ПРИЛОЖЕНИЕ А

Листинг программы с комментариями:

ball.h:

```
#pragma once
#include <QImage>
#include <QRect>

class Ball {
    public:
        Ball();
        ~Ball();
        void resetState();
        void autoMove();
        void setXDir(int);
        void setYDir(int);
        int getXDir();
        int getYDir();
        QRect getRect();
        QImage & getImage();
    private:
        int xdir;
        int ydir;
        QImage image;
        QRect rect;
        static const int INITIAL_X = 375;
        static const int INITIAL_Y = 680;
        static const int RIGHT_EDGE = 800;
};
```

ball.cpp:

```
#include <iostream>
#include "./Headers/ball.h"

Ball::Ball() {
    xdir = 1;
    ydir = -2;
    image.load(":/img/ball3.png");
    rect = image.rect();
    resetState();
}
Ball::~Ball() {
    std::cout << "Ball deleted" << std::endl;
}
void Ball::autoMove() {
    rect.translate(xdir, ydir);
    if (rect.left() == 0) {
        xdir = 1;
    }
    if (rect.right() == RIGHT_EDGE) {
        xdir = -1;
    }
    if (rect.top() == 0) {
        ydir = 1;
    }
}
void Ball::resetState() {
    rect.moveTo(INITIAL_X, INITIAL_Y);
}
void Ball::setXDir(int x) {
    xdir = x;
}
void Ball::setYDir(int y) {
    ydir = y;
}
int Ball::getXDir() {
    return xdir;
}
int Ball::getYDir() {
    return ydir;
}
QRect Ball::getRect() {
    return rect;
}
QImage & Ball::getImage() {
    return image;
}
```

paddle.h:

```
#pragma once
#include <QImage>
#include <QRect>

class Paddle {
    public:
        Paddle();
        ~Paddle();
        void resetState();
        void move();
        void move(int);
        void setDx(int);
        QRect getRect();
        QImage & getImage();
    private:
        QImage image;
        QRect rect;
        int dx;
        static const int INITIAL_X = 335;
        static const int INITIAL_Y = 700;
};
```

paddle.cpp:

```
#include <iostream>
#include "./Headers/paddle.h"

Paddle::Paddle() {
    dx = 0;
    image.load(":/img/paddle2.png");
    rect = image.rect();
    resetState();
}
Paddle::~Paddle() {
    std::cout << ("Paddle deleted") << std::endl;
}
void Paddle::setDx(int x) {
    dx = x;
}
void Paddle::move() {
    int x = rect.x() + dx;
    int y = rect.top();
    rect.moveTo(x, y);
}
void Paddle::move(int x) {
    rect.moveTo(x, INITIAL_Y);
}
void Paddle::resetState() {
    rect.moveTo(INITIAL_X, INITIAL_Y);
}
QRect Paddle::getRect() {
    return rect;
}
QImage & Paddle::getImage() {
    return image;
}
```

bonus.h:

```
#pragma once

#include <QImage>
#include <QRect>

class Bonus
{
public:
    Bonus(int, int, int);
    ~Bonus();
    void autoMove();
    void deleteObject();
    void resetState();
    bool isDestroyed();
    int getBonus(int);
    QRect getRect();
    QImage & getImage();
private:
    int xdir;
    int ydir;
    int typeBonus;
    bool destroyed;
    QImage image;
    QRect rect;
    int INITIAL_X;
    int INITIAL_Y;
};
```

