



**DEPARTAMENTO
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

RTPI - Flores vs Vampiros

Algoritmos y Estructuras de Datos I

Grupo: 07 - MOLOTOV

Integrante	LU	Correo electrónico
Bukovits, Nicolás Axel	546/14	nicobuk@gmail.com
Chizzoli, Lucas	782/14	chizzoli.lucas13@gmail.com
Frachtenberg Goldsmit, Kevin	247/14	kevinfra94@gmail.com
Garrett, Philip	318/14	garrett.phg@gmail.com



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

1. Flor.h

```
#pragma once
#include "Types.h"
#include <vector>
#include <iostream>
#include <string>

class Flor
{
private:
    Vida _vida;
    std::vector<Habilidad> _habilidades;
    int _cuantoPega;

public:
    Flor();
    Flor(Vida v, int cP, std::vector<Habilidad> hs);
    Vida vidaF();
    int cuantoPegaF();
    std::vector<Habilidad>& habilidadesF();

    void Mostrar(std::ostream& os);
    void Guardar(std::ostream& os);
    void Cargar(std::istream& is);
};
```

2. Flor.cpp

```
#include "Flor.h"

using namespace std;

string tipoHabilidad(Habilidad h){
    string poder;
    if(h==Generar){
        poder="Generar";
    }else if(h == Atacar){
        poder="Atacar";
    }else{
        poder="Explotar";
    }
    return poder;
}

Flor::Flor(){
    this->_vida = 100;
    this->_cuantoPega = 0;
}

Flor::Flor(Vida v, int cP, vector<Habilidad> hs){
    this->_habilidades = hs;
    this->_vida = v;
    this->_cuantoPega = cP;
}
```

```

Vida Flor::vidaF() {
    return this->_vida;
}

int Flor::cuantoPegaF() {
    return this->_cuantoPega;
}

vector<Habilidad>& Flor::habilidadesF() {
    return this->_habilidades;
}

void Flor::Mostrar(ostream& os) {
    os << "Flor_{";
    os << "Vida:_" << this->_vida;
    os << "_Pega:_" << this->_cuantoPega;
    os << "_Habilidades:_" ;
    int i = 0;
    int l = this->_habilidades.size();
    while(i < l) {
        os << tipoHabilidad(this->_habilidades[i]) << "_";
        i++;
    }
    os << "}" << endl;
}

void Flor::Guardar(ostream& os) {
    os << "{_F_" << this->_vida << "_" << this->_cuantoPega << "_[_";
    int i = 0;
    int l = this->_habilidades.size();
    while(i < l) {
        os << tipoHabilidad(this->_habilidades[i]) << "_";
        i++;
    }
    os << "]_}";
}

void Flor::Cargar(istream& is) {
    string flor;
    getline(is, flor, 'F');
    getline(is, flor, '_');
    string vidaC;
    getline(is, vidaC, '_');
    this->_vida = atoi(vidaC.c_str());
    string cuantoPegaC;
    getline(is, cuantoPegaC, '_');
    string basura;
    getline(is, basura, '[');
    this->_cuantoPega = atoi(cuantoPegaC.c_str());
    getline(is, flor, '_');
    string habF;
    while(habF.back() != ']') {
        getline(is, habF, '_');
        if(habF != "]") {
            if(habF == "Atacar") {
                this->_habilidades.push_back(Atacar);
            }
        }
    }
}

```

```

        if(habF == "Explotar"){
            this->_habilidades.push_back(Explotar);
        }
        if(habF == "Generar"){
            this->_habilidades.push_back(Generar);
        }
    }
}

```

3. Vampiro.h

```

#pragma once
#include "Types.h"
#include <vector>
#include <iostream>
#include <string>

class Vampiro
{
    Vida _vida;
    int _cuantoPega;
    ClaseVampiro _clase;

public:
    Vampiro();
    Vampiro(ClaseVampiro cv, Vida v, int cP);
    Vida vidaV();
    ClaseVampiro claseV();
    int cuantoPegaV();

    void Mostrar(std::ostream& os);
    void Guardar(std::ostream& os);
    void Cargar(std::istream& is);
};

```

4. Vampiro.cpp

```

#include "Vampiro.h"

using namespace std;

string TipoVampiro(ClaseVampiro clase)
{
    if(clase == Caminante){
        return "Caminante";
    }else if(clase == Desviado){
        return "Desviado";
    }
}

Vampiro::Vampiro(){
    this->_vida = 1;
    this->_cuantoPega = 1;
}

```

```

        this->_clase = Caminante;
    }

Vampiro::Vampiro(ClasVampiro cv, Vida v, int cP){
    this->_clase = cv;
    this->_vida = v;
    this->_cuantoPega = cP;
}

Vida Vampiro::vidaV()
{
    return this->_vida;
}

ClasVampiro Vampiro::claseV()
{
    return this->_clase;
}

int Vampiro::cuantoPegaV()
{
    return this->_cuantoPega;
}

void Vampiro::Mostrar(ostream& os)
{
    os << "Vampiro_{";
    os << "Vida:_ " << this->_vida;
    os << "_ ,Pega:_ " << this->_cuantoPega;
    os << "_ ,Clase:_ " << TipoVampiro(this->_clase);
    os << "}" << endl;
}

void Vampiro::Guardar(ostream& os)
{
    os << "{_V_" << TipoVampiro(this->_clase) << "_ " << this->_vida << "_ "
        << this->_cuantoPega << "_}";
}

void Vampiro::Cargar(istream& is)
{
    string vampiro;
    getline(is, vampiro, 'V');
    getline(is, vampiro, '_');
    string tipoV;
    getline(is, tipoV, '_');
    if(tipoV == "Caminante"){
        this->_clase = Caminante;
    }else if(tipoV == "Desviado"){
        this->_clase = Desviado;
    }
    string vidaC;
    getline(is, vidaC, '_');
    this->_vida = atoi(vidaC.c_str());
    string cuantoPegaC;
    getline(is, cuantoPegaC, '_');
    this->_cuantoPega = atoi(cuantoPegaC.c_str());
}

