Decision Recording Framework for Wiimote Interactions

Simon Zhang and Mark Overholt

Abstract- The Decision Recording Framework for Nintento Wiimote Interactions (WiiDoRF) is a software framework for recording user defined game events and easily making games for this purpose. Such events are paired with physical Wiimote interactions by a user of WiiDoRF. These events are called “Decisions” and model a user action in the game. WiiDoRF is 100% open source, highly modular, Java written, and customizable. Furthermore, WiiDoRF is designed to be scaled to for large scale statistical data mining with its back end database. Ideally WiiDoRF could be used by educators and psychologists to study and keep track of a user’s learning.

Index Terms— Wiimote, software, games

I. INTRODUCTION

The history behind the Wiimote is an interesting one. It was first introduced to the public in 2006 as the new “revolution” to the gaming industry. Instead of accepting traditional keyboard-like “button presses” as user input, the Wiimote accepted actual human motion and positioning. This is done with three built in accelerometers for motion detection and an infrared (IR) camera for 2D positions in the Wiimote. The Wiimote’s novel way of interpreting user interactions has made it hugely successful. As of 2009, over 50 million Wiimotes have been sold along with the Wii gaming console. Furthermore, many hardware extensions of the Wiimote have been introduced such as the nun chuck, Wii balance board, and Wii motion plus.

The question among many people who have observed the huge success of the Wiimote’s novel concept or have used the Wiimote themselves has been: how can we gain access to the Wiimote ourselves? This question was soon answered once people found out that the Wiimote can easily be reversed engineered by taking advantage of its Bluetooth capabilities. They discovered that the Wiimote sends out bytes of data indexed by address values via Bluetooth signal; by figuring out the correct mapping between memory addresses and raw physical Wiimote data sources, people have been able to directly map the Wiimote to the computer. A community specifically dedicated to reverse engineering the Wiimote can found on Wiili[1] and Wiibrew[2]. Countless drivers, applications, and information about the Wiimote have been compiled by this community.

What seemed lacking amongst the Wiimote community, however, was a framework in which one can extract non-physical data such as game decisions and write games for such an extraction. There are many successful drivers such as Brian Peak’s WiimoteLib[3], for example, that are great at mapping physical Wiimote data to the computer, and gesture recognizers such as wiigee[4] that interpret physical motion. Yet there exists few, if any, software for recording what cannot be physically sensed, such as programmer-defined ‘decisions’ during a game or an
environment in which to make a game for that purpose. Furthermore, traditional, non-computer science related, fields such as education and psychology are interested in studying the effects of the Wiimote and games on humans; yet there are few, if any, software frameworks in which they can work in. WiiDoRF addresses this new demand and weak supply by introducing a modular programming framework where human game decisions can be recorded and structured appropriately as computer data.

II. WiiDoRF

II.a. Wiimote Driver

The first technical challenge we encountered was in finding an existing driver to map the Wiimote’s physical data to the computer. WiiDoRF is intended to be scaled to store and process potentially millions of data; WiiDoRF is also intended to be multi-platform and accessible to well written graphics and data transferring libraries. To best meet these conditions, we decided on using Java as our programming language and to use a successfully Java written driver. Of the existing Wiimote drivers that are pure Java, all of the drivers use the JSR-82 Java specification. We found that such pure-Java drivers, unfortunately, were not compatible with the existing Bluetooth drivers in most computers. There was an existing driver that was not pure-Java, but partially Java, however, that was very convenient to use: WiiUseJ[5]. WiiUseJ is a Java driver that is actually a wrapper on top of the highly successful C written Wiimote driver: wiiuse[6]. WiiUseJ is compatible with the Linux and Windows operating systems, traditionally event driven, and single threaded. WiiUseJ was, by far, the simplest to use Java driver and would be best suited for WiiDoRF’s purposes.

