Contents

Main.cpp

```
#include "BaseFrame.hpp"
#include <QtWidgets/QApplication>

int main(int argc, char *argv[])
{
	QApplication app(argc, argv);
	BaseFrame w;
	w.show();
	return app.exec();
}
```

BaseFrame.hpp

```
#pragma once

#include <QtWidgets/QMainWindow>
#include <vector>
#include <QTimer>

#include <QFile>
#include <QApplication>
#include <QDesktopWidget>

#include <QHBoxLayout>
#include <QMessageBox>
#include <QLabel>
#include <QTextStream>
#include <QIcon>

#include "ui_BaseFrame.h"
#include "Settings.hpp"
#include "Quote.hpp"

#include <QDebug>
using namespace std;

class BaseFrame : public QMainWindow
{
	Q_OBJECT

public:
	BaseFrame(QWidget *parent = Q_NULLPTR);
	~BaseFrame();
	void setPicture(int currentPic); //set main picture

protected:
	void mousePressEvent(QMouseEvent *event) override;
	void mouseMoveEvent(QMouseEvent *event) override;
	void mouseReleaseEvent(QMouseEvent *event) override;

private slots :
	void owo();
	void patMode();
	void pokeMode();
	void dragdropmode();
	void hidePic();
	void showPic();
	void setting();
	void what();
	void selfmove();
	void selftalk();
	void delQuote();
	void discon();

private:
	Ui::BaseFrameClass ui;
	void createActions();//create actions
	void updatedPic(int index); //update main picture after changing from settings
	void updateValue();// update values from settings
	//Actions can contain mouse hover event, any action from user,however, sometimes
not needed
	QActionGroup *cursorGroup;//group of cursors
	QAction *OwOAct;
	QAction *PatAct;
	QAction *PokeAct;
	QAction *DragDropAct;
	QAction *HideAct;
	QAction *ShowAct;
	QAction *setAct;
	QAction *exitAct;
	QAction *aboutAct;

	QTimer *walkAct; //normal timers, count from start time to 0
	QTimer *talkAct;
	int timeCounterS = 0; //used as substitue for timer as when program uses, the
timing is wrong
	bool quoteStat = 1; //Enable/disable quote
	bool appear = false; //shows quote or not

	QVBoxLayout *layout = new QVBoxLayout();//layout or a container for objects
	QLabel *image = new QLabel(this);
	QRect shape; //shape of image

	QPoint dragPos; //current dragged position
	QPoint displacementLeft = QPoint(5, 0); //character movement distance towards
right
	QPoint displacementRight = QPoint(-5, 0);
	QPoint topLeft, topRight, bottomLeft, bottomRight; //4 corners of the app
	QPoint centre; //center of picture
	QPoint screenCenter;

	int screenWidth;
	int screenHeight;

	int width, height; bool direction = 1; //width and height of the picture,
direction where the picture is moving
	int second = 10, minute, walkTime; //second = 10 miliseconds

	Quote *a;
	int talkTime, lifespan; //for quote

	vector<QString> pictureDirectory;
	vector<int> values;
	QPixmap pix; //another kind of image
};
```

