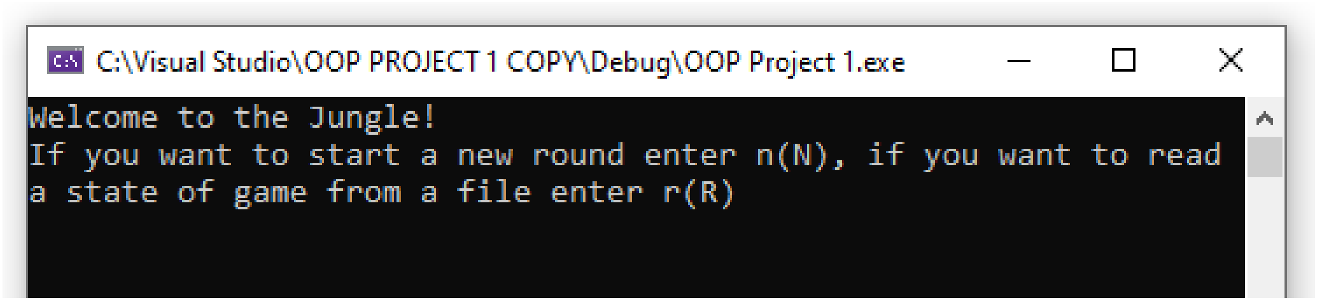
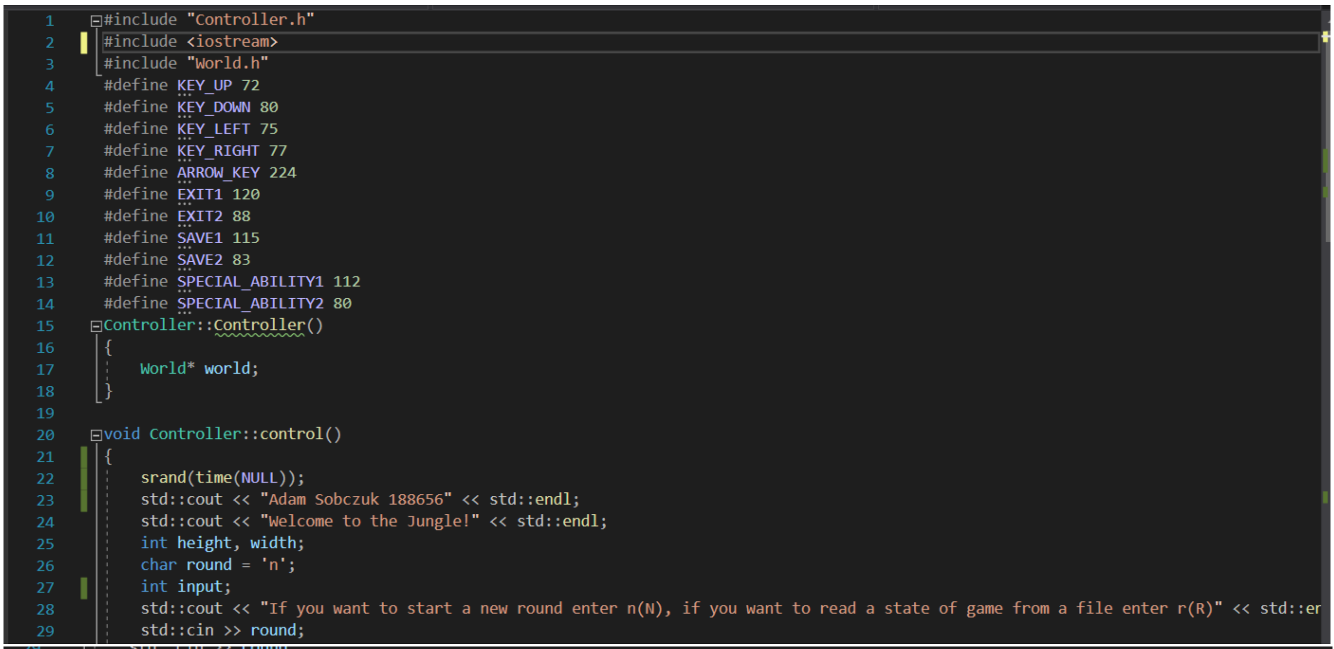
Firstly I Will show the console user interface, when we start the programme it shows:



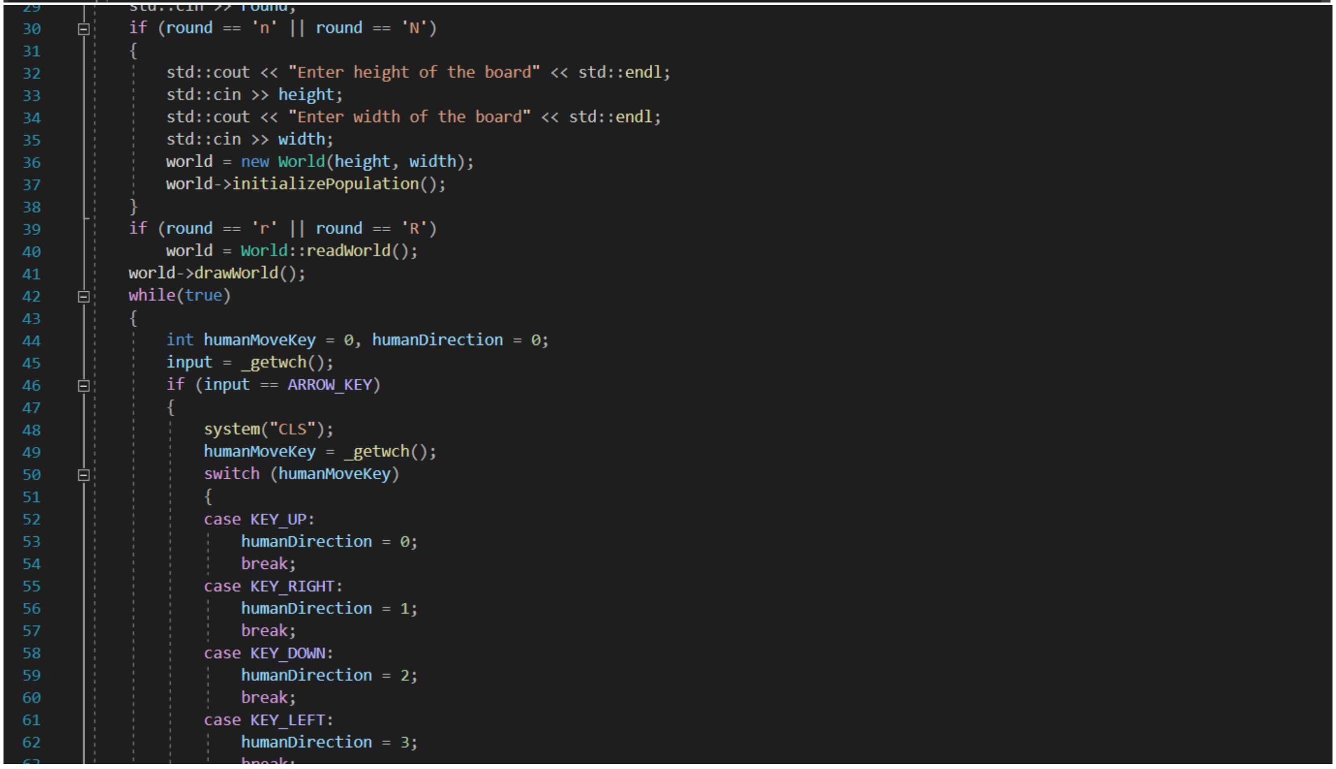
```
Welcome to the Jungle!
If you want to start a new round enter n(N), if you want to read
a state of game from a file enter r(R)
```

Class controller is responsible for showing the interface and handling the options its object is created in main cpp and control method is called:

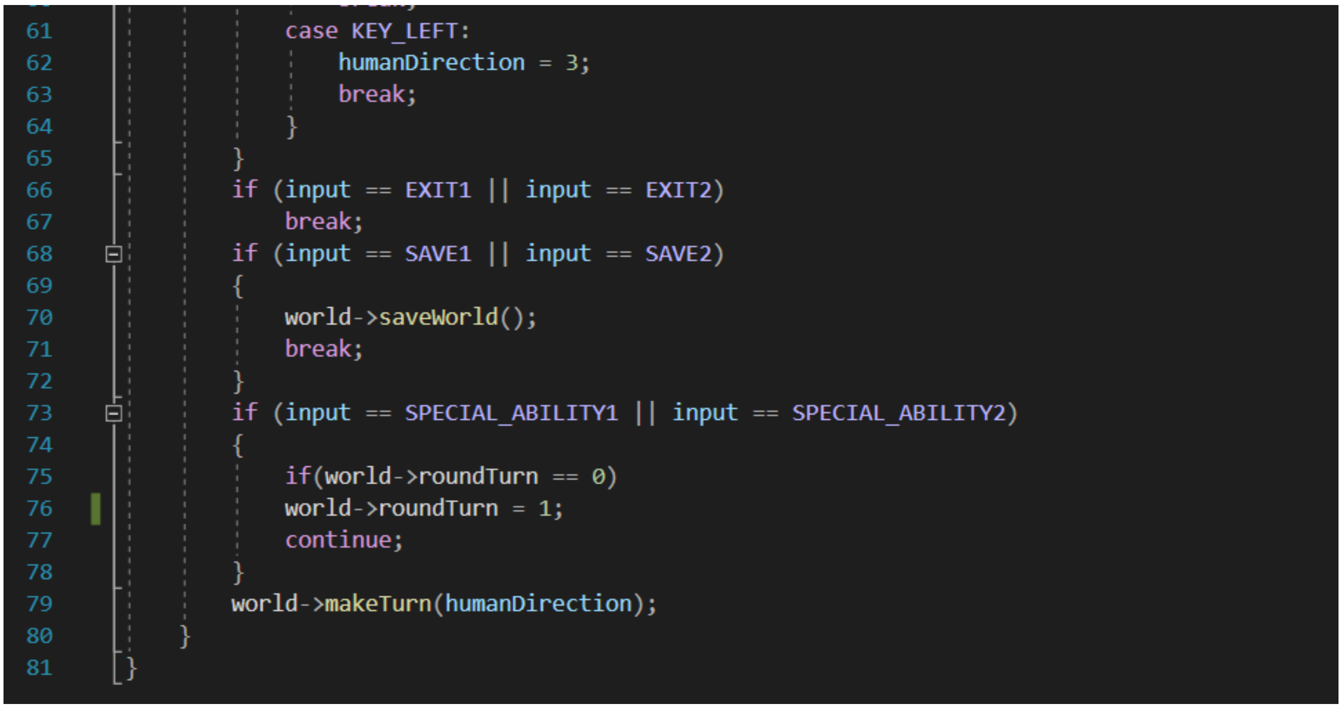


```cpp
#include "Controller.h"
#include <iostream>
#include "World.h"
#define KEY_UP 72
#define KEY_DOWN 80
#define KEY_LEFT 75
#define KEY_RIGHT 77
#define ARROW_KEY 224
#define EXIT1 120
#define EXIT2 88
#define SAVE1 115
#define SAVE2 83
#define SPECIAL_ABILITY1 112
#define SPECIAL_ABILITY2 80
Controller::Controller()
{
    World* world;
}

void Controller::control()
{
    srand(time(NULL));
    std::cout << "Adam Sobczuk 188656" << std::endl;
    std::cout << "Welcome to the Jungle!" << std::endl;
    int height, width;
    char round = 'n';
    int input;
    std::cout << "If you want to start a new round enter n(N), if you want to read a state of game from a file enter r(R)" << std::er
    std::cin >> round;
```



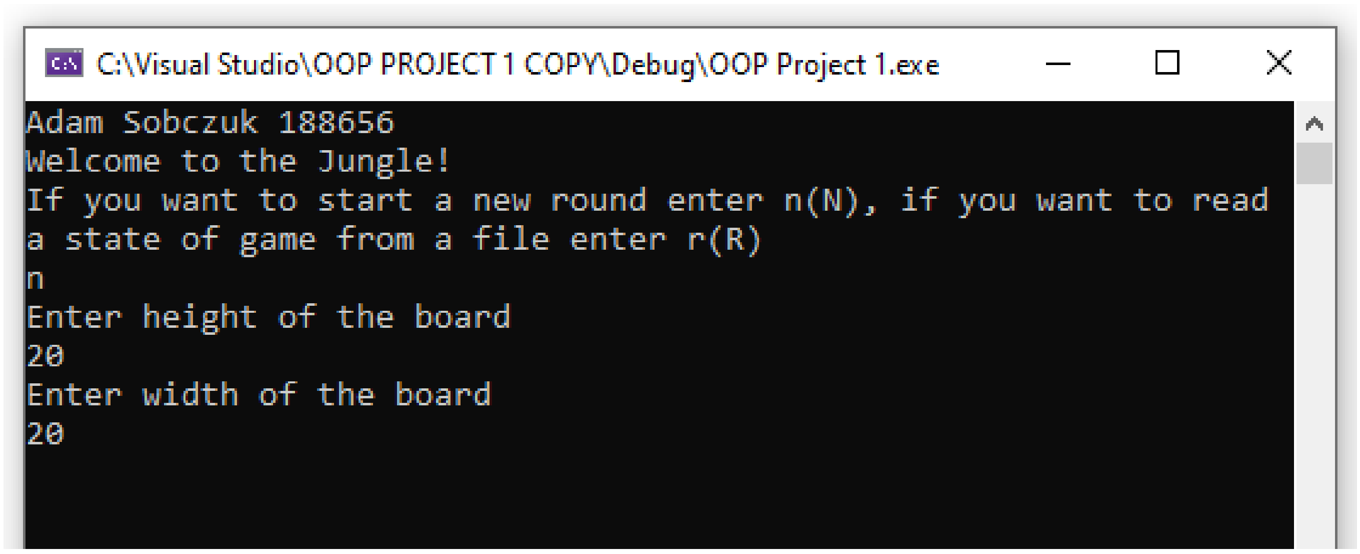
```cpp
    if (round == 'n' || round == 'N')
    {
        std::cout << "Enter height of the board" << std::endl;
        std::cin >> height;
        std::cout << "Enter width of the board" << std::endl;
        std::cin >> width;
        world = new World(height, width);
        world->initializePopulation();
    }
    if (round == 'r' || round == 'R')
        world = World::readWorld();
    world->drawWorld();
    while(true)
    {
        int humanMoveKey = 0, humanDirection = 0;
        input = _getwch();
        if (input == ARROW_KEY)
        {
            system("CLS");
            humanMoveKey = _getwch();
            switch (humanMoveKey)
            {
            case KEY_UP:
                humanDirection = 0;
                break;
            case KEY_RIGHT:
                humanDirection = 1;
                break;
            case KEY_DOWN:
                humanDirection = 2;
                break;
            case KEY_LEFT:
                humanDirection = 3;
```



```
            case KEY_LEFT:
                humanDirection = 3;
                break;
            }
        }
        if (input == EXIT1 || input == EXIT2)
            break;
        if (input == SAVE1 || input == SAVE2)
        {
            world->saveWorld();
            break;
        }
        if (input == SPECIAL_ABILITY1 || input == SPECIAL_ABILITY2)
        {
            if(world->roundTurn == 0)
            world->roundTurn = 1;
            continue;
        }
        world->makeTurn(humanDirection);
    }
}
```

