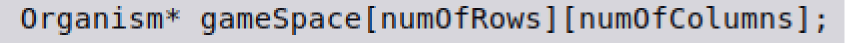
CSE 241 OOP – Programming Project Documentation

The goal is: programming project is to create a simple 2D predator-prey simulation. In this simulation the prey are ants and the predators are doodlebugs.

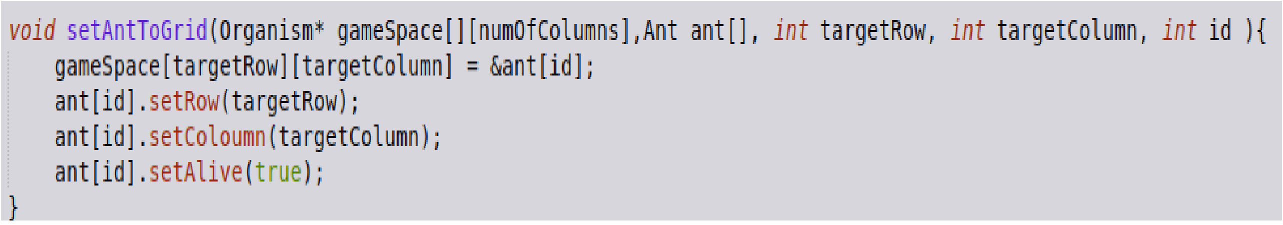
- *These critters live in a world composed of a grid of cells and only one critter may occupy a cell at a time.*



```
Organism* gameSpace[numOfRows][numOfColumns];
```

main.cpp line : 18

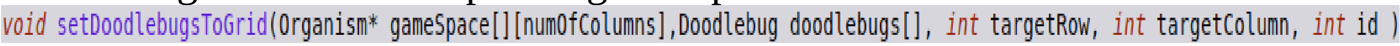
Every cells are kept in the organism pointer array thanks to upcasting.



```
void setAntToGrid(Organism* gameSpace[][numOfColumns],Ant ant[], int targetRow, int targetColumn, int id ){
    gameSpace[targetRow][targetColumn] = &ant[id];
    ant[id].setRow(targetRow);
    ant[id].setColoumn(targetColumn);
    ant[id].setAlive(true);
}
```

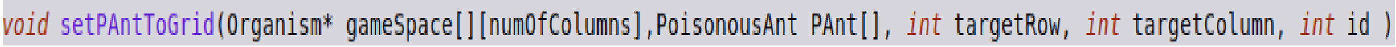
Ant.cpp line : 35

Same logic in the other upcasting examples.



```
void setDoodlebugsToGrid(Organism* gameSpace[][numOfColumns],Doodlebug doodlebugs[], int targetRow, int targetColumn, int id )
```

Doodlebug.cpp line : 150

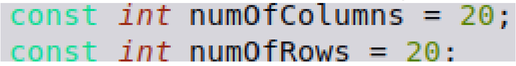


```
void setPAntToGrid(Organism* gameSpace[][numOfColumns],PoisonousAnt PAnt[], int targetRow, int targetColumn, int id )
```

PoisonousAnt.cpp line : 32

I used the these function as global function because I think these are bridge of the between main function and classes so its not appropriate of the principle of least privilege to make member function.

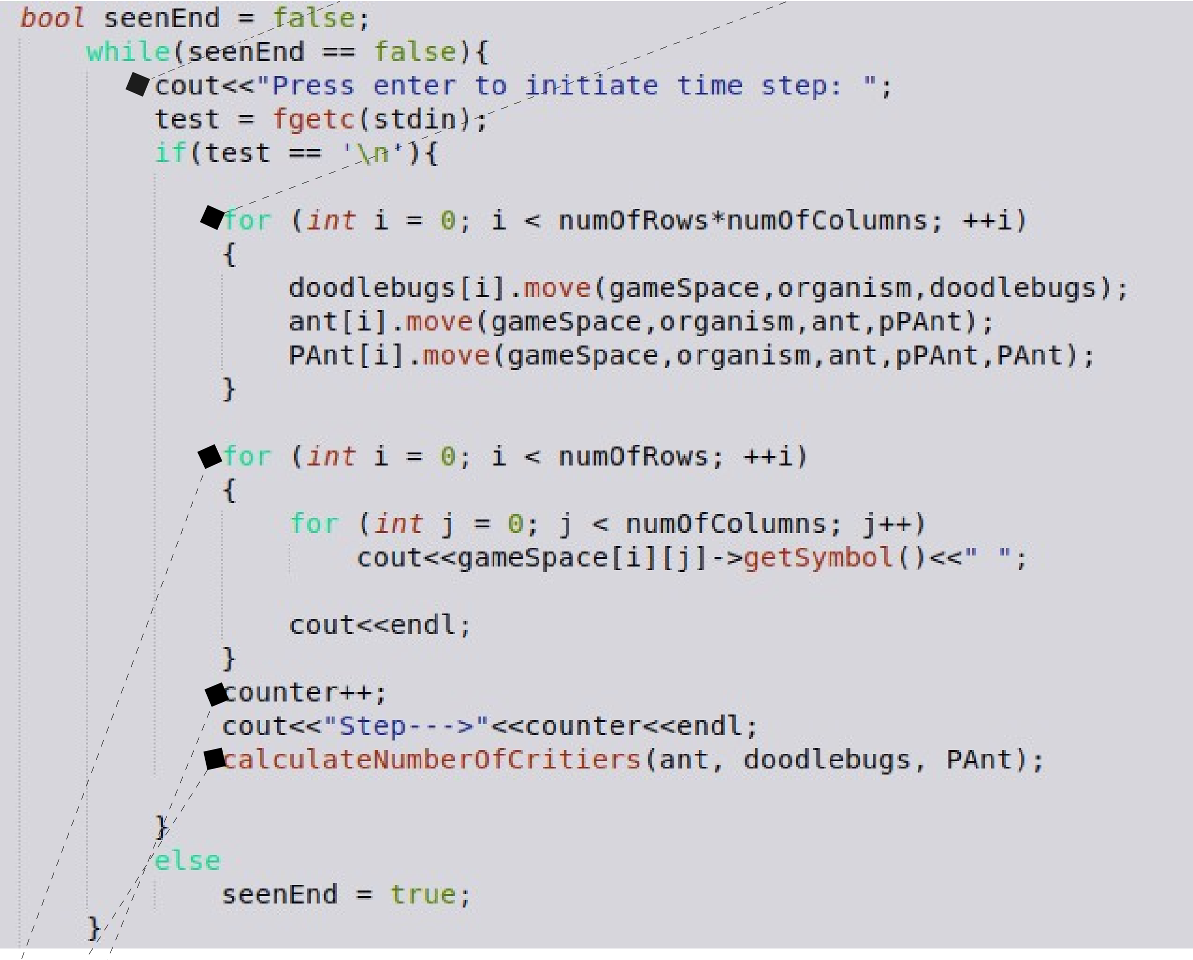
- *(grid size should be adjustable by changing constant global variables).*



```
const int numOfColumns = 20;
const int numOfRows = 20;
```

main.cpp line: 3-4

- *Time is simulated in time steps. Each critter performs some action every time step*

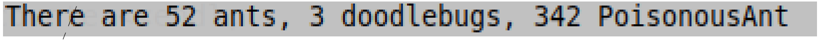


```
bool seenEnd = false;
    while(seenEnd == false){
        cout<<"Press enter to initiate time step: ";
        test = fgetc(stdin);
        if(test == '\n'){

            for (int i = 0; i < numOfRows*numOfColumns; ++i)
            {
                doodlebugs[i].move(gameSpace,organism,doodlebugs);
                ant[i].move(gameSpace,organism,ant,pPAnt);
                PAnt[i].move(gameSpace,organism,ant,pPAnt,PAnt);
            }

            for (int i = 0; i < numOfRows; ++i)
            {
                for (int j = 0; j < numOfColumns; j++)
                    cout<<gameSpace[i][j]->getSymbol()<<" ";

                cout<<endl;
            }
            counter++;
            cout<<"Step--->"<<counter<<endl;
            calculateNumberOfCritiers(ant, doodlebugs, PAnt);

        }
        else
            seenEnd = true;
    }
```

main.cpp line : 68-96

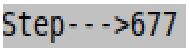
This for loop print the full of the grid.

