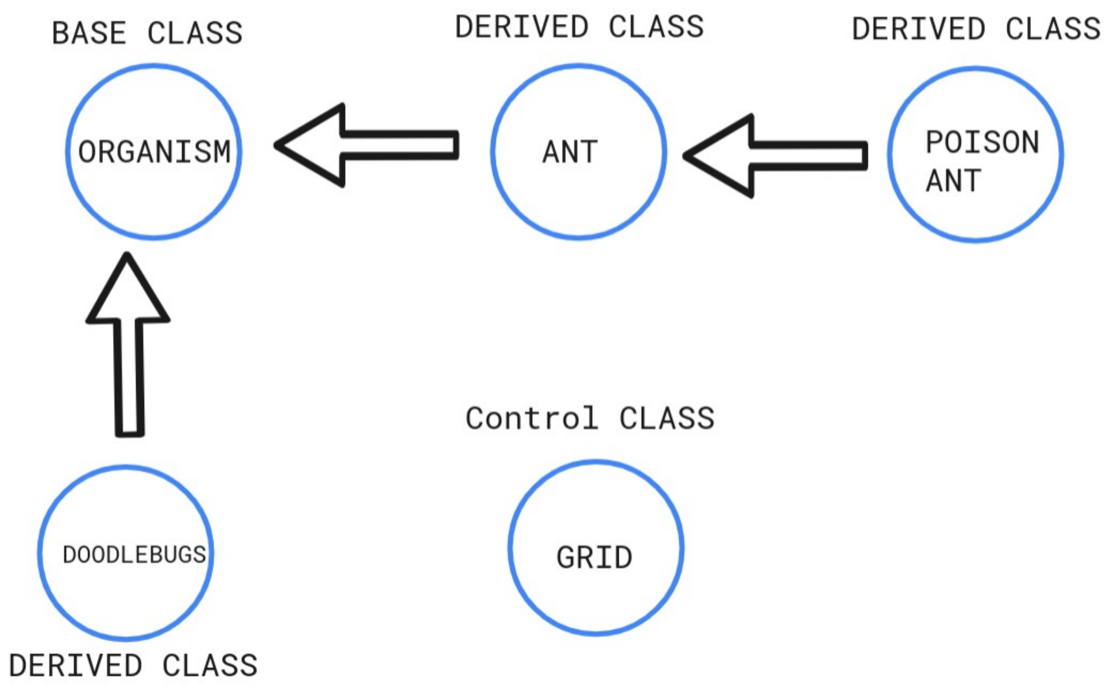
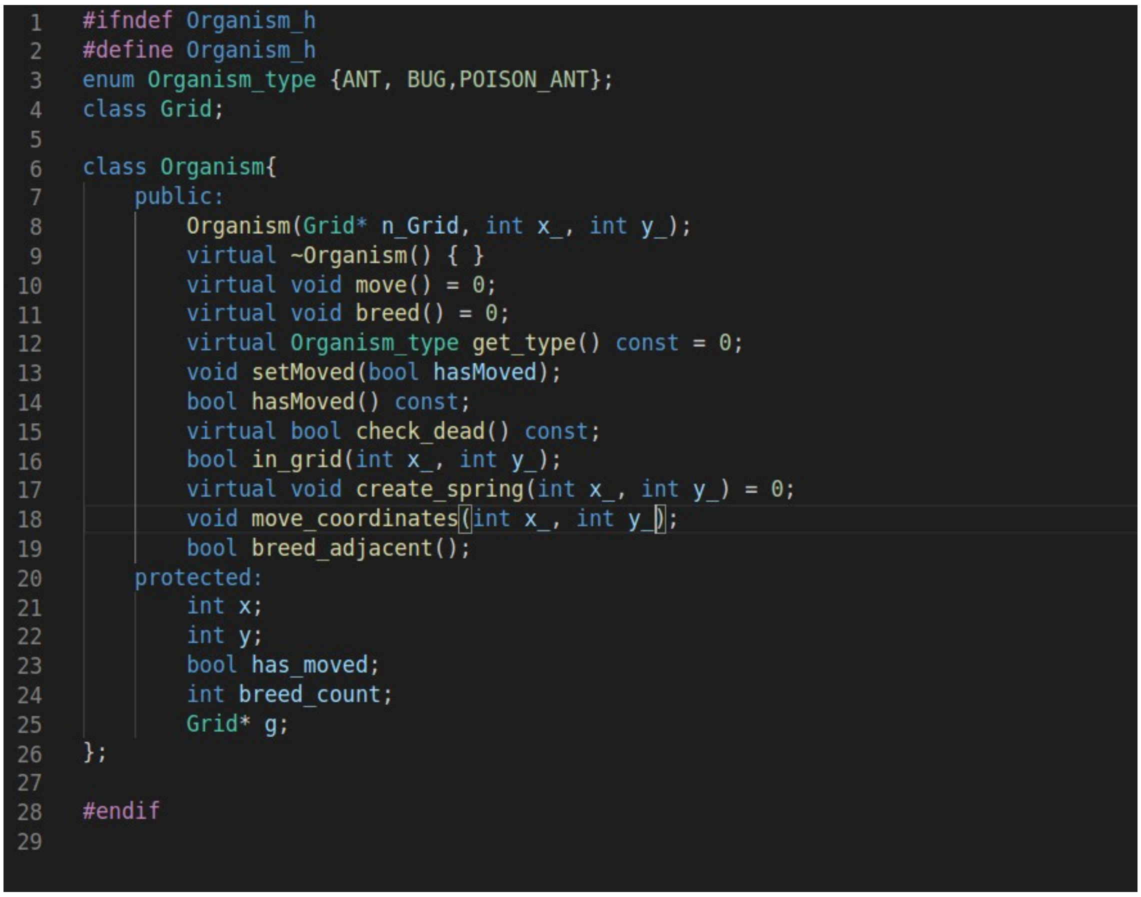
PROGRAMMİNG PROJECT DOCUMANTATİON