BaseFrame.cpp

```
#include "BaseFrame.hpp"


BaseFrame::BaseFrame(QWidget *parent): QMainWindow(parent)

{

	ui.setupUi(this);



	updatedPic(0); //init main picture

	updateValue();//store value

	createActions();



	QDesktopWidget *desktop = QApplication::desktop();//get data for desktop sizes

	screenWidth = desktop->width();

	screenHeight = desktop->height();

	screenCenter = QPoint((screenWidth / 2) - (width / 2), (screenHeight / 2) -
(height / 2));//picture center to desktop center



	setWindowFlags(Qt::Widget | Qt::FramelessWindowHint); // no frame

	setParent(0); // Create TopLevel-Widget

	setAttribute(Qt::WA_NoSystemBackground, true); //do not draw background

	setAttribute(Qt::WA_TranslucentBackground, true); //invisible background

}


BaseFrame::~BaseFrame()

{

	//Auto delete by Qt

	delete cursorGroup;

	delete OwOAct;

	delete PatAct;
```

```
        delete PokeAct;

        delete DragDropAct;

        delete HideAct;

        delete ShowAct;

        delete setAct;

        delete exitAct;

        delete aboutAct;

        delete walkAct;

        delete layout;

        delete image;

}


void BaseFrame::setPicture(int cPic)

{

        QImage image(pictureDirectory[cPic]); //get picture from index of directories

        pix = QPixmap::fromImage(image);

}




#include <QMouseEvent>

void BaseFrame::mousePressEvent(QMouseEvent * event)

{

        if (event->button() == Qt::RightButton) {


                QMenu menu(this); //due to invisible frame, no menu will be seen and
context menu will contain the following:

                menu.addAction(OwOAct);

                menu.addAction(PatAct);

                menu.addAction(PokeAct);
```

```
              menu.addAction(DragDropAct);

              menu.addSeparator();

              menu.addAction(HideAct);

              menu.addAction(ShowAct);

              menu.addSeparator();

              menu.addAction(setAct);

              menu.addAction(aboutAct);

              menu.addSeparator();

              menu.addAction(exitAct);

              menu.exec(event->globalPos());//on the postion where user clicked

       }

       if (event->button() == Qt::LeftButton && cursor().shape() == Qt::WhatsThisCursor)
{

              updatedPic(1);

       }

       else if (event->button() == Qt::LeftButton && cursor().shape() ==
Qt::OpenHandCursor) {

              updatedPic(2);

       }

       else if (event->button() == Qt::LeftButton && cursor().shape() ==
Qt::PointingHandCursor) {

              updatedPic(3);

       }

       else  if (event->button() == Qt::LeftButton && cursor().shape() ==
Qt::ArrowCursor) {

              dragPos = event->globalPos() - frameGeometry().topLeft();

              event->accept();

       }

}
```

```
void BaseFrame::mouseMoveEvent(QMouseEvent * event)

{

	if (event->buttons() & Qt::LeftButton && cursor().shape() == Qt::ArrowCursor) {

		topLeft = this->mapToGlobal(QPoint(0, 0));

		topRight = this->mapToGlobal(QPoint(width, 0));

		bottomLeft = this->mapToGlobal(QPoint(0, height));

		if (topLeft.x() >= 0 && topRight.x() <= screenWidth && topLeft.y() >= 0 &&
bottomLeft.y() <= screenHeight) {

			move(event->globalPos() - dragPos); //if within the screen, move the
picture

			event->accept();

		}

		else

			event->ignore();

	}

}


void BaseFrame::mouseReleaseEvent(QMouseEvent * event)

{

	if (event->button() == Qt::LeftButton && cursor().shape() == Qt::WhatsThisCursor)
{

		updatedPic(0);

	}

	else if (event->button() == Qt::LeftButton && cursor().shape() ==
Qt::OpenHandCursor) {

		updatedPic(0);

	}

	else if (event->button() == Qt::LeftButton && cursor().shape() ==
Qt::PointingHandCursor) {

		updatedPic(0);

	}
```

```
        else if (event->button() == Qt::LeftButton && cursor().shape() == Qt::ArrowCursor)
{
                topLeft = this->mapToGlobal(QPoint(0, 0)); //current position of the app's
window
                topRight = this->mapToGlobal(QPoint(width, 0));
                bottomLeft = this->mapToGlobal(QPoint(0, height));
                //QPoint a = dragPos;
                if(topLeft.x() < 0 ){//left
                        if (topLeft.y() < 0) {//top corner
                                move(QPoint(0, 0));
                                event->accept();
                        }
                        else if (bottomLeft.y() > screenHeight) {//down corner
                                move(QPoint(0, screenHeight - height));
                                event->accept();
                        }
                        else {//middle
                                move(QPoint(0, event->globalPos().y() - dragPos.y()));
                                event->accept();
                        }
                }
                else if (topRight.x() > screenWidth) {//right
                        if (topLeft.y() < 0) {//top corner
                                move(QPoint(screenWidth-width, 0));
                                event->accept();
                        }
                        else if (bottomLeft.y() > screenHeight) {//down corner
                                move(QPoint(screenWidth-width, screenHeight - height));
                        event->accept();
                    }
                    else {//middle
                        move(QPoint(screenWidth - width, event->globalPos().y() -
dragPos.y()));
                        event->accept();
                    }
                }
                else if (topLeft.y() < 0) {//top center
                    move(QPoint(event->globalPos().x() - dragPos.x(),0));
                    event->accept();
                }
                else if (bottomLeft.y() > screenHeight) {//down center
                    move(QPoint(event->globalPos().x() - dragPos.x(), screenHeight-
height));
                    event->accept();
                }
        }
}

void BaseFrame::owo() {
        OwOAct->setChecked(true);
        setCursor(Qt::WhatsThisCursor); //set cursor to the shape
}

void BaseFrame::patMode()
{
        PatAct->setChecked(true);
        setCursor(Qt::OpenHandCursor);
```

```
}

void BaseFrame::pokeMode()
{
	PokeAct->setChecked(true);
	setCursor(Qt::PointingHandCursor);
}

void BaseFrame::dragdropmode()
{
	DragDropAct->setChecked(true);
	setCursor(Qt::ArrowCursor);
}

void BaseFrame::hidePic()
{
	this->setWindowOpacity(0.05); //opacity or how clear it is
}

void BaseFrame::showPic()
{
	this->setWindowOpacity(1); // 100 not translucent
}

void BaseFrame::setting()
{
	Settings a;
	a.setModal(true); //cannot touch the main frame/character while the settings
window is open
```

```
	a.exec();//execute

	updatedPic(0); //update everything even if user didn't change any thing

	updateValue();

}


void BaseFrame::what()

{

	QMessageBox::about(this, tr("About this app"),

		tr("The app is about you wasting time with the character.\n Have fun."));

}


void BaseFrame::createActions()

{

	QFont font("Arial", 8); //init font

	OwOAct = new QAction(tr("&OwO??"), this);

	OwOAct->setFont(font);

	OwOAct->setStatusTip(tr("OwO???")); // tip/help is at the bottom of the character
towards the left

	connect(OwOAct, &QAction::triggered, this, &BaseFrame::owo); //connect the action,
when clicked/activated, it will call f(x) owo


	PatAct = new QAction(tr("&Pat Mode"), this);

	PatAct->setFont(font);

	PatAct->setStatusTip(tr("It's time to pat!"));

	connect(PatAct, &QAction::triggered, this, &BaseFrame::patMode);


	PokeAct = new QAction(tr("&Poke Mode"), this);

	PokeAct->setFont(font);

	PokeAct->setStatusTip(tr("Poke death"));
```

```
connect(PokeAct, &QAction::triggered, this, &BaseFrame::pokeMode);

DragDropAct = new QAction(tr("&User move mode"));
DragDropAct->setFont(font);
connect(DragDropAct, &QAction::triggered, this, &BaseFrame::dragdropmode);

HideAct = new QAction(tr("&Hide"));
HideAct->setFont(font);
connect(HideAct, &QAction::triggered, this, &BaseFrame::hidePic);

ShowAct = new QAction(tr("&Show"));
ShowAct->setFont(font);
connect(ShowAct, &QAction::triggered, this, &BaseFrame::showPic);

cursorGroup = new QActionGroup(this); //one of the modes can be on, not all
cursorGroup->addAction(OwOAct);
cursorGroup->addAction(PatAct);
cursorGroup->addAction(PokeAct);
cursorGroup->addAction(DragDropAct);
OwOAct->setChecked(true);

walkAct = new QTimer(this);
connect(walkAct, SIGNAL(timeout()), this, SLOT(selfmove()));
walkAct->start(walkTime); //start timer

talkAct = new QTimer(this);
connect(talkAct, SIGNAL(timeout()), this, SLOT(selftalk()));
talkAct->start(talkTime); //quote init
```

```
	setAct = new QAction(tr("&Settings"), this);

	setAct->setFont(font);

	setAct->setStatusTip(tr("It's the settings."));

	connect(setAct, &QAction::triggered, this, &BaseFrame::setting); //open settings
panel


	exitAct = new QAction(tr("&Exit"), this);

	exitAct->setFont(font);

	exitAct->setShortcuts(QKeySequence::Quit);

	exitAct->setStatusTip(tr("Exit the application"));

	connect(exitAct, &QAction::triggered, this, &BaseFrame::discon); //close other
functions than character

	connect(exitAct, &QAction::triggered, this, &QWidget::close); //close app


	aboutAct = new QAction(tr("&About"), this);

	aboutAct->setFont(font);

	aboutAct->setStatusTip(tr("What is this app about?"));

	connect(aboutAct, &QAction::triggered, this, &BaseFrame::what);

}



void BaseFrame::selfmove()

{

