Memoria de Trabajo de Prácticas
Mundo Wumpus

Procesadores del Lenguaje
Primer curso del Segundo Ciclo de Ingeniería Informática (2011-2012)
Escuela Politécnica Superior de Córdoba
Universidad de Córdoba
Rubén Salado Cid
José Carlos Garrido Pérez

28 de mayo de 2012

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Descripción del entorno a simular

En la presente memoria de prácticas va a comentarse cuáles han sido los pasos realizados para acometer el desarrollo de un intérprete que acepte y ejecute las sentencias necesarias para poder, en dos modos de ejecución distintos, crear niveles del juego "Mundo Wumpus" y también para que el usuario pueda jugar los niveles creados, moviendo al personaje por el mapa creado. Este entorno deberá, además, permitir la declaración de variables, el uso de esquemas propios de la programación como los esquemas condicionales, bucles de diverso tipo, operaciones lógicas y aritméticas, etc. En la medida de lo posible, deberá recuperarse de los errores e informar al usuario del problema acontecido, así como de las posibles soluciones por las que puede optarse.

A grandes rasgos, el entorno que se va a simular con el intérprete cuenta con los dos modos siguientes:

- **Modo de configuración**: denominado *configurationMode*, es el modo en el que se podrán crear y editar los niveles, con total libertad, configurando aspectos tales como el tamaño del tablero, las posiciones de los hoyos, tesoros, Wumpus, etc. También podrá configurarse el número de flechas que se desea que tenga el jugador a su disposición y las posiciones de comienzo y fin del nivel.

- **Modo de aventura**: denominado *adventureMode*, en el cual, el usuario solamente podrá realizar las acciones que están permitidas a un jugador del juego Mundo Wumpus, esto es, mover al aventurero (al que se le ha dado el nombre ficticio de *Mecca*) por el mapa, disparar para intentar matar al Wumpus, saber cuántos tesoros quedan, etc. En este modo, el jugador recibirá además información por parte del intérprete de los elementos que hay en el mapa, dependiendo de la casilla en el que se encuentre y de los resultados de sus movimientos o disparos. Por ejemplo, si se encuentra cerca del Wumpus se le informará de que existe un hedor, para que sepa que está en una casilla cercana. También se informará, en caso de que el juego llegue a su fin, de que ha logrado escapar del nivel con todos los tesoros existentes o, por el contrario, de que ha muerto.

1.2. Fases del proceso de generación del intépre-te

El proceso de generación de este intérprete ha contado con las siguientes fases:

1. Elección del entorno
2. Diseño del lenguaje
3. Desarrollo del análisis léxico
4. Desarrollo del análisis sintáctico
5. Atributos y funciones auxiliares
6. Elegir modo de ejecución

Capítulo 2

Definición del lenguaje diseñado

2.1. Sentencias o instrucciones para simular el entorno elegido

Como hay dos modos de ejecución, se van a comentar primero las funciones que son comunes a ambos modos:

- getArrows(): debe devolver el número de flechas que tiene el aventurero.
- getMecca(): debe devolver la posición de Mecca en el mapa.
- getTotalTreasures(): debe devolver el número total de tesoros.

Las funciones que pueden ser llamadas sólo desde el modo de configuración son:

- setBoardSize(rows, cols): debe hacer que el tamaño del tablero sea el que recibe como parámetro.
- setTreasure(row, col): ha de permitir insertar un tesoro, en caso de que sea posible, en la posición que recibe como parámetro.

- getTreasure(number): debe devolver la posición del tesoro cuyo número se le pasa como parámetro.
- setHole(row, col): debe permitir insertar un hoyo, en caso de que sea posible, en la posición que recibe como parámetro.
- getNumberOfHoles(): devolverá el número de hoyos existentes en el mapa.
- getHole(number): ha de devolver la posición del hoyo cuyo número se le pasa como parámetro.
- setWumpus(row, col): ha de permitir colocar el Wumpus en la posición que se le pasa como parámetro.
- getWumpus(): deovlverá la posición del Wumpus.
- setStart(row, col): deberá permitir colocar la casilla de comienzo en la posición que se le pasa como parámetro.
- getStart(): debe devolver la posición de comienzo del mapa.
- setExit(row, col): deberá permitir colocar la casilla de salida en la posición que se le pasa como parámetro.
- getExit(): debe devolver la posición de salida del mapa.
- incArrows(number): incrementa el número de flechas de Mecca en una cantidad igual a la que reciba como parámetro.
- decArrows(number): decrementa el número de flechas de Mecca en una cantidad igual a la que reciba como parámetro.

Desde el modo aventura pueden llamarse las siguientes funciones:

- Funciones que se usan para el movimiento del aventurero Mecca:
  - goUp(): moverá a Mecca una casilla hacia arriba.
  - goRight(): moverá a Mecca una casilla hacia la derecha.
  - goLeft(): moverá a Mecca una casilla hacia la izquierda.
  - goDown(): moverá a Mecca una casilla hacia abajo.
- Funciones que se usan para que Mecca dispare:

- shootUp(): permitirá disparar hacia arriba.
- shootRight(): permitirá disparar hacia la derecha.
- shootLeft(): permitirá disparar hacia la izquierda.
- shootUp(): permitirá disparar hacia abajo.

