Universität Regensburg

Philosophische Fakultät III
Sprach-, Literatur- und Kulturwissenschaften
Institut für Information und Medien, Sprache und Kultur (I:IMSK)
Lehrstuhl für Medieninformatik

Interaktionstechniken und –technologien (ITT)
Modul: MEI-M 32.1 + 2 (M.Sc.)
SS 2017
Leitung: Dr. Raphael Wimmer

IPlanPy – The New Charting Tool

Sebastian Peiser, Julia Sageder
1691711, 11694688
Medieninformatik
1. Semester M.Sc., 8. Semester B.A.

Email:
Sebastian.Peiser@stud.uni-regensburg.de
Julia.Sageder@stud.uni-regensburg.de

Abgegeben am 06.08.2017

List of figures

Contents

1 Concept and Usage

In everyday life, organizing is an important part to be prolific and efficient. Not only in private, also in professional life it's necessary to display structures. Especially at work there are many cases where building a structure is a big advantage for planning e.g. a work process/tasks etc. This is where our system steps in: IPlanPy is the perfect solution for creating diagrams. It is easy to handle and a great way of presenting, for example, abstract processes, hierarchies or complex systems. IPlanPy is designed for cooperation work in a team, especially with the Wiimote, but can also be used from a single person simply with the mouse instead of the Wii-controller. The support of collaboration is an important feature of the system. Sketching with IPlanPy will prove the capacity for teamwork in that team, because the best way of usage is in splitting the input roles at two different people. Certainly, our system can also be used as a single user, just with a mouse and a keyboard. The application areas are not limited at all. In every use case where a chart is needed, IPlanPy is your system to use.

2 System

To use our system IPlanPy you need a laptop or computer with Linux, a keyboard and a mouse (for single usage) or a Wiimote and therefore also IR-Sensors/LEDs (for team usage) and a Bluetooth connectivity. To get the best performance your system may have the following requirements: Intel i5, 4GB RAM

With IPlanPy it is possible to build diagrams. The system supports the following features:

- Create a new card
- Switch the card type between:
    - Standard (Title and Field)
    - Header (Title)
- Delete a card
- Change the card color between prescribed colors
- Build a connection between two cards presented as a line from card middle to card middle
- Delete a connection between two cards (Undo)
- Delete all connections from one card

- Save your chart
- Load your saved charts
- Connect your Wiimote (Stores connected Wiimotes automatically)
- New chart (Rejects all unsaved data and clears the screen)
- Feedback to the user through message boxes in sensitive situations (e.g. Over-write existing files, Chart saved, Warnings with error informations)

3 Implementation

3.1 iplanpy.py

This is the main python script, where all the corresponding threads run together. The script imports the custom scripts with their classes, like **wiimote**, **vectortransform, gestureclassifier**, **connectionsmanager** and **card**. It also contains the class *IPlanPy*, which includes all relevant handling processes. The user interface of IPlanPy is loaded from the **iplanpy.ui** script, which contains all start-widgets (see **3.2 iplanpy.ui**). The buttons are connected in the *init_ui* to their specific definitions:

The "New Chart" button deletes all cards and connections in the *on_btn_new_chart*. The *toggle_* definitions handle the visibility of the two menus ("Connection Settings", "Save/Load"). Charts can be loaded trough the *load_chart* definition, which opens and loads the selected **.chart** file from a list (*load_available_charts*). The loading is managed from the *create_* definitions, which interpret the data of the **.chart** file and display the saved *Card*s (with *card_info*) and *connections*. Charts can be saved through the *save_chart* definition, which encodes the current chart to a **.chart** file (see **3.10 demo.chart**). Is the "Save" button clicked, there is also checked if an existing file may be overwritten. This procedure ensures good usability (see **2 System**). The user is able to scan for new Wiimotes (*scan_for_wiimotes*), if he wants to register a new one (not to forget: sync button at the Wiimote has to be pressed). The known Wiimotes are persistently saved in the **wii.motes** file, which are shown in the available wiimotes list. Through the selection of the right address of the Wiimote which should be connected, the user is able to connect the Wiimote with a following click on the "Connect" Button (*connect_wiimote*) and the sync button at the Wiimote. The vibration of the Wiimote signals that it is correctly connected.

After this he is able to call the *disconnect_wiimote* by clicking the "Disconnect" Button (Text-changed "Connect" Button) to disconnect the Wiimote.

The *on_wiimote_button*, *on_wiimote_ir* and *on_wiimote_accelerometer* detect events of the Wiimote. To circumvent the problems of the different thread of the Wiimote events, the "Button B" events of the Wiimote are interpreted as mouse-click and mouse-release events. Key-press events are also registered and handled to ensure a single user usage. The different (simulated) mouse events manage the interaction of the user (see **5 Cheat Sheet** for detailed information). The *mouseReleaseEvent* is handling the checking if a card should be deleted (over "Delete" button) or connected (over another card) or just moved to this point.