II.b. WiiUseJ

Most of the WiiUseJ driver is directly used in WiiDoRF. While the driver itself works perfectly well on its own, in order to accommodate a more convenient programming environment for WiiDoRF, we made a couple of minor modifications to the driver’s listener interface. Originally, WiiUseJ first classified physical Wiimote data into three primary types: motion, IR, and button; WiiUseJ then attached conditions to each incoming physical data event such as “just pressed” or “held” for button presses or “x-axis”, “roll” for motions. These events were sent to very basic listener methods that were only classified by the incoming data’s primary type: motion, IR, or button. These general purpose methods were okay to use for simple script-like applications, however a real inconvenience for anything more complex. Our solution to this problem was to split up the WiiUseJ listening methods to listen on more specific physical user events. For example “onButtonPressed()” used to receive events for pressing any one of the eleven buttons and whether they were “just pressed” or “held”. The single “onButtonPressed()” event actually handled forty-four different actions. In order to abstract this some, we split the “onButtonPressed()” method into twelve different methods such as “onAButtonPressed().” This
meant that each event only had to worry about four actions: Pressed, Just Pressed, Held, and Just Released for each button. We also split the “onMotionSensing()” method into three different high linear-acceleration listening methods for each of the physical axes directions: X, Y and Z. The following code segment illustrates the difference in convenience between the partitioned methods and the general purpose methods:

Before:

```java
//pre: Any Button on the Wiimote was just pressed
public void onButtonPressed(WiimoteButtonsEvent arg0){
    if(arg0.isButtonAHeld()){
        //do something
    }else if(arg0.isButtonAJustPressed()){
        //do something
    }else if(arg0.isButtonBHeld()){
        //do something
    }else if(arg0.isButtonHomeJustPressed()){
        //do something
    }...
} else if(...
}
```

After:

```java
//pre: Button ‘A’ was just pressed
public void onAButtonPressed(WiimoteButtonsEvent arg0){
    if(arg0.isButtonAHeld()){
        //do something
    }else if(arg0.isButtonAJustPressed()){
        //do something
    }
}
```

We also wrote minor peripheral methods that made convenient the Wiimote’s feedback features such as its rumble and LED lighting abilities. The ultimate purpose behind our modifications is to make WiiDoRF an easy to use programming framework such that one can easily associate a user action with his game decision and write more complex yet easily manageable Java code.
II.c. SandBox UI

The central game logic and graphics environment of WiiDoRF is called the SandBox UI. The SandBox UI provides the logic that associates user physical actions with his game decisions. It also is the programming structure for which WiiDoRF games are built on top of. The SandBox UI graphics utilizes the highly successful Java Swing library. It is expected that any Java programmer will easily be able to understand the SandBox UI’s construction and quickly write games in WiiDoRF.

Some special features of the SandBox UI that will be covered are its data driven approach, its easy implementation, and its role as the logic that associates physical Wiimote data to nonphysical game decisions.

II.c.i. Input Data Driven Approach

Using the original concept introduced in *Data Driven Games* [7], games that use the SandBox UI are fed XML input data that define the game objects and their properties in the game; this kind of initialization separates game logic from game fields. Games in WiiDoRF are easily customizable under this data driven approach. If one were to write multiple versions of a puzzle game where there were different levels, say, all one would have to do in WiiDoRF is redesign the input XML files and keep the game logic and rules. Furthermore, the games themselves, under the data driven approach, can be associated with data and not hard-code. The ultimate goal of WiiDoRF is to be able to store and structure both a decision and the game it associates with as data for simplicity and reliability.

II.c.i.1. GOI

To facilitate writing the input game fields’ initialization XML files for WiiDoRF games and fulfill WiiDoRF’s role as an easy to use programming framework, we developed a What You See Is What You Get (WYSIWYG) editor called GOI. GOI eliminates the need to manually type out an application's initialization file and for the game developer to memorize WiiDoRF input XML syntax; GOI can also speed up development time and gives the game developer a convenient graphical way of designing his games. GOI is very intuitive to use; for each game object a user creates, he just needs to point and click for positions and specify its property values in GOI; his specifications will automatically be written to an input XML file. The input XML file contains game field initializations and will be parsed by the SandBox UI. The following picture illustrates GOI being used by a user.
Here is the input XML data generated from the above picture:

```xml
<?xml version="1.0" encoding="utf-8"?>
<test game>
  <draggableT>
    <text>I am draggable text!</text>
    <x>130</x>
    <y>93</y>
    <width>342</width>
    <height>33</height>
    <id>0</id>
    <fontname>Comic Sans MS</fontname>
    <fonttype>BOLD</fonttype>
    <fontsize>40</fontsize>
  </draggableT>
  <draggableO>
    <filename>images/bluesquare.jpg</filename>
    <x>131</x>
    <y>213</y>
    <width>100</width>
    <height>100</height>
    <id>0</id>
  </draggableO>
  <draggableO>
    <filename>images/orangesquare.jpg</filename>
    <x>232</x>
  </draggableO>
</test>
```
II.c.ii. Object Oriented Games: the SandBox UI game environment

II.c.ii.1. Game Objects

The data driven games written in WiiDoRF are naturally objected oriented. The input XML files, for example, are structured around a game object classification. To organize the vast amount of potential game object types, we designed an intuitive classification system. The following picture shows our object hierarchy:

![Object Hierarchy Diagram]

All Objects are classified in such a way that non physical game data recording simply becomes a matter of recording game objects and their changed properties. Game objects are designed to easily fit into simple game physics and rules.

