DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

RTPI

Algoritmos y Estructuras de Datos I
Primer Cuatrimestre de 2015

Grupo 18: Perl Jam

Integrante	LU	Correo electrónico
Joel Esteban Camera	257/14	joel.e.camera@gmail.com
Isakova, Olga	779/14	fallenapart@mail.ru
Martinez, Marcelo	501/82	pcpower99@hotmail.com
Valls, Marcelo	108/14	cmarcelovalls@hotmail.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

Types.h

```
1  #pragma once
2  #include <tuple>
3
4  enum Habilidad {Generar, Atacar, Explotar};
5  enum ClaseVampiro {Caminante, Desviado};
6  typedef int Vida;
7
8  struct Posicion
9  {
10   Posicion()
11     {
12       fila = 0;
13       columna = 0;
14     }
15     Posicion(int x, int y)
16     {
17       fila = x;
18       columna = y;
19     }
20
21     int fila, columna;
22 };
```

Flor.h

```
1  #pragma once
2  #include "Types.h"
3  #include <vector>
4  #include <iostream>
5  #include <string>
6
7  using namespace std;
8
9  class Flor
10 {
11 private:
12
13    Vida _vida;
14    vector<Habilidad> _habilidades;
15    int _cuantoPega;
16    void quitarRepetidosHabilidad(vector<Habilidad>& hs);
17    bool atacarPertenece(vector<Habilidad> hs);
18
19 public:
20    Flor();
21    Flor(Vida v, int cP, vector<Habilidad> hs);
22    Vida vidaF();
23    int cuantoPegaF();
24    vector<Habilidad>& habilidadesF();
25
26    void Mostrar(ostream& os) const;
27    void Guardar(ostream& os) const;
28    void Cargar(istream& is);
29
30 };
```

Flor.cpp

```
1  #include "Flor.h"
2  #include <cassert>
3
4  using namespace std;
5
6
7  Flor::Flor(){}
8
9  Flor::Flor(Vida v, int cP, vector<Habilidad> hs)
10 {
11   quitarRepetidosHabilidad(hs);
12
13   if(atacarPertenece(hs)){
14     this->_cuantoPega = 12 / hs.size();
15   } else {
16     this->_cuantoPega = 0;
17   }
18   this->_vida = 100 / (hs.size() + 1);
19   this->_habilidades = hs;
20 }
21
22 Vida Flor::vidaF()
23 {
24   return this->_vida;
25 }
26
27 int Flor::cuantoPegaF()
28 {
29   return this->_cuantoPega;
30 }
31
32 vector<Habilidad>& Flor::habilidadesF()
33 {
34   return this->_habilidades;
35 }
36
37 void Flor::Mostrar(ostream& os) const
38 {
39   int i = 0;
40
41   os << "INFO. DE LA FLOR:" << endl;
42   os << "Vida de la Flor: " << this->_vida << endl;
43   os << "Cuanto Pega la Flor: " << this->_cuantoPega << endl;
44   os << "Habilidad(es) de la Flor: [ ";
45
46   while(i < this->_habilidades.size()){
47
48     if(this->_habilidades.at(i) == Atacar){
49       os << "Atacar ";
50     } else if(this->_habilidades.at(i) == Generar) {
51       os << "Generar ";
52     } else {
```

```
53          os << "Explotar ";
54        }
55
56        i++;
57      }
58      os << "]" << endl << endl;
59    }
60
61    void Flor::Guardar(ostream& os) const
62    {
63      os << "{ F ";
64
65      string strVida = to_string(this->_vida);
66      os << strVida << " ";
67
68      string strCuantoPega = to_string(this->_cuantoPega);
69      os << strCuantoPega << " [ ";
70
71      int i = 0;
72
73      while (i < this->_habilidades.size()){
74        if(this->_habilidades.at(i) == 0){
75          os << "Generar ";
76        } else if(this->_habilidades.at(i) == 1){
77          os << "Atacar ";
78        } else{
79          os << "Explotar ";
80        }
81        i++;
82      }
83      os << "] }";
84    }
85
86    void Flor::Cargar(istream& is)
87    {
88      string buscaValores;
89      vector<Habilidad> hs;
90
91      //lo unico que me interesa saber es el vector habilidad, con eso creo la
            ↪ flor
92      while(buscaValores != "]"){
93
94        getline(is,buscaValores,' ');
95
96        if(buscaValores == "Atacar"){
97          hs.push_back(Atacar);
98        } else if(buscaValores == "Generar") {
99          hs.push_back(Generar);
100       } else if(buscaValores == "Explotar"){
101         hs.push_back(Explotar);
102       }
103     }
104
105     getline(is,buscaValores,' '); // "}" esto lo pongo para los otros cargar
```

```
106
107   quitarRepetidosHabilidad(hs);
108   if(atacarPertenece(hs)){
109     this->_cuantoPega = 12 / hs.size();
110   } else {
111     this->_cuantoPega = 0;
112   }
113   this->_vida = 100 / (hs.size() + 1);
114   this->_habilidades = hs;
115
116 }
117
118 void Flor::quitarRepetidosHabilidad(vector<Habilidad>& hs){
119   int i = hs.size() - 1;
120   int contadorA = 0;
121   int contadorE = 0;
122   int contadorG = 0;
123
124   while(i >= 0){
125
126     if(hs.at(i) == Generar){
127       contadorG += 1;
128     } else if(hs.at(i) == Atacar){
129       contadorA += 1;
130     } else {
131       contadorE += 1;
132     }
133     hs.pop_back();
134     i--;
135   }
136
137   if(contadorG > 0){
138     hs.push_back(Generar);
139   }
140   if(contadorA > 0){
141     hs.push_back(Atacar);
142   }
143   if(contadorE > 0){
144     hs.push_back(Explotar);
145   }
146 }
147
148 bool Flor::atacarPertenece(vector<Habilidad> hs){
149   int i = 0;
150   bool res = false;
151
152   while(i < hs.size()){
153     if(hs.at(i) == Atacar)
154       res = true;
155     i++;
156   }
157   return res;
158 }
```

Vampiro.h

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

class Vampiro
{
private:

  Vida _vida;
  int _cuantoPega;
  ClaseVampiro _clase;
  bool vidaEnRango(int v);
  void ponerVidaEnRango(int& v);
  bool cuantoPegaEnRango(int cP);
  void ponerCuantoPegaEnRango(int& cP);

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);

};
```

Vampiro.cpp

```
#include "Vampiro.h"
#include <cassert>

using namespace std;

Vampiro::Vampiro()
{
}

Vampiro::Vampiro(ClaseVampiro cv, Vida v, int cP)
{
  // si no esta en rango, pone el valor mas cercano dentro del rango
  // (v<0 --> v = 1 || v>100 --> v == 100) y (cP<=0 --> 1)
  if(!vidaEnRango(v)) ponerVidaEnRango(v);
  if(!cuantoPegaEnRango(cP)) ponerCuantoPegaEnRango(cP);
  this->_vida = v;
  this->_cuantoPega = cP;
  this->_clase = cv;
}

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

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

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

void Vampiro::Mostrar(ostream& os)
{
  os << "INFO. DEL VAMPIRO:" << endl;
  os << "Vida del Vampiro: " << this->_vida << endl;
  os << "Cuanto Pega el Vampiro: " << this->_cuantoPega << endl;
  os << "Clase del Vampiro: ";

  if(this->_clase == Caminante){
    os << "Caminante" << endl;
  } else {
    os << "Desviado" << endl;
  }
  os << endl;
}

void Vampiro::Guardar(ostream& os)
{
```

```
53    os << "{ V ";
54    if(this->_clase == Caminante){
55      os << "Caminante ";
56    } else {
57      os << "Desviado ";
58    }
59
60    string strVida = to_string(this->_vida);
61    os << strVida << " ";
62    string strCuantoPega = to_string(this->_cuantoPega);
63    os << strCuantoPega << " }";
64  }
65
66  void Vampiro::Cargar(istream& is)
67  {
68    string buscaValores;
69    int num = 0;
70
71    getline(is,buscaValores,'V');
72    getline(is,buscaValores,' ');
73    getline(is,buscaValores,' '); // aca veo la clase del vampiro
74
75    if(buscaValores == "Desviado"){
76      this->_clase = Desviado;
77    } else {
78      this->_clase = Caminante;
79    }
80
81    getline(is,buscaValores,' '); // aca veo la vida
82    this->_vida = atoi(buscaValores.c_str());
83    if(!vidaEnRango(this->_vida)) ponerVidaEnRango(this->_vida);
84
85    getline(is,buscaValores,' '); // aca veo cuanto pega
86
87    this->_cuantoPega = atoi(buscaValores.c_str());
88    if(!cuantoPegaEnRango(this->_cuantoPega)) ponerCuantoPegaEnRango(this->
        ↪ _cuantoPega);
89
90    getline(is,buscaValores,' '); // "}" para otros cargar pongo esta linea
91  }
92
93  bool Vampiro::vidaEnRango(int v){
94    return (v >= 0 && v <= 100);
95  }
96
97  void Vampiro::ponerVidaEnRango(int& v){
98    if(v < 0){
99      v = 0;
100   } else {
101     v = 100;
102   }
103 }
104
105 bool Vampiro::cuantoPegaEnRango(int cP){
```

```
106     return (cP > 0);
107 }
108
109 void Vampiro::ponerCuantoPegaEnRango(int& cP){
110     cP = 1;
111 }
```

Nivel.h

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

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

using namespace std;

struct FlorEnJuego
{
  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;
  }

```

```
53 };
54
55 class Nivel
56 {
57 private:
58
59   int _ancho;
60   int _alto;
61   int _turno;
62   int _soles;
63
64   vector<FlorEnJuego> _flores;
65   vector<VampiroEnJuego> _vampiros;
66   vector<VampiroEnEspera> _spawning;
67
68   bool valoresDeEntradaEnRango(int ancho, int alto, int soles, int
       ↪ spawnSize);
69   vector<VampiroEnEspera> ordenarSpawning(vector<VampiroEnEspera> spawning)
       ↪ ;
70   bool tieneTurnoMinimo(VampiroEnEspera v1, vector<VampiroEnEspera>
       ↪ spawning);
71   vector<VampiroEnEspera> eliminarSpawning(vector<VampiroEnEspera> lista,
       ↪ VampiroEnEspera vampiro);
72   bool igualesVampiros(VampiroEnEspera v1, VampiroEnEspera v2);
73   int vampirosEnCasa(vector<VampiroEnJuego> vampiros);
74   int cantidadFloresConHabilidad(Habilidad habilidad);
75   FlorEnJuego daniarFlor(FlorEnJuego flor, vector<VampiroEnJuego> vampiros)
       ↪ ;
76   bool florMuerta(FlorEnJuego flor, vector<VampiroEnJuego> vampiros);
77   bool florExploto(FlorEnJuego flor, vector<VampiroEnJuego> vampiros);
78   bool tieneHabilidad(FlorEnJuego florEnJuego, Habilidad habilidad);
79   bool vampiroEnMismaPosicion(FlorEnJuego florEnJuego, vector<
       ↪ VampiroEnJuego> vampiros);
80   bool vampiroMuerto(VampiroEnJuego vampiro, vector<FlorEnJuego> flores,
       ↪ vector<VampiroEnJuego> vampiros);
81   VampiroEnJuego daniarVampiro(VampiroEnJuego vampiroEnJuego, vector<
       ↪ FlorEnJuego> flores, vector<VampiroEnJuego> vampiros);
82   bool enMira(FlorEnJuego flor, VampiroEnJuego vampiro);
83   bool intercepta(FlorEnJuego flor, VampiroEnJuego vampiro, vector<
       ↪ VampiroEnJuego> vampiros);
84   vector<VampiroEnJuego> nuevosVampiros();
85   bool mismaPosicion(Posicion p1, Posicion p2);
86   bool florAtaca (FlorEnJuego f);
87   vector<VampiroEnJuego> moverVampiros(vector<VampiroEnJuego> vampiros);
88   VampiroEnJuego mover(VampiroEnJuego vampiro);
89   Posicion intentarAvanzar(VampiroEnJuego vampiro);
90   Posicion intentarRetroceder(Posicion posicion, int ancho);
91   Posicion intentarDesvio(VampiroEnJuego vampiro);
92   bool florExplotada(Posicion posicion, vector<FlorEnJuego> flores, vector<
       ↪ VampiroEnJuego> vampiros);
93   bool florSobreviviente(Posicion posicion, vector<FlorEnJuego> flores,
       ↪ vector<VampiroEnJuego> vampiros);
94   vector<VampiroEnJuego> concatenarVampiros(vector<VampiroEnJuego> v1,
       ↪ vector<VampiroEnJuego> v2);
```

```
95    vector<VampiroEnEspera> proximosVampiros(vector<VampiroEnEspera> spawning
          ↪ , int turno);
96    vector<FlorEnJuego> floresDaniadasYSobrevivientes(vector<FlorEnJuego>
          ↪ flores, vector<VampiroEnJuego> vampiros);
97    vector<VampiroEnJuego> vampirosDaniadosYSobrevivientes(vector<FlorEnJuego
          ↪ > flores, vector<VampiroEnJuego> vampiros);
98
99  public:
100
101   Nivel();
102   Nivel(int ancho, int alto, int soles, vector<VampiroEnEspera>&
          ↪ spawninglist);
103   int anchoN();
104   int altoN();
105   int turnoN();
106   int solesN();
107   vector<FlorEnJuego>& floresN();
108   vector<VampiroEnJuego>& vampirosN();
109   vector<VampiroEnEspera>& spawningN();
110   void agregarFlor(Flor f, Posicion p);
111   void pasarTurno();
112   bool terminado();
113   bool obsesivoCompulsivo();
114   void comprarSoles(int n);
115   void Mostrar(ostream& os);
116   void Guardar(ostream& os);
117   void Cargar(istream& is);
118
119 };
```