Class Design



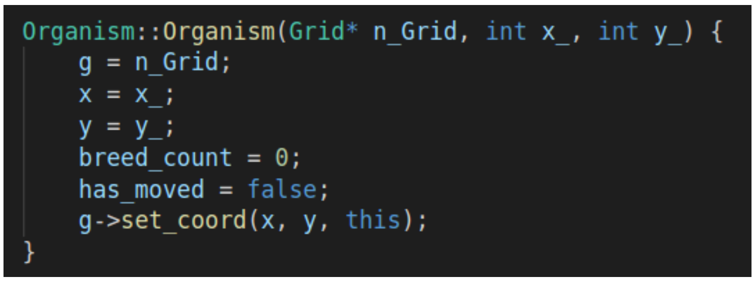
ORGANISM Class



```
#ifndef Organism_h
#define Organism_h
enum Organism_type {ANT, BUG,POISON_ANT};
class Grid;

class Organism{
    public:
        Organism(Grid* n_Grid, int x_, int y_);
        virtual ~Organism() { }
        virtual void move() = 0;
        virtual void breed() = 0;
        virtual Organism_type get_type() const = 0;
        void setMoved(bool hasMoved);
        bool hasMoved() const;
        virtual bool check_dead() const;
        bool in_grid(int x_, int y_);
        virtual void create_spring(int x_, int y_) = 0;
        void move_coordinates(int x_, int y_);
        bool breed_adjacent();
    protected:
        int x;
        int y;
        bool has_moved;
        int breed_count;
        Grid* g;
};

#endif
```

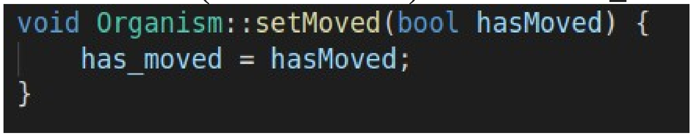
x = This variable keeps x coordinates for Class.
y = This variable keeps y coordinates for Class.
has_moved = This variable keeps the object whether it moves.
breed_count =It is a counter that when Organism will breed.
Grid* g = It is a Grid Class Pointer for control the move and breed etc. functions.

enum Organism_type = > It is a enum for indicate what type of organism have class.
Organism(Grid* n_Grid, int x_, int y) => Paramater Constructor for create new Organism object.



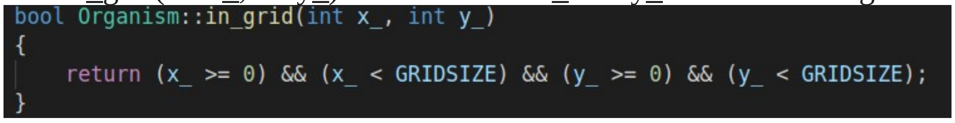
```
Organism::Organism(Grid* n_Grid, int x_, int y_) {
    g = n_Grid;
    x = x_;
    y = y_;
    breed_count = 0;
    has_moved = false;
    g->set_coord(x, y, this);
}
```

~Organism() = > Destructor
virtual void move() = > Pure virtual function for move Organism.
virtual void breed() = > Pure virtual function for breed Organism.
virtual Organism_type get_type() const = > Return what type of Organism(Such as Ant, Doodlebugs or Posion Ant)
void setMoved(bool hasMoved) = >Set the has_moved varible.



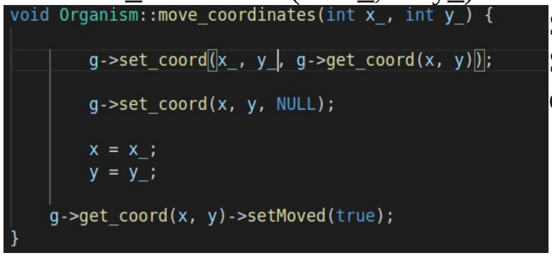
```
void Organism::setMoved(bool hasMoved) {
    has_moved = hasMoved;
}
```

virtual bool check_dead() = > Check the Organism object is dead or alive.
bool in_grid(int x_, int y_) = > Check the x_ and y_ coordinates in game map.



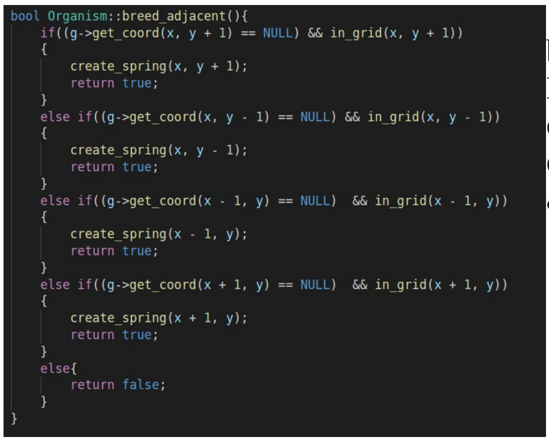
```
bool Organism::in_grid(int x_, int y_)
{
    return (x_ >= 0) && (x_ < GRIDSIZE) && (y_ >= 0) && (y_ < GRIDSIZE);
}
```

GRIDSIZE variable is size of the Grid.If the point is not in the Grid return false otherwise return true.
virtual void create_spring(int x_, int y_) = >This function create a offspring for Organism object
void move_coordinates(int x_, int y_) = > Moves the Organism object to another point



```
void Organism::move_coordinates(int x_, int y_) {

        g->set_coord(x_, y_, g->get_coord(x, y));

        g->set_coord(x, y, NULL);

        x = x_;
        y = y_;

    g->get_coord(x, y)->setMoved(true);
}
```