At the very top of the hierarchy is the most generic classification of a single game object: a SandBoxObject. SandBoxObjects are required to have a 2D position and size; in other words, they represent the essentials of a physical 2D entity. Of the SandBoxObjects there are two kinds
of objects: user controllable objects and game environment objects. The game environment objects are known as SandBoxStaticObjects while the controllable objects can be Cursors or SandBoxDraggables.

SandBoxStaticObjects are, from the technical perspective, just Java Swing images treated as physical entities. A user can specify SandBoxStaticObjects’ image filenames and their initial velocities. SandBoxStaticObjects are often used as discrete game locations for easy decision state and object recording (more on this later).

SandBoxDraggables are objects that can be controlled by a user. This is done by “grabbing” onto a SandBoxDraggable with a Cursor object. Cursor objects are just in-game mouse cursors that are positioned directly by physical Wiimote data. A SandBoxDraggable can be of two different types: SandBoxDraggableText or SandBoxDraggableObject. The only difference between the two is SandBoxDraggableObjects represent images while SandBoxDraggableTexts represent text. When a decision is recorded, SandBoxDraggables will be the objects that are selected and given a changed state.

Two other object types in the SandBox UI are the DraggableBlockObjects and StaticBlockObjects. Both objects are essentially structured groups of SandBoxDraggableObjects and SandBoxStaticObjects respectively. Both kinds of objects either have all of their individual parts change simultaneously or not at all.

II.c.ii.2. Peripherial Objects

The SandBox UI also supports data defined game instructions and two in-game utility bars. The data defined instructions, like the game objects, are initialized via XML and support pictures and text. Like an in-game object, the instructions can be used as an object in a decision.

II.d.iii. Output Data

II.d.iii.1. Game Decisions

Interaction Translation
The key part of WiiDoRF is the ability to take the actions that the user makes during a game and translate that into readable data for the developer. In order to do this, we developed a data structure called a decision. Decisions are saved throughout the run time of a game, and then saved into a database and an XML file at the end. This data can then be used to replay the game in real time, or have statistical analysis done on it.

To facilitate reliable data recording for the database as well as taking into account all kinds of user interactions, WiiDoRF defines a 'Decision' as a discrete cause and effect thought process made during the game. A decision has five distinct properties that define it. First is the idle time. This is the idle time. This is the time that is spent, in millisecond, before the user began the action. The idle time is calculated as the time from when they ended their previous action to the time when they
began their next action. The next two properties are written simultaneously. The object on the screen that is selected is recorded as the selected object. Also that object’s starting state is recorded. This state can be many things, whether it be the angle it is rotated, or the X, Y, position it is at, or even if it is on top of another object. The next property that is recorded is the time that the user interacted with the object. This could be how long it took them to drag the object to its new location, or how long it took them to rotate the object. The last part of a decision is the ending state of the selected object. This will look similar to the starting state in terms of its syntax. For example: say a user decides to roll the Wiimote 90 degrees to turn a figure in a WiiDoRF-made game; WiiDoRF would recognize the user's decision as: first idling, then selecting a figure (with a starting state at its known angle), processing an action while holding the selected figure and then giving the figure an ending state (angle). The idea behind a decision is to filter through data “noise” that is unimportant to the game or to the person recording data. With decisions, one can define exactly what one wants to record and have it directly relate to the game.

II.d.iii.2. Recording a Game Decision

To actually record a decision, one first needs to define the kind of decision being made and then its three game-dependant components (select, start and end). For example, say we want to define stabbing an enemy opponent in a game (where the Wiimote is thrusted forward) as a decision. We would first give the decision a type; let us call it “stab”; then we would define the components that make it up. Let us say ‘selected’ is the opponent being stabbed, ‘start’ is the object containing all of his properties such as position, health, etc. and ‘end’ is the same property object updated after the stab.

