color. At best this color should be randomly chosen to not confuse anybody, since there are usually a lot of vectors with the same similarity to a given cultural vector and this similarity is difficult to express in color. For example the vector $\vec{v}_1 = [0, 1, 1, 0]$ and $\vec{v}_2 = [0, 1, 2, 0]$ are very similar so one could come up with the idea to give them similar colors like orange and red. But then $\vec{v}_3 = [0, 2, 2, 0]$ shares a smaller similarity with $\vec{v}_1$ but the same with $\vec{v}_2$ therefore there is a problem to attach a color gradient to the similarity. This is why random colors were chosen.

I then tried to pre-generate the set of random colors for all possible cultural vectors which worked out for the small initial numbers of traits and features. But when these numbers are already slightly increased a combinatoric explosion occurs. The amount of possible cultural vectors for M traits and N features is given by M^N. For 4 features and 3 traits this is feasible with 81 possible combinations but already for 10 and 10 features and traits respectively this scales up to 10^{10} possibilities which takes way to long to compute and therefore would very much limit the application.

For this reason I chose to check for new cultural vectors in each iteration. Every new vector gets assigned with a number which is assigned to a random color. With this every vector keeps the same color throughout the simulation even if a certain vector disappears from the grid for some time due to the dynamics.

Since the color representing the cultural vector of the agent on the grid gives no clue about how similar it is to for example its neighbors I also added the possibility of showing separation lines on the grid where the dot density of the separation lines represents the similarity in steps of 20 percent, so less than 20 percent similarity is depicted by a solid line, for 80 percent or more no line is shown and a gradual increase of dot density for less than 80, 60 and 40 percent. The similarity for the separation lines has to be calculated for each neighbor of each individual agent in every time step. Therefore this takes a lot of computational effort and runs smoothly only on small grid sizes.

4.3. Embedding into a GUI

Most of the time of the project went into this part. This was the first time for me creating a GUI so I had to look into the PyQt documentation many times. I decided to realize the GUI by using the PyQt grid so that every container of the GUI covers one or several squares of the grid. This allowed for an easy assembling of the different features of the GUI and also made a stepwise addition of features possible.

The key element of the application is the aforementioned visualization of the grid which is simply realised by showing the matplotlib figure. matplotlib comes with a good compatibility with PyQt. The sliders to adjust the parameters, namely the grid size, the number of features and traits respectively and the number of neighbors up for interaction, were implemented with the PyQt QSlider class. Each of the respective values of the sliders is shown next to them and also passed on to the model.

Every time after adjusting one of the parameters the simulation is reset to lead to the right results. The pause and reset buttons are established using the QPushButton class and the checkboxes using the QCheckBox class.

The explanation of the model and its properties is embedded into a single scrollable QLabel in which the text elements and images in form of PDF files are placed.

5. Final remarks on the project

When offered to do this project instead of regular weekly exercises for the study achievement, I gladly took it. I generally preferred this option as it offers more freedom and allowed me to organise my working hours freely, for example during lecture-free periods. Also the models which were presented to examine in the project were interesting and I liked working on it. Although I have to say that I underestimated the amount of work that went into this. As already mentioned before in the report, most of the time I spent on the project went into the making of the GUI. Since I had no prior experience with JavaScript the level of the provided GitHub repository from the ComplexityExplorables was too high for me. It was very helpful that later on the option was given to also create the GUI in python. With this option it was possible to make the GUI in a reasonable amount of time even without having worked with PyQt before.