```

5. Nivel.h

```
#pragma once
#include <tuple>
#include <vector>
#include <iostream>
#include <string>

#include "Flor.h"
#include "Vampiro.h"

struct FlorEnJuego
{
    FlorEnJuego(Flor f, Posicion p, Vida v)
    {
        flor = f;
        pos = p;
        vida = v;
    }

    Flor flor;
    Posicion pos;
    Vida vida;
};

struct VampiroEnJuego
{
    VampiroEnJuego(Vampiro v, Posicion p, Vida vd)
    {
        vampiro = v;
        pos = p;
        vida = vd;
    }

    Vampiro vampiro;
    Posicion pos;
    Vida vida;
};

struct VampiroEnEspera
{
    Vampiro vampiro;
    int fila;
    int turno;

    VampiroEnEspera(Vampiro v, int f, int t)
    {
        vampiro = v;
        fila = f;
        turno = t;
    }
};

class Nivel
{
private:
    int _ancho;
    int _alto;
```

```

    int _turno;
    int _soles;

    std::vector<FlorEnJuego> _flores;
    std::vector<VampiroEnJuego> _vampiros;
    std::vector<VampiroEnEspera> _spawning;

public:
    Nivel();
    Nivel(int ancho, int alto, int soles, std::vector<VampiroEnEspera>
        & spawninglist);
    int anchoN();
    int altoN();
    int turnoN();
    int solesN();
    std::vector<FlorEnJuego>& floresN();
    std::vector<VampiroEnJuego>& vampirosN();
    std::vector<VampiroEnEspera>& spawningN();
    void agregarFlor(Flor f, Posicion p);
    void pasarTurno();
    bool terminado();
    bool obsesivoCompulsivo();
    void comprarSoles(int n);

    void Mostrar(std::ostream& os);
    void Guardar(std::ostream& os);
    void Cargar(std::istream& is);

};

```

6. Nivel.cpp

```

#include "Nivel.h"
#include <cmath>

using namespace std;

int danoFlor(FlorEnJuego flor, vector<VampiroEnJuego> vampiros){
    int cp = 0;
    int v = 0;
    int l = vampiros.size();
    while(v < l){
        if(vampiros[v].pos.x == flor.pos.x && vampiros[v].pos.y == flor.pos.y)
        {
            cp = cp + vampiros[v].vampiro.cuantoPegaV();
        }
        v++;
    }
    return cp;
}

bool hayVampiroEnPos(FlorEnJuego flor, vector<VampiroEnJuego> vampiros){
    int lv = vampiros.size();
    int v = 0;
    while(v < lv && !(vampiros[v].pos.x == flor.pos.x && vampiros[v].pos.y
        == flor.pos.y)){

```

```

        v++;
    }
    return v < lv;
}

bool tieneHabilidad(Habilidad h, FlorEnJuego flor){
    int v = 0;
    int lh = flor.flor.habilidadesF().size();
    while(v < lh && h != flor.flor.habilidadesF()[v]){
        v++;
    }
    return v < lh;
}

bool florExplota(FlorEnJuego flor, vector<VampiroEnJuego> vampiros){
    return tieneHabilidad(Explotar, flor) && hayVampiroEnPos(flora, vampiros);
}

vector<FlorEnJuego> floresVivas(vector<FlorEnJuego> flores, vector<
    VampiroEnJuego> vampiros){
    vector<FlorEnJuego> nuevaListaFlores;
    int l = flores.size();
    int v = 0;
    while(v < l){
        if(!florExplota(flores[v], vampiros) && flores[v].vida - danoFlor(
            flores[v], vampiros) > 0){
            nuevaListaFlores.push_back(FlorEnJuego(flores[v].flor, flores[v].
                pos, flores[v].vida - danoFlor(flores[v], vampiros)));
        }
        v++;
    }
    return nuevaListaFlores;
}

bool enMira(FlorEnJuego flor, VampiroEnJuego vamp){
    return ((vamp.pos.y == flor.pos.y) && (vamp.pos.x >= flor.pos.x));
}

bool noIntercepta(VampiroEnJuego vamp, FlorEnJuego flor, vector<
    VampiroEnJuego> vampiros){
    int v = 0;
    int largoVampiros = vampiros.size();
    while(v < largoVampiros && !(enMira(flora, vampiros[v]) && (vampiros[v].
        pos.x < vamp.pos.x))){
        v++;
    }
    return v == largoVampiros;
}

int danoV(VampiroEnJuego vamp, vector<FlorEnJuego> flores, vector<
    VampiroEnJuego> vampiros){
    int n = 0;
    int largoFlores = flores.size();
    int golpeFlores = 0;
    while(n < largoFlores){
        if(enMira(flores[n], vamp) && noIntercepta(vamp, flores[n], vampiros)
        ){

