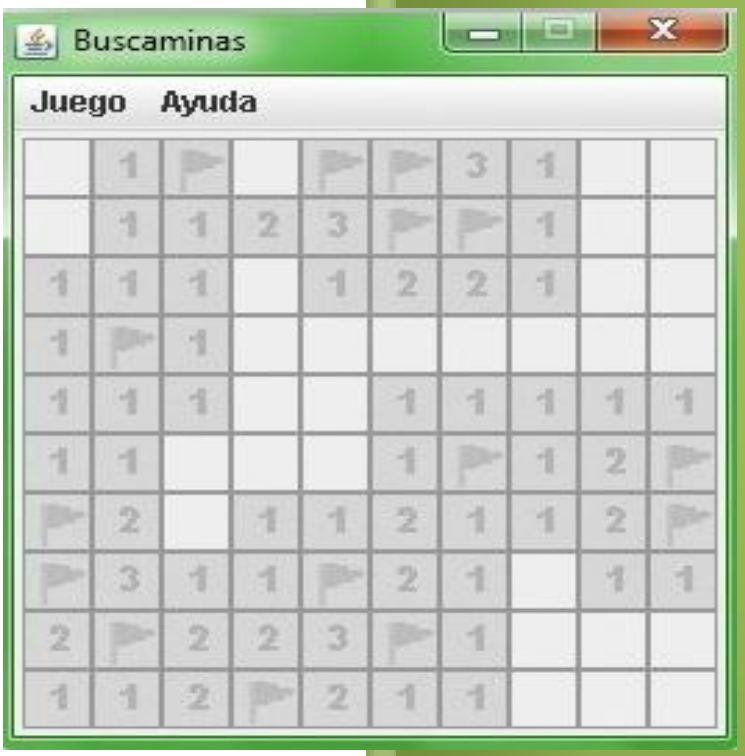
Programación Modular
Orientada a Objetos

Buscaminas



Grupo V3
Helen Paz Cortón, Egoitz Puerta Beldarrain,
Lucía Rodero Pascual, Itsaso Sabino Urbieta
Fecha de defensa: 13 de mayo de 2011
Fecha de Entrega: 21 de mayo de 2011

Índice

Introducción.....2

Diseño: diagrama de clases.....4

Diseño: diagrama de secuencia/algoritmo 6

Casos de prueba: diseño de las Junit 8

Excepciones.....9

Aspectos más destacables de la implementación 10

Conclusiones.....11

Introducción

El juego que presentamos es el Buscaminas, es un juego clásico muy conocido debido a que suele venir instalado en los ordenadores de Windows. En un principio, para la elección del juego nos basamos en una idea muy general de cómo podía ser tratado, no vimos excesiva complicación en su realización pero tampoco nos pareció sencillo. Al desconocer el funcionamiento del juego suponía un reto bastante grande por lo que nuestro primer objetivo residió en informarse sobre esto. En lo que se refiere al funcionamiento del Buscaminas realizado intenta ser lo más similar posible al Buscaminas de Windows. Se trata de ir resolviendo el tablero descubriendo sus diferentes casillas intentando no caer en ninguna bomba, para eso, las casillas marcadas muestran una serie de números, indicando las bombas que hay alrededor y en el caso de intuir que en una casilla hay una bomba existe la opción de marcar las casillas con una bandera. Se pierde cuando se pulsa una bomba, mientras que se gana cuando se han descubierto todas las casillas que no tenían bombas.

Por otro lado, nuestros objetivos principales radicaban en una realización correcta y eficiente de las clases pensadas y sus respectivas Junits aplicando los conocimientos adquiridos durante el curso así como la realización del diagrama de clases y de secuencia y en un plano más personal, el pleno conocimiento del funcionamiento de todas las partes del proyecto. Análogamente, declaramos nuestros objetivos secundarios que tenían como fin realizar una interfaz gráfica, varios niveles de dificultad, un cronómetro que midiese el tiempo jugado y un contador que indicase la puntuación. Tanto los objetivos principales como los secundarios han sido realizados con

