HW2 보고서

로빛 20기 인턴 2025407006 모시온

목차

1. 프로젝트 개요

본 프로젝트는 스택과 큐 중에서 스택을 2개를 활용하여 그림판의 실행 이전/이후의 그림을 불러오는 기능을 구현하였습니다.
undo/ redo를 사용하기 위해 2개의 스택을 활용하였으며 사용자가 마우스로 그림을 그릴 시 Qimage를 통해 해당 사진 레이어를 저장하는데 이때 사용자의 그림 횟수는 동적으로 변하기 때문에 이를 벡터를 활용하여 할당하고 그에 따라 undo와 redo를 적용할 수 있도록 하였습니다.

가장 중요한 핵심은 undo를 실행할 경우 이후에 redo를 사용할 수 있게 되는데 이를 사용하기 위해서는 undo하여 사라진 과거 현재의 사진 레이어를 redo에 저장하여 LIFO를 구현하였습니다.

2. 알고리즘 설계

2-1. .h/.hpp 헤더 파일 설계

```
#include <QMainWindow>
//벡터 사용
#include <vector>
//마우스 사용, 이미지 사용 모듈
#include <QMouseEvent>
#include <QGraphicsScene>
#include <QImage>
```

사용자의 마우스를 통한 선 입력이 동적으로 변하기 때문에 이를 벡터 모듈, QT의 마우스 이벤트와 이미지 레이어, 이 전체 결과를 출력하는데 사용할 모듈을 호출하였습니다.

```
QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE
```

Class 또한 활용하기 위하여 QT UI를 C++ 코드와 연결하며 충돌을 방지할 수 있도록 하였습니다.

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

protected:
    void mouse_condition(QMouseEvent *event);
    void mouse_move(QMouseEvent *event);
    void mouse_let(QMouseEvent *event);
    bool mouse_event(QObject *object, QEvent *event);
    bool eventFilter(QObject *obj, QEvent *event);


private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

    void on_horizontalSlider_sliderMoved(int value);
```

Class 메인윈도우에서 부모 위젯을 초기화하며 ui 객체를 만드는 생성자 호출과 함께 소멸자를 호출합니다.

그 다음 protected 영역으로 마우스를 통한 선 입력을 구현하기 위하여 각각의 함수를 저장하였습니다.

mouse_condition(QMouseEvent *event) : 상태를 기록하고, 시작점을 저장

mouse_move(QMouseEvent *event) : 현재 점과 마지막 점을 연결하여 그림 작성

mouse_let(QMouseEvent *event) : 그림 그리기를 종료, Undo에 상태를 저장

mouse_event(QObject *object, QEvent *event) : graphicsView의 viewport 이벤트를 처리하며, 위의 마우스 관련 함수를 호출하는 중계하는 함수

eventFilter(QObject *obj, QEvent *event) : Qt의 이벤트 필터를 지정 위젯에서 발생하는 이벤트를 중간에 처리하는 함수

private slot 영역에서는 UI로 구현한 각 버튼 및 graphicView를 다루는 함수를 저장합니다.

on_pushButton_clicked() : Undo 버튼 클릭 시 동작

on_pushButton_2_clicked() : Redo 버튼 클릭 시 동작

on_horizontalSlider_sliderMoved(int value) : 브러시 크기를 조절 슬라이더 값 감지

마지막으로 private영역에서는 UI에 접근하기 위한 ui, 그림 작성을 위한 scence를포인터 변수로 현재 그려진 이미지를 저장하는 currentImage와 스택을 동적을 할당하며 사용하기 위한 undo, redo, 마우스의 위치를 기록하는 모듈 선언과 함수 선언을 하였습니다.

2-2. .c/.cpp 소스파일 설계

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QPainter>
#include <QPen>
#include <QMouseEvent>
#include <QEvent>
```

이전 과제1에서도 호출한 그림 작성과 관련한 모듈과 마우스 이벤트를 처리할 수 있는 모듈을 호출하였습니다.

