Títol:	*”Un joc d’Scrabble”*
Volum:	*1/1*
Alumne:	*Rafael Olivera Cano*
Director/Ponent:	*Marc Vigo Anglada*
Departament:	*LSI – Llenguatges i Sistemes Informàtics*
Data:	

DADES DEL PROJECTE

Títol del Projecte: ***"Un joc d'Scrabble"***

Nom de l'estudiant: ***Rafael Olivera Cano***
Titulació: ***Enginyeria Informàtica***
Crèdits: ***37,5***
Director/Ponent: ***Marc Vigo Anglada***
Departament: ***LSI – Llenguatges i Sistemes Informàtics***

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: ***Jesus Alonso Alonso***

Vocal: ***Josep Carmona Vargas***

Secretari: ***Marc Vigo Anglada***

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

“*Vuestro honor no lo constituirá vuestro origen, sino vuestro fin.*”

- NIETZSCHE, Friedrich -

“*Obrar es fácil, pensar es difícil; obrar según se piensa es aún más difícil.*”

- GOETHE, Johann Wolfgang von -

AGRADECIMIENTOS

Creo indispensable empezar este proyecto agradeciendo a todas aquellas personas que de una manera u otra, me han ayudado disfrutar la vida y me han apoyado en la elaboración de este proyecto en todo momento. Quisiera por lo tanto darle las gracias a:

mi madre
por enseñarme el significado de la vida, y a vivirla,

mi padre
por enseñarme a seguir adelante frente a cualquier circunstancia,

mi hermana
por enseñarme a fijar una meta a conseguir,

Sílvia
por enseñarme a querer,

mis amigos
por brindarme su amistad,

Marc Vigo
por darme la oportunidad de realizar este proyecto y su apoyo.

ÍNDICE DEL CONTENIDO

1. INTRODUCCIÓN 16

1.1. Sinopsis 18
1.2. Historia del Scrabble 19
1.3. Organización 22

1.3.1. Organización de la memoria 22
1.3.2. Organización del CD 23

1.4. Objetivos 24
1.5. Justificación 24
1.6. Alternativas al proyecto 26
1.7. Planificación inicial 27

1.7.1. Determinación de tareas 27
1.7.2. Planificación 28

1.8. Terminología del *Scrabble* 30

2. ANÁLISIS DE REQUERIMIENTOS 34

2.1. Introducción 36

2.1.1. Objetivo del documento 36
2.1.2. Ámbito del documento 36
2.1.3. Descripción general del documento 37

2.2. Descripción general 38

2.2.1. Perspectiva de la aplicación 38
2.2.3. Características de los usuarios 39
2.2.4. Restricciones del sistema 40

2.3. Requerimientos específicos 40

2.3.1. Requerimientos de las interfaces externas 40
2.3.2. Requerimientos funcionales 40
2.3.3. Requerimientos de rendimiento 42
2.3.4. Restricciones de diseño 42
2.3.5. Atributos del sistema software 42

3. TECNOLOGÍAS UTILIZADAS 45

3.1. Lenguaje de programación 47

3.1.1. Java 47
3.1.2. Justificación 51

3.2. Visualización 3D 52

3.2.1. OpenGL 53
3.2.2. JOGL 56
3.2.3. Justificación 57

3.3. Persistencia de los datos 57

3.3.1. Ficheros de texto 58
3.3.2. Ficheros binarios 59
3.3.3. Justificación 59

3.4. Herramientas 61

3.4.1. Eclipse 61

4. ESPECIFICACIÓN 64

4.1. Introducción 66
4.2. Diagrama de clases 67
4.3. Casos de uso 68
4.4. Diagramas de casos de uso 70

4.4.1. Diagrama: Gestión de jugadores 71
4.4.2. Diagrama: Gestión de diccionarios 71
4.4.3. Diagrama: Gestión de idiomas 72
4.4.4. Diagrama: Gestión de reglas 72
4.4.5. Diagrama: Gestión de estadísticas 72
4.4.6. Diagrama: Gestión de partidas 73

4.5. Especificación de los casos de uso 75

4.5.1. Especificación: Gestión de jugadores 75

4.5.1.1. Crear jugador 75
4.5.1.2. Borrar jugador 76
4.5.1.3. Ver detalles jugador 76

4.5.2. Especificación: Gestión de diccionarios 77

4.5.2.1. Definir diccionario por omisión 77
4.5.2.2. Añadir palabra 78
4.5.2.3. Eliminar palabra 78

4.5.3. Especificación: Gestión de idiomas 79

4.5.3.1. Seleccionar idioma 79
4.5.3.2. Definir idioma por omisión 80

4.5.4. Especificación: Gestión de reglas 80

4.5.4.1. Modificar reglas 81

4.5.5. Especificación: Gestión de estadísticas 81

4.5.5.1. Ver ranking 81

4.5.6. Especificación: Gestión de partidas 82

4.5.6.1. Cargar partida 82
4.5.6.2. Salvar partida 83
4.5.6.3. Seleccionar jugador 83
4.5.6.4. Deseleccionar jugador 84
4.5.6.5. Seleccionar diccionario 84
4.5.6.6. Remover fichas 85
4.5.6.7. Cambiar fichas 85
4.5.6.8. Deshacer jugada 86
4.5.6.9. Pasar turno 87
4.5.6.10. Realizar jugada (colocar ficha) 87
4.5.6.11. Realizar jugada (quitar ficha) 88
4.5.6.12. Realizar jugada (mover ficha) 89
4.5.6.13. Validar jugada 89
4.5.6.14. Reciclar comodines 90
4.5.6.15. Fer senyors 91
4.5.6.16. Siguiente turno 91
4.5.6.17. Ver log 92
4.5.6.18. Salvar log 92

4.6. Modelo de comportamiento 93

4.6.1. Diagramas de secuencia 93

4.6.1.1. Diagramas de secuencia: Gestión de jugadores 94
4.6.1.2. Diagramas de secuencia: Gestión de diccionarios 95
4.6.1.3. Diagramas de secuencia: Gestión de idiomas 96
4.6.1.4. Diagramas de secuencia: Gestión de reglas 96
4.6.1.5. Diagramas de secuencia: Gestión de estadísticas 96
4.6.1.6. Diagramas de secuencia: Gestión de partidas 97

4.6.2. Contratos de las operaciones 100

4.6.2.1. Contratos: Gestión de jugadores 100
4.6.2.2. Contratos: Gestión de diccionarios 101
4.6.2.3. Contratos: Gestión de idiomas 102
4.6.2.4. Contratos: Gestión de reglas 103
4.6.2.5. Contratos: Gestión de estadísticas 103
4.6.2.6. Contratos: Gestión de partidas 104

4.7. Modelo de estados 110

4.7.1. Diagrama de estado de clases 110

4.7.1.1. Diagrama: Ficha 111
4.7.1.2. Diagrama: Casilla 111
4.7.1.3. Diagrama: Jugadores 111

4.7.2. Diagrama de estado de casos de uso 112

4.7.2.1. Diagrama: Alta genérica 112
4.7.2.2. Diagrama: Borrar genérico 112

4.7.2.3. Diagrama: Ver genérico 112
4.7.2.4. Diagrama: Modificar genérico 113
4.7.2.5. Diagramas principales 113

5. DISEÑO 117

5.1. Introducción 119
5.2. Diseño de la arquitectura 119

5.2.1. Patrones arquitectónicos 121
5.2.2. Arquitectura escogida 121

5.2.2.1. Patrón por capas 122
5.2.2.2. Orientación a objetos 123

5.3. Capa de presentación 125

5.3.1. Diagrama de la capa de presentación 126
5.3.2. Diseño de las pantallas 127
5.3.3. Componentes 131

5.4. Capa de datos 132

5.4.1. Descripción de los ficheros 133
5.4.2. Diagrama de la capa de datos 135

5.5. Capa de dominio 135

5.5.1. Normalización del diagrama de clases 136
5.5.2. Patrón de diseño: Controlador 137
5.5.3. Descripción de los controladores 138

5.5.3.1. Controlador jugadores 139
5.5.3.2. Controlador diccionarios 139
5.5.3.3. Controlador reglas 140
5.5.3.4. Controlador saco 140
5.5.3.5. Controlador tablero 141
5.5.3.6. Controlador estrategia 141
5.5.3.7. Controlador partida 141

5.5.4. Diagrama de la capa de dominio 142
5.5.5. Diagramas de secuencia 144

5.5.5.1. Diagrama: crearJugador 144
5.5.5.2. Diagrama: borrarJugador 144
5.5.5.3. Diagrama: verJugador 145
5.5.5.4. Diagrama: colocarFicha 145
5.5.5.5. Diagrama: realizarSiguienteTurno 146
5.5.5.6. Diagrama: realizarJugada 147
5.5.5.7. Diagrama: getSolucion 148
5.5.5.8. Diagrama: reponerFichasJugador 148

6. IMPLEMENTACIÓN 150

6.1. Introducción 152
6.2. Capa de presentación 152

6.2.1. Implementación menús 153
6.2.2. Implementación de las pantallas 155
6.2.3. Implementación del tablero 156

6.2.3.1. Inicialización del tablero 157
6.2.3.2. Display 158

6.3. Capa de datos 160

6.3.1. Acceso ficheros de texto 160
6.3.2. Acceso ficheros binarios 162
6.3.3. Directorio de ficheros 162

6.4. Capa de dominio 163

6.4.1. Diccionario 163

6.4.1.1. Primer estudio realizado 166
6.4.1.2. Segundo estudio realizado 168
6.4.1.3. Nodo 175

6.4.2. Estrategia 176

6.5. Pruebas del sistema 177

6.5.1. Pruebas unitarias 179
6.5.2. Pruebas de integración 180
6.5.3. Pruebas del sistema 180

7. ESTUDIO TEMPORAL Y ECONÓMICO 182

7.1. Planificación inicial 184
7.2. Proceso de desarrollo del proyecto 186
7.3. Estudio temporal real 187
7.4. Estudio económico 187

7.4.1. Costes hardware 188
7.4.2. Costes software 188
7.4.3. Costes recursos humanos 189
7.4.4. Costes totales 190

8. CONCLUSIONES 192

8.1. Objetivos alcanzados 194
8.2. Aportación personal 194
8.3. Desarrollo futuro 195

9. BIBLIOGRAFÍA 197

9.1. Bibliografía 199

9.1.1. Libros 199
9.1.2. Referencias 199
9.1.3. Proyectos 200

i. ANEXO 1. MANUAL DE INSTALACIÓN 202

i.1. Introducción 204
i.2. JRE 204
i.3. Librerías JOGL (Windows) 205
i.4. Librerías JOGL (Linux) 206

ii. ANEXO 2. INSTRUCCIONES DE JUEGO 208

ii.1. Scrabble 210
ii.2. Contenido 210
ii.3. Mecánica del juego 210
ii.4. Reglas de juego 210
ii.5. Explicación de reglas problemáticas 214

iii. ANEXO 3. MANUAL DE USUARIO 217

iii.1. Introducción 219
iii.2. Jugadores 220
iii.3. Diccionario 223
iii.4. Reglas 224
iii.5. Partida 225

iii.5.1. Turno jugador robot 226
iii.5.2. Turno jugador humano 227

iii.5.2.1. Acciones 228
iii.5.2.2. Cambiar fichas 230
iii.5.2.3. Validar jugada 231

iii.6. Ranking 233
iii.7. Idioma de localización 234
iii.8. Ayuda 235
iii.9. Acerca de... 236

1. INTRODUCCIÓN

1.1. Sinopsis 18
1.2. Historia del Scrabble 19
1.3. Organización 22

1.3.1. Organización de la memoria 22
1.3.2. Organización del CD 23

1.4. Objetivos 24
1.5. Justificación (motivación) 24
1.6. Alternativas al producto 26
1.7. Planificación inicial 27

1.7.1. Determinación de tareas 27
1.7.2. Planificación 28

1.8. Terminología del *Scrabble* 30

1.1. SINOPSIS

Un Joc d'Scrabble es una aplicación que pretende proporcionar la posibilidad de poder realizar partidas a través del ordenador al famoso y extendido *Scrabble*. Juego originalmente de sobremesa en el que diversos jugadores intentan ganar la partida formando palabras cruzadas sobre el tablero.

Está orientado principalmente a aquellos usuarios amantes del *Scrabble*, pero también a cualquier otro usuario con menos experiencia en el juego. Gracias a sus diferentes niveles de dificultad, se podrán conseguir partidas realmente equilibradas según la experiencia de cada jugador.

Para una mejor visualización y como atractivo visual, se ha construido el tablero de juego en un modelo 3D.

También incorporará un modulo de diccionarios a través del cual se cargarán los listados (externos a la aplicación) de palabras válidas al inicio de cada partida. Los diccionarios se podrán modificar directamente sobre los respectivos ficheros que contienen los listados de palabras, así como durante el transcurso de una partida, dando la posibilidad de añadir nuevas palabras dinámicamente enriqueciendo partida a partida el listado de palabras aceptadas por los diferentes idiomas.

Para la implementación de la aplicación se ha utilizado *Java*, ya que hoy en día es uno de los lenguajes de programación más potentes y que ofrecen una mayor portabilidad. Los usuarios de esta aplicación no necesitarán más que la *Java Runtime Environtment (JRE)* para poder ejecutar la aplicación, la cual es proporcionada en el CD adjunto a esta memoria, así como las librerías específicas para la visualización 3D. Dichas librerías son las desarrolladas en el proyecto *JOGL*, de código abierto, en las que se consigue acceder a la *API* de *OpenGL* mediante aplicaciones *Java*, proporcionadas también en el CD adjunto, y ya configuradas dentro de la *JRE*. Este proyecto se ha realizado mediante la suite de desarrollo *Eclipse*, una de las suites de desarrollo de código abierto más importantes ya que ofrece diversos *Frameworks* y extensiones que facilitan el desarrollo de nuevas aplicaciones.

1.2. HISTORIA DEL SCRABBLE

- **El arquitecto del juego.** En 1931, la población de Poughkeepsie, en Nueva York, se encontraba sumida en una situación similar a la del resto de Estados Unidos... hundida en lo más profundo de la Gran Depresión. Había una gran inestabilidad laboral y los estafadores estaban a la orden del día. Fue en estas circunstancias cuando el arquitecto Alfred Mosher Butts, residente en Poughkeepsie, perdió su trabajo y decidió explorar su pasión por los juegos y las palabras. A Butts, hombre tranquilo y apacible, no le gustaban los juegos con dados, porque en ellos todo dependía de la suerte. Pero por el otro lado, opinaba que los juegos que requerían muchas habilidades -como el ajedrez- resultaban demasiado intelectuales para el público en general.

- **Mitad suerte, mitad habilidad.** Así que decidió inventar un juego en el que intervinieran una mitad de suerte y otra mitad de habilidad, y a finales de 1931 ya había dado con la idea inicial del actual Scrabble, un juego al que denominó Lexico. Lexico se jugaba sin tablero de juego, y los jugadores anotaban puntos en base a la longitud de las palabras que iban formando. Se obtenían puntos adicionales por utilizar letras poco comunes (B, F, H, M, P, V, W, Y) y un mayor número de puntos adicionales por utilizar las letras más inusuales (J, K, Q, X, Z). La manera como Butts calculó la frecuencia de aparición de las letras en su lengua y el correspondiente valor de cada una de ellas fue mediante un riguroso análisis de la portada del New York Times. Llegó a la conclusión de que el juego resultaba muy fácil si se incluían muchas "S", así que redujo el número a 4.

- **El fracaso del juego.** En 1933 Butts solicitó una patente para su juego Lexico, que le fue denegada. Del mismo modo, cuando lo presentó a dos fabricantes de juegos, Parker Brothers y Milton Bradley, sólo recibió una cortés negativa. Butts no tiró la toalla, y en los siguientes 5 años fabricó él mismo casi 200 juegos que regaló o vendió a sus amigos, pero Lexico no llegó más lejos. No era un éxito comercial.

- **Los crucigramas entran en juego**. Corría el año 1938 cuando Butts introdujo un elemento verdaderamente innovador en su juego. La gran popularidad de los crucigramas le proporcionó la brillante idea de combinar las letras sobre un tablero de juego, formando palabras del mismo modo que en los crucigramas. Lexico cambió de nombre varias veces, llamándose sucesivamente New Anagrams, Alph, Criss-Cross y Criss-Crosswords. Butts dibujó él mismo con sus instrumentos de arquitectura los primeros tableros de juego para Criss-Crosswords, imprimiéndolos y pegándolos en

tableros de ajedrez. Asimismo, dibujó las letras de las fichas a mano, pegándolas en trozos de madera de 0,5 cm., cortados a la medida de las casillas del tablero. A lo largo de los años siguientes, la mecánica del juego cambió paulatinamente. Por ejemplo, en un momento dado, la primera palabra del juego debía colocarse en la esquina superior izquierda del tablero. Sin embargo, muchas de las características del juego original de Butts se han conservado hasta nuestros días, por ejemplo, el tablero de 15 x 15 casillas o el atril de 7 fichas, así como el número de fichas de cada letra y los valores de las mismas, que se han mantenido igual desde 1938 hasta hoy. Pero Criss-Crosswords obtuvo el mismo rechazo que Lexico. De nuevo le fue denegada la patente para el juego y varios fabricantes rechazaron su propuesta. ¡Una desafortunada decisión de la que unos años después a buen seguro se arrepentirían!

- **Con un poco de ayuda de sus amigos.** Durante un tiempo, Butts consideró la idea de fabricar y comercializar el juego con sus propios medios, pero pronto se dio cuenta de que no tenía espíritu de empresario y volvió a su trabajo como arquitecto. Con la llegada de la Segunda Guerra Mundial, Butts abandonó el desarrollo del juego hasta 1948. Y en ese momento se produjo un hito en la historia del juego, protagonizado por James Brunot, amigo de Butts y uno de los primeros en recibir uno de los juegos Criss-Crosswords que Butts había fabricado con sus propias manos. James Brunot y su esposa creían firmemente en el juego y decidieron arriesgarse a comercializarlo. James Brunot no sólo creía en el juego, sino que disponía del tiempo y del espíritu empresarial necesarios para embarcarse en esta aventura comercial. Así que Butts y Brunot llegaron a un acuerdo: a cambio de dar la autorización a Brunot para fabricar el juego, Butts recibiría un royalty por cada juego vendido.

- **16.12.48 El día "D".** Los Brunot decidieron que el juego necesitaba unas pequeñas modificaciones, por lo que reorganizaron la distribución de casillas con premio y simplificaron las reglas, que resultaban demasiado largas y complejas. Al mismo tiempo, empezaron a pensar en un nuevo nombre y solicitaron un "copyright" (derechos de autor), lo que se les concedió el 1 de Diciembre de 1948. Pocos días después llegó el gran día... tras largas consideraciones, decidieron llamar Scrabble® al juego y consiguieron registrarlo el 16 de Diciembre de 1948. ¡Por fin había nacido el Scrabble® que todos conocemos!

- **Scrabble va a al colegio.** Los Brunot empezaron vendiendo el juego en el salón de su casa de Newtown, en Connecticut. Brunot compraba las piezas para fabricar el juego y las montaba con ayuda de su esposa. Al principio, sólo fabricaban 18 unidades diarias,

debido al trabajoso proceso de estampar las letras en las fichas de madera una por una. Durante 1949, el primer año de producción del juego por parte de los Brunot, montaron y vendieron 2251 juegos, perdiendo 450$. En los años siguientes, continuaron luchando y trabajando duro para comercializar el juego. En 1952 seguían perdiendo dinero, por lo que empezaron a pensar en abandonar el proyecto y Brunot se tomó unas vacaciones para considerar el futuro de su empresa. Al volver, se encontró con una agradable sorpresa: una avalancha de pedidos, debido a que los que lo habían adquirido lo recomendaban con entusiasmo a sus amigos y conocidos. Había llegado el momento de trasladarse a un local más grande, así que se mudaron a un colegio abandonado cercano a su casa de Connecticut. En el último trimestre de 1952 se vendieron 37.000 unidades de Scrabble.

- **¿Qué quiere decir que no lo vendemos?** Fue también en el año 1952 cuando Jack Strauss, el presidente de Macy's de Nueva York -los grandes almacenes más importantes del mundo-, aprendió a jugar a Scrabble durante sus vacaciones. El juego le entusiasmó de tal manera que al volver a Nueva York pidió a su Departamento de Juegos que le mandaran unas cuantas unidades. No existen testimonios de cómo sucedió todo, pero el hecho es que su Departamento de Juegos tuvo que confesar que no vendían Scrabble... ¡hasta ese momento! Macy's no sólo empezó a vender Scrabble, sino que apoyó una campaña de promoción del juego que cautivó a miles de personas. En 1953, aunque la cifra de producción había aumentado a 6.000 unidades por semana, Brunot se dio cuenta de que no podía satisfacer la extraordinaria demanda de Scrabble, así que cedió la licencia de fabricación a Selchow and Righter, el fabricante de juegos líder en Estados Unidos, que precisamente lo había rechazado unos años antes.

- **Scrabble por fin pide la palabra.** Durante tres años, se tuvo que racionar el servicio de Scrabble a las tiendas, ya que la demanda no dejaba de aumentar y el fabricante no podía satisfacerla. Mientras, la fiebre de Scrabble se extendió hasta Australia en 1953. Ese mismo año, J. W. Spear & Sons lanzaron el juego en Gran Bretaña, donde se convirtió en un éxito de ventas inmediato. Brunot acabó vendiendo los derechos de Scrabble en 1968 a Spear's para todo el mundo excepto para los Estados Unidos, Canadá y Australia (unos años más tarde consiguieron también los derechos para este país). Los derechos quedaron repartidos de esta manera hasta la actualidad. En 1986 Selchow and Righter vendieron los derechos a Coleco, que quebró en 1987. Y así sucedió que, 53 años después de rechazar el juego, Milton Bradley adquirió los derechos de Scrabble para los Estados Unidos y Canadá. En 1991 tuvo lugar el primer campeonato mundial de Scrabble en Londres, y el segundo se organizó en Nueva York

en 1993. Desgraciadamente, James Brunot murió en Octubre de 1984, por lo que nunca llegó a ver el primer campeonato mundial del juego, pero Alfred Butts sí tuvo la suerte de verlo. Vivió hasta los 93 años, muriendo en Abril de 1993. Butts siguió jugando a Scrabble con su familia y sus amigos hasta el final de sus días. Este hombre modesto y sin pretensiones vivió para ver cómo su creación se convertía en un fenómeno mundial. En 1994 la compañía J. W. Spear and Sons fue adquirida por Mattel, Inc., uno de los mayores fabricantes de juguetes del mundo.

1.3. ORGANIZACIÓN

A continuación, se explica la línea que se ha seguido en la estructuración de este documento, así como del CD adjunto al mismo con el fin de facilitar y acomodar la lectura y compresión.

1.3.1. ORGANIZACIÓN DE LA MEMORIA

El documento se organiza en los siguientes capítulos:

1. **Introducción.** Visión general del proyecto (presentación, motivaciones, objetivos,…). Imprescindible para adquirir una idea general del proyecto, y para más adelante poder entrar en más detalle en todos los aspectos del mismo.

2. **Análisis de requerimientos.** Este capítulo estudia el problema expuesto en la introducción, con el objetivo de definir exactamente las necesidades a las que el sistema debería de dar soporte.

3. **Tecnologías utilizadas.** En este capítulo se hará una breve explicación de todas aquellas tecnologías que son necesarias para el proyecto y su funcionamiento, para así conseguir una mejor comprensión de que como funciona el sistema.

4. **Especificación.** La especificación constituye el primer paso en la construcción del programa. Este capítulo recoge los diagramas y los documentos más importantes de esta etapa del desarrollo.

5. **Diseño.** Decisiones que se han tomado sobre cómo se tiene que implementar el sistema.

6. **Implementación.** Se muestran los métodos que se han seguido para la implementación del sistema, así como las pruebas que se han realizado para verificar el buen funcionamiento de éste.

7. **Estudio temporal y económico.** Coste económico del sistema y tiempo que se ha invertido en el desarrollo de éste.

8. **Conclusiones.** Conocimientos adquiridos en el desarrollo del proyecto y líneas de desarrollo futuras.

9. **Bibliografía.**

i. **Anexo 1. Manual de instalación y configuración.**
ii. **Anexo 2. Instrucciones de juego.**
iii. **Anexo 3. Manual de usuario**

1.3.2. ORGANIZACIÓN DEL CD

En el CD adjunto en esta memoria encontramos:

- **El proyecto (Scrabble).** Podemos encontrar el conjunto de carpetas que forman la aplicación, en las cuales la carpeta src, es el código fuente de la misma. En el directorio raíz del proyecto, podemos encontrar adicionalmente un ejecutable .bat, a través del cual ejecutar la aplicación.

- **Requerimientos software.** Contiene el software necesario para el funcionamiento de la aplicación, además de las librerías específicas para la visualización 3D del tablero tanto para sistemas operativos Windows, como para sistema operativos Linux.

- **Manual de instalación y configuración.** Manual donde se explica como instalar el software necesario para el funcionamiento de la aplicación, así como, la misma aplicación. Es un extracto del segundo anexo de la memoria.

- **Instrucciones de juego.** Explicación en detalle de cómo jugar a Scrabble. Extraído de la memoria.

- **Manual de Usuario.** Manual de usuario donde se explica como llevar a cabo paso a paso todas las funcionalidades de la aplicación. Extraído de la memoria.

- **Memoria.** Versión digital de este documento.

1.4. OBJETIVOS

Un joc d'Scrabble debe permitir jugar una partida al juego conocido como *Scrabble* entre 2 y 4 jugadores. Estos jugadores pueden ser de tipo *Humano* o *Robots*. Los jugadores tipo *Robot* podrán configurarse con diferentes tipos de dificultad. Adicionalmente debe ofrecer la posibilidad de jugar un único jugador a modo de entrenamiento para la capacidad de resolver jugadas.

Los principales objetivos del proyecto son:

- **Gestión de partidas**
- **Gestión de jugadores**
- **Gestión de diccionarios**
- **Gestión de idiomas**
- **Gestión de reglas (variaciones de juego)**
- **Gestión de estadísticas (rankings)**

Nos limitaremos únicamente a listar los principales objetivos, ya que todos estos serán completamente detallados en el siguiente capítulo: *Análisis de requerimientos.*

1.5. JUSTIFICACIÓN

El proyecto final de carrera ofrece la posibilidad de consolidar mi formación como ingeniero y poner en práctica todos los conceptos adquiridos durante estos últimos cinco años. Su realización supone la dedicación de una gran cantidad de horas y de esfuerzos. Es por eso que la elección del proyecto a realizar ha de ser fruto de una decisión meditada y convencida. En este sentido, resulta fundamental saber cuales son los requisitos que debe cumplir el proyecto que se desea realizar y las posibles aportaciones personales al mismo. Desde un principio, sabía que la elección del proyecto tenía que satisfacer plenamente mis aspiraciones personales, y a la vez que resultará "divertido". Además quería que el proyecto pusiera en

práctica los máximos conceptos aprendidos durante la carrera, y que no fuera puramente un trámite. Otra motivación que tenía, era sentir que el proyecto que realizara fuera "algo" servible, es decir, que pudiera servir realmente en la vida cotidiana.

A parte de la principal motivación, claro, que es el hecho de acabar la carrera y obtener el título de ingeniero, existen otras motivaciones adicionales que me gustaría dejar constancia:

- Realizar un proyecto informático de programación desde el principio, con decisiones de diseño propias, con un conjunto amplio de funcionalidades diferentes y con una interfaz gráfica encarada principalmente a los usuarios.

- Utilización del lenguaje de programación *Java*, que entre los muchos lenguajes vistos a lo largo de la carrera, es con diferencia el que más me ha gustado.

- Utilización de librerías gráficas 3D, una de las partes más interesantes que he encontrado a lo largo de la carrera.

- La optimización en la búsqueda de soluciones a un problema concreto en el que la extensa cantidad de información a tratar hace que resulte un verdadero reto en costes de tiempo.

- Así como, el desarrollo de una inteligencia artificial con la que podemos dotar a nuestras aplicaciones para poderlas hacer interactuar con los usuarios, como ha sido en este caso, jugando una divertida partida de *Scrabble*.

- Conocer el mundo de los videojuegos, aún no teniéndose que tratar de ninguno con impresionantes gráficos, o grandes dosis de acción, sino con un gran alto nivel intelectual; rapidez y capacidad mental para obtener soluciones óptimas que nos lleven a la victoria.

- Realizar una aplicación que se utilice y sea útil al mismo tiempo, no como la mayoría de las prácticas que he ido realizando hasta ahora y que han ido quedando en el olvido.

Estas motivaciones y muchas otras han ido surgiendo durante el desarrollo del proyecto, y han hecho que apostara al 100% por esta magnífica idea desde el primer momento.

1.6. ALTERNATIVAS AL PROYECTO

Existen en el mercado diversos productos relacionados con Scrabble, por lo que antes de que este proyecto tuviera lugar ya existían alternativas a tener en cuenta. Aunque algunas de ellas cumplen bastantes de los requerimientos deseados, no los llegan a cumplir en su totalidad. Por lo que el desarrollo este proyecto consigue ofrecer todos aquellos aspectos positivos de ciertos productos reunidos en un único software.

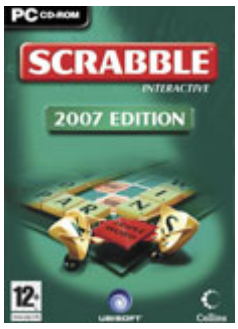
La principal carencia de este proyecto es no ofrecer la posibilidad de jugar partidas online con varios jugadores, ya sean de tipo *humano* o *robot*. Aunque como se definirá en el apartado de *Desarrollo futuro* en el bloque de *Conclusiones* se contempla la posibilidad de incluir dicha modalidad.

Por el contrario, considero que este proyecto destaca en los siguientes puntos:

- La posibilidad de incluir tantos diccionarios como el usuario desee.
- La posibilidad de actualizar en todo momento cualquier diccionario disponible. Dicha actualización puede realizar en el mismo transcurso de una partida, pasando la palabra a formar parte del diccionario desde ese mismo momento, y de esta manera pasando a ser válida.
- La obtención del software es totalmente gratuita ya que ha sido desarrollado bajo herramientas y tecnología de código abierto.

Algunos de los productos en el mercado son:

- **Scrabble 2007 Edition.**



  - Dispone de modalidad online.
  - Dispone de diccionario actualizable
  - **NO** dispone más idiomas disponibles que el propio.
  - La obtención del software es de pago (45.00 €)

- **Scrabble (Hasbro).**

  - **NO** dispone de modalidad online.
  - **NO** dispone de diccionario actualizable
  - **NO** dispone más idiomas disponibles que el propio.
  - Dispone de la definición de todas las palabras.
  - La obtención del software es de pago (19.90 €)

Aunque ambos son dos buenos productos, existen también muchas versiones Freeware de personas que han desarrollado en menor o mayor seriedad un software capaz de permitir jugar al Scrabble.

1.7. PLANIFICACIÓN INICIAL

A continuación se expone la planificación inicial que acordé en las diferentes reuniones de seguimiento con mi director de proyecto. Esta planificación abarca tanto el desarrollo de la aplicación, como la elaboración de la memoria y preparación de la presentación. Recordamos que ésta ha de cumplir los objetivos mencionados con anterioridad.

1.7.1. PLANIFICACIÓN DE TAREAS

Las principales tareas de que se compone este proyecto son las siguientes:

- **Definición del proyecto.** El primer paso para realizar el proyecto, es determinar todos los aspectos sin entrar en detalle, para poder conocer el alcance de éste. Es importante fijar los objetivos desde un inicio para evitarnos problemas.

- **Análisis de requerimientos.** Hemos de definir claramente cuales son las funcionalidades de la aplicación, como también los requisitos que ha de cumplir, como eficiencia, facilidad de utilización… Este análisis juntamente con la definición de los objetivos nos servirá para poder realizar una estimación temporal del proyecto.

- **Especificación.** En el momento que dispongamos de las diferentes funcionalidades definidas, pasaremos a especificar los casos de uso, para plasmar la interacción entre el usuario y el sistema.

- **Diseño.** Paso previo a la implementación, donde se indicará como se implementaran las diferentes funcionalidades.

- **Implementación.** Llevar a cabo la creación de código, siguiendo los pasos que se han indicado en los puntos anteriores.

- **Pruebas.** Ejecución de la aplicación para comprobar que todo funciona tal y como estaba previsto, siguiendo los objetivos marcados. En caso de encontrar problemas, se tendrán que estudiar y rectificarlos adecuadamente.

- **Memoria.** Crear el documento donde se expondrá todo el trabajo realizado para la elaboración del proyecto. Partes del documento ya se habrán realizado durante la elaboración del proyecto, con lo que sólo se tendrá que maquetar.

- **Preparación de la presentación.** Una vez finalizada la memoria y entregada a los miembros del tribunal, Presidente, Vocal y Tutor, se realizará la preparación de la exposición. Ésta incluirá la elaboración de unas transparencias y unos ejemplos de ejecución de la aplicación.

1.7.2. PLANIFICACIÓN

Para no encontrarnos con dificultades para llegar a la fecha de entrega del proyecto final de carrera, se realizó la siguiente planificación de las tareas anteriormente descritas. Con esta planificación no se ha pretendido establecer temporalizaciones rigurosas en la elaboración del proyecto sino disponer de una ayuda a la hora de estructurar las etapas de la realización del mismo.

Desde un primer momento, se planteó planificar el desarrollo del proyecto en un periodo equivalente a dos cuadrimestres. El motivo de la extensa planificación no es otro que la responsabilidad laboral que impide una plena dedicación diaria al desarrollo del proyecto. Entre las diferentes tareas planificadas se han contabilizando dos intervalos de un mes cada uno dedicado a las vacaciones (Agosto y Enero).

Como fecha de inicio consta el **01/06/2006**.

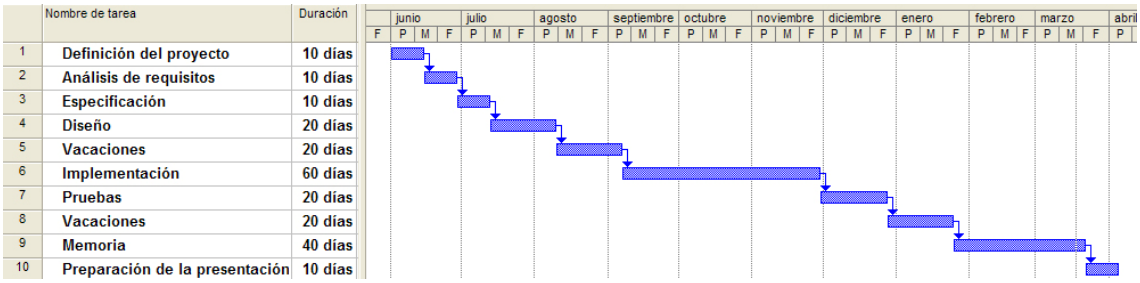


Figura 1.1. Diagrama de Gantt de la planificación inicial

Tarea	Duración (días)	Fecha inicio	Fecha fin
Definición del proyecto	10	01 / 06 / 06	14 / 06 / 06
Análisis de requisitos	10	15 / 06 / 06	28 / 06 / 06
Especificación	10	29 / 06 / 06	12 / 07 / 06
Diseño	20	13 / 07 / 06	09 / 08 / 06
Vacaciones	20	10 / 08 / 06	06 / 09 / 06
Implementación	60	07 / 09 / 06	29 / 11 / 06
Pruebas	20	30 / 11 / 06	27 / 12 / 06
Vacaciones	20	28 / 12 / 06	24 / 01 / 07
Memoria	40	25 / 01 / 07	21 / 03 / 07
Preparación de la presentación	10	22 / 03 / 07	04 / 04 / 07

Figura 1.2. Planificación inicial expresada en días

Tarea	Duración (horas)
Definición del proyecto	40
Análisis de requisitos	40
Especificación	40
Diseño	80
Implementación	240
Pruebas	80
Memoria	160
Preparación de la presentación	40
Total	**720**

Figura 1.3. Cómputo de horas de la planificación inicial

Para el cómputo total de horas dedicadas al proyecto en la planificación inicial, consideraremos una media aproximada de 4 horas por día contabilizado. Obteniendo así un total de 720 horas como planificación inicial.

1.8. TERMINOLOGÍA DEL SCRABBLE

Antes de continuar con el análisis de requisitos y entrar en materia, creo que es conveniente para el lector, explicar una serie de términos del *Scrabble* de los cuales algunos irán apareciendo a lo largo de este documento. De esta manera se pretende que el lector se situé en el ámbito del juego, sin que tenga que buscar otras fuentes más que esta sección del documento.

- **Abrir.** Colocar en el tablero una jugada que incrementa las posibilidades de los que juegan a continuación de poder colocar la suya.
- **Acabar.** Finalizar la partida por el hecho de gastar un jugador todas las fichas de su propio atril cuando ya no quedan más en el saco.
- **Anotación.** Procedimiento por levantar acta del desarrollo de una partida.
- **Anotación abreviada.** Anotación en la que únicamente constan los puntos obtenidos y acumulados por cada jugador.
- **Anotación estándar.** Anotación en la que se hace constancia cada mano con la jugada o el cero, y las coordenadas de la jugada, si hay, los puntos obtenidos en aquella mano con la jugada y el total acumulado.
- **Anotación integral.** Anotación que, adicionalmente a la información contenida en la modalidad estándar, hace constancia también del atril de cada jugador.
- **Arreglar el atril.** Hacer una quedada.
- **Atril.** Pieza con dos planos en ángulo recto, de base inclinada, donde descansan las fichas de un jugador. Por extensión, el conjunto de fichas de que un jugador dispone en un turno.
- **Benjamín.** Jugada que alarga la palabra prefijando el numero suficiente de fichas para cobrar un doble o triple de palabra.
- **Bicentenaria.** Jugada en la que se cobran más de 199 puntos.
- **Bonificación.** El hecho de cobrar más puntos que los de las fichas usadas. Puede ser por scrabble, o por casilla.
- **Bolsa.** Ver *saco*.
- **Casilla.** Cada una de las 225 celdas del tablero. Hay algunas que tienen un efecto especial:

  - **Azul celeste.** Duplica el valor de la ficha (L2).
  - **Azul oscuro.** Triplica el valor de la ficha (L3).
  - **Rosa.** Duplica el valor de la palabra (P2).
  - **Rojo.** Triplica el valor de la palabra (P3).

- **Cambiar.** Pasar para tener el derecho de sustituir algunas de las fichas del atril por otras del saco.
- **Centenaria.** Jugada en la que se cobran más de 99 puntos.
- **Cero.** Jugada nula en la que no se cobran puntos, por ejemplo al cambiar fichas.
- **Cerrar.** Colocar en el tablero una jugada que reduce las posibilidades de los que juegan a continuación de poder colocar la suya.
- **Cobrar.** Obtener puntos.

- **Coger.** Coger fichas del saco.
- **Coordenadas.** Letra y numero (si la palabra se sitúa en vertical) o número y letra (si se sitúa en horizontal) que indican la casilla correspondiente a la primera ficha de una jugada.
- **Comodín.** Ficha sin letra que puede valer por cualquiera de las otras. En el juego convencional no suma ningún punto.
- **Cuádricentenaria.** Jugada en la que se cobran más de 399 puntos.
- **Cuádruplo.** Jugada, generalmente pero no obligatoriamente scrabble, en la que se cobran dos P2.
- **Diccionario arbitral.** Aquel que sirve para determinar la validez de una jugada.
- **Ficha (letra) cara.** Cualquiera de las que valen 8 o 10 puntos.
- **Impugnar.** Solicitar comprobación de la validez de una jugada.
- **Nónuplo.** Jugada, generalmente pero no obligatoriamente scrabble, en la que se cobran dos P3.
- **Pared.** Jugada en que la palabra nueva se sitúa a lo largo de otra palabra provocando varias palabras perpendiculares a la palabra situada.
- **Partida.** Serie de turnos que forman un juego completo.
- **Partida de 2, 3, o 4.** Partida en la que intervienen el número de jugadores correspondiente.
- **Pasar.** Renunciar a hacer jugada en el propio turno.
- **Quedada.** Jugada en la que se usan únicamente una parte de las fichas del atril teniendo como objetivo no tanto lo que se cobre sino, principalmente, conservar fichas susceptibles de provocar un atril mas adecuado en el siguiente turno.
- **Saco.** El recipiente de tela que contiene las fichas todavía no repartidas ni jugadas.

- **Scrabble.** Nombre del juego. Jugada en la que se gastan las siete fichas del atril i que tiene una bonificación de 50 puntos.
- **Tablero.** Superficie del juego, subdividida en 225 unidades repartidas en 15 filas y 15 columnas, llamadas casillas; les files se identifican con un número y las columnas con una letra.
- **Tétrada.** Cobrar el mismo jugador cuatro scrabbles en una partida.
- **Triada.** Cobrar el mismo jugador tres scrabbles en una partida, por extensión cobrar tres o más.
- **Tricentenaria.** Jugada en la que se cobran más de 299 puntos.
- **Turno.** Ocasión de jugar, tanto si se hace jugada como si se pasa.

2. ANÁLISIS DE REQUERIMIENTOS

2.1. Introducción 36

2.1.1. Objetivo del documento 36
2.1.2. Ámbito del documento 36
2.1.3. Descripción general del doc. 37

2.2. Descripción general 38

2.2.1. Perspectiva de la aplicación 38
2.2.2. Características de usuarios 39
2.2.3. Restricciones del sistema 40

2.3. Requerimientos específicos 40

2.3.1. Req. de las interfaces ext. 40
2.3.2. Requerimientos funcionales 40
2.3.3. Req. de rendimiento 42
2.3.4. Restricciones de diseño 42
2.3.5. Atributos sistema software 42

2.1. INTRODUCCIÓN

El análisis de requerimientos en todo proyecto, es indispensable, ya que es el punto de partida. Éste nos guiará a etapas posteriores como son la especificación, el diseño y la posterior implementación del sistema. En esta primera etapa de análisis no se tomará ninguna decisión técnica, sino que se realizará un explicación clara y concisa de la ideas.

La idea principal de este punto, es que una vez finalizada su lectura, el lector esté familiarizado con el proyecto y conozca todos los objetivos del mismo. Otra razón por la cual este capítulo es importante, es por la definición clara de todos los objetivos y funcionalidades que ha de ofrecer el sistema, de manera que a posteriori podamos solucionar las dudas redirigiéndonos a este punto.

2.1.1. OBJETIVO DEL DOCUMENTO

Este capítulo tiene como objetivo realizar una primera descripción en detalle de los objetivos que debe cumplir la aplicación, a parte de identificar de forma clara y no ambigua todos los requerimientos de ésta.

2.1.2. ÁMBITO DEL DOCUMENTO

Este proyecto se ha desarrollado bajo el ámbito del departamento de *Lenguajes y Sistemas Informáticos (LSI)* de la *Facultad d'Informatica de Barcelona (FIB)* de la *Universitat Politècnica de Catalunya (UPC)*.

Las principales funciones de *Un joc d'Scrabble* son:

- Proporcionar a los usuarios la posibilidad de participar en partidas de *Scrabble* de dos a cuatro jugadores. Los jugadores de una partida podrán ser de tipo *Humano* o *Robot*.

- Permitir a los usuarios gestionar la creación y eliminación de los perfiles de jugadores. Un perfil de jugador es la representación del usuario en la aplicación, tal que reunirá toda la información relevante a éste. A través de los perfiles de jugadores los usuarios podrán participar en las partidas.

- Proporcionar los diccionarios para los correspondientes idiomas de juego de la aplicación. Pudiendo ser modificados por el usuario en todo momento enriqueciendo así el contenido de los mismos.

- Proporcionar diversos idiomas de localización (idiomas de la interfaz de la aplicación) para una mayor accesibilidad a los usuarios en todo momento.

- Permitir a los usuarios gestionar diversas reglas (variaciones de juego) para personalizar así cada partida según las preferencias de cada uno de ellos.

- Proporcionar a los usuarios información y estadísticas de los perfiles de jugadores existentes en la aplicación a modo de ranking según diversos criterios como pueden ser: número de partidas ganadas, puntuaciones totales, puntuaciones máximas de partida, número de palabras formadas, etc.

2.1.3. DESCRIPCIÓN GENERAL DEL DOCUMENTO

Para la realización del análisis de requerimientos se han utilizado las recomendaciones de la *IEEE* según su estándar *830-1993*. Dichas recomendaciones describen el proceso de creación de la especificación de requisitos *software*. Alguno de los contenidos del presente capítulo han sido ya esbozados previamente y la razón de ser incluidos nuevamente es cumplir el estándar de documentación en el ámbito de Ingeniería de Software. Este documento se organiza de la siguiente forma:

- Descripción general
  - Perspectiva de la aplicación
  - Funciones de la aplicación
  - Características de usuario
  - Restricciones
  - Suposiciones y dependencias

- Requerimientos específicos
  - Requerimientos de las interfaces externas
  - Requerimientos funcionales
  - Requerimientos de funcionamiento
  - Restricciones de diseño
  - Atributos del sistema software

2.2. DESCRIPCIÓN GENERAL

2.2.1. PERSPECTIVA DE LA APLICACIÓN

- **Interfaces de usuario.** Las interfaces que visualizará el usuario y con las que podrá interaccionar con el sistema serán del tipo *WIMP (Windows, Icons, Menus and Pointers)*. Una aplicación con una interfaz de usuario de tipo *WIMP* requiere siempre una ventana en la que ejecutarse, por lo tanto, la aplicación contendrá una o más ventanas principales donde se irán añadiendo otras ventanas con las que el usuario interaccionará con el sistema.

- **Interfaces software.** Las tecnologías que requiere la aplicación son principalmente, *Java* para implementación de la misma, *JOGL (Java OpenGL)* para la visualización en modo 3D del tablero de juego, y ficheros de texto, encargados de mantener la persistencia de los datos. La aplicación está pensada para poder ser ejecutada en cualquier sistema operativo que pueda ejecutar una *JVM (Java Virtual Machine)* y que incorpore *OpenGL*. Tanto el desarrollo como las pruebas realizadas sobre la aplicación han sido en un sistema operativo *Windows XP*. Aunque no ha sido probado, no debería haber ningún problema en ejecutar la aplicación en otras versiones del sistema operativo *Microsoft® Windows®* como *Windows 98, Me, 2000 y NT* como cualquier versión estándar del sistema operativo *Linux.* Destacar que para la correcta ejecución en los sistemas operativos mencionados se deberán configurar las correspondientes bibliotecas dedicadas a la visualización 3D de la aplicación (proporcionadas en el CD adjunto a este documento). Así como una versión de *Java* igual o superior a la *JDK1.5.0_07*, también proporcionada en el CD adjunto a este documento.

- **Interfaces hardware.** Las interfaces hardware mínimas que se requieren son un PC, pantalla, teclado, y ratón. Los requisitos software mencionados anteriormente requieren unos requisitos mínimos hardware:

    - Equipo personal con procesador a 486DX a 66 megahercios o superior
    - Entre 256 y 512 *megabytes* de memoria RAM.
    - 20 *megabytes* de espacio libre en disco duro.
    - Tarjeta gráfica de resolución mínima 800x600.
    - CD-ROM para la obtención del software (*véase anexo manual de instalación y configuración*).

Aunque con los anteriores requisitos mínimos la aplicación puede ejecutarse, se recomiendan los siguientes requisitos para un óptimo funcionamiento:

- Equipo personal con procesador a 300 Megahercios o superior.
- 512 *megabytes* como mínimo de memoria RAM.
- Entre 25 y 50 *megabytes* de espacio libre en disco duro.
- Tarjeta gráfica SVGA de igual o superior resolución a 800x600.
- CD-ROM para la obtención del software (*véase anexo manual de instalación y configuración*)

- **Interfaces de comunicación.** No es necesario ningún tipo de protocolo de comunicación ya que se trata de una aplicación local sin ninguna funcionalidad que requiera comunicarse con alguna otra aplicación externa.

- **Restricciones de memoria.** Para determinar las restricciones de memoria se ha considerado la necesidad de consumo de memoria por parte del módulo que implementa el diccionario ya que para poder ofrecer un máximo rendimiento en velocidad de respuesta se ha necesitado, como se describe más adelante en este documento, hacer un elevado consumo de memoria. Un diccionario de *Scrabble* suele comprender entre 100.000 y 150.000 palabras, para las cuales se ha hecho necesaria una capacidad de memoria de entre 150 y 200 *megabytes* de RAM.

- **Requerimientos de adaptación.** No se precisa ningún requerimiento de adaptación a la ubicación de la aplicación.

2.2.2. CARACTERÍSTICAS DE LOS USUARIOS

Los usuarios de la aplicación serán cualquier persona que esté interesada en el mundo del *Scrabble*, y necesite una aplicación para poder realizar partidas contra otros usuarios, contra una inteligencia artificial, o simplemente en solitario a modo de entrenamiento y/o entretenimiento.

La aplicación deberá ser lo suficientemente intuitiva como para permitir su uso a cualquier tipo de usuario, es decir, irá dirigida tanto a usuarios inexpertos en informática como a usuarios avanzados. Para ello, se ha decidido diseñar pantallas intuitivas y con el mínimo número de opciones necesarias para no dificultar el uso de la aplicación. No es necesario que el

usuario domine las técnicas del *Scrabble* para poder realizar partidas, ya que la aplicación ofrece todas las posibles acciones a realizar en el turno de juego del usuario, siendo éste quien decide cual de ellas realizar en cada momento.

2.2.3. RESTRICCIONES DEL SISTEMA

La aplicación ha sido desarrollada bajo tecnología libre y por lo tanto se permite cualquier sistema operativo. No obstante, tanto el desarrollo como las pruebas realizadas sobre la aplicación han sido en un sistema operativo *Windows XP*.

2.3. REQUERIMIENTOS ESPECÍFICOS

Esta sección de la especificación de requisitos *software* contiene todos los requerimientos hasta un nivel de detalle suficiente para permitir a los diseñadores diseñar un sistema que satisfaga dichos requerimientos, y que permita diseñar las pruebas que ratifiquen que el sistema cumple con los requerimientos.

2.3.1. REQUERIMIENTOS DE LAS INTERFACES EXTERNAS

- **Interfaces de usuario.** Las interfaces de usuario no tienen que contener lógica del programa, es decir, tiene que haber una clara separación entre presentación y dominio.

- **Interfaces hardware.** La aplicación no requiere ningún periférico específico en el caso de una estación de trabajo estándar.

- **Interfaces software.** No hay requisitos sobre las interfaces *software*.

- **Interfaces de comunicación.** No hay requisitos sobre las interfaces de comunicaciones.

2.3.2. REQUERIMIENTOS FUNCIONALES

Un joc d'Scrabble debe permitir jugar una partida al juego conocido como *Scrabble* entre 2 y 4 jugadores. Estos jugadores pueden ser de tipo *Humano* o *Robots*. Los jugadores tipo *Robot* podrán configurarse con diferentes tipos de dificultad. Adicionalmente debe ofrecer la

posibilidad de jugar un único jugador a modo de entrenamiento para la capacidad de resolver jugadas. Las principales funciones del proyecto son:

- **Gestión de jugadores.** Los usuarios podrán a través de la aplicación crear perfiles de nuevos jugadores de dos tipos, de tipo *Humano* y de tipo *Robot*, éste último con cinco posibles variaciones de dificultad (*noob*, *fácil*, *normal*, *experto*, y *élite*). Así mismo, a través de la aplicación los usuarios podrán eliminar aquellos perfiles de jugador que no deseen a través del menú correspondiente. O incluso ver en detalle todas las características del perfil de jugador seleccionado, así como sus puntuaciones máximas, puntuaciones totales y el índice de partidas jugadas, ganadas y perdidas.

- **Gestión de diccionarios.** Como diccionarios nos referiremos al idioma de la partida. Éstos contendrán el listado completo de las palabras válidas para el idioma al que corresponden. El usuario podrá añadir y eliminar todas aquellas palabras que crea convenientes, previamente a la partida como en plena ejecución de la misma, teniendo constancia la validez de la palabra desde el mismo momento de su inserción en el diccionario. Adicionalmente, se podrán modificar los diccionarios tratando los ficheros de texto correspondientes.

- **Gestión de idiomas.** Como idiomas nos referiremos al idioma de localización, es decir, el idioma en que se presentará la interfaz de la aplicación, ofreciendo así una mayor accesibilidad a todos los usuarios. En cualquier momento de la ejecución de la aplicación podremos configurar en qué idioma desea el usuario visualizarla. Adicionalmente, se podrán añadir nuevos idiomas de localización importando a la ruta correspondiente del idioma los nuevos ficheros de texto correspondientes.

- **Gestión de reglas (variaciones de juego).** Los usuarios podrán configurar las reglas (variaciones de juego) de las partidas. Entre las diversas reglas se encuentran el número de fichas en el atril (para ciertos idiomas se suele jugar con un número mayor o menor al habitual, que son siete), el valor de la puntuación extra ganada al realizar un *Scrabble*, configurar o desactivar el tiempo limite por turno o por jugador, etc.

- **Gestión de estadísticas (rankings).** Los usuarios podrán consultar en todo momento ciertas estadísticas de los perfiles de jugador existentes en la aplicación en forma de rankings según diversos criterios como pueden ser: número de partidas ganadas o perdidas, puntuaciones totales, puntuaciones máximas de partida, número de palabras formadas, etc.