	topLeft = this->mapToGlobal(QPoint(0, 0));

	topRight = this->mapToGlobal(QPoint(width, 0));

	bottomLeft = this->mapToGlobal(QPoint(0, height));

	bottomRight = this->mapToGlobal(QPoint(width, height));

	centre = this->mapToGlobal(QPoint(width/2, height/2));
```

```
        if (topLeft.x() >= 0 && topRight.x() <= screenWidth && topLeft.y() >= 0 &&
bottomLeft.y() <= screenHeight) {


                     if (direction == 1) {//towards right

                            if (topRight.x() + displacementLeft.x() < screenWidth)
{//within screen width

                                   move(topLeft + displacementLeft);

                                   topLeft = this->mapToGlobal(QPoint(0, 0));//update
current position of the app from top left

                                   topRight = this->mapToGlobal(QPoint(width, 0));

                            }

                            else if (topRight.x() + displacementLeft.x() >= screenWidth)
{//more than screenwidth

                                   direction = 0;//change direction to left

                                   move(topLeft + displacementRight);

                                   topLeft = this->mapToGlobal(QPoint(0, 0));//update
current position of the app from top left

                                   topRight = this->mapToGlobal(QPoint(width, 0));

                            }

                            else {

                                   move(screenCenter);//if at center of the screen

                                   topLeft = this->mapToGlobal(QPoint(0, 0));//update
current position of the app from top left

                                   topRight = this->mapToGlobal(QPoint(width, 0));

                            }

                     }


                     if(direction == 0) {//towards left

                            if (topLeft.x() + displacementRight.x() > 0) {

                                   move(topLeft + displacementRight);
```

```
                                    topLeft = this->mapToGlobal(QPoint(0, 0));//update
current position of the app from top left

                                    topRight = this->mapToGlobal(QPoint(width, 0));

                              }

                              else if(topLeft.x() + displacementRight.x() <= 0){

                                    direction = 1; //change direction to right

                                    move(topLeft + displacementLeft);

                                    topLeft = this->mapToGlobal(QPoint(0, 0));//update
current position of the app from top left

                                    topRight = this->mapToGlobal(QPoint(width, 0));

                              }

                              else {

                                    move(screenCenter);//if at center

                                    topLeft = this->mapToGlobal(QPoint(0, 0));//update
current position of the app from top left

                                    topRight = this->mapToGlobal(QPoint(width, 0));;

                              }

                        }

      }

      else {//when user successfully dragged out of screen

            move(screenCenter);

            topLeft = this->mapToGlobal(QPoint(0, 0));//update current position of the
app from top left

            topRight = this->mapToGlobal(QPoint(width, 0));

            QMessageBox::information(this, tr("Out of screen"), tr("Please be a bit
more considerate."));

      }

      update();



      if (quoteStat == 1) {//quote is enabled

            if (appear == true) {
                    if (centre.x() < screenWidth / 2 && centre.y() < screenHeight / 2) {
                           a->move(topRight);
                    }
                    else if (centre.x() > screenWidth / 2 && centre.y() < screenHeight /
2) {
                           a->move(topLeft - QPoint(a->width(), 0));
                    }
                    else if (centre.x() < screenWidth / 2 && centre.y() > screenHeight /
2) {
                           a->move(topRight);
                    }
                    else if (centre.x() > screenWidth / 2 && centre.y() > screenHeight /
2) {
                           a->move(topLeft - QPoint(a->width(), 0));
                    }
                    else if (centre == screenCenter) {
                           a->move(topRight);
                    }
                    else {//when out of screen
                           if (centre.x() < screenWidth / 2) {
                                  a->move(topRight);
                           }
                           else if (centre.x() > screenWidth / 2 ) {
                                  a->move(topLeft - QPoint(a->width(), 0));
                           }
                    }
             }
      }

      else {//quote is disabled
```

```
		if (appear == true) {//if disabled while running as before was enabled
			delete a;
			appear = false;
		}
	}

	update();
}
void BaseFrame::selftalk()
{
	if (quoteStat == 1) {
		if (appear == false) {
			a = new Quote; /init quote
			if (centre.x() < screenWidth / 2 && centre.y() < screenHeight / 2) {
				a->move(topRight);
			}
			else if (centre.x() > screenWidth / 2 && centre.y() < screenHeight / 
2) {
				a->move(topLeft - QPoint(a->width(), 0));
			}
			else if (centre.x() < screenWidth / 2 && centre.y() > screenHeight / 
2) {
				a->move(topRight);
			}
			else if (centre.x() > screenWidth / 2 && centre.y() > screenHeight / 
2) {
				a->move(topLeft - QPoint(a->width(), 0));
			}
			else {
				a->move(QPoint(screenWidth / 2, 0));
```

```
                    }

                    a->show();

                    appear = true;

                    QTimer::singleShot(lifespan, this, SLOT(delQuote()));//to be called
once per quote init then call F(x) in slot

            }

    }

    update();

}


void BaseFrame::delQuote()

{

    if (quoteStat == 1) {

            if (appear == true) {

                    delete a;

                    appear = false;

            }

    }

    update();

}


void BaseFrame::discon()

{

    QCoreApplication::exit();

}



void BaseFrame::updatedPic(int i)

{
    if (pictureDirectory.size() == 0) {
            QFile inputFile("Resources/picDirectory.txt");
            if (inputFile.open(QIODevice::ReadOnly)) //if able to open, readonly = 
delete previous content
            {
                    QTextStream in(&inputFile);
                    while (!in.atEnd())
                    {
                            QString line = in.readLine();
                            pictureDirectory.push_back(line);
                    }
                    inputFile.close();
            }

            setPicture(i);
            image->setPixmap(pix);
            layout->addWidget(image);

            shape.setSize(pix.size());
            layout->setGeometry(shape);
            layout->setAlignment(this, Qt::AlignBottom);
            this->setMaximumSize(pix.size());
            this->setMinimumSize(pix.size());
            setLayout(layout);
            width = pix.size().width();
            height = pix.size().height();
            update();
    }
    else if((pictureDirectory.size() != 0)) {
```

```
		pictureDirectory.clear();

		QFile inputFile("Resources/picDirectory.txt");

		if (inputFile.open(QIODevice::ReadOnly))

		{

			QTextStream in(&inputFile);


			while (!in.atEnd())

			{

				QString line = in.readLine();

				pictureDirectory.push_back(line);


			}

			inputFile.close();

		}

		setPicture(i);

		image->setPixmap(pix);

		shape.setSize(pix.size());

		layout->setGeometry(shape);

		layout->setAlignment(this, Qt::AlignBottom);

		this->setMaximumSize(pix.size());

		this->setMinimumSize(pix.size());

		setLayout(layout);

		width = pix.size().width();

		height = pix.size().height();

		layout->update(); //instead of deleting and adding, just replace by
updating as variable is the same (image)

		screenCenter = QPoint((screenWidth / 2) - (width / 2), (screenHeight / 2) -
(height / 2)); //when picture changes, center changes

		update();
```

```
	}
}

void BaseFrame::updateValue()
{
	if (values.size() == 0) {
		QFile inputFile("Resources/values.txt");
		if (inputFile.open(QIODevice::ReadOnly))
		{
			QTextStream in(&inputFile);
			while (!in.atEnd())
			{
				QString line = in.readLine();
				values.push_back(line.toInt());
			}
			inputFile.close();
		}
		minute = values[0];
		displacementLeft = QPoint(values[1], 0);
		displacementRight = QPoint(-values[1], 0);
		walkTime = minute * second;
		talkTime = values[2] * 1000;
		lifespan = values[3] * 1000;
		quoteStat = values[4];
	}
	else {
		values.clear();
		QFile inputFile("Resources/values.txt");
```

```
        if (inputFile.open(QIODevice::ReadOnly))
        {
            QTextStream in(&inputFile);
            while (!in.atEnd())
            {
                QString line = in.readLine();
                values.push_back(line.toInt());
            }
            inputFile.close();
        }
        minute = values[0];
        displacementLeft = QPoint(values[1],0);
        displacementRight = QPoint(-values[1], 0);
        walkTime = minute * second;
        talkTime = values[2] * 1000;
        lifespan = values[3] * 1000;
        quoteStat = values[4];
        walkAct->start(walkTime);//reset the time and start a new one
        update();
    }
}
```