This function prints number of the critiers



```
There are 52 ants, 3 doodlebugs, 342 PoisonousAnt
```

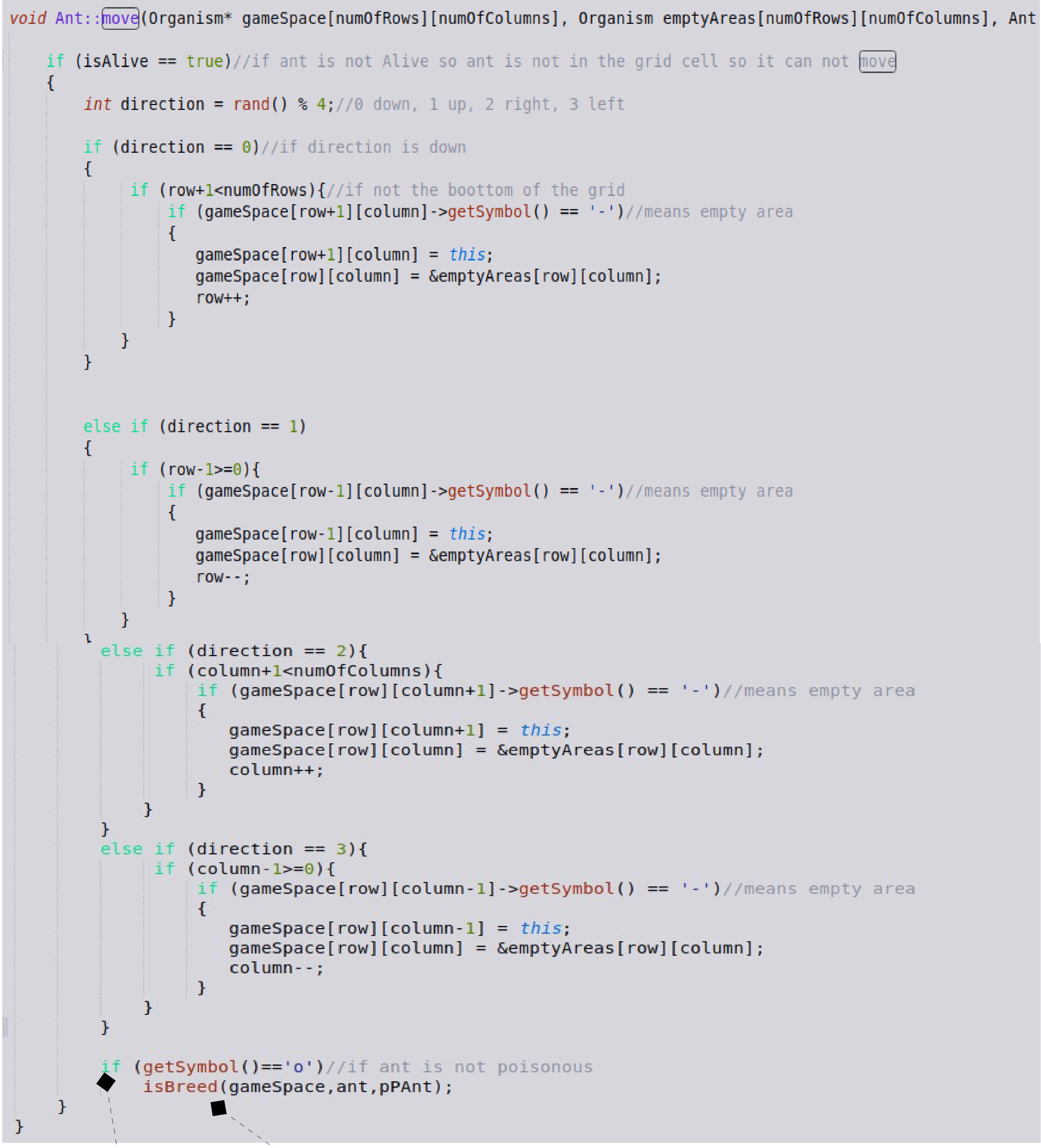
This counter counts the steps



```
Step--->677
```

The ants behave

- *Move. Every time step, randomly try to move up, down, left or right. If the neighboring cell in the selected direction is occupied or would move the ant off the grid, then the ant stays in the current cell.*

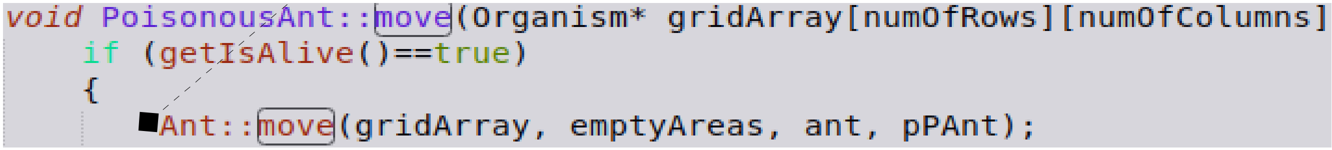


```
void Ant::move(Organism* gameSpace[numOfRows][numOfColumns], Organism emptyAreas[numOfRows][numOfColumns], Ant

    if (isAlive == true)//if ant is not Alive so ant is not in the grid cell so it can not move
    {
        int direction = rand() % 4;//0 down, 1 up, 2 right, 3 left

        if (direction == 0)//if direction is down
        {
            if (row+1<numOfRows){//if not the boottom of the grid
                if (gameSpace[row+1][column]->getSymbol() == '-')//means empty area
                {
                    gameSpace[row+1][column] = this;
                    gameSpace[row][column] = &emptyAreas[row][column];
                    row++;
                }
            }
        }


        else if (direction == 1)
        {
            if (row-1>=0){
                if (gameSpace[row-1][column]->getSymbol() == '-')//means empty area
                {
                    gameSpace[row-1][column] = this;
                    gameSpace[row][column] = &emptyAreas[row][column];
                    row--;
                }
            }
        }
        else if (direction == 2){
            if (column+1<numOfColumns){
                if (gameSpace[row][column+1]->getSymbol() == '-')//means empty area
                {
                    gameSpace[row][column+1] = this;
                    gameSpace[row][column] = &emptyAreas[row][column];
                    column++;
                }
            }
        }
        else if (direction == 3){
            if (column-1>=0){
                if (gameSpace[row][column-1]->getSymbol() == '-')//means empty area
                {
                    gameSpace[row][column-1] = this;
                    gameSpace[row][column] = &emptyAreas[row][column];
                    column--;
                }
            }
        }

        if (getSymbol()=='o')//if ant is not poisonous
            isBreed(gameSpace,ant,pPAnt);
    }
}
```

Ant.cpp line : 57-111 'Ant::move(...)'

- *Breed. If an ant survives for three time steps, then at the end of the time step (i.e. after moving) the ant will breed.*

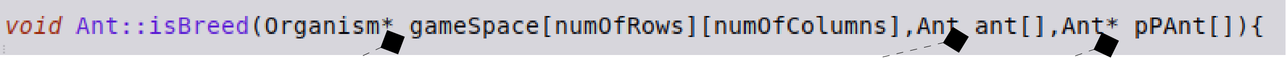
I used getSymbol() function to classified the whether object is Ant or not because I also use it for PosiounousAnt::Move(…)



```
void PoisonousAnt::move(Organism* gridArray[numOfRows][numOfColumns]
    if (getIsAlive()==true)
    {
        Ant::move(gridArray, emptyAreas, ant, pPAnt);
```

PousinousAnt.cpp line : 14

- *This is simulated by creating a new ant in an adjacent (up, down, left, or right) cell that is empty. If there is no empty cell available then no breeding occurs.*