It takes firstly two types of inputs as we can see in the screen from the console before

If we insert n or N a new game will start with a newly created board:

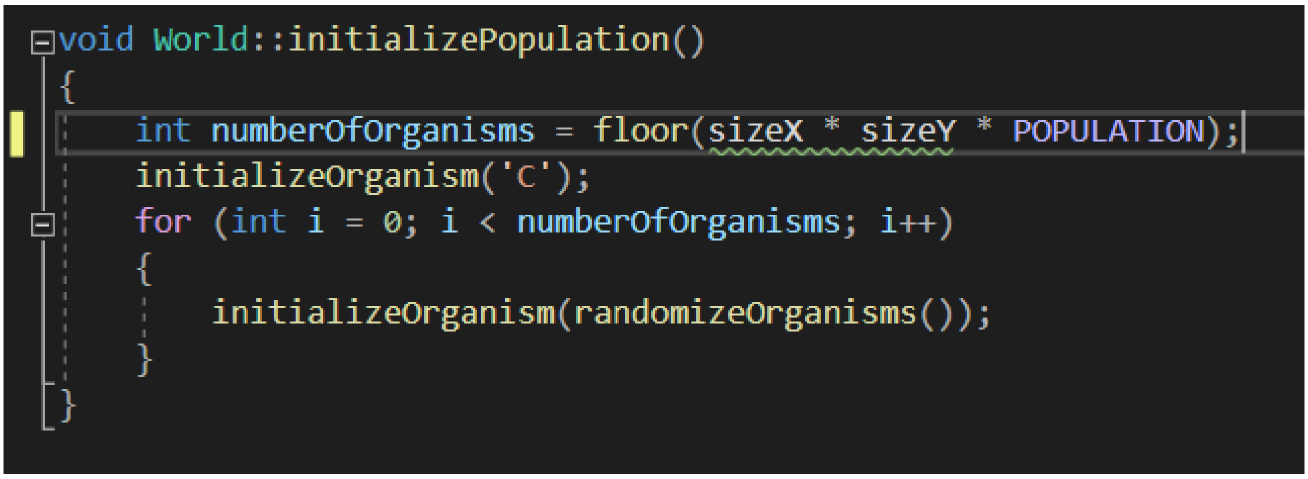


C:\Visual Studio\OOP PROJECT 1 COPY\Debug\OOP Project 1.exe

```
Adam Sobczuk 188656
Welcome to the Jungle!
If you want to start a new round enter n(N), if you want to read
a state of game from a file enter r(R)
n
Enter height of the board
20
Enter width of the board
20
```

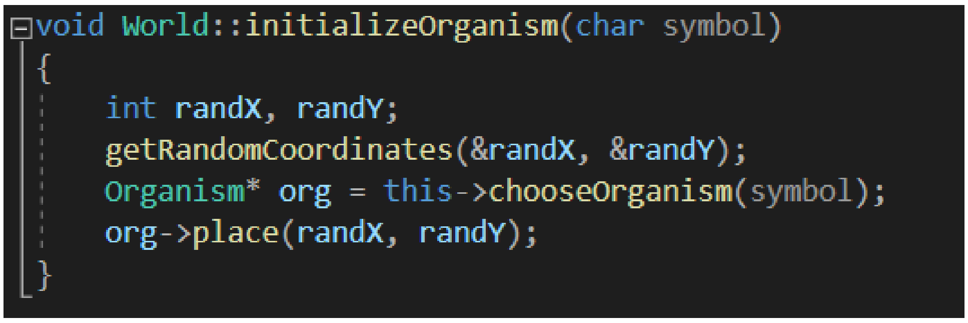
Before that it asks as for size of the board and when we type a object of world class is created:

Then called is the method initialize population which places several instances of organisms on the board randomly:



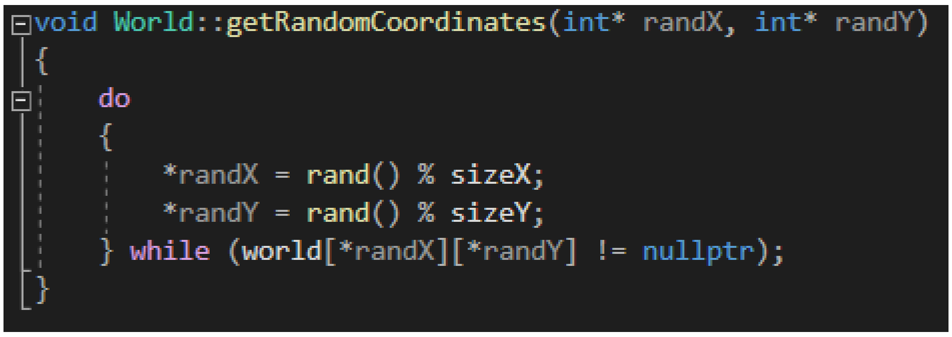
```cpp
void World::initializePopulation()
{
    int numberOfOrganisms = floor(sizeX * sizeY * POPULATION);
    initializeOrganism('C');
    for (int i = 0; i < numberOfOrganisms; i++)
    {
        initializeOrganism(randomizeOrganisms());
    }
}
```

It calls function initialize Organism which places an istance of human on the board:



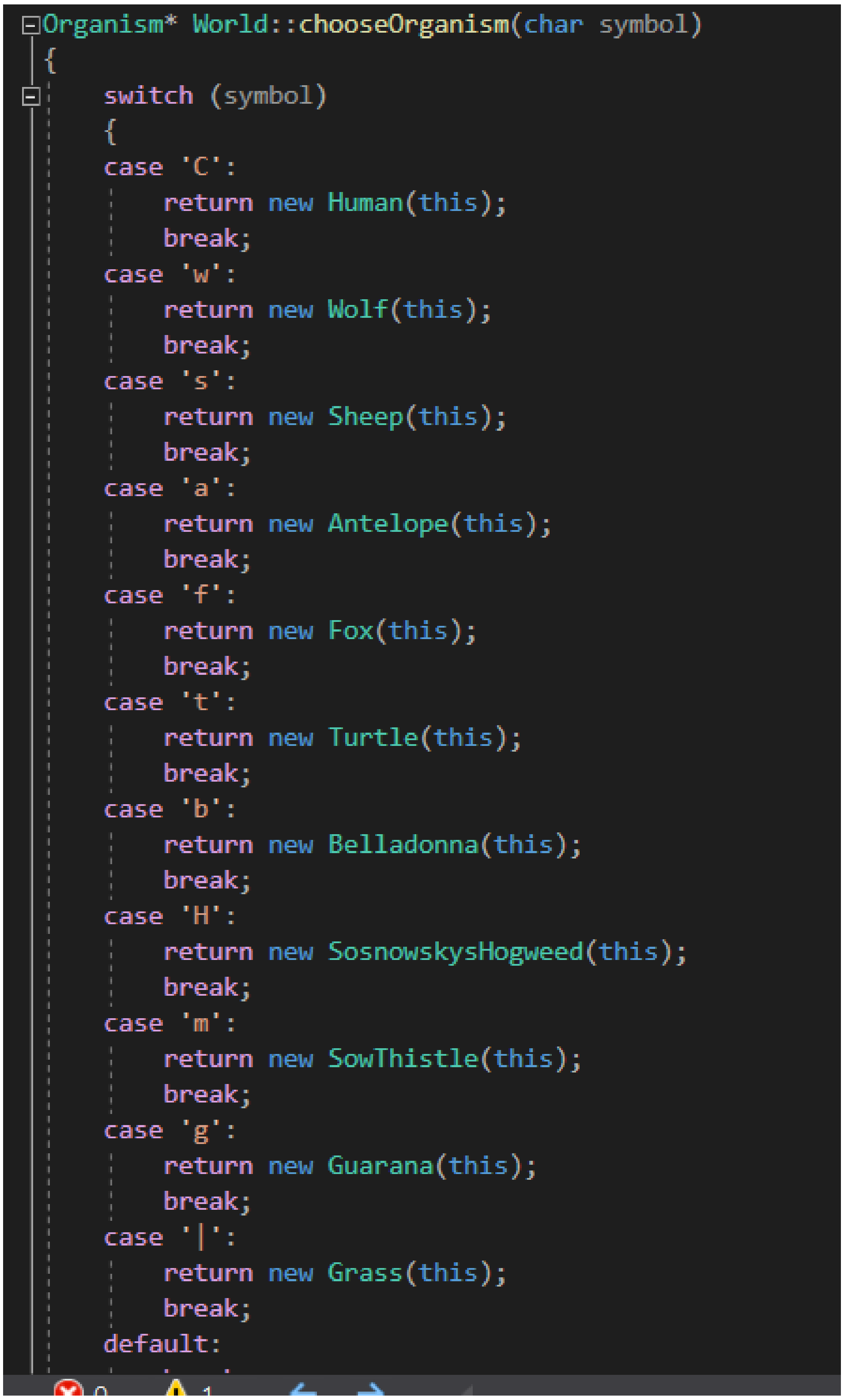
```cpp
void World::initializeOrganism(char symbol)
{
    int randX, randY;
    getRandomCoordinates(&randX, &randY);
    Organism* org = this->chooseOrganism(symbol);
    org->place(randX, randY);
}
```

Method getRandomCoordinates:



```cpp
void World::getRandomCoordinates(int* randX, int* randY)
{
    do
    {
        *randX = rand() % sizeX;
        *randY = rand() % sizeY;
    } while (world[*randX][*randY] != nullptr);
}
```

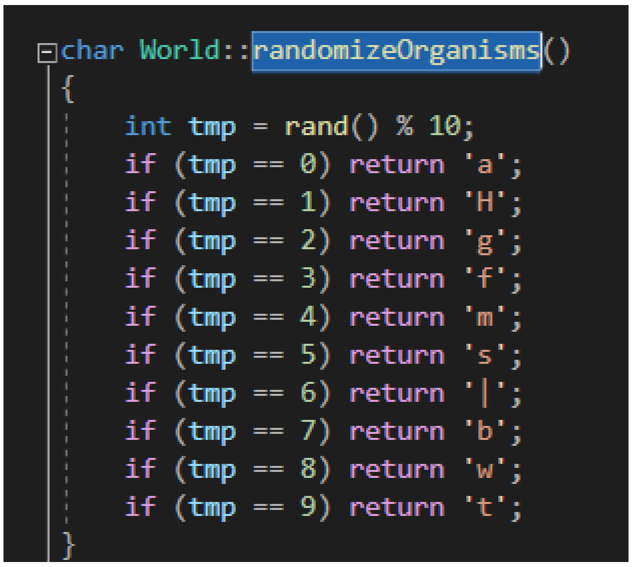
This function based on the symbol passed then chooses the Organism with choose organism method which should be returned to the board:



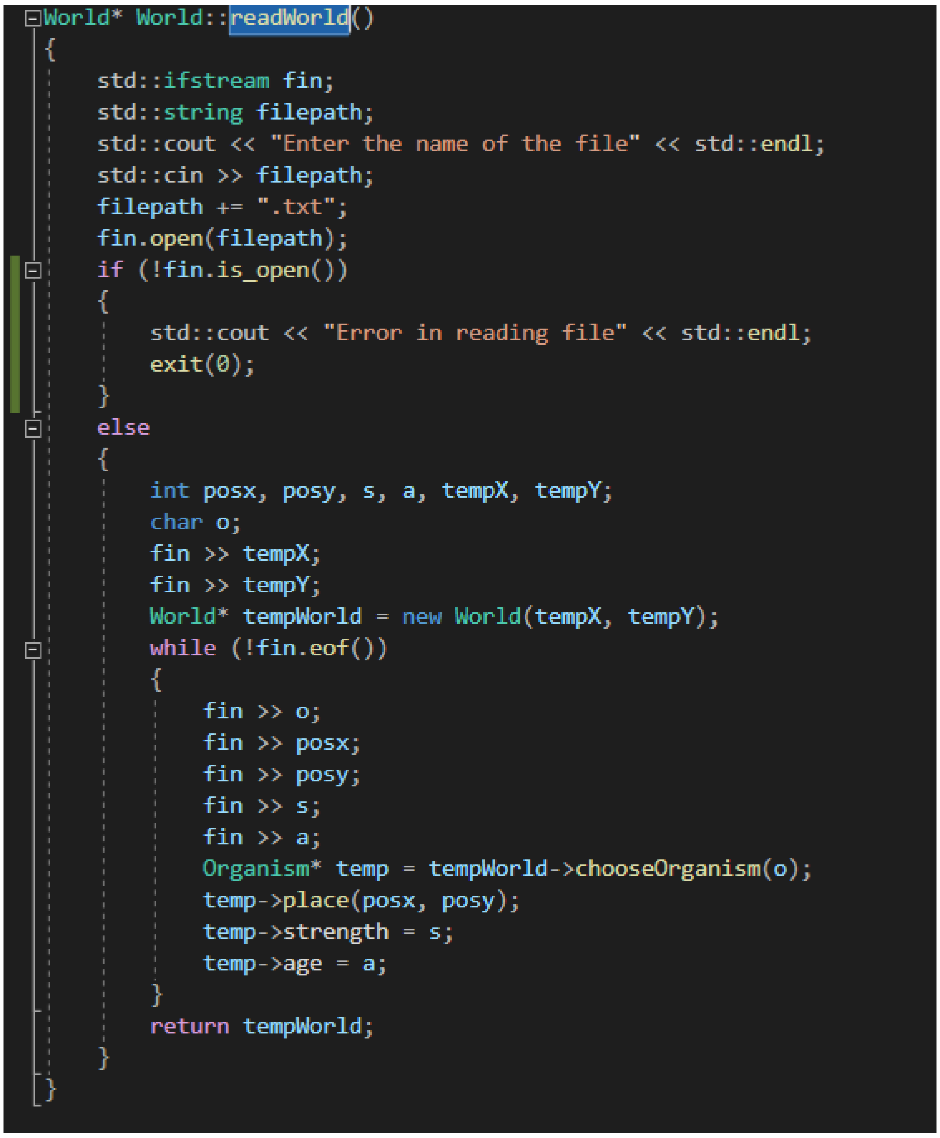
```cpp
Organism* World::chooseOrganism(char symbol)
{
    switch (symbol)
    {
    case 'C':
        return new Human(this);
        break;
    case 'w':
        return new Wolf(this);
        break;
    case 's':
        return new Sheep(this);
        break;
    case 'a':
        return new Antelope(this);
        break;
    case 'f':
        return new Fox(this);
        break;
    case 't':
        return new Turtle(this);
        break;
    case 'b':
        return new Belladonna(this);
        break;
    case 'H':
        return new SosnowskysHogweed(this);
        break;
    case 'm':
        return new SowThistle(this);
        break;
    case 'g':
        return new Guarana(this);
        break;
    case '|':
        return new Grass(this);
        break;
    default:
```

