KMPC

A Kinect Media Player Controller

University of Illinois at Urbana Champaign
CS428 : Software Engineering II
Prof. Danny Dig
Spring 2012

Author: KMPC Group

Table of Contents

Chapter 1 Brief description of project

1.1 KMPC

Our project is to make a Kinect media player controller which is a piece of software that can control all kinds of media players by providing varieties of gestures to the Kinect camera sensor. This is useful when movies are playing on a computer, and people sit on sofas far away from it because people don't have to go back and forward to control the media player, e.g., turn the volume up. They can control the media player by making all kinds of gestures, for example, lifting up the right hand up will turn the volume up.

1.2 The Team

We are a group of undergraduate students currently studying Computer Science and Computer Engineering at the University of Illinois at Urbana-Champaign. Our team of seven has managed all development coordination through wiki-ing, weekly meeting, and Subversion source controlling, with attempts at pair programming whenever possible.

The team consists of the following members:

1. Dawen Huang (dhuang22@illinois.edu)
2. Wanya Huang (huang139@illinois.edu)
3. Lik-heng Philip Chan (chan32@illinois.edu)
4. Cyrus Chan (chan15@illinois.edu)
5. Darshan Sanghani (dsangha2@illinois.edu)
6. Sandeep Bhattaram (bhattar1@illinois.edu)
7. Yogesh Italia (italia1@illinois.edu)

1.3 Explanation Of Terms

Kinect - a motion sensing input device by Microsoft for the Xbox 360 video game console and Windows PCs.

Microsoft Visual Studio Ultimate 2010 - Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop console and graphical user interface applications along with Windows Forms applications

1.4 Note To The Reader

The concepts in the document assume familiarity with the following programming languages, techniques, technologies, standards, libraries and frameworks:
Kinect for Windows SDK
UML 2.0
Coding4fun
Win32 API
Object oriented architecture and design

Chapter 2 Process

2.1 Overview

Our team uses a combination of Rational Unified Process (RUP) and Extreme Programming (XP). Our software development process is kind of unique since we are not strictly following a typical software development model. The main reason for that is because this is a class project, and we are not required to do business modeling, which is an essential procedure in Rational Unified Process (RUP). RUP is an iterative software development process framework that can guide us through the whole project. We also need to do our project in a way such that we could be graded on our work. Also another factor we have to consider is we have an unusually large team of 7 members. XP is a well-known iterative software development methodology that we learned for a whole semester so we are all familiar with it. Iterative process fits our project every well because this is a relatively small project, and we need to present our work after each iteration. Our development process, as other typical development processes, includes several activities and steps. They are iterative development, testing, refactoring, and collaborative development.

2.2 Iterative development

We decided to borrow the idea of Pair Programming from Extreme Programming development process. Because we have seven people in the team, it is hard to keep all the team members updated, especially if we are dividing the group into two teams for the two separate components. To facilitate the team communication, we decided to use Pair Programming and switch pairs every meeting. For example, if two developers worked on use case A last meeting, then one of them will continue to work on it with another developer to make sure the new developer understand the progress and how to continue to work on the code. Each developer will work on the same use case for two meetings, and then switch to other use cases. Because we further divide the two groups in the team into coding group and testing group, we have three pairs in the development team. Therefore, it is possible for everyone to switch groups every two meetings. We think this is the easiest way to keep team members on the same page in a large team. We also record what we did for each meeting so that we can find out who is responsible for which modules.

2.3 Testing

We did not choose to use the waterfall model because we want to test our system as we implement different modules. There are five stages in the waterfall model: Requirements, Design, Implementation, Verification and Maintenance stages. Coding is done in the Implementation stage, and Testing is done in the Verification stage. The waterfall model separate the coding stage and the testing stage, which means we won't test our system until we finish writing all the code. This is not a good plan for our project team. Because we have seven

people in the team, we would like to split the team into two groups so that we can work in parallel on different modules of the system, and then integrate them later. If we use the waterfall model, we will integrate our modules before we test any of them. This process greatly increases the difficulty of modifying our code if we find errors in the testing stage, because it is hard to determine whether the error is inside a module or due to the wrong integration. The RUP designs the structure of the system and split the system into modules early in the process. Once developers all understand the big picture of the system, they can work on different modules at the same time. Modules are tested by the testing group immediately after development of that module is completed by the development groups. Because the modules are tested before integration, although it might take longer to finish implementing individual modules, we can make sure they will work correctly. Therefore, we can eliminate some sources of error when we perform integration tests.

