Memory game

Goal: I will learn to design and implement a graphical user interface with Qt. I will learn to use Qt documentation.

Instructions: Create a new project: student/13/pairs_gui/.

Although you have no code template, you can copy code e.g. from the examples on this round, as well as from earlier pairs project on round 4.

Important

Before starting to write code, read carefully the whole assignment. Especially, remember to use version control with a sufficient number of commits.

Note

This project will be done independently. Working in pairs is not allowed, but it will be considered as plagiarism.

The task is to implement a graphical user interface for the pairs game. You can find its rules from the assignment of the pairs project on round 4 (unless you do not know/remember them already). You are allowed to slightly change the rules, if you document them. In this assignment, you will write a document describing those features you have implemented, the structure of your program, and design decisions.

The section *Minimal requirements* tells which features there at least must be in your implementation.

Header comment and feedback language

In the same way as earlier, the header comment is again required. You can use the first project's assignment as an example (see 4.6 (P) Pairs). It is sufficient to add header comments in the files you modify.

As in the previous projects, the feedback language will be chosen in the submission box (at the very end of this page). We will **only** use the language you have given in the submission box of the final submission.

Assignment

Your task is to implement a pairs game with a graphical user interface enabling the player to play the game (approximately) following the pairs game rules. (More precise requirements can be seen later.)

The game must allow variable number of cards. The amount of cards is either asked from the user, or there can be a constant for it in the program, and by changing the value of the constant, the game can played with different amounts of cards. The number of players can be fixed (two or more), or it can vary in the same way as in the pairs game on round 4. If desired and if it feels useful for you, you can use the Card and Player classes from the earlier pairs project. You can also freely modify these classes.

Cards will be laid on game board in the same way as in the project of round 4, i.e. the numbers of rows and columns must be as close to each other as possible (c.f. the assignment 2.3.2 Nearest factors, the code of which can be found from the template code of pairs project on round 4).

Cards will be laid on the game board in random order, and thus, you will need a random number generator. You can draw the cards in the same way as in the project on round 4, or you can implement a better way to draw them. (Note also the section Random number generator, a bit later.)

It must be possible to turn cards in some way. You can again number rows and

columns to enable the user to give four numbers totally (”coordinates” of two cards). You can use any widget (QLineEdit, QSpinBox, QDoubleSponBox etc.) for reading the given input, or you can use key commands. Perhaps the most natural way is to define cards as push buttons, whereupon you can turn a card by clicking it. In this case, you must take care that at most two cards can be visible at a time.

Turning a card means that its face will be visible. It is enough to have just letters on the face as in the project on round 4. Moreover, the sides of a card must at least have a different colors, whereupon turning also means that a color changes.

If the turned cards were not pairs, they will be turned hidden again. If they were pairs, they are removed from the game board, and the amount of points of the player in turn increases. Removing a card can mean that the card really disappears, or that the card becomes more difficult to detect, it can e.g. fade or its color can change more similar to the color of the game board.

In the program, turning a card may happen too quickly, and thus, the players cannot see, which figures (letters) the cards had. Therefore, the user interface can have a push button of its own for closing a card, which enables a player to look at the cards as long as they want.

Another possibility is to use delay, whereupon the cards stay visible for a certain time before they will be turned hidden again. This can be implemented with a timer. However, the timer need not be an attribute of a class as in the Timer exercise. It is simpler to delay an action with the following code:

```
QTimer::singleShot(DELAY, this, SLOT(wait()));
```

where DELAY is an integer constant telling the amount of delay in milliseconds and wait is a slot function, implemented by yourself, containing the delayed action. Remember to put the definition of the slot function at private slots in the header file. You can use different names from those shown in the code above. For the amount of delay, it is better to use a named constant (as here DELAY) instead of a magic number.

During the game or at least when the game is over, the situation is reported, which means telling the amount of cards/pairs collected by each player. During the game, you can show the situation as whenever found pairs disappear from the game board, they will move to a player’s card deck. The deck can be drawn in a QGraphicsView area as (thin/shallow) rectangles in the similar way as the circle was drawn in the example on moving circle. Each time a player gets a pair, their deck increases with new rectangles.

In laying cards, you need mathematics (arithmetics). However, the needed mathematics is very easy.

Random number generator

Random number generators used on this course have such a property that (exluding very big seed values) the first drawn number is the lower bound of the chosen interval. Thus, if the random number generator is a local variable of a function and if only one number is drawn per a function call, the drawn number is almost always the same.

To make the random number generator work better, it should be an attribute of MainWindow class. The same holds for the distribution variable that determines the interval. Both of these attributes will be initialized in the constructor, and then the first random number can be drawn and discarded. In this way, you can prevent the first number to always be the same.

You can always use the computer clock as the seed value of the random number generator, unless you want to add a widget for reading the seed value given by the user. During testing it is better to used some fixed seed value. In this way, you can repeat the same deal of cards several times.

