Universidad del Pais Vasco

Escuela Universitaria de Ingenieria Tecnica Industrial de Bilbao

Ingenieria del Software

Proyecto Buscaminas

Andrea Siles
Sergio Amorrosta
Andoni Teso
Ivan Salazar

Ingenieria Informática de Gestión y Sistemas de Información

May 13, 2016

Contents

1 Introduccion

El objetivo final de este proyecto será la realización (diseño e implementación) del popular juego "Buscaminas" que, de forma habitual, encontramos en los entornos de Windows.

El jugador puede hacer tres tipos de acciones sobre el tablero: descubrir una casilla, marcar una casilla y quitar la marca de una casilla.
El juego consiste en despejar todas las casillas de una pantalla que no oculten una mina. Algunas casillas tienen un número, el cual indica la cantidad de minas que hay en las casillas circundantes.
Si se descubre una casilla sin número indica que ninguna de las casillas vecinas tiene mina y éstas se descubren automáticamente. Si se descubre una casilla con una mina se pierde la partida. Se puede poner una marca en las casillas que el jugador piensa que hay minas para ayudar a descubrir las que están cerca.

El juego también posee un sistema de ranking con los jugadores con mejor puntuación para cada uno de los 3 niveles. Cuanto menos tiempo se finalice el juego, mayor puntuación se obtendrá. Los niveles son:
- Nivel principiante: 7x10 casillas y 10 minas
- Nivel intermedio: 10x15 casillas y 30 minas
- Nivel experto: 12x25 casillas y 75 minas

2 Organización

Nuestra manera de organizarnos se asemeja, en la medida de lo posible, a el proceso SCRUM. Por lo que el proyecto se ha llevado a cabo en una secuencia de 3 Sprints.

Cada uno con una duración de 4 semanas. El producto se diseña, implementa y prueba en cada Sprint. En cada Sprint seleccionábamos las tareas del proyecto (previamente elegidas) que íbamos a realizar. Al comienzo de éste componemos la Planificación de Tareas, en la que asignamos a cada miembro del grupo una serie de tareas que deberá terminar de cara a la finalización del Sprint. En caso de que no consiga finalizarlo se volverá a añadir a la Planificación de Tareas del siguiente Sprint.

Nos reunimos al menos una vez a la semana en el aula de estudio de la Biblioteca de la EUITI. En cuanto a las reuniones semanales, su duración

era de al menos 3 horas y nos dedicamos cada miembro a explicar lo que hemos avanzado en nuestra parte correspondiente para que la entiendan el resto del grupo y los problemas que le han surgido en caso de que otro lo pueda resolver, preparamos el próximo sprint en caso de que se acercase su entrega...

También hemos tenido ocasión de hacer reuniones de corta duración las veces que coincidíamos y donde exponíamos brevemente lo realizado.

Normalmente las tareas las dividíamos en parte de diseño (diagramas de clase y de secuencia), Implementación de las clases, pruebas del proyecto (documento de pruebas y JUnits), Interfaz gráfica (a partir del segundo sprint) y Documentación técnica (en el tercer sprint).

Las pruebas las realizamos en un documento el cual se anexará a la documentación, en donde apuntábamos las pruebas realizadas, las condiciones, lo que se esperaba que ocurriese y los errores que surgieron en caso de que hubiese. Además, para los métodos más importantes hemos hecho uso de las JUnits.

Hemos hecho también un acta de reunión de cada vez que nos reuníamos en donde se expone brevemente los objetivos y lo que hemos decidido (también se anexarán a la documentación) También hemos utilizado la herramienta GitHub donde está alojado nuestro proyecto y la plataforma drive.

Para la documentación técnica utilizamos el editor del lenguaje LaTeX TeXstudio.