bonus.cpp:

```
#include <iostream>
#include "./Headers/Bonus.h"

Bonus::Bonus(int x, int y, int type) {
    INITIAL_X = x + 35;
    INITIAL_Y = y;
    typeBonus = type;
    xdir = 0;
    ydir = +1;
    destroyed = true;
    image.load(":/img/bonus.png");
    rect = image.rect();
    resetState();
}
Bonus::~Bonus(){
    std::cout<<"Bonus deleted"<<std::endl;
}
void Bonus::deleteObject(){
    delete this;
}
void Bonus::resetState() {
    rect.moveTo(INITIAL_X, INITIAL_Y);
}
bool Bonus::isDestroyed(){
    return destroyed;
}
QRect Bonus::getRect() {
    return rect;
}
QImage & Bonus::getImage() {
    return image;
}
void Bonus::autoMove() {
    rect.translate(xdir, ydir);
}
int Bonus::getBonus(int score){
    switch (typeBonus) {
    case 1:
        score *=2;
        break;
    }
    return score;
}
```

yellowBrick.h:

```
#pragma once

#include <QImage>
#include <QRect>

class YellowBrick
{
public:
    YellowBrick(int, int);
    ~YellowBrick();
    bool isDestroyed();
    void setDestroyed(bool);
    QRect getRect();
    void setRect(QRect);
    QImage & getImage();
    int getXDir();
    int getYDir();
    int getScore();
    virtual void destroy();
    virtual void deleteObject();
protected:
    QImage image;
    QRect rect;
    int score = 10;
    bool destroyed;
    int xdir;
    int ydir;
};
```

yellowBrick.cpp:

```
#include <iostream>
#include "./Headers/YellowBrick.h"

YellowBrick::YellowBrick(int x, int y){
    image.load(":/img/brick_yellow.png");
    xdir = x;
    ydir = y;
    destroyed = false;
    rect = image.rect();
    rect.translate(x, y);
}
YellowBrick::~YellowBrick(){
    std::cout << "YellowBrick deleted" << std::endl;
}
QRect YellowBrick::getRect() {
    return rect;
}

void YellowBrick::setRect(QRect rct) {

    rect = rct;
}

QImage & YellowBrick::getImage() {
    return image;
}

bool YellowBrick::isDestroyed() {
    return destroyed;
}

void YellowBrick::setDestroyed(bool destroy) {
    destroyed = destroy;
}
int YellowBrick::getXDir(){
    return xdir;
}
int YellowBrick::getYDir(){
    return ydir;
}
int YellowBrick::getScore(){
    if (destroyed)
        return score;
    else
        return 0;
}
void YellowBrick::destroy(){
    destroyed = true;
}
void YellowBrick::deleteObject(){
    delete this;
}
```

redBrick.h:

```
#pragma once
#include <QRect>
#include "./Headers/YellowBrick.h"

class RedBrick : public YellowBrick
{
public:
    RedBrick(int, int);
    ~RedBrick();
    void destroy() override;
    void deleteObject() override;
protected:
    int stabillity = 1;
};
```

redBrick.cpp:

```
#include <iostream>
#include "./Headers/RedBrick.h"

RedBrick::RedBrick(int x, int y) : YellowBrick(x,y){
    image.load(":/img/brick_red.png");
    xdir = x;
    ydir = y;
    score = 20;
    destroyed = false;
    rect = image.rect();
    rect.translate(x, y);
}
RedBrick::~RedBrick(){
    std::cout << "RedBrick deleted" << std::endl;
}
void RedBrick::destroy(){
    if(stabillity == 1){
        image.load(":/img/brick_yellow.png");
        stabillity = 0;
    }
    else
        destroyed = true;
}
void RedBrick::deleteObject(){
    delete this;
}
```

blueBrick.h:

```
#pragma once
#include <QRect>
#include <QRandomGenerator>
#include "./Headers/RedBrick.h"
#include "./Headers/Bonus.h"

class BlueBrick : public RedBrick
{
public:
    BlueBrick(int, int);
    ~BlueBrick();
    void destroy() override;
    void deleteObject() override;
    bool getActiveBonus();
    Bonus getBonus();
protected:
    int stabillity = 2;
    Bonus *bonus;
    bool activeBonus;
};
```

