// Финальный вариант

#include<iostream>

#include<cstdlib>

#include<ctime>

#include<cmath>

#include<string>

#ifdef \_WIN32

#define CLEAR\_SCREEN "cls"

#else

#define CLEAR\_SCREEN "clear"

#endif

using namespace std;

const char predator\_symbol\_young = 'p';

const char predator\_symbol\_old = 'P';

const char herbivore\_symbol\_young = 'h';

const char herbivore\_symbol\_old = 'H';

const char grass\_symbol = '#';

const char river\_symbol = '~';

const char mountain\_symbol = '^';

int mountain\_population = 100; // количество гор

const int screen\_width = 200;

const int screen\_height = 30;

int predator\_population = 1200;

int herbivore\_population = 1200;

int grass\_population = 4000; // количество травы

const double grass\_regrowth\_rate\_summer = 0.031; // вероятность восстанавливаемости травы летом

const double grass\_regrowth\_rate\_springfall = 0.0155; // вероятность восстанавливаемости травы осенью и весной

const double grass\_regrowth\_rate\_winter = 0.0; // вероятность восстанавливаемости травы зимой

const double death\_probability\_per\_step = 0.1;

double current\_temperature = 20.0; // начальная температура

double tsunami\_probability = 0.3; // вероятность цунами

const int young\_herbivore\_hunger\_threshold = 3; // сколько шагов молодое травоядное животное может пройти без еды

const int young\_predator\_hunger\_threshold = 3; // сколько шагов молодой хищник может пройти без еды

const int old\_herbivore\_hunger\_threshold = 6; // сколько шагов старое травоядное животное может пройти без еды

const int old\_predator\_hunger\_threshold = 6; // сколько шагов старый хищник может пройти без еды

const int predator\_starvation\_threshold = 4; // сколько шагов хищник может пройти без еды

struct Animal {

char symbol;

int age;

int steps\_without\_eating;

};

int natural\_death\_predator\_count = 0;

int predator\_count = 0;

void placeRandomMountains(Animal grid[][screen\_width], int mountainCount) {

int count = 0;

while (count < mountainCount) {

int x = rand() % screen\_height;

int y = rand() % screen\_width;

if (grid[x][y].symbol == ' ') {

grid[x][y].symbol = mountain\_symbol;

++count;

}

}

}

void initializeGrid(Animal grid[][screen\_width]) {

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

grid[i][j].symbol = ' ';

grid[i][j].age = 0;

grid[i][j].steps\_without\_eating = 0;

}

}

// случайное место горы

placeRandomMountains(grid, mountain\_population);

}

void placeRandomRivers(Animal grid[][screen\_width], int riverCount) {

for (int i = 0; i < riverCount; ++i) {

int riverSize = rand() % 4 + 6; // случайный размер реки от 2х2 до 5х5

int riverX = rand() % (screen\_height - riverSize);

int riverY = rand() % (screen\_width - riverSize);

for (int x = riverX; x < riverX + riverSize; ++x) {

for (int y = riverY; y < riverY + riverSize; ++y) {

grid[x][y].symbol = river\_symbol;

}

}

}

}

void printGrid(const Animal grid[][screen\_width], int herbivore\_count, int predator\_count,

int young\_herbivore\_count, int young\_predator\_count, int old\_herbivore\_count, int old\_predator\_count,

int dead\_herbivore\_count, int remaining\_grass\_count, int current\_step, int current\_season,

int natural\_death\_predator\_count, double current\_temperature, int& old\_age\_death\_count) {

cout << "Herbivores: " << max(0, herbivore\_count)

<< " | Predators: " << max(0, predator\_count)

<< " | Young Herbivores (< 10 years): " << young\_herbivore\_count

<< " | Young Predators (< 10 years): " << young\_predator\_count

<< " | Old Herbivores (>= 10 years): " << old\_herbivore\_count

<< " | Old Predators (>= 10 years): " << old\_predator\_count

<< " | Dead Herbivores: " << dead\_herbivore\_count

<< " | Natural Deaths (Predators): " << natural\_death\_predator\_count

<< " | Old Age Deaths: " << old\_age\_death\_count

<< " | Remaining Grass: " << remaining\_grass\_count

<< " | Step: " << current\_step + 1

<< " | Current Temperature: " << current\_temperature << "^C"

<< " | Season: ";

switch (current\_season) {

case 0:

cout << "Summer";

break;

case 1:

cout << "Autumn";

break;

case 2:

cout << "Winter";

break;

case 3:

cout << "Spring";

break;

default:

cout << "+";

break;

}

cout << endl << endl;

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == grass\_symbol) {

if (current\_season == 1) {

cout << "\033[1;33m" << grid[i][j].symbol << "\033[0m"; // оранжевый цвет для травы осенью

}

// если сейчас зима - трава = белый цвет

else if (current\_season == 2) {

cout << "\033[1;37m" << grid[i][j].symbol << "\033[0m"; // белый цвет для травы осенью

}

else {

cout << "\033[1;32m" << grid[i][j].symbol << "\033[0m"; // зеленый цвет для травы

}

}

else if (grid[i][j].symbol == river\_symbol) {

cout << "\033[1;34m" << grid[i][j].symbol << "\033[0m"; // голубой цвет для реки

}

else if (grid[i][j].symbol == herbivore\_symbol\_young) {

cout << "\033[1;31m" << grid[i][j].symbol << "\033[0m"; // красный цвет для новорожденных особей травоядных

}

else if (grid[i][j].symbol == predator\_symbol\_young) {

cout << "\033[1;31m" << grid[i][j].symbol << "\033[0m"; // красный цвет для новорожденных особей хищников

}

else if (grid[i][j].symbol == mountain\_symbol) {

cout << "\033[1;37m" << grid[i][j].symbol << "\033[0m"; // белый цвет для гор

}

else {

cout << grid[i][j].symbol;

}

}

cout << endl;

}

}

void countAnimals(const Animal grid[][screen\_width], int& herbivore\_count, int& predator\_count,

int& young\_herbivore\_count, int& young\_predator\_count, int& old\_herbivore\_count, int& old\_predator\_count) {

herbivore\_count = 0;

predator\_count = 0;

young\_herbivore\_count = 0;

young\_predator\_count = 0;

old\_herbivore\_count = 0;

old\_predator\_count = 0;

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == herbivore\_symbol\_young) {

++herbivore\_count;

++young\_herbivore\_count;

}

else if (grid[i][j].symbol == herbivore\_symbol\_old) {

++herbivore\_count;

++old\_herbivore\_count;

}

else if (grid[i][j].symbol == predator\_symbol\_young) {

++predator\_count;

++young\_predator\_count;

}

else if (grid[i][j].symbol == predator\_symbol\_old) {

++predator\_count;

++old\_predator\_count;

}

}

}

}

void placeRandomAnimals(Animal grid[][screen\_width], char young\_symbol, char old\_symbol, int total\_population) {

for (int i = 0; i < total\_population; ++i) {

int x = rand() % screen\_height;

int y = rand() % screen\_width;

int age = rand() % 20 + 1; // устанавливаем возраст случайным образом от 1 до 20 лет

if (age <= 10) {

grid[x][y].symbol = young\_symbol; // если до 10 лет - молодая особь

}

else {

grid[x][y].symbol = old\_symbol; // если после 10 лет - взрослая особь

}

grid[x][y].age = age;

}

}

void placeRandomGrass(Animal grid[][screen\_width], int population) {

for (int i = 0; i < population; ++i) {

int x = rand() % screen\_height;

int y = rand() % screen\_width;

grid[x][y].symbol = grass\_symbol;

grid[x][y].age = 0; // возраст травы не имеет значения - установлен в 0

}

}

void moveRandomly(Animal grid[][screen\_width], char animal\_symbol) {

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == animal\_symbol) {

grid[i][j].symbol = ' ';