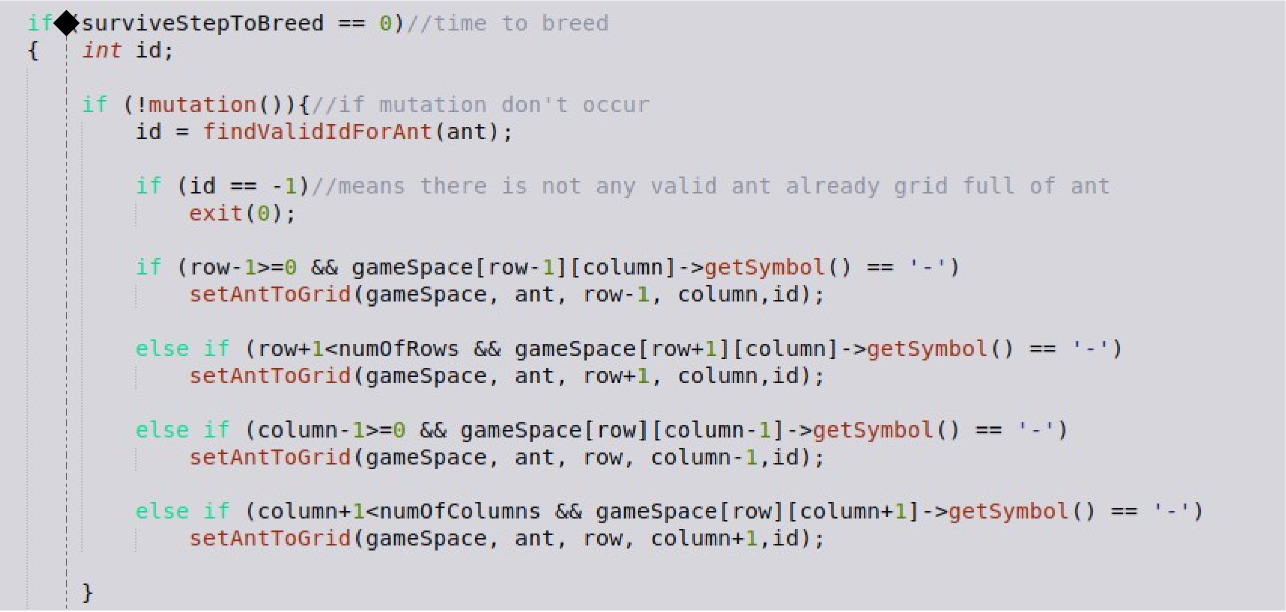
```
void Ant::isBreed(Organism* gameSpace[numOfRows][numOfColumns],Ant ant[],Ant* pPAnt[]){
```

Ant.cpp line : 113

Organism* gameSpace[][] is grid area it holds every objects(Ant, PoisousAnt, Doodlebug) thanks to upcasting.

Ant ant[] is the hold all ants which declared in the main() function, If breed will be succesful, program pick one of them which is not alive because living ant cannot born

Ant* pPant[] holds all Poisonous ants thanks to upcasting because there is a probability of the mutation if mutation will be succesful breed must be Poisonous ant

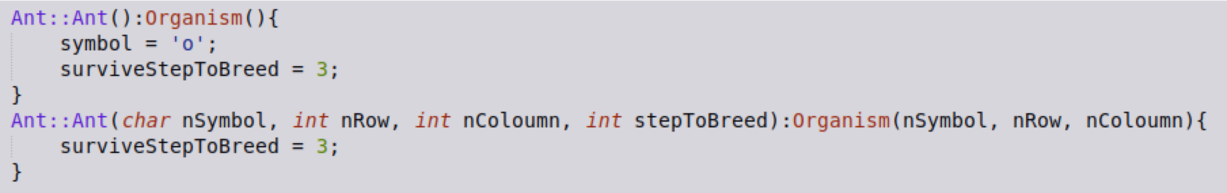


```
  if (surviveStepToBreed == 0)//time to breed
  {   int id;

      if (!mutation()){//if mutation don't occur
          id = findValidIdForAnt(ant);

          if (id == -1)//means there is not any valid ant already grid full of ant
              exit(0);

          if (row-1>=0 && gameSpace[row-1][column]->getSymbol() == '-')
              setAntToGrid(gameSpace, ant, row-1, column,id);

          else if (row+1<numOfRows && gameSpace[row+1][column]->getSymbol() == '-')
              setAntToGrid(gameSpace, ant, row+1, column,id);

          else if (column-1>=0 && gameSpace[row][column-1]->getSymbol() == '-')
              setAntToGrid(gameSpace, ant, row, column-1,id);

          else if (column+1<numOfColumns && gameSpace[row][column+1]->getSymbol() == '-')
              setAntToGrid(gameSpace, ant, row, column+1,id);

      }
```

findValidIdForAnt(ant);   this function finds dont Alive (which is not in the grid) ant’s id in the ant[] array. If id -1 grid is full of ant and simulation is over because ants are dont die, and dont eat each other.

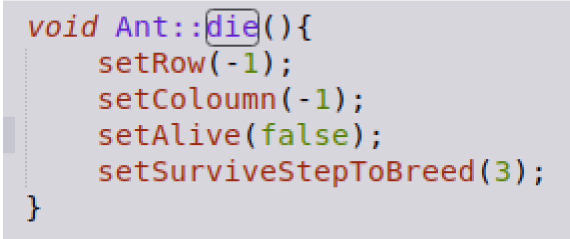
- *Once an offspring is produced an ant cannot producean offspring until three more time steps have elapsed.*

surviveStepToBreed is a protected variable in the Organism class it count remain step to breed. In the mutator functions surviveStepToBreed variable set to 3 and if a ant die there is a die() function to set to surviveStepToBreed 3.



```
Ant::Ant():Organism(){
    symbol = 'o';
    surviveStepToBreed = 3;
}
Ant::Ant(char nSymbol, int nRow, int nColoumn, int stepToBreed):Organism(nSymbol, nRow, nColoumn){
    surviveStepToBreed = 3;
}
```

Ant.cpp line : 7-13

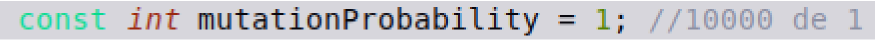


```
void Ant::die(){
    setRow(-1);
    setColoumn(-1);
    setAlive(false);
    setSurviveStepToBreed(3);
}
```

Ant.cpp line : 50-56

If an ant die its surviveStepToBreed variable will be 3. Then if this ant will again set int the grid it will be as a new created object.

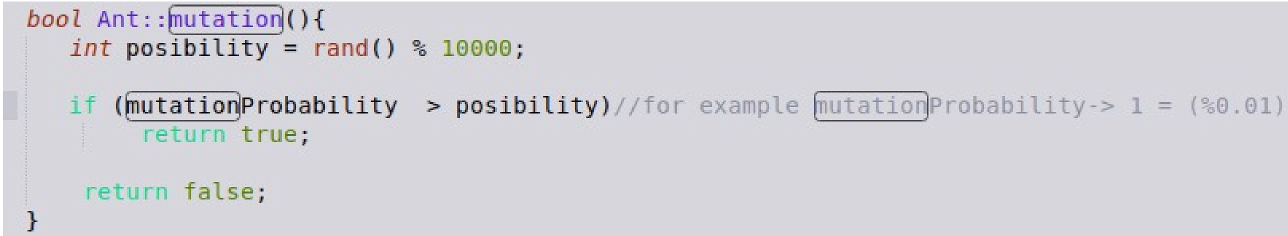
- *There is a possibility of mutation. Randomly, with a small probability, the new ant can be a mutated version.*



```
const int mutationProbability = 1; //10000 de 1
```

