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

1 Введение

1.1 Постановка задачи:

Игроку представляется поле размером N клеток на N клеток (в данной работе используется $N = 150$), реализованное в графическом режиме. Каждую секунду, или по нажатию клавиши E сменяется жизненный цикл. На клетках находятся живые организмы трёх типов:

- Тип **Растение** (Plant): является пищей для травоядного, не двигается. Определённое количество растений появляется раз в несколько жизненных циклов.
- Тип **Травоядное животное** (Prey): поглощает растения и является пищей для хищника. Каждое травоядное животное каждый жизненный цикл может сдвинуться на одну из соседних клеток, с некоторой вероятностью оставляя после себя травоядное животное. Наступая на клетку с растением, поглощает его. Размножается с вероятностью 35%.
- Тип **Хищник** (Hunter): поглощает травоядных, но не взаимодействует с растениями. Хищник каждый ход может сдвинуться на одну из соседних клеток (при голоде может сдвинуться на две клетки). Поглощает всех травоядных животных, которых встретит. С растениями не взаимодействует. Размножается только после поглощения минимум двух травоядных животных с вероятностью 25%.

Травоядные и хищники обладают системой голода. Изначальный показатель голода для хищников 36, для травоядных 11 (значения подобраны в ходе реализации для достижения умеренно стабильного состояния поля в ходе жизненного цикла). Каждое перемещение животного отнимает 2 единицы от показателя голода. Рождение отнимает 4 единицы. Поглощение прибавляет к показателю голода 4. Животное погибает, когда показатель голода становится равным нулю.

Растения живут 6 циклов, затем погибают. Определённое количество растений появляется на поле каждые 1-6 циклов.

2 Технологии ООП

Для реализации программы были использованы принципы ООП, такие как: наследование, полиморфизм, инкапсуляция, абстрактный класс.

3 Реализация

Программа написана на языке C++ под платформу Linux с ипользованием графической библиотеки SFML.

Описание классов:

- class Creature:

  Абстрактный класс, класс-родитель для всех классов живых существ. В нём задаются координаты объекта и цвет. Также оъявляются общие для всех объектов виртуальные функции, например isDead() или step() (функция, меняющая значения аттрибутов класса каждый жизненный цикл).

- class Plant:

Класс объекта Растение. Наследуется от класса Creature. В классе дополнительно приватно определена переменная m_lifetime и соответствующая функция getLifetime(). Графически обозначен зелёным цветом.

- class LivingCreature:

  Класс, наследующийся от класса Creature, но также являющийся классом-родителем для классов с системой голода (классы Prey и Hunter). В классе определены соответствующие переменные для контроля показателя голода и константные переменные для величины изменения показателя голода при определённом действии.

- class Prey и class Hunter:

  Классы объектов Травоядное животное и Хищник. Наследуются от класса LivingCreature. Наследуют аттрибуты родительских классов и переопределяют виртуальные функции. Объекты класса Травоядное животное обозначены жёлтым цветом, объекты класса Хищник – Красным.

- class Game:

  Самый крупный класс в программе. Создаёт игровое поле, контролирует весь игровой процесс,отображение фигур, создание и перемещение объектов.

4 Результаты работы

В ходе работы была реализована программа для игры Жизнь. Параметры программы настраиваемые, при экспериментировании получаются интересные различные постоянно изменяющиеся состояния игрового поля.

