

# Self-report log

*(dossier)*

Alina Maximova (s2032074)

[a.maximova@student.utwente.nl](mailto:a.maximova@student.utwente.nl)

Group 14 HFEP

## Table of contents

<b>December 2, 2020</b>	<b>3</b>
<b>December 9, 2020</b>	<b>3</b>
<b>December 11, 2020</b>	<b>4</b>
<b>December 15, 2020</b>	<b>5</b>
<b>December 16, 2020</b>	<b>7</b>
<b>December 17, 2020</b>	<b>9</b>
<b>December 19, 2020</b>	<b>9</b>
<b>December 22, 2020</b>	<b>10</b>
<b>December 23, 2020</b>	<b>11</b>
<b>January 17, 2021</b>	<b>11</b>

## December 2, 2020

Today I have created a Git repository for my group, so that we can work together on our project without sending each other the file. I have helped my fellow group members to install Git and explained how to use it and why I suggested using it for our project. My group members agreed with my arguments and we decided to start using it.

I have created a Git repository on gitlab, you can access our repository by this link: [https://gitlab.utwente.nl/s2032074/programming\\_project\\_group\\_14](https://gitlab.utwente.nl/s2032074/programming_project_group_14) . I have added Jana and Sylvan to the repository.

However, we have encountered some problems installing the project. First of all, we decided to work in VScode as Spyder has a very weird integration with git. Secondly, Jana has Mac, so the proper installation for her took much longer, she had some errors while installing it, but I helped her to find the solution and we finally managed to install everything and make it work.

Finally, I have started writing the README file, so that when we will be submitting the assignment we could add a link to our repository and a teacher could clearly understand what is going on in this repository.

The last thing we have done today as a group - we have decided on what project out of three we are going to work on. We decided to work on the third, the hardest one.

## December 9, 2020

I have created a file on GoogleDrive where I described the possible subtasks for the implementation of our assignment. Right now it looks like this:

Files that need to be implemented:

1. game.py  
In this class we will define the setup for the game. The sequences that will be run by a program (which a user will have to repeat) -> taken from, also the number of mistakes a user makes should be counted
2. Settings.py  
In this class we will define the drawing of blocks and the environment in general
3. Sequence.py -> Sylvan  
This file will keep the sequence that the user have to represent
4. data.py  
We will record the data to this file, also we will be able to retrieve data from it. It is drawn from the game.py and
5. Block.py -> Alina  
This file will contain blocks to use in this task
6. saveAs.py -> Jana  
This file will provide the ability to save the results to a file

After this we have discussed this division with my group and agreed that for now this is something we can work with, we also came to a conclusion that probably while implementing this subtasks may change a little bit, however, for now this is a direction where we are going.

We have decided that we will be working on separate branches, so that we do not disturb each other and only when we are done and sure that our code is perfect we will merge it into the master branch.

We decided that till 18/12/2020 I should finish Block.py, Jana - saveAs.py and Sylvan - Sequence.py

## December 11, 2020

Today I have started with implementing block.py file. For that I have created a separate branch on git, so that I do not mess up with the master branch. I decided to define a class with all the necessary functions. For now I thought that I will need to track the location of the block (the coordinates of the top left corner), and the length of the side will be constant, so I will be able to make a function which will define the area inside the block. It is necessary for tracking the user input: whether he/she pressed on the block or not. Also, I will probably add the id of the block, so that later we can refer to this block quite easily.

As long as we are working with git it is quite easy to see the contribution of each of the team members. I think that in the end of the project we will share the link on git or will just make some screenshots from it to illustrate how the progress was made.

While coding I have encountered a problem “module “pygame” has no “init” member”. I have spent some time trying to figure out how to solve this problem. Finally I found the solution, this is quite a common problem for those who are using VScode. To fix this I opened settings.json and then inserted in {} the following: “python.linting.pylintArgs”: [ “--extension-pkg-whitelist=pygame” ]

Also, I had several complaints from my compiler about not using docstrings, this was solved with ease: I just added them.

