Universität Zürich[UZH]

Master Project at the Department of Informatics

Master Project: Post-Fossil Cities

Mapping sensor input data to physical gameboard instances to create an expandable gameboard

Professor: Prof. Dr. Lorenz M. Hilty
Supervisor: João Gonçalves
Authors: David Wyss (14-734-453) & Vincent Rüegge (15-700-966)

Prototype v2 Project Documentation. This report serves as documentation and description of our second up and running prototype. It includes detailed descriptions and pictures of all the prototype's components and their respective subcomponents. Additionally, the report presents the prototype's limitations and potential future improvements.

Table of Contents

List of Figures

1 Introduction

This report serves as documentation and description of our second up and running prototype. It includes detailed descriptions and pictures of all the prototype's components and their respective subcomponents. This includes (a) a Main Controller module to which the sensors can be connected, (b) Sensor modules on which the cards can be played, (c) a separate module for the Oracle that accepts combinations of up to four cards, (d) a Python-based UI for training, and (e) a Python-based UI to capture incoming game actions and to specify and send the corresponding HTTP API calls.
Compared to the first prototype, the second prototype is built modularly. Instead of having one gameboard with a fixed size, the second prototype allows for a variety of gameboard designs. Figure 1 represents the modular design. The Main Controller (MC), the Sensor modules (S), and the Oracle (Oracle) can all be placed individually on any gameboard. The size and shape of the gameboard are irrelevant and thus allow for more flexibility.

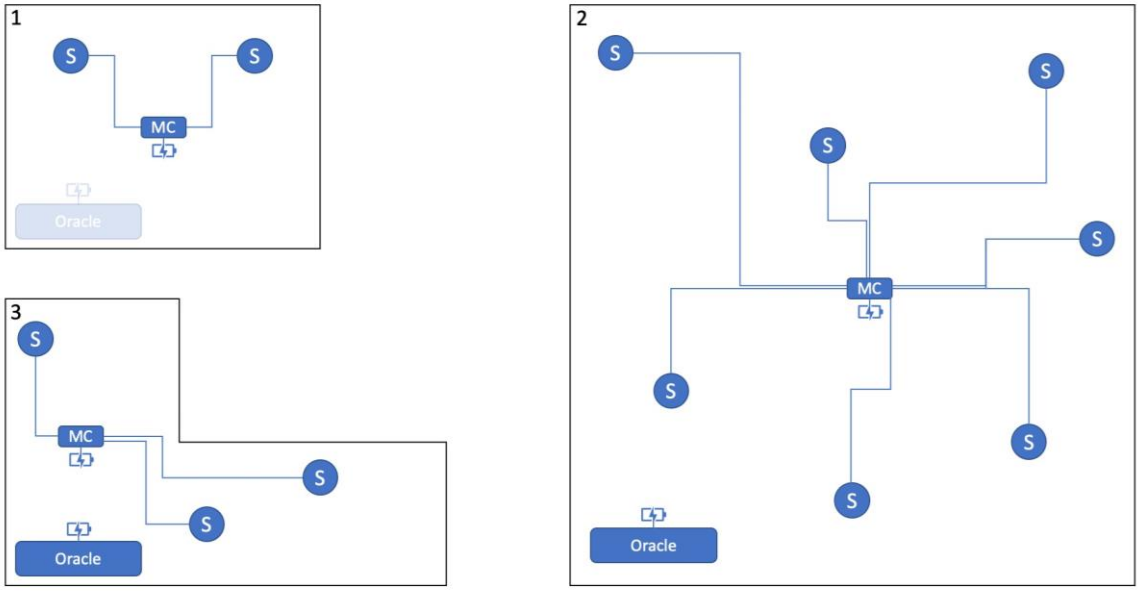


Figure 1: Modular design patterns

2 Hardware

2.1 Main Controller

The Main Controller is the centrepiece of our second prototype. It maintains and controls the data flow between the sensors and the connected device (usually a computer running e.g. the training dashboard). Additionally, it saves the mapped data generated after training. A serial connection is established via either Bluetooth (57600 baud, 8-N-1) or USB (57600 baud, 8-N-1).
It also includes three LEDs. A red LED (power) indicating whether the gameboard is connected to a power source, a green LED (status) indicating whether the Main Controller is ready to train or play, and a blue LED (communication) indicating whether a command was sent, or a valid command was received.
The current version allows for up to five Sensor modules to be connected to the Main Controller module via USB-C.

2.2 Sensor Modules

Each sensor (PN532 NFC/RFID Sensor) is placed on a PCB in its own dedicated Sensor module. Each module is then connected to the Main Controller module via USB-C. The sensors capture training and playing actions and forward the information to the Main Controller to be processed.
Each Sensor module has two LEDs. A red LED (power) indicating whether the module is connected to a power source, and a blue LED (communication) indicating whether a command was sent, or a valid command was received.

2.3 Oracle

The Oracle is a separate module with its own ESP-32 . The rationale behind this design is the way the Oracle handles sensor input. The Oracle can combine up to four sensor inputs and can forward the combination whenever a change happens.
The Oracle module also includes seven LEDs. A red LED (power) indicating whether the Oracle module is connected to a power source, a green LED (status) indicating whether the Oracle is ready to play, and a blue LED (communication) indicating whether a command was sent, or a valid command was received. Additionally, a blue LED is present for each sensor (communication) indicating whether a card is currently placed on and recognized by the sensor.

2.4 NFC Tags

Each playing card is equipped with an NFC Tag (Ntag213 13,56 MHz). The NFC Tag holds the card's ID which is captured by the sensor and forwarded to the Main Controller. The card ID is formatted in the following fashion: *CardID=xyz*, wherein three digits are required (e.g. card ID 1 would be stored as *CardID=001* in plain text (english) format).

2.5 Printed Circuit Boards (PCBs)

2.5.1 Description

A Printed Circuit Board (PCB) electrically and physically connects various components without the need for additional wiring. In our case, each module has its own custom PCB that allows for easy assembly as

well as reduced setup time and complexity. Additionally, having a digitalized PCB design simplifies replication by enabling easy ordering of additional modules.
Our PCBs were designed using EasyEDA and ordered fully assembled from PCBWay.com and Quick-Pick.ch.

2.5.2 Main Controller

The Main Controller PCB houses and provides mounting for the following components:

- NodeMCU ESP-32S
- Three LEDs (power, status, communication)
- Two MCP23016 I/O expanders (one for LEDs, one for sensors)
- Ten USB-C ports, to which Sensor modules can be connected

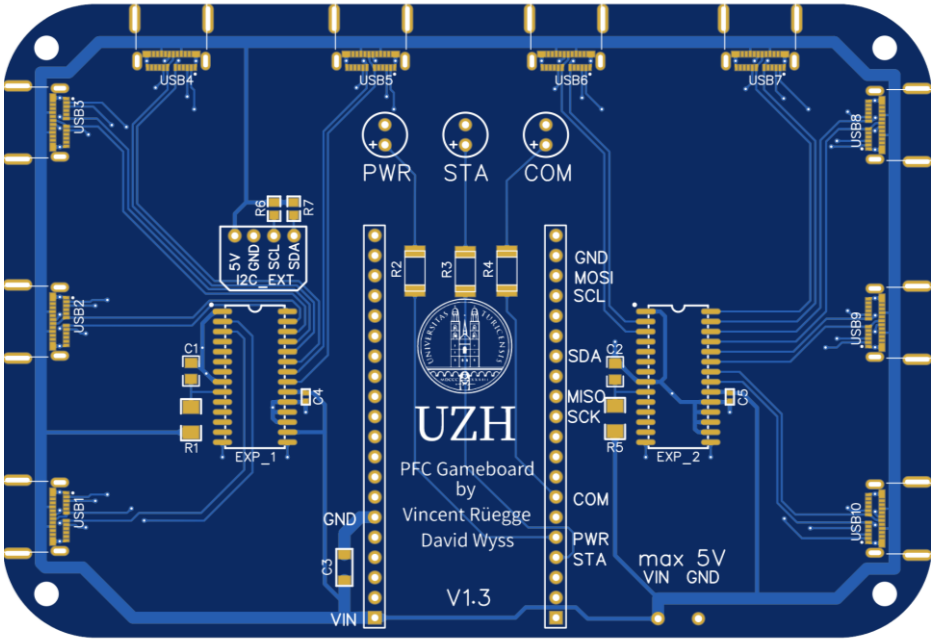


Figure 2: Main Controller PCB

2.5.3 Sensor Modules

The Sensor module PCB houses and provides mounting for the following components:

- PN532 NFC sensor
- Two LEDs (power, communication)
- One USB-C port to connect to the Main Controller

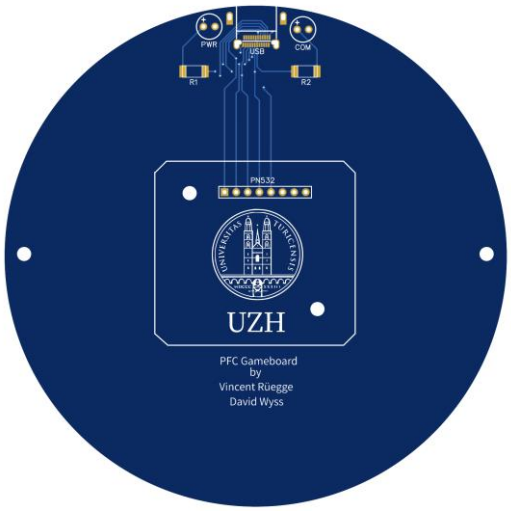


Figure 3: Sensor PCB

2.5.4 Oracle

The Oracle PCB houses and provides mounting for the following components:

- NodeMCU ESP-32S
- Four PN532 NFC sensors
- Three LEDs (power, status, communication)
- Four Sensor LEDs (communication)
- One MCP23016 I/O expander (sensors, LEDs)

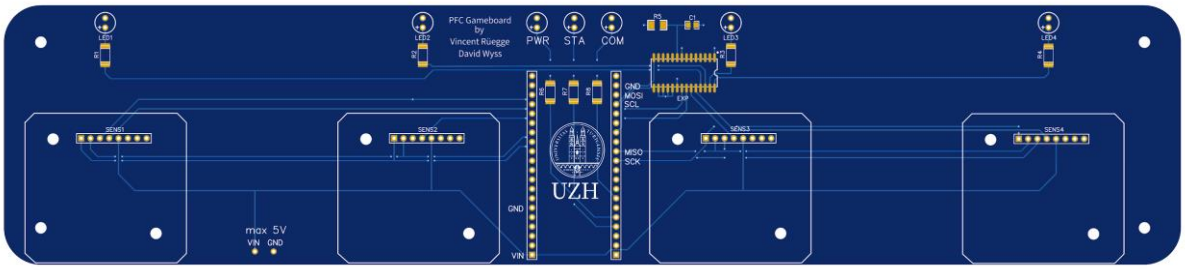


Figure 4: Oracle PCB

3 Software

3.1 Main Controller

3.1.1 Modes

We designed the behaviour of our software in a mode-dependent way. In other words, depending on the Main Controller's current mode, there exists a predefined set of commands the user has access to. Thus, the possibilities within a given mode are always clear, and no unexpected behaviour appears. Figure 5 describes each mode in more detail, as well as its available commands. Figure 6 represents the interdependence of the modes present in our main software component as well as its general flow for each step. Figure 7 and Figure 8 represent the flow that is repeated for each step of the training and playing mode respectively.

ID	Mode	Description	Commands	Follow-up Modes
0	READY	The READY mode is a passive mode with no behaviour on its own. It indicates that the game board is in a state where the user can chose whether to play the game or train the board. In this mode, power is supplied to the game board and it is ready for connection to either the training or playing dashboard	CHANGE_MODE=, REBOOT	TRAINING, PLAYING
1	TRAINING	In this mode, the gameboard is connected to the training dashboard and forwards raw sensor input data to it (via USB or Bluetooth). Each packet of training data needs to be acknowledged by the training dashboard with a TRAIN_OK message.	TRAIN_OK, CHANGE_MODE=, REBOOT, RESTART_TRAINING, TRAIN_UNDO=	UPLOAD
2	UPLOAD	In this mode, any data passed to the gameboard will be saved into its flash memory. This data consists of the mapped dataset sent by the computer running the training dashboard. Once the UPLOAD_END command is received, the upload ends and the gameboard automatically switches to READY mode.	UPLOAD_END	READY
3	PLAYING	This mode is entered after the game has started. The gameboard is connected to the playing dashboard and waits for sensor input (game action such as a card being played). From this, it then generates the corresponding API call for the game action, which is sent to the playing dashboard. The playing dashboard acknowledges the API call by sending back a PLAY_OK message.	PLAY_OK, CHANGE_MODE=	READY

