

Министерство цифрового развития, связи и массовых коммуникаций Российской  
Федерации  
ФГБОУ высшего образования  
"Сибирский Государственный Университет Телекоммуникаций и  
Информатики" (СибГУТИ)  
Кафедра прикладной математики и кибернетики

Курсовая работа  
по дисциплине «Объектно-ориентированное программирование»

Тема: Игра «Жизнь»

Выполнил: студент 2 курса группы ИП-012

Маланов Роман Игоревич

Проверил: доцент кафедры ПМиК

Ситняковская Е.И.

Новосибирск, 2021

# Содержание

<b>1</b>	<b>Введение</b>	<b>2</b>
1.1	Задача: . . . . .	2
1.2	Описание игры: . . . . .	2
<b>2</b>	<b>Описание используемых алгоритмов и функций</b>	<b>2</b>
<b>3</b>	<b>Код программы</b>	<b>4</b>
3.1	Листинг: . . . . .	4
3.2	Скриншоты с примером работы: . . . . .	18

# 1 Введение

## 1.1 Задача:

Написать программу для игры в "Пятнашки".

## 1.2 Описание игры:

Игра в 15 (пятнашки) – головоломка, придуманная Ноем Чепмэном в 1878 году. Представляет собой набор одинаковых квадратных костяшек с нанесёнными числами, заключённых в квадратную коробку. Длина стороны коробки в четыре раза больше длины стороны костяшек для набора из 15 элементов, соответственно в коробке остаётся незаполненным одно квадратное поле. Цель игры — перемещая костяшки по коробке, добиться упорядочивания их по номерам.

## 2 Описание используемых алгоритмов и функций

Программа написана на языке Си под платформу Linux с использованием ncurses, библиотеки для управления вводом-выводом в терминал.

Программа создаёт в терминале 16 клеток (фишек), заполненных цифрами от 1 до 15 и одной пустой. Управление фишками происходит с помощью мыши. Нажатие на фишку сдвигает её, если рядом с ней есть пустая клетка.

- В функции `main.c` происходит создание всех блоков, настройка параметров и отслеживание нажатий клавиш и мыши.
- Текущий порядок фишек хранится в массиве `numbers`.
- Для хранения фишек, их координат и содержащихся в них чисел используется структура `struct _cell`, определённая как тип `Cell`.
- Функция `randomize()` перемешивает числа в массиве `numbers`. Для перемешивания фишек используется клавиша 'r', перемешивание происходит до тех пор, пока не получаем разрешимую комбинацию (ровно половина возможных комбинаций неразрешима). Проверку на разрешимость текущего порядка фишек осуществляет функция `is_solvable()` (проверяет чётность текущего порядка фишек, см.

Пятнашки: Математическое описание).

- Функции `get_coord()` и `get_cell_idx()` взаимно обратны и используются для получения координат фишки по индексам и для получения индекса по известным координатам соответственно.

- Функция `get_motion()` проверяет, есть ли рядом с данной фишкой свободная клетка. Возвращает тип `Motion`, направление, в котором можно сместить фишку.
- Функция `make_movement` смещает фишку на пустую клетку (меняет местами числа в массиве `numbers`).
- Функция `check_win()` проверяет, получил ли игрок правильное положение фишек (проверяет возрастание чисел в массиве `numbers` и положение пустой клетки). Проверка происходит только когда фишка с цифрой '1' находится в левом верхнем углу, а пустая клетка в правом нижнем.

## 3 Код программы

### 3.1 Листинг:

language: C++

Creature.hpp

```
1  #ifndef CREATURE_HPP
2  #define CREATURE_HPP
3
4  #include <SFML/Graphics.hpp>
5
6  enum class CreatureType { Hunter, Prey, Plant };
7
8  class Creature {
9      int m_idxX;
10     int m_idxY;
11
12     sf::Color m_color;
13
14     float m_probability;
15
16     char* m_name;
17
18 public:
19     Creature(int, int);
20
21     virtual CreatureType getType() = 0;
22
23     void setX(int);
24     int getX();
25
26     void setY(int);
27     int getY();
28
29     void setPosition(int, int);
30
31     void setColor(sf::Color);
32     virtual sf::Color getColor();
33
34     void setProb(float);
35     virtual float getProb();
36
37     virtual void step();
38
39     virtual int isDead();
40
41     int getSpeed();
42
43     virtual const char* getTypeName();
44 };
45
46 #endif // #ifndef CREATURE_HPP
```