```



```

        golpeFlores = golpeFlores + flores[n].flor.cuantoPegaF();
    }
    n++;
}
return golpeFlores;
}

bool hayFlorSobreviviente(Posicion p, vector<FlorEnJuego> flores, vector<
    VampiroEnJuego> vampiros){
    int x = 0;
    int largoFlores = flores.size();
    while(x < largoFlores && !(flores[x].pos.x == p.x && flores[x].pos.y ==
        p.y && flores[x].vida - danoFlor(flores[x], vampiros) > 0)){
        x++;
    }
    return x < largoFlores;
}

bool hayFlorSobrevivienteExplota(Posicion p, vector<FlorEnJuego> flores,
    vector<VampiroEnJuego> vampiros){
    int n = 0;
    int largoFlores = flores.size();
    while(n < largoFlores && !(flores[n].pos.x == p.x && flores[n].pos.y ==
        p.y && tieneHabilidad(Explotar, flores[n]))){
        n++;
    }
    return n < largoFlores;
}

void avanzaV(Posicion &p, VampiroEnJuego vamp){
    if(vamp.vampiro.claseV() == Desviado){
        if(p.y > 1){
            p.y = p.y - 1;
        }
    }
    p.x = p.x - 1;
}

Posicion seMueve(VampiroEnJuego vamp, vector<FlorEnJuego> flores, vector<
    VampiroEnJuego> vampiros){
    Posicion p(vamp.pos.x, vamp.pos.y);
    if(hayFlorSobrevivienteExplota(p, flores, vampiros)){
        p.x++;
    } else if(!hayFlorSobreviviente(p, flores, vampiros)){
        avanzaV(p, vamp);
    }
    return p;
}

vector<VampiroEnJuego> vampirosVivos(vector<FlorEnJuego> preFlores,
    vector<VampiroEnJuego> preVampiros){
    int i = 0;
    int largoVampiros = preVampiros.size();
    std::vector<VampiroEnJuego> vampiros;
    while(i < largoVampiros){
        Vida vidaRestanteV = preVampiros[i].vida - danoV(preVampiros[i],
            preFlores, preVampiros);
        if(vidaRestanteV > 0){

```

```

        vampiros.push_back(VampiroEnJuego(preVampiros[i].vampiro, seMueve(
            preVampiros[i], preFlores, preVampiros), vidaRestanteV));
    }
    i++;
}
return vampiros;
}

int solesGenerados(vector<FlorEnJuego> preFlores, int preSoles){
    int s = preSoles+1;
    int v = 0;
    int lF = preFlores.size();
    while(v < lF){
        if(tieneHabilidad(Generar, preFlores[v])){
            s++;
        }
        v++;
    }
    return s;
}

void actualizarSpawning(vector<VampiroEnEspera>& spaw, int turno){
    vector<VampiroEnEspera> newSpaw;
    int v = 0;
    int largoSpaw = spaw.size();
    while(v < largoSpaw){
        if(spaw[v].turno > turno){
            newSpaw.push_back(spaw[v]);
        }
        v++;
    }
    spaw = newSpaw;
}

bool vampiroEnColumnaCero(vector<VampiroEnJuego> vampiros){
    int v = 0;
    int largoVamps = vampiros.size();
    while(v < largoVamps && !(vampiros[v].pos.x == 0)){
        v++;
    }
    return v < largoVamps;
}

bool floresDesordenadas(vector<FlorEnJuego> floresOrdenadas){
    bool b;
    int largoF = floresOrdenadas.size();
    int v = 1;
    while(v < largoF){
        if(floresOrdenadas[v-1].pos.y <= floresOrdenadas[v].pos.y){
            if((floresOrdenadas[v-1].pos.y == floresOrdenadas[v].pos.y) &&
                floresOrdenadas[v-1].pos.x <= floresOrdenadas[v].pos.x){
                b = false;
            }else if(floresOrdenadas[v-1].pos.y < floresOrdenadas[v].pos.y){
                b = false;
            }else{
                b = true;
                v = largoF;
            }
        }
    }
}

```

```

    }
    v++;
}
return b;
}

void swapF(int a, int b, vector<FlorEnJuego>& floresOrdenadas){
    FlorEnJuego florC = floresOrdenadas[a];
    floresOrdenadas[a] = floresOrdenadas[b];
    floresOrdenadas[b] = florC;
}

void ordenarFlores(vector<FlorEnJuego>& floresOrdenadas){
    int b;
    int largoFlores = floresOrdenadas.size();
    while(floresDesordenadas(floresOrdenadas)){
        b = 1;
        while(b < largoFlores){
            if((floresOrdenadas[b-1].pos.y > floresOrdenadas[b].pos.y) || ((
                floresOrdenadas[b-1].pos.y == floresOrdenadas[b].pos.y) && (
                floresOrdenadas[b-1].pos.x > floresOrdenadas[b].pos.x))){
                swapF(b-1, b, floresOrdenadas);
            }
            b++;
        }
    }
}

bool hayPatron(vector<FlorEnJuego> floresOrdenadas){
    int v = 1;
    int largoF = floresOrdenadas.size();
    while(v < largoF && tieneHabilidad(Atacar, floresOrdenadas[v-1]) == !(
        tieneHabilidad(Atacar, floresOrdenadas[v]))){
        v++;
    }
    return largoF <= 1 || v == largoF;
}

Nivel::Nivel(){}

Nivel::Nivel(int ancho, int alto, int soles, vector<VampiroEnEspera>&
    spawninglist){
    this->_ancho = ancho;
    this->_alto = alto;
    this->_soles = soles;
    this->_turno = 0;
    this->_spawning = spawninglist;
}

int Nivel::anchoN(){
    return this->_ancho;
}

int Nivel::altoN(){
    return this->_alto;
}

int Nivel::turnoN(){

```

```

    return this->_turno;
}

int Nivel::solesN() {
    return this->_soles;
}

vector<FlorEnJuego>& Nivel::floresN() {
    return this->_flores;
}

vector<VampiroEnJuego>& Nivel::vampirosN() {
    return this->_vampiros;
}

vector<VampiroEnEspera>& Nivel::spawningN() {
    return this->_spawning;
}

void Nivel::agregarFlor(Flor f, Posicion p) {
    this->_soles = this->_soles - pow(2, f.habilidadesF().size());
    this->_flores.push_back(FlorEnJuego(f, p, f.vidaF()));
}

void Nivel::pasarTurno() {
    this->_turno++;
    vector<FlorEnJuego> preFlores;
    preFlores = this->_flores;
    this->_flores = floresVivas(this->_flores, this->_vampiros);
    this->_vampiros = vampirosVivos(preFlores, this->_vampiros);
    this->_soles = solesGenerados(preFlores, this->_soles);
    int vs=0;
    int largoSpaw = this->_spawning.size();
    while(vs < largoSpaw) {
        if(this->_spawning[vs].turno <= this->_turno) {
            this->_vampiros.push_back(VampiroEnJuego(this->_spawning[vs].
                vampiro, Posicion(this->_ancho, this->_spawning[vs].fila), this
                ->_spawning[vs].vampiro.vidaV()));
        }
        vs++;
    }
    actualizarSpawning(this->_spawning, this->_turno);
}

bool Nivel::terminado() {
    return vampiroEnColumnaCero(this->_vampiros) || (this->_vampiros.empty()
        () && this->_spawning.empty());
}

bool Nivel::obsesivoCompulsivo() {
    vector<FlorEnJuego> floresOrdenadas = this->_flores;
    ordenarFlores(floresOrdenadas);
    return hayPatron(floresOrdenadas);
}

void Nivel::comprarSoles(int n) {
    this->_soles = this->_soles + n;
}