blueBrick.cpp:

```
#include <iostream>
#include "./Headers/BlueBrick.h"

BlueBrick::BlueBrick(int x, int y) : RedBrick(x,y){
    image.load(":/img/brick_blue.png");
    xdir = x;
    ydir = y;
    QRandomGenerator *rg = QRandomGenerator::global();
    if (rg->bounded(1, 10) % 2 == 0){
        bonus = new Bonus(x,y,1);
        activeBonus = true;
    }else
        activeBonus = false;
    score = 50;
    destroyed = false;
    rect = image.rect();
    rect.translate(x, y);
}
BlueBrick::~BlueBrick(){
    std::cout << "BlueBrick deleted" << std::endl;
}
Bonus BlueBrick::getBonus(){
    return *bonus;
}
bool BlueBrick::getActiveBonus(){
    if(destroyed)
        return activeBonus;
    else
        return false;
}
void BlueBrick::destroy(){
    switch (stabillity) {
    case 2:
        image.load(":/img/brick_blue1.png");
        stabillity--;
        break;
    case 1:
        image.load(":/img/brick_blue2.png");
        stabillity--;
        break;
    case 0:
        destroyed = true;
        break;
    }
}
void BlueBrick::deleteObject(){
    delete this;
}
```

breakout.h:

```
#pragma once

#include <QWidget>
#include <QVector>
#include <QKeyEvent>
#include "ball.h"
#include "paddle.h"
#include "BlueBrick.h"
#include "YellowBrick.h"
#include "RedBrick.h"
#include "Bonus.h"
#include "list.h"

class Breakout : public QWidget {
public:
    Breakout(QWidget *parent = 0);
    ~Breakout();
protected:
    void paintEvent(QPaintEvent *);
    void timerEvent(QTimerEvent *);
    void keyPressEvent(QKeyEvent *);
    void keyReleaseEvent(QKeyEvent *);
    void drawObjects(QPainter *);
    void finishGame(QPainter *, QString);
    void moveObjects();
    void drawScore(QPainter *,int,int,int);
    void startGame();
    void pauseGame();
    void stopGame();
    void victory();
    void checkCollision();
private:
    int x;
    int score = 0;
    int timerId;
    static const int DELAY = 10;
    static const int BOTTOM_EDGE = 750;
    Ball *ball;
    Paddle *paddle;
    List<YellowBrick*> yellowBricks;
    List<RedBrick*> redBricks;
    List<BlueBrick*> blueBricks;
    List<Bonus*> bonuses;
    bool gameOver;
    bool gameWon;
    bool gameStarted;
    bool paused;
    bool gameMenu;
};
```