Plant.hpp

```
1  #ifndef PLANT_HPP
```

```

2  #define PLANT_HPP
3
4  #include "Creature.hpp"
5
6  #include <SFML/Graphics.hpp>
7
8  class Plant : public Creature {
9      int m_lifetime = 6;
10
11  public:
12      Plant(int, int);
13
14      virtual CreatureType getType();
15
16      virtual const char* getTypeName();
17
18      int getLifetime();
19
20      virtual void step();
21
22      virtual int isDead();
23
24      virtual int getSpeed(); // -> 0
25 };
26
27 #endif // #ifndef PLANT_HPP

```

## LivingCreature.hpp

```

1  #ifndef LIVINGCREATURE_HPP
2  #define LIVINGCREATURE_HPP
3
4  #include "Creature.hpp"
5
6  #include <SFML/Graphics.hpp>
7
8  class LivingCreature : public Creature {
9  public:
10     int m_hunger = 15;
11     int m_eaten = 0;
12     const int m_hungerStep = -2;
13     const int m_gaveBirthStep = -4;
14     const int m_eatStep = 2;
15
16     uint8_t delta = 20;
17
18     LivingCreature(int x, int y);
19
20     void step();
21
22     virtual int isDead();
23
24     void eat();
25
26     void giveBirth();
27
28     int getCountEaten();
29 };
30

```

```
31 #endif // #ifndef LIVINGCREATURE_HPP
```

## Prey.hpp

```
1 #ifndef PREY_HPP
2 #define PREY_HPP
3
4 #include "LivingCreature.hpp"
5
6 class Prey : public LivingCreature {
7 public:
8     Prey(int x, int y);
9
10    virtual CreatureType getType();
11
12    virtual const char* getTypeName();
13
14    virtual int getSpeed(); //-> 1
15 };
16
17 #endif // #ifndef PREY_HPP
```

## Hunter.hpp

```
1 #ifndef HUNTER_HPP
2 #define HUNTER_HPP
3
4 #include "LivingCreature.hpp"
5
6 class Hunter : public LivingCreature {
7 public:
8     Hunter(int x, int y);
9
10    virtual CreatureType getType();
11
12    virtual const char* getTypeName();
13
14    virtual int getSpeed(); //-> 1 or 2
15 };
16
17 #endif // #ifndef HUNTER_HPP
```

## Game.hpp

```
1 #ifndef GAME_HPP
2 #define GAME_HPP
3
4 #include "Creature.hpp"
5 #include "Hunter.hpp"
6 #include "LivingCreature.hpp"
7 #include "Plant.hpp"
8 #include "Prey.hpp"
9
10 #include <SFML/Graphics.hpp>
11
12 #include <iostream>
13 #include <random>
14
15 class Game {
```

```

16     int m_Xsize;
17     int m_Ysize;
18
19     const char* m_gameTitle = "Game_of_life ";
20
21     sf::RenderWindow m_window;
22
23     int m_Nrow = 100; // square count per column or row
24
25     int m_borderSize = 2;
26
27     int m_squareSize;
28
29     typedef Creature** CreatureArray;
30
31     int m_sizeArrayCreatures = m_Nrow*m_Nrow/2;
32     int m_countCreatures = 0;
33
34     CreatureArray* m_board;
35
36     int m_steps = 0;
37
38     // probabilities
39
40     int m_probSizeArray;
41     int* m_probArray;
42
43     typedef enum CreatureInt {
44         CreatureInt_Hunter = 1,
45         CreatureInt_Prey = 2,
46         CreatureInt_Plant = 3
47     } CreatureInt;
48
49     void fillProbArray();
50
51     CreatureType chooseRandomCreatureType();
52
53 public:
54     Game(int Xsize, int Ysize);
55
56     Game(int Xsize);
57
58     Game();
59
60     sf::Vector2f getCoord(int, int);
61
62     sf::RectangleShape createShape(Creature* creature);
63
64     int isIndexAvailable(int x, int y);
65
66     int isSideIndexAvailable();
67
68     int isBoardFull();
69
70     Creature* getCreature(CreatureType type, int onSide = 0);
71
72     Creature* getRandomCreature();
73

```



```

74     void initializeArray();
75
76     void updateEnvironment();
77
78     void lifeStep();
79
80     int countHunters();
81
82     int countPreys();
83
84     void run();
85 };
86
87 #endif // #ifndef GAME_HPP