Figure 5: Mode Description

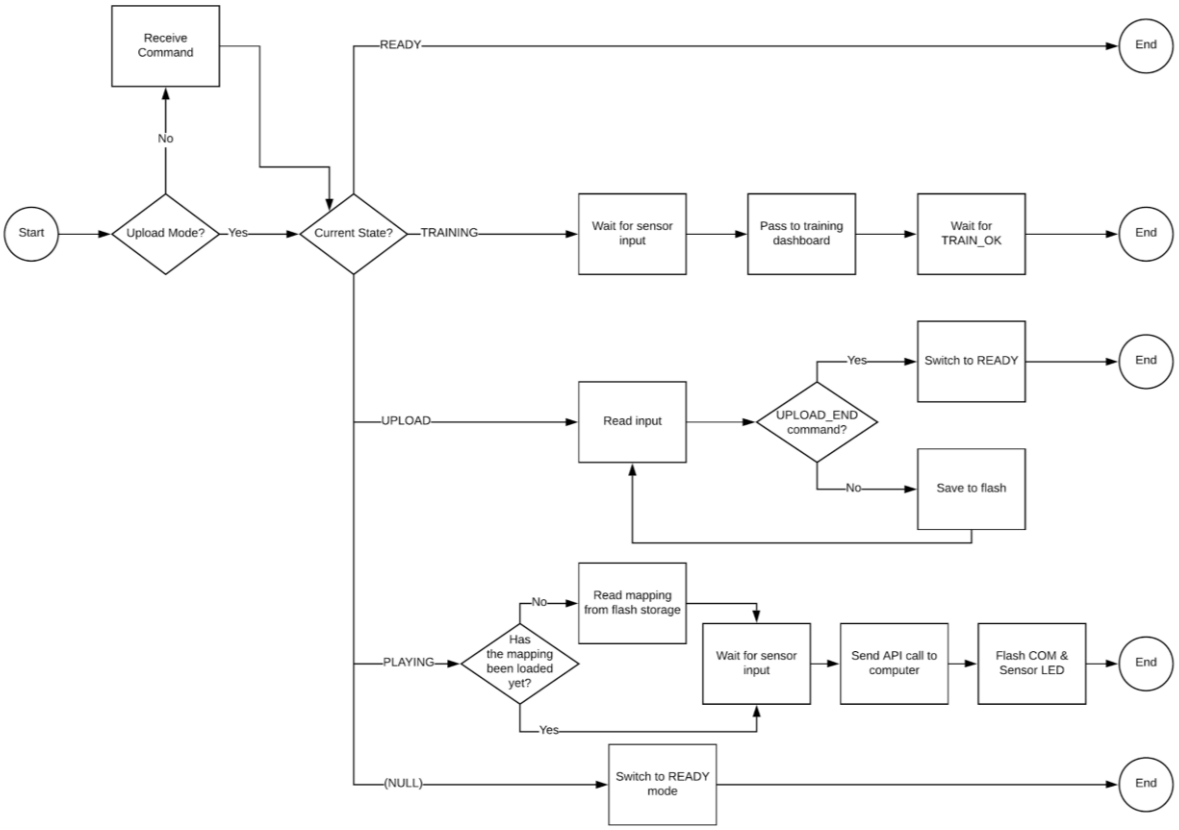


Figure 6: Flow Chart: Main

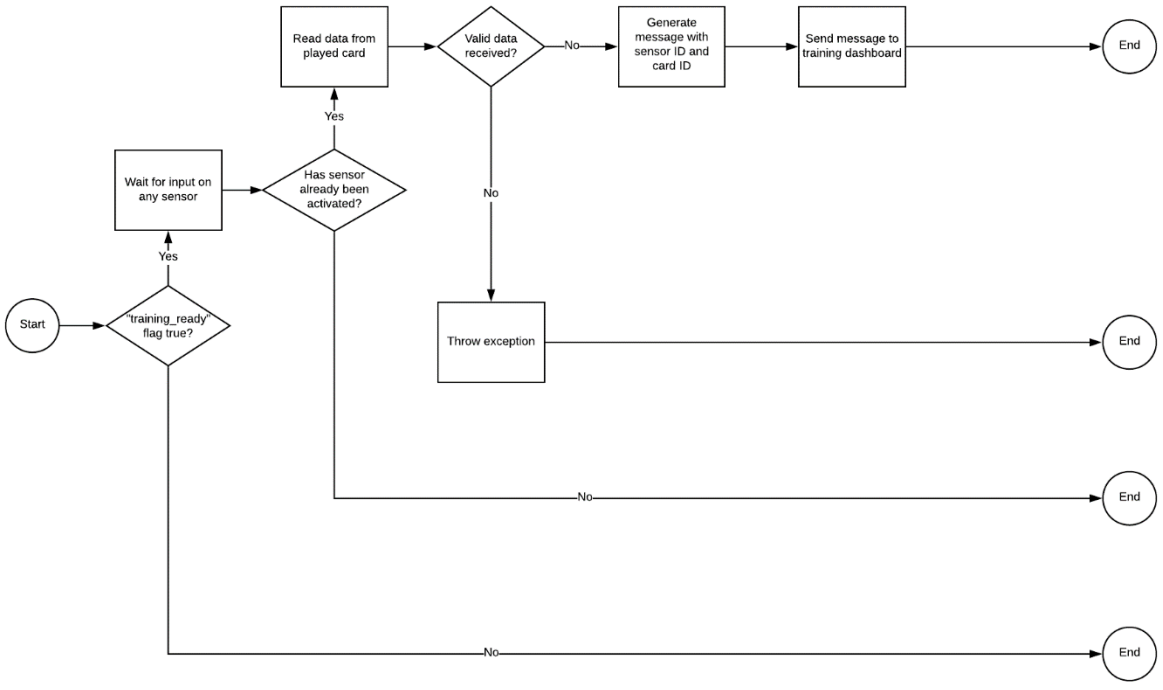


Figure 7: Flow Chart: Training

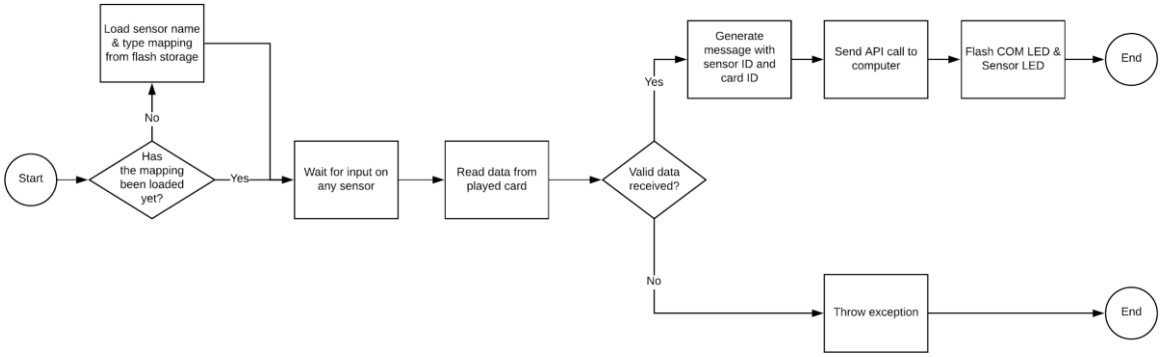


Figure 8: Flow Chart: Playing

3.1.2 Sensor Types

The training dashboard allows the user to choose between two behaviour types for each Sensor module. Firstly, there is a *regular* type, on which a user can place a card and remove it shortly afterwards; only one API call will be sent out. Secondly, a *combinatorial* type exists; for this type, cards have to remain on the Sensor module after playing them for as long as they shall be considered by the backend. A first API call is sent when the card is placed. Once the card is removed, a second API call is sent letting the backend know that the card was removed.
In the call sent by the Main Controller, the acronym *RP* stands for "regular play" and is one of three possible message types; this simply occurs when a card is played on a *regular* Sensor module. The other two message types are *CP* for "combinatorial play", which occurs when a card is placed on a *combinatorial* Sensor module. Lastly, there exists a message type *CR* for "combinatorial remove", which occurs when a card is removed from a *combinatorial* Sensor module.

3.1.3 Commands

Depending on the mode the Main Controller is currently in, the user has a predefined set of commands he has access to. These commands are used to control the software. Figure 9 describes each command. Figure 10 represents the 'Receive Command' process from Figure 6 in more detail. This entails the process that occurs upon receiving any input to the Main Controller.

Command Name	Description	Parameters	Parameters Description	Sender	Recipient
CHANGE_MODE=	Changes the Main Controller's operation mode	int mode	integer representing the operation mode	Training / Playing Dashboard	Main Controller
TRAIN=	Sends raw sensor values back to the training dashboard, which are then logged	int sensorID, int cardID	raw values from Main Controller and NFC tags	Main Controller	Training Dashboard
TRAIN_OK	Confirms reception of valid training data packet	None	-	Training Dashboard	Main Controller
TRAIN_UNDO=	Discards training data for a specific (usually the latest) sensor	int sensorID	sensor number from USB port	Training Dashboard	Main Controller
RESTART_TRAINING	Restarts the training process	None	-	Training Dashboard	Main Controller
UPLOAD_END	End flag for upload of mapped data	None	-	Training Dashboard	Main Controller
UPLOAD_OK	Confirms reception of valid mapped data	None	-	Main Controller	Training Dashboard
PLAY=	Sends information about played action (area, card ID, type) to playing dashboard	string area, int cardID, string type	mapped values from Main Controller and NFC tags, specified area type	Main Controller	Playing Dashboard
PLAY_OK	Confirms reception of valid played action	None	-	Playing Dashboard	Main Controller
REBOOT	Reboots/resets the Main Controller (putting it back into "Ready" state)	None	-	Training / Playing Dashboard	Main Controller

Figure 9: Command Description

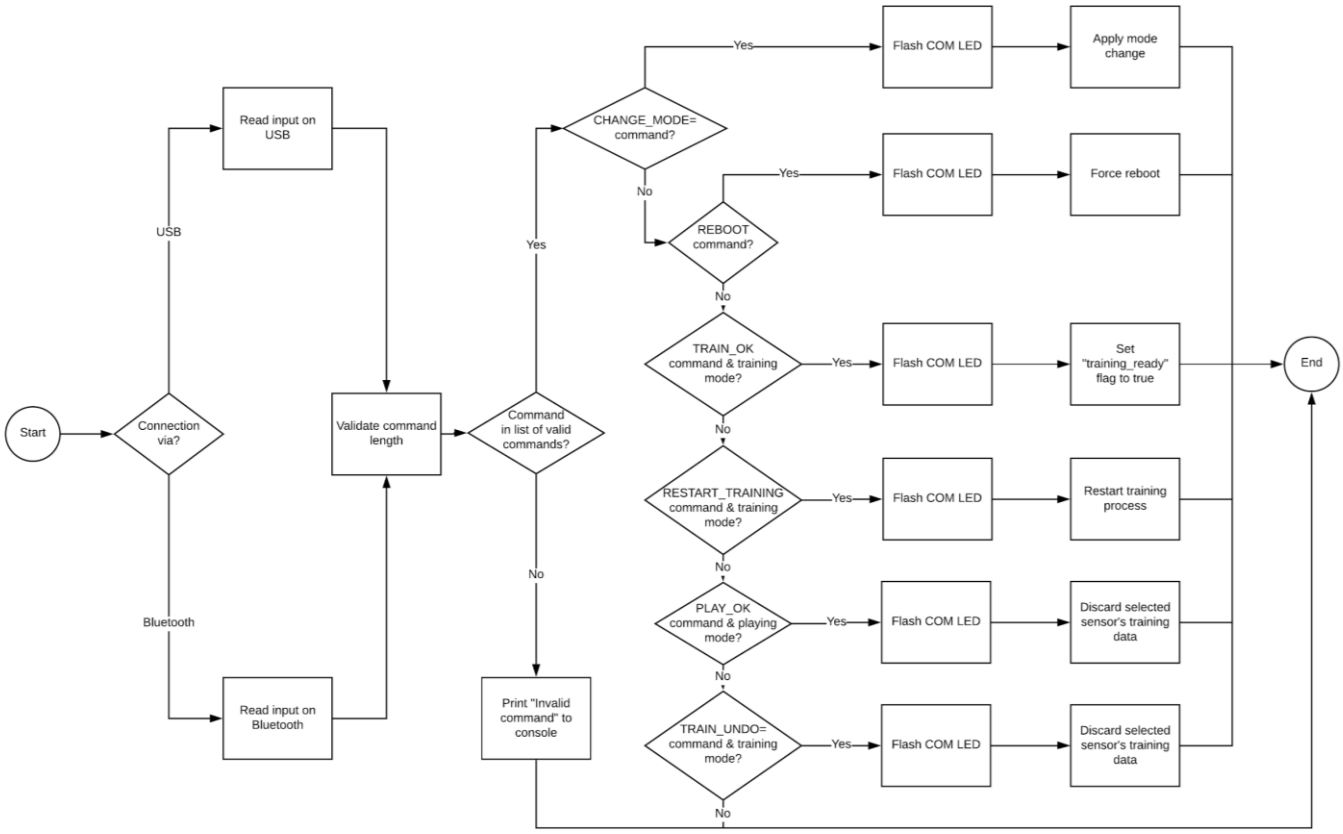


Figure 10: Flow Chart: Receive Command

3.1.4 Mode & Command Interactions

Example Training Sequence
Figure 11 shows an example command sequence as it might occur during the gameboard's training process.

Mode	ID
Ready	0
Training	1
Upload	2
Playing	3

Command	from	to	Mode
Training			
(Start)	-	-	Ready
CHANGE_MODE=1	Dashboard	Main Controller	Training
TRAIN={SensorID=3_CardID=72}	Main Controller	Dashboard	Training
TRAIN_OK	Dashboard	Main Controller	Training
TRAIN={SensorID=2_CardID=54}	Main Controller	Dashboard	Training
TRAIN_OK	Dashboard	Main Controller	Training
TRAIN={SensorID=1_CardID=72}	Main Controller	Dashboard	Training
TRAIN_OK	Dashboard	Main Controller	Training
CHANGE_MODE=2	Dashboard	Main Controller	Upload
Mapping data, save to Flash	Dashboard	Main Controller	Upload
UPLOAD_END	Dashboard	Main Controller	Upload
CHANGE_MODE=0	Dashboard	Main Controller	Ready

Figure 11: Training: Mode & Command Interaction Example

Example Playing Sequence
Figure 12 shows an example command sequence as it might occur during regular gameplay. Note that the playing dashboard acknowledges each *PLAY={...}* command from the gameboard by returning a *PLAY_OK* command.

Mode	ID
Ready	0
Training	1
Upload	2
Playing	3

Command	from	to	Mode
Playing			
(Start)	-	-	Ready
CHANGE_MODE=3	Dashboard	Main Controller	Playing
PLAY={Area='Oracle'_CardID=123_Type=CP}	Main Controller	Dashboard	Playing
PLAY_OK	Dashboard	Main Controller	Playing
PLAY={Area='Population'_CardID=5_Type=RP}	Main Controller	Dashboard	Playing
PLAY_OK	Dashboard	Main Controller	Playing
PLAY={Area='Oracle'_CardID=123_Type=CR}	Main Controller	Dashboard	Playing
PLAY_OK	Dashboard	Main Controller	Playing
CHANGE_MODE=0	Dashboard	Main Controller	Ready

Figure 12: Playing: Mode & Command Interaction Example

3.2 Oracle

The Oracle module software is based on the Main Controller software. However, its behaviour is not mode-dependent; instead, the Oracle permanently runs in *Playing* mode after initialization.
In its functionality, the Oracle also differs from the Main Controller: Instead of sending an individual message for every card that is placed, the Oracle sends the combination of currently placed cards in a single message every time a card is added or removed. This message adheres to the following format:

ORACLE={w_x_y_z}, wherein w,x,y, and z are the four IDs of the cards that are currently placed on the oracle. If less than four cards are present, the message gets shortened (e.g. *ORACLE={7_52}*). If the last remaining card is removed, an empty message *ORACLE={}* is sent. The rationale behind this is to signal that the Oracle shall currently not be considered by the backend.
In other words, the Oracle's sensors are permanently of type *combinatorial*, and all sensor inputs are condensed into a single message.