Then coming back to the initialize population method:

Based on the population parameter(in this project its set to 0.1) it initializes organisms at random places using the randomize organisms method, which returns organisms symbols that are used in choose organism method:

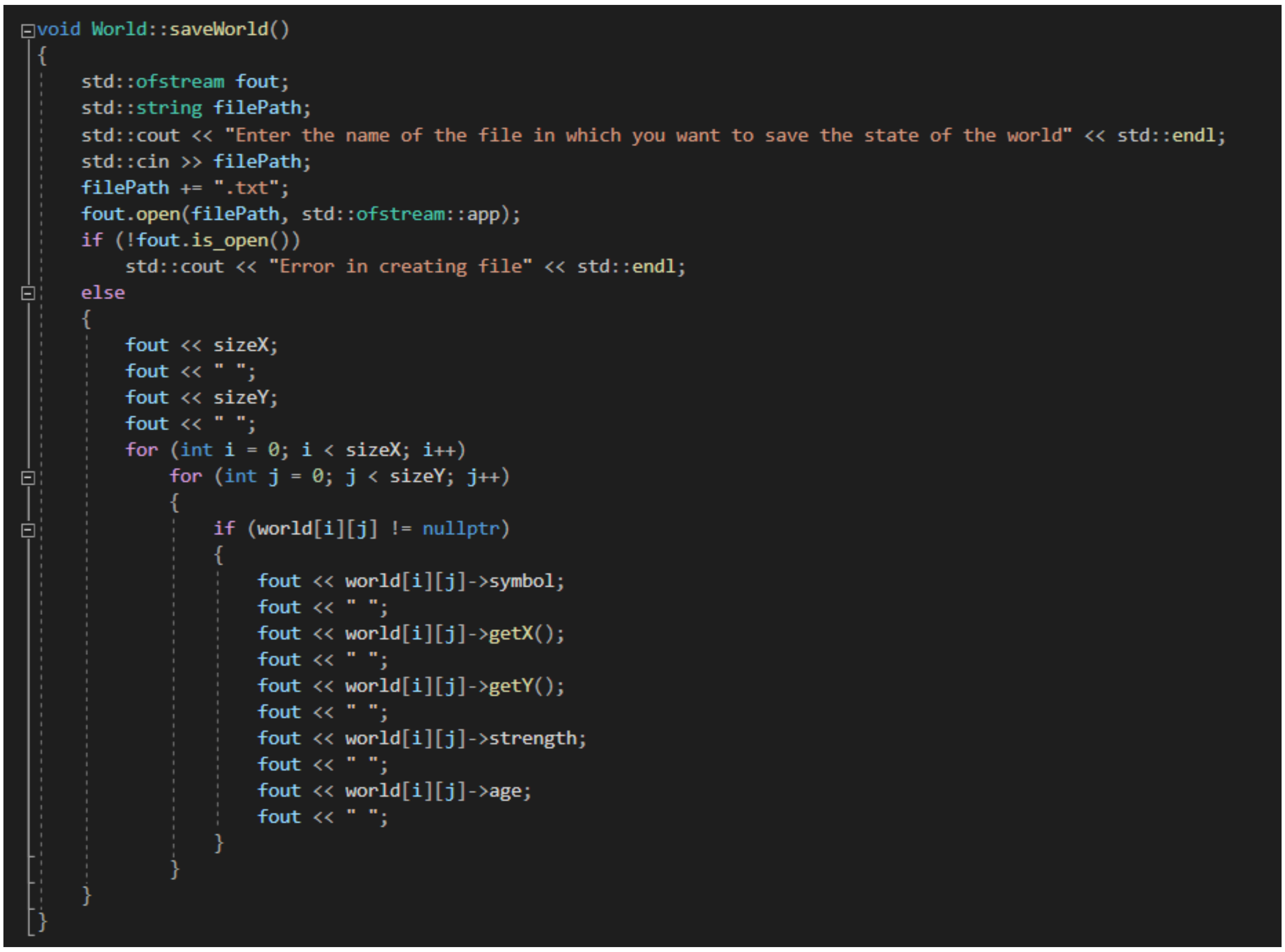

```cpp
char World::randomizeOrganisms()
{
    int tmp = rand() % 10;
    if (tmp == 0) return 'a';
    if (tmp == 1) return 'H';
    if (tmp == 2) return 'g';
    if (tmp == 3) return 'f';
    if (tmp == 4) return 'm';
    if (tmp == 5) return 's';
    if (tmp == 6) return '|';
    if (tmp == 7) return 'b';
    if (tmp == 8) return 'w';
    if (tmp == 9) return 't';
}
```

Second type of playing the game is reading a “save” file which has the instance of the world, which is called when we type r or R in the console:


```cpp
World* World::readWorld()
{
    std::ifstream fin;
    std::string filepath;
    std::cout << "Enter the name of the file" << std::endl;
    std::cin >> filepath;
    filepath += ".txt";
    fin.open(filepath);
    if (!fin.is_open())
    {
        std::cout << "Error in reading file" << std::endl;
        exit(0);
    }
    else
    {
        int posx, posy, s, a, tempX, tempY;
        char o;
        fin >> tempX;
        fin >> tempY;
        World* tempWorld = new World(tempX, tempY);
        while (!fin.eof())
        {
            fin >> o;
            fin >> posx;
            fin >> posy;
            fin >> s;
            fin >> a;
            Organism* temp = tempWorld->chooseOrganism(o);
            temp->place(posx, posy);
            temp->strength = s;
            temp->age = a;
        }
        return tempWorld;
    }
}
```

This method reads a file creates a world with a size given in a file and puts organisms on the board with its given parameters(that could change) such as age strength, x, y.

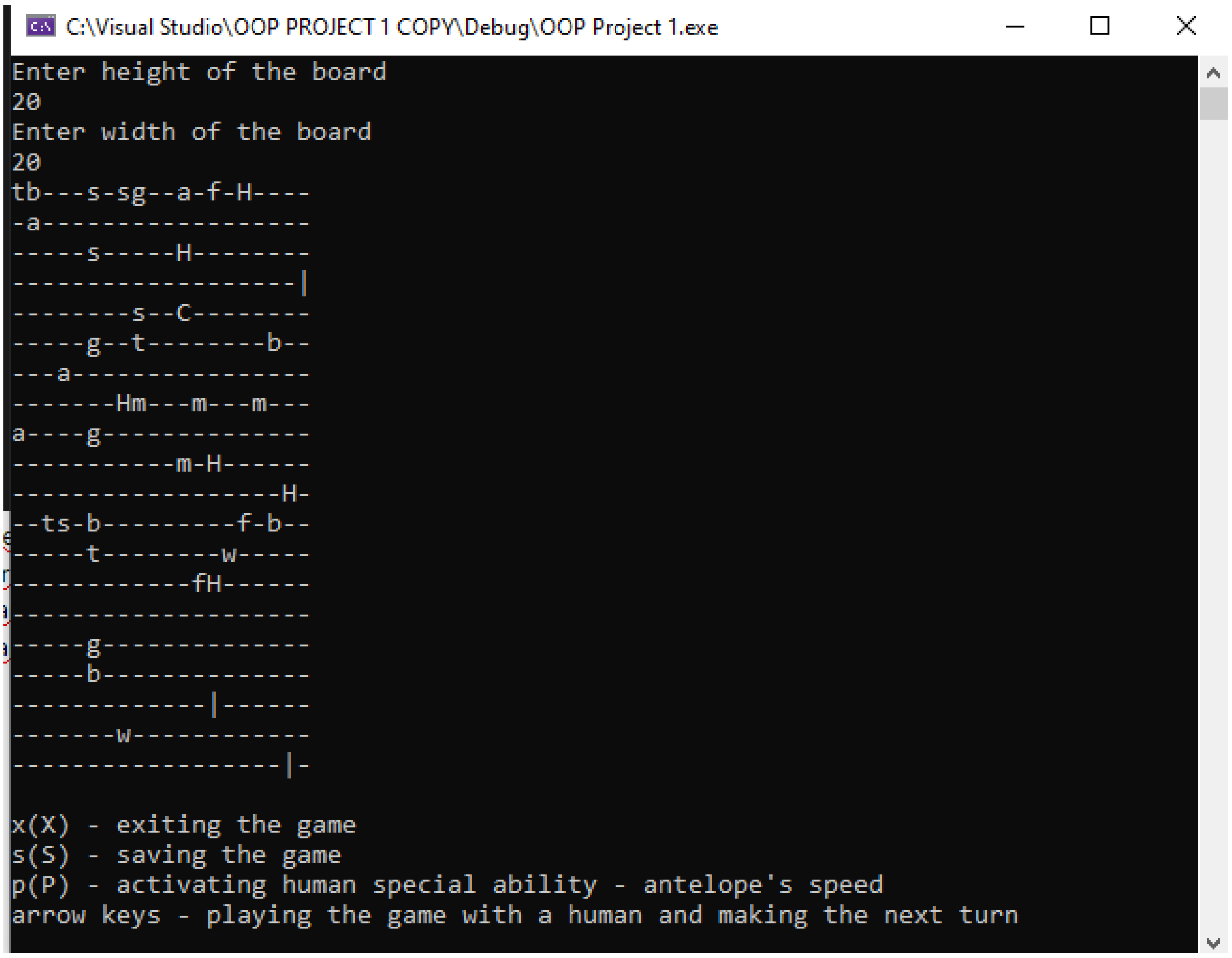
But to do this first you have to write a save file playing, for which method saveWorld is called when user inputs s or S:



```cpp
void World::saveWorld()
{
    std::ofstream fout;
    std::string filePath;
    std::cout << "Enter the name of the file in which you want to save the state of the world" << std::endl;
    std::cin >> filePath;
    filePath += ".txt";
    fout.open(filePath, std::ofstream::app);
    if (!fout.is_open())
        std::cout << "Error in creating file" << std::endl;
    else
    {
        fout << sizeX;
        fout << " ";
        fout << sizeY;
        fout << " ";
        for (int i = 0; i < sizeX; i++)
            for (int j = 0; j < sizeY; j++)
            {
                if (world[i][j] != nullptr)
                {
                    fout << world[i][j]->symbol;
                    fout << " ";
                    fout << world[i][j]->getX();
                    fout << " ";
                    fout << world[i][j]->getY();
                    fout << " ";
                    fout << world[i][j]->strength;
                    fout << " ";
                    fout << world[i][j]->age;
                    fout << " ";
                }
            }
    }
}
```

It creates a txt file and inputs there sizeX and sizeY of the board and all of the organisms on the board with its parameters: x, y, strength, age. Symbol of organism is used as a representation of organism so that way in readWorld function we can use chooseOrganism method to put instances of organisms on board when reading a file.

When we start the game an initial board is printed:


```
C:\Visual Studio\OOP PROJECT 1 COPY\Debug\OOP Project 1.exe
Enter height of the board
20
Enter width of the board
20
tb---s-sg--a-f-H----
-a------------------
-----s-----H--------
-------------------|
--------s--C--------
-----g--t--------b--
---a----------------
-------Hm---m---m---
a----g--------------
-----------m-H------
------------------H-
--ts-b---------f-b--
-----t--------w-----
------------fH------
--------------------
-----g--------------
-----b--------------
-------------|------
-------w------------
------------------|-

x(X) - exiting the game
s(S) - saving the game
p(P) - activating human special ability - antelope's speed
arrow keys - playing the game with a human and making the next turn
```

Symbols:

C – human

t – turtle

s- sheep

w- wolf

| - grass

g- guarana

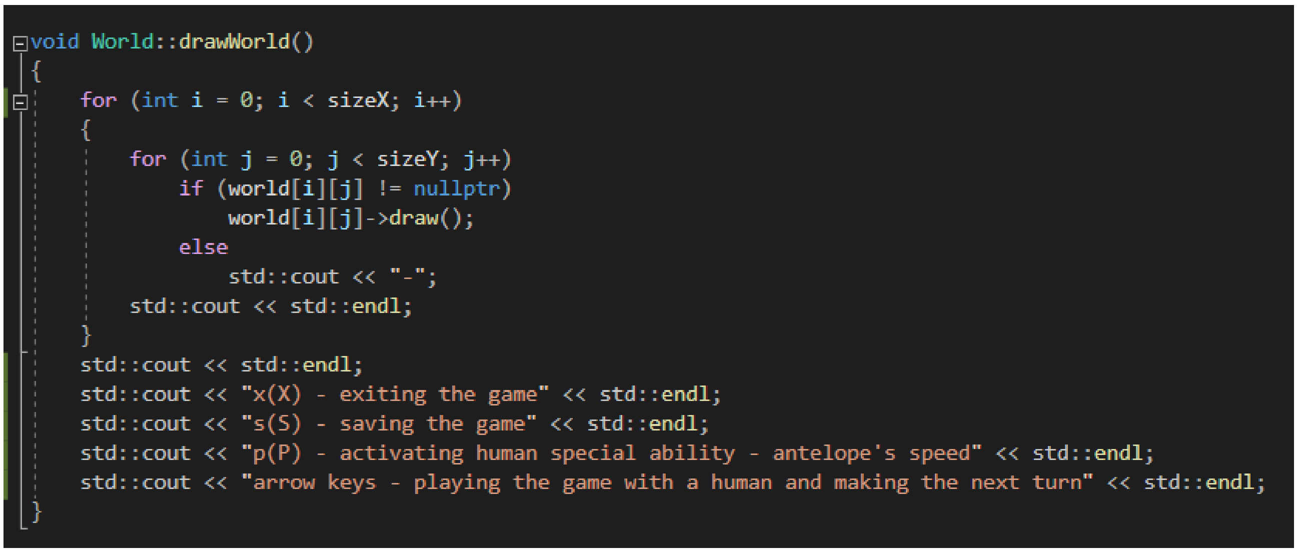
m-sowthistle

H-sosnowskyshogweed

a-antelope

f-fox

Method drawWorld is responsible for drawing the current world state:

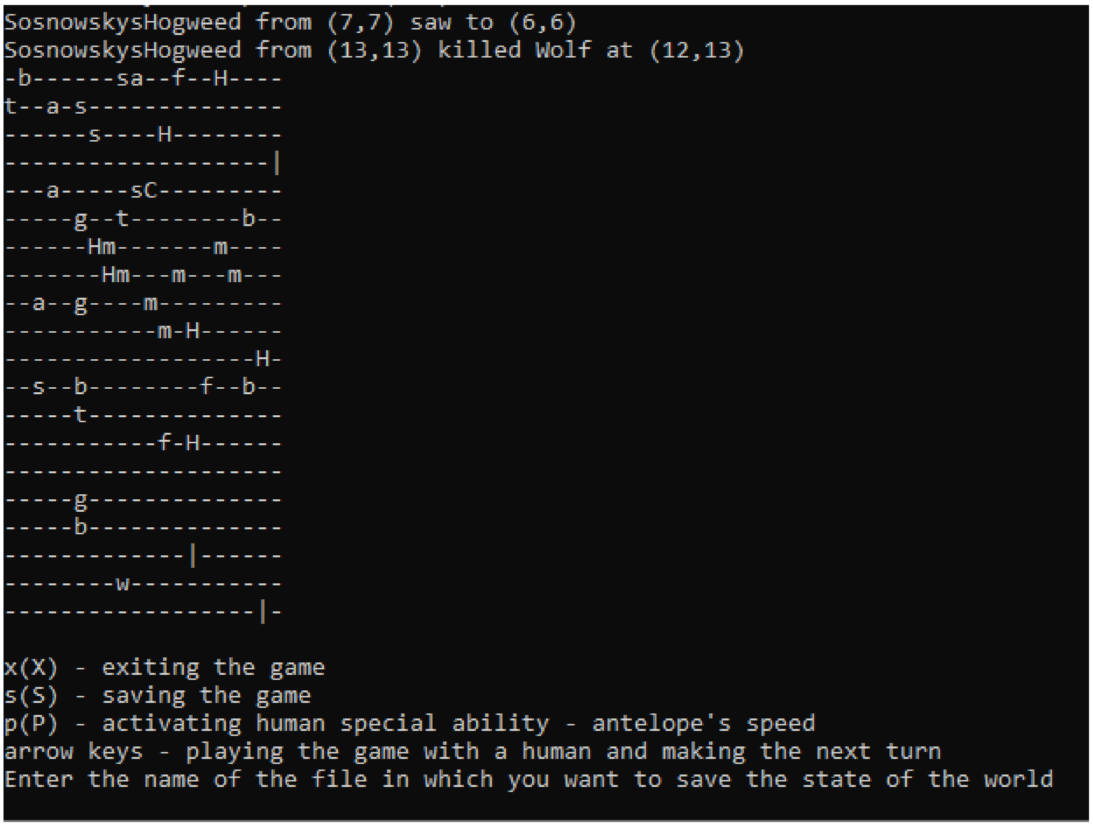


```cpp
void World::drawWorld()
{
    for (int i = 0; i < sizeX; i++)
    {
        for (int j = 0; j < sizeY; j++)
            if (world[i][j] != nullptr)
                world[i][j]->draw();
            else
                std::cout << "-";
        std::cout << std::endl;
    }
    std::cout << std::endl;
    std::cout << "x(X) - exiting the game" << std::endl;
    std::cout << "s(S) - saving the game" << std::endl;
    std::cout << "p(P) - activating human special ability - antelope's speed" << std::endl;
    std::cout << "arrow keys - playing the game with a human and making the next turn" << std::endl;
}
```

It simply iterates through the board(vector) and for each organism calls its method which prints a symbol of the animal.

I clicked the left arrow and human moved to the left field, and all instances of animals and plants updated as well.

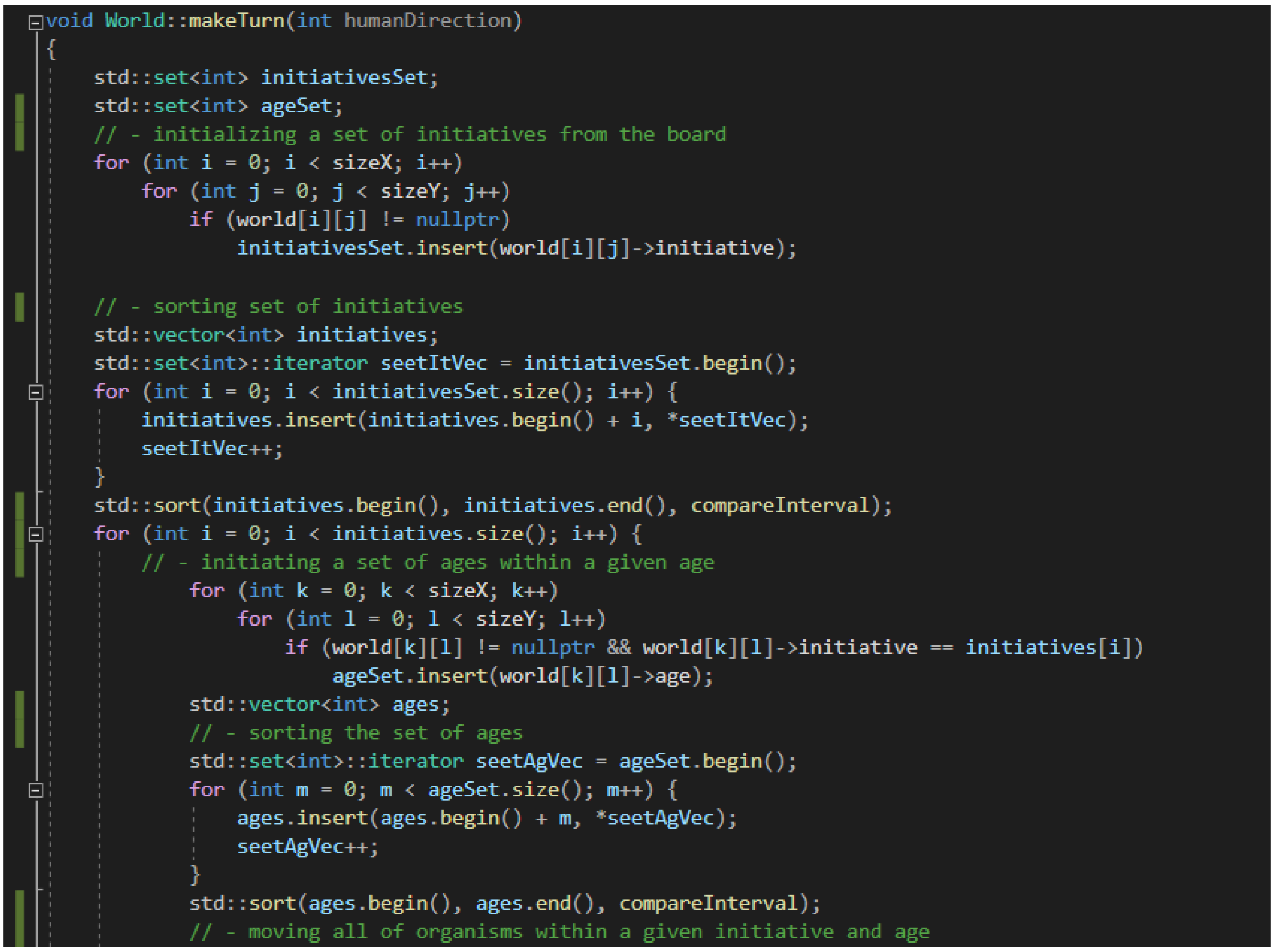
Which is showed by the action logs above the board(action logs show only specific actions such as sawing, breeding, killing, eating) I chose not to implement move of animals because of its common occurence as it would occupy too much space in the console:



```
SosnowskysHogweed from (7,7) saw to (6,6)
SosnowskysHogweed from (13,13) killed Wolf at (12,13)
-b------sa--f--H----
t--a-s--------------
------s----H--------
-------------------|
---a-----sC---------
-----g--t--------b--
------Hm-------m----
-------Hm---m---m---
--a--g----m---------
-----------m-H------
------------------H-
--s--b--------f--b--
-----t--------------
-----------f-H------
--------------------
-----g--------------
-----b--------------
-------------|------
--------w-----------
------------------|-

x(X) - exiting the game
s(S) - saving the game
p(P) - activating human special ability - antelope's speed
arrow keys - playing the game with a human and making the next turn
Enter the name of the file in which you want to save the state of the world
```

For making turn of the world method makeTurn is responsible(which is called when we press an arrow key – in case of moving a human, or when human is dead):



```cpp
void World::makeTurn(int humanDirection)
{
    std::set<int> initiativesSet;
    std::set<int> ageSet;
    // - initializing a set of initiatives from the board
    for (int i = 0; i < sizeX; i++)
        for (int j = 0; j < sizeY; j++)
            if (world[i][j] != nullptr)
                initiativesSet.insert(world[i][j]->initiative);

    // - sorting set of initiatives
    std::vector<int> initiatives;
    std::set<int>::iterator seetItVec = initiativesSet.begin();
    for (int i = 0; i < initiativesSet.size(); i++) {
        initiatives.insert(initiatives.begin() + i, *seetItVec);
        seetItVec++;
    }
    std::sort(initiatives.begin(), initiatives.end(), compareInterval);
    for (int i = 0; i < initiatives.size(); i++) {
        // - initiating a set of ages within a given age
            for (int k = 0; k < sizeX; k++)
                for (int l = 0; l < sizeY; l++)
                    if (world[k][l] != nullptr && world[k][l]->initiative == initiatives[i])
                        ageSet.insert(world[k][l]->age);
            std::vector<int> ages;
            // - sorting the set of ages
            std::set<int>::iterator seetAgVec = ageSet.begin();
            for (int m = 0; m < ageSet.size(); m++) {
                ages.insert(ages.begin() + m, *seetAgVec);
                seetAgVec++;
            }
            std::sort(ages.begin(), ages.end(), compareInterval);
            // - moving all of organisms within a given initiative and age
```

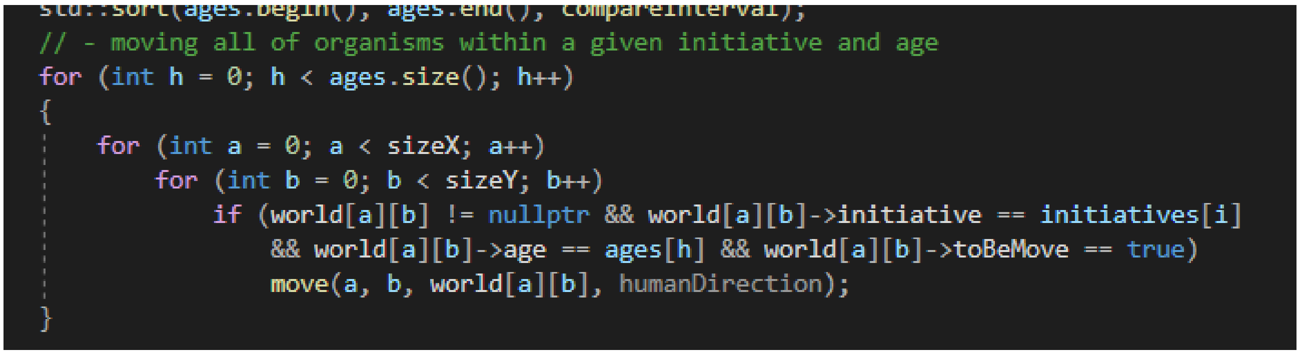
Here at first is created a set of initiatives from the organisms on the board, simply by iterating through the world and putting the values in the set

Then an iterator is created for the purpose of using a loop

Then the set is sorted in order to have individual distinct initiatives values in a descending order:

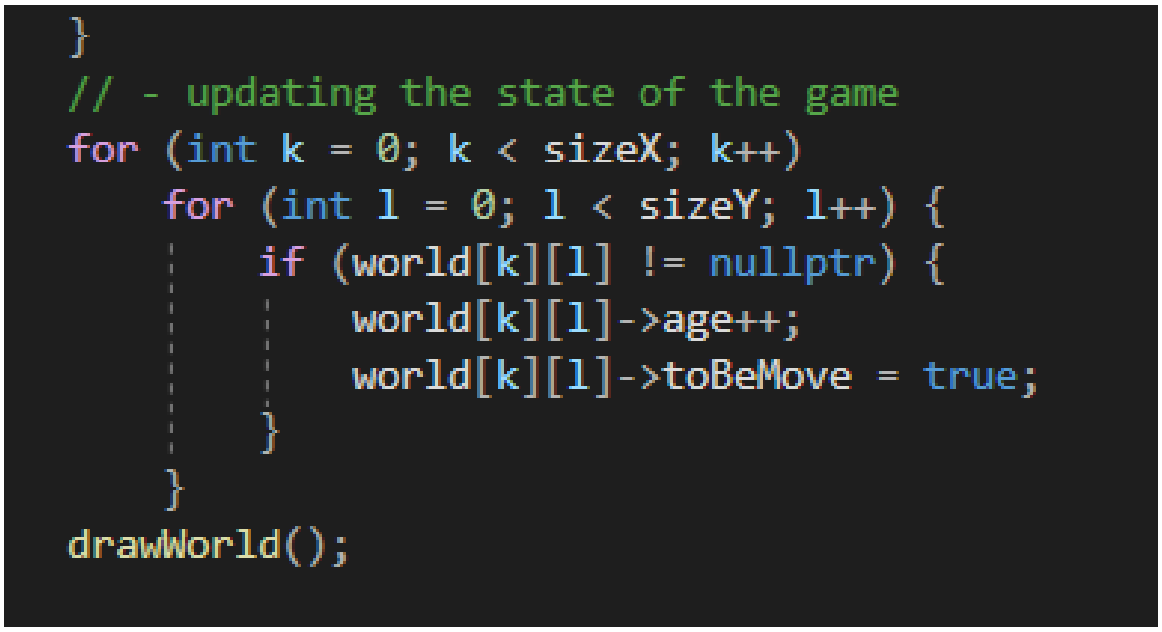
Then we iterate through all the initiatives and create an age set in a similar fashion

Then:



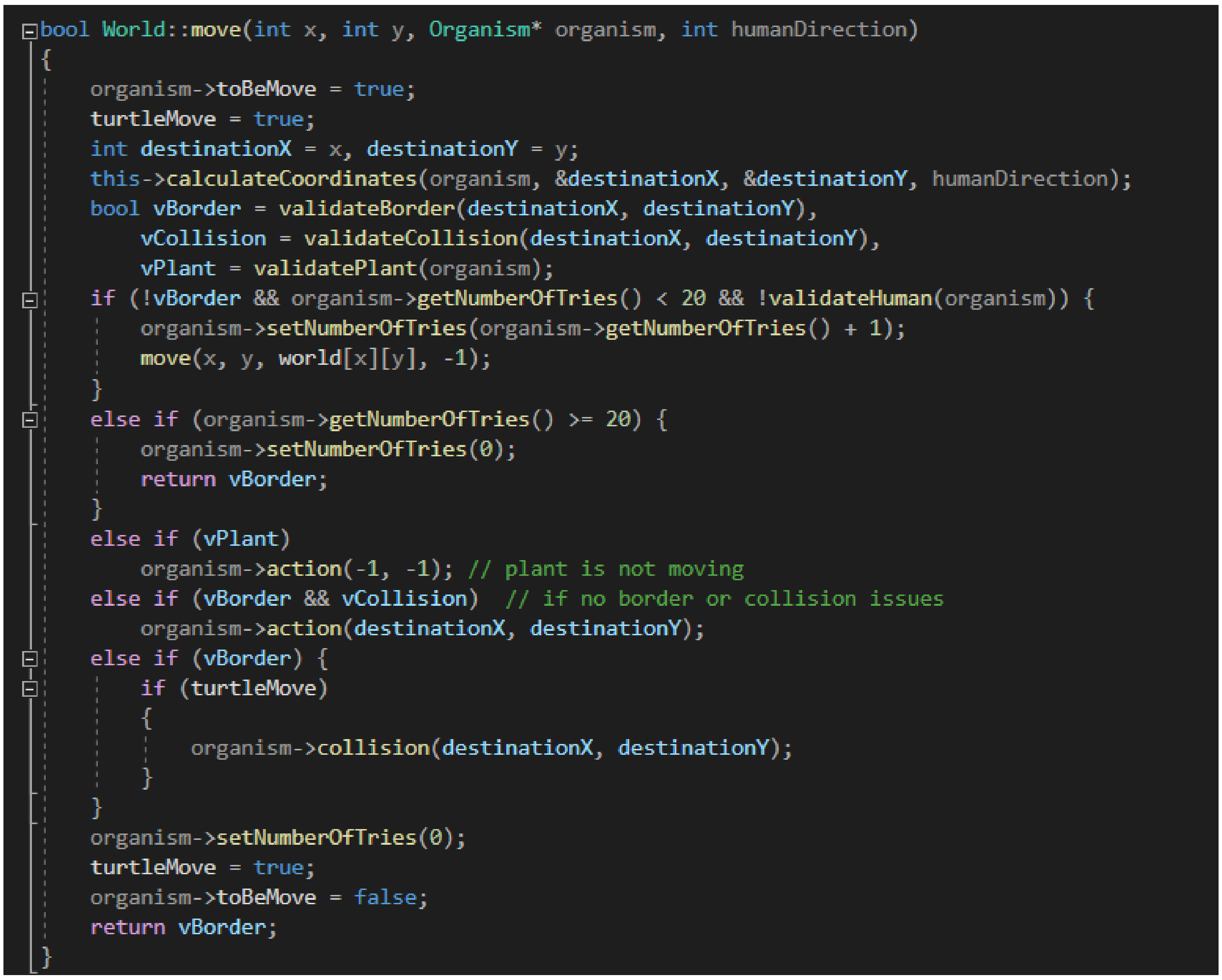
```cpp
std::sort(ages.begin(), ages.end(), compareInterval);
// - moving all of organisms within a given initiative and age
for (int h = 0; h < ages.size(); h++)
{
    for (int a = 0; a < sizeX; a++)
        for (int b = 0; b < sizeY; b++)
            if (world[a][b] != nullptr && world[a][b]->initiative == initiatives[i]
                && world[a][b]->age == ages[h] && world[a][b]->toBeMove == true)
                move(a, b, world[a][b], humanDirection);
}
```

We iterate through all of the ages within a given initiative and age and through iterating through the world organisms are moved in the correct order. For which method move is used but more about it further.



```
}
// - updating the state of the game
for (int k = 0; k < sizeX; k++)
    for (int l = 0; l < sizeY; l++) {
        if (world[k][l] != nullptr) {
            world[k][l]->age++;
            world[k][l]->toBeMove = true;
        }
    }
drawWorld();
```

Next we iterate through the world once again in order to update the state of the game, increasing the age of organisms and setting the flag for them to be moved, then method drawWorld is called and new world round is presented to us in the console.



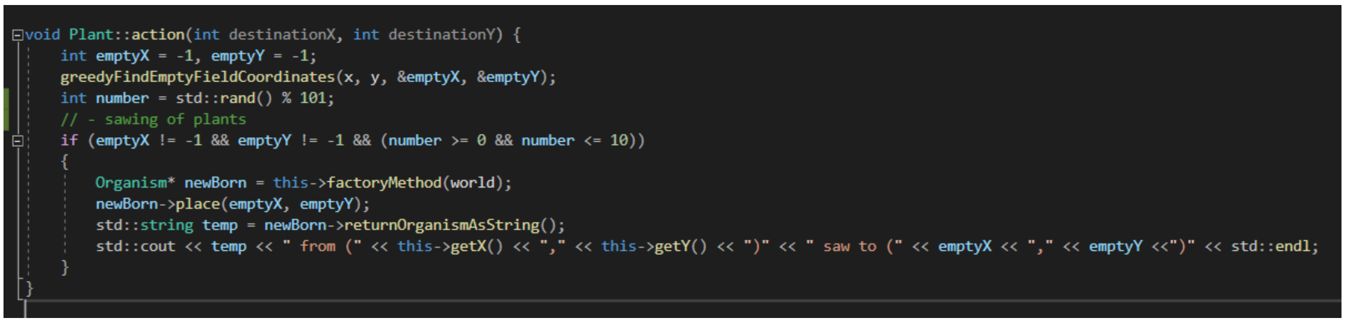
```
bool World::move(int x, int y, Organism* organism, int humanDirection)
{
    organism->toBeMove = true;
    turtleMove = true;
    int destinationX = x, destinationY = y;
    this->calculateCoordinates(organism, &destinationX, &destinationY, humanDirection);
    bool vBorder = validateBorder(destinationX, destinationY),
        vCollision = validateCollision(destinationX, destinationY),
        vPlant = validatePlant(organism);
    if (!vBorder && organism->getNumberOfTries() < 20 && !validateHuman(organism)) {
        organism->setNumberOfTries(organism->getNumberOfTries() + 1);
        move(x, y, world[x][y], -1);
    }
    else if (organism->getNumberOfTries() >= 20) {
        organism->setNumberOfTries(0);
        return vBorder;
    }
    else if (vPlant)
        organism->action(-1, -1); // plant is not moving
    else if (vBorder && vCollision)  // if no border or collision issues
        organism->action(destinationX, destinationY);
    else if (vBorder) {
        if (turtleMove)
        {
            organism->collision(destinationX, destinationY);
        }
    }
    organism->setNumberOfTries(0);
    turtleMove = true;
    organism->toBeMove = false;
    return vBorder;
}
```

This method is responsible for all the occurences on the board. It uses methods to calculate random coordinates, to validate if the coordinates are on the border or if there is a collision

If it fails on validation it calls itself again.

If an animal can't move for twenty times (an antelope case, it can't find a free field when running from an attacker) method ends.

If validation for plant occurs method action for plant is called:

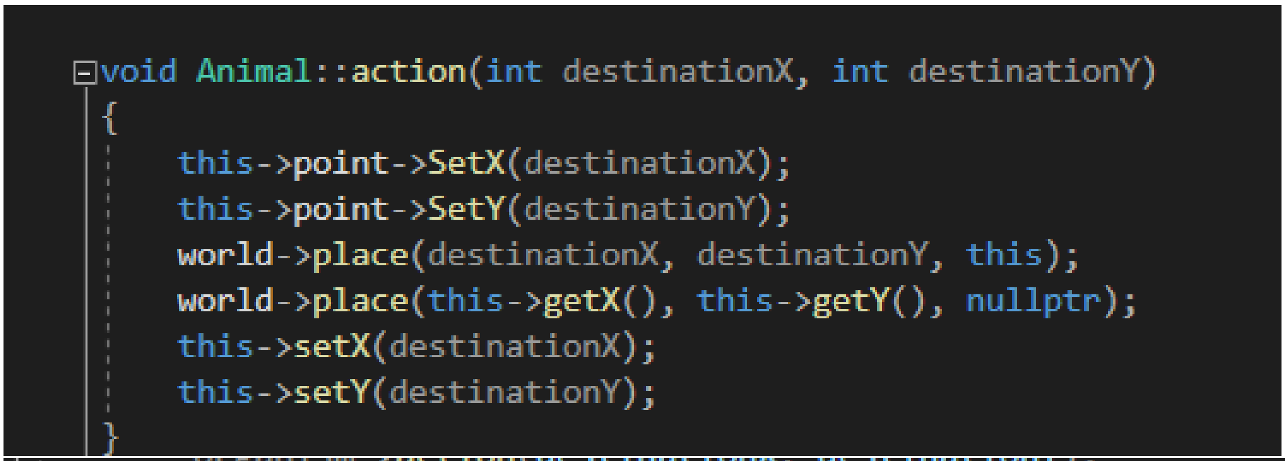


```cpp
void Plant::action(int destinationX, int destinationY) {
    int emptyX = -1, emptyY = -1;
    greedyFindEmptyFieldCoordinates(x, y, &emptyX, &emptyY);
    int number = std::rand() % 101;
    // - sawing of plants
    if (emptyX != -1 && emptyY != -1 && (number >= 0 && number <= 10))
    {
        Organism* newBorn = this->factoryMethod(world);
        newBorn->place(emptyX, emptyY);
        std::string temp = newBorn->returnOrganismAsString();
        std::cout << temp << " from (" << this->getX() << "," << this->getY() << ")" << " saw to (" << emptyX << "," << emptyY <<")" << std::endl;
    }
}
```

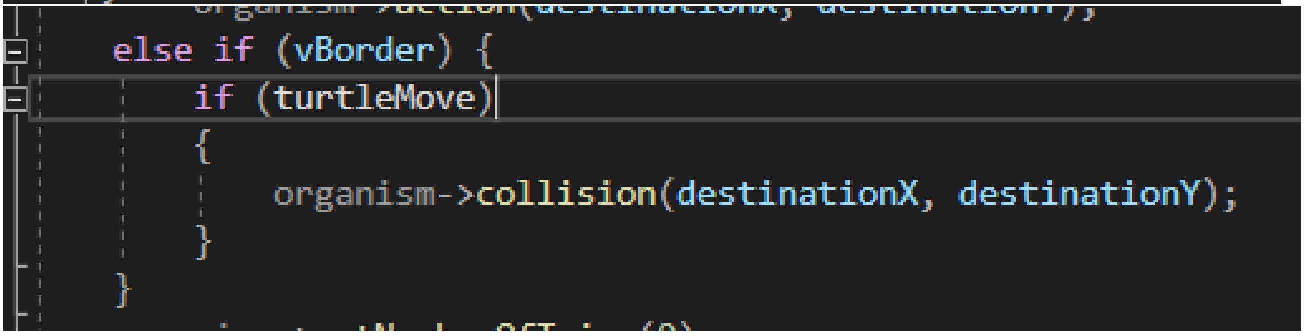
It saws a plant in a neighbouring field with a probability of 10%.

Coming back to move method.

If there are no border or collision issues just a simple moved is performed by calling the method action for animals:



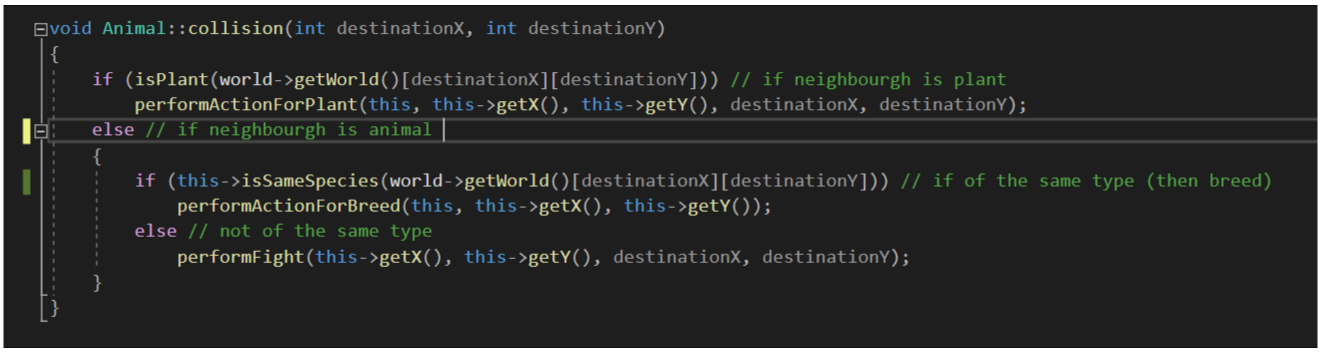
```cpp
void Animal::action(int destinationX, int destinationY)
{
    this->point->SetX(destinationX);
    this->point->SetY(destinationY);
    world->place(destinationX, destinationY, this);
    world->place(this->getX(), this->getY(), nullptr);
    this->setX(destinationX);
    this->setY(destinationY);
}
```



```cpp
else if (vBorder) {
    if (turtleMove)
    {
        organism->collision(destinationX, destinationY);
    }
}
```

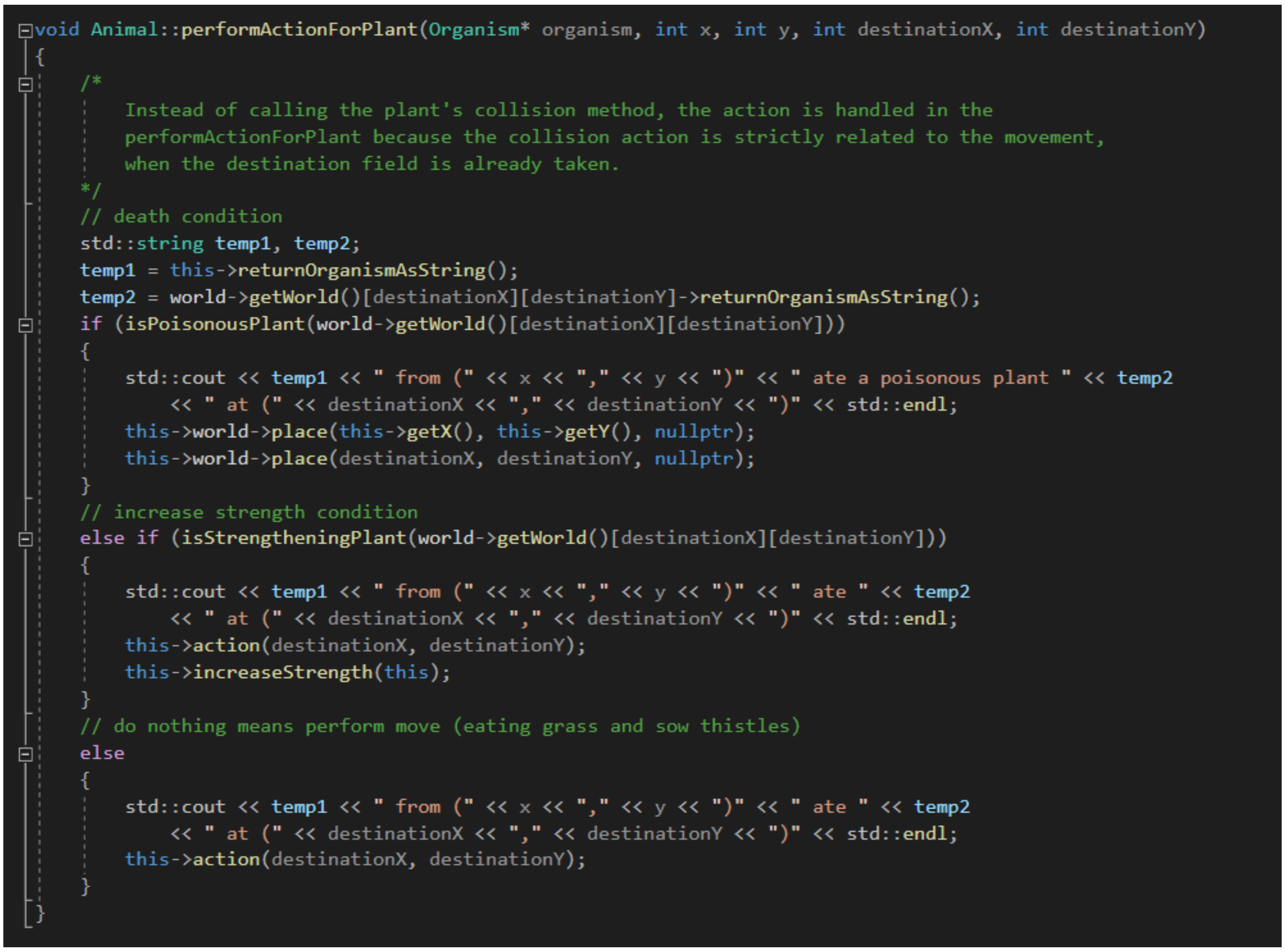
if there is no border issue and a collision happens collision method for each organism is called;