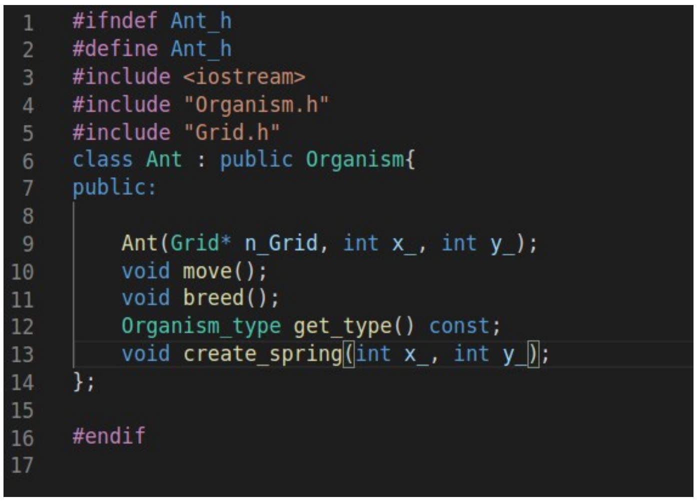
Set the new x and y points.
Set the old point to NULL and setMoved true until the other tour comes.



```
bool Organism::breed_adjacent(){
    if((g->get_coord(x, y + 1) == NULL) && in_grid(x, y + 1))
    {
        create_spring(x, y + 1);
        return true;
    }
    else if((g->get_coord(x, y - 1) == NULL) && in_grid(x, y - 1))
    {
        create_spring(x, y - 1);
        return true;
    }
    else if((g->get_coord(x - 1, y) == NULL)  && in_grid(x - 1, y))
    {
        create_spring(x - 1, y);
        return true;
    }
    else if((g->get_coord(x + 1, y) == NULL)  && in_grid(x + 1, y))
    {
        create_spring(x + 1, y);
        return true;
    }
    else{
        return false;
    }
}
```

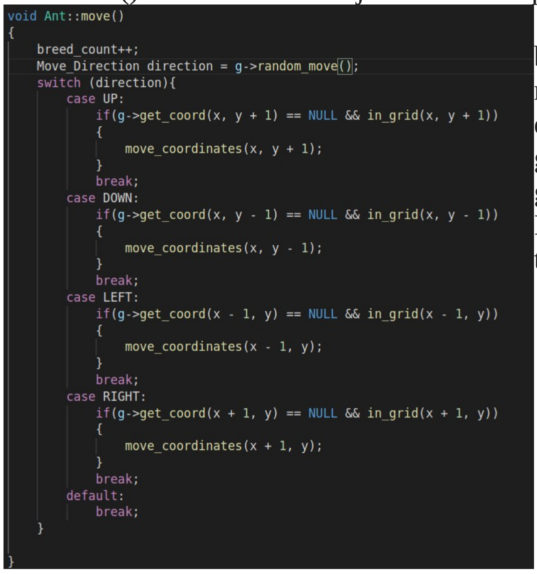
bool breed_adjacent() = >Check the adjacent point is NULL or not.And if adjacent point is NULL create new ofspring to that point.Use create_spring for create new offspring and return true for the not to do same breed again.

ANT CLASS



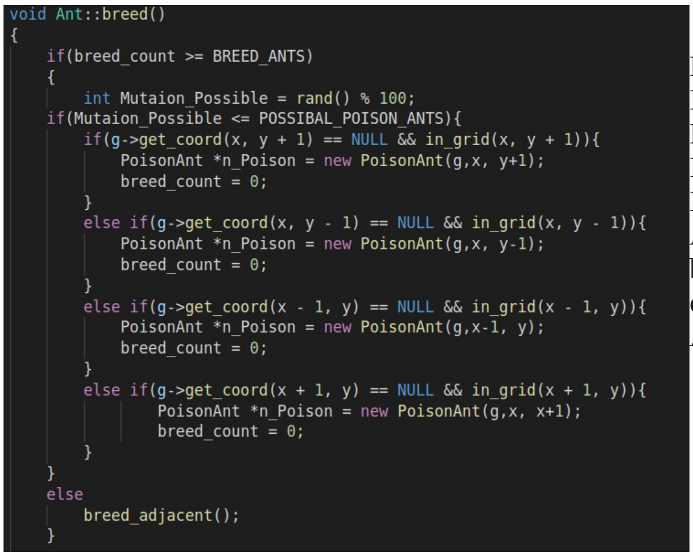
```cpp
#ifndef Ant_h
#define Ant_h
#include <iostream>
#include "Organism.h"
#include "Grid.h"
class Ant : public Organism{
public:

    Ant(Grid* n_Grid, int x_, int y_);
    void move();
    void breed();
    Organism_type get_type() const;
    void create_spring(int x_, int y_);
};

#endif

```

Ant(Grid* n_Grid, int x_, int y_) = >Paramater Constructor for create new Ant object.
void move() = > Move Ant objects from one point to another point and NULL the old point.



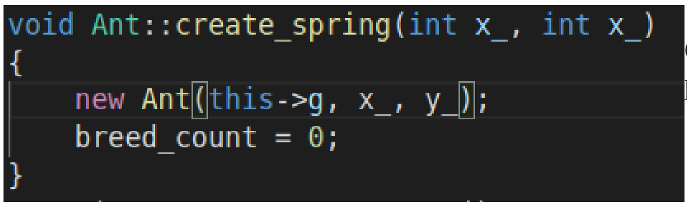
```cpp
void Ant::move()
{
    breed_count++;
    Move_Direction direction = g->random_move();
    switch (direction){
        case UP:
            if(g->get_coord(x, y + 1) == NULL && in_grid(x, y + 1))
            {
                move_coordinates(x, y + 1);
            }
            break;
        case DOWN:
            if(g->get_coord(x, y - 1) == NULL && in_grid(x, y - 1))
            {
                move_coordinates(x, y - 1);
            }
            break;
        case LEFT:
            if(g->get_coord(x - 1, y) == NULL && in_grid(x - 1, y))
            {
                move_coordinates(x - 1, y);
            }
            break;
        case RIGHT:
            if(g->get_coord(x + 1, y) == NULL && in_grid(x + 1, y))
            {
                move_coordinates(x + 1, y);
            }
            break;
        default:
            break;
    }

}
```

