Reporte Laboratorio 0:
El juego de la vida

Luis Adrian Aguilar Casacante - B00092
Robin González - B43011
Giancarlo Marín - B54099

31 de enero de 2017

Índice

---

1. Introducción

EL juego de la vida es un programa que busca simular el comportamiento un ecosistema biológico simple, para dicho fin es necesaria la programación orientada a objetos según lo resolvemos en el presente código en el lenguaje c++. El código incluido en este PDF contiene comentarios que explican su funcionamiento.

1.1. Objetivos

Crear un "juego de la vida"mediante POO en c++ utilizado la materia vista en clase.

2. Diagrama de clases

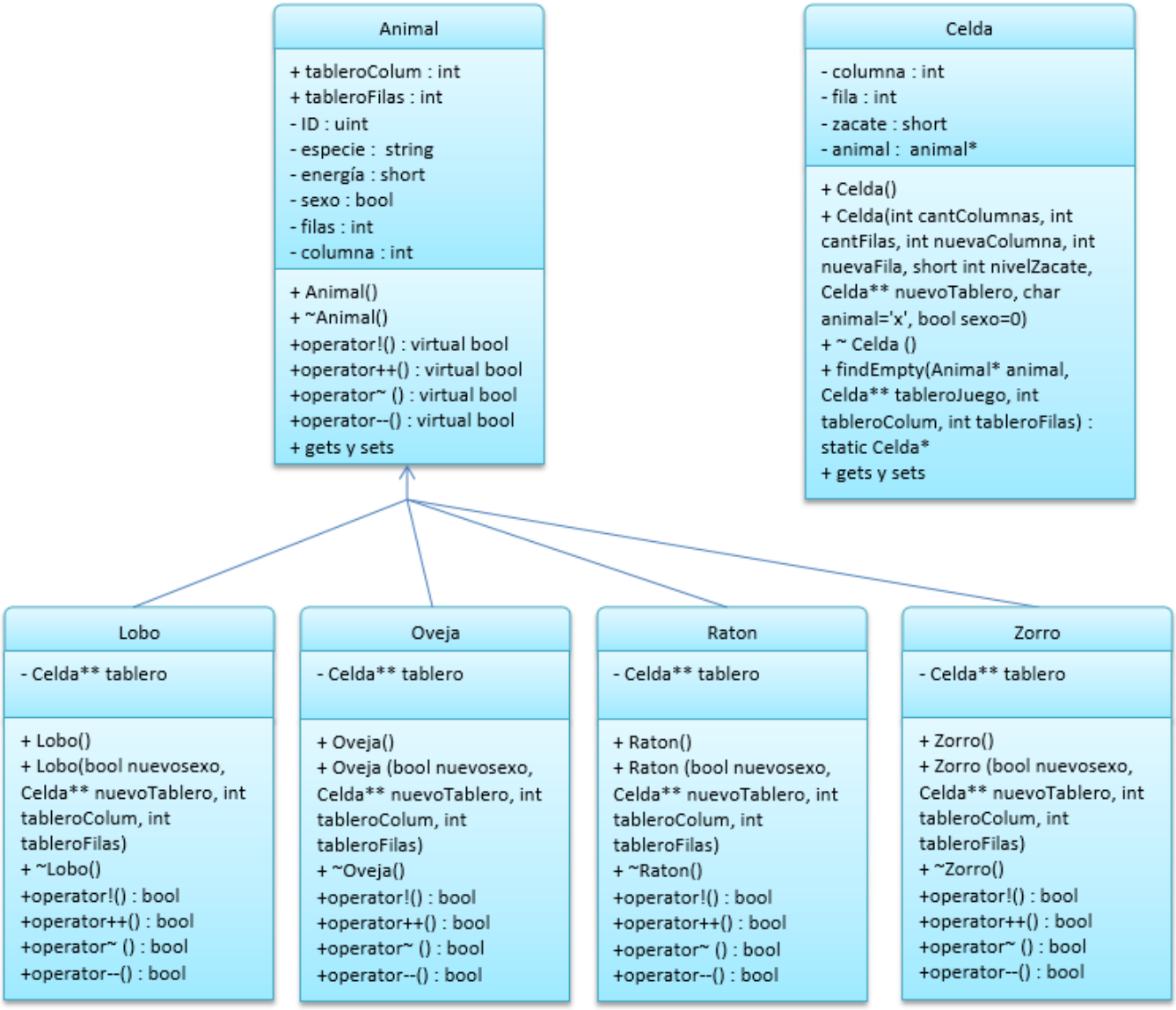

Figura 1: Diagrama de las clases utilizadas para la solución del laboratorio

3. Solución al reto

3.1. Clases

Se implementarion las clases ya planteadas en el diagrama de clases mediante el siguiente código de headers y archivos de código.

3.1.1. Animal

Animal.hpp

```
/*
 *@author Giancarlo Marin
 *@date  01-24-2017
 *@brief Archivo encabezado con la definicion de la clase Animal
*/

#ifndef ANIMAL_HPP
#define ANIMAL_HPP
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cstddef>
#include <string>
#include <sstream>

using namespace std;

static unsigned int identificador=0; //variable global para generar ID de cada animal

/*
 * Definicion de clase Animal
*/
class Animal{
public:
   //Metodos
   Animal(); //Constructor de la clase
   virtual ~Animal(); //Destructor de la clase
   //Sobrecarga de operadores para funcionamiento de funciones
   //True=Si realizo la funcion o FALSE de lo contrario
   virtual bool operator!()=0;//Moverse
   virtual bool operator++()=0;//Comer
   virtual bool operator~()=0;//Reproducirse
   virtual void operator--()=0;//Morir
   //metodos set y get de los atributos
   unsigned int getID();
   void setEspecie(string nuevaEspecie);
   string getEspecie();
   void setEnergia(short int nuevaEnergia);
   short int getEnergia();
   void setSexo(bool nuevoSexo);
   bool getSexo();
   void setFila(int nuevaFila);
   void setColumna(int nuevaFila);
   int getFila();
   int getColumna();
private:
   //Atributos
   unsigned int ID;
   string especie;
   short int energia;
```

```
   bool sexo;
   int fila;
   int columna;
};

#endif /* ANIMAL_HPP */
```

Animal.cpp

```
/*
 *@author Giancarlo Marin
 *@date  01-25-2017
 *@brief Implementacion de metodos de las clase Animal
*/

#include "../include/Animal.hpp"

/*
 * Constructor por defecto de la clase Animal
*/
Animal::Animal(){
   cout << "Creando Animal #" << identificador << endl;
   ID=identificador;
   identificador++;
}

/*
 * Destructor por defecto de la clase Animal
*/
Animal::~Animal(){
   cout << "Destruyendo Animal #" << getID() << endl;
}

/*
 * Metodo get del identifiacor del animal
 *@return unsigned int  ID del Animal
*/
unsigned int Animal::getID(){
   return ID;
}
/*
 * Metodo set del atributo especie
 *@param string  Definicion del Nuevo valor de energia asignado al animal
*/
void Animal::setEspecie(string nuevaEspecie){
   especie=nuevaEspecie;
}

/*
 * Metodo get del atributo especie
 *@return string  Nombre de la especie del animal
*/
string Animal::getEspecie(){
```

```
   return especie;
}

/*
 * Metodo set del atributo energia
 *@param short int  Nuevo valor de energia asignado al animal
*/
void Animal::setEnergia(short int nuevaEnergia){
   this->energia=nuevaEnergia;
}

/*
 * Metodo get del atributo energia
 *@return short int  Valor de energia asignado al animal
*/
short int Animal::getEnergia(){
   return energia;
}

/*
 * Metodo set del atributo sexo
 *@param bool  Sexo por asignar al animal
*/
void Animal::setSexo(bool nuevoSexo){
   sexo=nuevoSexo;
}

/*
 * Metodo get del atributo sexo
 *@return bool  Retorna el sexo del animal de la siguiente forma:
 * False=Macho True=Hembra
*/
bool Animal::getSexo(){
   return sexo;
}

/*
 * Metodo set del atributo fila
 *@param int  Nueva asignacion del espacio fila donde se encuentra el animal
*/
void Animal::setFila(int nuevaFila){
   this->fila=nuevaFila;
}

/*
 * Metodo set del atributo columna
 *@param int  Nueva asignacion del espacio columna donde se encuentra el animal
*/
void Animal::setColumna(int nuevaColumna){
   this->columna=nuevaColumna;
}

/*
 * Metodo get del atributo fila
```

```
 *@return int  Retorna la fila en la que se encuentra el animal
*/
int Animal::getFila(){
   return fila;
}

/*
 * Metodo get del atributo columna
 *@return int  Retorna la columna en la que se encuentra el animal
*/
int Animal::getColumna(){
   return columna;
}
```

3.1.2. Celda

Celda.hpp

```
/*
 *@author Giancarlo Marin
 *@date  01-24-2017
 *@brief Archivo encabezado con la definicion de la clase Celda
*/

#ifndef CELDA_HPP
#define CELDA_HPP
#include <iostream>
#include <cstdlib>
#include "Animal.hpp"

using namespace std;

/*
 * Definicion de clase Celda
*/
class Celda{
public:
   //Metodos
   Celda(); //Constructor de la clase
   Celda(int nuevaColumna, int nuevaFila, short int nivelZacate, Celda** nuevoTablero,
       char animal='x', bool sexo=0);
   ~Celda(); //Destructor de la clase
   //metodos set y get de los atributos
   int getFila();
   int getColumna();
   void setZacate(short int nivelZacate);
   short int getZacate();
   void setAnimal(Animal* nuevoAnimal);
   Animal& getAnimal();
   short int getContadorDias();
private:
   //Atributos
```

```
   int columna;
   int fila;
   short int zacate;
   Animal* animal;
   short int contadorDias;
};

#endif /* CELDA_HPP */
```

Celda.cpp

```
/*
 *@author Giancarlo Marin
 *@date  01-25-2017
 *@brief Implementacion de metodos de las clase Celda
*/

#include "../include/Celda.hpp"
#include "../include/print.hpp"
#include "../include/Lobo.hpp"
#include "../include/Oveja.hpp"
#include "../include/Raton.hpp"
#include "../include/Zorro.hpp"


/*
 * Constructor por defecto de la clase Celda
*/
Celda::Celda(){
   cout << "Creando Celda" << endl;
}

/*
 * Constructor sobrecargado de la clase Celda
*/
Celda::Celda(int nuevaColumna, int nuevaFila, short int nivelZacate, Celda**
    nuevoTablero, char animal, bool sexo){
   this->columna=nuevaColumna;
   this->fila=nuevaFila;
   this->zacate=nivelZacate;
   if (animal=='L'){
      Lobo* nuevoLobo = new Lobo(sexo, nuevoTablero);
      setAnimal(nuevoLobo);
   }
   else if(animal == 'O'){
      Oveja* nuevaOveja = new Oveja(sexo, nuevoTablero);
      setAnimal(nuevaOveja);
   }
   else if(animal == 'Z'){
      Zorro* nuevoZorro = new Zorro(sexo, nuevoTablero);
      setAnimal(nuevoZorro);
   }
   else if(animal == 'R'){
```

```
      Raton* nuevoRaton = new Raton(sexo, nuevoTablero);
      setAnimal(nuevoRaton);
   }
   else{
      //this->animal=nullptr;
   }
   this->contadorDias=0;
}

/*
 * Destructor por defecto de la clase Celda
*/
Celda::~Celda(){
   cout << "Destruyendo Celda " << fila << " " << columna << " y sus componentes:" <<
       endl;
   delete this->animal;
}

/*
 * Metodo get de la fila de la Celda
 *@return int  fila de la Celda
*/
int Celda::getFila(){
   return fila;
}

/*
 * Metodo get de la columna de la Celda
 *@return int  columna de la Celda
*/
int Celda::getColumna(){
   return columna;
}

/*
 * Metodo set del atributo zacate
 * @param short int  Nuevo valor de zacate de la Celda
*/
void Celda::setZacate(short int nivelZacate){
   this->zacate=nivelZacate;
}

/*
 * Metodo get del atributo zacate
 *@return short int  Nivel de zacate de la Celda
*/
short int Celda::getZacate(){
   return zacate;
}

/*
 * Metodo get del atributo contador de dias
 *@return short int  Numero de dias que ha estado sin cambios la celda
*/
```

```
short int Celda::getContadorDias(){
   return contadorDias;
}

/*
 * Metodo set del atributo animal
 * @param *Animal  Puntero a una instancia Animal
*/
void Celda::setAnimal(Animal* nuevoAnimal){
   this->animal=nuevoAnimal;
}

/*
 * Metodo get del atributo animal
 *@return &Animal  Direccion de memoria del Puntero Animal en la celda
*/
Animal& Celda::getAnimal(){
   return *(this->animal);
}
```

3.1.3. Lobo

Lobo.hpp

```
/*
 *@author Giancarlo Marin
 *@date  01-24-2017
 *@brief Archivo encabezado con la definicion de la clase Lobo
*/

#ifndef LOBO_HPP
#define LOBO_HPP
#include <iostream>
#include "Celda.hpp"

using namespace std;

/*
 * Definicion de clase Lobo
*/
class Lobo : public Animal{
public:
   //Metodos
   Lobo(); //Constructor de la clase
   Lobo(bool nuevoSexo, Celda** nuevoTablero); //Constructor sobrecargado de la clase
   virtual ~Lobo(); //Destructor de la clase
   //Sobrecarga de operadores para funcionamiento de funciones
   bool operator!();
   bool operator++();
   bool operator~();
   void operator--();
private:
```

```
   Celda** tablero;
};

#endif /* LOBO_HPP */
```

Lobo.cpp

```
/*
 *@author Luis Adrian Aguilar Cascante, @autor Giancarlo Marin Hernandez
 *@date  01-26-2017
 *@brief Implementacion de los metodos de la clase Lobo
*/

#include "../include/Lobo.hpp"

/*
 *Constructor por defecto de la clase Lobo
*/
Lobo::Lobo() {
}

/*
 * Constructor sobrecargado de la clase Lobo
 * @param bool  Sexo del Lobo
*/
Lobo::Lobo(bool nuevoSexo, Celda** nuevoTablero) {
    setEnergia(100); //Definido por el enunciado Regla 1
    setSexo(nuevoSexo);
    setEspecie("Lobo");
   this->tablero = nuevoTablero;
}

/*
 * Destructor por defecto de la clase Lobo
*/
Lobo::~Lobo() {
}

/*
 * Metodo moverse del Lobo definido por las regla
*/
bool Lobo::operator!(){
   return true;
}

bool Lobo::operator++(){
   return true;
}

bool Lobo::operator~(){
   return true;
}

void Lobo::operator--(){
```

```
}
```

3.1.4. Oveja

Oveja.hpp

```
/*
 *@author Giancarlo Marin
 *@date  01-24-2017
 *@brief Archivo encabezado con la definicion de la clase Oveja
*/

#ifndef OVEJA_HPP
#define OVEJA_HPP
#include <iostream>
#include "Animal.hpp"
#include "Celda.hpp"

using namespace std;

/*
 * Definicion de clase Oveja
*/
class Oveja : public Animal{
public:
   //Metodos
   Oveja(); //Constructor de la clase
   Oveja(bool nuevoSexo, Celda** nuevoTablero); //Constructor sobrecargado de la clase
   virtual ~Oveja(); //Destructor de la clase
   //Sobrecarga de operadores para funcionamiento de funciones
   bool operator!();
   bool operator++();
   bool operator~();
   void operator--();
private:
   Celda** tablero;
};

#endif /* OVEJA_HPP */
```

Oveja.cpp

```
/*
 *@author Luis Adrian Aguilar Cascante, @autor Giancarlo Marin Hernandez
 *@date  01-26-2017
 *@brief Implementacion de los metodos de la clase oveja
*/

#include "../include/Oveja.hpp"

/*
 *Constructor por defecto de la clase oveja
```

```
*/
Oveja::Oveja() {
}

/*
 * Constructor sobrecargado de la clase oveja
 * @param bool  Sexo del oveja
*/
Oveja::Oveja(bool nuevoSexo, Celda** nuevoTablero) {
    setEnergia(100); //Definido por el enunciado Regla 1
    setSexo(nuevoSexo);
    setEspecie("Oveja");
   this-> tablero = nuevoTablero;
}

/*
 * Destructor por defecto de la clase Oveja
*/
Oveja::~Oveja() {
}

/*
 * Metodo moverse del oveja definido por las regla
*/
bool Oveja::operator!(){
   return true;
}

bool Oveja::operator++(){
   return true;
}

bool Oveja::operator~(){
   return true;
}

void Oveja::operator--(){

}
```

3.1.5. Raton

Raton.hpp

```
/*
 *@author Giancarlo Marin
 *@date  01-24-2017
 *@brief Archivo encabezado con la definicion de la clase Raton
*/

#ifndef RATON_HPP
#define RATON_HPP
#include <iostream>
```

```
#include "Animal.hpp"
#include "Celda.hpp"

using namespace std;

/*
 * Definicion de clase Raton
*/
class Raton : public Animal{
public:
   //Metodos
   Raton(); //Constructor de la clase
   Raton(bool nuevoSexo, Celda** nuevoTablero); //Constructor sobrecargado de la clase
   virtual ~Raton(); //Destructor de la clase
   //Sobrecarga de operadores para funcionamiento de funciones
   bool operator!();
   bool operator++();
   bool operator~();
   void operator--();
private:
   Celda** tablero;
};

#endif /* RATON_HPP */
```

Raton.cpp

```
/*
 *@author Luis Adrian Aguilar Cascante, @autor Giancarlo Marin Hernandez
 *@date  01-26-2017
 *@brief Implementacion de los metodos de la clase Raton
*/

#include "../include/Raton.hpp"

/*
 *Constructor por defecto de la clase Raton
*/
Raton::Raton() {
}

/*
 * Constructor sobrecargado de la clase Raton
 * @param bool  Sexo del Raton
*/
Raton::Raton(bool nuevoSexo, Celda** nuevoTablero) {
    setEnergia(100); //Definido por el enunciado Regla 1
    setSexo(nuevoSexo);
    setEspecie("Raton");
   this-> tablero = nuevoTablero;
}

/*
 * Destructor por defecto de la clase Raton
```

```
*/
Raton::~Raton() {
}

/*
 * Metodo moverse del Raton definido por las regla
*/
bool Raton::operator!(){
   return true;
}

bool Raton::operator++(){
   return true;
}

bool Raton::operator~(){
   return true;
}

void Raton::operator--(){

}
```

3.1.6. Zorro

Zorro.hpp

```
/*
 *@author Giancarlo Marin
 *@date  01-24-2017
 *@brief Archivo encabezado con la definicion de la clase Zorro
*/

#ifndef ZORRO_HPP
#define ZORRO_HPP
#include <iostream>
#include "Animal.hpp"
#include "Celda.hpp"

using namespace std;

/*
 * Definicion de clase ZORRO
*/
class Zorro : public Animal{
public:
   //Metodos
   Zorro(); //Constructor de la clase
   Zorro(bool nuevoSexo, Celda** nuevoTablero); //Constructor sobrecargado de la clase
   virtual ~Zorro(); //Destructor de la clase
   //Sobrecarga de operadores para funcionamiento de funciones
   bool operator!();
   bool operator++();
```

```
   bool operator~();
   void operator--();
private:
   Celda** tablero;
};

#endif /* ZORRO_HPP */
```

Zorro.cpp

```
/*
 *@author Luis Adrian Aguilar Cascante, @autor Giancarlo Marin Hernandez
 *@date  01-26-2017
 *@brief Implementacion de los metodos de la clase Zorro
*/

#include "../include/Zorro.hpp"

/*
 *Constructor por defecto de la clase Zorro
*/
Zorro::Zorro() {
}

/*
 * Constructor sobrecargado de la clase Zorro
 * @param bool  Sexo del Zorro
*/
Zorro::Zorro(bool nuevoSexo, Celda** nuevoTablero) {
    setEnergia(100); //Definido por el enunciado Regla 1
    setSexo(nuevoSexo);
    setEspecie("Zorro");
   this-> tablero = nuevoTablero;
}

/*
 * Destructor por defecto de la clase Zorro
*/
Zorro::~Zorro() {
}

/*
 * Metodo moverse del Zorro definido por las regla
*/

/*
 * Metodo moverse del Zorro definido por las regla
*/
bool Zorro::operator!(){
   return true;
}

bool Zorro::operator++(){
   return true;
```

```
}

bool Zorro::operator~(){
   return true;
}

void Zorro::operator--(){

}
```

3.2. Main

En el método main ocurre también una parte fundamental del proceso la cual consiste en leer de un archivo de texto pasado como argumento la configuración inicial del juego de la vida, en este método se debe inicializar la matriz y todas las instancias de objetos necesarios para que corra el juego, además también se lee como primer argumento para el ejecutable la cantidad de iteraciones que se desea ejecutar el juego, llamadas días.

main.cpp

```
/*
 *Universidad de Costa Rica - Escuela de Ingenieria Electrica
 *IE-0217 - Lab0 - Grupo01
 *@author Giancarlo Marin, @author Luis Adrian Aguilar, @author Robin Gonzalez
 *@date  01-26-2017
 *@brief Clase main que prubea el funconamiento del juego de ecologia
*/

#include "../include/Celda.hpp"
#include "../include/print.hpp"

//Variables globales que indican las filas, columnas y tablero de juego
static int tFilas=0;
static int tColumnas=0;
static Celda** tablero=NULL;

/*
 * Metodo main que prueba el funcionamieno del juego de ecologia
 * @param int   Cantidad de parametros obtenidos en la ejecucion del programa
 * @param char** Vector de char* con los parametros enviados en la ejecucion
*/
int main(int argc, char *argv[]){
   int dias, fila, columna;
   short int zacate;
   bool genero;
   char animal, sexo;
   dias = atoi(argv[1]);
   //lectura del archivo de configuracion del juego
   ifstream file(argv[2]);
   string str,temp;
   getline(file, str);
   tFilas = atoi(str.c_str());//.c_srt() metodo para convertir string a vector de
       caracteres
```

```
   getline(file, str);
   tColumnas = atoi(str.c_str());
   //inicializacion de la matriz de objetos tipo celda
   tablero = new Celda*[tFilas];
   for(int i=0; i<tFilas;i++){
      tablero[i] = new Celda[tColumnas];
   }
   while (getline(file, str))//Mientras se encuentre texto al leer el archivo guardar
       cada linea en el string str
   {
      stringstream test(str);
      //separa el stringstream cada caracter por un tab y lo guarda en temp
      getline(test, temp,'\t');
      fila = atoi(temp.c_str());
      getline(test, temp,'\t');
      columna = atoi(temp.c_str());
      getline(test, temp,'\t');
      zacate = atoi(temp.c_str());
      getline(test, temp,'\n');
      if(temp != "\0"){
         //si existe animal
         animal = temp[0];
         sexo = (int) temp[1];
         if(sexo == 72){
            //el char 'H' tiene el valor entero: 72 que indica genero femenino (TRUE)
            genero = 1;
         }
         else{
            genero = 0;
         }
      }
      else{
         //caso en que no haya un animal en una celda
         animal='x';
         genero=0;
      }
      //Asignacion de informacion a cada celda
      Celda* nuevaCelda= new Celda(tColumnas, tFilas, columna, fila, zacate, tablero,
          animal, sexo);
      tablero[fila][columna]=*nuevaCelda;
      cout << fila << "\t" << columna << "\t" << zacate << "\t" <<
      animal << sexo << endl;
}
```

En las lineas anteriores se inicializan los objetos necesarios para el juego. En las siguientes lineas se lleva a cabo el juego en sí:

```
int contDias=0;
   while (dias>0){
      cout << "estoy en el while: " << dias << endl;
      contDias++;
      for (int i=0;i<tFilas;i++){
         for (int j=0;j<tColumnas;j++){
            Animal* animal=tablero[i][j].getAnimal(); //puntero al animal de la celda
```

```
      cout << "tengo al animal: " << animal->getID() << endl;
      if (animal!=NULL){ //si hay un animal
      //determinar la especie y realizar sus cuatro funciones
         if (animal->getEspecie()=="Lobo"){
            if (!animal && !animal){
               !animal; //los lobos se mueven 3 espacios al dia
               cout << "Los lobos se mueven" << endl;
            }
         }
         else if (animal->getEspecie()=="Oveja"){
            if (!animal){
               !animal; //las ovejas se mueven 2 espacios al dia
            }
         }
         else if(animal->getEspecie()=="Zorro"){
            if (!animal){
               !animal; //los zorros se mueven 2 espacios al dia
            }
         }
         else{
            cout << "Soy un Raton" << endl;
            !animal; //los ratones se mueven 1 espacio al dia
         }

      //Regla 5: La energia del animal se disminuye en 1 por dia
         animal->setEnergia(animal->getEnergia()-1);
         cout << "Tengo menos energia";
      }
   //Regla 6: El zacate aumenta en 5 su nivel cada 3 dias
      if (contDias == 3 && tablero[i][j].getZacate()>0){ //Regla 8
         tablero[i][j].setZacate(tablero[i][j].getZacate()+5);
      //Regla 7: El nivel de Zacate no puede ser mayor a 100
         if (tablero[i][j].getZacate()>100){
            tablero[i][j].setZacate(100);
         }
      }
   }
}
cout << "--- RESUMEN DIA " << dias << " ---" << endl;
for (int i=0;i<tFilas;i++){
   for (int j=0;j<tColumnas;j++){
      cout << i << "\t" << j << "\t" << tablero[i][j].getZacate() << "\t" <<
      (tablero[i][j].getAnimal())->getEspecie() <<
      (tablero[i][j].getAnimal())->getSexo() << endl;
   }
}
cout << "--- RESUMEN ANIMALES " << dias << " ---" << endl;
for (int i=0;i<tFilas;i++){
   for (int j=0;j<tColumnas;j++){
      Animal* animal=tablero[i][j].getAnimal(); //puntero al animal de la celda
      if (animal!=NULL){
         printVivo(animal);
      }
   }
```

```
      }
      dias--;
   }
   return 0;
}
```

3.3. Template print

Se creó una plantilla para poder usar un mismo método (printVivo) con diferentes tipos de variables (objetos de las clases hijas), de modo que se muestre información propia de cada uno. Print.hpp

```
/*
 *Template de impresion de seres vivos
*/

#ifndef PRINT_HPP
#define PRINT_HPP
#include <iostream>
#include <cstdlib>

using namespace std;

template <typename T>

void printVivo(T *serVivo){
   cout << "Sexo: " << serVivo->getSexo() <<
   " Energia: " << serVivo->getEnergia() <<
   " Energia: " << serVivo->getEnergia() << endl;
}

#endif /*PRINT_HPP*/
```

3.4. Makefile

Con el siguiente código y comandos el makefile realiza las acciones deseadas.

```
CXX= g++
RM= rm -f
CXXFLAGS= -Wall -g -std=c++11
CONFIG_FILE= ../include/config.txt
DAYS= 10
SRCS= ../src/main.cpp ../src/Animal.cpp ../src/Celda.cpp ../src/Lobo.cpp
    ../src/Oveja.cpp ../src/Raton.cpp ../src/Zorro.cpp
OBJS= $(SRCS:.cpp=.o)
#Nombre del archivo ejecutable
MAIN= Juego

.PHONY: depend clean

all: $(MAIN) run clean
```

```
$(MAIN): $(OBJS)
   $(CXX) $(OBJS) -o $(MAIN)
   @echo ----- PROYECTO COMPILADO ARCHIVO EJECUTABLE JUEGO GENERADO -----

.cpp.o:
   $(CXX) $(CXXFLAGS) -c $< -o $@

run:
   @echo ----- EJECUTANDO JUEGO -----
   ./$(MAIN) $(DAYS) $(CONFIG_FILE)

clean:
   @echo ----- LIMPIANDO ARCHIVOS INTERMEDIOS Y ARCHIVO EJECUTABLE -----
$(RM) ../src/*.o ../src/*~ $(MAIN)
```

- Make : compila el código.
- Make run : ejecuta el código.
- Make clean : borra archivos intermedios y el ejecutable.

4. Conclusiones

- La Poo fue la herramienta ideal para realizar un proyecto como éste ya que se requieren varios entes diferentes con propiedades similares que se alteran durante la ejecución del código.
- Se necesitó de la implementación de una plantilla para que una función pudiera recibir de argumento diferentes tipos de objetos sin problema, siempre y cuando tuvieran las características necesarias en común.