Default for animal:



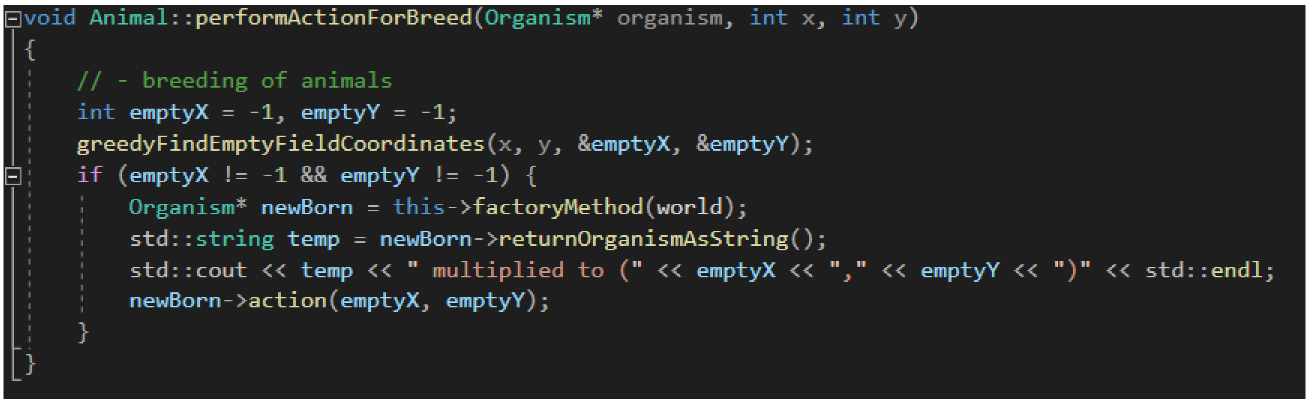
```cpp
void Animal::collision(int destinationX, int destinationY)
{
    if (isPlant(world->getWorld()[destinationX][destinationY])) // if neighbourgh is plant
        performActionForPlant(this, this->getX(), this->getY(), destinationX, destinationY);
    else // if neighbourgh is animal
    {
        if (this->isSameSpecies(world->getWorld()[destinationX][destinationY])) // if of the same type (then breed)
            performActionForBreed(this, this->getX(), this->getY());
        else // not of the same type
            performFight(this->getX(), this->getY(), destinationX, destinationY);
    }
}
```

Which uses three methods for eating a plant, for breeding and fighting

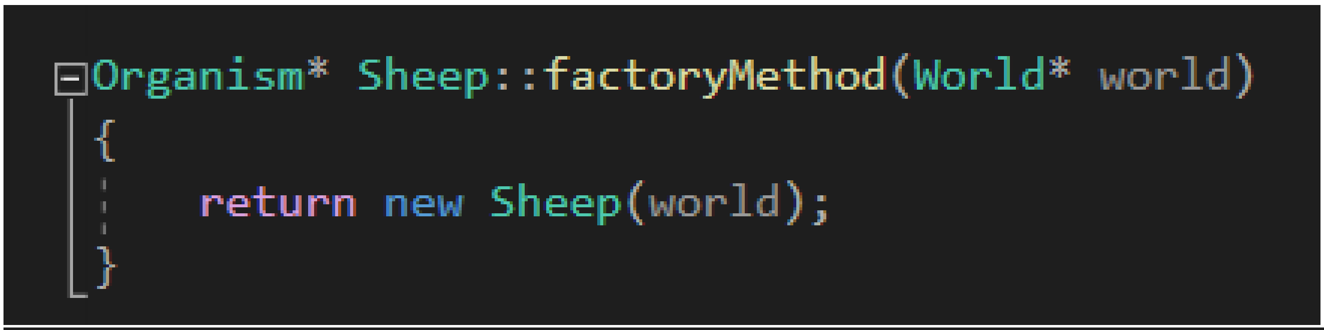


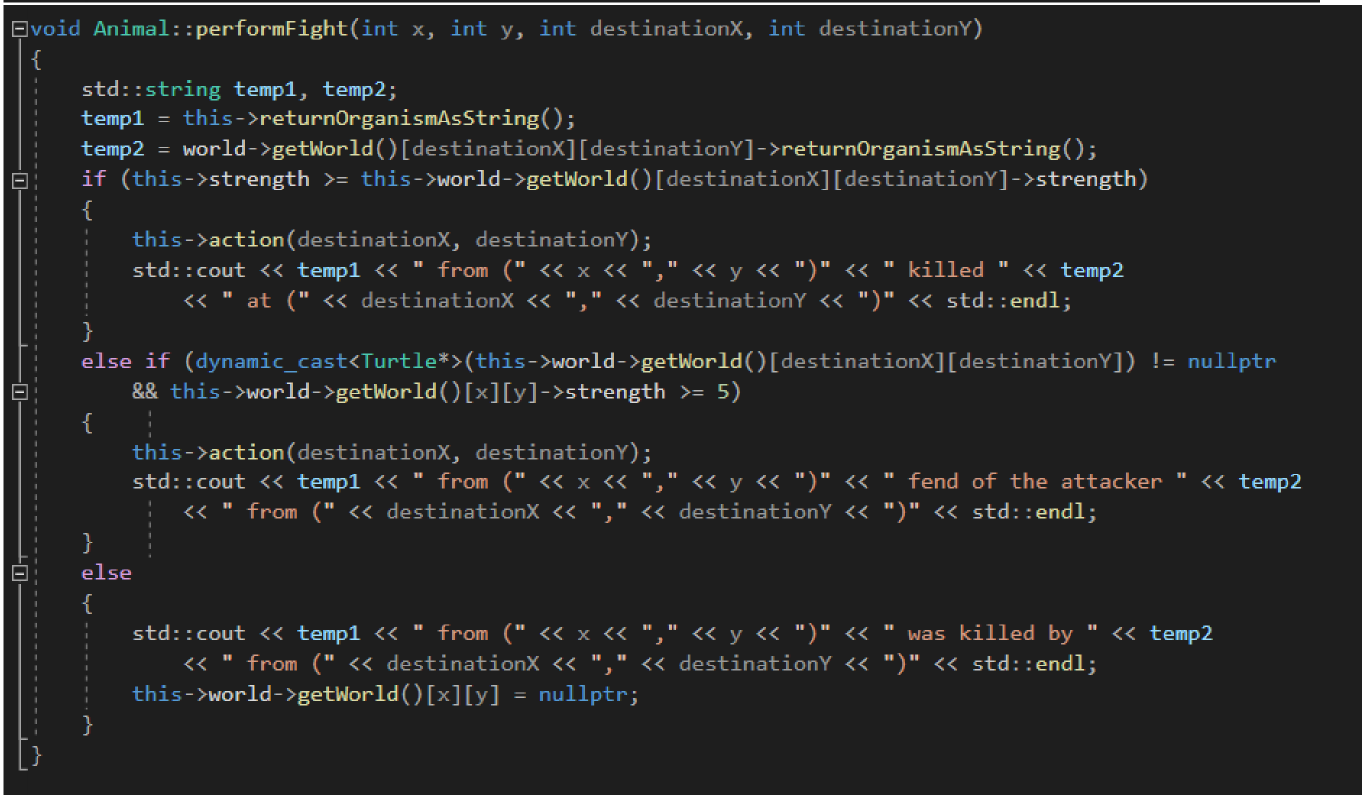
```cpp
void Animal::performActionForPlant(Organism* organism, int x, int y, int destinationX, int destinationY)
{
    /*
        Instead of calling the plant's collision method, the action is handled in the
        performActionForPlant because the collision action is strictly related to the movement,
        when the destination field is already taken.
    */
    // death condition
    std::string temp1, temp2;
    temp1 = this->returnOrganismAsString();
    temp2 = world->getWorld()[destinationX][destinationY]->returnOrganismAsString();
    if (isPoisonousPlant(world->getWorld()[destinationX][destinationY]))
    {
        std::cout << temp1 << " from (" << x << "," << y << ")" << " ate a poisonous plant " << temp2
            << " at (" << destinationX << "," << destinationY << ")" << std::endl;
        this->world->place(this->getX(), this->getY(), nullptr);
        this->world->place(destinationX, destinationY, nullptr);
    }
    // increase strength condition
    else if (isStrengtheningPlant(world->getWorld()[destinationX][destinationY]))
    {
        std::cout << temp1 << " from (" << x << "," << y << ")" << " ate " << temp2
            << " at (" << destinationX << "," << destinationY << ")" << std::endl;
        this->action(destinationX, destinationY);
        this->increaseStrength(this);
    }
    // do nothing means perform move (eating grass and sow thistles)
    else
    {
        std::cout << temp1 << " from (" << x << "," << y << ")" << " ate " << temp2
            << " at (" << destinationX << "," << destinationY << ")" << std::endl;
        this->action(destinationX, destinationY);
    }
}
```

This method checks whether plant is a death plant(belladonna, hogweed) or a strengthening plant(guarana) or in other case sowthistle and grass then it calls apropriate methods in order for animals to eat this plants and die or increase their strength or just simply eat them


```cpp
void Animal::performActionForBreed(Organism* organism, int x, int y)
{
    // - breeding of animals
    int emptyX = -1, emptyY = -1;
    greedyFindEmptyFieldCoordinates(x, y, &emptyX, &emptyY);
    if (emptyX != -1 && emptyY != -1) {
        Organism* newBorn = this->factoryMethod(world);
        std::string temp = newBorn->returnOrganismAsString();
        std::cout << temp << " multiplied to (" << emptyX << "," << emptyY << ")" << std::endl;
        newBorn->action(emptyX, emptyY);
    }
}
```

Perform action for breed works in a similar fashion as the sawing of plants method, it uses a factoryMethod, which is implemented in each class:

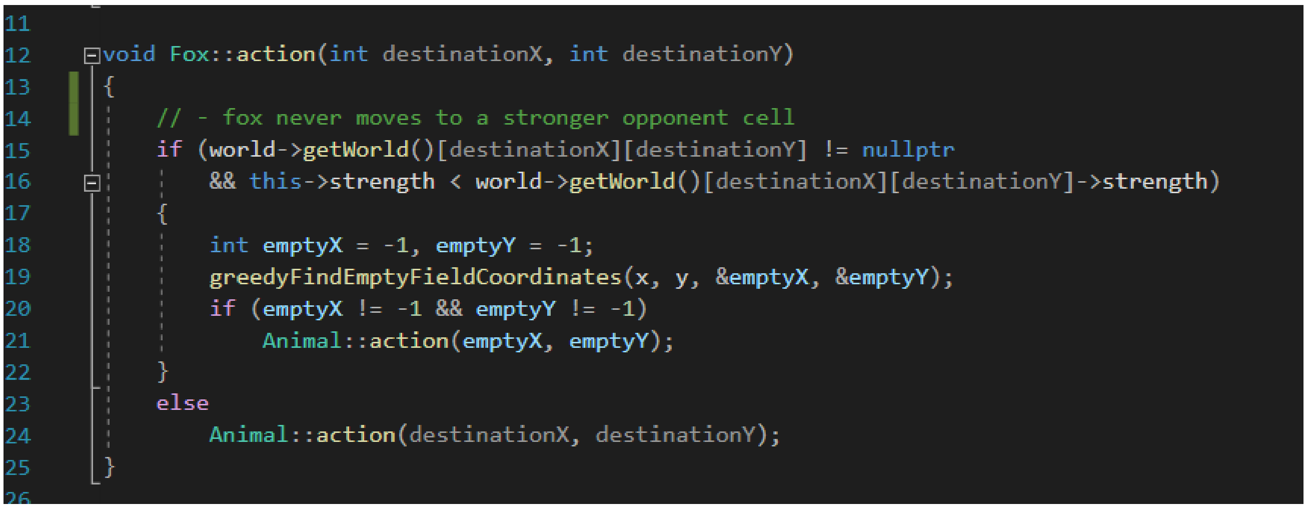

```cpp
Organism* Sheep::factoryMethod(World* world)
{
    return new Sheep(world);
}
```


```cpp
void Animal::performFight(int x, int y, int destinationX, int destinationY)
{
    std::string temp1, temp2;
    temp1 = this->returnOrganismAsString();
    temp2 = world->getWorld()[destinationX][destinationY]->returnOrganismAsString();
    if (this->strength >= this->world->getWorld()[destinationX][destinationY]->strength)
    {
        this->action(destinationX, destinationY);
        std::cout << temp1 << " from (" << x << "," << y << ")" << " killed " << temp2
            << " at (" << destinationX << "," << destinationY << ")" << std::endl;
    }
    else if (dynamic_cast<Turtle*>(this->world->getWorld()[destinationX][destinationY]) != nullptr
        && this->world->getWorld()[x][y]->strength >= 5)
    {
        this->action(destinationX, destinationY);
        std::cout << temp1 << " from (" << x << "," << y << ")" << " fend of the attacker " << temp2
            << " from (" << destinationX << "," << destinationY << ")" << std::endl;
    }
    else
    {
        std::cout << temp1 << " from (" << x << "," << y << ")" << " was killed by " << temp2
            << " from (" << destinationX << "," << destinationY << ")" << std::endl;
        this->world->getWorld()[x][y] = nullptr;
    }
}
```

Perform fight method compares the strengths of the opponents and carries the appropriate tasks or in case of a turtle it checks whether the attacker's strength is bigger than 5 and then kills the turtle

Because a turtle can reflect attacks of animals of only strength lower than 5.