main.cpp line : 6

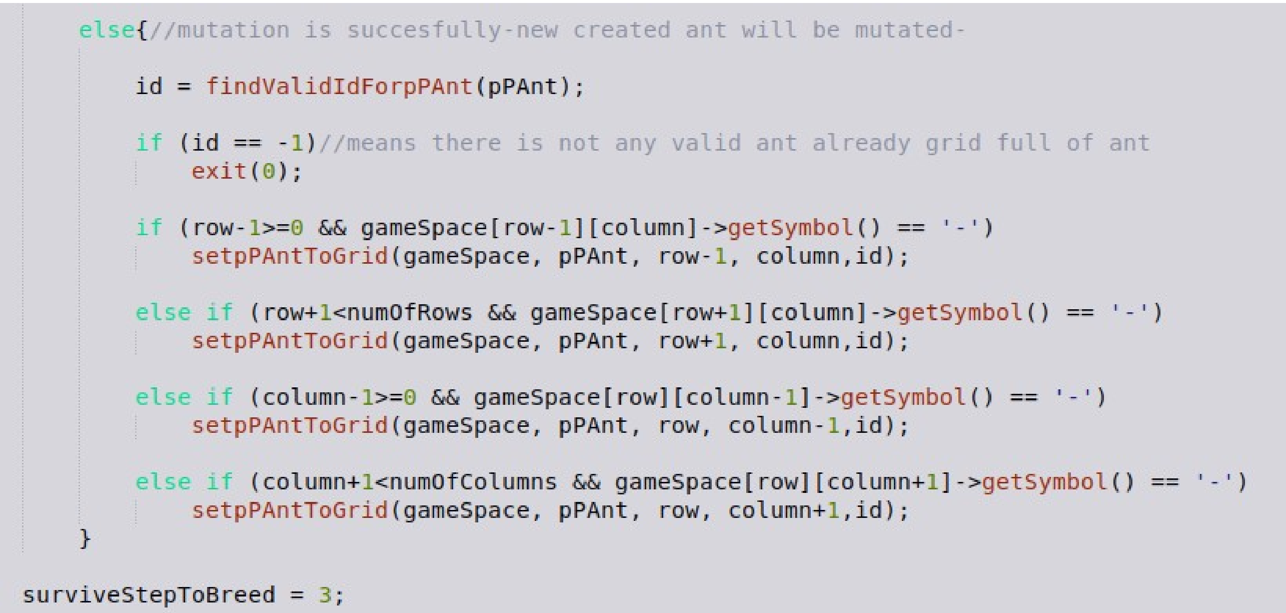
Its constant variable and user can change this probability



```
bool Ant::mutation(){
    int posibility = rand() % 10000;

    if (mutationProbability  > posibility)//for example mutationProbability-> 1 = (%0.01)
        return true;

     return false;
}
```

Ant.cpp line:165-172

If mutation succesfull

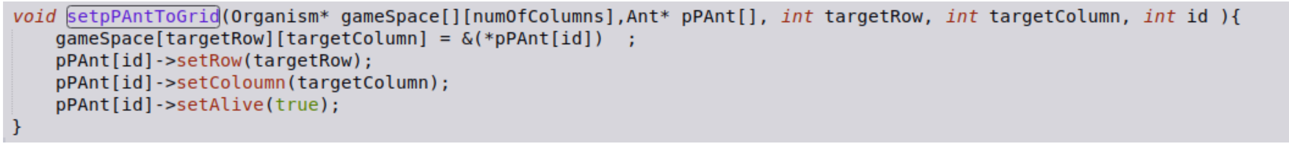


```
    else{//mutation is succesfully-new created ant will be mutated-

        id = findValidIdForpPAnt(pPAnt);

        if (id == -1)//means there is not any valid ant already grid full of ant
            exit(0);

        if (row-1>=0 && gameSpace[row-1][column]->getSymbol() == '-')
            setpPAntToGrid(gameSpace, pPAnt, row-1, column,id);

        else if (row+1<numOfRows && gameSpace[row+1][column]->getSymbol() == '-')
            setpPAntToGrid(gameSpace, pPAnt, row+1, column,id);

        else if (column-1>=0 && gameSpace[row][column-1]->getSymbol() == '-')
            setpPAntToGrid(gameSpace, pPAnt, row, column-1,id);

        else if (column+1<numOfColumns && gameSpace[row][column+1]->getSymbol() == '-')
            setpPAntToGrid(gameSpace, pPAnt, row, column+1,id);
    }

surviveStepToBreed = 3;
```

Ant.cpp line : 138 - 158

findValidIdForpPant(pPant) same logic with the [4]findValidIdForAnt(Ant[]) but pPant parameters is Ant* [] and it holds PosiounosAnt object by upcasting.

SetpPantToGrid(…) also same logic with the [1]these functions puts new objects in the field



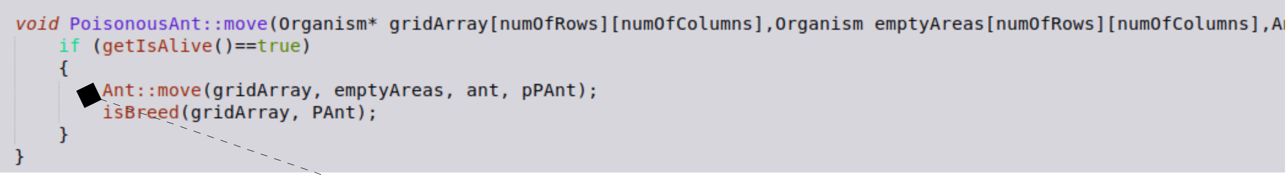
```
void setpPAntToGrid(Organism* gameSpace[][numOfColumns],Ant* pPAnt[], int targetRow, int targetColumn, int id ){
    gameSpace[targetRow][targetColumn] = &(*pPAnt[id])  ;
    pPAnt[id]->setRow(targetRow);
    pPAnt[id]->setColoumn(targetColumn);
    pPAnt[id]->setAlive(true);
}
```

Ant.cpp line : 42-47

The Poisonous ants behave

- *Move. Every time step, randomly try to move up, down, left or right. If the selected direction would move the ant off the grid, then the poisonous ant stays in the current cell.*

Same with the Ants move so I used the Ant::move()

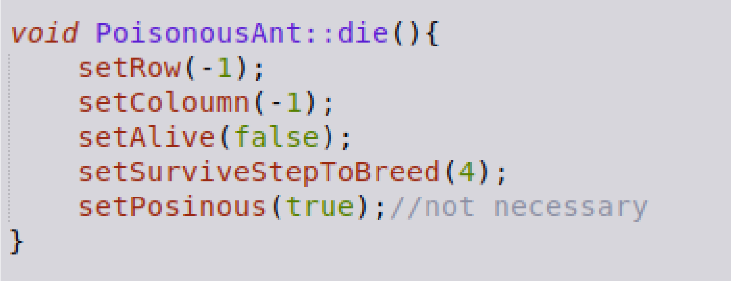


```
void PoisonousAnt::move(Organism* gridArray[numOfRows][numOfColumns],Organism emptyAreas[numOfRows][numOfColumns],A
    if (getIsAlive()==true)
    {
        Ant::move(gridArray, emptyAreas, ant, pPAnt);
        isBreed(gridArray, PAnt);
    }
}
```

PoisonousAnt.cpp line : 14-21

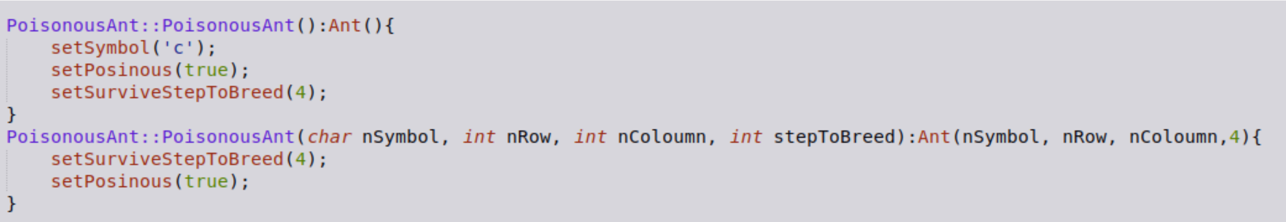
- *Breed. If a poisonous ant survives for four time steps, then at the end of the time step (i.e. after moving) the ant will breed.*

PoisonousAnt's [4]surviveStepToBreed is 4 so it set 4 by the constructors and PoisonousAnt::die() function will set surviveStepToBreed 4 as the Ant class.



```
void PoisonousAnt::die(){
    setRow(-1);
    setColoumn(-1);
    setAlive(false);
    setSurviveStepToBreed(4);
    setPosinous(true);//not necessary
}
```

