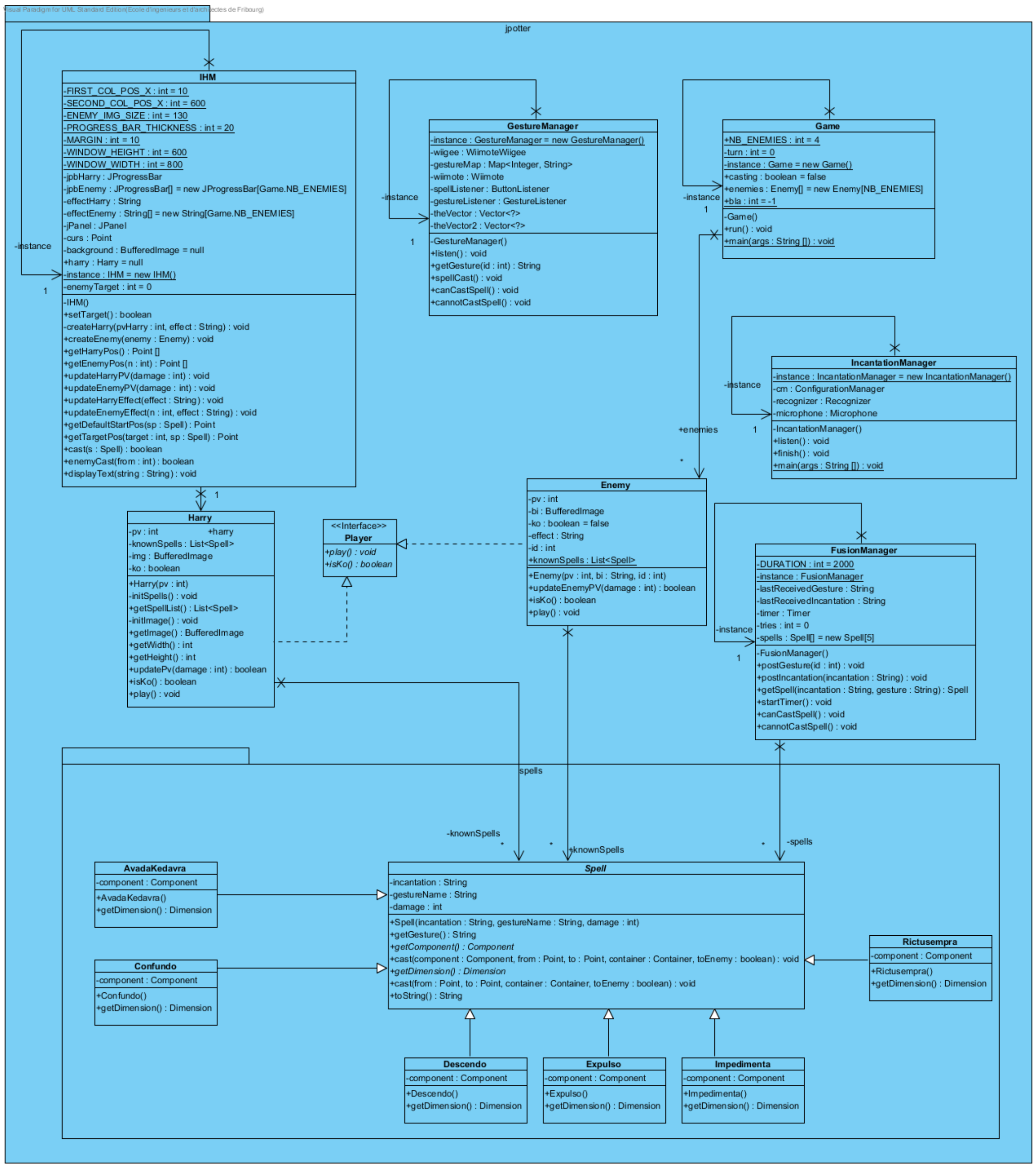
jpotter
IHM
-FIRST_COL_POS_X : int = 10
-SECOND_COL_POS_X : int = 600
-ENEMY_IMG_SIZE : int = 130
-PROGRESS_BAR_THICKNESS : int = 20
-MARGIN : int = 10
-WINDOW_HEIGHT : int = 600
-WINDOW_WIDTH : int = 800
-jpbHarry : JProgressBar
-jpbEnemy : JProgressBar[] = new JProgressBar[Game.NB_ENEMIES]
-effectHarry : String
-effectEnemy : String[] = new String[Game.NB_ENEMIES]
-jPanel : JPanel
-curs : Point
-background : BufferedImage = null
+harry : Harry = null
-instance : IHM = new IHM()
-enemyTarget : int = 0
-IHM()
+setTarget() : boolean
-createHarry(pvHarry : int, effect : String) : void
+createEnemy(enemy : Enemy) : void
+getHarryPos() : Point []
+getEnemyPos(n : int) : Point []
+updateHarryPV(damage : int) : void
+updateEnemyPV(damage : int) : void
+updateHarryEffect(effect : String) : void
+updateEnemyEffect(n : int, effect : String) : void
+getDefaultStartPos(sp : Spell) : Point
+getTargetPos(target : int, sp : Spell) : Point
+cast(s : Spell) : boolean
+enemyCast(from : int) : boolean
+displayText(string : String) : void
-instance
1
GestureManager
-instance : GestureManager = new GestureManager()
-wiigee : WiimoteWiigee
-gestureMap : Map<Integer, String>
-wiimote : Wiimote
-spellListener : ButtonListener
-gestureListener : GestureListener
-theVector : Vector<?>
-theVector2 : Vector<?>
-GestureManager()
+listen() : void
+getGesture(id : int) : String
+spellCast() : void
+canCastSpell() : void
+cannotCastSpell() : void
-instance
1
Game
+NB_ENEMIES : int = 4
-turn : int = 0
-instance : Game = new Game()
+casting : boolean = false
+enemies : Enemy[] = new Enemy[NB_ENEMIES]
+bla : int = -1
-Game()
+run() : void
+main(args : String []) : void
-instance
1
IncantationManager
-instance : IncantationManager = new IncantationManager()
-cm : ConfigurationManager
-recognizer : Recognizer
-microphone : Microphone
-IncantationManager()
+listen() : void
+finish() : void
+main(args : String []) : void
