Proyecto 3

Alejandro Hernández Cano
Martínez Santana Brayan
Trad Mateos Kethrim Guadalupe

24 de octubre de 2019

Repositorio: `https://github.com/BrayanCaro/BuscaminasVala`

1. Introducción

Todos estamos familiarizados con el juego de buscaminas. El objetivo de este es simple: descubrir totalmente un campo minado sin llegar a activar una de estas y las reglas del juego sencillas. Además, puede ser un reto interesante al momento de programar, pues estar sujeto a varios diseños y depende de cada uno el que le parezca más adecuado.

Este documento presenta y analiza la implementación de un Buscaminas en el lenguaje de programación Vala. Presentamos un proyecto completo con pruebas y documentación de un buen juego buscaminas que busca mejorar el que ya teníamos hecho al inicio de la carrera, haciendo un código más compacto, siguiendo principos antes extraños, que fueron vistos durante el curso y ofreciendo una interfaz al usuario más amigable que en el proyecto inicial.

2. Definición del problema

El reto de este proyecto consiste en implementar correctamente un juego de buscaminas en Vala, pero también se pueda guardar la partida en curso y poder cargar esta partida posteriormente. Esto se exitende desde aprender a utilizar el lenguaje Vala, manipular la serialización de objetos en este y poder realizar el diseño de el proyecto elegantemente.

3. Análisis del problema

Las reglas del buscaminas son sencillas. Inicia en un tablero de tamaño $n \times m$, con todas las casillas iguales. En cada turno el jugador decidirá presionar una de estas casillas o colocar una bandera. La bandera sirve únicamente para ayudarle al usuario a recordar las casillas en donde se cree hay una mina.

Si el jugador decide presionar una casilla y esta contiene una mina, se perderá la partida. Si no contenía una mina entonces se extenderá la zona descubierta hasta que todas las casillas de la zona tengan al menos a una casilla con bomba adyacente a ella. En cada una de estas casillas se mostrará un número, que indica cuántas casillas adyacentes contienen una mina. Este número le servirá al usuario para poder deducir en cuales casillas hay minas y en cuales no las hay.

Además, nuestro programa podrá ser capaz de:

1. Guardar y cargar partidas.
2. Tener en una tabla los mejores puntajes.
3. Dar la oportunidad de que el usuario decida el tamaño del tablero.

4. Mejor alternativa

Parte del reto de este proyecto fue el considerar en qué lenguaje realizar el juego. Después de varias opciones analizadas se terminó optando por, como ya fue mencionado anteriormente, Vala. A continuación daremos una pequeña descripción de este lenguaje y daremos varias justificaciones de que este lenguaje es buena propuesta para nuestro problema.

Vala es un lenguaje de alto nivel orientado a objetos con un compilador que genera código en C. Vala es sintácticamente similar a C# e incluye notables características como funciones anónimas, propiedades, genéricos, manejo de memoria asistida, manejo de excepciones, inferencia de tipos y el bucle foreach.

Creemos que esta es una buena alternativa para realizar nuestro buscaminas pues es relativamente sencillo sintáctimanete y además ofrece varias características útiles, como las mencionadas anteriormente, que facilitan el escribir código. Además al ser compilado a C podemos esperar un programa final relativamente ligero y fácilmente distribuible.

5. Pseudocódigo

En esta sección describiremos los algoritmos utilizados para hacer posible este proyecto. Hay que tomar en cuenta que todos estos algoritmos están encapsulados dentro de una clase `Tablero`, por lo tanto hacemos la suposición de que al inicio de cada algoritmo, el tablero se encuentra en un estado válido, además, dentro de estos, denotaremos por n y m la cantidad de filas y columnas en el tablero, respectivamente y por *casillas* la matriz que representa las casillas del tablero.

Algorithm 1 Presionar casilla

```
1:  procedure PresionarCasilla(x, y)
Require: CasillaValida(x, y)
2:      casillas[x, y].marcarPresionado( )
3:      Extender(x, y)
4:  procedure Extender(x, y)
5:      if casillas[x, y].bombasVecinas > 0 then                ▷ Caso base
6:          return
7:      coords ← [(x, y + 1), (x − 1, y), (x + 1, y), (x, y − 1)]   ▷ Vecinos arriba/abajo y de los lados
8:      for all (i, j) ∈ coords do
9:          if CasillaValida(i, j) then
10:             c ← casillas[i, j]
11:             if not c.minado and not c.abanderado and not c.presionado then
12:                 c.marcarPresionado( )
13:                 Extender(i, j)
14: procedure CasillaValida(x, y)
15:     if 0 ≤ x < n and 0 ≤ y < m then
16:         return true
17:     return false
```

6. Pruebas y mejoras

La mejora principal que vio nuestro programa es en el diseño. Como el buscaminas original fue realizado al inicio de la carrera, nuestros conocimientos en diseño eran muy pobres y como a penas estábamos practicando los conceptos básicos de programación orientada a objetos, termino estando muy mal diseñado y sin estar tan claro.

Por otro lado, el nuevo buscaminas ve una mejora significativa en este rubro, reducimos la cantidad de clases notablemente simplificando el código, el diseño y la implementación, tanto en la interfaz con el usuario como en el

modelo de nuestro tablero. Además esta vez incluimos pruebas unitarias, ayudándonos a depurar más facilemente nuestro código y poder tener más confianza en el correcto funcionamento.

También, al utilizar serialización utilizando el formato json, es más fácil poder guardar estos tableros y leerlos por otros programas, si eso llegara a ser necesario.

Otra gran mejora que se realizó fue la manera en la que se compila y ejecuta el programa. En las propuestas anteriores utilizabamos `make`, que a pesar de que cumplía con el objetivo, no había garantía de que lo hiciera en todos los ordenadores, pues nos enfrentamos con problema de compilación en algunos. Otra alternativa que se tenía era compilar todo el proyecto manualmente y tener que ejecutarlo utilizando el ejecutador de java. Esto resultaba algo engorroso y además se neciesitaba que el usuario tuviera conocimientos sobre java y tuviera y supiera utilizar las herramientas adecuadas.

En esta implementación utilizamos `mason`, que resulta ser una alternativa más fiable que `make`, dando la misma facilidad para compilar utilizando un solo comando, pero teniendo más fiabilidad de que lo haga correctamente. Además esta herramienta arroja un archivio ejecutable, en vez de un archivo empaquetado `jar`, lo que facilita la distribución y ejecución de nuestro proyecto, sin siquiera necesitar instalar ninguna herramienta más.

7. Pensamiento a futuro

El programa podría ponerse en las tiendas de alguna plataforma de software como en 'Software de Ubuntu' porque Vala esta diseñado para la creación de aplicaciones (usando las bibliotecas de GNOME).

Bibliografía

1. Tutorial de Vala
   https://wiki.gnome.org/Projects/Vala/Tutorial/es

2. Introducción a Vala
   https://www.youtube.com/watch?v=hkUnly_Viys