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 8

1.4 Результаты тестирования 8

Приложение А **9**

Game.h 9

GameItem.h 12

Bee.h 13

Cloud.h 14

FlyingObj.h 14

Bell.h 15

Bulletgen.h 15

Enemy.h 16

Initialization.h 19

MainWindow.h 21

Observer.h 21

GameItem.cpp 22

Bee.cpp 24

Bell.cpp 24

Cloud.cpp 25

Flyingobj.cpp 25

Observer.cpp 26

main.cpp 26

MainWindow.cpp 27

Settings.ini 29

Постановка задачи

Рассмотреть паттерн проектирования наблюдатель. Реализовать обработку нажатий клавиш клавиатуры с использованием механизма сигналов и слотов библиотеки *Qt*. Доработать и реализовать логику игры (при нажатии клавиш объекты должны изменять положение).

1 Основная часть

1.1 Общая идея решения задачи

В работе была реализована обработка движения пчелы при нажатии на клавиатуру с помощью слотов *keyReleaseEvent* и *keyPressEvent*. Была улучшена логика игры: появилась возможность убивать врагов, стрелять в облака и ловить колокольчик, выпадающий из облака. Также для более удобной разработки игры дополнительно были применены паттерны (одиночка, адаптер) и механизм таймера из последующих лабораторных работ. В проекте был применен поведенческий паттерн проектирования наблюдатель, который позволяет получать объектам оповещения информацию о состоянии уведомителя (таймера).

1.2 Структура и принципы действия

Класс *MainWindow* содержит слоты *keyReleaseEvent* и *keyPressEvent*, которые позволяют обрабатывать нажатия с клавиатуры. При нажатии на кнопки влево, вправо, вверх и вниз (обрабатывает слот *keyPressEvent*) пчела перемещается по полю в соответствии с заданным направлением. При нажатии на пробел создается экземпляр класса пули, которая летит вверх. Когда кнопка отпущена, в слоте *keyReleaseEvent* выполняется остановка движения.

В классе *Game* была реализована функция *Collide*, обрабатывающая столкновения объектов. Отдельно рассмотрено столкновение пули и врага (враг и пуля исчезают, игроку начисляются баллы), столкновение пули и облака (если из облака ранее не вылетал колокольчик, то создается экземпляр класса Bell, пуля исчезает), столкновение пули и колокольчика (изменяется тип движения колокольчика, он поделает вверх, пуля исчезает), столкновение пчелы и колокольчика (колокольчик исчезает, игроку начисляются очки).

В работе был реализован паттерн наблюдатель с использованием механизма таймера (класс *QTimer*). Класс *Notifer* - класс одиночка, который определяет методы подписки, отписки и уведомления наблюдателей. Класс наследуется от класса *QObject* и содержит макрос *Q_OBJECT*. *Notifer* включает в себя вектор подписчиков, поле *QTimer* (дополнительно фазу и период таймера, период можно задать в *INI*-файле). Содержит слот *Notify*, который уведомляет все объекты из вектора подписчиков. Класс *Observer* является чисто виртуальным классом-подписчиком, который может

получать уведомления от *Notifer*. Он содержит единственный метод *Update* для конкретного уведомителя. От класса *Observer* наследуются классы *MainWindow* (чтобы обновлять форму в соответствии с таймером), *GameItem* (чтобы пермещать объекты) и *Game* (для проверки столкновений).

В игру был добавлен класс *Bell*, который приблизит функционал игры к прототипу (класс описывает 2 типа движения в отличие от базового класса *FlyingObj*).

Был создан класс-адаптер *BulletGen* (также и одиночка), в котором был реализован механизм пистолета. Класс создает экземпляр пули, если текущее состояние кнопки - нажата, а предыдущее - отпущена, и интервал между выстрелами не должен быть меньше *shootTime*. Этот класс работает одновременно и с классом *MainWindow*, и с классом *FlyingObj*. Адаптер получает вызовы от *MainWindow* (клиента), а затем в правильном формате создает экземпляр класса *FlyingObj* (сервиса).