Nivel.cpp

```cpp
#include "Nivel.h"
#include <cassert>
#include <iostream>
#include <fstream>
#include <cmath>

using namespace std;

Nivel::Nivel()
{
}

Nivel::Nivel(int ancho, int alto, int soles, vector<VampiroEnEspera>&
    ↪ spawninglist)
{
  /* Si alguno de los valores no esta en el rango del invariante o la
   * longitud de spawninglist no es
   * la debida crea un "nivel estandar" con los valores minimos que cumplen
       ↪ .
   */
  if(valoresDeEntradaEnRango(ancho,alto,soles,spawninglist.size())){

    this->_ancho = ancho;
    this->_alto = alto;
    this->_soles = soles;
    this->_spawning = ordenarSpawning(spawninglist);

  } else {

    this->_ancho = 1;
    this->_alto = 1;
    this->_soles = 0;
    Vampiro vamp(Caminante,0,1);
    VampiroEnEspera vampEnEspera(vamp,1,1);
    vector<VampiroEnEspera> spaw;
    spaw.push_back(vampEnEspera);
    this->_spawning = spaw;
  }

  this->_turno = 0;
  this->_vampiros = vector<VampiroEnJuego>();
  this->_flores = vector<FlorEnJuego>();
}

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

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

```
51  }
52
53  int Nivel::turnoN()
54  {
55    return this->_turno;
56  }
57
58  int Nivel::solesN()
59  {
60    return this->_soles;
61  }
62
63  vector<FlorEnJuego>& Nivel::floresN()
64  {
65    return this->_flores;
66  }
67
68  vector<VampiroEnJuego>& Nivel::vampirosN()
69  {
70    return this->_vampiros;
71  }
72
73  vector<VampiroEnEspera>& Nivel::spawningN()
74  {
75    return this->_spawning;
76  }
77
78  void Nivel::comprarSoles(int s)
79  {
80    if(s > 0) this->_soles = this->_soles + s;
81  }
82
83  void Nivel::agregarFlor(Flor f, Posicion p)
84  {
85    int i = 0;
86    int valor = pow(2, (f.habilidadesF().size()));
87    bool hayEspacio = true;
88
89    if(this->_soles >= valor){
90
91      while(i < this->_flores.size()){
92        if(mismaPosicion(this->_flores.at(i).pos, p)){
93          hayEspacio = false;
94        }
95        i++;
96      }
97
98      if(hayEspacio){
99        this->_soles = this->_soles - valor;
100       vector<FlorEnJuego> flores = vector<FlorEnJuego>();
101       FlorEnJuego florEnJuego = FlorEnJuego(f, p, f.vidaF());
102       flores.push_back(florEnJuego);
103       i = 0;
104       while(i < this->_flores.size()){
```

```
105           flores.push_back(this->_flores.at(i));
106           i++;
107         }
108         this->_flores = flores;
109       }
110     }
111 }
112
113 void Nivel::pasarTurno(){
114   vector<FlorEnJuego> flores =
115       floresDaniadasYSobrevivientes(this->_flores, this->_vampiros);
116
117   vector<VampiroEnJuego> vampirosDaniados = vampirosDaniadosYSobrevivientes
      ↪ (this->_flores, this->_vampiros);
118
119   vector<VampiroEnJuego> vampirosMovidos = moverVampiros(vampirosDaniados);
120   vector<VampiroEnJuego> nuevosVampirosDeTurno = nuevosVampiros();
121   vector<VampiroEnEspera> proximosVampirosDeTurno = proximosVampiros(this->
      ↪ _spawning, this->_turno);
122
123   Habilidad habilidad = Generar;
124   this->_soles = this->_soles + cantidadFloresConHabilidad(habilidad) + 1;
125   this->_vampiros = concatenarVampiros(vampirosMovidos,
      ↪ nuevosVampirosDeTurno);
126   this->_flores = flores;
127   this->_spawning = ordenarSpawning(proximosVampirosDeTurno);
128   this->_turno = this->_turno + 1;
129 }
130
131 vector<VampiroEnEspera> Nivel::proximosVampiros(vector<VampiroEnEspera>
    ↪ spawning, int turno){
132
133   vector<VampiroEnEspera> proximosVampirosDeTurno = vector<VampiroEnEspera
      ↪ >();
134   int i = 0;
135   while(i < this->_spawning.size()){
136     if(this->_spawning.at(i).turno > this->_turno + 1){
137       proximosVampirosDeTurno.push_back(this->_spawning.at(i));
138     }
139     i++;
140   }
141   return proximosVampirosDeTurno;
142 }
143
144 vector<FlorEnJuego> Nivel::floresDaniadasYSobrevivientes(vector<FlorEnJuego
    ↪ > flores, vector<VampiroEnJuego> vampiros){
145
146   vector<FlorEnJuego> floresDaniadas = vector<FlorEnJuego>();
147   int i = 0;
148   while(i < this->_flores.size()){
149     FlorEnJuego florEnJuego = this->_flores.at(i);
150     if(!florMuerta(florEnJuego, this->_vampiros)){
151       floresDaniadas.push_back(daniarFlor(florEnJuego, this->_vampiros));
152     }
```

```
153       i++;
154     }
155     return floresDaniadas;
156
157   }
158
159   vector<VampiroEnJuego> Nivel::vampirosDaniadosYSobrevivientes(vector<
        ↪ FlorEnJuego> flores, vector<VampiroEnJuego> vampiros){
160
161     vector<VampiroEnJuego> vampirosDaniados = vector<VampiroEnJuego>();
162     int i = 0;
163     while(i < vampiros.size()){
164       VampiroEnJuego vampiroEnJuego = this->_vampiros.at(i);
165       if(!vampiroMuerto(vampiroEnJuego, flores, vampiros)){
166         vampirosDaniados.push_back(daniarVampiro(vampiroEnJuego, flores,
              ↪ vampiros));
167       }
168       i++;
169     }
170     return vampirosDaniados;
171   }
172
173   vector<VampiroEnJuego> Nivel::concatenarVampiros(vector<VampiroEnJuego> v1,
        ↪  vector<VampiroEnJuego> v2){
174
175     vector<VampiroEnJuego> vFinal;
176     int i = 0;
177     while (i < v1.size()){
178       vFinal.push_back(v1.at(i));
179       i++;
180     }
181     i = 0;
182     while (i < v2.size()){
183       vFinal.push_back(v2.at(i));
184       i++;
185     }
186     return vFinal;
187   }
188
189   vector<VampiroEnJuego> Nivel::moverVampiros(vector<VampiroEnJuego> vampiros
        ↪ ){;
190     vector<VampiroEnJuego> vampirosMovidos = vector<VampiroEnJuego>();
191     int i = 0;
192     while(i < vampiros.size()){
193       vampirosMovidos.push_back(mover(vampiros.at(i)));
194       i++;
195     }
196     return vampirosMovidos;
197   }
198
199   VampiroEnJuego Nivel::mover(VampiroEnJuego vampiro){
200     Vampiro prm = vampiro.vampiro;
201     Posicion sgd;
202     if(florSobreviviente(vampiro.pos, this->_flores, this->_vampiros)){
```

```
203       sgd = vampiro.pos;
204     }else{
205       sgd = intentarAvanzar(vampiro);
206     }
207     Vida trd = vampiro.vida;
208     VampiroEnJuego vampiroEnJuego = VampiroEnJuego(prm, sgd, trd);
209     return vampiroEnJuego;
210   }
211
212   Posicion Nivel::intentarAvanzar(VampiroEnJuego vampiro){
213     Posicion pos;
214     if(florExplotada(vampiro.pos, this->_flores, this->_vampiros)){
215       pos = intentarRetroceder(vampiro.pos, this->_ancho);
216     }else{
217       pos = intentarDesvio(vampiro);
218     }
219     return pos;
220   }
221
222   Posicion Nivel::intentarRetroceder(Posicion posicion, int ancho){
223     Posicion pos;
224     if(posicion.columna < ancho){
225       pos = Posicion(posicion.fila, posicion.columna+1);
226     }else{
227       pos = posicion;
228     }
229     return pos;
230   }
231
232   Posicion Nivel::intentarDesvio(VampiroEnJuego vampiro){
233     Posicion pos;
234     if(vampiro.vampiro.claseV() == Desviado && vampiro.pos.fila > 1){
235       pos = Posicion(vampiro.pos.fila-1, vampiro.pos.columna-1);
236     }else{
237       pos = Posicion(vampiro.pos.fila, (vampiro.pos.columna - 1));
238     }
239     return pos;
240   }
241
242   bool Nivel::florExplotada(Posicion posicion, vector<FlorEnJuego> flores,
243         vector<VampiroEnJuego> vampiros){
244     bool res = false;
245     int i = 0;
246     while(i < flores.size()){
247       if(mismaPosicion(flores.at(i).pos, posicion) && florExploto(flores.at(i
            ↪ ), vampiros)){
248
249         res = true;
250       }
251       i++;
252     }
253     return res;
254   }
255
```

```
256 bool Nivel::florSobreviviente(Posicion posicion, vector<FlorEnJuego> flores
    ↪ , vector<VampiroEnJuego> vampiros){
257
258   bool res = false;
259   int i = 0;
260   while(i < flores.size()){
261     if(mismaPosicion(flores.at(i).pos, posicion) &&
262         !florMuerta(flores.at(i), vampiros)){
263       res = true;
264     }
265     i++;
266   }
267   return res;
268 }
269
270 bool Nivel::terminado()
271 {
272   return (this->_vampiros.empty() && this->_spawning.empty()) ||
      ↪ vampirosEnCasa(this->_vampiros) > 0;
273 }
274
275 bool Nivel::obsesivoCompulsivo()
276 {
277   vector<FlorEnJuego> ord = this->_flores;
278   int i=0, j;
279   FlorEnJuego temp;
280   bool res = true;
281
282   if(this->_flores.size() > 0){
283
284     while (i < ord.size()) {
285       j=i+1;
286
287       while (j < ord.size()){
288         if (ord[j].pos.fila < ord[i].pos.fila ||
289         (ord[j].pos.fila == ord[i].pos.fila &&
290         ord[j].pos.columna < ord[i].pos.columna)){
291           temp = ord[i];
292           ord[i] = ord[j];
293           ord[j] = temp;
294         }
295         j++;
296       }
297       i++;
298     }
299
300       i = 0;
301     while (i < ord.size()-1){
302       if (florAtaca(ord[i])==florAtaca(ord[i+1])){
303           res = false;
304       }
305       i++;
306     }
307
```

```
308   } else {
309     res = false;
310   }
311   return res;
312 }
313 
314 
315 bool Nivel::florAtaca (FlorEnJuego f){
316   int i = 0;
317   bool res = false;
318 
319   while (i < f.flor.habilidadesF().size()) {
320     if (f.flor.habilidadesF()[i] == Atacar)
321       res = true;
322     i++;
323   }
324   return res;
325 }
326 
327 void Nivel::Mostrar(ostream& os)
328 {
329   int i = 0;
330 
331   os << "INFO. DEL NIVEL:" << endl;
332   os << "Ancho del Nivel: " << this->_ancho << endl;
333   os << "Alto del Nivel: " << this ->_alto << endl;
334   os << "Turno del Nivel: " << this->_turno << endl;
335   os << "Soles del Nivel: " << this->_soles << endl << endl;
336 
337   //Muestro las flores del Nivel
338   os << "** Flores del Nivel **" << endl;
339   while(i < this->_flores.size()){
340 
341     os << endl;
342     this->_flores.at(i).flor.Mostrar(os);
343     os << "PosiciÃşn en Nivel:" << endl;
344     os << "Fila: " << this->_flores.at(i).pos.fila << "  //  Columna: " <<
        ↪ this->_flores.at(i).pos.columna << endl;
345     os << "Vida de la flor en el Nivel: " << this->_flores.at(i).vida <<
        ↪ endl;
346     i++;
347   }
348 
349   i = 0;
350   //Muestro los vampiros en juego del Nivel
351   os << endl << "** Vampiros del Nivel **" << endl;
352 
353   while(i < this->_vampiros.size()){
354 
355     os << endl;
356     this->_vampiros.at(i).vampiro.Mostrar(os);
357     os << "PosiciÃşn en Nivel:" << endl;
358     os << "Fila: " << this->_vampiros.at(i).pos.fila << "  //  Columna: "
        ↪ << this->_vampiros.at(i).pos.columna << endl;
```