- **Gestión de partidas.** Los usuarios podrán seguir el transcurso de las partidas de una forma fácil, sencilla y cómoda gracias a la información mostrada en todo momento durante el transcurso de las mismas. Mediante los diversos botones, fácilmente identificables gracias a sus iconos y textos, podrán realizar todas aquellas acciones permitidas entre las cuales se encuentran: colocar, mover o quitar fichas del tablero, cambiar fichas del atril, pasar turno, deshacer la jugada, validar la jugada, etc.

2.3.3. REQUERIMIENTOS DE RENDIMIENTO

Se hace necesario dedicar especial atención a la velocidad de respuesta de la aplicación en el momento de obtener todas las jugadas posibles y qué jugada entre ellas llevar a cabo, cuando se trata del turno de partida de un jugador de tipo *Robot*. Esto es debido al gran número de posibles jugadas que se pueden realizar dado un tablero concreto y un atril. Cuanto más completo sea el diccionario usado, más grande será el conjunto de soluciones obtenido. Para poder obtener una velocidad de respuesta óptima, el consumo de memoria necesario para ello se verá notablemente incrementado. Es por lo tanto necesario, tener una cantidad de memoria mínima que poder asignar a la aplicación para que todo el proceso de obtención de jugadas no necesite ningún procesamiento adicional como pudiera ser la paginación a disco por falta de memoria. En los casos en que es posible ofrecer a la aplicación todos los recursos necesarios para su comportamiento óptimo, ésta se comporta de manera que los tiempos de obtención de jugadas son realmente ínfimos (escasos milisegundos generalmente).

2.3.4. RESTRICCIONES DE DISEÑO

La aplicación debe de ser una aplicación *Java Stand-alone*, es decir, no necesita de ningún elemento de soporte para su ejecución, como podría ser por ejemplo un servidor de aplicaciones.

2.3.5. ATRIBUTOS DEL SISTEMA SOFTWARE

- **Fiabilidad.** El sistema debe de soportar todas las funcionalidades anteriormente descritas y garantizar su buen funcionamiento durante todo el período de su ejecución. Para ello, se utilizarán patrones de diseño que han sido ampliamente estudiados y probados.

- **Disponibilidad.** Los ficheros utilizados para la persistencia de datos, tales como los diccionarios de juego e idiomas de localización, perfiles de jugadores, etc, deberán estar disponibles como mínimo al menos durante la ejecución de la aplicación.

- **Seguridad.** En un principio, será el propio usuario de la aplicación el encargado de garantizar la seguridad de los datos utilizados por ésta. De la misma manera, el propio usuario será el encargado de realizar *backups* de los ficheros con datos sensibles cuando él crea necesario.

- **Mantenibilidad.** El mantenimiento y modificaciones que se puedan hacer en el sistema, dependerá de las aplicaciones con las que se quiera integrar y las plataformas donde se quiera poner.

- **Eficiencia.** El sistema se ha de comportar de manera eficiente y rápida, tanto en la resolución de los problemas presentados a cada turno de juego, como en el desarrollo general de las partidas. Desde el punto de vista de la gestión de datos, la aplicación se ha diseñado de forma eficiente.

- **Facilidad de uso.** La aplicación va dirigida tanto a personas expertas en informática como a personas con conocimientos muy bajos o casi nulos, por tanto, su diseño ha de estar pensado para este último tipo de personas. Esto se deberá conseguir mediante menús y pantallas suficientemente intuitivas y fáciles de usar para cualquier usuario de la aplicación.

3. TECNOLOGÍAS UTILIZADAS

3.1. Lenguaje de programación 47

3.1.1. Java 47
3.1.2. Justificación 51

3.2. Visualización 3D 52

3.2.1. OpenGL 53
3.2.2. JOGL 56
3.2.3. Justificación 57

3.3. Persistencia de los datos 57

3.3.1. Ficheros de texto 58
3.3.2. Ficheros binarios 59
3.3.3. Justificación 59

3.4. Herramientas 61

3.4.1. Eclipse 61

3.1. LENGUAJE DE PROGRAMACIÓN

Existen innumerables lenguajes de programación, cada uno de ellos con sus ventajas e inconvenientes. La decisión de cual de ellos utilizar es muy complicada y se basa en las necesidades del usuario final y del desarrollador.

Un joc d'Scrabble se ha desarrollado mediante el lenguaje de programación *Java*. A continuación se describen las características y las razones de porqué se han utilizado este lenguaje y no otro.

3.1.1. JAVA

La tecnología *Java* consta de un lenguaje de programación y de una plataforma.

El lenguaje de programación Java

Java es un lenguaje de programación de alto nivel que tiene las siguientes características:

- Orientado a objetos
- Distribuido y dinámico
- Robusto
- Seguro
- Multitarea
- Portable

La mayoría de los lenguajes de programación se caracterizan por ser interpretados o compilados, lo que determina la manera en como serán ejecutados en un ordenador.

Java tiene la característica de ser al mismo tiempo compilado e interpretado. El compilador es el encargado de convertir el código fuente de un programa en un código intermedio llamado *bytecode* que es independiente de la plataforma en que se trabaje y que es ejecutado por el intérprete de *Java* que forma parte de la *Java Virtual Machine*.

Figura 3.1 Compilación y ejecución de programas en *Java*.

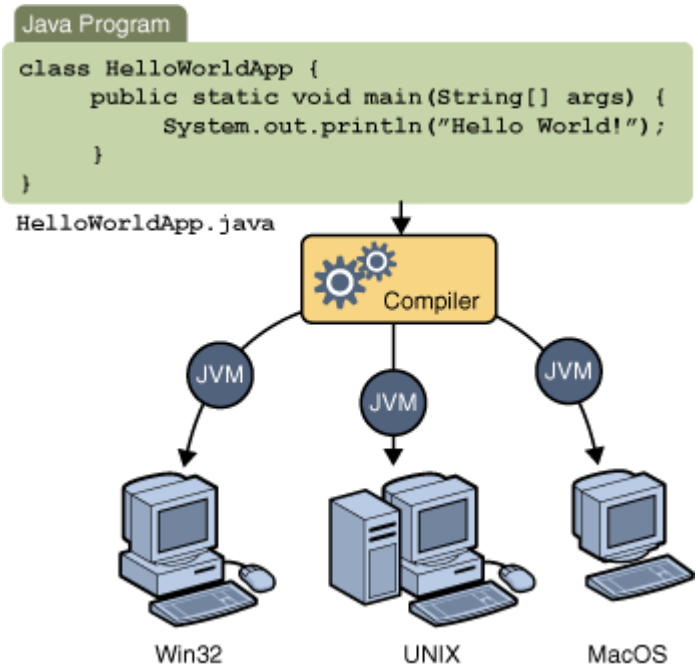


Figura 3.2 A través de la JVM, una misma aplicación puede ser ejecutada en múltiples plataformas.

La plataforma Java

Una plataforma es el ambiente de hardware o software en el cual se ejecutan los programas. En general, la mayoría de las plataformas pueden ser descritas como una combinación de hardware y sistema operativo. Algunas de las plataformas más populares son *Windows*, *Solaris*, *Linux* y *MacOS*. La plataforma *Java* difiere de las anteriores en que ésta es una plataforma basada únicamente en software que corre por encima de las plataformas basadas en hardware.

La plataforma *Java* consta de dos componentes:

- La *Java Virtual Machine* (JVM)
- La Interfaz de Programación de Aplicaciones de *Java* (API *Java*)

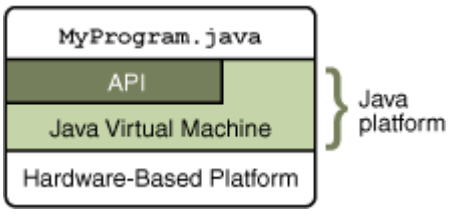


Figura 3.3 La API y la JVM aíslan el programa del hardware subyacente.

Tipos de programas en Java

Los programas en *Java* suelen estar en una de las siguientes categorías:

- **Aplicaciones.** Las aplicaciones son programas *stand-alone* de propósito general que normalmente se ejecutan desde la línea de comandos del sistema operativo. Con *Java* se puede realizar cualquier programa que normalmente se crearía con algún otro lenguaje de programación.

- **Applets.** Los *applets* son pequeños programas que se incorporan en una página *web* y que por lo tanto, necesitan de un navegador *web* compatible con *Java* para poder ejecutarse. A menudo los *applets* se descargan junto con una página HTML desde un servidor *web* y se ejecutan en la máquina cliente.

- **Servlets.** Los *servlets* al contrario de los *applets* son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones *web* que interactúen con los clientes. Los *servlets* son una alternativa de la programación CGI tradicional.

Compilación y ejecución de programas Java

El *kit* de desarrollo de *Java* (JDK) contiene las herramientas y librerías necesarias para crear y ejecutar *applets* y aplicaciones en *Java*. Se puede obtener directamente de forma gratuita del sitio *web* de *Java* http://java.sun.com/.

A continuación se listas algunas de las utilidades que se pueden encontrar en el JDK:

- **javac.** Es el compilador de *Java*. Se encarga de convertir el código fuente escrito en *Java* a *bytecode*.

- **java.** Es el intérprete de *Java*. Ejecuta el *bytecode* a partir de los archivos *class*.

- **appletviewer.** Es un visor de *applets*. En la mayoría de las ocasiones puede utilizarse en lugar de un navegador *web*.

- **javadoc.** Se utiliza para crear documentación en formato HTML a partir del código fuente *Java* y los comentarios que contiene.

- **javap.** Es un desensamblador de *Java*.

- **jar.** Es una herramienta utilizada para trabajar con los archivos JAR.

Un ejemplo del uso del JDK para construir aplicaciones es el siguiente:

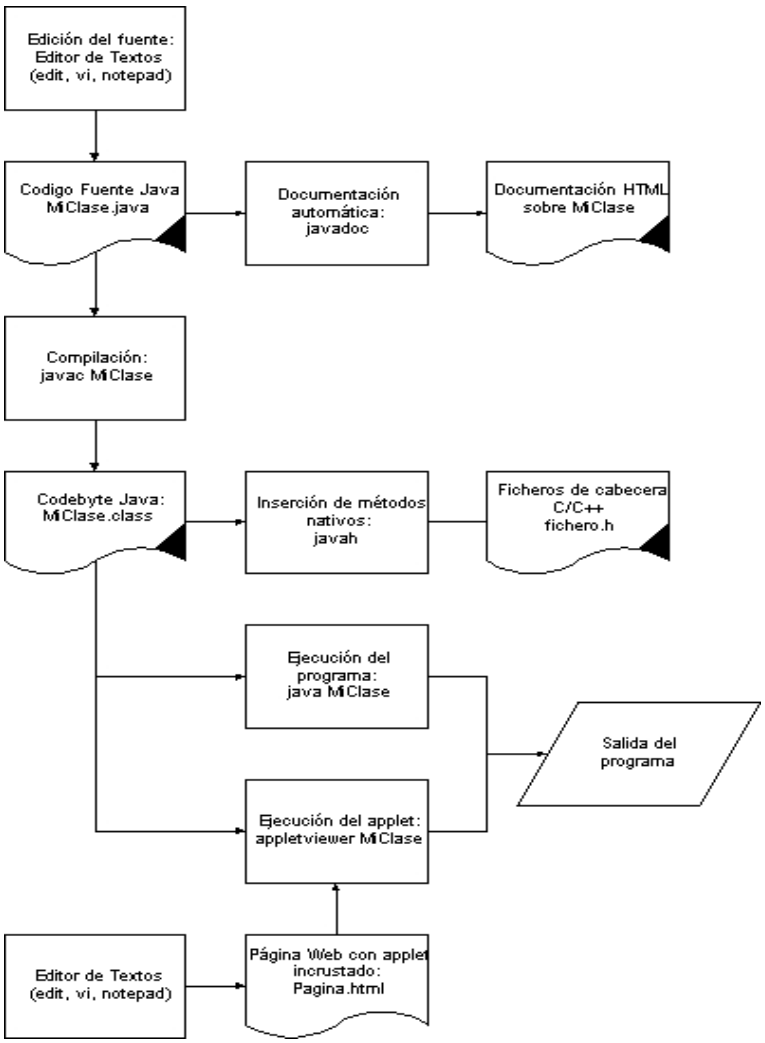


Figura 3.4 Ejemplo de uso del JDK

3.1.2. JUSTIFICACIÓN

Se ha escogido *Java* como lenguaje de programación, en vez de lenguajes como C, C++, entre otros, por los siguientes motivos:

- **Simplicidad.** El equipo original que desarrolló *Java,* diseño este lenguaje por necesidad, al no cumplir los lenguajes existentes con todo lo que necesitaban para el proyecto que tenían en marcha. Pero, como buenos diseñadores, no inventaron un lenguaje completamente nuevo, sino que se basaron en lo ya hecho y probado. Es por esta razón que el código *Java* se expresa en archivos de texto plano, y tiene una apariencia muy familiar para los programadores de C/C++ y para los programadores en general.

- **Orientación a objetos.** Posiblemente, ésta característica es una de las razones con más peso, de porqué se ha utilizado este lenguaje y no otro. A primera vista parece que todos los lenguajes de programación tienen algo "orientado a objetos". En el caso de *Java*, la aplicación de esa calificación es totalmente acertada. De hecho, en *Java*, todo es un objeto. *Java* se basa en la definición de clases, con atributos y métodos, agrupándose en paquetes, y exponiendo interfaces, permitiendo la herencia y el polimorfismo, implementando así la tecnología de objetos de una manera muy completa, y a la vez sencilla y agradable.

- **Soporte de múltiples plataformas.** El gran motivo de la elección de *Java*, ha sido éste, ya que el proceso de compilación no produce código para un procesador en particular, como en los compiladores C/C++, sino que genera código para un procesador ideal, denominado máquina virtual *Java* (*Java Virtual Machine*, o *Java VM*). La razón de esta elección es simple: la necesidad de poder ejecutar la aplicación en cualquier plataforma, sin necesidad de cambiar el código fuente, ni siquiera de recompilar. Todo programa *Java* se traduce en una serie de códigos (llamados *bytecodes*) para un procesador imaginario, el *Java VM*, que puede ser implementado fácilmente en cualquier plataforma. Es así como el lenguaje y sus aplicaciones se independizan de la máquina destino, a costa de algo de velocidad, provocada por la necesidad de "simular" la *Java VM*, mediante *software*. Otra característica es que los tipos elementales de datos quedan definidos para todas las plataformas. De esta manera nos evitamos así las pesadillas de portabilidad, tan comunes en programas desarrollados en C/C++, debido al cambio de

tamaño de los enteros o al conjunto de caracteres soportados. En *Java*, esos problemas no existen: sólo existe una máquina virtual.

- **Gestión automática de la memoria.** El lenguaje *Java* se compone de objetos, como se ha comentado anteriormente. Estos objetos se pueden y se deben crear, y tienen una vida que dura hasta su destrucción. Mientras que la creación de los objetos se deja bajo la voluntad del programador, la destrucción definitiva de un objeto ocurre cuando éste no es más referenciado por otros objetos del programa. De esta forma, se elimina una de las causas más comunes de error en otros lenguajes, como la destrucción por el programador de objetos aun en uso en el programa, o la falta de destrucción de objetos que ya son inútiles, pues no se usan en el resto de la ejecución, pero que molestan con empleo de recursos. Esta técnica de gestión automática de memoria se denomina *garbage collection*, recolección de basura. En una aplicación *Java* hay siempre un proceso, ejecutado como un hilo de ejecución separado, que se ocupa de recorrer la memoria donde se encuentran los objetos, y determinan cuáles se pueden destruir y cuáles no. Con esto nos evitamos los típicos problemas de punteros y problemas de memoria.

- **Java no se aísla y evoluciona.** *Java* no presenta demasiadas dificultades para comunicarse con librerías de otros lenguajes. Tiene capacidades definidas para implementar clases y métodos (o funciones), escritos, por ejemplo, en C/C++. Esto nos permite usar lo que ya está hecho, en cada plataforma de implementación. Además, debido al éxito que ha tenido *Java*, y a su difusión, cada día aparecen nuevas herramientas y paquetes de clases escritas por otros programadores, que podremos utilizar para desarrollar nuestra aplicación.

3.2. VISUALIZACIÓN 3D

Existen diversas maneras de visualizar la información de una aplicación cara al usuario. En este proyecto en concreto, al tratarse de una aplicación de entretenimiento y ocio, es muy importante que la información llegue al usuario de una manera clara y concisa, que no sea necesario un análisis tedioso de ésta por parte del usuario. Es por ello, que la mejor opción como en toda aplicación de entretenimiento es la visualización gráfica. Ya que mediante objetos, animaciones y todo aquello que sea realizable a través de gráficos podrá existir una interactividad sencilla, dinámica y fluida entre la aplicación y el usuario.

Llegados a éste punto, existen dos posibilidades, un modelo de visualización 2D, o un modelo de visualización 3D. Ambos modelos son totalmente válidos para el desarrollo de la aplicación a la que se refiere este documento. Ya que con ambos podemos conseguir representar toda aquella información necesaria de manera gráfica: el tablero de juego, las fichas dispuestas sobre éste y las correspondientes a cada jugador. Incluso, se podrá ofrecer al usuario la capacidad de mover las fichas él mismo sobre el tablero. Finalmente se ha optado por la utilización de un modelo 3D, principalmente porque a día de hoy, es mucho más actual y atractivo para el usuario un entorno de interacción 3D, asegurándonos así el no dar una apariencia ya anticuada.

Para poder ofrecer una visualización 3D, la aplicación se basa en *OpenGL*, y más concretamente *JOGL*, librerías en *Java* mediante las cuales se hace posible el enlace entre la aplicación y *OpenGL*.

3.2.1. OPENGL

OpenGL es una especificación estándar que define una *API* multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. Fue desarrollada por Silicon Graphics Inc. (SGI) en 1992. Su nombre viene del inglés *Open Graphics Library*, cuya traducción es *biblioteca de gráficos abierta* (o mejor, *libre*, teniendo en cuenta su política de licencias).

OpenGL se utiliza en campos como CAD, realidad virtual, representación científica y de información, simulación de vuelo o **desarrollo de videojuegos**, en el que su principal competidor es *Direct3D* de Microsoft Windows.

- **Especificación.** A grandes rasgos, *OpenGL* es una especificación, es decir, un documento que describe un conjunto de funciones y su comportamiento exacto. A partir de ella, los fabricantes de hardware crean implementaciones (bibliotecas de funciones creadas para enlazar con las funciones de la especificación *OpenGL*, utilizando aceleración hardware cuando sea posible). Dichos fabricantes tienen que superar pruebas específicas que les permitan calificar su implementación como una

implementación de *OpenGL*. Existen implementaciones eficientes de *OpenGL* suministradas por fabricantes para Mac OS, Microsoft Windows, Linux, varias plataformas Unix, y PlayStation 3. Existen también varias implementaciones software que permiten que *OpenGL* esté disponible para diversas plataformas sin soporte de fabricante. Es de señalar la biblioteca de software libre / código abierto Mesa 3D, una API de gráficos basada totalmente en software y completamente compatible con *OpenGL*. Sin embargo, para evitar los costes de la licencia para ser denominada formalmente como una implementación de *OpenGL*, afirma ser simplemente una API muy *similar*. La especificación *OpenGL* era revisada por el OpenGL Architecture Review Board (**ARB**), fundado en 1992. El ARB estaba formado por un conjunto de empresas interesadas en la creación de una API consistente y ampliamente disponible. Microsoft, uno de los miembros fundadores, abandonó el proyecto en 2003.

- **Diseño.** OpenGL tiene dos propósitos principales:

  - Ocultar la complejidad de la interfaz con las diferentes tarjetas gráficas, presentando al programador una API única y uniforme.

  - Ocultar las diferentes capacidades de las diversas plataformas hardware, requiriendo que todas las implementaciones soporten el conjunto completo de características de OpenGL (utilizando emulación software si fuese necesario).

  La operación básica de *OpenGL* es aceptar primitivas tales como puntos, línea y polígonos, y convertirlas en píxeles. Este proceso es realizado por una *pipeline* gráfica conocida como la *Máquina de estados de OpenGL*. *OpenGL* es una API basada en procedimientos de bajo nivel que requiere que el programador dicte los pasos exactos necesarios para renderizar una escena. Esto contrasta con las APIs descriptivas, donde un programador sólo debe describir la escena y puede dejar que la biblioteca controle los detalles para renderizarla. El diseño de bajo nivel de *OpenGL* requiere que os programadores conozcan en profundidad la *pipeline* gráfica, a cambio de la libertad ofrecida en la implementación de algoritmos novedosos de renderizado. *OpenGL* ha influido en el desarrollo de las tarjetas gráficas, promocionando un nivel básico de funcionalidad que actualmente es común en el hardware comercial; algunas de esas contribuciones son:

  - Primitivas básicas de puntos, líneas y polígonos rasterizados.
  - Una *pipeline* de transformación e iluminación.
  - Z-buffering

- Mapeado de texturas
- Alpha blending

Una descripción somera del proceso en la *pipeline* gráfica podría ser:

1. Evaluación, si procede, de las funciones polinomiales que definen ciertas entradas, como las superficies *NURBS*, aproximando curvas y la geometría de la superficie.

2. Operaciones por vértices, transformándolos, iluminándolos según su material y recortando partes no visibles de la escena para producir un volumen de visión.

3. Rasterización, o conversión de la información previa en píxeles. Los polígonos son representados con el color adecuado mediante algoritmos de interpolación.

4. Operaciones por fragmentos o segmentos, como actualizaciones según valores venideros o ya almacenados de profundidad y de combinaciones de colores, entre otros.

5. Por último, los fragmentos son volcados en el *Frame buffer*.

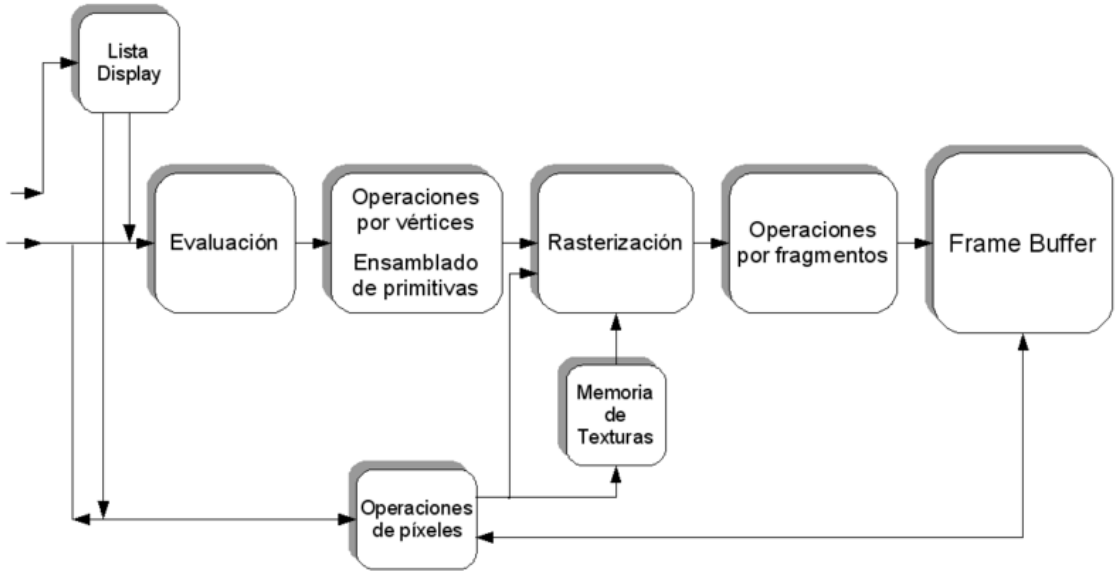


Figura 3.5 Proceso en la *pipeline* de gráficos

3.2.2. JOGL

Para enfatizar las características multilenguaje y multiplataforma de *OpenGL*, se han desarrollado varios *bindigs* en muchos lenguajes. *JOGL* se trata de uno de ellos, concretamente para *Java.*

Java OpenGL (JOGL) es una de muchas bibliotecas del entorno que permite que *OpenGL* sea utilizado en el lenguaje de programación *Java*. Actualmente está siendo desarrollada por el grupo de la tecnología de juegos de *Sun Microsystems*.

JOGL permite el acceso a la mayoría de las características disponibles para los programadores del lenguaje de programación *C*, con la notable excepción del sistema de ventanas relacionado llamado *GLUT* (mientras que *Java* contiene sus propios sistemas de ventanas, *AWT* y *SWING*, de alto nivel), y algunas extensiones. La base de *OpenGL C API* está alcanzada en *JOGL* a través de llamadas nativas del interfaz de *Java* (JNI). Como tal, el sistema subyacente debe apoyar *OpenGL* para que *JOGL* trabaje.

JOGL se diferencia de algunas otras bibliotecas del entorno de *Java OpenGL* en que expone simplemente la *OpenGL API* a través de métodos de algunas clases, más bien procurando traspasar la funcionalidad de *OpenGL* sobre el paradigma de programación orientada a objetos de *Java*. De hecho, la mayoría del código de *JOGL* es autogenerado por los ficheros de cabecera *OpenGL C*, a través de una herramienta de conversión nombrada *Gluegen*, que fue programada específicamente para facilitar la creación de *JOGL*.

Esta decisión de diseño tiene sus ventajas y desventajas. La naturaleza procedural y de máquina de estados de *OpenGL* es contraria con el método típico de programación bajo *Java*, que es fastidioso a muchos programadores. La delgada capa de la abstracción proporcionada por *JOGL* hace la ejecución runtime absolutamente eficiente, pero es por consiguiente más difícil al código comparada a las bibliotecas de alto nivel de abstracción como *Java3D*. Porque la mayor parte del código está autogenerado, los cambios producidos en *OpenGL*, pueden ser añadidos mucho más rápidamente a *JOGL*.

En fecha de 2006, *JOGL* proporciona total acceso a la especificación de *OpenGL 2.0*.

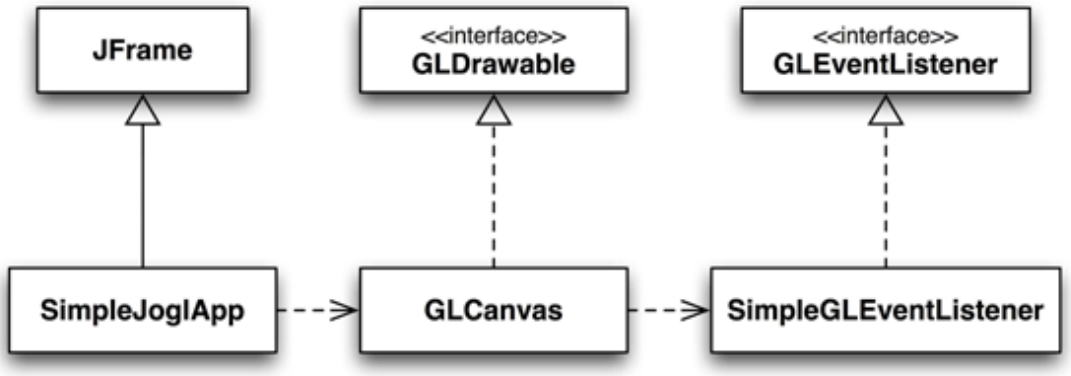


Figura 3.6 Estructura básica de una aplicación *JOGL*

3.2.3. JUSTIFICACIÓN

Se ha escogido el binding de *OpenGL* conocido como *JOGL* por:

- **Menor carga en el motor 3D.** A los desarrolladores de juegos les gusta que su código se ejecute lo más cerca del hardware como sea posible. La carga inherente en el motor de escenas genérico de cómo por ejemplo *Java3D* puede ser mucho más de lo que estos desarrolladores quieran soportar. Para este tipo de desarrollos, los bindings *OpenGL* (*JOGL*) son probablemente más apropiados.

- **Funcionalidades.** *JOGL* al tratarse de accesos a más bajo nivel que otras librerías o bindings *OpenGL*, ofrece las últimas funcionalidades aparecidas en *OpenGL* con el máximo de rendimiento y el acceso al nivel más bajo a los recursos gráficos.

- **Familiarización con *OpenGL*.** *JOGL*, tiene también la ventaja de utilizar un API estándar para gráficos 3D. Así que si estás ya familiarizado con *OpenGL* y quieres usar Java, esta es una gran forma de usar ambas cosas juntas.

3.3. PERSISTENCIA DE LOS DATOS

Para implementar la persistencia de los datos existían principalmente dos opciones totalmente opuestas:

- Ficheros
- SGBD *(Sistemas Gestores de Bases de Datos)*

Después de estudiar ambas posibilidades con sus correspondientes ventajas y desventajas, se ha optado por utilizar ficheros. A continuación se expone los motivos por los que la balanza se ha declinado por ficheros y no sobre un SGBD que a primera vista puede parecer la mejor opción.

3.3.1. FICHEROS DE TEXTO

Un fichero de texto (o fichero de texto plano) es un fichero electrónico que contiene únicamente caracteres textuales ordinarios con esencialmente ningún formato. El término *fichero de texto* se utiliza típicamente como contrario al término *fichero binario*, aún cuando cualquier fichero es fundamentalmente una secuencia de bits arbitrarios, y muchos componentes de los ordenadores (por ejemplo, todo el trazado de circuito del disco duro y la mayoría del software del sistema) no hacen ninguna distinción entre los dos tipos de fichero. Sin embargo, un gran porcentaje de los programas usados puede entender y utilizar ficheros de texto de cierta manera, pero pocos programas pueden entender y utilizar típicamente el contenido de cualquier fichero binario particular. Por lo tanto la distinción puede ser útil a los usuarios del ordenador.

Los ficheros de texto son los ficheros donde la mayoría de los *bytes* (o secuencias cortas de *bytes*) representan caracteres legibles ordinarios tales como letras, dígitos, y puntuación (espacios incluidos), e incluyen algunos caracteres de control tales como tabuladores, avances de línea y retorno de carro. Esta simplicidad permite que una amplia variedad de programas exhiba su contenido.

Generalmente, un fichero de texto contiene caracteres en una codificación basada en *ASCII*, o mucho menos comúnmente una codificación basada en *EBCDIC*, sin ninguna información incluida tal como información de la fuente, hyperlinks o imágenes en línea. Los ficheros de texto se codifican a menudo en una extensión del *ASCII*; éstos incluyen *ISO 8859*, *EUC*, una codificación especial para Windows, una codificación *Mac-Romana* especial para el OS del Mac, y los esquemas de codificación de *Unicode* (comunes en muchas plataformas) por ejemplo *UTF-8* o *UTF-16*.

Aunque los ficheros de texto se presentan para que puedan ser leídos por seres humanos, son también de uso general para el almacenaje de datos por programas informáticos.

3.3.2. FICHEROS BINARIOS

Un fichero binario es un archivo informático que contiene información de cualquier tipo, codificada en forma binaria para el propósito de almacenamiento y procesamiento en ordenadores. Por ejemplo los archivos informáticos que almacenan texto formateado o fotografías.

Muchos formatos binarios contienen partes que pueden ser interpretados como texto. Un archivo binario que *sólo* contiene información de tipo textual sin información sobre el formato del mismo se dice que es un archivo de texto plano. Habitualmente se contraponen los términos *archivo binario* y *archivo de texto* de forma que los primeros no contienen solamente texto.

Tal y como se detallará posteriormente en este documento, la aplicación tratará ficheros de ambos tipos, de texto y binarios, según el propósito de éstos. Ya que, por ejemplo, para almacenar listados de palabras válidas (diccionarios), un fichero de texto será el más adecuado, pudiendo así un usuario comprender su contenido para posteriormente modificarlo, mientras que para almacenar los datos de los perfiles de jugadores, un fichero binario impedirá a cualquier usuario, poder modificar los datos contenidos, y evitar así la falsificación de los diferentes tipos puntuaciones y contadores.

3.3.3. JUSTIFICACIÓN

Finamente he optado por la utilización de ficheros de texto como la base de la persistencia de datos de la aplicación, que engloba la información de los idiomas (tanto de idiomas de juego, como de aplicación, así como de la propia configuración de la misma) por los siguientes motivos aquí expuestos:

- **Concepto de videojuego.** Como uno de los principales motivos, ha sido el concepto de videojuego local en el que no encaja una base de datos para mantener la información, sino ciertos ficheros en el número que sean necesarios para almacenarla y cargarla en memoria en el momento que se necesite en el transcurso de éste. Como excepción podemos comentar los videojuegos online donde toda la información reside centralizada en un servidor y en volumen realmente grande, donde si necesitamos de un sistema de almacenaje realmente eficiente y fiable.

- **Instalación y configuración.** Juntamente al concepto de videojuego, encuentro que de cara al usuario no es factible que deba instalar y configurar un *SGBD* para poder ejecutar la aplicación que nos traemos entre manos por dos motivos:

    - Posiblemente, el usuario no desee más que instalar la propia aplicación para poder hacer uso de ella, y puede mostrarse reacio a la idea de tener que instalar más aplicaciones adicionales.

    - La mayor parte de usuarios que de aplicaciones informáticas no tiene porqué tener conocimientos informáticos y mucho menos de *Bases de Datos*, por lo que instalar y configurar una puede suponer todo un reto o incluso la imposibilidad de poder disfrutar de la aplicación.

- **Volumen de información.** La cantidad de información con la que la aplicación trabaja no supone un volumen ni mucho menos excesivo como para tener que descansar sobre una base de datos. Si bien es cierto que la cantidad de palabras válidas puede ser realmente elevada, entre 150.000 y 500.000, no deja de ser un simple listado de información totalmente independiente del contenido que no contiene ninguna relación entre ella misma.

- **Simplicidad, y comodidad para el usuario.** De cara a las modificaciones y ampliaciones de los diccionarios por parte de los usuarios, es mucho más simple y cómodo poderlo hacer sobre ficheros de texto, los cuales son entendibles al lenguaje humano y pueden ser abiertos con cualquier editor de textos de los sistemas operativos. No teniendo más que modificar la misma información ya existente o añadiendo la nueva. De manera que no es necesario que deban aprender ningún nuevo lenguaje como *SQL* para poder manipular y visualizar la información contenida en al *Base de Datos*.

  Si bien es cierto, que la propia aplicación permite añadir nuevas palabras de forma transparente al sistema de almacenaje de los datos, no es un camino factible para introducir cualquier colección de tamaño considerable de nuevas palabras que se haya podido construir, siendo mucho más simple y cómoda, la inserción directa sobre el propio fichero de texto.

3.4. HERRAMIENTAS

Como suite de desarrollo para aplicación *Java* se ha escogido *Eclipse* entre la gran cantidad de posibilidades, por sus numerosas ventajas que a continuación expondremos.

3.4.1. ECLISPSE

Eclipse es una plataforma de software de *código abierto* independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de cliente enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar un *Entorno integrado de desarrollo* (del inglés *IDE*), como el *IDE* de *Java* llamado *Java Development Toolkit* (*JDT*) y el compilador (*ECJ*) que se embarca como parte de *Eclipse* (y que son usados también para desarrollar el mismo *Eclipse*).

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado *Eclipse Modeling Project*, cubriendo casi todas las áreas de *Model Driven Engineering*.

Eclipse fue desarrollado originalmente por *IBM* como el sucesor de su familia de herramientas para *VisualAge*. *Eclipse* es ahora desarrollado por la *Fundación Eclipse*, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

El entorno integrado de desarrollo (*IDE*) de *Eclipse* emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son *C/C++* y *Phyton*, permite a *Eclipse* trabajar con lenguajes para procesado de texto como *LaTeX*, aplicaciones en red como *Telnet* y *Sistema de gestión de base de datos*. La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente,

como seria Gestión de la configuración. Se provee soporte para *Java* y *CVS* en el *SDK* de *Eclipse*. Este no debe ser usado solamente para soportar otros lenguajes de programación.

La definición que da el proyecto *Eclipse* acerca de su software es: "*una especie de herramienta universal - un IDE abierto y extensible para todo y nada en particular*".

El *SDK* de *Eclipse* incluye las herramientas de desarrollo de *Java*, ofreciendo un *IDE* con un compilador de *Java* interno y un modelo completo de los archivos fuente de *Java*. Esto permite técnicas avanzadas de refactorización y análisis de código.

Eclipse fue liberado originalmente bajo la *Common Public License*, pero después fue re-licenciado bajo la *Eclipse Public License*. La *Free Software Foundation* ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con GNU *General Public License* (*GPL*). Mike Milinkovich, de la fundación *Eclipse* comento que moverse a la *GPL* será considerado cuando la versión 3 de la *GPL* sea liberada.

4. ESPECIFICACIÓN

4.1. Introducción 66
4.2. Diagramas de clases 67
4.3. Casos de uso 68
4.4. Diagramas de casos de uso 70

4.4.1. Gestión de jugadores 71
4.4.2. Gestión de diccionarios 71
4.4.3. Gestión de idiomas 72
4.4.4. Gestión de reglas 72
4.4.5. Gestión de estadísticas 72
4.4.6. Gestión de partidas 73

4.5. Especificación de los casos de uso 74

4.5.1. Gestión de jugadores 75
4.5.2. Gestión de diccionarios 77
4.5.3. Gestión de idiomas 79
4.5.4. Gestión de reglas 80
4.5.5. Gestión de estadísticas 81
4.5.6. Gestión de partidas 82

4.6. Modelo de comportamiento 93

4.6.1. Diagramas de secuencia 93
4.6.2. Contratos de las operaciones 100

4.7. Modelo de estados 110

4.7.1. Diagramas de clases 110
4.7.2. Diagramas de casos de uso 112

4.1. INTRODUCCIÓN

La especificación es la etapa de la ingeniera del *software* que, una vez analizados los requerimientos que ha de cumplir el sistema, nos permite representarlos de una manera más formal para poder conseguir con éxito la implementación de este. La especificación pretende dar respuesta a la pregunta ¿qué ha de hacer el sistema?, y trata de definir el comportamiento del sistema especificando formalmente los requerimientos anteriormente descritos.

A lo largo de este capítulo se describirá conceptualmente el sistema, es decir, la especificación de la aplicación. Para realizar esta descripción del sistema se ha utilizado UML (*Unified Modeling Language*), que es un lenguaje modular muy potente y popular.

La parte de especificación normalmente se realiza mediante diagramas y especificación de casos de uso, diagramas de secuencia, diagramas de estado, modelo conceptual y contratos de las operaciones del sistema.

Primero de todo se mostrará el diagrama de clases del sistema. Los diagramas de clases son diagramas de estructura estática que muestran las clases del sistema y sus interrelaciones (incluyendo herencia, agregación, asociación, etc.). Los diagramas de clase son el pilar básico del modelado con UML, siendo utilizados tanto para mostrar lo que el sistema puede hacer (análisis), como para mostrar cómo puede ser construido (diseño).

En este apartado se mostrará el diagrama de casos de uso, y su especificación, que servirá para identificar las funcionalidades del sistema y saber qué función desempeñaran. También se describirán los diagramas de secuencia que nos ayudaran a identificar los sucesos y las operaciones del sistema. Éstos se basan en los casos de uso, y su función es describir con más detalle los casos de uso, mostrando las operaciones implicadas en estos, y el orden en el cual se van produciendo.

Para poder realizar la especificación se ha utilizado el libro *Enginyeria del software: Especificació*, que describe la notación UML y define el flujo de trabajo básico para la especificación de sistemas de información.

4.2. DIAGRAMA DE CLASES

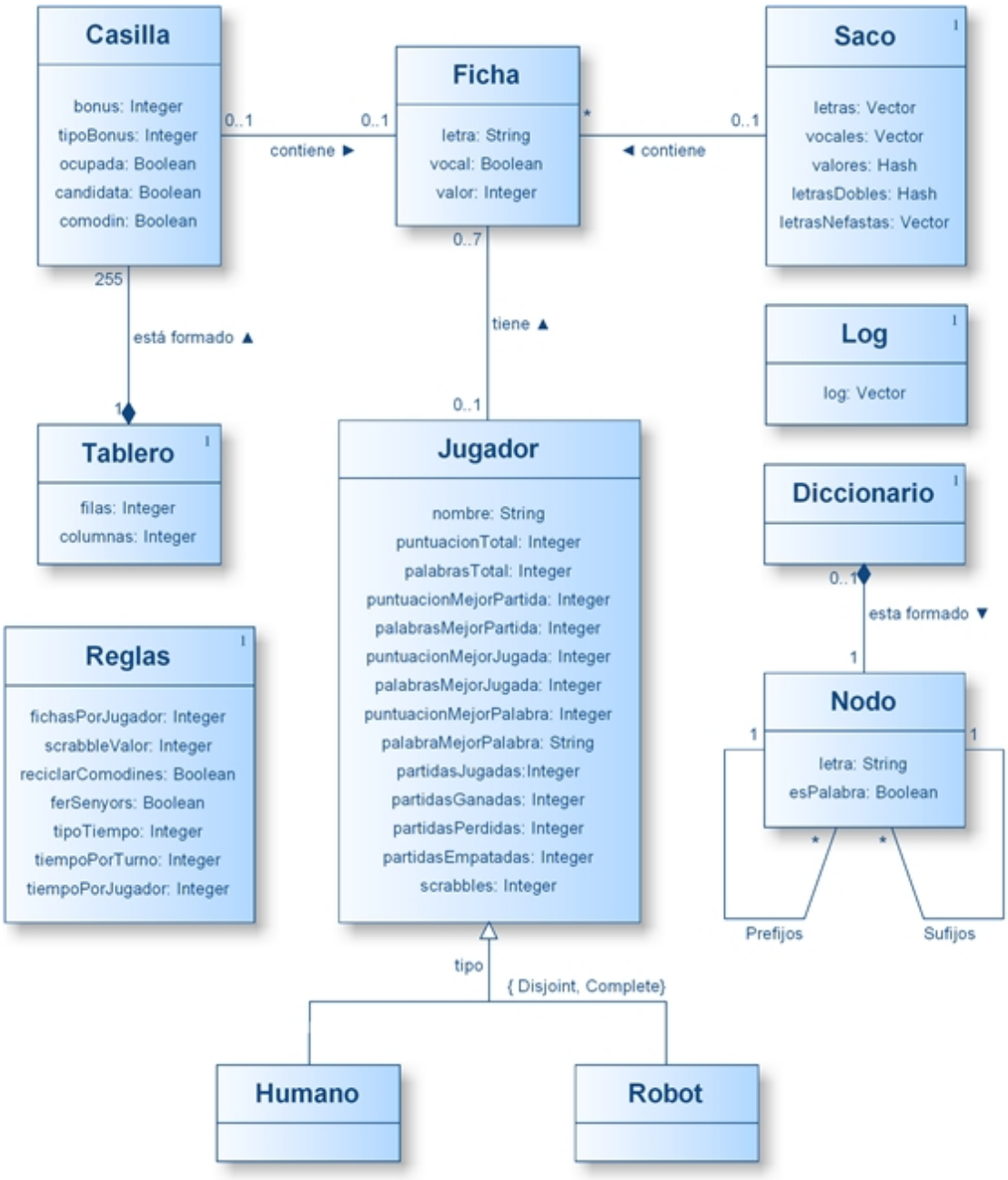


Figura 4.1 Diagrama de clases

Restricción de integridad:

- Una instancia concreta de ficha o es contenida en el *Saco*, o es contenida en una *Casilla* o pertenece a un *Jugador* en un mismo instante de tiempo.
- *Jugador* tiene como clave Nombre.

4.3. CASOS DE USO

Un caso de uso es un documento que describe los actores utilizando un sistema para satisfacer un objetivo. Es una forma particular de utilizar un sistema, normalmente es iniciado por uno de los actores y produce un efecto en los actores participantes y en el sistema. Es una descripción de alto nivel del sistema donde se han de evitar conceptos de diseño.

Los casos de uso describen, mediante acciones y reacciones, el comportamiento del sistema des del punto de vista del usuario. Permiten definir los límites del sistema y las relaciones entre el sistema y el entorno. Son descriptores de las funcionalidades del sistema independientemente de su implementación.

Nos hemos de asegurar que cada caso de uso describe una parte significativa del funcionamiento del sistema. Éstos han de ser lo suficientemente entendedores tanto por los desarrolladores de *software* como por los expertos del dominio. En el lenguaje *UML*, un caso de uso se representa de la siguiente manera:

Figura 4.2 Representación *UML* de un caso de uso

Un actor es una entidad externa al sistema que participa en la historia del caso de uso. Puede ser una persona, un conjunto de personas, un sistema *hardware*, un sistema *software* o un reloj. El actor que genera el suceso que provoca la ejecución del proceso se le denomina iniciador, y el resto de actores que intervienen en el proceso son los participantes. En el lenguaje *UML*, un actor se representa de la siguiente manera:

Figura 4.3 Representación *UML* de un actor

Los casos de uso son requerimientos, en particular requerimientos funcionales. La identificación de casos de uso requiere un conocimiento de los requerimientos, y se basa en la revisión de los documentos de requerimientos existentes.

Inicialmente se muestran todos los casos de uso, organizados por funcionalidades en las siguientes tablas:

Gestión de jugadores
Crear jugador
Borrar jugador
Ver detalles jugador

Gestión de diccionarios
Definir diccionario por defecto
Añadir palabra
Eliminar palabra

Gestión de idiomas
Seleccionar idioma
Definir idioma por defecto

Gestión de reglas
Modificar reglas

Gestión de estadísticas
Ver ranking

Gestión de partidas
Cargar partida
Salvar partida
Seleccionar jugador
Deseleccionar jugador
Seleccionar diccionario
Remover fichas
Cambiar fichas
Deshacer jugada
Pasar turno
Realizar jugada (colocar ficha en tablero)
Realizar jugada (quitar ficha de tablero)
Realizar jugada (mover ficha de tablero)

Validar jugada
Reciclar comodines
Fer senyors
Siguiente turno
Ver log
Salvar log

4.4. DIAGRAMA DE CASOS DE USO

El diagrama de casos de uso muestra conjuntamente los diferentes casos de uso de un sistema *software*, los actores y las relaciones entre actores y casos de uso.

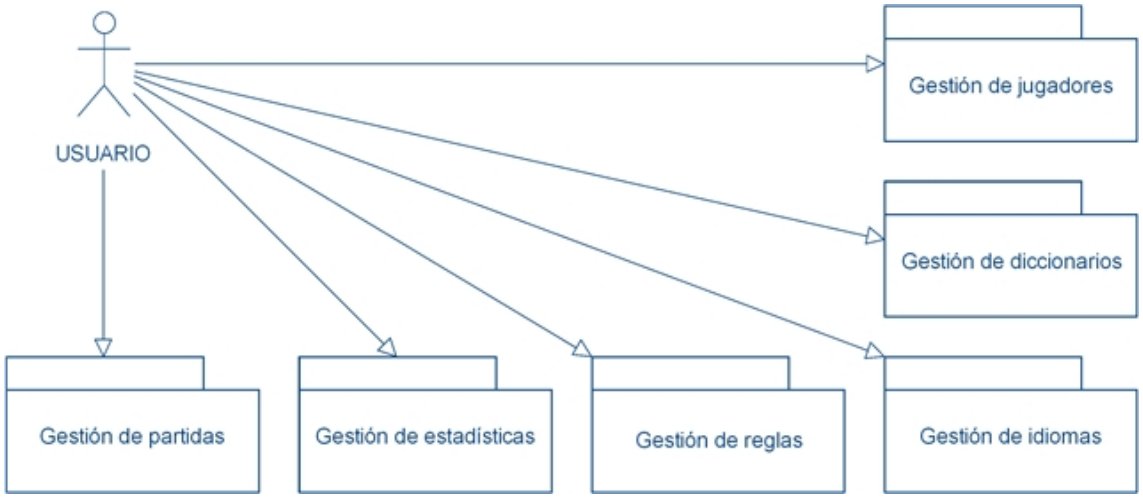


Figura 4.4 Diagrama de casos de uso

Se ha divido el diagrama de casos de uso en diagramas más pequeños para que se pueda visualizar de forma más estructurada las relaciones que existen entre los casos de uso. La separación se ha realizado según la funcionalidad a la que pertenecen.

4.4.1. DIAGRAMA: GESTIÓN DE JUGADORES

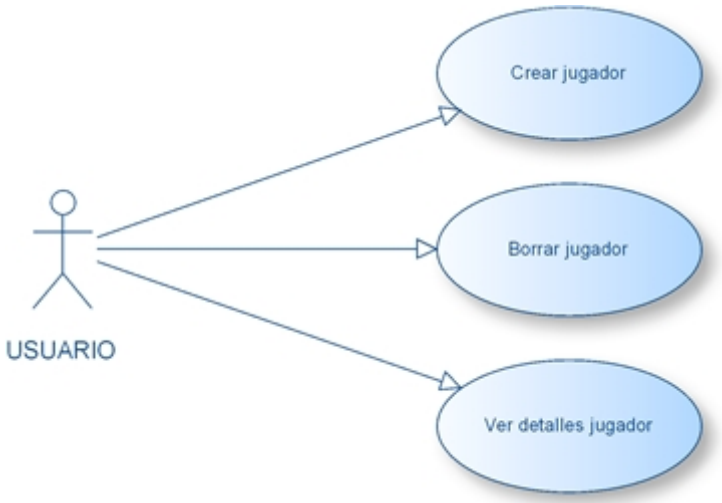


Figura 4.5 Diagrama: Gestión de jugadores

4.4.2. DIAGRAMA: GESTIÓN DE DICCIONARIOS

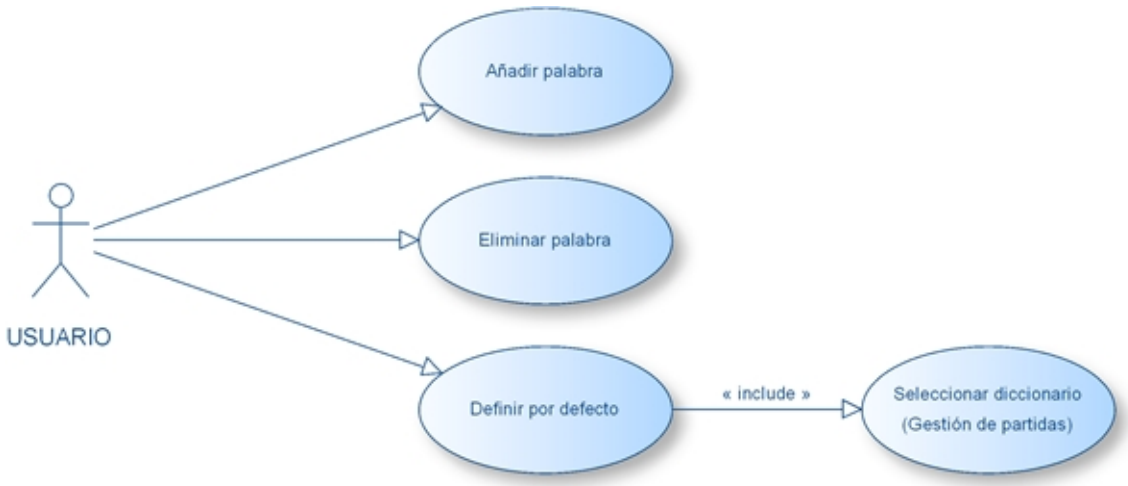


Figura 4.6 Diagrama: Gestión de diccionarios

4.4.3. DIAGRAMA: GESTIÓN DE IDIOMAS

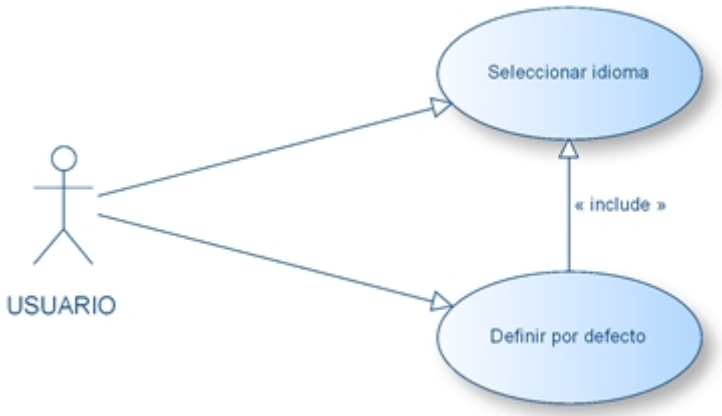


Figura 4.7 Diagrama: Gestión de idiomas

4.4.4. DIAGRAMA: GESTIÓN DE REGLAS

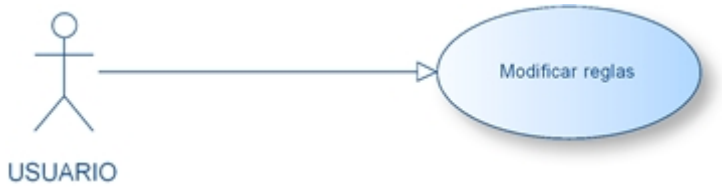


Figura 4.8 Diagrama: Gestión de reglas

4.4.5. DIAGRAMA: GESTIÓN DE ESTADÍSTICAS

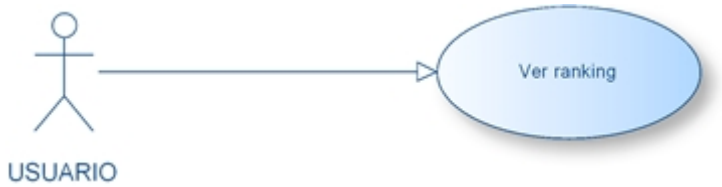


Figura 4.9 Diagrama: Gestión de estadísticas

4.4.6. DIAGRAMA: GESTIÓN DE PARTIDAS

Este diagrama se ha divido en subdiagramas según el contexto en el que actúan dentro de todo el proceso de gestión de las partidas para poderlo mostrar de una forma más sencilla y clara de cara al lector.