The second thing one needs to do is define how and when a decision is recorded. In other words, one needs to define the physical Wiimote data and in-game conditions that trigger whether a kind of decision gets recorded. There are two checkpoints to a decision: when it starts and when it ends. We have developed the start and end methods in the SandBox UI. Their method signatures are:

void start__Decision(int player id, Selected Object)
void end__Decision(int playerid, Ending Object)

Time is continuously recorded in-between start and end method calls, so decisions directly correlate with real time.
II.e. The Database

The WiiDoRF database was designed to store the decisions that WiiDoRF creates. The actions that the user takes in the game are translated into a data structure called a decision. The database was designed to easily accept the decision data and store it in table form. The database was implemented in MySQL [8].

The database is relational and is structured into several tables to store the decision data. The player table consists of biographical information about the different players that have used WiiDoRF. Currently it has only the player name, but it can easily be expanded to add other biographical information. Other data that could be relevant to studies are the gender and age of a participant. Another table is the game info table. This table holds some top level information about different games that were played. Information here includes the game name and version as well as the initialization file used to start the game. It also stores the total time the game took as well as how long it took for the player to read the instructions. Lastly, it stores the randomization info used during the game, the number of players that played, whether they succeeded or failed, and the number of decisions it took them. Each game that is played has a row in this table. Another table is the decision list table. This is a table used to normalize the database. It contains a list of all the different types of decisions that have been defined in WiiDoRF. This table can be modified manually to add more decisions as more are added. It currently holds decision names for Drag and Drop, Rotation, Jerk, and Read Instructions. The main table is the Decision table. This table holds all of the actual data from a decision. There are fields for all five properties of a decision, which include the idle time, object selected, starting state, time held, and ending state. The decision table also has references to decision list in order to determine what type of decision was performed. It also has a reference to the player table for a reference of who performed the decision. Lastly, it has a reference to the game
info table so that we know what game type and number the decision belongs to. The last field is
decision order. This is used to group all decision from the same game, and specify what order
they were performed in.

II.e.2. Extracting from the Database

Many things can be pulled directly out of the database using simple SQL queries. A complete
SQL primer is out of scope for this paper, but a few examples are below.

For example, all of the decisions from a particular game could be retrieved (where # is the game
ID number):

```
“SELECT * FROM DragDropDecision d, player p, gameInfo g, decisionList l WHERE
d.playerID = p.playerID AND d.gameID = g.gameID AND d.decisionID = l.decisionID AND
g.gameID = ‘#’”;
```

All of the games that a particular person has played could also be retrieved (where # is the player
ID number):

```
“SELECT * FROM player p, gameInfo g, DragDropDecision d, decisionList l WHERE
d.gameID = g.gameID AND p.playerID = d.playerID AND l.decisionID = d.decisionID AND
p.playerID = ‘#’”;
```

These are just two examples of many different pieces of information that you can query out of
this database.

We have also provided an untested JDBC utilizing class intended for extracting data from the
database. The class has methods for querying the database and then outputting a Java-typed
vector containing statistically relevant data such as game success, the total times, etc.

II.e.3 Decision Replay

As stated above, WiiDoRF can replay any game that has been properly recorded. WiiDoRF can
read in the decisions recorded in the database or XML file and can visually display exactly what
happened in the game. This effectively gives the user or data analyzer a chance to visually
assess actions taken during the game. The game will be played back in real time, as if the player
were actually making the exact moves seen on screen. The benefit of this is that this allows one
to look for trends in game play with respect to real time, visuals, and the user's perspective,
rather than just looking through pure data. After all, human generated data is often best analyzed
from the human's perspective.

There are two ways for the replayer to get data. The first way is by parsing an XML file made
right after running an application in WiiDoRF. That XML data file is generated from the
SandBox UI; the XML file contains all the decision and game property data just recorded in a
game. The second way is by parsing an XML file made from a database query and JDBC. The process to create the XML file is essentially the same as if the data from the database came directly from a recent game.

The Replayer can also be used to verify how effective one’s decision recording implementation is in one’s WiiDoRF games.

II.f. Sample Games
Many sample games were written with WiiDoRF to prove its viability to translate human actions into readable and useable computer data. These games were also intended to be educational in nature. A description of a few of them follows:

Sorting Game:
The Sorting Game is a game designed to teach the computational task of sorting an array. The player is given 20 numbers, in a random order. You have an array of 20 spaces and a temp space. Using the temp space, the player must sort the array from lowest number to highest number in the quickest amount of time. As a number can not be out of the array (i.e. memory) the player must use the temp space to swap numbers that they wish to swap. Using the database and the replay functionality of WiiDoRF, trends in computational thinking of the users could be seen.