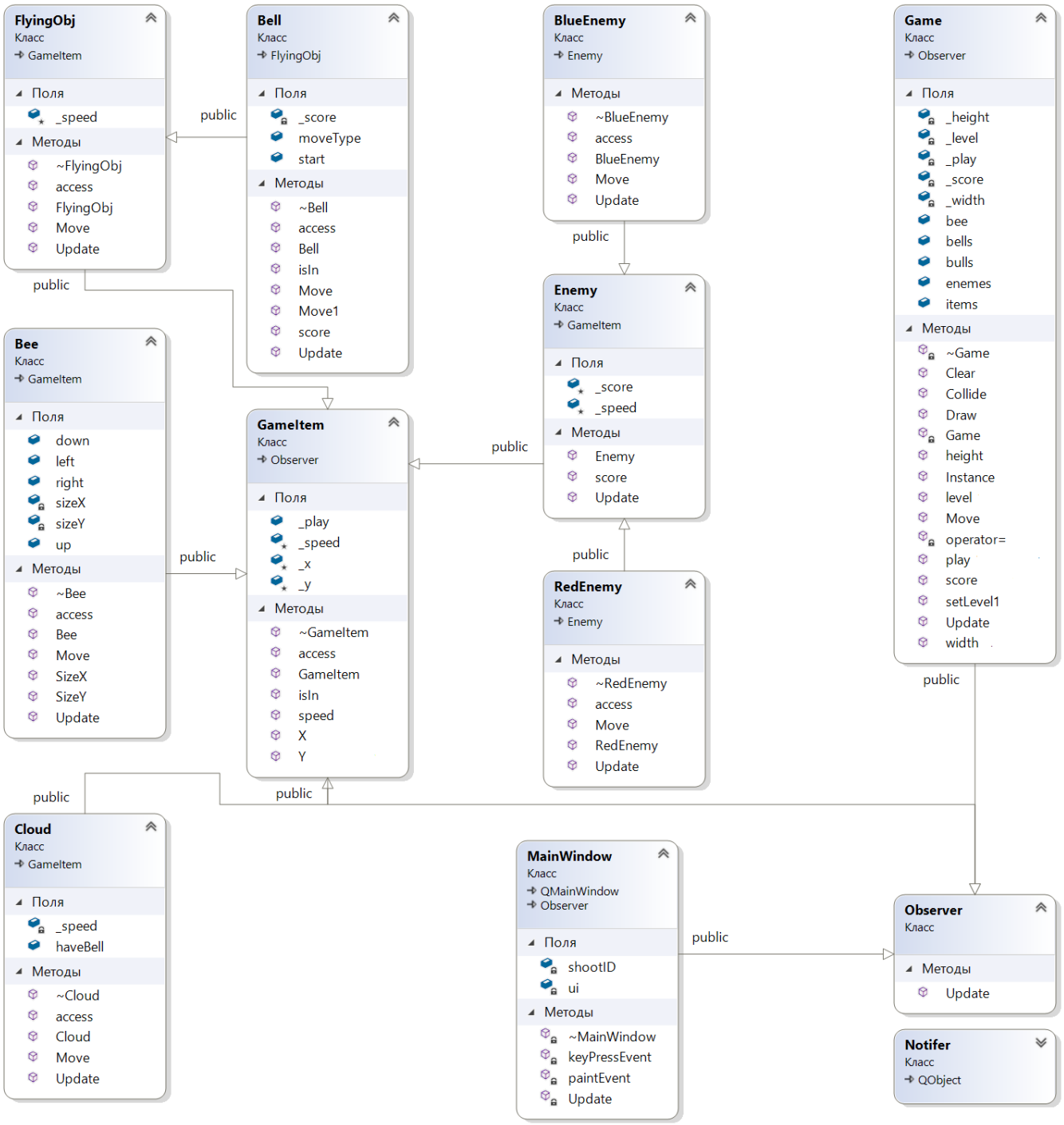

Рис. 1: Диграма классов

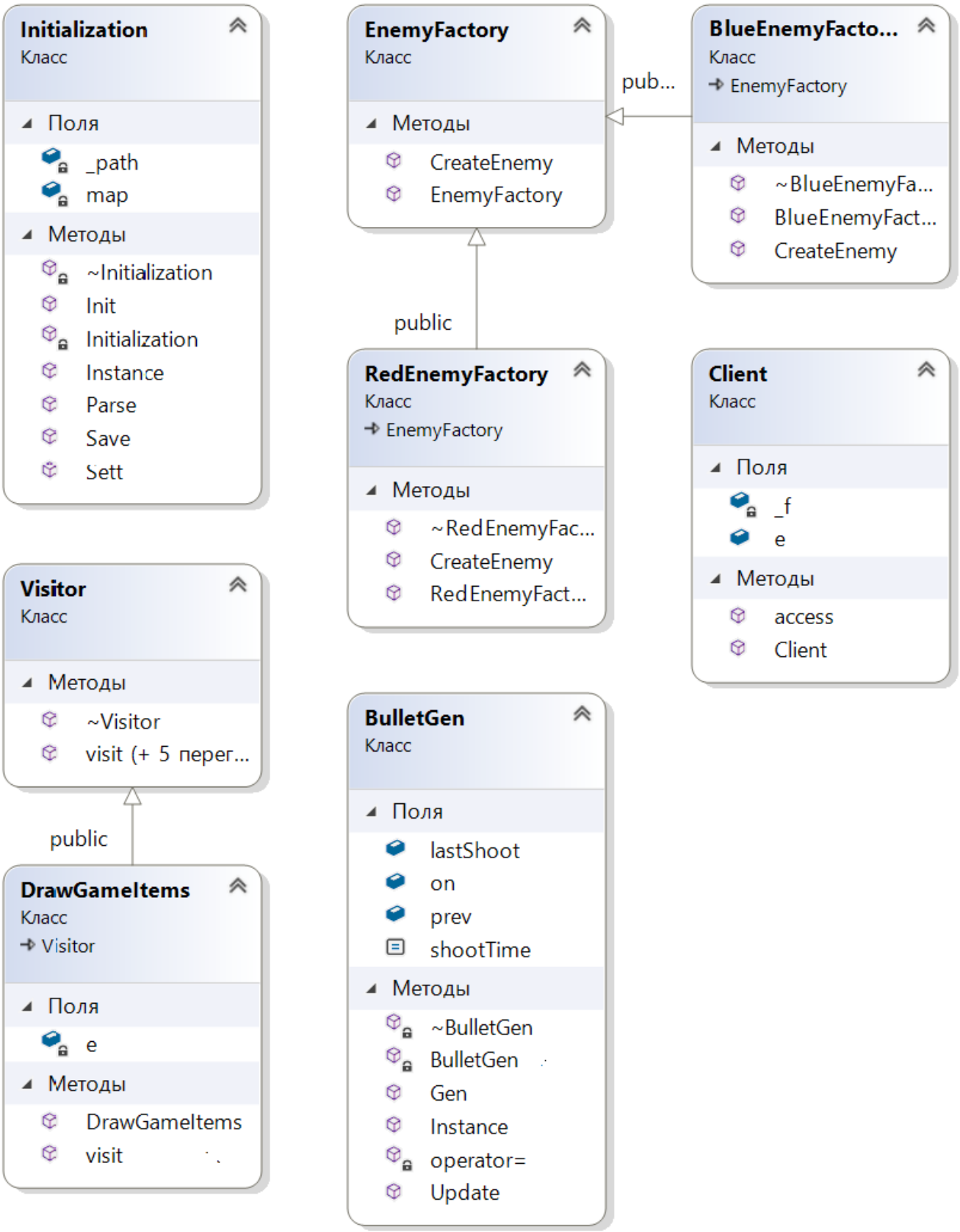

Рис. 2: Диграма классов (продолжение)

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* (убито 2 врага, получен один колокольчик):

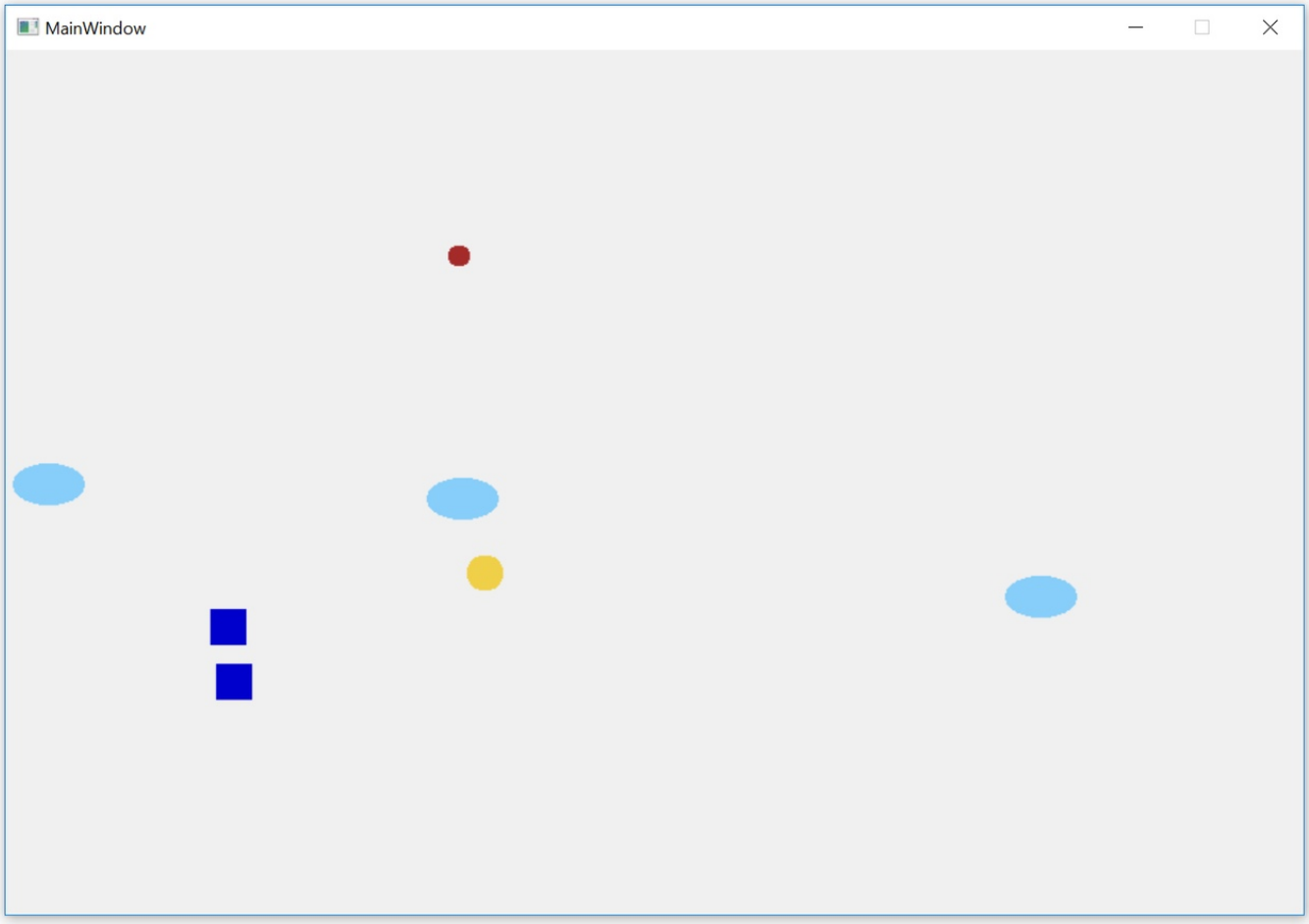

Рис. 3: Результат тестирования

Приложение А

полный код программы

A.1 - 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(){
74        for (auto b: bulls){
75          for (auto i: enemes){
76            if (b->X() > i->e->X() && b->X() + 10 < i->e->X() + 25 &&
77                b->Y() < i->e->Y() + 25 && b->Y() + 10 > 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() + 10 < i->X() + 50 &&
87                b->Y() < i->Y() + 30 && b->Y() + 10 > i->Y()){
88              if(i->haveBell){
89                i->haveBell = false;
90                bells.push_back(new Bell(i->X() + 15, 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() + 10 < i->X() + 15 &&
99                b->Y() < i->Y() + 15 && b->Y() + 10 > 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() > bee->X() && i->X() + 15 < bee->X() + 25 &&
```