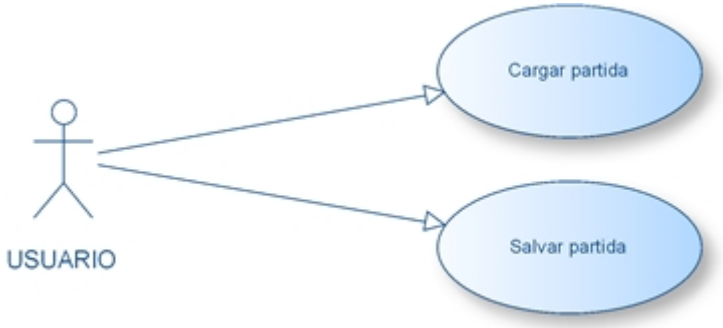


Figura 4.10 Diagrama: Gestión de partidas

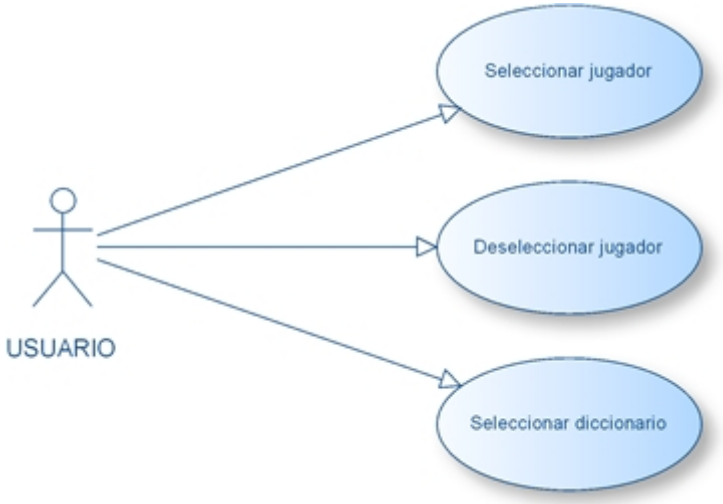


Figura 4.11 Diagrama: Gestión de partidas

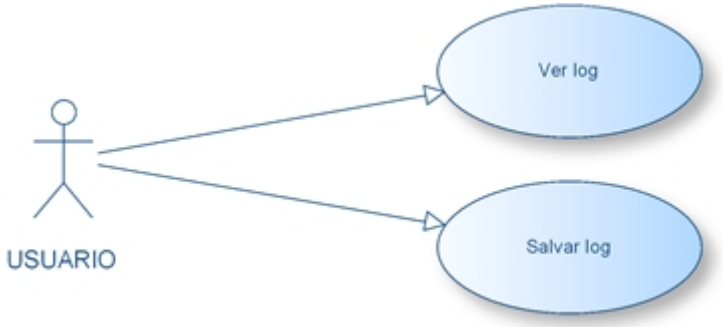


Figura 4.12 Diagrama: Gestión de partidas

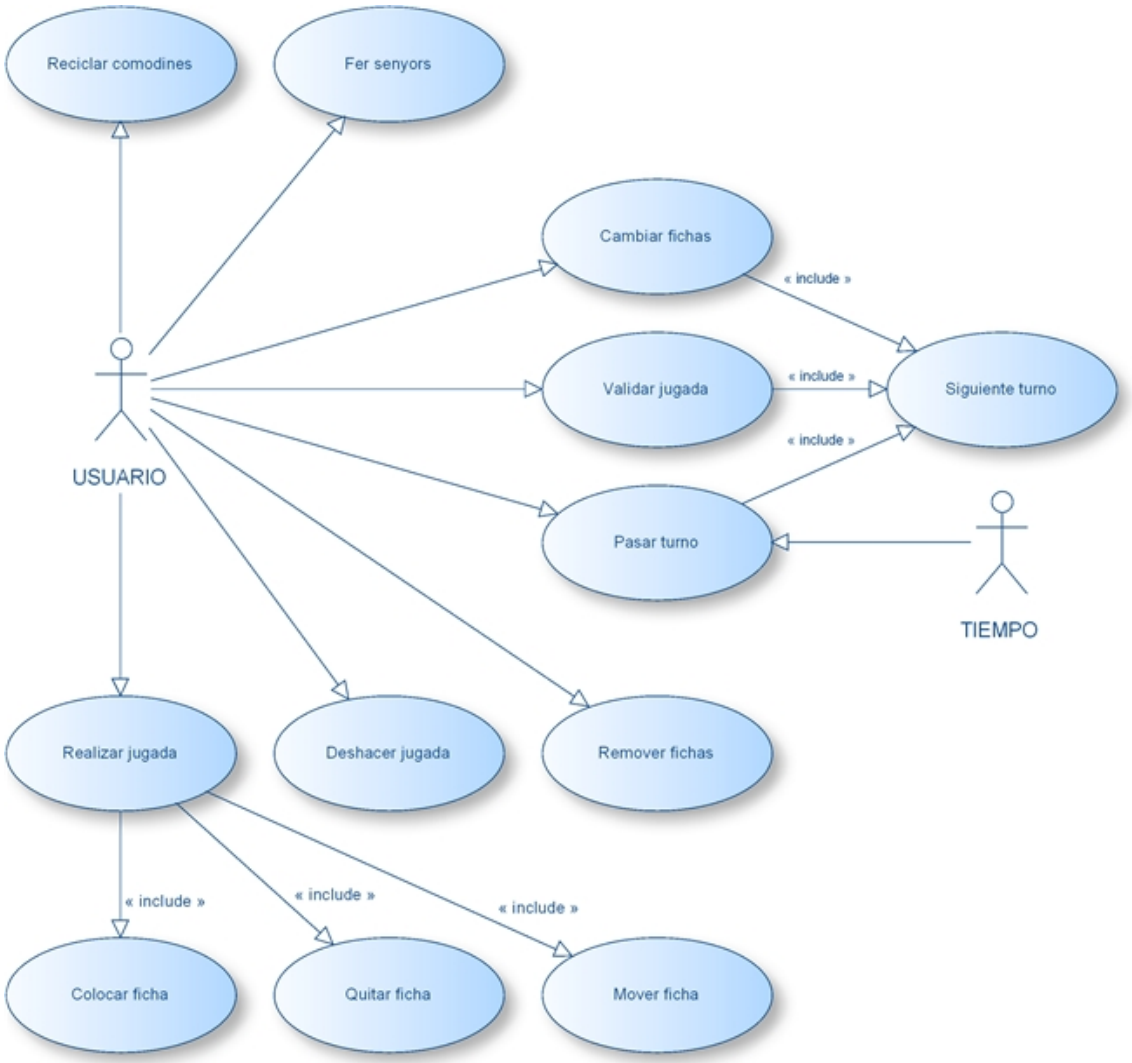


Figura 4.13 Diagrama: Gestión de partidas

4.5. ESPECIFICACIÓN DE LOS CASOS DE USO

La especificación de los casos de uso nos permite conocer el comportamiento externo que define la posible secuencia de mensajes intercambiados entre los actores y el sistema. Hay dos maneras de especificar los casos de uso, de alto nivel o expandida. La manera de alto nivel es una breve descripción de las acciones de los casos de uso que incluye los siguientes apartados:

- **Caso de uso:** Nombre del caso de uso.
- **Actores:** Lista de los actores, iniciador.
- **Propósito:** Objetivo del caso de uso.
- **Resumen:** Breve descripción de las actividades que se han de llevar a cabo.

- **Tipo:** 1 – primario, secundario, opcional.
  2 – real o esencial.

Si los especificamos de manera expandida, lo que haremos será describir detalladamente las acciones y los requerimientos. Añadiremos los siguientes apartados:

- **Referencias cruzadas:** Requerimientos a los que hace referencia.
- **Curso típico de sucesos:** Descripción detallada de la interacción entre los actores y el sistema. Se describen los sucesos paso a paso.
- **Cursos alternativos:** Descripción de las excepciones del curso típico.

A continuación se muestran las especificaciones de los casos de uso comentados en los apartados anteriormente en este mismo documento.

4.5.1. ESPECIFICACIÓN: GESTIÓN DE JUGADORES

A continuación se especifican todos aquellos contratos de los casos de uso relacionados con la gestión de jugadores dentro de la aplicación.

4.5.1.1. CREAR JUGADOR

- **Caso de uso:** Crear jugador
- **Actores:** Usuario (iniciador)
- **Propósito:** Crear una nuevo perfil de jugador
- **Resumen:** Se piden los datos necesarios, se comprueba su validez y se realiza la alta.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de crear un nuevo perfil de jugador.	
	2. El sistema solicita los datos necesarios para crear un perfil de jugador.
3. El usuario introduce los datos solicitados.	
	4. El sistema comprueba que no existe el perfil de jugador que el usuario desea crear.

5. Da de alta el nuevo perfil de jugador.

- **Cursos alternativos:**
  - **Línea 4:** Si ya existe un perfil de jugador con el mismo nombre que se ha introducido, el sistema muestra un error y vuelve a la pantalla de jugadores.

4.5.1.2. BORRAR JUGADOR

- **Caso de uso:** Borrar jugador
- **Actores:** Usuario (iniciador)
- **Propósito:** Borrar un perfil de jugador existente
- **Resumen:** Se realiza el borrado del perfil seleccionado.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de borrar un perfil de jugador entre los existentes.	
	2. El sistema borra todos los datos del perfil de jugador indicado.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.1.3. VER DETALLES JUGADOR

- **Caso de uso:** Ver detalles jugador
- **Actores:** Usuario (iniciador)
- **Propósito:** Ver los detalles del perfil de jugador existente
- **Resumen:** Se muestras todos los datos del perfil del jugador seleccionado.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden ver los detalles del perfil de jugador seleccionado entre los existentes.	

2. El sistema muestra al usuario todos los detalles del perfil de jugador indicado.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.2. ESPECIFICACIÓN: GESTIÓN DE DICCIONARIOS

A continuación se especifican todos aquellos contratos de los casos de uso relacionados con la gestión de diccionarios dentro de la aplicación.

4.5.2.1. DEFINIR POR OMISIÓN

- **Caso de uso:** Definir por defecto
- **Actores:** Usuario (iniciador)
- **Propósito:** Definir por defecto en el sistema un idioma para los diccionarios.
- **Resumen:** Se registra el idioma seleccionado como por defecto para futuras ejecuciones de la aplicación.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario activa la opción para establecer el idioma del diccionario seleccionado por defecto.	
2. El usuario da la orden de seleccionar diccionario.	
	3. El sistema registra el idioma seleccionado como idioma de diccionario por defecto.
	4. *Include*: Seleccionar diccionario (*Gestión de partidas*)

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.2.2. AÑADIR PALABRA

- **Caso de uso:** Añadir palabra
- **Actores:** Usuario (iniciador)
- **Propósito:** Añadir una nueva palabra válida en el diccionario actual.
- **Resumen:** Se añade como palabra válida en el diccionario actual la palabra indicada por el usuario.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario indica la nueva palabra que desea añadir al diccionario actual.	
2. El usuario da la orden de añadir la palabra al diccionario.	
	3. El sistema comprueba que no existe la palabra que el usuario desea añadir al diccionario.
	4. El sistema añade la palabra al diccionario actual.
	5. Si existe una partida en proceso, el sistema carga la nueva palabra al diccionario en memoria.

- **Cursos alternativos:**
  - **Línea 3:** Si ya existe la palabra en el diccionario actual, el sistema muestra un error y vuelve a la pantalla de diccionarios.

4.5.2.3. ELIMINAR PALABRA

- **Caso de uso:** Eliminar palabra
- **Actores:** Usuario (iniciador)
- **Propósito:** Eliminar una palabra válida del diccionario actual.
- **Resumen:** Se elimina como palabra válida del diccionario actual la palabra indicada por el usuario.
- **Tipo:** Primario, esencial

- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario indica la palabra que desea eliminar del diccionario actual.	
2. El usuario da la orden de eliminar la palabra al diccionario.	
	3. El sistema comprueba que existe la palabra que el usuario desea eliminar del diccionario.
	4. El sistema elimina la palabra del diccionario actual.
	5. Si existe una partida en proceso, el sistema descarga la palabra del diccionario en memoria.

- **Cursos alternativos:**
  - **Línea 3:** Si no existe la palabra en el diccionario actual, el sistema muestra un error y vuelve a la pantalla de diccionarios.

4.5.3. ESPECIFICACIÓN: GESTIÓN DE IDIOMAS

A continuación se especifican todos aquellos contratos de los casos de uso relacionados con la gestión de idiomas dentro de la aplicación.

4.5.3.1. SELECCIONAR IDIOMA

- **Caso de uso:** Seleccionar idioma
- **Actores:** Usuario (iniciador)
- **Propósito:** Seleccionar el idioma de la aplicación.
- **Resumen:** Seleccionar el idioma en que se visualizarán todos los menús, botones y textos de la aplicación entre los existentes.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de seleccionar el idioma indicado.	

2. El sistema carga todos los menús, botones y textos de la aplicación en el idioma seleccionado.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.3.2. DEFINIR POR OMISIÓN

- **Caso de uso:** Definir por defecto
- **Actores:** Usuario (iniciador)
- **Propósito:** Definir por defecto en el sistema un idioma para la aplicación.
- **Resumen:** Se registra el idioma seleccionado como por defecto para futuras ejecuciones de la aplicación.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario activa la opción para establecer el idioma de la aplicación seleccionado por defecto.	
2. El usuario da la orden de seleccionar idioma.	
	3. El sistema registra el idioma seleccionado como idioma de aplicación por defecto.
	4. *Include*: Seleccionar idioma

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.4. ESPECIFICACIÓN: GESTIÓN REGLAS

A continuación se especifican todos aquellos contratos de los casos de uso relacionados con la gestión de las reglas dentro de la aplicación.

4.5.4.1. MODIFICAR REGLAS

- **Caso de uso:** Modificar reglas
- **Actores:** Usuario (iniciador)
- **Propósito:** Establecer la configuración de las reglas (variaciones de juego).
- **Resumen:** Se registra la configuración de las reglas tal como las ha definido el usuario.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario configura todas las reglas tal y como desea.	
2. El usuario da la orden de modificar reglas.	
	3. El sistema carga la nueva configuración de reglas en la aplicación.
	4. Si el usuario ha activado establecer por defecto, el sistema registrará la nueva configuración de reglas para futuras ejecuciones de la aplicación.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.5. ESPECIFICACIÓN: GESTIÓN DE ESTADÍSTICAS

A continuación se especifican todos aquellos contratos de los casos de uso relacionados con la gestión de estadísticas dentro de la aplicación.

4.5.5.1. VER RANKING

- **Caso de uso:** Ver ranking
- **Actores:** Usuario (iniciador)
- **Propósito:** Visualizar el ranking entre los perfiles de jugadores existentes según diferentes conceptos.
- **Resumen:** Se muestra por pantalla un ranking completo de todos los perfiles de jugadores existentes según un concepto seleccionado entre los posibles.
- **Tipo:** Primario, esencial

- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de visualizar el ranking según uno de los posibles conceptos.	
	2. El sistema muestra un listado completo de todos los perfiles de jugadores existentes ordenados según el concepto seleccionado por el usuario.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6. ESPECIFICACIÓN: GESTIÓN DE PARTIDAS

A continuación se especifican todos aquellos contratos de los casos de uso relacionados con la gestión de partidas dentro de la aplicación.

4.5.6.1. CARGAR PARTIDA

- **Caso de uso:** Cargar partida
- **Actores:** Usuario (iniciador)
- **Propósito:** Cargar una partida guardada anteriormente.
- **Resumen:** Se carga una partida que fue guardada anteriormente pudiendo continuar en el mismo punto en que fue salvada.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de cargar partida.	
	2. El sistema muestra un listado de todas las partidas salvadas disponibles.
3. El usuario selecciona una de entre todas las partidas salvadas disponibles.	
	4. El sistema carga la partida seleccionada para poder continuar jugando ésta.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.2. SALVAR PARTIDA

- **Caso de uso:** Salvar partida
- **Actores:** Usuario (iniciador)
- **Propósito:** Guardar la situación actual de la partida actual.
- **Resumen:** Se salvan todos los datos necesarios para poder posteriormente cargar y continuar jugando dicha partida.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de salvar partida.	
	2. El sistema los datos necesarios para salvar una partida.
3. El usuario indica los datos necesarios para salvar la partida (nombre y ruta del fichero)	
	4. El sistema salva todos los datos necesarios de la partida en el fichero indicado por el usuario.

- **Cursos alternativos:**
  - **Línea 4:** Ya existe un fichero con el mismo nombre y ruta que el indicado por el usuario. El sistema muestra un error, y preguntará al usuario si desea sobrescribir el fichero existente por el nuevo.

4.5.6.3. SELECCIONAR JUGADOR

- **Caso de uso:** Seleccionar jugador
- **Actores:** Usuario (iniciador)
- **Propósito:** Seleccionar un jugador para la partida.
- **Resumen:** Se selecciona el jugador indicado para que forme parte de los jugadores de la partida.
- **Tipo:** Primario, esencial

- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de seleccionar el jugador seleccionado entre todos los perfiles de jugadores disponibles.	
	2. El sistema registra el jugador seleccionado como tal para la partida.

- **Cursos alternativos:**
  - **Línea 2:** Si ya existen 4 jugadores seleccionados el sistema ignorará esta acción por parte del usuario.

4.5.6.4. DESELECCIONAR JUGADOR

- **Caso de uso:** Deseleccionar jugador
- **Actores:** Usuario (iniciador)
- **Propósito:** Deseleccionar un jugador para de la partida.
- **Resumen:** Se deselecciona el jugador indicado para que no forme parte de los jugadores de la partida.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de deseleccionar el jugador seleccionado entre todos los perfiles de jugadores seleccionados.	
	2. El sistema registra el jugador seleccionado como no seleccionado para la partida.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.5. SELECCIONAR DICCIONARIO

- **Caso de uso:** Seleccionar diccionario
- **Actores:** Usuario (iniciador)
- **Propósito:** Seleccionar un diccionario para la partida.

- **Resumen:** Se selecciona el diccionario indicado para la partida.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de seleccionar el diccionario seleccionado entre los posibles.	
	2. El sistema registra el diccionario seleccionado como tal para la partida.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.6. REMOVER FICHAS

- **Caso de uso:** Remover fichas
- **Actores:** Usuario (iniciador)
- **Propósito:** Remover las fichas del atril del jugador actual.
- **Resumen:** Se permutan las fichas del atril del jugador de manera totalmente aleatoria.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de remover fichas.	
	2. El sistema permuta las fichas del usuario de manera aleatoria.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.7. CAMBIAR FICHAS

- **Caso de uso:** Cambiar fichas
- **Actores:** Usuario (iniciador)
- **Propósito:** Cambiar las fichas seleccionadas del atril del jugador actual.

- **Resumen:** Se cambian las fichas seleccionadas del atril de jugador actual por un número igual de fichas procedentes del saco.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de cambiar fichas.	
	2. El sistema muestra al usuario la pantalla de cambio de fichas con sus respectivas fichas.
3. El usuario selecciona aquellas fichas que desea cambiar.	
4. El usuario da la orden de cambiar.	
	5. El sistema retira del atril del usuario aquellas fichas seleccionadas.
	6. El sistema repone las fichas retiradas por un número igual de fichas procedentes del saco.
	7. Las fichas retiradas son devueltas al saco.
	8. *Include*: Siguiente turno.

- **Cursos alternativos:**
  - **Línea 3:** Si el usuario selecciona más fichas de las disponibles al saco el sistema no se lo permitirá.

4.5.6.8. DESHACER JUGADA

- **Caso de uso:** Deshacer jugada
- **Actores:** Usuario (iniciador)
- **Propósito:** Deshacer la jugada realizada.
- **Resumen:** Se deshará la jugada realizada devolviendo todas las fichas colocadas en el tablero en el turno actual por el usuario a su atril.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de deshacer jugada.	

2. El sistema devuelve todas las fichas colocadas en el tablero en el turno actual al atril del usuario.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.9. PASAR TURNO

- **Caso de uso:** Pasar turno
- **Actores:** Usuario (iniciador)
- **Propósito:** Pasar el turno del jugador actual.
- **Resumen:** Se pasará al siguiente turno del jugador actual sin que éste pueda realizar ninguna acción.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario o el tiempo dan la orden de pasar turno.	
	2. El sistema no permitirá realizar ninguna acción más al usuario durante este turno.
	3. *Include*: Siguiente turno.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.10. REALIZAR JUGADA (COLOCAR FICHA)

- **Caso de uso:** Colocar ficha
- **Actores:** Usuario (iniciador)
- **Propósito:** Colocar la ficha seleccionada del atril en el tablero.
- **Resumen:** Se coloca la ficha seleccionada en el atril del jugador actual en el tablero.
- **Tipo:** Primario, esencial

- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario desea colocar una ficha de su atril en el tablero.	
2. El usuario selecciona la ficha deseada de su atril.	.
3. El usuario indica en el tablero, la posición donde sea colocar la ficha seleccionada.	
	4. El sistema retira la ficha seleccionada del atril del usuario.
	5. El sistema registra la ficha seleccionada como colocada en el tablero en la posición indicada por el usuario.

- **Cursos alternativos:**
  - **Línea 3:** Si la posición indicada en el tablero ya es ocupada por otra ficha, el sistema no permitirá la colocación de la misma.

4.5.6.11. REALIZAR JUGADA (QUITAR FICHA)

- **Caso de uso:** Quitar ficha
- **Actores:** Usuario (iniciador)
- **Propósito:** Quitar la ficha seleccionada del tablero.
- **Resumen:** Se quita la ficha seleccionada del tablero colocada en el mismo turno actual por el jugador para ser devuelta a su atril.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario desea quitar una ficha colocada en el tablero en el turno actual.	
2. El usuario selecciona la ficha deseada del tablero.	
3. El usuario da la orden de quitar la ficha.	
	4. El sistema retira la ficha seleccionada del tablero.
	5. El sistema devuelve la ficha seleccionada al atril del usuario.

- **Cursos alternativos:**
  - **Línea 2:** Si la ficha seleccionada no le pertenece o no ha sido colocada por él en este turno actual el sistema no le permitirá la selección.

4.5.6.12. REALIZAR JUGADA (MOVER FICHA)

- **Caso de uso:** Mover ficha
- **Actores:** Usuario (iniciador)
- **Propósito:** Mover la ficha seleccionada del tablero de posición.
- **Resumen:** Se mueve la ficha seleccionada del tablero colocada en el mismo turno actual por el jugador para ser colocada en otra posición.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario desea mover una ficha colocada en el tablero en el turno actual.	
2. El usuario selecciona la ficha deseada del tablero.	
3. El usuario da la orden de mover la ficha.	
	4. El sistema retira la ficha seleccionada del tablero.
5. El usuario indica la nueva posición para la ficha en el tablero.	
	6. El sistema registra la ficha seleccionada como colocada en el tablero en la posición indicada por el usuario.

- **Cursos alternativos:**
  - **Línea 2:** Si la ficha seleccionada no le pertenece o no ha sido colocada por él en este turno actual el sistema no le permitirá la selección.
  - **Línea 5:** Si la posición indicada en el tablero ya es ocupada por otra ficha, el sistema no permitirá la colocación de la misma.

4.5.6.13. VALIDAR JUGADA

- **Caso de uso:** Validar jugada
- **Actores:** Usuario (iniciador)

- **Propósito:** Validar la jugada realizada.
- **Resumen:** Se validará la jugada realizada mostrando al jugador todas las palabras formadas y sus respectivas puntuaciones.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de validar la jugada actual.	
	2. El sistema comprueba que la colocación de las fichas sea válida.
	3. El sistema comprueba que todas las palabras formadas sean válidas según el diccionario de la partida.
	4. El sistema registra todas las nuevas palabras formadas, así como actualiza las puntuaciones.
	5. *Include*: Siguiente turno.

- **Cursos alternativos:**
  - **Línea 2:** Si la colocación de las fichas no es válida según las reglas de juego, el sistema muestra error y no procede a la validación continuando el jugador su turno.
  - **Línea 3:** Si alguna de las palabras formadas no es válida según el diccionario de la partida, el sistema muestra error así como la lista de todas aquellas palabras inválidas. El jugador continúa su turno sin proceder la validación.

4.5.6.14. RECICLAR COMODINES

- **Caso de uso:** Reciclar comodines
- **Actores:** Usuario (iniciador)
- **Propósito:** Reciclar un comodín colocado en el tablero.
- **Resumen:** Se cambiará el comodín colocado en el tablero, por la ficha del atril del jugador que representa la misma letra que el comodín.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de reciclar el comodín.	

2. El sistema intercambia la ficha del atril del usuario que representa la misma letra que el comodín, por éste mismo.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.15. FER SENYORS

- **Caso de uso:** Fer senyors
- **Actores:** Usuario (iniciador)
- **Propósito:** Cambiar todas las fichas del atril si son vocales o consonantes.
- **Resumen:** Se cambiarán todas las fichas del atril si estas son vocales o consonantes y en el saco quedan como mínimo el total de fichas por jugador.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de fer senyors.	
	2. El sistema retira del atril todas las fichas del usuario.
	3. El sistema repone el mismo número de fichas retiradas por fichas aleatorias procedentes del saco.
	4. El sistema devuelve las fichas retiradas del atril del usuario al saco.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.16. SIGUIENTE TURNO

- **Caso de uso:** Siguiente turno
- **Actores:** -
- **Propósito:** Avanzar al turno del siguiente jugador.
- **Resumen:** Se avanza el turno al turno del siguiente jugador.
- **Tipo:** Primario, esencial

- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
	1. El sistema avanza el turno actual al turno del siguiente jugador de la partida.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.17. VER LOG

- **Caso de uso:** Ver log
- **Actores:** Usuario (iniciador)
- **Propósito:** Ver el log de la partida actual.
- **Resumen:** Se visualizará el log de la partida actual.
- **Tipo:** Primario, esencial
- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de ver el log de la partida actual.	
	2. El sistema muestra al usuario el log en el estado en que se encuentra la partida actual.

- **Cursos alternativos:**
  - No existen cursos alternativos.

4.5.6.18. SALVAR LOG

- **Caso de uso:** Reciclar comodines
- **Actores:** Usuario (iniciador)
- **Propósito:** Reciclar un comodín colocado en el tablero.
- **Resumen:** Se cambiará el comodín colocado en el tablero, por la ficha del atril del jugador que representa la misma letra que el comodín.
- **Tipo:** Primario, esencial

- **Curso típico de sucesos:**

Acciones de los actores	Respuesta del sistema
1. El usuario da la orden de salvar el log de la partida actual.	
	2. El sistema solicita los datos necesarios para salvar el log de la partida actual.
3. El usuario indica los datos necesarios para salvar el log de la partida actual (nombre y ruta del fichero).	
	4. El sistema salva el log de la partida actual en el fichero indicado por el usuario.

- **Cursos alternativos:**
  - **Línea 4:** Ya existe un fichero con el mismo nombre y ruta que el indicado por el usuario. El sistema muestra un error, y preguntará al usuario si desea sobrescribir el fichero existente por el nuevo.

4.6. MODELO DE COMPORTAMIENTO

El modelo de comportamiento del sistema es una visión del sistema que describe las interacciones entre los elementos del sistema y la evolución del estado del sistema y sus componentes. Se define mediante los diagramas de secuencia y el contrato de las operaciones.

4.6.1. DIAGRAMAS DE SECUENCIA

Los diagramas de secuencia son aquellos que nos muestran las interacciones entre objetos, tanto cuando se produce un suceso externo como cuando se invoca una operación de un objeto. Todo esto lo hacen describiendo el orden temporal en el cual se producen las invocaciones de las operaciones.

En UML, los diagramas de secuencia se representan de la siguiente manera:

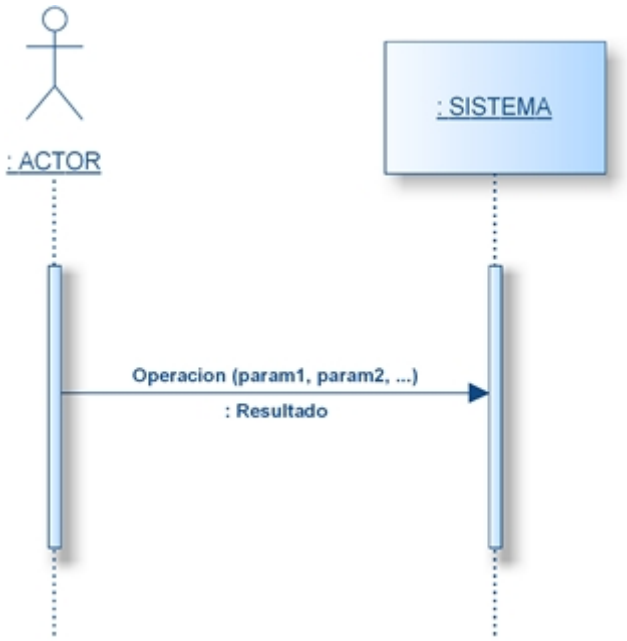


Figura 4.14 Representación genérica de un diagrama de secuencia en *UML*.

Para representar los diagramas de secuencia tenemos el usuario que es el actor que realiza las diferentes llamadas al sistema. El sistema es quien tiene todas las operaciones. Las operaciones se representan con un nombre situado encima de la fecha que comunica al usuario y al sistema. Éstas pueden tener parámetros, representados entren paréntesis, y, a veces las operaciones devuelven un resultado, este suceso se representa mediante dos puntos seguidos de la operación y el valor que retornan.

4.6.1.1. DIAGRAMAS: GESTIÓN DE JUGADORES

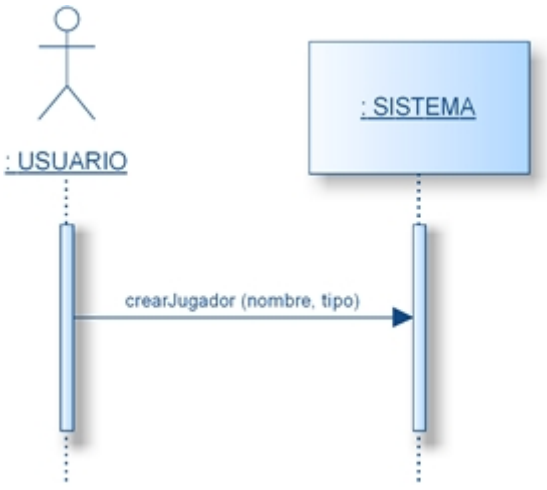


Figura 4.15 Crear jugador

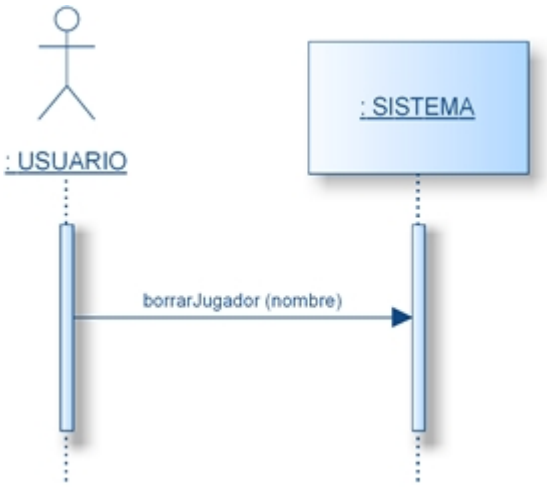


Figura 4.16 Borrar jugador

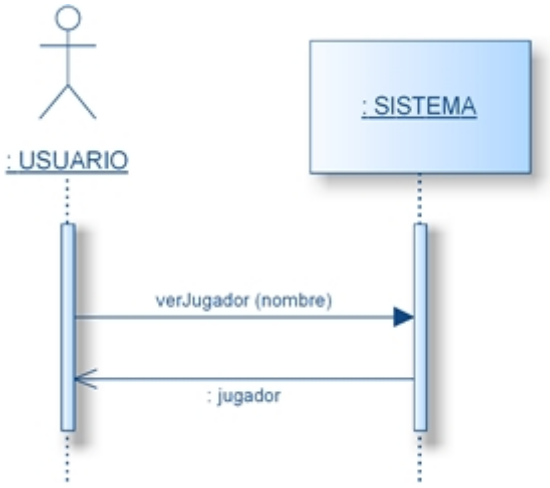


Figura 4.17 Ver detalles jugador

4.6.1.2. DIAGRAMAS: GESTIÓN DE DICCIONARIOS

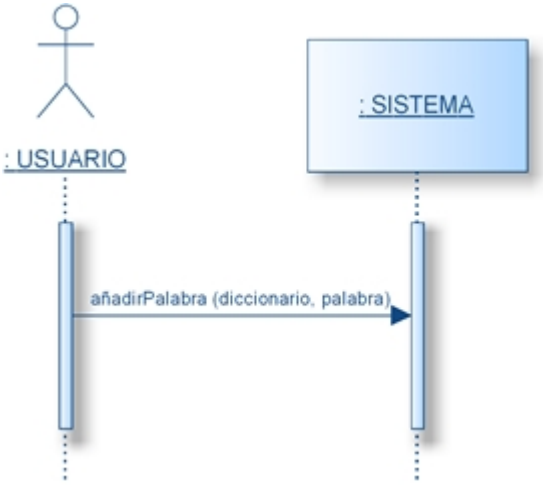


Figura 4.18 Añadir palabra

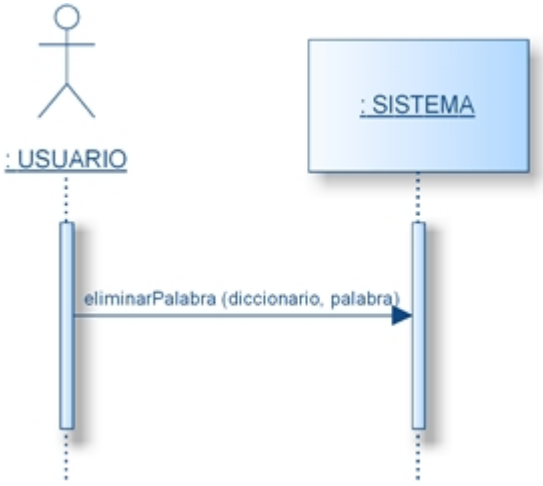


Figura 4.19 Eliminar palabra

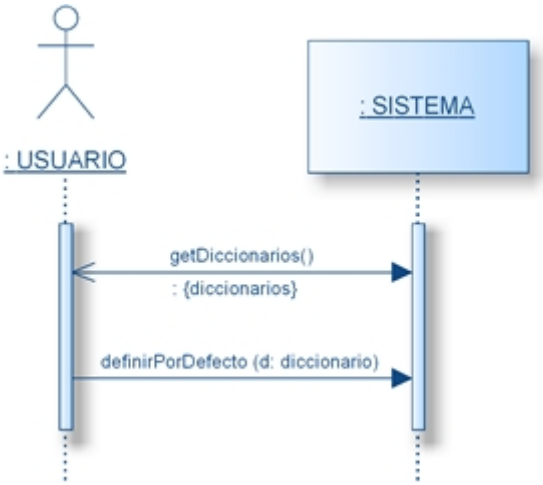


Figura 4.20 Definir por defecto (diccionario)

4.6.1.3. DIAGRAMAS: GESTIÓN DE IDIOMAS

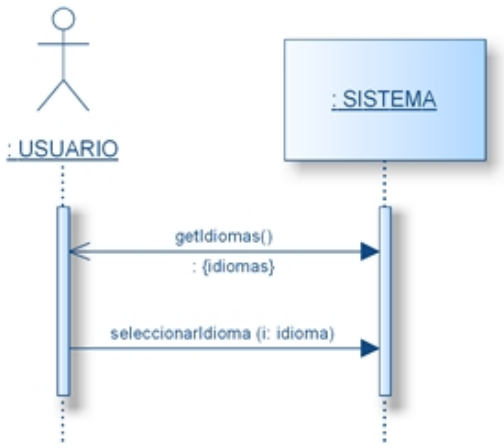


Figura 4.21 Seleccionar idioma

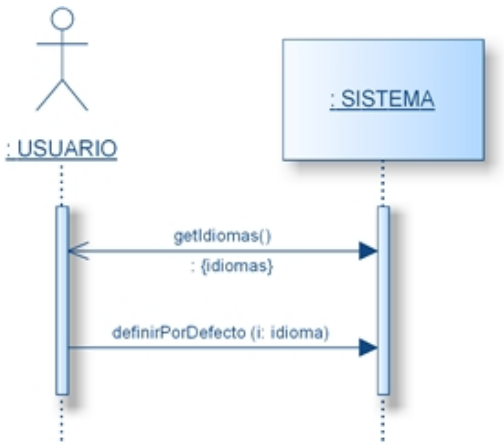


Figura 4.22 Definir por defecto

4.6.1.4. DIAGRAMAS: GESTIÓN DE REGLAS

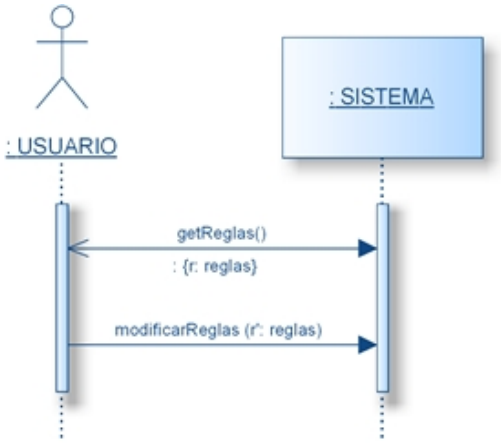


Figura 4.23 Modificar reglas

4.6.1.5. DIAGRAMAS: GESTIÓN DE ESTADÍSTICAS

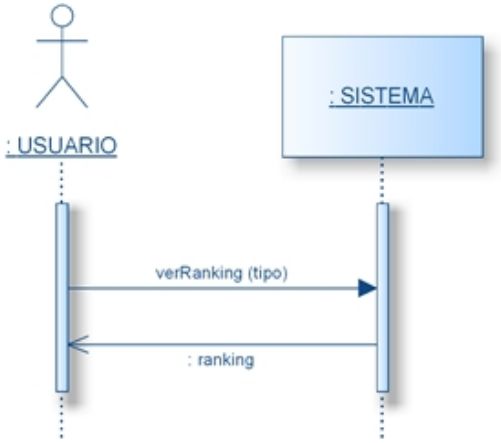


Figura 4.24 Ver ranking

4.6.1.6. DIAGRAMAS: GESTIÓN DE PARTIDAS

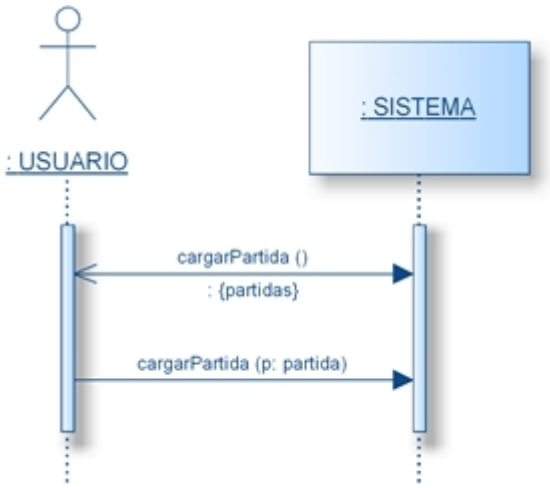


Figura 4.25 Cargar partida

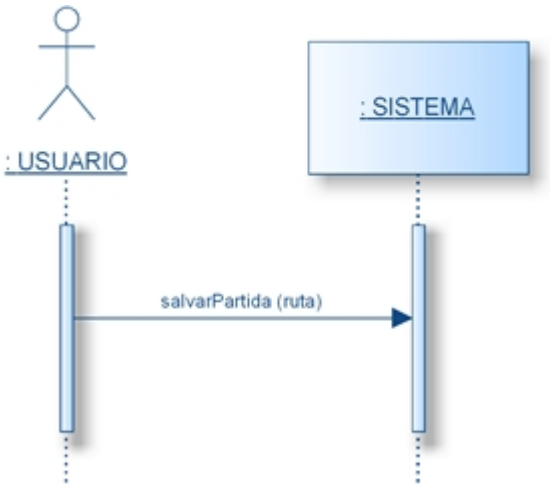


Figura 4.26 Salvar partida

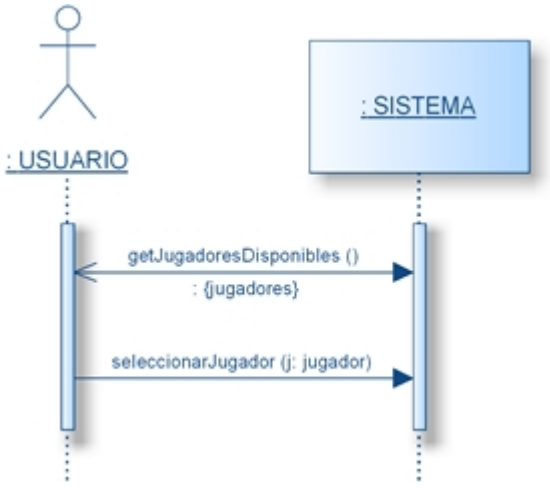


Figura 4.27 Seleccionar jugador

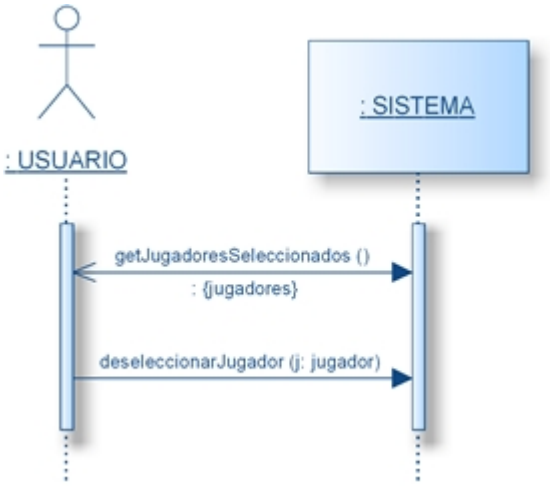


Figura 4.28 Deseleccionar jugador

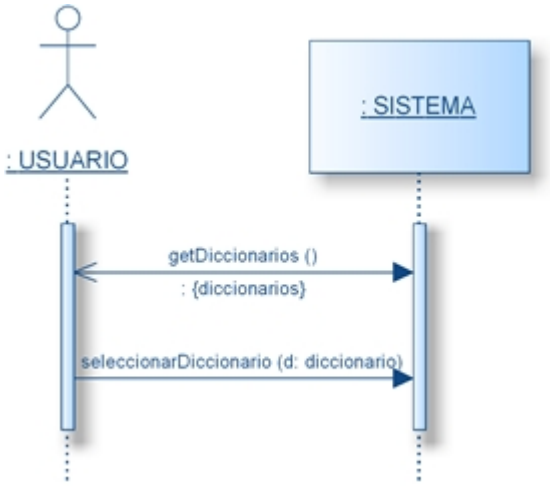


Figura 4.29 Seleccionar diccionario

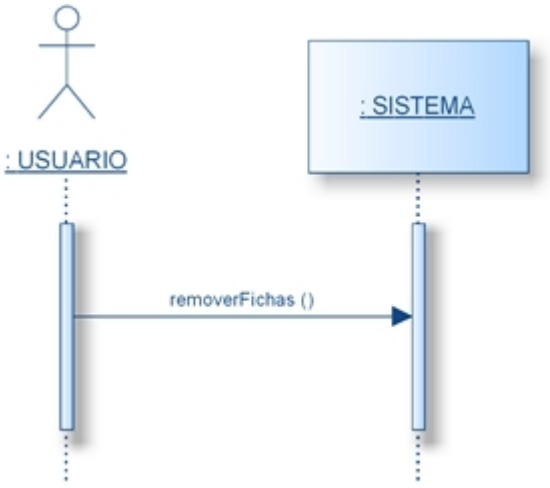


Figura 4.30 Remover fichas

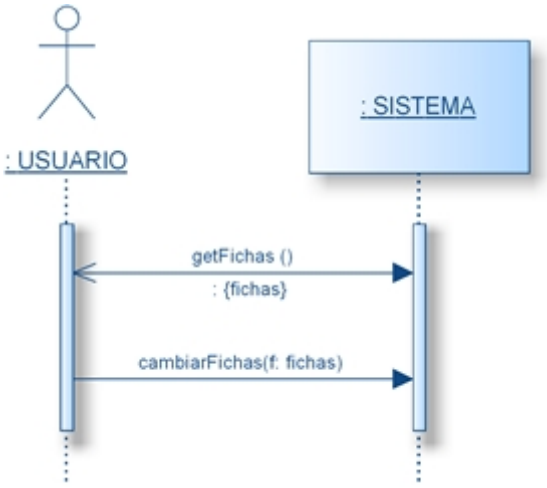


Figura 4.31 Cambiar fichas

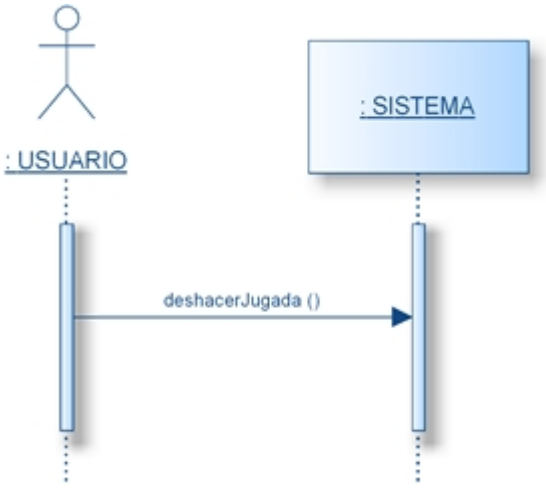


Figura 4.32 Deshacer jugada

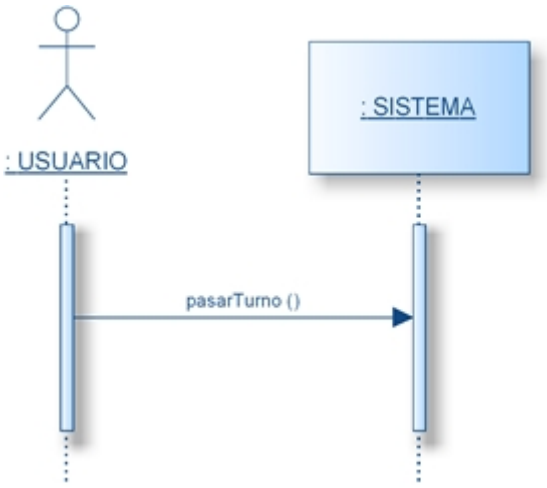


Figura 4.33 Pasar turno

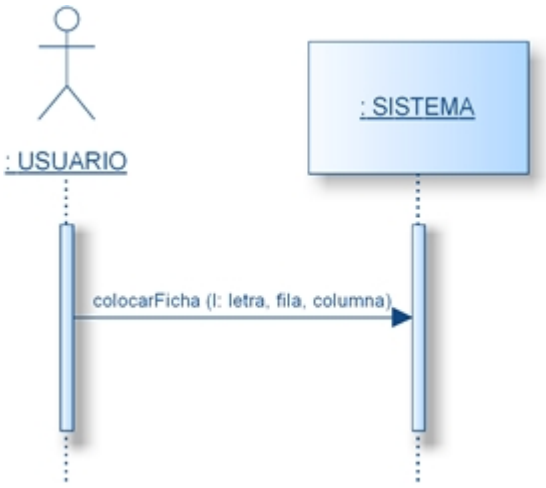


Figura 4.34 Colocar ficha

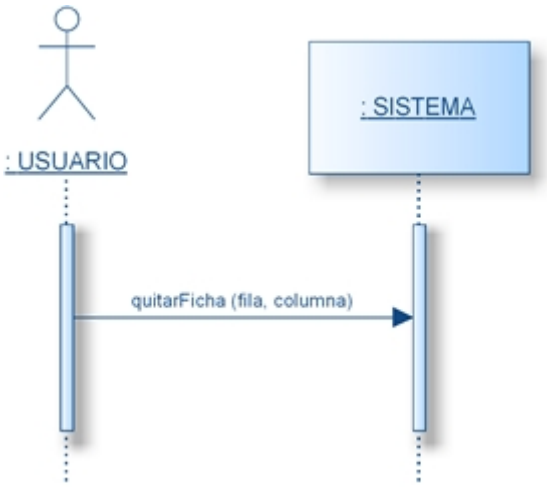


Figura 4.35 Quitar ficha

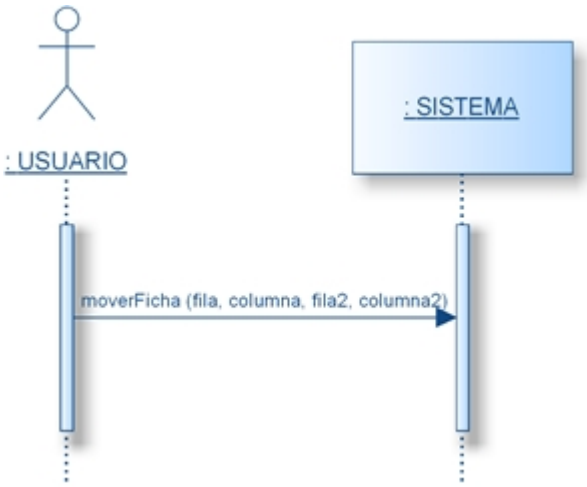


Figura 4.36 Mover ficha

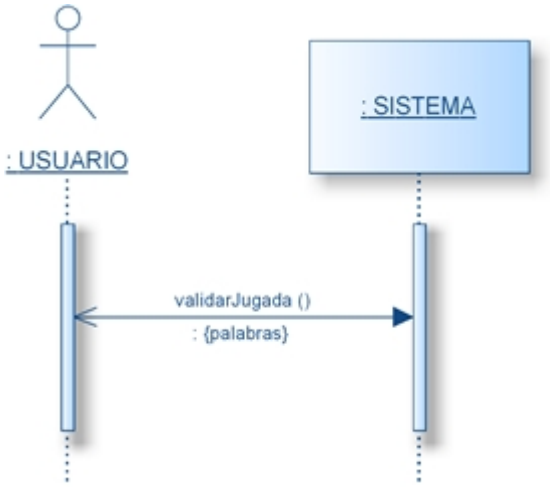


Figura 4.37 Validar jugada

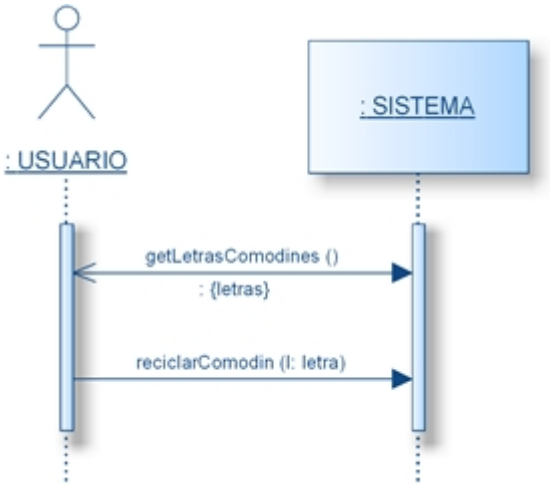


Figura 4.38 Reciclar comodines

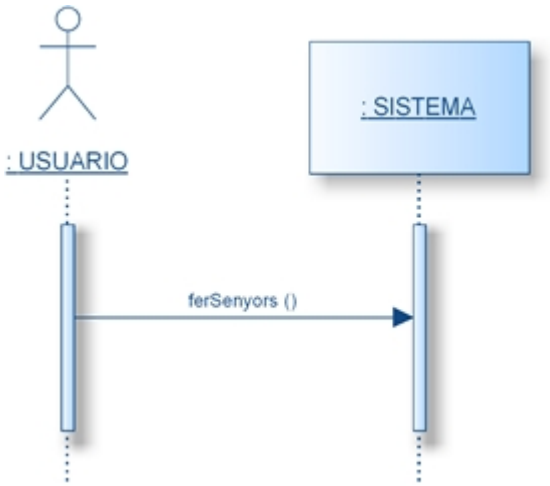


Figura 4.39 Fer senyors

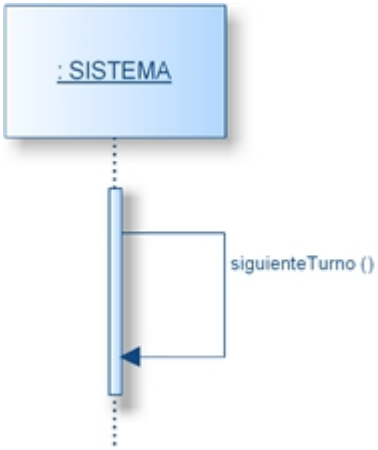


Figura 4.40 Siguiente turno

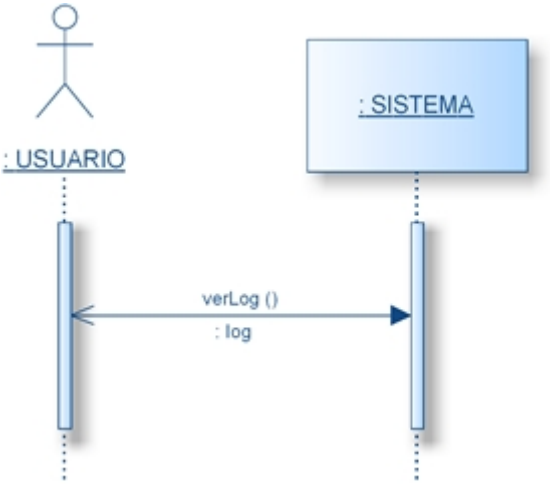


Figura 4.41 Ver log

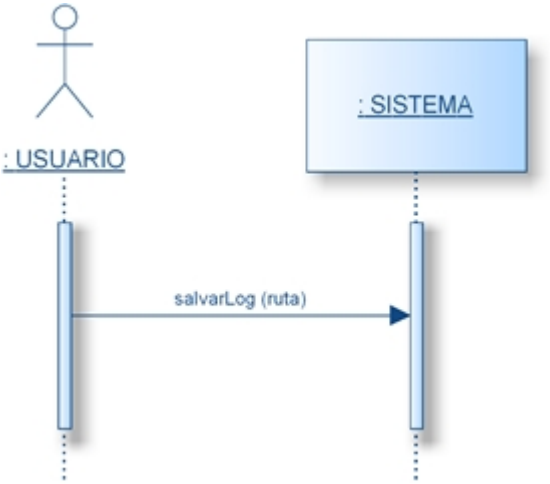


Figura 4.42 Salvar log

4.6.2. CONTRATOS DE LAS OPERACIONES

Los contratos de las operaciones describen el comportamiento del sistema en términos de cuales son los cambios de estado de información y cuales son las salidas que el sistema proporciona cuando se invoca a la operación. Incluyen precondiciones y poscondiciones que describen los cambios de estado y salidas. El tipo de descripción es declarativo, por tanto, el énfasis se ponen en lo qué hará la operación más que en como lo hará. Cada diagrama de secuencia ha permitido definir una serie de operaciones. En este apartado veremos los contratos de estas operaciones.

