МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Брестский Государственный Технический Университет
Кафедра интеллектуальных информационных технологий

Лабораторная работа №1-2
по дисциплине «ОсиСП» за 5 семестр
на тему «Проектирование и разработка приложений с графическим пользовательским интерфейсом в ОС Windows средствами Qt»

Выполнила: студентка
2 курса
Факультета ЭИС
Андросюк Мария
Михайловна
Проверила:
Дряпко А.В.

Брест, 2021

Вариант 1:
Игра «Xonix» (Ксоникс). Реализовать один уровень с тремя точками. Суть игры состоит в том, что игрок управляет условным кораблем, представляющим собой точку. Игровое поле содержит т.н. море, представленное первоначально практически всей игровой областью (черный прямоугольник, расположенный в центре экрана и почти полностью закрывающий собой поле).
По морю двигаются вражеские корабли, представленные точками. Двигаться они могут только лишь по диагонали и не могут выходить за пределы области моря. Игрок может двигаться либо по вертикали, либо по горизонтали. При этом, если он попадает в область моря, за точкой появляется линия. Если игрок успевает выйти за область моря, то часть, очерченная кораблем, становится сушей. Если же линию, оставляемую кораблем, пересекает вражеский корабль или же он сталкивается с кораблем игрока, игра заканчивается. Смысл игры в том, чтобы максимально уменьшить область моря. Игра считается выигранной, если игроку удается сократить размеры моря до 25% или менее от первоначальной.

game.h

```
#ifndef GAME_H
#define GAME_H

#include <QGraphicsScene>
#include <QTimer>
#include <QSet>
#include "gitem.h"
#include "engine/xonix.h"

class Game : public QGraphicsScene
{
    Q_OBJECT
private:
    int m_width;
    int m_height;
    QTimer m_timer;
    QSet<int> pressedKeys;
    QMap<int,GItem*> m_items;
    bool m_blocks_removed;
    int m_key_mask;
    bool m_victory;
    int m_evils;

    void drawBlock(bool busy, int x, int y, QPainter *painter, const QRectF &
rect );
    void drawBackground ( QPainter * painter, const QRectF & rect );
    void victory();


    void keyPressEvent( QKeyEvent *keyEvent);
    void keyReleaseEvent( QKeyEvent * keyEvent);
    void focusInEvent ( QFocusEvent * focusEvent);
    void focusOutEvent ( QFocusEvent * focusEvent);

public:
    explicit Game(QObject *parent = 0, int width=40, int height=30);
    void startGame();
    void xonixEvent( XonixEvent *e);

signals:
    void score(int s);

public slots:
    void myAdvance();
```

```
};

#endif // GAME_H
```

