Reporte del Laboratorio 0
Juego de la vida

Luis Diego Fernández Coto - B22492
Daniel Jiménez León - B23467

1 de febrero de 2017

Índice

1. Enunciado

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

`IE-0217` Estructuras abstractas de datos y algoritmos para ingeniería

Laboratorio 0: Juego de ecología en C++

M. Sc. Ricardo Román Brenes - `ricardo.roman@ucr.ac.cr`

III-2016

Tabla de contenidos

---

1. Enunciado

Diseñe, implemente y documente una simulación simple de ecológica.

La simulación se llevará acabo en una tabla de `Campos` de $N \times M$. En cada `Campo` habrá una cantidad de `energía` y puede o no haber un `Animal`. La simulación correrá por una cantidad de terminada de `días`.

Los `Animales` pueden ser uno de cuatro tipos: `Lobos`, `Ovejas`, `Zorros`, `Ratones`.

Cada `Animal` tiene, al menos, una cantidad de energía, un identificador, un sexo, una especie y su ubicación.

Cada `Animal` puede realizar alguno de las siguientes cuatro acciones: `mover`, `comer`, `reproducir`, `morir`. Estos comportamientos se verán reflejados por los operadores **unarios** !, ++, ~ y −− respectivamente.

Los `Animales` tiene diferentes velocidades; al día, un Lobo puede moverse 3 veces, un Oveja y un Zorro pueden moverse 2 veces y un Ratón puede moverse 1 vez.

Cada día que pase, todos los animales deben moverse, comer, reproducirse y morir, en caso **de ser posible** y deben hacer cada acción **exactamente una vez al día** y **en ese orden**. Por ejemplo si un lobo se encuentra en una casilla rodeado por lobas, este solo se reproducirá una vez y dicha loba no se reproducirá más por ese día.

Los `Animales` y el `Campo` interactúan bajo las siguientes reglas (considere como adyacentes a un Campo la vecindad de Moore):

1. Los Lobos inician con una energía de 100.
2. Las Ovejas inician con una energía de 75.
3. Los Zorros inician con una energía de 50.
4. Los Ratones inician con una energía de 25.
5. Al final de cada día, los Animales pierden 1 punto de energía.
6. Al cabo de 3 días, cada Campo que contenga una cantidad positiva de energía ganará 5 puntos de energía.
7. Un Campo no puede acumular más de 100 puntos de energía.
8. Un Campo con energía 0 no gana más puntos de energía.
9. Una Oveja come energía de un Campo, consume 10 puntos del Campo y recupera la misma cantidad.
10. Un Ratón come energía de un Campo, consume 5 puntos del Campo y recupera la misma cantidad.
11. Un Zorro come energía de un Ratón, lo consume por completo y recupera 2 puntos de energía.
12. Un Lobo come energía de cualquier otro animal, lo consume por completo y recupera 10 puntos por cada Oveja, 5 por cada Zorro y 2 por cada Ratón.
13. Un Lobo macho que tenga en un Campo adyacente a otro Lobo macho causa que alguno de los dos muera.
14. Un Lobo que tenga en un Campo adyacente a cualquier otro Animal menos un Lobo causa que el Lobo consuma al Animal.
15. Un Zorro que tenga en un Campo adyacente a un Ratón causa que el Zorro consuma al Ratón.
16. Un Animal que tenga en un Campo adyacente a otro Animal de la misma especie pero de sexo opuesto y contando con otro Campo adyacente desocupado, se reproducirá y genera un nuevo Animal de la misma especie de cualquier sexo.

El programa que genere recibirá por argumento la cantidad de días por los que se ejecutará la simulación y la ruta de un archivo de configuración que proporcionará la configuración inicial del juego.

Este archivo contiene la siguiente información:

```
3                   # M
3                   # N
0   0   25  LH      # celda (0,0); 25e; lobo hembra
0   1   75  CM      # celda (0,1); 75e; conejo macho
0   2   0           # celda (0,2); 00e; sin animales
1   0   100         # ...
1   1   50  LM      # ...
1   2   25  CH      # ...
2   0   75  OM      # ...
2   1   0   OM      # ...
2   2   0   OH      # ...
```

El anterior archivo representaría el siguiente mapa:

	0		1		2	
0	A:	LH	A:	CM	A:	
	en:	25	en:	75	en:	0
1	A:		A:	LM	A:	CH
	en:	100	en:	50	en:	25
2	A:	OM	A:	OM	A:	OH
	en:	75	en:	0	en:	0

Mapa de ejemplo.

Al inicio y al final de la simulación es necesario que imprima en pantalla un reporte de los animales disponibles en el mapa. Para esto construya una función emplantillada que reporte, el identificador, la especie, el sexo y las coordenadas del Animal.

En cada día de ejecución de la simulación imprima el estado del juego de una forma clara en pantalla.

Documente su trabajo utilizando la herramienta *doxygen* y cree un diagrama de clases UML de su trabajo; así como los archivos README e INSTALL.

Utilice un Makefile para facilitar la construcción y prueba de su programa.

Recuerde utilizar todas las capacidades disponibles en C++: programación orientada a objetos, herencia, polimorfismo, memoria dinámica, sobrecarga de métodos y operadores, plantillas; es el objetivo de este laboratorio.

2. Consideraciones

- Haga grupos de 2 personas.
- Genere un reporte en LaTeX que incluya su código, su abordaje para la solución, el diagrama de clases, y sus conclusiones.
- Suba su código y documentación (doxygen, README, INSTALL) al GitHub respectivo de su grupo y el directorio del laboratorio.
- Cada estudiante debe subir el reporte a Schoology. (`https://app.schoology.com/assignment/949072914/`).
- Recuerde que por cada día tardío de entrega se le rebajaran puntos de acuerdo con la formula: 3^d, donde $d > 1$ es la cantidad de días tardíos.

2. Código

2.1. main.cpp

```
#include <stdlib.h>

#include "Controlador.h"

//se reciben 2 argumentos:
//el primero la cantidad de dias que van a pasar
//el segundo el nombre del archivo del cual se van a leer los datos
int main(int argc, char *argv[]) {
    int amountOfDays = atoi(argv[1]); //argv[] recibe un char* entonces
        hay que pasarlo a int

    //la clase controlador va a manejar el juego
    Controlador controlador;

    //se llama a la funcion run del controlador con los parametros
       respectivos
    return controlador.run(amountOfDays, argv[2]);
}
```

2.2. Controlador.h

```
#ifndef CONTROLADOR_H
#define CONTROLADOR_H

#include <fstream>
#include <iostream>

#include "Animal.h"
#include "Celda.h"
#include "Lobo.h"
#include "Oveja.h"
#include "Raton.h"
#include "Zorro.h"

using namespace std;

class Controlador {
//atributos
public:
    static int columns;
    static int rows;
    static Celda*** terreno;

//Metodos
```

```
public:
    Controlador();
    virtual ~Controlador();
    void printTerreno(int columns, int rows, Celda*** terreno);
    void resetMarks(int columns, int rows, Celda*** terreno); //resetea
        el atributo alreadyReproduced al final de cada dia
    int run(int amountOfDays, char* fileName); //corre el juego
};

#endif // CONTROLADOR_H
```

2.3. Controlador.cpp

```
/**
*@file Controlador.cpp
*@version 1.0
*@date 29/01/17
*@author Luis Diego Fernandez, Daniel Jimenez
*@title Juego de la vida
*@brief Clase Controlador
*/

#include "Controlador.h"

int Controlador::columns;
int Controlador::rows;
Celda*** Controlador::terreno;

/*! \brief Constructor por defecto.
 */
Controlador::Controlador() {

}

/*! \brief Destructor.
 */
Controlador::~Controlador() {

}
/*! \brief funcion de impresion en plantilla
 * \param objeto es un objeto de cualquier tipo
 */
//funcion plantilla
template <typename DataType>
void printTemplate (DataType* objeto) {
    cout << objeto << endl;
    cout << "Posicion: [" << objeto->Fila << ", " << objeto->Columna <<
        "]" << endl;
```

```
    cout << "Animal: " << objeto->tipoAnimal << endl;
    if (objeto->Sexo == 1)
        cout << "Sexo: macho" << endl;
    else
        cout << "Sexo: hembra" << endl;
    cout << "Energia: " << objeto->Energia << "\n" << endl;
}

/*! \brief funcion de impresion en plantilla especifico para una celda
 */
template<>
void printTemplate <Celda> (Celda* celda) {
    cout << "Posicion: [" << celda->fila << ", " << celda->columna << "
       ]" << endl;
    cout << "Nivel zacate: " << celda->zacate << endl;
    cout << "Ocupante: " << celda->ocupante << endl;
}

/*! \brief Metodo que imprime el estado del sistema (terreno)
 *
 *  \param columns Posicion en la cual se desea crear el animal y
    establecer los datos a la celda.
 *  \param rows Posicion en la cual se desea crear el animal y
    establecer los datos a la celda.
 *  \param terreno La matriz de celdas con el cual esta basado el juego
 */
void Controlador::printTerreno(int columns, int rows, Celda ***terreno)
    {
    for (int colIndex = 0; colIndex < columns; ++colIndex)
        for (int rowIndex = 0; rowIndex < rows; ++rowIndex)
            terreno[colIndex][rowIndex]->print();
}

/*! \brief Metodo que ejecuta la logica del juego.
 *
 *  \param animal Almacenar temporalmente el animal que se lee del
    archivo de texto.
 *  \param columns Posicion en la cual se desea crear el animal y
    establecer los datos a la celda.
 *  \param rows Posicion en la cual se desea crear el animal y
    establecer los datos a la celda.
 *  \param zacate Nivel de zacate en la celda.
 *  \param line Variable de utilidad a la hora de leer el archivo de
    datos.
 *  \param posicionColumna Variable para recorrer el terreno.
 *  \param posicionFila Variable para recorrer el terreno.
*/
int Controlador::run(int amountOfDays, char* fileName) {
```

```
cout << "Bienvenido al  Juego de la Vida! \nMay the odds be in
    your favor..!\n" << endl;

//declaracion de variables de interes
//int amountOfDays = 5; //esta se utilizaba antes de pasarla como
    argumento, para pruebas
string animal; //Almacenar temporalmente el animal que se lee del
    archivo de texto.
int zacate = 0; //Posicion en la cual se desea crear el animal y
    datos de la celda.
string line; //Variable de utilidad a la hora de leer el archivo de
     datos.
int posicionColumna = 0, posicionFila = 0; //Variables para
    recorrer el terreno.

//se abre el archivo donde esta el estado inicial
ifstream dataFile;
dataFile.open(fileName);

//se extraen la cantidad de columnas y filas del archivo
//estas estan en las primeras 2 lineas
if (dataFile.is_open()) {
     dataFile >> columns;
     dataFile >> rows;
}
//se crea una matriz de objetos tipo Celda
terreno = new Celda**[columns];
for (int index = 0; index < columns; ++index) {
     terreno[index] = new Celda*[rows];
}

cout << "Estado inicial (crecion de la Tierra): " << endl;

//se llena la matriz con inicializaciones de celdas con sus
    respectivos datos
//el estado inicial
for (int colIndex = 0; colIndex < columns; ++colIndex) {
     for (int rowIndex = 0; rowIndex < rows; ++rowIndex) {
          //las siguientes lineas obtienen valores del archivo de
              datos
          dataFile >> posicionColumna;
          dataFile >> posicionFila;
          dataFile >> zacate;
          getline(dataFile, animal);
          if (animal == " ")
               animal = "Vacio";
          //se crea la celda con los datos del archivo de datos
          terreno[posicionColumna][posicionFila] = new Celda(zacate,
```

```
            animal, posicionColumna, posicionFila);
        //se imprime cada celda como un estado inicial del terreno
        terreno[posicionColumna][posicionFila]->print();
    }
}

dataFile.close(); //se cierra el archivo ya que no se ocupa mas

//EMPEZAMOS A CORRER LOS DIAS
for (int daysIndex = 1; daysIndex <= amountOfDays; ++daysIndex) {
    //imprimo el estado del sistema al inicio del dia
    cout << "\nEstado al inicio del dia " << daysIndex << ": \n" <<
        endl;
    printTerreno(columns, rows, terreno);

    //Para estar seguros que cada dia se apliquen todos los metodos
        sobre los animales recorremos la matriz dos veces
    for (int banderaControl = 0; banderaControl < 2; ++
       banderaControl) {
        for (int colIndex = 0; colIndex < columns; ++colIndex) {
            for (int rowIndex = 0; rowIndex < rows; ++rowIndex) {
                //cada 3 dias el terreno gana 5 de energia
                if(banderaControl == 1) {
                    if (terreno[colIndex][rowIndex]->zacate > 0 &&
                       daysIndex %3 == 0) {
                        if (terreno[colIndex][rowIndex]->zacate <=
                           95)
                            terreno[colIndex][rowIndex]->zacate +=
                               5;
                        else {
                            //si el terreno tiene mas de 95 de
                               energia solo se recupera a 100
                            if (terreno[colIndex][rowIndex]->zacate
                                > 95)
                                terreno[colIndex][rowIndex]->zacate
                                    = 100;
                        }
                    }
                }
                //si hay algun animal en el terreno, ejecutamos las
                    acciones de los animales
                if (terreno[colIndex][rowIndex]->ocupante != "Vacio
                   ") { //revisa que el campo tenga un animal
                    //Funcion mover
                    !(*terreno[colIndex][rowIndex]->animal);
                    //revisa nuevamente que el campo tenga un
                       animal y que no se hayan aplicado los
                       metodos sobre el
```

```
                    if (terreno[colIndex][rowIndex]->ocupante != "
                        Vacio" && terreno[colIndex][rowIndex]->
                        animal->allFunctions == false) {
                        //Funcion Comer
                        ++(*terreno[colIndex][rowIndex]->animal);
                        //Funcion reproducir
                        ~(*terreno[colIndex][rowIndex]->animal);
                        //al final de cada dia los animales pierden
                            1 de energia
                        terreno[colIndex][rowIndex]->animal->
                            Energia -= 1;
                        //si el animal perdio energia y llego a
                            cero, se muere
                        //Funcion morir sobrecargada con el operado
                            --
                        if(--(*terreno[colIndex][rowIndex]->animal)
                            ){
                            terreno[colIndex][rowIndex]->ocupante =
                                "Vacio";
                            delete terreno[colIndex][rowIndex]->
                                animal;
                        }
                        //Inico que al animal se le aplicaron todos
                            los metodos
                        terreno[colIndex][rowIndex]->animal->
                            allFunctions = true;
                    }
                }
            }
        }
    }

    resetMarks(columns, rows, terreno); //se resetean las marcas de
        reproduccion cada dia

    //imprimo el estado del sistema al final del dia
    cout << "Estado al final del dia " << daysIndex << ":\n" <<
        endl;
    printTerreno(columns, rows, terreno);
}
cout << endl;


//al acabar la ejecucion del juego, al pasar todos los dias hay que
    liberar toda la memoria dinamica
//que se utilizo. En este caso hay que liberar todos los animales y
    las celdas
for (int colIndex = 0; colIndex < columns; ++colIndex) {
```

```
        for (int rowIndex = 0; rowIndex < rows; ++rowIndex) {
            if(terreno[colIndex][rowIndex]->ocupante != "Vacio")
                delete terreno[colIndex][rowIndex]->animal; //libero
                    cada objeto animal que cree
            delete terreno[colIndex][rowIndex]; //libero cada celda que
                se creo
        }
        delete terreno[colIndex];//libero cada puntero dentro del
            vector de punteros
    }
    delete terreno; //libero el vector de punteros a celda que se creo

    cout << "Finalizacion del programa..!" << endl;


    /*Lobo* l1 = new Lobo(1, 1, 1);
    Zorro* z1 = new Zorro(5, 3, 2);
    Celda* c1 = new Celda(33, "Cachorro de perro", 6, 9);
    printTemplate(l1);
    printTemplate(z1);
    printTemplate<Celda>(c1); */

    return 0;
}

/// \brief Metodo para resetear banderas de control. Cada animal tiene
    una marca que dice si ya se reprodujo, se movio
///        y aplicaron demas metodos cada dia, entonces al finalizar
    cada dia se resetean las marcas de todos los animales
///        para que al dia siguiente puedan ser aplicadas nuevamente.
void Controlador::resetMarks(int columns, int rows, Celda*** terreno) {
    for (int colIndex = 0; colIndex < columns; ++colIndex) {
        for (int rowIndex = 0; rowIndex < rows; ++rowIndex) {
            if (terreno[colIndex][rowIndex]->ocupante.compare("Vacio")
                != 0) {
                terreno[colIndex][rowIndex]->animal->allFunctions =
                    false;
                terreno[colIndex][rowIndex]->animal->alreadyMoved =
                    false;
                terreno[colIndex][rowIndex]->animal->alreadyReproduced
                    = false;
            }
        }
    }
}
```