Packing Game:
The packing game takes the concept of graphically representing the standard packing optimization problem. The object here is to pack all of the shapes into bins using the least amount of bins possible. Points are awarded based on the density in each bin. This game is also timed. The blocks can not only be moved, but they can also be rotated to fit like in Tetris.

Story Problem Game:
The Story Problem Game combines the two concepts of unit cost and inequalities. The user is presented with a story problem. The user must then drag the correct symbols and numbers to make an inequality formula. For example: Megan buys 7 charms for $49.70. How many charms can Megan buy for $112.45? The user will then write an equation that looks like: \(7.10 \times X \leq 112.45\) where \(X\) is the number of charms. Like the previous two math games, if the first player gets the question wrong, then the next player gets a chance to answer that same question.

Micro Game:
The Micro game is essentially a multi-dimensional Game of Chicken where timing and control decide the winner. It is a competitive two player game; the users are given five bases and two base attackers/defenders. The goal of both players is destroy as many enemy bases as fast as he can. A user can attack an enemy base by entering a base attacker/defender into one of his bases and quickly ‘jerking’ the Wiimote in the direction of the enemy base across from him. More
damage is done to an empty base than one with a base attacker/defender inside of it; players are also awarded for killing faster than slower. By observing the win/lose record and decisions of users in this game, the development of the users' familiarity with the Wiimote and how they learn from each others' 'Micro' strategies can be seen.

III. Future Work
There exist many varied possibilities for future work with WiiDoRF. From the software perspective, Gesture Recognition could be implemented into WiiDoRF. Wiigee is an open source Java gesture recognizer that uses sophisticated statistics and data mining techniques to train and recognize human gestures with data acquired from the accelerometers in the Wiimote. We attempted during the course of our work to implement Wiigee into WiiDoRF; however, because of time constraints, we were not able to finish. The integration of Wiigee with WiiDoRF would further enhance WiiDoRF’s data collecting abilities and allow for a broader range of possible user actions. A sword swing or a “zorro move”, for example, could become an action associated with a game decision. Another item that would be useful for WiiDoRF is the Wiimotion Plus. The Wiimotion plus is a hardware add-on to the Wiimote that gives the Wiimote the ability to track its 3D position in space. The Wiimotion plus was released in June 2009. Unfortunately, sufficient support for the Wiimotion plus was not available during WiiDoRF’s development for the Wiimotion Plus to be used. When support and drivers for it become available, the Wiimotion plus would be an excellent addition to WiiDoRF. It would add a whole new level of physical Wiimote data for WiiDoRF to use and make physical data capturing much more accurate.

From the applications perspective, work could be done with WiiDoRF in an educational setting. Teachers, for example, can use WiiDoRF to get their students to learn interactively in an interesting and unconventional way. Educators as well as students themselves, for example, can also use WiiDoRF to track student learning progress or design their own games for learning. WiiDoRF also has applications in the psychology field. With well written games, psychologists can keep track of user game decisions as well as the physical motions the users make while thinking during a game. Another application that could be realized with WiiDoRF is stroke rehab. One can track, for example, how well a stroke victim is mentally recovering or gaining back hand eye coordination. The applications for WiiDoRF are numerous and its potential is limitless.

IV. Conclusion
The Nintendo Wiimote is a powerful tool that is a very cost effective way of providing a motion and point environment. WiiDoRF is a framework that was designed in the interest of taking that captured motion, associating it with an onscreen game environment and then storing that relevant data for playback and study. WiiDoRF was designed from the ground up in Java to be a highly modular way of creating games that use the Wiimote. It also makes it very easy to take the relevant Wiimote actions a user makes during a game and translate them into computer data. The
goal of WiiDoRF is to allow other fields such as education or psychology study people ability to think, learn, or rehabilitate. And with WiiDoRF, it should be possible to easily make applications for the study of Wiimote interactions and thinking.

V. Further Information

More information about WiiDoRF can be found on our Wiki website, located at http://netscale.cse.nd.edu/WiiDoRF. From there, all the information that is needed to get the WiiDoRF source code and get it up and running on your system. Also on the wiki, there is a small tutorial on creating a new game using WiiDoRF as well as more descriptions of the WiiDoRF sample games.

VI. References