4.6.2.1. CONTRATOS: GESTIÓN DE JUGADORES

- **crearJugador (*nombre, tipo*)**

  Nombre: crearJugador (*nombre*, *tipo*)
  Responsabilidades: Crear un nuevo perfil de jugador
  Caso de uso: Crear jugador
  Excepciones: Si ya existe un jugador con el mismo nombre.
  Precondiciones: -
  Poscondiciones: Existe un nuevo jugador con nombre *nombre* y tipo *tipo*.
  Salida: -

- **borrarJugador (*nombre*)**

  Nombre: borrarJugador (*nombre*)
  Responsabilidades: Borrar el perfil de un jugador
  Caso de uso: Borrar jugador
  Excepciones: -
  Precondiciones: Existe el jugador con nombre *nombre*.
  Poscondiciones: No existe ningún jugador con nombre *nombre*.
  Salida: -

- **verJugador (*nombre*)**

  Nombre: verJugador (*nombre*)
  Responsabilidades: Mostrar toda la información sobre un jugador
  Caso de uso: Ver detalles jugador

Excepciones: -

Precondiciones: Existe el jugador con nombre *nombre*.

Poscondiciones: -

Salida: Toda la información sobre el jugador con nombre *nombre*.

4.6.2.2. CONTRATOS: GESTIÓN DE DICCIONARIOS

- **añadirPalabra (*diccionario, palabra*)**

  Nombre: añadirPalabra (*diccionario, palabra*)

  Responsabilidades: Añadir una nueva palabra válida a un diccionario

  Caso de uso: Añadir palabra

  Excepciones: Si ya existe la palabra *palabra* en el diccionario *diccionario*.

  Precondiciones: Existe un diccionario *diccionario*.

  Poscondiciones: La palabra *palabra* está incluida en el diccionario *diccionario*.

  Salida: -

- **eliminarPalabra (*diccionario, palabra*)**

  Nombre: eliminarPalabra (*diccionario, palabra*)

  Responsabilidades: Eliminar una palabra válida de un diccionario

  Caso de uso: Eliminar palabra

  Excepciones: Si no existe la palabra *palabra* en el diccionario *diccionario*.

  Precondiciones: Existe un diccionario *diccionario*.

  Poscondiciones: La palabra *palabra no* está incluida en el diccionario *diccionario*.

  Salida: -

- **getDiccionarios ()**

  Nombre: getDiccionarios ()

  Responsabilidades: Obtener una lista de los diccionarios disponibles

  Caso de uso: Seleccionar diccionario / Definir por defecto

  Excepciones: -

  Precondiciones: -

  Poscondiciones: -

  Salida: Una lista de los diccionarios disponibles en la aplicación.

- **definirPorOmision (*diccionario*)**

  Nombre: definirPorDefecto (*diccionario*)
  Responsabilidades: Definir un diccionario por defecto
  Caso de uso: Definir por defecto
  Excepciones: -
  Precondiciones: Existe un diccionario *diccionario*.
  Poscondiciones: El diccionario *diccionario* es configurado por defecto para próximas ejecuciones de la aplicación.
  Salida: -

4.6.2.3. CONTRATOS: GESTIÓN DE IDIOMAS

- **getIdiomas ()**

  Nombre: getIdiomas ()
  Responsabilidades: Obtener una lista de los idiomas disponibles
  Caso de uso: Seleccionar idioma / Definir por defecto
  Excepciones: -
  Precondiciones: -
  Poscondiciones: -
  Salida: Una lista de los idiomas disponibles en la aplicación.

- **seleccionarIdioma (*idioma*)**

  Nombre: seleccionarIdioma (*idioma*)
  Responsabilidades: Seleccionar un idioma para la aplicación
  Caso de uso: Seleccionar idioma
  Excepciones: -
  Precondiciones: Existe un idioma *idioma*.
  Poscondiciones: La aplicación es visualizada en el idioma *idioma*.
  Salida: -

- **definirPorOmision (*idioma*)**

  Nombre: definirPorDefecto (*idioma*)
  Responsabilidades: Definir un idioma por defecto

Caso de uso: Definir por defecto

Excepciones: -

Precondiciones: Existe un idioma *idioma*.

Poscondiciones: El idioma *idioma* es configurado por defecto para próximas ejecuciones de la aplicación.

Salida: -

4.6.2.4. CONTRATOS: GESTIÓN DE REGLAS

- **getReglas ()**

  Nombre: getReglas ()

  Responsabilidades: Obtener las reglas de la aplicación

  Caso de uso: Modificar reglas

  Excepciones: -

  Precondiciones: -

  Poscondiciones: -

  Salida: La configuración de reglas de la aplicación.

- **modificarReglas (*reglas*)**

  Nombre: modificarReglas (*reglas*)

  Responsabilidades: Configurar las reglas de la aplicación

  Caso de uso: Modificar reglas

  Excepciones: -

  Precondiciones: -

  Poscondiciones: Las reglas de la aplicación quedan configuradas tal como *reglas*.

  Salida: -

4.6.2.5. CONTRATOS: GESTIÓN DE ESTADÍSTICAS

- **verRanking (*tipo*)**

  Nombre: verRanking (*tipo*)

  Responsabilidades: Mostrar un tipo de ranking

  Caso de uso: Ver ranking

  Excepciones: -

Precondiciones: -

Poscondiciones: -

Salida: Un listado de perfiles de jugadores ordenados según el tipo de ranking *tipo*.

4.6.2.6. CONTRATOS: GESTIÓN DE PARTIDAS

- **cargarPartida ()**

  Nombre: cargarPartida ()

  Responsabilidades: Cargar una partida salvada

  Caso de uso: Cargar partida

  Excepciones: -

  Precondiciones: -

  Poscondiciones: La partida seleccionada es cargada en memoria.

  Salida: -

- **salvarPartida (*ruta*)**

  Nombre: salvarPartida (ruta)

  Responsabilidades: Salvar una partida

  Caso de uso: Salvar partida

  Excepciones: -

  Precondiciones: Existe una partida en curso.

  Poscondiciones: La partida en juego es salvada en el fichero con path *ruta*.

  Salida: Un fichero de texto que contiene todos los datos necesarios para salvar una partida para que pueda ser cargada posteriormente.

- **getJugadoresDisponibles ()**

  Nombre: getJugadoresDisponibles ()

  Responsabilidades: Obtener una lista de los jugadores disponibles

  Caso de uso: Seleccionar jugador

  Excepciones: -

  Precondiciones: -

  Poscondiciones: -

  Salida: Listado de todos los perfiles de jugadores disponibles.

- **seleccionarJugador(*jugador*)**

  Nombre: seleccionarJugador (*jugador*)
  Responsabilidades: Seleccionar un jugador para la partida
  Caso de uso: Seleccionar jugador
  Excepciones: -
  Precondiciones: Existe el jugador disponible *jugador*.
  Poscondiciones: El jugador *jugador* es seleccionado para la partida.
  Salida: -

- **getJugadoresSeleccionados ()**

  Nombre: getJugadoresSeleccionados ()
  Responsabilidades: Obtener una lista de los jugadores seleccionados
  Caso de uso: Deseleccionar jugador
  Excepciones: -
  Precondiciones: -
  Poscondiciones: -
  Salida: Listado de todos los perfiles de jugadores seleccionados.

- **deseleccionarJugador(*jugador*)**

  Nombre: deseleccionarJugador (*jugador*)
  Responsabilidades: Deseleccionar un jugador de la partida
  Caso de uso: Deseleccionar jugador
  Excepciones: -
  Precondiciones: Existe el jugador seleccionado *jugador*.
  Poscondiciones: El jugador *jugador* no es seleccionado para la partida.
  Salida: -

- **seleccionarDiccionario(*diccionario*)**

  Nombre: seleccionarDiccionario (*diccionario*)
  Responsabilidades: Seleccionar un diccionario para la partida
  Caso de uso: Seleccionar diccionario
  Excepciones: -
  Precondiciones: Existe el diccionario *diccionario*.
  Poscondiciones: El diccionario *diccionario* es seleccionado para la partida.
  Salida: -

- **removerFichas()**

  Nombre: removerFichas ()
  Responsabilidades: Remover las fichas del jugador actual
  Caso de uso: Remover fichas
  Excepciones: -
  Precondiciones: Existe una partida en curso.
  Poscondiciones: Las fichas del jugador actual son removidas aleatoriamente.
  Salida: -

- **getFichas()**

  Nombre: getFichas ()
  Responsabilidades: Obtener las fichas del jugador actual
  Caso de uso: Cambiar fichas
  Excepciones: -
  Precondiciones: Existe una partida en curso.
  Poscondiciones: El jugador *jugador* es seleccionado para la partida.
  Salida: -

- **cambiarFichas(*fichas*)**

  Nombre: cambiarFichas (*fichas*)
  Responsabilidades: Cambiar ciertas fichas concretas del jugador actual
  Caso de uso: Cambiar fichas
  Excepciones: -
  Precondiciones: Existen las fichas *fichas* en el atril del jugador actual.
  Poscondiciones: Las fichas *fichas* del atril del jugador actual han sido reemplazadas por el mismo número de fichas procedentes del saco aleatoriamente.
  Salida: -

- **deshacerJugada()**

  Nombre: deshacerJugada ()
  Responsabilidades: Deshacer la jugada realizada hasta ahora por el jugador actual
  Caso de uso: Deshacer jugada
  Excepciones: -
  Precondiciones: Existe una partida en curso.

Poscondiciones: La situación vuelve a la del inicio del turno del jugador actual.
Salida: -

- **pasarTurno()**

  Nombre: pasarTurno ()
  Responsabilidades: Pasar el turno del jugador actual
  Caso de uso: Pasar turno
  Excepciones: -
  Precondiciones: Existe una partida en curso.
  Poscondiciones: El jugador actual no realiza ninguna acción más y cede el siguiente turno.
  Salida: -

- **colocarFicha(*ficha, fila, columna*)**

  Nombre: colocarFicha (*ficha, fila, columna*)
  Responsabilidades: Colocar una ficha del atril al tablero
  Caso de uso: Colocar ficha
  Excepciones: -
  Precondiciones: Existe una partida en curso.
  Poscondiciones: Se coloca una ficha del atril del jugador actual en la casilla (*fila, columna*) del tablero.
  Salida: -

- **quitarFicha(*fila, columna*)**

  Nombre: colocarFicha (*fila, columna*)
  Responsabilidades: Quitar una ficha del tablero al atril
  Caso de uso: Quitar ficha
  Excepciones: -
  Precondiciones: Existe una ficha en la casilla (*fila, columna*) del tablero.
  Poscondiciones: Se retira la ficha de la casilla (*fila, columna*) del tablero al atril del jugador actual.
  Salida: -

- **moverFicha(*fila, columna, fila2, columna2*)**

  Nombre: moverFicha (*fila*, *columna*, *fila2*, *columna2*)
  Responsabilidades: Mover una ficha de una casilla a otra del tablero
  Caso de uso: Mover ficha
  Excepciones: -
  Precondiciones: Existe una ficha en la casilla (*fila*, *columna*) del tablero. No existe ninguna ficha en la casilla (*fila2*, *columna2*) del tablero.
  Poscondiciones: Se mueve la ficha de la casilla (*fila*, *columna*) del tablero a la casilla (*fila2*, *columna2*) del tablero.
  Salida: -

- **validarJugada()**

  Nombre: validarJugada ()
  Responsabilidades: Validar la jugada realizada en el tablero
  Caso de uso: Validar jugada
  Excepciones: Las fichas colocadas en el tablero no lo están de forma correcta. O si hay formada alguna palabra no válida según el diccionario actual.
  Precondiciones: -
  Poscondiciones: Las fichas colocadas en el tablero que forman la jugada pasan a ser parte del tablero.
  Salida: Lista de palabras válidas formadas por la jugada realizada, así como sus respectivas puntuaciones.

- **getLetrasComodines()**

  Nombre: getLetrasComodines ()
  Responsabilidades: Obtener las letras que representan los comodines jugados
  Caso de uso: Reciclar comodines
  Excepciones: -
  Precondiciones: -
  Poscondiciones: -
  Salida: Lista de las letras que representan los comodines jugados en el tablero.

- **reciclarComodin(*letra*)**

  Nombre: reciclarComodin (*letra*)

Responsabilidades: Intercambiar la ficha de un comodín por su respectiva letra del atril
Caso de uso: Reciclar comodines
Excepciones: -
Precondiciones: Existe un comodín jugado en el tablero que representa la letra *letra*.
Poscondiciones: Se intercambia la ficha del comodín jugado en el tablero que representa la letra *letra*, por una ficha del atril del jugador actual con letra *letra*.
Salida: -

- **ferSenyors()**

Nombre: ferSenyors ()
Responsabilidades: Cambiar todas las fichas del atril
Caso de uso: Fer senyors
Excepciones: -
Precondiciones: Las fichas del atril son todas vocales y comodines, o todas consonantes y comodines. Hay como mínimo el mismo número de fichas en el saco.
Poscondiciones: Se intercambia todas las fichas del jugador actual por fichas del saco.
Salida: -

- **siguienteTurno()**

Nombre: siguienteTurno ()
Responsabilidades: Pasar el turno al siguiente jugador de la partida
Caso de uso: Siguiente turno
Excepciones: -
Precondiciones: -
Poscondiciones: El turno actual corresponde al siguiente jugador de la partida.
Salida: -

- **verLog()**

Nombre: verLog ()
Responsabilidades: Ver el log desde el inicio al turno actual de la partida actual
Excepciones: -
Precondiciones: -
Poscondiciones: -
Salida: Listado de todos los turnos de la partida actual con toda la información sobre cada uno de ellos.

- **salvarLog(*ruta*)**

  Nombre: salvarLog (ruta)
  Responsabilidades: Salvar el log de la partida actual
  Caso de uso: Salvar log
  Excepciones: -
  Precondiciones: Existe una partida recién finalizada.
  Poscondiciones: El log de la partida finalizada es salvado en el fichero con path *ruta*.
  Salida: Un fichero de texto que contiene todos los datos necesarios para salvar un log partida para que pueda ser visualizado posteriormente.

4.7. MODELO DE DATOS

El objetivo del modelo de estados es crear diagramas de estado para objetos y casos de uso, mediante:

- Sucesos: todo aquello que requiere una respuesta del sistema *software*.
- Estados: condición de un objeto o de un caso de uso en un momento del tiempo.
- Transiciones: cambio de estado como consecuencia de un suceso.

4.7.1. DIAGRAMA DE ESTADO DE CLASES

Primero especificaremos en detalle los diagramas de estado de las clases de objetos que pueden variar su estado durante el transcurso de ejecución de la aplicación. Esta especificación nos puede resultar útil para:

- Describir porque los objetos cambian de subclase.
- Las subclases de un diagrama de estados no tienen porque aparecer explícitamente en el esquema conceptual.
- Describir clases de objetos con un importante comportamiento dinámico.

4.7.1.1. DIAGRAMA: FICHA

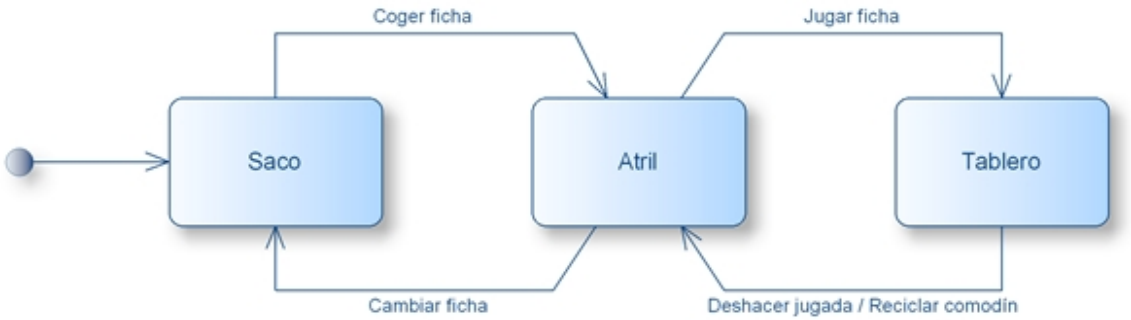


Figura 4.43 Diagrama de estados de la clase Ficha

4.7.1.2. DIAGRAMA: CASILLA

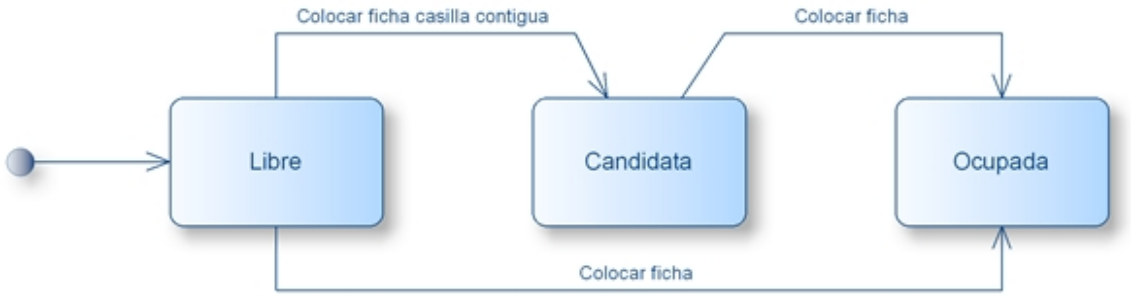


Figura 4.44 Diagrama de estados de la clase Casilla

4.7.1.3. DIAGRAMA: JUGADOR

Figura 4.45 Diagrama de estados de la clase Jugador

4.7.2. DIAGRAMA DE ESTADOS DE CASOS DE USO

El diagrama de estados del caso de uso describe la secuencia legal con la que los sucesos se pueden producir en el mundo real.

A continuación, definiremos los diagramas de estado de las altas, bajas, modificaciones y vistas de manera genérica ya que en cuanto a comportamiento en términos de objeto se comportan de forma similar. Estas altas, bajas, modificaciones y vistas genéricas, en nuestra aplicación podrían traducirse como altas y bajas de jugadores, modificaciones de reglas, añadir y eliminar palabras válidas en los diccionarios, etc…

4.7.2.1. DIAGRAMA: ALTA GENÉRICA

Figura 4.46 Diagrama de estados del caso de uso genérico Alta.

4.7.2.2. DIAGRAMA: BORRAR GENÉRICO

Figura 4.47 Diagrama de estados del caso de uso genérico Borrar.

4.7.2.3. DIAGRAMA: VER GENÉRICO

Figura 4.48 Diagrama de estados del caso de uso genérico Ver.

4.7.2.4. DIAGRAMA: MODIFICAR GENÉRICO

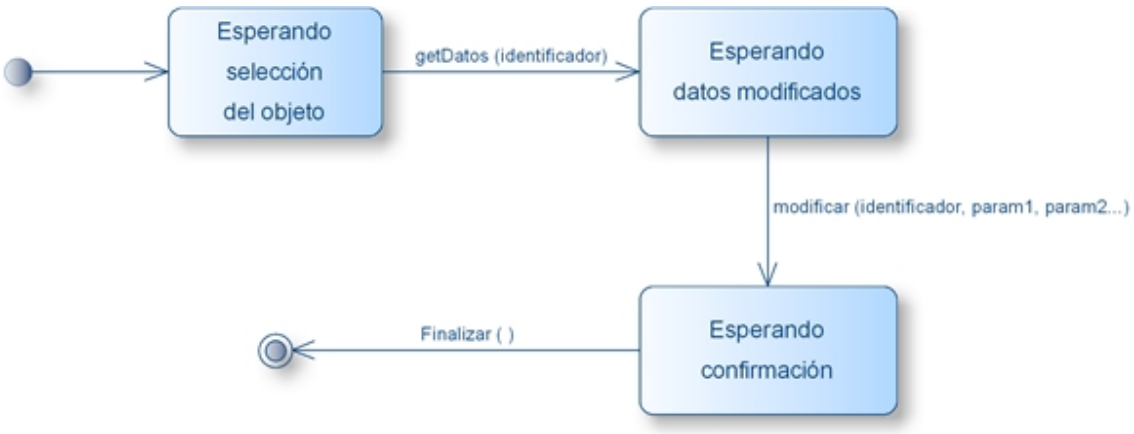


Figura 4.49 Diagrama de estados del caso de uso genérico Modificar.

4.7.2.5. DIAGRAMAS PRINCIPALES

Especificaremos a continuación algunos de los posibles casos de uso que pueden tener lugar a lo largo de la ejecución de la aplicación. Como casos de uso más importantes para este apartado, se han considerado aquellos que tienen relación directa con el transcurso de una partida, como pueden ser la preparación previa de la configuración de la misma, el transcurso de los turnos de los jugadores, y como bloque principal todas las funcionalidades que puede llevar a cabo un usuario ya sea de tipo *Humano* o *Robot* en el transcurso de su propio turno.

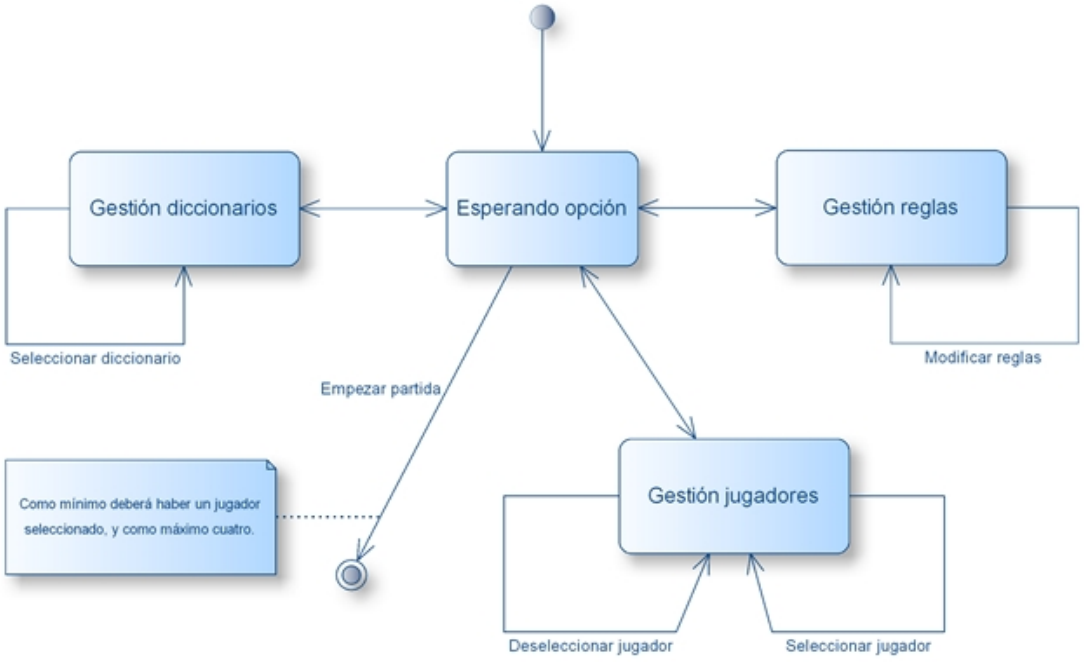


Figura 4.50 Diagrama de estados que referencia a la configuración de una partida.

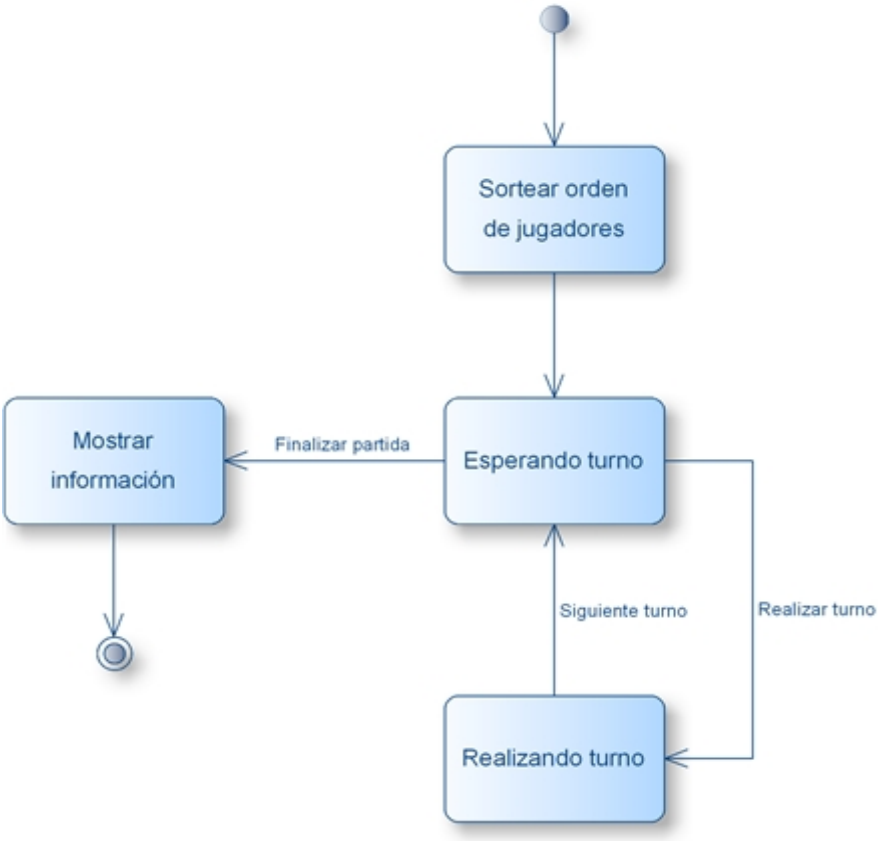


Figura 4.51 Diagrama de estados que referencia al transcurso de turnos en una partida.

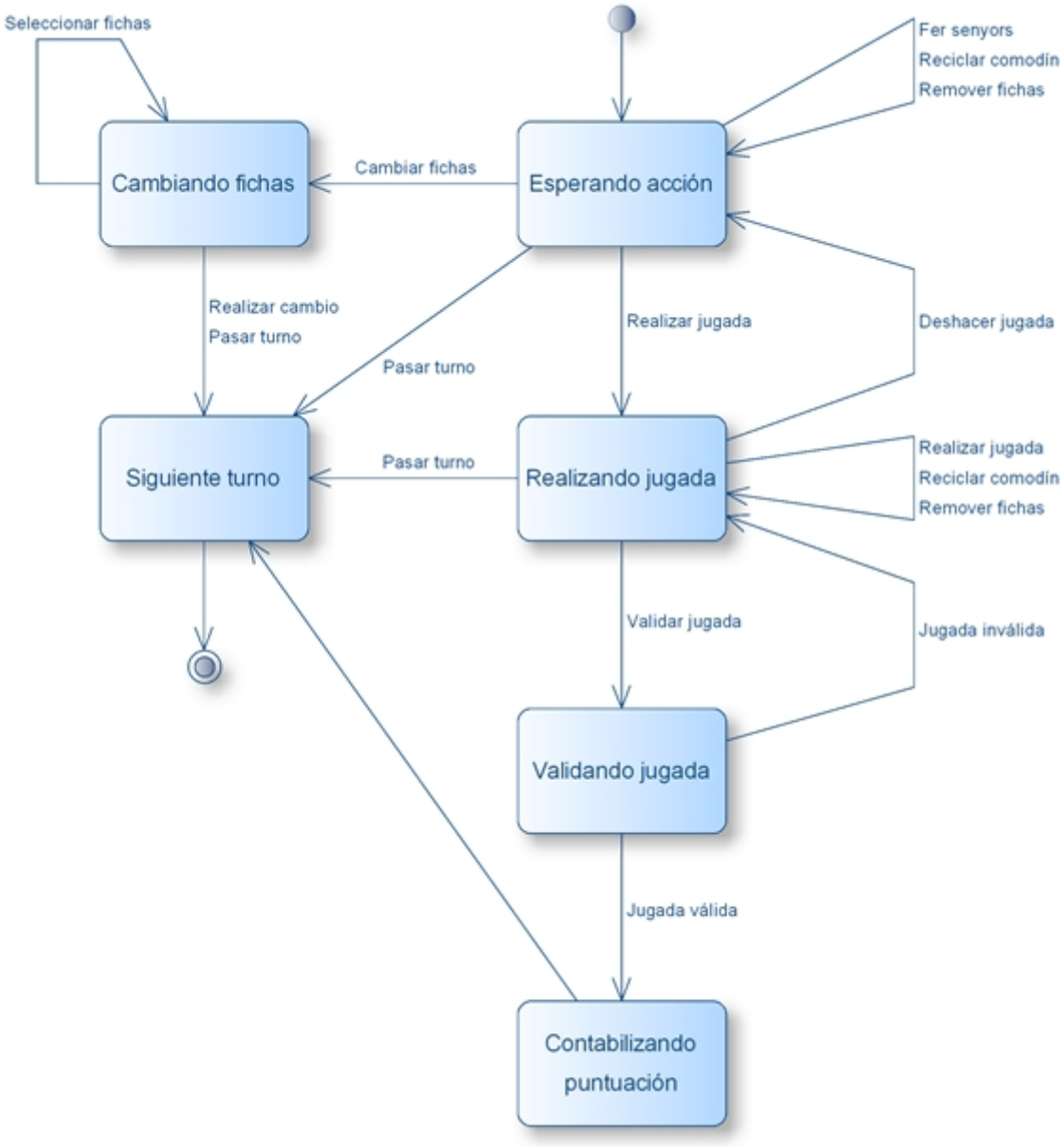


Figura 4.52 Diagrama de estados que referencia la realización de un turno.

5. DISEÑO

5.1. Introducción 119
5.2. Diseño de la arquitectura 119

5.2.1. Patrones arquitectónicos 121
5.2.2. Arquitectura escogida 121

5.3. Capa de presentación 125

5.3.1. Diagrama de presentación 126
5.3.2. Diseño de las pantallas 127
5.3.3. Componentes 131

5.4. Capa de datos 132

5.4.1. Descripción de los ficheros 133
5.4.2. Diagrama datos 135

5.5. Capa de dominio 135

5.5.1. Normalización diagrama 136
5.5.2. Patrón diseño: Controladores 137
5.5.3. Descripción de controladores 138
5.5.4. Diagrama de dominio 142
5.5.5. Diagramas de secuencia 144

5.1. INTRODUCCIÓN

En el capítulo anterior se ha realizado el análisis del problema mediante la especificación, y hemos conseguido identificar tanto el problema que queremos resolver, como las necesidades que ha de satisfacer el sistema a desarrollar. Pero, el resultado de la especificación no es suficiente para llevar a cabo la implementación del sistema, debido a que ésta se orienta a la descripción del problema, independientemente de las tecnologías a utilizar. Por tanto, una vez identificado el problema, hace falta decidir como resolverlo.

La etapa de diseño consiste precisamente en la toma de diversas decisiones que nos lleven a desarrollar un sistema capaz de resolver satisfactoriamente el problema planteado. Si el objetivo de la etapa de especificación es conseguir qué ha de hacer el sistema, el objetivo de la etapa de diseño es determinar las pautas que se han de seguir, es decir, cómo se ha de hacer.

Es en este capítulo, en la fase de diseño, donde se tienen en cuenta aspectos como la tecnología de que se dispone. El diseño comenzará con un estudio comparativo entre las posibles arquitecturas. Se ha de destacar que, la solución al problema planteado no es única, ya que existen diferentes opciones correctas para el mismo problema. Lo que se pretende en este capítulo es exponer y justificar las decisiones que se han tomado en el momento de desarrollar el sistema que resuelva el problema planteado.

La etapa de diseño se trata de un proceso crucial para poder finalizar un proyecto con éxito. Ésta dará paso a la etapa de implementación, donde se comenzará a materializar los esfuerzos en forma de sistema de información.

Para poder llevar a cabo el diseño se ha utilizado el libro *Enginyeria del software: Disseny I. Disseny de sistemes orientats a objectes amb notació UML*. que describe el diseño de sistemas orientados a objetos con notación UML.

5.2. DISEÑO DE LA ARQUITECTURA

El primer paso en el diseño, es decidir qué arquitectura será la más adecuada para nuestra aplicación. Al hablar de arquitectura software nos referimos a la estructura fundamental del sistema. Esta estructura determinará las características que tendrá el software implementado; características como eficiencia, reutilización de código, fiabilidad, etc.…

Para definir la arquitectura software existen diferentes patrones arquitectónicos, que ofrecen soluciones generales para conseguir que el sistema disponga de determinadas características. Según los requerimientos funcionales escogeremos una o otra solución.

La determinación de la arquitectura consiste en la toma de decisiones respecto a:

- La organización del sistema software.
- La selección de los elementos estructurales y sus interfaces.
- El comportamiento de estos elementos estructurales.
- La posible composición de los elementos estructurales en subsistemas más grandes.
- El estilo que guía esta organización.

Antes de definir la arquitectura del sistema, y establecer su estructura, es necesario identificar las propiedades que tendrá que ofrecer el sistema una vez construido. La elección de las propiedades que tiene que cumplir nuestra aplicación, guiarán la posterior elección del estilo y patrones arquitectónicos.

Las diferentes propiedades de la arquitectura son las siguientes:

- **Cambiable.**
  - **Extensible**. Nuevas funcionalidades o mejoras de los componentes.
  - **Portable**. Cambios de plataforma *hardware*, sistemas operativos, lenguajes,…
  - **Mantenible**. Detección y reparación de errores.
  - **Reestructurable**. Reorganización de componentes.

- **Interoperable**. Capacidad de intercambiar funcionalidades y datos entre dos o más entidades *software*.

- **Eficiente**. Buena utilización de los recursos *hardware*, alto rendimiento, tiempos reducidos de respuesta.

- **Fiable**. Capacidad de tolerancia a fallos y perdidas de conexión, robusteza frente a la incorrecta utilización y tratamiento de errores inesperados.

- **Probable**. Facilitar las pruebas de sistema.

- **Reutilizable**. Conseguir lo que se quiere con lo que se tiene, es decir, reutilizar *software* existente y hacer el nuevo para ser reutilizado.

5.2.1. PATRONES ARQUITECTÓNICOS

Existen diferentes tipos de estilos o patrones arquitectónicos, clasificables según los siguientes estilos:

- **Modelo Vista – Controlador**
- **Llamada y retorno**
  - **Programación principal y subrutinas**
  - **Orientado a objetos**
  - **Por capas**
- **Concentrado en los datos**
- **Flujo de datos**
  - **Batch secuencial**
  - **Tubos y filtros**

Ya que el cometido de este documento no es explicar los patrones arquitectónicos, sino detallar qué patrones se han decido usar y por qué, tan solo se detallarán aquellos patrones que tengan lugar en el diseño de la aplicación.

5.2.2. ARQUITECTURA ESCOGIDA

Tal y como se puede deducir de la explicación de los diferentes patrones arquitectónicos anteriormente mencionados, es muy difícil encontrar un patrón que favorezca la eficiencia del sistema, y al mismo tiempo, la cambiabilidad y la portabilidad. Esto sucede porque garantizar la cambiabilidad supone agrupar y organizar los datos y funciones en clases, módulos o capas de abstracción que, de alguna forma, esconden la implementación de estos componentes. En consecuencia obtenemos una mayor complejidad en la implementación del sistema, cosa que dificulta la eficiencia, ya que para realizar una misma tarea se han de consultar a varios componentes o atravesar diferentes capas de software.

Los requisitos funcionales de la aplicación nos indican que el sistema debe estar preparado para sufrir cambios, también se pide una buena interacción con el usuario y facilidad de mantenimiento. Para cumplir estos objetivos hemos decidido combinar dos patrones

arquitectónicos. Primero el patrón de orientación a objetos, con el cual conseguiremos encapsular los datos y las operaciones, y así tratar los datos en objetos, lo que facilitará su reutilización y la facilidad de mantenimiento del software. Segundo, dividiremos el sistema en capas. *Capa de presentación*, *capa de dominio* y *capa de datos*, esto nos permitirá que el sistema pueda fácilmente adaptarse a otros tipos de persistencia de datos y a cambios en la presentación.

A continuación, explicaremos brevemente, el patrón por capas y la orientación a objetos un poco más en detalle.

5.2.2.1. PATRÓN POR CAPAS

El patrón por capas trata de separar el código de la aplicación de la presentación. Cada capa puede acceder únicamente al nivel contiguo, por eso, uno no puede programar un nivel de acceso a datos que sea llamado desde el nivel de presentación o viceversa.

El diseño de una aplicación de acuerdo con esta arquitectura de capas presenta la gran ventaja de producir código modular en el que la modificación de uno de sus componentes, por ejemplo, la interfaz de usuario (la forma en la cual el usuario navega por la aplicación o la forma en la que se presentan los datos) no requiere modificar ningún de los elementos del resto de capas. A parte, también permite la reutilización del código para futuras aplicaciones.

Figura 5.1. Estructura de la arquitectura de aplicaciones en tres capas

Cada una de las capas que se muestran esta comunicada con la inmediatamente superior y la inferior consiguiendo una abstracción del sistema muy notable.

- **Capa de Presentación.** Para el desarrollo de la interfaz de usuario, la cual ofrecerá todas las funcionalidades del sistema utilizaremos el siguiente software:

Producto	Versión
Java Swing	1.5

- **Capa de Dominio.** En la capa de dominio tendremos la lógica de negocio, que es donde se implementarán las funcionalidades del sistema.

- **Capa de datos.** En el componente de acceso a los datos, se generan todos los accesos a ficheros de disco para obtener así los datos necesarios.

5.2.2.2. ORIENTACIÓN A OBJETOS

La orientación a objetos más que un patrón arquitectónico a seguir, es una filosofía de diseño. Es un paradigma de programación que define los programas en términos de clases de objeto. La idea de orientación a objetos es subir el nivel de programación al nivel objeto, concepto más entendedor para el ser humano que un *byte* o un *string*.

Un objeto es un paquete de datos del sistema (los atributos) acompañados por las operaciones que trabajan con estos datos (los métodos.) Los atributos son las operaciones más importantes del objeto, y los métodos son herramientas para poder manipular estos atributos. Las clases definen los objetos, es decir, los tipos de datos que almacenaran y los procedimientos que tendrán. La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas. Esta filosofía es completamente diferente a los lenguajes tradicionales basados en procedimientos, en los que los datos y los procedimientos están separados y sin relación. La orientación a objetos está pensada para realizar los programas y módulos más fáciles de escribir, mantener y reutilizar.

Las características más importantes de la orientación a objetos son:

- **Abstracción.** Cada objeto en el sistema sirve como un agente abstracto que puede realizar trabajo, informar, cambiar de estado y comunicar-se con otros objetos en el

sistema sin tener que revelar como se implementan estas características. Los procesos, las funciones o los métodos también puede ser abstraídos.

- **Encapsulación.** También llamada "ocultación de la información", asegura que los objetos no puedan cambiar de estado interno de otros objetos de manera inesperada; sólo los propios métodos internos del objeto pueden acceder a su estado. Cado tipo de objeto expone una interfaz a los otros objetos que especifica como los objetos pueden interactuar con él.

- **Polimorfismo.** Las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos. La invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del referente. Cuando esto sucede en tiempo de ejecución, a esta última característica se le denomina asignación dinámica. Algunos lenguajes proporcionan medios más estáticos, en tiempo de compilación, como pueden ser las plantillas y la redefinición de operadores.

- **Herencia.** Organiza y facilita el polimorfismo y la encapsulación permiten a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que reimplementar su código. Esto se suele a hacer habitualmente agrupando los objetos en clases y las clases en árboles reflejando un comportamiento común.

Las ventajas y los inconvenientes que supone este paradigma son los siguientes:

- **Ventajas**

  - Favorece la reutilización del software.
  - El coste de implementación es menor respecto a otros paradigmas.
  - Permite modelar mejor los dominios, facilitando así el análisis y el diseño.
  - Permite dividir las responsabilidades más fácilmente en un mismo grupo de trabajo.
  - Existen múltiples herramientas, metodologías y lenguajes de programación que hacen que trabajar con objetos sea la opción adecuada.
  - Los sistemas orientados a objetos resultan más viables económicamente a medio y largo plazo.
  - Facilita un análisis y diseño más elegante de la aplicación.
  - Mayor seguridad de las aplicaciones gracias a la ocultación de datos y la implementación interna de las clases (encapsulamiento).

- Extensibilidad del programa de manera fácil mediante la herencia y adición de nuevos módulos.

- **Inconvenientes**

  - Es una técnica que se aprende rápidamente, pero muy difícil de dominar completamente.
  - Los costes de desarrollo a corto plazo son más elevados y se tarda más en ver una primera versión del sistema que funcione.
  - Los mecanismos de la orientación a objetos presentan un cierto coste en eficiencia.

En vista de mi dominio personal de las herramientas de las que disponía, no existía duda de que el mejor paradigma para desarrollar este proyecto en la actualidad fuera el de la orientación a objetos.

5.3. CAPA DE PRESENTACIÓN

La capa de presentación es la encargada de interactuar con el usuario y se corresponde tradicionalmente a lo que se conoce como interfaz de usuario. Esta capa se encarga de mostrar una serie de pantallas que permiten interactuar al usuario con el sistema.

El sistema comunica al usuario la información y captura la información introducida por el usuario, realizando un filtrado previo para comprobar que no hay errores de formato. Esta capa sólo se comunica con la capa de dominio.

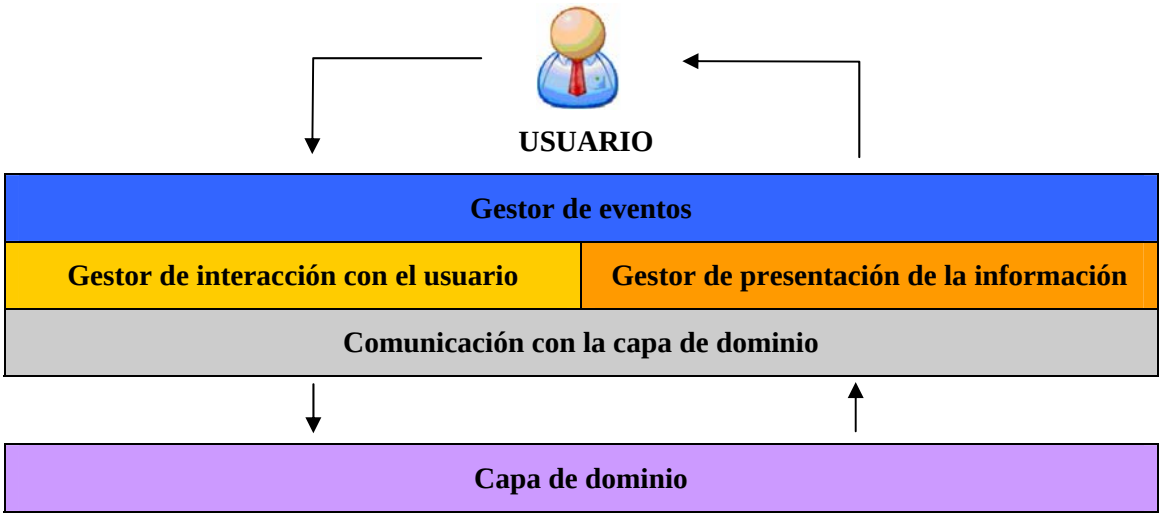


Figura 5.2. Estructura de la capa de presentación

5.3.1. DIAGRAMA DE LA CAPA DE PRESENTACIÓN

Para comprender mejor la comunicación entre las diversas pantallas de la capa de presentación, así como la comunicación de éstas con la capa de dominio podemos observar el siguiente diagrama. En éste se puede ver como toda la información enviada por el usuario a través de las diferentes pantallas como la recibida de la capa de dominio para ser presentada al usuario, es controlada por una clase principal en la capa de presentación, la clase *Scrabble*. Gracias a este diseño conseguimos poder comunicar todas las pantallas entre sí, y poder gestionar la interacción del usuario hacia la capa de dominio y viceversa de una manera modular. Ya que permitimos una total cambiabilidad entre las diferentes pantallas y la capa de negocio, delegando la responsabilidad de conocer la comunicación con la capa de dominio únicamente a la clase *Scrabble.*

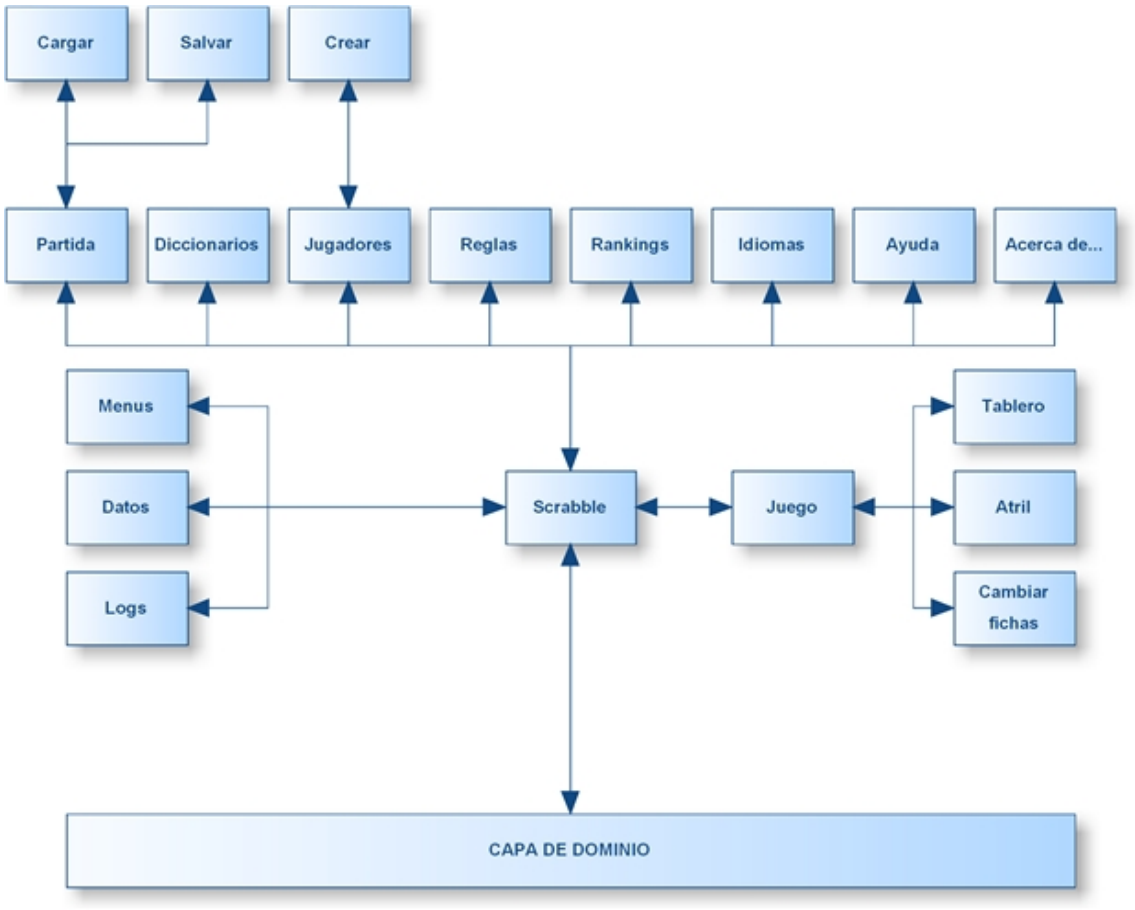


Figura 5.3. Diagrama de pantallas y controladores de la capa de presentación

5.3.2. DISEÑO DE LAS PANTALLAS

Tal y como se ha indicado anteriormente, la interfaz de usuario se realizará mediante la tecnología *Java Swing*. *Swing* es una biblioteca gráfica para *Java* que forma parte de las *Java Foundation Classes* (*JFC*). Incluye *widgets* para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

La aplicación dispondrá de la siguiente estructuración de pantalla:

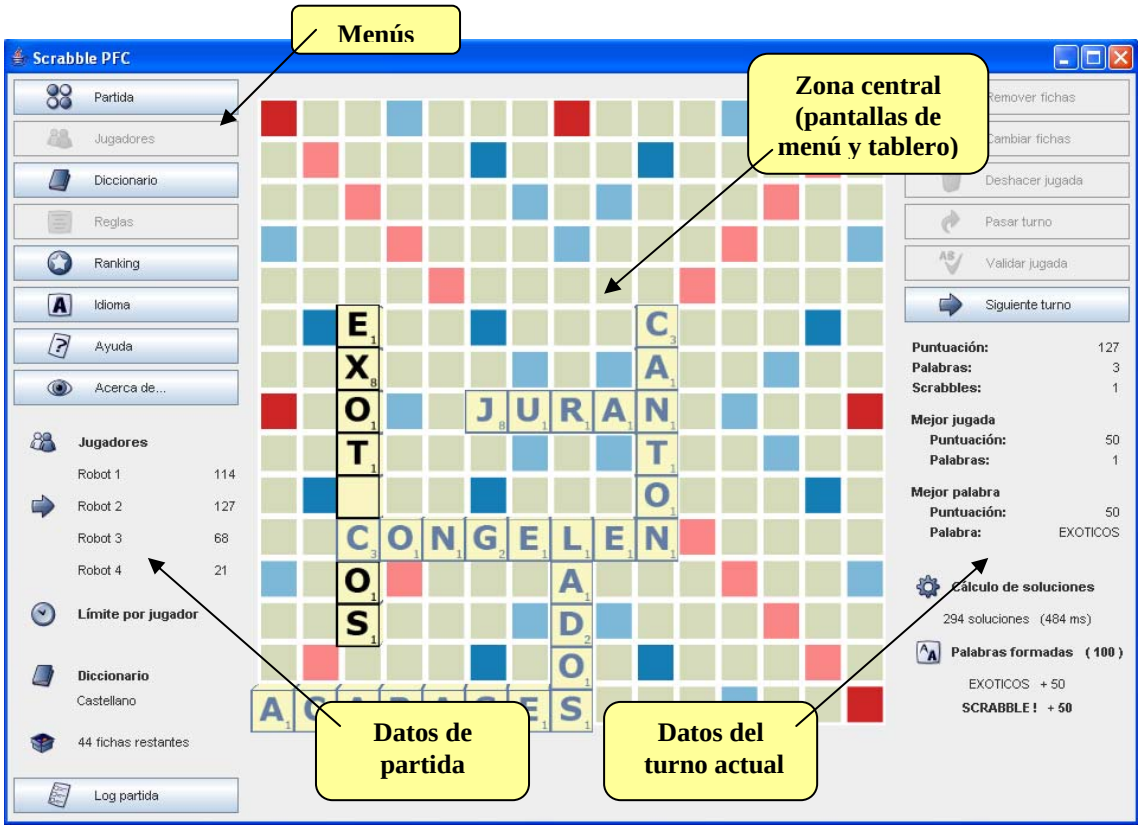


Figura 5.4 Distribución de la pantalla principal

Podemos diferenciar cuatro partes bien definidas:

- **Menús.** Permitirán acceder a todas las pantallas necesarias para configurar una partida, así como para crear/eliminar perfiles de jugadores, y visualizar datos como rankings y ayuda.

