ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Департамент прикладной математики

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ 3
по дисциплине «Компьютерный практикум»

Работу выполнила
студентка группы БПМ 173 _____ М.В. Самоделкина
дата, подпись

Работу проверил _____ С.А. Булгаков
дата, подпись

Москва 2019

Содержание

Постановка задачи **3**

1 Основная часть **4**

1.1 Общая идея решения задачи 4

1.2 Структура и принципы действия 4

1.3 Процедура получения исполняемых программных модулей 7

1.4 Результаты тестирования 7

Приложение А **8**

MainWindow.cpp 8

MainWindow.h 8

GameItem.cpp 9

Game.h 10

Bee.h 14

Bee.cpp 15

Постановка задачи

Реализовать загрузку растровых изображений и переработать функционал отрисовки объектов таким образом, чтобы для отображения объектов использовались изображения. Окончательно доработать логику игры.

1 Основная часть

1.1 Общая идея решения задачи

В работе была реализована отрисовка объектов с помощью растровых изображений вместо геометрических фигур, используя класс *QImage*. Была завершена разработка логики игры: появилась возможность отображать счет и количество жизней на экране, обработка столкновений пчелы и врагов. Был улучшен механизм взаимодействий объектов.

1.2 Структура и принципы действия

В классе конкретного посетителя *DrawGameItems* был изменен механизм отрисовки объектов (функция *visit*). С помощью класса *QImage* и его методов *load* и *scaled* было загружено растровое изображение, которое масштабировалось под заданные размеры (в классе *GameItem* появились поля *sizeX* и *sizeY* для хранения ширины и высоты объекта). Таким образом, объекты классов *Bee*, *Cloud*, *Bell*, *RedEnemy* и *BlueEnemy* изображаются на экране с помощью выбранных картинок. Также был добавлен задний фон и установлены необходимые шрифты для отображения текста.

В классе *Game* была улучшена функция *Collide*, обрабатывающая столкновения объектов: стали использоваться поля *sizeX* и *sizeY* для расчета столкновений, была добавлена обработка взаимодействия врагов и пчелы; при их столкновении у пчелы отнимается жизнь, при этом количество жизней не может быть отрицательным и потеря жизни не может происходить чаще заданного времени *lifeTime*. Данный алгоритм описан в классе *Bee* в функциях *Update* и *SubLifes*.

Был доработан механизм окончания игры: когда количество жизней достигает 0, то игра останавливается (перестает производиться обработка нажатий и отрисовка), выводится надпись *”Game over”*.