2.2. Sentencias del lenguaje de programación de pseudocódigo

Las sentencias del lenguaje de programación serán:

- Sentencia condicional simple
- Sentencia condicional compuesta
- Bucle "mientras"
- Bucle "repetir"
- Bucle "para"

Dado que las otras órdenes serán en inglés, se ha decidido, para mantener una cierta homogeneidad en este aspecto, que estas sentencias también usen ese idioma.

2.2.1. Sentencia condicional simple

if condicion **then**
sentencias
end_if;

2.2.2. Sentencia condicional compuesta

if condicion **then**
sentencias

else
sentencias
end_if;

2.2.3. Bucle “mientras”

while condicion **do**
sentencias
end_while;

2.2.4. Bucle “repetir”

repeat
sentencias
until condicion**;**

2.2.5. Bucle “para”

for identificador
from expresión numérica 1
until expresión numérica 2
step expresión numérica 3
do
sentencias
end_for;

Capítulo 3

Descripción de la gramática asociada al lenguaje definido

Capítulo 4

Descripción del intérprete ANTLR construido para la gramática

4.1. Analizador léxico: componentes léxicos

El analizador léxico se ha codificado en el fichero *Meccalex.g* y cuenta con la definición de los siguientes *tokens*:

- Modos de utilización del intérprete

```
// Modos de utilizaci n del int rprete
BEGIN_CONF          = "ConfigurationMode";
END_CONF            = "endConfigurationMode";
BEGIN_ADV           = "AdventureMode";
END_ADV             = "endAdventureMode";
```

- Funciones que pueden realizarse

```
//Funciones
FUNC_LEER = "read";
FUNC_ESCRIBIR = "write";
FUNC_SHOWBOARD = "showBoard";
FUNC_SHOWADVENTURESTATE = "showAdventureState";
```

```
FUNC_SETBOARDSIZE = "setBoardSize";
FUNC_GETBOARDROWS = "getBoardRows";
FUNC_GETBOARDCOLUMNS = "getBoardColumns";
FUNC_GETBOARDSIZE = "getBoardSize";
FUNC_SETTREASURE = "setTreasure";
FUNC_GETTOTALTREASURES = "getTotalTreasures";
FUNC_GETTREASURE = "getTreasure";
FUNC_SHOWTREASURES = "showTreasures";
FUNC_REMOVETREASURE = "removeTreasure";
FUNC_SETHOLE = "setHole";
FUNC_REMOVEHOLE = "removeHole";
FUNC_GETNUMBEROFHOLES = "getNumberOfHoles";
FUNC_GETHOLE = "getHole";
FUNC_SHOWHOLES = "showHoles";
FUNC_SETWUMPUS = "setWumpus";
FUNC_GETWUMPUS = "getWumpus";
FUNC_SETSTART = "setStart";
FUNC_GETSTART = "getStart";
FUNC_SETEXIT = "setExit";
FUNC_GETEXIT = "getExit";
FUNC_GETMECCA = "getMecca";
FUNC_GETREMAININGTREASURES = "getRemainingTreasures";
FUNC_SETARROWS = "setArrows";
FUNC_GETARROWS = "getArrows";
FUNC_INCARROWS = "incArrows";
FUNC_DECARROWS = "decArrows";
FUNC_SHOOTLEFT = "shootLeft";
FUNC_SHOOTRIGHT = "shootRighT";
FUNC_SHOOTUP = "shootUp";
FUNC_SHOOTDOWN = "shootDown";
FUNC_GOLEFT = "goLeft";
FUNC_GORIGHT = "goRight";
FUNC_GOUP = "goUp";
FUNC_GODOWN = "goDown";
```

- Palabras reservadas para los tipos de variables

```
// Tipos de variables
TIPO_CADENA  = "String";
TIPO_NUMERO = "Number";
```

- Palabras reservadas para bucles y esquemas condicionales

```
//Palabras reservadas para bucles y esquemas condicionales
RES_MIENTRAS      = "while" ;
RES_HACER         = "do" ;
RES_REPETIR           = "repeat";
RES_HASTA         = "until";
RES_FIN_MIENTRAS = "end_while" ;
RES_SI            = "if" ;
RES_ENTONCES      = "then" ;
RES_SI_NO         = "else" ;
RES_FIN_SI        = "end_if" ;
RES_PARA          = "for" ;
RES_PASO          = "step" ;
RES_DESDE         = "from" ;
RES_FIN_PARA      = "end_for" ;
```

4.2. Analizador sintáctico

El analizador sintáctico está codificado en el fichero *Meccasint.g* y está dividido en las siguientes partes.