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //QGraphicsScene 초기화
    scene = new QGraphicsScene(this);
    ui->graphicsView->setScene(scene);

    //이미지 초기화
    currentImage = QImage(750, 500, QImage::Format_ARGB32);
    currentImage.fill(Qt::white);

    //초기 Undo 상태 저장
    UNDO();
    update();

    //graphicsView의 viewport에서 마우스 이벤트 감지
    ui->graphicsView->viewport()->installEventFilter(this);
}
```

부모 위젯을 받아서 값을 초기화하는 생성자를 QMainwindow에 종속되도록 하며 디자이너로 만든 UI를 동적할당하여 버튼과 슬라이더 입력을 접근할 수 있게 하였습니다.

그 다음으로, UI 위젯들을 메모리에 할당하며 부모 위젯인 this(메인윈도우)에 종속하고 그림판 동작시 사용할 scene생성과 함께 해당 scene이 UI의 graphicView에 적용되도록 하였습니다.

currentImage에는 QImage함수를 통해 코드 실행 시 생성되는 창을 저장할 공간(버퍼)를 이미지로서 할당하여 생성하도록 하였습니다.
이때, 초기 코드 실행시 해당 이미지를 모두 흰화면으로 채워 초기화를 하였으며 이를 초기상태로서 UNDO에 저장 및 update 함수 동작을 통해 표시하였습니다.

여기서 graphicView에서 사용자의 입력을 포함한 관여를 감지하고 이를 eventFilter의 필터함수를 동작하여 인식시키도록 하였습니다.

```
void MainWindow::mouse_condition(QMouseEvent *event)
{
    if(event->button() == Qt::LeftButton) {
        draw1 = 1;
        lastPoint = event->pos() - QPoint(15,25);
    }
    else if(event->button() == Qt::RightButton){
        draw2 = 1;
        lastPoint = event->pos() - QPoint(15,25);
    }
}

void MainWindow::mouse_move(QMouseEvent *event)
{
    if(draw1) {
        QPainter painter(&currentImage);
        painter.setPen(QPen(Qt::black, size, Qt::SolidLine, Qt::RoundCap));
        painter.drawLine((lastPoint), event->pos() - QPoint(15,25));
        lastPoint = event->pos() - QPoint(15,25);;
        update();
    }
    else if(draw2){
        QPainter painter(&currentImage);
        painter.setPen(QPen(Qt::white, size, Qt::SolidLine, Qt::RoundCap));
        painter.drawLine(lastPoint, event->pos() - QPoint(15,25));
        lastPoint = event->pos() - QPoint(15,25);
        update();
    }
}
```

마우스의 동작을 담당하는 멤버함수입니다.
첫번째 함수에서는 좌클릭, 우클릭을 각각 담당 인식하여 좌클릭일 경우 선그리기를 인식, 우클릭일 경우 지우기를 인식하도록 하였습니다. 선이 그려지는 좌표는 event->pos()를 통해 지정하였으나 이때 생성되는 창의 크기와 관련하여 정확히 포인터에 지정되지 않는 문제가 있었습니다.
이를 해결하기 위하여 pos()에 QPoint(15,25)를 가감하여 이를 보정할 수 있도록 하

였습니다.

두번째 함수입니다.
좌클릭, 우클릭의 감지에 따라 선그리기, 선 지우기를 구현하였습니다.
선그리기일 경우 Qpainter painter를 통해 현재 그려지는 이미지를 저장하도록 설정하며 이후 SetPen을 통한 브러쉬의 상태를 검은색, 슬라이드 값에 따라 바뀌는 사이즈, 선 모양, 선 끝을 지정하며 drawLine을 통해 가장 마지막의 마우스좌표부터 시작하여 마우스의 움직임에 따라 따라가며 선이 그려지도록 하였습니다.

선 지우기일 경우 배경이 흰화면인 점을 감안해 검은석이 아닌 흰선을 그리는 방식으로 결과적으로 윈도우 제공 그림판과 동일하게 지우개가 동작합니다.

```
void MainWindow::mouse_let(QMouseEvent *event)
{
    if(draw1 && event->button() == Qt::LeftButton) {
        draw1 = 0;
        UNDO();
        redo.clear();
    }
    else if(draw2 && event->button() == Qt::RightButton){
        draw2 = 0;
        UNDO();
        redo.clear();
    }
}
```

마우스의 버튼 입력이 해제됨을 인식하여 버튼이 입력되지 않은 경우 draw1, 2값을 초기화하며 그때의 이미지를 undo 함수 동작과 함께 저장합니다.
이때, redo를 리셋하여 현재의 그림이 가장 마지막에 작성된 것을 인지하도록 하였습니다.