breakout.cpp:

```
 #include <QPainter>
#include <QApplication>
#include "./Headers/breakout.h"
#include <QRandomGenerator>
#include <QDateTime>
#include <QCoreApplication>
#include <QDateTime>
```

```
#include <iostream>

Breakout::Breakout(QWidget *parent): QWidget(parent) {
    QPixmap background(":/img/backgroundMenu.png");
    QPalette palette;
    palette.setBrush(QPalette::Window, background);
    this->setPalette(palette);
    x = 0;
    gameOver = false;
    gameWon = false;
    paused = false;
    gameMenu = true;
    gameStarted = false;
    ball = new Ball();
    paddle = new Paddle();
}

Breakout::~Breakout() {
    delete ball;
    delete paddle;

    yellowBricks.clearList();
}

void Breakout::paintEvent(QPaintEvent *event) {
    Q_UNUSED(event);

    QPainter painter(this);

    if (gameOver) {
        finishGame(&painter, "Game lost");
    }
    else if(gameWon) {
        finishGame(&painter, "Victory");
    }
    else {
        drawObjects(&painter);
    }
}

void Breakout::finishGame(QPainter *painter, QString message) {

    if(QString::compare(message,"Game lost") != 0)
    {
        QPixmap background(":/img/backgroundGameLost.png");
        QPalette palette;
        palette.setBrush(QPalette::Window, background);
        this->setPalette(palette);
        drawScore(painter,300,450, 30);
    }
    else
    {
        QPixmap background(":/img/backgroundVictory.png");
        QPalette palette;
        palette.setBrush(QPalette::Window, background);
        this->setPalette(palette);
        drawScore(painter,300,450,30);
    }
}

void Breakout::drawObjects(QPainter *painter) {
    if(!gameMenu)
    {
        painter->drawImage(ball->getRect(), ball->getImage());
        painter->drawImage(paddle->getRect(), paddle->getImage());

        for (int i = 0; i < bonuses.getSize(); ++i) {
            if((bonuses.peek(i))->isDestroyed())
                painter->drawImage((bonuses.peek(i))->getRect(),
(bonuses.peek(i))->getImage());
        }

        drawScore(painter,10,50,15);

        for (int i=0; i<yellowBricks.getSize(); i++) {
            if (!(yellowBricks.peek(i))->isDestroyed()) {
                painter->drawImage((yellowBricks.peek(i))->getRect(),
(yellowBricks.peek(i))->getImage());
            }
        }
        for (int i=0; i<redBricks.getSize(); i++) {
            if (!(redBricks.peek(i))->isDestroyed()) {
                painter->drawImage((redBricks.peek(i))->getRect(),
(redBricks.peek(i))->getImage());
            }
        }
        for (int i=0; i<blueBricks.getSize(); i++) {
            if (!(blueBricks.peek(i))->isDestroyed()) {
                painter->drawImage((blueBricks.peek(i))->getRect(),
(blueBricks.peek(i))->getImage());
            }
        }
    }

}

void Breakout::timerEvent(QTimerEvent *event) {
    Q_UNUSED(event);

    moveObjects();
    checkCollision();
    repaint();
}

void Breakout::moveObjects() {
    ball->autoMove();
    paddle->move();
    for (int i = 0; i < bonuses.getSize(); ++i) {
        (bonuses.peek(i))->autoMove();
    }
}

void Breakout::keyReleaseEvent(QKeyEvent *event) {

    if(event->key() == Qt::Key_Left || event->key() == Qt::Key_A)
    {
        if(paddle->getRect().left() <= 10)
            paddle->move(10);
        paddle->setDx(0);
    }

    if(event->key() == Qt::Key_Right || event->key() == Qt::Key_D)
    {
        if(paddle->getRect().right() >= 790)
```

```
            paddle->move(690);
        paddle->setDx(0);
    }
}

void Breakout::keyPressEvent(QKeyEvent *event) {

    if(event->key() == Qt::Key_Left || event->key() == Qt::Key_A)
    {
        if(paddle->getRect().left() <= 10) {
            paddle->setDx(0);
            paddle->move(10);
        }
        else
            paddle->setDx(-2);
    }

    if(event->key() == Qt::Key_Right || event->key() == Qt::Key_D)
    {
        if(paddle->getRect().right() >= 790) {
            paddle->setDx(0);
            paddle->move(690);
        }
        else
            paddle->setDx(2);
    }

    switch (event->key()) {
    case Qt::Key_P:
        pauseGame();
        break;

    case Qt::Key_Space:
        startGame();
        break;

    case Qt::Key_Escape:
        qApp->exit();
        break;

    default:
        QWidget::keyPressEvent(event);
    }
}

void Breakout::startGame() {
    if (!gameStarted) {