3.3 Training Dashboard

The training dashboard is a Python-/Kivy-based program that enables all the training-relevant functionality. This includes (a) establishing a connection to the gameboard, (b) defining the names of the Sensor modules, (c) choosing each Sensor module's type (*regular* or *combinatorial*), (d) uploading the data to the gameboard, and (e) optionally undoing the most recent step of the training process or (f) cancelling the entire training process. Upon successful completion of the training process, the user can start playing the game with the previously defined area names. For example, a user first connects his computer to the Main Controller via USB or Bluetooth. He then specifies three areas, (1) *Actions*, (2) *Buildings*, and (3) *Investors*. He activates each Sensor module with a card and chooses its type. Should the user make a mistake, he can undo the most recent step of the training process. After specifying the areas, he uploads the mapping data to the gameboard.

3.4 API Call Specification Dashboard

The API call specification dashboard is a Python-/Kivy-based program that enables all API call-relevant functionality. This includes (a) establishing a connection to the gameboard, (b) entering the API endpoints, (c) entering the API keys, (d) defining the format of the data to be sent with each API call, and (e) saving and reloading the current settings. After specification, every playing action sends the area name, the card ID, and the area type in the previously defined format to the API endpoint. This can be specified individually for both the Main Controller and the Oracle.
For example, a user first connects his computer to the Main Controller via USB or Bluetooth. Then, he specifies the API endpoint (e.g. *https://post-fossil-cities.com/play*) and the corresponding API key (e.g. *e3435g288fcdcsw8776dshbw*). He then specifies the data format in which the area name, the card ID, and the area type are forwarded (e.g. *pfc_play_action=[AreaName]_[CardID]_[Type]*). Now, the user can save the specifications and keep the dashboard running in the background.
Assuming a player now plays the card with the ID 57 on the area named *Actions*, which is of type *regular*, a message containing the data *pfc_play_action=Actions_57_RP* is sent to *https://post-fossil-cities.com/play*, using *e3435g288fcdcsw8776dshbw* as the API key.

4 Manufacturing

4.1 Introduction

This chapter describes the manufacturing process for all the gameboard modules. It describes the parameters for ordering components, as well as sources that were used for this prototype.

4.2 PCBs

For each of the three PCBs, multiple files are necessary for ordering production and assembly of additional units. Firstly, the Gerber file (.zip) contains the basic board layout for production of the bare PCB. Additionally, the Bill of Materials (BOM, .csv) contains the components needed for PCB assembly (e.g. USB ports, IO expanders). Lastly, the Pick and Place file (.csv) describes each component's location on the PCB. Because we opted to use small scale SMD components, PCB assembly was also outsourced due to increased complexity.
The PCBs used in the second prototype are of the following specifications:

- Layers: 2
- Thickness: 1.6mm
- Material: FR-4 TG130
- Surface Finish: HASL with lead
- Min. Track spacing: 6/6mil
- Silkscreen: White
- Min. Hole size: 0.3
- Finished Copper: 1 oz Cu

The PCBs used in our second prototype were ordered fully assembled from PCBWay.com and Quick-Pick.ch.

4.3 Cases

The cases used for each of the gameboard modules were 3D printed using the University's own Ultimaker 3 3D printer, located at the Tiny Makerspace (UZH Irchel).
They were printed using the following specifications:

- Material: PLA
- Print speed: 70mm/s
- Support: enabled
- Infill: 30%
- Printing temperature: 200°C

For each case, we include two files:

- .skp file (Google SketchUp): The original file, which can be edited to accommodate additional changes.
- .stl file: The file used for 3D printing with a slicer program (e.g. Ultimaker Cura), using the specifications described above.

5 Assembly

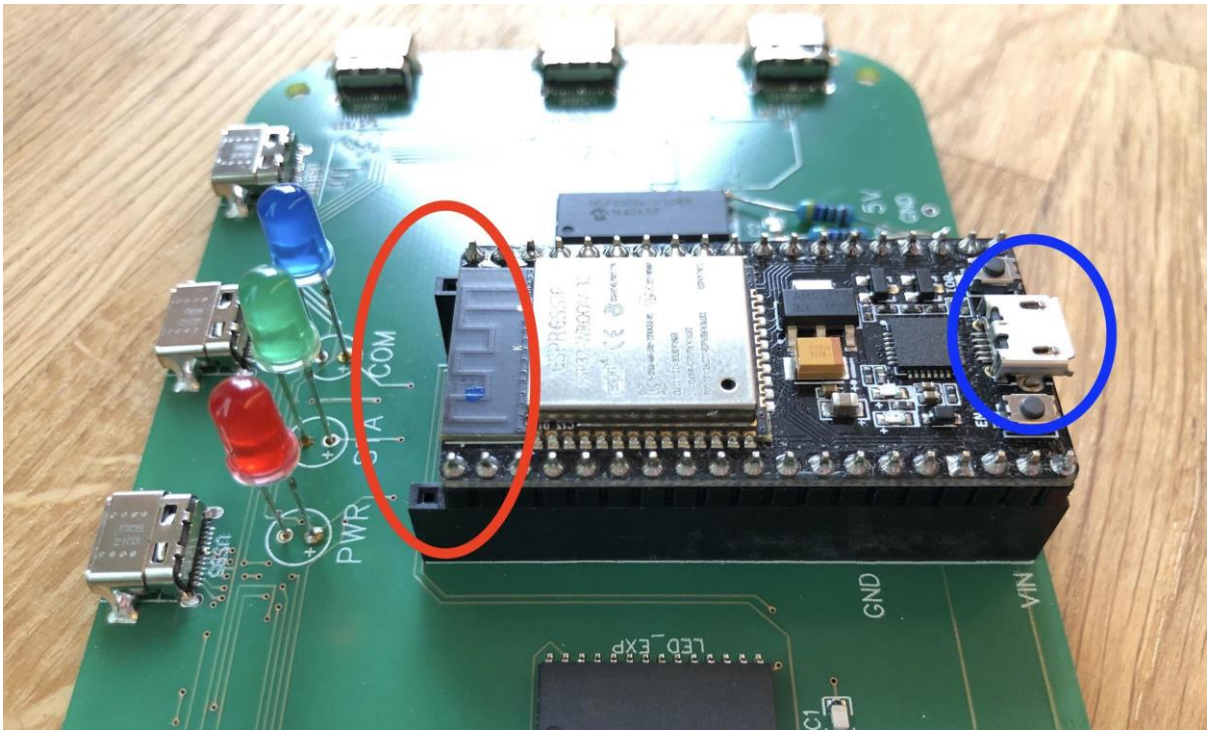
5.1 Introduction

This chapter describes, for each module, what parts are needed and how it is assembled. Each module is composed of multiple components: A 3D printed case, a PCB, and, for the Oracle and Main Controller modules, an ESP32 controller.

5.2 Main Controller module

The Main Controller module is housed in a 3D printed case. It contains the Main Controller PCB, to which an ESP32 controller is mounted. The assembly process is as follows:

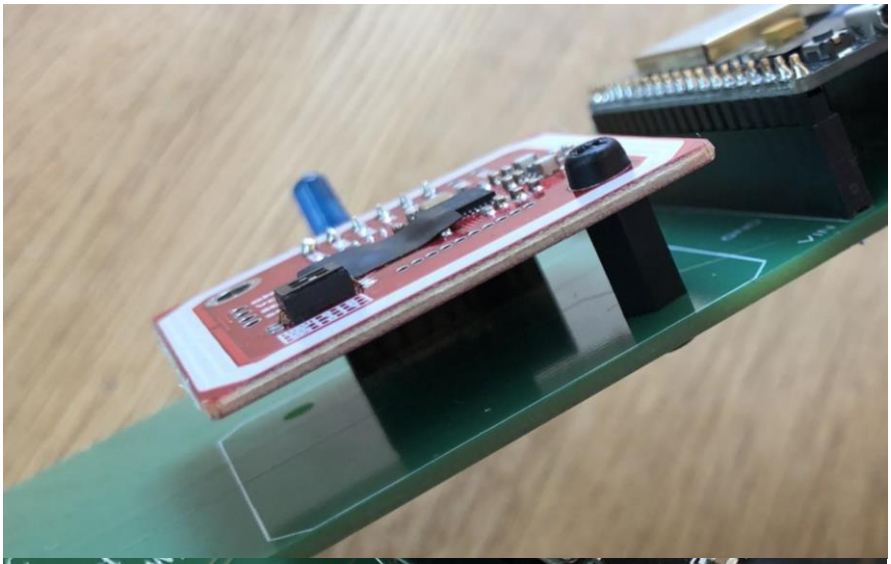
- Solder the three LEDs to the PCB, making sure they are in the correct order (red, green, blue from left to right). Also make sure their polarity matches the indications on the PCB.
- Mount the NodeMCU ESP32-S to the PCB, being careful to align its pins leaving open the topmost row (shown in red in the picture below). Also make sure the USB connector is pointing away from the LEDs (shown in blue in the picture below).
- Mount the PCB to the case using four 5-8mm long M3 screws.