2.4 Refactoring

Throughout the whole development process, we will have consistent code reviews. Different pairs will look at the other pairs' code and suggest what can be done better. When better solutions to the problem are found, or when codes are too messy and unmanageable, we will refactor our code. Refactoring means to modifying the code so the code is more simple, easier to understand and more maintainable, but at the same time leaving the behaviour unchanged. In our project, refactoring usually happens when testing pairs create tests for the corresponding modules. This is because testing pairs first look through the code and create tests. In the process, they might able to think of a better way to solve the problem, or be able to give advice to the development team for improvement and refactoring. Pairs who develop the program will also refactor their code when they find out their code is unmanageable.

2.5 Collaborative development

Collaborative software development is done by peer review. During the milestone 5, another group and our group are assigned to review each other code. Each group is required to send some specific parts of code to each other. Then, both of us will write a checklist about what code smell they find for each other. In the end, we will meet each other and walk through the checklist.

Chapter 3 Requirement & Specifications

3.1 Introduction

- Purpose: Many people connect their PC to their big flat screen TV to watch movies. Usually HDMI cords are not long enough for people to put their PCs on the sofa. So people need to get up from the sofa each time when they want to control the movie (play, stop, pause etc). It will be wonderful if there is a software that can use a kinect camera to detect hand gestures and control the PC.

- System overview: The application works as a mediator between Kinect and the media player. Kinect recognizes the gesture, communicates with the application, the application translates the gesture to a command to be fed to the media player, media player then executes that command.

3.2 Overall description

- Product perspective: The aim of this application is to ease the user and media player interaction when the video needs to be played/paused/full-screened and several other features. We do not want the user to do the intermediate action of getting up and controlling the media player.

- Product functions: The product gives the user the ability to control a media player without going upto the computer just by doing gestures in front of a kinect.

- User characteristics: The user should be sitting a comfortable distance from the kinect we also assume the complete body of the user is visible to the kinect.

- Constraints, assumptions and dependencies: We are assuming for this project the user has a kinect with him/her and the code is being run on a windows machine. The windows does install the required drivers when it is connected to the kinect.

3.3 Specific requirements

- System requirement:
    - Windows installed machine.
    - Kinect
    - Any Media Player Installed

- Specific Sub-Requirements For Project:
    - Automatically get information about a media player by clicking it through a input window.
    - Ability to add key press shortcuts to control media player in this window.
    - Ability to save all the shortcuts to use for later purposes.
    - Show appropriate warning for No media player running or Kinect not connected.
    - Create interpreter to convert gestures into keypresses
    - Make gestures to Kinect to send key presses

Chapter 4 Architecture & Design

4.1 Microsoft Kinect Architecture

Knowledge of the Windows SDK for Kinect and the Kinect architecture is beneficial for properly understanding the Media player controller and the Gesture detection framework. More information at with a comprehensive guide can be found at:
http://research.microsoft.com/en-us/events/fs2011/jancke_kinect_programming.pdf

4.2 Kinect Media Player Controller (KMPC) Architecture

The KMPC architecture has two major components, namely the controller component, and the gesture detection component. The controller is responsible for the interaction between KMPC and the media players (e.g. VLC, Windows Media Player, etc.) that KMPC is controlling. The gesture detection component is responsible for detecting the hand gestures made by the user which are captured by the Kinect sensor.

4.2.1 Media Player Controller

Rather than working with a specific media player (like the popular VLC, or Windows Media Player, etc), KMPC is designed to work with any media player. To achieve this, the KMPC daemon does not interact directly with the media players like writing a plugin for VLC, which interacts directly and specifically with VLC. KMPC is actually not aware it is controlling a media player. Instead, KMPC interacts with Windows (the operating system) windows (the actual windows you can drag around in the GUI). This way, it will work with any media player the user desires after a simple one-time setup process. The theory is that all popular media players have assigned shortcut keys for major functions such as play/pause/next/prev/etc. KMPC will “remember” the unique window classname of the media players, and send the corresponding shortcut key combinations to the media player window (e.g. [SPACE] for play/pause in VLC). Since the window classname for media players are typically unique, this solution works decently well for our purposes. Therefore, although not design for this purpose, KMPC can functionally work with any type of window, not limited to media players.

In order to get necessary system information and interact with visible windows, the controller component uses a few Windows APIs and libraries extensively, some notable ones include the Win32 library, System.Windows namespace, System.Windows.Forms namespace. The majority of the controller functions are implemented in the `ControlFunction` class and `MainWindow` class.