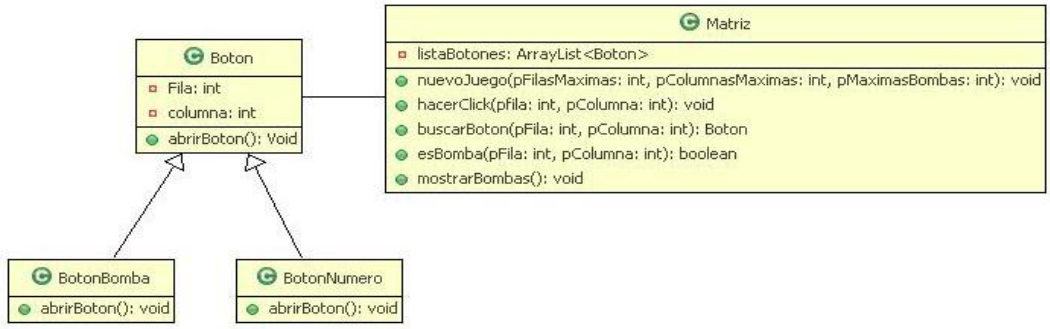
éxito a excepción de los dos últimos objetivos secundarios propuestos, el cronómetro y el contador, que por razones de tiempo no pudimos elaborar.

Del mismo modo que con los objetivos, realizamos una planificación de las horas que dedicaríamos al desarrollo del proyecto. En un primer momento se habló de tres horas individuales semanalmente más las dos horas de los laboratorios y hora y media de reuniones. Las horas se han cumplido tal y como se proyectaron con alguna variación en las horas individuales que hubo algunas semanas en las que fueron más y otras menos.

Para el reparto de tareas, en un principio se planteó que Egoitz Puerta y Helen Paz realizarían las Junits pertinentes a las clases y que Itxaso Sabino y Lucía Rodero harían las clases. Finalmente, nos dividimos las clases, de ésta manera Helen Paz realizó las clases *JuegoNuevo* y *TableroVisual*, Egoitz Puerta desarrolló la clase *Tablero* y Lucía Rodero e Itxaso Sabino elaboraron las clases *Casilla*, *CasillaNumero* y *CasillaBomba* y, por consiguiente, cada uno realizó las Junits de las clases asignadas. Cabe destacar que para realizar el proyecto utilizamos el servicio de alojamiento de archivos Dropbox, lo utilizamos como *workspace* y, de esta manera, pudimos tener el proyecto actualizado en todo momento. Además, gracias a esto, localizamos errores existentes en el código, ya fuese en el que teníamos asignado o no. Esto ayudó a que el trabajo avanzase a un buen ritmo y todos tuviésemos conocimientos sobre lo que se hacía en el resto de clases.

Diagrama de clases

Diagrama Inicial



En un principio, las clases pensadas para el juego eran *Boton,* con su herencia *BotonBomba* y *BotonNumero* y *Matriz*. Durante el desarrollo del juego hemos tenido que implementar más clases como son *JuegoNuevo*, la cual se encontraba en este diseño inicial en la clase *Matriz*, que se encarga de iniciar un juego nuevo. Asimismo, debido al desarrollo de la interfaz gráfica hemos añadido la clase *TableroVisual* cuya función es la de transformar la información que tenemos de manera gráfica, así como, facilitar la interacción del usuario con el juego ya que jugar por consola es más complicado y con la interfaz se hace más atractivo.

Por otra parte, y siguiendo el consejo de Javier López, cambiamos los nombres de las clases para que reflejen de manera más precisa su función. En nuestro diagrama final la clase Matriz se denomina *Tablero*, las clases *Boton*, *BotonNumero* y *BotonBomba* se renombraron como *Casilla*, *CasillaNumero* y *CasillaBomba* respectivamente. Del mismo modo, siguiendo el consejo de Javier Lopez, se realizó la clase *JuegoNuevo* mencionada antes.

Los métodos que aparecen en el primer diagrama los hemos mantenido todos aunque algunos tienen una denominación diferente así como: *hacerClick* se corresponde con *pulsarCasilla* al igual que *buscarBoton* que ahora se denomina *buscarCasilla* y *abrirBoton* de las clases *Boton*, *BotonNumero*, *BotonBomba* es el método *mostrarCasilla* de *Casilla*, *CasillaNumero* y *CasillaBomba*.