Finally, I spent 4 hours on writing, debugging and testing this file, but now it is complete and is working. Currently it has the following functionality:

1. Class with functions for a block.

2. Constructor where the block will be initialized using x and y coordinates of the top left corner of the square, length of a side, and a boolean value for tracking whether the block was pressed on or not
3. A function for drawing a block.
4. A function for checking if a user pressed on the block and changing the color of the block if user pressed. This is important for further development of the test.

## December 15, 2020

Today I was working with git. Started with renaming branches and deleting unnecessary ones. Why renaming? Because any branch should clearly show what is done/kept on this branch. For example, we had Sylvan's branch where he was working on sequence. He called this branch "sylvan\_branch". I renamed it into "Sequence".

Also, Sylvan just created the template for the sequence, so my main task for today was to figure out what he did, and, the most important part, build-in my Block class into the sequence.

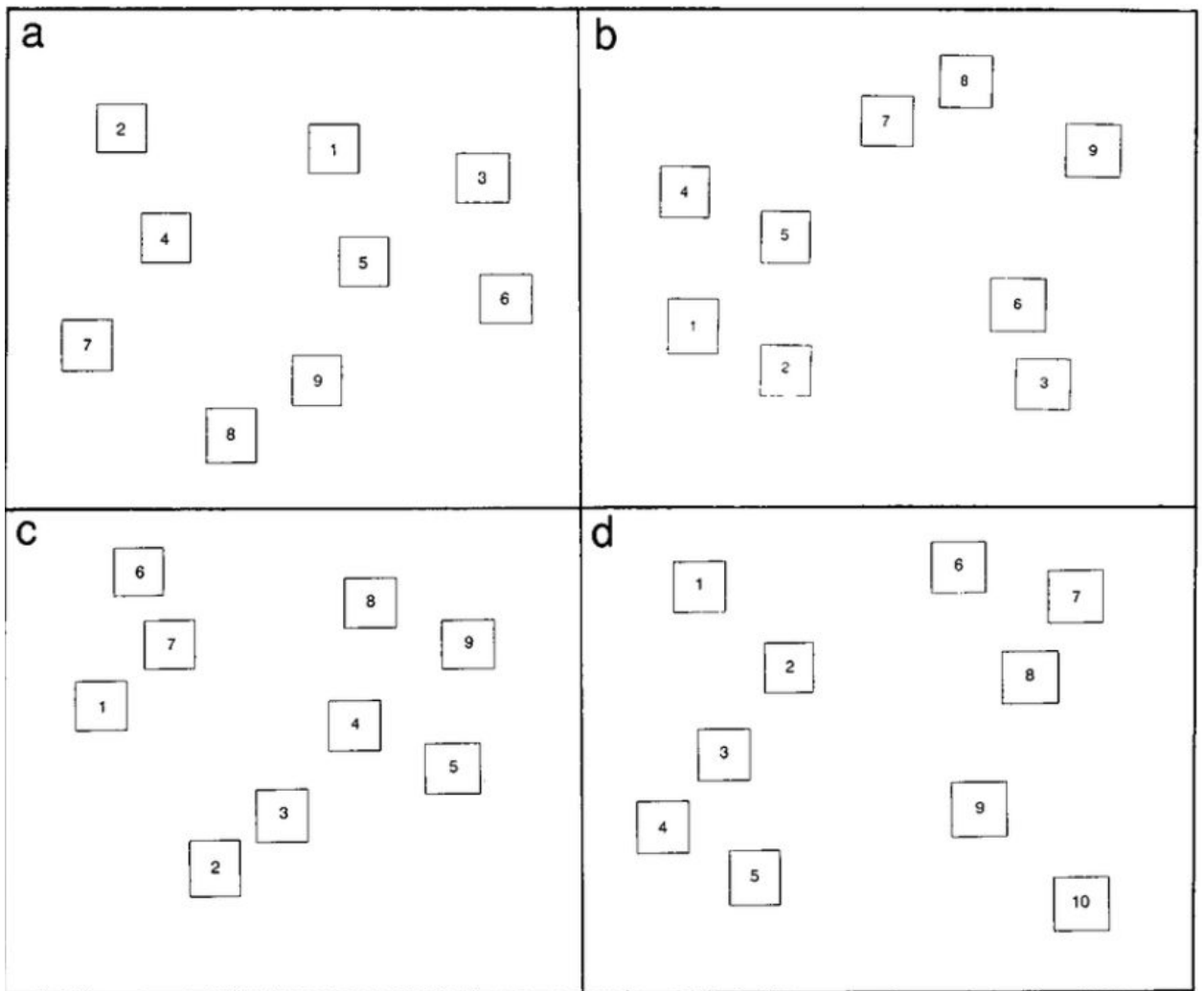
I have imported the Block.py into the Sequence\_update.py, so now I am able to use methods that I have defined earlier in the new class. In a couple of hours I managed to implement the function for creating the sequence. I have thought that the program will have 4 different predefined sequences which all should be used in the experiment. By now I have created only one sequence, but before continuing with others I need to test it and see if blocks are located correctly, as I want them.