- **Datos de partida.** En el momento en que una partida este en proceso, se podrán consultar los datos sobre los jugadores que forman parte en ella, así como el tiempo

restante para el jugador o turno actual, el diccionario que está en uso en la partida y la cantidad de fichas restantes en el saco.

- **Datos de turno actual.** En el momento en que una partida este en proceso, se podrán consultar los datos sobre el jugador que actualmente este en posesión del turno de juego, así como acceder a todas las acciones posibles para la realización del turno.

- **Zona central.** Es en esta área de la pantalla principal en la que se podrá visualizar aquellas pantallas accedidas a través de los diferentes menús, o si hay una partida en juego, en la que se podrá acceder al tablero de juego como al atril del jugador del turno en juego.

A continuación se listan algunos ejemplos de pantallas correspondientes a cada una de las secciones principales del menú.

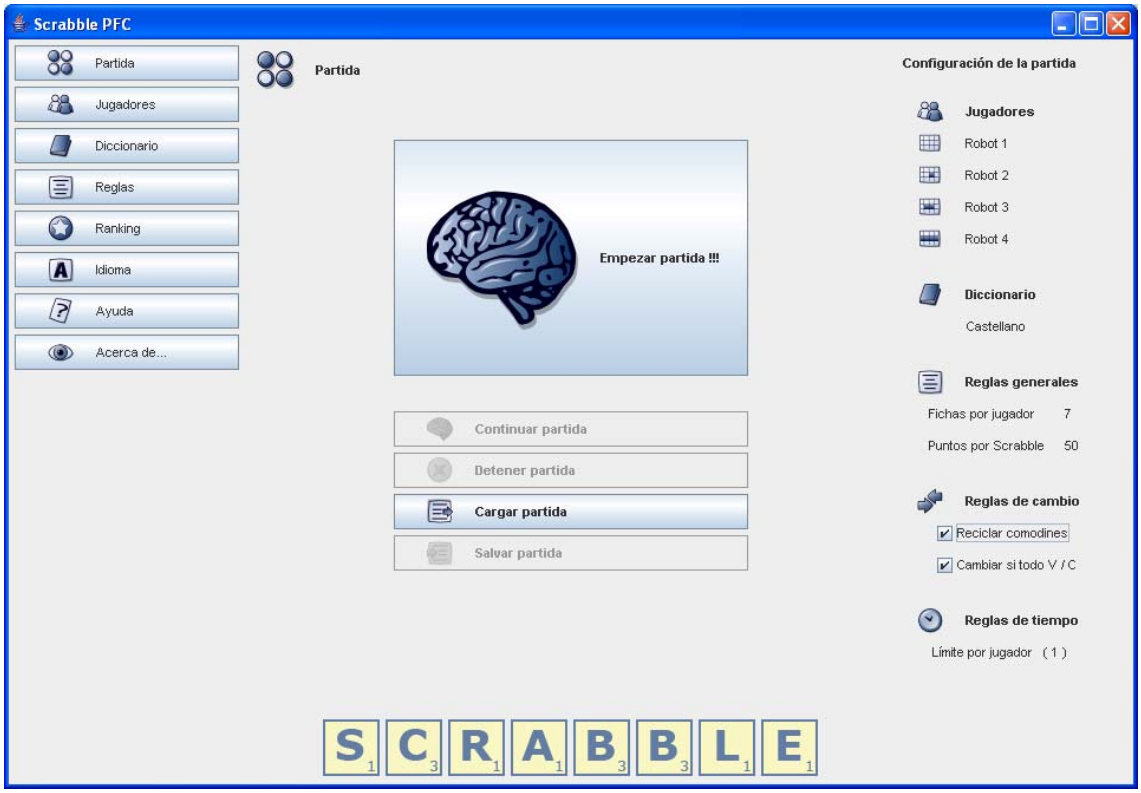


Figura 5.5. Pantalla principal

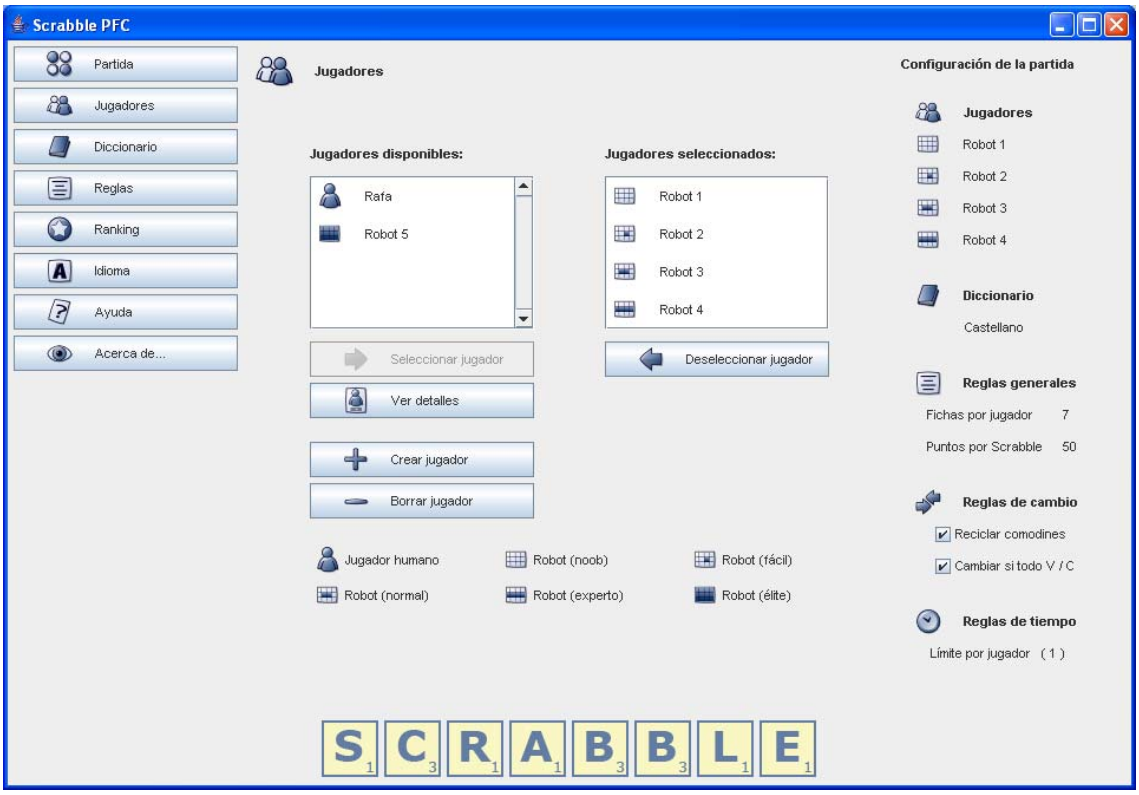


Figura 5.6. Pantalla gestión de jugadores

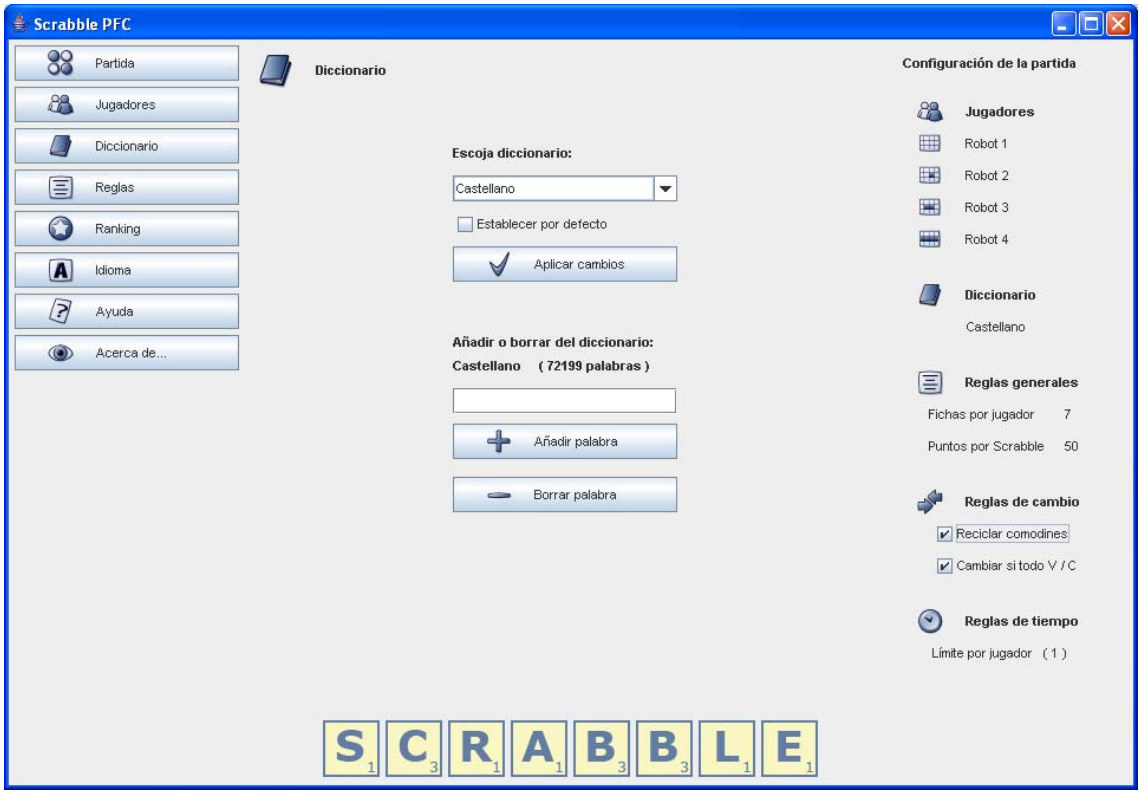


Figura 5.7. Pantalla de gestión de diccionarios

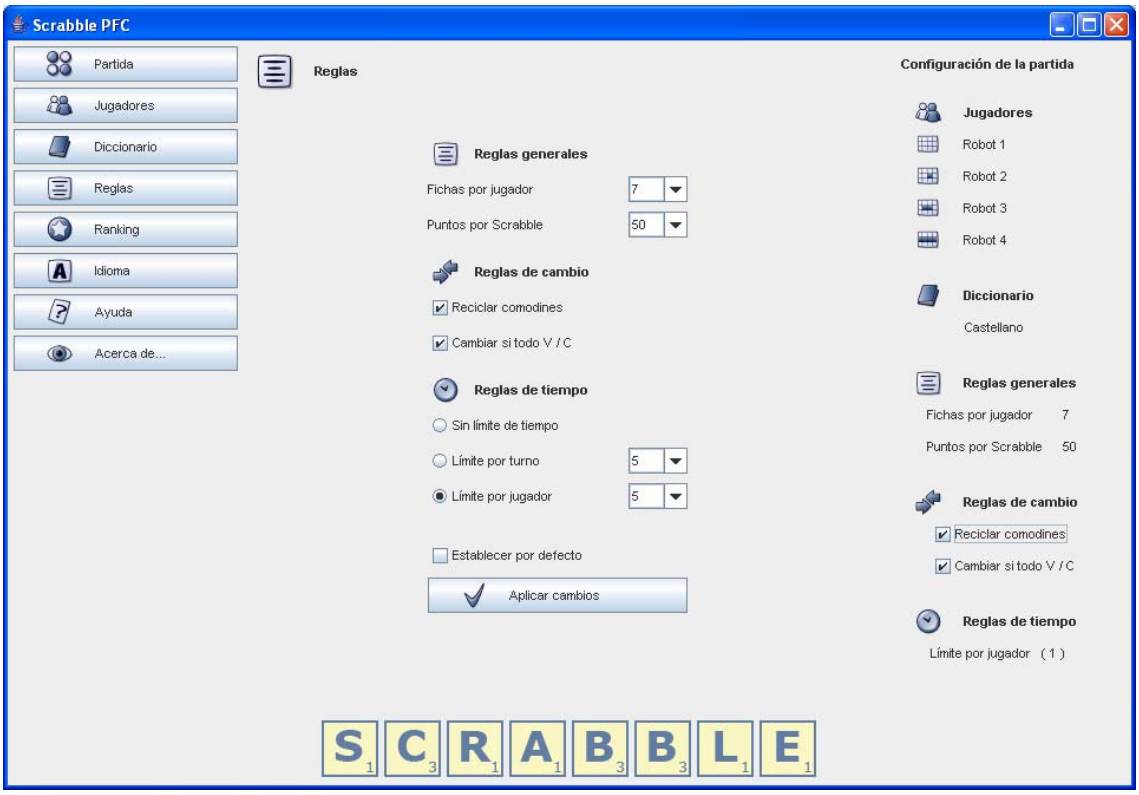


Figura 5.8. Pantalla gestión de reglas

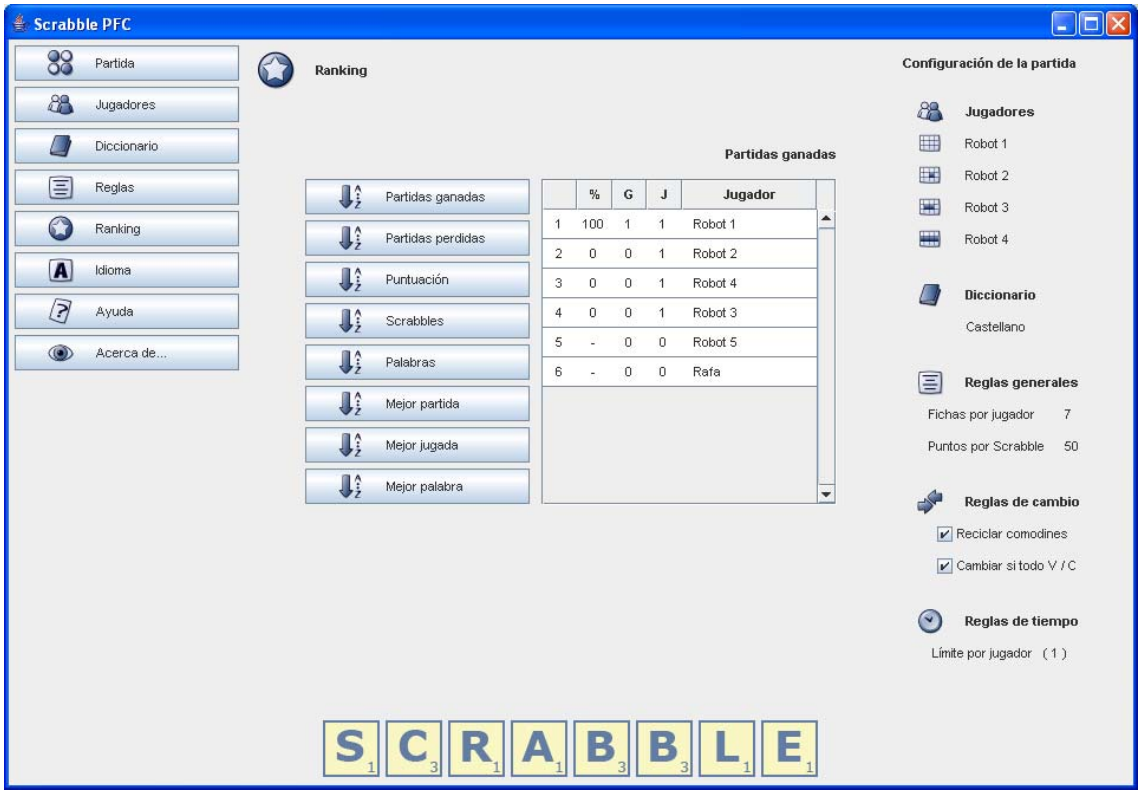


Figura 5.9. Pantalla de rankings

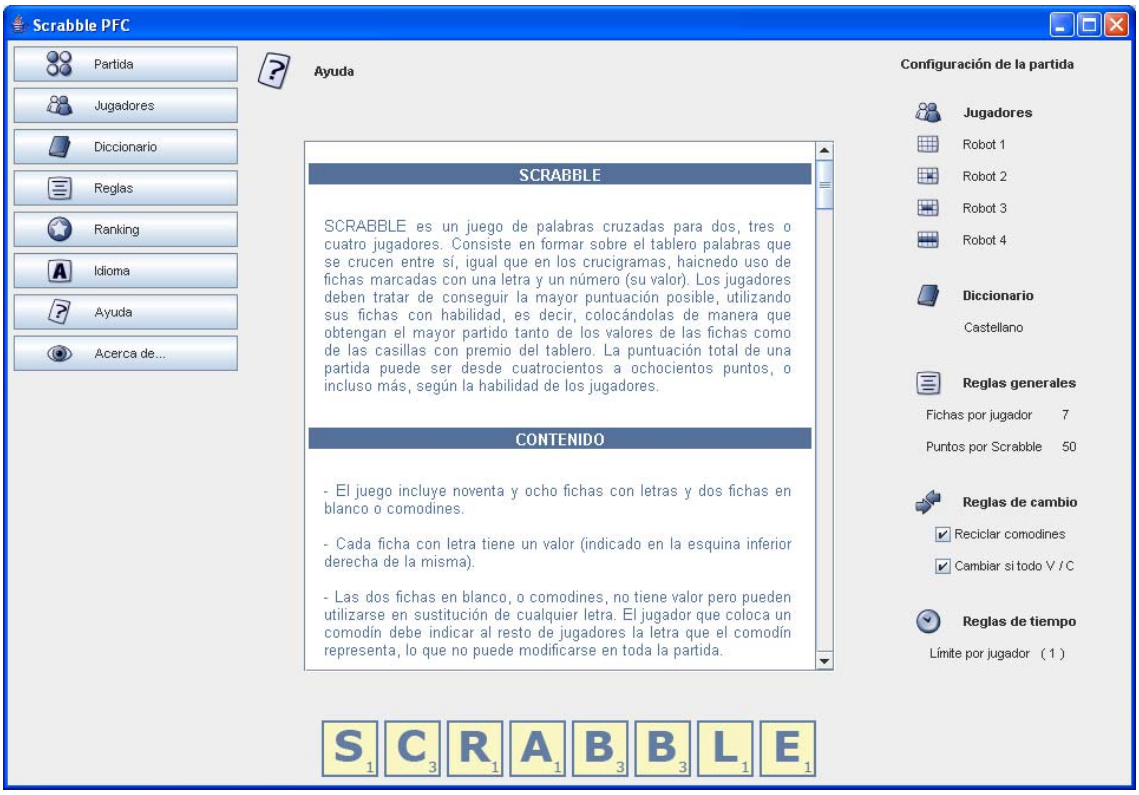


Figura 5.10. Pantalla gestión de ayuda

5.3.3. COMPONENTES

Destacar que todos los componentes visuales utilizados extienden de sus respectivos en la plataforma J2EE. De esta manera conseguimos unificar ciertas propiedades que serán comunes en toda la aplicación como pueden ser el tipo fuente, el color, el tamaño, etc.

Así, por ejemplo el componente de texto *ScrabbleText* utilizado en la aplicación extiende directamente del componente ya existente en *Java Swing*, *JTextField*. Heredando de esta manera todas sus propiedades y métodos.

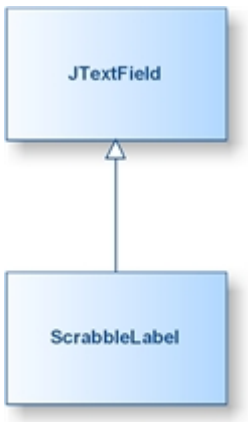


Figura 5.11 Ejemplo de extensión de componentes

5.4. CAPA DE DATOS

La capa de gestión de datos es la encargada de hacer de puente entre la capa de dominio y el tipo de persistencia de datos que hemos escogido, en este caso ficheros. Las clases de esta capa están atadas a la tecnología que se utilice para realizar la persistencia de los datos. Se debe diseñar la capa de gestión de datos de tal manera que cambios en la forma de almacenar los datos no repercuta a las otras capas.

Como sistema de persistencia de datos, se ha optado por implementar un sistema de gestión de ficheros. Para acceder a los correspondientes ficheros encargados de almacenar la información necesaria en nuestro sistema la propia plataforma J2EE nos ofrece clases específicas para ello.

Dichos ficheros almacenarán los datos para los perfiles de jugadores, partidas salvadas, así como todos los datos necesarios en cada idioma importado al sistema como son el diccionario (listado completo de palabras válidas), el saco (listado completo de todas las letras del abecedario del idioma correspondiente, junto con sus respectivos valores y cantidades), los textos que serán visualizados en los menús y pantallas de la aplicación, y la ayuda para el usuario. Existirá también un fichero dedicado a almacenar la configuración por omisión de la aplicación, como puede ser el idioma de localización, el idioma de diccionario, la configuración de las diversas reglas, etc. La aplicación ofrecerá la posibilidad de guardar en ficheros de texto el log completo de las partidas que se deseen, con toda la información correspondiente al desarrollo de las mismas.

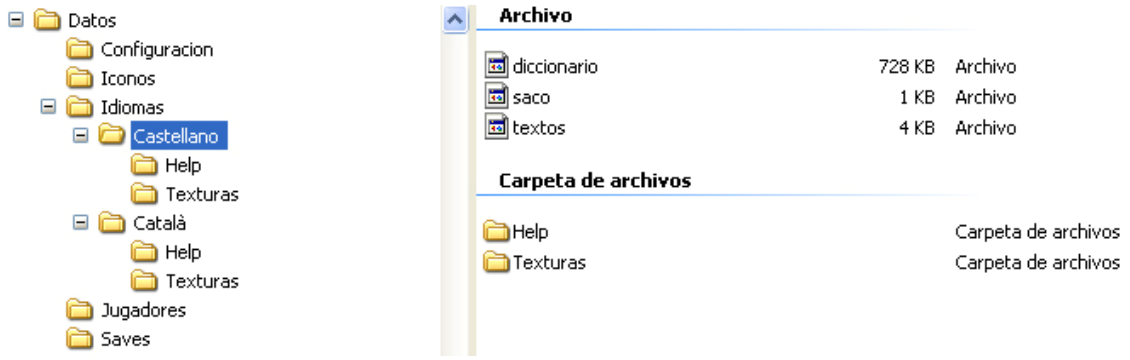


Figura 5.12 Ejemplo de ubicación de los ficheros

5.4.1. DESCRIPCIÓN DE LOS FICHEROS

A continuación se describen los anteriores ficheros mencionados para conservar la persistencia de los datos de nuestro sistema.

- **Ficheros de jugadores.** Los perfiles de jugadores serán almacenados en ficheros de tipo binario. Esta decisión ha sido tomada para poder asegurar la fiabilidad de los datos contenidos ellos. Pues son datos referentes a las puntuaciones totales y máximas, así como el número de palabras y scrabbles que ha ido formando el jugador correspondiente al perfil a lo largo de las diversas partidas en las que haya participado. Dichos ficheros, se obtendrán a partir de la serialización que nos permite *Java* de la clase *Jugador*. Es por ello que existirá un fichero por cada perfil de jugador existente en la aplicación. La ubicación de éstos será una carpeta *Jugadores*.

- **Ficheros de partidas salvadas.** Las partidas salvadas serán almacenas en ficheros de tipo binario. Al igual que los ficheros de perfiles de jugador, esta decisión ha sido tomada para poder garantizar la fiabilidad de los datos contenidos. Se almacenará toda la información necesaria para posteriormente crear la partida en el mismo estado en que fue guardada. La ubicación de estos ficheros será en la carpeta *Saves*.

- **Ficheros de diccionarios.** Los diccionarios serán almacenados en ficheros de texto plano. La estructura de éstos consiste en una simple lista de palabras separadas por un salto de línea. De esta manera, se consigue que cualquier usuario con conocimientos mínimos sobre ficheros, pueda abrir un fichero de diccionario, comprender el contenido de éste, y modificarlo, de manera que pueda incluir, eliminar o modificar palabras a la lista. Adicionalmente, la aplicación ofrecerá la posibilidad de añadir y eliminar palabras del diccionario de manera que el usuario no necesite ningún tipo de conocimientos sobre ficheros. Existirá por lo tanto un fichero de diccionario por cada idioma incorporado en la aplicación. La ubicación de éstos será dentro de la carpeta del propio idioma al que pertenece.

- **Ficheros de sacos.** Los sacos serán almacenados en ficheros de texto plano. La estructura de éstos consiste en la lista de letras incluidas en el alfabeto de juego del idioma al que corresponde el saco. Junto a cada una de las letras, se proporciona la información sobre el valor de éstas y la cantidad de fichas con dichas letras que contendrá el saco. En los casos en que el idioma contenga letras dobles, se almacenarán

también dichas letras junto con su carácter alfanumérico que la representará internamente en la aplicación. Ya que como se detallará en la implementación, toda letra será representada por un único carácter alfanumérico. Existirá un fichero de saco por cada idioma incorporado en la aplicación. La ubicación de éstos será dentro de la carpeta del propio idioma al que pertenece.

- **Ficheros de localización.** Los textos de localización serán almacenados en ficheros de texto plano. La estructura de estos consiste en una lista de correspondencias entre un código alfanumérico y su descripción. De esta manera, desde la aplicación a la hora de cargar el texto de un menú, botón, etiqueta o mensaje, se hará referencia al código, obteniendo así a través de éste la descripción del texto a mostrar. Al tratarse de un fichero de texto plano, un usuario con conocimientos mínimos sobre ficheros, podrá modificar dichas descripciones. Existirá un fichero de localización por cada idioma incorporado en la aplicación. La ubicación de éstos será dentro de la carpeta del propio idioma al que pertenece.

- **Ficheros de ayuda.** Las ayudas serán almacenadas en ficheros de texto plano. La estructura de éstos será en formato *HTML*, de manera que la información contenida pueda visualizarse con cualquier navegador web. El motivo principal de esta decisión es poder mostrar toda la información contenida dentro de la aplicación en el mismo formato que lo mostraría un navegador, consiguiendo así una ayuda más visual y fácil de comprender para el usuario a poder incorporar imágenes, y diferentes formatos de letra. Existirá un fichero de ayuda por cada idioma incorporado en la aplicación. La ubicación de éstos será dentro de la carpeta del propio idioma al que pertenece.

- **Ficheros de logs.** Los logs de partidas serán almacenados en ficheros de texto plano. De manera que el usuario pueda comprender el contenido de éstos, pues su finalidad no es otra que informar a éste sobre el desarrollo de la partida a la que hacen referencia. La estructura de éstos es de manera que puedan mostrar la lista de jugadores que formaron parte de la partida, seguidamente de la lista completa de turnos que tuvieron lugar con toda la información sobre éstos como puede ser el jugador de cada uno de los turnos, con que fichas contaba, que jugada realizó y la puntuación de la misma, si realizó cualquier otro tipo de acción como cambiar fichas o pasar turno, etc. Finalmente, se almacenará también el estado final del tablero con todas las palabras que se formaron sobre éste a lo largo de la partida y en que posición se dispusieron. Existirá por lo tanto tantos ficheros de logs de partidas como el usuario haya creado, siendo la ubicación de éstos totalmente libre a elección del usuario.

- **Ficheros de configuración.** Existirán dos ficheros de configuración únicos en la aplicación, *configuración* y *reglas*. El primero de ellos c*onfiguración*, estará destinado al almacenar toda la información sobre la configuración de la aplicación como puede ser la lista de idiomas de localización, la lista de idiomas de diccionarios, idiomas por omisión (tanto de localización como de diccionarios). El segundo fichero de configuración *reglas*, estará destinado a almacenar toda la configuración por omisión sobre las diversas reglas configurables de la partida.

5.4.2. DIAGRAMA DE LA CAPA DE DATOS

A continuación se muestra el diagrama correspondiente a la capa de datos.

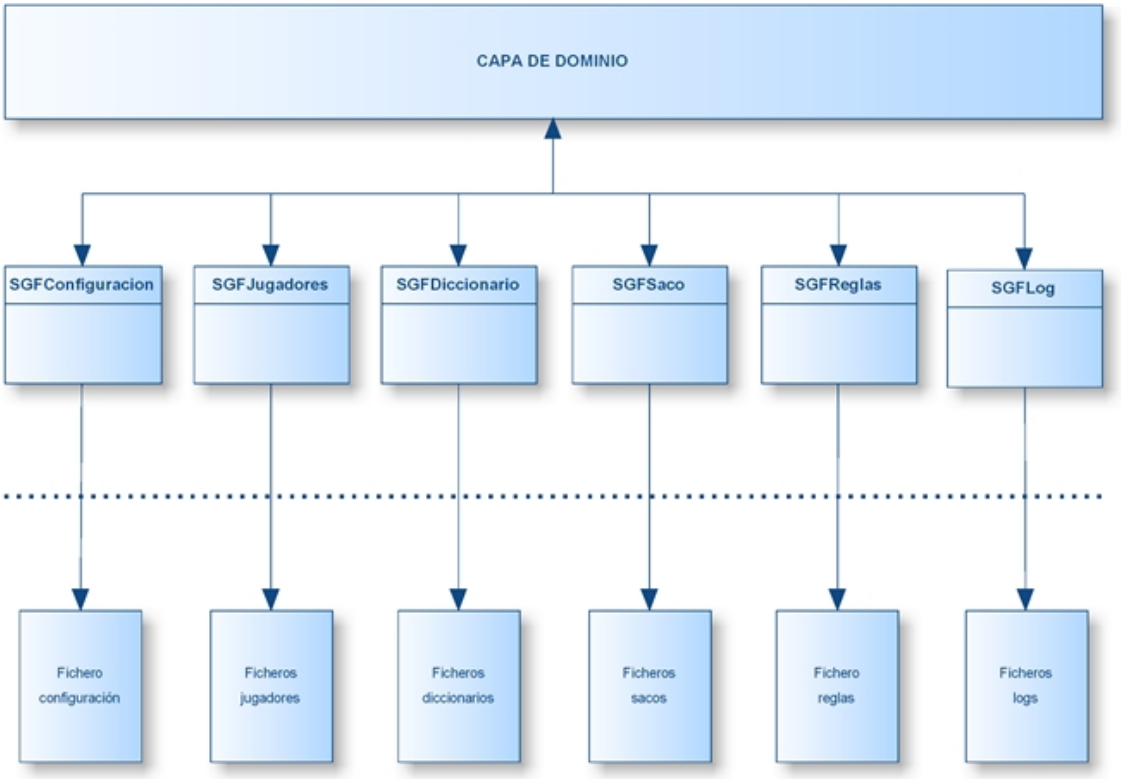


Figura 5.13. Diagrama de la capa de datos

5.5. CAPA DE DOMINIO

La capa de dominio será la encargada de tratar los sucesos generados en la capa de presentación y actuar de enlace con la capa de gestión de datos, devolviendo los resultados de las peticiones realizadas por el usuario. Como primer paso, se debe normalizar el modelo conceptual de la especificación. Esto es debido a que en el diseño tenemos *componentes software* y no *conceptos del dominio*. Debido a la arquitectura escogida orientada a objetos

existe una limitación que no permite implementar directamente todos los conceptos que se hayan podido usar en la especificación, como puede ser:

- Asociaciones n-arias, con n > 2
- Clases asociativas
- Información derivada
- Control de las restricciones de integridad

5.5.1. NORMALIZACIÓN DIAGRAMA DE CLASES

En el modelo conceptual descrito en la sección destinada a la especificación podemos ver que no existe ninguna de las limitaciones mencionadas, a excepción de la única restricción de integridad, la cual ya se tuvo en cuenta a la hora de especificar los contratos de las funcionalidades del sistema. Por lo que nuestro diagrama de la capa de dominio no sufre ningún cambio a causa de la normalización.

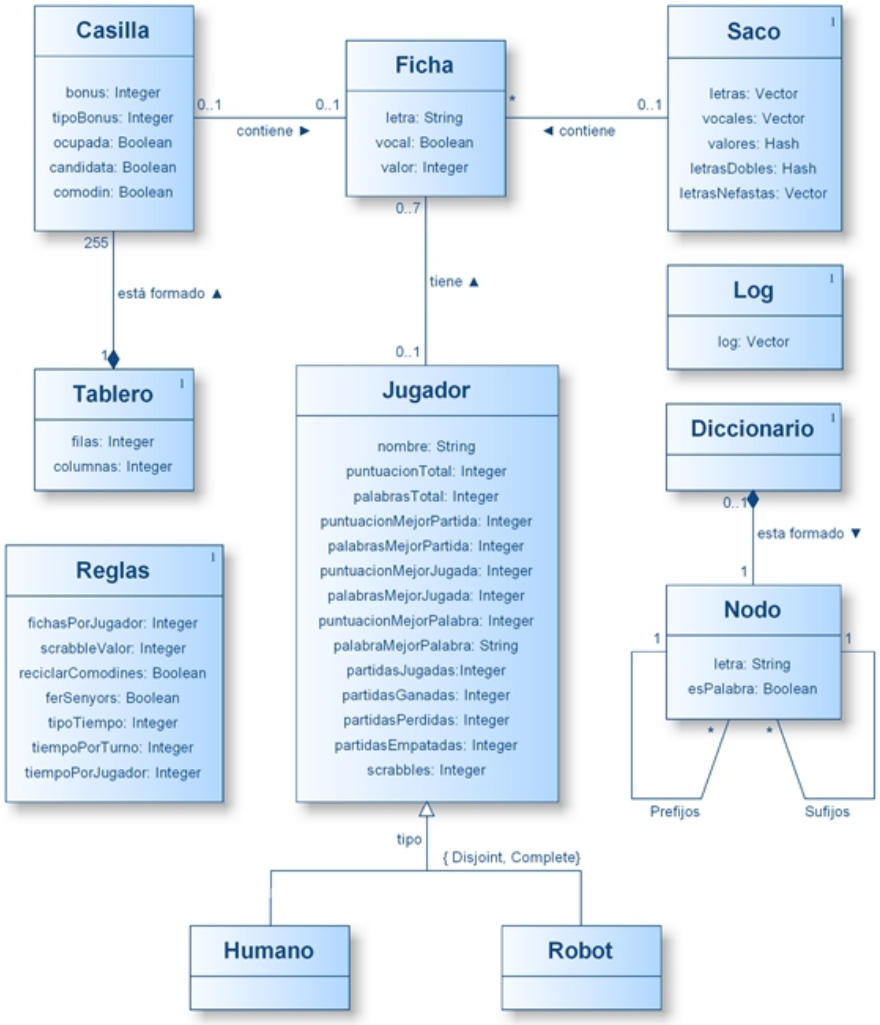


Figura 5.14. Diagrama del dominio inicial al diseño

5.5.2 PATRÓN DE DISEÑO: CONTROLADOR

Durante el transcurso del diseño de la capa de dominio, se ha recurrido en varias ocasiones a uno de los patrones de diseño más comunes. El patrón *Controlador*. El motivo de tal decisión es que:

- Los sistemas software reciben eventos externos
- Una vez interceptado un evento en la *capa de presentación*, algún objeto de la *capa de dominio* tiene que recibir este evento y ejecutar las acciones correspondientes.

Problema: ¿Qué objeto es el encargado de tratar un evento externo?

La solución reside en asignar esta responsabilidad a un controlador. El controlador es un objeto de una cierta clase. Existen posibles tipos de controladores:

- **Fachada – Sistema.** Un objeto que representa todo el sistema.
- **Fachada – Empresa.** Un objeto que representa todo el dominio (empresa, etc…)
- **Papel.** Un objeto que representa a un actor.
- **Caso de uso.** Un objeto que representa una instancia de un caso de uso.
- **Transacción.** Un objeto que representa una instancia de un suceso externo.

Finalmente, para llevar a cabo la solución se ha optado por utilizar en el diseño, controladores de tipo *Fachada – Empresa*. Dividiendo el dominio del problema en subproblemas con cierta correspondencia con los contextos de las funcionalidades del sistema. De esta manera, no existirá un único controlador inflado de posibles eventos que pueda recibir, sino que todo el trabajo ha sido repartido según el contexto de cada uno de los eventos en diversos controladores (más simples) ofreciendo una mayor facilidad a la hora de modificar el comportamiento del sistema en un futuro si fuera necesario.

Sí será necesario un controlador superior encargado de recibir todos los eventos por parte de la *capa de presentación*, y que comunique al controlador específico la recepción de tal evento. Este controlador, será el encargado también, de mantener la comunicación entre los diferentes controladores específicos. Es por eso que se le asignará la responsabilidad adicional, de llevar a cabo todas aquellas acciones (en el transcurso de una partida) más complejas que necesiten la resolución de varias acciones más simples llevadas a cabo por los diferentes controladores específicos.

Como resultado, se han obtenido los siguientes controladores:

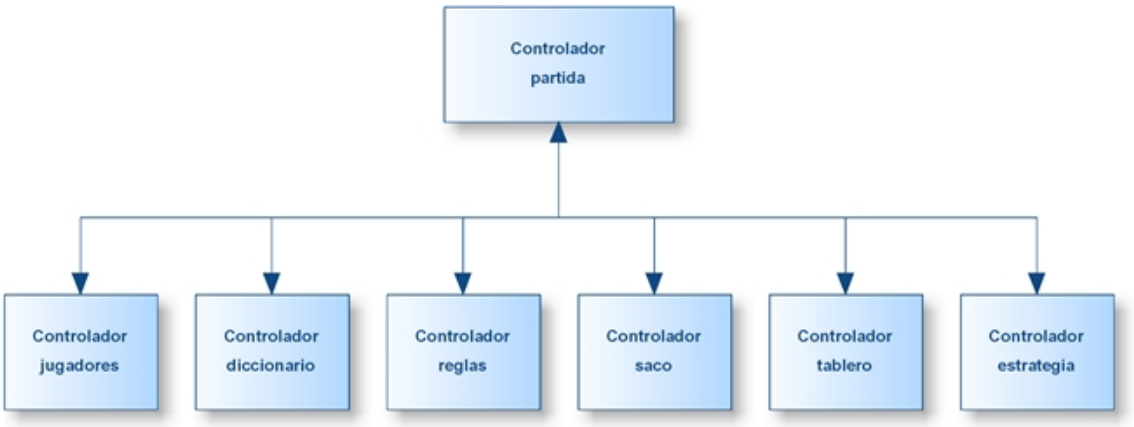


Figura 5.15. Controladores en la capa de diseño

5.5.3. DESCRIPCIÓN DE LOS CONTROLADORES

Se detallarán a continuación todos los controladores que han sido necesarios en el sistema para la realización de la aplicación. Pudiendo así, ver que tipo de eventos son responsabilidad de cada uno de ellos. Se describirán únicamente aquellos métodos de principal importancia, ya que enumerarlos todos dificultaría le lectura de los controladores cuando muchos de ellos son acciones sin complejidad alguna y similares entre todas ellas.

5.5.3.1. CONTROLADOR JUGADORES

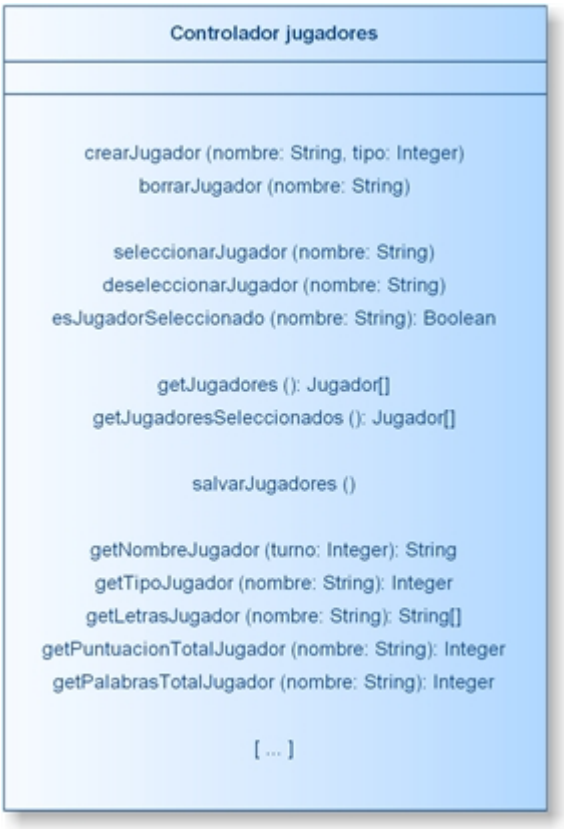


Figura 5.16. Descripción del controlador específico de jugadores

5.5.3.2. CONTROLADOR DICCIONARIOS

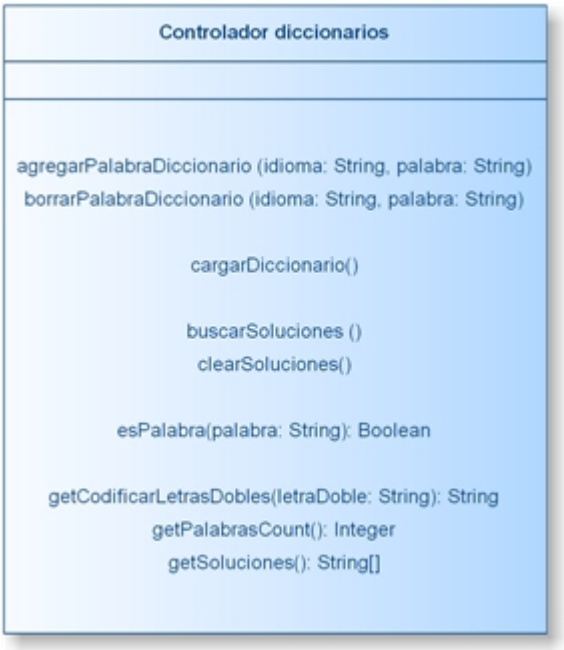


Figura 5.17. Descripción del controlador específico de diccionarios

5.5.3.3. CONTROLADOR REGLAS

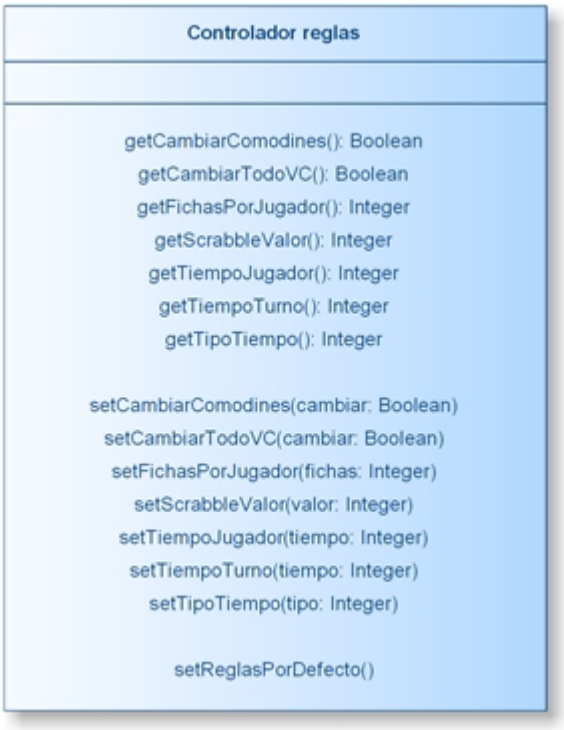


Figura 5.18. Descripción del controlador específico de reglas

5.5.3.4. CONTROLADOR SACO

Figura 5.19. Descripción del controlador específico de saco

5.5.3.5. CONTROLADOR TABLERO

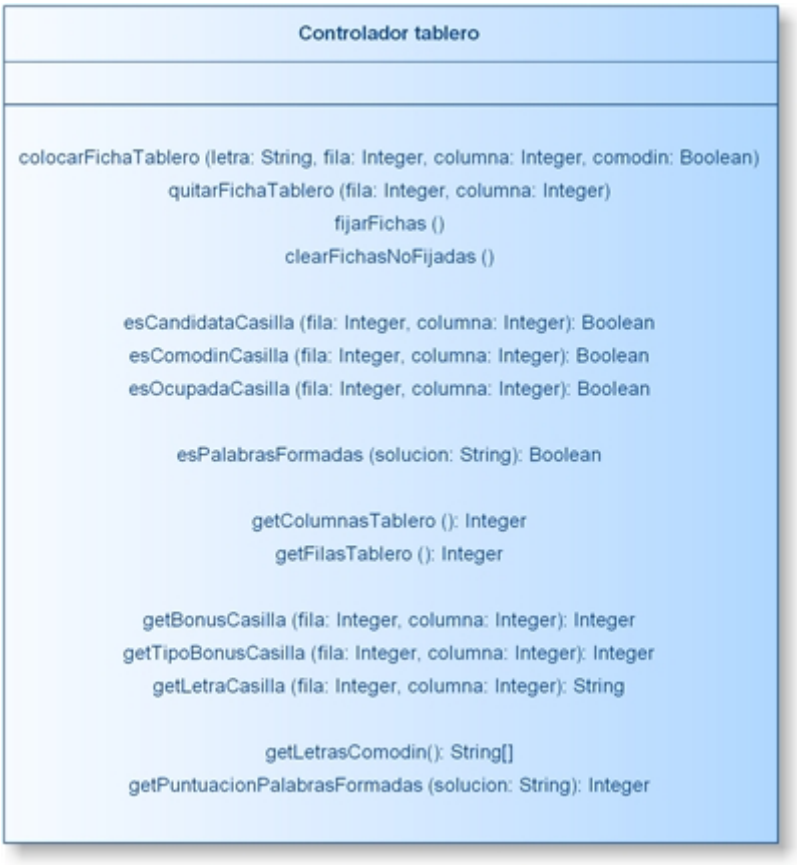


Figura 5.20. Descripción del controlador específico de tablero

5.5.3.6. CONTROLADOR ESTRATEGIA

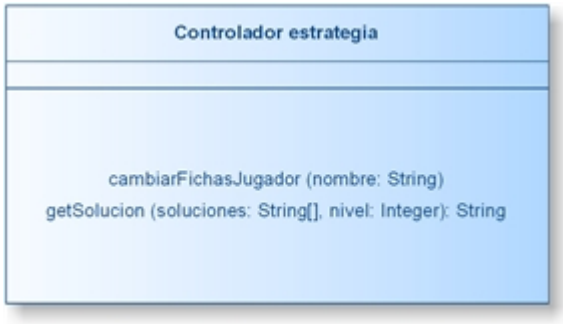


Figura 5.21. Descripción del controlador específico de estrategia

5.5.3.7. CONTROLADOR PARTIDA

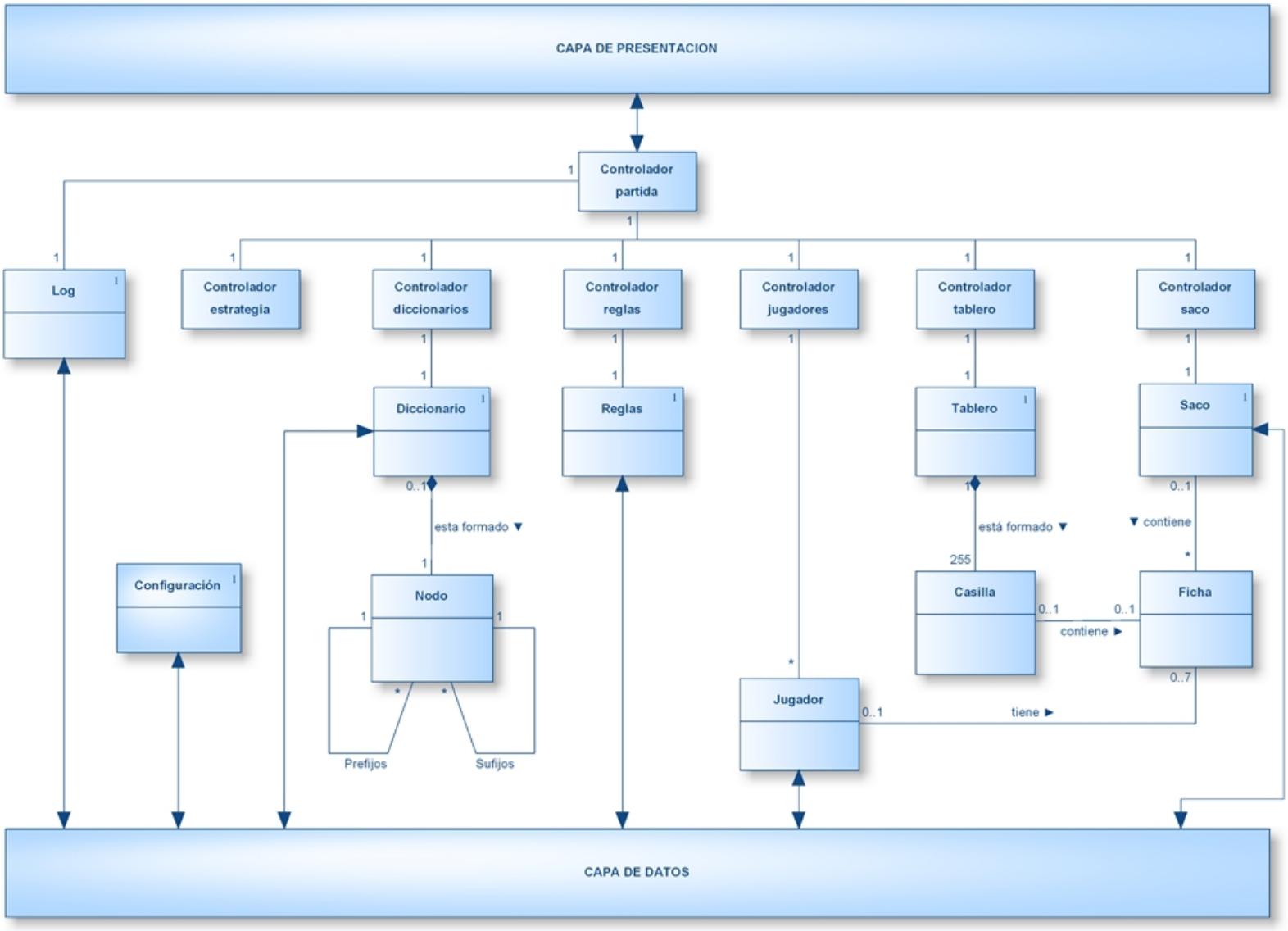
El controlador de partida es el controlador de nivel superior a través del cual se comunican el resto de controladores específicos entre ellos. También es el enlace de comunicación entre la capa de dominio y la capa de presentación.

En la siguiente descripción tan solo se citan aquellos métodos que son propios al controlador para poder gestionar las partidas tal y como se ha mencionado anteriormente. No se detallan por tanto, todos aquellos métodos cuya única funcionalidad no es otra que comunicar o traspasar información a los controladores específicos.

Figura 5.22. Descripción del controlador superior de partida

5.5.4. DIAGRAMA DE LA CAPA DE DOMINIO

Una vez introducidos todos los controladores en la capa de dominio con el fin de poder gestionar de manera totalmente modular el sistema, se ha obtenido un diagrama de la capa de dominio mucho más completo al inicial, tal como se puede ver a continuación.


CAPA DE PRESENTACION
Controlador partida
Log
Controlador estrategia
Controlador diccionarios
Controlador reglas
Controlador jugadores
Controlador tablero
Controlador saco
Diccionario
Reglas
Tablero
Saco
Configuración
Nodo
Prefijos
Sufijos
esta formado
está formado
contiene
Casilla
Ficha
Jugador
tiene
CAPA DE DATOS

5.5.5. DIAGRAMAS DE SECUENCIA

A continuación se presentan los diagramas de secuencia de aquellas operaciones que se ha creído conveniente detallar más a fondo. Se ha descartado la opción de indicar formalmente el diagrama de secuencia de todas y cada una de las funcionalidades ya que como se ha podido ver en el capítulo anterior de especificación, gran parte de las funcionalidades son inserciones, consultas y eliminación de datos del sistema, por lo que en su defecto se presentarán los diagramas de un tipo de inserción, consulta y eliminación como ejemplo.