// движение в случайном направлении

int direction = rand() % 4;

int newI = i;

int newJ = j;

switch (direction) {

case 0: // двигаться вверх

if ((i > 0) && (grid[i - 1][j].symbol != river\_symbol) && (grid[i - 1][j].symbol != mountain\_symbol)) {

newI = i - 1;

}

break;

case 1: // двигаться вниз

if ((i < screen\_height - 1) && (grid[i + 1][j].symbol != river\_symbol) && (grid[i + 1][j].symbol != mountain\_symbol)) {

newI = i + 1;

}

break;

case 2: // двигаться влево

if ((j > 0) && (grid[i][j - 1].symbol != river\_symbol) && (grid[i][j - 1].symbol != mountain\_symbol)) {

newJ = j - 1;

}

break;

case 3: // двигаться направо

if ((j < screen\_width - 1) && (grid[i][j + 1].symbol != river\_symbol) && (grid[i][j + 1].symbol != mountain\_symbol)) {

newJ = j + 1;

}

break;

}

// если новая позиция = трава = отменяем изменения

if (grid[newI][newJ].symbol == grass\_symbol) {

grid[newI][newJ].steps\_without\_eating = 0;

}

grid[newI][newJ].symbol = animal\_symbol; // обновляем позицию

grid[newI][newJ].age = grid[i][j].age + 1; // увеличиваем возраст

}

}

}

}

void eatGrass(Animal grid[][screen\_width], int x, int y, int& remaining\_grass\_count) {

if (grid[x][y].symbol == grass\_symbol && grid[x][y].symbol != mountain\_symbol) {

grid[x][y].symbol = ' '; // очищаем блок травы

--remaining\_grass\_count;

}

}

void updateTemperature(double& current\_temperature, int steps, int current\_season) {

int random\_temperature\_change = 0;

switch (current\_season) {

case 0: // лето

random\_temperature\_change = (steps / 3) % 2 == 0 ? rand() % 13 + 18 : -(rand() % 13 + 1);

break;

case 1: // осень

random\_temperature\_change = (steps / 3) % 2 == 0 ? min(rand() % 11, 10) : -(rand() % 14 + 4);

// проверка = что при изменении температуры температура не поднимается выше 10 градусов

if (current\_temperature + random\_temperature\_change > 10) {

random\_temperature\_change = max(-10.0, 10.0 - current\_temperature);

}

break;

case 2: // зима

random\_temperature\_change = (steps / 3) % 2 == 0 ? max(-(rand() % 21 + 10), -30) : min(rand() % 26 - 25, -10);

break;

case 3: // весна

random\_temperature\_change = (steps / 3) % 2 == 0 ? min(rand() % 11 + 5, 15) : max(-(rand() % 11 + 5), -10);

// проверка = что при изменении температуры температура не поднимается выше 20 градусов

if (current\_temperature + random\_temperature\_change > 20) {

random\_temperature\_change = max(-10.0, 20.0 - current\_temperature);

}

break;

}

// обновляем темпу

current\_temperature += random\_temperature\_change;

// проверяем что температура остается в разумных пределах (например = от -10 до 30 градусов)

current\_temperature = max(-10.0, min(30.0, current\_temperature));

}

bool isAdjacent(int x1, int y1, int x2, int y2, const Animal grid[][screen\_width]) {

return abs(x1 - x2) <= 1 && abs(y1 - y2) <= 1 && grid[x2][y2].symbol != mountain\_symbol;

}

void herbivoreEatGrass(Animal grid[][screen\_width], int herbivoreX, int herbivoreY, int& remaining\_grass\_count) {

for (int i = max(0, herbivoreX - 1); i < min(screen\_height, herbivoreX + 2); ++i) {

for (int j = max(0, herbivoreY - 1); j < min(screen\_width, herbivoreY + 2); ++j) {

if (grid[i][j].symbol == grass\_symbol && (i != herbivoreX || j != herbivoreY) && grid[i][j].symbol != river\_symbol && grid[i][j].symbol != mountain\_symbol) {

eatGrass(grid, i, j, remaining\_grass\_count);

}

}

}

}

void predatorEatHerbivore(Animal grid[][screen\_width], int predatorX, int predatorY, int herbivoreX,

int herbivoreY, int& dead\_herbivore\_count, int& predator\_count) {

if (isAdjacent(predatorX, predatorY, herbivoreX, herbivoreY, grid) && grid[herbivoreX][herbivoreY].symbol != mountain\_symbol) {

// проверка на возраст хищника и травоядного животного

bool is\_young\_predator = (grid[predatorX][predatorY].symbol == predator\_symbol\_young);

bool is\_young\_herbivore = (grid[herbivoreX][herbivoreY].symbol == herbivore\_symbol\_young);

if ((is\_young\_predator && is\_young\_herbivore) || (!is\_young\_predator && !is\_young\_herbivore)) {

// хищник и травоядное животное имеют одинаковую возрастную группу - хищник ест травоядное животное

grid[herbivoreX][herbivoreY].symbol = ' '; // мертвое травоядное - очищаем herbivore

++dead\_herbivore\_count;

grid[predatorX][predatorY].steps\_without\_eating = 0; // сбрасываем счетчик голода

}

else {

// хищники и травоядные животные принадлежат к разным возрастным группам - хищник ест травоядное животное

if (grid[predatorX][predatorY].steps\_without\_eating >= 4) {

// хищник умирает - если не наелся

--predator\_count; // умирает

grid[predatorX][predatorY].symbol = ' '; // очищаем доску экран

grid[predatorX][predatorY].steps\_without\_eating = 0; // сбрасываем счетчик голода

}

else {

// хищник сьедает травоядное животное и сбрасывает счетчик голода

grid[predatorX][predatorY].symbol = ' '; // очищаем доску экран

grid[predatorX][predatorY].steps\_without\_eating = 0; // сбрасываем счетчик голода

}

}

}

}

void ageAnimals(Animal grid[][screen\_width], int& herbivore\_count, int& predator\_count, int& dead\_herbivore\_count,

int& current\_step, int& old\_age\_death\_count) {

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == predator\_symbol\_young || grid[i][j].symbol == herbivore\_symbol\_young) {

// увеличиваем счетчик возраста и голода каждые 24 шага (1 год)

if ((current\_step + 1) % 24 == 0) {

grid[i][j].age += 2; // увеличиваем возраст на 2

grid[i][j].steps\_without\_eating += 1;

if (grid[i][j].age >= 480) {

// если хищник достиг возраста 20 = он обязательно должен умереть

if (rand() % 100 < 10) { // вероятность смерти

--predator\_count;

grid[i][j].symbol = ' ';

grid[i][j].age = 0;

++old\_age\_death\_count;

}

}

else if (grid[i][j].age >= 350) {

// если хищник достиг возраста 10 = он становится старым

if (grid[i][j].symbol == predator\_symbol\_young) {

grid[i][j].symbol = predator\_symbol\_old;

}

else {

grid[i][j].symbol = herbivore\_symbol\_old;

}

}

}

}

}

}

}

void countYoungAnimals(const Animal grid[][screen\_width], int& young\_herbivore\_count, int& young\_predator\_count) {

young\_herbivore\_count = 0;

young\_predator\_count = 0;

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == herbivore\_symbol\_young && grid[i][j].age < 10) {

++young\_herbivore\_count;

}

else if (grid[i][j].symbol == predator\_symbol\_young && grid[i][j].age < 10) {

++young\_predator\_count;