```

```

void Nivel::Mostrar(ostream& os)
{
    os << "Nivel_{";
    os << "Ancho:_ " << this->_ancho;
    os << "_ ,Alto:_ " << this->_alto;
    os << "_ ,Turno:_ " << this->_turno;
    os << "_ ,Soles:_ " << this->_soles << "\n";
    os << "\t_Flores_en_juego:_{" << "\n";
    int i = 0;
    int lFlores = this->_flores.size();
    while (i < lFlores){
        os << "\t\t_{";
        this->_flores[i].flor.Mostrar(os);
        os << "\t\t_Posicion:_ (" << this->_flores[i].pos.x << ", " << this->
            _flores[i].pos.y << ")_";
        os << "Vida:_ " << this->_flores[i].vida << "}" << "\n";
        i++;
    }
    os << "\t_}" << "\n";
    os << "\t_Vampiros_en_juego:_{" << "\n";
    int j = 0;
    int lVampiros = this->_vampiros.size();
    while(j < lVampiros){
        os << "_\t\t_{";
        this->_vampiros[j].vampiro.Mostrar(os);
        os << "\t\t_Posicion:_ (" << this->_vampiros[j].pos.x << ", " << this
            ->_vampiros[j].pos.y << ")_";
        os << "Vida:_ " << this->_vampiros[j].vida << "}" << "\n";
        j++;
    }
    os << "\t_}" << "\n";
    os << "\t_Spawning:_{" << "\n";
    int s = 0;
    int lSpawning = this->_spawning.size();
    while(s < lSpawning){
        os << "_\t\t_{";
        this->_spawning[s].vampiro.Mostrar(os);
        os << "\t\t_Fila:_ " << this->_spawning[s].fila;
        os << "_ ,Turno:_ " << this->_spawning[s].turno << "}" << "\n";
        s++;
    }
    os << "\t_}" << "\n";
    os << "}" << endl;
}

void Nivel::Guardar(ostream& os)
{
    os << "{_N_" << this->_ancho << "_" << this->_alto << "_" << this->
        _turno << "_" << this->_soles << "_[";
    int i = 0;
    int lFlores = this->_flores.size();
    while (i < lFlores){
        os << "_(_";
        this->_flores[i].flor.Guardar(os);
        os << "_(_" << this->_flores[i].pos.x << "_" << this->_flores[i].pos.y
            << "_)" << this->_flores[i].vida << "_)"";
        i++;
    }
}

```

```

    }
    os << "┘┘[";
    int j = 0;
    int lVampiros = this->_vampiros.size();
    while(j < lVampiros){
        os << "┘(";
        this->_vampiros[j].vampiro.Guardar(os);
        os << "┘(" << this->_vampiros[j].pos.x << "┘" << this->_vampiros[j].
            pos.y << "┘)" << this->_vampiros[j].vida << "┘)";
        j++;
    }
    os << "┘┘[";
    int s = 0;
    int lSpawning = this->_spawning.size();
    while(s < lSpawning){
        os << "┘(" << std::endl;
        this->_spawning[s].vampiro.Guardar(os);
        os << "┘" << this->_spawning[s].fila << "┘" << this->_spawning[s]
            ].turno << "┘)";
        s++;
    }
    os << "┘┘}" << endl;
}

void Nivel::Cargar(istream& is){
    string nivel;
    getline(is, nivel, 'N');
    getline(is, nivel, '┘');
    string anchoNivel;
    getline(is, anchoNivel, '┘');
    this->_ancho = atoi(anchoNivel.c_str());
    string altoNivel;
    getline(is, altoNivel, '┘');
    this->_alto = atoi(altoNivel.c_str());
    string turnoNivel;
    getline(is, turnoNivel, '┘');
    this->_turno = atoi(turnoNivel.c_str());
    string solesNivel;
    getline(is, solesNivel, '┘');
    this->_soles = atoi(solesNivel.c_str());
    string basura;
    getline(is, basura, '┘');
    string cambioTipo;
    getline(is, cambioTipo, '┘');
    char ultimoCambio=cambioTipo.back();
    while(ultimoCambio != '┘') {
        Flor cargaflor;
        Posicion t;
        Vida v;
        cargaflor.Cargar(is);
        getline(is, basura, '(');
        getline(is, basura, '┘');
        string posicionFx;
        getline(is, posicionFx, '┘');
        t.x = atoi(posicionFx.c_str());
        string posicionFy;
        getline(is, posicionFy, '┘');
        t.y = atoi(posicionFy.c_str());
    }
}