```
108             i->Y() < bee->Y() + 25 && i->Y() + 15 > bee->Y()){
109           _score += i->score();
110           i->_play = false;
111         }
112       }
113     }
114     void Update(const Notifer& n) {
115       Collide();
116     }
117     void setLevel1(){
118       bee = new Bee();
119       EnemyFactory *factory = new BlueEnemyFactory;
120       for (int i = 0; i < 3; i++) {
121         Client* enemy = static_cast<Client*>(new Client(factory));
122         enemes.push_back(enemy);
123       }
124       for (int i = 0; i < 3; i++) {
125         Cloud* item = static_cast<Cloud*>(new Cloud);
126         items.push_back(item);
127       }
128       delete factory;
129     }
130     void Clear(){
131       for (auto it = bulls.begin(); it != bulls.end(); it++) {
132         if (!(*it)->isIn() || !(*it)->_play){
133           delete *it;
134           bulls.erase(it);
135           it--;
136         }
137       }
138       for (auto it = enemes.begin(); it != enemes.end(); it++) {
139         if (!(*it)->e->isIn() || !(*it)->e->_play){
140           delete *it;
141           enemes.erase(it);
142           it--;
143         }
144       }
145       for (auto it = items.begin(); it != items.end(); it++) {
146         if (!(*it)->isIn()){
147           delete *it;
148           items.erase(it);
149           it--;
150         }
151       }
152       for (auto it = bells.begin(); it != bells.end(); it++) {
153         if (!(*it)->isIn() || !(*it)->_play){
154           delete *it;
155           bells.erase(it);
156           it--;
157         }
158       }
159     }
160     static Game& Instance()
161     {
162       static Game g;
163       return g;
164     }
```

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

A.2 - GameItem.h

```
1  #pragma once
2  #include <iostream>
3  #include <QPainter>
4  #include <QMainWindow>
5  #include "observer.h"
6
7  class Bee;
8  class Cloud;
9  class FlyingObj;
10 class RedEnemy;
11 class BlueEnemy;
12 class Bell;
13
14 using namespace std;
15
16 class Visitor{
17 public:
18   virtual void visit(Bee &b) = 0;
19   virtual void visit(Cloud &c) = 0;
20   virtual void visit(FlyingObj &f) = 0;
21   virtual void visit(Bell &b) = 0;
22   virtual void visit(RedEnemy &r) = 0;
23   virtual void visit(BlueEnemy &b) = 0;
24   virtual ~Visitor() = default;
25 };
```

```
26
27 //element
28 class GameItem: public Observer
29 {
30 protected:
31   int _x, _y, _speed;
32 public:
33   bool _play;
34   GameItem(){
35     _play = true;
36     _x = 0;
37     _y = 0;
38     _speed = 0;
39   }
40   virtual void access(Visitor &v) = 0;
41   virtual ~GameItem() = 0;
42   //virtual void Move() = 0;
43   bool isIn();
44   int X(){ return _x;}
45   int Y(){ return _y;}
46   int speed() {return _speed;}
47   void X( int x ){ _x = x;}
48   void Y( int y ){ _y = y;}
49 };
50
51 //concrete visitor
52 class DrawGameItems:
53     public Visitor{
54 private:
55   QMainWindow *e;
56 public:
57   DrawGameItems(QMainWindow *event): e(event){}
58   void visit(Bee &b)override ;
59   void visit(Cloud &c) override;
60   void visit(FlyingObj &f) override;
61   void visit(Bell &b) override;
62   void visit(RedEnemy &r) override;
63   void visit(BlueEnemy &b) override;
64 };
```

A.3 - Bee.h

```
1 #pragma once
2 #include "GameItem.h"
3 #include "initialization.h"
4
5 class Bee :
6   public GameItem
7 {
8 private:
9   int sizeX;
10  int sizeY;
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        sizeX = 25;
28        sizeY = 25;
29        right= false;
30        left= false;
31        up= false;
32        down= false;
33      }
34      int SizeX(){
35        return sizeX;
36      }
37      int SizeY(){
38        return sizeY;
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  };
```

A.4 - Cloud.h

```
1  #pragma once
2  #include "GameItem.h"
3  #include "initialization.h"
4
5  class Cloud :
6    public GameItem
7  {
8    int _speed;
9  public:
10   int haveBell;
11   Cloud();
12   ~Cloud() override;
13   void access(Visitor &v) override;
14   void Update(const Notifer& n) override;
15   void Move();
16 };
```

A.5 - FlyingObj.h

```
1  #pragma once
2  #include "GameItem.h"
3  #include "initialization.h"
4
5  //class visitor
6  class FlyingObj :
```

```
 7    public GameItem
 8  {
 9  protected:
10    int _speed;
11  public:
12    FlyingObj();
13    ~FlyingObj();
14    void access(Visitor &v) override {
15      v.visit(*this);
16    }
17    void Update(const Notifer& n) override;
18    void Move(){
19      _y -= _speed;
20    }
21  };
```

A.6 - Bell.h