You can draw the letters for cards in any way you want, but one possible way is as follows. Denote the numbers of cards as *n*. Take a string and assign the value ”AABBCC…” for it such that the number of characters is *n*. Mix the letters in the string by using the function shuffle from algorithm library as shown in section 5.3 or as you probable did in the exercise 5.3.1 Mixing alphabets.

Minimal requirements

By implementing the minimum functionality of the project, you can earn 50 points at maximum. At least, you must implement the following features:

- The number of players can be fixed, but it must be at least two.
- It is shown which player is currently (or next) in turn.
- It must be possible to play the game with a variable amount of cards. However, there can be an upper bound for the amount of cards. The upper bound cannot be less than 20.
    - The amount of cards is either asked from the user, or there is a constant in the program, and by changing the constant the game can be played with different amounts of cards.
- Cards must be graphical user interface components (e.g. push buttons or rectangles), not just text.
- Cards must be laid such the amounts of rows and columns are as close to each other as possible.
- There must be a way to turn cards.
- When cards are turned, there can only be at most two cards visible at a time.
- It enough to use letters as the figures of the cards. There are always two same letters/figures. The back of a card can be empty, but the back and the face of a card must differ from each other at least by their color.
- When turning two cards visible, their faces must be visible long enough.
    - This can be implemented such that there is a delay after which the cards are turned hidden, or such that there is a push button for closing cards.
- The result of the game is told/shown at least at the end of the game.
- You must document the functionality of the game. The document must include instructions to play the game, for example, what are the rules of the game. In

addition, it must tell what happens and what can happen in different situations. The document must also describe the program structure and the design decisions you have made.

You must implement the points above in such a way that they work. The better your implementation is and the better programming style you have followed, the more points you will get.

To get more points, pay attention also to the user-friendliness of the game. You can, for example, control the push buttons to make them either enabled or disabled. For this purpose, the class QWidget has the slots setEnabled and setDisabled.

It is especially important to pay attention to the robustness of the game. This means that the program works in a sensible way (and do not, for example, crash), even if the user does something stupid. For example, in the case of buttons, you can disable those ones that cannot be used in a sensible way at the moment. In any case, try to think beforehand, which kind of use cases there can be. Test your program by imaging that you are an evil person, whose purpose is to make the program crash. If you succeed in this, fix your code to prevent such situations.

In general, we will pay attention to the naming convention used in your code. However, the names generated by Qt do not always follow a good programming style. For example, if you automatically generate a slot, the name can be as on_pushButton_clicked. This can be considered as a bad style, since it uses both underscores and CamelCase style in dividing the name into parts. However, you should keep the names given by yourself consistent in their style.

Extra features

As said earlier, by implementing the minimum functionality of the game, you can earn 50 points at maximum. By implementing extra features for the game, you can earn another 50 points at maximum. You can get points only from working and documented extra features, i.e. also the extra features must be documented.

Assistants will add bonus points into the later section in Plussa while reviewing the projects. Besides the functionality of extra features, assistants take into account how

clean your solution is.

Examples of extra features are listed below (approximate maximum points given in parenthesis):

1. The game allows a variable number of players, and that number is asked from the user. You can name the players as Player1, Player2 etc. (10 p.)
2. The user can name the players. (5 p.)
3. The game status is shown graphically, for example with a QGraphicsView widget, which has space for each players' card piles. A pile grows when a player finds pairs. (10 p.)
4. Besides the amounts of cards collected by each players, the program also tells, which cards each player has collected. (5 p.)
5. The time used in the game is counted. For this extra feature, you have two choices, one of which will be evaluated:
    1. The time used is shown to the user, when the game ends. (5 p.)
    2. The time used so far is shown during the game. (10 p.)
6. After the game ends (or anyway), it is possible to start the game again from the beginning without starting the program again. (10 p.)
7. The game includes a score board for storing the points the player has earned. The stored data is preserved between games (i.e. the data is stored into a file). The score board may include points from different players, which means that the user can give their user name at the beginning of the game (at least they wish). (10 p.)
8. There is a figure on the back of cards (the same figure on all cards). (10 p.)
9. Instead of letters, there are figures on the face of cards. (20 p.)
    1. As said earlier, the game has a variable number of cards, but on the other hand you can set an upper bound for that number. Therefore, you must have as many different figures as is the upper bound divided by

two. If the game is played with a smaller number of cards, you need not draw all figures.

If you, in implementing last two extra features, take png figures from internet, do not violate copyrights.

In addition you can suggest an additional feature of your own. The suggestions must be made at least a week before the deadline. Suggestions can be made after the exercise classes or by sending e-mail to the course's e-mail address. The course staff has the right to accept/reject the suggestions, and define the maximum points for the feature.

Note that by successfully implementing all the features listed above, you seem to get more than 50 points. However, the maximum amount of points is 50, so if you get more of them, they are decreased to 50. It is still worth trying to implement as much extra features as you can, since you might not get the maximum points from each implemented feature. (Note that the evaluation criteria of the projects at the end of the course is stricter than at the beginning.)

