Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

кафедра ПМ иК

КУРСОВАЯ РАБОТА по дисциплине «Объектно-ориентированное программирование»

Тема: «\_Жизненный цикл в заданной экосистеме \_»

Выполнили: студенты группы ИП-011

Игнатьев Данил Андреевич

Проверил: ассистент кафедры ПМиК

Бублей Д.А.

Новосибирск – 2021

Содержание

[Постановка задачи 3](#__RefHeading___Toc213_4041067640)

[Технологии ООП 3](#__RefHeading___Toc215_4041067640)

[Структура классов 4](#__RefHeading___Toc358_4041067640)

[Программная реализация 4](#__RefHeading___Toc360_4041067640)

[Результаты работы 8](#__RefHeading___Toc5109_4041067640)

[Заключение 8](#__RefHeading___Toc5111_4041067640)

[Приложение. Листинг 9](#__RefHeading___Toc5113_4041067640)

# Постановка задачи

Вариант №7: Растение – банан, травоядное животное - обезьяна, хищник – ягуар. Бананы живут 10 жизненных циклов. Обезьяны размножаются с вероятностью 25%, имеют показатель голода 1 и могут перемещаться на любую случайную клетку в радиусе 3 клеток от них. Ягуары имеют показатель голода 1. На поле может находится до двух ягуаров. В случае уменьшения популяции ягуаров, на краю поля появляются ещё ягуары. Пока показатель голода ягуара больше 0,4, он перемещается хаотично. В случае если показатель голода будет меньше, он автоматически нападает на любую обезьяну в радиусе 2 клеток, сразу же поедая её.

# Средства разработки

Code::Blocks IDE – среда разработки ПО, в ней разрабатывалось само приложение

# Технологии ООП

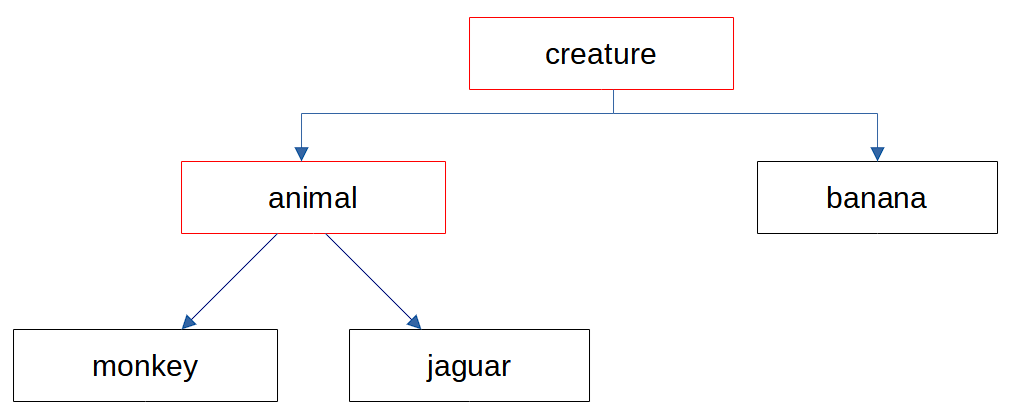
В курсовой работе были применены ниже перечисленные технологии ООП:

* Инкапсуляция (все поля данных не доступны из внешних функций)
* Наследование (4 класса, один из которых - абстрактный)
* Полиморфизм
* Конструкторы
* Списки инициализации
* Статические переменные
* Виртуальные функции

Инкапсуляция

# 

# Структура классов

Рисунок 1: Структура классов

# Программная реализация

Класс creature

|  |
| --- |
| class creature  {  protected:  int x, y;  char name;  public:  char getName();  void setName(char pname);  int getX();  int getY();  void setX(int coord);  void setY(int coord);  void die(creature die\_creature);  virtual bool TryDie(creature\* die\_creature)=0;  creature(): x(1 + rand() % 10) , y(1 + rand() % 10)  {    }  }; |

Класс содержит переменные координат(*int x, y*) и переменную имени(*char name*). В конструкторе случайно генерируются координаты существа.

Также присутствуют геттеры и сеттеры для переменных и полностью виртуальная функция *TryDie*, которая не имеет реализации.