```
 1  #pragma once
 2  #include "flyingobj.h"
 3  #include <cmath>
 4
 5  //class visitor
 6  class Bell :
 7    public FlyingObj
 8  {
 9    int _score;
10  public:
11    int moveType;
12    int start;
13    Bell(int x, int y);
14    ~Bell(){
15      Notifer::Instance().Unsubscribe(this);
16    }
17    void access(Visitor &v) override {
18      v.visit(*this);
19    }
20    void Move(const Notifer& n){
21      _x += 0.2 * (n.getStage() - start) * cos(1.5);
22      _y -= (0.06 * (n.getStage() - start) * sin(1.5) - 0.001 * (n.
       getStage() - start)*(n.getStage() - start));
23    }
24    void Move1(const Notifer& n){
25      _y -= (0.06 * (n.getStage() - start) - 0.001 * (n.getStage() -
       start)*(n.getStage() - start));
26    }
27    void Update(const Notifer& n) override{
28      if (!moveType)
29        Move(n);
30      else
31        Move1(n);
32    }
33    bool isIn();
34    int score() const { return _score; }
35  };
```

A.7 - Bulletgen.h

```
1  #pragma once
2  #include "flyingobj.h"
3  #include "Game.h"
4
5  class BulletGen{
6  public:
7    bool on, prev;
8    int lastShoot;
9    static const int shootTime = 5;
10   static BulletGen& Instance()
11   {
12     static BulletGen b;
13     return b;
14   }
15   void Gen(){
16     if (on && !prev){
17       Game& game = Game::Instance();
18       FlyingObj* item = new FlyingObj();
19       game.bulls.push_back(item);
20       prev = true;
21     }
22   }
23   void Update(const Notifer& n){
24     if(n.getStage() - lastShoot >= BulletGen::shootTime){
25       lastShoot = n.getStage();
26       Gen();
27     }
28   }
29 private:
30   BulletGen(){
31     on = false;
32     prev = false;
33     lastShoot = -1;
34   }
35   ~BulletGen(){}
36   BulletGen(BulletGen const&) = delete;
37   BulletGen& operator= (BulletGen const&) = delete;
38 };
```

A.8 - Enemy.h

```
1  #pragma once
2  #include "GameItem.h"
3  #include "initialization.h"
4
5  //Abstract base product
6  class Enemy :
7    public GameItem
8  {
9  protected:
10   int _score;
11   int _speed;
12 public:
13   Enemy() {
14     Initialization& ini = Initialization::Instance();
15     _x = ini.Sett("setcoord/enemy_x") + + rand()%70;
16     if (_x < 0){
17       _x = 0;
```

```
18      }
19      _y = ini.Sett("setcoord/enemy_y") + rand()%70;
20      if (_y < 0){
21        _y = 0;
22      }
23      _score = 0;
24      _speed = 0;
25    }
26    void Update(const Notifer& n) override{}
27    int score() const { return _score; }
28  };
29  //Concrete product type Red
30  class RedEnemy :
31    public Enemy
32  {
33  public:
34    RedEnemy(){
35      Initialization& ini = Initialization::Instance();
36      _score = ini.Sett("setscore/RedEnemy");
37      if (_score < 0){
38        _score = 0;
39      }
40      _speed = ini.Sett("setspeed/RedEnemy");
41      if (_speed < 0){
42        _speed = 0;
43      }
44      Notifer::Instance().Subscribe(this);
45    }
46    ~RedEnemy() override {
47      Notifer::Instance().Unsubscribe(this);
48    }
49    void Update(const Notifer& n){
50      Move();
51    }
52    void access(Visitor &v) override {
53      v.visit(*this);
54    }
55    void Move() {
56      _x += _speed;
57      _y += _speed;
58    }
59  };
60  //Concrete product type Blue
61  class BlueEnemy :
62    public Enemy
63  {
64  public:
65    BlueEnemy(){
66      Initialization& ini = Initialization::Instance();
67      _score = ini.Sett("setscore/BlueEnemy");
68      if (_score < 0){
69        _score = 0;
70      }
71      _speed = ini.Sett("setspeed/BlueEnemy");
72      if (_speed < 0){
73        _speed = 0;
74      }
```

```
75      Notifer::Instance().Subscribe(this);
76    }
77    ~BlueEnemy() override {
78      Notifer::Instance().Unsubscribe(this);
79    }
80    void Update(const Notifer& n){
81      Move();
82    }
83    void access(Visitor &v) override {
84      v.visit(*this);
85     }
86    void Move() {
87      _x -= _speed;
88      _y += _speed;
89    }
90  };
91  //Abstract factory
92  class EnemyFactory {
93  public:
94    EnemyFactory(){}
95    virtual Enemy* CreateEnemy() = 0;
96    //virtual ~EnemyFactory() = 0;
97  };
98  //Conctete factory type Red
99  class RedEnemyFactory :
100   public EnemyFactory {
101 public:
102   RedEnemyFactory() {}
103   Enemy* CreateEnemy() override{
104     return new RedEnemy;
105   }
106   ~RedEnemyFactory() {}
107 };
108 //Conctete factory type Blue
109 class BlueEnemyFactory :
110   public EnemyFactory {
111 public:
112   BlueEnemyFactory() {}
113   Enemy* CreateEnemy() override {
114     return new BlueEnemy;
115   }
116   ~BlueEnemyFactory() {}
117 };
118 //Working through abstract interface
119 class Client {
120   EnemyFactory *_f;
121 public:
122   Enemy *e;
123   //QVector<Enemy*> army;
124   Client(EnemyFactory *f): _f(f) {
125 //     for(int i = 0; i < 3; i++){
126 //       Enemy *e = _f->CreateEnemy();
127 //       army.push_back(e);
128 //     }
129     e = _f->CreateEnemy();
130   }
131   void access(Visitor &v)
```