5.5.5.1. DIAGRAMA: CREARJUGADOR

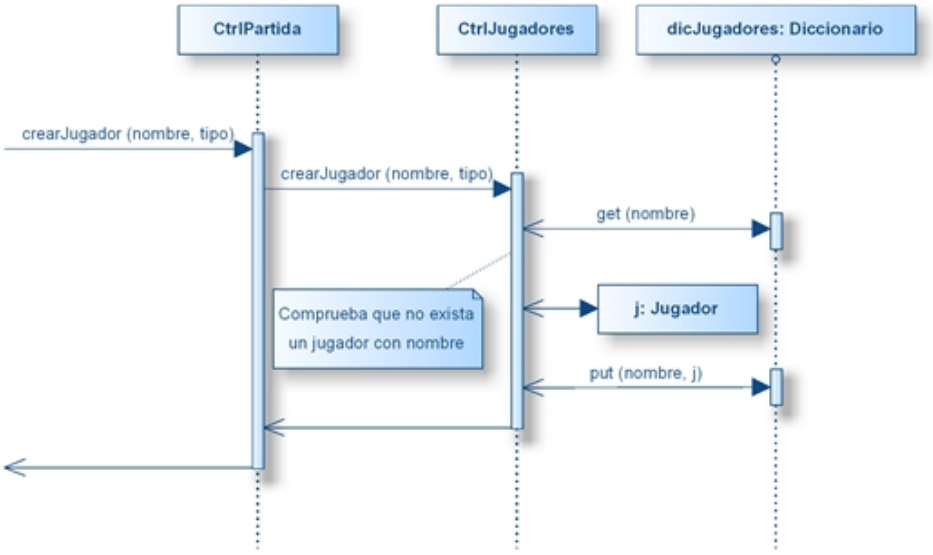


Figura 5.23. Diagrama de secuencia crearJugador

5.5.5.2. DIAGRAMA: BORRARJUGADOR

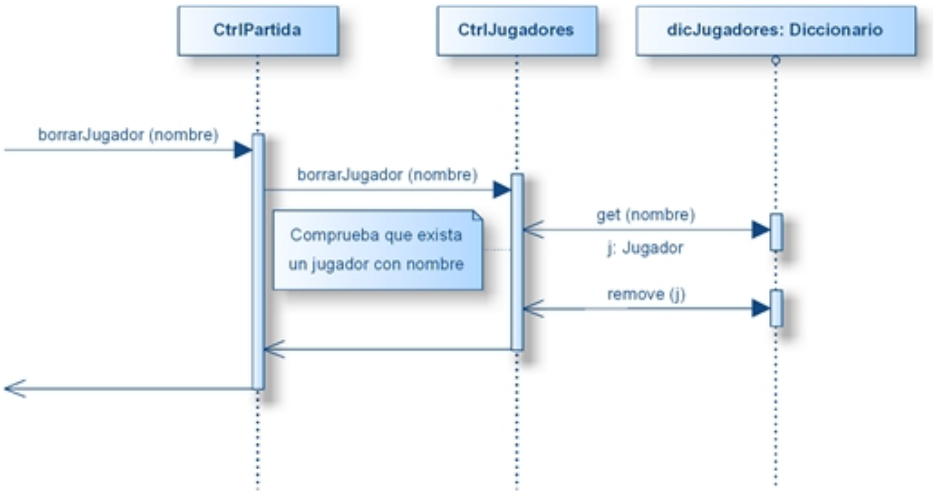


Figura 5.24. Diagrama de secuencia borrarJugador

5.5.5.3. DIAGRAMA: VERJUGADOR

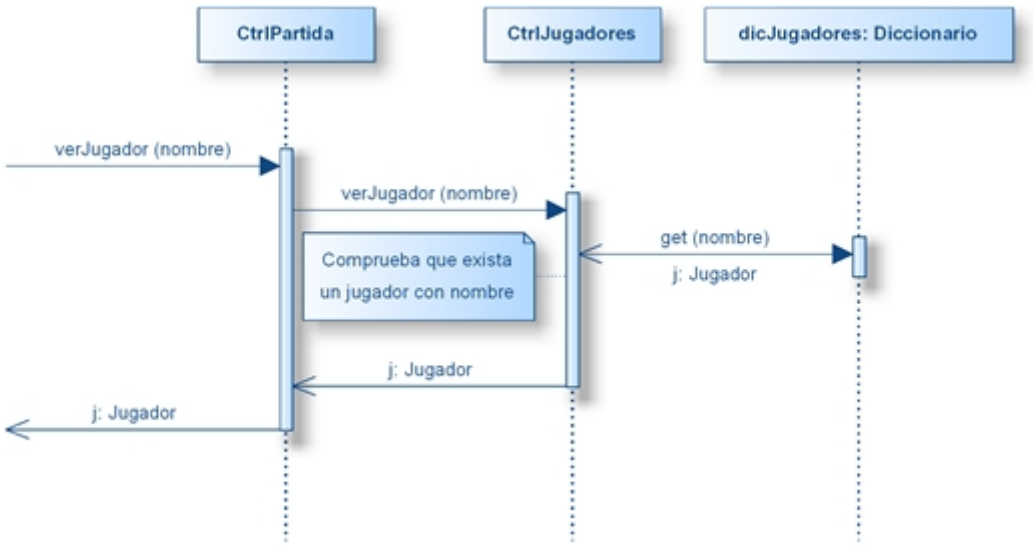


Figura 5.25. Diagrama de secuencia verJugador

5.5.5.4. DIAGRAMA: COLOCARFICHA

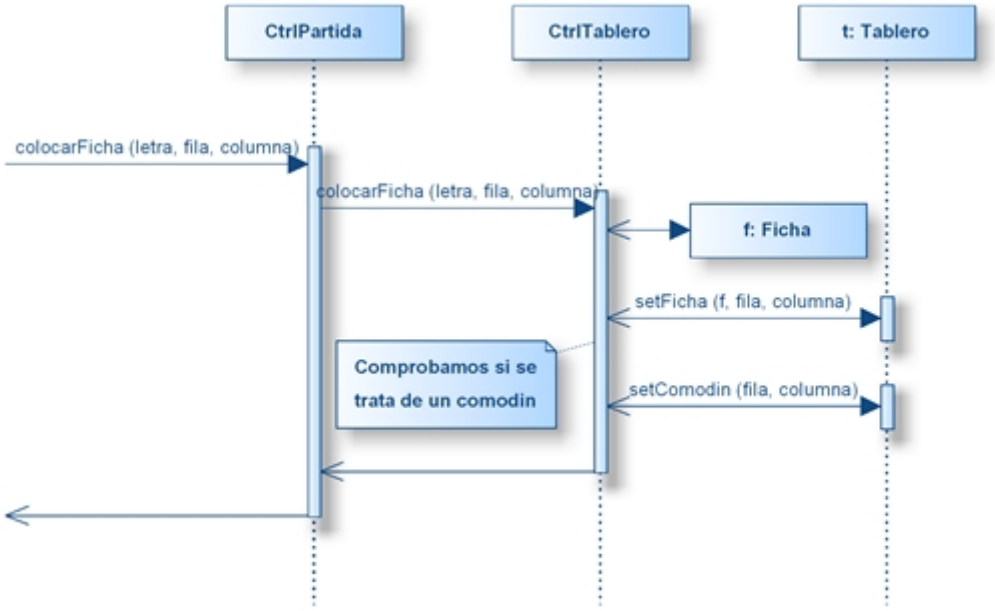


Figura 5.26. Diagrama de secuencia colocarFicha

5.5.5.5. DIAGRAMA: REALIZARSIGUIENTETURNO

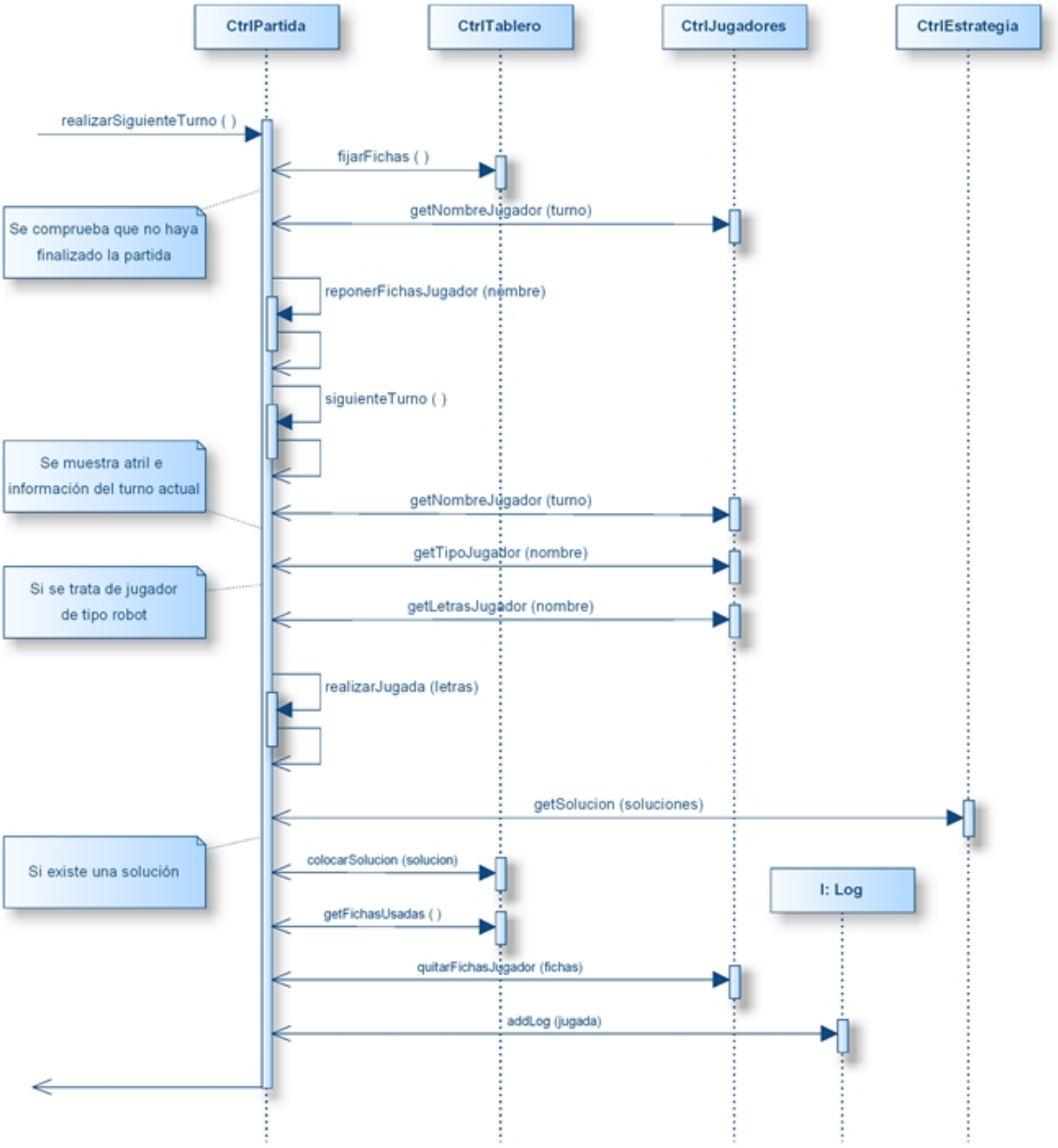


Figura 5.27. Diagrama de secuencia realizarSiguienteTurno

5.5.5.6. DIAGRAMA: REALIZARJUGADA

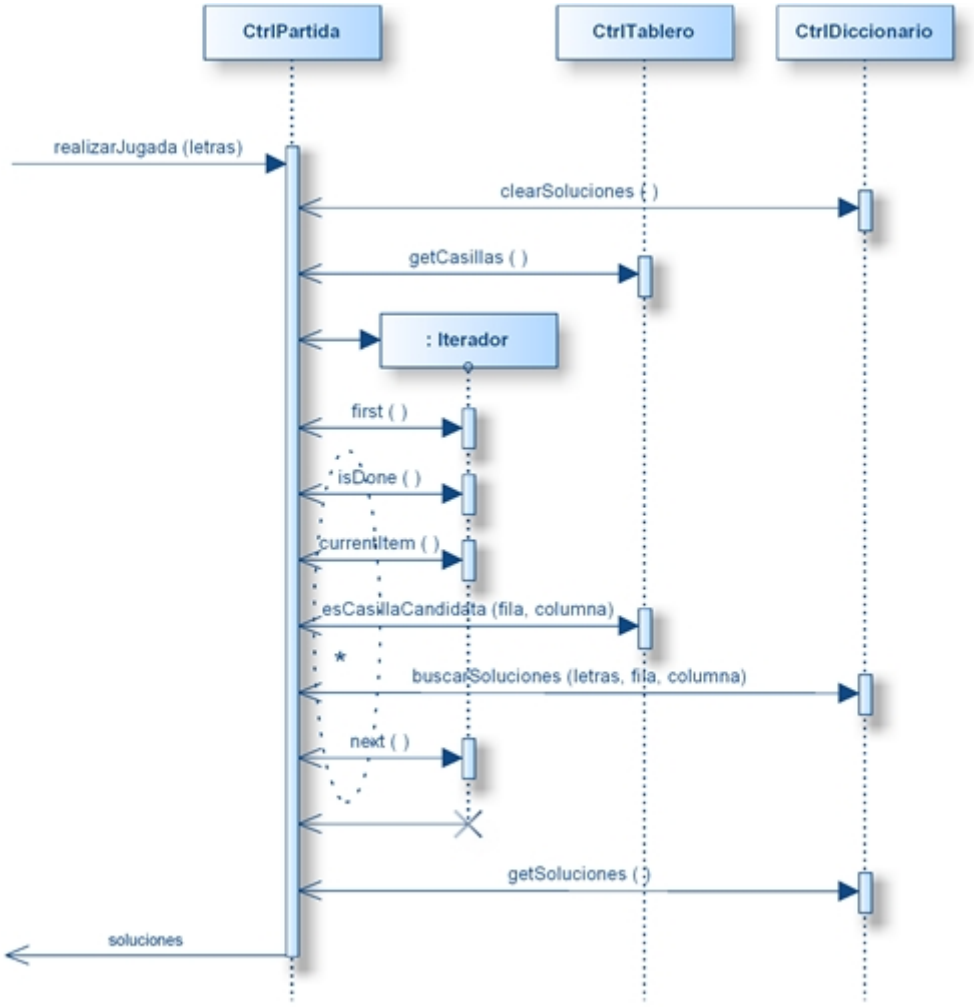


Figura 5.28. Diagrama de secuencia realizarJugada

5.5.5.7. DIAGRAMA: GETSOLUCION

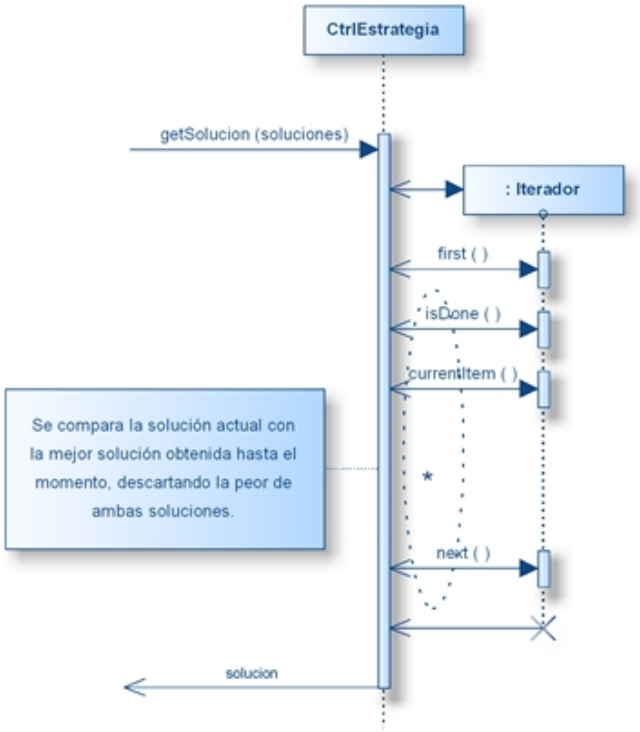


Figura 5.29. Diagrama de secuencia getSolucion

5.5.5.8. DIAGRAMA: REPONERFICHASJUGADOR

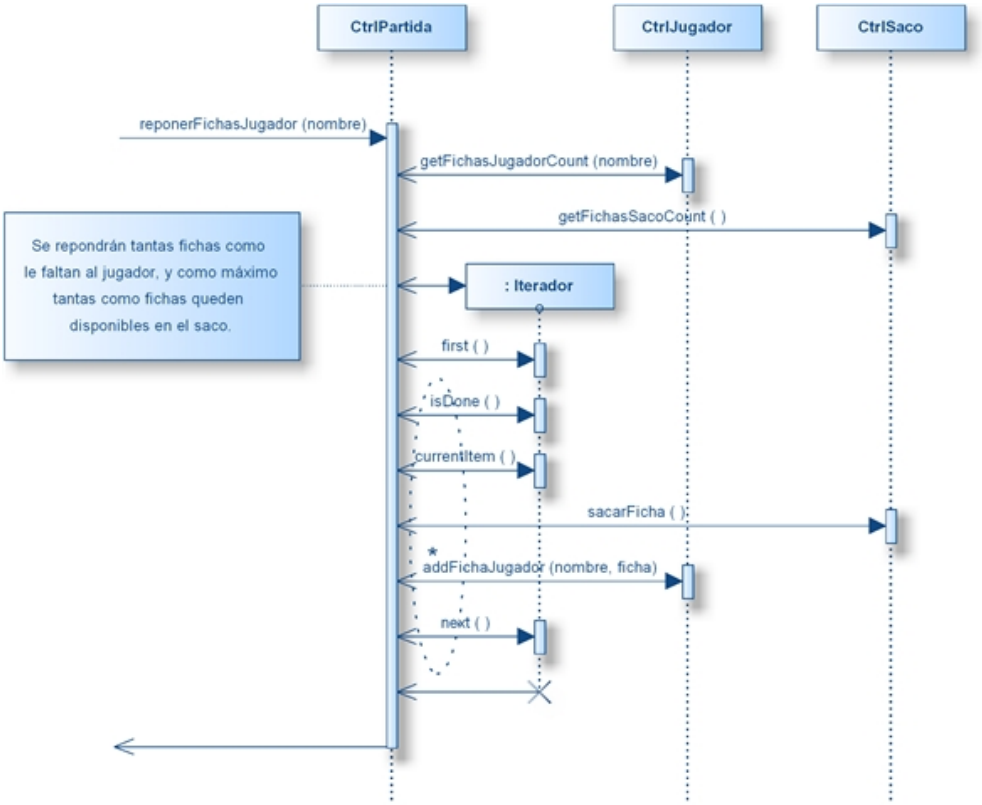


Figura 5.34 Diagrama de secuencia reponerFichasJugador

6. IMPLEMENTACIÓN

6.1. Introducción 152
6.2. Capa de presentación 152

6.2.1. Implementación menús 153
6.2.2. Implementación pantallas 155
6.2.3. Implementación tablero 156

6.3. Capa de datos 160

6.3.1. Acceso ficheros de texto 160
6.3.2. Acceso ficheros binarios 162
6.3.3. Directorio de ficheros 162

6.4. Capa de dominio 163

6.4.1. Diccionario 163
6.4.2. Estrategia 176

6.5. Pruebas del sistema 177

6.5.1. Pruebas unitarias 179
6.5.2. Pruebas de integración 180
6.5.3. Pruebas del sistema 180

6.1. INTRODUCCIÓN

El objetivo de este capítulo es la implementación del diseño obtenido en la etapa anterior. Con el diseño se ha obtenido un sistema que estaba construido pero no implementado. La codificación de un sistema con una tecnología concreta permite poderlo ejecutar y llevar a la practica todo el desarrollo realizado en las etapas anteriores.

En este capítulo de la memoria se explicarán los aspectos más destacables de la fase de implementación del sistema, es decir, la creación de la capa de presentación, la capa de datos, y la capa de dominio.

Para finalizar, se incluye la explicación de la etapa de pruebas del sistema, ya que esta etapa se ha ido realizando de forma paralela a la de implementación.

6.2. CAPA DE PRESENTACIÓN

Para la implementación de la capa de presentación se ha utilizado *Java Swing* mediante la herramienta *Eclipse.* Tal y como se ha explicado en el diseño del sistema, se ha estructurado la pantalla principal diversas zonas bien definidas. La importancia de la capa de presentación no es otra que permitir al usuario acceder a la información y funcionalidades que ofrece la aplicación de una manera lo más sencilla y cómodamente posible.

Lo que se explicará en este apartado es cómo se ha implementado la barra de menús para ofrecer una total accesibilidad al usuario a todas las funcionalidades ofrecidas, y cómo se han implementado las diversas pantallas siguiendo el ejemplo de una de ellas, en concreto la pantalla de partida, ya que la metodología seguida en todas ellas ha sido la misma, a excepción de la pantalla de juego en la que se muestra el tablero en un escenario 3D y el atril del jugador. La implementación de dicha pantalla se explicará en detalle más adelante.

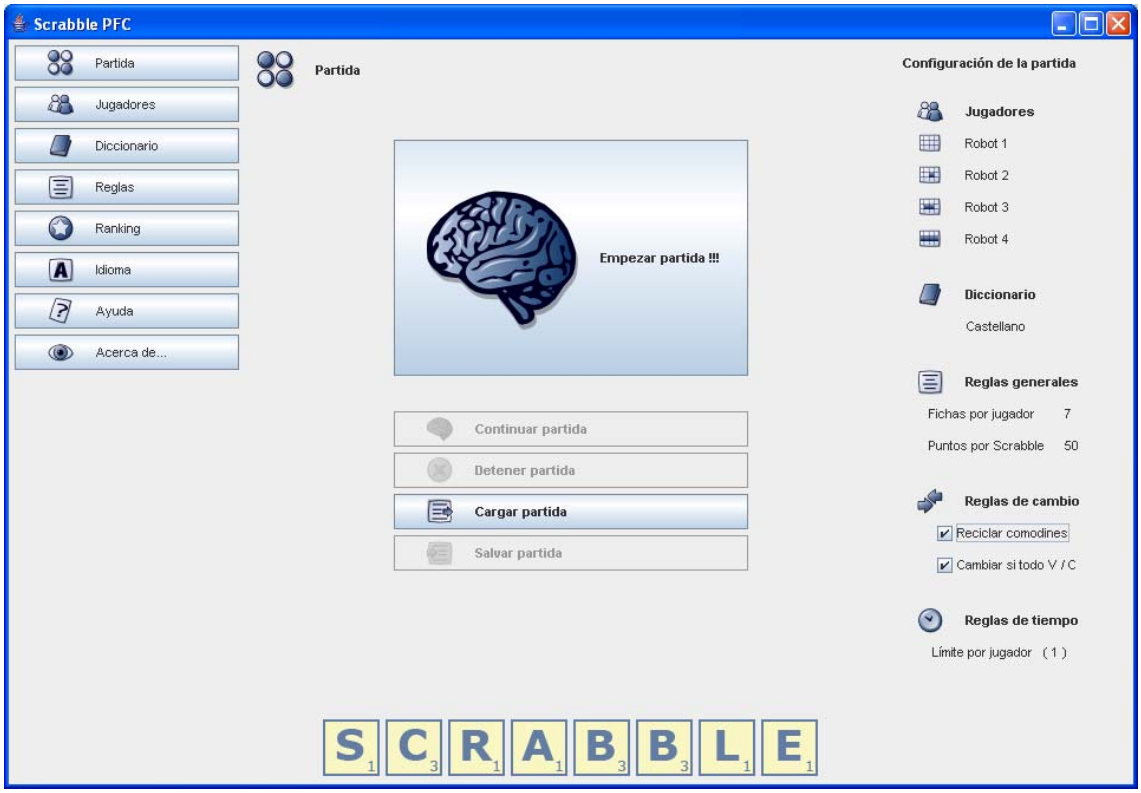


Figura 6.1 Pantalla de partida

6.2.1. IMPLEMENTACIÓN MENÚS

Las partes de código mostrado en esta sección pertenecen a la clase de la capa de presentación *PanelToolbar.java*. Para poder acceder a cualquier pantalla desde el panel de menús se ha responsabilizado a la clase Scrabble, tal y como se ha comentado en el apartado de diseño en este mismo documento, de conocer la manera de mostrar en la zona central de la ventana principal de la aplicación cualquiera de las pantallas. Se indicará entonces cómo comunicarse con la clase Scrabble a la clase *PanelToolbar*, y de esta manera poder ordenar mostrar una u otra pantalla según sea necesario.

```
public PanelToolBar(Scrabble pScrabble, int pWidth, int pHeight)
{
        super(pWidth, pHeight);
        scrabble = pScrabble;

      // Inicializamos interficie
        initGUI();
}
```

Figura 6.2. Se indica la clase Scrabble por referencia

A cada uno de los botones creados en el menú se le añade un *Listener* encargado de estar pendiente de recibir cualquier evento de acción como puede ser un *click* sobre el propio botón para ejecutar así el código indicado, en nuestro caso, el código encargado de avisar a la clase Scrabble de mostrar la correspondiente pantalla. Podemos ver como ejemplo, el código implementado en la función *clickPartida* del botón de *Partida* para ver cómo se realiza la invocación a la clase Scrabble para indicarle que muestre la pantalla correspondiente.

```
private void clickPartida()
{
	scrabble.setPanelCuerpo(new PanelPartida(scrabble));
}
```

Figura 6.3. Código ejecutado al recibir un *click* sobre el botón *Partida*

La función *setPanelCuerpo* de la clase Scrabble es la encargada de mostrar los paneles pasados como parámetro correspondientes a las diferentes pantallas, en este ejemplo, el panel de la pantalla de partida.

Una vez han sido añadidos todos los botones en el panel de menú, el resultado tendrá un aspecto como el mostrado en la siguiente figura.

Figura 6.4. Barra de menús

6.2.2. IMPLEMENTACIÓN DE LAS PANTALLAS

Las partes de código mostrado en esta sección pertenecen a la clase de la capa de presentación *PanelPartida.java*.

Toda pantalla extiende de la clase *ScrabblePanel*, que contiene todas las propiedades visuales comunes entre las diferentes pantallas como pueden ser el tipo de fuente, el tamaño de fuente, color de fuente, color de fondo, etc, que a su vez extiende la clase *JPanel* propia de la plataforma *Java Swing*.

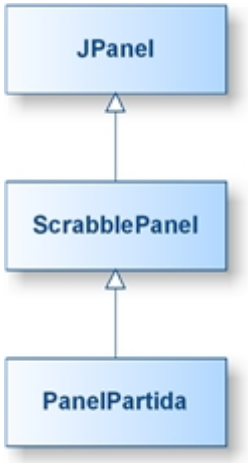


Figura 6.5. Herencia clases de presentación

A cada una de las pantallas se les indicará como comunicarse con la clase Scrabble para poder así comunicarse con la capa de dominio, ya que es la única que conoce cómo hacerlo. Tal y como se explicó en el diseño, la responsabilidad de comunicación con la capa de dominio para enviar o recibir información era de la clase Scrabble de la capa de presentación.

En el método *initGUI( )* contenido en cada constructora tiene lugar la creación de todos los componentes que van a visualizarse a través de los cuales tendrá lugar la interacción entre el usuario y la aplicación.

```
public PanelPartida(Scrabble pScrabble)
{
        super();
        scrabble = pScrabble;

        // Inicializamos interficie
        initGUI();
}
```

Figura 6.6. Método constructor del panel de partida

Todas las pantallas siguen la misma estructura para proporcionar un aspecto totalmente homogéneo en la aplicación. La parte central de la ventana principal estará compuesta por el panel *PanelCuerpo* que es el encargado de contener los paneles correspondientes a las diversas pantallas. Cada una de las pantallas estará compuesta por un panel *PanelNull* que se situará siempre centrado dentro de su contenedor *PanelCuerpo*, independientemente del tamaño de *PanelNull*. Todos los componentes creados para ofrecer la accesibilidad al usuario a las funcionalidades de la aplicación se añadirán a *PanellNull* de manera que la posición de éstos será completamente absoluta respecto a su contenedor *PanelNull*, obteniendo así de cara al programador un control total sobre el posicionamiento de los diversos componentes.

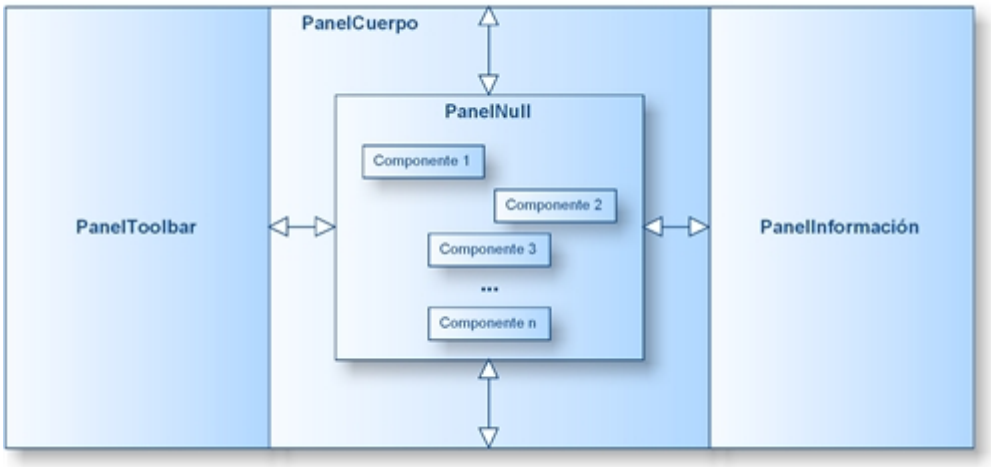


Figura 6.7. Estructuración de los paneles

6.2.3. IMPLEMENTACIÓN DEL TABLERO

Para la implementación del tablero se han utilizado librerías adicionales destinadas a la visualización 3D. Estas librerías pertenecen al proyecto de código abierto *JOGL* (*Java OpenGL*). *JOGL* proporciona un total acceso a la especificación de la API de *OpenGL 2.0*, y se integra perfectamente con *AWT* y *Swing*.

A continuación se explicarán las principales partes del código implementado para la obtención del tablero en 3D, así como para la completa interacción del usuario con éste, de manera que puedan ser los mismos usuarios los encargados de colocar, quitar y mover directamente las fichas sobre las casillas del tablero.

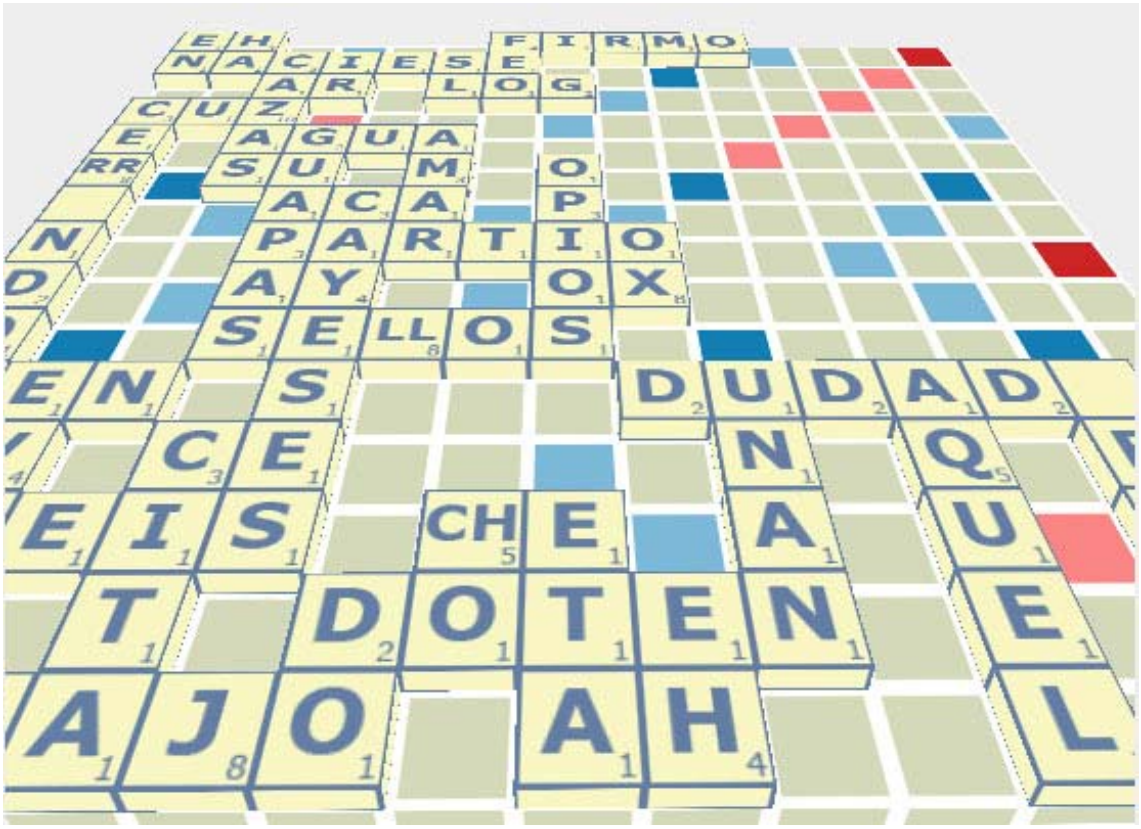


Figura 6.8. Tablero de juego 3D

6.2.3.1. INICIALIZACIÓN DEL TABLERO

En la inicialización del tablero nos encargaremos de inicializar las propiedades de visualización 3D (*GL_DEPH_TEST, GL_CULL_FACE, GL_SMOOTH, etc)*. Adicionalmente, se ha optado que sea en este punto, donde tenga lugar la carga de las texturas a utilizar. Texturas que se corresponderán a cada una de las letras contenidas en las fichas del saco del idioma de juego seleccionado, así como las texturas comunes para todos los idiomas como son las propias de las casillas del tablero. Estas texturas, serán cargadas en una Hash, de manera que el acceso a ellas posteriormente en el *display* del tablero, momento de pintado del tablero, sea en tiempo mínimo y constante. Para las letras de las fichas, se almacenarán dos texturas de igual forma y tamaño, pero diferente color. De esta forma, se consigue ayudar a diferenciar al usuario qué fichas son las recién colocadas en el turno actual, y qué fichas son fijas en el tablero colocadas en turnos anteriores.

Figura 6.9. Ejemplo de texturas de ficha

6.2.3.2. DISPLAY

El método *display*, es el encargado de pintar continuamente la escena 3D que tiene lugar. Para ello, lo primero que se deberá hacer en cada repintado, es limpiar la escena para poder dibujar a continuación la posición actual de cada uno de los objetos a pintar.

Antes de proceder al pintado de cada uno de los objectos, se calculará si el usuario se encuentra interaccionando con el tablero. Dicha interacción tan solo podrá ser con una casilla del tablero a la vez y en el único caso en que el usuario tenga una ficha seleccionada. Para poder obtener la casilla en la que se encuentra el ratón, a cada una de las casillas que forman el tablero se le asigna un identificador, en el caso que nos ocupa el identificador de cada casilla será el número de casilla dentro del tablero que ocupa ésta. De esta manera, una vez se obtiene el identificador de la casilla de interacción, en caso de existir, se podrá tratar dicha casilla de manera que sea posible dibujar sobre ella la ficha que el usuario ha seleccionado para ser colocada o movida sobre el tablero, dando así una total sensación de movimiento y control por parte del usuario sobre la disposición de las fichas sobre el tablero de juego.

```
// Inicializamos hits a 0
int hits = 0;

// Definimos el modo de render como SELECCION
gl.glRenderMode(GL.GL_SELECT);
fijarPerspectiva(gl, true);

// Colocamos la camara en su posición actual
gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();

gl.glTranslatef(0, 0, -distancia);
gl.glRotatef(rotacionZ, 0, 0, 1);
gl.glRotatef(rotacionY, 0, 1, 0);
gl.glRotatef(rotacionX, 1, 0, 0);
gl.glTranslatef(0, -ancho, 0);

// Inicializamos nombres para identificar las zonas de interacción
gl.glInitNames();


// Se define una zona de interacción por cada casilla del tablero
int i = 0;
for (int c = 0; c < columnas; c++)
       for (int r = 0; r < filas; r++)
       {
```

```
                try
                {
                    // Como id. establecemos el número de casilla dentro del tablero
                    gl.glPushName(i);
                    dibujarCasilla(gl, c, r, true);
                    gl.glPopName();
                }
                catch (Exception e) {}
                i++;
        }

// Obtenemos el numero de hits (interacciones), y pasamos a RENDER
hits = gl.glRenderMode(GL.GL_RENDER);

// Obtenemos el identificador de la interacción (id. de la casilla)
hit = bufferSel.get(3);
```

Figura 6.10. Parte del código del método *display* dedicado a obtener la casilla de interacción

Seguidamente se procederá al pintado de cada una de las casillas y de las fichas dispuestas sobre éstas. Así como al pintado de la ficha seleccionada por el usuario sobre la casilla de interacción en caso de ser necesario.

```
// Colocamos la camara en su posición actual
gl.glMatrixMode(GL.GL_MODELVIEW);
gl.glLoadIdentity();

gl.glTranslatef(0, 0, -distancia);
gl.glRotatef(rotacionZ, 0, 0, 1);
gl.glRotatef(rotacionY, 0, 1, 0);
gl.glRotatef(rotacionX, 1, 0, 0);
gl.glTranslatef(0, -ancho, 0);

// Dibujamos cada una de las casillas del tablero
for (int c = 0; c < columnas; c++)
        for (int r = 0; r < filas; r++)
                dibujarCasilla(gl, c, r, false);

// Si existen interacción y hay una ficha seleccionada
if (hits > 0 && letraSeleccionada != null)
{

        // Ocultamos puntero del ratón
        Image cursorImage = Toolkit.getDefaultToolkit().getImage("");

        Cursor blankCursor =
        Toolkit.getDefaultToolkit().createCustomCursor(cursorImage, new Point(0,0), "");
        setCursor( blankCursor );
```

```
        // Obtenemos columna y fila de la casilla a partir del id.
        int c = hit / columnas;
        int r = hit % filas;

        // Dibujamos la ficha seleccionada sobre la casilla objetivo
        dibujarCasillaSeleccionada(gl, c, r, letraSeleccionada);
}
else setCursor(new Cursor(Cursor.HAND_CURSOR));
```

Figura 6.11 Parte del código del método *display* dedicado a dibujar el tablero y sus fichas colocadas

6.3. CAPA DE DATOS

La aplicación trabaja sobre ficheros de texto y ficheros binarios para poder garantizar la persistencia de los datos. A continuación se detallará la implementación de las clases que tienen la responsabilidad de obtener y actualizar dichos ficheros según sean de tipo texto, o binario.

6.3.1. ACCESO FICHEROS DE TEXTO

El acceso a los ficheros de texto en toda la aplicación tiene lugar a través de las clases que la plataforma *Java* nos proporciona para tal efecto, como son:

- `java.io.File`
- `java.io.BufferedReader`
- `java.io.BufferedWriter`
- `java.io.FileReader`
- `java.io.FileWriter`

La lectura del contenido de cada uno de ellos será siempre secuencial, línea a línea. Para mostrar un ejemplo, nos fijaremos en la clase de la capa de datos encargada de obtener y mantener el listado de palabras de un diccionario *SGFDiccionario*. Para realizar la carga de todas las palabras válidas, dicha clase se comunicará con la clase de dominio encargada de representar el diccionario de manera que le indicará todas aquellas palabras que deberá contener.

```
public SGFDiccionario(Diccionario pDiccionario)
{
        // Obtenemos diccionario de dominio por referencia
        diccionario = pDiccionario;
}
```

Figura 6.12. Constructora de *SGFDiccionario* que recibe la referencia a la clase de dominio *Diccionario*

```
public void cargarDiccionario(String pIdioma)
{
	try
	{
		// Acceso al fichero físico del diccionario
		File file = new File(Configuracion.PATH_IDIOMAS + pIdioma+
			"/diccionario");

		// Creamos buffer de lectura del fichero de diccionario
		FileReader fileReader = new FileReader(file);
		BufferedReader bReader = new BufferedReader(fileReader);

		// Lectura linea a linea del fichero de diccionario
		String l;
		while ((l = bReader.readLine()) != null)
		{
			// Si la linea contiene palabra la incluimos
			if (!l.trim().equals(""))
				diccionario.addPalabra(l);
		}

		// Cerramos los accesos abiertos al fichero
		bReader.close();
		fileReader.close();
	}
	catch (Exception e) { throw e; }
}
```

Figura 6.13. Método encargado de cargar el listado de palabras válidas en el diccionario

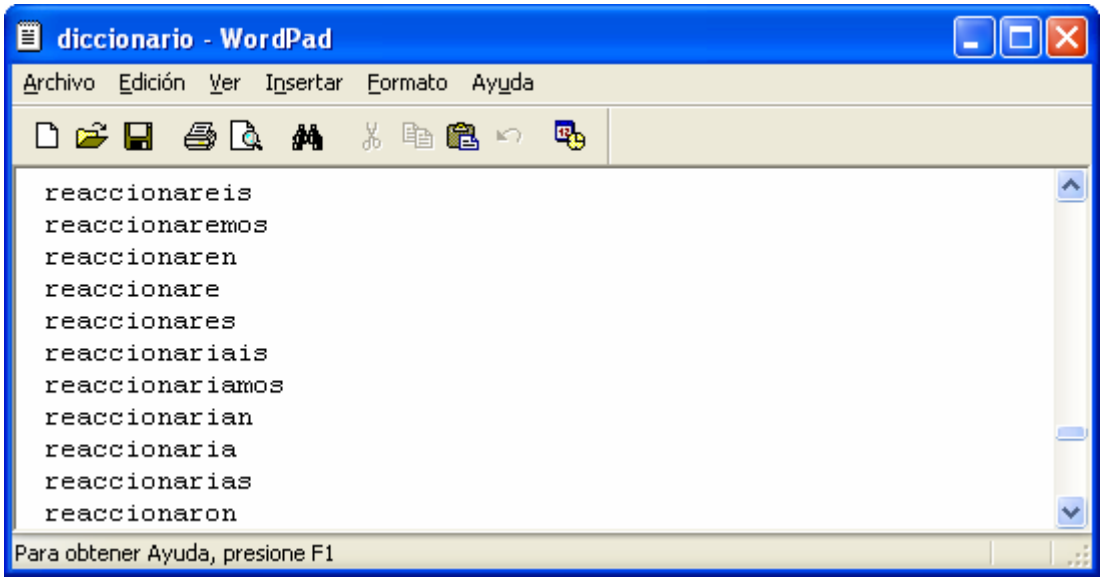


Figura 6.14. Parte del contenido del diccionario *Castellano*

6.3.2. ACCESO FICHEROS BINARIOS

La propia plataforma *Java* en que la aplicación está desarrollada nos ofrece una interfaz *Serializable* a través de la cual se puede realizar una conversión de objeto a fichero y viceversa de forma totalmente transparente para el programador, mediante los métodos *writeObject* y *readObject*.

```
private void writeObject(java.io.ObjectOutputStream out)
     throws IOException

 private void readObject(java.io.ObjectInputStream in)
     throws IOException, ClassNotFoundException;
```

Figura 6.15. Funciones *writeiObject* y *readObject* de la interfaz *Serializable*

6.3.3. DIRECTORIO DE FICHEROS

El directorio de los ficheros de datos de la aplicación se estructura de la siguiente manera:

- **Configuración.** En la carpeta configuración se encontrarán los ficheros de configuración general y configuración de reglas.

- **Iconos.** En la carpeta iconos se encontrarán todos aquellos iconos necesarios para la aplicación.

- **Idiomas.** En la carpeta idiomas se encontrarán los ficheros de *diccionario*, *saco* y *textos (textos de localización)*. Así como el fichero de ayuda, y las texturas correspondientes a las letras del idioma de juego.

- **Jugadores.** En la carpeta jugadores se encontrarán todos los ficheros binarios correspondientes a los perfiles de jugador de la aplicación.

- **Saves.** Carpeta por omisión donde se encontrarán los ficheros de partidas salvadas.

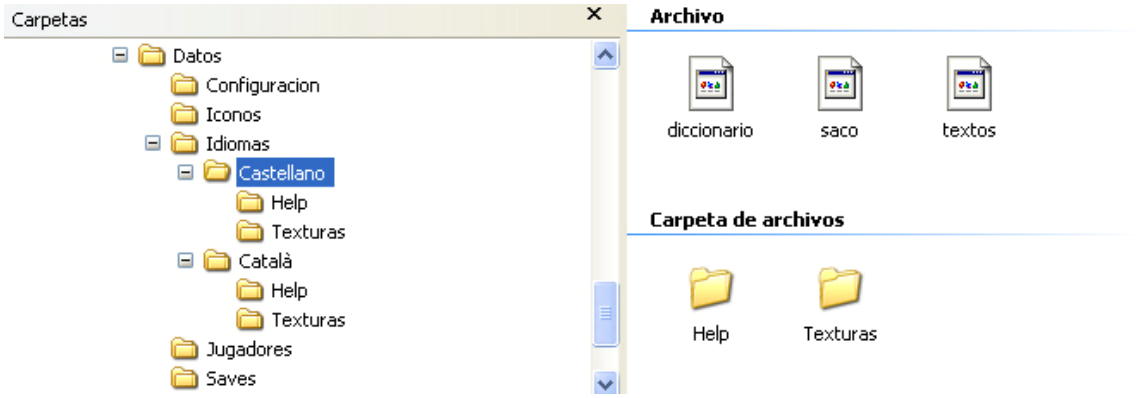


Figura 6.16. Directorio de los ficheros de datos

6.4. CAPA DE DOMINIO

Respecto a la implementación de la capa de dominio, este documento se centrará en la explicación de aquellos puntos clave de la aplicación como son:

- Implementación del diccionario
- Implementación de la estrategia

6.4.1. DICCIONARIO

La implementación del diccionario es el punto más importante de toda la aplicación debido a que se deberá implementar una estructura capaz de almacenar el listado completo de palabras válidas del idioma de juego seleccionado de manera que ofrezca dos funcionalidades principales:

- Validación de palabras (por parte del usuario)
- Obtención de todas las posibles jugadas en un tiempo mínimo

De estas dos funcionalidades, se ha centrado el estudio en la obtención en un tiempo mínimo de todas las posibles jugadas, teniendo siempre presente que la implementación resultante posibilite la necesidad de validar palabras. Posteriormente, tal como se detallará, del conjunto de posibles jugadas se seleccionará la jugada óptima a través del modulo de *Estrategia*.

¿Por qué es tan importante centrarnos en la obtención de todas las posibles jugadas en un tiempo mínimo?

Bien, para justificar esta decisión veremos que sin una buena estructura que posibilite una gran optimización en la obtención de las jugadas, la gran cantidad de cálculos necesarios imposibilita la implementación del diccionario. Los siguientes cálculos realizados son de manera global ya que se generaliza una posible situación de cálculo de una casilla a las 224 restantes del tablero, pero para obtener una idea de la cantidad de posibles jugadas a tener en cuenta ya nos es válido.

La finalidad de un jugador en su turno es la de formar la palabra a partir de las fichas disponibles en su atril más, como mínimo, una de las fichas colocadas sobre el tablero en turnos anteriores, respetando siempre el cumplimiento de todas las reglas de juego.

El número habitual de fichas en un atril será de siete, exceptuando ciertos idiomas en que es posible que cierto número varíe, o cuando en el saco no queden más fichas por coger. Supondremos que únicamente se utiliza una de las fichas ya colocadas en el tablero a la hora de obtener las posibles jugadas desde cualquier casilla del tablero. Por lo que obtendremos un total de 109.592 posibles jugadas que será necesario comprobar que formen una palabra válida.

$V_{n,r} = n! / (n - r)!$
(Variaciones de n elementos cogidos de r en r sin repetición)

$V_{8,8} = 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 =$ **40.320**

$V_{8,7} = 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 =$ **40.320**

$V_{8,6} = 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 =$ **20.160**

$V_{8,5} = 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 =$ **6.720**

$V_{8,4} = 8 \cdot 7 \cdot 6 \cdot 5 =$ 1**.680**

$V_{8,3} = 8 \cdot 7 \cdot 6 =$ **336**

$V_{8,2} = 8 \cdot 7 =$ **56**

40.320 + 40.320 + 20.160 + 6.720 + 1.680 + 336 + 56 = **109.592**

Si dicho cálculo se realiza para cada una de las 225 casillas del tablero de juego, se deberán realizar 24.658.200 comprobaciones.

109.592 · 225 = **24.658.200**

Si se tienen en cuenta la existencia de los comodines, podemos suponer que en el atril una de las fichas sea un comodín, o incluso dos de las fichas sean comodín, en cuyo caso el número de cálculos aumentará de forma considerable ya que cada comodín puede ser utilizado por cada una de las letras que forman el saco. Suponiendo por ejemplo, que en el saco utilizado existan 28 letras diferentes, el número de comprobaciones a realizar en caso de poseer un comodín sería de 690.429.600, y en caso de poseer ambos comodines sería de 19.332.028.800.

$24.658.200 \cdot 28 = \mathbf{690.429.600} \cdot 28 = \mathbf{19.332.028.800}$

En el caso en que la partida estuviera lo suficientemente avanzada como para que la gran mayoría de jugadas a realizar deban tener en cuenta no una, sino dos fichas colocadas sobre el tablero tan solo anotar que el número de comprobaciones en caso de poseer un comodín ya sería de 6.214.320.000, y en caso de poseer ambos comodines sería de 174.000.960.000.

$V_{9,9} = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = \mathbf{362.880}$

$V_{9,8} = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 = \mathbf{362.880}$

$V_{9,7} = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = \mathbf{181.440}$

$V_{9,6} = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 = \mathbf{60.480}$

$V_{9,5} = 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 = \mathbf{15.120}$

$V_{9,4} = 9 \cdot 8 \cdot 7 \cdot 6 = \mathbf{3.024}$

$V_{9,3} = 9 \cdot 8 \cdot 7 = \mathbf{504}$

$V_{9,2} = 9 \cdot 8 = \mathbf{72}$

$362.880 + 362.880 + 181.440 + 60.480 + 15.120 + 3.024 + 504 + 72 =$

$\mathbf{986.400}$

$986.400 \cdot 225 = \mathbf{221.940.000}$

$221.940.000 \cdot 28 = 6.214.320.000 \cdot 28 = \mathbf{174.000.960.000}$

Se puede observar, como una solución para la implementación del diccionario que no optimice realmente todo el proceso de obtención de las posibles jugadas, imposibilita totalmente la realización de la aplicación.

A continuación se detallarán los dos estudios realizados para la implementación del diccionario durante el desarrollo del proyecto. El primero de ellos fallido en el momento de contemplar la posibilidad de disponer de fichas de tipo comodín. Mientras que el segundo estudio, más detallado, ha podido ser llevado a cabo satisfactoriamente.

Recordar de nuevo, que los cálculos han sido realizados a la alza, generalizando un caso particular de una posible casilla a las 225 casillas totales del tablero, pero que aún así la cantidad de posibles jugadas a comprobar su validez es realmente abismal.

6.4.1.1. PRIMER ESTUDIO REALIZADO

De este primer estudio realizado para implementar el diccionario, detallaremos los puntos más importantes para ver como un mal planteamiento inicial acaba comportando resultados negativos.

El primer estudio que se realizó fue enfocado en la utilización de anagramas para evitar el cálculo de todas las variaciones posibles para formar y obtener la lista de posibles jugadas. Realmente, parecía un buena idea, pero como más adelante se vió, tenía ciertas carencias que se detallarán a continuación.

El enfoque en anagramas se basaba en encontrar el anagrama común de todas aquellas palabras incorporadas en el diccionario. Este anagrama común era el formado por las letras que formaban cada una de las palabras ordenadas alfabéticamente. Por ejemplo, el anagrama común para la palabra *camión* es *acimno*. Por lo que todas las palabras formadas por las letras *acimno* tendrían esta propiedad común, como por ejemplo puede ser la palabra *camino*. De esta manera, si juntábamos las letras del atril con todas aquellas letras de las fichas ya colocadas en el tablero con las que podía cruzarse desde la casilla tratada, y las ordenábamos alfabéticamente formando así el anagrama común, obteníamos todas las palabras formadas por una correspondencia directa. Dicha correspondencia era guardada en una Hash donde el anagrama común era la clave, y la lista de palabras formadas, el valor.

Llegados a este punto se identificaron tres factores negativos a tener en cuenta, que se intentaron solventar y optimizar:

- Para obtener el conjunto total de posibles jugadas, se debían realizar combinaciones sobre el anagrama común.

- Se debían comprobar todas las casillas.

- Una vez obtenido el conjunto de posibles jugadas para una casilla, se debía comprobar una a una que realmente fuera posible realizar la jugada debido a las fichas ya colocadas en el tablero.

- **Problema 1.** Para obtener el conjunto total de posibles jugadas desde una casilla concreta, se debían realizar combinaciones sobre el anagrama a tratar. Es decir, dado el anagrama común únicamente se podía obtener el conjunto de posibles jugadas de la misma longitud que éste. No era posible obtener las posibles jugadas formadas por subanagramas sin necesidad de combinar qué letras del anagrama común se debían ir omitiendo, para ir añadiendo las nuevas palabras encontradas de menor longitud al conjunto de posibles jugadas.

- **Solución.** Se pudo resolver de manera que tan sólo fuera necesario calcular para un anagrama común todas las posibles palabras formadas por todos sus subanagramas la primera vez que se consultara durante el transcurso de la partida. Para ello se pasó a memorizar todas las jugadas de igual o menor longitud en los anagramas desde cuales era posible obtenerlas. Por el contrario, comportaba un crecimiento en recursos de memoria considerable, ya que consistía en repetir información para todas aquellas claves de Hash que iban siendo consultadas.

- **Problema 2.** Se debían comprobar desde cada una de las 225 casillas del tablero si era posible desde cada una de ellas alcanzar alguna de las fichas ya dispuestas sobre el tablero para proceder a la obtención de las posibles jugadas.

- **Solución.** Aunque se llegó a la idea de marcar aquellas casillas desde las cuales era posible formar una posible jugada, en el momento en que los atriles de los jugadores se quedaran con menos fichas de las indicadas debido a la falta de fichas en el saco, ciertas casillas estarían marcadas erróneamente ya que la longitud máxima de las posibles palabras formadas habría disminuido, por lo que sería necesario un calculo auxiliar. Aunque sí se observó que tan solo era necesario comprobar aquellas casillas situadas a la izquierda o arriba de alguna casilla que tuviera una ficha colocada debido a la dirección de escritura de las palabras, de izquierda a derecha, o de arriba abajo.

- **Problema 3.** Una vez obtenido el conjunto de posible jugadas era necesario un post-procesado para descartar todas aquellas soluciones que no coincidieran con las fichas ya colocadas en el tablero. Esto era debido a qué en la obtención de las palabras formadas a partir de un anagrama no se tenía en cuenta si alguna de las letras que las formaban debía aparecer fija en una posición concreta.