5 Скриншоты

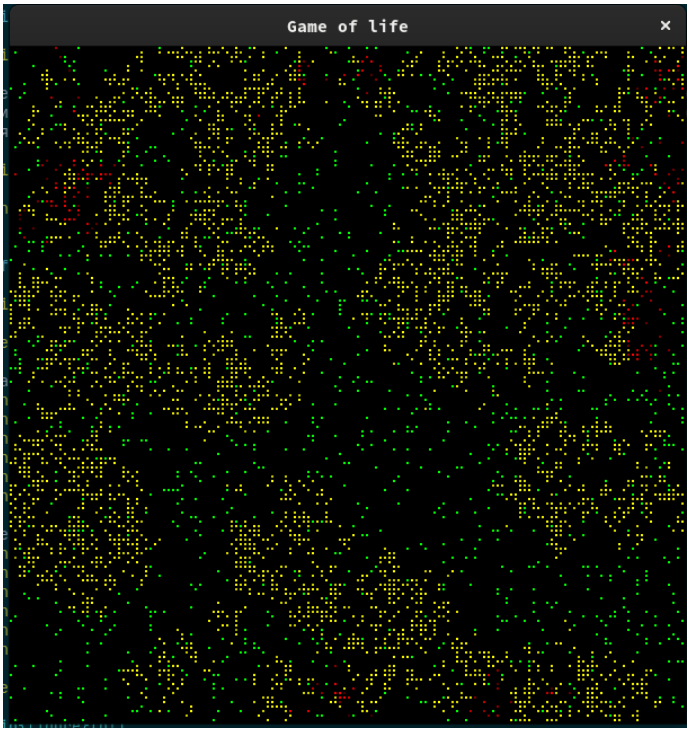

Рис. 1: Одно из состояний программы (хищники почти отсутствуют)

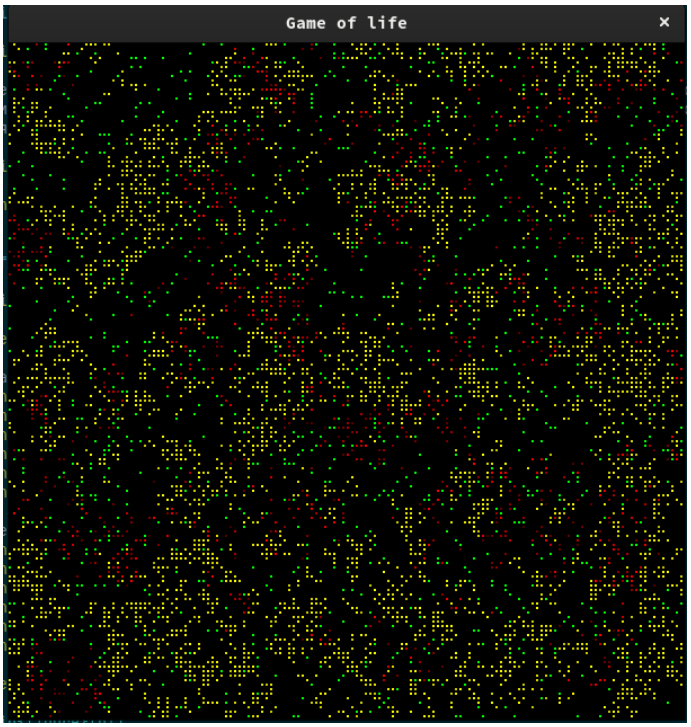

Рис. 2: Одно из состояний программы

6 Код программы

language: C++

Creature.hpp

```cpp
#ifndef CREATURE_HPP
#define CREATURE_HPP

#include <SFML/Graphics.hpp>

enum class CreatureType { Hunter, Prey, Plant };

class Creature {
    int m_idxX;
    int m_idxY;

    sf::Color m_color;

    float m_probability;

public:
    Creature(int, int);

    virtual CreatureType getType() = 0;

    void setX(int);
    int getX();

    void setY(int);
    int getY();

    void setPosition(int, int);

    void setColor(sf::Color);
    virtual sf::Color getColor();

    void setProb(float);
    virtual float getProb();

    virtual void step();

    virtual int isDead();

    int getSpeed();

    virtual const char* getTypeName();
};

#endif // #ifndef CREATURE_HPP
```

Plant.hpp

```cpp
#ifndef PLANT_HPP
#define PLANT_HPP

#include "Creature.hpp"

#include <SFML/Graphics.hpp>