}

}

}

}

void reproduce(Animal grid[][screen\_width], int x1, int y1, int x2, int y2, char young\_symbol, char old\_symbol,

int& herbivore\_count, int& predator\_count) {

// проверка - принадлежат ли два животного к одному типу и находятся ли они рядом

if ((grid[x1][y1].symbol == grid[x2][y2].symbol) && isAdjacent(x1, y1, x2, y2, grid)) {

if (rand() % 100 < 0.0005) {

for (int i = max(0, x1 - 1); i < min(screen\_height, x1 + 2); ++i) {

for (int j = max(0, y1 - 1); j < min(screen\_width, y1 + 2); ++j) {

if (grid[i][j].symbol == ' ') {

if (grid[x1][y1].symbol == predator\_symbol\_young || grid[x1][y1].symbol == herbivore\_symbol\_young) {

grid[i][j].symbol = (grid[x1][y1].age <= 10) ? young\_symbol : old\_symbol;

grid[i][j].age = 0;

grid[i][j].steps\_without\_eating = 0;

if (grid[i][j].symbol == young\_symbol) {

if (grid[i][j].symbol == herbivore\_symbol\_young) {

++herbivore\_count;

}

else {

++predator\_count;

}

}

}

break;

}

}

}

}

}

}

void checkStarvation(Animal grid[][screen\_width], int& herbivore\_count, int& predator\_count, int& dead\_herbivore\_count,

int& natural\_death\_predator\_count) {

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == predator\_symbol\_young || grid[i][j].symbol == predator\_symbol\_old) {

int hunger\_threshold = (grid[i][j].symbol == predator\_symbol\_young) ? young\_predator\_hunger\_threshold : old\_predator\_hunger\_threshold;

int starvation\_counter = (grid[i][j].symbol == predator\_symbol\_young) ? predator\_starvation\_threshold : predator\_starvation\_threshold \* 2;

if (grid[i][j].steps\_without\_eating >= hunger\_threshold) {

// хищник умирает от голода

--predator\_count;

++natural\_death\_predator\_count; // увеличиваем счетчик

grid[i][j].symbol = ' ';

grid[i][j].steps\_without\_eating = 0;

}

else if (grid[i][j].steps\_without\_eating >= starvation\_counter) {

// хищник умер от голода - не сьел нужное колво

--predator\_count;

grid[i][j].symbol = ' ';

grid[i][j].steps\_without\_eating = 0;

}

else {

// проверка успешно ли охотился хищник за последний месяц

bool has\_hunted = false;

for (int k = 0; k < screen\_height; ++k) {

for (int l = 0; l < screen\_width; ++l) {

if (grid[k][l].symbol == herbivore\_symbol\_young || grid[k][l].symbol == herbivore\_symbol\_old) {

if (isAdjacent(i, j, k, l, grid)) {

// хищник успешно охотился на травоядное животное

has\_hunted = true;

grid[i][j].steps\_without\_eating = 0; // сбрасываем счетчик голода

break;

}

}

}

if (has\_hunted) {

break;

}

}

// если хищник не охотился - увеличиваем счетчик голодания

if (!has\_hunted) {

grid[i][j].steps\_without\_eating++;

}

}

}

}

}

}

void checkStarvationHerbivores(Animal grid[][screen\_width], int& herbivore\_count, int& dead\_herbivore\_count,

int& remaining\_grass\_count) {

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == herbivore\_symbol\_young || grid[i][j].symbol == herbivore\_symbol\_old) {

int hunger\_threshold = (grid[i][j].symbol == herbivore\_symbol\_young) ? young\_herbivore\_hunger\_threshold : old\_herbivore\_hunger\_threshold;

if (grid[i][j].steps\_without\_eating >= hunger\_threshold) {

// травоядное умирает от голода

--herbivore\_count;

++dead\_herbivore\_count;

grid[i][j].symbol = ' ';

grid[i][j].steps\_without\_eating = 0;

}

else {

// проверка успешно ли ел хищник за последний месяц

bool has\_eaten = false;

for (int k = max(0, i - 1); k < min(screen\_height, i + 2); ++k) {

for (int l = max(0, j - 1); l < min(screen\_width, j + 2); ++l) {

if (grid[k][l].symbol == grass\_symbol && (k != i || l != j)) {

// травоядный успешно сьедает траву - не голодает временно

has\_eaten = true;

grid[i][j].steps\_without\_eating = 0; // сбрасываем счетчик голода

eatGrass(grid, k, l, remaining\_grass\_count);

break;

}

}

if (has\_eaten) {

break;

}

}

// если травоядное не поело - увеличиваем счетчик голодания

if (!has\_eaten) {

grid[i][j].steps\_without\_eating++;

}

}

}

}

}

}

int main() {

srand(static\_cast<unsigned>(time(0)));

Animal grid[screen\_height][screen\_width];

initializeGrid(grid);

// размещение животных - травы и рек

placeRandomAnimals(grid, predator\_symbol\_young, predator\_symbol\_old, predator\_population);

placeRandomAnimals(grid, herbivore\_symbol\_young, herbivore\_symbol\_old, herbivore\_population);

placeRandomGrass(grid, grass\_population);

placeRandomRivers(grid, 5); // количество рек

int herbivore\_count = 0;

int young\_herbivore\_count = 0;

int young\_predator\_count = 0;

int old\_herbivore\_count = 0;

int old\_predator\_count = 0;

int dead\_herbivore\_count = 0;

int remaining\_grass\_count = grass\_population;

int steps = 0;

int current\_season = 0;

double current\_regrowth\_rate = 0.0;

int old\_age\_death\_count = 0;

bool is\_tsunami = false;

bool game\_ended = false;

string user\_input; // переменная для хранения пользовательских данных

while (steps < 576 && !game\_ended) {

system(CLEAR\_SCREEN);

if (rand() % 100 < tsunami\_probability) { // вероятность цунами

is\_tsunami = true;

}

if (is\_tsunami) {

// все синие - цунами

cout << "\033[1;34m";

// все синие - цунами

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == ' ') {

cout << ' ';

}

else {

cout << grid[i][j].symbol;

}

}

cout << endl;

}

// сбрасываем цвет терминала

cout << "\033[0m";

// удялаем всех живых животных из сетки

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == predator\_symbol\_young || grid[i][j].symbol == predator\_symbol\_old ||

grid[i][j].symbol == herbivore\_symbol\_young || grid[i][j].symbol == herbivore\_symbol\_old) {

grid[i][j].symbol = ' ';

}

}

}

game\_ended = true;

}

else {

// вывод информации о симуляции - счетчик

printGrid(grid, herbivore\_count, predator\_count, young\_herbivore\_count, young\_predator\_count, old\_herbivore\_count,

old\_predator\_count, dead\_herbivore\_count, remaining\_grass\_count, steps, current\_season,

natural\_death\_predator\_count, current\_temperature, old\_age\_death\_count);

cout << "\nEnter 'exit'/'EXIT' to exit the game. Or press Enter move animals... ";

getline(cin, user\_input);

if (user\_input == "exit" or user\_input == "EXIT") {

cout << "\nokay. goodbye :( \n";

break;

}

// движение животных и размножение

moveRandomly(grid, predator\_symbol\_young);

moveRandomly(grid, predator\_symbol\_old);

moveRandomly(grid, herbivore\_symbol\_young);

moveRandomly(grid, herbivore\_symbol\_old);

// рождение детей

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == predator\_symbol\_young || grid[i][j].symbol == predator\_symbol\_old) {

for (int k = 0; k < screen\_height; ++k) {

for (int l = 0; l < screen\_width; ++l) {

if (grid[k][l].symbol == predator\_symbol\_young || grid[k][l].symbol == predator\_symbol\_old) {

reproduce(grid, i, j, k, l, predator\_symbol\_young, predator\_symbol\_old, herbivore\_count, predator\_count);