```

```

        getline(is, basura, '_');
        string vidaFi;
        getline(is, vidaFi, '_');
        v = atoi(vidaFi.c_str());
        getline(is, basura, '_');
        FlorEnJuego fCargar(cargaflor, t, v);
        this -> _flores.push_back(fCargar);
        getline(is, cambioTipo, '_');
        ultimoCambio = cambioTipo.back();
    }
    getline(is, basura, '_');
    getline(is, cambioTipo, '_');
    ultimoCambio = cambioTipo.back();
    while(ultimoCambio != '']') {
        Vampiro cargaVampiro;
        Posicion p;
        Vida v;
        cargaVampiro.Cargar(is);
        getline(is, basura, '(');
        getline(is, basura, '_');
        string posicionVx;
        getline(is, posicionVx, '_');
        p.x = std::atoi(posicionVx.c_str());
        string posicionVy;
        getline(is, posicionVy, '_');
        p.y = std::atoi(posicionVy.c_str());
        getline(is, basura, '_');
        string vidaVj;
        getline(is, vidaVj, '_');
        v = atoi(vidaVj.c_str());
        VampiroEnJuego vCargar(cargaVampiro, p, v);
        this -> _vampiros.push_back(vCargar);
        getline(is, basura, '_');
        getline(is, cambioTipo, '_');
        ultimoCambio = cambioTipo.back();
    }
    getline(is, basura, '_');
    getline(is, cambioTipo, '_');
    ultimoCambio = cambioTipo.back();
    while(ultimoCambio != '']') {
        Vampiro cargarVS;
        int cargarFila;
        int cargarTurnoS;
        cargarVS.Cargar(is);
        getline(is, basura, '_');
        string fila;
        getline(is, fila, '_');
        cargarFila = atoi(fila.c_str());
        getline(is, basura, '_');
        string turnoVs;
        getline(is, turnoVs, '_');
        cargarTurnoS = std::atoi(turnoVs.c_str());
        VampiroEnEspera vSpawC(cargarVS, cargarFila, cargarTurnoS);
        this -> _spawning.push_back(vSpawC);
        getline(is, cambioTipo, '_');
        ultimoCambio = cambioTipo.back();
    }
}

```

7. Juego.h

```
#pragma once
#include <vector>

#include <iostream>
#include <fstream>

#include "Flor.h"
#include "Vampiro.h"
#include "Nivel.h"
#include "Types.h"

class Juego
{
private:
    std::vector<Flor> _flores;
    std::vector<Vampiro> _vampiros;
    std::vector<Nivel> _niveles;
    int _nivelActual;

public:
    Juego();
    Juego(std::vector<Flor>& flores, std::vector<Vampiro>& vamps);
    int nivelActual();
    void pasarNivel();
    std::vector<Flor>& floresJ();
    std::vector<Vampiro>& vampirosJ();
    std::vector<Nivel>& nivelesJ();
    void agregarNivel(Nivel& n, int i);
    void jugarNivel(Nivel& n, int i);
    std::vector<Nivel> estosSaleFacil();
    void altoCheat(int n);
    bool muyDeExactas();

    void Mostrar(std::ostream& os);
    void Guardar(std::ostream& os);
    void Cargar(std::istream& is);
};
```


8. Juego.cpp

```
#include "Juego.h"

using namespace std;

bool vampirosIguales(Vampiro v1, Vampiro v2){
    return (v1.vidaV() == v2.vidaV() && v1.cuantoPegaV() == v2.
        cuantoPegaV() && v1.claseV() == v2.claseV());
}

bool perteneceV(Vampiro v, vector<Vampiro> vamps){
    int l = vamps.size();
    int i = 0;
    while(i<l && !(vampirosIguales(v, vamps[i]))) {
        i++;
    }
    return i < l;
}

int cuenta(Habilidad h, vector<Habilidad> hs){
    int cant = 0;
    int l = hs.size();
    int i = 0;
    while(i<l){
        if(h == hs[i]){
            cant++;
        }
        i++;
    }
    return cant;
}

bool cuentaMismosHabilidades(Flor f1, Flor f2){
    int l = f1.habilidadesF().size();
    int i = 0;
    while(i<l && cuenta(f1.habilidadesF()[i], f1.habilidadesF()) == cuenta(
        f1.habilidadesF()[i], f2.habilidadesF()))
    {
        i++;
    }
    return i==l;
}

bool floresIguales(Flor f1, Flor f2){
    return (f1.habilidadesF().size() == f2.habilidadesF().size() &&
        cuentaMismosHabilidades(f1, f2));
}

bool perteneceF(Flor f, vector<Flor> flores){
    int l = flores.size();
    int i = 0;
    while(i<l && !(floresIguales(f, flores[i]))) {
        i++;
    }
    return i < l;
}
```

```

vector<Vampiro> sinRepetidosV(vector<Vampiro>& vamps){
    vector<Vampiro> _vampsSinRepetir;
    int l = vamps.size();
    int i = 0;
    while(i<l){
        if(!perteneceV(vamps[i], _vampsSinRepetir)){
            _vampsSinRepetir.push_back(vamps[i]);
        }
        i++;
    }
    return _vampsSinRepetir;
}

vector<Flor> sinRepetidosF(vector<Flor>& flores){
    vector<Flor> _floresSinRepetir;
    int l = flores.size();
    int i = 0;
    while(i<l){
        if(!perteneceF(flores[i], _floresSinRepetir)){
            _floresSinRepetir.push_back(flores[i]);
        }
        i++;
    }
    return _floresSinRepetir;
}

vector<int> nivelesGanados(vector<Nivel> niveles){
    vector<int> ganados;
    int i = 0;
    int niv = niveles.size();
    while(i<niv){
        if(niveles[i].vampirosN().size() == 0 && niveles[i].spawningN().
            size() == 0){
            ganados.push_back(i);
        }
        i++;
    }
    return ganados;
}

int maxSoles(vector<Nivel> niveles){
    int max = 0;
    int i = 0;
    int cantNiveles = niveles.size();
    while(i<cantNiveles){
        if(niveles[i].solesN() > max){
            max = niveles[i].solesN();
        }
        i++;
    }
    return max;
}

int maxFloresmaxSoles(vector<Nivel> niveles, int soles){
    int max = 0;

