MineSweeper The Game

[a] Castro Renzo Gustavo; [b] Paredes Samuel Octavio; [c] Ramírez Eduardo Manuel; [d] Saucedo Gonzalo Nicolás

[a] U.T.N F.R.Re, ISI B, renzogustavocastro99@gmail.com
[b] samuelocta215@gmail.com
[c] edu.ramirez645@gmail.com
[d] gonzalosaucedoo15@gmail.com

Resumen

En el presente trabajo exploraremos el desarrollo del juego "Buscaminas", en el lenguaje de programación SmallTalk en Pharo 8.0. El desarrollo fue pensado para tener una interfaz gráfica sencilla que será a través de la ventana Transcript y un sistema de puntero y movilidad en el tablero a través de letras del teclado. A grandes rasgos el juego está compuesto por dos tableros superpuestos de manera lógica, un tablero que será lo que se muestra por pantalla y otro donde tendremos los elementos del tablero tradicional, es decir, elementos vacíos, pistas y bombas, estas se irán develando en el tablero visible a medida que se decida mostrar la celda.

Además se agregaron algunas características extras, primeramente podremos elegir un grado de dificultad para el juego, esto hace que las dimensiones del tablero y el número de minas sean diferentes entre grado y grado. También se puede ganar diferentes puntajes dependiendo de la característica del elemento el cual fue develado. Por último se agregó que el jugador pueda seguir jugando y al final de sus intentos se le muestra un top de jugadas de la peor a la mejor puntuación que obtuvo.

1. Introducción

El "*Buscaminas*" es un juego cuyo objetivo es descubrir la ubicación de todas las minas del tablero sin pisarlas (ver Fig. 1). El tablero tendrá una dimensión y cantidad de minas determinadas por el grado de dificultad elegida por el jugador, las minas serán distribuidas de manera aleatoria y el jugador debe desactivarlas. Si este llegara a tocar una mina, habrá perdido y el juego terminará. El jugador tiene la habilidad de marcar las casillas que crea contiene una mina con una bandera, y si consigue marcar correctamente todas las ubicaciones del tablero, ganará la partida. Las casillas pueden indicar con números que hay una cierta cantidad de minas adyacentes a ella, variando desde 1 hasta 8, si se revela una casilla y esta muestra el número 1, por ejemplo, significa que en una de las 8 casillas adyacentes, existe una única bomba. Al seleccionar una ubicación vacía, si no hay ninguna mina adyacente a ésta, todas las casillas vacías adyacentes también se desbloquearán, liberando una parte del tablero delimitada por pistas que indican que hay bombas adyacentes; en cambio, si hay una mina el juego termina y se mostrarán todas las minas que existen dentro del tablero.

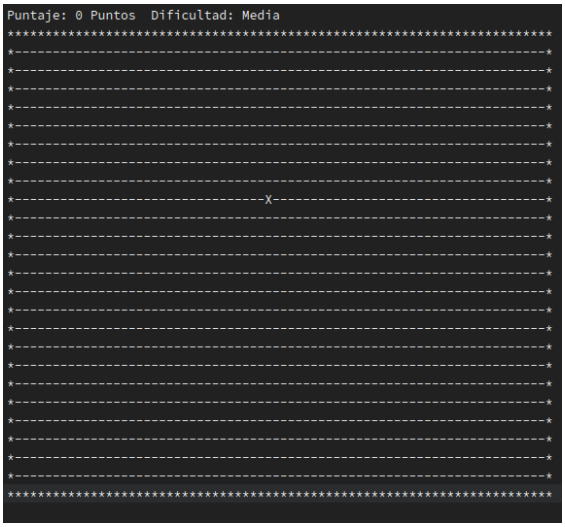

Fig. 1: Juego iniciado

2. Desarrollo

Comenzando con el desarrollo del programa, decidimos hacer un modelado de clases (ver Fig. 2) que consiste en una clase Buscaminas que será la clase contenedora, ésta contendrá los métodos y atributos necesarios para llevar adelante la lógica del juego, una clase Tablero que tendrá los elementos necesarios para crear los dos tableros que usaremos a lo largo del juego y colocar el contenido necesario, una clase Elemento que representa a los valores elementales del tablero, llámese espacio vacío

o pista y una clase Bomba que será una clase particular de la clase Elemento ya que es la encargada de representar a una mina dentro del tablero.

La lógica del diagrama es la siguiente, la clase Buscaminas contendrá una instancia de la clase Tablero la cual contiene 2 tableros, lo llamaremos *tableroOculto* y *tableroVisible*, estos tableros serán creados una vez ejecutemos el método *creaTablero*, ambos tableros tendrán las mismas dimensiones determinada por la dificultad que se elija, los tableros tendrán una forma de creación similar pero al *tableroVisible* lo rellenaremos con el string '-' y al *tableroOculto* lo rellenaremos con instancias de la clase Elemento, dicha clase tendrá un valor inicial de '0'. La clase Tablero también tendrá los atributos *filas*, *columnas* y *cantMinas* que en un principio tendrán un valor de 0, esto cambia cuando ejecutamos el método *eligeDificultad*, dependiendo de lo que se elija se determina el valor de estos atributos y también tendrá un atributo llamado *minasOcultas* que será una lista que contendrá las coordenadas de las minas que coloquemos, este atributo será completado cuando ejecutamos el método *colocaMinas*, dicho método coloca instancias de la clase Bomba de manera aleatoria en el *tableroOculto*, y luego el proceso será complementado con la ejecución del método *colocaPistas* que tiene como objetivo recorrer la matriz de *tableroOculto* y cada vez que encuentre una bomba visitar todos los casilleros adyacentes incrementando el valor en uno, para esto la clase Elemento contendrá un método llamado *incrementarValor* que hará lo antes mencionado.

Teniendo en cuenta estas relaciones volvemos a la clase Buscaminas, la misma contendrá como atributos, además de *tableros*, *score*, donde se guardará el puntaje de cada juego ejecutado, y *dificultad*, que guardará un string con el nombre de la dificultad que se elige. Esta clase tiene la capacidad de imprimir los tableros (ver Fig. 3 y Fig. 4) con el método *muestraTablero*, también se irá modificando la puntuación del jugador con el método *sumaPuntos*, y por último contendrá un algoritmo de relleno por difusión en el método rellenado que es llamado cada vez que develamos un espacio en blanco en el *tableroOculto* representado con un '0', esto hace que se develen todos los espacios en blanco hasta que encuentre una pista, la cual también será develada.

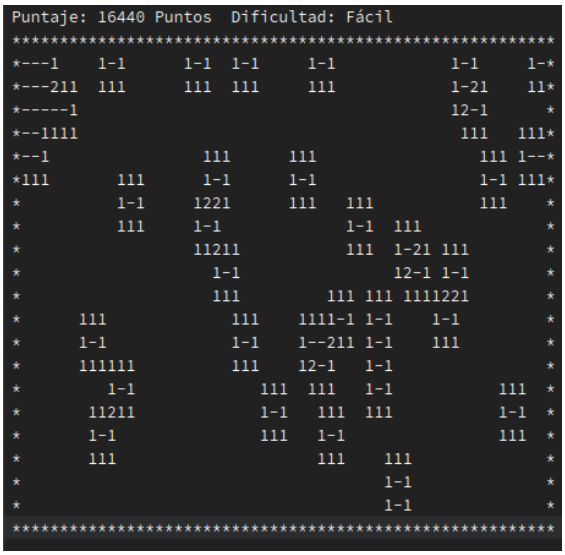

Fig. 3: Tablero Visible

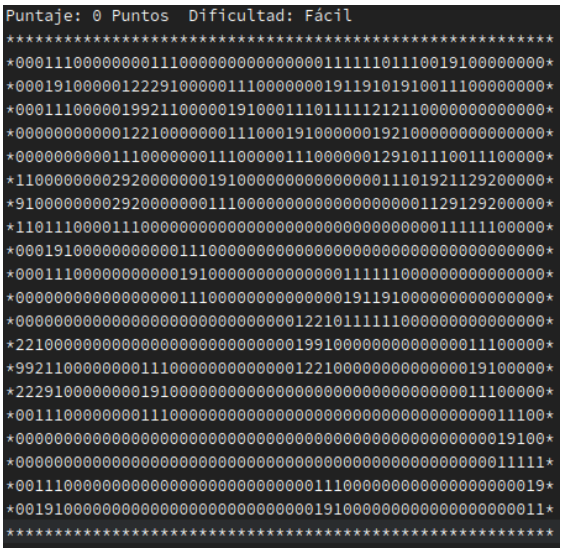

Fig. 4: Tablero Oculto

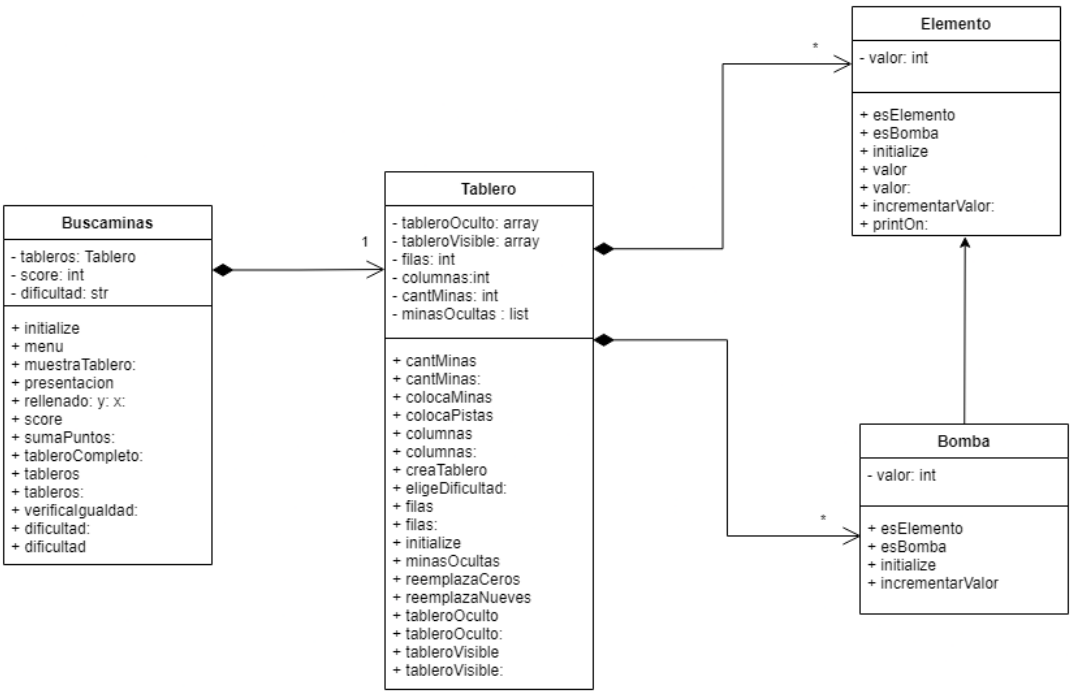

Fig. 2: Diagrama de Clases del Buscaminas

Para poder llevar a cabo esta tarea tuvimos que enfocarnos en desarrollar desde la clase componente más abajo en la jerarquía la cuales fueron Elemento y Bomba, que como tienen una jerarquía de herencia entre ellas (ver Fig. 2).

Clase Elemento

Object subclass: #Elemento

instanceVariableNames: 'valor'

classVariableNames: ''

package: 'juegoBuscaminas'

Con sus métodos más importantes:

esBomba

^ false

esElemento

^ true

Estos dos métodos son importantes ya que a la hora de determinar si un casillero dentro del tablero es un elemento normal o una bomba, con estos métodos lo determinamos de inmediato.

incrementarValor

valor := valor + 1

Éste método es necesario a la hora de colocar las pistas, ya que en un principio tendremos valor 0, y luego de colocado las minas hay que recorrer los casilleros adyacentes de cada bomba para incrementar en uno indicando así que existe una bomba en las cercanías de ese casillero.

initialize

super initialize.

valor := 0

El valor inicial también será importante ya que nosotros decidimos que el espacio vacío dentro del tablero oculto tendrá la representación con ese valor.

Clase Bomba

Elemento subclass: #Bomba

instanceVariableNames: ''

classVariableNames: ''

package: 'juegoBuscaminas'

Con sus métodos más importantes:

esBomba

^ true

esElemento

^ false

Estos métodos fueron redefinidos para indicar que cuando estemos en presencia de una instancia tipo Bomba lo sepamos.

initialize

valor := 9

El valor inicial de este las instancias de Bomba serán '9' ya que así decidimos representar en el tablero oculto la presencia de una mina.

incrementarValor

valor := valor + 0

Este método lo redefinimos ya que una instancia de Bomba no puede incrementar su valor, tiene un valor fijo y es el valor '9'.

Clase Tablero

Object subclass: #Tablero

instanceVariableNames:'tableroOculto tableroVisible filas columnas cantMinas minasOcultas'

classVariableNames: ''

package: 'juegoBuscaminas'

Con sus métodos más importantes:

initialize

super initialize .

minasOcultas := OrderedCollection new.

cantMinas:= 0.

filas:= 0.

columnas:= 0.

Este método coloca valores iniciales a los atributos de la instancia de Tablero.

eligeDificultad: unNumero

unNumero = '1' ifTrue:[

self filas: 20.

self columnas: 35.

self cantMinas: 30.

].

unNumero = '2' ifTrue:[

self filas: 24.

self columnas: 60.

self cantMinas: 50.

].

unNumero = '3' ifTrue:[

self filas: 28.

self columnas: 75.

self cantMinas: 70.

].

unNumero = '4' ifTrue:[

self filas: 10.

self columnas: 10.

self cantMinas: 2.

].

Este método settea los componentes más importantes del tablero que son sus filas, columnas y la cantidad de minas que tendrán.

creaTablero

```
|nuevo|
tableroOculto:= Array new: filas.
tableroVisible:= Array new: filas.

1 to: filas do:[:i|
        tableroOculto at: i put: (Array new: columnas).
        tableroVisible at: i put: (Array new: columnas).
        1 to: columnas do:[:j|
                nuevo := Elemento new.
                (tableroOculto at: i) at: j put: nuevo.
                (tableroVisible at: i) at: j put: '-'
                ]
        ].
```

Este es el método encargado de crear los dos tableros que vamos a utilizar en todo el juego, como se puede observar *tableroOculto* será llenado de instancias de Elemento de manera inicial, y *tableroVisible* será llenado del string '-'.

colocaMinas

```
|numero x y bombaNueva arreglo|
numero:= 0.
[ numero < cantMinas ] whileTrue:[
        y:= (1 to: filas) atRandom.
        x:= (1 to: columnas) atRandom.
        (((tableroOculto at: y) at: x) esElemento) ifTrue:[
                bombaNueva:= Bomba new.
                (tableroOculto at: y) at: x put: bombaNueva.
                arreglo:= Array new: 2.
                arreglo at: 1 put: y.
                arreglo at: 2 put: x.
                minasOcultas add: arreglo.
                numero:= numero + 1.] ]
```

Este método coloca las instancias de Bomba de manera aleatoria a través del método *atRandom*, y por último guardamos en un arreglo de dos posiciones las coordenadas (y,x) de las minas que colocamos y las guardamos en la lista minasOcultas.

colocaPistas

```
1 to: filas do:[:y|
        1 to: columnas do:[:x|
                ((tableroOculto at:y) at:x) esBomba ifTrue:[
        -1 to: 1 do:[:i|
                -1 to: 1 do:[:j|
(((1 <= (y+i)) and:((y+i) <= filas)) and: ((1 <= (x+j)) and:((x+j) <= columnas)) ) ifTrue: [
((tableroOculto at:(y+i)) at:(x+j)) esElemento ifTrue: [
((tableroOculto at:(y+i)) at:(x+j)) incrementarValor ]]]]]]
```

Este método coloca las pistas en el *tableroOculto*, de una manera sencilla, recorre todo el *tableroOculto* y cada vez que encuentra un objeto de tipo Bomba, que lo determinamos ejecutando el método *esBomba*, recorremos todos los casilleros adyacentes e incrementamos en un uno.

Clase Buscaminas

```
Object subclass: #Buscaminas
        instanceVariableNames: 'tableros score dificultad'
        classVariableNames: ''
        package: 'juegoBuscaminas'
```

Con sus métodos más importantes:

initialize

```
|op|
super initialize .
score:= 0.
tableros:= Tablero new.
```

```
op:= UIManager default request: 'Ingrese dificultad 1- Fácil 2- Medio 3- Difícil'.
tableros eligeDificultad: op.
tableros creaTablero.
tableros colocaMinas.
tableros colocaPistas.
op = '1' ifTrue: [ dificultad := 'Fácil' ]
ifFalse: [ op = '2' ifTrue: [ dificultad := 'Media' ]
ifFalse: [ op = '3' ifTrue: [ dificultad := 'Difícil' ]
ifFalse: [ op = '4' ifTrue: [ dificultad := 'Misteriosa' ] ] ] ]
```

En el método *initialize* de la clase Buscaminas vamos a determinar, además del valor inicial de *score*, la instancia de la clase Tablero, luego ejecutamos en orden los métodos *eligeDIficultad, creaTablero, colocaMinas y colocaPistas*, ya que si no lo hacemos en orden nos dará error.

```
sumaPuntos: unPuntaje
score:= score + unPuntaje.
```

Este método realiza la suma del puntaje actual del juego con un puntaje que viene como parámetro.

```
tableroCompleto: unValor
1 to: tableros filas do:[:i|
        1 to: tableros columnas do:[:j|
                (((tableros tableroVisible at: i) at: j) = unValor) ifTrue:[
                        ^false
                ]
        ].
^true
```

Este método retornará verdadero si recorre todo el *tableroVisible* y no encuentra ningún valor que se le pasa por parámetro que en este caso será el string '-' , falso en caso contrario. Se usa para determinar si terminamos el juego.

```
verificaIgualdad: unaLista
|cont|
cont:= 0.

tableros minasOcultas do:[:coord1|
        unaLista do:[:coord2|
                (coord1 = coord2) ifTrue:[
                        cont:= cont + 1
                        ]
                ]
        ].
(cont = (tableros minasOcultas) size) ifTrue:[
        ^true
        ].
^false
```

Con éste método realizamos una comparación entre la lista interna *minasOcultas*, la cual tiene todas las coordenadas de las minas que colocamos en un principio, y una lista externa que guarda todas las minas que marcamos, si las coordenadas guardadas en una es igual a la que pasamos por parámetro éste método devolverá verdadero, falso en caso contrario.

```
rellenado: val y: unY x: unX
|coord ceros y x tam|
ceros:= OrderedCollection new.
coord:= Array new: 2.
coord at: 1 put: unY.
coord at: 2 put: unX.
ceros add: coord.

[(ceros size) > 0] whileTrue:[
        tam:= ceros size.
        y:= (ceros at:tam) at: 1.
        x:= (ceros at:tam) at: 2.
        coord:= Array new:2.
        coord at: 1 put: y.
```

```
coord at: 2 put: x.

ceros remove: coord.

-1 to: 1 do:[:i|

-1 to: 1 do:[:j|

(((1 <= (y+i)) and:((y+i) <= tableros filas)) and: ((1 <= (x+j)) and:((x+j) <= tableros columnas))) ifTrue:[

((((tableros tableroVisible at:(y+i)) at:(x+j)) = val) and: (((tableros tableroOculto at:(y+i)) at:(x+j)) valor = 0)) ifTrue:[
(tableros tableroVisible at:(y+i)) at: (x+j) put: 0.

self sumaPuntos: 20.
coord:= Array new: 2.

coord at: 1 put: (y+i).

coord at: 2 put: (x+j).

(ceros includes: coord) ifFalse:[
        ceros add: coord ]

] ifFalse:[
        self sumaPuntos: 60.
(tableros tableroVisible at:(y+i)) at: (x+j) put: ((tableros tableroOculto at:(y+i)) at: (x+j)) valor. ] ]]] ].

^tableros tableroVisible
```

El método *rellenado*[1] realiza una función primordial en el desarrollo de la lógica del juego ya que como parámetro recibimos las coordenadas que indican que hay un valor igual a '0' en el *tableroOculto* por lo tanto un espacio vacío en el *tableroVisible* y esto indica que hay que revelar todos los valores 0 adyacentes al '0' actual pasado por parámetro y a su vez pasarlo al *tableroVisible* para que muestre al jugador, su desarrollo tiene una similitud con el método *colocaPistas* ya que en ella cuando encontrábamos una instancia de la clase Bomba visitábamos los ochos casilleros adyacentes y aumentábamos su valor en uno con el método *incrementarValor*. En este caso lo que haremos es tomar el '0' actual (ver Fig. 5), tomamos sus coordenadas, que también son pasadas por parámetro, y colocamos dentro de una lista temporal y abrimos un bucle que va a terminar cuando esta lista ya esté vacía, entonces lo hacemos es como primera acción guardar en 'y' y en 'x' las coordenadas actuales y luego eliminar de la lista ya que estamos por analizar este punto, y simplemente realizamos el mismo proceso para ver sus ochos casilleros adyacentes, y si tenemos valor 0 primeramente lo develamos al *tableroVisible* y guardamos cada una de las posiciones en la lista temporal que mencionamos antes, si no tenemos un valor 0 quiere decir que estamos en presencia de una pista por lo que también develaremos al *tableroVisible* pero no haremos nada más. Este proceso se repite con cada coordenada guardada en la lista temporal *ceros* , hasta que nos quedamos sin elementos. Esto tiene el efecto que cuando pisamos un 0 en *tableroOculto* se devele automáticamente todos los 0 hasta un límite marcado por las pistas de bombas.

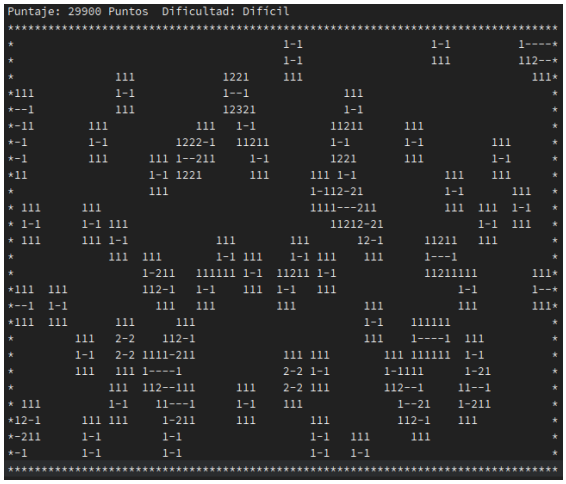

Fig. 5: Algoritmo Floodfill

Todo lo antes explicado conforma la parte de modelado que creamos para el juego, tomamos algunas decisiones de modelo a la hora de hacer este proyecto, y consideramos que dicho proyecto comprende completamente todas las funciones principales del juego.

Por último queremos explicar la interfaz gráfica y cómo lo llevamos a cabo, decidimos hacerlo por la ventana Trascript haciendo uso la variable de clase Transcript, colocando un cursor que estará en *tableroVisible* representado por el string 'X'(ver Fig. 3), con las posibilidad de movimiento a través del ingreso por teclado de las letras 'w' (arriba) 's' (abajo) 'a' (izquierda) 'd' (derecha). A esto le sumamos las funciones principales del juego, las cuales son: 'm' (mostrar/develar casillero) 'b' (marcar bomba) 'v' (desmarcar bomba), la estructura en la ventana del PlayGround es la siguiente:

```
[ jugando ] whileTrue:[

mov:= juego menu.
```

```
mov = 'w' ifTrue:[
        y = 1 ifTrue:[
        y:= 1] ifFalse:[
        (juego tableros tableroVisible at: y) at: x put: real.
        y:= y - 1.
        real:= (juego tableros tableroVisible at: y) at: x.
        (juego tableros tableroVisible at:y) at:x put: 'X'.  ]].

mov = 's' ifTrue:[
        y = juego tableros filas ifTrue:[
        y:= juego tableros filas] ifFalse:[
        (juego tableros tableroVisible at: y) at: x put: real.
        y:= y + 1.
        real:= (juego tableros tableroVisible at: y) at: x.
        (juego tableros tableroVisible at:y) at:x put: 'X'.  ]].

mov = 'a' ifTrue:[
        x = 1 ifTrue:[
        x:= 1] ifFalse:[
        (juego tableros tableroVisible at: y) at: x put: real.
        x:= x - 1.
        real:= (juego tableros tableroVisible at: y) at: x.
        (juego tableros tableroVisible at:y) at:x put: 'X'.  ]].

mov = 'd' ifTrue:[
        x = juego tableros columnas ifTrue:[
        x:= juego tableros columnas] ifFalse:[
        (juego tableros tableroVisible at: y) at: x put: real.
        x:= x + 1.
        real:= (juego tableros tableroVisible at: y) at: x.
        (juego tableros tableroVisible at:y) at:x put: 'X'.  ]].

mov = 'b' ifTrue:[
        real = '-' ifTrue:[
        (juego tableros tableroVisible at:y) at:x put: '#'.
        real:= (juego tableros tableroVisible at:y) at:x.
        arregloSop:= Array new:2.
        arregloSop at: 1 put: y.
        arregloSop at: 2 put: x.
        (minasMarcadas includes: arregloSop) ifFalse:[ minasMarcadas add: arregloSop.] ] ].

mov = 'v' ifTrue:[
        real = '#' ifTrue:[
        (juego tableros tableroVisible at:y) at:x put: '-'.
        real:= (juego tableros tableroVisible at:y) at:x.
        arregloSop:= Array new:2.
        arregloSop at: 1 put: y.
        arregloSop at: 2 put: x.
        (minasMarcadas includes: arregloSop) ifTrue:[ minasMarcadas remove: arregloSop.]]].

mov = 'm' ifTrue:[
        (((juego tableros tableroOculto at:y) at: x) esBomba) ifTrue:[
        (juego tableros tableroVisible at:y) at:x put: '@'.
        jugando:= false ].
        (((juego tableros tableroOculto at:y) at: x) esElemento) ifTrue:[
        (juego tableros tableroVisible at:y) at:x put: ((juego tableros tableroOculto at:y) at: x) valor.
        juego sumaPuntos: 60.
        real:= (juego tableros tableroVisible at:y) at:x].
```

(((juego tableros tableroOculto at:y) at: x) valor = 0) ifTrue:[

(juego tableros tableroVisible at:y) at:x put: 0.

juego tableros tableroVisible: (juego rellenado: '-' y: y x: x).

juego tableros tableroVisible: (juego tableros reemplazaCeros).

real:= (juego tableros tableroVisible at:y) at:x].].

Como se puede observar por cada ingreso de opción se realiza un movimiento u otro dentro del tablero visible, si decidimos movernos en el tablero ingresaremos 'w', 'a', 's', 'd' y lo que harán estos condicionales es verificar si están en el borde del tablero para los cuales cada uno realiza una acción que determina que el cursor no puede salir del tablero, y si no se encuentra en los bordes realiza la acción de colocar el cursor en la posición donde se indique. Por otro lado tenemos los ingresos de las letras 'b' y 'v' que hacen algo similar entre ellas y es que si nosotros marcamos una potencial bomba en el tablero, en caso de 'b', guardaremos en una lista externa llamada minasMarcadas o eliminaremos de esta lista, caso de ingreso de 'v', por último tenemos el tratamiento del ingreso de la letra 'm' que tiene que hacer tres verificaciones diferentes, primero verificaremos si estamos en presencia de una bomba, por lo que hacemos una consulta al *tableroOculto* utilizando el método *esBomba*, si esto es verdadero a la bandera que designamos para manejar el juego llamada 'jugando' le asignaremos falso para que el juego termine ya que perdimos la partida por develar una bomba y pasaremos al tablero visible en la posición actual el sring '@' indicando que develamos una bomba. En segundo lugar verificaremos si la casilla donde estamos parados es un elemento cualquiera, por lo que haciendo la misma consulta pero esta vez utilizando el método *esElemento*, dentro de la misma consulta, con condicionales tendremos dos vertientes diferentes, uno si el valor de objeto Elemento es '0' o si el valor es diferente de '0', para ambos tendremos que lo primero es mostrar el valor que tiene en el *tableroVisible*, y si se trata de un valor igual a '0' ejecutaremos el método llamado *rellenado*[1] que tiene como función develar todos los 0 adyacentes al '0' actual, como lo explicamos cuando hicimos la explicación del método. Si no estamos en presencia de un valor igual a '0' simplemente sumaremos un puntaje y lo revelaremos al *tableroVisible* ya que estamos en presencia de una pista.

3. Conclusiones

El Buscaminas es un juego fácil de entender, con escasas reglas y muy sencillas, pero aún con la facilidad de entender el juego, el proyecto representó un desafío logístico a la hora de implementar. Como equipo enfrentamos varias dificultades no solamente con el desarrollo del código encargado de la lógica detrás del juego per se sino también su implementación gráfica, fue un reto a superar poder visualizar los medios que utilizaríamos para llevar a cabo esta tarea, al final tomamos la decisión de hacer con una interfaz gráfica sencilla a través de la ventana Transcript de Pharo. Uno de estos problemas fue el tener que pensar todo el juego como dos matrices superpuestas entre sí, pero una vez superado este problema, en realidad este planteo del juego como tal nos trajo grandes beneficios, como por ejemplo ver con nuestro propio desarrollo el poder que tienen los lenguajes Orientados a Objetos, ya que con escasos métodos resolvimos la lógica del juego, consideramos que el planteo del modelo de clases fue suficiente para poder abarcar todos los aspectos más importantes del juego, la colaboración entre objetos, el envío de mensajes y cómo esto nos dio un punto de vista más amplio a la hora de desarrollar un software.

Por último, queremos destacar que el trabajo en éste tipo de lenguajes nos hizo ver la importancia de la programación en bloques[2], ya que el equipo mantuvo una comunicación fluida pero a su vez cada integrante tenía encargado un bloque de código a desarrollar y esto hizo que el avance del proyecto haya sido un éxito en términos de resolver el problema en un tiempo definido por la cátedra.

Bibliografía

[1] https://es.wikipedia.org/wiki/Algoritmo_de_relleno_por_difusi%C3%B3n#:~:text=El%20algoritmo%20de%20relleno%20por,contiguos%20en%20una%20matriz%20multidimensional.

[2] Andrew P. Black, Damien Pollet, Stéphane Ducasse, Oscar Nierstrasz (2010) Pharo by Example 5.0