Diagrama final

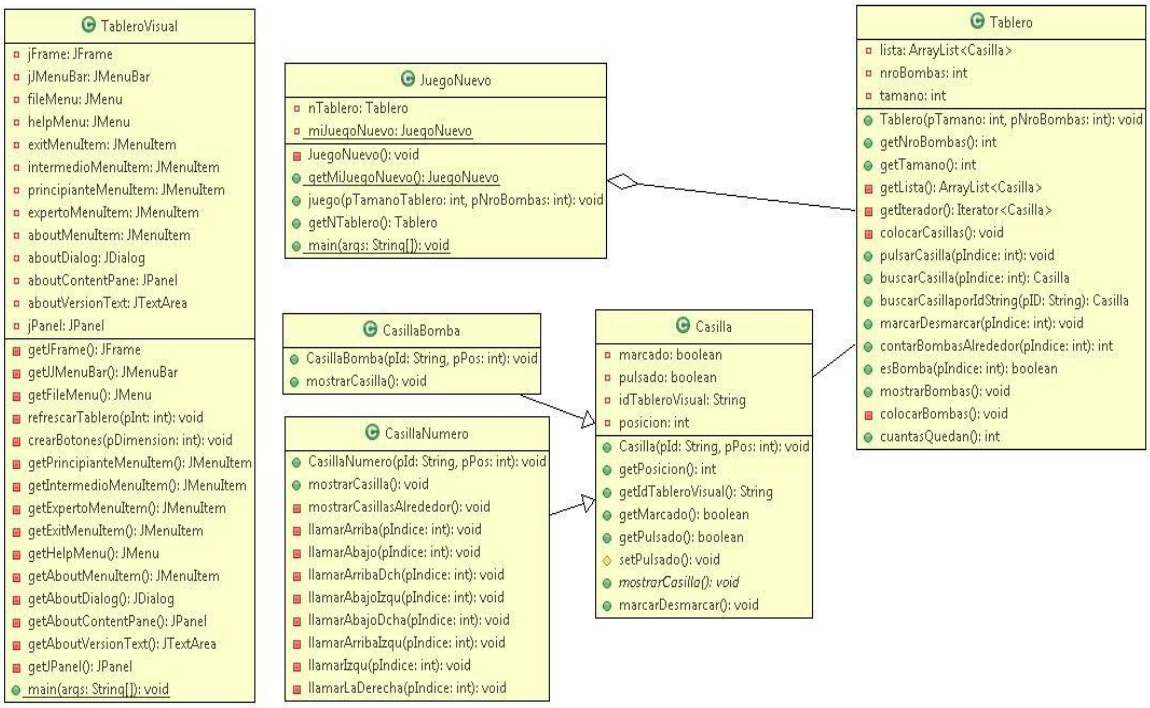
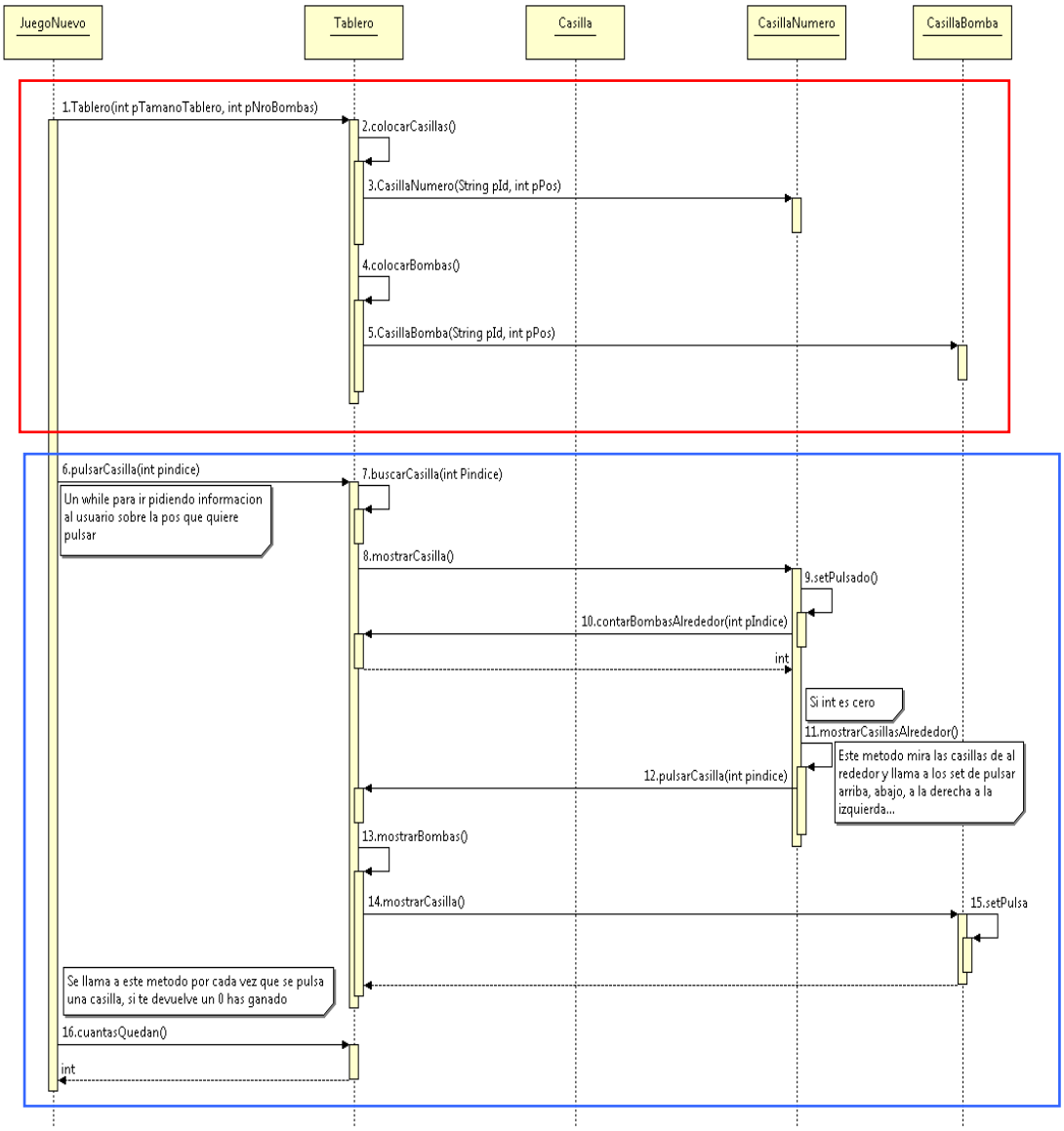