Finally, I have released a minor update in Block.py, now the function 'clicked' can draw the block even if it is not clicked.

Evening update:

I have created the first test for the sequence file, it went successfully, so now I can move on to create 3 more sequences. I created one more sequence, it has different locations of blocks. Then I created a function which allows the computer to show the sequence to be repeated to a user. Also, I have created a function that compares the sequence that the user enters to the sequences presented by program. Moreover, I have added a field ID to Blocks.py because using IDs of blocks it will be very easy to compare these blocks which is extremely important in comparing the user input sequence and the computer made sequence.

pic 1 illustrates the four predefined sequences:



[pic 1]

Then I decided that I need to create a dictionary for each of the sequences mapping the blocks' IDs with the actual blocks. Thus, I will be able to track which block was tapped by the user.

I had a very interesting issue with timer. Apparently if the code looks like in pic 2, the program will not wake up for the for-loop, to fix it I added a simple print function as shown in pic 3.

```
#waiting 2 seconds to allow user see all the blocks
time.sleep(2)
#print('')

#changing color back to black
for block in self.sequence:
    block.pressed = False

for block in self.sequence:
    block.draw_block(screen)
    pygame.display.update()
```

[pic 2]

```
#waiting 2 seconds to allow user see all the blocks
time.sleep(2)
print('')

#changing color back to black
for block in self.sequence:
    block.pressed = False

for block in self.sequence:
    block.draw_block(screen)
pygame.display.update()
```

[pic 3]

December 16, 2020

Today I have changed the file with what needs to be done, now it looks like this:

Files that need to be implemented:

1. game.py  
In this class we will define the setup for the game. The sequences that will be run by a program (which a user will have to repeat) -> taken from, also the number of mistakes a user makes should be counted

2. Settings.py  
In this class we will define ~~the drawing of blocks~~ and the environment in general

This class should have the following methods (all screens are in width: 1000, height: 800): -->

1. Welcome screen
2. Screen with the description of the task
3. Screen with the rules for using (e.g. do not touch blocks which are grey already, they are selected) → depends on implementation
4. Intermediate screen (There will be 4 tests for 1 user → intermediate screen should be shown 3 times and once the final screen)
5. Final screen
6. Goodbye screen
7. "Repeat the sequence" screen → Screen design Jana

3. Sequence.py -> Alina  
This file will keep the sequence that the user have to represent. Challenge rn: add timer for user and save/print the result
4. Data -? The last thing to do, maybe will record data as text to .doc or as table to .xls  
We will record the data to this file, also we will be able to retrieve data from it. It is drawn from the game.py and
5. DONE Block.py -> Alina  
This file will contain blocks to use in this task
6. saveAs  
This file will provide the ability to save the results to a file
7. Transition table. It is not smth in code, but the requirement from uni (Jana)

In the beginning of the tutorial today I have shown this to my group and we discussed it and we all agreed on it. saveAs function appeared to be too difficult for Jana, therefore, she is now responsible for making screens as it is easier. Also, she is the one responsible for the transition table. Starting from today Sylvan will be working on save-to-file function.