Класс animal

|  |
| --- |
| class animal : public creature  {  protected:  float bornpower;  float movepower;  float foodpower;  float hunger;  static int population;  public:  virtual bool food(creature\* monkey) = 0;  virtual void move() = 0;  virtual bool reproduction();  float getHunger();  void setHunger(float hung);  bool TryDie(creature\* die\_creature) override;  int getPopulation();  animal()  {  name = '0';  bornpower = 0.4;  movepower = 0.2;  foodpower = 0.2;  hunger = 1;  }  };  int animal::population = 0; |

Класс содержит показатели уменьшения голода в следствии различных действий(*float bornpower, movepower, foodpower*) и само значение голода(*float hunger*). Статическая целочисленная переменная *population* хранит в себе кол-во созданных обезьян. Также присутствуют полностью виртуальные функции *food, move,* которые не имеют реализации. Функция *reproduction* отвечает за размножение обезьян. Функция *TryDie* проверяет, что голод животного не опустился до критического значения.

Класс banana

|  |
| --- |
| class banana : public creature  {  private:  int lifeCycle;  int year;  public:  banana(): lifeCycle(10), year(0)  {  name = 'B';  }  int getLifeCycle();  int getYear();  void setYear(int pyear);  bool TryDie(creature\* die\_creature) override;  void reproduction();  }; |

Переменная *lifeCycle* содержит количество жизненных циклов, отведённых банану, а переменная *year*​, содержит количество уже пройденных циклов. Функция *TryDie* проверяет, что количество прожитых циклов не превышает максимальное значение. Функция *reproduction* отвечает за появление нового банана, в случае смерти другого.

Класс monkey

|  |
| --- |
| class monkey : public animal  {  public:  void move() override;  bool food(creature\* banana) override;  monkey()  {  name = 'M';  population++;  }  }; |

Функция *move* генерирует новые координаты в соответствие с условием задачи. Функция *food* отвечает за поедание бананов, с последующим восстановлением голода.

Класс jaguar

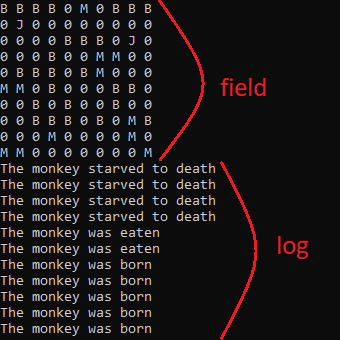
|  |
| --- |
| class jaguar : public animal  {  public:  void move() override;  bool food(creature\* monkey) override;  bool reproduction() override;  jaguar()  {  name = 'J';  }  }; |

Функция *move* генерирует новые координаты в соответствие с условием задачи. Функция *food* отвечает за поедание обезьян, с последующим восстановлением голода. Функция *reproduction* отвечает за появление нового ягура, в случае смерти другого.

Source.cpp

В функции main вызывается *createfield*, в которой создаётся двумерный массив. Далее запускается цикл, выход из которого производится нажатием любой клавиши. Потом происходит размещение существ на поле(двумерный массив), проверка жизненных показателей, рождение новых существ, движение животных. Так же вызывается функция *log*, которая выводит статистику по каждому жизненному циклу.

# Результаты работы

Рисунок 2: Пример одного из жизненных циклов

# Заключение

В ходе выполнения курсовой работы, мы реализовали жизненный цикл в заданной экосистеме с использованием технологии Объектно-Ориентированного Программирования.