```
132     {
133 //      for(auto e: army)
134 //        e->access(v);
135       e->access(v);
136     }
137 };
```

A.9 - Initialization.h

```
  1 #pragma once
  2 #include <QCoreApplication>
  3 #include <QSettings>
  4 #include <QString>
  5 #include <QTextStream>
  6 #include <QMap>
  7 #include <QFile>
  8 using namespace std;
  9
 10 class Initialization
 11 {
 12   QString _path;
 13   QMap<QString, QString> map;
 14   Initialization(){}
 15   ~Initialization(){}
 16 public:
 17   static Initialization& Instance()
 18   {
 19     static Initialization i;
 20     return i;
 21   }
 22   int Sett(QString key, const int & val = 0){
 23     if (map.find(key) == map.end()){
 24       return val;
 25     }
 26     return map.value(key).toInt();
 27   }
 28   void Save(const QString &section,
 29         const QString &variable, int value){
 30     QSettings sett(_path, QSettings::IniFormat);
 31     QFile file(_path);
 32     if ((file.open(QIODevice::ReadWrite)))
 33     {
 34       QString line;
 35       QString comment;
 36       bool flag = false;
 37       QTextStream stream( &file );
 38       do {
 39         line = file.readLine();
 40         if (line.contains(section, Qt::CaseSensitive) && !line.isEmpty
    () && line[0] != ';')
 41         {
 42           stream << line;
 43           line = file.readLine();
 44           while(!line.isEmpty() && line != "\n" && line[0] != ';'){
 45             if (line.contains(variable, Qt::CaseSensitive) && !line.
    isEmpty() && line[0] != ';')
 46             {
 47               flag = true;
```

```
48                line = variable + "=" + QString::number(value) + '\n';
49              }
50              stream << line;
51              line = file.readLine();
52            }
53            if (!flag){
54              flag = true;
55              line = variable + "=" + QString::number(value) + "\n\n";
56            }
57          }
58          stream << line;
59        } while (!line.isEmpty());
60        if (!flag){
61          stream << "\n\n[" + section + "]" + '\n';
62          stream << variable + "=" + QString::number(value) + "\n";
63        }
64        file.resize(0);
65        file.close();
66      }
67    }
68    QString Parse(QString &str, QString &sec){
69      //ignore comments
70      if (str[0] == ';')
71        return "";
72      //init section
73      if (str[0] == '['){
74        int i = 1;
75        sec="";
76        while (str[i] != ']'){
77          sec.append(str[i]);
78          i++;
79        }
80        sec.append('/');
81        return sec;
82      }
83      //init variable
84      int i = 0;
85      QString key="";
86      while(str[i] != 0 && str[i] != '='){
87        if (str[i] == ' ')
88          i++;
89        else{
90          key.append(str[i]);
91          i++;
92        }
93      }
94      QString val="";
95      i++;
96      while(str[i] != 0 && str[i] != ';'){
97        if (str[i] == ' ')
98          i++;
99        else{
100         val.append(str[i]);
101         i++;
102       }
103     }
104     if(val.length())
```

```
105         map.insert(sec + key, val);
106       return sec;
107     }
108     void Init(QString p)
109     {
110       _path = QCoreApplication:: applicationDirPath();
111       _path.append("/" + p);
112       QSettings sett(_path, QSettings::IniFormat);
113       QFile file(_path);
114       if ((file.open(QIODevice::ReadOnly)))
115       {
116         QString str, sec;
117         while(!file.atEnd())
118         {
119           str = file.readLine();
120           sec = Parse(str, sec);
121         }
122         file.close();
123       }
124     }
125 };
```

A.10 - MainWindow.h

```
1  #pragma once
2  #include <QMainWindow>
3  #include <QKeyEvent>
4  #include "GameItem.h"
5  #include "observer.h"
6
7  namespace Ui {
8  class MainWindow;
9  }
10
11 class MainWindow : public QMainWindow, public Observer
12 {
13   Q_OBJECT
14
15 public:
16   explicit MainWindow(QWidget *parent = nullptr);
17   ~MainWindow();
18   void paintEvent(QPaintEvent *event);
19   void Update(const Notifer& n) {
20     update();
21   }
22 private:
23   Ui::MainWindow *ui;
24   int shootID = 0;
25   QSet<Qt::Key> keysPressed;
26 public slots:
27   void keyReleaseEvent(QKeyEvent*);
28   void keyPressEvent(QKeyEvent*);
29 };
```

A.11 - Observer.h

```
1  #pragma once
2  #include <iostream>
3  #include <vector>
```

```
 4 #include <time.h>
 5 #include <sys/types.h>
 6 #include <sys/timeb.h>
 7 #include <string.h>
 8 #include <algorithm>
 9 #include <QTimer>
10 using namespace std;
11
12 class Notifer;
13
14 //GameItem is class Observer
15 class Observer
16 {
17 public:
18   virtual void Update(const Notifer&) = 0;
19 };
20
21 //observable object
22 class Notifer: public QObject
23 {
24    Q_OBJECT
25 public:
26   static Notifer& Instance()
27   {
28     static Notifer n;
29     return n;
30   }
31   void Subscribe(Observer* item);
32   void Unsubscribe(Observer* item);
33
34   int getStage() const
35   {
36     return stage;
37   }
38   int getPeriod() const
39   {
40     return period;
41   }
42 public slots:
43   void Notify();
44 protected:
45   vector<Observer*> _items;
46 private:
47   Notifer();
48   ~Notifer();
49   Notifer(Notifer const&) = delete;
50   Notifer& operator= (Notifer const&) = delete;
51   QTimer *timer;
52   int stage;
53   int period;
54 };
```

A.12 - GameItem.cpp