```
359       os << "Vida del vampiro en el Nivel: " << this->_vampiros.at(i).vida <<
              ↪  endl;
360       i++;
361     }
362
363     i = 0;
364     //Muestro la lista de spawning del Nivel
365     os << endl << "** Lista del Spawning del Nivel **" << endl;
366     while(i < this->_spawning.size()){
367
368       os << endl;
369       this->_spawning.at(i).vampiro.Mostrar(os);
370       os << "Fila en que aparece: " << this->_spawning.at(i).fila << endl;
371       os << "Turno en que aparece: " << this->_spawning.at(i).turno << endl;
372       i++;
373     }
374     cout << endl;
375   }
376
377   void Nivel::Guardar(ostream& os)
378   {
379     os << "{ N ";
380     os << this->_ancho << " ";
381     os << this->_alto << " ";
382     os << this->_turno << " ";
383     os << this->_soles << " [ ";
384
385     if(this->_flores.size() == 0){
386       os << "] ";
387     } else {
388       int i = 0;
389       while(i < this->_flores.size()){
390         os << "( ";
391         this->_flores.at(i).flor.Guardar(os);
392         os << " ( ";
393
394         os << to_string(this->_flores.at(i).pos.fila) << " ";
395         os << to_string(this->_flores.at(i).pos.columna) << " ) ";
396         os << to_string(this->_flores.at(i).vida) << " ) ";
397
398         i++;
399       }
400       os << "] ";
401     }
402
403     os << "[ ";
404     if(this->_vampiros.size() == 0){
405       os << "] ";
406     } else {
407       int j = 0;
408       while(j < this->_vampiros.size()){
409         os << "( ";
410         this->_vampiros.at(j).vampiro.Guardar(os);
411         os << " ( ";
```

```
412         os << to_string(this->_vampiros.at(j).pos.fila) << " ";
413         os << to_string(this->_vampiros.at(j).pos.columna) << " ) ";
414         os << to_string(this->_vampiros.at(j).vida) << " ) ";
415         j++;
416       }
417       os << "] ";
418     }
419
420     os << "[ ";
421     if(this->_spawning.size() == 0){
422       os << "] }";
423     } else {
424       int k = 0;
425       while(k < this->_spawning.size()){
426         os << "( ";
427         this->_spawning.at(k).vampiro.Guardar(os);
428         os << " ";
429         os << this->_spawning.at(k).fila << " ";
430         os << this->_spawning.at(k).turno << " ";
431         k++;
432         os << ") ";
433       }
434       os << "] }";
435     }
436 }
437
438 void Nivel::Cargar(istream& is)
439 {
440     string buscaValores;
441     int i,fila,columna,turno;
442     Vida vidaFlor,vidaVampiro;
443     ClaseVampiro claseVamp;
444
445     //cargo ancho, alto, turno y soles
446     getline(is,buscaValores,'N'); // 'N'
447     getline(is,buscaValores,' ');
448     getline(is,buscaValores,' '); // ancho
449     this->_ancho = atoi(buscaValores.c_str());
450     getline(is,buscaValores,' '); // alto
451     this->_alto = atoi(buscaValores.c_str());
452     getline(is,buscaValores,' '); // turno
453     this->_turno = atoi(buscaValores.c_str());
454     getline(is,buscaValores,' '); // soles
455     this->_soles = atoi(buscaValores.c_str());
456
457     getline(is,buscaValores,' '); // '[' comienzo de la lista de flores del
        ↪ nivel
458
459     while(buscaValores != "]"){
460       getline(is,buscaValores,' '); // '('
461
462       if(buscaValores == "]"){ // lista vacia
463
464         vector<FlorEnJuego> floresEnJuegoVacio;
```

```
465         this->_flores = floresEnJuegoVacio;
466         break;
467       }
468
469       Flor flor;
470       flor.Cargar(is);
471
472       getline(is,buscaValores,' '); // '(' empieza posicion
473       getline(is,buscaValores,' '); // fila
474       fila = atoi(buscaValores.c_str());
475       getline(is,buscaValores,' '); // columna
476       columna = atoi(buscaValores.c_str());
477       Posicion pos(fila,columna);
478
479       getline(is,buscaValores,' '); // ')'
480       getline(is,buscaValores,' '); // vida en juego
481       vidaFlor = atoi(buscaValores.c_str());
482
483       FlorEnJuego florEJ(flor,pos,vidaFlor);
484       this->_flores.push_back(florEJ);
485
486       getline(is,buscaValores,' '); // ')'
487       getline(is,buscaValores,' '); // '(' || ']'
488     } // fin de la carga de flores en el nivel
489
490     getline(is,buscaValores,' '); // '[' comienzo de la lista de vampiros del
        ↪  nivel
491
492     while(buscaValores != "]"){
493       getline(is,buscaValores,' '); // '('
494
495       if(buscaValores == "]") { //lista vacia
496
497         vector<VampiroEnJuego> vampirosEnJuegoVacios;
498         this->_vampiros = vampirosEnJuegoVacios;
499         break;
500       }
501
502       Vampiro vamp;
503       vamp.Cargar(is);
504
505       getline(is,buscaValores,' '); // '(' empieza posicion
506       getline(is,buscaValores,' '); // fila
507       fila = atoi(buscaValores.c_str());
508       getline(is,buscaValores,' '); // columna
509       columna = atoi(buscaValores.c_str());
510       Posicion pos(fila,columna);
511
512       getline(is,buscaValores,' '); // ')'
513       getline(is,buscaValores,' '); // vida del vampiro
514       vidaVampiro = atoi(buscaValores.c_str());
515
516       VampiroEnJuego vampiroEJ(vamp,pos,vidaVampiro);
517       this->_vampiros.push_back(vampiroEJ);
```

```
518
519       getline(is,buscaValores,' '); // ')'
520       getline(is,buscaValores,' '); // '(' || ']'
521     } // fin de la carga de vampiros del nivel
522
523     getline(is,buscaValores,' '); // '[' comienzo de la lista de spawning
524
525     while(buscaValores != "]"){
526       getline(is,buscaValores,' '); // '('
527
528       if(buscaValores == "]"){ // lista vacia
529
530         vector<VampiroEnEspera> spawningVacio;
531         this->_spawning = spawningVacio;
532         break;
533       }
534
535       Vampiro vamp;
536       vamp.Cargar(is);
537
538       getline(is,buscaValores,' '); // fila
539       fila = atoi(buscaValores.c_str());
540       getline(is,buscaValores,' '); // turno
541       turno = atoi(buscaValores.c_str());
542
543       VampiroEnEspera vEnEspera(vamp,fila,turno);
544       this->_spawning.push_back(vEnEspera);
545
546       getline(is,buscaValores,' '); // ')'
547       getline(is,buscaValores,' '); // '(' || ']'
548     } // fin de la carga de spawning
549
550     getline(is,buscaValores,' '); // '}' recorro hasta el final para juego.
        ↪ Cargar()
551 }
552
553 vector<VampiroEnEspera> Nivel::ordenarSpawning(vector<VampiroEnEspera>
      ↪ spawning){
554     vector<VampiroEnEspera> spawningFinal = vector<VampiroEnEspera>();
555     vector<VampiroEnEspera> spawningAuxiliar = spawning;
556     int i = 0;
557     while(i < spawning.size()){
558       int j = 0;
559       while(j < spawning.size()){
560         VampiroEnEspera spawningSeleccionado = spawningAuxiliar.at(j);
561         if(tieneTurnoMinimo(spawningSeleccionado, spawningAuxiliar)){
562           spawningFinal.push_back(spawningSeleccionado);
563           spawningAuxiliar = eliminarSpawning(spawningAuxiliar,
                ↪ spawningSeleccionado);
564           break;
565         }
566         j++;
567       }
568       i++;
```

```
569     }
570     return spawningFinal;
571   }
572
573   bool Nivel::tieneTurnoMinimo(VampiroEnEspera v1, vector<VampiroEnEspera>
        ↪ spawning){
574     bool result = true;
575     int i = 0;
576     while(i < spawning.size()){
577       if(spawning.at(i).turno < v1.turno || (spawning.at(i).turno == v1.turno
            ↪ && spawning.at(i).fila < v1.fila)){
578
579         result = false;
580         break;
581       }
582       i++;
583     }
584     return result;
585   }
586
587   vector<VampiroEnEspera> Nivel::eliminarSpawning(vector<VampiroEnEspera>
        ↪ lista, VampiroEnEspera vampiro){
588
589     vector<VampiroEnEspera> listaReturn = vector<VampiroEnEspera>();
590     int i = 0;
591     while(i < lista.size()){
592       if(!igualesVampiros(lista.at(i), vampiro)){
593         listaReturn.push_back(lista.at(i));
594       }
595       i++;
596     }
597     return listaReturn;
598   }
599
600   bool Nivel::igualesVampiros(VampiroEnEspera v1, VampiroEnEspera v2){
601     bool res = false;
602     if(v1.vampiro.claseV() == v2.vampiro.claseV() &&
603         v1.fila == v2.fila &&
604         v1.turno == v2.turno){
605       res = true;
606     }
607     return res;
608   }
609
610   int Nivel::cantidadFloresConHabilidad(Habilidad habilidad){
611     vector<FlorEnJuego> floresConHabilidad = vector<FlorEnJuego>();
612     int i = 0;
613     while(i < this->_flores.size()){
614       FlorEnJuego florSeleccionada = this->_flores.at(i);
615       vector<Habilidad> habilidadesFlorSeleccionada =
616         florSeleccionada.flor.habilidadesF();
617
618       int j = 0;
619       while(j < habilidadesFlorSeleccionada.size()){
```

```
620         if(habilidadesFlorSeleccionada.at(j) == habilidad){
621           floresConHabilidad.push_back(florSeleccionada);
622           break;
623         }
624         j++;
625       }
626       i++;
627     }
628     return floresConHabilidad.size();
629   }
630
631   int Nivel::vampirosEnCasa(vector<VampiroEnJuego> vampiros){
632     int result = 0;
633     int i = 0;
634     while(i < vampiros.size()){
635       VampiroEnJuego vampiro = vampiros.at(i);
636       if(vampiro.pos.fila == 0){
637         result++;
638       }
639       i++;
640     }
641     return result;
642   }
643
644   bool Nivel::valoresDeEntradaEnRango(int ancho, int alto, int soles, int
        ↪ spawnSize){
645     return ancho > 0 && alto > 0 && soles >= 0 && spawnSize > 0;
646   }
647
648   FlorEnJuego Nivel::daniarFlor(FlorEnJuego flor, vector<VampiroEnJuego>
        ↪ vampiros){
649     FlorEnJuego res = FlorEnJuego(flor.flor, flor.pos, flor.vida);
650     int danioTotal = 0;
651     int i = 0;
652     while(i < vampiros.size()){
653       VampiroEnJuego vej = vampiros.at(i);
654       if(mismaPosicion(vej.pos, flor.pos)){
655         danioTotal += vej.vampiro.cuantoPegaV();
656       }
657       i++;
658     }
659     res.vida = res.vida - danioTotal;
660     return res;
661   }
662
663   bool Nivel::florMuerta(FlorEnJuego flor, vector<VampiroEnJuego> vampiros){
664     return (florExploto(flor, vampiros) || daniarFlor(flor, vampiros).vida <=
          ↪ 0);
665   }
666
667   bool Nivel::florExploto(FlorEnJuego flor, vector<VampiroEnJuego> vampiros){
668     Habilidad hab = Explotar;
669     return (tieneHabilidad(flor, hab) && vampiroEnMismaPosicion(flor,
          ↪ vampiros));
```