Settings.hpp

```
#include <QtWidgets/QMainWindow>
#include <vector>
#include <QFileDialog>
#include <QTextStream>
#include <QMessageBox>
#include <QSignalMapper>

#include "ui_Settings.h"
#include "SetTime.hpp"
#include "SetDistance.hpp"
#include "SetApTimeQ.hpp"
#include "SetLifespanQ.hpp"
#include "ModQuotes.hpp"
#include "QuoteED.hpp"
using namespace std;

class Settings : public QDialog {
	Q_OBJECT

public:
	Settings(QWidget *parent = Q_NULLPTR);
	~Settings();

private slots:
	void browse(int a); //which browse button did user press
	void change();//set picture(s) for change
	void setTime();//GUIs
	void setDistance();
	void setInterQ();
	void setLifespan();
	void modQuote();
	void quoteED();
	void movpic();

private:
	Ui::SettingsUI ui;
	void createActions();
	bool fit = false; //size of picture
	int pressed = 0, owoPress = 0, patPress = 0, pokePress = 0; //how many times user
has pressed the buttons

	QAction *browseAct;
	QAction *changeAct;

	QAction *setTimeAct;
	QAction *setDistanceAct;
	QSignalMapper* signalMapper; //Map to assign value

	QAction *setIntQAct;
	QAction *setLifeAct;
	QAction *modQuoteAct;

	vector<QString> pictureDirectory; //loaded directories from file
	QString picDirectory; //uploaded picture directory
};
```