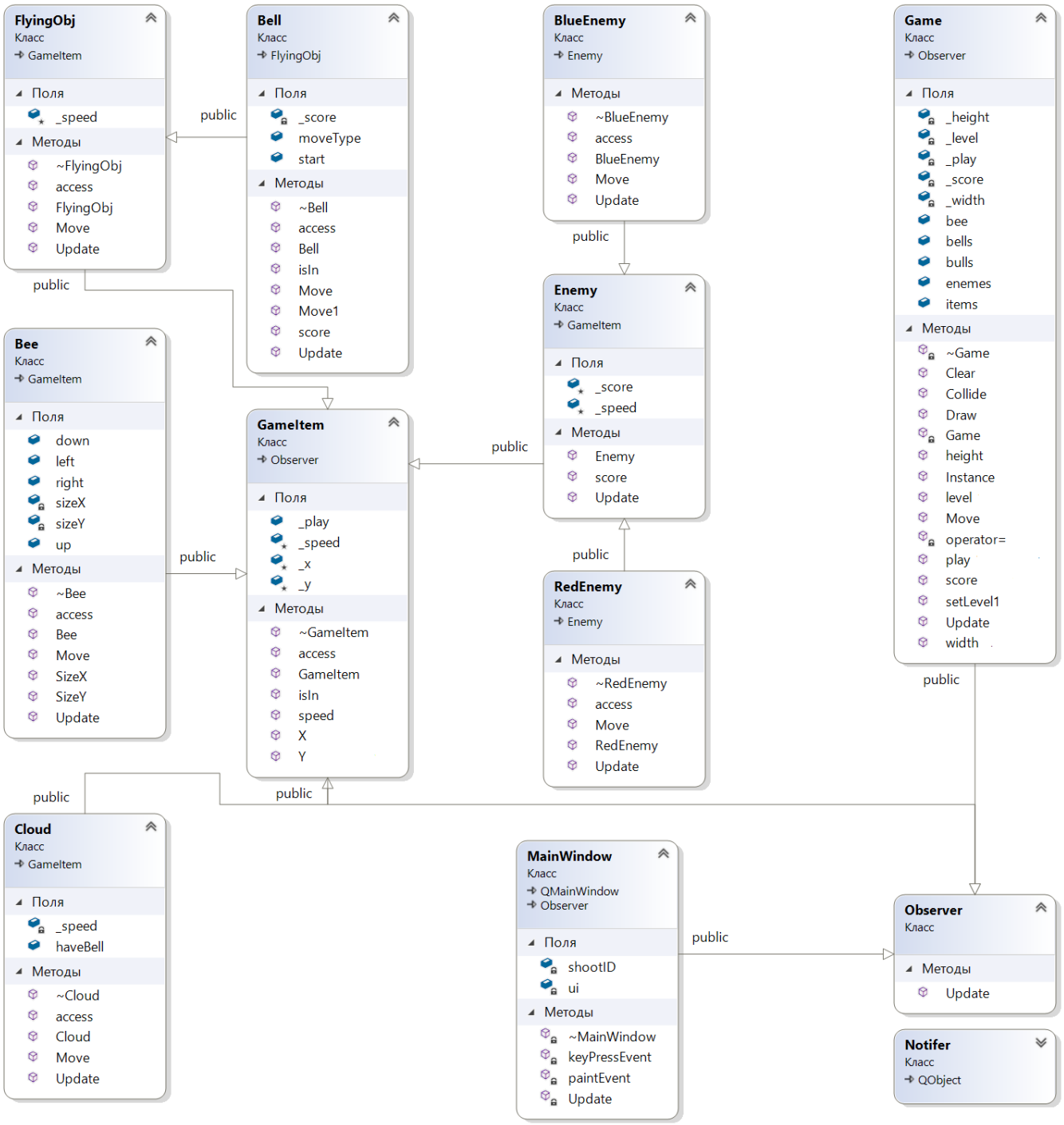

Рис. 1: Диграма классов

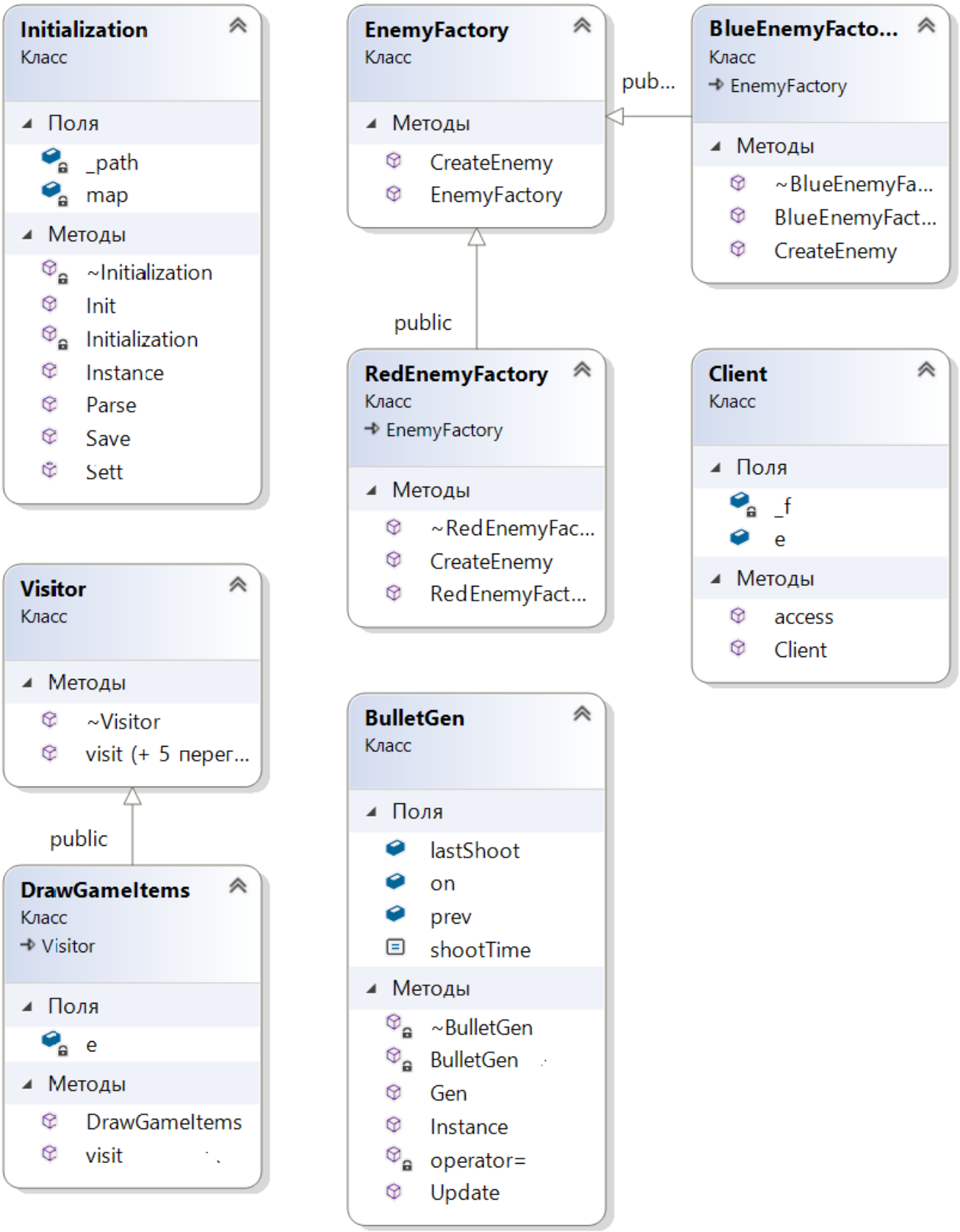

Рис. 2: Диграма классов (продолжение)

1.3 Процедура получения исполняемых программных модулей

Программный код был скомпилирован с среде *Qt Creator*. Компиляция раздельная: исходный код программы разделён на несколько файлов. Никаких дополнительных ключей не добавлялось, использовались ключи, которые добавляются по умолчанию. Параметры сборки: компилятор C++ MinGW 4.8.2, профиль Qt: Qt 4.8.7, отладчик GDB.

1.4 Результаты тестирования

Тестирование программы представлено в файле *"main.cpp"* в функции *Main()*. Ожидаемая отрисовка объектов в окне *MainWindow*:

Рис. 3: Результат тестирования

Приложение А

полный код программы

A.1 - MainWindow.cpp

```
1  MainWindow::MainWindow(QWidget *parent) :
2    QMainWindow(parent),
3    ui(new Ui::MainWindow)
4  {
5    ui->setupUi(this);
6    Game& game = Game::Instance();
7    this->setFixedSize(game.width(),game.height());
8    Notifer::Instance().Subscribe(this);
9    QPixmap bkgnd("images/back.png");
10   bkgnd = bkgnd.scaled(this->size(), Qt::IgnoreAspectRatio);
11   QPalette palette;
12   palette.setBrush(QPalette::Background, bkgnd);
13   this->setPalette(palette);
14   QFont f( "Arial", 15, QFont::Bold);
15
16   score = new QLabel(this);
17   score->setFrameStyle(QFrame::Sunken);
18   score->setGeometry(QRect(10,30,200,20));
19   score->setStyleSheet("QLabel {color : white; }");
20   score->setFont( f);
21   lifes = new QLabel(this);
22   lifes->setFrameStyle(QFrame::Sunken);
23   lifes->setGeometry(QRect(10,10,200,20));
24   lifes->setStyleSheet("QLabel {color : white; }");
25   lifes->setFont( f);
26
27   QFont f1( "Arial", 30, QFont::Bold);
28   over = new QLabel(this);
29   over->setFrameStyle(QFrame::Sunken);
30   over->setGeometry(QRect(game.width()*0.36,game.height()/2,220,30));
31   over->setStyleSheet("QLabel {color : white; }");
32   over->setFont( f1);
33 };
```