breed_count is incread for breed() function
random_move() is create random move direction(Up,Down,Left,Right) and keep in direction variable.
get_coord is return the coordinates(x and y) of this object in grid array.
If Up, Down, Right, Left direction is empty Ant object move there If 4 direction is not empty Ant object stays in point.



```cpp
void Ant::breed()
{
    if(breed_count >= BREED_ANTS)
    {
        int Mutaion_Possible = rand() % 100;
    if(Mutaion_Possible <= POSSIBAL_POISON_ANTS){
        if(g->get_coord(x, y + 1) == NULL && in_grid(x, y + 1)){
            PoisonAnt *n_Poison = new PoisonAnt(g,x, y+1);
            breed_count = 0;
        }
        else if(g->get_coord(x, y - 1) == NULL && in_grid(x, y - 1)){
            PoisonAnt *n_Poison = new PoisonAnt(g,x, y-1);
            breed_count = 0;
        }
        else if(g->get_coord(x - 1, y) == NULL && in_grid(x - 1, y)){
            PoisonAnt *n_Poison = new PoisonAnt(g,x-1, y);
            breed_count = 0;
        }
        else if(g->get_coord(x + 1, y) == NULL && in_grid(x + 1, y)){
                PoisonAnt *n_Poison = new PoisonAnt(g,x, x+1);
                breed_count = 0;
        }
    }
    else
        breed_adjacent();
    }
```

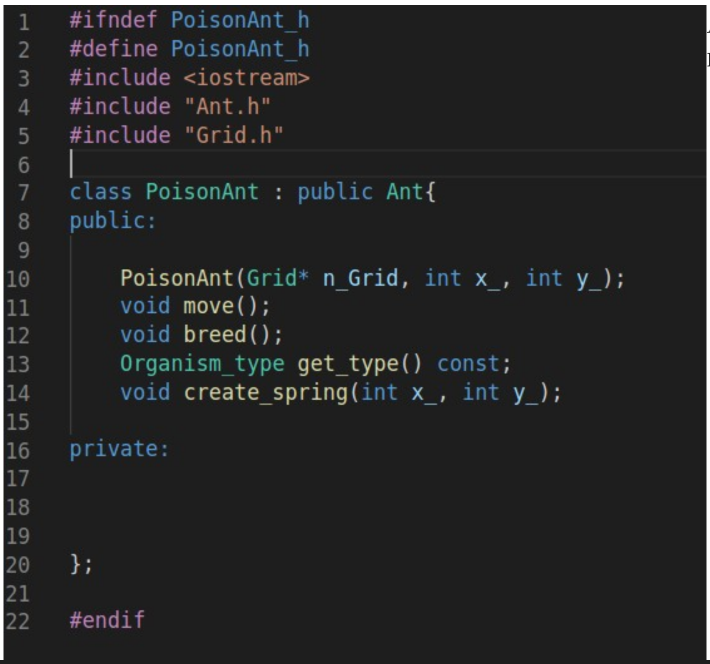
void breed();
BREED_ANTS = Survives number of Ant Class
If Mutation possible occurs Ant object create Poison ant offspring instead Ant offspring
If Mutation possible doesnt occurs;
If any direction is empty and breed count is equal Ant object create new offspring to that point call the breed_adjacent() function from Organism class overload.
And of the breeding breed_count reset.



```cpp
void Ant::create_spring(int x_, int x_)
{
    new Ant(this->g, x_, y_);
    breed_count = 0;
}
```

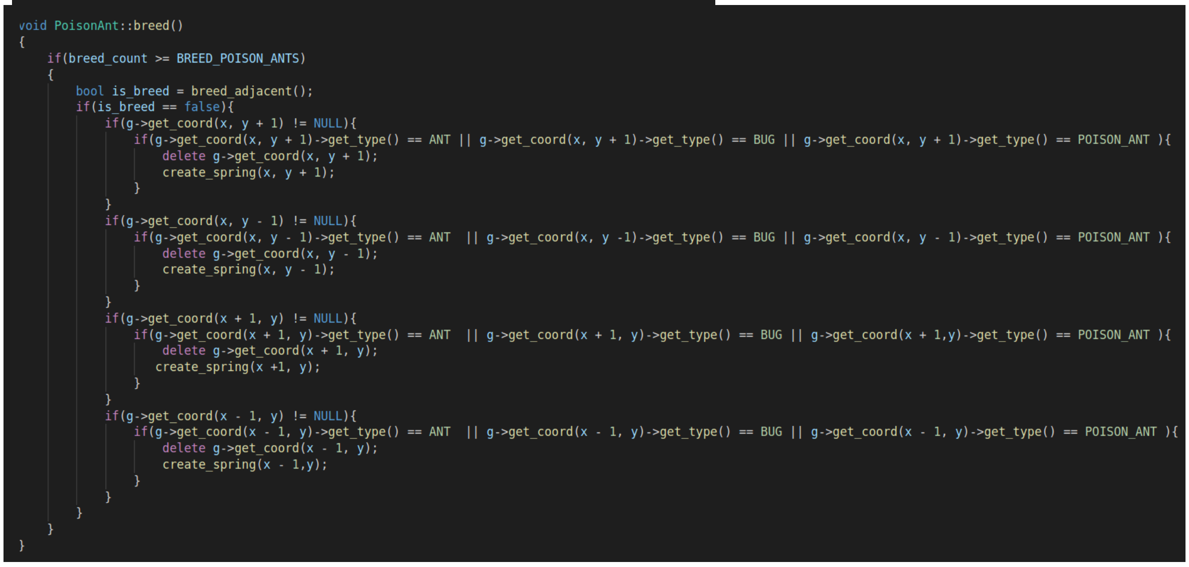
Create new Ant offspring to point x_ and y_ from parameter and breed_count reset.