2.4. Celda.h

```
#ifndef CELDA_H
#define CELDA_H

#include <iostream>

#include "Lobo.h"
#include "Oveja.h"
#include "Zorro.h"
#include "Raton.h"

using namespace std;

class Animal;

class Celda {

//Atributos
public:
    ///Animal que se encuetra en la celda.
    Animal* animal;
    ///Posicion de la celda.
    int columna, fila;
    ///Tipo de animal que ocupa la celda.
    string ocupante;
    ///Cantidad de zacate presente en la celda.
    int zacate;

//Metodos
public:
    Celda();
    Celda(int cantidadZacate, string tipoOcupante, int columna, int
        fila);
    virtual ~Celda();

    void print();
};

#endif // CELDA_H
```

2.5. Celda.cpp

```
/**
*@file Celda.cpp
*@version 1.0
*@date 29/01/17
*@author Luis Diego Fernandez, Daniel Jimenez
*@title Juego de la vida
*@brief Clase Celda
```

```
*/

#include "Celda.h"
#include "Animal.h"

/*! \brief Constructor por defecto.
 */
Celda::Celda() {

}

/*! \brief Constructor para crear Celda con datos especificos.
 */
Celda::Celda(int cantidadZacate, string tipoOcupante, int fila, int
    columna) {
    zacate = cantidadZacate;
    ocupante = tipoOcupante;
    this->fila = fila;
    this->columna = columna;

    //dependiendo del tipo de animal que diga el archivo de datos
    //ese tipo de animal es creado
    if(ocupante.compare(" LM") == 0)
        animal = new Lobo(fila, columna, 1);
    else if(ocupante.compare(" LH") == 0)
        animal = new Lobo(fila, columna, 2);
    else if(ocupante.compare(" OM") == 0)
        animal = new Oveja(fila, columna, 1);
    else if(ocupante.compare(" OH") == 0)
        animal = new Oveja(fila, columna, 2);
    else if(ocupante.compare(" ZM") == 0)
        animal = new Zorro(fila, columna, 1);
    else if(ocupante.compare(" ZH") == 0)
        animal = new Zorro(fila, columna, 2);
    else if(ocupante.compare(" RM") == 0)
        animal = new Raton(fila, columna, 1);
    else if(ocupante.compare(" RH") == 0)
        animal = new Raton(fila, columna, 2);
}


/*! \brief Destructor.
 */
Celda::~Celda() {

}

///Se imprime toda la informacion relevane de una celda y de el animal
```

```
    que esta dentro de ella.
void Celda::print() {
    cout << "Posicion: [" << fila << ", " << columna << "]" << endl;
    cout << "Nivel zacate: " << zacate << endl;
    cout << "Ocupante: " << ocupante << endl;

    //si la celda no tiene una animal dentro entonces no se imprime la
        informacion del animal
    if (ocupante != "Vacio")
        animal->Print();
    cout << endl;
}
```

2.6. Animal.h

```
#ifndef ANIMAL_H
#define ANIMAL_H

#include <iostream>

using namespace std;

class Celda;

class Animal {

//Atributos
public:
    ///Bandera para saber si a un animal ya se le aplicaron todos los
        metodos.
    bool allFunctions;
    ///Bandera para saber si un animal ya se movio en un dia.
    bool alreadyMoved;
    ///Bandera para saber si un animal ya se repdodujo en un dia.
    bool alreadyReproduced;
    ///Energia del aimal.
    int Energia;
    ///Posicion del animal.
    int Fila, Columna;
    ///Sexo del animal.
    int Sexo;
    ///Se define en las clases hijo, dependiendo de que tipo de animal
        que sea.
    string tipoAnimal;

//Metodos
public:
    Animal();
```

```
    virtual ~Animal();

    virtual int operator!() = 0; //metodo mover
    virtual int operator++() = 0; //metodo de comer
    virtual void operator~() = 0; //metodo de reproducir
    bool operator--(); // metodo de morir
    void Print();

};

#endif // ANIMAL_H
```

2.7. Animal.cpp

```
/**
*@file Animal.cpp
*@version 1.0
*@date 29/01/17
*@author Luis Diego Fernandez, Daniel Jimenez
*@title Juego de la vida
*@brief Clase Animal
*/

#include "Animal.h"
#include "Celda.h"

/*! \brief Constructor por defecto.
 */
Animal::Animal() {
    //por defecto a un animal no se le va a haber aplicado ningun
       metodo al crearse
    allFunctions = false;
    alreadyMoved = false;
    alreadyReproduced = false;
}

/*! \brief Destructor.
 */
Animal::~Animal(){

}

/*! \brief Metodo para determinar si un animal debe morir.
 */
bool Animal::operator--() {
    //si un animal se queda sin energia se muere
    if(Energia == 0){
        cout << "Murio: " << this << endl; //mensaje de muerte de un
```

```
            animal
        return 1; //al retornar 1 se le hace delete al animal por fuera
    }
    return 0;
}

/// \brief Esta funcion se llama dentro de la funcion imprimir de celda
///        ya que un atributo de una celda es un animal
void Animal::Print() {
    cout << "Energia: " << Energia << endl;
    if(Sexo == 1)
        cout << "Sexo: Macho" << endl;
    else
        cout << "Sexo: Hembra" << endl;
}
```

2.8. Lobo.h

```
#ifndef LOBO_H
#define LOBO_H

#include <cstdlib>

#include "Animal.h"
#include "Celda.h"
#include "Controlador.h"

class Lobo : public Animal {

//Metodos
public:
    Lobo();
    Lobo(int Fila, int Columna, int Sexo);
    virtual ~Lobo();

    int operator!(); //metodo mover
    int operator++(); //metodo de comer
    void operator~(); //metodo reproducir
    //void PrintLobo();
};

#endif // LOBO_H
```