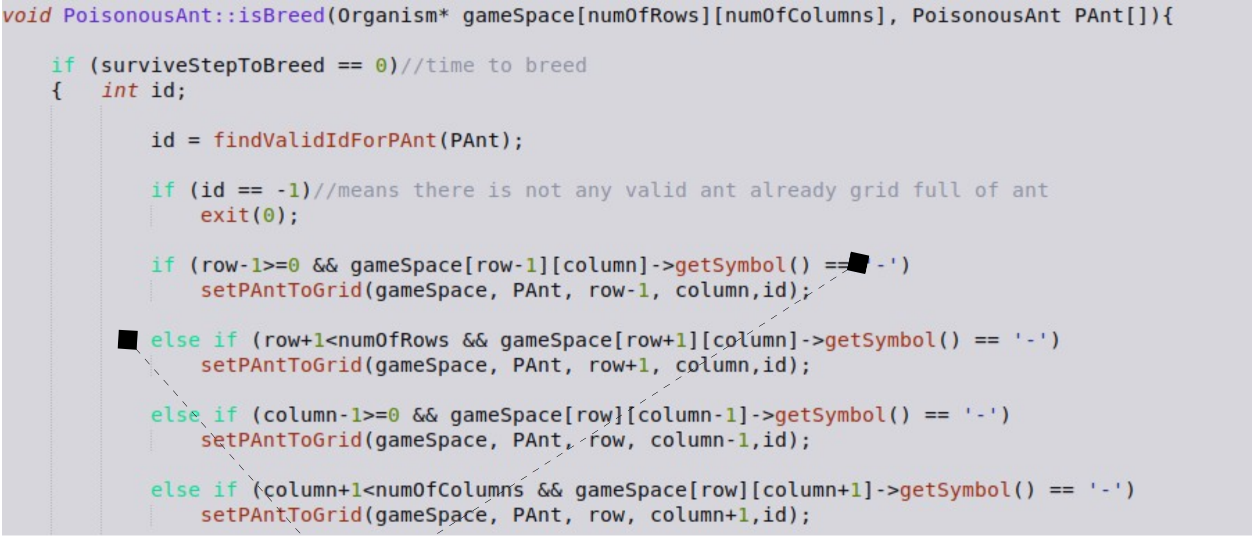
PoisonousAnt.cpp line : 15 - 20



```
PoisonousAnt::PoisonousAnt():Ant(){
    setSymbol('c');
    setPosinous(true);
    setSurviveStepToBreed(4);
}
PoisonousAnt::PoisonousAnt(char nSymbol, int nRow, int nColoumn, int stepToBreed):Ant(nSymbol, nRow, nColoumn,4){
    setSurviveStepToBreed(4);
    setPosinous(true);
}
```

PoisonousAnt.cpp line : 5 – 13

- *This is simulated by creating a new poisonous ant in an adjacent (up, down, left, or right) cell that is empty.*

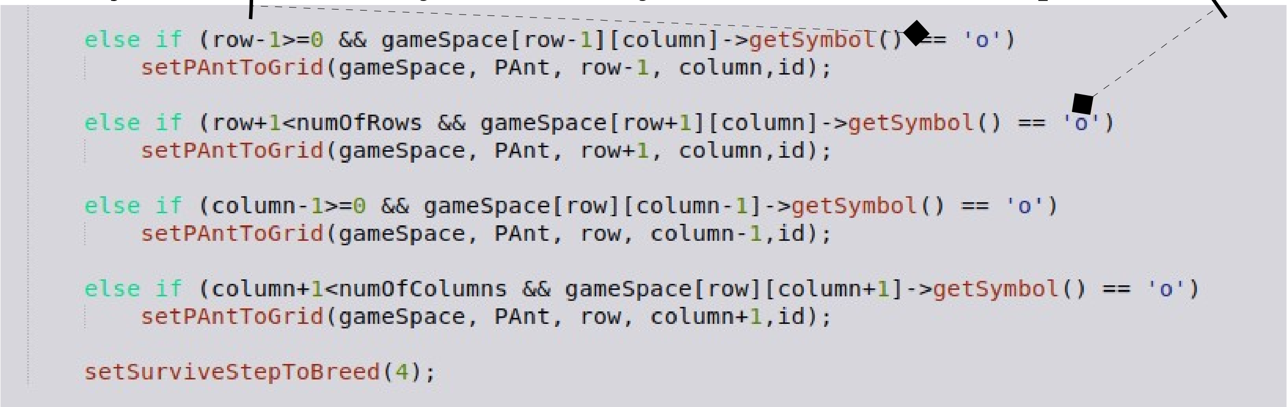


```
void PoisonousAnt::isBreed(Organism* gameSpace[numOfRows][numOfColumns], PoisonousAnt PAnt[]){

    if (surviveStepToBreed == 0)//time to breed
    {   int id;

            id = findValidIdForPAnt(PAnt);

            if (id == -1)//means there is not any valid ant already grid full of ant
                exit(0);

            if (row-1>=0 && gameSpace[row-1][column]->getSymbol() == '-')
                setPAntToGrid(gameSpace, PAnt, row-1, column,id);

            else if (row+1<numOfRows && gameSpace[row+1][column]->getSymbol() == '-')
                setPAntToGrid(gameSpace, PAnt, row+1, column,id);

            else if (column-1>=0 && gameSpace[row][column-1]->getSymbol() == '-')
                setPAntToGrid(gameSpace, PAnt, row, column-1,id);

            else if (column+1<numOfColumns && gameSpace[row][column+1]->getSymbol() == '-')
                setPAntToGrid(gameSpace, PAnt, row, column+1,id);
```

PoisonousAnt.cpp line : 49 – 69

- *If all of the adjacent cells are occupied, the new poisonous ant fights and kills the occupant in the last cell tried(killing means replacing)*

‘- ’represents the empty cell in here code try to put a new offspring in the up, below, right and left adjacent cell if there is not any empty cell code code try to find an ant object in the adjacent cell to kill. ‘o’ represents ant.

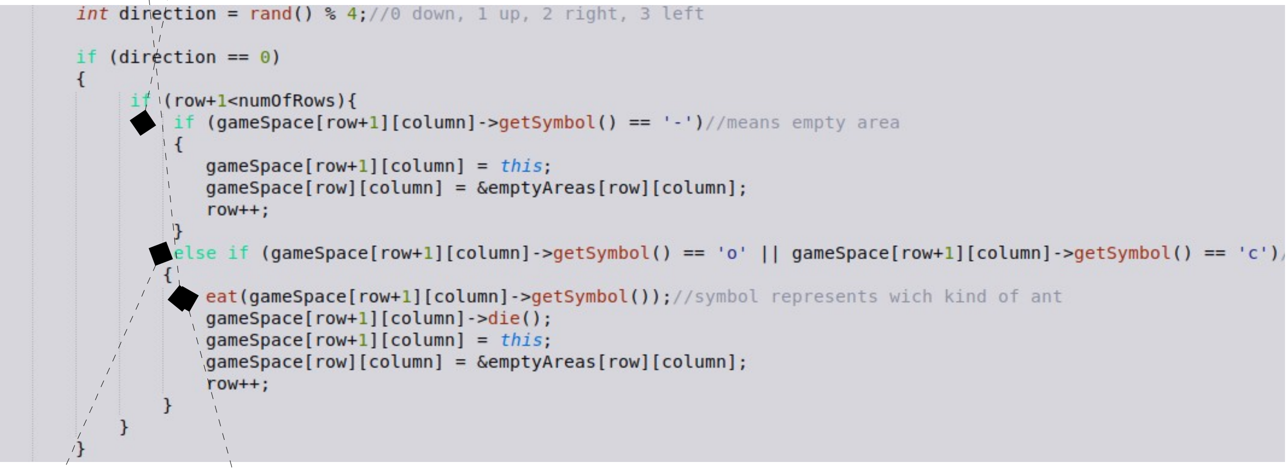


```
            else if (row-1>=0 && gameSpace[row-1][column]->getSymbol() == 'o')
                setPAntToGrid(gameSpace, PAnt, row-1, column,id);

            else if (row+1<numOfRows && gameSpace[row+1][column]->getSymbol() == 'o')
                setPAntToGrid(gameSpace, PAnt, row+1, column,id);

            else if (column-1>=0 && gameSpace[row][column-1]->getSymbol() == 'o')
                setPAntToGrid(gameSpace, PAnt, row, column-1,id);

            else if (column+1<numOfColumns && gameSpace[row][column+1]->getSymbol() == 'o')
                setPAntToGrid(gameSpace, PAnt, row, column+1,id);

            setSurviveStepToBreed(4);
```