The `ControlFunction` class only contains of two methods: `Execute`, and `KeyPressInterpreter`. Execute is the final method in our code that will make the media player perform an action. This is done by the Win32 API `SetForegroundWindow` and System.Windows.Forms namespace API `SendKeys`, which will first set the media player as the foreground window, and then sending the parsed shortcut key of the action to the media player in the foreground. The `KeyPressInterpreter` is called by `Execute` to convert control strings saved in the KMPC library to a string that consists the key combination in the format accepted by `Sendkey.SendWait`.

The `MainWindow` class contains two significant methods, namely `Report` and `ParseGesture`. `Report` is the callback method that is repeatedly called by the Win32 library API `EnumWindows`. For each enumerated window, `Report` checks the windows class name against a list of known media players saved in the KMPC library and tries to load the shortcut key mappings from the library XML file. `Report` will return false upon a match and successful XML read, causing `EnumWindows` to return. If multiple known media players are opened, `Report` will load the media player first returned by `EnumWindows`. Therefore, it is highly recommended that only one media player is opened at a time when using KMPC. `ParseGesture` is responsible for checking the detected gesture against a list of known gestures, and calling `ControlFunction:Execute` with the corresponding control string (e.g. control string for play) based on the list retrieved from the data XML file loaded in `Report`.

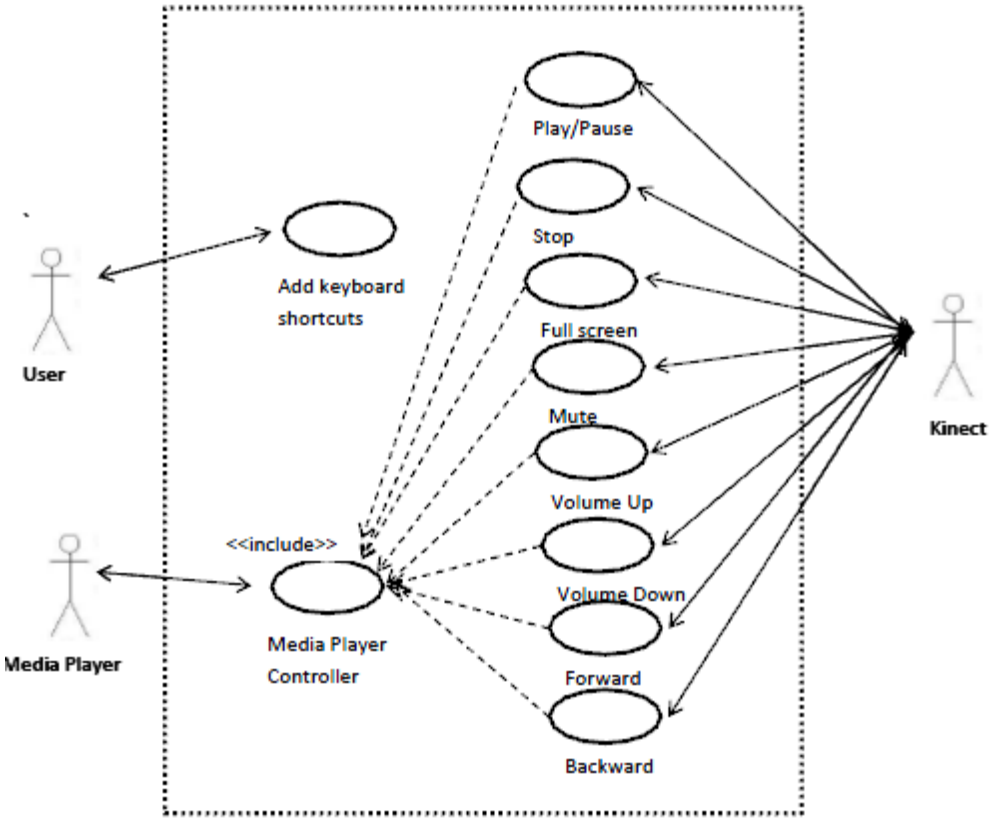


Figure 1: Media Player Controller Use Case Diagram

4.2.1.1 Read from XML file

KMPC has keeps a record of all the media players the user wishes to control in a XML file. This XML file saves two types of information: the media player window's class name, and the shortcut key combinations for each of the 8 available controls (play/pause/next/previous/ etc). The reading of the XML file is invoked `MainWindow:Report`, when `Report` tries to read the node with the class name of the current window returned by `EnumWindows` (see 4.2.1 description for details). Figure 2 and Figure 3 show how classes interact during an XML read.