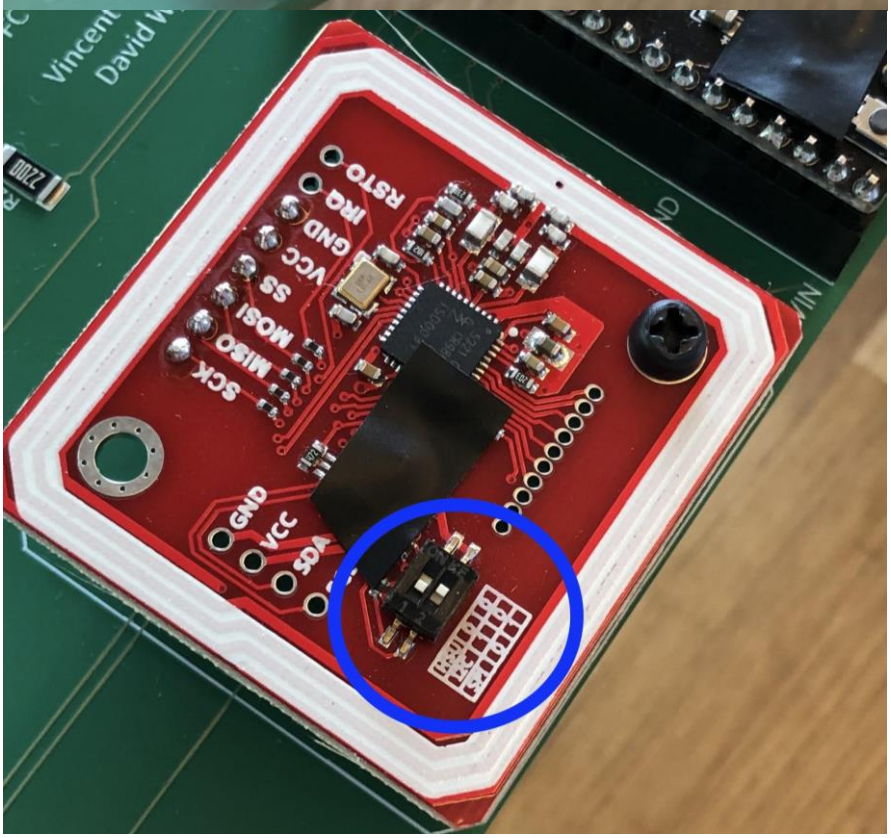


5.3 Oracle module

The Oracle module is housed in a 3D printed case. It contains the Oracle PCB, to which an ESP32 controller is mounted. Also, four PN532 NFC sensors are mounted to the Oracle PCB. The assembly process is as follows:

- Solder the three main LEDs to the PCB, making sure they are in the correct order (red, green, blue from left to right). Also make sure their polarity matches the indications on the PCB.
- Solder the four blue sensor LEDs to the PCB. Make sure their polarity matches the indications on the PCB.
- Mount the NodeMCU ESP32-S to the PCB, exactly as shown for the Main Controller (picture above).
- Connect the four PN532 NFC sensors to the PCB so their outlines match the white markings on the PCB.
- For each NFC sensor, install a 10mm nylon standoff with two M3 screws as shown in the first picture below.
- Set each sensor to SPI mode with the white DIP switches, as shown in blue in the second picture below. We also opted to cover the onboard LED with black tape.
- Mount the PCB to the case using three 5-8mm long M3 screws.



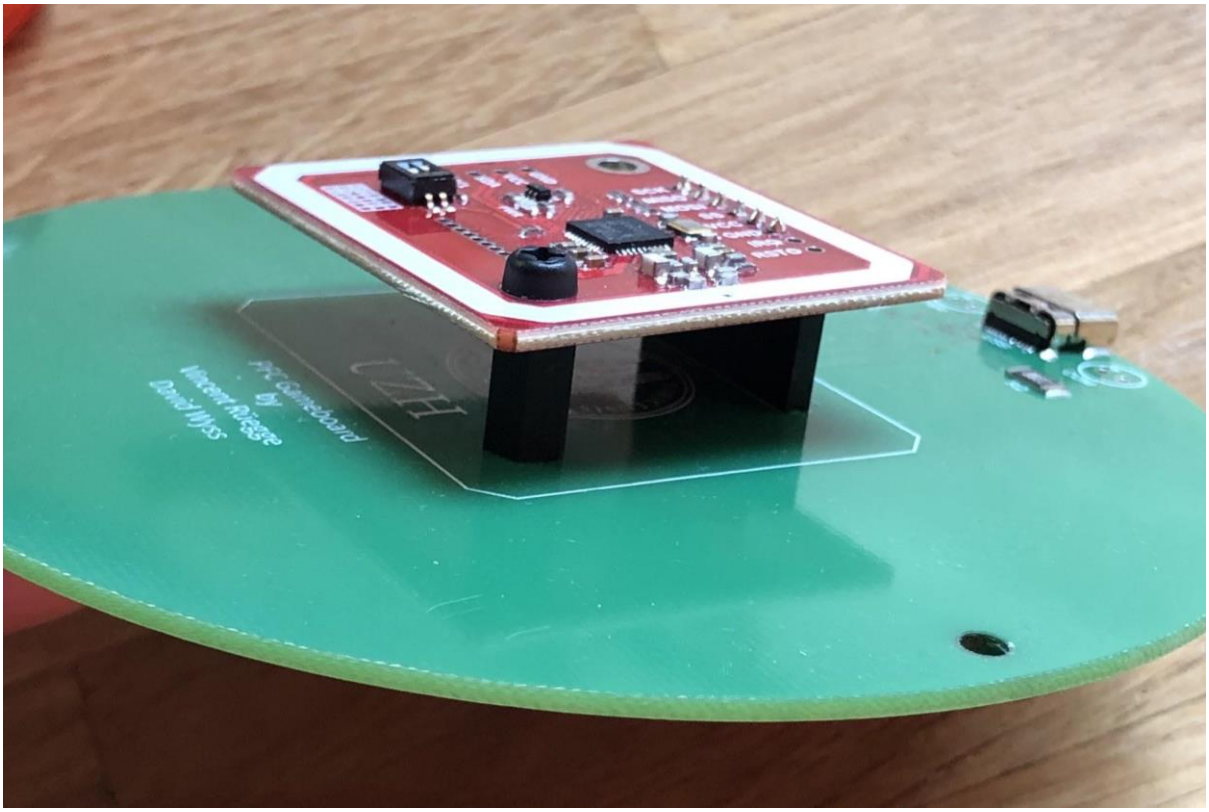


5.4 Sensor module

Each Sensor module is housed in a 3D printed case. It contains the Sensor PCB, to which a PN532 NFC sensor is mounted. The assembly process is as follows:

- Solder the two LEDs to the PCB (red for PWR, blue for COM). Also make sure their polarity matches the indications on the PCB.
- Connect the PN532 NFC sensor to the PCB so its outlines matches the white markings on the PCB.
- For the NFC sensor, install a 10mm nylon standoff with two M3 screws as shown in the first picture below.

- Set the sensor to SPI mode with the white DIP switches, as shown for the Oracle module. We also opted to cover the onboard LED with black tape.
- Mount the PCB to the case using two 5-8mm long M3 screws.



6 Setup

6.1 Software

Training and API specification software

- To run the training dashboard, using any Python development software (we chose to use *PyCharm*), run the file *layout.py* located in the *trainingDashboard* folder, using a Python 3.7 interpreter. You may need to create a configuration by pressing the "Edit configurations" button in the top right corner, then pressing the + button to add a new configuration. Choose the *layout.py* file as the script path, and Python 3.7 as the interpreter.
- To run the API specification dashboard, using any Python development software (we chose to use *PyCharm*), run the file *APISpecification.py* located in the *APISpecification* folder, using a Python 3.7 interpreter. You may need to create a configuration by pressing the "Edit configurations" button in the top right corner, then pressing the + button to add a new configuration. Choose the *layout.py* file as the script path, and Python 3.7 as the interpreter.

ESP32 Software

- Connect the ESP32 to your computer using a micro USB cable.
- Using the free editor *Atom* with the *PlatformIO* plugin, open the folder of the ESP32 software you want to install (either *ESP32 Game Board* or *ESP32 Oracle*).
- The *platformio.ini* file specifies the upload configuration; if using different ESP32 modules, setting may need to be adapted here.
- Change the *upload_port* parameter to the respective port name on your computer that the ESP32 is connected to.
- Within *Atom* with *PlatformIO* enabled, press the "*Upload*" button on the top left (third button from the top, which looks like an arrow facing to the right)
- The software will then be uploaded to the ESP32. Upon completion, you may close *Atom* and start using the module your ESP32 is used in.

6.2 Modules

Connection

- Connect up to five Sensor modules as you need to the Main Controller using USB-C cables. The cables have to be connected in numerical order, starting from port 1 without skipping any ports. Due to hardware constraints, no more than five Sensor modules are currently usable; more information on limitations in the section *Future Work.*
- For the Oracle module, nothing needs to be connected.

Power

- To power the Main Controller, connect a power bank or powered micro USB cable to the port opposite of the LEDs on the Main Controller. **Note**: Do not use any USB extensions or USB ports that cannot provide adequate power (e.g. from smartphones).
- To power the Oracle module, connect a power bank or powered micro USB cable to the built-in USB extension. **Note**: Do not use any USB extensions or USB ports that cannot provide adequate power (e.g. from smartphones).
- Both Main Controller and Oracle Modules will turn on and initialize automatically once power is supplied.

6.3 Playing Cards

Download the application and label the cards

- Download the *NFC Tools* application (Android, iOS) and launch it
- Switch to the *Write* tab
- Press *Add a record*
- Press *Text* and write *CardID=xyz* in the dedicated entry field (where x, y, and z represent the card's ID). Three digits are required (e.g. card ID 1 would be stored as *CardID=001*)
- Press *OK* and write the contents to the NFC tag.

6.4 Training Process

Connection

- Turn on the Main Controller and connect it to your computer via USB or Bluetooth
- Open the training dashboard application
- Wait until the Main Controller's status LED (green) stays lit
- Select the correct device from the dropdown menu in the top right corner, labeled *Select device...*

Training

- Press the "Start Training" button and wait for the Main Controller to acknowledge it by flashing the communication LED (blue)
- Place a card on any area, and enter the corresponding area name when prompted
- Press *Regular* for a check-in / check-out sensor, or *Combinatorial* for a check-in / leave-in sensor. Wait for the Main Controller to acknowledge it by flashing the communication LED (blue)
- Repeat the previous two steps for each area you wish to specify
- *Optional:* Tick the checkbox in the top left corner named *Areas* to get a list of all already trained sensors
- *Optional:* Tick the checkbox in the top right corner named *Log* to get a list of all the actions performed via the training dashboard

Completion

- Once done, press "Finish and Upload"
- Wait for the Main Controller to acknowledge the end of the training process by flashing the communication LED (blue)
- After the training dashboard indicates that the program may be closed, the Main Controller is ready to start playing

Undo / Cancel

- Press the *Undo* button in the bottom left corner to delete the most recently trained area
- Press the *Cancel Training* button in the bottom right corner to discard all training data and start over

6.5 API Call Specification

Connection

- Turn on the Main Controller or Oracle module and connect it to your computer via USB or Bluetooth

- Open the API Call Specification dashboard application
- Wait until the Main Controller's / Oracle module's status LED (green) stays lit
- Choose which device you want to specify the API calls for by choosing the appropriate screen in the bottom right corner (*Areas* or *Oracle)*
- Select the correct device from the dropdown menu in the top right corner

Specification and save

- Enter the API key
- Enter the API endpoint
- Specify the data format. You can add card IDs, area names, sensor types, and card combinations by pressing the respective buttons (*Add CardId, Add Area, Add Type, Add Card Combination*
- Press *Save & Apply* to save the current settings and apply them to incoming play actions

Load settings

- Press *Load current settings* in the bottom left corner to load the currently saved specifications in the respective fields. Only empty fields will be overwritten by loading current settings.

7 Challenges

Over the course of our master project, we faced various challenges spanning from software to hardware to a global crisis.
From a software perspective, the choice of the dashboard design language (Python Kivy) revealed itself to be suitable for the scope of this project, but not recommendable for more sophisticated applications. Additionally, hardware-specific libraries for the ESP32 had to be significantly altered to fulfill our intended purpose (e.g. the NFC sensors' library had to be adapted to get up to five sensors working together).
From a hardware perspective, designing the PCBs proved a significant challenge as neither of us had any experience on that aspect of the project. All things considered, the PCB design was a crucial part of the project, as the second prototype's functionality and design are based on having a working PCB for each module. Failing to achieve this would have resulted in a significant deviation from previously established requirements (e.g. small size and user-friendly, modular design).
All of these challenges were additionally embedded in the global COVID-19 crisis. This lead to significant delays in the production and delivery of multiple hardware components, which postponed the completion of the entire project by several months. Therefore, in the closing stages, project management became increasingly important.

8 Future Work

The second post-fossil cities game prototype offers a first connection between the software running in the background and the user playing the game. Additionally, it allows for great flexibility and is able to cope with various use cases. Different numbers of sensors can be connected, the Oracle module can optionally be incorporated, two different sensor types are available (regular or combinatorial), the area names can be specified, as well as the outgoing API calls for both the Main Controller and the Oracle.
Regarding the maximum number of connectable sensors, we reached practical hardware limitations with only five sensors as opposed to ten that were initially planned. This is not caused by our custom PCB, but by the used PN532 sensor boards themselves, which were not designed by us. As they communicate using the SPI protocol, all sensors utilise a single *MISO* (Master In Slave Out) line to send data to the ESP32. However, the sensors' hardware design does not permit inactive sensors to leave the *MISO* line unchanged, instead forcing them to pull its voltage towards 0V. This behaviour depends on how many sensors are connected; the more are connected, the more difficult it becomes for any active sensor to send data to the ESP32. Adding an additional pull-up resistor between *MISO* and the board's 5V supply allows communication to work reliably with up to five sensors, however connecting any more will prohibit all sensors from sending data back to the ESP32. There are multiple ways of redesigning the hardware in order to overcome this issue including, but not limited to, the following:

- Using different SPI compatible NFC sensors that are capable of leaving the *MISO* line floating (this would require redesigning the Sensor module PCBs)
- Redesigning the Sensor module PCBs to directly include an NFC sensor, bypassing the need for a prefabricated sensor board
- Using an I²C controlled switch, (e.g. ADG728BRUZ) to dynamically assign the MISO line to the active sensor (this would require redesigning the Main Controller PCB as well as adapting the main controller's software, which may impact possible playing speed)

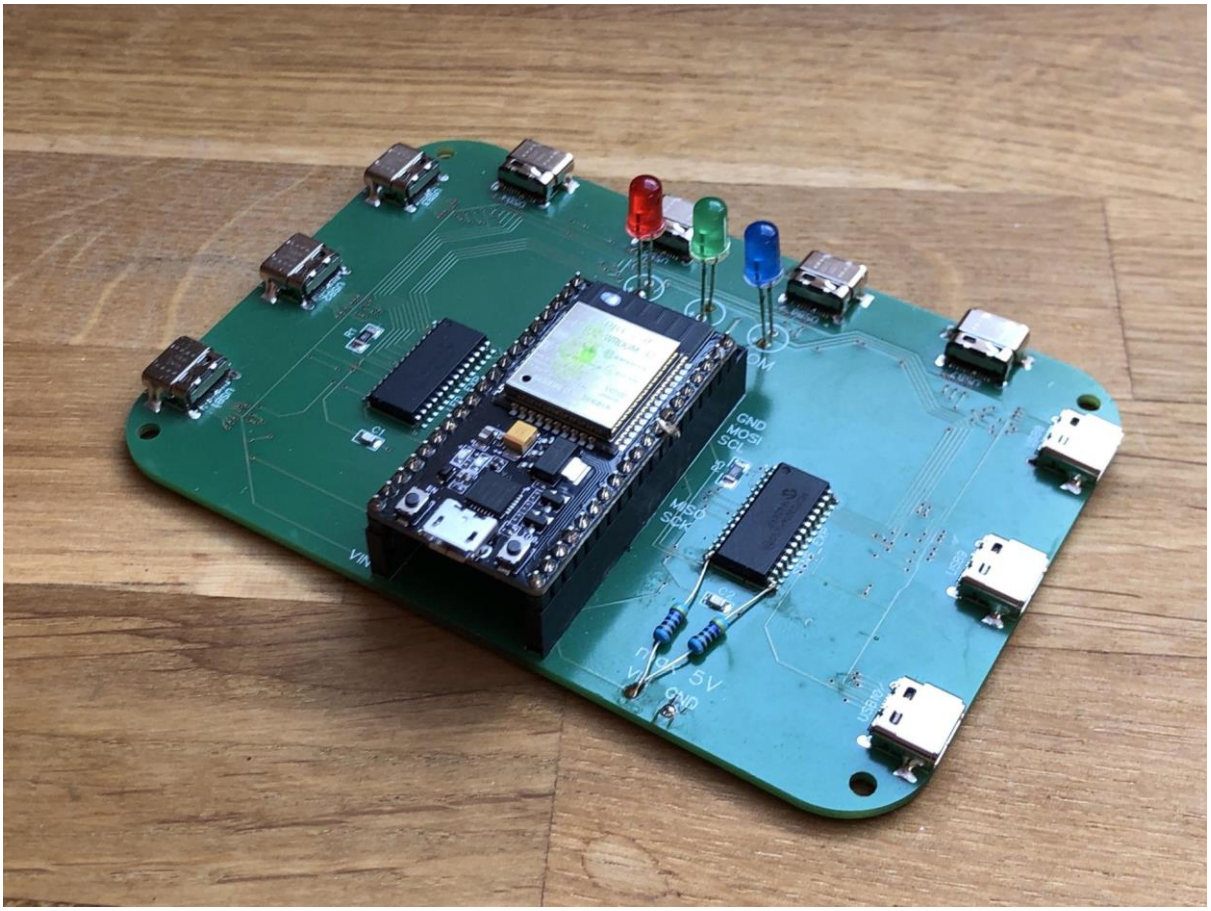
Nevertheless, the prototype can still be expanded and future work can use it (or certain components) as a baseline to build upon. Furthermore, a database could be connected to the gameboard and incoming

playing data could be captured and stored. This would allow for future analyses of the playing data. Additionally, hardware changes such as the inclusion of different types of NFC cards could be made.

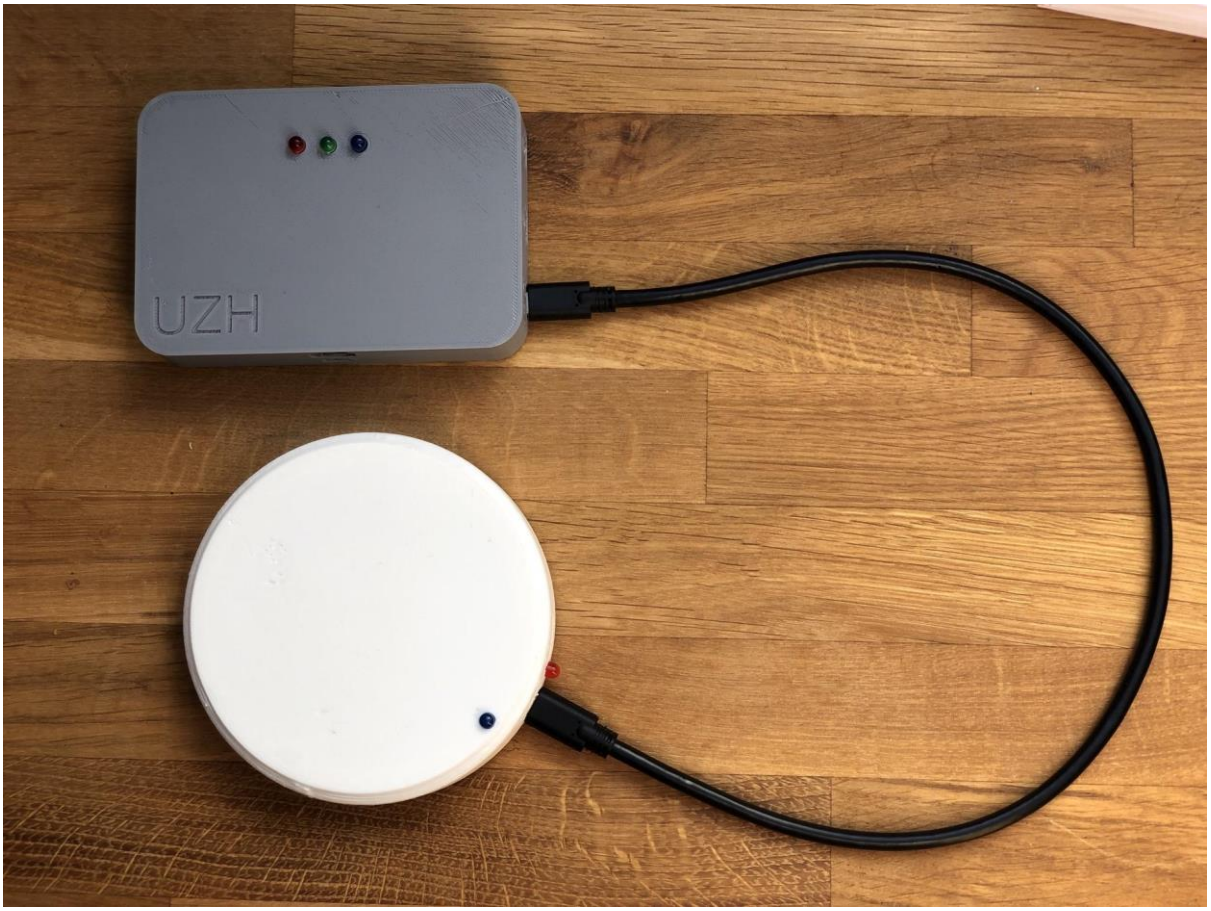
9 Pictures

9.1 Main Controller


UZH
UZH
7
6
5
4

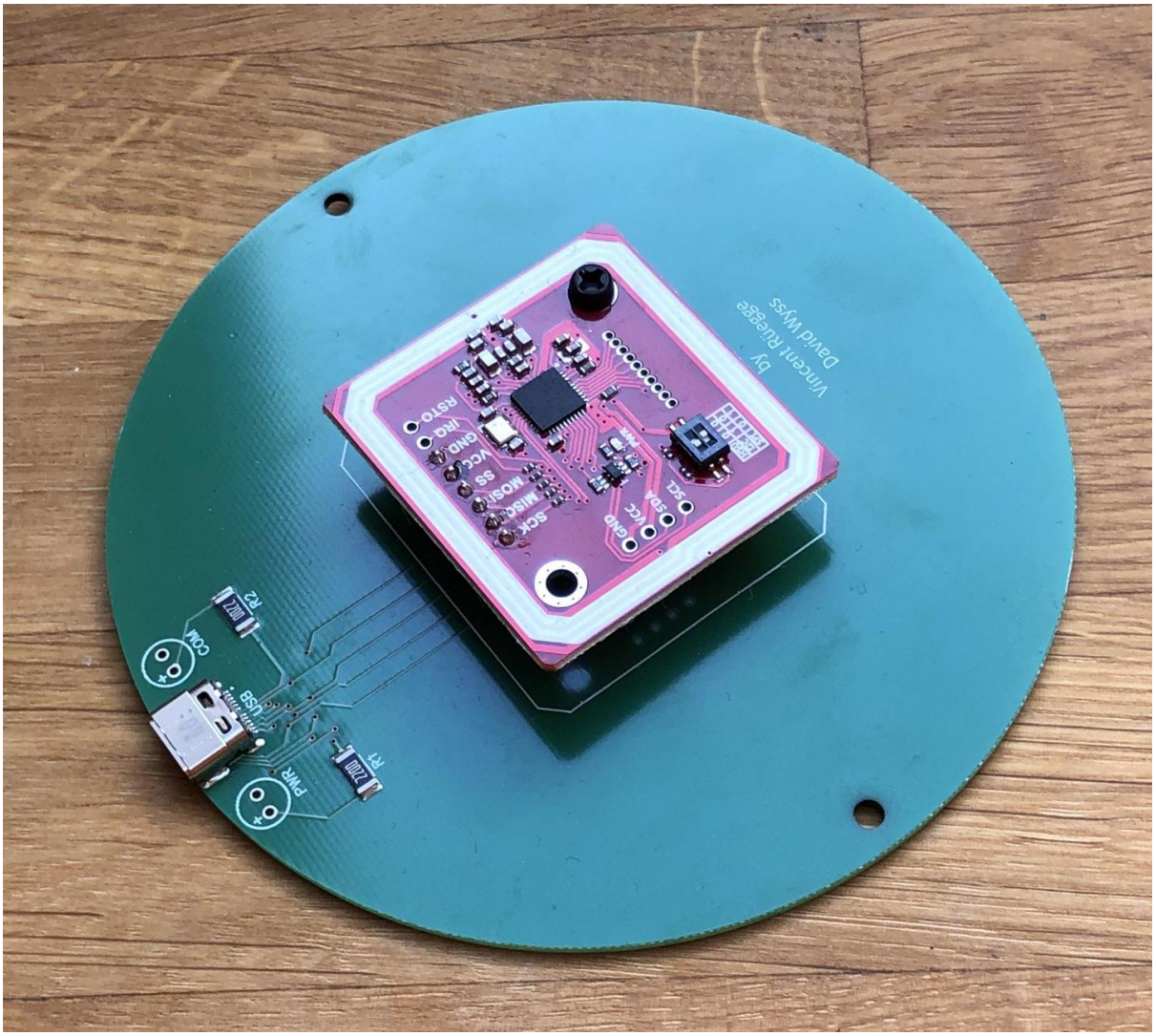

GND
MOSI
SCL
5V
GND


UZH
Invest in Stocks
Buy shares of companies on the stock market.
Action: To invest in stocks, transfer money to the accumulation field.
Return: Either through model / through industry -> could be a lever / driver for paradigm shift! tbd.
Effect:
Decrease Carbon Intensity
Decrease the carbon intensity of construction and store carbon in construction materials.
Action: Add a red cube to building/ mobility that is renovated or newly built. Maximum height of red cubs indicated by politics.
Return: None.
Effect: Less GHG emissions, carbon storage
Services Instead of Material
Instead of selling products (e.g. a house) you sell services (e.g. dwelling).
Action: Prerequisite: "Longer life Time" implemenented.
Return: Higher endowment.
Effect: You have an interest to produce long-living goods.
Prosumer Services
Invest in prosumer service (e.g. grid provision and long-term storage)
Action:
Return: If the consumer plays teh card "prosumer" increased returns.
Effect: Decentralized energy production.

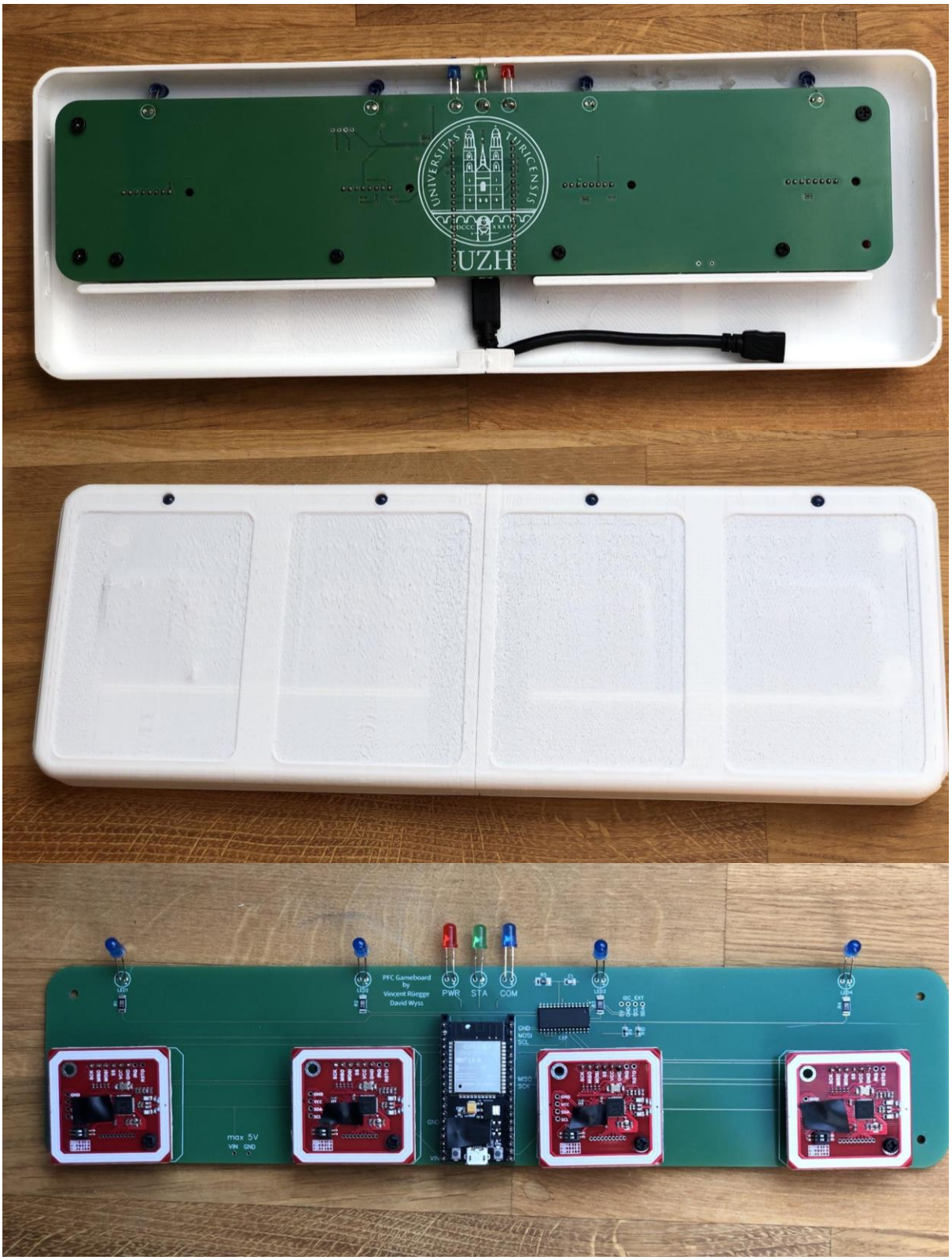

UZH

9.2 Sensor Modules

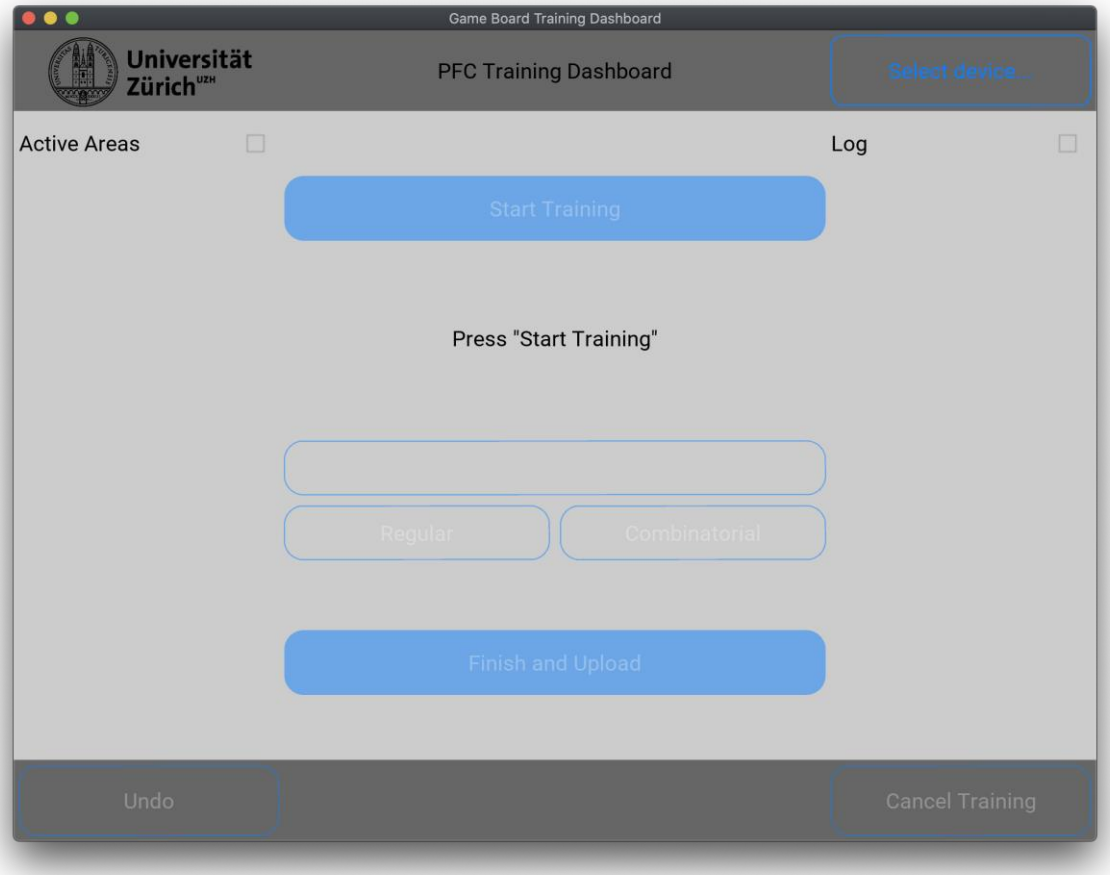

HZ
UNIVERSITAS
TURICENSIS


by
Vincent Rueegge
David Wyss
RST0
IRQ
GND
VCC
SS
MOSI
MISO
SCK
GND
VCC
SDA
SCL
R2
220Ω
COM
USB
PWR
R1
220Ω

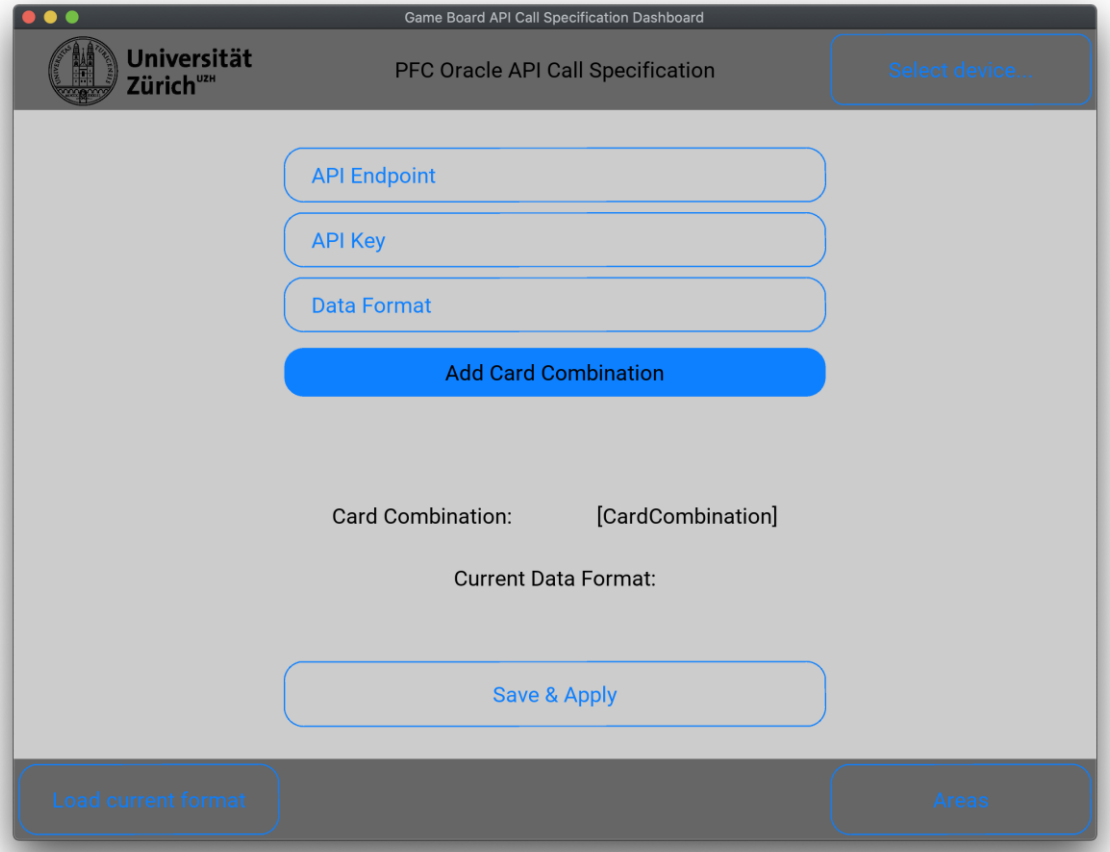
9.3 Oracle Module



9.4 Training Dashboard



9.5 API Call Specification Dashboard


Game Board API Call Specification Dashboard
Universität Zürich
PFC Oracle API Call Specification
Select device...
API Endpoint
API Key
Data Format
Add Card Combination
Card Combination:
[CardCombination]
Current Data Format:
Save & Apply
Load current format
Areas


Game Board API Call Specification Dashboard
Universität Zürich
PFC Area API Call Specification
Select device...
API Endpoint
API Key
Data Format
Add Area
Add Card ID
Add Type
Area:
[AreaName]
Card ID:
[CardID]
Type:
[Type]
Current Data Format:
Save & Apply
Load current format
Oracle