3 Diseño completo

3.1 Diagrama de clase

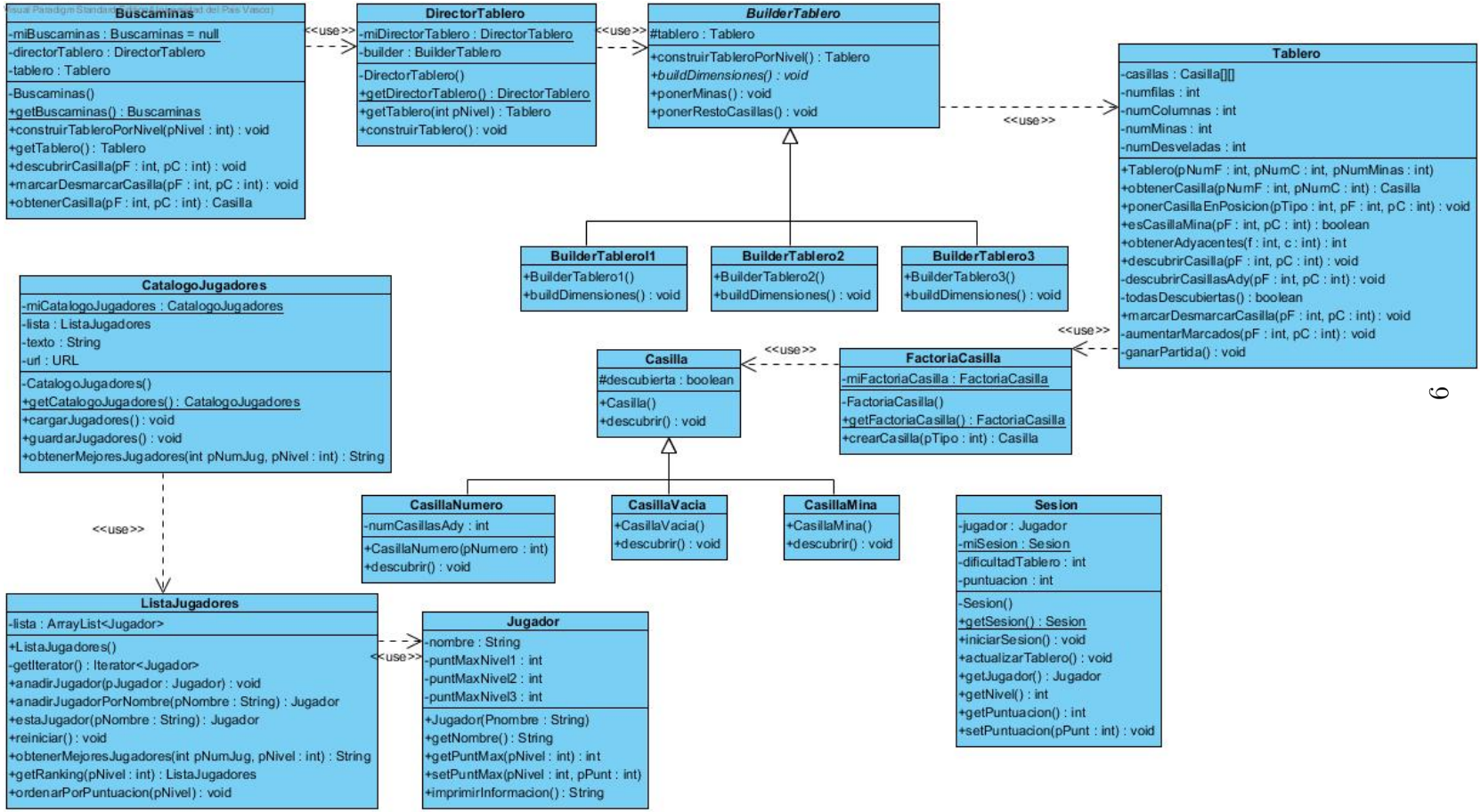
En el cual se pueden distinguir algunos de los patrones que hemos utizado, como el singleton para asegurar que hay una única instancia de las clases Buscaminas, DirectorTablero, FactoriaCasilla, Sesion y CatalogoJugadores.

También hemos utilizado el patrón builder para la construcción de los tres tipos de tableros, en el que la clase DirectorTablero controla la construcción paso a paso dependiendo del nivel introducido, y en los builders se construyen los tableros con unas dimensiones diferentes para cada tipo de nivel.

Por último, tenemos un patrón Factory para controlar la creación de casillas (que pueden ser con mina, con número y vacías). El método ponerCasillaEnPosición (int pTipo, int fila, int columna) llama al crearCasilla de FactoriaCasilla y nos situa una casilla del tipo que le indiquemos en la posición que elijamos. De esta forma se facilitan el mantenimiento y la extensibilidad en caso por ejemplo de que queramos crear otro tipo de casilla.