- **Solución 3.** Para éste problema no existía ningún tipo de solución ya que era una comprobación que no se podía omitir para ninguna de las palabras del conjunto de posibles jugadas.

Una vez sorteados estos contratiempos, la obtención de todas las posibles jugadas, aún no siendo todo lo eficiente que pudiera ser, se realizaba en tiempos bajos, aunque para ciertas situaciones eran algo más elevados, pero aún así no impedían la fluidez de juego.

¿Qué imposibilitó realmente continuar con este enfoque de implementación?

Pero el problema realmente serio apareció cuando se quiso incluir en la implementación del diccionario la utilización de los comodines. Ya que como se ha podido observar en los cálculos anteriormente descritos, la inclusión de comodines repercute seriamente en el número de cálculos a tener en cuenta, sobretodo en el caso extremo (pero posible) en que en un mismo atril puedan encontrarse ambos comodines. El aumento en el número de cálculos provocaba que los tiempos de obtención de soluciones fueran exageradamente elevados al tratarse con algún comodín, por no contar si se producía la situación en que en un mismo atril se encontraban ambos comodines. En este momento, me di cuenta de que necesitaría un enfoque radicalmente distinto si quería conseguir una implementación realmente eficiente en tiempo y memoria. Por lo que se empezó desde cero una nueva investigación sobre el tema.

6.4.1.1. SEGUNDO ESTUDIO REALIZADO

Una vez se vieron los principales problemas que existían en la implementación del diccionario se decidió empezar desde cero intentando buscar un nuevo enfoque, radicalmente distinto que por definición evitara sino todos, el mayor número de problemas ya conocidos:

- Realizar variaciones para obtener todas las posibles palabras a jugar.
- Tratar cada una de las casillas del tablero.
- Comprobar posteriormente las jugadas obtenidas ya que aún podían ser no válidas.
- La utilización del comodín comportaba un exagerado número de nuevas posibilidades.

Finalmente, tras varios días de investigación y desarrollo, se consiguió llegar a una solución realmente eficaz basada en el algoritmo que presentaron Appel y Jacobson para generar todas las posibles jugadas.

Este algoritmo está basado en un enorme autómata finito a través del cual puede formarse cada una de las palabras de un idioma. Esta gran estructura fue llamada *Directed Acyclic Word Graph (DAWG)*. Estructuras equivalentes al *DAWG* eran muy utilizadas para representar grandes léxicos para diccionarios, autocorrección de palabras, etc. Aunque esta estructura era muy propicia para aplicaciones de éste tipo no era la más eficiente para generar las posibles jugadas en una partida de Scrabble.

La clave reside en observar que todo el léxico contenido en una estructura de tipo *DAWG*, es generado de izquierda a derecha, tal y como seria escrito o leído por la definición de su idioma. Exactamente esta generación de izquierda a derecha, es la que imposibilita la eficiencia en la generación de palabras en el Scrabble, ya que la naturaleza de la generación de palabras en éste no es unidireccional, sino bidireccional. Más concretamente, consiste en anclar las nuevas palabras en palabras ya existentes. Por lo que la generación se inicia en la letra ya dispuesta en el tablero y no en la letra inicial de la palabra.

A partir de aquí, se construyó una estructura similar al *DAWG*, donde el inicio de la formación de cada palabra puede ser cada una de las mismas letras que la forman, pudiendo ser de esta manera anclada desde cualquier punto de la palabra. A esta estructura se le llamó *GADDAG*.

Como contra del *GADDAG* respecto al *DAWG*, es que ocupa aproximadamente cinco veces más memoria. Pero lo que realmente lo hace el más indicado es que el *GADDAG* es más del doble de rápido que el *DAWG* al estar enfocada la construcción de palabras de manera muy similar a como se forman en una partida de Scrabble.

Por lo tanto, basándome en la estructura *GADDAG*, el algoritmo implementado en este proyecto para la realización del diccionario consiste en la formación de palabras a partir de una letra de anclaje de manera que primero se formará el sufijo a partir de dicho anclaje, para posteriormente formar el prefijo, con lo que todo unido se formará la palabra.

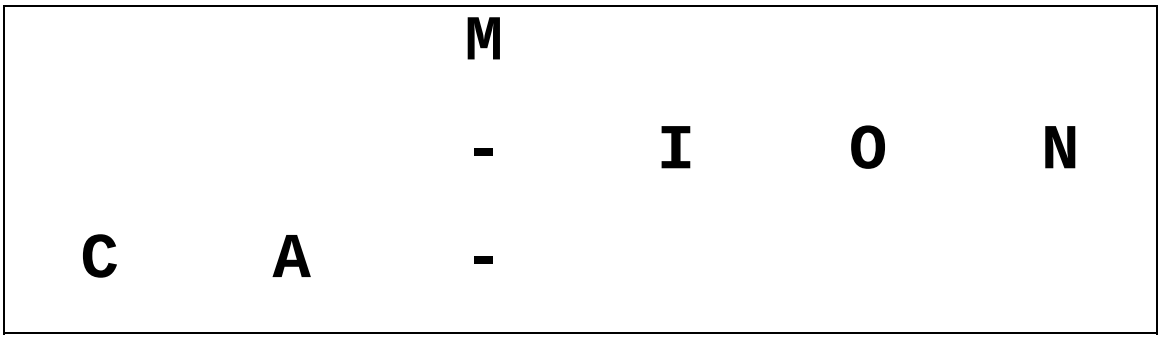


Figura 6.17 Formación de la palabra CAMIÓN, mediante sufijo y prefijo a partir de la letra de anclaje M.

La estructura del diccionario a implementar, estará por lo tanto, basada en la formación de palabras a partir de cada una de las letras que las forman sin importar qué letra se toma como inicio ya que conoceremos el sufijo y prefijo a formar a partir de cada una de ellas. De esta manera, se puede asegurar ya que tan solo se tratarán aquellas casillas desde las cuales se pueda realmente iniciar una jugada.

Como se puede observar en la siguiente figura, las casillas candidatas no son las que contienen fichas, sino todas las contiguas a éstas y que se encuentren libres, ya que no tan solo pueden formarse palabras cruzando con la letra de anclaje, sino también formando lo que se denomina una "pared".

Figura 6.18 Área de casillas candidatas

Respecto al primer estudio, se consiguió realmente tratar únicamente aquellas casillas desde las cuales se puede iniciar la formación de las jugadas. Por lo que el estudio actual ya dispone de un punto a su favor respecto al primer estudio.

Para definir la estructura final del diccionario, primero definiremos como se estructurará una única palabra en memoria. Siendo la estructura del diccionario la unión de todas las palabras.

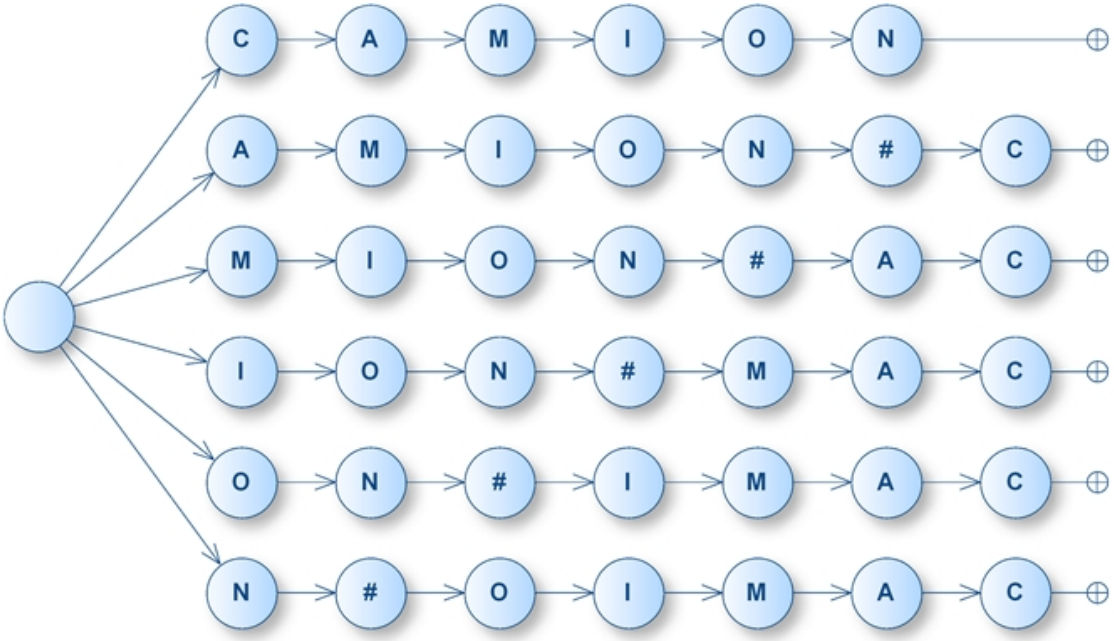


Figura 6.19 Estructura básica de la palabra CAMIÓN en el diccionario

En esta estructura resultante se puede comprender fácilmente la idea desarrollada hasta el momento para la formación de palabras dentro del diccionario. Vemos que el símbolo # se ha utilizado para indicar en qué momento deja de formase el sufijo a partir del anclaje, para pasar a formar el prefijo en orden inverso, ya que se construirá de derecha a izquierda a partir del anclaje. Vemos ahora a partir de la figura, cómo se ha conseguido dejar de lado la idea de formación de palabras de forma convencional, de izquierda a derecha, para pasar a una formación bidireccional que nos permite iniciar cada palabra desde cualquier letra que la forme, tal y como se formarían en un partida de Scrabble.

A continuación se minimizará al máximo el autómata construido para ofrecer así un menor coste de memoria necesaria.

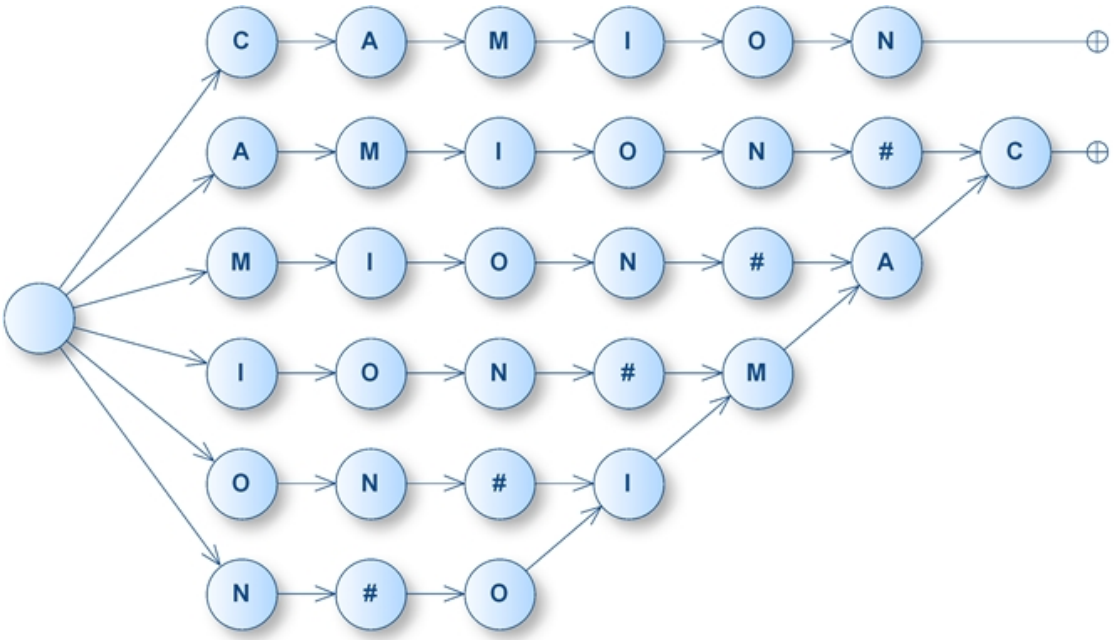


Figura 6.20 Estructura de la palabra CAMIÓN en el diccionario

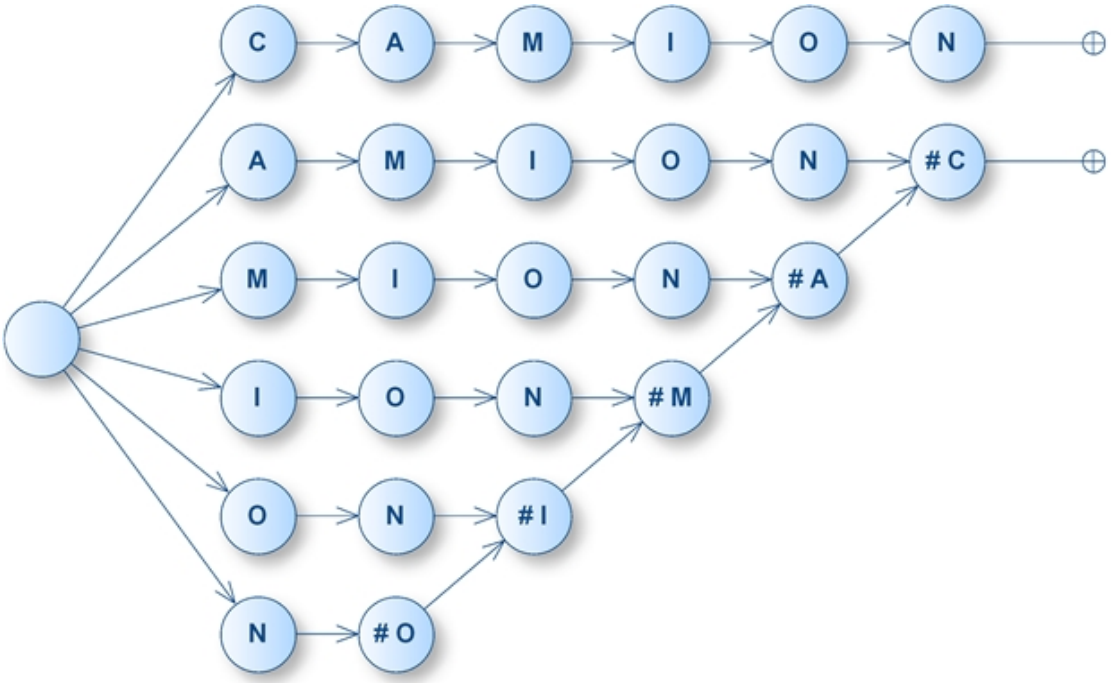


Figura 6.21 Estructura minimizada de la palabra CAMIÓN en el diccionario

En el primer paso de minimización, se han minimizado los estados que pertenecen a la formación de los prefijos, ya que todos los estados que añaden la misma letra son equivalentes entre ellos a la hora de formar el prefijo en orden inverso. Esto es debido a que una vez se decide dejar de formar el sufijo y pasar a formar el prefijo, únicamente existe una manera de formar la parte inicial de la palabra (prefijo en orden inverso).

En el segundo paso de minimización, para eliminar el estado en que se indica el cambio de sufijo a prefijo, se ha añadido una "marca" a la siguiente letra esperada. Por lo que en un mismo estado, podrán existir dos caminos con una misma letra. Esto no será ningún problema ya que no existirá ninguna ambigüedad, pues al marcar uno de ellos como camino de prefijos, estamos diferenciando entre ambas letras. Por lo que una vez que se pasa a formar el prefijo de la palabra, no es posible volver a un camino que forme parte de la construcción de los sufijos de la palabra en ningún momento.

La estructura interna de cada uno de los nodos constará de:

- **Letra.**
  Letra necesaria para poder acceder al estado del autómata.

- **esPalabra.**
  Indicará si el estado del autómata es final, por lo que se habrá formado en este punto una palabra válida

- **Prefijos.**
  Listado de estados tal que las letras de éstos forman parte del prefijo.

- **Sufijos.**
  Listado de estados tal que las letras de éstos forman parte del sufijo.

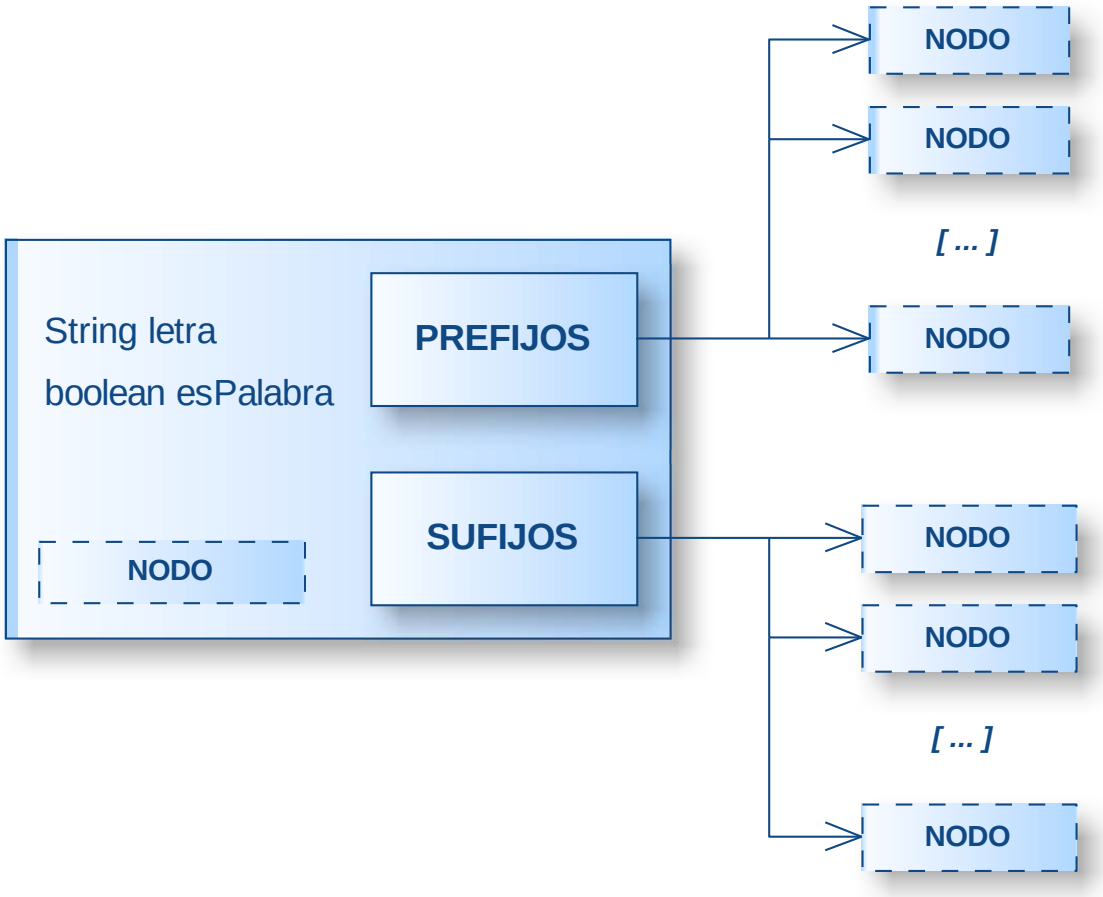


Figura 6.22 Estructura interna de un nodo del diccionario

Una vez detallada la estructura del diccionario veremos en detalle el proceso de obtención de todas las posibles jugadas a partir de aquellas casillas marcadas como candidatas del tablero.

Desde cada casilla candidata se realizarán dos búsquedas, una en dirección horizontal y otra en dirección vertical. El proceso de búsqueda de jugadas desde una casilla tal y como se puede prever, una vez visto en detalle el autómata definido, consta de la búsqueda de todos los sufijos posibles según las fichas de las que dispone el jugador en su atril, y para cada uno de ellos, la búsqueda de todos los prefijos posibles teniendo en cuenta las fichas disponibles del jugador no usadas para formar el sufijo en cuestión. Por lo tanto, vemos que tan solo trataremos aquellas ramas del nodo actual del autómata tal que correspondan con alguna de las fichas disponibles que no hayan sido utilizadas todavía para llegar hasta dicho nodo. Para cada cambio de estado en el autómata, avanzaremos a la siguiente casilla del tablero en el caso de formar sufijos, o la previa en caso de formar prefijos. Una gran ventaja, es que a través del propio proceso de búsqueda, al encontrarnos en una casilla que ya esté ocupada por una ficha, se obtiene directamente el próximo estado del autómata en caso de existir camino mediante la letra

de la ficha encontrada. De esta manera obtendremos únicamente, y sin necesidad de ningún post-procesado tal y como pasaba en el primer estudio del diccionario, el conjunto de jugadas que pueden ser realmente dispuestas en el tablero. Por lo que de nuevo se ha vuelto a resolver uno de los principales problemas que se habían encontrado en el primer estudio. Adicionalmente en cada paso, se comprobará que la letra correspondiente al nuevo estado no forme ninguna palabra inválida con aquellas fichas contiguas en dirección perpendicular a la dirección de la búsqueda, de manera que si así fuera, inmediatamente dicho nodo quedaría descartado, evitando así procesar el resto de nodos accesibles desde éste.

Una vez recorridos todos los nodos posibles, habremos ido obteniendo todas las posibles jugadas a realizar. Se puede observar como este algoritmo no necesita del cálculo de variaciones en ningún caso, teniendo en todo momento un número finito de posibles ramas a escoger, y donde tan solo se recorrerán de siete a nueve niveles de media. Nuevamente, se ha resuelto otro de los principales problemas encontrados en el primer estudio.

Si estudiamos la situación especial en que definitivamente se rechazó el primer estudio como válido, nos daremos cuenta de la potencia real de esta nueva estructura del diccionario. Ya que en el caso de disponer de un comodín entre las fichas del atril del jugador, tan sólo se realizarán las sustituciones por letras del alfabeto que realmente formen parte de alguna palabra válida. Esto es debido a que a la hora de tratar un comodín desde un nodo concreto, éste tan solo podrá ser sustituido por aquellas letras tales que permitan avanzar a un nuevo estado desde dicho nodo, evitando así la comprobación exhaustiva de la totalidad de letras del alfabeto. Esta propiedad es la que hace realmente que este tipo de diccionario sea realmente eficiente.

6.4.1.3. NODO

```
private class Nodo
{
	protected String letra;
	protected boolean esPalabra;
	protected Map sufijos;
	protected Map prefijos;

	public Nodo(String _letra)
	{
		letra = _letra;
		esPalabra = false;
		sufijos = null;
		prefijos = null;
	}
}
```

Figura 6.23. Implementación de un nodo del diccionario

Se creará un nodo principal sin ninguna letra asignada, a partir del cual empezarán todas las búsquedas que se realicen (nodo inicial).

```
public Diccionario(CtrlDiccionario pCtrlDiccionario)
{
        // Obtenemos la referencia al controlador del diccionario
        ctrlDiccionario = pCtrlDiccionario;

        // Nodo principal del diccionario
        diccionario = new Nodo("");

        soluciones = new TreeMap();
        sgfDiccionario = new SGFDiccionario(this);
}
```

Figura 6.24. Método constructor de la clase *Diccionario*

No debemos olvidar que dicha estructura debe ofrecer la posibilidad de validar si una palabra pertenece o no al diccionario. Podemos observar que dicha validación equivaldría a la formación de la palabra desde tomando como letra inicial la primera letra de palabra, y ver si existe un camino letra a letra que nos lleve a un estado final una vez hayamos llegado a la última. Por lo que podemos asegurar que la validación de palabras es posible mediante la estructura de diccionario formada.

6.4.2. ESTRATEGIA

El desarrollo de la estrategia se ha llevado a cabo teniendo en cuenta la realización de cinco niveles de dificultad. Se ha querido que la dificultad simule el nivel de conocimiento de palabras y capacidad de ver que palabras pueden formarse del jugador. De esta manera, a menor dificultad menor el número de posibles jugadas obtenidas ya sea por desconocimiento de la palabra o simplemente por que no se percatado de que podía formarse dicha palabra. Para simular dicho comportamiento se ha desarrollado en un primer paso el nivel mayor de dificultad, obteniendo en éste el conjunto completo de jugadas entre las que escoger una de ellas para llevar a cabo. Para obtener en un segundo paso el resto de niveles de dificultad, se ha procedido a eliminar de forma aleatoria un número determinado de jugadas del conjunto, simulando así que el jugador no ha sabido reconocer ciertas jugadas ya bien por desconocimiento o por no darse cuenta de su posibilidad.

Al implementar cinco niveles de dificultad, se ha decidido que el jugador obtenga una quinta parte del conjunto total de soluciones por cada nivel, de manera que para el nivel mas sencillo (el nivel uno), el jugador resuelve una quinta parte del conjunto de jugadas total; para un jugador de nivel dos, el jugador es capaz de resolver dos quintas partes del conjunto de jugadas total; y así sucesivamente hasta el nivel máximo (nivel cinco), en el que el jugador es capaz de resolver el conjunto de jugadas en su totalidad.

Una vez se ha obtenido el conjunto final de jugadas resueltas por el jugador, se deberá escoger la jugada óptima para llevar a cabo. En este punto no se ha distinguido entre niveles de dificultad ya que se entiende que independientemente del nivel del jugador, éste sabe reconocer que jugada es mejor que otra, por lo que siempre se obtendrá la jugada óptima entre las posibles.

Para decidir entre dos jugadas cual de ellas es mejor, se han seguido una serie de criterios como pueden ser:

- Puntuación conseguida
- La jugada se realiza utilizando comodín o no
- La jugada realizada forma un Scrabble (utilización de todas las fichas)
- Utilización de fichas desaconsejables de guardar en el atril
- Utilización de fichas caras

Es en la estrategia también donde tiene lugar la realización de reglas especiales como pueden ser:

- Reciclar comodines
- "Fer senyors"

6.5. PRUEBAS DEL SISTEMA

Esta etapa es muy importante para el desarrollar cualquier tipo de software que funcione de manera correcta. Las pruebas han de permitir determinar si la implementación del sistema satisface las funcionalidades, las restricciones y la especificación. Esta etapa no es trivial, ya que es necesario tener un buen conocimiento de los requisitos de software y no se puede dejar para cuando el sistema esté totalmente implementado. Es por eso que a lo largo de la implementación se han ido realizando diferentes pruebas, revisando que el sistema funcionará

tal y como se había previsto. De esta forma, el trabajo de probar el sistema se ha ido repitiendo durante la etapa de implementación.

Es importante mencionar que probar un software no asegura que un sistema sea 100% correcto; las pruebas no pueden asegurar la ausencia total de errores, sólo pueden detectar la presencia. Una prueba no es más que la ejecución del sistema con la intención de descubrir un error.

Podemos hacer básicamente cinco tipos de pruebas:

- **Pruebas unitarias.** Prueba de un único componente elemental.
- **Pruebas de integración**. Prueba de las interacciones entre componentes.
- **Pruebas del sistema**. Prueba global del sistema como una unidad de ejecución.
- **Pruebas de validación.** Para asegurar que se satisfacen los requerimientos des del punto de vista del usuario.
- **Pruebas de regresión.** Para asegurar que cambios realizados no introducen nuevos errores en componentes ya probados.

Para llevar a cabo estas pruebas existen básicamente tres enfoques posibles:

- **Caja negra.** Con este método no se tiene en cuenta la estructura interna del sistema. Es decir, el sistema se ve como una caja negra, desconociendo por completo la estructura interna de este. Esta metodología permite obtener entradas que prueben todos los requerimientos funcionales del sistema y está orientada a detectar:

    - Funciones incorrectas o ausentes.
    - Errores en las interfaces.
    - Errores en estructuras de datos o en el acceso a la base de datos.
    - Errores de rendimiento.
    - Errores de inicialización y finalización de la ejecución.

- **Caja blanca.** A diferencia del método caja negra donde la estructura interna del sistema era desconocida, el método caja blanca parte del análisis de la estructura interna del sistema, la implementación propiamente dicha.

Este método se basa en:

- La ejecución de todos los caminos independientes.
- La ejecución de todos los bucles.
- La ejecución de todas las ramas condicionales.
- Utilización de todas las estructuras.

- **Caja gris.** Combinación de las dos anteriores. A medida que se iba avanzando en la etapa de implementación, se ha ido validando el *software* desarrollado, corrigiendo los errores detectados. Para la realización de las pruebas de los componentes desarrollados se han aplicado sobretodo conceptos de la metodología de pruebas de la caja negra, encaminadas a comprobar que los componentes desarrollados correspondían a los requerimientos funcionales del sistema.

Para realizar las diversas pruebas, se añadieron al sistema dos idiomas de diccionario, castellano y catalán (obtenidos de las listas de iSpell), junto a todas sus fichas del saco. Se crearon también varios perfiles de jugador, como mínimo uno de cada nivel de dificultad y un perfil de jugador de tipo *humano*.

6.5.1. PRUEBAS UNITARIAS

Para cada componte del sistema, se han realizado pruebas sobre él, intentando descubrir posibles errores a nivel de código y a nivel de resultados. Se ha comprobado que para diferentes entradas de cada función, el resultado devuelto era el esperado. También se ha comprobado que en los casos que se debía de mostrar error, se daba. A parte también se revisado que todos los casos posibles dentro de cada componente quedaran cubiertos dentro del código.

Ejemplo de este tipo de pruebas que se han realizado son:

- Crear nuevos perfiles de jugadores con nombres válidos e inválidos.
- Añadir/eliminar palabras del diccionario existentes e inexistentes.
- Realizar movimientos inválidos de las fichas sobre el tablero.
- Etc.

6.5.2. PRUEBAS DE INTEGRACIÓN

Según se han ido implementado los diferentes componentes del sistema, se han ido integrando los diferentes módulos entre la capa de dominio y datos, y dominio y presentación, a parte de las diferentes clases. De esta manera se comprueba que los diferentes componentes se entienden y se demuestra que pueden funcionar perfectamente.

A continuación se enumeran algunas de las pruebas realizadas, así como sus efectos:

- Crear perfil de jugador ya existente ➔ no permitido, indica error.
- Añadir una palabra al diccionario ya existente ➔ no permitido, indica error.
- Eliminar una palabra del diccionario inexistente ➔ no permitido, indica error.
- Colocar ficha sobre casilla ocupada ➔ no permitido.

6.5.3. PRUEBAS DEL SISTEMA

Una vez hemos llegado a este punto, ya tenemos todas las funcionalidades, capas y clases implementadas, y por tanto, es el momento de realizar una prueba global de todo el sistema, probando cada uno de los casos de uso.

Para realizar unas pruebas más exhaustivas y efectivas del sistema, se ha dejado probar a una serie de personas cercanas como familiares, compañeros de facultad y amigos, la aplicación con tal de probar si existen errores y posibles sugerencias de mejorar la aplicación para un uso más fácil e intuitivo.

7. ESTUDIO TEMPORAL Y ECONÓMICO

7.1. Planificación inicial 184
7.2. Proceso de desarrollo del proyecto 186
7.3. Estudio temporal real 187
7.4. Estudio económico 187

7.4.1. Costes hardware 188
7.4.2. Costes software 188
7.4.3. Costes recursos humanos 189
7.4.4. Costes totales 190

7.1. PLANIFICACIÓN INICIAL

Una vez finalizados los objetivos e identificados los diferentes requerimientos del sistema, es el momento de realizar una planificación inicial proyecto. Para realizar un correcto desarrollo de un proyecto es imprescindible hacer una buena planificación de las tareas a realizar, establecer un orden cronológico y asignar una duración a cada una de ellas.

Aunque esta planificación y sus estimaciones no sean del todo ajustadas, hacerla inicialmente tiene diferentes ventajas, entre las que se pueden destacar:

- Asegurar el proceso de desarrollo, estableciendo los recursos y el tiempo necesario para su finalización.
- Adquirir plazos de tiempo fiables para cada una de las fases del proyecto.
- Permite llevar un control del proceso e identificar desviaciones y retardos en las fechas de entrega.
- Facilita la toma de decisiones.

Para planificar el proyecto, se ha dividido en varias etapas:

- Definición del proyecto
- Análisis de requerimientos
- Especificación
- Diseño
- Implementación
- Pruebas
- Memoria
- Preparación de la presentación

Estas etapas se han ido desarrollando secuencialmente, estableciendo varios puntos de control para así hacer un mejor seguimiento y control del proyecto. Tenemos que partir que el proyecto es de 37,5 créditos, que en términos de tiempo, equivale a 750 horas de trabajo. El proyecto fue inscrito y comenzado a realizarse a principios de Junio del 2006, tomando como fecha límite para la entrega a finales de Junio del 2007.

Tarea	Duración (días)	Fecha inicio	Fecha fin
Definición del proyecto	10	01 / 06 / 06	14 / 06 / 06
Análisis de requisitos	10	15 / 06 / 06	28 / 06 / 06
Especificación	10	29 / 06 / 06	12 / 07 / 06
Diseño	20	13 / 07 / 06	09 / 08 / 06
Vacaciones	20	10 / 08 / 06	06 / 09 / 06
Implementación	60	07 / 09 / 06	29 / 11 / 06
Pruebas	20	30 / 11 / 06	27 / 12 / 06
Vacaciones	20	28 / 12 / 06	24 / 01 / 07
Memoria	40	25 / 01 / 07	21 / 03 / 07
Preparación de la presentación	10	22 / 03 / 07	04 / 04 / 07

Figura 7.1. Planificación inicial

Para el cómputo total de horas dedicadas al proyecto en la planificación inicial, consideraremos una media aproximada de 4 horas por día contabilizado. Obteniendo así un total de 720 horas como planificación inicial.

Tarea	Duración (horas)
Definición del proyecto	40
Análisis de requisitos	40
Especificación	40
Diseño	80
Implementación	240
Pruebas	80
Memoria	160
Preparación de la presentación	40
Total	**720**

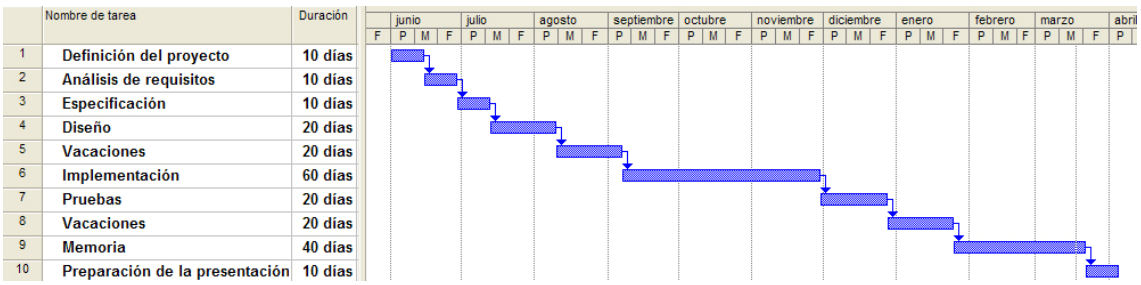
Figura 7.2. Cómputo de horas inicial

Para no encontrarnos con dificultades para llegar a la fecha de entrega del proyecto final de carrera, se realizó planificación de las tareas anteriormente descritas. Con esta planificación no se ha pretendido establecer temporalizaciones rigurosas en la elaboración del proyecto sino disponer de una ayuda a la hora de estructurar las etapas de la realización del mismo.

Desde un primer momento, se planteó planificar el desarrollo del proyecto en un periodo equivalente a dos cuadrimestres. El motivo de la extensa planificación no es otro que la

responsabilidad laboral que impide una plena dedicación diaria al desarrollo del proyecto. Entre las diferentes tareas planificadas se han contabilizando dos intervalos de un mes cada uno dedicado a las vacaciones (Agosto y Enero).

Como fecha de inicio consta el **01/06/2006**.



	Nombre de tarea	Duración
1	Definición del proyecto	10 días
2	Análisis de requisitos	10 días
3	Especificación	10 días
4	Diseño	20 días
5	Vacaciones	20 días
6	Implementación	60 días
7	Pruebas	20 días
8	Vacaciones	20 días
9	Memoria	40 días
10	Preparación de la presentación	10 días

Figura 7.3. Diagrama de Gantt de la planificación inicial

7.2. PROCESO DE DESARROLLO DEL PROYECTO

El desarrollo de un proyecto, en cualquier ámbito, se realiza mediante la secuencia del logro de diferentes etapas. En concreto veremos cuales son las etapas cuando hablamos de un proyecto informático. Las etapas que presentaremos, son muy genéricas y de esta manera presentan un esqueleto robusto para cualquier proyecto. No se puede decir cual es duración, porqué dependen mucho del proyecto en concreto.

El paradigma más sencillo y más lógico a primera vista, es el lineal secuencial, o paradigma de vida clásico. Donde las diferentes etapas van linealmente estructuradas, y la finalización de una implica el inicio de la siguiente. Este proyecto ha seguido en gran medida esta línea de desarrollo.

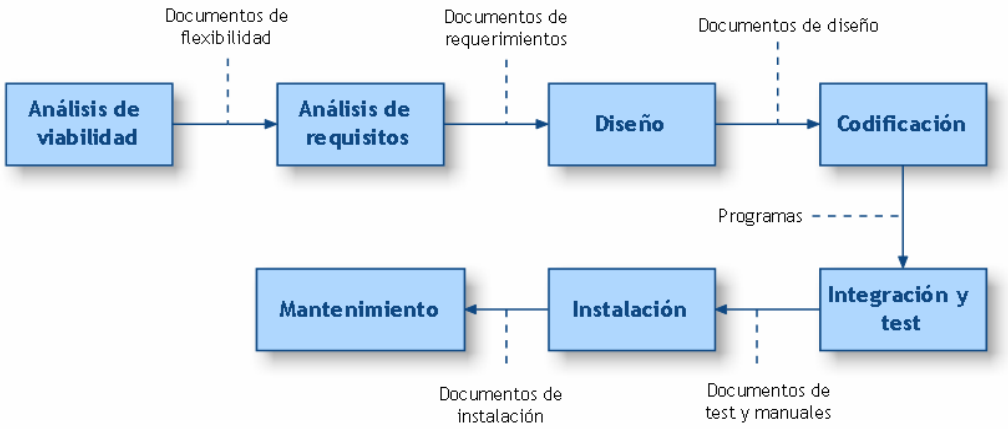


Figura 7.4 Ciclo de vida clásico de un proyecto informático

7.3. ESTUDIO TEMPORAL REAL

En este apartado veremos la duración real del proyecto, y haremos una comparación con la previsión inicial que se hizo, y así poder comparar el desvío que ha habido entre lo que se esperaba y la realidad.

Tarea	Duración estimada (días)	Duración real (días)	Desviación (días)
Definición del proyecto	10	10	0
Análisis de requisitos	10	10	0
Especificación	10	10	0
Diseño	20	20	0
Implementación	60	80	+ 20
Pruebas	20	20	0
Memoria	40	50	+ 10
Preparación de la presentación	10	10	0
TOTAL	**180 (720 horas)**	**210 (840 horas)**	**+ 30 (+ 120 horas)**

Figura 7.5. Planificación real

Como se puede apreciar hay un desvío entre la planificación inicial y el tiempo real dedicado al desarrollo del proyecto. El cálculo de la duración real ha sido medido de manera aproximada. Los motivos de la desviación notable, han sido principalmente la realización de un segundo estudio de implementación por parte del modulo de diccionario, y una corta estimación inicial del tiempo necesario para realizar el desarrollo de la memoria. Escribir esta memoria ha conllevado un mayor tiempo previsto, ya que se ha querido realizar un documento siguiendo las etapas aprendidas en las asignaturas de ingeniería del software, provocando un tiempo extra en especificar y diseñar a la hora de documentar.

7.4. ESTUDIO ECONÓMICO

Realizar un estudio del coste económico de un proyecto no es una tarea trivial ni fácil. Se ha de tener en cuenta que el proyecto ha sido realizado por una sola persona, y que en esta persona han recaído los diferentes roles que comporta la realización de un proyecto software: desde el análisis de requerimientos, hasta el diseño y la implementación de éste. También se han tenido en cuenta los gastos de software adicional y de hardware, como pueden ser el sistema operativo, el software destinado a hacer este documento o un ordenador para realizar el

proyecto. Aunque el hardware que se ha utilizado ha estado proporcionado por la facultad, y en algunos casos, se ha utilizado el ordenador personal, y en cuanto al software, no ha añadido coste ya que ha sido proporcionado por la facultad, contaremos los gastos que se hubieran producido de no haber sido esto así para poder ofrecer así una estimación de los costes más real y aproximada. Además, hay que destacar que todas las tecnologías utilizadas por la aplicación en sí, son de libre distribución.

7.4.1. COSTES HARDWARE

Para realizar el proyecto ha sido necesario un ordenador personal de gama media:

EQUIPO	CARACTERÍSTICAS	PRECIO
	▪ PC Pentium IV ▪ 2,0 Ghz ▪ 1,0 GB de RAM ▪ 60 GB HD ▪ Monitor de 17 pulgadas	**1.100 €**

Figura 7.6. Costes hardware

A este equipo informático, se le atribuye una vida media de tres años, por tanto se amortiza la parte proporcional según la duración del proyecto (12 meses).

Precio PC	Plazo amortización	% proyecto	Coste
1.100€	36 meses	(12 / 36) * 100 = 33,3	**366,3 €**

Figura 7.7. Coste hardware real

7.4.2. COSTES SOFTWARE

Por lo que respecta al software, esta es la lista de aplicaciones y herramientas utilizadas durante el desarrollo del proyecto y de sus precios.

PRODUCTO	PRECIO
	130 €

	610 €
	850 €
	0 €
JOGL	**0 €**
TOTAL:	**1.590 €**

Figura 7.8. Costes software

7.4.3. COSTES RECURSOS HUMANOS

Aunque el proyecto ha estado realizado íntegramente por mi, para ver los costes humanos del proyecto, se ha dividido el trabajo en cuatro perfiles diferentes, ya que según la tarea desempeñada tiene un mayor o menor coste por hora.

A continuación se detalla brevemente que hace cada perfil:

- **Jefe de proyecto.** Encargado de que el proyecto cumpla los requisitos solicitados, de realizar reuniones periódicas (puntos de control) y responsable de hacer que se cumplan los plazos.
- **Analista.** Su función es hacer el análisis de requerimientos, especificación y diseño del sistema.

- **Programador.** Realiza la implementación del sistema.

- **Diseñador Gráfico.** Encargado de realizar la parte del diseño que corresponde a la interfaz gráfica.

PERFIL	HORAS	PRECIO (€/hora)	SUBTOTAL
Jefe de proyecto	80	60 €	4.800 €
Analista	320	45 €	14.400 €
Programador	400	20 €	8.000 €
Diseñador gráfico	40	35 €	1.400 €
TOTAL	**840**		**28.600 €**

Figura 7.9. Costes recursos humanos

La documentación es una tarea compartida por la mayoría de los participantes en el proyecto. Aún así, es el analista quien realiza la mayor parte de la documentación ya que hace falta plasmar los conceptos técnicos y de diseño que se han realizado, ayudado por el resto de perfiles.

7.4.4. COSTES TOTALES

TIPO COSTE	COSTE
Hardware	**366 €**
Software	**1.590 €**
Recursos humanos	**28.600 €**
TOTAL	**30.556 €**

Figura 7.10. Costes totales

Tan solo recordar que en la realidad el total es algo inferior a los 30.556 €, debido a que los costes por hardware y software han sido nulos debido a que ya se disponía de dichos recursos antes de la inicialización del proyecto. Aún así, se ha preferido tenerlos en cuenta en la contabilización total para poder ofrecer un coste más aproximado a la realidad.

8. CONCLUSIONES

8.1. Objetivos alcanzados 194
8.2. Aportación personal 194
8.3. Desarrollo futuro 195

8.1. OBJETIVOS ALCANZADOS

El objetivo principal de este proyecto, *Un joc d'Scrabble*, es permitir jugar una partida al juego conocido como *Scrabble* entre 2 y 4 jugadores. Estos jugadores pueden ser de tipo *Humano* o *Robots*. Los jugadores tipo *Robot* podrán configurarse con diferentes tipos de dificultad. Adicionalmente debe ofrecer la posibilidad de jugar un único jugador a modo de entrenamiento para la capacidad de resolver jugadas. Podemos decir que el objetivo principal ha sido alcanzado, así como los diferentes objetivos listados a continuación:

- Gestión de partidas
- Gestión de jugadores
- Gestión de diccionarios
- Gestión de idiomas
- Gestión de reglas (variaciones de juego)
- Gestión de estadísticas (rankings)

8.2. APORTACIÓN PERSONAL

Personalmente este proyecto me ha ayudado a conocer tecnologías que antes prácticamente desconocía, y ampliar mis conocimientos en las ya conocidas. Y además, he podido aplicar en un mismo proyecto todos los conocimientos adquiridos en las diferentes asignaturas de la carrera.

Me he dado cuenta que el desarrollo de un proyecto es algo muy complejo que requiere un gran esfuerzo de dedicación y sobretodo un estricto control de "*timings*" para poder cumplir los objetivos que de manera personal o a través del tutor se van marcando para poder alcanzar los objetivos que se plantean. Aún así, la cantidad de contratiempos que pueden aparecer es muy grande y esto demuestra el hecho que aún con una buena organización no siempre es fácil desarrollar un proyecto.

Como antes he comentado en el primer párrafo, gracias al proyecto he podido conocer y ampliar mis conocimientos en diferentes aspectos como pueden ser:

- Diseño UML
- Análisis y programación en *Java*
- Programación con librerías gráficas *JOGL*

- Optimización de algoritmos en la resolución de un problema
- Desarrollo de una inteligencia artificial
- Planificación y estudio económico de un proyecto
- Documentar un proyecto

También he podido aplicar los conocimientos obtenidos en diferentes asignaturas como *ES1*, *ES2* y *PESBD* en lo referente a ingeniería del software. Asignaturas como *IP*, *PM*, *EDA*, *PP*, *PCD* me han ayudado en lo que refiere a programación. *IA* me ha permitido aplicar bastantes conceptos en el desarrollo de la inteligencia artificial de los jugadores de tipo *robot*.

En definitiva, considero que la realización del proyecto final de carrera es una tarea fundamental por parte de cualquier ingeniero en informática, sea técnico o superior.

8.3. DESARROLLO FUTURO

Aunque el proyecto ha adquirido los objetivos que se marcaron en un principio, éste puede mejorarse en muchos aspectos, a continuación se presenta una lista con las posibles mejoras a hacer:

- Como una de las principales posibles mejoras, se encuentra la implementación de un módulo adicional que permitiera jugar partidas online entre jugadores de tipo *humano* cada uno de ellos desde su propio PC.

- Adicionalmente a la mejora de partidas online, podrían centralizarse los datos como los diccionarios de manera que todos las partidas online utilicen los mismos diccionarios. De esta manera se enriquecerán mucho más rápidamente los diccionarios con nuevas palabras.

- Si se centralizasen también los datos de jugadores, se podría llegar a crear una comunidad web, desde donde organizar partidas, y poder crear torneos y ligas.

9. BIBLIOGRAFÍA

9.1. Bibliografía 199

9.1.1. Libros 199
9.1.2. Referencias 199
9.1.3. Proyectos 200

9.1. BIBLIOGRAFÍA

A continuación se detalla toda la bibliografía consultada para la creación de este documento, así como para el diseño e implementación de todo el proyecto. Ésta se encuentra detallada en tres bloques definidos como: *Libros*, *Referencias* y *Proyectos*.

9.1.1. LIBROS

- **Dolors Costal, M. Ribera Sancho, Ernest Teniente**
  Enginyeria del software especificació: especificació de sistemes orientats a objectes amb la notació UML
  Edicions UPC, 2000 ISBN: 84-8301-379-7

- **Cristina Gómez, Enric Mayol, Antoni Olivé, Ernest Teniente**
  Enginyeria del software Disseny I: disseny de sistemes orientats a objectes amb notació UML
  Edicions UPC, 2000 ISBN: 84-8301-540-4

9.1.2. REFERENCIAS

- **Scrabble**
  - Mattel Scrabble
    http://www.mattelscrabble.com
  - Wikipedia
    http://es.wikipedia.org/wiki/Scrabble
  - Terminología
    http://www.pancarta.net/scrabble/terminologia.htm
  - Amigos del Scrabble
    http://usuarios.sion.com/yniss/index2.htm

- **Análisis de requerimientos**
  - IEEE *Std.* 830-1993
    http://standards.ieee.org/reading/ieee/std_public/description/se/830-1993_desc.html

  - Estudio del estándar IEEE 830-1993
    http://www.infor.uva.es/~descuder/proyectos/ipo/requidoc.htm

- **Tecnologías**

  - JAVA
    http://java.sun.com/j2se/1.5.0/docs/api/
  - JOGL
    http://jogl.dev.java.net
  - OpenGL
    http://es.wikipedia.org/wiki/OpenGL
  - Algoritmos
    http://www.gtoal.com/wordgames/scrabble.html
    http://www.gtoal.com/wordgames/jacobson+appel/aj.pdf
  - Eclipse
    http://www.eclipse.org

- **Diccionarios (listados de palabras)**

  - iSpell
    http://lasr.cs.ucla.edu/geoff/ispell-dictionaries.html

9.1.3. PROYECTOS

- **Aplicación para el control de producción**
  Autor: *Rubén Rodríguez Requena*

- **Diseño de una agenda digital con servicio de mensajería**
  Autor: *José Luís Zayas Pascual*

- **Disseny i implementació aplicació d'anàlisi estadístic de series financeres**
  Autor: *Javier Meler Playán*

- **Integración de GRID superescalar con brokers externos**
  Autor: *Daniel Rubio Catalán*

- **Soporte didáctico de Bridge 2**
  Autor: *Raúl Molina Ordóñez*

i. ANEXO 1. MANUAL DE INSTALACIÓN

i.1. Introducción 204
i.2. JRE 204
i.3. Librerías JOGL (Windows) 205
i.4. Librerías JOGL (Linux) 206

i.1. INTRODUCCIÓN

Este manual ha sido construido con el propósito de explicar y facilitar la instalación del proyecto *Un joc d'Scrabble* y de todos los requisitos necesarios para su funcionamiento.

A continuación se muestra una lista de los diferentes componentes a instalar y más adelante se explicará como instalar uno por uno:

- JRE (Máquina virtual de *Java*)
- Librerías *JOGL* (*Windows*)
- Librerías *JOGL* (*Linux*)

i.2. JRE

La aplicación está implementada utilizando la tecnología *Java* por tanto necesitaremos la *JVM* (Máquina Virtual de *Java*) para ejecutar los *bytecodes* de nuestro programa. Aquí mostraremos como instalar el paquete *Java 2 Platform, Standard Edition* que contiene *Java 2 SDK* (*kit* para desarrollar aplicaciones *Java*) y *Java 2 Runtime Environment* (Máquina Virtual de *Java*).

Podemos encontrar el J2SE en http://www.sun.com en el apartado de *Downloads*, o también en el CD que viene adjunto con la memoria del proyecto (carpeta jre1.5.0_11). Si optamos por la opción de descarga, debemos seleccionar la descarga de **J2SE 5.0**, dado que serán necesarias las API's de esta versión o posteriores para el funcionamiento de la aplicación (no en todas las versiones anteriores están incluidas).

Figura i.1. Descarga de la J2SE 5.0

Una vez hemos dispongamos del archivo de instalación sólo tenemos que ejecutarlo y el archivo de instalación nos irá guiando.

i.3. LIBRERIAS JOGL (WINDOWS)

Será necesario incorporar en la instalación de la *JVM* (Java Virtual Machine) las librerías que contienen las API para las llamadas a *OpenGL*. Dichas librerías podemos encontrarlas en el CD entregado junto a este documento, concretamente en la carpeta JOGL.

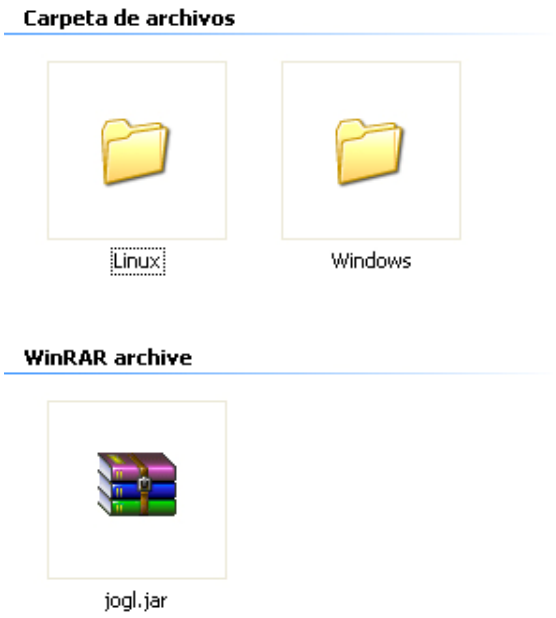


Figura i.2. Carpeta JOGL del CD entregado junto a la memoria

El primer paso será copiar el fichero **jogl.jar** al directorio *lib\ext* de la máquina virtual de Java utilizada. Por ejemplo, si la instalación de Java se encuentra en el directorio *C:\Archivos de programa\Java\jre1.5.0_11*, la ruta completa de la carpeta destino será ***C:\Archivos de programa\Java\jre1.5.0_11\lib\ext***.

El segundo paso será copiar las bibliotecas **dll** correspondientes a la carpeta *Windows* al directorio *bin* de la máquina virtual de Java utilizada. Por ejemplo, al igual que en el caso anterior, si la instalación de Java se encuentra en el directorio *C:\Archivos de programa\Java\jre1.5.0_11*, la ruta completa de la carpeta destino será ***C:\Archivos de programa\Java\jre1.5.0_11\bin***.