```
1 #pragma once
2 #include "Game.h"
3
4 bool GameItem::isIn(){
```

```
5     Game& game = Game::Instance();
6     return (0 < _x && _x < game.width() && 0 <_y && _y <game.height() );
7   }
8
9   GameItem::~GameItem(){}
10
11  void DrawGameItems::visit(Bee &b){
12    QPainter painter(e);
13    QColor yellow("#f0d048");
14    Qt::BrushStyle style = Qt::SolidPattern;
15    QBrush brush(yellow, style);
16    painter.setBrush(brush);
17    painter.setPen(Qt::NoPen);
18    painter.drawEllipse(b.X(),b.Y(), b.SizeX(),b.SizeY());
19    painter.end();
20  }
21
22  void DrawGameItems::visit(Cloud &c){
23    QPainter painter(e);
24    QColor gray("#87CEFA");
25    Qt::BrushStyle style = Qt::SolidPattern;
26    QBrush brush(gray, style);
27    painter.setBrush(brush);
28    painter.setPen(Qt::NoPen);
29    painter.drawEllipse(c.X(),c.Y(), 50,30);
30    painter.end();
31  }
32
33  void DrawGameItems::visit(FlyingObj &f){
34    QPainter painter(e);
35    QColor gray("#A52A2A");
36    Qt::BrushStyle style = Qt::SolidPattern;
37    QBrush brush(gray, style);
38    painter.setBrush(brush);
39    painter.setPen(Qt::NoPen);
40    painter.drawEllipse(f.X(),f.Y(), 10,10);
41    painter.end();
42  }
43
44  void DrawGameItems::visit(Bell &b){
45    QPainter painter(e);
46    QColor gray("#A52A2A");
47    Qt::BrushStyle style = Qt::SolidPattern;
48    QBrush brush(gray, style);
49    painter.setBrush(brush);
50    painter.setPen(Qt::NoPen);
51    painter.drawEllipse(b.X(),b.Y(), 15,15);
52    painter.end();
53  }
54
55  void DrawGameItems::visit(RedEnemy &r){
56    QPainter painter(e);
57    QColor red("#ff00ff");
58    Qt::BrushStyle style = Qt::SolidPattern;
59    QBrush brush(red, style);
60    painter.setBrush(brush);
61    painter.setPen(Qt::NoPen);
```

```
62    painter.drawRect(r.X(),r.Y(), 25,25);
63    painter.end();
64  }
65
66  void DrawGameItems::visit(BlueEnemy &b){
67    QPainter painter(e);
68    QColor blue("#0000CD");
69    Qt::BrushStyle style = Qt::SolidPattern;
70    QBrush brush(blue, style);
71    painter.setBrush(brush);
72    painter.setPen(Qt::NoPen);
73    painter.drawRect(b.X(),b.Y(), 25,25);
74    painter.end();
75  }
```

A.13 - 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 }
```

A.14 - Bell.cpp

```
1  #include "bell.h"
2  #include "Game.h"
3
4  Bell::Bell(int x, int y){
5    Initialization& ini = Initialization::Instance();
6    Game& game = Game::Instance();
7    _speed = ini.Sett("setspeed/Bell");
8    if (_speed < 0){
9      _speed = 0;
10   }
11   _score = ini.Sett("setscore/Bell");
12   if (_score < 0){
13     _score = 0;
14   }
15   _x = x;
16   _y = y;
17   Notifer::Instance().Subscribe(this);
18   start = Notifer::Instance().getStage();
19   moveType = 0;
20 }
```

```
21
22 bool Bell::isIn() {
23   Game& game = Game::Instance();
24   return (0 < _x && _x < game.width() && game.height() > _y);
25 }
```

A.15 - Cloud.cpp

```
 1 #include "Cloud.h"
 2 #include "observer.h"
 3
 4 Cloud::Cloud(){
 5   Initialization& ini = Initialization::Instance();
 6   _speed = ini.Sett("setspeed/Cloud");
 7   if (_speed < 0){
 8     _speed = 0;
 9   }
10   int x = ini.Sett("setgame/width", 900);
11   if (x < 0){
12     x = 900;
13   }
14   x -= 50;
15   int y = ini.Sett("setgame/height", 700);
16   if (y < 0){
17     y = 700;
18   }
19   y /= 2;
20   _x = rand()%x;
21   _y = rand()%y;
22   haveBell = true;
23   Notifer::Instance().Subscribe(this);
24 }
25
26 Cloud::~Cloud() {
27   Notifer::Instance().Unsubscribe(this);
28 }
29
30 void Cloud::access(Visitor &v) {
31   v.visit(*this);
32 }
33 void Cloud::Move(){
34   _y+=_speed;
35 }
36 void Cloud::Update(const Notifer& n){
37   Move();
38 }
```