El Catalogo jugadores es una MAE que contiene una ListaJugadores y el método obtenerMejoresJugadores(int pNumJug, int pNivel) que me devuelve un String con los pNumJug mejores jugadores del nivel que indiquemos. Esto nos servirá para elaborar nuestro ranking en la VentanaRanking

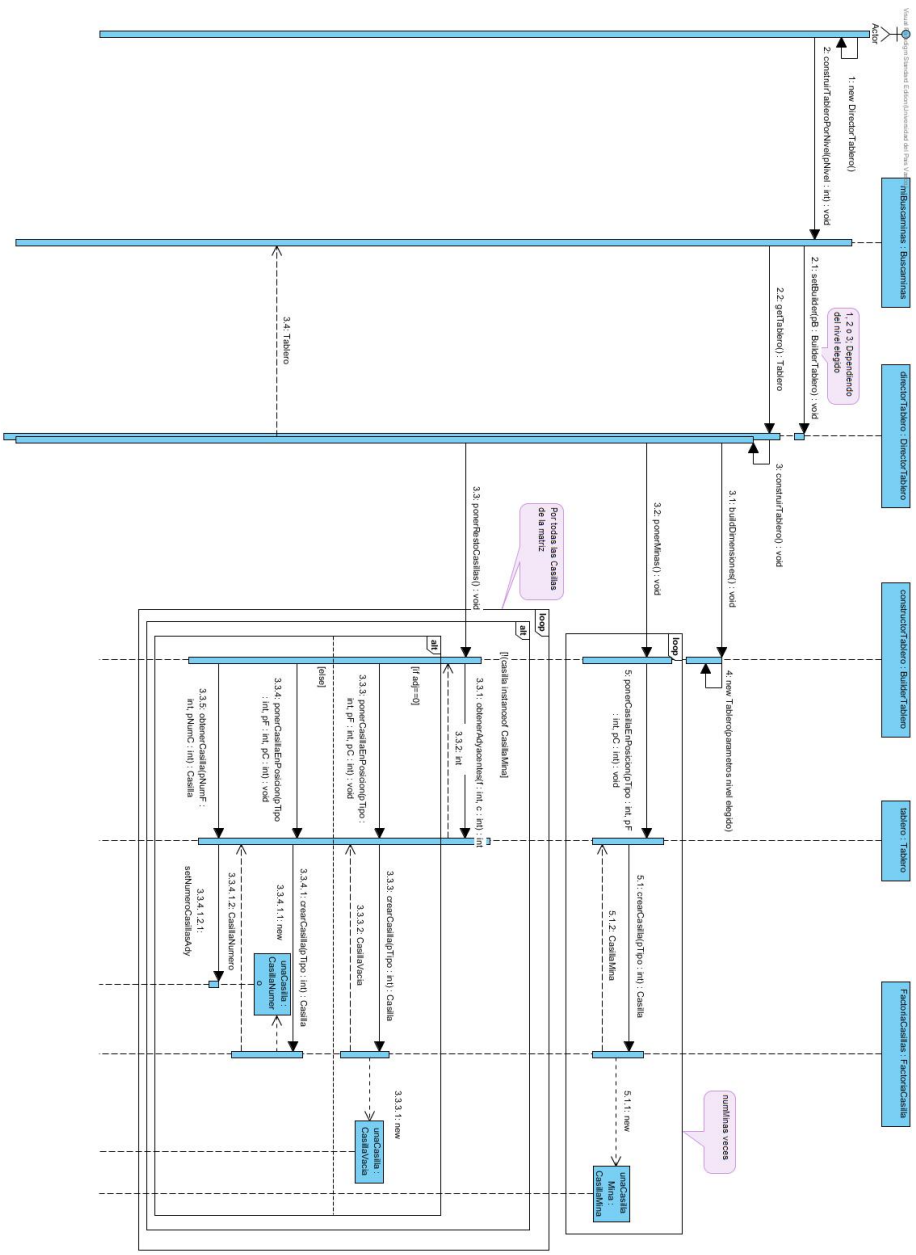
En el caso de la interfaz gráfica hemos utilizado el Modelo Vista Controlador (MVC) para separar el Modelo, la Vista y el Controlador. También hemos utilizado el patrón Observer.