POISON ANT CLASS



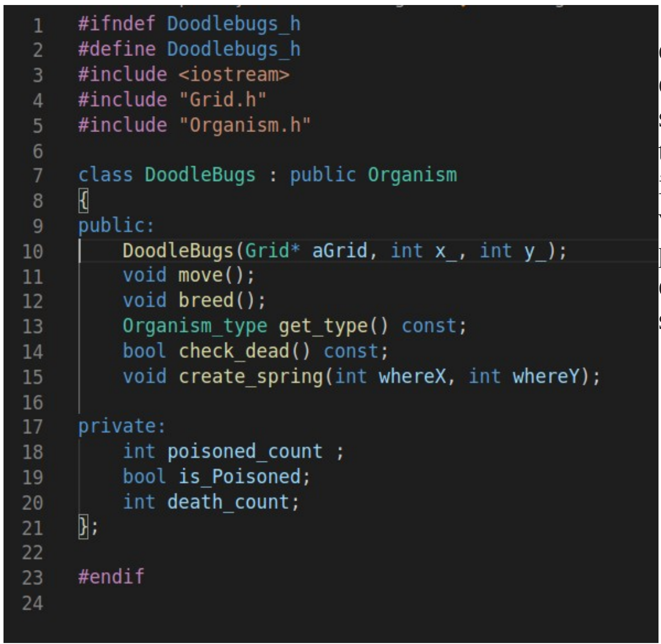
```
#ifndef PoisonAnt_h
#define PoisonAnt_h
#include <iostream>
#include "Ant.h"
#include "Grid.h"

class PoisonAnt : public Ant{
public:

    PoisonAnt(Grid* n_Grid, int x_, int y_);
    void move();
    void breed();
    Organism_type get_type() const;
    void create_spring(int x_, int y_);

private:



};

#endif
```

Ant and Poison ant class have almost same member functions.



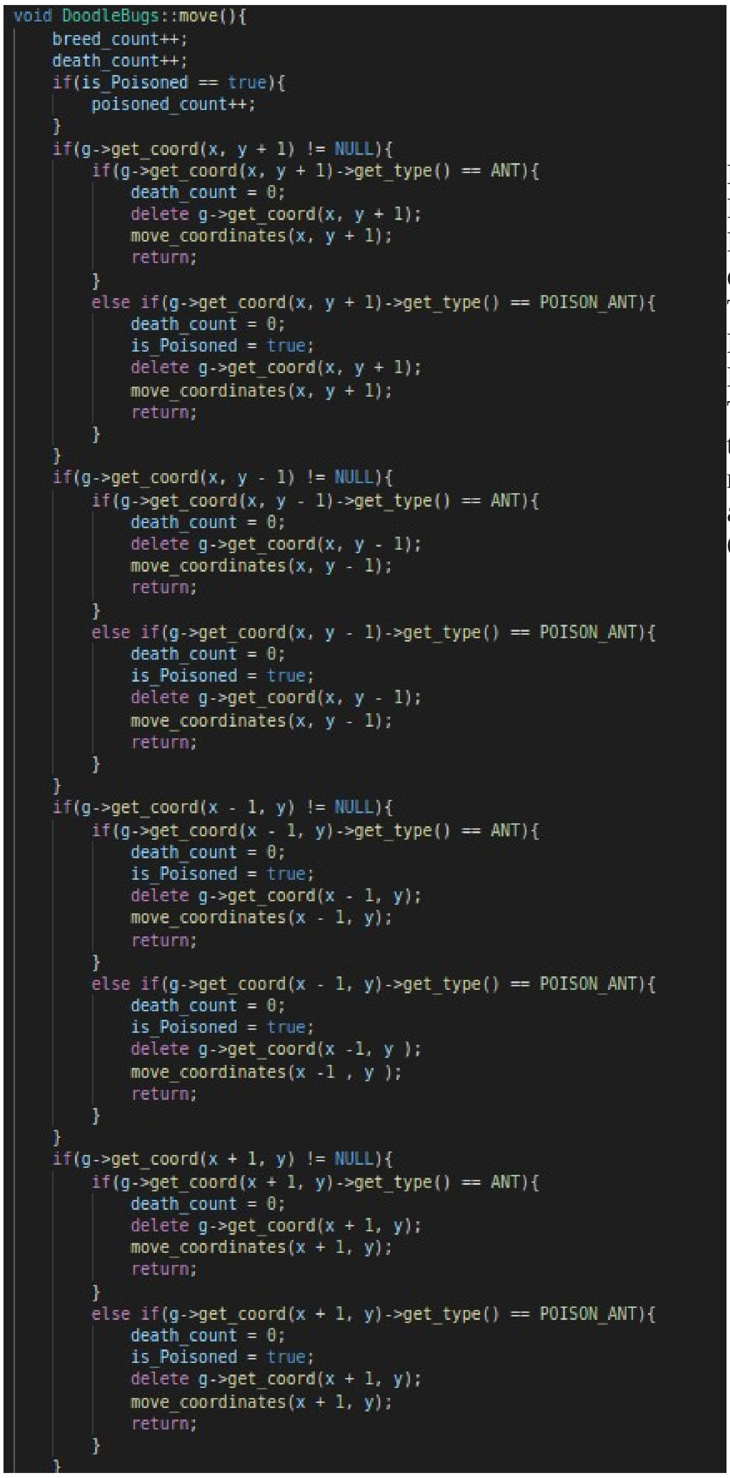
```
void PoisonAnt::breed()
{
    if(breed_count >= BREED_POISON_ANTS)
    {
        bool is_breed = breed_adjacent();
        if(is_breed == false){
            if(g->get_coord(x, y + 1) != NULL){
                if(g->get_coord(x, y + 1)->get_type() == ANT || g->get_coord(x, y + 1)->get_type() == BUG || g->get_coord(x, y + 1)->get_type() == POISON_ANT ){
                    delete g->get_coord(x, y + 1);
                    create_spring(x, y + 1);
                }
            }
            if(g->get_coord(x, y - 1) != NULL){
                if(g->get_coord(x, y - 1)->get_type() == ANT  || g->get_coord(x, y -1)->get_type() == BUG || g->get_coord(x, y - 1)->get_type() == POISON_ANT ){
                    delete g->get_coord(x, y - 1);
                    create_spring(x, y - 1);
                }
            }
            if(g->get_coord(x + 1, y) != NULL){
                if(g->get_coord(x + 1, y)->get_type() == ANT  || g->get_coord(x + 1, y)->get_type() == BUG || g->get_coord(x + 1,y)->get_type() == POISON_ANT ){
                    delete g->get_coord(x + 1, y);
                  create_spring(x +1, y);
                }
            }
            if(g->get_coord(x - 1, y) != NULL){
                if(g->get_coord(x - 1, y)->get_type() == ANT  || g->get_coord(x - 1, y)->get_type() == BUG || g->get_coord(x - 1, y)->get_type() == POISON_ANT ){
                    delete g->get_coord(x - 1, y);
                    create_spring(x - 1,y);
                }
            }
        }
    }
}
```