```

Source-файлы:

Creature.cpp

```

1  #include "Creature.hpp"
2
3  Creature::Creature(int x, int y) : m_idxX(x), m_idxY(y)
4  {
5  }
6
7  void Creature::setX(int newX)
8  {
9      m_idxX = newX;
10 }
11
12 int Creature::getX()
13 {
14     return m_idxX;
15 }
16
17 void Creature::setY(int newY)
18 {
19     m_idxY = newY;
20 }
21
22 int Creature::getY()
23 {
24     return m_idxY;
25 }
26
27 void Creature::setPosition(int newX, int newY)
28 {
29     m_idxX = newX;
30     m_idxY = newY;
31 }
32
33 void Creature::setColor(sf::Color color)
34 {
35     m_color = color;
36 }
37
38 sf::Color Creature::getColor()
39 {
40     return m_color;
41 }

```

```

42
43 float Creature::getProb()
44 {
45     return m_probability;
46 }
47
48 void Creature::setProb(float prob)
49 {
50     m_probability = prob;
51 }
52
53 void Creature::step()
54 {
55 }
56
57 int Creature::isDead()
58 {
59     return 0;
60 }
61
62 int Creature::getSpeed()
63 {
64     return 1;
65 }
66
67 const char* Creature::getTypeName()
68 {
69     return "???";
70 }

```

## Plant.cpp

```

1  #include "Plant.hpp"
2  #include "Creature.hpp"
3
4  Plant::Plant(int x, int y) : Creature(x, y)
5  {
6      setColor(sf::Color::Green);
7      setProb(0.6f);
8  }
9
10 CreatureType Plant::getType()
11 {
12     return CreatureType::Plant;
13 }
14
15 const char* Plant::getTypeName()
16 {
17     return "Plant";
18 }
19
20 int Plant::getLifetime()
21 {
22     return m_lifetime;
23 }
24
25 void Plant::step()
26 {
27     Creature::step();

```

```

28     —m_lifetime;
29 }
30
31 int Plant::isDead()
32 {
33     if (m_lifetime <= 0)
34         return 1;
35     return 0;
36 }
37
38 int Plant::getSpeed()
39 {
40     return 0;
41 }

```

## LivingCreature.cpp

```

1  #include "LivingCreature.hpp"
2
3  LivingCreature::LivingCreature(int x, int y) : Creature(x, y)
4  {
5  }
6
7  void LivingCreature::step()
8  {
9      Creature::step();
10     m_hunger += m_hungerStep;
11     sf::Color color = getColor();
12     if (color.a - delta >= delta)
13         setColor(sf::Color(color.r, color.g, color.b, color.a - delta));
14 };
15
16 int LivingCreature::isDead()
17 {
18     if (m_hunger <= 0)
19         return 1;
20     return 0;
21 }
22
23 void LivingCreature::eat()
24 {
25     m_hunger += m_eatStep;
26     ++m_eaten;
27
28     sf::Color color = getColor();
29     if (color.a + delta >= 254)
30         setColor(sf::Color(color.r, color.g, color.b, color.a + delta));
31 }
32
33 void LivingCreature::giveBirth()
34 {
35     m_hunger += m_gaveBirthStep;
36 }
37
38 int LivingCreature::getCountEaten()
39 {
40     return m_eaten;
41 }

```

## Prey.cpp

```
1  #include "Prey.hpp"
2  #include "LivingCreature.hpp"
3
4  Prey::Prey(int x, int y) : LivingCreature(x, y)
5  {
6      setColor(sf::Color::Yellow);
7      setProb(0.4f);
8  }
9
10 CreatureType Prey::getType()
11 {
12     return CreatureType::Prey;
13 }
14
15 const char* Prey::getTypeName()
16 {
17     return "Prey";
18 }
19
20 int Prey::getSpeed()
21 {
22     return 1;
23 }
```

## Hunter.cpp

```
1  #include "Hunter.hpp"
2
3  Hunter::Hunter(int x, int y) : LivingCreature(x, y)
4  {
5      setColor(sf::Color::Red);
6      setProb(0.4f);
7  }
8
9  CreatureType Hunter::getType()
10 {
11     return CreatureType::Hunter;
12 }
13
14 const char* Hunter::getTypeName()
15 {
16     return "Hunter";
17 }
18
19 int Hunter::getSpeed()
20 {
21     if (m_hunger <= 5)
22         return 2;
23     return 1;
24 }
```