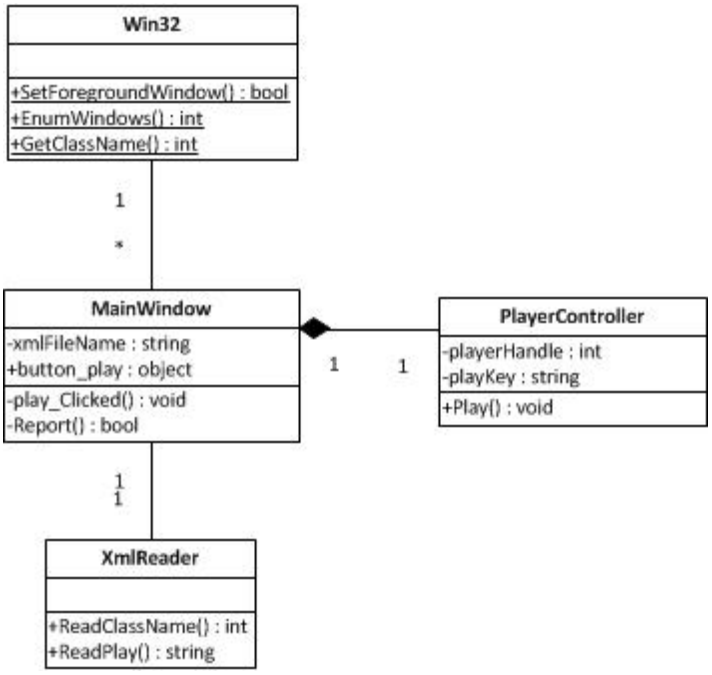


Figure 2: Controller Read Class Diagram

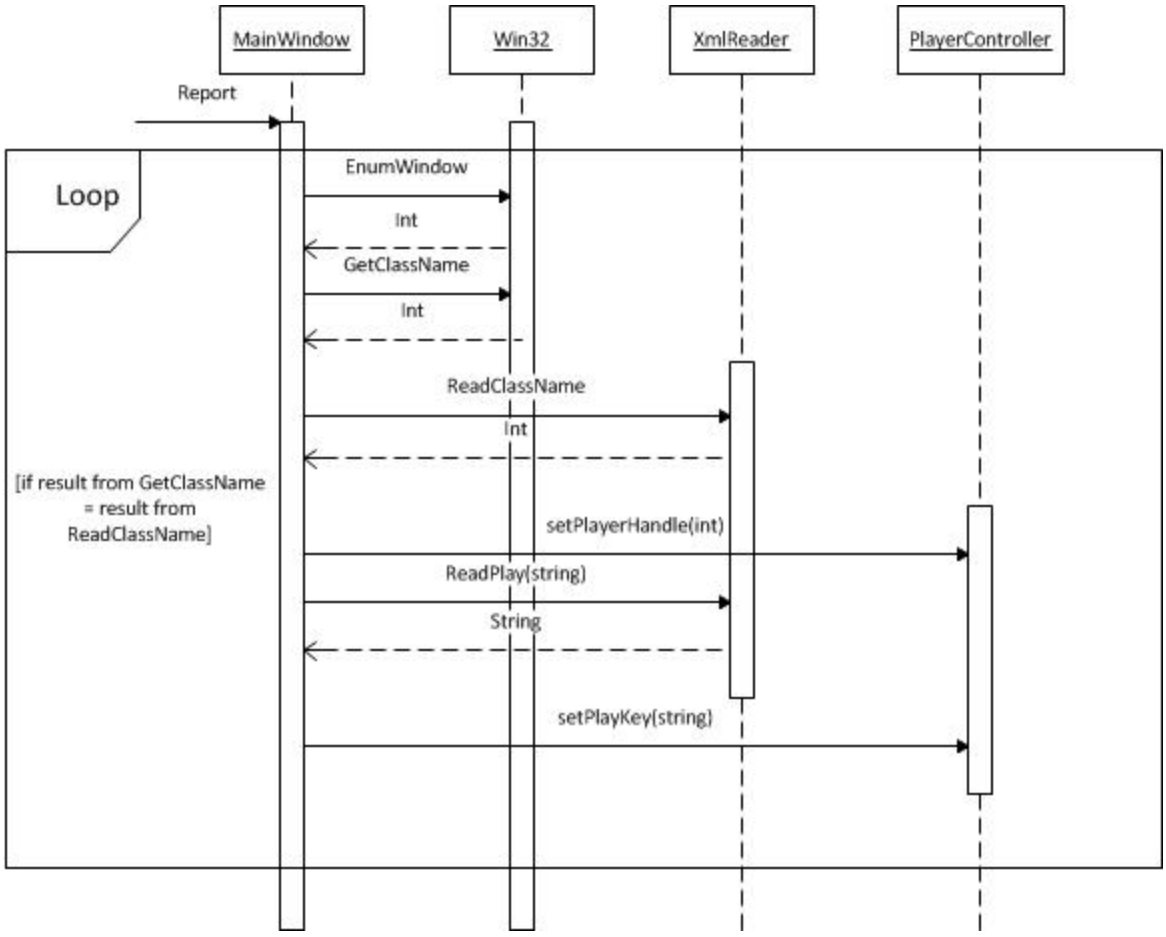


Figure 3: Controller Read Sequence Diagram

4.2.1.2 Write to XML file

One of the most crucial steps of setting up KMPC for use is the initial step of adding a new media player to the KMPC media player list/library. This involves manually entering shortcut key combinations for the media player functions (e.g. [space] for play/pause in VLC) into a Windows Form. The information captured in that form will be saved to a data XML file that saves all the key combinations and media player window class names that will be used to control the player.

To manipulate XML file types, the `XmlDocument` class in the `System.Xml` namespace is used. For this section, the key methods used are `Load`, `WriteTo`, and `Close`. When the user clicks on the "Edit Player Shortcuts" button on the home window of KMPC, a separate form is invoked. This form has a series of textboxes that, when clicked on, will capture