```
670 }
671
672 bool Nivel::tieneHabilidad(FlorEnJuego florEnJuego, Habilidad habilidad){
673   int i = 0;
674   bool res = false;
675   vector<Habilidad> habilidades = florEnJuego.flor.habilidadesF();
676   while(i < habilidades.size()){
677     if(habilidades.at(i) == habilidad){
678       res = true;
679     }
680     i++;
681   }
682   return res;
683 }
684
685 bool Nivel::vampiroEnMismaPosicion(FlorEnJuego florEnJuego, vector<
    ↪ VampiroEnJuego> vampiros){
686
687   bool res = false;
688   int i = 0;
689   while(i < vampiros.size()){
690     if(mismaPosicion(vampiros.at(i).pos, florEnJuego.pos)){
691       res = true;
692     }
693     i++;
694   }
695   return res;
696 }
697
698 bool Nivel::vampiroMuerto(VampiroEnJuego vampiro, vector<FlorEnJuego>
    ↪ flores, vector<VampiroEnJuego> vampiros){
699
700   return daniarVampiro(vampiro, flores, vampiros).vida <= 0;
701 }
702
703 VampiroEnJuego Nivel::daniarVampiro(VampiroEnJuego vampiroEnJuego, vector<
    ↪ FlorEnJuego> flores, vector<VampiroEnJuego> vampiros){
704
705   VampiroEnJuego vampiro = VampiroEnJuego(vampiroEnJuego.vampiro,
      ↪ vampiroEnJuego.pos, vampiroEnJuego.vida);
706
707   int i = 0;
708   int danioTotal = 0;
709   while(i < flores.size()){
710     if(enMira(flores.at(i), vampiro) && !intercepta(flores.at(i), vampiro,
        ↪ vampiros)){
711       danioTotal += flores.at(i).flor.cuantoPegaF();
712     }
713     i++;
714   }
715   vampiro.vida = vampiro.vida - danioTotal;
716   return vampiro;
717 }
718
```

```
719 bool Nivel::enMira(FlorEnJuego flor, VampiroEnJuego vampiro){
720   return flor.pos.fila == vampiro.pos.fila && flor.pos.columna <= vampiro.
        ↪ pos.columna;
721 }
722
723 bool Nivel::intercepta(FlorEnJuego flor, VampiroEnJuego vampiro, vector<
      ↪ VampiroEnJuego> vampiros){
724
725   bool res = false;
726   int i = 0;
727   while(i < vampiros.size()){
728     if(vampiros.at(i).pos.fila == flor.pos.fila && flor.pos.columna <=
          ↪ vampiros.at(i).pos.columna && vampiros.at(i).pos.columna <
          ↪ vampiro.pos.columna){
729       res = true;
730     }
731     i++;
732   }
733   return res;
734 }
735
736 vector<VampiroEnJuego> Nivel::nuevosVampiros(){
737   int i = 0;
738   vector<VampiroEnJuego> nuevosVamps = vector<VampiroEnJuego>();
739   while(i < this->_spawning.size()){
740     VampiroEnEspera vee = this->_spawning.at(i);
741     if(vee.turno == (this->_turno + 1)){
742       Posicion pos = Posicion(vee.fila, this->_ancho);
743       VampiroEnJuego vej = VampiroEnJuego(vee.vampiro, pos, vee.vampiro.
            ↪ vidaV());
744       nuevosVamps.push_back(vej);
745     }
746     i++;
747   }
748   return nuevosVamps;
749 }
750
751 bool Nivel::mismaPosicion(Posicion p1, Posicion p2){
752   return p1.fila == p2.fila && p1.columna == p2.columna;
753 }
```

Juego.h

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

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

class Juego
{
private:

   vector<Flor> _flores;
   vector<Vampiro> _vampiros;
   vector<Nivel> _niveles;
   int _nivelActual;

   bool nivelVacio(Nivel n);
   void quitarRepetidosFlores(vector<Flor>& flores);
   bool mismasHabilidadesFlor(Flor f1,Flor f2);
   bool habilidadPerteneceAFlor(int h, Flor f2);
   void quitarRepetidosVampiros(vector<Vampiro>& vamp);
   bool vampirosIguales(Vampiro v1, Vampiro v2);
   bool igualesVampirosEnEspera(VampiroEnEspera v1, VampiroEnEspera v2);
   vector<int> darNivelesGanados();
   void reducirVidaALaMitad(VampiroEnJuego& vampiro);
   void chitearNivel(Nivel& nivelTrucado);
   bool estadoFuturo(Nivel& ni, Nivel& nf);

public:

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

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

Juego.cpp

```
1  #include "Juego.h"
2
3  using namespace std;
4
5  Juego::Juego()
6  {
7  }
8
9  Juego::Juego(vector<Flor>& flores, vector<Vampiro>& vamps)
10 {
11   quitarRepetidosFlores(flores);
12   quitarRepetidosVampiros(vamps);
13   this->_flores = flores;
14   this->_vampiros = vamps;
15   this->_niveles = vector<Nivel>();
16 }
17
18
19 int Juego::nivelActual()
20 {
21 }
22
23
24 void Juego::pasarNivel()
25 {
26 }
27
28
29 vector<Flor>& Juego::floresJ()
30 {
31   return this->_flores;
32 }
33
34 vector<Vampiro>& Juego::vampirosJ()
35 {
36   return this->_vampiros;
37 }
38
39 vector<Nivel>& Juego::nivelesJ()
40 {
41   return this->_niveles;
42 }
43
44 void Juego::agregarNivel(Nivel& n, int i)
45 {
46   if(nivelVacio(n)){
47
48     vector<Nivel> niveles = vector<Nivel>();
49     int j = 0;
50
51     if(i == this->_niveles.size()){
52
```

```
53        this->_niveles.push_back(n);
54
55      }
56
57      else if (i>=0 && i<this->_niveles.size()) {
58
59        while(j < this->_niveles.size()){
60          if(j == i) niveles.push_back(n);
61          niveles.push_back(this->_niveles.at(j));
62          j++;
63        }
64        this->_niveles = niveles;
65      }
66
67    }
68
69  }
70
71  void Juego::jugarNivel(Nivel& n, int i)
72  {
73    if(i >= 0 && i < this->_niveles.size() &&
74      estadoFuturo(this->_niveles[i], n)) this->_niveles[i] = n;
75  }
76
77  bool Juego::estadoFuturo(Nivel& ni, Nivel& nf){
78    bool futuro = true;
79    int i = 0, j = 0, cuenta1 = 0, cuenta2 = 0;
80    vector<VampiroEnEspera> vampirosFuturos;
81
82    if (ni.turnoN() >= nf.turnoN() || ni.altoN() != nf.altoN() || ni.anchoN()
        ↪ != nf.anchoN()) futuro=false;
83
84    while (futuro && i < ni.spawningN().size()){
85
86      if (ni.spawningN()[i].turno > nf.turnoN()) {
87        vampirosFuturos.push_back(ni.spawningN()[i]);
88      }
89      i++;
90    }
91
92    if(vampirosFuturos.size() != nf.spawningN().size()) futuro=false;
93
94    i = 0;
95    while (futuro && i < vampirosFuturos.size()){
96
97      cuenta1++;
98
99      j = 0;
100     while(j < nf.spawningN().size()){
101       if(igualesVampirosEnEspera(vampirosFuturos[i], nf.spawningN()[j]))
            ↪ cuenta2++;
102       j++;
103     }
104
```

```
105     if (cuenta1 != cuenta2) futuro=false;
106     i++;
107   }
108
109   return futuro;
110 }
111
112 bool Juego::igualesVampirosEnEspera(VampiroEnEspera v1, VampiroEnEspera v2)
      ↪ {
113   bool res = false;
114
115   if(v1.vampiro.claseV() == v2.vampiro.claseV() &&
116       v1.fila == v2.fila &&
117       v1.turno == v2.turno){
118     res = true;
119   }
120   return res;
121 }
122
123 vector<Nivel> Juego::estosSaleFacil()
124 {
125   int i = 0, maxSoles=0, maxFlores=0;
126   vector<Nivel> faciles;
127
128   while (i < this->_niveles.size()){
129     if(this->_niveles[i].solesN() > maxSoles) maxSoles = this->_niveles[i].
        ↪ solesN();
130
131     i++;
132   }
133
134   i = 0;
135   while (i < this->_niveles.size()){
136     if(this->_niveles[i].solesN() == maxSoles &&
137        this->_niveles[i].floresN().size() > maxFlores)
138
139       maxFlores = this->_niveles[i].floresN().size();
140
141     i++;
142   }
143
144   i = 0;
145   while (i < this->_niveles.size()){
146     if(this->_niveles[i].solesN() == maxSoles &&
147         this->_niveles[i].floresN().size() == maxFlores)
148
149       faciles.push_back(this->_niveles[i]);
150
151     i++;
152   }
153   return faciles;
154 }
155
156 void Juego::altoCheat(int n)
```

```
157 {
158   if(n >= 0 && n < this->_niveles.size() && this->_niveles[n].vampirosN().
        ↪ size()!=0){
159
160     int i = 0;
161     while(i < this->_niveles[n].vampirosN().size()){
162         this->_niveles[n].vampirosN()[i].vida /= 2;
163         i++;
164     }
165   }
166 }
167
168 bool Juego::muyDeExactas()
169 {
170   vector<int> nivelesGanados = darNivelesGanados();
171   bool res = true;
172   if(nivelesGanados.size() == 0) res = false;
173   if(nivelesGanados.size() >= 1 && nivelesGanados.at(0) != 1) res = false;
174   if(nivelesGanados.size() >= 2 && nivelesGanados.at(1) != 2) res = false;
175
176   int i = 2;
177   while(i < nivelesGanados.size()){
178     if(nivelesGanados.at(i) != nivelesGanados.at(i-1) +
179       nivelesGanados.at(i-2)) res = false;
180
181     i++;
182   }
183   return res;
184 }
185
186 void Juego::Mostrar(ostream& os)
187 {
188   int i = 0;
189   os << "JUEGO:" << endl;
190   //Muestro vector de flores del juego
191   os << endl << "** Flores del JUEGO **" << endl;
192   while(i < this->_flores.size()){
193
194     os << endl;
195     this->_flores.at(i).Mostrar(os);
196     i++;
197   }
198
199   i = 0;
200   //Muestro el vector de vampiros del Juego
201   os << endl << "** Vampiros del JUEGO **" << endl;
202   while(i < this->_vampiros.size()){
203
204     os << endl;
205     this->_vampiros.at(i).Mostrar(os);
206     i++;
207   }
208
209   i = 0;
```

```
210   //Muestro el vector de niveles del Juego
211   os << endl << "** Niveles del JUEGO **" << endl;
212   while(i < this->_niveles.size()){
213
214     os << endl;
215     this->_niveles.at(i).Mostrar(os);
216     i++;
217   }
218   os << endl;
219 }
220
221 void Juego::Guardar(ostream& os)
222 {
223   os << "{ J [";
224
225   if(this->_flores.size() > 0){
226     int i = 0;
227     while(i < this->_flores.size()){
228       os << " ";
229       Flor flor = this->_flores.at(i);
230       flor.Guardar(os);
231       i++;
232     }
233   }
234
235   os << " ] [" ; //termina la lista de flores, empieza la de vampiros
236
237   if(this->_vampiros.size() > 0){
238
239     int i = 0;
240     while(i < this->_vampiros.size()){
241       os << " ";
242       Vampiro vampiro = this->_vampiros.at(i);
243       vampiro.Guardar(os);
244       i++;
245     }
246   }
247
248   os << " ] [" ; // termina la lista de vampiros, empieza la de niveles
249
250   if(this->_niveles.size() > 0){
251     int i = 0;
252     while(i < this->_niveles.size()){
253       os << " ";
254       Nivel nivel = this->_niveles.at(i);
255       nivel.Guardar(os);
256       i++;
257     }
258
259   }
260   os << " ] }" ; // termina la lista de niveles y el juego
261 }
262
263 void Juego::Cargar(istream& is)
```

```
264 {
265   string buscaValores;
266   vector<Flor> vectorFlorVacio;
267   vector<Vampiro> vectorVampiroVacio;
268   vector<Nivel> vectorNivelVacio;
269 
270   //inicializo todos los vectores en cero
271   this->_flores = vectorFlorVacio;
272   this->_vampiros = vectorVampiroVacio;
273   this->_niveles = vectorNivelVacio;
274 
275   getline(is,buscaValores,' '); // '{'
276   getline(is,buscaValores,' '); // 'J'
277   getline(is,buscaValores,' '); // '[' comienzo de la lista de flores del
      ↪ juego
278 
279   while(buscaValores != "]"){
280 
281     getline(is,buscaValores,' '); // '{' // ']'
282 
283     if(buscaValores != "]"){
284       Flor flor;
285       flor.Cargar(is);
286       this->_flores.push_back(flor);
287     }
288   } // fin de la carga de flores del nivel
289 
290   getline(is,buscaValores,' '); // '[' comienzo de la lista de vampiros del
      ↪ juego
291 
292   while(buscaValores != "]"){
293 
294     getline(is,buscaValores,' '); // '{' // ']'
295 
296     if(buscaValores != "]"){
297       Vampiro vamp;
298       vamp.Cargar(is);
299       this->_vampiros.push_back(vamp);
300     }
301   } // fin de la carga de vampiros del nivel
302 
303   getline(is,buscaValores,' '); // '[' comienzo de la lista de niveles del
      ↪ juego
304 
305 
306   while(buscaValores != "]"){
307 
308     getline(is,buscaValores,' ');
309 
310     if(buscaValores != "]"){
311       Nivel nivel;
312       nivel.Cargar(is);
313       this->_niveles.push_back(nivel);
314     }
```