-instance
1
+enemies
*
Harry
-pv : int
-knownSpells : List<Spell>
-img : BufferedImage
-ko : boolean
+Harry(pv : int)
-initSpells() : void
+getSpellList() : List<Spell>
-initImage() : void
+getImage() : BufferedImage
+getWidth() : int
+getHeight() : int
+updatePv(damage : int) : boolean
+isKo() : boolean
+play() : void
+harry
1
<<Interface>>
Player
+play() : void
+isKo() : boolean
Enemy
-pv : int
-bi : BufferedImage
-ko : boolean = false
-effect : String
-id : int
+knownSpells : List<Spell>
+Enemy(pv : int, bi : String, id : int)
+updateEnemyPV(damage : int) : boolean
+isKo() : boolean
+play() : void
FusionManager
-DURATION : int = 2000
-instance : FusionManager
-lastReceivedGesture : String
-lastReceivedIncantation : String
-timer : Timer
-tries : int = 0
-spells : Spell[] = new Spell[5]
-FusionManager()
+postGesture(id : int) : void
+postIncantation(incantation : String) : void
+getSpell(incantation : String, gesture : String) : Spell
+startTimer() : void
+canCastSpell() : void
+cannotCastSpell() : void
-instance
1
spells
-knownSpells
*
*
+knownSpells
*
-spells
Spell
-incantation : String
-gestureName : String
-damage : int
+Spell(incantation : String, gestureName : String, damage : int)
+getGesture() : String
+getComponent() : Component
+cast(component : Component, from : Point, to : Point, container : Container, toEnemy : boolean) : void
+getDimension() : Dimension
+cast(from : Point, to : Point, container : Container, toEnemy : boolean) : void
+toString() : String
AvadaKedavra
-component : Component
+AvadaKedavra()
+getDimension() : Dimension
Confundo
-component : Component
+Confundo()
+getDimension() : Dimension
Rictusempra
-component : Component
+Rictusempra()
+getDimension() : Dimension
Descendo
-component : Component
+Descendo()
+getDimension() : Dimension
Expulso
-component : Component
+Expulso()
+getDimension() : Dimension
Impedimenta
-component : Component
+Impedimenta()
+getDimension() : Dimension