```

```
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 = 11;
11     int m_eaten = 0;
12     const int m_hungerStep = -2;
13     const int m_gaveBirthStep = -4;
14     const int m_eatStep = 4;
15
16     uint8_t delta = 10;
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     void setInitialHunger(int);
31 };
32
33 #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 = 150; // 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;
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*);
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  }
42
43  void LivingCreature::setInitialHunger(int hunger)
44  {
45      m_hunger = hunger;
46  }
```

Prey.cpp

```
1   #include "Prey.hpp"
```

```cpp
#include "LivingCreature.hpp"

Prey::Prey(int x, int y) : LivingCreature(x, y)
{
    setColor(sf::Color::Yellow);
    setProb(0.4f);
}

CreatureType Prey::getType()
{
    return CreatureType::Prey;
}

const char* Prey::getTypeName()
{
    return "Prey";
}

int Prey::getSpeed()
{
    return 1;
}
```

Hunter.cpp

```cpp
#include "Hunter.hpp"

Hunter::Hunter(int x, int y) : LivingCreature(x, y)
{
    setColor(sf::Color::Red);
    setInitialHunger(36);
    setProb(0.2f);
}

CreatureType Hunter::getType()
{
    return CreatureType::Hunter;
}

const char* Hunter::getTypeName()
{
    return "Hunter";
}

int Hunter::getSpeed()
{
    if (m_hunger <= 18)
        return 2;
    return 1;
}
```