```

```

        int i = 0;
        int cantNiveles = niveles.size();
        while(i<cantNiveles){
            if(niveles[i].solesN() == soles && niveles[i].floresN().size() >
                max){
                max = niveles[i].floresN().size();
            }
            i++;
        }
        return max;
    }

Juego::Juego()
{
}

Juego::Juego(vector<Flor>& flores , vector<Vampiro>& vamps)
{
    this->_niveles = vector<Nivel>();
    this->_nivelActual = 0;
    this->_vampiros = sinRepetidosV(vamps);
    this->_flores = sinRepetidosF(flores);
}

int Juego::nivelActual()
{
    return this->_nivelActual;
}

void Juego::pasarNivel()
{
    this->_nivelActual++;
}

vector<Flor>& Juego::floresJ()
{
    return this->_flores;
}

vector<Vampiro>& Juego::vampirosJ()
{
    return this->_vampiros;
}

vector<Nivel>& Juego::nivelesJ()
{
    return this->_niveles;
}

void Juego::agregarNivel(Nivel& n, int i)
{
    vector<Nivel> _nivelesSiguientes;
    int l = this->_niveles.size();
    int j = i;
    while(j<l){
        _nivelesSiguientes.push_back(this->_niveles[j]);
    }
}

```

```

        j++;
    }
    this->_niveles.resize(i+1);
    this->_niveles[i] = n;
    j = 0;
    l = _nivelesSiguientes.size();
    while(j<l){
        this->_niveles.push_back(_nivelesSiguientes[j]);
        j++;
    }
}

void Juego::jugarNivel(Nivel& n, int i)
{
    this->_niveles[i] = n;
}

vector<Nivel> Juego::estosSaleFacil()
{
    int _maxSoles = maxSoles(this->_niveles);
    int _maxFloresmaxSoles = maxFloresmaxSoles(this->_niveles,
        _maxSoles);
    vector<Nivel> nivelesFaciles;
    int i = 0;
    int cantNiveles = this->_niveles.size();
    while(i<cantNiveles){
        if(this->_niveles[i].solesN() == _maxSoles && this->_niveles[i].
            floresN().size() == _maxFloresmaxSoles){
            nivelesFaciles.push_back(this->_niveles[i]);
        }
        i++;
    }
    return nivelesFaciles;
}

vector<VampiroEnJuego> vampirosMitadVida(vector<VampiroEnJuego>& vampiros
)
{
    vector<VampiroEnJuego> _vampsMitad;
    int i = 0;
    int l = vampiros.size();
    while(i<l){
        if(!((vampiros[i].vida / 2) == 0)){
            vampiros[i].vida = vampiros[i].vida / 2;
            _vampsMitad.push_back(vampiros[i]);
        }
        i++;
    }
    return _vampsMitad;
}

void Juego::altoCheat(int n)
{
    this->_niveles[n].vampirosN() = vampirosMitadVida(this->_niveles[
        n].vampirosN());
}

bool esFibonacciCasoBaseUno(vector<int> niveles){

```

```

        return (niveles.size() == 1 && niveles[0] == 1);
    }

    bool esFibonacciCasoBaseDos(vector<int> niveles){
        return (niveles.size() == 2 && niveles[0] == 1 && niveles[1] == 2);
    }

    bool Juego::muyDeExactas()
    {
        vector<int> ganados = nivelesGanados(this->_niveles);
        int i = 2;
        int l = ganados.size();
        while(i < l && (ganados[i] == ganados[i-1] + ganados[i-2])){
            i++;
        }
        return (i == l || esFibonacciCasoBaseUno(ganados) ||
                esFibonacciCasoBaseDos(ganados));
    }

    void Juego::Mostrar(ostream& os)
    {
        os << "Juego_{" << endl;
        os << "_t_Flores:_" << endl;
        int i = 0;
        int lFlores = this->_flores.size();
        while (i < lFlores){
            os << "_t_";
            this->_flores[i].Mostrar(os);
            i++;
        }
        os << "_t_" << endl;
        os << "_t_Vampiros:_" << endl;
        int j = 0;
        int lVampiros = this->_vampiros.size();
        while(j < lVampiros){
            os << "_t_";
            this->_vampiros[j].Mostrar(os);
            j++;
        }
        os << "_t_" << endl;
        os << "_t_Niveles:_" << endl;
        int n = 0;
        int lNiveles = this->_niveles.size();
        while(n < lNiveles){
            os << "_t_";
            this->_niveles[n].Mostrar(os);
            n++;
        }
        os << "_t_" << endl;
        os << "}" << endl;
    }

    void Juego::Guardar(ostream& os){
        os << "{_J_";
        int i = 0;
        int lFlores = this->_flores.size();
        while (i < lFlores){
            this->_flores[i].Guardar(os);

```

```

        i++;
        os << " " ;
    }
    os << "]" _[" ";
    int j = 0;
    int lVampiros = this->_vampiros.size();
    while(j < lVampiros){
        this->_vampiros[j].Guardar(os);
        j++;
        os << " " ;
    }
    os << "]" _[" ";
    int n = 0;
    int lNiveles = this->_niveles.size();
    while(n < lNiveles){
        this->_niveles[n].Guardar(os);
        n++;
        os << " " ;
    }
    os << "]" _}" << endl;
}

void Juego::Cargar(istream& is)
{
    string juego;
    getline(is, juego, 'J');
    string basura;
    string cambioTipo;
    getline(is, cambioTipo, '_');
    char ultimoCambio=cambioTipo.back();
    while (ultimoCambio != ']' ) {
        Flor cargaF;
        cargaF.Cargar(is);
        this->_flores.push_back(cargaF);
        getline(is, basura, '_');
        getline(is, cambioTipo, '_');
        ultimoCambio = cambioTipo.back();
    }
    getline(is, basura, '_');
    getline(is, cambioTipo, '_');
    ultimoCambio = cambioTipo.back();
    while (ultimoCambio != ']' ) {
        Vampiro cargaF;
        cargaF.Cargar(is);
        this->_vampiros.push_back(cargaF);
        getline(is, basura, '_');
        getline(is, cambioTipo, '_');
        ultimoCambio = cambioTipo.back();
    }
    getline(is, basura, '_');
    getline(is, cambioTipo, '_');
    ultimoCambio = cambioTipo.back();
    while (ultimoCambio != ']' ) {
        Nivel cargaN;
        cargaN.Cargar(is);
        this->_niveles.push_back(cargaN);
        getline(is, basura, '_');
        getline(is, cambioTipo, '_');
    }
}