game.cpp

```
#include "game.h"
#include <QPainter>
#include <QPen>
#include <QKeyEvent>
#include <QDebug>

QPen freePen;
QBrush blockBrush(QColor(200,200,200));
QBrush bgBrush(QColor(255,255,255));

QMap<int,int> keyMap;

const int period=17;

void xonix_callback(void *tag, XonixEvent *e)
{
    if( tag) {
        ((Game *)tag)->xonixEvent(e);
    }
}

Game::Game(QObject *parent, int width, int height) :
    QGraphicsScene(parent),m_width(width),m_height(height)
{
    setSceneRect(0, 0, m_width, m_height);

    freePen.setColor(QColor(200,200,200));
    freePen.setWidth(0);

    keyMap[Qt::Key_Up] = X_KEY_UP;
    keyMap[Qt::Key_Down] = X_KEY_DOWN;
    keyMap[Qt::Key_Left] = X_KEY_LEFT;
    keyMap[Qt::Key_Right] = X_KEY_RIGHT;
    m_evils = 0;

    m_key_mask = 0;
    m_timer.setInterval(period);
    QObject::connect(&m_timer, SIGNAL(timeout()), this, SLOT(myAdvance()));

    //startGame();
}

void Game::startGame()
{
    QList<int> keys = m_items.keys();
    foreach(int k, keys) {
        GItem *i = m_items[k];
        removeItem(i);
        delete i;
    }
    m_evils++;
    m_key_mask = 0;
    m_victory = false;
    pressedKeys.clear();
```

```
    invalidate(sceneRect(), QGraphicsScene::BackgroundLayer);
    xonix_free();
    xonix_init(m_width, m_height, period, m_evils, xonix_callback, this);
    m_timer.start();
}

void Game::drawBackground ( QPainter * painter, const QRectF & rect )
{
    painter->setPen(freePen);
    //qDebug() << "rect: " << rect;
    painter->setRenderHint(QPainter::Antialiasing);
    for(int x=0; x<m_width; x++) {
        for(int y=0; y<m_height; y++) {
            bool busy=false;
            Cell c;
            xonix_get_cell(x, y, &c);
            if( xonix_cell_is(&c, BLOCK)) {
                busy=true;
            }
            drawBlock(busy, x, y, painter, rect);
        }
    }
    if( m_victory) {
        QBrush textBrush(QColor(0,220,0));
        QPen textPen;
        textPen.setWidth(0);
        QFont font;

        textPen.setColor(QColor(0,220,0));
        painter->setBrush(textBrush);
        painter->setPen(textPen);
        QRectF r=sceneRect();

        font = painter->font();

        font.setFamily("Arial");

        font.setPointSizeF(1.0*m_width/15);
        painter->setFont(font);

        painter->drawText(0,0, m_width, m_height/1.5, Qt::AlignCenter, "Level
cleared");

        font.setPointSizeF(1.0*m_width/30);
        painter->setFont(font);

        painter->drawText(0,0, m_width, m_height*1.1, Qt::AlignCenter, "Press
space to continue");
    }
}

void Game::drawBlock(bool busy, int ix, int iy, QPainter *painter, const
QRectF & rect)
{
    qreal x = ix;
    qreal y = iy;
    qreal m = 0.05;
    QRectF r(x+m, y+m, 1-2*m, 1-2*m);
    QRectF cell_rect(x, y, 1, 1);

    if( !cell_rect.intersects(rect)) {
        return;
    }
```

```
    painter->fillRect(QRectF(x, y, 1, 1), bgBrush);

    if(busy) {
        painter->fillRect(r, blockBrush);
    } else {
        painter->drawRect(r);
    }

}

void Game::keyPressEvent( QKeyEvent *keyEvent)
{
    int k = keyEvent->key();
    if( m_victory) {
        if( k == Qt::Key_Space) {
            startGame();
        }
    }
    pressedKeys.insert(k);
    if(keyMap.contains(k)) {
        m_key_mask |= keyMap[k];
    }
}

void Game::keyReleaseEvent( QKeyEvent * keyEvent)
{
    pressedKeys.remove(keyEvent->key());
}

void Game::focusInEvent ( QFocusEvent * )
{
    pressedKeys.clear();
    m_timer.start();
}

void Game::focusOutEvent( QFocusEvent * )
{
    pressedKeys.clear();
    m_timer.stop();
}

void Game::myAdvance()
{

    foreach(int k, pressedKeys){
        if(keyMap.contains(k)) {
            m_key_mask |= keyMap[k];
        }
    }

    m_blocks_removed = false;
    xonix_advance(m_key_mask);
    m_key_mask = 0;
    if(m_blocks_removed) {
        invalidate(sceneRect(), QGraphicsScene::BackgroundLayer);
    }

    advance();
}

void Game::victory()
{
    m_victory = true;
    m_key_mask = 0;
```

```
    pressedKeys.clear();
    invalidate(sceneRect(), QGraphicsScene::BackgroundLayer);
}

void Game::xonixEvent(XonixEvent *e)
{
    if(e->et == eVictory) {
        victory();
        return;
    }

    if(e->et == eScore) {
        emit score(e->x);
    }

    if(e->et == eObjAdded) {
        if(e->ot == objBlock) {
            return;
        }
        GItem *i = new GItem(e);
        addItem(i);
        m_items[e->id] = i;
    }

    if(e->et == eObjMoved) {
        (m_items[e->id])->move(e->x, e->y, e->t_len);
    }

    if(e->et == eObjDeleted) {
        if(e->ot == objBlock) {
            m_blocks_removed = true;
            //victory();
            return;
        }

        GItem *i = m_items[e->id];
        m_items.remove(e->id);
        removeItem(i);
        delete i;
    }
}
```

gitem.h

```
#ifndef GITEM_H
#define GITEM_H

#include <QGraphicsItem>
#include "engine/xonix.h"

class GItem : public QGraphicsItem
{
private:
    QColor m_color;
    ObjType m_type;
    int m_id;
    int m_x;
    int m_y;
    int m_new_x;
    int m_new_y;
    int m_old_x;
    int m_old_y;
    int m_t_len;
```