Diagrama de secuencia



En la primera parte, la clase *JuegoNuevo* invoca a *Tablero* enviandole dos parámetros: la dimensión que va a tener el tablero y el número de bombas. La constructora de *Tablero* llama a su método *colocarCasillas* y éste invoca a la constructora de *CasillaNumero* pasándole dos parámetros: un id (identificador que utiliza la clase *TableroVisual*) y la posición (el indice que tiene cada casilla). La constructora del *Tablero* llama a *colocarBombas* al que le pasa un integer (el número de bombas) y éste, a su vez, llama a la constructora de *CasillaBomba*.

La segunda parte representaría una nueva partida, la clase *JuegoNuevo* llama a *pulsarCasilla* al que le pasa como parámetro la posición de esa casilla. De este modo la clase *Tablero* llama a su método *buscarCasilla*, a la que le envia la posición de la casilla y aquí, se pueden dar dos casos:

- El caso en el que la casilla sea una bomba. En este caso el tablero llama al método *mostrarBombas* que se encarga de descubrir todas las bombas y de finalizar el juego.
- El caso en el que la casilla sea un número. En este caso el tablero, mediante el método *mostrarCasillas*, llama a la clase *casillaNumero*, ésta se encarga de ejecutar el *setPulsado* y de llamar a la clase *Tablero* mediante el método de *contarBombasAlrededor* al que se le envia la posición de la casilla. La clase *Tablero* devuelve el número de bombas que hay alrededor y, dependiendo del resultado que nos devuelva, se hará una cosa u otra:
    - Si la casilla es cero: la casilla llamará a su método *mostrarCasillasAlrededor*, que se encarga de pulsar las casillas contiguas utilizando diferentes métodos, y por cada casilla de alrededor se llama nuevamente a *pulsarCasilla*.
    - Si la casilla es un número: la casilla es pulsada.

Al pulsar una casilla, la clase *JuegoNuevo* llama al método *cuantasQuedan* de la clase *Tablero*, la cual te devuelve cuantas casillas de números quedan por descubrir. En el caso de que no quede ninguna finaliza el juego.