A.16 - Flyingobj.cpp

```
 1 #include "flyingobj.h"
 2 #include "Game.h"
 3
 4 FlyingObj::FlyingObj(){
 5   Initialization& ini = Initialization::Instance();
 6   Game& game = Game::Instance();
 7   _speed = ini.Sett("setspeed/FlyingObj");
 8   if (_speed < 0){
 9     _speed = 0;
```

```
10    }
11    _x = game.bee->X() + game.bee->SizeX()/2 - 5;
12    _y = game.bee->Y() - game.bee->SizeY()/2;
13    Notifer::Instance().Subscribe(this);
14  }
15
16  FlyingObj::~FlyingObj(){
17    Notifer::Instance().Unsubscribe(this);
18  }
19
20  void FlyingObj::Update(const Notifer &n){
21    Move();
22  }
```

A.17 - Observer.cpp

```
 1  #include "observer.h"
 2  #include "GameItem.h"
 3  #include "initialization.h"
 4
 5  Notifer::Notifer() {
 6    timer = new QTimer();
 7    connect(timer, SIGNAL(timeout()), this, SLOT(Notify()));
 8    stage = 0;
 9    Initialization& ini = Initialization::Instance();
10    period = ini.Sett("timer/peroid", 30);
11    timer->setInterval(period);
12    timer->start();
13  }
14
15  Notifer::~Notifer() {
16    delete timer;
17  }
18
19  void Notifer::Subscribe(Observer *item)
20  {
21    _items.push_back(item);
22  }
23
24  void Notifer::Unsubscribe(Observer *item)
25  {
26    auto it = find(_items.begin(), _items.end(), item);
27    if ( it != _items.end() ){
28      _items.erase(it);
29    }
30  }
31
32  void Notifer::Notify(){
33    stage++;
34    for(auto item: _items)
35      item->Update(*this);
36  }
```

A.18 - main.cpp

```
1  #include <QCoreApplication>
2  #include <iostream>
3  #include <QSettings>
```

```
 4 #include <QVariant>
 5 #include <QString>
 6 #include "Game.h"
 7 #include "mainwindow.h"
 8 #include <QApplication>
 9 #include "Initialization.h"
10 #include <ctime>
11
12 class QPaintEvent;
13 using namespace std;
14
15 int main(int argc, char *argv[])
16 {
17     QApplication a(argc, argv);
18     srand((unsigned int)time(NULL));//helps to generate random coords of
       objects
19     Initialization& ini = Initialization::Instance();
20     ini.Init("settings.ini");
21     Game& game = Game::Instance();
22     MainWindow w;
23     w.show();
24     return a.exec();
25 }
```

A.19 - MainWindow.cpp

```
 1 #include "mainwindow.h"
 2 #include "ui_mainwindow.h"
 3 #include "Game.h"
 4 #include "observer.h"
 5 #include "bulletgen.h"
 6
 7 MainWindow::MainWindow(QWidget *parent) :
 8     QMainWindow(parent),
 9     ui(new Ui::MainWindow)
10 {
11     ui->setupUi(this);
12     Game& game = Game::Instance();
13     this->setFixedSize(game.width(),game.height());
14     Notifer::Instance().Subscribe(this);
15     shootID = 0;
16 }
17
18 MainWindow::~MainWindow()
19 {
20     delete ui;
21 }
22
23 void MainWindow::paintEvent(QPaintEvent *event)
24 {
25     Game& game = Game::Instance();
26     game.Draw(this);
27 }
28
29 void MainWindow::keyPressEvent(QKeyEvent * event)
30 {
31     Game& game = Game::Instance();
32
```

```
33    switch(event->key()){
34
35    case Qt::Key_Up:
36      game.bee->up = true;
37      break;
38
39    case Qt::Key_Right:
40      game.bee->right = true;
41      break;
42
43    case Qt::Key_Left:
44      game.bee->left = true;
45      break;
46
47    case Qt::Key_Down:
48      game.bee->down = true;
49      break;
50
51    case Qt::Key_Space:
52      //if(!event->isAutoRepeat()){
53        BulletGen& gen = BulletGen::Instance();
54        gen.prev = gen.on;
55        gen.on = true;
56        gen.Update(Notifer::Instance());
57      //}
58      break;
59    }
60    game.bee->Move();
61    update();
62  }
63
64  void MainWindow::keyReleaseEvent(QKeyEvent *event){
65
66    Game& game = Game::Instance();
67
68    switch(event->key()){
69
70    case Qt::Key_Up:
71      game.bee->up = false;
72      break;
73
74    case Qt::Key_Right:
75      game.bee->right = false;
76      break;
77
78    case Qt::Key_Left:
79      game.bee->left = false;
80      break;
81
82    case Qt::Key_Down:
83      game.bee->down = false;
84      break;
85
86    case Qt::Key_Space:
87      BulletGen& gen = BulletGen::Instance();
88      //if(!event->isAutoRepeat()){
89        gen.on = false;
90      //}
```

```
91          break;
92      }
93      game.bee->Move();
94  }
```

A.20 - Settings.ini

```
 1  ;here comes the game config
 2  [setgame]
 3  width=900
 4  height=600
 5  score=660
 6  level=1
 7
 8  ;see the score for game items
 9  [setscore]
10  RedEnemy=100
11  BlueEnemy=200
12  Cloud=20
13  Bell=20
14
15  [setspeed]
16  RedEnemy=2
17  BlueEnemy=2
18  FlyingObj=20
19  Bee=10
20  Cloud=1
21  Bell=1
22
23  [setcoord]
24  bee_x=450
25  bee_y=350
26  enemy_x=450
27  enemy_y=0
28
29  [timer]
30  peroid=30
31
32  [logs]
33  no logs=1
```