```

```
        ultimoCambio = cambioTipo.back();  
    }  
    this->_nivelActual = 0;  
}
```

9. Types.h

```
#pragma once
#include <tuple>

enum Habilidad {Generar, Atacar, Explotar};
enum ClaseVampiro {Caminante, Desviado};
typedef int Vida;

struct Posicion
{
    Posicion()
    {
        x = 0;
        y = 0;
    }
    Posicion(int ax, int ay)
    {
        x = ax;
        y = ay;
    }

    int x;
    int y;
};
```

10. main.cpp

```
#include "Juego.h"
#include <iostream>
#include <assert.h>

using namespace std;

bool NivelesIgualesParaTest(Nivel n1, Nivel n2){
    return (n1.anchaN() == n2.anchaN() && n1.altoN() == n2.altoN() && n1.
        turnoN() == n2.turnoN() && n1.solesN() == n2.solesN());
}

int main(){

    cout << "└Trabajo┐practico┐Flores┐vs┐Vampiros" << endl;

    //-----Test Flor-----
    cout << "Test_Flor" << endl;

    //Test constructor flor vacia

    Flor fVacia;

    assert(fVacia.vidaF() == 100);
    assert(fVacia.cuantoPegaF() == 0);
    assert(fVacia.habilidadesF().size() == 0);

    fVacia.Mostrar(cout);

    vector<Habilidad> hab1;
```



```
hab1.push_back(Atacar);

vector<Habilidad> hab2;
hab2.push_back(Atacar);
hab2.push_back(Generar);

vector<Habilidad> hab3;
hab3.push_back(Explotar);
hab3.push_back(Generar);

vector<Habilidad> hab4;
hab4.push_back(Explotar);
hab4.push_back(Atacar);
hab4.push_back(Generar);

Flor f1(33,0,hab3);

Flor f2(33,6,hab2);

Flor f3(50,12,hab1);

Flor f4(25,6,hab4);

Flor f5(33,0,hab3);

Flor f6(33,6,hab2);

Flor f7(33,0,hab3);

Flor f8(33,6,hab2);

assert(f1.vidaF() == 33);
assert(f1.cuantoPegaF() == 0);

cout << "f1_size:" << f1.habilidadesF().size() << endl;
cout << "f2_size:" << f2.habilidadesF().size() << endl;
cout << "f3_size:" << f3.habilidadesF().size() << endl;
cout << "f4_size:" << f4.habilidadesF().size() << endl;

f1.Mostrar(cout);
ofstream guardarFlor("guardarFlor.txt");
f1.Guardar(guardarFlor);
ifstream cargarFlor("guardarFlor.txt");

f4.Mostrar(cout);
ofstream guardarFlor2("guardarFlor2.txt");
f4.Guardar(guardarFlor2);
ifstream cargarFlor2("guardarFlor2.txt");

//-----Fin test Flor-----
//-----Test Vampiro-----

//Test de constructor vacio de vampiro

cout << "Test_vampiro" << endl;
```

```

Vampiro vVacio;

assert(vVacio.vidaV() == 1);
assert(vVacio.cuantoPegaV() == 1);
assert(vVacio.claseV() == Caminante);

vVacio.Mostrar(cout);

Vampiro v1(Desviado,50,50);

Vampiro v2(Caminante,100,100);

Vampiro v3(Desviado,50,50);

Vampiro v4(Caminante,100,100);

Vampiro v5(Caminante,50,500000);

Vampiro v6(Desviado,100,100);

Vampiro v7(Desviado,1,50);

Vampiro v8(Desviado,1,100);

Vampiro v9(Caminante,0,50);

Vampiro v10(Caminante,98,100);

Vampiro v11(Caminante,33,50);

Vampiro v12(Caminante,100,100);

assert(v1.vidaV() == 50);
assert(v1.cuantoPegaV() == 50);
assert(v1.claseV() == Desviado);
assert(v9.vidaV() == 0);
assert(v9.cuantoPegaV() == 50);
assert(v9.claseV() == Caminante);
assert(v5.cuantoPegaV() == 500000);

v1.Mostrar(cout);
ofstream guardarVampiro("guardarVampiro.txt");
v1.Guardar(guardarVampiro);
ifstream cargarVampiro("guardarVampiro.txt");

//-----Fin test vampiro-----
//-----Test Nivel-----

vector<VampiroEnEspera> spawn;
spawn.push_back(VampiroEnEspera(v6,2,2));
spawn.push_back(VampiroEnEspera(v1,3,3));
spawn.push_back(VampiroEnEspera(v2,4,4));
spawn.push_back(VampiroEnEspera(v3,4,5));
spawn.push_back(VampiroEnEspera(v4,5,5));
spawn.push_back(VampiroEnEspera(v5,6,5));