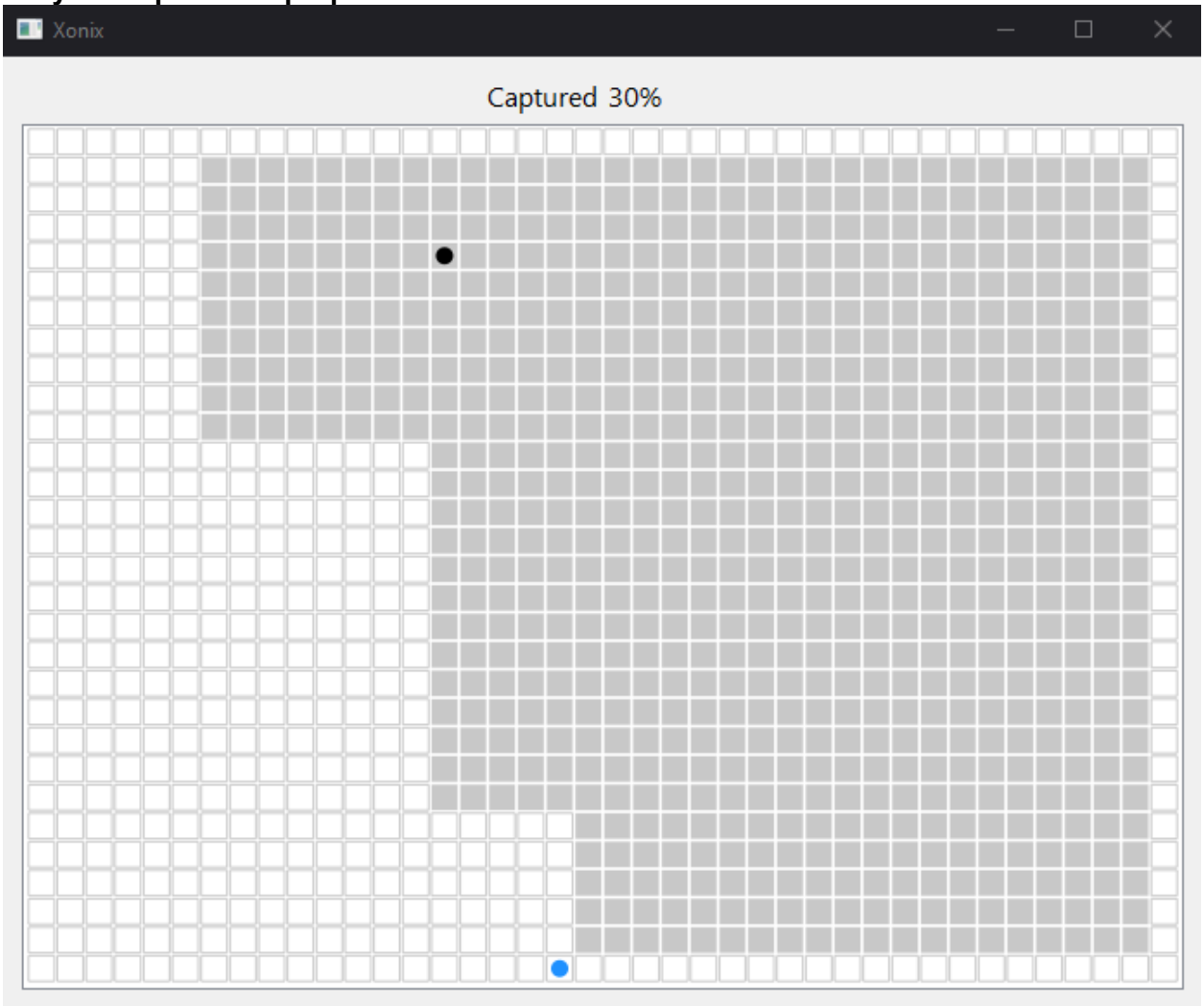
```
    int m_t_curr;
    bool m_transition;


public:
    GItem(XonixEvent *e);
    void paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
            QWidget *widget);
    QRectF boundingRect() const;
    void advance(int phase);
    void move(int x, int y, int t_len);
};

#endif // GITEM_H
```

gitem.cpp

```
#include "gitem.h"
#include <QBrush>
#include <QPainter>

GItem::GItem( XonixEvent *e)
{
    m_type = e->ot;
    m_id = e->id;
    m_x = e->x;
    m_y = e->y;
    m_new_x = -1;
    m_new_y = -1;
    m_t_curr = 0;
    m_t_len = 0;
    switch(m_type) {

    case objPlayer:
        m_color = QColor(30,144,255);
        setZValue(1.0);
        break;

    case objEvil:
        m_color = QColor(0,0,0);
        break;

    case objPath:
        m_t_curr = 0;
        m_t_len = e->t_len;
        setZValue(0.5);
        m_color = QColor(255,215,0);
    }

    setPos(m_x, m_y);
}

QRectF GItem::boundingRect() const
{
    qreal adjust = 0.05;
    return QRectF( -adjust, -adjust, 1+2*adjust, 1+2*adjust);
}

void GItem::paint(QPainter *painter, const QStyleOptionGraphicsItem *,
QWidget *)
{
    if( m_type == objPath) {
        if( m_t_curr != m_t_len) {
```

```
            return;
        }
    }

    QBrush b = painter->brush();
    QPen p = painter->pen();
    p.setStyle(Qt::NoPen);
    painter->setPen(p);

    b.setColor(this->m_color);
    b.setStyle(Qt::SolidPattern);
    painter->setBrush(b);
    painter->drawEllipse(QPointF(0.5,0.5), 0.3, 0.3);
}

void GItem::move(int x, int y, int t_len)
{
    m_old_x = m_x;
    m_old_y = m_y;
    m_x = x;
    m_y = y;
    m_t_len = t_len;
    m_t_curr = 0;
}

void GItem::advance(int phase)
{
    if(phase == 0) {
        return;
    }

    if( m_t_len == m_t_curr) {
        return;
    }

    m_t_curr++;

    if(m_type == objPath) {
        return;
    }

    qreal x = 1.0*m_x*m_t_curr/m_t_len + 1.0*(m_old_x)*(m_t_len -
m_t_curr)/m_t_len;
    qreal y = 1.0*m_y*m_t_curr/m_t_len + 1.0*(m_old_y)*(m_t_len -
m_t_curr)/m_t_len;
    setPos(x, y);
}
```

Результат работы программы:



Вывод: приобрести практические навыки проектирования и разработки приложений с графическим пользовательским интерфейсом в ОС Windows средствами Qt.