## Game.cpp

```
1  #include "Game.hpp"
2
3  void Game::fillProbArray()
4  {
5      int coef = 100;
```

```

6
7     int hunterProb = Hunter(0, 0).getProb() * coef;
8     int preyProb = Prey(0, 0).getProb() * coef;
9     int plantProb = Plant(0, 0).getProb() * coef;
10    m_probSizeArray = hunterProb + preyProb + plantProb;
11
12    m_probArray = new int[m_probSizeArray];
13    for (int i = 0; i < hunterProb; ++i)
14        m_probArray[i] = CreatureInt_Hunter;
15    for (int i = 0; i < preyProb; ++i)
16        m_probArray[hunterProb + i] = CreatureInt_Prey;
17    for (int i = 0; i < plantProb; ++i)
18        m_probArray[m_probSizeArray - 1 - i] = CreatureInt_Plant;
19 }
20
21 CreatureType Game::chooseRandomCreatureType()
22 {
23     int randIdx = rand() % m_probSizeArray;
24     switch (m_probArray[randIdx]) {
25     case CreatureInt_Hunter:
26         return CreatureType::Hunter;
27     case CreatureInt_Prey:
28         return CreatureType::Prey;
29     case CreatureInt_Plant:
30         return CreatureType::Plant;
31     default:
32         std::cerr << "Errrr_choose_(" << m_probArray[randIdx]
33             << ")!_That_not_supposed_to_happen!" << std::endl;
34     }
35     return CreatureType::Plant;
36 }
37
38 Game::Game(int Xsize, int Ysize)
39     : m_Xsize(Xsize),
40       m_Ysize(Ysize),
41       m_window(sf::VideoMode(m_Xsize, m_Ysize), m_gameTitle)
42 {
43     m_squareSize
44         = (m_Xsize < m_Ysize ? m_Xsize : m_Ysize) / m_Nrow - m_borderSize;
45     m_board = new Creature**[m_Nrow];
46     for (int i = 0; i < m_Nrow; ++i) {
47         m_board[i] = new Creature*[m_Nrow];
48         for (int j = 0; j < m_Nrow; ++j)
49             m_board[i][j] = nullptr;
50     }
51     initializeArray();
52 }
53
54 Game::Game(int Xsize) : Game(Xsize, Xsize)
55 {
56 }
57
58 Game::Game() : Game(600, 600)
59 {
60 }
61
62 sf::Vector2f Game::getCoord(int idxX, int idxY)
63 {

```

```

64     float coordX = idxX * (m_squareSize + m_borderSize);
65     float coordY = idxY * (m_squareSize + m_borderSize);
66     return sf::Vector2f(coordX, coordY);
67 }
68
69 sf::RectangleShape Game::createShape(Creature* creature)
70 {
71     sf::RectangleShape shape
72         = sf::RectangleShape(sf::Vector2f(m_squareSize, m_squareSize));
73     shape.setPosition(getCoord(creature->getX(), creature->getY()));
74     shape.setFillColor(creature->getColor());
75     return shape;
76 }
77
78 int Game::isIndexAvailable(int x, int y)
79 {
80     if (m_board[x][y] == nullptr)
81         return 1;
82     return 0;
83 }
84
85 int Game::isSideIndexAvailable()
86 {
87     for (int i = 0; i < m_Nrow; ++i)
88         if (m_board[i][0] == nullptr || m_board[i][m_Nrow - 1] == nullptr)
89             return 1;
90     for (int j = 0; j < m_Nrow; ++j)
91         if (m_board[0][j] == nullptr || m_board[m_Nrow - 1][j] == nullptr)
92             return 1;
93     return 0;
94 }
95
96 int Game::isBoardFull()
97 {
98     for (int i = 0; i < m_Nrow; ++i)
99         for (int j = 0; j < m_Nrow; ++j)
100             if (m_board[i][j] == nullptr)
101                 return 0;
102     return 1;
103 }
104
105 Creature* Game::getCreature(CreatureType type, int onSide)
106 {
107     int randX, randY;
108     if (m_countCreatures + 1 > m_Nrow * m_Nrow || isBoardFull() == 1) {
109         std::cerr << "No_available_cell!" << std::endl;
110         return nullptr;
111     }
112     if (onSide == 1) {
113         if (isSideIndexAvailable() == 0) {
114             return nullptr;
115         }
116     }
117     int counter = 0;
118     do {
119         randX = rand() % m_Nrow;
120         randY = rand() % m_Nrow;
121         if (onSide) {