4.2.1. Atributos de la clase Meccasint.g

La variable de tipo cadena *mode* es muy importante, ya que es la encargada de controlar en qué modo estamos dentro del intérprete. Los valores que puede contener están definidos en las tres cadenas de tipo estático del comienzo del código mostrado:

- NO_MODE: será el valor de la cadena *mode* cuando la ejecución no se encuentre en ninguno de los dos modos
- CONFIGURATION_MODE: será el valor de la cadena *mode* cuando la ejecución se encuentre en el modo de configuración
- ADVENTURE_MODE: será el valor de la cadena *mode* cuando la ejecución se encuentre en el modo de aventura

Más tarde, a la hora de llamar a las funciones dentro de las reglas sintácticas, se comprobará el modo en el que se encuentra la ejecución y se realizarán las acciones pertinentes o se mostrará un mensaje de error en caso de que esa acción no pueda realizarse en ese modo.

La variable de tipo *Board* almacena una instancia de esa clase. El cometido de esta clase es el de almacenar toda la información relativa al tablero y realizar las operaciones y movimientos que sean necesarios en él.

La variable de tipo *TablaSimbolos* almacena una instancia de esa clase. Su objetivo es el de almacenar la tabla de símbolos, con el uso de instancias de la clase *Variable* guardando el nombre del símbolo, el tipo (que puede ser *string* o *number*), y el valor asociado a este símbolo.

4.2.2. Símbolos no terminales

Símbolo mecca

El símbolo inicial de la gramática es denominado *mecca* y su regla es la siguiente:

```
mecca: (instruction)* configuration (instruction)* adventure
```

Se pueden utilizar tantas instrucciones como se desee dentro y fuera de los métodos (el caso de que puedan o no ejecutarse será controlado por la variable de tipo cadena *mode* como se ha comentado previamente).

Símbolo configuration

Este símbolo sirve para incluir todas las instrucciones realizadas en el modo de configuración, está delimitado por los símbolos no terminales BEGIN_CONF y END_CONF.

```
configuration:  BEGIN_CONF {mode = CONFIGURATION_MODE;} (inst
```

Símbolo adventure

Este símbolo sirve para incluir todas las instrucciones realizadas en el modo de aventura, está delimitado por los símbolos no terminales BEGIN_ADV y END_ADV.

```
adventure: BEGIN_ADV {if(board.initGame()) mode = ADVENTURE_MODE;}
```

Símbolo instruction

Este símbolo no terminal es el más extenso de la gramática y es en el que se encuentran todas las funciones que pueden ser llamadas. Contiene dos variables de tipo, valga la redundancia, *Variable* que se utilizan para recoger los resultados de la evaluación de las reglas de tipo *expression* y que son utilizadas dentro de las instrucciones.

4.2.3. Reglas de producción de la gramática

4.3. Análisis semántico

4.3.1. Atributos: heredados y sintetizados

4.3.2. Funciones auxiliares

Clase Position

Se trata de una clase simple cuyo cometido es el de almacenar una posición en el tablero. Los atributos con los que cuenta son los siguientes:

- x: es una variable privada de tipo entero entero que almacena la posición x de una instancia de esta clase.

- y: es una variable privada de tipo entero entero que almacena la posición y de una instancia de esta clase.

Los métodos de que dispone son:

- Position(): constructor vacío de la clase que pondrá la posición a (-1,-1).
- Position(int newX, int newY): constructor parametrizado que pondrá la posición en los valores que reciba como parámetros.
- Position(Position copyPos): constructor de copia de la clase Position.
- setX(int newX): función que recibe un valor entero y lo almacena en la variable x.
- setY(int newY): función que recibe un valor entero y lo almacena en la variable y.
- getX(): función que devuelve un entero con el valor guardado para la posición x.
- getY(): función que devuelve un entero con el valor guardado para la posición y.
- toString(): función que devuelve una cadena de texto que muestra la posición almacenada en esta instancia.
- equals(): función que devuelve true si la posición que se le pasa como parámetro es igual a esta instancia y false en caso contrario.

Clase Size

Se trata de una clase simple cuyo cometido es el de almacenar el tamaño de algún elemento del tablero (o del propio tablero). Sus atributos son:

- x: es una variable privada de tipo entero entero que almacena la anchura de una instancia de esta clase.
- y: es una variable privada de tipo entero entero que almacena la altura de una instancia de esta clase.

Los métodos de que dispone son:

- Size(): constructor vacío de la clase que pondrá el tamaño a (0,0).
- Size(int newWidth, int newHeight): constructor parametrizado que pondrá el tamaño en los valores que reciba como parámetros.
- Size(Size copyPos): constructor de copia de la clase Size.
- setWidth(int newWidth): función que recibe un valor entero y lo almacena en la variable width.
- setHeight(int newHeight): función que recibe un valor entero y lo almacena en la variable height.
- getWidth(): función que devuelve un entero con el valor guardado para la variable width.
- getHeight(): función que devuelve un entero con el valor guardado para la posición height.
- toString(): función que devuelve una cadena de texto que muestra la información de tamaños almacenada en esta instancia.

Clase Mecca

Clase que representa al aventurero del juego. Permite guardar y modificar información sobre el número de flechas que tiene, su posición, y una lista de los tesoros ganados. Los atributos con los que cuenta son los siguientes:

- nArrows: variable privada de tipo entero que almacena el número de flechas que tiene Mecca.
- position: variable privada de tipo Position que almacena la posición de Mecca en el tablero.
- treasuresWon: variable privada de tipo ArrayList de enteros que almacena los tesoros que han sido conseguidos por Mecca.

Sus métodos son:

- Mecca(): constructor vacío de la clase. Inicializará la posición con el constructor vacío de Position y el número de flechas a 0, así como los tesoros conseguidos.
- getNArrows(): función que devuelve un entero que contiene el número de flechas que le quedan a Mecca.
- setNArrows(int newArrows): función que recibe un entero que actualizará el valor de la variable nArrows.
- incNarrows(int inc): función que incrementa, en el valor entero que reciba como parámetro, el valor de nArrows.
- decNarrows(int inc): función que decrementa, en el valor entero que reciba como parámetro, el valor de nArrows.
- incNarrows(): función que incrementa nArrows una unidad.
- decNarrows(): función que decrementa nArrows una unidad.
- getPos(): función que devuelve un objeto de tipo Position que contiene la posición de Mecca.
- setPos(): función que recibe un objeto de tipo Position que será la nueva posición de Mecca.

Clase Board

Esta clase representa al tablero del juego. Permite guardar y modificar información sobre qué hay en cada casilla y mover a Mecca por todo el tablero. Gestiona todas las acciones que pueden realizarse, además de los distintos resultados que puede tener la partida.

En cuanto a los atirbutos que contiene, en primer lugar, existen unas variables estáticas que contienen los distintos valores que puede albergar una casilla para representar lo que en ella hay dentro del juego. Son las siguientes:

```
public static final String WUMPUS = "Wumpus";
    public static final String TREASURE = "Treasure";
    public static final String HOLE = "Hole";
    public static final String BREEZE = "Breeze";
    public static final String SMELL = "Smell";
```

```
public static final String START = "Start";
public static final String EXIT = "Exit";
public static final String EMPTY = "Empty"; // No Wumpus, no hole
```

En el caso de "Empty", cabe destacar que una casilla puede ser empty aún conteniendo el olor o la brisa, ya que esos estados son simplemente utilizados para informar a Mecca de las cosas que están cerca, pero no afectan de ninguna manera al estado del juego. El resto de atributos que contiene son:

- boardMatrix: es una matriz de ArrayLists de cadenas que contiene, para cada casilla, una lista de cadenas con los elementos que existen en ella.
- boardMatrixVisited: es una matriz booleana, del mismo tamaño que el tablero, que controla qué casillas han sido visitadas y cuáles no. Es utilizada para mostrar por pantalla sólo la información de las casillas ya visitadas y que el resto del tablero sea un misterio.
- boardSize: atributo de tipo Size que guarda el tamaño del tablero.
- mecca: atributo de tipo Mecca que guarda toda la información del aventurero.
- wumpusPos: atributo de tipo Position que guarda información sobre la posición del Wumpus.
- startPos: atributo de tipo Position que guarda información sobre la posición de la casilla de comienzo.
- exitPos: atributo de tipo Position que guarda información sobre la posición de la casilla de salida.
- treasuresPos: ArrayList de posiciones que contiene la posición de los tesoros existentes en el tablero.
- holesPos: ArrayList de posiciones que contiene la posición de los hoyos existentes en el tablero.
- isWumpusAlive: atributo booleano que será true si el Wumpus está vivo y false en caso contrario.
- endOfGame: atributo booleano que será true si el juego ha terminado ya y falso en caso contrario.

Los métodos de esta clase son:

- Board(): constructor vacío de la clase Board. Por defecto todos los tamaños y posiciones son inicializados con sus respectivos constructores vacíos.
- restartBoard: método que inicializa el tablero colocando todas sus casillas como vacías.
- isEmpty(Position pos): metodo que devuelve verdadero si la casilla que se le pasa como argumento está vacía y falso en caso contrario.
- isInsideBoard(Position pos): metodo que devuelve verdadero si la casilla que se le pasa como argumento está dentro del trablero y falso en caso contrario.
- writeOnBoard(Position pos, String element): escribe en la posición del tablero que se le pasa como primer parámetro el elemento que recibe como segundo.
- removeFromBoard(String element, Position position): borra de la posición del tablero que se le pasa como segundo parámetro el elemento que se le pasa como primero.
- readFromBoardVisited(Position pos): función privada que devuelve verdadero si la posición que se le pasa como parámetro ya ha sido visitada por Mecca y falso en caso contrario.
- writeOnBoardVisited(Position pos, boolean element): función privada que escribe (en la matriz de casillas visitadas) en la posición que se le pasa como primer parámetro el booleano que se le pasa como segundo parámetro.
- getSize(): función que devuelve el tamaño del tablero en una variable de tipo Size.
- setSize(): función que recibe como parámetro una variable de tipo Size y hace que este sea el nuevo tamaño del tablero, además de reiniciarlo.
- getWumpusPos(): función que devuelve una variable de tipo Position con la posición del Wumpus.

- setWumpusPos(Position newPos): función que coloca el Wumpus en la posición que recibe como parámetro. Devolverá true si todo funciona y falso en caso contrario.
- removeWumpus(): función que borra al Wumpus del tablero, eliminado también su olor.
- getStartPos(): función que devuelve una variable de tipo Position que contiene la posición de la casilla de comienzo.
- setStartPos(Position newPos): función que coloca, si es posible, la casilla de salida en la posición que se le pasa como parámetro. Devolverá true si todo va bien y false en caso contrario.
- getExitPos(): función que devuelve una variable de tipo Position que contiene la posición de la casilla de salida.
- setExitPos(Position newPos): función que coloca, si es posible, la casilla de salida en la posición que se le pasa como parámetro. Devolverá true si todo va bien y false en caso contrario.
- setTreasurePos(Position newPos): función que permite añadir tesoros a la lista de tesoros en la posición que se le pase como parámetro. Devuelve un entero que contiene la posición en la que se encuentra el tesoro en la lista.
- getTreasurePos(int treasureNo): función que devuelve la posición en el trablero del tesoro que se encuentra en la posición de la lista que indica el entero que recibe como parámetro.
- getTreasuresPos(): función que devuelve toda la lista de los tesoros con sus posiciones.
- showTreasures(): función que muestra un mensaje por consola que indica la posición de cada tesoro.
- editTreasurePos(int treasureNo, Position newPos): función que permite editar la posición del tesoro cuya posición en el vector se le pasa como primer parámetro. La posición que se le pondrá es la indicada por el segundo parámetro.
- removeTreasure(int treasureNumber): función que borra de la lista de tesoros aquél cuya posición en esa lista sea la que se le pasa como parámetro.

- removeTreasure(Position position): borra de la lista de tesoros el que se encuentre en la posición del tablero que se le pase como parámetro.
- setHolePos(Position newPos): función que permite añadir hoyos a la lista de hoyos en la posición que se le pase como parámetro. Devuelve un entero que contiene la posición en la que se encuentra el tesoro en la lista.
- getHolePos(int holeNo): función que devuelve la posición en el trablero del hoyo que se encuentra en la posición de la lista que indica el entero que recibe como parámetro.
- getHolesPos(): función que devuelve toda la lista de los hoyos con sus posiciones.
- editHolePos(int holeNo, Position newPos): función que permite editar la posición del hoyo cuya posición en el vector se le pasa como primer parámetro. La posición que se le pondrá es la indicada por el segundo parámetro.
- removeHole(int holeNumber): función que borra de la lista de hoyos aquél cuya posición en esa lista sea la que se le pasa como parámetro.
- showHoles(): función que muestra un mensaje por consola que indica la posición de cada hoyo.
- getTotalTreasures(): función que devuelve un entero con la cantidad de tesoros existentes.
- getNumberOfHoles(): función que devuelve un entero con la cantidad de hoyos existentes.
- getMecca(): función que devuelve la instancia de la clase Mecca que se contiene en este tablero.
- getMeccaPos(): función que devuelve un objeto de tipo Position que contiene la posición de Mecca.
- setMeccaPos(Position newPos): función que recibe un objeto de tipo Position que será la nueva posición de Mecca.
- getMeccaNArrows(): función que devuelve un entero que contiene el número de flechas que le quedan a Mecca.

- setMeccaNArrows(int nArrows): función que recibe un entero que actualizará el valor de la variable que contiene el número de flechas que le quedan a Mecca.
- incMeccaNArrows(int inc): función que incrementa, en el valor entero que reciba como parámetro, el valor del número de flechas que le quedan a Mecca.
- decMeccaNArrows(int dec): función que decrementa, en el valor entero que reciba como parámetro, el valor del número de flechas que le quedan a Mecca.
- meccaGoUp(): función que mueve a Mecca una casilla hacia arriba.
- meccaGoDown(): función que mueve a Mecca una casilla hacia abajo.
- meccaGoLeft(): función que mueve a Mecca una casilla hacia la izquierda.
- meccaGoRight(): función que mueve a Mecca una casilla hacia la derecha.
- meccaShoot(int direction): función booleana que permite a Mecca disparar. Devolverá true si acaba con el Wumpus y false en caso contrario. Recibe un entero que indica la dirección de disparo con los siguientes valores:
  - 1.- Arriba
  - 2.- Derecha
  - 3.- Izquierda
  - 4.- Abajo
- meccaGoPosition(Position position): función privada de tipo booleano que coloca a Mecca en la posición que recibe como parámetro. Devolverá true si el movimiento pudo ser efectuado y falso en caso contrario.
- checkMovement(Position position): función privada de tipo booleano que comprueba el resultado de un movimiento de Mecca. Devuelve true si puede realizarse y false en caso contrario. Informa por consola del resultado de ese movimiento.
- getBoardSize(): función que devuelve un objeto de tipo Size con el tamaño del tablero.

- toString(): función que devuelve una cadena que contiene toda la información relativa al tablero.
- isWumpusAlive(): función booleana que devuelve true si el Wumpus está vivo y false en caso contrario.
- setWumpusAlive(boolean isWumpusAlive): función que recibe un booleano que será el que indique si el Wumpus está vivo o no y lo guarda.
- initGame(): función booleana que comprueba si, al entrar en el modo aventura, el mapa definido tiene todo lo necesario para comenzar el juego. En caso afirmativo devuelve true, en caso negativo false.
- isGameFinished(): función booleana que devuelve true si el juego ha terminado y false en caso contrario.
- checkStart(): función booleana que comprueba si la casilla de comienzo ha sido colocada. Devuelve true en caso de que lo haya sido y false en caso contrario.
- checkExit(): función booleana que comprueba si la casilla de salida ha sido colocada. Devuelve true en caso de que lo haya sido y false en caso contrario.
- checkWumpus(): función booleana que comprueba si la casilla del Wumpus ha sido colocada. Devuelve true en caso de que lo haya sido y false en caso contrario.
- showAdventureState(): fnción que muestra por consola el estado de la aventura. Es decir, muestra todos los elementos que el jugador conoce porque ya ha visitado esas casillas.

4.4. Tabla de símbolos

En este proyecto, la tabla de símbolos se implementa con una clase que contiene como atributo un ArrayList de objetos de la clase Variable. Por esta razón, va a explicarse primero esa clase y posteriormente se explicará la tabla propiamente dicha.

4.4.1. Clase Variable

Esta clase, realizada por Luis Del Moral, Juan María Palomo y Profesor, y modificada por los autores de este trabajo contiene lo necesario para representar una variable de las que se almacenarán en la tabla de símbolos. Sus atributos, todos privados son los siguientes:

- _nombre: que representa el nombre del identificador.
- _tipo: que representa el tipo del identificador. En esta aplicación concreta los tipos serán number y string.
- _valor: que representa el valor que contiene ese identificador.

En cuanto a los métodos que contiene, son los siguientes:

- Variable(): constructor vacío que no realiza ninguna operación.
- Variable(String nombre,String tipo, String valor): constructor parametrizado que recibe el nombre, tipo y valor de una variable a crear y los guarda en los atributos privados ya comentados.
- setNombre(String nombre): método que permite guardar el nombre que se le pasa como parámetro.
- setValor(String valor): método que permite guardar el valor que se le pasa como parámetro.
- getNombre(): método que devuelve un entero con el nombre de la variable.
- getTipo(): método que devuelve un entero con el tipo de la variable.
- escribirVariable(): método que escribe por pantalla toda la información relativa a una variable, es decir, nombre, tipo y valor.
- isNumber(): función booleana que devuelve true si la variable es de tipo number y false en caso contrario.
- isString(): función booleana que devuelve true si la variable es de tipo string y false en caso contrario.

Una vez definida esta clase, ya se puede entender perfectamente la clase TablaSimbolos.

4.4.2. Clase TablaSimbolos

Esta clase, realizada por Luis Del Moral, Juan María Palomo y Profesor, y modificada por los autores de este trabajo, contiene lo necesario para guardar los identificadores, tipos y valores de todas las variables creadas durante la ejecución del intérprete. El único método que contiene es un ArrayList privado de objetos de la clase Variable ya explicada. Los métodos que implementa son:

- TablaSimbolos(): constructor vacío en el que se inicializa el Arraylist.
- insertarSimbolo(Variable nuevoSimbolo): función cuya misión es insertar la variable que recibe como parámetro en la tabla de símbolos.
- getSimbolo(int indice): devuelve la variable que ocupa la posición señalada por indice.
- existeSimbolo(String nombreSimbolo): función que comprueba si existe el símbolo cuyo nombre recibe como parámetro en la tabla de símbolos. Si existe, devuelve el índice que indica la posición del símbolo, en caso contrario, devuelve -1.
- eliminarSimbolo(String nombreSimbolo): función que elimina de la tabla de símbolos el identificador cuyo nombre recibe como parámetro.
- escribirSimbolos(): función que escribe por pantalla los identificadores de la tabla de símbolos.

4.5. Elementos auxiliares para la simulación del entorno diseñado

Capítulo 5

Modo de obtención del intérprete: descripción del fichero makefile

5.1. Nombre y descripción de cada uno de los ficheros utilizados

5.2. Modo de generación de intérprete

Capítulo 6

Modo de ejecución del intérprete

6.1. Interactiva

6.2. A partir de un fichero

Capítulo 7

Ejemplos

En el presente capítulo van a mostrarse algunos ficheros de ejecución realizados para realizar la fase de pruebas y, además, mostrar las diferentes posibilidades del intérprete desarrollado. Cada uno de los ejemplos intenta enfatizar una de las distintas posibilidades del intéprete. En todos ellos hay comentarios tanto de una línea como de varias. Los ejemplos que van a verse son los siguientes:

- Ejemplo 1: Juego Básico
- Ejemplo 2: Variables (declaración y uso)
- Ejemplo 3: Uso de los esquemas condicionales
- Ejemplo 4: Juego avanzado
- Ejemplo 5:

7.1. Ejemplo 1: Juego Básico

En este fichero de ejemplo se muestra cómo crear un tablero de juego, realizando las siguientes acciones:

- Entrada en el modo de configuración

- Selección del tamaño del tablero
- Comprobación de que el comando anterior funcionó, llamando a las funciones encargadas de devolver el valor del número de filas y columnas
- Selección de las casillas de comienzo y de salida
- Selección de las posiciones de 3 hoyos
- Comprobación de las posiciones de los hoyos
- Selección del número de flechas que tendrá Mecca
- Selección de la posición de un tesoro
- Comprobación de la posición del tesoro
- Selección de la posición del Wumpus
- Salida del modo de configuración

- Entrada en el modo aventura
  - Realización de movimientos, comprobando que se muestra el olor del Wumpus
  - Llegada de Mecca a la casilla del tesoro
  - Llegada de Mecca a la casilla de salida

Código del ejemplo:

```
{ Test file number 01
We will show how to create a MeccaBoard
and play one game}

ConfigurationMode   #This is the mode to create the board
setBoardSize(5,5);  #Size of 5 rows and 5 columns
getBoardRows();     #Yeah, it works!
getBoardColumns();  #Again!
setStart(0,0);      #Down left corner
setExit(4,4);           #Up right corner
setHole(1,1);
setHole(2,2);
setHole(0,1);
showHoles();
setArrows(3);
setTreasure(4,3);
```

```
showTreasures();
setWumpus(4,0);
endConfigurationMode
{
This is what the level should look like

        -   -       -       -       X
        -   ~       ~       ~       T
        ~   ~       H       ~       -
        H   H       ~       ~       =
        +   ~       ~       =       W

}
#Let's start the adventure!!!
AdventureMode
goRight();
goRight();
goRight();  #It should smell bad right now
goUp();             #Let's avoid the Wumpus
goRight();
goUp();
goUp();             #You should have found a treasure here
goUp();             #Winner
endAdventureMode
```

7.2. Ejemplo 2: Variables (declaración y uso)

En este ejemplo se muestra cómo se declaran y utilizan las variables. Como puede observarse, éstas pueden ser definidas y utilizadas fuera de los modos de configuración o aventura. Las acciones realizadas son las siguientes:

- Declaración, con asignación de valor, de tres variables numéricas
- Declaración, también con asignación de valor, de cuatro variables de tipo cadena
- Declaración de una cadena de cara a su impresión por pantalla siéndole asignado el resultado de una operación de suma entre las tres variables

de cadena creadas anteriormente

- Utilización del comando de impresión por pantalla para mostrar la cadena que ha sido creada en el paso anterior
- Asignación a la cadena que ha sido imprimida de valores diferentes y realización de nuevas impresiones
- Creación de una nueva variable numérica a la que se le asigna el valor de multiplicar las tres variables numéricas creadas al comienzo del ejemplo
- Impresión del valor de la variable que se acaba de crear

Código del ejemplo:

```
{ Test file number 02
We will show how to declare and use variables }

#Variables can be declared outside the modes
number jordan:=23;
number magic:=32;
number bird:=33;
string jor:="Jordan";
string mag:="Magic";
string bir:="Bird";
string player_presentation:=" number is ";


#Now we are going to print them
string printout := jor+player_presentation+jordan;
read(printout);

printout := mag+player_presentation+magic;
read(printout);

printout := bir+player_presentation+bird;
read(printout);

#Let's do a mathematical operation
number result:=jordan*magic*bird;
read(result);
```

7.3. Ejemplo 3: Uso de los esquemas condicionales

En este ejemplo se crean tres variables numéricas, se utilizan esquemas condicionales comparando los valores de las mismas y se crean las variables que corresponda dependiendo de estos valores y, posteriormente se muestran por pantalla. Los pasos realizados son los siguientes:

- Creación de tres variables numéricas
- Esquema condicional en el que se comparan dos valores numéricos y se asigna un valor a una variable de tipo cadena dependiendo del resultado de esa comparación
- Impresión de la variable creada
- Utilización de dos esquemas condicionales anidados para la declaración de una segunda variable cadena, cuyo valor depende del resultado de la declaración
- Impresión de la variable creada

Código del ejemplo:

```
{ Test file number 03
We will show how to use conditional schemes }

#Variables can be declared outside the modes
number jordan:=23;
number magic:=32;
number bird:=33;

if jordan > magic then
        string print:=jordan+" is bigger than "+magic;
else
        string print:=jordan+" is not bigger than "+magic;
end_if;

read(print);
```

```
if jordan < magic then
        if jordan < bird then
                string print2:=jordan+" is smaller than "+magic+"
        end_if;
end_if;

read(print2);

#You can also do it like this

if jordan < magic _and jordan < bird then
        string print2:=jordan+" is smaller than "+magic+" and "+b
end_if;

read(print2);
```

7.4. Ejemplo 4: Juego Avanzado

En este fichero de ejemplo se muestra cómo crear un tablero de juego como el del ejemplo 1 pero haciendo uso de elementos más avanzados como bucles, esquemas condicionales y variables:

- Entrada en el modo de configuración
  - Creación de las variables de filas y columnas, denominadas rows y cols respectivamente
  - Asignación de un valor a la variable de filas
  - Asignación a la variable de columnas del valor contenido en la de filas
  - Selección del tamaño del tablero usando ambas variables
  - Comprobación de que el comando anterior funcionó, llamando a las funciones encargadas de devolver el valor del número de filas y columnas
  - Selección de la posición de la casilla de comienzo, cuyas coordenadas son el resultado de restar las dos variables anteriores

- Selección de la posición de la casilla de finalización, cuyas coordenadas son el resultado de restarle uno a las dos variables
- Utilización de un esquema condicional para insertar posiciones de hoyos
- Comprobación de las posiciones de los hoyos
- Utilización de un esquema de tipo "mientras" para la selección del número de flechas que tendrá Mecca
- Selección de la posición de un tesoro
- Comprobación de la posición del tesoro
- Selección de la posición del Wumpus
- Salida del modo de configuración

▪ Entrada en el modo aventura

- Utilización de un bucle del tipo "para" con el objetivo de mover a Mecca tres casillas a la derecha
- Realización de dos movimientos directamente
- Creación de una variable de tipo numérico que nos servirá en el bucle siguiente
- Utilización de un bucle "repetir hasta" que moverá a Mecca tres veces hacia arriba que producirá los siguientes eventos:
  - Llegada de Mecca a la casilla del tesoro
  - Llegada de Mecca a la casilla de salida
- Creación de dos variables de tipo cadena para mostrar un mensaje de despedida
- Impresión en consola de las dos cadenas recién creadas usando *read*
- Utilización de *read* para imprimir por pantalla una cadena directamente insertada por el usuario

Código del ejemplo:

```
{ Test file number 04
We will show how to create a MeccaBoard
and play one game while we use all kind of loops}
```

```
ConfigurationMode   #This is the mode to create the board
number rows, cols;
rows:=5;
cols:=rows;
setBoardSize(rows,cols); #Size of 5 rows and 5 columns
getBoardRows();     #Yeah, it works!
getBoardColumns(); #Again!
setStart(cols-rows,rows-cols);      #Down left corner
setExit(cols-1,rows-1);     #Up right corner
if cols == rows then    #Conditional
setHole(1,1);
setHole(2,2);
setHole(0,1);
else
setHole(1,1);
end_if;
showHoles();
while rows > 2 do    #While Loop
incArrows(1);
rows:=rows-1;
end_while;
setTreasure(4,3);
showTreasures();
setWumpus(4,0);
endConfigurationMode
{
This is what the level should look like

        -       -       -       -       X
        -       ~       ~       ~       T
        ~       ~       H       ~       -
        H       H       ~       ~       =
        +       ~       ~       =       W

}
#Let's start the adventure!!!
AdventureMode
for i from 0 until 3 step 1
do
goRight();
end_for;        #We are using a for loop for moving 3 times
```

```
goUp();
goRight();
number limit:=0;
repeat          #Repeat-until loop
goUp();
limit:=limit+1;
until limit==3;
string final:= "/************************/";
string final2:= "/*****see you soon!******/";
write(final);
write(final2);
write("/************************/");
endAdventureMode
```

Bibliografía

[1] Edgar Allan Poe. El escarabajo de oro. Editorial: Losada. Año 2010. I.S.B.N : 9789500397124.

[2] Stephen W. Hawking. El universo en una cáscara de nuez. Editorial: Círculo de lectores. Año 2005. I.S.B.N : 84-843-2293-9.

[3] Stephen W. Hawking. Brevísima historia del tiempo. Editorial: Círculo de lectores. Año 2005. I.S.B.N : 84-672-1469-4.

[4] Stephen Bartlett. Lecture 5 on Quantum Computing. NITP Summer School, Adelaide, Australia. Año: 2003. Dirección web: `http://www.cs.duke.edu/~reif/courses/complectures/AltModelsComp/QuantCrypt/QuantCryptOverview/Bartlett/QuantCryptOverviewBartlett.pdf`. Fecha de última consulta: 13 de Mayo de 2012.

[5] Charles H. Bennett, Gilles Bras Sard and Artur K. Ekert. Quantum Cryptography. Editorial: Scientific American. Año: 1992. Dirección web: `http://www.dhushara.com/book/quantcos/aq/qcrypt.htm`. Fecha de última consulta: 13 de Mayo de 2012.

[6] Jesús Martínez Mateo. PFC: Criptografía cuántica aplicada. Universidad Politécnica de Madrid. Año: 2008. Dirección web: `http://oa.upm.es/1298/1/PFC_JESUS_MARTINEZ_MATEO.pdf`. Fecha de última consulta: 13 de Mayo de 2012.

[7] Nuevo impulso a la criptografía cuántica. Universidad Politécnica de Madrid. Año: 2012. Dirección web: `http://www.fi.upm.es/?id=tablon&acciongt=consulta1&idet=1068`. Fecha de última consulta: 13 de Mayo de 2012.

[8] Jesús Martínez Mateo. Tésis Doctoral: Reconciliación Eficiente de Información para la Distribución Cuántica de Claves. Universidad Politécnica de Madrid. Año: 2011. Dirección web: `http://oa.upm.es/9717/1/Jesus_Martinez_Mateo.pdf`. Fecha de última consulta: 13 de Mayo de 2012.