A.2 - MainWindow.h

```
1  #pragma once
2  #include <QMainWindow>
3  #include <QKeyEvent>
4  #include "GameItem.h"
5  #include "observer.h"
6  #include "Game.h"
7  #include "bulletgen.h"
8  #include <QLabel>
9
10 namespace Ui {
11 class MainWindow;
12 }
13
14 class MainWindow : public QMainWindow, public Observer
15 {
```

```
16    Q_OBJECT
17
18  public:
19    explicit MainWindow(QWidget *parent = nullptr);
20    ~MainWindow();
21    void paintEvent(QPaintEvent *event);
22    void Update(const Notifer& n) {
23      Game& game = Game::Instance();
24      if(game.play()){
25        game.bee->up = keysPressed.contains(Qt::Key_Up);
26        game.bee->right = keysPressed.contains(Qt::Key_Right);
27        game.bee->left = keysPressed.contains(Qt::Key_Left);
28        game.bee->down = keysPressed.contains(Qt::Key_Down);
29
30        BulletGen& gen = BulletGen::Instance();
31        gen.on = keysPressed.contains(Qt::Key_Space);
32        gen.Update(Notifer::Instance());
33
34        game.bee->Move();
35      }
36      else{
37        over -> setText(QString("Game␣over"));
38      }
39      score -> setText(QString("Score:␣%1").arg(game.score()));
40      lifes -> setText(QString("Lifes:␣%1").arg(game.bee->Lifes() ));
41      update();
42    }
43  private:
44    Ui::MainWindow *ui;
45    QLabel *score;
46    QLabel *lifes;
47    QLabel *over;
48    int shootID = 0;
49    QSet<int> keysPressed;
50  public slots:
51    void keyReleaseEvent(QKeyEvent*);
52    void keyPressEvent(QKeyEvent*);
53  };
```

A.3 - GameItem.cpp

```
1  #pragma once
2  #include "Game.h"
3
4  bool GameItem::isIn(){
5    Game& game = Game::Instance();
6    return (0 < _x || _x - sizeX < game.width() || 0 <_y || _y - sizeY <
       game.height() );
7  }
8
9  GameItem::~GameItem(){}
10
11 void DrawGameItems::visit(Bee &b){
12   QPainter painter(e);
13   QImage img;
14   img.load("images/Bee.png");
15   img = img.scaled(b.SizeX(), b.SizeY(), Qt::KeepAspectRatio);
16   painter.drawImage(b.X(),b.Y(),img);
```

```
17    painter.end();
18  }
19
20  void DrawGameItems::visit(Cloud &c){
21    QPainter painter(e);
22    QImage img;
23    img.load("images/cloud.png");
24    img = img.scaled(c.SizeX(), c.SizeY(), Qt::KeepAspectRatio);
25    painter.drawImage(c.X(),c.Y(),img);
26
27    painter.end();
28  }
29
30  void DrawGameItems::visit(FlyingObj &f){
31    QPainter painter(e);
32    QColor gray("#A52A2A");
33    Qt::BrushStyle style = Qt::SolidPattern;
34    QBrush brush(gray, style);
35    painter.setBrush(brush);
36    painter.setPen(Qt::NoPen);
37    painter.drawEllipse(f.X(),f.Y(), 10,10);
38    painter.end();
39  }
40
41  void DrawGameItems::visit(Bell &b){
42    QPainter painter(e);
43    QImage img;
44    img.load("images/bell.png");
45    img = img.scaled(b.SizeX(), b.SizeY(), Qt::KeepAspectRatio);
46    painter.drawImage(b.X(),b.Y(),img);
47    painter.end();
48  }
49
50  void DrawGameItems::visit(RedEnemy &r){
51    QPainter painter(e);
52    QImage img;
53    img.load("images/Putan.png");
54    img = img.scaled(r.SizeX(), r.SizeY(), Qt::KeepAspectRatio);
55    painter.drawImage(r.X(),r.Y(),img);
56    painter.end();
57  }
58
59  void DrawGameItems::visit(BlueEnemy &b){
60    QPainter painter(e);
61    QImage img;
62    img.load("images/Tobikame.png");
63    img = img.scaled(b.SizeX(), b.SizeY(), Qt::KeepAspectRatio);
64    painter.drawImage(b.X(),b.Y(),img);
65
66    painter.end();
67  }
```