Void breed();
If a poisonous ant survives BREED_POISON_ANTS(initally 4) times then the ant will breed.
If the all randomly selected cell is not empty Poison ants kill the cells ant create new Poison ant offspring
Other functions same as Ant class.
DOODLEBUGS CLASS



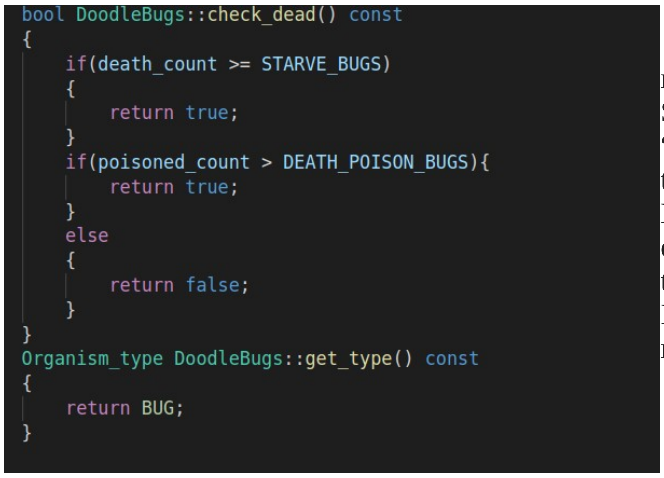
```
#ifndef Doodlebugs_h
#define Doodlebugs_h
#include <iostream>
#include "Grid.h"
#include "Organism.h"

class DoodleBugs : public Organism
{
public:
    DoodleBugs(Grid* aGrid, int x_, int y_);
    void move();
    void breed();
    Organism_type get_type() const;
    bool check_dead() const;
    void create_spring(int whereX, int whereY);

private:
    int poisoned_count ;
    bool is_Poisoned;
    int death_count;
};

#endif

```

death_count = It is a count that If a doodlebug has not eaten an ant within the last STARVE_BUGS time steps and if death_count equal to STARVE_BUGS, at the end of the time step it will starve and die.
is_Poisened = If Doodlebugs eat a Poison ant this variable become true.
poison_count = It is a count that If a doodlebug has eaten a poison ant it can only live two time steps.



```cpp
void DoodleBugs::move(){
    breed_count++;
    death_count++;
    if(is_Poisoned == true){
        poisoned_count++;
    }
    if(g->get_coord(x, y + 1) != NULL){
        if(g->get_coord(x, y + 1)->get_type() == ANT){
            death_count = 0;
            delete g->get_coord(x, y + 1);
            move_coordinates(x, y + 1);
            return;
        }
        else if(g->get_coord(x, y + 1)->get_type() == POISON_ANT){
            death_count = 0;
            is_Poisoned = true;
            delete g->get_coord(x, y + 1);
            move_coordinates(x, y + 1);
            return;
        }
    }
    if(g->get_coord(x, y - 1) != NULL){
        if(g->get_coord(x, y - 1)->get_type() == ANT){
            death_count = 0;
            delete g->get_coord(x, y - 1);
            move_coordinates(x, y - 1);
            return;
        }
        else if(g->get_coord(x, y - 1)->get_type() == POISON_ANT){
            death_count = 0;
            is_Poisoned = true;
            delete g->get_coord(x, y - 1);
            move_coordinates(x, y - 1);
            return;
        }
    }
    if(g->get_coord(x - 1, y) != NULL){
        if(g->get_coord(x - 1, y)->get_type() == ANT){
            death_count = 0;
            is_Poisoned = true;
            delete g->get_coord(x - 1, y);
            move_coordinates(x - 1, y);
            return;
        }
        else if(g->get_coord(x - 1, y)->get_type() == POISON_ANT){
            death_count = 0;
            is_Poisoned = true;
            delete g->get_coord(x -1, y );
            move_coordinates(x -1 , y );
            return;
        }
    }
    if(g->get_coord(x + 1, y) != NULL){
        if(g->get_coord(x + 1, y)->get_type() == ANT){
            death_count = 0;
            delete g->get_coord(x + 1, y);
            move_coordinates(x + 1, y);
            return;
        }
        else if(g->get_coord(x + 1, y)->get_type() == POISON_ANT){
            death_count = 0;
            is_Poisoned = true;
            delete g->get_coord(x + 1, y);
            move_coordinates(x + 1, y);
            return;
        }
    }
```