Figure 2: class diagram

3.4 CARE and CASE models

The CASE model indicates that we will have to handle a “synergistic” kind of fusion because the incantation and the gesture happen in parallel and are not completely independent.

According to the CARE model, our use of the gesture and speech is of type “Redundancy” because neither the speech nor the gesture is stronger than the other in the decision-making process; both are equivalent although needed to perform the spell.

3.5 Summary

This section explained how we want the game to run and how the player is supposed to interact with it. It also provides the reader with the technical architecture of our game. The next section will describe details of the implementation of the game and the libraries we used.

4 Implementation

In this chapter we reveal how we implemented the different modalities we used.

Figure 3 below shows what the game window looks like.

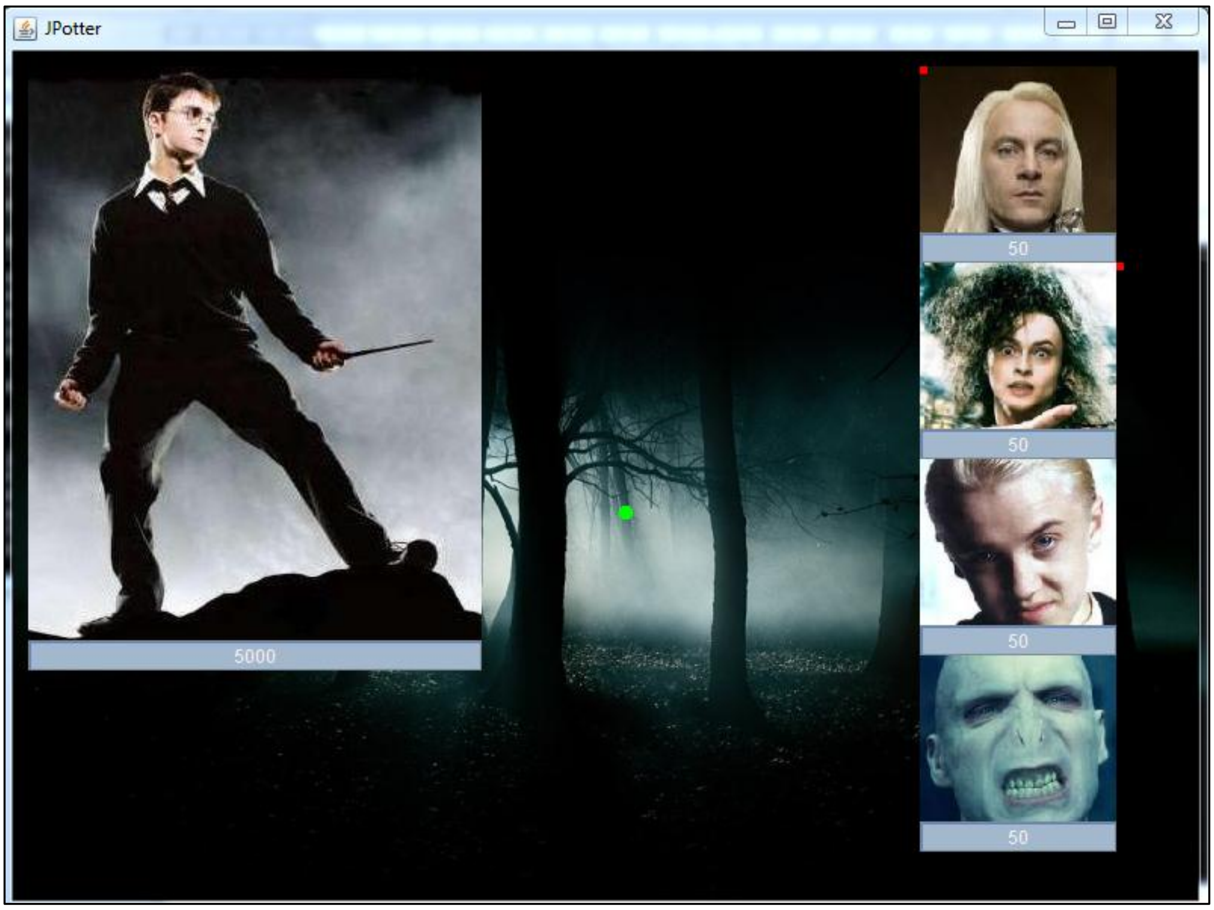

Figure 3: game window

The player is represented on the left hand side. The enemies you are currently fighting are on the right hand side. The green dot in the middle of the screen is the cursor we chose for the user to select the enemy who will be affected by the spell cast by the player.

4.1 Libraries used

This subsection lists the different libraries we used to handle the multiple modalities used in this game and the animations. Indeed, we attempted to animate the game to make it look slightly fancier.

4.1.1 Sphinx4

Sphinx4 [1] is the library we used to handle speech recognition. In order to use it, we downloaded it from their website and added the relevant libraries to the classpath of the application.

The HelloWorld application available from their website was our starting point to try out Sphinx4. We started from that HelloWorld and augmented it to match our needs for JPotter.

All we had to do was to define a dictionary with the list of the valid incantations and a grammar file. Everything is well explained on the website regarding that part and this really did not cause any issue.

4.1.2 Wiigee

Wiigee [2] is the library we used to handle the Wiimote and to recognize gesture. The library is available for downloading on the website and, in order to make it work, we had to add the “wiigee-lib” and “wiigee-plugin-wiimote” jar files to the application classpath.

Again, there is a very easy to understand example code on the website that got us started. We adapted it to allow the player to select a target and to recognize our own gestures.

4.1.3 TimingFramework

The goal of this project was by no mean to spend time dealing with 2D graphics. We still wanted our game to have some fancy effects and therefore we used TimingFramework [5] which is a very nice library to handle animation-related actions.