Buscaminas
-miBuscaminas : Buscaminas = null
-directorTablero : DirectorTablero
-tablero : Tablero
-Buscaminas()
+getBuscaminas() : Buscaminas
+construirTableroPorNivel(pNivel : int) : void
+getTablero() : Tablero
+descubrirCasilla(pF : int, pC : int) : void
+marcarDesmarcarCasilla(pF : int, pC : int) : void
+obtenerCasilla(pF : int, pC : int) : Casilla
<<use>>
DirectorTablero
-miDirectorTablero : DirectorTablero
-builder : BuilderTablero
-DirectorTablero()
+getDirectorTablero() : DirectorTablero
+getTablero(int pNivel) : Tablero
+construirTablero() : void
<<use>>
BuilderTablero
#tablero : Tablero
+construirTableroPorNivel() : Tablero
+buildDimensiones() : void
+ponerMinas() : void
+ponerRestoCasillas() : void
<<use>>
Tablero
-casillas : Casilla[][]
-numfilas : int
-numColumnas : int
-numMinas : int
-numDesveladas : int
+Tablero(pNumF : int, pNumC : int, pNumMinas : int)
+obtenerCasilla(pNumF : int, pNumC : int) : Casilla
+ponerCasillaEnPosicion(pTipo : int, pF : int, pC : int) : void
+esCasillaMina(pF : int, pC : int) : boolean
+obtenerAdyacentes(f : int, c : int) : int
+descubrirCasilla(pF : int, pC : int) : void
-descubrirCasillasAdy(pF : int, pC : int) : void
-todasDescubiertas() : boolean
+marcarDesmarcarCasilla(pF : int, pC : int) : void
-aumentarMarcados(pF : int, pC : int) : void
-ganarPartida() : void
BuilderTablero1
+BuilderTablero1()
+buildDimensiones() : void
BuilderTablero2
+BuilderTablero2()
+buildDimensiones() : void
BuilderTablero3
+BuilderTablero3()
+buildDimensiones() : void
CatalogoJugadores
-miCatalogoJugadores : CatalogoJugadores
-lista : ListaJugadores
-texto : String
-url : URL
-CatalogoJugadores()
+getCatalogoJugadores() : CatalogoJugadores
+cargarJugadores() : void
+guardarJugadores() : void
+obtenerMejoresJugadores(int pNumJug, pNivel : int) : String
<<use>>
Casilla
#descubierta : boolean
+Casilla()
+descubrir() : void
<<use>>
FactoriaCasilla
-miFactoriaCasilla : FactoriaCasilla
-FactoriaCasilla()
+getFactoriaCasilla() : FactoriaCasilla
+crearCasilla(pTipo : int) : Casilla
CasillaNumero
-numCasillasAdy : int
+CasillaNumero(pNumero : int)
+descubrir() : void
CasillaVacia
+CasillaVacia()
+descubrir() : void
CasillaMina
+CasillaMina()
+descubrir() : void
Sesion
-jugador : Jugador
-miSesion : Sesion
-dificultadTablero : int
-puntuacion : int
-Sesion()
+getSesion() : Sesion
+iniciarSesion() : void
+actualizarTablero() : void
+getJugador() : Jugador
+getNivel() : int
+getPuntuacion() : int
+setPuntuacion(pPunt : int) : void
ListaJugadores
-lista : ArrayList<Jugador>
+ListaJugadores()
-getIterator() : Iterator<Jugador>
+anadirJugador(pJugador : Jugador) : void
+anadirJugadorPorNombre(pNombre : String) : Jugador
+estaJugador(pNombre : String) : Jugador
+reiniciar() : void
+obtenerMejoresJugadores(int pNumJug, pNivel : int) : String
+getRanking(pNivel : int) : ListaJugadores
+ordenarPorPuntuacion(pNivel) : void
<<use>>
Jugador
-nombre : String
-puntMaxNivel1 : int
-puntMaxNivel2 : int
-puntMaxNivel3 : int
+Jugador(Pnombre : String)
+getNombre() : String
+getPuntMax(pNivel : int) : int
+setPuntMax(pNivel : int, pPunt : int)
+imprimirInformacion() : String