2.9. Lobo.cpp

```
/**
*@file Lobo.cpp
*@version 1.0
*@date 29/01/17
```

```
*@author Luis Diego Fernandez, Daniel Jimenez
*@title Juego de la vida
*@brief Clase Lobo
*/

#include "Lobo.h"

/*! \brief Constructor por defecto.
 */
Lobo::Lobo() {

}

/*! \brief Constructor para crear un Lobo con datos especificos.
 */
Lobo::Lobo(int Fila, int Columna, int Sexo) {
    this->Columna = Columna;
    Energia = 100;
    this->Fila = Fila;
    this->Sexo = Sexo;
    tipoAnimal = "Lobo";
}

/*! \brief Destructor.
 */
Lobo::~Lobo() {

}

/*! \brief Metodo para que el Lobo se mueva en el terreno. Es capaz de
   moverse
 *         hasta tres espacios en el terrreno siempre que estan
    desocupados y
 *         sean aledannos.
 *
 *  \param xActual Posicion actual del animal.
 *  \param yActual Posicion actual del animal.
 *  \param xPrevio Posicion previa del animal.
 *  \param yPrevio Posicion previa del animal.
 *  \param contador Numero de veces que se ha desplzado el animal.
 */
int Lobo::operator!() {
    int xActual = this->Columna; //Posicion actual del animal.
    int yActual = this->Fila;
    int yPrevio, xPrevio; //Posicion previa del animal.
    int contador = 0; //Numero de veces que se ha desplazado el animal.
    int temp;
```

```
//Verifico que el animal no se haya movido.
if(Controlador::terreno[this->Fila][this->Columna]->animal->
    alreadyMoved == false) {
     //Si el animal se mueve se deben empezar de nuevo los ciclos
        for por lo tanto se utiliza este ciclo while para ello.
     while (temp) {
         temp = 0;
         //Se busca si en las posiciones aledannas si hay algun
            espacio libre en donde se pueda mover el animal.
         //Este doble ciclo for esta configurado de manera que se
            busca alrededor de la celda en la cual se esta
         //evita tambien salirse de la matriz para no dar errores de
             segmentacion y evita utilizarse a si misma.
         for (int xpos = xActual-1; xpos <= xActual+1; ++xpos) {
             for (int ypos = yActual-1; ypos <= yActual+1; ++ypos) {
                 if (!(xpos == xActual && ypos == yActual)) { //no
                    se mete en si mismo
                     if ((xpos >=  0 && xpos < Controlador::columns)
                         && (ypos >=  0 && ypos < Controlador::rows)
                        )
                         //Si la celda no tiene animal y ademas no
                             es la posicion previa en la cual estuvo,
                              se mueve.
                         if((Controlador::terreno[ypos][xpos]->
                            ocupante.compare("Vacio") == 0) && ypos
                            != yPrevio && xpos != xPrevio){
                             //Creo el nuevo animal con las mismas
                                 caracteristicas que el original.
                             Controlador::terreno[ypos][xpos]->
                                 animal = new Lobo(ypos, xpos,
                                 Controlador::terreno[yActual][
                                 xActual]->animal->Sexo);
                             Controlador::terreno[ypos][xpos]->
                                 animal->Energia = Controlador::
                                 terreno[yActual][xActual]->animal->
                                 Energia;
                             Controlador::terreno[ypos][xpos]->
                                 ocupante = Controlador::terreno[
                                 yActual][xActual]->ocupante;
                             //Indico que ya se movio.
                             Controlador::terreno[ypos][xpos]->
                                 animal->alreadyMoved = true;
                             //Elimino el animal de la posicion de
                                 la cual se esta desplazando para
                                 dejar la celda vacia.
                             delete Controlador::terreno[yActual][
                                 xActual]->animal;
                             Controlador::terreno[yActual][xActual
```

```
                                        ]->ocupante = "Vacio";
                                    //Actualizo variables de control.
                                    yPrevio = yActual;
                                    xPrevio = xActual;
                                    yActual = ypos;
                                    xActual = xpos;
                                    contador += 1;
                                    temp = 1;
                                    if(contador == 3)
                                        return 0;
                                }
                            }
                        }
                    }
                }

        }
        return 0;
}

/// \brief Metodo para comer. Se alimenta de cualquier otro animal que
    esten en posiciones aledannas. Recive 10 puntos
///         por Ovejas, 5 puntos por Zorros y 2 por ratones. Mata al
    otro animal. No puede excederse de 100 la energia
///         del Zorro. Si hay dos lobos machos alguno de los dos muere.
int Lobo::operator++() {
    //Se busca si en las posiciones aledannas hay un Animal de
        cualquier especie para eliminarlo y subir la energia del Lobo
    //este doble ciclo for esta configurado de manera que se busca
        alrededor de la celda en la cual se esta
    //evita tambien salirse de la matriz para no dar errores de
        segmentacion y evita utilizarse a si misma
    for (int xpos = this->Columna-1; xpos <= this->Columna+1; ++xpos) {
        for (int ypos = this->Fila-1; ypos <= this->Fila+1; ++ypos) {
            if (!(xpos == this->Columna && ypos == this->Fila)) { //no
                se mete en si mismo
                if ((xpos >=  0 && xpos < Controlador::columns) && (
                    ypos >=  0 && ypos < Controlador::rows)){
                    //Verifico cual tipo de animal es porque la
                        cantidad de puntos que recupera el lobo son
                        diferentes
                    if(Controlador::terreno[ypos][xpos]->ocupante.
                        compare("OM") == 0 || Controlador::terreno[
                        ypos][xpos]->ocupante.compare("OH") == 0) {
                        delete Controlador::terreno[ypos][xpos]->animal
                            ; //Mato al animal.
                        Controlador::terreno[ypos][xpos]->ocupante = "
                            Vacio"; //Asigno vacia la celda.
```

```
        Controlador::terreno[this->Fila][this->Columna
            ]->animal->Energia += 10; //Aumento energia
            del Lobo.
        //Verifico que no se pase de 100 la energia.
        if(Controlador::terreno[this->Fila][this->
            Columna]->animal->Energia > 100)
             Controlador::terreno[this->Fila][this->
                 Columna]->animal->Energia = 100;
        return 0;
    }else if(Controlador::terreno[ypos][xpos]->ocupante
        .compare(" ZM") == 0 || Controlador::terreno[
        ypos][xpos]->ocupante.compare(" ZH") == 0) {
        delete Controlador::terreno[ypos][xpos]->animal
            ;
        Controlador::terreno[ypos][xpos]->ocupante = "
            Vacio";
        Controlador::terreno[this->Fila][this->Columna
            ]->animal->Energia += 5;
        if(Controlador::terreno[this->Fila][this->
            Columna]->animal->Energia > 100)
             Controlador::terreno[this->Fila][this->
                 Columna]->animal->Energia = 100;
        return 0;
    }else if(Controlador::terreno[ypos][xpos]->ocupante
        .compare(" RM") == 0 || Controlador::terreno[
        ypos][xpos]->ocupante.compare(" RH") == 0) {
        delete Controlador::terreno[ypos][xpos]->animal
            ;
        Controlador::terreno[ypos][xpos]->ocupante = "
            Vacio";
        Controlador::terreno[this->Fila][this->Columna
            ]->animal->Energia += 2;
        if(Controlador::terreno[this->Fila][this->
            Columna]->animal->Energia > 100)
             Controlador::terreno[this->Fila][this->
                 Columna]->animal->Energia = 100;
        return 0;
    //Si se encuentra otro Lobo macho utilizo una
        funcion para generar un numero aleatorio y asi
        decidir cual muere.
    }else if(Controlador::terreno[ypos][xpos]->ocupante
        .compare(" LM") == 0 && Controlador::terreno[
        this->Fila][this->Columna]->ocupante.compare(" 
        LM") == 0){
        if((rand() % 10) < 5){
             delete Controlador::terreno[ypos][xpos]->
                 animal;
             Controlador::terreno[ypos][xpos]->ocupante
```

```
                                = "Vacio";
                            return 0;
                        }else{
                            delete Controlador::terreno[this->Fila][
                                this->Columna]->animal;
                            Controlador::terreno[this->Columna][this->
                                Fila]->ocupante = "Vacio";
                            return 0;
                        }
                    }
                }
            }
        }

    return 0;
}

/// \brief Metodo para reproducirse. Solo se puede reproducir con un
    animal y solo si es de su misma
///        especie pero de distinto sexo si se reproduce tanto este
    como la pareja no puede volver a
///        resproducirse durante el dia.
///
/// \param reproduzcase Mediante esta variable se informa si hay una
    pareja disponible.
/// \param x Posicion del animal con el cual se puede reproducir.
/// \param y Posicion del animal con el cual se puede reproducir.
//metodo reproducir
void Lobo::operator~() {
    bool reproduzcase = false; //Mediante esta variable se informa si
        hay una pareja disponible.
    int x =0, y = 0; //Posicion del animal con el cual se puede
        reproducir.

    //en este doble ciclo se busca que haya una pareja disponible y que
        NO se haya reproducido en el presente dia
    //este doble ciclo for esta configurado de manera que se busca
        alrededor de la celda en la cual se esta
    //evita tambien salirse de la matriz para no dar errores de
        segmentacion y evita utilizarse a si misma
    for (int xpos = this->Columna-1; xpos <= this->Columna+1; ++xpos) {
        for (int ypos = this->Fila-1; ypos <= this->Fila+1; ++ypos) {
            if (!(xpos == this->Columna && ypos == this->Fila) && !
                reproduzcase) { //no se mete en si mismo
                if ((xpos >=  0 && xpos < Controlador::columns) && (
                    ypos >=  0 && ypos < Controlador::rows)) { //evita
                    que se salga de la matriz
```

```
                    if (Controlador::terreno[ypos][xpos]->ocupante.
                        compare("Vacio") != 0) { //si hay un animal en
                        la celda entra
                         if (Controlador::terreno[ypos][xpos]->animal->
                            tipoAnimal.compare(this->tipoAnimal) == 0) {
                            //revisa que ambos animales sean del mismo
                            tipo
                             if (Controlador::terreno[ypos][xpos]->
                                animal->Sexo != this->Sexo && !
                                Controlador::terreno[ypos][xpos]->animal
                                ->alreadyReproduced) { //revisa que
                                ambos animales sean de distinto sexo
                                 reproduzcase = true;//pone la bandera
                                 Controlador::terreno[ypos][xpos]->
                                    animal->alreadyReproduced = true;//
                                    marca que el animal target ya se va
                                    a reproducir por este dia
                                 //estas posiciones se guardan en caso
                                    de que no se encuentro un espacio
                                    para poner la
                                 //cria, entonces se desmarca la pareja
                                    para que se pueda reproducir de
                                    nuevo
                                 x = xpos;
                                 y = ypos;
                             }
                         }
                    }
                }
            }
        }
    }
}

//este doble ciclo busca una celda del terreno vacia en la cual se
   pueda poner un futuro hizo del apareaminteo
for (int xpos1 = this->Columna-1; xpos1 <= this->Columna+1; ++xpos1
   ) {
    for (int ypos1 = this->Fila-1; ypos1 <= this->Fila+1; ++ypos1)
       {
        if (!(xpos1 == this->Columna && ypos1 == this->Fila)) { //
           no se mete en si mismo
            if ((xpos1 >=  0 && xpos1 < Controlador::columns) && (
               ypos1 >=  0 && ypos1 < Controlador::rows)) {
                if (Controlador::terreno[ypos1][xpos1]->ocupante.
                   compare("Vacio") == 0) {//busca uno que este
                   vacio
                    if(reproduzcase && !this->alreadyReproduced){
                        //crea un animal nuevo con el sexo del
```

```
                                    animal ->this
                                Controlador::terreno[ypos1][xpos1]->animal
                                    = new Lobo(ypos1, xpos1, Controlador::
                                    terreno[this->Fila][this->Columna]->
                                    animal->Sexo);
                                Controlador::terreno[ypos1][xpos1]->
                                    ocupante = Controlador::terreno[this->
                                    Fila][this->Columna]->ocupante;
                                reproduzcase = false;
                                Controlador::terreno[this->Fila][this->
                                    Columna]->animal->alreadyReproduced =
                                    true;
                            }
                        }
                    }
                }
            }
        }
    //en caso de que ->this no haya encontrado campo para poner el hijo
        , entonces desmarco a la pareja
    if (!Controlador::terreno[this->Fila][this->Columna]->animal->
        alreadyReproduced)
         Controlador::terreno[y][x]->animal->alreadyReproduced = false;
}

/*void Lobo::PrintLobo() {
    cout << "Especie: Lobo" << endl;
    this->Print();
}*/
```

2.10. Oveja.h

```
#ifndef OVEJA_H
#define OVEJA_H

#include "Animal.h"
#include "Celda.h"
#include "Controlador.h"

class Oveja : public Animal {

//Metodos
public:
    Oveja();
    Oveja(int Fila, int Columna, int Sexo);
    virtual ~Oveja();

    int operator!(); //metodo mover
```

```
    int operator++(); //funcion de comer
    void operator~(); //metodo reproducir
    //void PrintOveja();
};

#endif // OVEJA_H
```