keyboard inputs. This allows the user to input a key combination for each of the 8 available functions. For example, in the Play/Pause box, the user can press [space], and the key will be captured. When the user clicks on the "Save" button, the form will invoke a XmlDocument object, and write all the captured information on that form to a data XML file that resides in the same directory as the KMPC executable.

Figure 4 and 5 below shows the class diagram and sequence diagram for the use case of saving the 'Play/Pause' shortcut respectively. The MainWindow class instance is the KMPC home window. When "Edit Player Shortcut (button7) is clicked, a `Form` instance is created. The user can capture a window and obtain the class name by `GetClassName` of the Win32 library

API. `textBox_Play_KeyDown` captures the keyboard inputs. When “Save” is clicked, the `Form` will invoke a `XmlDocument` instance to write the data to the XML file.

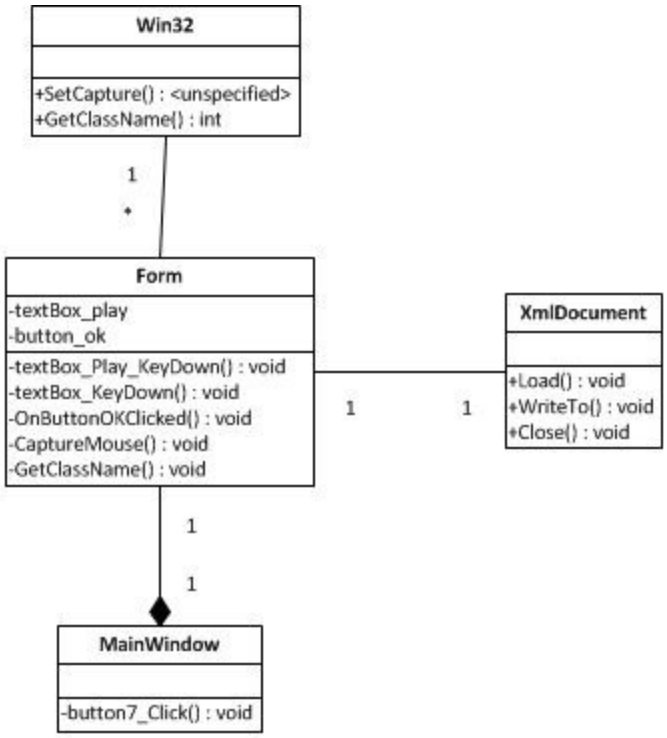
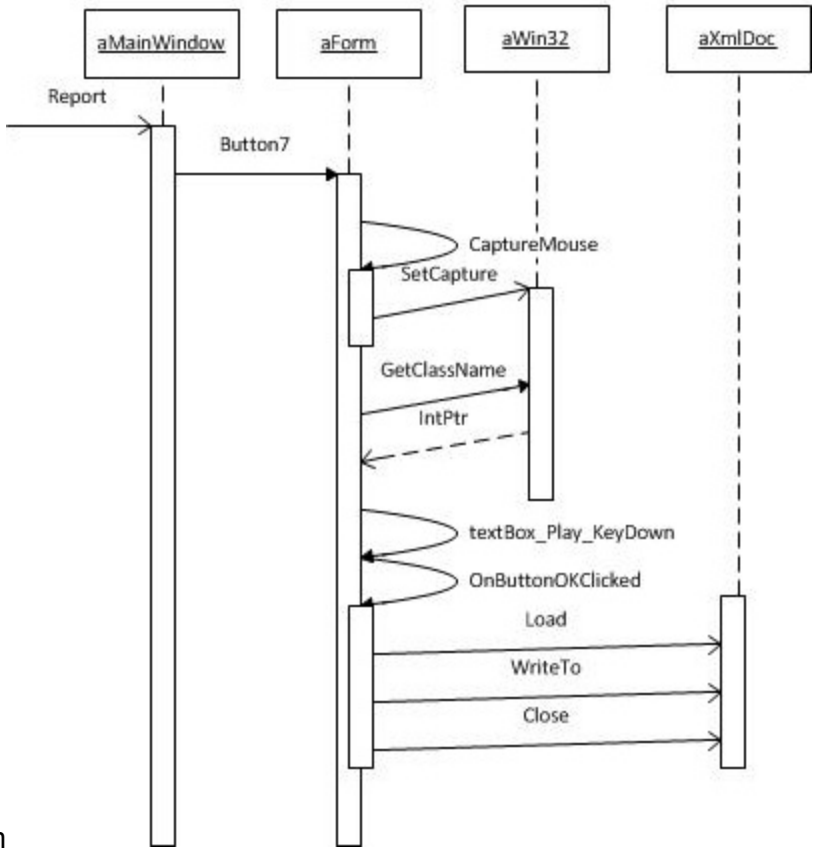