PoisonousAnt.cpp line : 69 – 89

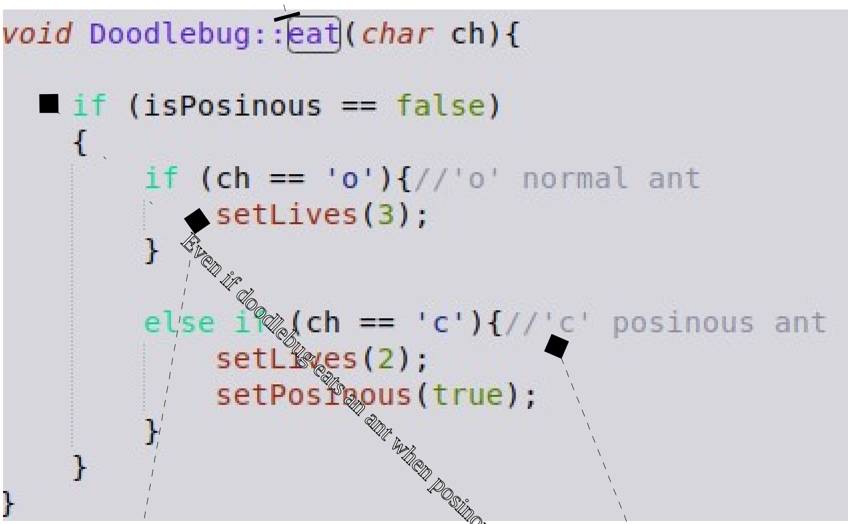
The doodlebugs behave

- *Move. Every time step, if there is an adjacent ant (up, down, left, or right) then the doodlebug will move to that cell and eat the ant. Otherwise the doodlebug moves according to the same rules as the ant. Note that a doodlebug cannot eat other doodlebugs.*



```
int direction = rand() % 4;//0 down, 1 up, 2 right, 3 left

if (direction == 0)
{
      if (row+1<numOfRows){
        if (gameSpace[row+1][column]->getSymbol() == '-')//means empty area
        {
            gameSpace[row+1][column] = this;
            gameSpace[row][column] = &emptyAreas[row][column];
            row++;
        }
        else if (gameSpace[row+1][column]->getSymbol() == 'o' || gameSpace[row+1][column]->getSymbol() == 'c')
        {
            eat(gameSpace[row+1][column]->getSymbol());//symbol represents wich kind of ant
            gameSpace[row+1][column]->die();
            gameSpace[row+1][column] = this;
            gameSpace[row][column] = &emptyAreas[row][column];
            row++;
        }
      }
}
```

This is eat check in here if the target direction is ‘o’Ant or ‘c’ PoisonousAnt, the doodlebug will eat the this ant.



```
void Doodlebug::eat(char ch){

    if (isPosinous == false)
    {
        if (ch == 'o'){//'o' normal ant
            setLives(3);
        }

        else if (ch == 'c'){//'c' posinous ant
            setLives(2);
            setPosinous(true);
        }
    }
}
```

Even if doodlebug eats an ant when posinous its lives dont increase!

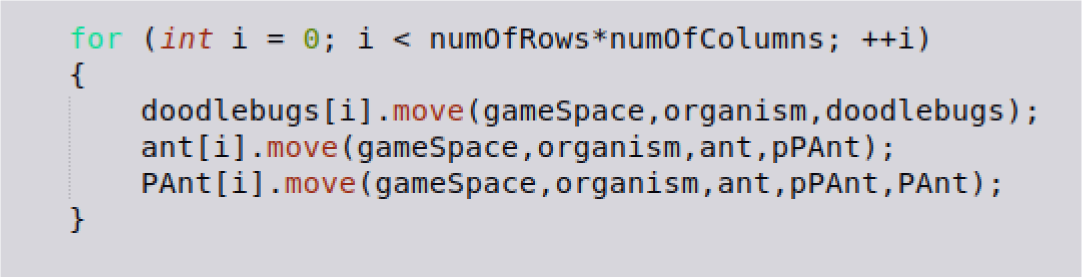
- *Starve. If a doodlebug has not eaten an ant within the last three time steps, then at the end of the third time step it will starve and die. The doodlebug should then be removed from the grid of cells.*
- *A doodlebug can eat ants and poisonous ants. If a doodlebug eats a poisonous ant, it can only live two time steps.*

- *Breed. If a doodlebug survives for eight time steps, then at the end of the time step it will spawn off a new doodlebug in the same manner as the ant.*

Its same as the ant so I’m not gonno mentione about it.

General Rules

- During one turn, all the doodlebugs should move before the ants. All the ants move before the poisonous ants.

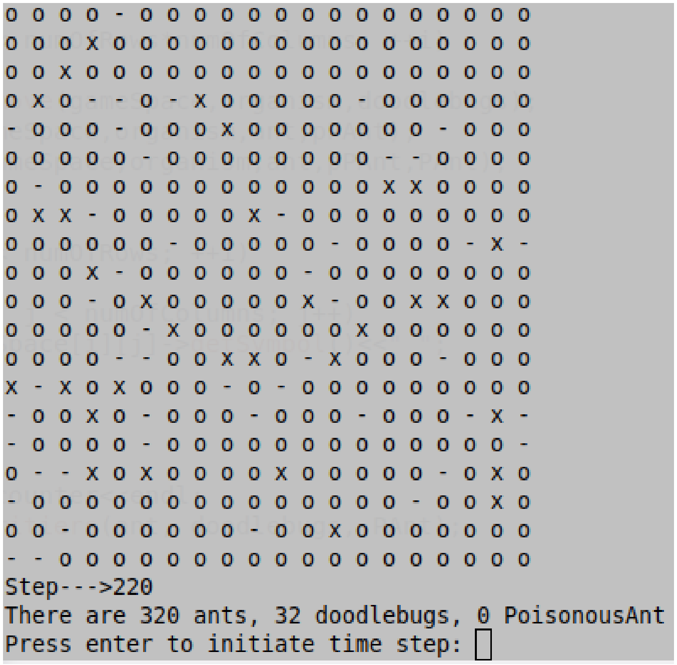


```
for (int i = 0; i < numOfRows*numOfColumns; ++i)
{
    doodlebugs[i].move(gameSpace,organism,doodlebugs);
    ant[i].move(gameSpace,organism,ant,pPAnt);
    PAnt[i].move(gameSpace,organism,ant,pPAnt,PAnt);
}
```

main.cpp line : 75-80

Here first doodlebugs then ant at the end poisonous ants move.