When we started the application, the library version was 1.0 and at the time of writing, it looks like they have updated it, although the library was released in 2008. If you want to get it from the website’s download section, you are looking for the “classic” version of the library.

4.2 Feedbacks

It is important to provide feedbacks to the user, therefore we provided some. In JPotter, it is quite easy to perform a spell incorrectly. Incidentally, we wanted to use speech synthesis to provide a message to the user when that happened. That was our first intention but eventually we decided to simply display a message on the screen that says the spell failed. When a spell is successfully cast though, the Wiimote is vibrating to indicate the spell is being cast to the enemy.

Another thing that is important to the user, since the game is turn based, is to know when to play. When the user can play, the application issues a text message on the screen to indicate the user can play.

4.3 Summary

The implementation was not a big deal as far as we are concerned. There was good source codes available that we could use as a starting point and it really helped a great deal.

The next chapter, which happens to be the last one, is the conclusion of this project. In that section, we compare the initial goals with what we have achieved. We also list possible future work on this game.

5 Conclusion

We believe the project is a success! We managed to model a sorcerer who can cast spells based on an incantation and a certain gesture. We see our project more as a prototype than as an actual game. Of course we did implement a game logic but it is not very entertaining as it is right now. Time was an issue and it is possible that we spent too much time on the analysis part. If we had not done so perhaps we would have had time to go further into the game design and we would have been able to make the game a little bit more entertaining.

Nonetheless, we believe the concept of our game is good and has a lot of potential to be turned into a real game. The next subsections list the different elements we identified that could make this game even greater than it already is.

5.1 Enhancing JPotter

Currently the player is able to cast spells at enemies, and enemies are able to retaliate. If the player is not good enough at invoking the spells, he will fail and thus the enemies will kill him before he kills them all.

The following is a list of additional features we could have added if we had had (much) more time.

5.1.1 Training mode

In the current version, the player needs to know the spells incantations and corresponding gesture in order to cast a spell. Also, the player is required to know a certain amount of all the available spells.

In the Harry Potter saga, Harry goes to a school of Witchcraft and Wizardry. He learns spells, among other things, at this school.

We could imagine having several places in the game where Harry could go. One of them would be a place where the user has to spend some time and learn spells after he successfully pronounced the incantation and performed the proper gesture a certain amount of times.

5.1.2 Levelling

We could think of JPotter as a role playing game where the player evolves in the magical world. He would start at level one and as he progresses by killing monsters, he would gain experience and would gain levels. Spells Harry can learn could require a certain level in order to be used.

5.1.3 Player versus Player

We believe it would be really fun to be able to play not only versus computer controlled enemies but also against other players.

5.1.4 Protecting spells

In the current version, all spells are offensive spells that deal damage to the opponent. But in the wizardry world, there are also defensive spells that we could implement in the game. They could act as healing spells or small bonuses.

Appendix A: CD-ROM content

This CD-ROM contains additional documentation to the project. Below is the list of all the available documents and where to find them on the CD structure.

- Videos
  We made two videos for this project. One called **jpotter.wmw** that is a demonstration of our prototype. The second one is called **jpotter_outtakes.wmw** and this one contains funny scenes that happened while we were making the videos. These are not for official publishing. The demonstration video comes with a descriptive file called **video_description.pdf**. All these files are inside the **videos** folder at the root of the cd.
- Report, class diagram, presentation and installation documents
  The **documents** folder contains the **jpotter_report.pdf** file which is our final report for this project. It also contains **class_diagram.png** which is provided for convenience if the reader is interested in having a bigger view on this specific diagram. The slides of the presentation we showed is also available in this folder, it is named **jpotter_presentation.pdf**. Finally, we provided a document to explain how to install and run JPotter, it is called **install_jpotter.pdf**.
- Source code and binaries
  This project was developed using the Eclipse IDE. We put the project files inside the **jpotter** folder and it contains the source code, the binaries and the libraries we used to develop JPotter. Source files are inside the **src** subfolder, we provided an executable jar file for convenience, called **JPotter.jar**. Last but not least, libraries are inside the **lib** subfolder.

Note that each document is also provided in its editable form.

Appendix B: References

[1] Carnegie Mellon University. Sphinx-4 Application Programmer's Guide, 2004, http:// http://cmusphinx.sourceforge.net/sphinx4/

[2] Benjamin Poppinga and Thomas Schlömer. Wiigee - A Java-based gesture recognition library for the Wii remote, 2008, http://www.wiigee.org/

[3] Brian Peek, WiimoteLib - Managed Library for Nintendo’s Wiimote , 2009, http://wiimotelib.codeplex.com

[4] BlueCove Team, BlueCove – Java library for Bluetooth, 2008, http://bluecove.org

[5] Chet Haaste and Romain Guy, Timing Framework - a library for making Java animation and timing-based control easier, https://timingframework.dev.java.net/

Appendix C: Table of Figures