A.4 - Game.h

```
1  #pragma once
2  #include "Bee.h"
3  #include "Enemy.h"
4  #include "Cloud.h"
```

```
 5 #include "Bell.h"
 6 #include "flyingobj.h"
 7 #include "initialization.h"
 8
 9
10 class Game : public Observer
11 {
12   Game(){
13     Initialization& ini = Initialization::Instance();
14     _level = ini.Sett("setgame/level", 1);
15     if (_level < 0){
16       _level = 1;
17     }
18     _play = true;
19     _width = ini.Sett("setgame/width", 900);
20     if (_width < 0){
21       _width = 900;
22     }
23     _height = ini.Sett("setgame/height", 700);
24     if (_width < 0){
25       _width = 700;
26     }
27     _score = ini.Sett("setgame/score");
28     if (_score < 0){
29       _score = 0;
30     }
31     setLevel1();
32     Notifer::Instance().Subscribe(this);
33   }
34   ~Game(){
35     Initialization& ini = Initialization::Instance();
36     ini.Save("setgame", "level", 1);
37     ini.Save("setgame", "score", 0/*_score*/);
38     ini.Save("logs", "no logs", 1);
39     delete bee;
40     for (auto it = items.begin(); it != items.end(); it++) {
41       delete *it;
42       items.erase(it);
43       it--;
44     }
45     for (auto it = enemes.begin(); it != enemes.end(); it++) {
46       delete *it;
47       enemes.erase(it);
48       it--;
49     }
50     for (auto it = bulls.begin(); it != bulls.end(); it++) {
51       delete *it;
52       bulls.erase(it);
53       it--;
54     }
55     for (auto it = bells.begin(); it != bells.end(); it++) {
56       delete *it;
57       bells.erase(it);
58       it--;
59     }
60   }
61   Game(Game const&) = delete;
```

```
62    Game& operator= (Game const&) = delete;
63    int _width, _height;
64    int _score;
65    bool _play;
66    int _level;
67  public:
68    Bee *bee;
69    QVector<FlyingObj*> bulls;
70    QVector<Cloud*> items;
71    QVector<Client*> enemes;
72    QVector<Bell*> bells;
73    void Collide(const Notifer& n){
74      for (auto b: bulls){
75        for (auto i: enemes){
76          if (b->X() > i->e->X() && b->X() + b->SizeX() < i->e->X() + i->
      e->SizeX() &&
77              b->Y() < i->e->Y() + i->e->SizeY() && b->Y() + b->SizeY() >
       i->e->Y()){
78            i->e->_play = false;
79            b->_play = false;
80            _score += i->e->score();
81          }
82        }
83      }
84      for (auto b: bulls){
85        for (auto i: items){
86          if (b->X() > i->X() && b->X() + b->SizeX() < i->X() + i->SizeX
      () &&
87              b->Y() < i->Y() + i->SizeY() && b->Y() + b->SizeY() > i->Y
      ()){
88            if(i->haveBell){
89              i->haveBell = false;
90              bells.push_back(new Bell(b->X() + b->SizeX()/2, i->Y()));
91            }
92            b->_play = false;
93          }
94        }
95      }
96      for (auto b: bulls){
97        for (auto i: bells){
98          if (b->X() > i->X() && b->X() + b->SizeX() < i->X() + i->SizeY
      () &&
99              b->Y() < i->Y() + i->SizeY() && b->Y() + b->SizeY() > i->Y
      ()){
100           i->moveType = 1;
101           i->start = Notifer::Instance().getStage();
102           b->_play = false;
103         }
104       }
105     }
106     for (auto i: bells){
107       if (i->X() + i->SizeX()/2 > bee->X() && i->X() + i->SizeX()/2 <
      bee->X() + bee->SizeX() &&
108           i->Y() + i->SizeY()/2 > bee->Y() && i->Y() + i->SizeY()/2 <
      bee->Y() + bee->SizeY()){
109         _score += i->score();
110         i->_play = false;
```

```
111         }
112       }
113       for (auto i: enemes){
114         if (i->e->X() + i->e->SizeX()/2 > bee->X() && i->e->X() + i->e->
      SizeX()/2 < bee->X() + bee->SizeX() &&
115             i->e->Y() + i->e->SizeY()/2 > bee->Y() && i->e->Y() + i->e->
      SizeY()/2 < bee->Y() + bee->SizeY()){
116           bee->Update(n);
117         }
118       }
119     }
120     void Update(const Notifer& n) {
121       Collide(n);
122     }
123     void setLevel1(){
124       bee = new Bee();
125       EnemyFactory *factory = new BlueEnemyFactory;
126       for (int i = 0; i < 3; i++) {
127         Client* enemy = static_cast<Client*>(new Client(factory));
128         enemes.push_back(enemy);
129       }
130       for (int i = 0; i < 3; i++) {
131         Cloud* item = static_cast<Cloud*>(new Cloud);
132         items.push_back(item);
133       }
134       delete factory;
135     }
136     void Clear(){
137       for (auto it = bulls.begin(); it != bulls.end(); it++) {
138         if (!(*it)->isIn() || !(*it)->_play){
139           delete *it;
140           bulls.erase(it);
141           it--;
142         }
143       }
144       for (auto it = enemes.begin(); it != enemes.end(); it++) {
145         if (!(*it)->e->isIn() || !(*it)->e->_play){
146           delete *it;
147           enemes.erase(it);
148           it--;
149         }
150       }
151       for (auto it = items.begin(); it != items.end(); it++) {
152         if (!(*it)->isIn()){
153           delete *it;
154           items.erase(it);
155           it--;
156         }
157       }
158       for (auto it = bells.begin(); it != bells.end(); it++) {
159         if (!(*it)->isIn() || !(*it)->_play){
160           delete *it;
161           bells.erase(it);
162           it--;
163         }
164       }
165     }
```

```
166     static Game& Instance()
167     {
168       static Game g;
169       return g;
170     }
171     void Draw(QMainWindow *e)
172     {
173       DrawGameItems visitor(e);
174       for(auto c: enemes){
175         c->access(visitor);
176       }
177       for(auto c: items){
178         c->access(visitor);
179       }
180       for(auto c: bulls){
181         c->access(visitor);
182       }
183       for(auto c: bells){
184         c->access(visitor);
185       }
186       bee->access(visitor);
187       Clear();
188     }
189     void Move(){}
190     int width() const { return _width; }
191     int height() const { return _height; }
192     int score() const { return _score; }
193     bool play() const { return _play; }
194     int level() const { return _level; }
195     void height(int height) { _height = height; }
196     void width(int width) { _width = width; }
197     void score(int score) { _score = score; }
198     void play(bool play) { _play = play; }
199     void level(int level) { _level = level; }
200 };
```