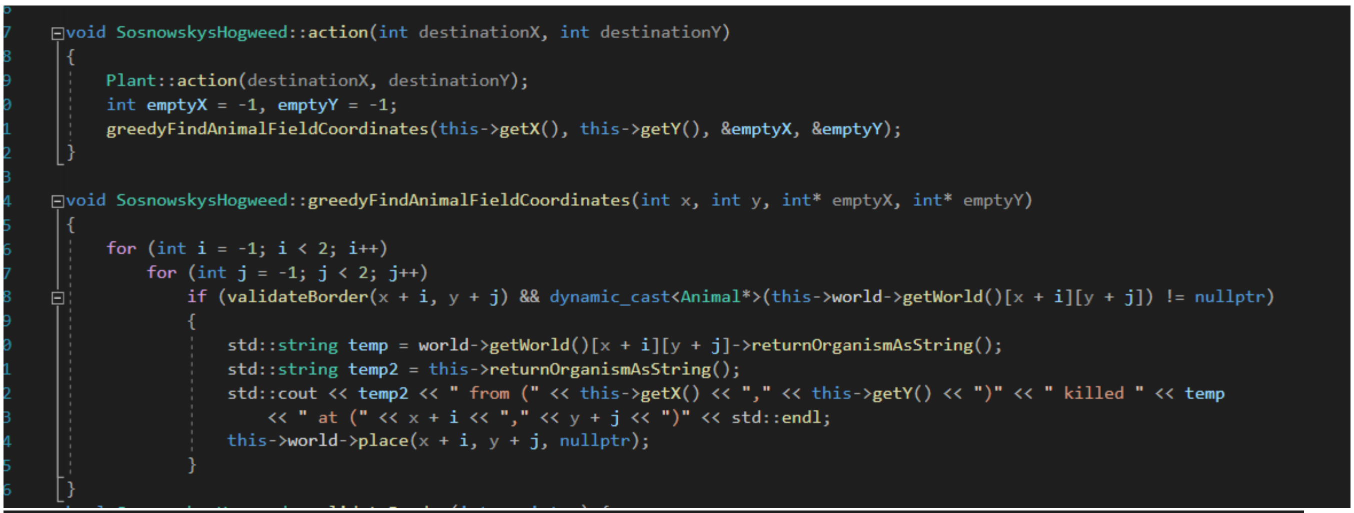
Fox has a special action



```cpp
void Fox::action(int destinationX, int destinationY)
{
    // - fox never moves to a stronger opponent cell
    if (world->getWorld()[destinationX][destinationY] != nullptr
        && this->strength < world->getWorld()[destinationX][destinationY]->strength)
    {
        int emptyX = -1, emptyY = -1;
        greedyFindEmptyFieldCoordinates(x, y, &emptyX, &emptyY);
        if (emptyX != -1 && emptyY != -1)
            Animal::action(emptyX, emptyY);
    }
    else
        Animal::action(destinationX, destinationY);
}
```

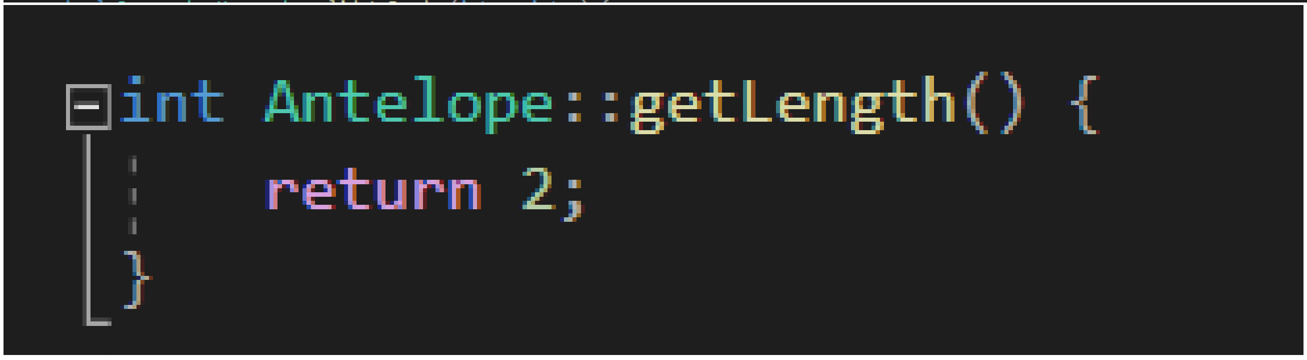
Which makes that fox never moves to a stronger opponent’s cell

Sosnowskys hogweed has a special action that kills all animals withing the neighbouring fields



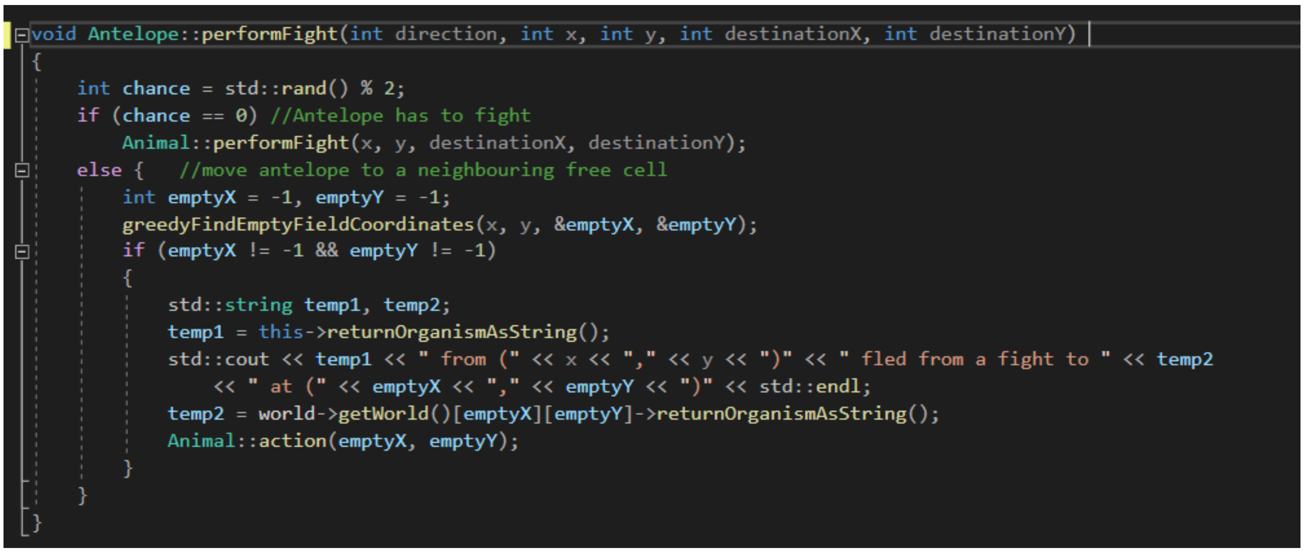
```cpp
void SosnowskysHogweed::action(int destinationX, int destinationY)
{
    Plant::action(destinationX, destinationY);
    int emptyX = -1, emptyY = -1;
    greedyFindAnimalFieldCoordinates(this->getX(), this->getY(), &emptyX, &emptyY);
}

void SosnowskysHogweed::greedyFindAnimalFieldCoordinates(int x, int y, int* emptyX, int* emptyY)
{
    for (int i = -1; i < 2; i++)
        for (int j = -1; j < 2; j++)
            if (validateBorder(x + i, y + j) && dynamic_cast<Animal*>(this->world->getWorld()[x + i][y + j]) != nullptr)
            {
                std::string temp = world->getWorld()[x + i][y + j]->returnOrganismAsString();
                std::string temp2 = this->returnOrganismAsString();
                std::cout << temp2 << " from (" << this->getX() << "," << this->getY() << ")" << " killed " << temp
                    << " at (" << x + i << "," << y + j << ")" << std::endl;
                this->world->place(x + i, y + j, nullptr);
            }
}
```



```cpp
int Antelope::getLength() {
    return 2;
}
```

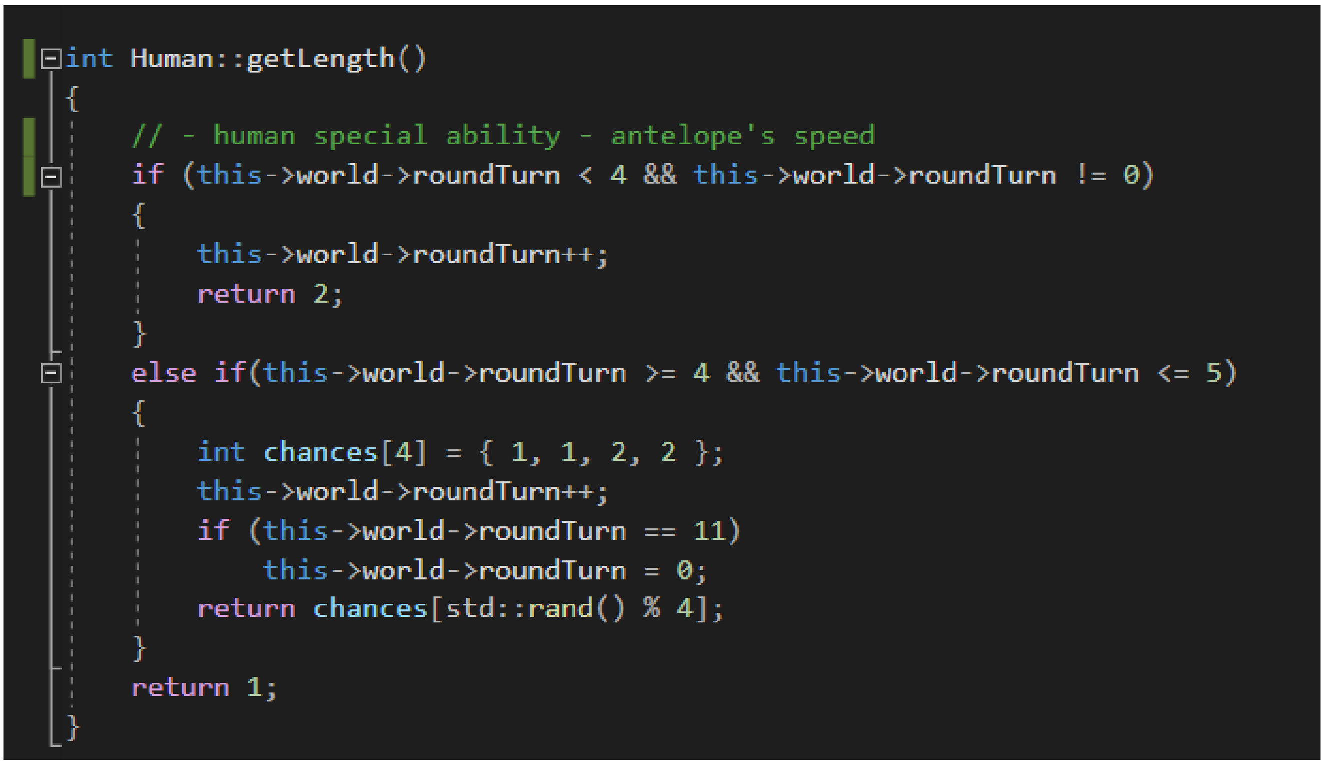
Antelope can move 2 blocks so method getLength in calculating coordinatesmethod returns 2 for antelope not 1 as for other animals



```cpp
void Antelope::performFight(int direction, int x, int y, int destinationX, int destinationY)
{
    int chance = std::rand() % 2;
    if (chance == 0) //Antelope has to fight
        Animal::performFight(x, y, destinationX, destinationY);
    else {   //move antelope to a neighbouring free cell
        int emptyX = -1, emptyY = -1;
        greedyFindEmptyFieldCoordinates(x, y, &emptyX, &emptyY);
        if (emptyX != -1 && emptyY != -1)
        {
            std::string temp1, temp2;
            temp1 = this->returnOrganismAsString();
            std::cout << temp1 << " from (" << x << "," << y << ")" << " fled from a fight to " << temp2
                << " at (" << emptyX << "," << emptyY << ")" << std::endl;
            temp2 = world->getWorld()[emptyX][emptyY]->returnOrganismAsString();
            Animal::action(emptyX, emptyY);
        }
    }
}
```

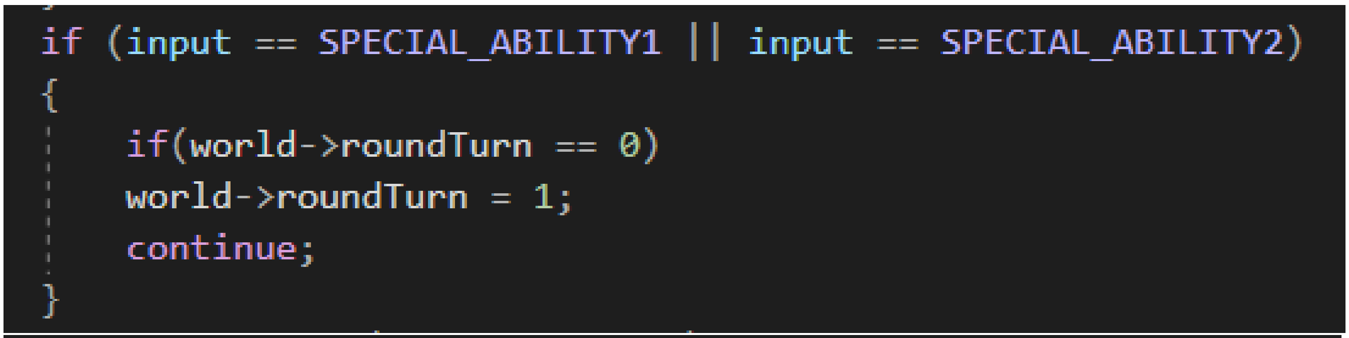
In case of collisiion antelope has a 50% chance to flee from the battle for which the performFight method is called

Human's special ability is activated when pressing p or P it then changes the roundTurn variable which is used in human's getLength method

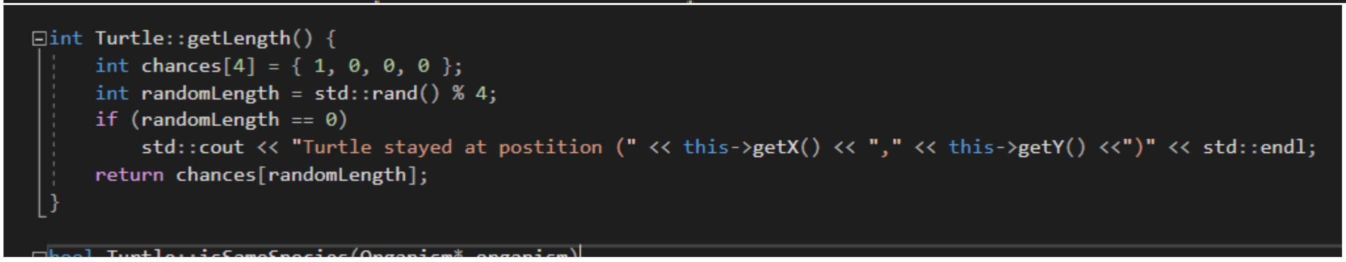


```cpp
int Human::getLength()
{
    // - human special ability - antelope's speed
    if (this->world->roundTurn < 4 && this->world->roundTurn != 0)
    {
        this->world->roundTurn++;
        return 2;
    }
    else if(this->world->roundTurn >= 4 && this->world->roundTurn <= 5)
    {
        int chances[4] = { 1, 1, 2, 2 };
        this->world->roundTurn++;
        if (this->world->roundTurn == 11)
            this->world->roundTurn = 0;
        return chances[std::rand() % 4];
    }
    return 1;
}
```