        QPixmap background(":/img/background.png");
        QPalette palette;
        palette.setBrush(QPalette::Window, background);
        this->setPalette(palette);

        ball->resetState();
        paddle->resetState();

        yellowBricks.clearList();
        redBricks.clearList();
        blueBricks.clearList();
        bonuses.clearList();

        QRandomGenerator *rg = QRandomGenerator::global();
        for (int i=0; i<5; i++) {
            for (int j=0; j<6; j++) {
                if (rg->bounded(1, 10) % 2 == 0)
                    yellowBricks.pushBack(new YellowBrick(j * 90 + 130, i *
30 + 100));
                else {
                    redBricks.pushBack(new RedBrick(j * 90 + 130, i * 30 +
100));
                    if (rg->bounded(1, 10) %2 == 0){
                        redBricks.popBack();
                        blueBricks.pushBack(new BlueBrick(j * 90 + 130, i *
30 + 100));
                    }
                }
            }
        }

        gameOver = false;
        gameWon = false;
        gameStarted = true;
        gameMenu = false;
        timerId = startTimer(DELAY);
    }
}

void Breakout::pauseGame() {
    if (paused) {
        timerId = startTimer(DELAY);
        paused = false;
    }
    else {
        paused = true;
        std::cout << "Pause" << std::endl;
        killTimer(timerId);

    }
}

void Breakout::stopGame() {
    killTimer(timerId);
    gameOver = true;
    gameStarted = false;

}

void Breakout::victory() {
    killTimer(timerId);
    gameWon = true;
    gameStarted = false;
}

void Breakout::drawScore(QPainter *painter, int x, int y, int size){
    QString stringNumber;
    std::cout << score<<std::endl;
    QString messageScore = "Score: " + stringNumber.setNum(score);
    QFont font("Courier", size, QFont::DemiBold);
    QFontMetrics fm(font);
    painter->setFont(font);
    painter->translate(QPoint(0, 0));
    painter->drawText(messageScore.length()/2 + x, y, messageScore);

}
```

```
void Breakout::checkCollision() {
    if (ball->getRect().bottom() > BOTTOM_EDGE) {
        stopGame();
    }

    for (int i = 0; i < bonuses.getSize(); ++i)
        if((bonuses.peek(i))->getRect().bottom() > BOTTOM_EDGE)
            bonuses.erase(i);

    for (int i = 0; i < bonuses.getSize(); ++i) {
        if(((bonuses.peek(i))->getRect()).intersects(paddle->getRect())){
            score = (bonuses.peek(i))->getBonus(score);
            bonuses.erase(i);
        }
    }

    if (ball->getRect().bottom() < 0) {
        QRandomGenerator *rg = QRandomGenerator::global();
        if (rg->bounded(1, 10) % 2 == 0) {
            ball->setXDir(-1);
        }
        else    {
            ball->setXDir(1);
        }
        ball->setYDir(+2);
    }

    int numberOfBlocks = 0;

    for (int i=0; i<yellowBricks.getSize(); i++)
        if ((yellowBricks.peek(i))->isDestroyed())
            numberOfBlocks++;

    for (int i=0; i<redBricks.getSize(); i++)
        if ((redBricks.peek(i))->isDestroyed())
            numberOfBlocks++;

    for (int i=0; i<blueBricks.getSize(); i++)
        if ((blueBricks.peek(i))->isDestroyed())
            numberOfBlocks++;


    if (numberOfBlocks == 30) {
        victory();
    }

    if ((ball->getRect()).intersects(paddle->getRect())) {
        int paddleLPos = paddle->getRect().left();
        int ballLPos = ball->getRect().left();

        int first = paddleLPos + 3;
        int second = paddleLPos + 25;
        int third = paddleLPos + 50;
        int fourth = paddleLPos + 75;

        if (ballLPos < first) {
            ball->setXDir(-1);
            ball->setYDir(-2);
        }

        if (ballLPos >= first && ballLPos < second) {
            ball->setXDir(-1);
            ball->setYDir(-1*ball->getYDir());
        }

        if (ballLPos >= second && ballLPos < third) {
            ball->setXDir(0);
            ball->setYDir(-3);
        }

        if (ballLPos >= third && ballLPos < fourth) {
            ball->setXDir(1);
            ball->setYDir(-1*ball->getYDir());
        }

        if (ballLPos > fourth) {
            ball->setXDir(1);
            ball->setYDir(-2);
        }
    }

    for (int i=0; i<yellowBricks.getSize(); i++) {
        if ((ball->getRect()).intersects((yellowBricks.peek(i))->getRect()))
{
            int ballLeft = ball->getRect().left();
            int ballHeight = ball->getRect().height();