```
315   } // fin de la carga de niveles
316
317   quitarRepetidosFlores(this->_flores);
318   quitarRepetidosVampiros(this->_vampiros);
319 }
320
321 bool Juego::nivelVacio(Nivel n){
322   return n.turnoN() == 0 && n.floresN().size() == 0 && n.vampirosN().size()
        ↪  == 0;
323 }
324
325 void Juego::quitarRepetidosFlores(vector<Flor>& flores){
326   int i = flores.size() - 1;
327   int j;
328   Flor f1;
329   Flor f2;
330
331   while (0 < i){
332     f1 = flores.at(i);
333     j = i - 1;
334
335     while(0 <= j){
336       f2 = flores.at(j);
337       if(mismasHabilidadesFlor(f1,f2)){
338         flores.pop_back();
339         break;
340       }
341       j--;
342     }
343     i--;
344   }
345 }
346
347 bool Juego::mismasHabilidadesFlor(Flor f1, Flor f2){
348   int res = true;
349
350   if(f1.habilidadesF().size() != f2.habilidadesF().size()){
351     res = false;
352   } else {
353     int i = 0;
354
355     while(i < f1.habilidadesF().size()){
356
357       if(!habilidadPerteneceAFlor(f1.habilidadesF().at(i),f2))
358         res = false;
359       i++;
360     }
361   }
362   return res;
363 }
364
365 bool Juego::habilidadPerteneceAFlor(int h, Flor f2){
366   bool res = false;
367   int i = 0;
```

```
368
369   while(i < f2.habilidadesF().size()){
370     if(f2.habilidadesF().at(i) == h)
371       res = true;
372     i++;
373   }
374
375   return res;
376 }
377
378 void Juego::quitarRepetidosVampiros(vector<Vampiro>& vamp){
379   Vampiro v1;
380   Vampiro v2;
381   int i = vamp.size() - 1;
382   int j;
383
384   while(0 < i){
385     v1 = vamp.at(i);
386     j = i - 1;
387
388     while(0 <= j){
389       v2 = vamp.at(j);
390
391       if(vampirosIguales(v1,v2)){
392         vamp.pop_back();
393         break;
394       }
395
396       j--;
397     }
398
399     i--;
400   }
401 }
402
403 bool Juego::vampirosIguales(Vampiro v1, Vampiro v2){
404   bool res = false;
405   if(v1.vidaV() == v2.vidaV() && v1.claseV() == v2.claseV() &&
406      v1.cuantoPegaV() == v2.cuantoPegaV())
407     res = true;
408
409   return res;
410 }
411
412 vector<int> Juego::darNivelesGanados(){
413   vector<int> nivelesGanados = vector<int>();
414   int i = 0;
415   while(i < this->_niveles.size()){
416     Nivel nivel = this->_niveles.at(i);
417     if(nivel.vampirosN().empty() && nivel.spawningN().empty()){
418       nivelesGanados.push_back(i+1);
419     }
420     i++;
421   }
```

```
422     return nivelesGanados;
423 }
```

Main.cpp

```
#include "Juego.h"
#include "Vampiro.h"
#include "Nivel.h"
#include <iostream>
#include <fstream>

using namespace std;

int main(){

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

   vector<Habilidad> hab;
   hab.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);

   Flor f(25,0,hab3);

   cout << f.vidaF() << ' ' << f.cuantoPegaF() << endl;

   Vampiro v(Caminante,50,50);

   cout << v.vidaV() << ' ' << v.cuantoPegaV() << endl;

   vector<VampiroEnEspera> spawn;
   spawn.push_back(VampiroEnEspera(v,2,2));

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

   cout << n.solesN() << endl;

   n.agregarFlor(f, Posicion(2,2));

   cout << n.solesN() << endl;

   n.pasarTurno();

   cout << n.solesN() << endl;

   cout << n.terminado() << ' ' << n.obsesivoCompulsivo() << endl;


   vector<Flor> fs;
   fs.push_back(f);

```

```
53    vector<Vampiro> vs;
54    vs.push_back(v);
55
56    Juego j(fs,vs);
57
58    j.agregarNivel(n,0);
59
60    cout << j.muyDeExactas() << endl;
61
62    cout << endl;
63
64    /*     TESTS DE JUEGO PARA ENTREGAR     */
65    cout << "TEST DE JUEGO" << endl << endl;
66
67    vector<Habilidad> habilidad1;
68    habilidad1.push_back(Atacar);
69    habilidad1.push_back(Explotar);
70    habilidad1.push_back(Generar);
71    vector<Habilidad> habilidad2;
72    habilidad2.push_back(Atacar);
73    habilidad2.push_back(Explotar);
74    vector<Habilidad> habilidad3;
75    habilidad3.push_back(Generar);
76
77    //el constructor de flor genera la vida y el cuanto pega segun las
          ↪ habilidades
78    Flor f1(1,1,habilidad1);
79    Flor f2(1,1,habilidad2);
80    Flor f3(1,1,habilidad3);
81
82    Vampiro vampiro1(Caminante,10,10);
83    Vampiro vampiro2(Caminante,12,12);
84    Vampiro vampiro3(Desviado,15,15);
85
86    vector<Flor> floresJuego;
87    floresJuego.push_back(f1);
88    floresJuego.push_back(f2);
89    floresJuego.push_back(f3);
90
91    vector<Flor> floresVacio;
92
93    vector<Vampiro> vampirosJuego;
94    vampirosJuego.push_back(vampiro1);
95    vampirosJuego.push_back(vampiro2);
96    vampirosJuego.push_back(vampiro3);
97
98    vector<Vampiro> vampirosVacio;
99
100   VampiroEnEspera v1 = VampiroEnEspera(vampiro1, 1, 1);
101   VampiroEnEspera v2 = VampiroEnEspera(vampiro2, 2, 2);
102   VampiroEnEspera v3 = VampiroEnEspera(vampiro3, 3, 3);
103   VampiroEnEspera v4 = VampiroEnEspera(vampiro1, 4, 4);
104   VampiroEnEspera v5 = VampiroEnEspera(vampiro2, 3, 6);
105   VampiroEnEspera v6 = VampiroEnEspera(vampiro3, 4, 2);
```

```
106
107     vector<VampiroEnEspera> spawningList1;
108     spawningList1.push_back(v1);
109     spawningList1.push_back(v2);
110     spawningList1.push_back(v3);
111     vector<VampiroEnEspera> spawningList2;
112     spawningList2.push_back(v4);
113     spawningList2.push_back(v5);
114     spawningList2.push_back(v6);
115     vector<VampiroEnEspera> spawningList3;
116     spawningList3.push_back(v1);
117     spawningList3.push_back(v4);
118     spawningList3.push_back(v6);
119     vector<VampiroEnEspera> spawningList4;
120     spawningList4.push_back(v2);
121     spawningList4.push_back(v5);
122     spawningList4.push_back(v3);
123     vector<VampiroEnEspera> spawningVacio;
124
125     Nivel n1(10,10,50,spawningList1);
126     Nivel n2(9,9,13,spawningList2);
127     Nivel n3(12,12,24,spawningList3);
128     Nivel n4(15,15,47,spawningList4);
129     Nivel n5(15,15,0, spawningVacio);
130
131     Juego juegoTest(floresJuego,vampirosJuego);
132
133     //  TESTS AGREGAR NIVEL
134
135     /*
136     Test 1:
137     Agrego el nivel n1 a juegoTest, la lista de niveles deberia tener un
            ↪ nivel.
138     */
139
140     cout << "TEST 1 AGREGAR NIVEL" << endl;
141     cout << "Longitud de la lista de niveles de juego: " << juegoTest.
            ↪ nivelesJ().size() << endl;
142     juegoTest.agregarNivel(n1,0);
143     cout << "Longitud de la lista de niveles de juego: " << juegoTest.
            ↪ nivelesJ().size() << endl;
144
145     /*
146     Salida del primer test:
147     TEST 1 AGREGAR NIVEL
148     Longitud de la lista de niveles de juego: 0
149     Longitud de la lista de niveles de juego: 1
150     */
151     cout << endl << endl;
152
153     /*
154     Test 2:
155     Agrego el nivel n2 a juegoTest en la posicion 1. El nivel deberia
            ↪ agregarse despues de n1.
```

```
156   Reviso si el nivel se agrego correctamente viendo el ancho, algo y soles
          ↪ del nivel en
157   cada posicion.
158   */
159
160   cout << "TEST 2 AGREGAR NIVEL" << endl;
161   cout << "Longitud de la lista de niveles de juego: " << juegoTest.
          ↪ nivelesJ().size() << endl;
162   juegoTest.agregarNivel(n2,1);
163   cout << "Longitud de la lista de niveles de juego: " << juegoTest.
          ↪ nivelesJ().size() << endl;
164
165   cout << endl << "Veo si los niveles estan en sus posiciones:" << endl <<
          ↪ endl;
166   if(juegoTest.nivelesJ().at(0).anchoN() == 10 && juegoTest.nivelesJ().at
          ↪ (0).altoN() == 10 &&
167     juegoTest.nivelesJ().at(0).solesN() == 50){
168     cout << "Nivel n1 en la posicion correcta." << endl;
169   } else {
170     cout << "Nivel n1 NO esta en la posicion correcta." << endl;
171   }
172
173   if(juegoTest.nivelesJ().at(1).anchoN() == 9 && juegoTest.nivelesJ().at(1)
          ↪ .altoN() == 9 &&
174     juegoTest.nivelesJ().at(1).solesN() == 13){
175     cout << "Nivel n2 en la posicion correcta." << endl;
176   } else {
177     cout << "Nivel n2 NO esta en la posicion correcta." << endl;
178   }
179
180   /*
181   Salida del segundo test:
182   TEST 2 AGREGAR NIVEL
183   Longitud de la lista de niveles de juego: 1
184   Longitud de la lista de niveles de juego: 2
185
186   Veo si los niveles estan en sus posiciones:
187
188   Nivel n1 en la posicion correcta.
189   Nivel n2 en la posicion correcta.
190   */
191   cout << endl << endl;
192
193   /*
194   Test 3:
195   Agrego el nivel n3 a juegoTest en la posicion 1. El nivel deberia
          ↪ agregarse entre el n1 y n2.
196   */
197
198   cout << "TEST 3 AGREGAR NIVEL" << endl;
199   cout << "Longitud de la lista de niveles de juego: " << juegoTest.
          ↪ nivelesJ().size() << endl;
200   juegoTest.agregarNivel(n3,1);
201   cout << "Longitud de la lista de niveles de juego: " << juegoTest.
```

```
    ↪ nivelesJ().size() << endl;
202
203 cout << endl << "Veo si los niveles estan en sus posiciones:" << endl <<
    ↪ endl;
204 if(juegoTest.nivelesJ().at(0).anchoN() == 10 && juegoTest.nivelesJ().at
    ↪ (0).altoN() == 10 &&
205   juegoTest.nivelesJ().at(0).solesN() == 50){
206   cout << "Nivel n1 en la posicion correcta." << endl;
207 } else {
208   cout << "Nivel n1 NO esta en la posicion correcta." << endl;
209 }
210
211 if(juegoTest.nivelesJ().at(1).anchoN() == 12 && juegoTest.nivelesJ().at
    ↪ (1).altoN() == 12 &&
212   juegoTest.nivelesJ().at(1).solesN() == 24){
213   cout << "Nivel n3 en la posicion correcta." << endl;
214 } else {
215   cout << "Nivel n3 NO esta en la posicion correcta." << endl;
216 }
217
218 if(juegoTest.nivelesJ().at(2).anchoN() == 9 && juegoTest.nivelesJ().at(2)
    ↪ .altoN() == 9 &&
219   juegoTest.nivelesJ().at(2).solesN() == 13){
220   cout << "Nivel n2 en la posicion correcta." << endl;
221 } else {
222   cout << "Nivel n2 NO esta en la posicion correcta." << endl;
223 }
224
225 /*
226 Salida del tercer test:
227 TEST 3 AGREGAR NIVEL
228 Longitud de la lista de niveles de juego: 2
229 Longitud de la lista de niveles de juego: 3
230
231 Veo si los niveles estan en sus posiciones:
232
233 Nivel n1 en la posicion correcta.
234 Nivel n3 en la posicion correcta.
235 Nivel n2 en la posicion correcta.
236 */
237 cout << endl << endl;
238
239 /*
240 Test 4:
241 Agrego el nivel n4 a juegoTest en la posicion 2. El nivel deberia
    ↪ agregarse entre el n3 y n2.
242 */
243
244 cout << "TEST 4 AGREGAR NIVEL" << endl;
245 cout << "Longitud de la lista de niveles de juego: " << juegoTest.
    ↪ nivelesJ().size() << endl;
246 juegoTest.agregarNivel(n4,2);
247 cout << "Longitud de la lista de niveles de juego: " << juegoTest.
    ↪ nivelesJ().size() << endl;
```