```
void MainWindow::UNDO()
{
    undo.push_back(currentImage);
    if(undo.size() > 50)
        undo.erase(undo.begin());
}

void MainWindow::update()
{
    scene->clear();
    scene->addPixmap(QPixmap::fromImage(currentImage));
}

void MainWindow::on_pushButton_clicked()
{
    if(undo.size() > 1) {
        redo.push_back(undo.back());
        undo.pop_back();
        currentImage = undo.back();
        update();
    }
}

void MainWindow::on_pushButton_2_clicked()
{
    if(!redo.empty()) {
        undo.push_back(redo.back());
        currentImage = redo.back();
        redo.pop_back();
        update();
    }
}
```

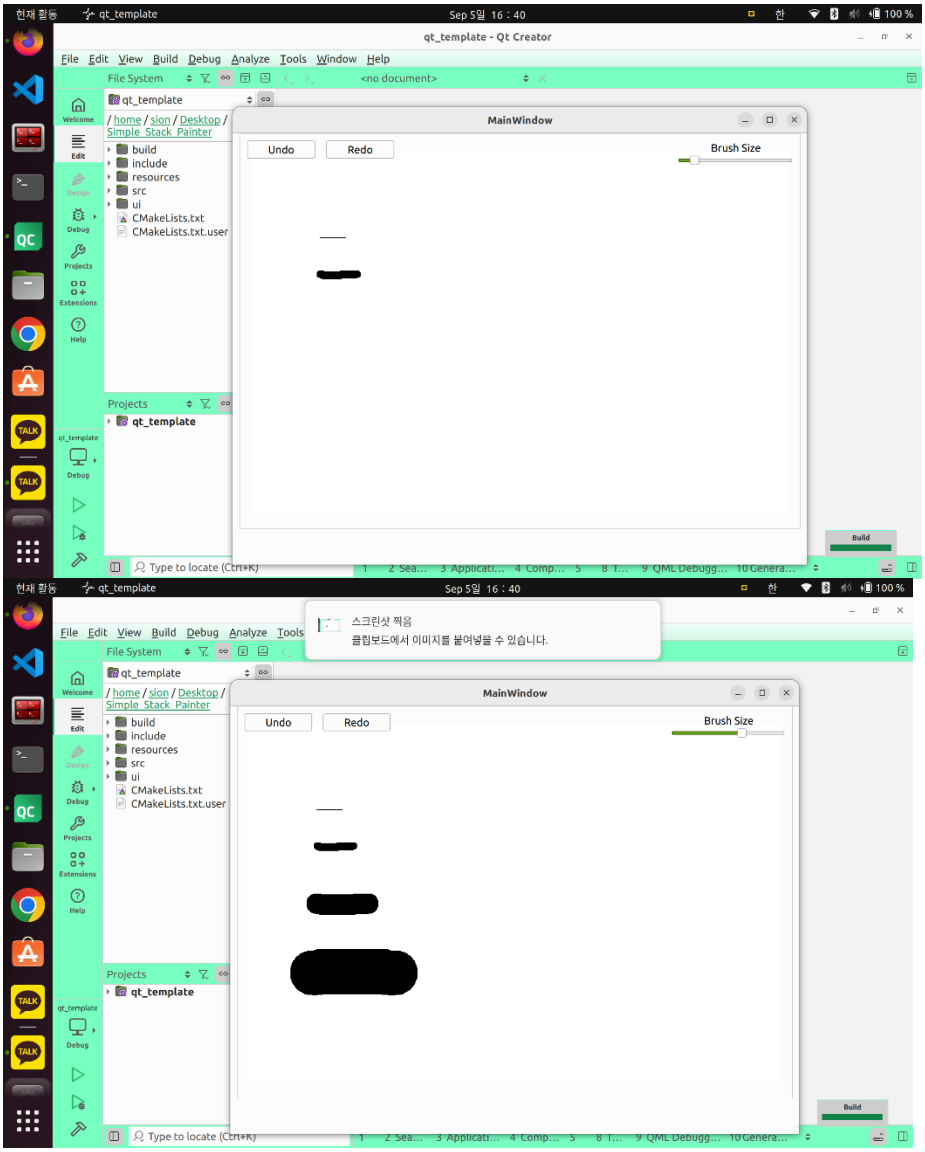
undo함수에서는 push_back을 통해 가장 마지막에 저장된 currentImage를 불러오도록하며 이때 undo에 저장되는 그림 목록을 50개로 제한하였습니다.
update함수에서는 현재의 Qimage를 화면에 반영하며 clear를 통해 이전 이미지를 제거할 수 있도록 하며,
scene->addPixmap(QPixmap::fromImage(currentImage))
위의 명령어를 통해 Qimage를 변환하여 scene에 추가되도록 하였습니다.
인터넷 참고 시 QPixmap으로 변환하여 렌더링을 진행하는 경우 그래픽카드 활용으로 인해 속도가 빨라진다는 점을 착안하였습니다.

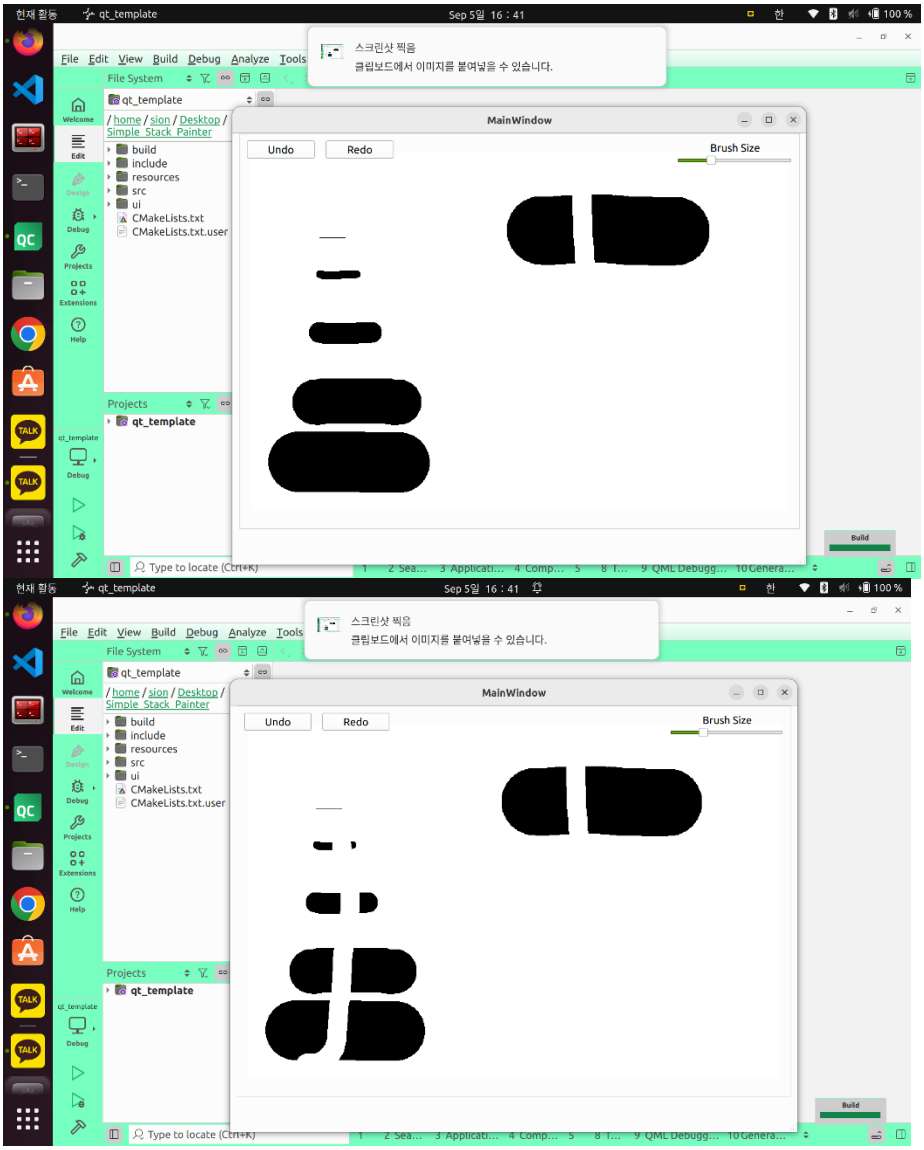
undo 버튼을 눌렀을때를 감지하는 함수
on_pushButton_clicked
해당 함수 동작시 스택을 pop하며 해당 pop값을 redo에 저장하여 2개의 스택이 이

중으로 상호작용을 이루도록 하였습니다.

redo를 담당하는 함수
on_pushButton_2_clicked
위 함수 동작시 redo 스택에 저장된 이미지를 불러옵니다.

3. 실행 결과





(사진의 단편적인 인식으로 인하여 정확한 보고의 미숙한 점 죄송합니다.)