Figure 4: Controller Write Class



Diagram

Figure 5: Controller Write Sequence Diagram

4.2.2 Kinect - Gesture Detection

The gesture detection is Kinect, for this project, is done using the Kinect Toolbox. This package is available online and was really helpful while working on the gesture detection. Kinect Toolbox allows the developer to maintain templates which are basically new gestures which can be used to obtain a richer set of identifiable gestures. Kinect identifies the gestures made by the skeleton of the user, mainly functioning under the Skeleton class in the Toolbox. Once a gesture is made, we have a 1 second window in which Kinect sleeps and does not look for any more gestures. On the event of obtaining a gesture, Kinect first looks for it in the default available gestures - `SwipeRight, SwipeLeft, SwipeUp, SwipeDown`. If the gesture obtained is one of these then `RaiseGestureDetected` of the `GestureDetector` class is invoked.

`GestureDetector` has one more sub-class called `TemplateGestureDetector`. Which has the similar functionalities as the `GestureDetector` but for templated gestures. Templates are stored in individual files and once a gesture is discarded as a non-default gesture (`SwipeGestureDetector`), then a new instance of `TemplateGestureDetector` is made and the control loops through all the templates, which basically contain vectors for a specific gesture. Upon match, `RaiseGestureDetected` is raised. If the gesture is completely unrecognizable or invalid, then `invalidGesture` is raised, which basically does nothing in our application but we left it there for sake of sanity checking, if needed in the future.

Besides this, the toolbox also has a record feature which can be used to record the skeleton. This was primarily used in making the testing framework, which is fed a previous recording and a list of right gestures in that recording, in proper sequence. Kinect looks at the recording and guesses the gesture, and cross checks it with the list of right gestures. If there is a mismatch, a proper exception is passed.