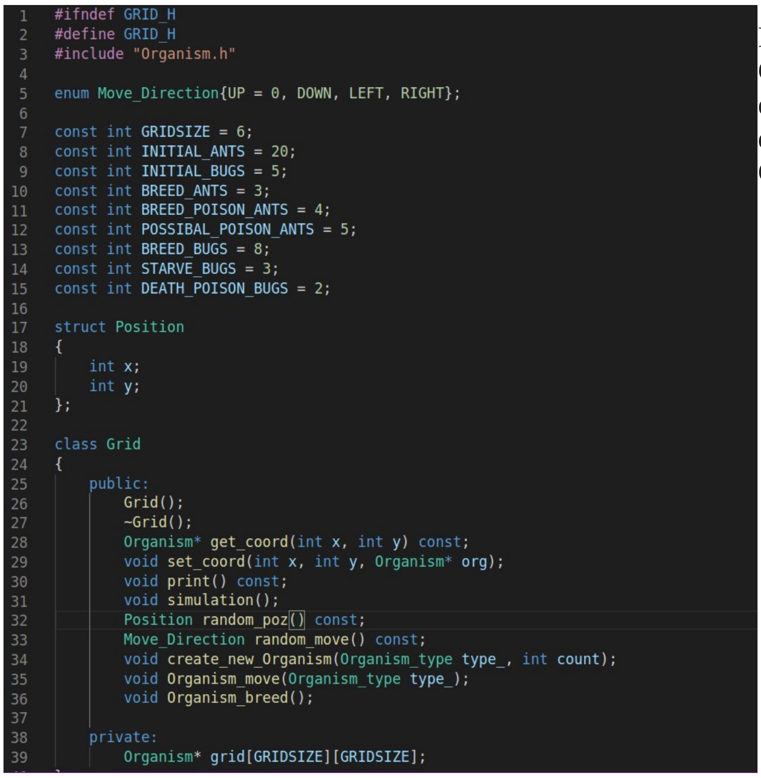
Move function is almost same as Ant and Poison ant.
But if there is an adjacent ant(or poison ant) on up down and right Doodlebugs eat ant.
This function if Doodlebugs eat ant move Doodlebugs to Ant pozition and NULL the Doodlebugs old pozition.
This function use get_coord function for takes Doodledugs pozition in Grid.
move_coordinates() => move Doodlebugs x and y to new x and y.
Other function same as Ant and Poison ant



```cpp
bool DoodleBugs::check_dead() const
{
    if(death_count >= STARVE_BUGS)
    {
        return true;
    }
    if(poisoned_count > DEATH_POISON_BUGS){
        return true;
    }
    else
    {
        return false;
    }
}
Organism_type DoodleBugs::get_type() const
{
    return BUG;
}
```

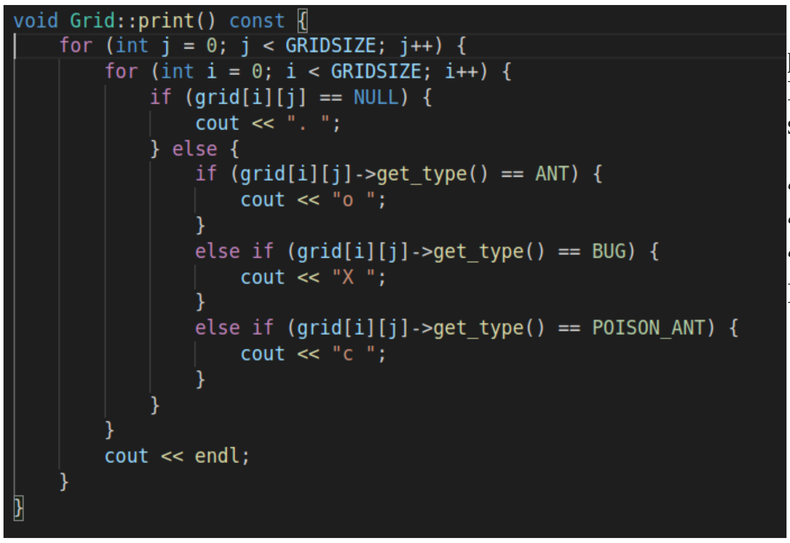
check_dead funcion determine Doddlebugs die or not.
STARVE_BUGS(initially 3) is keeps in variable “If a doodlebug has not eaten an ant within the last three time steps” this explanation
DEAT_POİSON_BUGS is keeps in variable “If a doodlebug eats a poisonous ant, it can only live two time steps.”
If Doodlebugs satisfy this condition function return true.

GRID CLASS



```cpp
#ifndef GRID_H
#define GRID_H
#include "Organism.h"

enum Move_Direction{UP = 0, DOWN, LEFT, RIGHT};

const int GRIDSIZE = 6;
const int INITIAL_ANTS = 20;
const int INITIAL_BUGS = 5;
const int BREED_ANTS = 3;
const int BREED_POISON_ANTS = 4;
const int POSSIBAL_POISON_ANTS = 5;
const int BREED_BUGS = 8;
const int STARVE_BUGS = 3;
const int DEATH_POISON_BUGS = 2;

struct Position
{
    int x;
    int y;
};

class Grid
{
    public:
        Grid();
        ~Grid();
        Organism* get_coord(int x, int y) const;
        void set_coord(int x, int y, Organism* org);
        void print() const;
        void simulation();
        Position random_poz() const;
        Move_Direction random_move() const;
        void create_new_Organism(Organism_type type_, int count);
        void Organism_move(Organism_type type_);
        void Organism_breed();

    private:
        Organism* grid[GRIDSIZE][GRIDSIZE];
```