I am continuing working on sequence class. Added a function that calculates the longest correct sequence that the user remembered and repeated. Finally, I added a timer that calculates the time the user spent on finding the sequence.



By the end of this day it turned out that we as a group have almost finished our implementation. Starting from now Jana and Sylvan will be working on screens (7 screens in total), I will be working on final file `game.py` where I will use all the predefined classes and will use the `game/test/task` logic.

## December 17, 2020

Today I merged `save` and `sequence` branches into the master branch and then from master I have created a new branch `game_without_screens` where I started working on the task logic. As name suggests, on that branch will be the complete test including saving to results to file, but without all the screens.

I started with issuing a minor update for `save.py`. Sylvan did a great job, so I just added some commenting and tested this function again. It works perfectly. Also, I added the rounding of the double variable of time to two decimals, so that in the file it looks nice.

Finally, I wrote and tested the `corsi tapping` task, implementing every feature that our team has made so far. Basically, all the functionality except for the screens.

## December 19, 2020

During our meeting on Wednesday, Martin mentioned that it may be nice to add some flexibility to our project. More specifically, Martin suggested that we may create a table to define our sequences outside of our Python program.

I have conducted some research on the matter and found out that the most used options are to use either a JSON file or a CSV file. In the former case, the file itself may not be as easily readable as an Excel-type CSV file (when using Excel), but it is more flexible in that it does not require me to create a separate CSV file for every sequence but allows me to define them all in one file.

Moreover, CSV files are generally less readable when not using an application such as Excel. The main issue I have found with CSV files is that they work very similarly to databases in that you have to specify all the table headers in advance. Afterwards, every table entry (or CSV entry in this case) will have the same number of headers. This makes it significantly harder to create sequences of various lengths if I wish to use a single file.