Testing your program and user instructions

This project will not be tested automatically, but you are responsible for testing your program with sufficient coverage by yourself. However, you must still submit your work in Plussa to get it evaluated. Note that the program must work on Linux desktop.

Create the file instructions.txt and write in it short instructions on how to use the program, what are the rules of your game, and which functionalities you have implemented. The instructions must be clear and exact enough to enable the assistant to test and evaluate your program. Check from the minimal requirements all the points, the document must contain. Remember to push also this file in version control.

If you use version control from command line, you can push the document file as any other file. If you use version control from Qt, you can add the document file into Qt project as follows. On the left from the edit window of Qt, you can a list of the

files included in the project. On top of this list, you can see the project name. Click it with the right button of the mouse and from the opening menu, select Add Existing Files ... (if the file instructions.txt already exists) or Add New ... (if you are about to begin to write the file).

If you have implemented an additional feature, but you have not documented it in the file instructions.txt, **you will not get points from the additional feature.**

Tips

- It is recommended to study the examples of the current round. In their code, you can find points that are useful in implementing the project.

- When you add new widgets in the file mainwindow.ui, their relative position to a possible graphicsView widget might not be correct, since the position of the aforementioned widget is reasonable to define in code (in the same way as in the example on moving circle). Try to set the widgets such that final user interface is visually as good as possible.

- Implementing the game will most probably require you to read Qt documentation. There you can find useful functions for implementing different features.

- You can exploit version control also in your experiments. After pushing the latest working version in Git, you can do even large modifications safely, since you know that you can return to the earlier version if needed. You can do as follows (for each modified file):

    - In Qt: Tools -> Git -> Current File -> Undo Unstaged Changes/Undo Uncommitted Changes

    - In command line: git checkout – <filename>

Special requirements

If you want your assignment to pass the evaluation, you must meet these requirements:

- The program must not have errors related to memory management. Therefore executing valgrind is recommended.
    - If you allocate memory with new, you must deallocate it with delete, unless deallocation happens via the parent-child mechanism (see round 12 at About widgets).
- Containers only from STL library are allowed, not e.g. from Boost.

Evaluation

To end up to assistants' evaluation, you must submit your work in Plussa, and it must pass the automated tests. (In this case, automated tests check only, if the program can be compiled without warnings.) If the automated tests give 0 points for your work, also your final points will be 0, and your work will not be evaluated by an assistant.

The assistant evaluates the submissions that have passed the automated testing (= 1 p) **and fulfilled the special requirements of the assignment** based on the last commit before the deadline, according to the following criteria:

- The functionality and user instructions of the program: 0-10 points
    - All the features mentioned in the list of minimal requirements have been implemented, and they are working.
    - The program does not crash.
    - The instructions inform how to use the game, and which additional features have been implemented. The instructions have been written clearly enough to enable assistants to understand them easily.
    - The game works as said in the instructions.

- The overall principle of the solution: 0-30 points:
    - The learning goals of the exercise have been achieved.
    - The program code has been split into logical, suitably long segments using functions, classes, and/or methods.
    - If the program uses classes and objects, these have been implemented according to the basics of object-oriented programming (see from round 4 About programming style at Object-oriented programming).
    - The data structure does not include repetitive data nor unnecessary parts. The chosen data structures have been used in a reasonable way.
    - The program code does not include unnecessary repetitions nor other unnecessary parts.
    - The program code does not include unnecessary limitations or assumptions or other forced solutions.
    - The implementations of the program’s structures are easy to understand.
    - Global variables have not been used in the program code (global constants are OK).
    - The program does not terminate with the exit function.
    - The program has no errors related to memory management (use valgrind, and note the updates in section 10.1).
- Programming style: -20-0 points:
    - Variables and functions are named clearly and appropriately.
    - The program uses named constants instead of magic numbers.
    - The program code is neatly formatted.
    - Each program line has at most 80 characters.
    - At the beginning of each file written/edited by yourself, there is a

comment explaining the purpose of the file, your name and student number and other necessary information. (see the assignment of the first project on round 4).

    - At the beginning of each function/method (in the header file if possible), there is a comment describing its working, return value and parameters.
    - There are comments in the code where necessary.
    - Comments are related to the current version of the program, not to an older one.
    - All the variables have been initialized.
    - The compiler does not give warnings while compiling.
- Using the version control: 0-10 points:
    - There are enough commits.
    - The content of commit messages is clear and relevant.

From the above list, please note that in this phase of the course it is assumed that the student follows good programming style. Therefore, you will not get points from following the style guidelines, but if you do not follow them, your points will be decreased. So, it is recommended to revisit the material about programming style on rounds 4 and 11.

Note

After the final submission, check from the web user interface of TUT GitLab that the submission is really in your central repository. Assistants will evaluate the latest commit that is submitted into your central repository before the deadline.

This project will not be tested automatically, instead each program that can be compiled without warnings receives one point. Testing is your own duty. The assistant gives 0-50 points to your submission after completing their evaluation. The maximum amount of points is 50 (it is not possible to get 51 points).