```
248
249   cout << endl << "Veo si los niveles estan en sus posiciones:" << endl <<
          ↪ endl;
250
251   if(juegoTest.nivelesJ().at(0).anchoN() == 10 && juegoTest.nivelesJ().at
          ↪ (0).altoN() == 10 &&
252     juegoTest.nivelesJ().at(0).solesN() == 50){
253     cout << "Nivel n1 en la posicion correcta." << endl;
254   } else {
255     cout << "Nivel n1 NO esta en la posicion correcta." << endl;
256   }
257
258   if(juegoTest.nivelesJ().at(1).anchoN() == 12 && juegoTest.nivelesJ().at
          ↪ (1).altoN() == 12 &&
259     juegoTest.nivelesJ().at(1).solesN() == 24){
260     cout << "Nivel n3 en la posicion correcta." << endl;
261   } else {
262     cout << "Nivel n3 NO esta en la posicion correcta." << endl;
263   }
264
265   if(juegoTest.nivelesJ().at(2).anchoN() == 15 && juegoTest.nivelesJ().at
          ↪ (2).altoN() == 15 &&
266     juegoTest.nivelesJ().at(2).solesN() == 47){
267     cout << "Nivel n4 en la posicion correcta." << endl;
268   } else {
269     cout << "Nivel n4 NO esta en la posicion correcta." << endl;
270   }
271
272   if(juegoTest.nivelesJ().at(3).anchoN() == 9 && juegoTest.nivelesJ().at(3)
          ↪ .altoN() == 9 &&
273     juegoTest.nivelesJ().at(3).solesN() == 13){
274     cout << "Nivel n2 en la posicion correcta." << endl;
275   } else {
276     cout << "Nivel n2 NO esta en la posicion correcta." << endl;
277   }
278
279   /*
280   Salida del cuarto test:
281   TEST 4 AGREGAR NIVEL
282   Longitud de la lista de niveles de juego: 3
283   Longitud de la lista de niveles de juego: 4
284
285   Veo si los niveles estan en sus posiciones:
286
287   Nivel n1 en la posicion correcta.
288   Nivel n3 en la posicion correcta.
289   Nivel n4 en la posicion correcta.
290   Nivel n2 en la posicion correcta.
291   */
292   cout << endl << endl;
293
294
295   //  TESTS JUGAR NIVEL
296
```

```
297     vector<VampiroEnEspera> spawningList5;
298     spawningList5.push_back(v1);
299     spawningList5.push_back(v2);
300     spawningList5.push_back(v3);
301     spawningList5.push_back(v4);
302     Nivel nivelTest(9,9,100,spawningList5);
303 
304     /*
305     Test:
306     Agrego el nivelTest a la posicion 0 de la lista de niveles de juegoTest.
307     Voy usando la funcion pasarturno en nivelTest para obtener un "estado
            ↪ futuro" del juego y
308     luego aplico la funcion jugarNivel.
309     El nivel deberia ir cambiandose al estado nuevo de nivelTest.
310     */
311 
312     juegoTest.agregarNivel(nivelTest,0);
313 
314     cout << "TEST 1 JUGAR NIVEL" << endl;
315     cout << "Nivel antes de usar la funcion: " << endl;
316     cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
317     cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
318     cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
319     cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
320     cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
            ↪ vampirosN().size() << endl;
321     cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
            ↪ spawningN().size() << endl;
322     nivelTest.pasarTurno();
323     juegoTest.jugarNivel(nivelTest,0);
324     cout << "Nivel despues de usar la funcion: " << endl;
325     cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
326     cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
327     cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
328     cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
329     cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
            ↪ vampirosN().size() << endl;
330     cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
            ↪ spawningN().size() << endl;
331 
332     /*
333     Salida del primer test:
334     TEST 1 JUGAR NIVEL
335     Nivel antes de usar la funcion:
336     Ancho: 9
337     Alto: 9
338     Soles: 100
339     Turno: 0
340     Longitud lista de vampiros: 0
341     Longitud lista de spawning: 4
342     Nivel despues de usar la funcion:
343     Ancho: 9
344     Alto: 9
345     Soles: 101
```

```
346     Turno: 1
347     Longitud lista de vampiros: 1
348     Longitud lista de spawning: 3
349     */
350     cout << endl << endl;
351 
352     cout << "TEST 2 JUGAR NIVEL" << endl;
353     cout << "Nivel antes de usar la funcion: " << endl;
354     cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
355     cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
356     cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
357     cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
358     cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
        ↪ vampirosN().size() << endl;
359     cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
        ↪ spawningN().size() << endl;
360     nivelTest.pasarTurno();
361     juegoTest.jugarNivel(nivelTest,0);
362     cout << "Nivel despues de usar la funcion: " << endl;
363     cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
364     cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
365     cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
366     cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
367     cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
        ↪ vampirosN().size() << endl;
368     cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
        ↪ spawningN().size() << endl;
369 
370     /*
371     Salida del segundo test:
372     TEST 2 JUGAR NIVEL
373     Nivel antes de usar la funcion:
374     Ancho: 9
375     Alto: 9
376     Soles: 101
377     Turno: 1
378     Longitud lista de vampiros: 1
379     Longitud lista de spawning: 3
380     Nivel despues de usar la funcion:
381     Ancho: 9
382     Alto: 9
383     Soles: 102
384     Turno: 2
385     Longitud lista de vampiros: 2
386     Longitud lista de spawning: 2
387     */
388     cout << endl << endl;
389 
390     cout << "TEST 3 JUGAR NIVEL" << endl;
391     cout << "Nivel antes de usar la funcion: " << endl;
392     cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
393     cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
394     cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
395     cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
```

```
396   cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
          ↪ vampirosN().size() << endl;
397   cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
          ↪ spawningN().size() << endl;
398   nivelTest.pasarTurno();
399   juegoTest.jugarNivel(nivelTest,0);
400   cout << "Nivel despues de usar la funcion: " << endl;
401   cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
402   cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
403   cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
404   cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
405   cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
          ↪ vampirosN().size() << endl;
406   cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
          ↪ spawningN().size() << endl;
407
408   /*
409   Salida del tercer test:
410   TEST 3 JUGAR NIVEL
411   Nivel antes de usar la funcion:
412   Ancho: 9
413   Alto: 9
414   Soles: 102
415   Turno: 2
416   Longitud lista de vampiros: 2
417   Longitud lista de spawning: 2
418   Nivel despues de usar la funcion:
419   Ancho: 9
420   Alto: 9
421   Soles: 103
422   Turno: 3
423   Longitud lista de vampiros: 3
424   Longitud lista de spawning: 1
425   */
426   cout << endl << endl;
427
428   cout << "TEST 4 JUGAR NIVEL" << endl;
429   cout << "Nivel antes de usar la funcion: " << endl;
430   cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
431   cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
432   cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
433   cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
434   cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
          ↪ vampirosN().size() << endl;
435   cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
          ↪ spawningN().size() << endl;
436   nivelTest.pasarTurno();
437   juegoTest.jugarNivel(nivelTest,0);
438   cout << "Nivel despues de usar la funcion: " << endl;
439   cout << "Ancho: " << juegoTest.nivelesJ().at(0).anchoN() << endl;
440   cout << "Alto: " << juegoTest.nivelesJ().at(0).altoN() << endl;
441   cout << "Soles: " << juegoTest.nivelesJ().at(0).solesN() << endl;
442   cout << "Turno: " << juegoTest.nivelesJ().at(0).turnoN() << endl;
443   cout << "Longitud lista de vampiros: " << juegoTest.nivelesJ().at(0).
```

```
        ↪ vampirosN().size() << endl;
444   cout << "Longitud lista de spawning: " << juegoTest.nivelesJ().at(0).
        ↪ spawningN().size() << endl;
445
446   /*
447   Salida del cuarto test:
448   TEST 4 JUGAR NIVEL
449   Nivel antes de usar la funcion:
450   Ancho: 9
451   Alto: 9
452   Soles: 103
453   Turno: 3
454   Longitud lista de vampiros: 3
455   Longitud lista de spawning: 1
456   Nivel despues de usar la funcion:
457   Ancho: 9
458   Alto: 9
459   Soles: 104
460   Turno: 4
461   Longitud lista de vampiros: 4
462   Longitud lista de spawning: 0
463   */
464   cout << endl << endl;
465
466
467
468   //  TESTS GUARDAR JUEGO
469
470     //Test 1: guardo juego sin flores, con vampiros, sin niveles
471
472   cout << "TEST 1 GUARDAR JUEGO" << endl;
473   Juego juegoTest1(floresVacio, vampirosJuego);
474   Juego juegoTestTemp1;
475
476   ofstream ost1 ("TestGuardarJuego1_2");
477
478   if (ost1.is_open()) {
479     juegoTest1.Guardar(ost1);
480     ost1.close();
481   }
482
483   ifstream ist1("TestGuardarJuego1_2");
484   if (ist1.is_open()) {
485     juegoTestTemp1.Cargar(ist1);
486     ist1.close();
487   }
488
489   cout << "Se guardÃş el juego: " << endl;
490   juegoTestTemp1.Mostrar(cout);
491
492   /* Salida del primer test:
493
494     TEST 1 GUARDAR JUEGO
495     Se guardÃş el juego:
```

```
496       JUEGO:
497
498       ** Flores del JUEGO **
499
500       ** Vampiros del JUEGO **
501
502       INFO. DEL VAMPIRO:
503       Vida del Vampiro: 10
504       Cuanto Pega el Vampiro: 10
505       Clase del Vampiro: Caminante
506
507
508       INFO. DEL VAMPIRO:
509       Vida del Vampiro: 12
510       Cuanto Pega el Vampiro: 12
511       Clase del Vampiro: Caminante
512
513
514       INFO. DEL VAMPIRO:
515       Vida del Vampiro: 15
516       Cuanto Pega el Vampiro: 15
517       Clase del Vampiro: Desviado
518
519
520       ** Niveles del JUEGO **
521
522       */
523     cout << endl << endl;
524
525     //Test 2: guardo juego sin inicializar, sobreescribiendo el archivo del
          ↪ primer test
526
527
528     cout << "TEST 2 GUARDAR JUEGO" << endl;
529     Juego juegoTest2;
530     Juego juegoTestTemp2;
531
532     ofstream ost2 ("TestGuardarJuego1_2");
533     if (ost2.is_open()) {
534       juegoTest2.Guardar(ost2);
535       ost2.close();
536     }
537
538     ifstream ist2("TestGuardarJuego1_2");
539     if (ist2.is_open()) {
540       juegoTestTemp2.Cargar(ist2);
541       ist2.close();
542     }
543
544     cout << "Se guardÃş el juego: " << endl;
545     juegoTestTemp2.Mostrar(cout);
546
547
548     /*  Salida del segundo test:
```

```
549
550       TEST 2 GUARDAR JUEGO
551       Se guardÃş el juego:
552       JUEGO:
553
554       ** Flores del JUEGO **
555
556       ** Vampiros del JUEGO **
557
558       ** Niveles del JUEGO **
559     */
560
561     cout << endl << endl;
562
563     // Test 3: guardo un juego sin flores, sin vampiros, con niveles. El
        ↪ Ãžltimo nivel se construye con spawning vacÃyo
564
565     cout << "TEST 3 GUARDAR JUEGO" << endl;
566     Juego juegoTest3(floresVacio, vampirosVacio);
567     Juego juegoTestTemp3;
568     juegoTest3.agregarNivel(n1, 0);
569     juegoTest3.agregarNivel(n5, 1);
570
571     ofstream ost3("TestGuardarJuego3");
572     if (ost3.is_open()) {
573       juegoTest3.Guardar(ost3);
574       ost3.close();
575     }
576
577     ifstream ist3("TestGuardarJuego3");
578     if (ist3.is_open()) {
579       juegoTestTemp3.Cargar(ist3);
580       ist3.close();
581     }
582
583     if(juegoTestTemp3.nivelesJ().size() == 2 && juegoTestTemp3.floresJ().size
        ↪ () == 0 &&
584        juegoTestTemp3.vampirosJ().size() == 0){
585       cout << "El juego se guardÃş y cargo correctamente." << endl;
586     } else {
587       cout << "El juego no es el correcto" << endl;
588     }
589
590
591     /*
592     Salida del tercer test:
593     TEST 3 GUARDAR JUEGO
594     El juego se guardÃş y cargo correctamente.
595     */
596     // El spawning no es vacÃyo, ya que el constructor de nivel crea un
        ↪ vampiro "generico"
597     cout << endl << endl;
598
599
```