}

}

}

}

else if (grid[i][j].symbol == herbivore\_symbol\_young || grid[i][j].symbol == herbivore\_symbol\_old) {

for (int k = 0; k < screen\_height; ++k) {

for (int l = 0; l < screen\_width; ++l) {

if (grid[k][l].symbol == herbivore\_symbol\_young || grid[k][l].symbol == herbivore\_symbol\_old) {

reproduce(grid, i, j, k, l, herbivore\_symbol\_young, herbivore\_symbol\_old, herbivore\_count, predator\_count);

}

}

}

}

}

}

// проверка на голод перед тем - как животные поедят

checkStarvation(grid, herbivore\_count, predator\_count, dead\_herbivore\_count, natural\_death\_predator\_count);

// травоядные едят траву после шагов

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == herbivore\_symbol\_young || grid[i][j].symbol == herbivore\_symbol\_old) {

herbivoreEatGrass(grid, i, j, remaining\_grass\_count);

}

}

}

// хищники едят травоядных

for (int i = 0; i < screen\_height; ++i) {

for (int j = 0; j < screen\_width; ++j) {

if (grid[i][j].symbol == predator\_symbol\_young || grid[i][j].symbol == predator\_symbol\_old) {

for (int k = 0; k < screen\_height; ++k) {

for (int l = 0; l < screen\_width; ++l) {

if (grid[k][l].symbol == herbivore\_symbol\_young || grid[k][l].symbol == herbivore\_symbol\_old) {

predatorEatHerbivore(grid, i, j, k, l, dead\_herbivore\_count, predator\_count);

}

}

}

}

}

}

// увеличение возраста на 1 - животных кажждые 24 шага

ageAnimals(grid, herbivore\_count, predator\_count, dead\_herbivore\_count, steps, old\_age\_death\_count);

// счет молодых особей

countYoungAnimals(grid, young\_herbivore\_count, young\_predator\_count);

// проверка травоядных на предмет голодания

checkStarvationHerbivores(grid, herbivore\_count, dead\_herbivore\_count, remaining\_grass\_count);

// расчет отрастание травы на основе текущего сезона

double current\_regrowth\_rate = 0.0;

switch (current\_season) {

case 0: // лето

current\_regrowth\_rate = grass\_regrowth\_rate\_summer;

break;

case 1: // осень

current\_regrowth\_rate = grass\_regrowth\_rate\_springfall;

break;

case 2: // зима

current\_regrowth\_rate = grass\_regrowth\_rate\_winter;

break;

case 3: // весна

current\_regrowth\_rate = grass\_regrowth\_rate\_springfall;

break;

}

// случайная траву только в том случае - если скорость отрастания превышает 0

if (current\_regrowth\_rate > 0.0) {

int regrowth\_amount = static\_cast<int>(grass\_population \* current\_regrowth\_rate);

placeRandomGrass(grid, regrowth\_amount);

// пересчет отсчета

remaining\_grass\_count += regrowth\_amount;

}

// обновления сезона каждые 6 шагов

if ((steps + 1) % 6 == 0) {

current\_season = (current\_season + 1) % 4;

}

// вызов countAnimals для обновления подсчета

countAnimals(grid, herbivore\_count, predator\_count, young\_herbivore\_count, young\_predator\_count, old\_herbivore\_count, old\_predator\_count);

// обновление температуры

updateTemperature(current\_temperature, steps, current\_season);

++steps;

}

}

return 0;

}



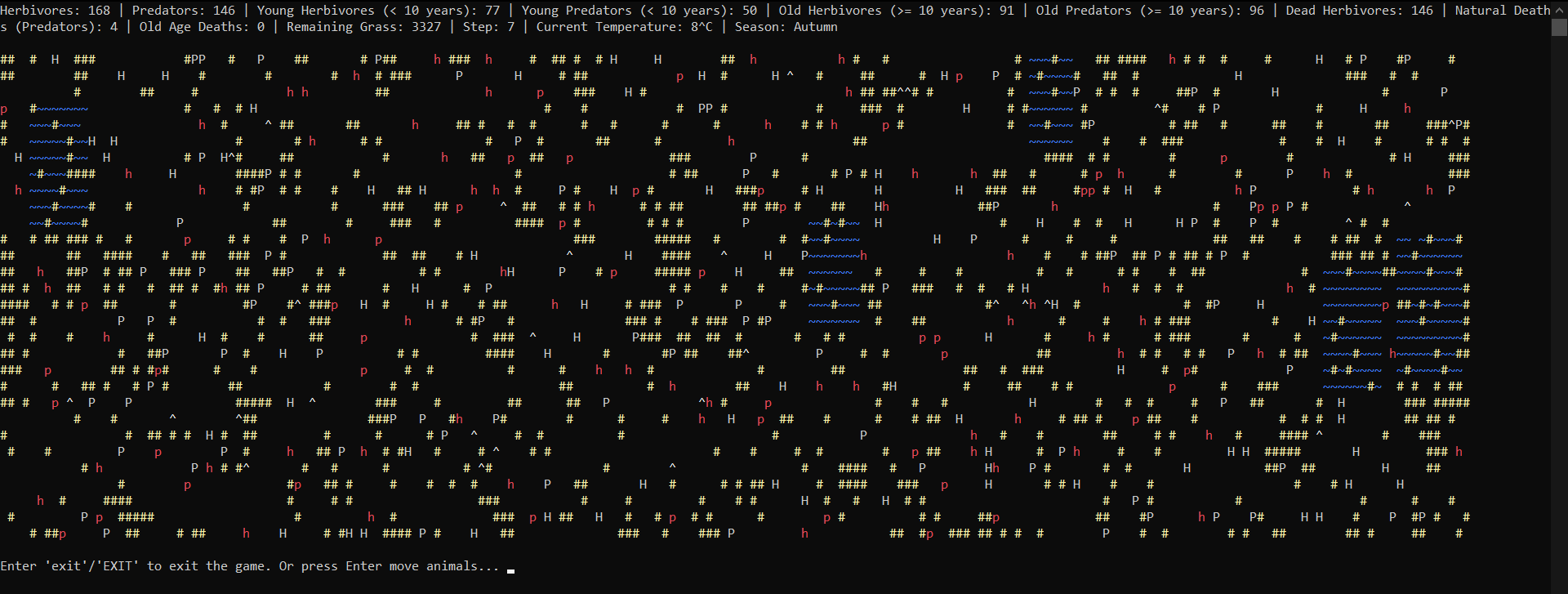
2 шаг:



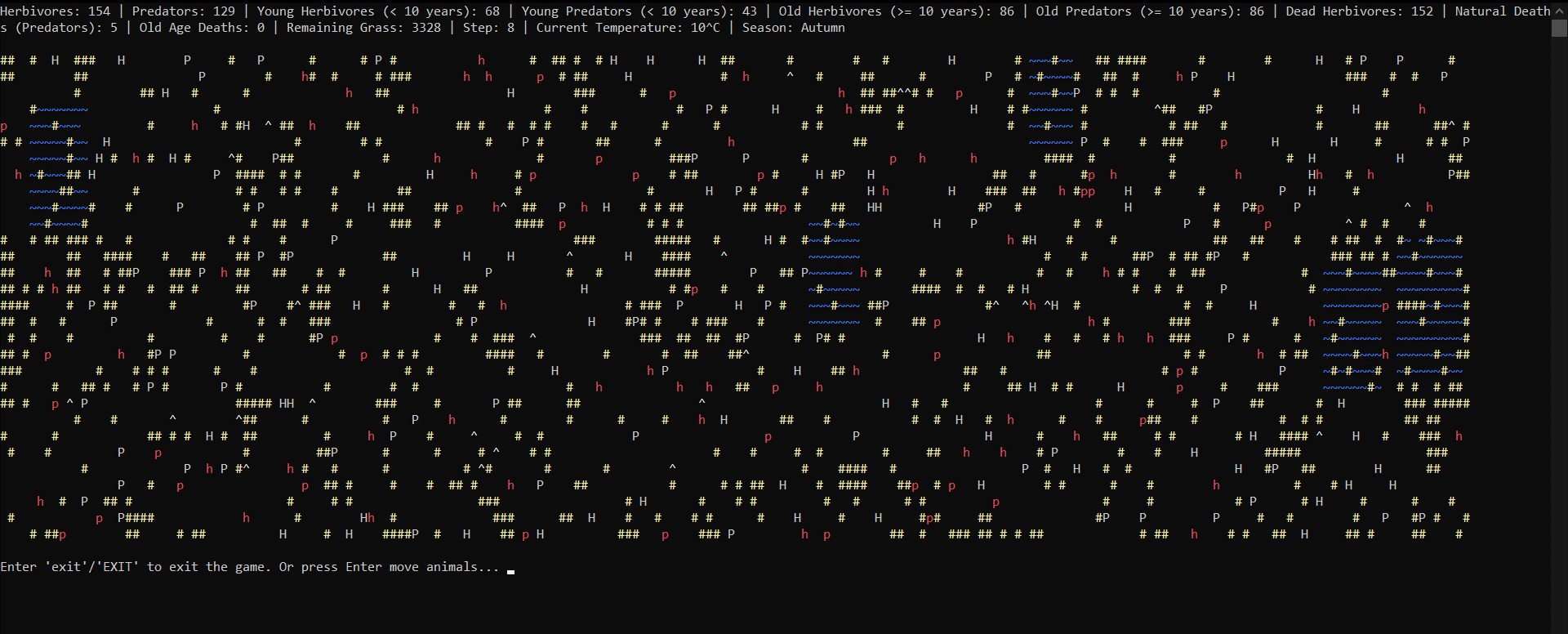
3 шаг:



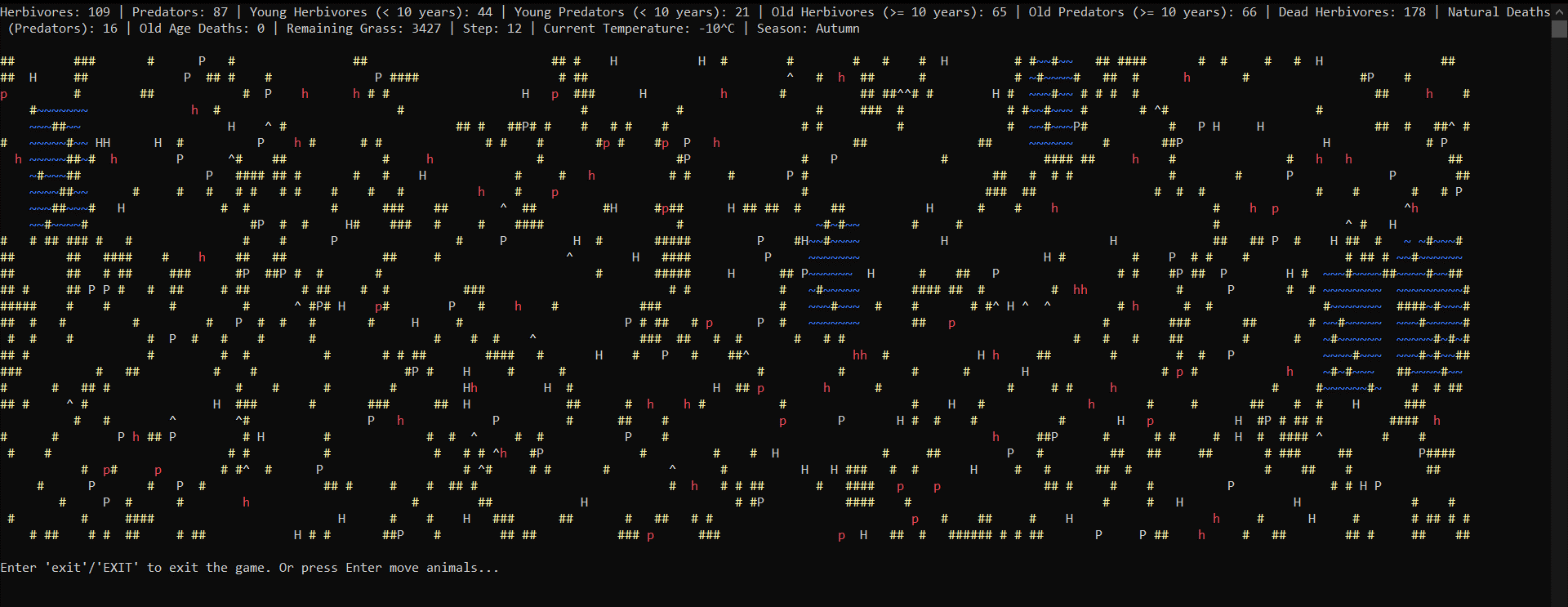
7 шаг:



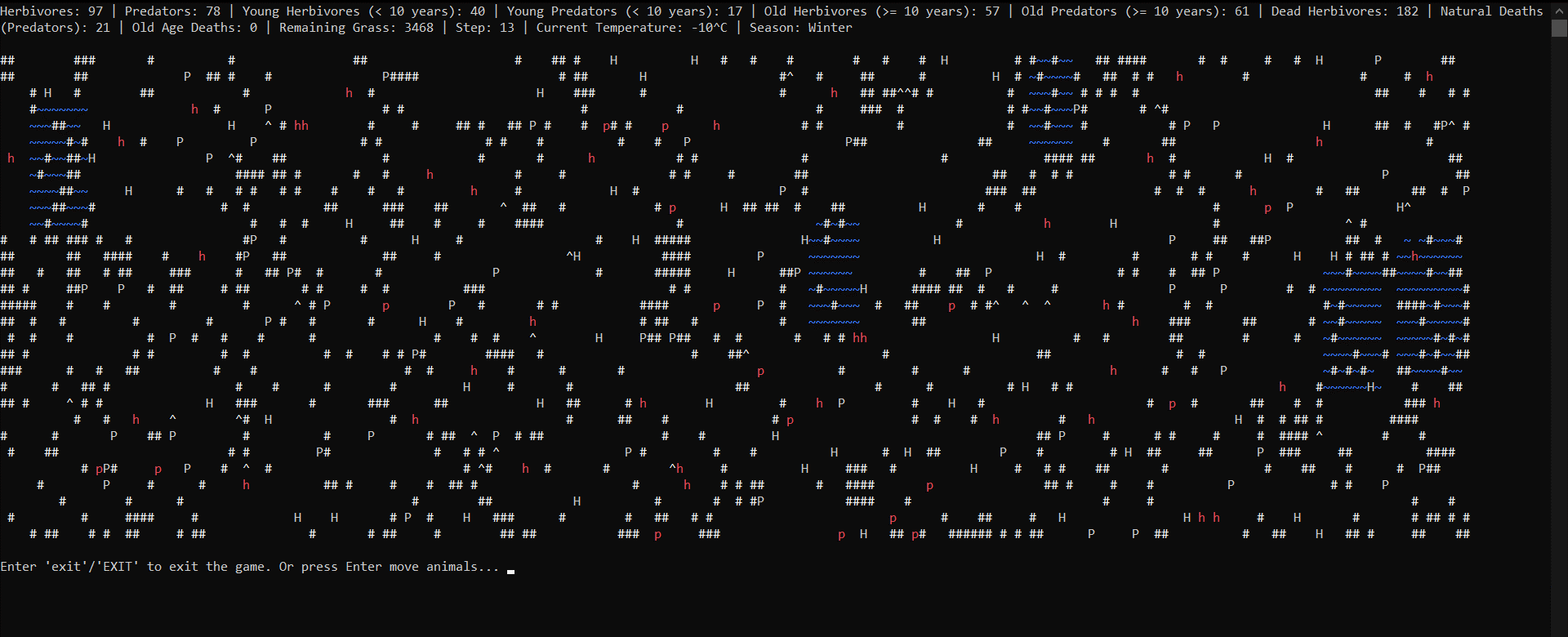
8 шаг:



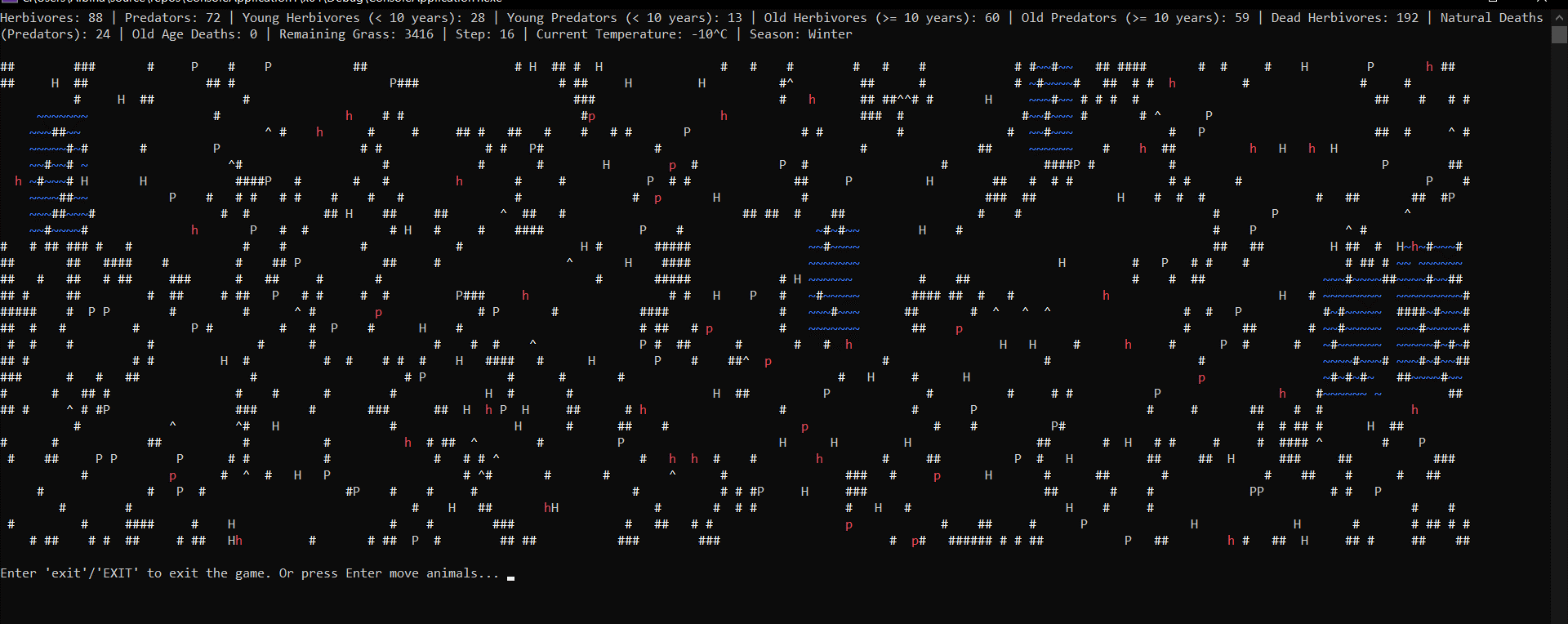
12 шаг:



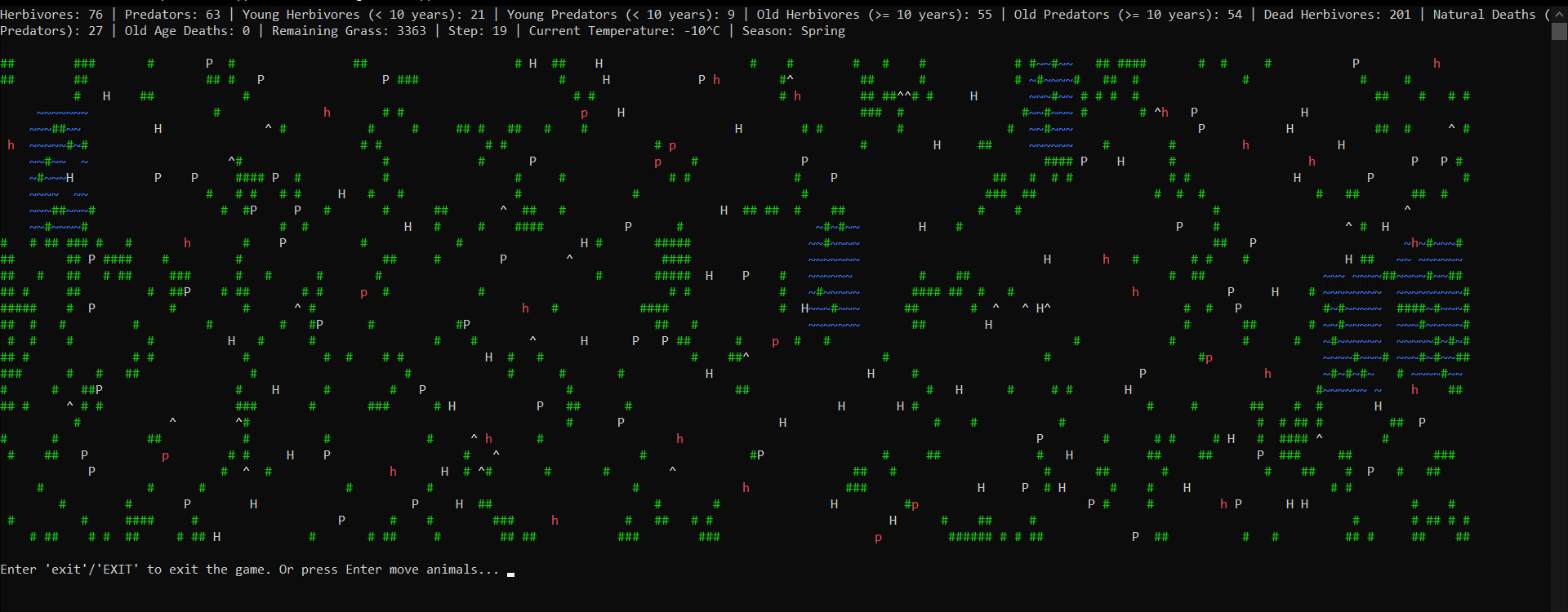
13 шаг:



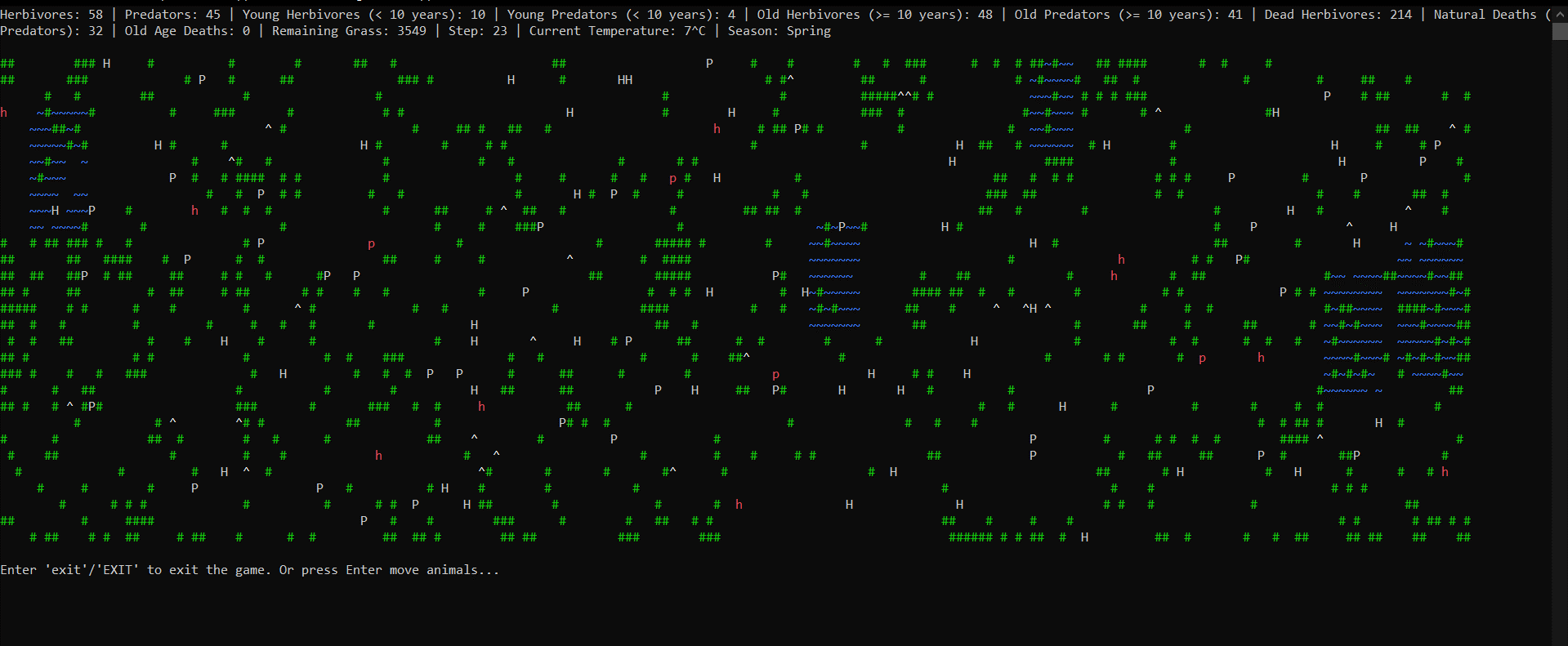
16 шаг:



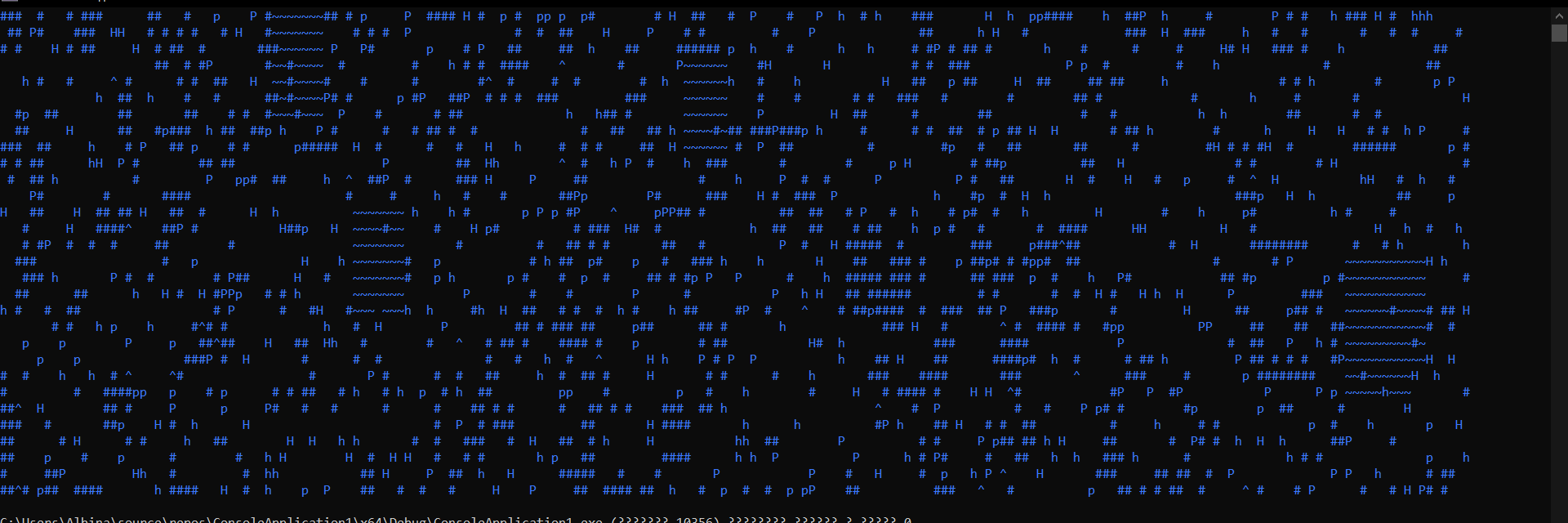
19 шаг:



23 шаг:



Пример цунами:



#ifdef \_WIN32

#define CLEAR\_SCREEN "cls"

#else

#define CLEAR\_SCREEN "clear"

Определяется строка CLEAR\_SCREEN, которая содержит команду для очистки экрана.

const char predator\_symbol\_young = 'p'; - символ для отображения молодого хищника на экране

const char predator\_symbol\_old = 'P'; - символ для отображения старого хищника на экране

const char herbivore\_symbol\_young = 'h'; - символ для отображения молодого травоядного на экране

const char herbivore\_symbol\_old = 'H'; - символ для отображения старого травоядного на экране

const char grass\_symbol = '#'; - символ для отображения травы на экране

const char river\_symbol = '~'; - символ для отображения речки на экране

const char mountain\_symbol = '^'; - символ для отображения горы на экране

int mountain\_population = 100; // количество гор

const int screen\_width = 200; - ширина экрана

const int screen\_height = 30; - высота экрана

int predator\_population = 1200; - количество хищников

int herbivore\_population = 1200; - количество травоядных

int grass\_population = 4000; // количество травы

const double grass\_regrowth\_rate\_summer = 0.031; // вероятность восстанавливаемости травы летом

const double grass\_regrowth\_rate\_springfall = 0.0155; // вероятность восстанавливаемости травы осенью и весной

const double grass\_regrowth\_rate\_winter = 0.0; // вероятность восстанавливаемости травы зимой

const double death\_probability\_per\_step = 0.1;

double current\_temperature = 20.0; // начальная температура

double tsunami\_probability = 0.3; // вероятность цунами

const int young\_herbivore\_hunger\_threshold = 3; // сколько шагов молодое травоядное животное может пройти без еды

const int young\_predator\_hunger\_threshold = 3; // сколько шагов молодой хищник может пройти без еды

const int old\_herbivore\_hunger\_threshold = 6; // сколько шагов старое травоядное животное может пройти без еды

const int old\_predator\_hunger\_threshold = 6; // сколько шагов старый хищник может пройти без еды

const int predator\_starvation\_threshold = 4; // сколько шагов хищник может пройти без еды

struct Animal - структура данных для представления животных в программе. Она включает символ (symbol) для их отображения, возраст (age) и количество шагов без еды (steps\_without\_eating).

Функция placeRandomMountains(Animal grid[][screen\_width], int mountainCount) предназначена для случайного размещения гор на игровом поле. В ней:

1. grid: двумерный массив типа Animal, представляющий игровое поле.
2. mountainCount: количество гор, которые нужно разместить.

Функция initializeGrid(Animal grid[][screen\_width]) предназначена для иницилизации игрового поля значениями по умолчанию, заполняя его пробелами. Она случайным образом располагает горы с помощью вызова функции placeRandomMountains(grid, mountain\_population).

Функция placeRandomRivers(Animal grid[][screen\_width], int riverCount) предназначена для случайного размещения рек на игровом поле. В ней:

1. grid: двумерный массив типа Animal, представляющий игровое поле.
2. riverCount: количество рек, которые нужно разместить.