```

```

vector<VampiroEnEspera> spawn2;
spawn2.push_back(VampiroEnEspera(v7,1,0));
spawn2.push_back(VampiroEnEspera(v8,2,0));
spawn2.push_back(VampiroEnEspera(v9,3,0));
spawn2.push_back(VampiroEnEspera(v10,3,0));

Nivel n(10,10,100,spawn);

Nivel n2(5,5,30,spawn);

Nivel n3(2,8,100,spawn);

Nivel n4(5,5,100,spawn);

Nivel n5(2,4,100,spawn);

Nivel n6(5,5,110,spawn);

Nivel n7(12,12,100,spawn2);

n.agregarFlor(f1, Posicion(4,3));
n.agregarFlor(f2, Posicion(5,5));
n.agregarFlor(f5, Posicion(6,6));
n.agregarFlor(f8, Posicion(7,1));

assert(n.obsesivoCompulsivo() == true);

n2.agregarFlor(f1, Posicion(4,15));
n2.agregarFlor(f2, Posicion(3,15));
n2.agregarFlor(f5, Posicion(5,15));
n2.agregarFlor(f8, Posicion(2,15));
assert(n2.obsesivoCompulsivo() == false);

cout << "El_largo_de_spawning_es:" << n.spawningN().size() << endl;
cout << endl;
cout << "Pasar_turno" << endl;

int pases=0;
while(n.terminado() == false){
    n.pasarTurno();
    n.Mostrar(cout);
    cout << endl;
}

string inNivel = "nGuardado.txt";

ofstream guardadon(inNivel);

n.Guardar(guardadon);

n.agregarFlor(f1, Posicion(3,3));

string solo="nGuardado.txt";

ifstream asds(solo);

Nivel nivelCargado;
nivelCargado.Cargar(asds);

```

```

cout << "Nivel_cargado" << endl;

assert (nivelCargado.anchaN() == 10);
assert (nivelCargado.altoN() == 10);
assert (nivelCargado.turnoN() == 12);
assert (nivelCargado.solesN() == 135);
assert (nivelCargado.floresN().size() == 1);
assert (nivelCargado.vampirosN().size() == 6);
assert (nivelCargado.spawningN().size() == 0);

//-----Fin test Nivel-----
//-----Test Juego-----
cout << "Test_Juego" << endl;

vector<Flor> fs;
fs.push_back(f1);
fs.push_back(f1);
fs.push_back(f1);
fs.push_back(f1);
fs.push_back(f1);
fs.push_back(f2);
fs.push_back(f3);

vector<Vampiro> vs;
vs.push_back(v1);
vs.push_back(v1);
vs.push_back(v1);
vs.push_back(v2);
vs.push_back(v5);
vs.push_back(v9);

Juego j(fs, vs);

assert(j.floresJ().size() == 3);
assert(j.vampirosJ().size() == 4);
assert(j.nivelesJ().size() == 0);

j.agregarNivel(n,0);

assert(j.nivelesJ().size() == 1);

j.agregarNivel(n5,1);

assert(j.nivelesJ().size() == 2);

j.agregarNivel(n2,0);

assert(NivelesIgualesParaTest(j.nivelesJ()[0],n2) == true);
assert(NivelesIgualesParaTest(j.nivelesJ()[1],n) == true);
assert(NivelesIgualesParaTest(j.nivelesJ()[2],n5) == true);

j.agregarNivel(n3,1);

assert(NivelesIgualesParaTest(j.nivelesJ()[0],n2) == true);
assert(NivelesIgualesParaTest(j.nivelesJ()[1],n3) == true);
assert(NivelesIgualesParaTest(j.nivelesJ()[2],n) == true);
assert(NivelesIgualesParaTest(j.nivelesJ()[3],n5) == true);

```

```

j.jugarNivel(n6,2);

assert(NivelesIgualesParaTest(j.nivelesJ()[2],n6) == true);

assert(j.estosSaleFacil().size() == 1);
assert(j.estosSaleFacil()[0].solesN() == 110);

n7.pasarTurno();

j.agregarNivel(n7,4);

cout << "Largo_vampiros_de_nivelesJ[4]_es" << j.nivelesJ()[4].vampirosN
    ().size() << endl;
assert(j.nivelesJ()[4].vampirosN().size() == 4);
j.altoCheat(4);
assert(j.nivelesJ()[4].vampirosN().size() == 1);
assert(j.nivelesJ()[4].vampirosN()[0].vida == 49);

vector<VampiroEnEspera> spawnVacio;

Juego j2(fs,vs);

Nivel nVacio(10,10,40,spawnVacio);
Nivel n1Vacio(10,10,40,spawnVacio);
Nivel n2Vacio(10,10,40,spawnVacio);
Nivel n3Vacio(10,10,40,spawnVacio);
Nivel n4Vacio(10,10,40,spawnVacio);

Nivel nLleno(10,10,30,spawn2);
Nivel nLleno1(10,12,40,spawn2);
Nivel nLleno2(10,10,30,spawn2);
Nivel nLleno3(10,10,30,spawn2);

j2.agregarNivel(nVacio,0);
j2.agregarNivel(n1Vacio,1);
j2.agregarNivel(n2Vacio,2);

assert(j2.muyDeExactas() == false);

Juego j3(fs,vs);

j3.agregarNivel(nLleno,0);
j3.agregarNivel(nVacio,1);
j3.agregarNivel(n1Vacio,2);
j3.agregarNivel(n2Vacio,3);
j3.agregarNivel(nLleno1,4);
j3.agregarNivel(n3Vacio,5);
j3.agregarNivel(nLleno2,6);
j3.agregarNivel(nLleno3,7);
j3.agregarNivel(n4Vacio,8);

assert(j3.muyDeExactas() == true);

Juego j4(fs,vs);

```

```
j4.agregarNivel(nLleno,0);
j4.agregarNivel(nVacio,1);

assert(j4.muyDeExactas() == true);

Juego j5(fs,vs);

j5.agregarNivel(nLleno,0);
j5.agregarNivel(nVacio,1);
j5.agregarNivel(n2Vacio,2);

assert(j5.muyDeExactas() == true);

ofstream juegoGuardado("juego.txt");
j.Guardar(juegoGuardado);

cout << "Juego_guardado" << endl;

string juegoPCarga="juego.txt";
ifstream juegoCargado(juegoPCarga);
Juego juegoLoaded;
juegoLoaded.Cargar(juegoCargado);

cout << "Cargo_juego" << endl;

assert(juegoLoaded.floresJ().size() == 3);
assert(juegoLoaded.vampirosJ().size() == 4);
assert(juegoLoaded.nivelesJ().size() == 5);

return 0;
}
```