# Приложение. Листинг

**Header.h**

|  |
| --- |
| #pragma once  #include <iostream>  #include<conio.h>  #include <stdlib.h>  #include <time.h>  #include <windows.h>  #include <vector>  #define SIZEFIELD 11  #define JAGMOVE 0.4  using namespace std;  class creature  {  protected:  int x, y;  char name;  public:  char getName();  void setName(char pname);  int getX();  int getY();  void setX(int coord);  void setY(int coord);  void die(creature die\_creature);  virtual bool TryDie(creature\* die\_creature)=0;  creature(): x(1 + rand() % 10) , y(1 + rand() % 10)  {    }  };  class animal : public creature  {  protected:  float bornpower;  float movepower;  float foodpower;  float hunger;  static int population;  public:  virtual bool food(creature\* monkey) = 0;  virtual void move() = 0;  virtual bool reproduction();  float getHunger();  void setHunger(float hung);  bool TryDie(creature\* die\_creature) override;  int getPopulation();  animal()  {  name = '0';  bornpower = 0.4;  movepower = 0.2;  foodpower = 0.2;  hunger = 1;  }  };  int animal::population = 0;  class monkey : public animal  {  public:  void move() override;  bool food(creature\* banana) override;  monkey()  {  name = 'M';  population++;  }  };  class jaguar : public animal  {  public:  void move() override;  bool food(creature\* monkey) override;  bool reproduction() override;  jaguar()  {  name = 'J';  }  };  class banana : public creature  {  private:  int lifeCycle;  int year;  public:  banana(): lifeCycle(10), year(0)  {  name = 'B';  }  int getLifeCycle();  int getYear();  void setYear(int pyear);  bool TryDie(creature\* die\_creature) override;  void reproduction();  };  //creature  int creature::getX()  {  return x;  }  int creature::getY()  {  return y;  }  void creature::setX(int coord)  {  x = coord;  }  void creature::setY(int coord)  {  y = coord;  }  char creature::getName()  {  return name;  }  void creature::setName(char pname)  {  name = pname;  }  //animal  float animal::getHunger()  {  return hunger;  }  void animal::setHunger(float hung)  {  hunger = hung;  }  int animal::getPopulation()  {  return population;  }  bool animal::TryDie(creature\* die\_creature)  {  if (name == '0')  {  return false;  }    if (hunger <= 0) {  die\_creature->setName('0');  return true;  }  return false;  }  bool animal::reproduction()  {  if (name=='0')  {  return false;  }  int chanceTOborn = rand() % 4;  if (chanceTOborn == 2)  {  hunger -= 0.4;  return true;  }  return false;  }  //monkey  void monkey::move()  {  if (hunger > 0) {  int chanceX = rand() % 2;  int chanceY = rand() % 2;  if (chanceX == 0)  x = x - (1 + rand() % 3);  else  x = x + (1 + rand() % 3);  if (chanceY == 0)  y = y - (1 + rand() % 3);  else  y = y + (1 + rand() % 3);  if (x > 10)  x = 10;  if (x < 1)  x = 1;  if (y > 10)  y = 10;  if (y < 1)  y = 1;  hunger -= movepower;  }  }  bool monkey::food(creature\* banana)  {  if ((x == banana->getX()) && (y == banana->getX()) && (banana->getName() != '0'))  {  hunger += foodpower;  banana->setName('0');  return true;  }  return false;  }  //Jaguar  void jaguar::move() {  int chancex = rand() % 2;  int chancey = rand() % 2;  if (hunger > 0) {  if (chancex == 0)  x++;  else  x--;  if (chancey == 0)  y++;  else  y--;  if (x > 10)  x = 10;  else if (x < 1)  x = 1;  if (y > 10)  y = 10;  else if (y < 1)  y = 1;  hunger -= movepower;  }  }  bool jaguar::food(creature\* monkey)  {  if (hunger > JAGMOVE)  {  if ((x == monkey->getX()) && (y == monkey->getY()) && (monkey->getName() != '0'))  {  hunger += foodpower;  monkey->setName('0');  return true;  }  }  else  {  for (int i = (x - 2); i < (x + 3); i++)  {  for (int j = (y - 2); j < (y + 3); j++)  {  if (i == (monkey->getX()) && (j == monkey->getY()) && (monkey->getName() != '0'))  {  hunger += foodpower;  monkey->setName('0');  return true;  }  }  }  }  return false;  }  bool jaguar::reproduction()  {  if (name == '0')  {  name = 'J';  hunger = 1;  int chancex = rand() % 2;  int chancey = rand() % 2;  if (chancex == 0)  x = 1;  else  x = 10;  if (chancey = 1)  y = 1;  else  y = 10;  return true;  }  return false;  }  //Banana  int banana::getLifeCycle()  {  return lifeCycle;  }  int banana::getYear()  {  return year;  }  void banana::setYear(int pyear)  {  year = pyear;  }  bool banana::TryDie(creature\* die\_creature)  {  year++;  if (year >= 10) {  die\_creature->setName('0');  return true;  }  return false;  }  void banana::reproduction()  {  if (name == '0') {  name = 'B';  x = 1 + rand() % 10;  y = 1 + rand() % 10;  year = 0;  }  } |