```

```

122         int onSideCoef = rand() % 2;
123         int onSideCoef9 = rand() % 2;
124         randX *= onSideCoef;
125         randY *= !onSideCoef;
126         randX += randX ? 0 : onSideCoef9 ? m_Nrow - 1 : 0;
127         randY += randY ? 0 : onSideCoef9 ? m_Nrow - 1 : 0;
128     }
129     ++counter;
130 } while (isIndexAvailable(randX, randY) == 0);
131
132 switch (type) {
133 case CreatureType::Hunter:
134     return new Hunter(randX, randY);
135 case CreatureType::Prey:
136     return new Prey(randX, randY);
137 case CreatureType::Plant:
138     return new Plant(randX, randY);
139 default:
140     std::cerr << "Errrr_get!_That_not_supposed_to_happen!" << std::endl;
141     break;
142 }
143 return nullptr;
144 }
145
146 Creature* Game::getRandomCreature()
147 {
148     switch (chooseRandomCreatureType()) {
149     case CreatureType::Hunter:
150         return getCreature(CreatureType::Hunter);
151     case CreatureType::Prey:
152         return getCreature(CreatureType::Prey);
153     case CreatureType::Plant:
154         return getCreature(CreatureType::Plant);
155     default:
156         std::cerr << "Errrr!_That_not_supposed_to_happen!" << std::endl;
157     }
158     return nullptr;
159 }
160
161 void Game::initializeArray()
162 {
163     fillProbArray();
164     for (int i = 0; i < m_sizeArrayCreatures; ++i) {
165         Creature* creature = getRandomCreature();
166
167         if (creature == nullptr)
168             break;
169
170         m_board[creature->getX()][creature->getY()] = creature;
171         ++m_countCreatures;
172     }
173 }
174
175 void Game::updateEnvironment()
176 {
177     if (m_steps % 5 == 0)
178         for (int i = 0; i < 15; ++i) {
179             Creature* creature = getCreature(CreatureType::Plant);

```

```

180         if (creature == nullptr)
181             break;
182         m_board[creature->getX()][creature->getY()] = creature;
183         ++m_countCreatures;
184     }
185
186     if (countHunters() < 15) {
187         for (int i = 0; i < 15; ++i) {
188             Creature* creature = getCreature(CreatureType::Hunter, 1);
189             if (creature == nullptr)
190                 break;
191             else
192                 m_board[creature->getX()][creature->getY()] = creature;
193             ++m_countCreatures;
194         }
195     }
196     /*
197     if (countPreys() < 3) {
198         for (int i = 0; i < 2; ++i) {
199             Creature* creature = getCreature(CreatureType::Prey, 1);
200             if (creature == nullptr)
201                 break;
202             else
203                 m_board[creature->getX()][creature->getY()] = creature;
204             ++m_countCreatures;
205         }
206     }
207     */
208 }
209
210 void Game::lifeStep()
211 {
212     ++m_steps;
213
214     int dx, dy;
215     for (int i = 0; i < m_Nrow; ++i) {
216         for (int j = 0; j < m_Nrow; ++j) {
217             Creature* creature = m_board[i][j];
218             if (!creature)
219                 continue;
220             creature->step();
221             if (creature->isDead()) {
222                 m_board[i][j] = nullptr;
223                 --m_countCreatures;
224                 continue;
225             }
226
227             int speed = creature->getSpeed();
228             if (speed == 1) {
229                 dx = rand() % 3 - 1;
230                 dy = rand() % 3 - 1;
231             } else {
232                 dx = rand() % 5 - 2;
233                 dy = rand() % 5 - 2;
234             }
235
236             if (i + dx < 0 || i + dx >= m_Nrow)
237                 dx = 0;