Settings.cpp

```
#include "Settings.hpp"

Settings::Settings(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);
	createActions();
	ui.lineEdit->setDisabled(true); //user unable to write/type here
	QFile inputFile("Resources/picDirectory.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);

		while (!in.atEnd())
		{
			QString line = in.readLine();
			pictureDirectory.push_back(line);
		}
		inputFile.close();
	}
	QImage idle(pictureDirectory[0]);
	ui.Piclabel->setPixmap(QPixmap::fromImage(idle)); //set picture on settings
	ui.lineEdit->setText(pictureDirectory[0]); //picture's directory

	QImage owo(pictureDirectory[1]);
	ui.owoImage->setPixmap(QPixmap::fromImage(owo));

	QImage pat(pictureDirectory[2]);
	ui.patImage->setPixmap(QPixmap::fromImage(pat));

	QImage poke(pictureDirectory[3]);
	ui.pokeImage->setPixmap(QPixmap::fromImage(poke));
}

void Settings::browse(int direct) {

	QFileDialog dialog(this);
	dialog.setNameFilter(tr("Images (*.png *.xpm *.jpg)"));
	dialog.setViewMode(QFileDialog::Detail);
	picDirectory = QFileDialog::getOpenFileName(this,
		tr("Open Images"), QString(), tr("Image Files (*.png *.jpg *.bmp)"));

	if (!picDirectory.isEmpty())
	{
		QImage image(picDirectory); //check size
		bool width = image.size().width() <= 300 && image.size().width() >= 150;
		bool height = image.size().height() <= 300 && image.size().height() >= 150;

		if (width && height) {//if fit conditions
			QImage unscaled(picDirectory);
			if (direct == 0) {
				QImage image = unscaled.scaled(300, 300, Qt::KeepAspectRatio);
				ui.Piclabel->setPixmap(QPixmap::fromImage(image));
				ui.lineEdit->setText(picDirectory);
				pictureDirectory[0] = picDirectory;
```

```
                                fit = true;
                                pressed += 1;
                        }
                        else if (direct == 1) {
                                QImage image = unscaled.scaled(200, 200, Qt::KeepAspectRatio);
                                ui.owoImage->setPixmap(QPixmap::fromImage(image));
                                pictureDirectory[1] = picDirectory;
                                fit = true;
                                owoPress += 1;
                        }
                        else if (direct == 2) {
                                QImage image = unscaled.scaled(200, 200, Qt::KeepAspectRatio);
                                ui.patImage->setPixmap(QPixmap::fromImage(image));
                                pictureDirectory[2] = picDirectory;
                                fit = true;
                                patPress += 1;
                        }
                        else if (direct == 3) {
                                QImage image = unscaled.scaled(200, 200, Qt::KeepAspectRatio);
                                ui.pokeImage->setPixmap(QPixmap::fromImage(image));
                                pictureDirectory[3] = picDirectory;
                                fit = true;
                                pokePress += 1;
                        }
                }
                else {
                        QMessageBox::warning(this, tr("Error!"), tr("Image size unfit!"));
                        fit = false;
                }
        }
}

Settings::~Settings()
{
        delete browseAct;
        delete changeAct;
        delete setTimeAct;
        delete setDistanceAct;
        delete setIntQAct;
        delete setLifeAct;
}

void Settings::change()
{
        if (fit == false && (pressed > 0 || owoPress > 0 || patPress > 0 || pokePress >
0)) {
                QMessageBox::warning(this, tr("Error!"), tr("Go and recheck your image
size."));
                return;
        }
        else {

                QFile outputFile("Resources/picDirectory.txt");
                if (outputFile.open(QIODevice::WriteOnly | QIODevice::Text))
                {
                        QTextStream out(&outputFile);
                        for (int i = 0; i < pictureDirectory.size(); i++) {
                                out << pictureDirectory[i] << endl;
```

```
                    }
                    outputFile.close();
                }
                QMessageBox::information(this, tr("Message"), tr("Operation successful."));
        }
        accept();
}

void Settings::setTime()
{
        SetTime b;
        b.setModal(true);
        b.exec();
}

void Settings::setDistance()
{
        SetDistance c;
        c.setModal(true);
        c.exec();
}

void Settings::setInterQ()
{
        SetApTimeQ d;
        d.setModal(true);
        d.exec();
}

void Settings::setLifespan()
{
        SetLifespanQ e;
        e.setModal(true);
        e.exec();
}

void Settings::modQuote()
{
        ModQuotes f;
        f.setModal(true);
        f.exec();
}

void Settings::quoteED()
{
        QuoteED i;
        i.setModal(true);
        i.exec();
}

void Settings::movpic()//pls wait for next update
{
        QMessageBox::information(this, tr("Sorry"), tr("Please wait for next patch.\nSorry
for your inconvenience."));
}

void Settings::createActions()
{
        QSignalMapper* signalMapper = new QSignalMapper(this);
        //same browse but for different index as each browse changes different picture's
directory
        browseAct = new QAction(tr("&Browse"), this);
        browseAct->setStatusTip(tr("Browse for picture"));
        connect(ui.BrowsePic, SIGNAL(clicked()), signalMapper, SLOT(map()));
        connect(ui.owoBrowse, SIGNAL(clicked()), signalMapper, SLOT(map()));
        connect(ui.patBrowse, SIGNAL(clicked()), signalMapper, SLOT(map()));
        connect(ui.pokeBrowse, SIGNAL(clicked()), signalMapper, SLOT(map()));
        signalMapper->setMapping(ui.BrowsePic, 0);
        signalMapper->setMapping(ui.owoBrowse, 1);
        signalMapper->setMapping(ui.patBrowse, 2);
        signalMapper->setMapping(ui.pokeBrowse, 3);
        connect(signalMapper, SIGNAL(mapped(int)), this, SLOT(browse(int)));

        changeAct = new QAction(tr("&Change"), this);
        changeAct->setStatusTip(tr("Change the picture"));
        connect(ui.ConfirmChanges, SIGNAL(clicked()), this, SLOT(change()));

        setTimeAct = new QAction(tr("&Set Time"), this);
        setTimeAct->setStatusTip(tr("Set the time for movement."));
        connect(ui.setIntervalB, SIGNAL(clicked()), this, SLOT(setTime()));

        setDistanceAct = new QAction(tr("&Set Distance"), this);
        setDistanceAct->setStatusTip(tr("Set the distance for movement."));
        connect(ui.setDistanceB, SIGNAL(clicked()), this, SLOT(setDistance()));

        setIntQAct = new QAction(tr("&Set Appearance Interval"), this);
        setIntQAct->setStatusTip(tr("Set quote spawn interval time."));
        connect(ui.setApTimeQ, SIGNAL(clicked()), this, SLOT(setInterQ()));

        setLifeAct = new QAction(tr("&Set lifespan"), this);
        setLifeAct->setStatusTip(tr("Set quote's lifespan."));
        connect(ui.setLifespanQ, SIGNAL(clicked()), this, SLOT(setLifespan()));

        modQuoteAct = new QAction(tr("&Modify quote"), this);
        modQuoteAct->setStatusTip(tr("Modify quote(s)."));
        connect(ui.setQuotes, SIGNAL(clicked()), this, SLOT(modQuote()));

        connect(ui.QuoteEnDis, SIGNAL(clicked()), this, SLOT(quoteED()));

        connect(ui.mpBrowse, SIGNAL(clicked()), this, SLOT(movpic()));
}
```