3.2 Diagramas de Secuencia

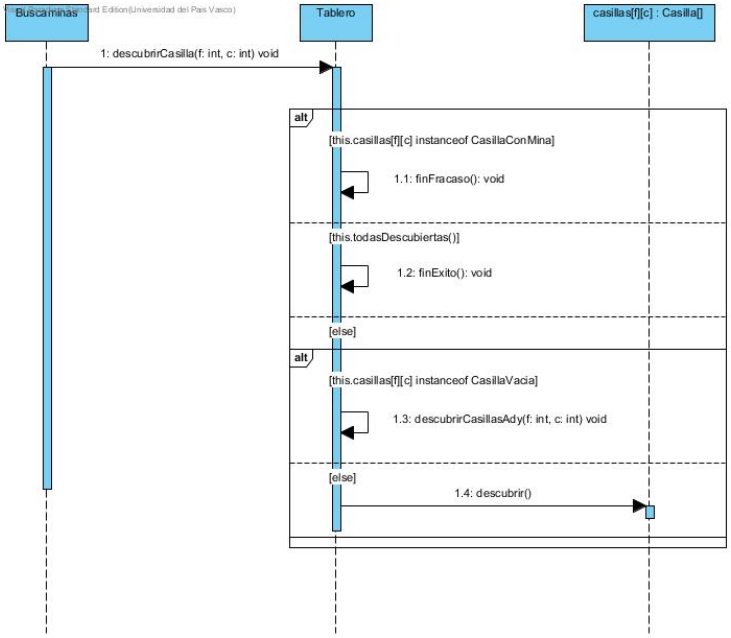
3.2.1 construirTableroPorNivel

El cuál sigue un patrón builder en la construcción de diferentes tableros que se diferencian por sus dimensiones.



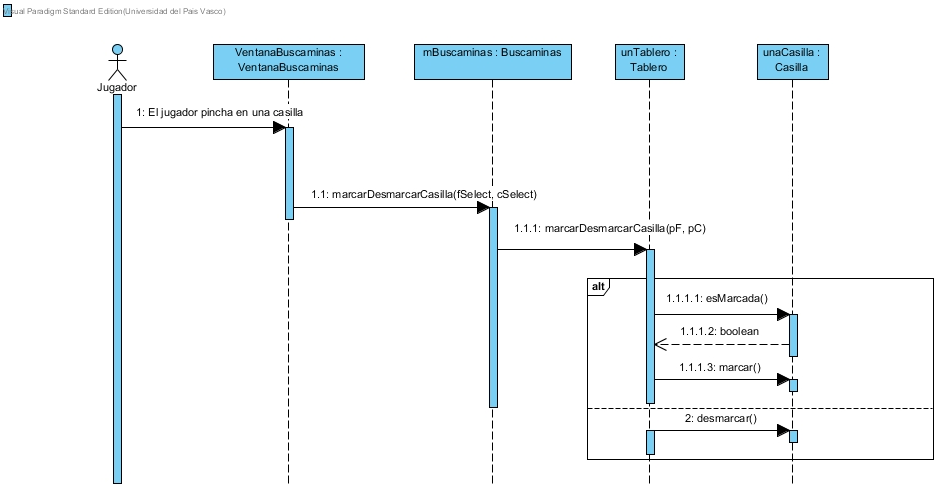
3.2.2 descubrirCasilla

En caso de que al descubrir resulta que todas las casillas sin minas están descubiertas, se ganan la partida. Y si es una mina, se pierde.



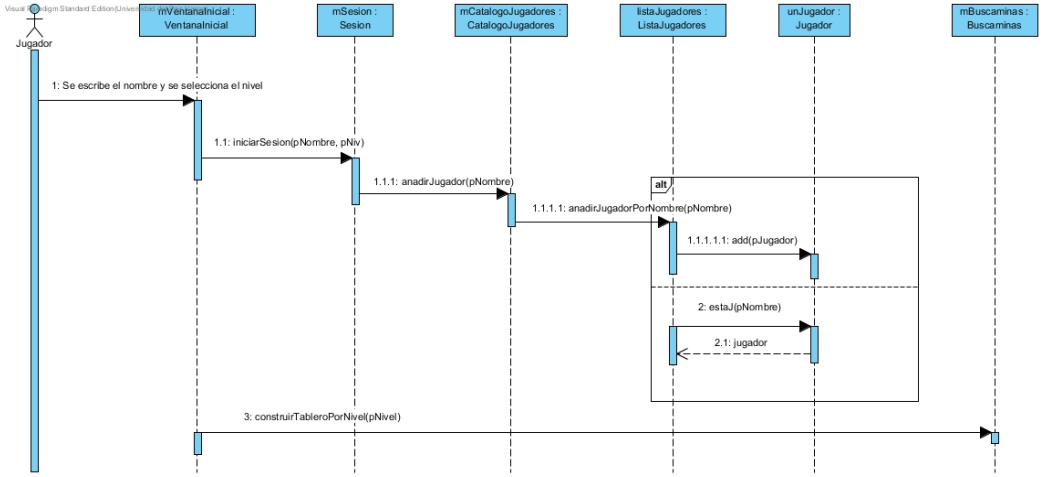
3.2.3 marcarDesmarcarCasilla

Si la casilla esta sin marcar la marca y viceversa, poniendo una bandera en el caso en que lo esté.