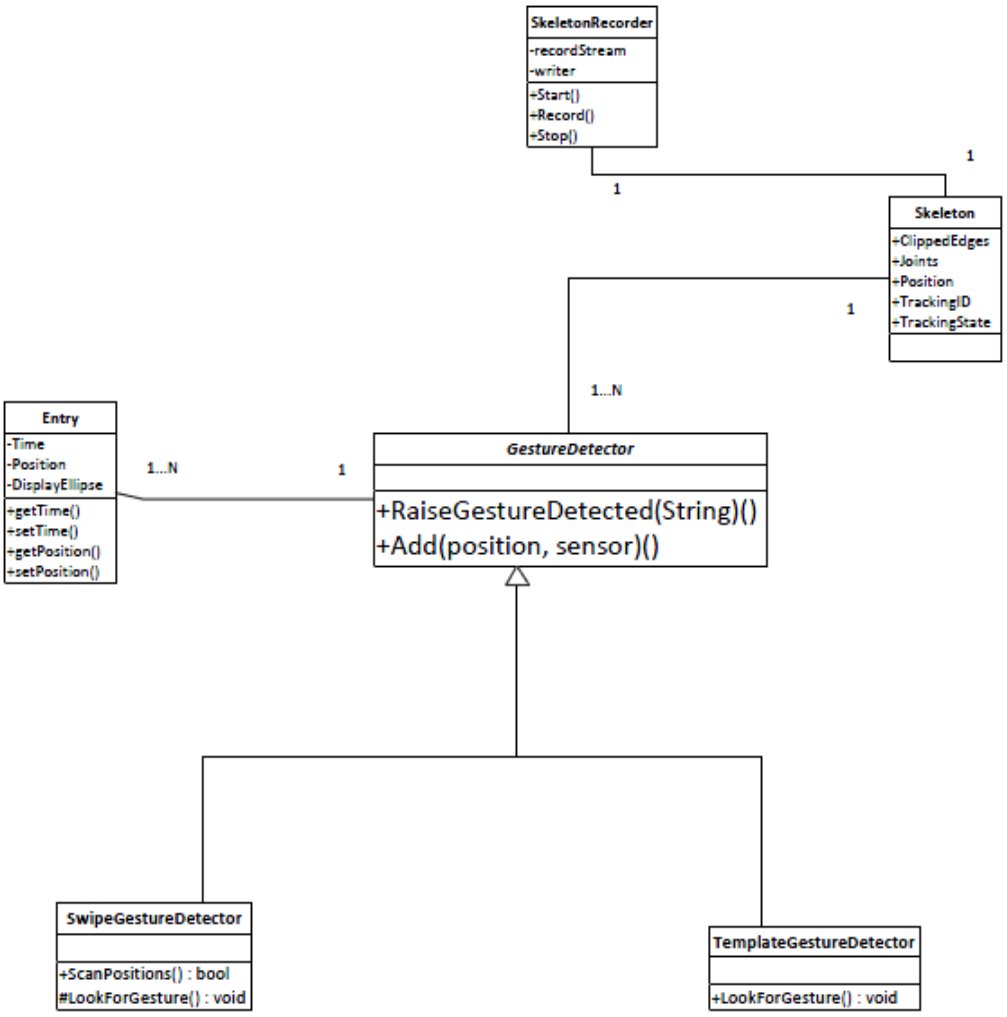


Figure 6: Gesture detector class diagram.