            int ballWidth = ball->getRect().width();
            int ballTop = ball->getRect().top();

            QPoint pointRight(ballLeft + ballWidth + 1, ballTop);
            QPoint pointLeft(ballLeft - 1, ballTop);
            QPoint pointTop(ballLeft, ballTop -1);
            QPoint pointBottom(ballLeft, ballTop + ballHeight + 1);

            if (!(yellowBricks.peek(i))->isDestroyed()) {
                //x
                if((yellowBricks.peek(i))->getRect().contains(pointRight)) {
                    ball->setXDir(-1);
                }
                else
if((yellowBricks.peek(i))->getRect().contains(pointLeft)) {
                    ball->setXDir(1);
                }
                //y
                if((yellowBricks.peek(i))->getRect().contains(pointTop)) {
                    ball->setYDir(2);
                }
                else
if((yellowBricks.peek(i))->getRect().contains(pointBottom)) {
                    ball->setYDir(-2);
                }
                (yellowBricks.peek(i))->destroy();
                score += (yellowBricks.peek(i))->getScore();
            }
        }
    }

    for (int i=0; i<redBricks.getSize(); i++) {
        if ((ball->getRect()).intersects((redBricks.peek(i))->getRect())) {
            int ballLeft = ball->getRect().left();
            int ballHeight = ball->getRect().height();
            int ballWidth = ball->getRect().width();
            int ballTop = ball->getRect().top();
```

```
                QPoint pointRight(ballLeft + ballWidth + 1, ballTop);
                QPoint pointLeft(ballLeft - 1, ballTop);
                QPoint pointTop(ballLeft, ballTop -1);
                QPoint pointBottom(ballLeft, ballTop + ballHeight + 1);

                if (!(redBricks.peek(i))->isDestroyed()) {
                    //x
                    if((redBricks.peek(i))->getRect().contains(pointRight)) {
                        ball->setXDir(-1);
                    }
                    else if((redBricks.peek(i))->getRect().contains(pointLeft)) {
                        ball->setXDir(1);
                    }
                    //y
                    if((redBricks.peek(i))->getRect().contains(pointTop)) {
                        ball->setYDir(2);
                    }
                    else if((redBricks.peek(i))->getRect().contains(pointBottom))
    {
                        ball->setYDir(-2);
                    }

                    (redBricks.peek(i))->destroy();
                    score += (redBricks.peek(i))->getScore();
                }
            }
        }

        for (int i=0; i<blueBricks.getSize(); i++) {
            if ((ball->getRect()).intersects((blueBricks.peek(i))->getRect())) {
                int ballLeft = ball->getRect().left();
                int ballHeight = ball->getRect().height();
                int ballWidth = ball->getRect().width();
                int ballTop = ball->getRect().top();

                QPoint pointRight(ballLeft + ballWidth + 1, ballTop);
                QPoint pointLeft(ballLeft - 1, ballTop);
                QPoint pointTop(ballLeft, ballTop -1);
                QPoint pointBottom(ballLeft, ballTop + ballHeight + 1);

                if (!(blueBricks.peek(i))->isDestroyed()) {
                    //x
                    if((blueBricks.peek(i))->getRect().contains(pointRight)) {
                        ball->setXDir(-1);
                    }
                    else if((blueBricks.peek(i))->getRect().contains(pointLeft))
    {
                        ball->setXDir(1);
                    }
                    //y
                    if((blueBricks.peek(i))->getRect().contains(pointTop)) {
                        ball->setYDir(2);
                    }
                    else
    if((blueBricks.peek(i))->getRect().contains(pointBottom)) {
                        ball->setYDir(-2);
                    }

                    (blueBricks.peek(i))->destroy();
                    score += (blueBricks.peek(i))->getScore();
                    if((blueBricks.peek(i))->getActiveBonus()){
                        bonuses.pushBack(new
    Bonus((blueBricks.peek(i))->getXDir(),(blueBricks.peek(i))->getYDir(),1));
                        //bonuses.push_back(blueBricks[i]->getBonus());
                    }

                }
            }
        }
    }
```