2.11. Oveja.cpp

```
/**
*@file Oveja.cpp
*@version 1.0
*@date 29/01/17
*@author Luis Diego Fernandez, Daniel Jimenez
*@title Juego de la vida
*@brief Clase Oveja
*/

#include "Oveja.h"

/*! \brief Constructor por defecto.
 */
Oveja::Oveja() {

}

/*! \brief Constructor para crear una Oveja con datos especificos.
 */
Oveja::Oveja(int Fila, int Columna, int Sexo) {
    this->Fila = Fila;
    this->Columna = Columna;
    this->Sexo = Sexo;
    Energia = 75;
    tipoAnimal = "Oveja";
}

/*! \brief Destructor.
 */
Oveja::~Oveja() {

}

/*! \brief Metodo para que la oveja se mueva en el terreno. Es capaz de
    moverse
 *          hasta dos espacios en el terrreno siempre que estan
    desocupados y
 *          sean aledannos.
 *
```

```
 *  \param xActual Posicion actual del animal.
 *  \param yActual Posicion actual del animal.
 *  \param xPrevio Posicion previa del animal.
 *  \param yPrevio Posicion previa del animal.
 *  \param contador Numero de veces que se ha desplzado el animal.
 */
int Oveja::operator!() {
    int xActual = this->Columna; //Posicion actual del animal.
    int yActual = this->Fila;
    int yPrevio, xPrevio; //Posicion actual del animal.
    int contador = 0; //Numero de veces que se ha desplazado el animal.
    int temp;

    //Verifico que el animal no se haya movido.
    if(Controlador::terreno[this->Fila][this->Columna]->animal->
       alreadyMoved == false) {
        //Si el animal se mueve se deben empezar de nuevo los ciclos
           for por lo tanto se utiliza este ciclo while para ello.
        while (temp) {
            temp = 0;
            //Se busca si en las posiciones aledannas si hay algun
               espacio libre en donde se pueda mover el animal.
            //Este doble ciclo for esta configurado de manera que se
               busca alrededor de la celda en la cual se esta
            //evita tambien salirse de la matriz para no dar errores de
                segmentacion y evita utilizarse a si misma.
            for (int xpos = xActual-1; xpos <= xActual+1; ++xpos) {
                for (int ypos = yActual-1; ypos <= yActual+1; ++ypos) {
                    if (!(xpos == xActual && ypos == yActual)) { //no
                       se mete en si mismo
                        if ((xpos >=  0 && xpos < Controlador::columns)
                            && (ypos >=  0 && ypos < Controlador::rows)
                           )
                            //Si la celda no tiene animal y ademas no
                                es la posicion previa en la cual estuvo,
                                 se mueve.
                            if((Controlador::terreno[ypos][xpos]->
                               ocupante.compare("Vacio") == 0) && ypos
                               != yPrevio && xpos != xPrevio){
                                //Creo el nuevo animal con las mismas
                                    caracteristicas que el original.
                                Controlador::terreno[ypos][xpos]->
                                    animal = new Oveja(ypos, xpos,
                                    Controlador::terreno[yActual][
                                    xActual]->animal->Sexo);
                                Controlador::terreno[ypos][xpos]->
                                    animal->Energia = Controlador::
                                    terreno[yActual][xActual]->animal->
```

```
                                        Energia;
                                    Controlador::terreno[ypos][xpos]->
                                        ocupante = Controlador::terreno[
                                        yActual][xActual]->ocupante;
                                    //Indico que ya se movio.
                                    Controlador::terreno[ypos][xpos]->
                                        animal->alreadyMoved = true;
                                    //Elimino el animal de la posicion de
                                        la cual se esta desplazando para
                                        dejar la celda vacia.
                                    delete Controlador::terreno[yActual][
                                        xActual]->animal;
                                    Controlador::terreno[yActual][xActual
                                        ]->ocupante = "Vacio";
                                    //Actualizo variables de control.
                                    yPrevio = yActual;
                                    xPrevio = xActual;
                                    yActual = ypos;
                                    xActual = xpos;
                                    contador += 1;
                                    if(contador == 2)
                                        return 0;
                                }
                            }
                        }
                    }
                }
            }
            return 0;
}

/// \brief Metodo para comer. Se alimenta del zacate, consume 10 puntos
    del zacate y recupera la misma cantidad.
///         En caso de no haber suficiente zacate, consume lo que haya y
    recupera eso mismo. No puede excederse
///         de 75 la energia de la oveja.
int Oveja::operator++() {
    //Si el zacate tiene mas de 10 puntos, consume 10 y los adquiere el
        animal.
    if(Controlador::terreno[this->Fila][this->Columna]->zacate >= 10){
        Controlador::terreno[this->Fila][this->Columna]->zacate -= 10;
        Controlador::terreno[this->Fila][this->Columna]->animal->
            Energia += 10;
        //Se verifica que no se haya pasado de 75
        if(Controlador::terreno[this->Fila][this->Columna]->animal->
            Energia > 75)
            Controlador::terreno[this->Fila][this->Columna]->animal->
                Energia = 75;
```

```
    //En caso de que haya menos que 10 puntos consume la cantidad de
       puntos que haya nada mas.
    }else{
        Controlador::terreno[this->Fila][this->Columna]->animal->
           Energia += Controlador::terreno[this->Fila][this->Columna]->
           zacate;
        Controlador::terreno[this->Fila][this->Columna]->zacate = 0;
        //Se verifica que no se haya pasado de 75
        if(Controlador::terreno[this->Fila][this->Columna]->animal->
           Energia > 75)
            Controlador::terreno[this->Fila][this->Columna]->animal->
               Energia = 75;
    }

    return 0;
}

/// \brief Metodo para reproducirse. Solo se puede reproducir con un
   animal y solo si es de su misma
///        especie pero de distinto sexo si se reproduce tanto este
   como la pareja no puede volver a
///        resproducirse durante el dia.
///
/// \param reproduzcase Mediante esta variable se informa si hay una
   pareja disponible.
/// \param x Posicion del animal con el cual se puede reproducir.
/// \param y Posicion del animal con el cual se puede reproducir.
//metodo reproducir
void Oveja::operator~() {
    bool reproduzcase = false; //Mediante esta variable se informa si
       hay una pareja disponible.
    int x =0, y = 0; //Posicion del animal con el cual se puede
       reproducir.

    //en este doble ciclo se busca que haya una pareja disponible y que
        NO se haya reproducido en el presente dia
    //este doble ciclo for esta configurado de manera que se busca
       alrededor de la celda en la cual se esta
    //evita tambien salirse de la matriz para no dar errores de
       segmentacion y evita utilizarse a si misma
    for (int xpos = this->Columna-1; xpos <= this->Columna+1; ++xpos) {
        for (int ypos = this->Fila-1; ypos <= this->Fila+1; ++ypos) {
            if (!(xpos == this->Columna && ypos == this->Fila) && !
               reproduzcase) { //no se mete en si mismo
                if ((xpos >=  0 && xpos < Controlador::columns) && (
                   ypos >=  0 && ypos < Controlador::rows)) { //evita
                   que se salga de la matriz
                    if (Controlador::terreno[ypos][xpos]->ocupante.
```

```
                    compare("Vacio") != 0) { //si hay un animal en
                    la celda entra
                     if (Controlador::terreno[ypos][xpos]->animal->
                        tipoAnimal.compare(this->tipoAnimal) == 0) {
                        //revisa que ambos animales sean del mismo
                        tipo
                         if (Controlador::terreno[ypos][xpos]->
                            animal->Sexo != this->Sexo && !
                            Controlador::terreno[ypos][xpos]->animal
                            ->alreadyReproduced) { //revisa que
                            ambos animales sean de distinto sexo
                             reproduzcase = true;//pone la bandera
                             Controlador::terreno[ypos][xpos]->
                                animal->alreadyReproduced = true;//
                                marca que el animal target ya se va
                                a reproducir por este dia
                             //estas posiciones se guardan en caso
                                de que no se encuentro un espacio
                                para poner la
                             //cria, entonces se desmarca la pareja
                                para que se pueda reproducir de
                                nuevo
                             x = xpos;
                             y = ypos;
                         }
                     }
                 }
             }
         }
     }
}

//este doble ciclo busca una celda del terreno vacia en la cual se
    pueda poner un futuro hizo del apareaminteo
for (int xpos1 = this->Columna-1; xpos1 <= this->Columna+1; ++xpos1
    ) {
     for (int ypos1 = this->Fila-1; ypos1 <= this->Fila+1; ++ypos1)
        {
         if (!(xpos1 == this->Columna && ypos1 == this->Fila)) { //
            no se mete en si mismo
             if ((xpos1 >=  0 && xpos1 < Controlador::columns) && (
                ypos1 >=  0 && ypos1 < Controlador::rows)) {
                 if (Controlador::terreno[ypos1][xpos1]->ocupante.
                    compare("Vacio") == 0) {//busca uno que este
                    vacio
                     if(reproduzcase && !this->alreadyReproduced){
                         //crea un animal nuevo con el sexo del
                            animal ->this
```

```
                                Controlador::terreno[ypos1][xpos1]->animal
                                    = new Oveja(ypos1, xpos1, Controlador::
                                    terreno[this->Fila][this->Columna]->
                                    animal->Sexo);
                                Controlador::terreno[ypos1][xpos1]->
                                    ocupante = Controlador::terreno[this->
                                    Fila][this->Columna]->ocupante;
                                reproduzcase = false;
                                Controlador::terreno[this->Fila][this->
                                    Columna]->animal->alreadyReproduced =
                                    true;
                            }
                        }
                    }
                }
            }
        }
    //en caso de que ->this no haya encontrado campo para poner el hijo
        , entonces desmarco a la pareja
    if (!Controlador::terreno[this->Fila][this->Columna]->animal->
        alreadyReproduced)
         Controlador::terreno[y][x]->animal->alreadyReproduced = false;
}

/*void Oveja::PrintOveja() {
    cout << "Especie: Oveja" << endl;
    this->Print();
}*/
```

2.12. Raton.h

```
#ifndef RATON_H
#define RATON_H

#include "Animal.h"
#include "Celda.h"
#include "Controlador.h"

class Raton : public Animal {

//Metodos
public:
    Raton();
    Raton(int Fila, int Columna, int Sexo);
    virtual ~Raton();

    int operator!(); //metodo mover
    int operator++(); //funcion de comer
```

```
    void operator~(); //metodo reproducir
    //void PrintRaton();
};

#endif // RATON_H
```

2.13. Raton.cpp

```
/**
*@file Raton.cpp
*@version 1.0
*@date 29/01/17
*@author Luis Diego Fernandez, Daniel Jimenez
*@title Juego de la vida
*@brief Clase Raton
*/

#include "Raton.h"

/*! \brief Constructor por defecto.
 */
Raton::Raton() {

}

/*! \brief Constructor para crear un Raton con datos especificos.
 */
Raton::Raton(int Fila, int Columna, int Sexo) {
    this->Fila = Fila;
    this->Columna = Columna;
    this->Sexo = Sexo;
    Energia = 25;
    tipoAnimal = "Raton";
}

/*! \brief Destructor.
 */
Raton::~Raton() {

}

/*! \brief Metodo para que el raton se mueva en el terreno. Es capaz de
    moverse
 *          solo un espacio en el terrreno siempre que este desocupado y
 *          sea aledanno.
 */
int Raton::operator!() {
    //Verifico que el animal no se haya movido.
```

```
if(Controlador::terreno[this->Fila][this->Columna]->animal->
    alreadyMoved == false) {
     //Se busca si en las posiciones aledannas si hay algun espacio
         libre en donde se pueda mover el animal.
     //Este doble ciclo for esta configurado de manera que se busca
         alrededor de la celda en la cual se esta
     //evita tambien salirse de la matriz para no dar errores de
         segmentacion y evita utilizarse a si misma.
     for (int xpos = this->Columna-1; xpos <= this->Columna+1; ++
        xpos) {
          for (int ypos = this->Fila-1; ypos <= this->Fila+1; ++ypos)
              {
              if (!(xpos == this->Columna && ypos == this->Fila)) {
                 //no se mete en si mismo
                  if ((xpos >=  0 && xpos < Controlador::columns) &&
                     (ypos >=  0 && ypos < Controlador::rows)) {
                      //Si la celda no tiene animal, se mueve.
                      if(Controlador::terreno[ypos][xpos]->ocupante.
                         compare("Vacio") == 0){
                          //Creo el nuevo animal con las mismas
                              caracteristicas que el original.
                          Controlador::terreno[ypos][xpos]->animal =
                             new Raton(ypos, xpos, Controlador::
                             terreno[this->Fila][this->Columna]->
                             animal->Sexo);
                          Controlador::terreno[ypos][xpos]->animal->
                             Energia = Controlador::terreno[this->
                             Fila][this->Columna]->animal->Energia;
                          Controlador::terreno[ypos][xpos]->ocupante
                             = Controlador::terreno[this->Fila][this
                             ->Columna]->ocupante;
                          //Indico que ya se movio.
                          Controlador::terreno[ypos][xpos]->animal->
                             alreadyMoved = true;
                          //Elimino el animal de la posicion de la
                              cual se esta desplazando para dejar la
                              celda vacia.
                          delete Controlador::terreno[this->Fila][
                             this->Columna]->animal;
                          Controlador::terreno[this->Fila][this->
                             Columna]->ocupante = "Vacio";
                          return 0;
                      }

                  }
              }
          }
     }
```

```
    }
    return 0;
}

/// \brief Metodo para comer. Se alimenta del zacate, consume 5 puntos
   del zacate y recupera la misma cantidad.
///        En caso de no haber suficiente zacate, consume lo que haya y
    recupera eso mismo. No puede excederse
///        de 25 la energia del Raton.
int Raton::operator++() {
    //Si el zacate tiene mas de 5 puntos, consume 5 y los adquiere el
       animal.
    if(Controlador::terreno[this->Fila][this->Columna]->zacate >= 5){
        Controlador::terreno[this->Fila][this->Columna]->zacate -= 5;
        Controlador::terreno[this->Fila][this->Columna]->animal->
           Energia += 5;
        //Se verifica que no se haya pasado de 25
        if(Controlador::terreno[this->Fila][this->Columna]->animal->
           Energia > 25)
            Controlador::terreno[this->Fila][this->Columna]->animal->
               Energia = 25;
    //En caso de que haya menos que 5 puntos consume la cantidad de
       puntos que haya nada mas.
    }else{
        Controlador::terreno[this->Fila][this->Columna]->animal->
           Energia += Controlador::terreno[this->Fila][this->Columna]->
           zacate;
        Controlador::terreno[this->Fila][this->Columna]->zacate = 0;
        //Se verifica que no se haya pasado de 25
        if(Controlador::terreno[this->Fila][this->Columna]->animal->
           Energia > 25)
            Controlador::terreno[this->Fila][this->Columna]->animal->
               Energia = 25;
    }

    return 0;
}

/// \brief Metodo para reproducirse. Solo se puede reproducir con un
   animal y solo si es de su misma especie
///        pero de distinto sexo si se reproduce tanto este como la
   pareja no puede volver a resproducirse durante el dia.
///
/// \param reproduzcase Mediante esta variable se informa si hay una
   pareja disponible.
/// \param x Posicion del animal con el cual se puede reproducir.
/// \param y Posicion del animal con el cual se puede reproducir.
//metodo reproducir
```

```
void Raton::operator~() {
    bool reproduzcase = false; //Mediante esta variable se informa si
       hay una pareja disponible.
    int x =0, y = 0; //Posicion del animal con el cual se puede
       reproducir.

    //en este doble ciclo se busca que haya una pareja disponible y que
        NO se haya reproducido en el presente dia
    //este doble ciclo for esta configurado de manera que se busca
       alrededor de la celda en la cual se esta
    //evita tambien salirse de la matriz para no dar errores de
       segmentacion y evita utilizarse a si misma
    for (int xpos = this->Columna-1; xpos <= this->Columna+1; ++xpos) {
        for (int ypos = this->Fila-1; ypos <= this->Fila+1; ++ypos) {
            if (!(xpos == this->Columna && ypos == this->Fila) && !
               reproduzcase) { //no se mete en si mismo
                if ((xpos >=  0 && xpos < Controlador::columns) && (
                   ypos >=  0 && ypos < Controlador::rows)) { //evita
                   que se salga de la matriz
                    if (Controlador::terreno[ypos][xpos]->ocupante.
                       compare("Vacio") != 0) { //si hay un animal en
                       la celda entra
                        if (Controlador::terreno[ypos][xpos]->animal->
                           tipoAnimal.compare(this->tipoAnimal) == 0) {
                           //revisa que ambos animales sean del mismo
                           tipo
                            if (Controlador::terreno[ypos][xpos]->
                               animal->Sexo != this->Sexo && !
                               Controlador::terreno[ypos][xpos]->animal
                               ->alreadyReproduced) { //revisa que
                               ambos animales sean de distinto sexo
                                reproduzcase = true;//pone la bandera
                                Controlador::terreno[ypos][xpos]->
                                   animal->alreadyReproduced = true;//
                                   marca que el animal target ya se va
                                   a reproducir por este dia
                                //estas posiciones se guardan en caso
                                   de que no se encuentro un espacio
                                   para poner la
                                //cria, entonces se desmarca la pareja
                                   para que se pueda reproducir de
                                   nuevo
                                x = xpos;
                                y = ypos;
                            }
                        }
                    }
                }
            }
```

```
            }
        }
    }

    //este doble ciclo busca una celda del terreno vacia en la cual se
       pueda poner un futuro hizo del apareaminteo
    for (int xpos1 = this->Columna-1; xpos1 <= this->Columna+1; ++xpos1
       ) {
        for (int ypos1 = this->Fila-1; ypos1 <= this->Fila+1; ++ypos1)
           {
            if (!(xpos1 == this->Columna && ypos1 == this->Fila)) { //
               no se mete en si mismo
                if ((xpos1 >=  0 && xpos1 < Controlador::columns) && (
                   ypos1 >=  0 && ypos1 < Controlador::rows)) {
                    if (Controlador::terreno[ypos1][xpos1]->ocupante.
                       compare("Vacio") == 0) {//busca uno que este
                       vacio
                        if(reproduzcase && !this->alreadyReproduced){
                            //crea un animal nuevo con el sexo del
                               animal ->this
                            Controlador::terreno[ypos1][xpos1]->animal
                               = new Raton(ypos1, xpos1, Controlador::
                               terreno[this->Fila][this->Columna]->
                               animal->Sexo);
                            Controlador::terreno[ypos1][xpos1]->
                               ocupante = Controlador::terreno[this->
                               Fila][this->Columna]->ocupante;
                            reproduzcase = false;
                            Controlador::terreno[this->Fila][this->
                               Columna]->animal->alreadyReproduced =
                               true;
                        }
                    }
                }
            }
        }
    }
    //en caso de que ->this no haya encontrado campo para poner el hijo
       , entonces desmarco a la pareja
    if (!Controlador::terreno[this->Fila][this->Columna]->animal->
       alreadyReproduced)
        Controlador::terreno[y][x]->animal->alreadyReproduced = false;
}

/*void Raton::PrintRaton() {
    cout << "Especie: Raton" << endl;
    this->Print();
}*/
```

2.14. Zorro.h

```
#ifndef ZORRO_H
#define ZORRO_H

#include "Animal.h"
#include "Celda.h"
#include "Controlador.h"


class Zorro : public Animal {

//Metodos
public:
    Zorro();
    Zorro(int Fila, int Columna, int Sexo);
    virtual ~Zorro();

    int operator!(); //metodo mover
    int operator++(); //funcion de comer
    void operator~(); //metodo reproducir
    //void PrintZorro();
};

#endif // ZORRO_H
```

2.15. Zorro.cpp

```
/**
*@file Zorro.cpp
*@version 1.0
*@date 29/01/17
*@author Luis Diego Fernandez, Daniel Jimenez
*@title Juego de la vida
*@brief Clase Zorro
*/

#include "Zorro.h"

/*! \brief Constructor por defecto.
 */
Zorro::Zorro() {

}

/*! \brief Constructor para crear un Zorro con datos especificos.
 */
Zorro::Zorro(int Fila, int Columna, int Sexo) {
    this->Fila = Fila;
```

```
    this->Columna = Columna;
    this->Sexo = Sexo;
    Energia = 50;
    tipoAnimal = "Zorro";
}

/*! \brief Destructor.
 */
Zorro::~Zorro() {

}

/*! \brief Metodo para que el zorro se mueva en el terreno. Es capaz de
    moverse
 *          solo un espacio en el terrreno siempre que este desocupado y
 *          sea aledanno.
 *
 *  \param xActual Posicion actual del animal.
 *  \param yActual Posicion actual del animal.
 *  \param xPrevio Posicion previa del animal.
 *  \param yPrevio Posicion previa del animal.
 *  \param contador Numero de veces que se ha desplzado el animal.
 */
int Zorro::operator!() {
    int xActual = this->Columna; //Posicion actual del animal.
    int yActual = this->Fila;
    int yPrevio, xPrevio; //Posicion actual del animal.
    int contador = 0; //Numero de veces que se ha desplazado el animal.
    int temp;

    //Verifico que el animal no se haya movido.
    if(Controlador::terreno[this->Fila][this->Columna]->animal->
       alreadyMoved == false) {
        //Si el animal se mueve se deben empezar de nuevo los ciclos
           for por lo tanto se utiliza este ciclo while para ello.
        while (temp) {
            temp = 0;
            //Se busca si en las posiciones aledannas si hay algun
               espacio libre en donde se pueda mover el animal.
            //Este doble ciclo for esta configurado de manera que se
               busca alrededor de la celda en la cual se esta
            //evita tambien salirse de la matriz para no dar errores de
                segmentacion y evita utilizarse a si misma.
            for (int xpos = xActual-1; xpos <= xActual+1; ++xpos) {
                for (int ypos = yActual-1; ypos <= yActual+1; ++ypos) {
                    if (!(xpos == xActual && ypos == yActual)) { //no
                       se mete en si mismo
                        if ((xpos >=  0 && xpos < Controlador::columns)
```

```
                                    && (ypos >=  0 && ypos < Controlador::rows)
                                   )
                                    //Si la celda no tiene animal y ademas no
                                        es la posicion previa en la cual estuvo,
                                         se mueve.
                                    if((Controlador::terreno[ypos][xpos]->
                                        ocupante.compare("Vacio") == 0) && ypos
                                        != yPrevio && xpos != xPrevio){
                                         //Creo el nuevo animal con las mismas
                                             caracteristicas que el original.
                                         Controlador::terreno[ypos][xpos]->
                                             animal = new Zorro(ypos, xpos,
                                             Controlador::terreno[yActual][
                                             xActual]->animal->Sexo);
                                         Controlador::terreno[ypos][xpos]->
                                             animal->Energia = Controlador::
                                             terreno[yActual][xActual]->animal->
                                             Energia;
                                         Controlador::terreno[ypos][xpos]->
                                             ocupante = Controlador::terreno[
                                             yActual][xActual]->ocupante;
                                         //Indico que ya se movio.
                                         Controlador::terreno[ypos][xpos]->
                                             animal->alreadyMoved = true;
                                         //Elimino el animal de la posicion de
                                             la cual se esta desplazando para
                                             dejar la celda vacia.
                                         delete Controlador::terreno[yActual][
                                             xActual]->animal;
                                         Controlador::terreno[yActual][xActual
                                             ]->ocupante = "Vacio";
                                         //Actualizo variables de control.
                                         yPrevio = yActual;
                                         xPrevio = xActual;
                                         yActual = ypos;
                                         xActual = xpos;
                                         contador += 1;
                                         if(contador == 2)
                                             return 0;
                                    }
                                }
                            }
                        }
                    }
                }
                return 0;
            }
```

```
/// \brief Metodo para Comer. Se alimenta de Ratones que esten en
    posiciones aledannas, consume 2 puntos
///         y mata al otro animal. No puede excederse de 50 la energia
    del Zorro.
int Zorro::operator++() {

    //Se busca si en las posiciones aledannas hay un Raton de cualquier
        sexo para eliminarlo y subir la energia del Zorro
    //este doble ciclo for esta configurado de manera que se busca
        alrededor de la celda en la cual se esta
    //evita tambien salirse de la matriz para no dar errores de
        segmentacion y evita utilizarse a si misma
    for (int xpos = this->Columna-1; xpos <= this->Columna+1; ++xpos) {
        for (int ypos = this->Fila-1; ypos <= this->Fila+1; ++ypos) {
            if (!(xpos == this->Columna && ypos == this->Fila)) { //no
                se mete en si mismo
                if ((xpos >=  0 && xpos < Controlador::columns) && (
                    ypos >=  0 && ypos < Controlador::rows)){
                    if(Controlador::terreno[ypos][xpos]->ocupante.
                        compare("RM") == 0 || Controlador::terreno[
                        ypos][xpos]->ocupante.compare("RH") == 0) {
                        delete Controlador::terreno[ypos][xpos]->animal
                            ; //Mato al animal.
                        Controlador::terreno[ypos][xpos]->ocupante = "
                            Vacio"; //Asigno vacia la celda.
                        Controlador::terreno[this->Fila][this->Columna
                            ]->animal->Energia += 2; //Aumento energia
                             del zorro
                        //Verifico que no se pase de 50 la energia.
                        if(Controlador::terreno[this->Fila][this->
                            Columna]->animal->Energia > 50)
                             Controlador::terreno[this->Fila][this->
                                 Columna]->animal->Energia = 50;
                        return 0;
                    }
                }
            }
        }
    }

    return 0;
}

/// \brief Metodo para reproducirse. Solo se puede reproducir con un
    animal y solo si es de su misma especie
///         pero de distinto sexo si se reproduce tanto este como la
    pareja no puede volver a resproducirse durante el dia.
///
```

```
/// \param reproduzcase Mediante esta variable se informa si hay una
    pareja disponible.
/// \param x Posicion del animal con el cual se puede reproducir.
/// \param y Posicion del animal con el cual se puede reproducir.
//metodo reproducir
void Zorro::operator~() {
    bool reproduzcase = false; //Mediante esta variable se informa si
        hay una pareja disponible.
    int x =0, y = 0; //Posicion del animal con el cual se puede
        reproducir.

    //en este doble ciclo se busca que haya una pareja disponible y que
        NO se haya reproducido en el presente dia
    //este doble ciclo for esta configurado de manera que se busca
        alrededor de la celda en la cual se esta
    //evita tambien salirse de la matriz para no dar errores de
        segmentacion y evita utilizarse a si misma
    for (int xpos = this->Columna-1; xpos <= this->Columna+1; ++xpos) {
        for (int ypos = this->Fila-1; ypos <= this->Fila+1; ++ypos) {
            if (!(xpos == this->Columna && ypos == this->Fila) && !
                reproduzcase) { //no se mete en si mismo
                if ((xpos >=  0 && xpos < Controlador::columns) && (
                    ypos >=  0 && ypos < Controlador::rows)) { //evita
                    que se salga de la matriz
                    if (Controlador::terreno[ypos][xpos]->ocupante.
                        compare("Vacio") != 0) { //si hay un animal en
                        la celda entra
                        if (Controlador::terreno[ypos][xpos]->animal->
                            tipoAnimal.compare(this->tipoAnimal) == 0) {
                            //revisa que ambos animales sean del mismo
                            tipo
                            if (Controlador::terreno[ypos][xpos]->
                                animal->Sexo != this->Sexo && !
                                Controlador::terreno[ypos][xpos]->animal
                                ->alreadyReproduced) { //revisa que
                                ambos animales sean de distinto sexo
                                reproduzcase = true;//pone la bandera
                                Controlador::terreno[ypos][xpos]->
                                    animal->alreadyReproduced = true;//
                                    marca que el animal target ya se va
                                    a reproducir por este dia
                                //estas posiciones se guardan en caso
                                    de que no se encuentro un espacio
                                    para poner la
                                //cria, entonces se desmarca la pareja
                                    para que se pueda reproducir de
                                    nuevo
                                x = xpos;
```

```
                                y = ypos;
                            }
                        }
                    }
                }
            }
        }

        //este doble ciclo busca una celda del terreno vacia en la cual se
            pueda poner un futuro hizo del apareaminteo
        for (int xpos1 = this->Columna-1; xpos1 <= this->Columna+1; ++xpos1
            ) {
            for (int ypos1 = this->Fila-1; ypos1 <= this->Fila+1; ++ypos1)
                {
                if (!(xpos1 == this->Columna && ypos1 == this->Fila)) { //
                    no se mete en si mismo
                    if ((xpos1 >=  0 && xpos1 < Controlador::columns) && (
                        ypos1 >=  0 && ypos1 < Controlador::rows)) {
                        if (Controlador::terreno[ypos1][xpos1]->ocupante.
                            compare("Vacio") == 0) {//busca uno que este
                            vacio
                            if(reproduzcase && !this->alreadyReproduced){
                                //crea un animal nuevo con el sexo del
                                    animal ->this
                                Controlador::terreno[ypos1][xpos1]->animal
                                    = new Zorro(ypos1, xpos1, Controlador::
                                    terreno[this->Fila][this->Columna]->
                                    animal->Sexo);
                                Controlador::terreno[ypos1][xpos1]->
                                    ocupante = Controlador::terreno[this->
                                    Fila][this->Columna]->ocupante;
                                reproduzcase = false;
                                Controlador::terreno[this->Fila][this->
                                    Columna]->animal->alreadyReproduced =
                                    true;
                            }
                        }
                    }
                }
            }
        }
        //en caso de que ->this no haya encontrado campo para poner el hijo
            , entonces desmarco a la pareja
        if (!Controlador::terreno[this->Fila][this->Columna]->animal->
            alreadyReproduced)
            Controlador::terreno[y][x]->animal->alreadyReproduced = false;
    }
```

```
/*void Zorro::PrintZorro() {
    cout << "Especie: Zorro" << endl;
    this->Print();
}*/
```

3. Solución propuesta

Se nos propuso crear un programa en el lenguaje C++ que resolviera el problema conocido como el "Juego de la vida". Este consiste en una simulación de un ecosistema donde interactúan diferentes tipos de animales en un terreno de tamaño limitado. El terreno se ve dividido por espacios, cada espacio tiene una cantidad de zacate y puede albergar un animal. El juego se resume en la creación de los animales y la simulación del comportamiento de unos con otros a través de una cantidad n de días. Para este juego hay ciertas reglas que rigen sobre el comportamiento de cada animal y sobre la interacción de unos con otros, estas reglas se pueden encontrar en la sección 1.

3.1. Clase Controlador

Ahora entrando en el detalle con la solución propuesta por el presente grupo, lo primero que se hizo fue crear una clase que manejara el juego, esta clase se llamó controlador y se puede ver su código en las secciones 2.2 y 2.3. El método de principal importancia dentro de esta clase se llama **run()** (vista con más detalle en la sección 3.6) y controla el comportamiento del juego. Dentro de este método se crea el terreno de juego, se crean los animales, se simulan las interacciones durante los días y se acaba el juego. Es importante destacar que este método recibe 2 argumentos:

- int amountOfDays: este es un número entero que le dice al juego cuantos dias de simulación tiene que correr. De esta forma dentro del método **run()** hay un ciclo *for* que itera una cantidad de veces igual a este argumento.

  Esta variable es provista por el main y viene de la línea de comandos, en este sentido, cuando el juego se corre desde la línea de comandos, el primer parámetro que recibe será esta variable y queda almacenada en el vector argv[] en la posición 1.

- char* fileName: esta variable de tipo *char pointer* es un string que nos da el nombre del archivo que guarda los datos necesarios para inicializar el juego. Este archivo se encuentra en la misma carpeta del proyecto y tiene como datos los que el profesor nos dio y que se puede observar en la sección 1.

  De nuevo esta variable es provista por el main y viene de la línea de comandos como segundo parámetro. Al recibirla el controlador utiliza ciertas funciones de la biblioteca *fstream* para manejo de archivos en donde se abre el archivo, se lee y se maneja su contenido a gusto, en nuestro caso para crear el terreno de juego (que se va a ver en detalle en la sección 3.2) y por último se cierra el archivo.

3.2. Terreno de juego

El terreno de juego va a simular un espacio de tierra en el cual los diferentes animales creados para la ejecución del programa van a interactuar. Este terreno está creado por una matriz de tamaño NxM de objetos tipo Celda (ver sección 3.3). De esta manera vamos a tener un terreno de convivencia para los animales cuadriculado en el cual en cada celda de la matriz se puede alojar 1 animal.

Para crear el terreno de juego se reciben datos de un archivo de texto que se encuentra en la misma carpeta del proyecto. Se abre este archivo y en las primeras 2 líneas vienen las dimensiones del terreno, de esta manera se puede crear la matriz que va a contener este terreno. Luego vienen datos sobre la posición y contenido de cada celda con el fin de inicializarlas adecuadamente.

3.3. Clase Celda

Con el fin de crear un ambiente óptimo para la ejecución del programa, se decide crear una clase llamada **Celda**. Esta clase se utilizará luego para crear el terreno de juego en donde se colocarán los animales e interactuarán estos mismos con el fin de lograr una simulación exitosa, dicha creación del terreno se detalla en la sección 3.2. Por ahora es importante saber que cada objeto tipo celda va a tener 4 atributos escenciales para una clara interacción de clases en la ejecución del método **run()** de la clase Controlador. Estos atributos son:

- Objeto animal: cada celda tiene o puede tener dentro de ella un objeto de tipo animal (el cuál se explicará a fondo en la sección 3.4). Esto con el fin de darle a los animales la opción de poder posicionarse en algún lugar en específico dentro del terreno de juego. Es probable que en una celda no haya un animal y esta se encuentre desalojada.
- Atributos de posición: estos serán 2 enteros que guardan la posición de fila y columna de la celda dentro de la matriz del terreno de juego, esto con el fin de poder saber en todo momento con cual celda estamos trabajando y poder interactuar con otras celdas alrededor.
- string ocupante: este string nos va a servir de identificador para saber que tipo de animal y su sexo reside en cada celda.
- int zacate: cada celda, como parte de un terreno de juego que simula un ecosistema viable y capaz de albergar vida animal, puede tener zacate en ella con el fin de que los animales herbívoros pueden comer.

Para crear una celda es fundamental que en su método constructor le pasemos como argumentos cada uno de los atributos mencionados anteriormente. Al hacer esto se puede construir una celda totalmente definida y capaz de valerse por sí misma dentro de la matriz del terreno de juego. Este constructor se puede observar en la sección 2.5.

3.4. Clase Animal

Como ya se ha podido observar en las secciones anteriores y como sería lógico pensar para un juego que simula interacción de animales, se crea una clase Animal. Esta clase nos define características fundamentales que posee un animal así como también tendrá características que nos ayudarán a manejarlo dentro del programa. El tener una clase Animal va a ser de suma importancia para este programa por que nos va a dejar implementar un concepto de programación conocido como **Polimorfismo**. La clase animal (clase padre) es una generalidad de especies específicas de animales (clases hijas). En este juego se van a crear 4 clases hijas distintas: lobo, oveja, zorro y ratón.

Con el polimorfismo podemos crear métodos para un animal que regirán sobre las clases hijas, sin necesidad de ellas de implementar estos métodos un ejemplo de esto sería un método en el cual el animal muera ya que todos los animales mueren de la misma manera. No obstante, hay ciertos cosas que no todos los animales hacen de la misma manera, por ejemplo un lobo que es carnívoro come otros animales, es diferente a una oveja que come zacate, esto se puede manejar también gracias al polimorfismo donde en la clase Animal estos métodos solamente se declaran, pero se hace de una forma especial en la cual uno tiene que implementar dichos métodos en las clases hijo.

De esta manera la clase animal va a tener 5 métodos, estos son:

- Mover: esta función va a ser sobrecargada con el operador ! . Se encarga de que los animales se puedan mover entre celdas aledañas y que estén vacías. Va a ser un método virtual puro por que los animales se mueven de maneras distintas dependiende de que especie sean.

- Comer: esta función va a ser sobrecargada con el operador ++ . Se encarga de que los animales puedan comer y va a ser virtual puro ya que unos son carnívoros y comen otros animales y otros herbíboros y comen zacate.
- Reproducir: esta función va a ser sobrecargada con el operador $\sim$ y va a ser virtual puro por que cada animal se reproduce solamente con animales de su misma especie.
- Morir: esta función se va a sobrecargar con el operador – y va a ser un método de la clase Animal ya que todos los animales mueren de la misma forma. Esta función revisa la energía del animal, si es 0 lo elimina, sino sigue vivo.
- Print: este método imprime el estado de la celda, dejando ver toda la información relevante de ella.

3.5. Clases Hijas

Como ya se mencionó en la sección 3.4 en el programa se crearon varias clases hijas de la clase Animal. Estas clases son las de Lobo, Oveja, Zorro y Ratón y se puede ver sus implementaciones en las secciones 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14 y 2.15. Cada una de estas clases tiene la implementación de los métodos Mover, Comer y Reproducir ya que son métodos virtuales puros. Algunos aspectos importante sobre estas clases hijas están definidas en la sección 1, sin embargo, mencionaremos algunas generalidades importantes:

- Lobo:
  - Los lobos inician con 100 de energía.
  - El lobo se puede mover hasta 3 espacios por día.
  - El lobo es carnívoro y come ovejas, ratones y zorros.
  - El lobo macha se mata con otro lobo macho.
- Oveja:
  - Las ovejas inician con 75 de energía.
  - Las ovejas son herbíboras.
  - Pueden moverse 2 espacios por día.
- Zorro:
  - Los zorros inician con 50 de energía.
  - Los zorroz son carnívoros y se comen a los ratones.
  - Pueden moverse 2 espacios por día.
- Ratón:
  - Los ratones inician con 25 de energía.
  - Los ratones son herbívoros y comen zacate.
  - Pueden moverse 1 espacio por día.

Al hablar de mover una cantidad n de espacios se refiere a n cantidad de celdas dentro de una vecindad de Moore.

3.6. Método run

Mencionadas todas las secciones anteriores y manejándolas de buena manera se logra crear una relación entre ellas de manera que el juego cobre vida y tenga sentido. En la clase Controlador en su método **run** se crean objetos de las clases mencionadas anteriormente, estos utlizan funciones de estos objetos y se implementa una lógia que permita que el sistema se desarrolle exitosamente. Este método run funciona de la siguiente manera y se puede ver su implementación en la sección 2.3:

1. Se declaran ciertas variables de utilidad para la ejecución del método.
2. Se abre el archivo de datos, recordar que el nombre de este viene dado como argumento del método como se mencionó en la sección 3.1.
3. Se extrae del archivo las dimensiones del terreno de juego y se define este como una matriz de objetos tipo celda.
4. Se inicializan todas las celdas de la matriz con los datos del archivo de datos y se imprime el estado inicial del juego.
5. Se cierra el archivo de datos.
6. Se inicia el ciclo que simula los días que pasan.
7. Al inicio de cada día se imprime el estado del terreno.
8. Se controla el crecimiento del zacate.
9. Se llama a la función mover para cada animal.
10. Se llama a la función comer de cada animal.
11. Se llama a la función reproducir de cada animal
12. Se le resta a cada animal la energía que perdío durante el día.
13. Se llama a la función morir de cada animal.
14. Se imprime el estado del terreno de juego al final del día.
15. Se resetean las banderas de control utilizadas.
16. Se libera toda la memoria dinámica utilizada para evitar fugas de información.
17. Se acaba la ejecución del juego.

3.7. Plantilla de impresión

En el enunciado del laboratorio se nos pide que creemos una función emplantillada para hacer la impresión del estado de cualquier objeto. En este caso los objetos que se quieren imprimir son Animales, Lobos, Ovejas, Zorros, Ratones y Celdas. Dicho esto, se puede crear una plantilla que imprima cualquier tipo de animal así a la plantilla no le importa si recibe un Lobo, una Oveja o cualquier otro animal, esta debe de ser capaz de imprimirla. Luego se sobrecarga este método de impresión emplantillado y se hace uno que reciba como argumento un objeto de tipo Celda, de este modo la funición funciona cuando reciba una Animal cualquiera o una Celda.

La implementación de este método y su sobrecarga se puede observar en la sección 2.2. Es de suma importancia resaltar que en la ejecución del juego la impresión de los estados **no** se hace utilizando esta función emplantillada si no que se realiza utilizando una función que imprime una Celda (esta función a la vez llama a una que imprime a los animales y está dentro de la clase Celda). Es por esto que para probar el correcto funcionamiento de esta función se crearon 3 objetos (al final del método run en el Controlador.cpp) uno tipo Lobo, otro tipo Zorro y otro tipo Celda y posteriormente se imprimieron con la función emplantillada (los resultados se muestran en la sección 5.3). Sin embargo, en la entrega del programa estas líneas mencionadas anteriormente se encuentran comentadas como se observa en la sección 2.3.

4. Diagrama de clases

Los diagramas UML son una herramienta que nos permite, además de describir de manera general los atributos y métodos presentes en una clase, su relación con las demás clases, herencias, agregación y demás conexiones posibles entre clases, todo esto de manera visual.

Como ya fue expuesto anteriormente, la construcción de los animales que pueden estar presentes en el ecosistema se realizó mediante una clase padre animal y cuatro mas que heredan de esta. En la figura 1, mostrada seguidamente, se expone el diagrama correspondiente a estas clases.

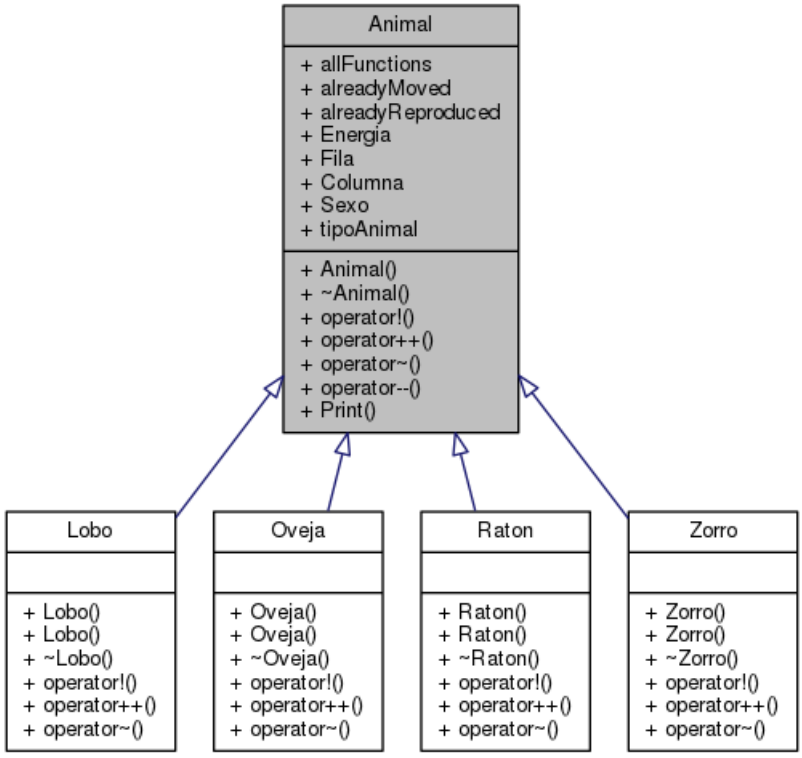

Figura 1: Diagrama de la clase Animal y sus respectivas clases hijo.

Es posible observar y verificar como dentro de cada clase se encuentran sus respectivos métodos y atributos. Ademas, las flechas indican la propiedad de herencia que tienen las clases hijo (Lobo, Oveja, Ratón y Zorro) de la clase padre (Animal).

Además se utilizó una clase celda para la construcción del terreno, y también una clase controlador donde se generaba toda la lógica del juego. Estas dos clases no heredan ni son padres de ninguna otra, pero tienen la particularidad que dentro de su lista de atributos se encuentra un objeto de alguna otra clase ya mencionada. Para apreciar esto de mejor manera se muestra en la figura 2 el diagrama UML correspondiente.

Dentro de Celda, uno de los atributos es un objeto de tipo Animal, el cual fue llamado animal. Y es por esto que se genera una linea para unir la clase Celda y la clase Animal, con el texto al lado $+animal$ para indicar que ese atributo corresponde a un objeto de esa clase. De la misma manera sucede con la clase Controlador. Acá uno de los atributos corresponde a una matriz de objetos Celda, llamada terreno. Este atributo, utilizando la misma linea, se indica que es de tipo Celda y el texto al lado $+terreno$ especifica a cual atributo corresponde. A esta propiedad se le conoce como agregación.

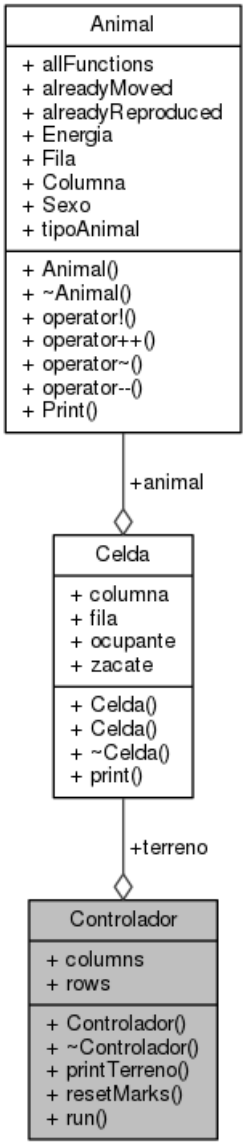

Figura 2: Diagrama de las clases Animal, Celda y Controlador.

5. Resultados

Antes de presentar los resultados obtenidos, es importante mencionar que se va a trabajar con un ecosistema de tamaño 3x3 por simplicidad además de que no se nos proporcionó un archivo de pruebas de mayor tamaño. A continuación en la figura 3 se muestra el contenido del archivo de pruebas que se utilizó.

```
3
3
0 0 25 LH
0 1 75 RM
0 2 0
1 0 100
1 1 50 LH
1 2 25 RH
2 0 75 OM
2 1 0 OM
2 2 0 ZH
```

Figura 3: Datos correspondientes a la inicialización del terreno de juego.

Para mostrar los diferentes resultados y el correcto funcionamiento del juego es probable que en algún momento se modifique este archivo o inclusive el código, en dados casos será indicado como comentario al resultado.

5.1. Ejecución desde la línea de comandos

Este programa está diseñado para ser ejecutado desde la línea de comandos, es por esto que para hacerlo se debe de compilar el programa escribiendo en la línea de comando

```
root@user: ~/path$ make build
```

Una vez compilado el programa se puede correr tomando en cuenta las consideraciones mencionadas en la sección 3.1 sobre los argumentos a utilizar. A continuación en la figura 4 se muestra como se corre el programa.

```
luisdifc@luisdifc:~/documents/estructuras/IE-0217-III-16-G0/Laboratorio0$ ./main 9 datos.txt
Bienvenido al  Juego de la Vida!
May the odds be in  your favor..!

Estado inicial (creción de la Tierra):
Posición: [0, 0]
Nivel zacate: 25
Ocupante:  LH
Energia: 100
Sexo: Hembra
```

Figura 4: Ejecución del programa desde la línea de comandos.

En la figura 4 se observa que se corre el ejecutable del programa con el primer parámetro siendo un 9 este le indica al programa que tiene que hacer una simulación de 9 días. Además se puede observar que el segundo parámetro es una hilera de carácteres “datos.txt”, esto indica que el programa debe de buscar dentro de la carpeta donde está el código fuente un archivo con este nombre y de él obtener los datos para inicializar el juego.

5.2. Impresión de una celda

Cada celda es capaz de imprimirse y mostras su estado con sus datos de mayor relevancia. Se imprimirán las siguientes cosas:

- Posicion: de la forma [x, y].
- Nivel del zacate.
- Ocupante: en caso de estar vacía lo indica Las siguientes 2 características solamente las imprime si tiene un animal dentro.
- Energía.
- Sexo.

5.3. Impresión con plantilla

A continuación se puede observar en la figura 5 como se imprime el estado de una celda que está ocupada por un Lobo de sexo femenino en la cual hay un 35 % de zacate y está en la posición [1, 2].

```
Posición: [1, 2]
Nivel zacate: 35
Ocupante:  LH
Energia: 95
Sexo: Hembra
```

Figura 5: Impresión de una celda ocupada por un animal.

Ahora podemos observar en la figura 6 la impresión de una celda que está vacía.

```
Posición: [1, 2]
Nivel zacate: 40
Ocupante: Vacío
```

Figura 6: Impresión de una celda vacía.

5.4. Estado inicial

El programa antes de empezar a correr las simulaciones e interacciones de los animales muestra al usuario el estado inicial del juego. Esto se puede observar en las figuras 7 y 8. Se puede comparar este estado con los datos mostrados en la figura 3 con el fin de corroborar que está correcta la ejecución.

También se puede observar que los animales, dependiendo de su especie inician con una cantidad de energía diferente. Esto es una regulación impuesta en las indicaciones del juego y se puede observar en la sección 1 y se puede observar como se impementa en el constructor de cada clase hija de animal en las secciones 2.9, 2.11, 2.13 y 2.15.

```
Estado inicial (creción de la Tierra):
Posición: [0, 0]
Nivel zacate: 25
Ocupante:  LH
Energia: 100
Sexo: Hembra

Posición: [0, 1]
Nivel zacate: 75
Ocupante:  RM
Energia: 25
Sexo: Macho

Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante: Vacío

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  LH
Energia: 100
Sexo: Hembra
```

Figura 7: Muestra del estado inicial del juego.

```
Posición: [1, 2]
Nivel zacate: 25
Ocupante:  RH
Energia: 25
Sexo: Hembra

Posición: [2, 0]
Nivel zacate: 75
Ocupante:  OM
Energia: 75
Sexo: Macho

Posición: [2, 1]
Nivel zacate: 0
Ocupante:  OM
Energia: 75
Sexo: Macho

Posición: [2, 2]
Nivel zacate: 0
Ocupante:  ZH
Energia: 50
Sexo: Hembra
```

Figura 8: Muestra del estado inicial del juego.

5.5. Inicio y final de los días

El programa al ejecutarse muestra el estado del sistema todos los días, sin embargo, lo hace una vez al inicio del día y otra vez al final del día con la finalidad de poder observar los comportamientos del sistema durante el día.

Se puede observar entonces en la figura 9 como se muestra un mensaje que indica el inicio del día 7 y luego se imprime el estado de cada celda en ese momento, se muestra solo el estado de la celda en la posición [0, 0] con fines ilustrativos nada maś.

```
Estado al inicio del día 7:

Posición: [0, 0]
Nivel zacate: 35
Ocupante:  LH
Energia: 96
Sexo: Hembra
```

Figura 9: Muestra del estado del sistema al inicio del día 7.

Luego se observa en la figura 10 que se indica que se va a mostrar el estado del sistema al final del dia 5.

```
Estado al final del día 5:

Posición: [0, 0]
Nivel zacate: 30
Ocupante: Vacío
```

Figura 10: Muestra del estado del sistema al final del día 5.

Estos mensajes se imprimen para todos los días que corra la simulación y muestran el estado de todo el sistema.

5.6. Finalización del programa

Al acabar la ejecución del programa se realizan varias tareas en el backend, sin embargo, se muestra un mensaje que se puede observar en la figura 11 que le comunica al usuario que la ejecución del programa llegó a su fin.

```
Finalización del programa..!
```

Figura 11: Finalización de la ejecución del programa.

5.7. Crecimiento del zacate

El zacate en este sistema crece un total de 5 puntos cada 3 días. En caso de que un terreno no tenga zacate, este no crecerá nunca más ahí. Para mostrar estos comportamientos biológicos del zacate tomamos la celda [1, 1] que tiene a una loba (la cual no come zacate) y la celda [0, 2] que no tiene zacate desde un inicio.

```
Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante: Vacío

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  LH
Energia: 100
Sexo: Hembra
```

Figura 12: Nivel del zacate al final del día 1.

La figura 12 es del final del día 1 y se observa que la celda [1, 1] tiene 50 puntos de energía y la celda [0, 2] tiene 0 puntos.

```
Posición: [0, 2]
Nivel zacate: 0
Ocupante:  RM
Energia: 24
Sexo: Macho

Posición: [1, 0]
Nivel zacate: 100
Ocupante:  RM
Energia: 23
Sexo: Macho

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  LH
Energia: 98
Sexo: Hembra
```

Figura 13: Nivel del zacate al final del día 2.

La figura 13 es del final del día 2 y se observa que la celda [1, 1] tiene 50 puntos de energía y la celda [0, 2] tiene 0 puntos.

```
Posición: [0, 2]
Nivel zacate: 0
Ocupante:  RM
Energia: 23
Sexo: Macho

Posición: [1, 0]
Nivel zacate: 100
Ocupante:  RM
Energia: 22
Sexo: Macho

Posición: [1, 1]
Nivel zacate: 55
Ocupante:  LH
Energia: 97
Sexo: Hembra
```

Figura 14: Nivel del zacate al final del día 3.

Ahora la figura 14 es del final del día 3 y se observa que la celda [1, 1] tiene 55 puntos de energía y la celda [0, 2] sigue con 0 puntos. Esto nos deja observar que al cabe de 3 días en la celda [1, 1] aumentó el zacate en 5 puntos y que en la celda [0, 2] no creció zacate nunca.

5.8. Pérdida de energía diaria

Como se menciona en las lineamientos del juego, al final de cada día todos los animales existentes pierden un punto de energía. Si tomamos en cuenta que los animales no comen (para que no ganen energía) y que no se mueven (para poder rastrearlos siempre en la misma posición) podemos observar las siguientes situaciones. En las figuras 15 y 16 la loba que se encuentra en la posición [0, 0] tiene 2 de energía al inicio del día 99 y esto es correcto ya que recordamos que estas empiezan con 100 de energía y al final del día 2 observamos que perdía otro punto de energía ya que tiene solo

1. Esto mismo pasa para cualquier tipo de animal y es un comportamiento esperado según las especificaciones.

```
Estado al inicio del día 99:

Posición: [0, 0]
Nivel zacate: 100
Ocupante:  LH
Energia: 2
Sexo: Hembra
```

Figura 15: Energía al inicio del día 99.

```
Estado al final del día 99:

Posición: [0, 0]
Nivel zacate: 100
Ocupante:  LH
Energia: 1
Sexo: Hembra
```

Figura 16: Energía al final del día 99.

5.9. Muerte de los animales

Un animal se puede morir de distintas maneras, sin embargo, vamos a mostrar el caso de una muerte por falta de energía. Observamos en la figura 17 que al inicio del día 100 la loba de la casilla [0, 0] tiene 1 punto de energía. Ahora al finalizar el día esta loba debe de perder un punto de energía más y por ende en la figura 18 se observa que al final del día 100 la loba ya no existe esto dado que se murió.

```
Estado al inicio del día 100:

Posición: [0, 0]
Nivel zacate: 100
Ocupante:  LH
Energia: 1
Sexo: Hembra
```

Figura 17: Energía al inicio del día 100.

```
Estado al final del día 100:

Posición: [0, 0]
Nivel zacate: 100
Ocupante: Vacío
```

Figura 18: Energía al final del día 100.

5.10. Reproducción de los animales

Vamos a tener que en este sistema cuando tenemos 2 animales de la misma especie y de diferente sexo en una vecindad de Moore, estos se van a aparear y nacerá un hijo de ellos si tiene una celda en el mismo vecindario de Moore libre. Es por esto que observamos la interacción del sistema en el día 1. En la figura 19 estamos al inicio del día 1 y vemos que hay un ratón macho y un ratón hembra en las celdas [0, 1] y [1, 2], dichas celdas se encuentran dentro de una vecindad de Moore, así como también vemos dentro de la misma vecindad las celdas [0, 2] y [1, 0] vacías. Ahora entonces vemos la figura 20 que se encuentra temporalmente al final del día 1 y notamos que la celda [1, 0] que se encontraba vacía antes ahora tiene la cría de los ratones.

```
Posición: [0, 1]
Nivel zacate: 75
Ocupante:  RM
Energia: 25
Sexo: Macho

Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante: Vacío

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  LH
Energia: 100
Sexo: Hembra

Posición: [1, 2]
Nivel zacate: 25
Ocupante:  RH
Energia: 25
Sexo: Hembra
```

Figura 19: Apareamiento de ratones al inicio del día 1.

```
Posición: [0, 1]
Nivel zacate: 75
Ocupante:  RM
Energia: 24
Sexo: Macho

Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante:  RM
Energia: 24
Sexo: Macho

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  LH
Energia: 99
Sexo: Hembra

Posición: [1, 2]
Nivel zacate: 25
Ocupante:  RH
Energia: 24
Sexo: Hembra
```

Figura 20: Nacimiento de un nuevo ratón al final del día 1.

Importante destacar que para obtener esta simulación comentamos el método de comer (ya que los lobos se hubieran comido ambos ratones) y no se hubieran podido reproducir.

5.11. Movimiento de los animales

A la hora de moverse por el terreno de juego cada animal lo puede hacer de manera diferente como se menciona en la sección 3.5. De esta manera para comprobar que todos los animales de movían correctamente se hicieron pruebas en donde estos no comían ni se reproducían con el fin de no alterar el número de animales en el terreno. Así también liberamos un poco el terreno de juego para que hubieran más espacios libres por los cuales los animales que quedaban pudieran moverse, se utilizó la configuración que se muestra en la figura 21.

```
3
3
0 0 25 LH
0 1 75 RM
0 2 0
1 0 100
1 1 50
1 2 25
2 0 75 OM
2 1 0
2 2 0 ZH
```

Figura 21: Datos de la inicialización del tablero para la prueba del método de moverse.

Podemos observar que en el tablero van a haber entonces 4 animales (cada uno de una especia distinta) y 5 espacio libres en los cuales los animales se podrán mover.

```
Estado al inicio del día 1:

Posición: [0, 0]
Nivel zacate: 25
Ocupante:  LH
Energia: 100
Sexo: Hembra

Posición: [0, 1]
Nivel zacate: 75
Ocupante:  RM
Energia: 25
Sexo: Macho

Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante: Vacío

Posición: [1, 1]
Nivel zacate: 50
Ocupante: Vacío
```

Figura 22: Estado del sistema al inicio del día 1.

```
Posición: [1, 2]
Nivel zacate: 25
Ocupante: Vacío

Posición: [2, 0]
Nivel zacate: 75
Ocupante:  OM
Energia: 75
Sexo: Macho

Posición: [2, 1]
Nivel zacate: 0
Ocupante: Vacío

Posición: [2, 2]
Nivel zacate: 0
Ocupante:  ZH
Energia: 50
Sexo: Hembra
```

Figura 23: Estado del sistema al inicio del día 1.

En las figuras 22 y 23 podemos observar el estado del juego al inicio del día 1, los animales están posicionados sobre el terreno de juego cómo se muestra en la figura 21.

```
Estado al final del día 1:

Posición: [0, 0]
Nivel zacate: 25
Ocupante:  RM
Energia: 24
Sexo: Macho

Posición: [0, 1]
Nivel zacate: 75
Ocupante:  OM
Energia: 74
Sexo: Macho

Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante: Vacío

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  ZH
Energia: 49
Sexo: Hembra
```

Figura 24: Estado del sistema al final del día 1.

```
Posición: [1, 1]
Nivel zacate: 50
Ocupante:  ZH
Energia: 49
Sexo: Hembra

Posición: [1, 2]
Nivel zacate: 25
Ocupante: Vacío

Posición: [2, 0]
Nivel zacate: 75
Ocupante: Vacío

Posición: [2, 1]
Nivel zacate: 0
Ocupante:  LH
Energia: 99
Sexo: Hembra

Posición: [2, 2]
Nivel zacate: 0
Ocupante: Vacío
```

Figura 25: Estado del sistema al final del día 1.

Luego al pasar todo el día y al haberse movido los animales observamos en las figuras 24 y 25 el estado del terreno de juego al final del día 1 podemos notar los siguientes aspectos:

- La loba se movió de la posicón [0, 0] a la posición [2, 1]. Esto significa que se movió 3 espacios: [1, 0], [1, 1] y [2, 1].
- El ratón se movió un espacio de [0, 1] a [0, 0].
- La oveja se movió 2 espacios, pasó de [2, 0] a [0, 1] pasando por [1, 1].
- El zoroo se movió 2 espacios, de [2, 2] a [1, 1] pasando por [1, 2].

5.12. Alimentación de los animales

En este juego como ya se mencionó anteriormente los animales pueden comer, en este caso los lobos pueden comer cualquier otro animal y los zorros pueden comer ratones, esto por el lado de los carnívoros; los animales herbívoros comen zacate.

Para realizar esta simulación quitamos la capacidad de los animales de moverse para que se hiciera más evidente el comportamiento de comer, por tanto tenemos los siguientes resultados.

```
Estado al inicio del día 1:

Posición: [0, 0]
Nivel zacate: 25
Ocupante:  LH
Energia: 100
Sexo: Hembra

Posición: [0, 1]
Nivel zacate: 75
Ocupante:  RM
Energia: 25
Sexo: Macho

Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante: Vacío

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  LH
Energia: 100
Sexo: Hembra
```

Figura 26: Estado del sistema al inicio del día 1.

```
Posición: [1, 2]
Nivel zacate: 25
Ocupante:  RH
Energia: 25
Sexo: Hembra

Posición: [2, 0]
Nivel zacate: 75
Ocupante:  OM
Energia: 75
Sexo: Macho

Posición: [2, 1]
Nivel zacate: 0
Ocupante:  OM
Energia: 75
Sexo: Macho

Posición: [2, 2]
Nivel zacate: 0
Ocupante:  ZH
Energia: 50
Sexo: Hembra
```

Figura 27: Estado del sistema al inicio del día 1.

```
Estado al final del día 1:

Posición: [0, 0]
Nivel zacate: 25
Ocupante:  LH
Energia: 99
Sexo: Hembra

Posición: [0, 1]
Nivel zacate: 75
Ocupante: Vacío

Posición: [0, 2]
Nivel zacate: 0
Ocupante: Vacío

Posición: [1, 0]
Nivel zacate: 100
Ocupante: Vacío

Posición: [1, 1]
Nivel zacate: 50
Ocupante:  LH
Energia: 99
Sexo: Hembra
```

Figura 28: Estado del sistema al inicio del día 1.

```
Posición: [1, 2]
Nivel zacate: 20
Ocupante: Vacío

Posición: [2, 0]
Nivel zacate: 75
Ocupante: Vacío

Posición: [2, 1]
Nivel zacate: 0
Ocupante:  OM
Energia: 74
Sexo: Macho

Posición: [2, 2]
Nivel zacate: 0
Ocupante:  ZH
Energia: 49
Sexo: Hembra
```

Figura 29: Estado del sistema al inicio del día 1.

En el caso de los animales carnívoros podemos observar en las figuras 26 y 27 que la loba ubicada

en [0, 0] tiene a su alrededor un ratón macho, la loba ubicada en [1, 1] tiene 2 ratones y 2 ovejas a su alrededor y el zorro tiene un ratón que puede comer.

Si observamos al final del día en las figuras 28 y 29 observamos lo siguiente:

- La loba de la posición [0, 0] se comió al único animal que podía comerse, el ratón ubicado en [0, 1].
- La loba de la posición [1, 1] se comió a la oveja ubicada en [2, 0].
- El zorro se comió al ratón ubicado en la posición [1, 2].

Ahora si buscamos el comportamiento de algún animal herbívoro e ignoramos el hecho de que un lobo o un zorro se lo vaya a comer, vemos en las figuras 30 y 31 que hay un ratón en la posición [0, 1] y tiene 75 puntos de zacate en su posición al inicio del día. Notamos que al final del día en la figura 31 que el nivel del zacate disminuyó en 5, esto es por que el ratón come 5 puntos de zacate por día.

```
Posición: [0, 1]
Nivel zacate: 75
Ocupante:  RM
Energia: 25
Sexo: Macho
```

Figura 30: Estado del sistema al inicio del día 1.

```
Posición: [0, 1]
Nivel zacate: 70
Ocupante:  RM
Energia: 24
Sexo: Macho
```

Figura 31: Estado del sistema al inicio del día 1.

5.13. Impresión de plantilla

Para hacer la prueba de esta función plantilla se crearon 3 objetos y se les llamó a la función como se observa a continuación:

```
Lobo* l1 = new Lobo(1, 1, 1);
Zorro* z1 = new Zorro(5, 3, 2);
Celda* c1 = new Celda(33, "Cachorro de perro", 6, 9);
printTemplate(l1);
printTemplate(z1);
printTemplate<Celda>(c1);
```

Esto nos da como resultado lo que observamos en la figura 32.

```
Bienvenido al  Juego de la Vida!
May the odds be in  your favor..!

0x1a1c260
Posicion: [1, 1]
Animal: Lobo
Sexo: macho
Energía: 100

0x1a1c2c0
Posicion: [5, 3]
Animal: Zorro
Sexo: hembra
Energía: 50

Posición: [6, 9]
Nivel zacate: 33
Ocupante: Cachorro de perro
```

Figura 32: Impresión de la función plantilla.

Como se observa en la figura 32 aunque se pasen como argumento diferentes tipos de objetos, la función es capaz de imprimirlos.

6. Conclusiones

- Fue posible construir un programa que simulara un ecosistema con animales los cuales ejecutan algunas funciones básicas de supervivencia como alimentarse, reproducirse, desplazarse y morirse.
- La lógica del juego funciona de manera correcta y la interacción de unos animales con otros ocurre de manera coherente y razonable dentro de las normas naturales en el ecosistema creado. Es decir, se reproducen con animales de su misma especie, animales pequeños no cazan animales grandes, no se desplazan a espacios ocupados, entre otros.
- Las características que presenta la programación orientada a objetos y específicamente en C++, como la herencia y el polimorfismo, además de otras cualidades como la sobrecarga de operadores y plantillas son herramientas que facilitan la creación de programas y permiten minimizar la cantidad de lineas de código.
- Dentro de la programación, la documentación es un importante elemento que permite a otros usuarios y los autores mismos entender el trabajo que se ha hecho. Por lo tanto es siempre importante mantener el código bien documentado y ordenado.