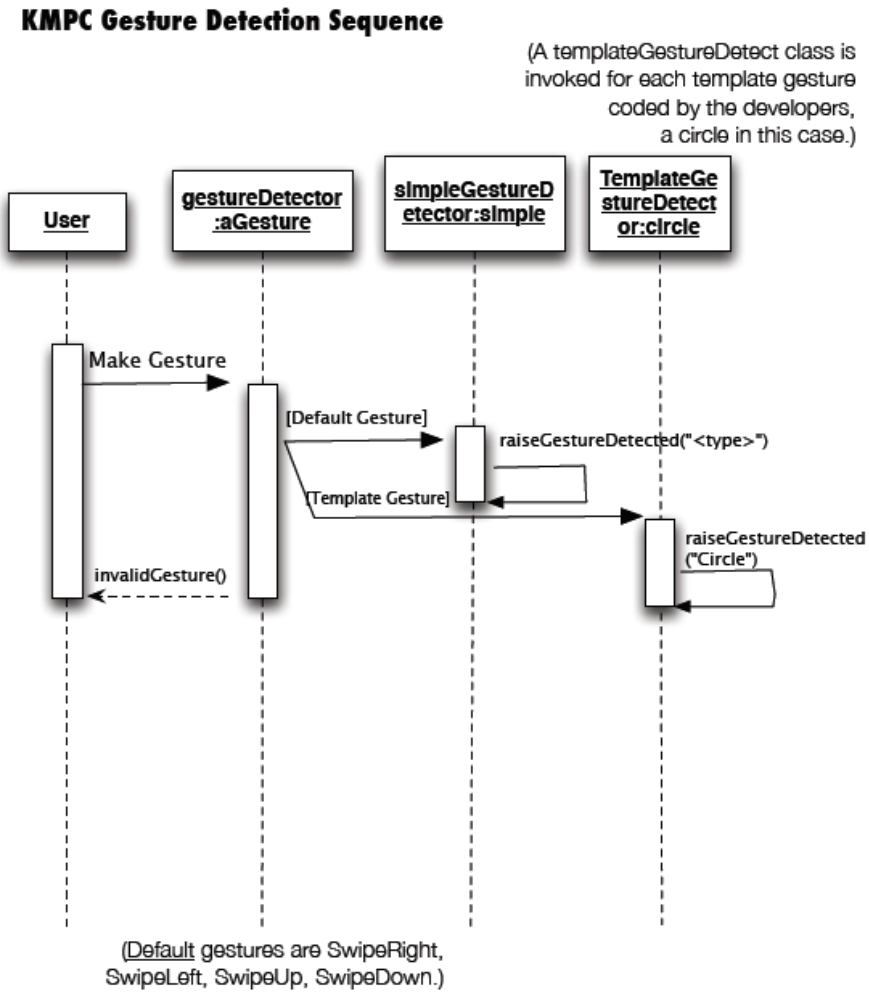


Figure 7: Gesture Detection sequence diagram.

Chapter 5 Future plans

5.1 Future Development

We will open source our project and upload it to the Github for other developers to modify and improve. If time permits, we will continue to improve the functionality and customizability of the program. For example, we will add the capability for users to record their own hand gestures, and the ability to choose which gesture maps to which functionalities.

5.2 Personal Reflection

1. Dawen Huang:
   I wasn't sure we can finish the project because we don't familiar with Kinect. We didn't know what kind of libraries we can use. As we did more research, we found lots of useful materials. My research skill enhanced through this project, and I know more about C#, Kinect and Visual Studio. I'm so glad we can finish some things that seemed "impossible" at first.

2. Wanya Huang:
   I didn't have much experience writing C# code before this project. It is a challenging project to me because I had to learn about the framework and the Kinect SDK. I researched and learned from many similar projects in order to implement some small features in our software. I also learned a lot about how to work in a big team and used what I learned from CS428.

3. Lik-heng Philip Chan:
   This was a fun project, I had the chance to learn more about the .NET framework while working on this project. It took me a while to get familiar with how C# works, and how the Kinect APIs work, so it was a slow start for me. Learnt a lot in the process though. I also spent most of my time writing unit tests for the Windows daemon code, which gave me the opportunity to learn more about writing unit tests with good code coverage as well.

4. Cyrus Chan:
   This is a challenging but fun project. Before the project, I didn't know anything about C# and .NET framework. However, after reading different documents online, the project is not that hard as I thought. Moreover, it is interesting that we are working in a big group and we always we have to use the knowledge we have learnt in class. Finally, I learn a lot on unit testing and also figure out tests that are difficult to develop in our project.

5. Sandeep Bhattaram:
   This was an interesting project with some nice challenges. I didn't really have any experience with C# before this project, so I was able to get some nice experience with the language, as well as use the Kinect, which was fun to work with, especially debugging. It was cool to work on and add gesture recognition, which helped make the provided toolbox more solid. I was also able to work on creating a unique way to test gestures.

6. Darshan Sanghani:
   This was my first time using the .NET framework. I was somewhat lost in the beginning but once I understood the control flow, it was just a matter of learning C# syntax and some concepts unique to that language. I used a lot of software development paradigms

taught in the class and realised that they actually help towards a smoother development process. I also learned how to work in a big group of developers. All in all this was a fun project to work on.

7. Yogesh Italia:
   This was a really fun and an interesting project to work on. Also it was the first time for me working on any of the technologies that are used in this project. The size of the team was also the largest, that i have ever worked with in college. All in all this project was a huge learning curve in which the concepts of software engineering were deeply applied. Also the fact that it was a semester long project gave us a chance to spend time learning and exploring new technologies which i personally think will be a really useful skill in the industry.

Chapter 6 Appendix

6.1 How to install the program

1. We have a executable file (.exe) for our project, so users don't have to install anything, simply run KMPC executable.
2. A Kinect sensor is required to access the features offered by KMPC. Under normal cicumstances, Windows should automatically install the required drivers for the Kinect sensor when it is plugged in the first time. However, if manual installation is required, the required drivers can be found at:
   http://www.microsoft.com/en-us/kinectforwindows/develop/overview.aspx

6.2 How to run the program

Just double click the executable file. (Make sure the Kinect sensor is connected to the computer)

6.3 Basic environmental requirements for using our program

1. Users must have a Kinect connected to computer to user our program.
2. Due to the library we use, our program can only run in Windows environment, so users must have a Windows operating system.
3. Because of the limitation of the Kinect, users must use it as the official document suggests. For example, users shouldn't place the Kinect in front of speakers, near things that vibrate or make noise or in a place that has direct sunlight on the sensor. They should put the Kinect on a safe and secure surface, between two to six feet off the floor and centered with the computer, higher is generally better.