3.2.4 iniciarSesion

Comienza escribiendo el usuario su nombre y el nivel deseado y se añade el jugador al CatalogoJugadores en caso de que no exista (si existe devuelve el jugador) y creando un tablero del nivel indicado



4 Desarrollo

4.0.1 Primer Sprint

En el primer sprint nos centramos en las historias de usuario 2 y 3: Generar Buscaminas del nivel seleccionado y descubrir Casilla. Las principales clases que identificamos fueron dirigidas al patrón builder junto al factory de manera razonada. Nos dimos cuenta que estos patrones de una manera lógica eran los que más se ajustaban a nuestra resolución.

Hemos elegido el builder porque nos permite en un futuro añadir nuevas funcionalidades a los objetos de forma más fácil, rápida y modular. El Factory controla la creación de casilla facilitando como en el caso anterior su reutilización.

Los principales errores que cometimos en este primer sprint fueron no poner como tarea la realización de la interfaz gráfica, y olvidarnos de rellenar las actas de reunión y la documentación escrita, por lo demás se hizo un buen trabajo en general.

4.0.2 Segundo Sprint

En este segundo Sprint fue dirigido a las historias de usuario 1 y 4: Iniciar sesión con un nivel de dificultad y marcar/desmarcar Mina. Se enfocó sobretodo en la interfaz gráfica de los distintos objetos. Utilizando el patrón observer para separar la interfaz del modelo. El tablero pasaría a ser un Observable y la ventanaBuscaminas un Observer.

También hemos diseñado e implementado la opción de marcar/desmarcar una mina (que se manifestará en la interfaz con la imagen de una bandera) y la clase Sesión con patrón Singletón que contendrá el jugador de la partida, su puntuación y el nivel.

Además, rellanos una documentación de pruebas en la que informamos de lo que vamos probando y los errores surgidos en el proceso junto con las JUnits y las actas de reunión.

4.0.3 Tercer Sprint

Se enfocó en las historias de usuario restantes: Abortar Buscaminas, seguir jugando y mostrar el Ranking. Hemos decidido hacer tres rankings diferentes dependiendo del nivel de dificultad. El sistema de puntuación sigue una resolución en la cual, cuantos menos segundos, más puntos se obtiene. Hemos implementado el método obtenerMejoresJugadores en el que le decimos el número de mejores jugadores (con más puntos) que queremos obtener (en este caso 10). Para ello ordenamos la lista de jugadores utilizando una de los novedades de Java 8, el Comparator y utilizando para ello una clase anónima que nos facilita la implementación.

Los jugadores y sus puntuaciones se guardaran en un archivo de texto para que permanezcan una vez acabada la partida. Este archivo se cargará cada vez que se inicie el programa.

Además, hemos permitido que el usuario pueda abortar la partida en todo momento y ver el ranking. Al perder o ganar la partida, se nos permitirá seguir jugando, abortar el juego o ir a la pantalla de inicio.

5 Conclusiones

Los patrones hacen nuestro programa más reusable, pero es verdad que en ocasiones hace de la tarea de implementar algo más complicado.

El modelo MVC tiene muchas ventajas ya que hace que la implementación esté separada en partes gracias al patrón Observer y el mantenimiento es mucho más sencillo al tener una buena organización. Si quitamos la interfaz gráfica, no repercutiría en el modelo y se podría poner otra interfaz sin modificar lo que ya está implementado.

Nos hemos adaptado rápidamente a la metodología SCRUM. Aunque somos conscientes que al contar nuestro grupo con solo cuatro personas y no tratarse de un proyecto profesional real, dista bastante de la metodología REAL del SCRUM.

Hemos visto las ventajas de usar Github, aunque esta vez no hemos sacado del todo su rendimiento al no tener suficiente tiempo.

Nos ha resultado un poco engorroso el separar el controlador de al interfaz, pero lo vemos más organizado.

Nos parece mejor una puntuación basada en segundos (cuantos menos segundos, más arriba en el ranking).

Hemos apreciado las ventajas de JAVA 8 con el uso de Comparator