Diseño de jUnits

En el test de la MAE *JuegoNuevo* se testean sus diferentes métodos. El test *testGetMiJuegoNuevo* y *testGetNTablero* se encargan de comprobar que no se devuelve un null. Por otra parte, *testJuego* comprueba que el tablero ha sido creado con las bombas y las filas indicadas. Cabe destacar que no se implementó el método *reiniciar*, por lo que no hizo falta hacer su test. Asimismo la constructora no se testeo puesto que no tenía código que probar.

En la clase *Tablero* se probaron todos los métodos y, en principio, todos los casos posibles. En la constructora *Tablero* se comprobó que se crease bien el tablero, es decir, que se colocasen bien los atributos. En los *getters* se verificó que no devolvían null. Por otra parte, en *testColocarCasillas* se cotejó que el número fuera el correcto. En *testColocarBombas* nos aseguramos que el número de bombas fuera el mismo que el indicado en el atributo. Para el método *testPulsarCasilla* se comprobó que se cambiaba el atributo correspondiente en la casilla asignada, del mismo modo se miró que imprimía el fallo correspondiente. En el método *testBuscarCasilla* se examinó que la casilla devuelta por el método y la casilla buscada en la prueba coincidían, y, como anteriormente, se miró si imprimía el fallo. Algo semejante ocurre con el test *testBuscarCasillaporIdString*. En *testMarcarDesmarcar* constatamos que se cambiaba correctamente el atributo de la casilla introducida. Para *testContarBombasAlrededor* se crean dos tableros que nunca se darían en el juego: uno completo de bombas y otro que carece de ellas y se comprueban todas las posibilidades que se pueden presentar, considerando, también, el caso de que sea fuera de rango, en cuyo caso se presenta un error por consola. Lo que se hace en *testEsBomba* es comprobar que la casilla es o no bomba. En *testMostrarBombas* se mira por pantalla que se han impreso por

pantalla todas las posiciones de las bombas. Por ultimo en *testCuantasQuedan* nos aseguramos de que se devuelve el número correcto de casillas que hay sin pulsar.

En la clase Casilla se probaron los *getters*. En el test del método *testSetPulsado* se aseguraba que se cambiaba correctamente el valor del booleano. El *testMarcarDesmarcar* coteja que se cambia correctamente el valor del atributo *marcado*. Esta clase es abstracta por lo que no es necesario realizar un test de la constructora *Casilla* ni del método *mostrarCasilla*.

Análogamente, en la clase *CasillaNumero* se comprueban de nuevo los métodos que hereda de la clase *Casilla*. En la prueba de la constructora nos aseguramos de que los atributos son inicializados de manera correcta. En *testMostrarCasillasAlrededor* se constata que se pulsan todas las casillas de un tablero sin bombas, que se ha creado previamente. Por último, se prueban los similares métodos "*llamar*", que utiliza el método *MostrarCasillasAlrededor*, en los que se comprueba que ha sido pulsada la casilla correspondiente. Estos métodos, en un principio, son métodos privados pero para poder realizar sus pruebas es necesario cambiar su visibilidad.

En la clase *CasillaBomba*, como en la anterior, se cotejan los métodos heredados de Casilla y la constructora. Además, en *testMostrarCasilla* se verifica que se ha impreso por pantalla lo que debe, es decir, la posición de la bomba y el valor del atributo *pulsado* modificado.

Excepciones

En cuanto a las excepciones hemos identificado dos: el caso excepcional que indica si está fuera de rango y la excepción que comprobaría que la casilla no está marcada ni pulsada.

La que hemos tratado es la que indica si el indice está fuera de rango. Si el parámetro introducido está fuera del rango o es distinto a un integer, entonces se produce la excepción. La excepción ha sido tratada en el método main de TableroVisual.

Aspectos más destacables de la implementación

Los aspectos más detacables de la implementación los hemos dividido en tres bloques: recursividad, herencia y tratamiento de algunos métodos. En lo concerniente a la recursividad se ve reflejada en el método *mostrarCasillasAlrededor* de la clase *CasillaNumero*. La función de éste método es que cada vez que el número de bombas que se encuentran alrededor de una casilla es cero el método se encarga de ir pulsando las casillas alrededor y así, de manera continua, hasta que el número de bombas de la casilla en cuestión sea distinto de cero.