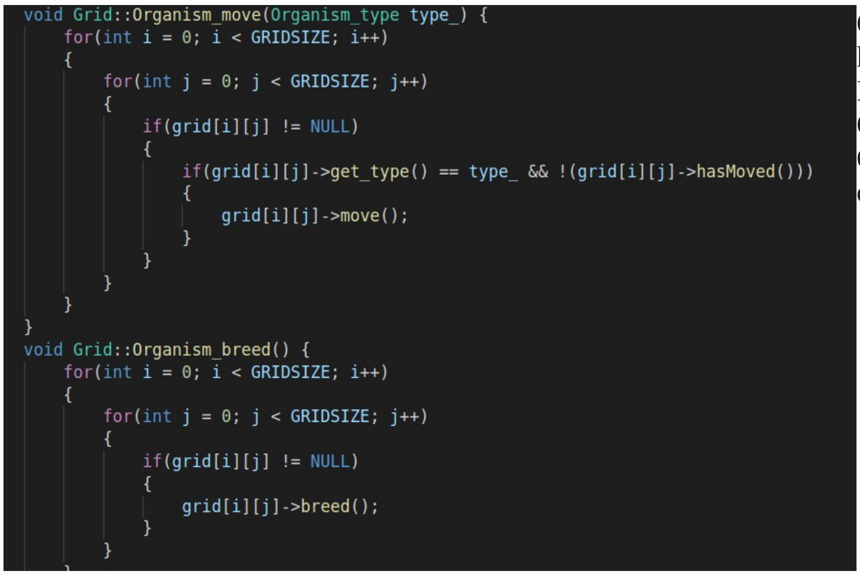
Position struct using that Create a random pozition in Grid for put initial ants and bugs.
enum Move_Direction = It is a enum for create move direction such as UP,DOWN,LEFT,RİGHT.
GRIDSIZE = Size of grid



```cpp
void Grid::print() const {
    for (int j = 0; j < GRIDSIZE; j++) {
        for (int i = 0; i < GRIDSIZE; i++) {
            if (grid[i][j] == NULL) {
                cout << ". ";
            } else {
                if (grid[i][j]->get_type() == ANT) {
                    cout << "o ";
                }
                else if (grid[i][j]->get_type() == BUG) {
                    cout << "X ";
                }
                else if (grid[i][j]->get_type() == POISON_ANT) {
                    cout << "c ";
                }
            }
        }
        cout << endl;
    }
}
```

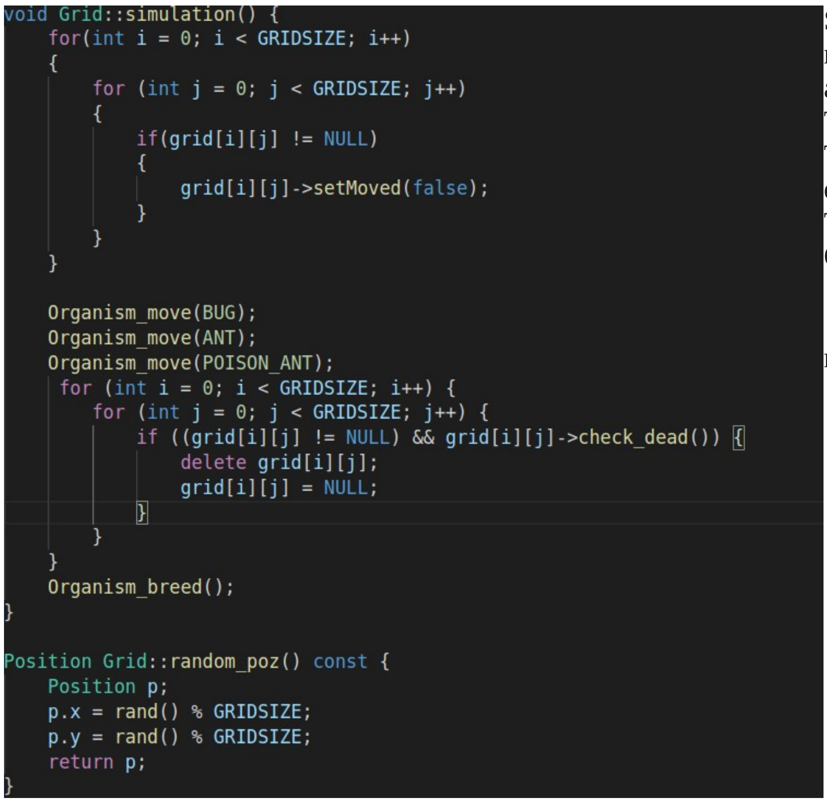
print() = This function display Ant,Posion Ant and Doodlebugs on screen with their character symbol(X,c,o).

“o” for Ant
“c” for Poison Ant
“X” for Doodlebugs
If there is no Orgonism print “.”



```cpp
void Grid::Organism_move(Organism_type type_) {
    for(int i = 0; i < GRIDSIZE; i++)
    {
        for(int j = 0; j < GRIDSIZE; j++)
        {
            if(grid[i][j] != NULL)
            {
                if(grid[i][j]->get_type() == type_ && !(grid[i][j]->hasMoved()))
                {
                    grid[i][j]->move();
                }
            }
        }
    }
}
void Grid::Organism_breed() {
    for(int i = 0; i < GRIDSIZE; i++)
    {
        for(int j = 0; j < GRIDSIZE; j++)
        {
            if(grid[i][j] != NULL)
            {
                grid[i][j]->breed();
            }
        }
```

Organism_move funciton move operation for all kind of Organism.
If organism has never move in tour then Organism has to move in order.
Organism_breed function is act breed operations for all kind of Organism in order.



```
void Grid::simulation() {
    for(int i = 0; i < GRIDSIZE; i++)
    {
        for (int j = 0; j < GRIDSIZE; j++)
        {
            if(grid[i][j] != NULL)
            {
                grid[i][j]->setMoved(false);
            }
        }
    }

    Organism_move(BUG);
    Organism_move(ANT);
    Organism_move(POISON_ANT);
     for (int i = 0; i < GRIDSIZE; i++) {
        for (int j = 0; j < GRIDSIZE; j++) {
            if ((grid[i][j] != NULL) && grid[i][j]->check_dead()) {
                delete grid[i][j];
                grid[i][j] = NULL;
            }
        }
    }
    Organism_breed();
}

Position Grid::random_poz() const {
    Position p;
    p.x = rand() % GRIDSIZE;
    p.y = rand() % GRIDSIZE;
    return p;
}
```

Simulation function firstly all has_moved variable makes false because no Organism has moved yet at the beginning of the round
Then All Organism has to move in order.
Then if Doodlebugs is dead function NULL all dead Doodlebugs in Grid.
Then all function breed in order in Organism_breed() function.

random_poz() function create random Pozition.

Erdi Bayır 171044063