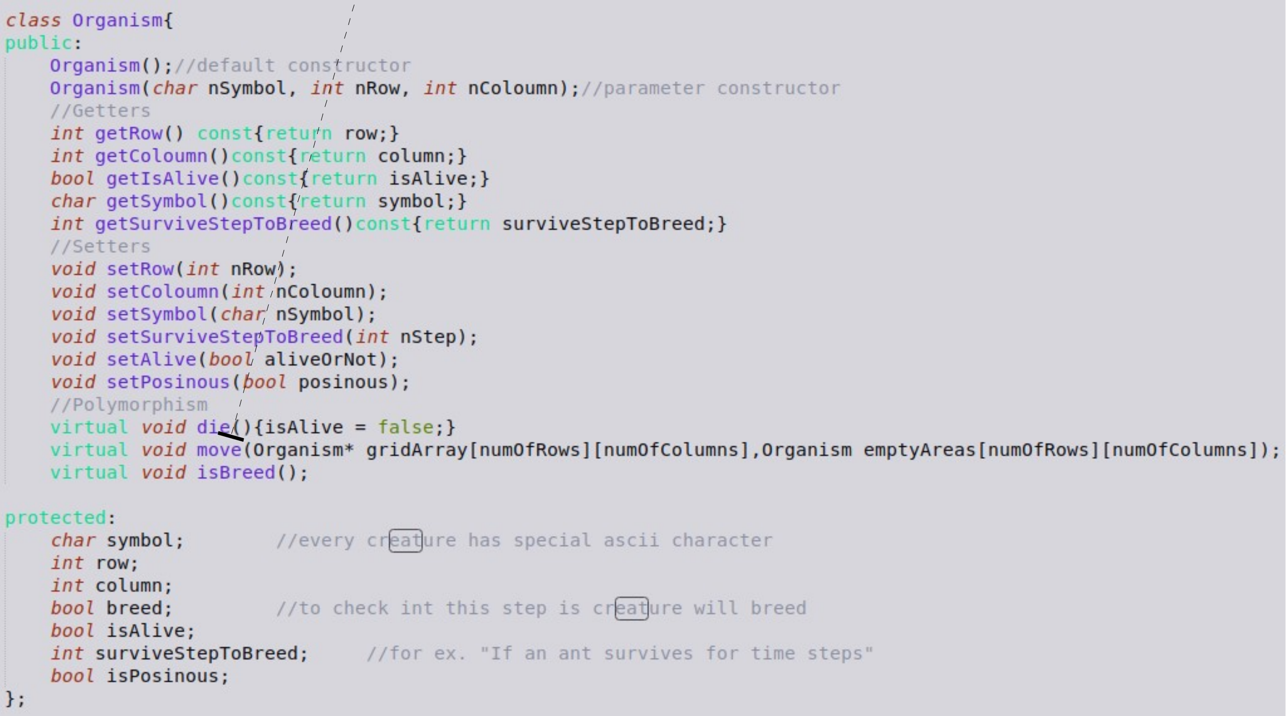
- Write a program to implement this simulation and draw the world using ASCII characters of “o” for an ant, “X” for a doodlebug and “c” for poisonous ant.



```
o o o o - o o o o o o o o o o o o o o o
o o o x o o o o o o o o o o o o o o o o
o o x o o o o o o o o o o o o o o o o o
o x o - - o - x o o o o o - o o o o o o
- o o o - o o o x o o o o o o o - o o o
o o o o o - o o o o o o o o - - o o o o
o - o o o o o o o o o o o o x x o o o o
o x x - o o o o o x - o o o o o o o o o
o o o o o o - o o o o o - o o o o - x -
o o o x - o o o o o o o - o o o o o o o
o o o - o x o o o o o x - o o x x o o o
o o o o o - x o o o o o o x o o o o o o
o o o o - - o o x x o - x o o o - o o o
x - x o x o o o - o - o o o o o o o o o
- o o x o - o o o - o o o - o o o - x -
- o o o o - o o o o o o o o o o o o o -
o - - x o x o o o o x o o o o o - o x o
- o o o o o o o o o o o o o o - o o x o
o o - o o o o o o - o o x o o o o o o o
- - o o o o o o o o o o o o o o o o o o
Step--->220
There are 320 ants, 32 doodlebugs, 0 PoisonousAnt
Press enter to initiate time step: 
```

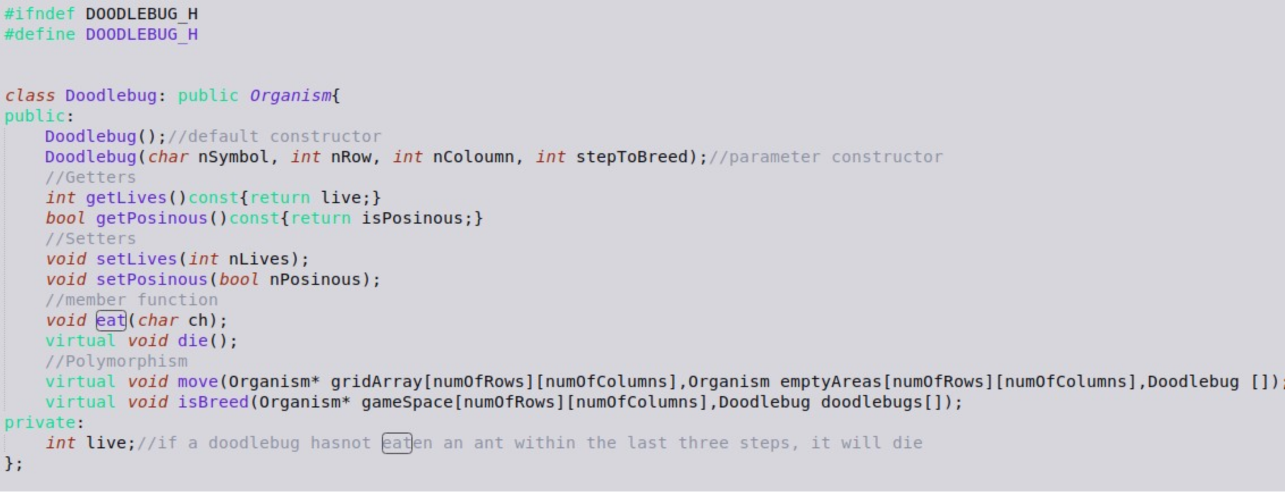
A screen shot when code running

- *Create a class named Organism that encapsulates basic data common to both ants, and doodlebugs. This class should have a virtual function named move that is defined in the derived classes of Ant and Doodlebug.*



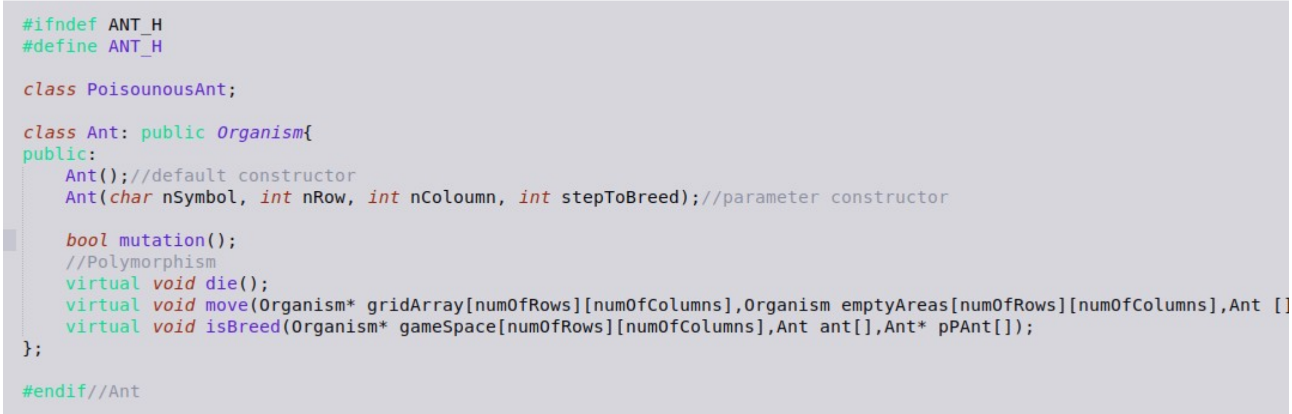
```
class Organism{
public:
    Organism();//default constructor
    Organism(char nSymbol, int nRow, int nColoumn);//parameter constructor
    //Getters
    int getRow() const{return row;}
    int getColoumn()const{return column;}
    bool getIsAlive()const{return isAlive;}
    char getSymbol()const{return symbol;}
    int getSurviveStepToBreed()const{return surviveStepToBreed;}
    //Setters
    void setRow(int nRow);
    void setColoumn(int nColoumn);
    void setSymbol(char nSymbol);
    void setSurviveStepToBreed(int nStep);
    void setAlive(bool aliveOrNot);
    void setPosinous(bool posinous);
    //Polymorphism
    virtual void die(){isAlive = false;}
    virtual void move(Organism* gridArray[numOfRows][numOfColumns],Organism emptyAreas[numOfRows][numOfColumns]);
    virtual void isBreed();

protected:
    char symbol;         //every creature has special ascii character
    int row;
    int column;
    bool breed;          //to check int this step is creature will breed
    bool isAlive;
    int surviveStepToBreed;      //for ex. "If an ant survives for time steps"
    bool isPosinous;
};
```