Quote.hpp

```
#pragma once
#include <QtWidgets/QMainWindow>
#include <QFile>
#include <QTextStream>
#include <QDesktopWidget>
#include <QTime>
#include <QFont>
#include <QMessageBox>
#include "ui_Quote.h"
using namespace std;

class Quote : public QDialog {
	Q_OBJECT

public:
	Quote(QWidget *parent = Q_NULLPTR);
	~Quote();

private:
	Ui::QuoteLog ui;

	int fontSize;
	int sizeQ;
	int randomValue;
	int randInt(int l, int h);
	vector<QString> quotes;
	vector<int> values;

};
```

Quote.cpp

```
#include "Quote.hpp"

Quote::Quote(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);

	QDesktopWidget *desktop = QApplication::desktop();
	int screenWidth = desktop->width();

	if (values.size() != 0 && quotes.size() != 0) {
		values.clear();
		quotes.clear();
	}

	QFile inputFile("Resources/quotes.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);
		while (!in.atEnd())
		{
			QString line = in.readLine();
			quotes.push_back(line);
		}
		inputFile.close();
```

```
        }

        QFile iputFile("Resources/values.txt");
        if (iputFile.open(QIODevice::ReadOnly))
        {
                QTextStream in(&iputFile);
                while (!in.atEnd())
                {
                        QString line = in.readLine();
                        values.push_back(line.toInt());
                }
                iputFile.close();
        }

        QTime time = QTime::currentTime();//desktop time for random
        qsrand((uint)time.msec()); // seed

        randomValue = randInt(0, quotes.size()-1);
        sizeQ = quotes[randomValue].size();//number of characters of quote
        if (sizeQ <= 15)
                fontSize = 20;
        else if (sizeQ <= 40)
                fontSize = 15;
        else if (sizeQ <= 100)
                fontSize = 12;
        QFont quoteFont("Arial", fontSize);
        ui.text->setFont(quoteFont);
        ui.text->setText(quotes[randomValue]);

        setWindowFlags(Qt::Widget | Qt::FramelessWindowHint);
        setAttribute(Qt::WA_NoSystemBackground, true);
        setAttribute(Qt::WA_TranslucentBackground, true);
}

Quote::~Quote()
{
        //QMessageBox::information(this, tr("Message"), tr("Operation successful."));
}

int Quote::randInt(int low, int high)
{
        return qrand() % ((high + 1) - low) + low;
}
```

ModQuotes.hpp

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_ModQuotes.h"
#include <QLineEdit>
#include <QSpinBox>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QMessageBox>
using namespace std;

class ModQuotes : public QDialog {
	Q_OBJECT
public:
	ModQuotes(QWidget *parent = Q_NULLPTR);

private slots:
	void updateQuote();
	void modQuote();//set quote to lineEdit
	void addQuote();
	void delQuote();
	void save();

private:
	Ui::ModQuotes ui;
	void createActions();

	int currentSize; //number of quotes
	QAction *saveAct;
	vector<QString> quotes;

};
```

ModQuotes.cpp

```
#include "ModQuotes.hpp"

ModQuotes::ModQuotes(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);
	createActions();

	QFile inputFile("Resources/quotes.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);
		while (!in.atEnd())
		{
			QString line = in.readLine();
			quotes.push_back(line);
		}
		inputFile.close();
		ui.lineEdit->setText(quotes[0]);
	}
	ui.spinBox->setMaximum(quotes.size());
	currentSize = quotes.size();
}

void ModQuotes::updateQuote() {
	ui.lineEdit->setText(quotes[ui.spinBox->value()-1]);
}

void ModQuotes::modQuote()
{
	quotes[ui.spinBox->value()-1] = ui.lineEdit->text();
}

void ModQuotes::addQuote()
{
	ui.spinBox->setMaximum(currentSize+1);
	quotes.push_back("");
	currentSize += 1;
	QMessageBox::information(this, tr("Sucessful"), tr("Added a new space for a 
quote.\nIt is located at the last number."));
}

void ModQuotes::delQuote()
{
	if (currentSize > 1) {
		quotes.erase(quotes.begin() + (ui.spinBox->value() - 1));
		ui.spinBox->setMaximum(currentSize - 1);
		currentSize -= 1;
		ui.lineEdit->setText(quotes[ui.spinBox->value() - 1]);
		QMessageBox::information(this, tr("Sucessful"), tr("Deleted the current 
quote. \n It is possible to recover by clicking cancel button."));
	}
	else {
		QMessageBox::warning(this, tr("Error!"), tr("There must be at least one 
quote!"));
	}
}
```

```
void ModQuotes::save()
{
	QFile outputFile("Resources/quotes.txt");
	if (outputFile.open(QIODevice::WriteOnly | QIODevice::Text))
	{
		QTextStream out(&outputFile);
		{
			QTextStream out(&outputFile);
			for (int i = 0; i < quotes.size(); i++) {
				out << quotes[i] << endl;
			}
			outputFile.close();
		}
		QMessageBox::information(this, tr("Message"), tr("Operation successful."));
	}
	accept();
}

void ModQuotes::createActions()
{
	connect(ui.spinBox, SIGNAL(valueChanged(int)), this, SLOT(updateQuote()));
	connect(ui.lineEdit, SIGNAL(textEdited(const QString&)), this, SLOT(modQuote()));

	connect(ui.addQte, SIGNAL(clicked()), this, SLOT(addQuote()));
	connect(ui.delQte, SIGNAL(clicked()), this, SLOT(delQuote()));

	saveAct = new QAction(tr("&Save"), this);
	connect(ui.okButton, SIGNAL(clicked()), this, SLOT(save()));
}
```

QuoteED.hpp

```cpp
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_QuoteED.h"
#include <QLineEdit>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QMessageBox>
using namespace std;

class QuoteED : public QDialog {
	Q_OBJECT

public:
	QuoteED(QWidget *parent = Q_NULLPTR);

private slots:
	void enable();
	void disable();
	void save();

private:
	Ui::QuoteED ui;
	void createActions();

	int stat; //status if enable or disable
	vector<int> values;
};
```

QuoteED.cpp

```cpp
#include "QuoteED.hpp"

QuoteED::QuoteED(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);
	createActions();
	ui.lineEdit->setDisabled(true);