list.h:

```
#pragma once
#include <iostream>
#include <conio.h>
#include <iostream>
#include <iomanip>
#include "./Headers/NoSuchElementException.h"

template<typename T>
class List
{
public:
    List();
    ~List();
    List(const List&);
    void pushBack(T data);
    int getSize();
    void erase(int num);
    void clearList();
    void pushFront(T data);
    void popFront();
    void popBack();
    T& peek(int num);
private:

    struct Node
    {
        T data;
        Node* next, * prev;
        Node(T data = T(), Node* next = nullptr, Node* prev = nullptr)
        {
            this->data = data;
            this->next = next;
            this->prev = prev;
        }
    };
    Node* head;
    int Size;

};

template<typename T>
List<T>::List()
{
    Size = 0;
    head = nullptr;
}
template<typename T>
List<T>::~List()
```

```
{
    while (Size)
    {
        Node* temp = head;
        head = head->next;
        delete temp;
        Size--;
    }

}

template<typename T>
List<T>::List(const List<T>& copy)
{
    try
    {
        Node* temp_cpy = copy.head;
        head = new Node(temp_cpy->data);
        temp_cpy = temp_cpy->next;
        Node* temp = head;

        for (int i = 0; i < copy.Size - 1; i++)
        {
            temp->next = new Node(temp_cpy->data, nullptr, temp);
            temp = temp->next;
            temp_cpy = temp_cpy->next;
        }
    }
    catch (bad_alloc& e)
    {
        std::cout << e.what() << std::endl;
    }
    this->Size = copy.Size;
}


template<typename T>
void List<T>::pushBack(T data)
{
    if (head == nullptr)
    {
        try
        {
            head = new Node(data);
        }
        catch (bad_alloc& e)
        {
            std::cout << e.what() << std::endl;
        }
    }
    else
    {
        try
        {
            Node* temp = this->head;
            while (temp->next != nullptr)
            {
                temp = temp->next;
            }
            temp->next = new Node(data, nullptr, temp);
        }
        catch (bad_alloc& e)
        {
            std::cout << e.what() << std::endl;
        }
    }
    Size++;
}

template<typename T>
inline int List<T>::getSize()
{
    return Size;
}

template<typename T>
void List<T>::erase(int num)//удаление по индексу
{
    if (!Size)
        return;

    Node* temp = head;
    try {
        if (num > this->Size || num < 0)
            throw NoSuchElementException();

    }
    catch (NoSuchElementException& exp)
    {
        exp.show();
        return;
    }
    if (num == 0)
    {
        head = head->next;
        head->prev = nullptr;
        delete temp;
        Size--;
        return;
    }
    for (int i = 0; i < num; i++)
    {
        temp = temp->next;
    }
    temp->prev->next = temp->next;
    temp->next = temp->prev;
    Size--;
    delete temp;

}

template<typename T>
void List<T>::clearList()//очистка очереди
{
    if (!Size) return;
    while (Size)
    {
        Node* temp = head;
        head = head->next;
        delete temp;
        Size--;
    }
}
```

```
template<typename T>
void List<T>::pushFront(T data)
{
    try
    {
        if (head == nullptr)
        {
            head = new Node(data);
        }
        else
        {
            this->head->prev = new Node(data, head);
            head = head->prev;
        }
    }
    catch (bad_alloc& e)
    {
        std::cout << e.what() << std::endl;
    }

    Size++;
}

template<typename T>
void List<T>::popFront()
{
    this->erase(0);

}

template<typename T>
void List<T>::popBack()
{

    this->erase(this->Size - 1);

}

template<typename T>
T& List<T>::peek(int num)
{
    if (this->Size == 0)
    {
        cout << "empty list" << endl;
        exit(1);
    }

    Node* temp = this->head;
    for (int i = 0; i < num; i++)temp = temp->next;
    return temp->data;
}
```