```
600   // Test 4: uso para guardar el juego juegoTest del principio con 3
          ↪ vampiros, 3 flores y 5 niveles
601   cout << "TEST 4 GUARDAR JUEGO" << endl;
602   Juego juegoTestTemp4;
603
604   ofstream ost4("TestGuardarJuego4");
605     if(ost4.is_open()) {
606       juegoTest.Guardar(ost4);
607       ost4.close();
608   }
609
610   ifstream is11("TestGuardarJuego4");
611   if(is11.is_open()) {
612     juegoTestTemp4.Cargar(is11);
613     is11.close();
614   }
615
616   if(juegoTestTemp4.nivelesJ().size() == 5 && juegoTestTemp4.floresJ().size
          ↪ () == 3 &&
617       juegoTestTemp4.vampirosJ().size() == 3){
618       cout << "El juego se guardÃş y cargo correctamente." << endl;
619     } else {
620       cout << "El juego no es el correcto" << endl;
621   }
622
623   /*
624   Salida del cuarto test:
625   TEST 4 GUARDAR JUEGO
626   El juego se guardÃş y cargo correctamente.
627   */
628   cout << endl << endl;
629
630
631   //  TESTS ESTOS SALE FACIL
632
633   vector<Nivel> nivelesFaciles;
634   /*
635   Test 1:
636   Pruebo juegoTest con la funcion, deberia devolver un vector con el nivel
          ↪ nivelTest que
637   quedo de la funcion jugarNivel, ya es el que tiene mas soles.
638   */
639
640   nivelesFaciles = juegoTest.estosSaleFacil();
641
642   cout << "TEST 1 ESTOS SALE FACIL" << endl;
643   cout << "Longitud de la lista de niveles faciles: " << nivelesFaciles.
          ↪ size() << endl;
644
645   if(nivelesFaciles.at(0).anchoN() == 9 && nivelesFaciles.at(0).altoN() ==
          ↪ 9 &&
646     nivelesFaciles.at(0).solesN() == 104){
647     cout << "Nivel correcto en la lista" << endl;
648   } else {
```

```
649       cout << "No es el nivel correcto" << endl;
650     }
651
652     /*
653     Salida del primer test:
654
655     TEST 1 ESTOS SALE FACIL
656     Longitud de la lista de niveles faciles: 1
657     Nivel correcto en la lista
658     */
659     cout << endl << endl;
660
661     /*
662     Test 2:
663     Cargo un juego con niveles y vampiros para hacer el proximo test.
664     El juego posee un vector con las mismas flores y los mismos vampiros y 2
            ↪ niveles, ambos
665     con 100 de soles y una flor cada uno. La funcion deberia devolver los 2
            ↪ niveles.
666     Nivel en posicion 0: ancho = 12, alto = 12, turno = 3, soles = 100.
667     Nivel en posicion 1: ancho = 9, alto = 9, turno = 2, soles = 100
668     */
669     ifstream is("TestEstosSaleFacil2");
670
671     if(is.is_open()){
672       juegoTest.Cargar(is);
673       is.close();
674     }
675
676     nivelesFaciles = juegoTest.estosSaleFacil();
677
678     cout << "TEST 2 ESTOS SALE FACIL" << endl;
679     cout << "Longitud de la lista de niveles faciles: " << nivelesFaciles.
            ↪ size() << endl;
680
681     if(nivelesFaciles.at(0).anchoN() == 12 && nivelesFaciles.at(0).altoN() ==
            ↪  12 &&
682       nivelesFaciles.at(0).solesN() == 100 && nivelesFaciles.at(0).turnoN()
            ↪ == 3){
683       cout << "Nivel correcto" << endl;
684     } else {
685       cout << "No es el nivel correcto" << endl;
686     }
687
688     if(nivelesFaciles.at(1).anchoN() == 9 && nivelesFaciles.at(1).altoN() ==
            ↪ 9 &&
689       nivelesFaciles.at(1).solesN() == 100 && nivelesFaciles.at(1).turnoN()
            ↪ == 2){
690       cout << "Nivel correcto" << endl;
691     } else {
692       cout << "No es el nivel correcto" << endl;
693     }
694
695     /*
```

```
696   Salida del segundo test:
697
698   TEST 2 ESTOS SALE FACIL
699   Longitud de la lista de niveles faciles: 2
700   Nivel correcto
701   Nivel correcto
702   */
703   cout << endl << endl;
704
705   /*
706   Test 3:
707   Cargo un juego con niveles y vampiros para hacer el proximo test.
708   El juego posee un vector con las mismas flores y los mismos vampiros y 3
        ↪ niveles.
709   Nivel en posicion 0: ancho = 12, alto = 12, turno = 3, soles = 100,
        ↪ cantidad flores = 1.
710   Nivel en posicion 1: ancho = 9, alto = 9, turno = 2, soles = 100,
        ↪ cantidad flores = 1.
711   Nivel en posicion 2: ancho = 10, alto = 10, turno = 6, soles = 100,
        ↪ cantidad flores = 2.
712   Deberia devolver el nivel de la posicion 2 por tener mas flores.
713   */
714
715   ifstream is2("TestEstosSaleFacil3");
716
717   if(is2.is_open()){
718     juegoTest.Cargar(is2);
719     is2.close();
720   }
721
722   nivelesFaciles = juegoTest.estosSaleFacil();
723
724   cout << "TEST 3 ESTOS SALE FACIL" << endl;
725   cout << "Longitud de la lista de niveles faciles: " << nivelesFaciles.
        ↪ size() << endl;
726
727   if(nivelesFaciles.at(0).anchoN() == 10 && nivelesFaciles.at(0).altoN() ==
        ↪  10 &&
728     nivelesFaciles.at(0).solesN() == 100 && nivelesFaciles.at(0).turnoN()
        ↪ == 6){
729     cout << "Nivel correcto" << endl;
730   } else {
731     cout << "No es el nivel correcto" << endl;
732   }
733
734   /*
735   Salida del tercer test:
736
737   TEST 3 ESTOS SALE FACIL
738   Longitud de la lista de niveles faciles: 1
739   Nivel correcto
740   */
741   cout << endl << endl;
742
```

```
743   /*
744   Test 4:
745   Cargo un juego con niveles y vampiros para hacer el proximo test.
746   El juego posee un vector con las mismas flores y los mismos vampiros y 4
          ↪ niveles.
747   Nivel en posicion 0: ancho = 12, alto = 12, turno = 3, soles = 100,
          ↪ cantidad flores = 1.
748   Nivel en posicion 1: ancho = 9, alto = 9, turno = 2, soles = 100,
          ↪ cantidad flores = 1.
749   Nivel en posicion 2: ancho = 10, alto = 10, turno = 6, soles = 100,
          ↪ cantidad flores = 2.
750   Nivel en posicion 3: ancho = 8, alto = 8, turno = 1, soles = 100,
          ↪ cantidad flores = 2.
751   Deberia devolver los ultimos dos niveles por tener la misma cantidad de
          ↪ soles y mayor
752   cantidad de flores.
753   */
754
755   ifstream is3("TestEstosSaleFacil4");
756
757   if(is3.is_open()){
758     juegoTest.Cargar(is3);
759     is3.close();
760   }
761
762   nivelesFaciles = juegoTest.estosSaleFacil();
763
764   cout << "TEST 4 ESTOS SALE FACIL" << endl;
765   cout << "Longitud de la lista de niveles faciles: " << nivelesFaciles.
          ↪ size() << endl;
766
767   if(nivelesFaciles.at(0).anchoN() == 10 && nivelesFaciles.at(0).altoN() ==
          ↪  10 &&
768     nivelesFaciles.at(0).solesN() == 100 && nivelesFaciles.at(0).turnoN()
          ↪ == 6){
769     cout << "Nivel correcto" << endl;
770   } else {
771     cout << "No es el nivel correcto" << endl;
772   }
773
774   if(nivelesFaciles.at(1).anchoN() == 8 && nivelesFaciles.at(1).altoN() ==
          ↪ 8 &&
775     nivelesFaciles.at(1).solesN() == 100 && nivelesFaciles.at(1).turnoN()
          ↪ == 1){
776     cout << "Nivel correcto" << endl;
777   } else {
778     cout << "No es el nivel correcto" << endl;
779   }
780
781   /*
782   Salida del cuarto test:
783
784   TEST 4 ESTOS SALE FACIL
785   Longitud de la lista de niveles faciles: 2
```

```
786     Nivel correcto
787     Nivel correcto
788     */
789     cout << endl << endl;
790
791
792     // TESTS ALTO CHEAT
793
794     /*
795     Test 1:
796     Cargo un juego con niveles y vampiros para hacer el test.
797     El juego posee un vector con las mismas flores y los mismos vampiros y 1
            ↪ nivel.
798     El nivel posee un vampiro con 10 de vida.
799     Deberia la vida ponerse a la mitad cuando aplico el test.
800     */
801
802     ifstream is4("TestAltoCheat1");
803
804     if(is4.is_open()){
805       juegoTest.Cargar(is4);
806       is4.close();
807     }
808
809     cout << "TEST 1 ALTO CHEAT" << endl;
810     juegoTest.altoCheat(0);
811
812     if(juegoTest.nivelesJ().at(0).vampirosN().at(0).vida == 5){
813       cout << "Cumple con el test!" << endl;
814     } else {
815       cout << "No cumple con el test." << endl;
816     }
817
818     /*
819     Salida del primer test:
820
821     TEST 1 ALTO CHEAT
822     Cumple con el test!
823     */
824     cout << endl << endl;
825
826     /*
827     Test 2:
828     Cargo un juego con niveles y vampiros para hacer el test.
829     El juego posee un vector con las mismas flores y los mismos vampiros y 2
            ↪ niveles.
830     Aplico la funcion altoCheat en el segundo nivel.
831     El segundo nivel posee 2 vampiros, uno con 12 de vida y el otro con 15 de
            ↪ vida.
832     Deberia la vida de ambos ponerse a la mitad (parte entera).
833     */
834
835     ifstream is5("TestAltoCheat2");
836
```

```
837   if(is5.is_open()){
838     juegoTest.Cargar(is5);
839     is5.close();
840   }
841
842   cout << "TEST 2 ALTO CHEAT" << endl;
843   juegoTest.altoCheat(1);
844
845   if(juegoTest.nivelesJ().at(1).vampirosN().at(0).vida == 6 &&
846     juegoTest.nivelesJ().at(1).vampirosN().at(1).vida == 7){
847     cout << "Cumple con el test!" << endl;
848   } else {
849     cout << "No cumple con el test." << endl;
850   }
851
852   /*
853   Salida del segundo test:
854
855   TEST 2 ALTO CHEAT
856   Cumple con el test!
857   */
858   cout << endl << endl;
859
860   /*
861   Test 3:
862   Cargo un juego con niveles y vampiros para hacer el test.
863   El juego posee un vector con las mismas flores y los mismos vampiros y 1
        ↪ nivel.
864   El nivel posee 3 vampiros, con vidas 12, 15 y 8.
865   Deberia la vida de los tres ponerse a la mitad (parte entera).
866   */
867
868   ifstream is6("TestAltoCheat3");
869
870   if(is6.is_open()){
871     juegoTest.Cargar(is6);
872     is6.close();
873   }
874
875   cout << "TEST 3 ALTO CHEAT" << endl;
876   juegoTest.altoCheat(0);
877
878   if(juegoTest.nivelesJ().at(0).vampirosN().at(0).vida == 6 &&
879     juegoTest.nivelesJ().at(0).vampirosN().at(1).vida == 7 &&
880     juegoTest.nivelesJ().at(0).vampirosN().at(2).vida == 4){
881     cout << "Cumple con el test!" << endl;
882   } else {
883     cout << "No cumple con el test." << endl;
884   }
885
886   /*
887   Salida del tercer test:
888
889   TEST 3 ALTO CHEAT
```