Функция printGrid(const Animal grid[][screen\_width], int herbivore\_count, int predator\_count,

int young\_herbivore\_count, int young\_predator\_count, int old\_herbivore\_count, int old\_predator\_count,

int dead\_herbivore\_count, int remaining\_grass\_count, int current\_step, int current\_season,

int natural\_death\_predator\_count, double current\_temperature, int& old\_age\_death\_count) выводит текущее состояние игрового поля на экран, включая информацию о популяции травоядных, хищников, сезоне, температуре и т.д., иными словами – счетчик. В зависимости от сезона (Лето, Зима, Весна, Осень) меняется цвет травы, животных, речкию.

Функция countAnimals(const Animal grid[][screen\_width], int& herbivore\_count, int& predator\_count,

int& young\_herbivore\_count, int& young\_predator\_count, int& old\_herbivore\_count, int& old\_predator\_count) подсчитывает количество животных различных видов на игровом поле. В ней:

1. grid: двумерный массив типа Animal, представляющий игровое поле.
2. herbivore\_count: переменная для хранения общего количества травоядных.
3. predator\_count: переменная для хранения общего количества хищников.
4. young\_herbivore\_count: переменная для хранения количества молодых травоядных.
5. young\_predator\_count: переменная для хранения количества молодых хищников.
6. old\_herbivore\_count: переменная для хранения количества старых травоядных.
7. old\_predator\_count: переменная для хранения количества старых хищников.

Функция placeRandomAnimals(Animal grid[][screen\_width], char young\_symbol, char old\_symbol, int total\_population) случайным образом размещает травоядных и хищников на игровом поле. В ней:

1. grid: двумерный массив типа Animal, представляющий игровое поле.
2. young\_symbol: символ для представления молодых особей.
3. old\_symbol: символ для представления старых особей.
4. total\_population: общее количество особей для размещения.

Функция placeRandomGrass(Animal grid[][screen\_width], int population) случайным образом размещает траву на игровом поле. В ней:

1. grid: двумерный массив типа Animal, представляющий игровое поле.
2. population: количество блоков травы для размещения.

Функция moveRandomly(Animal grid[][screen\_width], char animal\_symbol) отвечает за случайное перемещение животных по игровому полю. В ней:

1. grid: двумерный массив типа Animal, представляющий игровое поле.
2. animal\_symbol: символ, представляющий вид животного.

Функция eatGrass(Animal grid[][screen\_width], int x, int y, int& remaining\_grass\_count) предназначена для осуществления процесса поедания травы животными. В ней:

1. grid: двумерный массив типа Animal, представляющий игровое поле.
2. x, y: координаты блока травы, который животное собирается съесть.
3. remaining\_grass\_count: количество оставшейся травы на поле.

Функция updateTemperature(double& current\_temperature, int steps, int current\_season) предназначена для обновления температуры в симуляции. В ней:

1. Принимается текущая температура (current\_temperature), количество шагов (steps), и текущий сезон (current\_season).
2. В зависимости от сезона изменяется температура с учетом случайных факторов.
3. Проверяется, чтобы температура оставалась в разумных пределах (-10 до 30 градусов).

Функция isAdjacent(int x1, int y1, int x2, int y2, const Animal grid[][screen\_width]) предназначена для проверки, находятся ли две точки на соседних ячейках сетки. В ней:

1. Принимаются координаты двух точек (x1, y1, x2, y2) и сетка grid.
2. Возвращает true, если точки находятся на соседних ячейках и символ в ячейке (x2, y2) не равен символу горы.

Функция herbivoreEatGrass(Animal grid[][screen\_width], int herbivoreX, int herbivoreY, int& remaining\_grass\_count) предназначена для травоядных, чтобы они могли поедать траву вокруг себя. В ней:

1. Принимаются сетка grid, текущие координаты травоядного (herbivores, herbivoreY) и количество оставшейся травы (remaining\_grass\_count).
2. Проверяется окружающая трава в радиусе 3х3 и, если она найдена, вызывается функция eat Grass для уменьшения количества травы.

Функция predatorEatHerbivore(Animal grid[][screen\_width], int predatorX, int predatorY, int herbivoreX,

int herbivoreY, int& dead\_herbivore\_count, int& predator\_count) предназначена для определения взаимодействия между хищниками и травоядными. В ней:

1. Принимаются сетка grid и координаты хищника и травоядного.
2. Определяется, может ли хищник съесть травоядное в зависимости от их возрастных групп.
3. Обновляется состояние сетки в соответствии с результатом взаимодействия.

Функция ageAnimals(Animal grid[][screen\_width], int& herbivore\_count, int& predator\_count, int& dead\_herbivore\_count,

int& current\_step, int& old\_age\_death\_count) предназначена для обновления возраста и счетчика голода животных. В ней:

1. Принимаются сетка grid, счетчики различных видов животных, текущий шаг симуляции и счетчик естественных смертей.
2. Увеличивается возраст и счетчик голода каждого молодого животного каждые 24 шага.
3. Проверяется, стал ли хищник старым или достиг возраста 20, что приводит к его смерти.

Функция countYoungAnimals(const Animal grid[][screen\_width], int& young\_herbivore\_count, int& young\_predator\_count) предназначена для подсчета количества молодых животных в сетке. В ней:

1. Принимается сетка grid и счетчики молодых травоядных и молодых хищников.
2. Проходится по каждой ячейке сетки и увеличивает соответствующий счетчик, если обнаружено молодое животное.

Функция reproduce(Animal grid[][screen\_width], int x1, int y1, int x2, int y2, char young\_symbol, char old\_symbol,

int& herbivore\_count, int& predator\_count)предназначена для размножения животных. В ней:

1. Принимаются сетка grid, координаты двух животных, символы молодых и старых животных, а также счетчики животных каждого типа.
2. Проверяется возможность размножения с учетом вероятности и наличия свободного места рядом.
3. Если размножение происходит, создается новое молодое животное на свободном месте.

Функция checkStarvation(Animal grid[][screen\_width], int& herbivore\_count, int& predator\_count, int& dead\_herbivore\_count,

int& natural\_death\_predator\_count)предназначена для проверки голода у Функция хищников. В ней:

1. Принимается сетка grid и счетчики хищников и естественных смертей.
2. Проверяется счетчик голода каждого хищника и принимаются соответствующие меры, если голод превысил порог.

Функция checkStarvationHerbivores(Animal grid[][screen\_width], int& herbivore\_count, int& dead\_herbivore\_count,

int& remaining\_grass\_count) предназначена для проверки голода у травоядных. В ней:

1. Принимается сетка grid и счетчики травоядных и мертвых травоядных.
2. Проверяется счетчик голода каждого травоядного и принимаются соответствующие меры, если голод превысил порог.

Функция int main() предназначена для запуска эмуляции игры. В ней :

1. Инициализируется генератор случайных чисел на основе текущего времени.
2. Создается двумерный массив grid, представляющий собой экосистему, и инициализируется функцией initializeGrid.
3. Размещаются животные (хищники и травоядные), трава, и реки на сетке с помощью соответствующих функций placeRandomAnimals, placeRandomGrass и placeRandomRivers.
4. Инициализируются переменные, хранящие информацию о состоянии экосистемы, такие как количество животных разных видов, количество мертвых травоядных и т.д.
5. Запускается основной цикл симуляции, который продолжается, пока не достигнут максимальный предел шагов или не произошло событие цунами.
6. В каждой итерации цикла происходит обновление симуляции, включая перемещение животных, размножение, проверку на голод, питание, обновление возраста и другие события.
7. Выводится информация о текущем состоянии симуляции и предоставляется пользователю возможность взаимодействия с программой, например, выход из симуляции по запросу пользователя.