	QFile inputFile("Resources/values.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);
		while (!in.atEnd())
		{
			QString line = in.readLine();
			values.push_back(line.toInt());
		}
		inputFile.close();
	}
	stat = values[4];

	if (values[4] == 1)
		ui.lineEdit->setText("Enabled");
```

```
        else
                ui.lineEdit->setText("Disabled");
}

void QuoteED::enable()
{
        stat = 1;
        ui.lineEdit->setText("Enabled");
}

void QuoteED::disable()
{
        stat = 0;
        ui.lineEdit->setText("Disabled");
}

void QuoteED::save()
{
        values[4] = stat; //index 4 is reserved for quote enable/disable
        QFile outputFile("Resources/values.txt");
        if (outputFile.open(QIODevice::WriteOnly | QIODevice::Text))
        {
                QTextStream out(&outputFile);
                {
                        QTextStream out(&outputFile);
                        for (int i = 0; i < values.size(); i++) {
                                out << values[i] << endl;
                        }
                        outputFile.close();
                }
                QMessageBox::information(this, tr("Message"), tr("Operation successful."));
        }
        accept();
}

void QuoteED::createActions()
{
        connect(ui.enable, SIGNAL(clicked()), this, SLOT(enable()));
        connect(ui.disable, SIGNAL(clicked()), this, SLOT(disable()));

        connect(ui.okButton, SIGNAL(clicked()), this, SLOT(save()));
                                                   }
```

SetApTimeQ.hpp

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_SetApTimeQ.h"
#include <QLineEdit>
#include <QSpinBox>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QMessageBox>
using namespace std;

class SetApTimeQ : public QDialog {
	Q_OBJECT
public:
	SetApTimeQ(QWidget *parent = Q_NULLPTR);

private slots:
	void setTime();//for spinbox
	void save();

private:
	Ui::SetApTimeQ ui;
	void createActions();

	QAction *saveAct;

	int time = 60;
	QString t;
	vector<int> values;
};
```

SetApTimeQ.cpp

```
#include "SetApTimeQ.hpp"

SetApTimeQ::SetApTimeQ(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);
	createActions();
	ui.lineEdit->setDisabled(true);

	QFile inputFile("Resources/values.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);
		while (!in.atEnd())
		{
			QString line = in.readLine();
			values.push_back(line.toInt());
		}
		inputFile.close();
		ui.lineEdit->setText(QString::number(values[2]));
	}
}

void SetApTimeQ::setTime() {
        time = ui.spinBox->value();
        t = QString::number(time);
        ui.lineEdit->setText(t);
}

void SetApTimeQ::save() {
        values[2] = time; //index 2 is reserved for quote appearing interval
        QFile outputFile("Resources/values.txt");
        if (outputFile.open(QIODevice::WriteOnly | QIODevice::Text))
        {
                QTextStream out(&outputFile);
                {
                        QTextStream out(&outputFile);
                        for (int i = 0; i < values.size(); i++) {
                                out << values[i] << endl;
                        }
                        outputFile.close();
                }
                QMessageBox::information(this, tr("Message"), tr("Operation successful."));
        }
        accept();
}

void SetApTimeQ::createActions()
{
        connect(ui.spinBox, SIGNAL(valueChanged(int)), this, SLOT(setTime()));

        saveAct = new QAction(tr("&Save"), this);
        connect(ui.okButton, SIGNAL(clicked()), this, SLOT(save()));
}
```

SetDistance.hpp

```
#include <QtWidgets/QMainWindow>
#include "ui_SetDistance.h"
#include <QLineEdit>
#include <QSpinBox>
#include <QSignalMapper>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QMessageBox>
using namespace std;

class SetDistance : public QDialog {
	Q_OBJECT
public:
	SetDistance(QWidget *parent = Q_NULLPTR);
	~SetDistance();

private slots:
	void setDistance(int a); //buttons
	void setSBox();//spinbox
	void save();

private:
	Ui::SetDistance ui;
	void createActions();

	QSignalMapper* signalMapper;
	QAction *saveAct;

	int distance = 5;
	QString d;
	vector<int> values;
};
```

SetDistance.cpp

```
#include "SetDistance.hpp"

SetDistance::SetDistance(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);
	createActions();
	ui.outputShow->setDisabled(true);

	QFile inputFile("Resources/values.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);
		while (!in.atEnd())
		{
			QString line = in.readLine();
			values.push_back(line.toInt());
		}
		inputFile.close();
		ui.outputShow->setText(QString::number(values[1]));
	}
}

void SetDistance::setDistance(int dist)
{
        distance = dist;
        d = QString::number(distance);
        ui.outputShow->setText(d);
}

SetDistance::~SetDistance()
{
        delete saveAct;
}

void SetDistance::setSBox()
{
        distance = ui.spinBox->value();
        d = QString::number(distance);
        ui.outputShow->setText(d);
}

void SetDistance::save() {
        values[1] = distance; //index 1 is reserved for time interval
        QFile outputFile("Resources/values.txt");
        if (outputFile.open(QIODevice::WriteOnly | QIODevice::Text)){
                QTextStream out(&outputFile);
                {
                        QTextStream out(&outputFile);
                        for (int i = 0; i < values.size(); i++) {
                                out << values[i] << endl;
                        }
                        outputFile.close();
                }
                QMessageBox::information(this, tr("Message"), tr("Operation successful."));
        }
        accept();
}

void SetDistance::createActions(){
        QSignalMapper* signalMapper = new QSignalMapper(this);
        connect(ui.dist0, SIGNAL(clicked()), signalMapper, SLOT(map()));
        connect(ui.dist5, SIGNAL(clicked()), signalMapper, SLOT(map()));
        connect(ui.dist10, SIGNAL(clicked()), signalMapper, SLOT(map()));
        connect(ui.dist20, SIGNAL(clicked()), signalMapper, SLOT(map()));

        connect(ui.spinBox, SIGNAL(valueChanged(int)), this, SLOT(setSBox()));

        signalMapper->setMapping(ui.dist0, 0);
        signalMapper->setMapping(ui.dist5, 5);
        signalMapper->setMapping(ui.dist10, 10);
        signalMapper->setMapping(ui.dist20, 20);

        connect(signalMapper, SIGNAL(mapped(int)), this, SLOT(setDistance(int)));

        saveAct = new QAction(tr("&Save"), this);
        connect(ui.okButton, SIGNAL(clicked()), this, SLOT(save()));
}
```