If a new *Card* should be created, the *Card* class is invoked and a default *Card* is created, only the *id* is set individually from the beginning (see **3.4 card.py**).

The *paintEvent* is called by the *update()* method manually. It draws the *connections* between the *Card*s by iterating over the *connections.connections* list.

3.2 iplanpy.ui

The user interface was built in the *Qt Creator (Community)* and contains all start-widgets. This are the following buttons and labels: "New Card", "Delete Card", "Connection Settings", "New Chart" and "Save/Load". It is loaded in the *iplanpy.py* script.

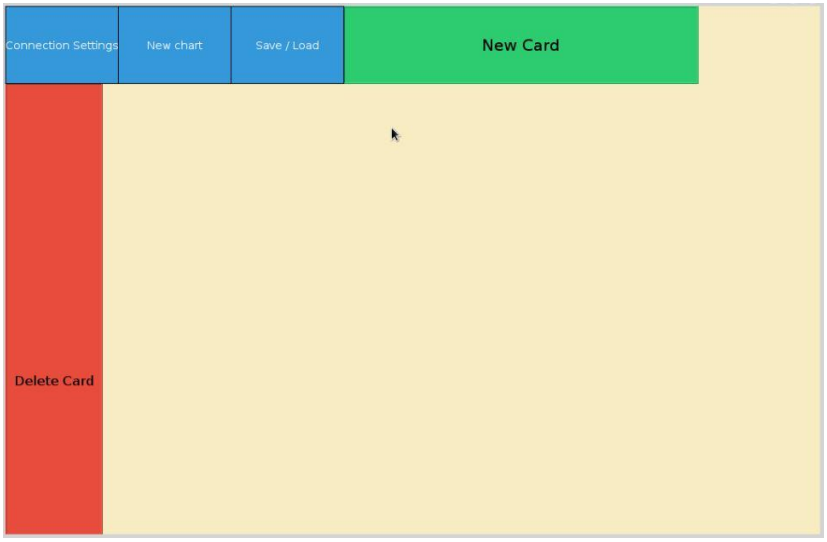

Figure 1 – User Interface

The “Connection Settings” and the “Save/Load” contain in addition menus.

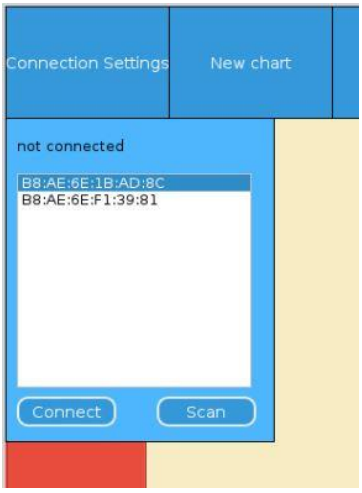

Figure 2 - Connection Menu

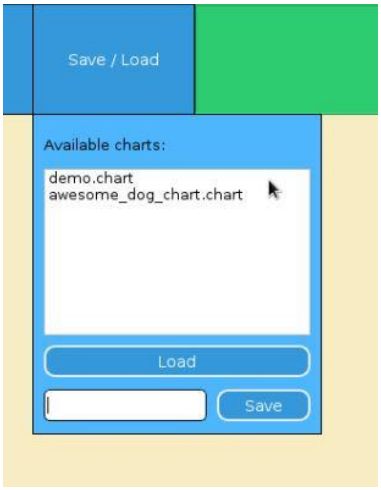

Figure 3 - Save/Load Menu

3.3 connectionmanager.py

The **connectionmanager.py** script contains the *ConnectionManager* class, which handles the connections between the cards. The definitions contained are:

connect: This one saves a new connection and proves first if it isn’t already existing.

delete_all_card_connections: deletes all connections where the handed over card is involved. In addition the deleted connections are saved in the restorable_connections list to make them flexible for undo and redo (except the card is being deleted!).

remove_last_connection: deletes the last connection of the *connections* list and restores it in the *restorable_connections* list to make it flexible for undo and redo.

restore_connection: restores the last deleted connection (if one exists) and adds it to the connections list for redo.

get_centers: determines the centers of the two cards where a connection should be build.