Organism.h



```
#ifndef DOODLEBUG_H
#define DOODLEBUG_H


class Doodlebug: public Organism{
public:
    Doodlebug();//default constructor
    Doodlebug(char nSymbol, int nRow, int nColoumn, int stepToBreed);//parameter constructor
    //Getters
    int getLives()const{return live;}
    bool getPosinous()const{return isPosinous;}
    //Setters
    void setLives(int nLives);
    void setPosinous(bool nPosinous);
    //member function
    void eat(char ch);
    virtual void die();
    //Polymorphism
    virtual void move(Organism* gridArray[numOfRows][numOfColumns],Organism emptyAreas[numOfRows][numOfColumns],Doodlebug []);
    virtual void isBreed(Organism* gameSpace[numOfRows][numOfColumns],Doodlebug doodlebugs[]);
private:
    int live;//if a doodlebug hasnot eaten an ant within the last three steps, it will die
};
```

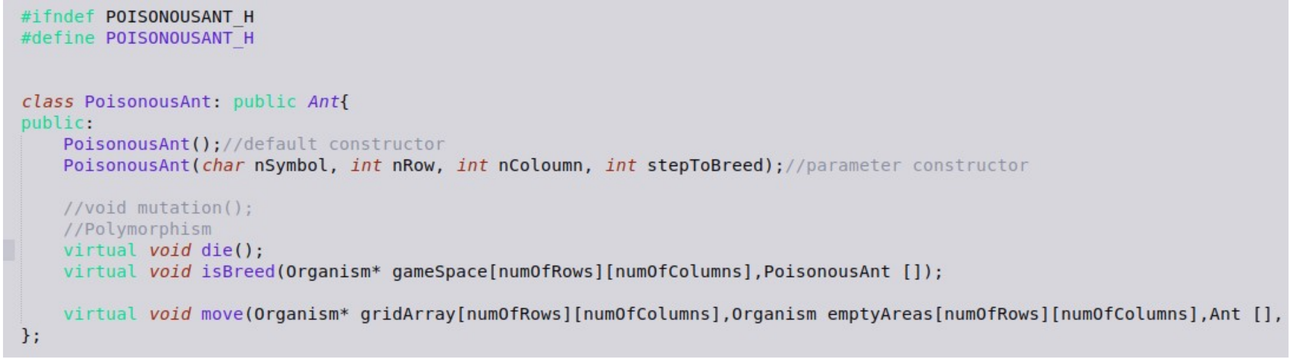
Doodlebug.h



```cpp
#ifndef ANT_H
#define ANT_H

class PoisounousAnt;

class Ant: public Organism{
public:
    Ant();//default constructor
    Ant(char nSymbol, int nRow, int nColoumn, int stepToBreed);//parameter constructor

    bool mutation();
    //Polymorphism
    virtual void die();
    virtual void move(Organism* gridArray[numOfRows][numOfColumns],Organism emptyAreas[numOfRows][numOfColumns],Ant []
    virtual void isBreed(Organism* gameSpace[numOfRows][numOfColumns],Ant ant[],Ant* pPAnt[]);
};

#endif//Ant
```

Ant.h

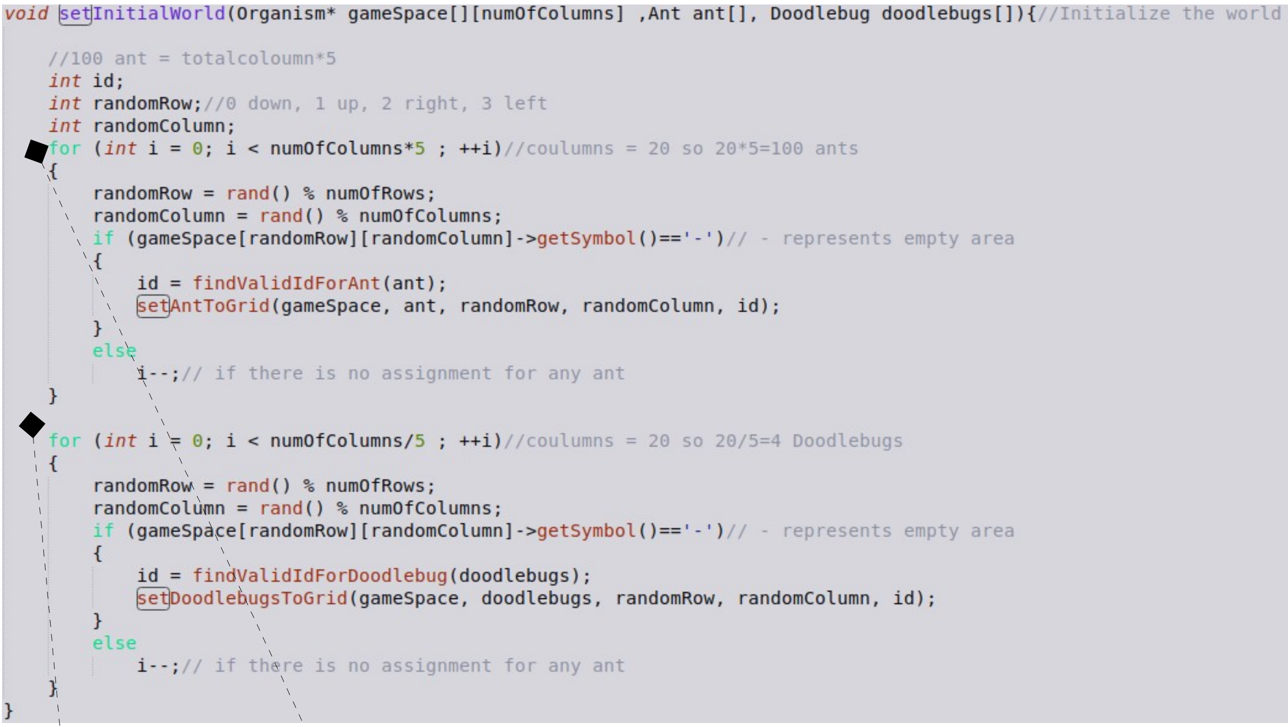
- *Derive poisonous ant class from the ant class.*



```cpp
#ifndef POISONOUSANT_H
#define POISONOUSANT_H


class PoisonousAnt: public Ant{
public:
    PoisonousAnt();//default constructor
    PoisonousAnt(char nSymbol, int nRow, int nColoumn, int stepToBreed);//parameter constructor

    //void mutation();
    //Polymorphism
    virtual void die();
    virtual void isBreed(Organism* gameSpace[numOfRows][numOfColumns],PoisonousAnt []);

    virtual void move(Organism* gridArray[numOfRows][numOfColumns],Organism emptyAreas[numOfRows][numOfColumns],Ant [],
};
```

PoisonousAnt.h

- *Initialize the world with 5 doodlebugs and 100 ants. No poisonous ants initially. After each time step prompt the user to press enter to move to the next time step. You should see a cyclical pattern between the population of predators and prey, although random perturbations may lead to the elimination of one or all species.*



```
void setInitialWorld(Organism* gameSpace[][numOfColumns] ,Ant ant[], Doodlebug doodlebugs[]){//Initialize the world

    //100 ant = totalcoloumn*5
    int id;
    int randomRow;//0 down, 1 up, 2 right, 3 left
    int randomColumn;
    for (int i = 0; i < numOfColumns*5 ; ++i)//coulumns = 20 so 20*5=100 ants
    {
        randomRow = rand() % numOfRows;
        randomColumn = rand() % numOfColumns;
        if (gameSpace[randomRow][randomColumn]->getSymbol()=='-')// - represents empty area
        {
            id = findValidIdForAnt(ant);
            setAntToGrid(gameSpace, ant, randomRow, randomColumn, id);
        }
        else
            i--;// if there is no assignment for any ant
    }

    for (int i = 0; i < numOfColumns/5 ; ++i)//coulumns = 20 so 20/5=4 Doodlebugs
    {
        randomRow = rand() % numOfRows;
        randomColumn = rand() % numOfColumns;
        if (gameSpace[randomRow][randomColumn]->getSymbol()=='-')// - represents empty area
        {
            id = findValidIdForDoodlebug(doodlebugs);
            setDoodlebugsToGrid(gameSpace, doodlebugs, randomRow, randomColumn, id);
        }
        else
            i--;// if there is no assignment for any ant
    }
}
```

SetInitialWorld() sets the initial world in project size is adjustable so ant number and doodlebug number can be changeable according to size of grid so the ant number is (numOfColumns*5) = 100 and doodlebug number is (numOfColumns/4) = 5