Game.cpp

```cpp
#include "Game.hpp"

void Game::fillProbArray()
{
    int coef = 100;

    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 % (rand() % 6 + 1) == 0)
178         for (int i = 0; i < m_Nrow * 3; ++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() < m_Nrow / 2) {
187             for (int i = 0; i < m_Nrow / 2; ++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() < m_Nrow / 2) {
198             for (int i = 0; i < m_Nrow / 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     void Game::lifeStep()
210     {
211         ++m_steps;
212
213         int dx, dy;
214         for (int i = 0; i < m_Nrow; ++i) {
215             for (int j = 0; j < m_Nrow; ++j) {
216                 Creature* creature = m_board[i][j];
217                 if (!creature)
218                     continue;
219                 creature->step();
220                 if (creature->isDead()) {
221                     m_board[i][j] = nullptr;
222                     --m_countCreatures;
223                     continue;
224                 }
225
226                 int speed = creature->getSpeed();
227                 if (speed == 1) {
228                     dx = rand() % 3 - 1;
229                     dy = rand() % 3 - 1;
230                 } else {
231                     dx = rand() % 5 - 2;
232                     dy = rand() % 5 - 2;
233                 }
234
235                 if (i + dx < 0 || i + dx >= m_Nrow)
236                     dx = 0;
237                 if (j + dy < 0 || j + dy >= m_Nrow)
238                     dy = 0;
239                 Creature* nextCell = m_board[i + dx][j + dy];
240                 int isChild;
241                 switch (creature->getType()) {
242                 case CreatureType::Plant:
243                     break;
```

```
244                 case CreatureType::Prey:
245                     isChild = rand() % 100 < 35 ? 1 : 0;
246                     if (nextCell == nullptr) {
247                         m_board[i + dx][j + dy] = m_board[i][j];
248 
249                         if (isChild) {
250                             Creature* child = getCreature(CreatureType::Prey);
251                             if (child) {
252                                 m_board[i][j] = child;
253                                 ++m_countCreatures;
254                             } else
255                                 m_board[i][j] = nullptr;
256                         } else
257                             m_board[i][j] = nullptr;
258                     } else {
259                         if (nextCell
260                             && nextCell->getType() == CreatureType::Plant) {
261                             Prey* creature = (Prey*)m_board[i][j];
262                             creature->eat();
263                             --m_countCreatures;
264                             m_board[i + dx][j + dy] = m_board[i][j];
265                             if (isChild) {
266                                 Creature* child = getCreature(CreatureType::Prey);
267                                 if (child) {
268                                     child->setPosition(i, j);
269                                     m_board[i][j] = child;
270                                     ++m_countCreatures;
271                                     creature->giveBirth();
272                                 } else
273                                     m_board[i][j] = nullptr;
274                             } else {
275                                 m_board[i][j] = nullptr;
276                             }
277                         }
278                     }
279                     break;
280                 case CreatureType::Hunter:
281                     if (nextCell == nullptr) {
282                         m_board[i + dx][j + dy] = m_board[i][j];
283                         m_board[i][j] = nullptr;
284                     } else {
285                         if (nextCell && nextCell->getType() == CreatureType::Prey) {
286                             Hunter* creature = (Hunter*)m_board[i][j];
287                             creature->eat();
288                             if (creature->getCountEaten() >= 2)
289                                 isChild = rand() % 100 < 25 ? 1 : 0;
290                             m_board[i + dx][j + dy] = nullptr;
291                             --m_countCreatures;
292                             m_board[i + dx][j + dy] = m_board[i][j];
293                             if (isChild) {
294                                 Creature* child = getCreature(CreatureType::Hunter);
295                                 if (child != nullptr) {
296                                     child->setPosition(i, j);
297                                     m_board[i][j] = child;
298                                     ++m_countCreatures;
299                                     creature->giveBirth();
300                                 } else {
301                                     m_board[i][j] = nullptr;
302                                 }
```

```
303                             } else {
304                                 m_board[i][j] = nullptr;
305                             }
306                         }
307                     }
308                     break;
309 
310                 default:
311                     break;
312                 }
313             }
314         }
315 
316         // update in-class positions
317         for (int i = 0; i < m_Nrow; ++i)
318             for (int j = 0; j < m_Nrow; ++j)
319                 if (m_board[i][j] != nullptr)
320                     m_board[i][j]->setPosition(i, j);
321 
322         updateEnvironment();
323     }
324 
325     int Game::countHunters()
326     {
327         int count = 0;
328         for (int i = 0; i < m_Nrow; ++i)
329             for (int j = 0; j < m_Nrow; ++j)
330                 if (m_board[i][j]
331                     && m_board[i][j]->getType() == CreatureType::Hunter)
332                     ++count;
333         return count;
334     }
335 
336     int Game::countPreys()
337     {
338         int count = 0;
339         for (int i = 0; i < m_Nrow; ++i)
340             for (int j = 0; j < m_Nrow; ++j)
341                 if (m_board[i][j] && m_board[i][j]->getType() == CreatureType::Prey)
342                     ++count;
343         return count;
344     }
345 
346     void Game::run()
347     {
348         time_t startTime = time(0);
349         while (m_window.isOpen()) {
350             sf::Event event;
351             while (m_window.pollEvent(event)) {
352                 if (event.type == sf::Event::Closed
353                     || sf::Keyboard::isKeyPressed(sf::Keyboard::Q))
354                     m_window.close();
355                 else if (
356                         event.type == sf::Event::KeyPressed
357                         && sf::Keyboard::isKeyPressed(sf::Keyboard::E)) {
358                     lifeStep();
359                 }
360             }
361 
```

```
362             if (time(0) - startTime == 1) {
363                 lifeStep();
364                 startTime = time(0);
365             }
366 
367             // update inside-class coordinates
368             for (int i = 0; i < m_Nrow; ++i)
369                 for (int j = 0; j < m_Nrow; ++j)
370                     if (m_board[i][j] != nullptr)
371                         m_board[i][j]->setPosition(i, j);
372 
373             m_window.clear(sf::Color::Black);
374 
375             for (int i = 0; i < m_Nrow; ++i) {
376                 for (int j = 0; j < m_Nrow; ++j) {
377                     Creature* creature = m_board[i][j];
378                     if (creature == nullptr)
379                         continue;
380                     m_window.draw(createShape(m_board[i][j]));
381                 }
382             }
383 
384             m_window.display();
385         }
386     }
```