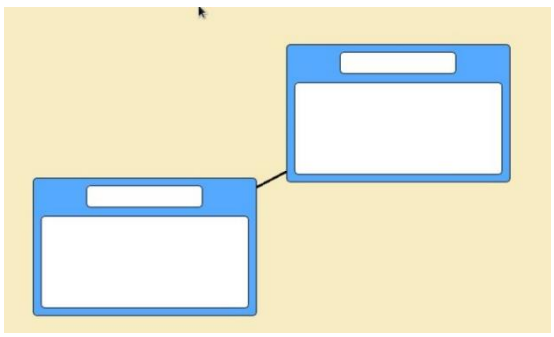

Figure 4 – Card Connection

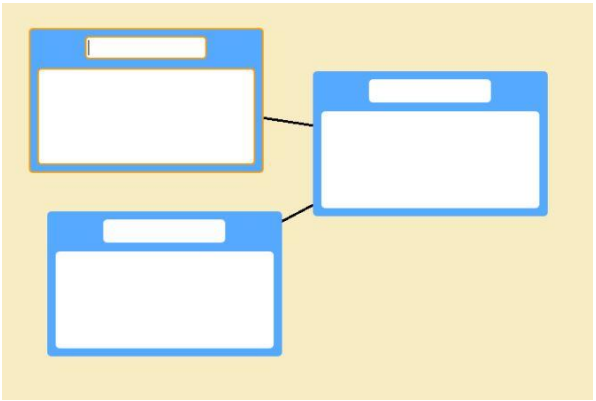

Figure 5 – Card Connections

3.4 card.py

The **card.py** script contains the card class, which inherits from the *QFrame class*, representing the cards of the system. The *__init__* function establishes the main properties of the *Card*s, such as *DEFAULT_COLOR* or *available_colors* list. The *setup_ui function* creates the user interface of the created *Card*, which contains a *QLineEdit title_field* and a *QTextEdit content_field*. There is also a type of *Card* where the *has_text_field* is *False*, and so there is only a *title_field*. The card type is switched in the *toggle_type* and the particular *setup_* definitions.

Figure 6 - Standard Card Type

Figure 7 – Header Card Type

The colors of the *Card*s can be switched by the *next_color* and *previous_color* definitions, which handle this by iterating over the *available_colors* list. Different looks of the *Card*s are handled over the *stylesheet*, manipulating *color*, *border* and *border-radius*.

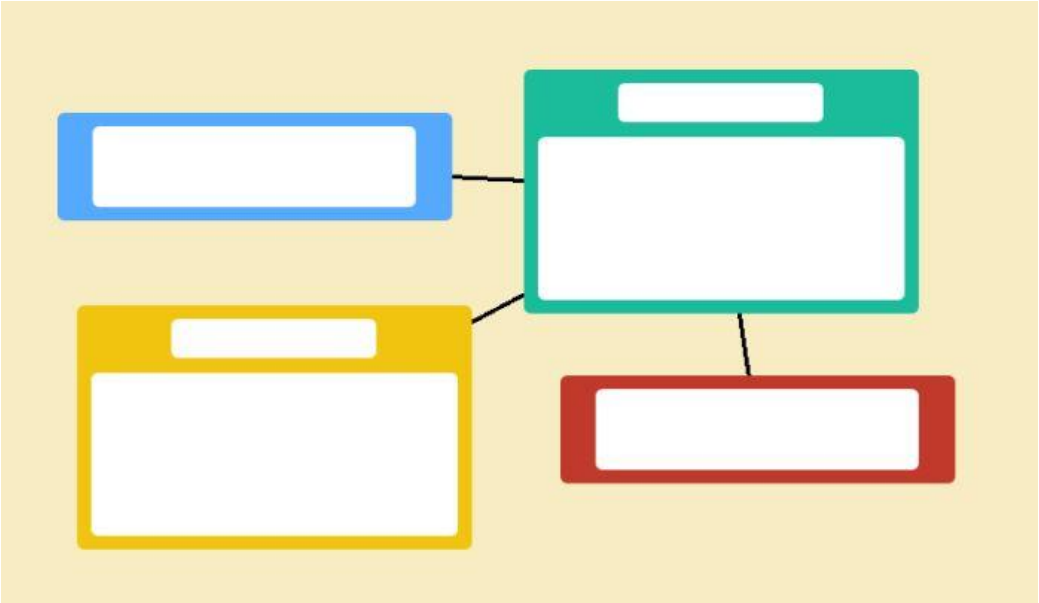

Figure 8 – Card Colors

Important actions of the *Card*s like *delete*, *focus*, *unfocus*, *move_to*, *collides_with*, *collides* and *hits_window_frame* are defined here. The *(un)focus* definitions are important for the interaction of the user (see **5 Cheat Sheet**). The *collision*, *hit* and *move* definitions are defining the movement and behavior of the *Card*s.

Figure 9 – Unfocused Card

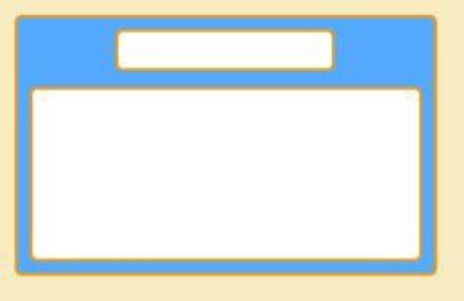

Figure 10 - Focused Card

3.5 gestureclassifier.py

The code of the *GestureClassifier* is adapted and adjusted from results achieved in Assignment 8 and from the provided files in the course. The result is from the Team F2 – Ariane Demleitner and Sebastian Peiser, who is also a Team Member of the IPlanPy project. The *GestureClassifier* contains the *CATEGORIES steady* and *shake*, which train the support vector machine with the two defined .csv files. The accelerometer data from the Wiimote is compared to the trained data and detects a possible *shake gesture*.
This shake.csv file contains the values for classifying the *shake gesture* with the **gestureclassifier.py** script. The **steady.csv** file contains data of expected standard movement.

3.6 vectortransform.py

Contains the class *VectorTransform* which is used in the **iplanpy.py** file. The calculations in this file are adapted and adjusted from the provided *Jupyter Notebook* files from the course.

The *VectorTransform* class handles the interpretation of the IR-sensor data of the Wiimote. The incoming data is proofed (there must be at least four signals), filtered and sorted, ascending by their size (strongest signal first). This should proof that the system uses the LEDs from all incoming signals, not e.g. the sun (which has infrared radiation, too). Next, the signals are sorted by their y value, so the first and the second values are from the both bottom LEDs, the third and the fourth from the both upper.

3.7 wiimote.py

This python script was provided of the university lecturer of the course, Dr. Raphael Wimmer. It is a Wiimote wrapper in in Python 3.[1] Based on this script it is able to handle the Wiimote callbacks in the **iplanpy.py** main script.

3.8 demo.chart

The **.chart** files represent the saved charts of the user. They encode the card and connection data of the saved chart in a csv-like format and can be loaded to continue the work on previous (saved) charts.

4 IPlanPy-Team Collaboration

The IPlanPy-Team contains two People: Sebastian and Julia. I (Julia) did the implementation of the features and logic, which Sebastian revised. Sebastian also was responsible for implementing the *VectorTransform* class, the gesture recognition and the interaction with the user interface which he built. He made the video of the system, while I wrote the documentation. In general, it was a very good cooperation with big discussions, which helped on the whole project. We got a result we are proud of, because we think IPlanPy is an easy-to-use but useful product for everyone.

Sebastian	**Julia**
Concept	Concept
Video	Documentation
Presentation Concept and Demo Charts	Presentation Concept
UI (Setup and Modifications)	UI (Modifications)
Code Cleanup and Comments	Comments
Bug fixes	Bug fixes
Card Class, New Card, Card Drag and Drop (with Collisions and Focusing)	Drag and Drop (modified)
Github Setup	Delete
Wiimote Connection (with User Interface Interaction)	New Card (Basics)
Vector Transform	Color (Basics)
Gesture Recognition	Stylesheet (Basics)
Wiimote Saving	Connections Drawing
Charts Saving and Loading	Connections Movement
Connection Manager Class (with Undo and Redo)	Paint Event
Wiimote Button Events (with Mouse Click Simulation)	Button Highlighting (Basics)

[1] Copyright © 2014 Raphael Wimmer <Raphael.wimmer@ur.de>

Application Control via Keyboard	Wiimote Button Events (Basics)
Button Highlighting	
Wiimote IR Tracking	

5 Cheat Sheet

The list of all possible interactions with IPlanPy:

Create a new card	Mouse: Click Button "New Card" Wii: Cursor over "New Card" + Button B
Focus card	Mouse: Click card Wii: Cursor over card + Button B
Switch the card type	Mouse: Cursor over card + Alt + Left/Right Wii: Cursor over card + Left/Right
Delete a card	Mouse: Drag and Drop card to "Delete" Wii: Drag and Drop with Button B to "Delete"
Change the card color	Mouse: Cursor over card + Alt + Up/Down Wii: Cursor over card + Up/Down
Build a connection between two cards	Mouse: Drag and Drop card to card Wii: Drag and Drop with Button B to card OR (Redo case) Mouse: Alt + Plus Wii: Plus
Delete a connection between two cards (Undo)	Mouse: Strg + Minus Wii: Minus
Delete all connections from one card	Mouse: Not possible! Wii exclusive! Wii: Focus card + shake gesture
Save your chart	Mouse: Click "Save/Load" + new chart name + Click "Save" Wii: Cursor over "Save/Load" + Button B + new chart name + Cursor over "Save" + Button B
Load a saved chart	Mouse: Click "Save/Load" + select chart + click "Load" Wii: Cursor over "Save/Load" + Button B + select chart with Button B + Cursor over "Save" + Button B
New chart	Mouse: Click "New Chart" Wii: Cursor over "New Chart" + Button B
Connect your Wiimote	Mouse: (Click Scan +) select your Wiimote + Click "Connect" Wii: Not possible! Just for changing performing Wiimote!