La herencia viene determinada en las clases *CasillaNumero* y *CasillaBomba* que heredan de *Casilla*. La finalidad de realizar el diseño de ésta manera es evitar la repetición de código ya que ambas clases utilizan los mismos atributos.

Tanto en el método *contarBombasAlrededor* de la clase *Tablero* como en el método *mostrarCasillasAlrededor* de la clase *CasillaNumero* el código es bastante extenso debido a la utilización de varios if's anidados. Sin embargo, esto se podría haber evitado modulando el código, es decir, creando más métodos.

Conclusión

En un principio, cuando todavía no teníamos decidido hacer el buscaminas, simplemente era una posibilidad entre muchas, pensamos que podía resultar complicado, ya que visto desde fuera el código nos parecía algo fuera de nuestro alcance. A medida que la idea de desarrollar dicho juego se iba haciendo más fuerte, empezamos a plantearnos las clases pertinentes para su codificación. Durante la *brainstorming* fuimos observando que las clases eran mínimas, únicamente cuatro, y los métodos relacionados con ellas también. Esto nos motivo, y por ello decidimos entre tantos, escoger el buscaminas.

Durante el desarrollo del juego, nuestra idea principal se fue desmoronando, ya que vimos que necesitábamos bastantes más métodos y algunas clases más de las que habíamos ideado en un principio. Tuvimos que replantearnos nuevos retos, que ni siquiera se nos habían ocurrido, sobre todo a nivel de diseño. También, algunos métodos demasiado extensos nos supusieron problemas porque teníamos pequeños errores que no éramos capaces de detectar.

Al final, el juego lo hemos podido realizar con éxito. Además, la implementación de la interfaz gráfica ha hecho el juego mucho más atractivo para el usuario. En general, nos sentimos muy orgullosos de haber sido capaces de hacer el buscaminas, y el resultado de nuestra dedicación y tenacidad está a la vista.

La mayoría de los objetivos que nos habíamos marcado los hemos cumplido, nuestra prioridad ha sido cumplir aquellos que creíamos necesarios, y en función del tiempo disponible ir realizando los otros que habíamos marcado como objetivos secundarios. Entre los objetivos principales, los cuales hemos cumplido todos ellos, se encuentran la realización de los diagramas UML y el diagrama de clases. Asimismo, hemos implementado las clases y jUnits

pertinentes, Para realizar cada uno de estos propósitos hemos empleado los conocimientos adquiridos a lo largo del curso.

Por otra parte, aunque nos hubiera encantado poder decir que también hemos conseguido todos los objetivos secundarios, no es así, debido al tiempo. Los objetivos secundarios que hemos logrado son: realización de la interfaz gráfica, además hemos capacitado al juego de varios niveles de dificultad, van del principiante al nivel experto, la diferencia radica en el número de casillas y el número de bombas asignado en cada panel. Aquellos otros que se han quedado en el tintero eran el de realizar un cronómetro, al igual que tiene el buscaminas de Windows, así como realizar un contador para la puntuación. Como se puede observar son aplicaciones que hubieran hecho al juego más atractivo y a semejanza plena del buscaminas de Windows, pero son detalles que no perturban la calidad del mismo.

En cuanto a mejoras son varias las que podemos destacar. En primer lugar, deberíamos haber controlado todas las excepciones que se encuentran a lo largo del desarrollo del juego, ya que solamente hemos capturado algunas de ellas. Del mismo modo, deberíamos haber modulado ciertos métodos que nos han quedado muy extensos, porque de este modo hubiera resultado más fácil a la hora de trabajar.

Siguiendo en esta línea, aunque en un orden de menor prioridad respecto a las anteriores ventajas, para que el juego hubiera sido idéntico al de Windows, las bombas deberían haber salido al segundo click. También, fue algo que nos planteamos, pero no lo llevamos a cabo porque la dificultad del juego aumentaba notablemente. Como hemos mencionado anteriormente, no hemos conseguido cumplir todos los objetivos secundarios, con lo que de haberlos cumplido hubiéramos mejorado el juego.

Otra de las mejoras es en vez de tener simplemente tres niveles de dificultad, haber tenido solamente un nivel pero con dificultad personal. Es decir, que el usuario tenga la posibilidad de elegir el número de bombas y el número de casillas y filas del juego. Por último, deberíamos haber asignado a cada casilla unas coordenadas x e y.

Los problemas encontrados durante el desarrollo del juego han sido mínimos. Simplemente pequeños fallos cometidos a la hora de realizar el código, uno de nuestros problemas lo teníamos en la parte recursiva del juego, ya que se nos presentaba un bucle, el cual no salía de dos casilla contiguas. Gracias a la realización de pequeñas modificaciones en el código conseguimos arreglarlo, porque el problema era que no estaban bien realizadas alguna de las llamadas. Por otra parte, tuvimos problemas en otro de los métodos, el de *mostrarCasillasAlrededor* que realizaba su función correctamente, este método está formado por varios ifs anidados, con lo que nos resultó costoso encontrar el fallo, al final con paciencia lo encontramos, era simplemente un paréntesis que se nos había olvidado.

Las conclusiones que hemos extraído a lo largo del proyecto son las siguientes:

En primer lugar, el buscaminas nos parecía un juego complicado, no sólo a la hora de realizar el juego, sino porque nunca habíamos sabido jugar con éxito. Nos parecía un reto hacer un juego, el cual nunca habíamos sabido cómo funcionaba. Por eso, una de nuestras conclusiones es que al realizar el juego hemos aprendido a jugar.

Por otra parte, uno de nuestros objetivos secundarios era realizar la interfaz gráfica, objetivo el cual hemos cumplido, con lo que podemos decir que hemos adquirido conocimientos para representar la información y acciones disponibles utilizando para ello un conjunto de imágenes y objetos gráficos. Además, hemos facilitado la interacción del usuario con el juego, ya que jugar a través de la consola se hace duro.

Nos ha servido para aclarar conceptos generales de la asignatura, así como para afianzar la teoría desarrollada. En general, hemos consolidado lo aprendido durante el curso, y hemos visto cómo podemos aplicar los conocimientos que tenemos a un caso que creemos más real como es en la aplicación para un juego.

También hay un gran cambio desde la idea inicial que tuvimos sobre el juego, es decir, en un primer momento cuando nos planteamos cuales debían ser las clases, los atributos y los métodos de dichas clases, eran cuatro. Aunque en cuanto al número de clases no ha habido mucha diferencia, el conjunto de métodos si ha aumentado notablemente. Es decir, hasta que no nos metimos en el juego y empezamos a profundizar en él, no hemos sido conscientes de todo lo que necesitábamos.

Por último, el trabajo en equipo es duro. En muchas ocasiones y siendo personas tan diferentes nos ha costado unir nuestros puntos de vista, y ha sido difícil consensuar las opiniones de cada uno de los miembros del grupo. Esta es una de las competencias que tenemos que adquirir a lo largo de la carrera porque el día de mañana nuestro trabajo diario va a consistir en eso, por ello tiene que ser algo que debemos ir mejorando.

En un principio el tiempo de dedicación al trabajo que nos habíamos planteado era de seis horas y media a la semana, dividido en tres horas semanales individuales, dos horas en equipo y una hora y media de reuniones conjuntas. Al final el tiempo real destinado a la realización del buscaminas ha sido de dos horas en conjunto, dependiendo de la semana una o dos veces, las horas de los laboratorios, y horas individuales, cada uno de nosotros las horas que ha visto necesarias para llevar a cabo su parte.

Asimismo, en un principio pensamos dividir el trabajo en dos partes que eran: la realización de las jUnits y el desarrollo de las clases. La primera parte la iban a desarrollar Helen Paz y Egoitz Puerta, así como la segunda parte estaría a cargo de Lucía Rodero e Itsaso Sabino. Aunque nos parecía una buena partición vimos que no era correcto realizarlo de este modo, por lo que finalmente Egoitz Puerta ha realizado la clase Tablero, Helen Paz las clases *JuegoNuevo* y *TableroVisual*, y Lucía Rodero e Itsaso Sabino las clases Casilla, *CasillaBomba* y *CasillaNumero*, cada uno con sus respectivas jUnits.

En general nos parece que hemos elegido un correcto diseño del juego, no se nos ocurre otra forma de haberlo hecho, salvo en un detalle. La idea original era desarrollar el juego con una matriz en vez de con un arraylist, pero al final Javier López nos aconsejó que lo realizásemos con un arraylist porque nos iba a resultar en su opinión más sencillo. Por este motivo, utilizamos un arraylist en vez de una matriz.