```



```

238     if (j + dy < 0 || j + dy >= m_Nrow)
239         dy = 0;
240     Creature* nextCell = m_board[i + dx][j + dy];
241     int isChild;
242     switch (creature->getType()) {
243     case CreatureType::Plant:
244         break;
245     case CreatureType::Prey:
246         isChild = rand() % 100 < 20 ? 1 : 0;
247         if (nextCell == nullptr) {
248             m_board[i + dx][j + dy] = m_board[i][j];
249
250             if (isChild) {
251                 Creature* child = getCreature(CreatureType::Prey);
252                 if (child) {
253                     m_board[i][j] = child;
254                     ++m_countCreatures;
255                 } else
256                     m_board[i][j] = nullptr;
257             } else
258                 m_board[i][j] = nullptr;
259         } else {
260             if (nextCell
261                 && nextCell->getType() == CreatureType::Plant) {
262                 Prey* creature = (Prey*)m_board[i][j];
263                 creature->eat();
264                 --m_countCreatures;
265                 m_board[i + dx][j + dy] = m_board[i][j];
266                 if (isChild) {
267                     Creature* child = getCreature(CreatureType::Prey);
268                     if (child) {
269                         child->setPosition(i, j);
270                         m_board[i][j] = child;
271                         ++m_countCreatures;
272                         creature->giveBirth();
273                     } else
274                         m_board[i][j] = nullptr;
275                 } else {
276                     m_board[i][j] = nullptr;
277                 }
278             }
279         }
280         break;
281     case CreatureType::Hunter:
282         if (nextCell == nullptr) {
283             m_board[i + dx][j + dy] = m_board[i][j];
284             m_board[i][j] = nullptr;
285         } else {
286             if (nextCell && nextCell->getType() == CreatureType::Prey) {
287                 Hunter* creature = (Hunter*)m_board[i][j];
288                 creature->eat();
289                 if (creature->getCountEaten() == 2)
290                     isChild = rand() % 100 < 100 ? 1 : 0;
291                 m_board[i + dx][j + dy] = nullptr;
292                 --m_countCreatures;
293                 m_board[i + dx][j + dy] = m_board[i][j];
294                 if (isChild) {
295                     Creature* child = getCreature(CreatureType::Hunter);

```

```

296             if (child != nullptr) {
297                 child->setPosition(i, j);
298                 m_board[i][j] = child;
299                 ++m_countCreatures;
300                 creature->giveBirth();
301             } else {
302                 m_board[i][j] = nullptr;
303             }
304         } else {
305             m_board[i][j] = nullptr;
306         }
307     }
308 }
309 break;
310
311 default:
312     break;
313 }
314 }
315 }
316
317 // update in-class positions
318 for (int i = 0; i < m_Nrow; ++i)
319     for (int j = 0; j < m_Nrow; ++j)
320         if (m_board[i][j] != nullptr)
321             m_board[i][j]->setPosition(i, j);
322
323 updateEnvironment();
324 }
325
326 int Game::countHunters()
327 {
328     int count = 0;
329     for (int i = 0; i < m_Nrow; ++i)
330         for (int j = 0; j < m_Nrow; ++j)
331             if (m_board[i][j]
332                 && m_board[i][j]->getType() == CreatureType::Hunter)
333                 ++count;
334     return count;
335 }
336
337 int Game::countPreys()
338 {
339     int count = 0;
340     for (int i = 0; i < m_Nrow; ++i)
341         for (int j = 0; j < m_Nrow; ++j)
342             if (m_board[i][j] && m_board[i][j]->getType() == CreatureType::Prey)
343                 ++count;
344     return count;
345 }
346
347 void Game::run()
348 {
349     time_t startTime = time(0);
350     while (m_window.isOpen()) {
351         sf::Event event;
352         while (m_window.pollEvent(event)) {
353             if (event.type == sf::Event::Closed

```

```

354         || sf::Keyboard::isKeyPressed(sf::Keyboard::Q))
355         m_window.close();
356     else if (
357         event.type == sf::Event::KeyPressed
358         && sf::Keyboard::isKeyPressed(sf::Keyboard::E)) {
359         lifeStep();
360     }
361 }
362
363 if (time(0) - startTime == 1) {
364     lifeStep();
365     startTime = time(0);
366 }
367
368 // update inside-class coordinates
369 for (int i = 0; i < m_Nrow; ++i)
370     for (int j = 0; j < m_Nrow; ++j)
371         if (m_board[i][j] != nullptr)
372             m_board[i][j]->setPosition(i, j);
373
374 m_window.clear(sf::Color::Black);
375
376 for (int i = 0; i < m_Nrow; ++i) {
377     for (int j = 0; j < m_Nrow; ++j) {
378         Creature* creature = m_board[i][j];
379         if (creature == nullptr)
380             continue;
381         m_window.draw(createShape(m_board[i][j]));
382     }
383 }
384
385 m_window.display();
386 }
387 }

```

## 3.2 Скриншоты с примером работы: