Programming Journey – Escaperoom 'Where is my Emma?'

What additional things (libraries, paradigms, etc.) did you learn during the project?

The main new aspect we learned is the `pygame` library. We learned how to display a frame and '`blit`' images on it. We learned how to handle userinputs or so-called '`pygame.events`' (keyboardinputs as well as mouseclicks). Further because of the high interactivity we learned the use of threads in python. The concept of threads was already known ('Einführung in die Softwareentwicklung'), so we decided to use them for handling simultaneous processes.

With a (for us) new app 'Poti poti' we learned to design pixel-images and display them. The pixel-look helped a lot in identifying the placement of the many 'buttons'. With paint, we could reduce all images to our wanted resolution in pixel*pixel. For design aspects, we also learned to use and load a .tff font and display it on our game.

We learned how to handle blocking user-input (with blocking while-loop) graphically. We used many global variables as status-variables to keep track of players' actions. Here we learned that you need to state at the beginning 'global x' if you want to change x in this function. Access is possible by only referring to x.

We learned how to implement backgroundmusic and play sounds in our game with '`pygame.mixer`'.

[For the scientific-ier part of the program I learned how to connect `matplotlib` with `pygame` and display plots there, use `scipy` for linear regression and intensified the work with `pandas dataframes`]

What challenges did you face?

One challenge we faced, was an updating problem. We learned that if file 'a.py' has a global variable that has been changed, file 'b.py' , that has an import statement at the beginning: 'from a import *' the variable is not automatically updated in file b.py ' after the change. We approached this problem by printing this variable in both files and recognizing asynchronicity. First, we tried to import it again (after the change was made) and it worked. Later we solved that problem more elegantly by implementing a getter-function (modularity and visibility reasons).

Another challenge with the import was the meaningful connection between several .py files. This was clear as we learned that one cannot import file b.py in a.py and vice versa simultaneously (which is logical because then you could write everything to one file). We approached this by splitting the program concepts into files `startscreen.py`, `endscreen.py` and `door1.py`, `door2.py` and `door3.py` at the beginning. Because the starscreen and `endscreen` should be equal no matter your doorchoice. After implementing and some redundancy we implemented `display_components` which holds functions and status variables that are used by every file (or the majority of files). The 'bigger' function `handle_userinput` was put in one extra file. The output on the resultscreen was put in `resultscreen.py`.

A further but quickly solved challenge was the simultaneous gameplay and time display. Because (as already stated in the beginning) the concept was known, we decided to use threads. One (probably the main one, like in java) handles the gameplay and user-input and the other one the displaying of time. First (naive) approach was that everything is handled by the main process but even after setting the fps to 60, one second was incremented every three (or so) because that thread had so many others to handle in between.

Another tricky problem was the evaluating of one click as 'one click'. We observed that if we click on exactly one pixel and hold it, the click is only counted as one. But if you click and slightly move your mouse the click often is evaluated as more. Because we first checked if the event '`pygame.mouse.get_pressed()`' happened and if it was a left-click. We assume that this event does not check if the mousebutton was released, only that it was or is pressed. We tried solving this by blocking functions like blocking while-loop or time-sleep. These functions prevent the evaluation of further clicks but make the game unnecessary long. After some research, we found another `pygame.event` which

made our lives better: `pygame.event.MOUSEBUTTONUP`. When we checked for that, we know that a previous click happened. So now the player can interact with left and right-clicks, but that is not a problem for our game.

What further additions could be made?

I think the first obvious addition could be increasing the complexity of the rooms task (like in real escaperoomgames) or adding levels/new rooms. If that is not wanted, one could also add a look-around function which lets you look around in the room from different perspectives (also like in real escaperoom games). We already implemented some zooming functions so equally we could let the user 'turn around' to look at other things in the rooms.

We also have implemented one task, where the key to the next door is collected but neither acknowledged nor displayed, so here could an itembox be added that displays it. Furthermore, we had the idea of an Avatar which is seen in the loading screen or the final_words room, but we did not entiretly embed it. Here is room for improvement.

In game we could add more soundeffects/interactivity posiblities.

But since there is no button-object in this library and we were forced to laboriously implement and calculate all button-coordinates by ourself, we would use other libraries/languages for this kind of game-development in future projects.