A.5 - Bee.h

```
1  #pragma once
2  #include "GameItem.h"
3  #include "initialization.h"
4
5  class Bee :
6    public GameItem
7  {
8    int _lifes;
9    int lastShoot;
10   int lifeTime;
11 public:
12   bool right, left, up, down;
13   Bee(){
14     Initialization& ini = Initialization::Instance();
15     _x = ini.Sett("setcoord/bee_x");
16     if (_x < 0){
17       _x = 0;
18     }
19     _y = ini.Sett("setcoord/bee_y");
20     if (_y < 0){
```

```
21        _y = 0;
22      }
23      _speed = ini.Sett("setspeed/Bee");
24      if (_speed < 0){
25        _speed = 0;
26      }
27      _lifes = ini.Sett("setgame/lifes");
28      if (_lifes < 0){
29        _lifes = 0;
30      }
31      sizeX = 60;
32      sizeY = 53;
33      right= false;
34      left= false;
35      up= false;
36      down= false;
37      lastShoot = -1;
38      lifeTime = 20;
39    }
40    ~Bee(){}
41    void access(Visitor &v) override {
42      v.visit(*this);
43     }
44    void Move();
45    void Update(const Notifer& n) override {
46      if(n.getStage() - lastShoot >= lifeTime){
47        lastShoot = n.getStage();
48        SubLifes();
49      }
50    }
51    void SubLifes();
52    int Lifes() {return _lifes;}
53  };
```

A.6 - Bee.cpp

```
1  #include "Bee.h"
2  #include "Game.h"
3
4  void Bee::Move(){
5    Game& game = Game::Instance();
6    if (left && _x - _speed >= 0){
7      _x -= _speed;
8    }
9    if (right && _x + sizeX + _speed <= game.width()) {
10     _x += _speed;
11   }
12   if (up && _y - _speed >= 0) {
13     _y -= _speed;
14   }
15   if (down && _y + sizeY /*+ _speed*/ <= game.height()) {
16     _y += _speed;
17   }
18 }
19
20 void Bee::SubLifes() {
21   if(_lifes > 1)
22     _lifes--;
```

```
23      else{
24          _lifes = 0;
25          Game& game = Game::Instance();
26          game.play(false);
27      }
28  }
```