MyException.h:

```
#pragma once
#include<cstring>
#include<iostream>

class MyException
```

```
{
protected:
    char message[80];
    int code;
public:
    MyException(const char* message);
    MyException();
    virtual void show();
};
```

MyException.cpp:

```
#include "./Headers/MyException.h"

MyException::MyException(const char * message) {

    strcpy_s(this->message, message);
    this->code = code;
}

MyException::MyException() {
    std::cout << "Exception was generated" << std::endl;
}

void MyException::show() {
    std::cout << "Base exception was generated" << std::endl;
}
```

IOException.h:

```
#pragma once
#include "./Headers/MyException.h"
class IOException : public MyException
{
public :
    IOException(int code,const char* message);
    IOException();
    void show() override;
    int getCode();
};
```

IOException.cpp:

```
#include "./Headers/IOException.h"

IOException::IOException(int code,const char * message) :
MyException(message){
    this->code = code;
}

IOException::IOException() {
    this->code = 2;
}

int IOException::getCode() {
    return code;
}

void IOException::show() {
    std::cout << "IOException was generated" << std::endl;
    std::cout << "Code: " << this->getCode() << std::endl;
}
```

FileNotFoundException.h:

```
#pragma once
#include "./Headers/IOException.h"

class FileNotFoundException : public IOException{

  FileNotFoundException(const char* message,int code);
  FileNotFoundException();
  int getCode();
  void show() override;
};
```

FileNotFoundException.cpp:

```
#include "./Headers/FileNotFoundException.h"


FileNotFoundException::FileNotFoundException(const char* message, int code) :
IOException(code, message) {
  this->code = code;
}

FileNotFoundException::FileNotFoundException() {
  this->code = 3;
}

void FileNotFoundException::show() {
  std::cout << "FileNotFoundException was generated" << std::endl;
  std::cout << "Code: " << this->getCode() << std::endl;
}

int FileNotFoundException::getCode() {
  return code;
}
```

NoSuchElementException.h:

```
#pragma once
#include "./Headers/MyException.h"
class NoSuchElementException : public MyException
{
public:
    NoSuchElementException(int code, const char* message);
    NoSuchElementException();
    void show() override;
    int getCode();
};
```

NoSuchElementException.cpp:

```
#include "./Headers/NoSuchElementException.h"

NoSuchElementException::NoSuchElementException(int code, const char* message)
: MyException(message) {
    this->code = code;
}
```

```
NoSuchElementException::NoSuchElementException() {
    this->code = 1;
}

int NoSuchElementException::getCode() {
    return code;
}

void NoSuchElementException::show() {
    std::cout << "NoSuchElementException was generated" << std::endl;
    std::cout << "Code: " << this->getCode() << std::endl;
}
```

MainWindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
    int time;
    QTimer *timer;


private slots:
    void TimerSlot();
};


#endif // MAINWINDOW_H
```

MainWindow.cpp:

```
#include "./Headers/mainwindow.h"
#include "./Headers/ui_mainwindow.h"
#include <QTimer>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    time = 0;
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()),this,SLOT(TimerSlot()));
    timer->start(1);

}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::TimerSlot(){
    time++;
    ui->label->setText(QString::number(time));
}
```

Main.cpp:

```
#include <QIcon>
#include <QApplication>
#include "./Headers/breakout.h"

int main(int argc, char *argv[]) {

  QApplication app(argc, argv);
  app.setWindowIcon(QIcon(":/img/breakout.png"));
  Breakout window;

  window.setFixedSize(QSize(800, 750));//660 900
  window.setWindowTitle("Arcanoid");
  window.show();

  return app.exec();
}
```