If the round turn is 1,2,3 human moves two blocks, if it's 4 or 5 human has a 50% chance of walking two blocks and then the ability is blocked for the 5 rounds when it can be enabled again which can be seen in the controller

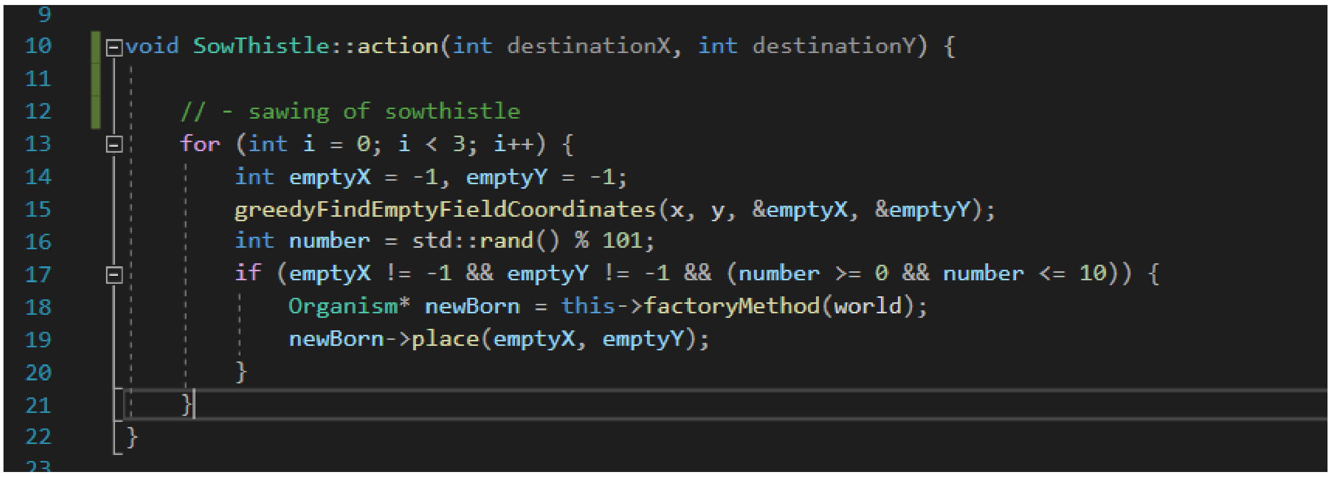


```
if (input == SPECIAL_ABILITY1 || input == SPECIAL_ABILITY2)
{
    if(world->roundTurn == 0)
    world->roundTurn = 1;
    continue;
}
```



```
int Turtle::getLength() {
    int chances[4] = { 1, 0, 0, 0 };
    int randomLength = std::rand() % 4;
    if (randomLength == 0)
        std::cout << "Turtle stayed at postition (" << this->getX() << "," << this->getY() <<")" << std::endl;
    return chances[randomLength];
}
```

Turtle has a 75% chance to stay in one place so this method is used for that case

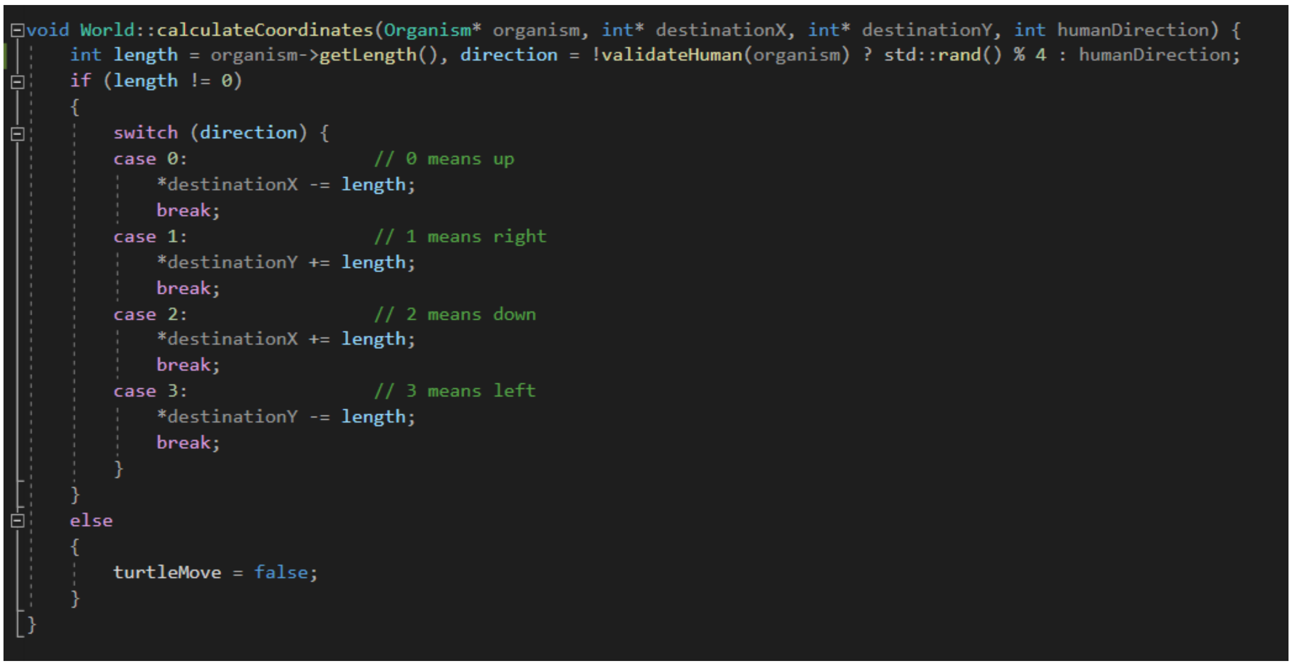


```
void SowThistle::action(int destinationX, int destinationY) {

    // - sawing of sowthistle
    for (int i = 0; i < 3; i++) {
        int emptyX = -1, emptyY = -1;
        greedyFindEmptyFieldCoordinates(x, y, &emptyX, &emptyY);
        int number = std::rand() % 101;
        if (emptyX != -1 && emptyY != -1 && (number >= 0 && number <= 10)) {
            Organism* newBorn = this->factoryMethod(world);
            newBorn->place(emptyX, emptyY);
        }
    }
}
```

Sowthistle action is different because it tries 3 times to saw a plant which can be seen above.

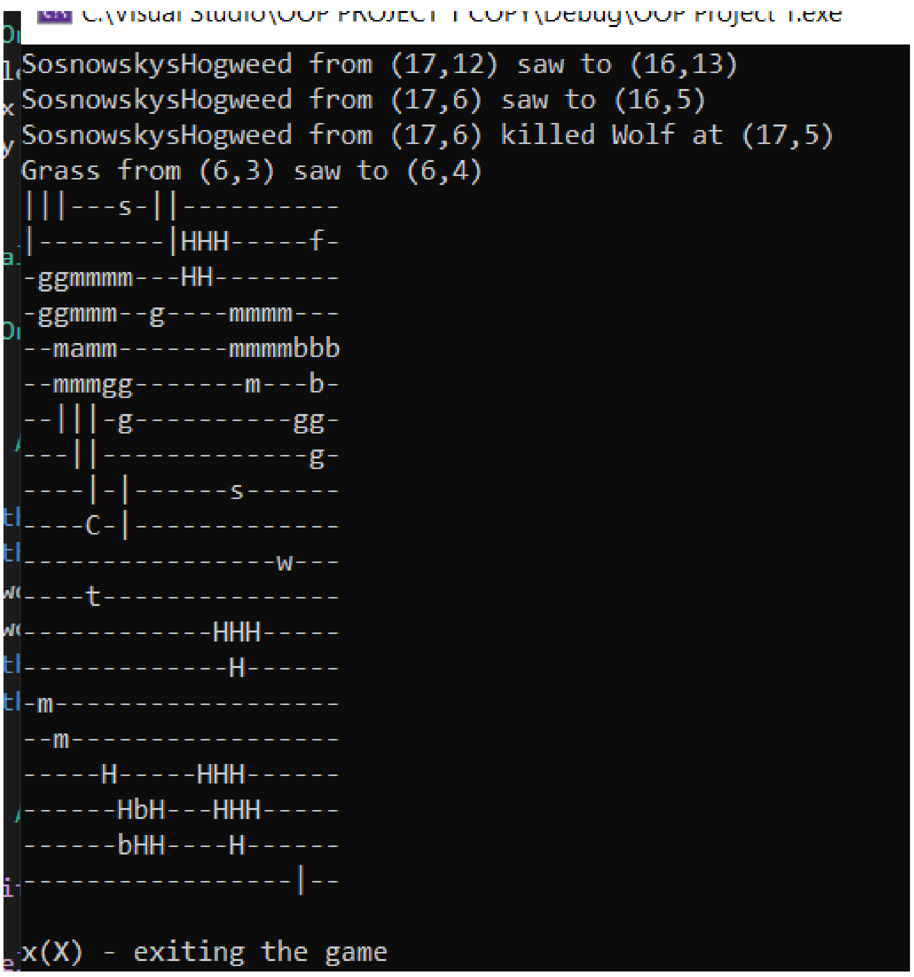
Method for calculating coordinates for animal moves which calls getLength method for a given animal



```cpp
void World::calculateCoordinates(Organism* organism, int* destinationX, int* destinationY, int humanDirection) {
    int length = organism->getLength(), direction = !validateHuman(organism) ? std::rand() % 4 : humanDirection;
    if (length != 0)
    {
        switch (direction) {
        case 0:                 // 0 means up
            *destinationX -= length;
            break;
        case 1:                 // 1 means right
            *destinationY += length;
            break;
        case 2:                 // 2 means down
            *destinationX += length;
            break;
        case 3:                 // 3 means left
            *destinationY -= length;
            break;
        }
    }
    else
    {
        turtleMove = false;
    }
}
```

I implemented all of the required functions for 5 points.

Here are some additional screens of the game:

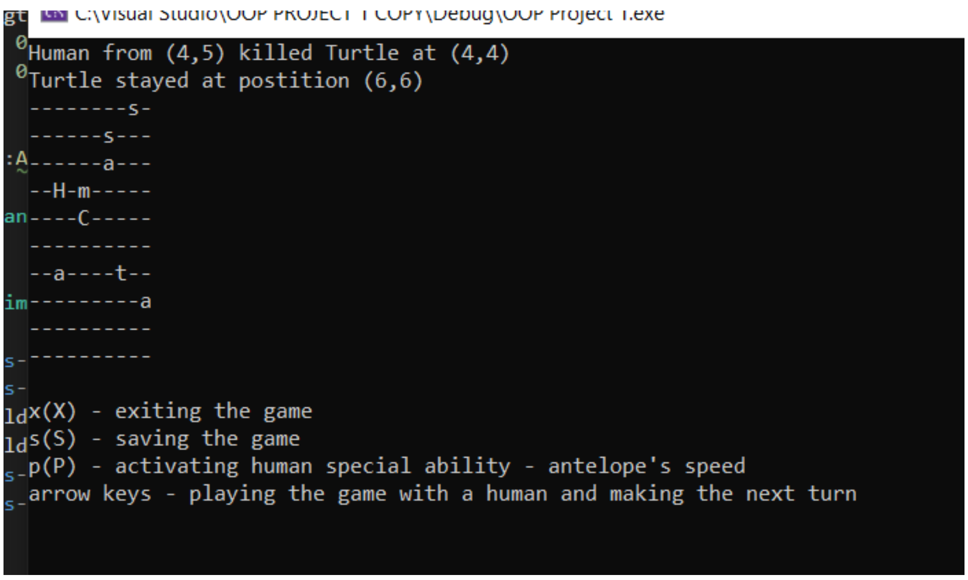


C:\Visual Studio\OOP PROJECT 1 COPY\Debug\OOP Project 1.exe

```
SosnowskysHogweed from (17,12) saw to (16,13)
SosnowskysHogweed from (17,6) saw to (16,5)
SosnowskysHogweed from (17,6) killed Wolf at (17,5)
Grass from (6,3) saw to (6,4)
|||---s-||----------
|--------|HHH-----f-
-ggmmmm---HH--------
-ggmmm--g----mmmm---
--mamm-------mmmmbbb
--mmmgg-------m---b-
--|||-g----------gg-
---||-------------g-
----|-|------s------
----C-|-------------
----------------w---
----t---------------
------------HHH-----
-------------H------
-m------------------
--m-----------------
-----H-----HHH------
------HbH---HHH-----
------bHH----H------
----------------|--

x(X) - exiting the game
```

A later stage of the game when a lot of plants are sawing

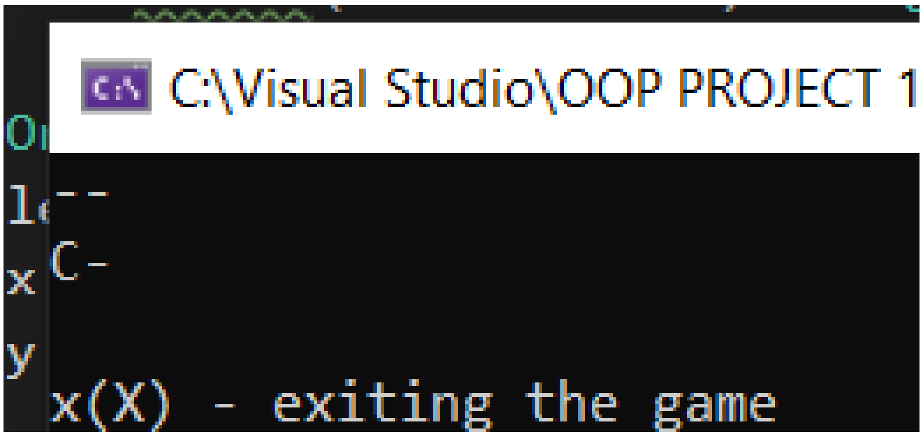


```
C:\Visual Studio\OOP PROJECT 1 COPY\Debug\OOP Project 1.exe
Human from (4,5) killed Turtle at (4,4)
Turtle stayed at postition (6,6)
--------s-
------s---
------a---
--H-m-----
----C-----
----------
--a----t--
---------a
----------
----------

x(X) - exiting the game
s(S) - saving the game
p(P) - activating human special ability - antelope's speed
arrow keys - playing the game with a human and making the next turn
```

A smaller board with less organisms

Or even a 2 by 2 board with just a human!



```
C:\Visual Studio\OOP PROJECT 1
--
C-

x(X) - exiting the game
```

Coming back to the action logs. All are implemented when an specific action happens and they print just the message indicating that.

Thank you for your patience.