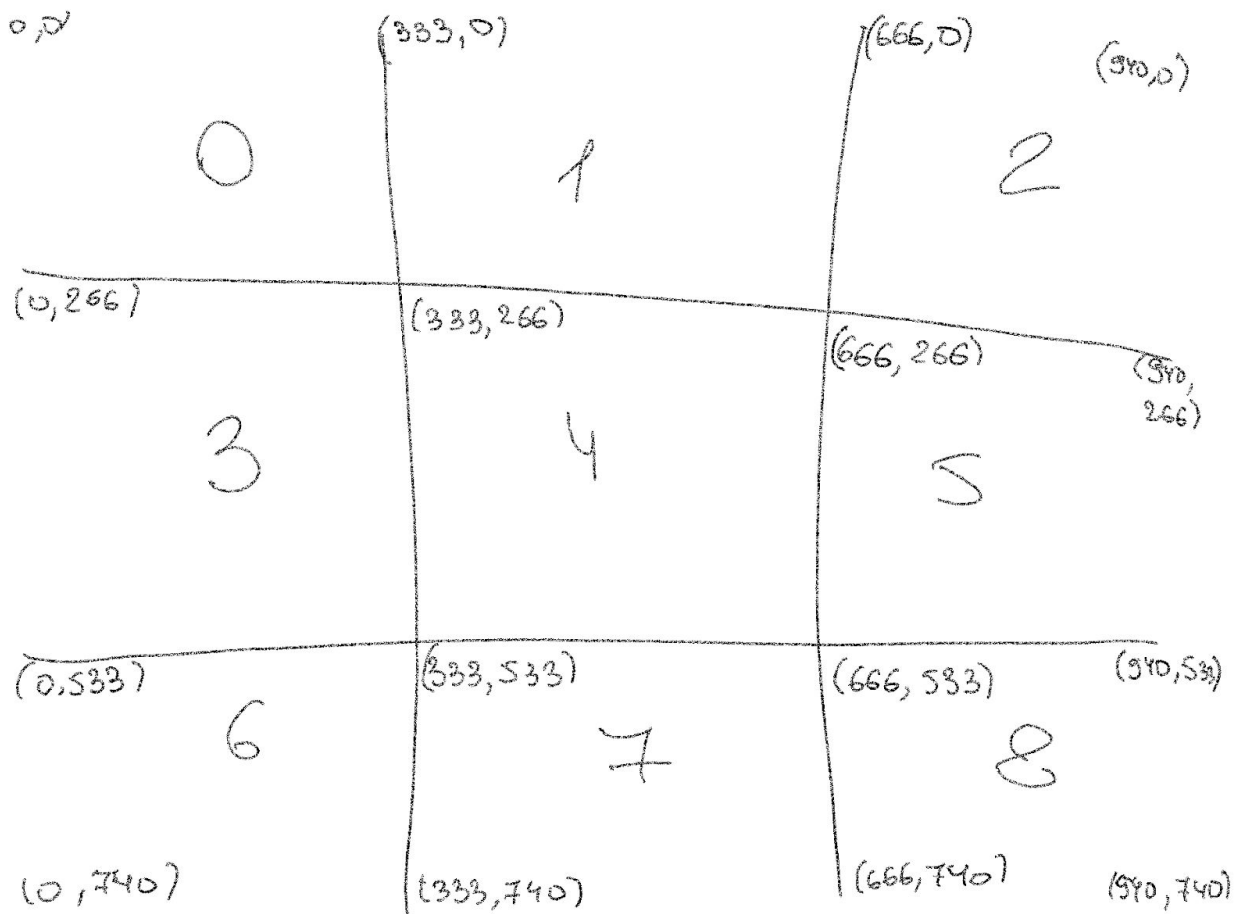
In addition, the code required to read all entries from a CSV file is significantly more extensive than the code required to read a JSON file and convert its contents into a Python object.

I would like to be able to define all of our sequences in a single file to avoid having to go over an entire folder, so my preference in this situation would be to use a JSON file. However, since Martin mentioned that he would like to see a table (or, as I understood it, a CSV file), I thought it best to ask his thoughts on the matter.

According to his reply, it became clear that he wants us to use CSV file instead of JSON. Thus, this is what I am going to do later during the holiday.

## December 22, 2020

Today I have started adding flexibility to the project, I started with the possibility to generate randomized sequences. To make the length of the sequence more or less flexible, I have decided to invisibly separate the screen into 9 zones (one zone for one block) as you can see in the picture below:



The idea is the following: the list with numbers 0 to 9 is randomly generated. Then blocks are created in the spaces above, if 0 is the last number in the list - sequence will consist of 8 blocks, 9 otherwise.

I have defined and tested this method, it works perfectly fine giving different unpredictable sequences. Moreover, such an approach allowed me to avoid blocks overlapping.

Finally, I have updated the README file, adding there the description of what happens when the program is running, what sequences are used.

## December 23, 2020

Today I completely finished my part of the project. I have implemented the ability to read a sequence from the CSV file. I was thinking about using Pandas library, but then decided to use a method from Numpy which appears to be much more helpful in the current situation.

Also, I have updated the README file adding all the necessary instructions, also I updated the game.py, now it is 95% done. The only thing that still needs to be implemented - seven screens should be written and used. As we have separated the work, these screens are not my part, therefore, I have no idea when they will be done.

## January 17, 2021

Today I updated and finished the transitional table. I also uploaded it on Git and made all the necessary changes in the README file on Git. Sylvan has finished all six screens, so today we are going to add them to the rest of the program together.

First of all we added all the functions to one class, then we called the relevant functions in game.py. We also had to add proper screen work to the functions, but it was not very challenging. Finally, we tested our program, it works perfectly. I will submit our project later in the evening.