SetLifespanQ.hpp

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_SetLifespanQ.h"
#include <QLineEdit>
#include <QSpinBox>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QMessageBox>
using namespace std;

class SetLifespanQ : public QDialog {
	Q_OBJECT
public:
	SetLifespanQ(QWidget *parent = Q_NULLPTR);

private slots:
	void setTime();
	void save();

private:
	Ui::SetLifespanQ ui;
	void createActions();

	QAction *saveAct;

	int time = 10;
	QString lp;
	vector<int> values;
};
```

SetLifespanQ.cpp

```
#include "SetLifespanQ.hpp"

SetLifespanQ::SetLifespanQ(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);
	createActions();
	ui.lineEdit->setDisabled(true);

	QFile inputFile("Resources/values.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);
		while (!in.atEnd())
		{
			QString line = in.readLine();
			values.push_back(line.toInt());
		}
		inputFile.close();
		ui.lineEdit->setText(QString::number(values[3]));
	}

	if (values[2] < 60) {
		ui.spinBox->setMaximum(values[2]-5);/*set limit so that lifespan of quote
```

```
                                                                      does not go over the
appearance interval*/ //<------ values[2]-5
       }
}

void SetLifespanQ::setTime() {
       time = ui.spinBox->value();
       lp = QString::number(time);
       ui.lineEdit->setText(lp);
}

void SetLifespanQ::save() {
       values[3] = time; //index 3 is reserved for quote's lifespan
       QFile outputFile("Resources/values.txt");
       if (outputFile.open(QIODevice::WriteOnly | QIODevice::Text))
       {
              QTextStream out(&outputFile);
              {
                     QTextStream out(&outputFile);
                     for (int i = 0; i < values.size(); i++) {
                            out << values[i] << endl;
                     }
                     outputFile.close();
              }
              QMessageBox::information(this, tr("Message"), tr("Operation successful."));
       }
       ui.spinBox->setMaximum(60);/*Set limit back to 60*/
       accept();
}

void SetLifespanQ::createActions()
{
       connect(ui.spinBox, SIGNAL(valueChanged(int)), this, SLOT(setTime()));

       saveAct = new QAction(tr("&Save"), this);
       connect(ui.okButton, SIGNAL(clicked()), this, SLOT(save()));
}
```

SetTime.hpp

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_SetTime.h"
#include <QLineEdit>
#include <QSpinBox>
#include <QSignalMapper>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QMessageBox>
using namespace std;

class SetTime : public QDialog {
	Q_OBJECT
public:
	SetTime(QWidget *parent = Q_NULLPTR);
	~SetTime();

private slots:
	void setTime(int a);
	void setSBox();
	void save();

private:
	Ui::SetTime ui;
	void createActions();

	QSignalMapper* signalMapper;
	QAction *saveAct;

	int time = 60;
	QString t;
	vector<int> values;
};
```

SetTime.cpp

```
#include "SetTime.hpp"

SetTime::SetTime(QWidget *parent) :QDialog(parent) {
	ui.setupUi(this);
	createActions();
	ui.timeInput->setDisabled(true);

	QFile inputFile("Resources/values.txt");
	if (inputFile.open(QIODevice::ReadOnly))
	{
		QTextStream in(&inputFile);
		while (!in.atEnd())
		{
			QString line = in.readLine();
			values.push_back(line.toInt());
		}
		inputFile.close();
		ui.timeInput->setText(QString::number(values[0]));
	}
}

void SetTime::setTime(int tim) {
	time = tim;
	t = QString::number(time);
	ui.timeInput->setText(t);
}

SetTime::~SetTime(){

	delete saveAct;
}

void SetTime::setSBox(){

	time = ui.spinBox->value() * 100;
	t = QString::number(time);
	ui.timeInput->setText(t);
}

void SetTime::save() {
	values[0] = time; //index 0 is reserved for time interval
	QFile outputFile("Resources/values.txt");
	if (outputFile.open(QIODevice::WriteOnly | QIODevice::Text))
	{
		QTextStream out(&outputFile);
		{
			QTextStream out(&outputFile);
			for (int i = 0; i < values.size(); i++) {
				out << values[i] << endl;
			}
			outputFile.close();
		}
		QMessageBox::information(this, tr("Message"), tr("Operation successful."));
	}
	accept();
}

void SetTime::createActions()
{
	QSignalMapper* signalMapper = new QSignalMapper(this);
	connect(ui.sec1, SIGNAL(clicked()), signalMapper, SLOT(map()));
	connect(ui.sec3, SIGNAL(clicked()), signalMapper, SLOT(map()));
	connect(ui.sec5, SIGNAL(clicked()), signalMapper, SLOT(map()));
	connect(ui.sec10, SIGNAL(clicked()), signalMapper, SLOT(map()));
	connect(ui.sec30, SIGNAL(clicked()), signalMapper, SLOT(map()));
	connect(ui.sec60, SIGNAL(clicked()), signalMapper, SLOT(map()));

	connect(ui.spinBox, SIGNAL(valueChanged(int)), this, SLOT(setSBox()));//update 
lineEdit value for spinbox

	signalMapper->setMapping(ui.sec1, 100);
	signalMapper->setMapping(ui.sec3, 300);
	signalMapper->setMapping(ui.sec5, 500);
	signalMapper->setMapping(ui.sec10, 1000);
	signalMapper->setMapping(ui.sec30, 3000);
	signalMapper->setMapping(ui.sec60, 6000);

	connect(signalMapper, SIGNAL(mapped(int)), this, SLOT(setTime(int)));

	saveAct = new QAction(tr("&Save"), this);
	connect(ui.okButton, SIGNAL(clicked()), this, SLOT(save()));
}
```