```
890     Cumple con el test!
891     */
892     cout << endl << endl;
893
894     /*
895     Test 4:
896     Utilizo el juego cargado anterior pero esta vez aplico la funcion
            ↪ altoCheat en un nivel que no
897     existe.
898     La vida de los vampiros no deberia modificarse.
899     */
900
901     cout << "TEST 4 ALTO CHEAT" << endl;
902     juegoTest.altoCheat(2);
903
904     if(juegoTest.nivelesJ().at(0).vampirosN().at(0).vida == 6 &&
905       juegoTest.nivelesJ().at(0).vampirosN().at(1).vida == 7 &&
906       juegoTest.nivelesJ().at(0).vampirosN().at(2).vida == 4){
907       cout << "Cumple con el test!" << endl;
908     } else {
909       cout << "No cumple con el test." << endl;
910     }
911
912     /*
913     Salida del cuarto test:
914
915     TEST 4 ALTO CHEAT
916     Cumple con el test!
917     */
918     cout << endl << endl;
919
920
921     // TESTS MUY DE EXACTAS
922
923     /*
924     Test 1
925     Cargo un juego que posee 2 niveles sin vampiros en el juego y sin
            ↪ vampiros en la lista
926     de spawning, ambos con una flor.
927     Los dos niveles deberian ser considerados ganados y entrarian en el rango
            ↪  de fibonacci.
928     1 1 2
929     */
930
931     ifstream is7("TestMuyDeExactas1");
932
933     if(is7.is_open()){
934       juegoTest.Cargar(is7);
935       is7.close();
936     }
937
938     cout << "TEST 1 MUY DE EXACTAS" << endl;
939     if(juegoTest.muyDeExactas()){
940       cout << "Cumple!" << endl;
```

```
941     } else{
942       cout << "No cumple." << endl;
943     }
944 
945     /*
946     Salida del primer test:
947 
948     TEST 1 MUY DE EXACTAS
949     Cumple!
950     */
951     cout << endl << endl;
952 
953     /*
954     Test 2
955     Cargo un juego que posee 4 niveles sin vampiros en el juego y sin
            ↪ vampiros en la lista
956     de spawning, todos con una flor.
957     Los 4 niveles deberian ser considerados ganados y por las posiciones no
            ↪ entrarian en el
958     rango de fibonacci.
959     1 1 2 3 5
960     */
961 
962     ifstream is8("TestMuyDeExactas2");
963 
964     if(is8.is_open()){
965       juegoTest.Cargar(is8);
966       is8.close();
967     }
968 
969     cout << "TEST 2 MUY DE EXACTAS" << endl;
970     if(!juegoTest.muyDeExactas()){
971       cout << "Cumple!" << endl;
972     } else{
973       cout << "No cumple." << endl;
974     }
975 
976     /*
977     Salida del segundo test:
978 
979     TEST 2 MUY DE EXACTAS
980     Cumple!
981     */
982     cout << endl << endl;
983 
984     /*
985     Test 3
986     Cargo un juego que posee 5 niveles.
987     El primer nivel sin vampiros en espera ni en spawning.
988     El segundo nivel sin vampiros en espera ni en spawning.
989     El tercer nivel sin vampiros en espera ni en spawning.
990     El cuarto nivel con vampiros en espera y en spawning.
991     El quinto nivel sin vampiros en espera ni en spawning.
992     Los niveles sin vampiros deberian ser considerados como ganados y entran
```

```
    ↪ en el rango
993 de fibonacci.
994 1 1 2 3 5
995 */
996
997 ifstream is9("TestMuyDeExactas3");
998
999 if(is9.is_open()){
1000   juegoTest.Cargar(is9);
1001   is9.close();
1002 }
1003
1004 cout << "TEST 3 MUY DE EXACTAS" << endl;
1005 if(juegoTest.muyDeExactas()){
1006   cout << "Cumple!" << endl;
1007 } else{
1008   cout << "No cumple." << endl;
1009 }
1010
1011 /*
1012 Salida del tercer test:
1013
1014 TEST 3 MUY DE EXACTAS
1015 Cumple!
1016 */
1017 cout << endl << endl;
1018
1019 /*
1020 Test 4
1021 Cargo un juego que posee 6 niveles.
1022 Solo el cuarto nivel con vampiros en espera y en spawning.
1023 Los niveles sin vampiros deberian ser considerados como ganados y no
    ↪ entran en el rango
1024 de fibonacci.
1025 1 1 2 3 5 8
1026 */
1027
1028 ifstream is10("TestMuyDeExactas4");
1029
1030 if(is10.is_open()){
1031   juegoTest.Cargar(is10);
1032   is10.close();
1033 }
1034
1035 cout << "TEST 4 MUY DE EXACTAS" << endl;
1036 if(!juegoTest.muyDeExactas()){
1037   cout << "Cumple!" << endl;
1038 } else {
1039   cout << "No cumple." << endl;
1040 }
1041
1042 /*
1043 Salida del cuarto test:
1044
```

```
1045    TEST 4 MUY DE EXACTAS
1046    Cumple!
1047    */
1048    cout << endl;
1049 }
```

TestAltoCheat1

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 12 12 3 100 [ ( { F 1 1 [ Atacar
    ↪ Explotar Generar ] } ( 1 1 ) 10 ) ] [ ( { V Desviado 15 15 } ( 1 2 )
    ↪ 10 ) ] [ ( { V Caminante 10 10 } 2 5 ) ] } ] }
```

TestAltoCheat2

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 12 12 3 100 [ ( { F 1 1 [ Atacar
    ↪ Explotar Generar ] } ( 1 1 ) 10 ) ] [ ( { V Desviado 15 15 } ( 1 2 )
    ↪ 10 ) ] [ ( { V Caminante 10 10 } 2 5 ) ] } { N 8 8 1 100 [ ( { F 1 1
    ↪ [ Generar ] } ( 3 3 ) 5 ) ( { F 1 1 [ Atacar Explotar ] } ( 1 1 ) 10
    ↪ ) ] [ ( { V Caminante 12 12 } ( 1 2 ) 12 ) ( { V Desviado 15 15 } ( 3
    ↪  3 ) 15 ) ( { V Desviado 15 15 } ( 3 3 ) 15 ) ] [ ( { V Caminante 12
    ↪ 12 } 2 5 ) ] } ] }
```

TestAltoCheat3

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 8 8 1 100 [ ( { F 1 1 [ Generar ] } ( 3
    ↪ 3 ) 5 ) ( { F 1 1 [ Atacar Explotar ] } ( 1 1 ) 10 ) ] [ ( { V
    ↪ Caminante 12 12 } ( 1 2 ) 12 ) ( { V Desviado 15 15 } ( 3 3 ) 15 ) (
    ↪ { V Caminante 10 10 } ( 2 4 ) 8 ) ] [ ( { V Caminante 12 12 } 2 5 ) ]
    ↪  } ] }
```

TestEstosSaleFacil2

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 12 12 3 100 [ ( { F 1 1 [ Atacar
    ↪ Explotar Generar ] } ( 1 1 ) 10 ) ] [ ( { V Desviado 15 15 } ( 1 2 )
    ↪ 10 ) ] [ ( { V Caminante 10 10 } 2 5 ) ] } { N 9 9 2 100 [ ( { F 1 1
    ↪ [ Atacar Explotar ] } ( 1 1 ) 10 ) ] [ ( { V Caminante 12 12 } ( 1 2
    ↪ ) 10 ) ] [ ( { V Caminante 12 12 } 2 5 ) ] } ] }
```

TestEstosSaleFacil3

```
{ J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } { F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 } { V Desviado 15 15 } ] [ { N 12 12 3 100 [ ( { F 1 1 [ Atacar Explotar Generar ] } ( 1 1 ) 10 ) ] [ ( { V Desviado 15 15 } ( 1 2 ) 10 ) ] [ ( { V Caminante 10 10 } 2 5 ) ] } { N 9 9 2 100 [ ( { F 1 1 [ Atacar Explotar ] } ( 1 1 ) 10 ) ] [ ( { V Caminante 12 12 } ( 1 2 ) 10 ) ] [ ( { V Caminante 12 12 } 2 5 ) ] } { N 10 10 6 100 [ ( { F 1 1 [ Generar ] } ( 3 3 ) 5 ) ( { F 1 1 [ Atacar Explotar ] } ( 1 1 ) 10 ) ] [ ( { V Caminante 12 12 } ( 1 2 ) 10 ) ] [ ( { V Caminante 12 12 } 2 5 ) ] } ] }
```

TestEstosSaleFacil4

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 12 12 3 100 [ ( { F 1 1 [ Atacar
    ↪ Explotar Generar ] } ( 1 1 ) 10 ) ] [ ( { V Desviado 15 15 } ( 1 2 )
    ↪ 10 ) ] [ ( { V Caminante 10 10 } 2 5 ) ] } { N 9 9 2 100 [ ( { F 1 1
    ↪ [ Atacar Explotar ] } ( 1 1 ) 10 ) ] [ ( { V Caminante 12 12 } ( 1 2
    ↪ ) 10 ) ] [ ( { V Caminante 12 12 } 2 5 ) ] } { N 10 10 6 100 [ ( { F
    ↪ 1 1 [ Generar ] } ( 3 3 ) 5 ) ( { F 1 1 [ Atacar Explotar ] } ( 1 1 )
    ↪  10 ) ] [ ( { V Caminante 12 12 } ( 1 2 ) 10 ) ] [ ( { V Caminante 12
    ↪  12 } 2 5 ) ] } { N 8 8 1 100 [ ( { F 1 1 [ Generar ] } ( 3 3 ) 5 ) (
    ↪  { F 1 1 [ Atacar Explotar ] } ( 1 1 ) 10 ) ] [ ( { V Caminante 12 12
    ↪  } ( 1 2 ) 10 ) ] [ ( { V Caminante 12 12 } 2 5 ) ] } ] }
```

TestMuyDeExactas1

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 9 9 1 11 [ ( { F 1 1 [ Atacar Explotar ]
    ↪  } ( 1 1 ) 15 ) ] [ ] [ ] } { N 9 9 2 22 [ ( { F 1 1 [ Atacar
    ↪ Explotar ] } ( 2 2 ) 10 ) ] [ ] [ ] } ] }
```

TestMuyDeExactas2

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 9 9 1 11 [ ( { F 1 1 [ Atacar Explotar ]
    ↪  } ( 1 1 ) 15 ) ] [ ] [ ] } { N 9 9 2 22 [ ( { F 1 1 [ Atacar
    ↪ Explotar ] } ( 2 2 ) 10 ) ] [ ] [ ] } { N 9 9 3 33 [ ( { F 1 1 [
    ↪ Atacar Explotar ] } ( 3 3 ) 10 ) ] [ ] [ ] } { N 9 9 4 44 [ ( { F 1 1
    ↪  [ Atacar Explotar ] } ( 4 4 ) 10 ) ] [ ] [ ] } ] }
```

TestMuyDeExactas3

```
1 { J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } {
    ↪  F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 }
    ↪ { V Desviado 15 15 } ] [ { N 9 9 1 11 [ ( { F 1 1 [ Atacar Explotar ]
    ↪  } ( 1 1 ) 15 ) ] [ ] [ ] } { N 9 9 2 22 [ ( { F 1 1 [ Atacar
    ↪ Explotar ] } ( 2 2 ) 10 ) ] [ ] [ ] } { N 9 9 3 33 [ ( { F 1 1 [
    ↪ Atacar Explotar ] } ( 3 3 ) 10 ) ] [ ] [ ] } { N 9 9 4 44 [ ( { F 1 1
    ↪  [ Atacar Explotar ] } ( 4 4 ) 10 ) ] [ ( { V Desviado 50 8 } ( 1 2 )
    ↪  10 ) ] [ ( { V Caminante 30 7 } 2 5 ) ] } { N 9 9 5 55 [ ( { F 1 1 [
    ↪  Atacar Explotar ] } ( 3 3 ) 10 ) ] [ ] [ ] } ] }
```

TestMuyDeExactas4

```
{ J [ { F 1 1 [ Atacar Explotar Generar ] } { F 1 1 [ Atacar Explotar ] } { F 1 1 [ Generar ] } ] [ { V Caminante 10 10 } { V Caminante 12 12 } { V Desviado 15 15 } ] [ { N 9 9 1 11 [ ( { F 1 1 [ Atacar Explotar ] } ( 1 1 ) 15 ) ] [ ] [ ] } { N 9 9 2 22 [ ( { F 1 1 [ Atacar Explotar ] } ( 2 2 ) 10 ) ] [ ] [ ] } { N 9 9 3 33 [ ( { F 1 1 [ Atacar Explotar ] } ( 3 3 ) 10 ) ] [ ] [ ] } { N 9 9 4 44 [ ( { F 1 1 [ Atacar Explotar ] } ( 4 4 ) 10 ) ] [ ( { V Desviado 50 8 } ( 1 2 ) 10 ) ] [ ( { V Caminante 30 7 } 2 5 ) ] } { N 9 9 5 55 [ ( { F 1 1 [ Atacar Explotar ] } ( 4 4 ) 10 ) ] [ ] [ ] } { N 9 9 6 66 [ ( { F 1 1 [ Atacar Explotar ] } ( 3 3 ) 10 ) ] [ ] [ ] } ] }
```