**Source.cpp**

|  |
| --- |
| #include "Header.h"  using namespace std;  char\*\* createField();  void log(int MonHungDie, int JagHungDie, int MonFoodDie, int MonRep,int JagRep);  int main()  {  srand(time(NULL));  int monkpop = 10;    vector<monkey> monfam(monkpop);  jaguar jagfam[2];  banana banfam[40];  char\*\* field = createField();  while (!\_kbhit())  {  int MonHungDie = 0, JagHungDie = 0, MonFoodDie = 0, MonRep = 0, JagRep = 0;  for (int i = 0; i < monfam[1].getPopulation(); i++)  field[monfam[i].getX()][monfam[i].getY()] = monfam[i].getName();  field[jagfam[0].getX()][jagfam[0].getY()] = jagfam[0].getName();  field[jagfam[1].getX()][jagfam[1].getY()] = jagfam[1].getName();  for (int i = 0; i < 40; i++) {  field[banfam[i].getX()][banfam[i].getY()] = banfam[i].getName();  banfam[i].TryDie(&banfam[i]);  banfam[i].reproduction();  }  for (size\_t i = 1; i < SIZEFIELD; i++)  {  for (size\_t j = 1; j < SIZEFIELD; j++)  {  cout << field[i][j] << ' ';  field[i][j] = '0';  }  cout << endl;  }    for (int i = 0; i < monfam[1].getPopulation(); i++) {  bool chance = false;  monfam[i].move();  for (int j = 0; j < 40; j++)  monfam[i].food(&banfam[j]);  chance = monfam[i].TryDie(&monfam[i]);  if (chance == true)  MonHungDie++;  chance = false;  chance = monfam[i].reproduction();  if (chance == true)  {  monkey mon;  monfam.push\_back(mon);  MonRep++;  }    }  for (int i = 0; i < 2; i++) {  bool chance = false;  jagfam[i].move();  for (int j = 0; j < monfam[1].getPopulation(); j++)  {  chance=jagfam[i].food(&monfam[j]);  if (chance==true)  MonFoodDie++;  chance = false;    }  chance=jagfam[i].TryDie(&jagfam[i]);  if (chance == true)  JagHungDie++;  chance = false;  chance=jagfam[i].reproduction();  if (chance == true)  JagRep++;  }  log(MonHungDie, JagHungDie, MonFoodDie, MonRep, JagRep);  // chanceBORN = firstMonk.reproduction();  //cout << firstMonk.getHunger() << endl;  Sleep(1000);  system("cls");  }  return 0;  }  char\*\* createField()  {  char\*\* matrix = new char\* [SIZEFIELD];  for (int i = 0; i < SIZEFIELD; i++) {  matrix[i] = new char[SIZEFIELD];  }  for (int i = 0; i < SIZEFIELD; i++) {  for (int j = 1; j < matrix[i][0] + 1; j++) {  matrix[i][j] = '0';  }  }  return matrix;  }  void log(int MonHungDie, int JagHungDie, int MonFoodDie, int MonRep, int JagRep)  {  for (size\_t i = 0; i < MonHungDie; i++)  {  cout << "The monkey starved to death" << endl;  Sleep(200);  }  for (size\_t i = 0; i < JagHungDie; i++)  {  cout << "Jaguar starved to death" << endl;  Sleep(200);    }  for (size\_t i = 0; i < MonFoodDie; i++)  {  cout << "The monkey was eaten" << endl;  Sleep(200);    }  for (size\_t i = 0; i < MonRep; i++)  {  cout << "The monkey was born" << endl;  Sleep(200);  }  for (size\_t i = 0; i < JagRep; i++)  {  cout << "Jaguar was born" << endl;  Sleep(200);    }  } |