Figura i.3. Bibliotecas dll correspondientes a la carpeta Windows

i.4. LIBRERIAS JOGL (LINUX)

Aunque la aplicación no haya sido desarrollada ni ejecutada bajo un sistema operativo *Linux*, se explicará brevemente como instalar las librerías *JOGL* en un entorno de este tipo. Al igual que las librerías para *Windows*, también se adjuntan en el CD entregado las librerías *JOGL* para distribuciones *Linux*.

1.- Localizar la instalación de *Java SDK* o *JRE* en la máquina.

2.- Copiar los ficheros **libjogl.so** y **libjogl_cg.so** de la carpeta linux del directorio JOGL del CD entregado en la carpeta */lib/i386/* del directorio de la instalación de *Java SDK* o *JRE*. Por ejemplo:

[jre_location]/lib/i386/libjogl.so

[jdk_location]/jre/lib/i386/libjogl.so

3.- Copiar el fichero **jogl.jar** en la carpeta */lib/ext/* del directorio de la instalación de *Java SDK* o *JRE*. Por ejemplo:

[jre_location]/lib/ext/jogl.jar

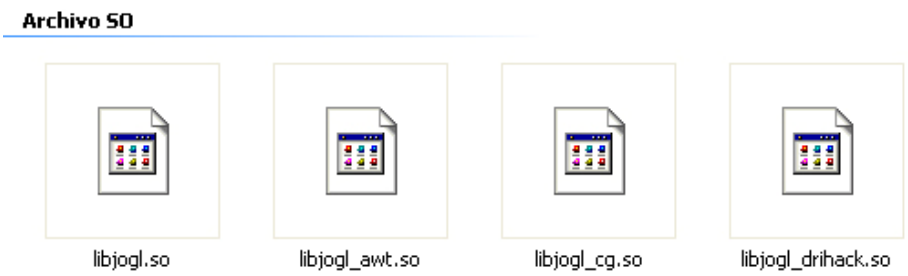


Figura i.4. Ficheros .so correspondientes a la carpeta Linux

ii. ANEXO 2. INSTRUCCIONES DE JUEGO

ii.1. Scrabble 210
ii.2. Contenido 210
ii.3. Mecánica del juego 210
ii.4. Reglas de juego 210
ii.5. Explicación de reglas problemáticas 214

ii.1. SCRABBLE

Scrabble es un juego de palabras cruzadas para dos, tres o cuatro jugadores. Consiste en formar sobre el tablero palabras que se crucen entre sí, igual que en los crucigramas, haciendo uso de fichas marcadas con una letra y un número (su valor). Los jugadores deben tratar de conseguir la mayor puntuación posible, utilizando sus fichas con habilidad, es decir, colocándolas de manera que obtengan el mayor partido tanto de los valores de las fichas como de las casillas con premio del tablero. La puntuación total de una partida puede ser desde cuatrocientos a ochocientos puntos, o incluso más, según la habilidad de los jugadores.

ii.2. CONTENIDO

- El juego incluye noventa y ocho fichas con letras y dos fichas en blanco comodines.
- Cada ficha con letra tiene un valor (indicado en la esquina inferior derecha de la misma).
- Las dos fichas en blanco, o comodines, no tiene valor pero pueden utilizarse en sustitución del cualquier letra. El jugador que coloca un comodín debe indicar al resto de jugadores la letra que el comodín representa, lo que no puede modificarse en toda la partida.

ii.3. MECÁNICA DEL JUEGO

- Para decidir qué jugador empieza a jugar, todos cogen una ficha y el que tenga la letra más próxima al principio del alfabeto será el primero en jugar. El comodín se considera anterior a la letra "A". Se vuelven a dejar estas fichas dentro de la bolsa, con las demás, y se mezclan todas.
- A continuación, cada jugador toma siete fichas y las coloca en su atril.

ii.4. REGLAS DE JUEGO

- **Cambio de fichas**

Los jugadores pueden utilizar su turno para cambiar algunas o todas las fichas de su atril. En primer lugar, deberán dejar las fichas que deseen cambiar boca abajo, sobre la mesa; seguidamente, tomarán tantas fichas nuevas como hayan dejado; y finalmente colocarán dentro

de la bolsa las que han dejado. El jugador que ha cambiado fichas debe esperar al siguiente turno para jugar.

- **Pasar (perder un turno)**

En lugar de colocar fichas en el tablero o cambiarlas por nuevas fichas de la bolsa, los jugadores pueden "pasar", es decir, anunciar que pierden su turno sin realizar ninguna jugada. Está permitido pasar incluso si el jugador puede formar alguna palabra o palabras con las fichas que posee. Sin embargo, cuando todos los jugadores pasan dos veces consecutivas, se da por finalizada la partida.

- **Colocación de la primera palabra en el tablero**

El primer jugador combina dos o más de sus letras para formar una palabra, colocándolas en el tablero en línea horizontal o vertical; está obligado a poner una de las letras que forman la palabra en la casilla central (señalada con una estrella). No se permite formar palabras en diagonal. Sólo se pueden formar palabras que puedan leerse de izquierda a derecha o de arriba a abajo, estando prohibidas las que deban leerse de derecha a izquierda y de abajo a arriba.

- **Palabras permitidas**

Se permite formar cualquier palabra que se encuentre en el diccionario. Existe un diccionario oficial para jugar a SCRABBLE, el "VOCABULARIO OFICIAL DEL SCRABBLE".

- **Casillas con premio**

El tablero consta de una cuadrícula de quince x quince casillas, de las cuales varias son casillas con premio. Al colocar una o más fichas en una casilla con premio, el jugador duplica o triplica la puntuación obtenida, según el tipo de casilla con premio.

Figura ii.1: Casilla normal

- **Casillas con premio para letras**

Al colocar una ficha en una casilla de color azul celeste, se dobla el valor de dicha ficha,

y al colocarla en una casilla de color azul marino se triplica su valor.

Figura ii.2: Casillas de bonus de tipo letra

- **Casillas con premio para palabras**

Al colocar una de las letras de una palabra en una casilla de color rosa, se dobla el valor de la palabra, y al colocarla en una casilla roja, se triplica el valor de la palabra.

Figura ii.3: Casillas de bonus de tipo palabra

Si se ha colocado una palabra sobre una o más casillas con premio para letras y una o más casillas con premio para palabras, al hacer el recuento de puntos de la palabra, se deben sumar primero los premios para letras y luego doblar o triplicar el valor de la palabra, según sea el caso. Los premio para letras y palabras se aplican solamente la primera vez que se ocupa la casilla con premio. Posteriormente, las fichas que ocupan estas casillas cuentan simplemente como su valor. Al colocar un comodín en una casilla rosa o roja, la suma de los valores de las letras que forma la palabra se dobla o triplica, según sea el caso, aunque el comodín por sí solo vale cero. Si se coloca en una casilla azul (doble o triple tanto de letras), como el valor del comodín es cero, al doblarlo o triplicarlo el resultado sigue siendo cero.

- **Finalizando turno**

Al final de su turno, el jugador toma tantas fichas de la bolsa como letras haya utilizado, con lo que durante toda la partida dispondrá siempre de siete fichas en el atril.

- **Hacer un "Scrabble"**

Cuando un jugador coloca las siete fichas de su atril en un solo turno, hace un "Scrabble", por el que obtiene cincuenta puntos extras, a sumar a la puntuación que obtenga en ese turno después de doblar o triplicar dicha puntuación si procede.

- **Continuación de la partida**

El juego continúa hacia la derecha del primer jugador. El siguiente jugador puede añadir una o más fichas a las que ya hay sobre el tablero para formar nuevas palabras, cambiar una o más de sus fichas o anunciar que pasa, perdiendo su turno. La partida continúa así sucesivamente. Todas las fichas jugadas en un mismo turno deben colocarse en una sola línea horizontal o vertical, nunca en diagonal. Si además están en contacto con fichas de hileras contiguas, también deberán formar palabras completas con las mismas, igual que en los crucigramas. El jugador obtiene puntos por todas las nuevas palabras que haya formado durante su turno.

- **Hay cinco maneras de formar palabras sobre el tablero:**

1. Añadiendo una o más fichas a una palabra que ya esté sobre el tablero. Se pueden añadir al principio de la palabra, al final, o simultáneamente al principio y al final.

2. Colocando una palabra que cruce a otra palabra del tablero. La nueva palabra debe utilizar una de las letras de la palabra que ya estaba sobre el tablero.

3. Colocando una palabra completa paralelamente a una palabra ya formada sobre el tablero, de modo que las fichas contiguas también formen palabras completas.

4. La nueva palabra también puede añadir una letra a una palabra ya formada sobre el tablero.

5. La última manera posible de formar palabras consiste en "hacer un puente" entre dos letras ya colocadas sobre el tablero (esto sólo puede hacerse a partir del cuarto movimiento aproximadamente, porque para realizar esta jugada debe haber varias palabras sobre el tablero).

A veces, una palabra puede ocupar dos casillas de Doble Tanto de Palabra o de Triple Tanto de Palabra. En estos casos, primero se duplica el valor de la palabra y luego se reduplica (es decir, se multiplica por 4 la suma de puntos de sus letras; o se triplica y se vuelve a triplicar, es decir, se multiplica por 9 la suma de puntos de sus letras).

- **Final de la partida**

La partida finaliza cuando:

- Uno de los jugadores utiliza todas las fichas de su atril y en la bolsa no quedan más fichas para coger.
- Cuando ningún jugador puede colocar fichas en el tablero para formar una nueva palabra.
- O cuando todos los jugadores pasan dos veces consecutivas.

Al final del juego, cada jugador debe restar de su puntuación total la suma de los valores de las fichas que le quedan en el atril. Si un jugador ha utilizado todas sus fichas, se suma a su puntuación el valor total de las fichas que han sobrado al resto de jugadores. Por ejemplo, si al acabar la partida un jugador tiene en su atril una X y una A, debe restar nueve puntos a su puntuación total. El jugador que haya jugado todas las fichas de su atril sumará a su puntuación total estos nueve puntos.

¡Recuerda que una partida se puede ganar o perder por la última letra que quede en la bolsa!

ii.5. EXPLICACIÓN DE REGLAS PROBLEMÁTICAS

- Si al colocar una palabra, una o más de sus fichas están en contacto con fichas de hileras contiguas, también deberán formar palabras completas con las mismas, igual que en los crucigramas.
- Se puede jugar la misma palabra más de una vez en la misma partida.
- Los plurales están permitidos, así como los femeninos y todas las formas verbales.
- Se puede alargar una palabra por ambos extremos en un mismo turno.
- Todas las fichas jugadas en un mismo turno deben colocarse en una sola línea horizontal o vertical, nunca en diagonal.
- No está permitido añadir letras a varias palabras distintas o formar varias palabras en diferentes lugares del tablero en un mismo turno.

- Los premios para letras y palabras se aplican solamente la primera vez que se ocupa la casilla con premio. Posteriormente, las fichas que ocupan estas casillas cuentan simplemente como su valor.

- Cuando en un mismo tuno se forman varias palabras, se contabiliza la puntuación de cada una de ellas. Las letras comunes se incluyen en la suma de puntos de cada una de las palabras (incluyendo los puntos adicionales por casilla con premio si alguna de las fichas de la nueva palabra ha sido colocada en una de estas casillas).

- Si se forma una palabra que ocupa dos casillas con premio para palabras, si son rosas, primero se dobla el valor de la palabra y luego se redobla (es decir, se multiplica por cuatro la suma de puntos de sus letras); y si son rojas, primero se triplica y luego se vuelve a triplicar (es decir, se multiplica por nueve la suma de puntos de sus letras).

- Al colocar un comodín en una casilla de Doble o Triple Tanto de Palabra, la suma de los valores de las letras que forman la palabra se dobla o triplica, según sea el caso, aunque el comodín por sí solo vale cero. Si se coloca el comodín en una casilla de Doble o Triple tanto de Letra, el valor del comodín es cero.

- Cuando uno de los jugadores utiliza todas las fichas de su atril y en la bolsa no quedan más fichas para coger, la partida se da por finalizada y no se pueden realizar más jugadas. A veces, ningún jugador consigue utilizar todas sus fichas cuando se agotan las fichas de la bolsa. En este caso, la partida continúa normalmente hasta que ningún jugador puede colocar fichas en el tablero para formar una nueva palabra. Si uno de los jugadores no puede formar ninguna palabra, debe pasar. Cuando todos los jugadores pasan dos vences consecutivas, se termina la partida.

iii. ANEXO 3. MANUAL DE USUARIO

iii.1. Introducción 219
iii.2. Jugadores 220
iii.3. Diccionario 223
iii.4. Reglas 224
iii.5. Partida 225

iii.5.1. Turno jugador robot 226
iii.5.2. Turno jugador humano 227

iii.6. Ranking 233
iii.7. Idioma 234
iii.8. Ayuda 235
iii.9. Acerca de... 236

iii.1. INTRODUCCIÓN

La creación de este manual tiene la finalidad de guiar a los nuevos usuarios de *Un joc d'Scrabble*, y que de ésta manera puedan familiarizarse con la interfaz de la aplicación más rápidamente.

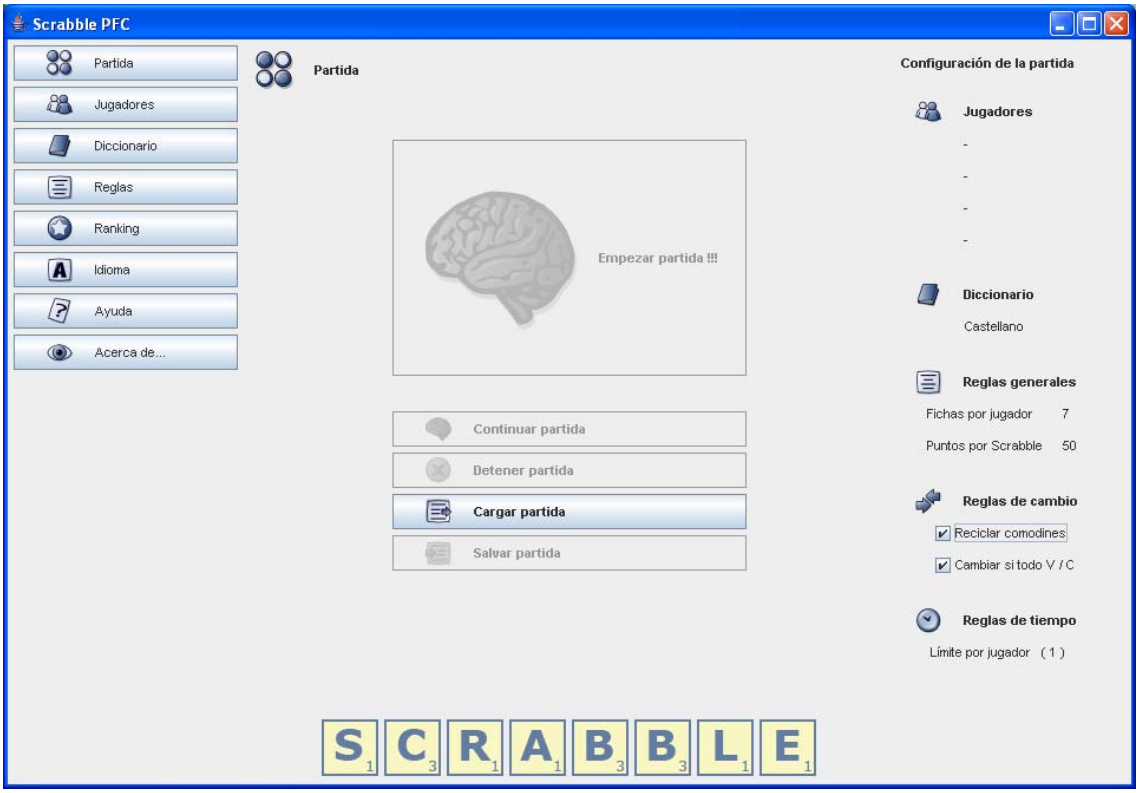


Figura iii.1. Pantalla inicial (pantalla de partida)

Como se puede observar la pantalla esta dividida en tres secciones:

- **Menú.** Desde donde se podrá acceder a todas las pantallas de la aplicación.

- **Sección central.** Sección en la que se mostrará el contenido de las pantallas a las que se acceda a través del menú.

- **Información de partida.** Desde donde se podrá consultar en todo momento la configuración actual de la partida: jugadores, idioma de diccionario, reglas generales, reglas de cambio de fichas y reglas de tiempo.

Esta división variará como veremos un poco más adelante en el momento en que sea iniciada la partida configurada. A continuación se detallarán las pantallas a través de las que se podrán configuran las partidas:

- Jugadores
- Diccionario
- Reglas

Seguidamente se detallará la pantalla de partida como la de juego mediante la que el jugador podrá llevar a cabo todas las acciones deseadas en su turno, para finalmente detallar el resto de pantallas:

- Ranking
- Idioma de localización
- Ayuda
- Acerca de...

Recién ejecutada la aplicación, tal como se ha comentado, se mostrará la pantalla de partida como pantalla de inicio. Se puede ver como no es posible empezar una partida en este momento hasta que no se haya configurado como mínimo los jugadores que formarán parte en ella (mínimo un jugador).

iii.2. JUGADORES

En la pantalla de gestión de jugadores se pueden ver dos listas, jugadores disponibles y jugadores seleccionados. Desde esta pantalla podremos indicar que jugadores serán los que formarán parte en la partida. Aquellos jugadores seleccionados aparecerán en la sección de información para que una vez el usuario cambie de pantalla, pueda seguir consultando qué jugador formarán parte sin tener que volver a esta pantalla. Como máximo podrán seleccionarse hasta cuatro jugadores.

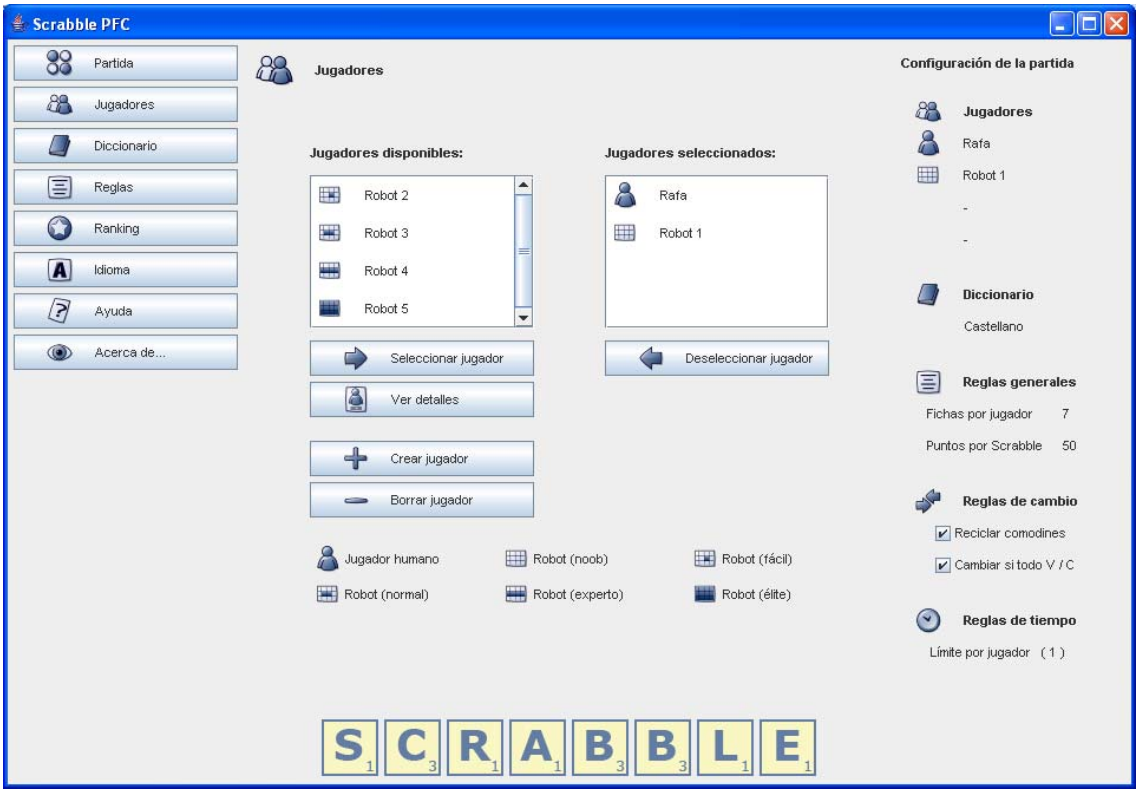


Figura iii.2. Pantalla de gestión de jugadores

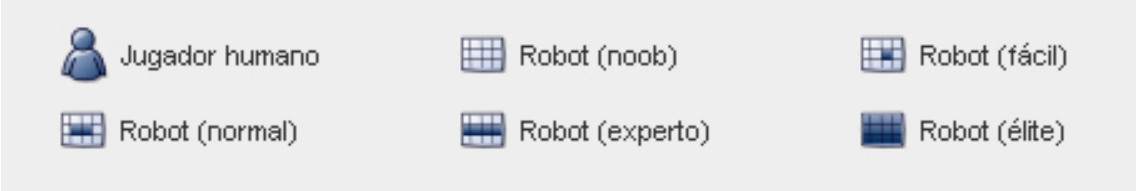


Figura iii.3. Descripción de iconos de tipo de jugador

Se podrán crear nuevos perfiles de jugador a través de la pantalla de gestión de jugadores. En la pantalla de creación de un nuevo perfil de jugador se deberá indicar el nombre del nuevo jugador así como el tipo de jugador que será. Esta información no podrá ser modificada posteriormente.

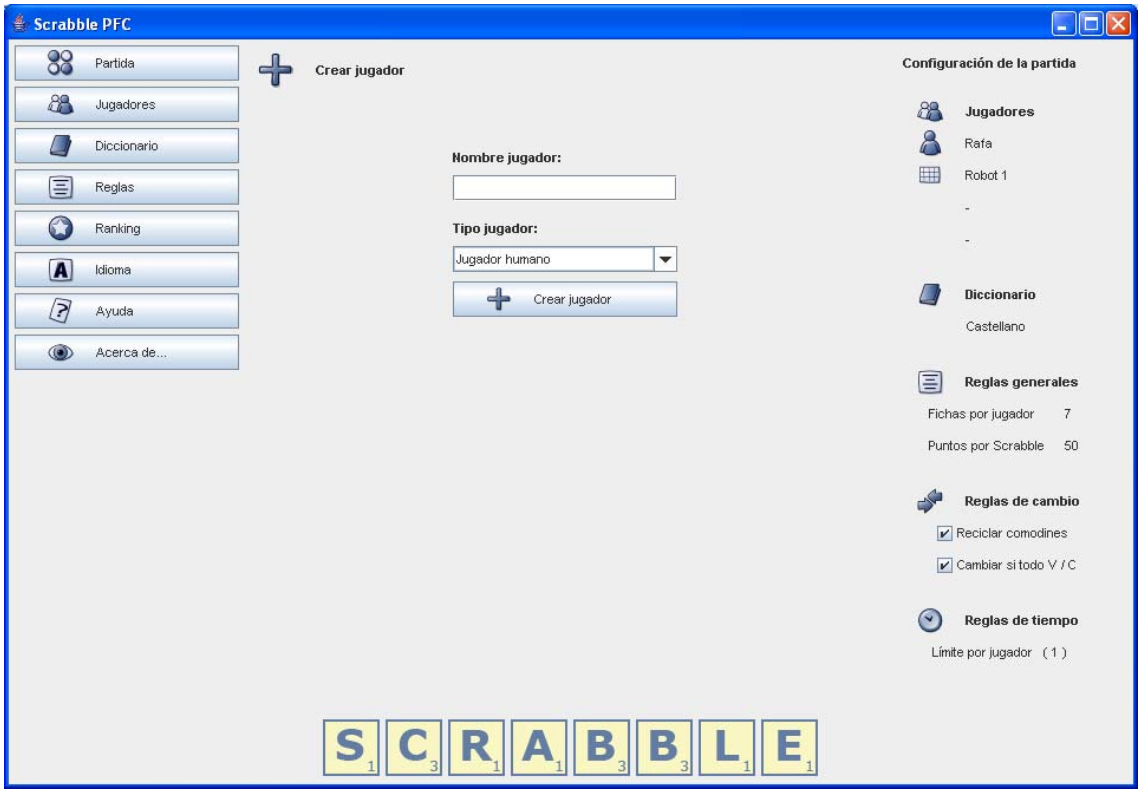


Figura iii.4. Pantalla de creación de un nuevo perfil de jugador

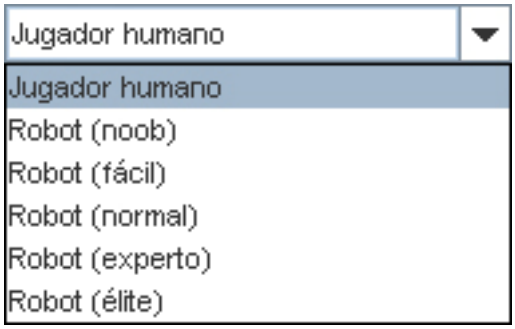


Figura iii.5. Tipos de jugador disponibles.

También es posible eliminar el perfil de jugador deseado. Para ello, se deberá seleccionar en la lista de jugadores disponibles el nombre del jugador que se desea eliminar e indicar a la aplicación mediante el botón correspondiente su eliminación.

De la misma manera que se puede eliminar un perfil de jugador, se podrá consultar los detalles de un jugador concreto, pudiendo ver así todas sus puntuaciones y estadísticas.

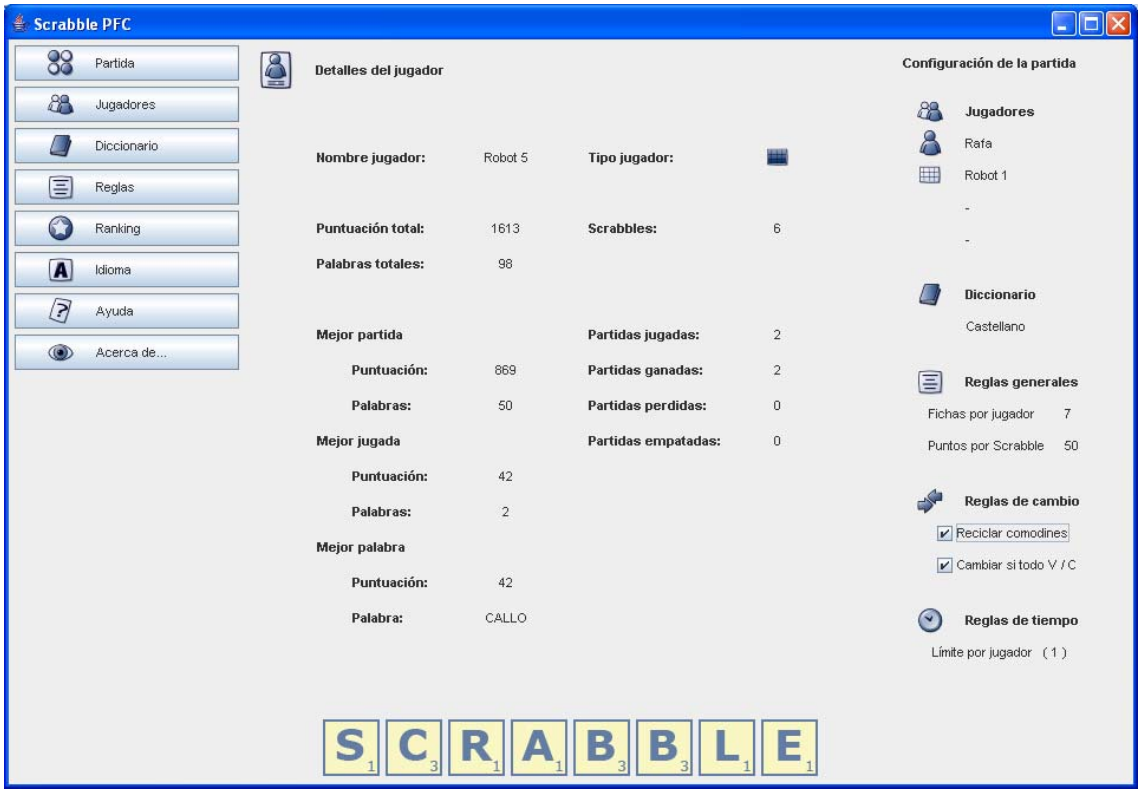


Figura iii.6. Pantalla de detalles de un perfil de jugador

iii.3. DICCIONARIO

Desde la pantalla de gestión de diccionarios, el usuario podrá seleccionar el idioma del diccionario de juego que desea, así como definirlo por omisión para futuras ejecuciones de la aplicación. El diccionario seleccionado podrá consultarse en la sección de información de manera que no será necesario volver a esta pantalla para ver qué diccionario está seleccionado.

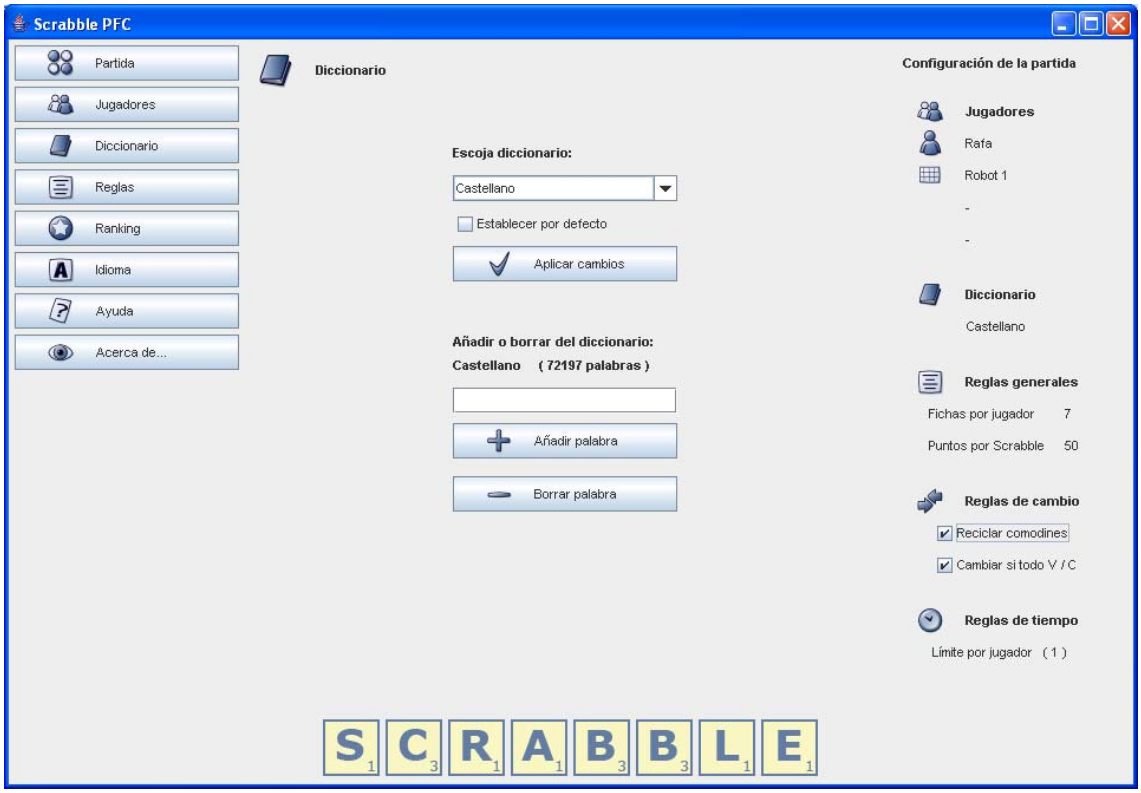


Figura iii.7. Pantalla de gestión diccionarios

Adicionalmente, se podrá ver el número de palabras válidas contenidas en el diccionario seleccionado, así como añadir nuevas palabras o eliminar aquellas que sean incorrectas. Esta funcionalidad seguirá estando disponible durante el desarrollo de una partida teniendo validez las nuevas palabras añadidas en el mismo momento de su incorporación.

iii.4. REGLAS

Desde la pantalla de reglas el usuario podrá configurar las diversas reglas de tipo: generales, de cambio, y de tiempo. Todos los cambios realizados podrán consultarse desde la sección de información de manera que no será necesario volver a está pantalla para conocer la configuración de reglas. El usuario podrá establecer la configuración de reglas por omisión para futuras ejecuciones de la aplicación.

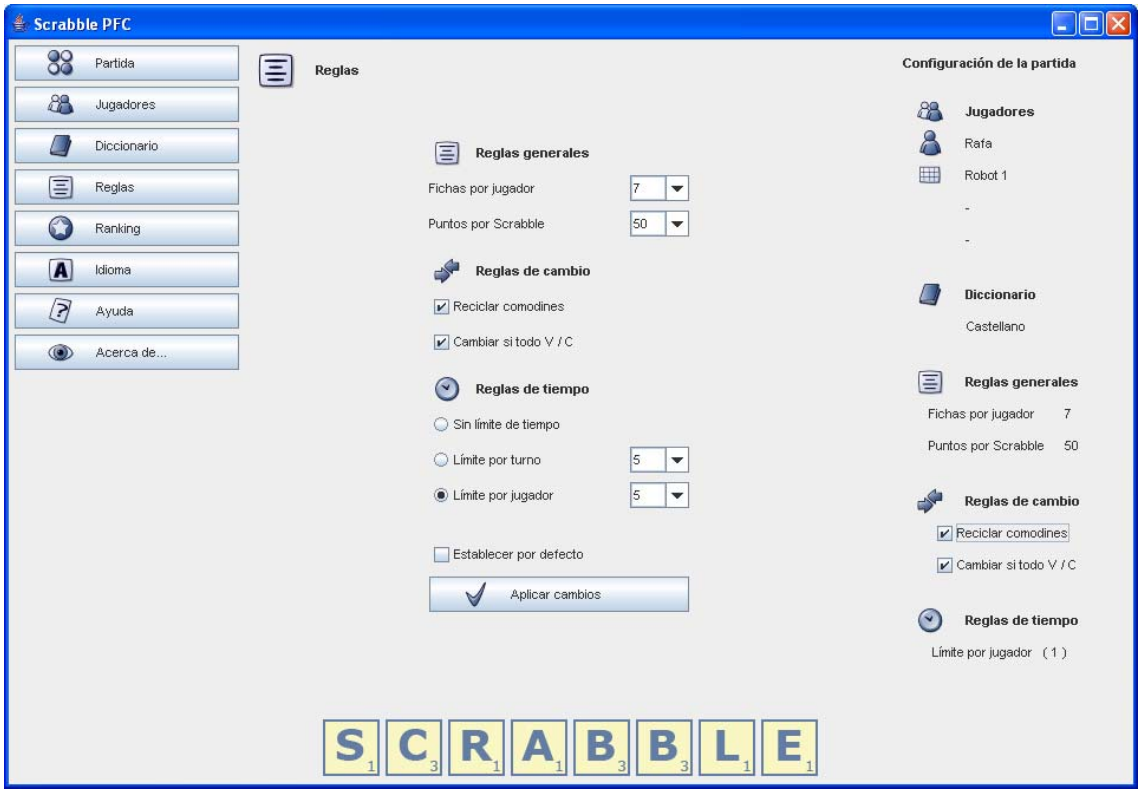


Figura iii.8. Pantalla de gestión reglas

iii.5. PARTIDA

Como se ha visto en la introducción de este manual, hasta que no hay seleccionado como mínimo un jugador para la partida no será posible iniciarla.

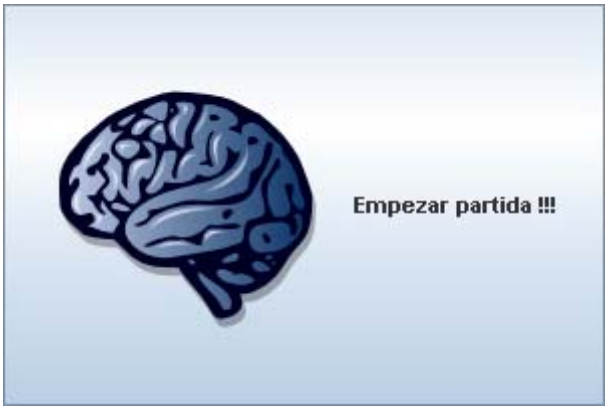


Figura iii.9. Botón empezar partida activado

Una vez la partida ha sido iniciada, se puede observar como han variado las secciones en que estaba dividida la aplicación:

Figura iii.10. Pantalla de una partida iniciada

- **Menú.** Se siguen mostrando los diferentes menús, y adicionalmente se muestra la información de la partida actual: jugadores (indicando el turno de jugador y las puntuaciones correspondientes de cada uno de ellos), el tiempo restante del jugador (si así estuviera configurado en las reglas), el idioma del diccionario de juego, y el número de fichas restantes en el saco. También se podrá crear el log del transcurso de la partida hasta el turno actual si así se desea.

- **Sección central.** En la sección central se encontrará el escenario 3D en el que se mostrará el tablero con el que el usuario podrá interaccionar para realizar las jugadas, así como las fichas en posesión del jugador actual (atril).

- **Acciones e información de jugador.** Se mostrarán todas las acciones disponibles a realizar por parte del jugador del turno actual, así como toda la información de éste referente a la partida en juego.

iii.5.1. TURNO JUGADOR ROBOT

Los jugadores de tipo de robot realizan la resolución de su turno automáticamente. El usuario podrá visualizar fácilmente que jugada es la que se ha llevado a cabo debido a que la

palabra formada aparecerá destacada al ser las fichas de un color diferente a las ya dispuestas sobre el tablero en turnos anteriores. Se podrá observar el número de jugadas válidas encontradas por el jugador de tipo robot, así como el tiempo necesitado para la obtención de todas ellas y escoger la más adecuada.

Adicionalmente, se mostrará el listado de palabras formadas junto con sus correspondientes puntuaciones.

El usuario no deberá más que indicar *Siguiente turno* en el momento que desee que juegue el siguiente jugador de la partida.

Figura iii.11. Turno de jugador de tipo *robot*

iii.5.2. TURNO JUGADOR HUMANO

Los jugadores de tipo humano, deberán realizar la jugada interaccionando con el tablero de juego y las fichas contenidas en su atril. Para ello no tendrán mas que escoger al ficha del atril que desean colocar sobre el tablero haciendo *click* sobre ella, para posteriormente colocarla en la casilla del tablero deseada volviendo a hacer *click* sobre ésta. La ficha escogida

desaparecerá del atril ya que será visualizada allá donde se situé el ratón sobre el tablero de juego, dando así una total sensación de control sobre la formación de palabras.

Para mover una ficha colocada en el mismo turno que tiene lugar a una casilla diferente, el usuario no deberá más que seleccionar la ficha del tablero que desea cambiar de casilla para volverla a colocar sobre otra casilla del tablero.

Para que la ficha seleccionada vuelva al atril, el usuario deberá hacer *click* con el botón derecho, y automáticamente la ficha será devuelta al atril del jugador.

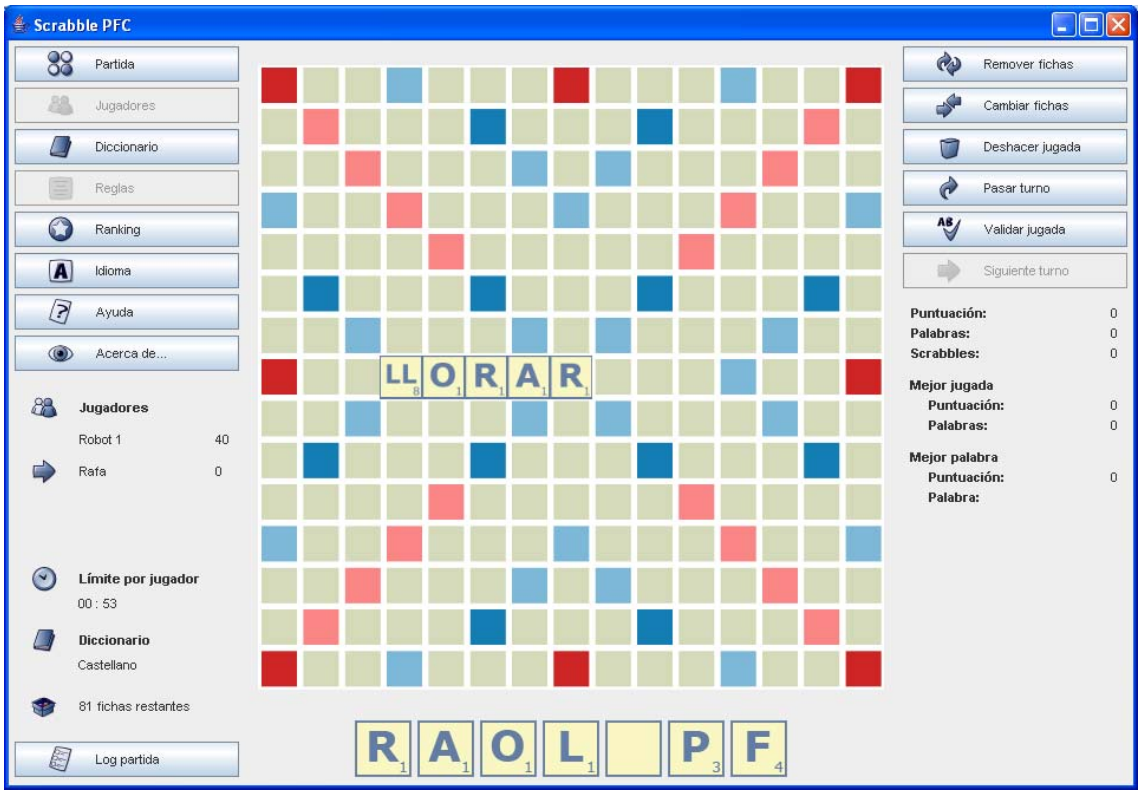


Figura iii.12. Turno de jugador de tipo *humano*

iii.5.2.1. ACCIONES

Un jugador de tipo humano podrá realizar las siguiente acciones durante el transcurso de su turno:

- Remover fichas
- Cambiar fichas
- Deshacer jugada

- Pasar turno
- Validar jugada

De forma puntual, cuando la situación del tablero y fichas contenidas en su propio atril se lo permitan, también podrá realizar las acciones de:

- Reciclar comodines
- Cambiar todas las fichas si estas son todas vocales o todas consonantes (sin contar las fichas comodín).

Figura iii.13. Listado de acciones por parte del jugador humano

iii.5.2.2. CAMBIAR FICHAS

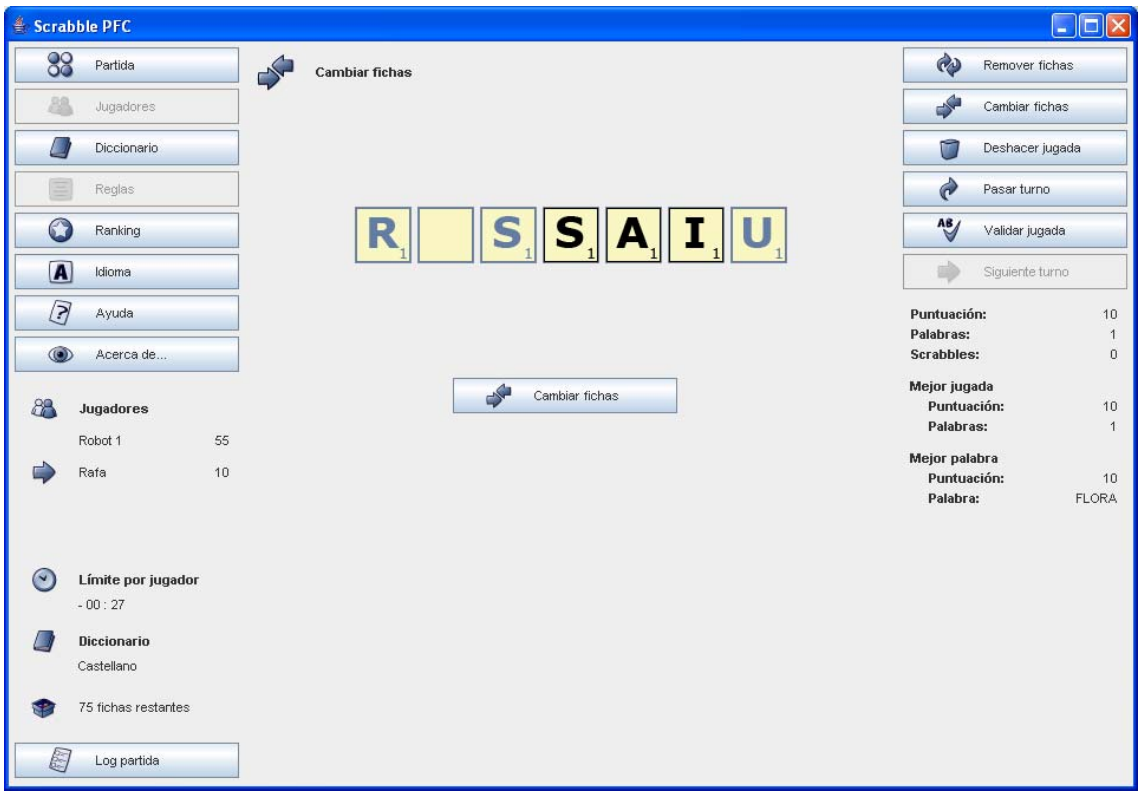


Figura iii.14. Pantalla de cambio de fichas

El usuario podrá cambiar las fichas que considere más oportunas de su atril si así lo desea en lugar de realizar alguna jugada. Para ello, no debe más que indicarlo en *Cambiar fichas*. Se mostrará una pantalla similar a la figura *iii.14* con sus propias fichas del atril, de las que deberá seleccionar aquellas que desee devolver al saco haciendo *click* sobre ellas. Las fichas seleccionadas podrán identificarse ya que resaltarán sobre el resto de fichas.

Para realizar el cambio de fichas tan solo deberá indicarlo en el botón *Cambiar fichas* situado bajo las fichas mostradas, y automáticamente serán reemplazadas por las nuevas fichas obtenidas del saco.

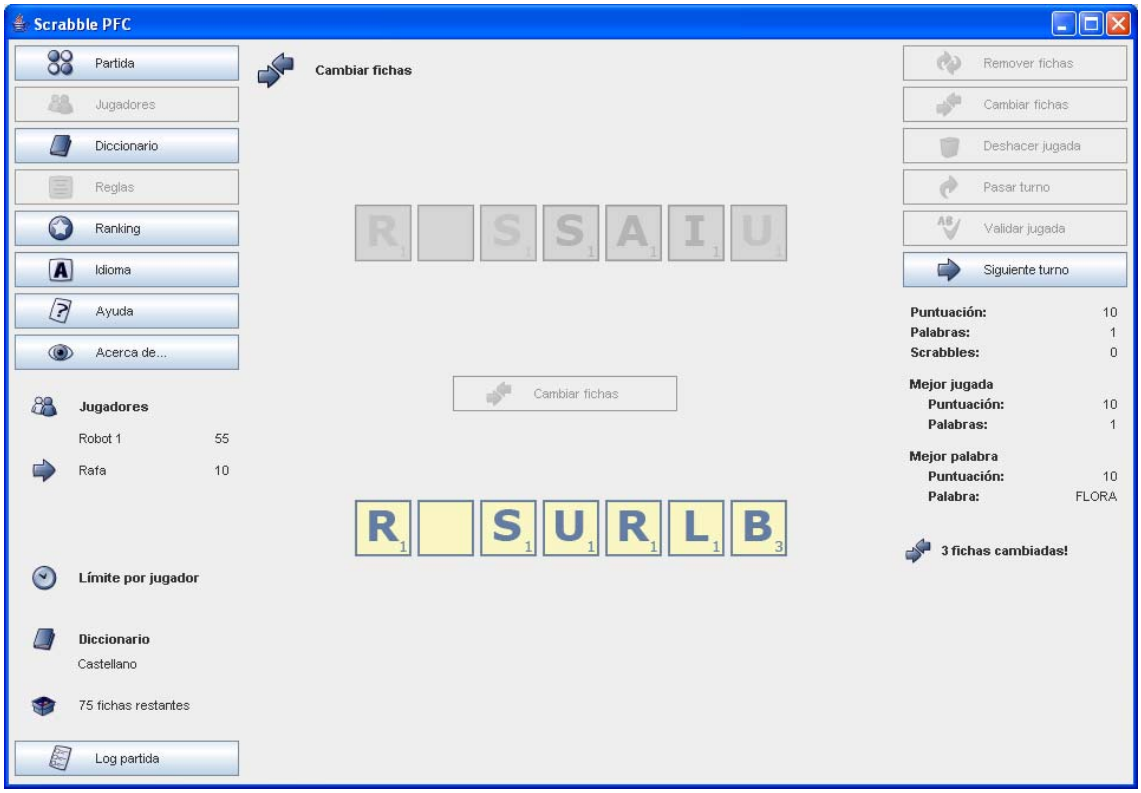


Figura iii.15. Pantalla de cambio de fichas

iii.5.2.3. VALIDAR JUGADA

Para validar la jugada realizada sobre el tablero de juego, el usuario no debe más que indicarlo en *Validar jugada*. Automáticamente se comprobará que la colocación de todas las fichas utilizadas sea correcta, y que todas las nuevas palabras formadas sean validas según el diccionario de juego escogido en la configuración de la partida.

Seguidamente, se mostrará la lista de todas las palabras formada junto a sus correspondientes puntuaciones en caso en que la jugada haya sido correcta. Si por el contrario la jugada es incorrecto debido a que alguna de las palabras formadas no es válida, se mostrará una lista con todas aquellas palabras formadas que sean invalidas. Si el usuario cree que alguna de las palabras invalidadas realmente sí que es válida, puede incluirla en ese mismo momento en el diccionario, de manera que al volver a validar la jugada, dicha palabra o palabras sí que serán válidas y quedarán incorporadas en el diccionario para futuras partidas.

Figura iii.16. Validación de jugada correcta

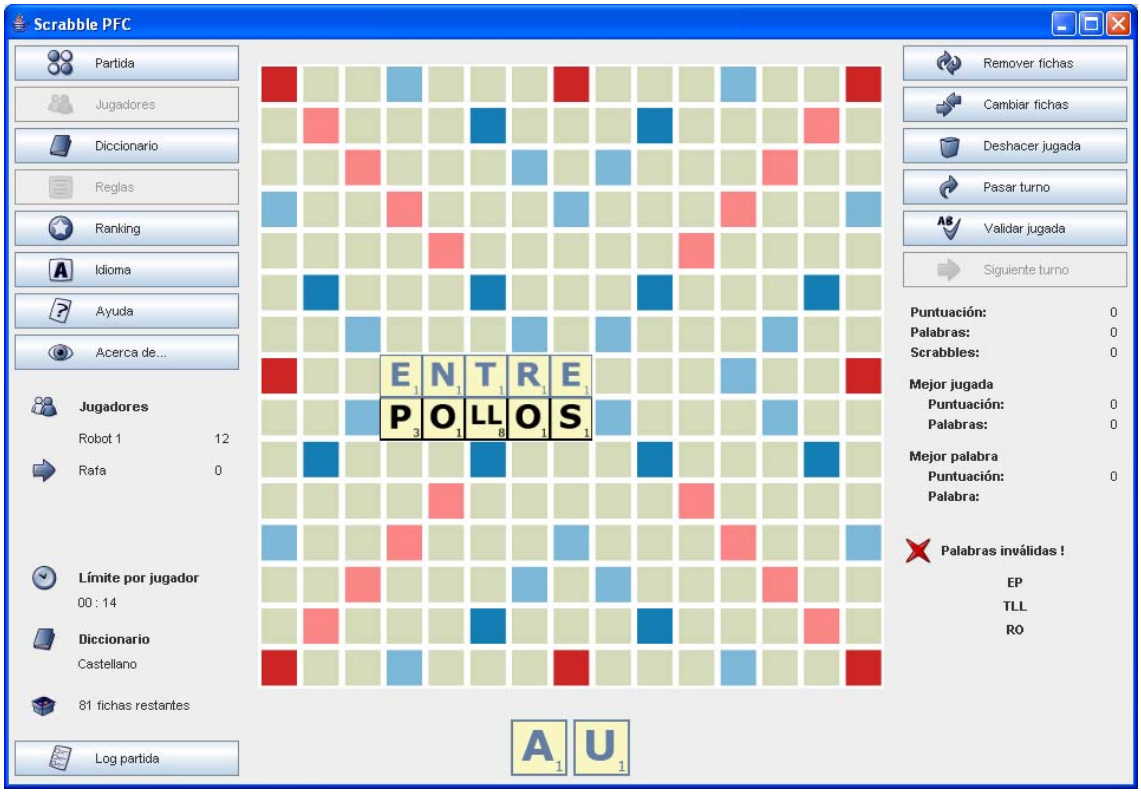


Figura iii.17. Validación de jugada incorrecta

iii.6. RANKING

Desde la pantalla de ranking el usuario podrá consultar diferentes clasificaciones de los jugadores de la aplicación según diversos criterios:

- Partidas ganadas
- Partidas perdidas
- Puntuación
- Scrabbles
- Palabras formadas
- Puntuación de la mejor partida
- Puntuación de la mejor jugada
- Puntuación de la mejor palabra formada

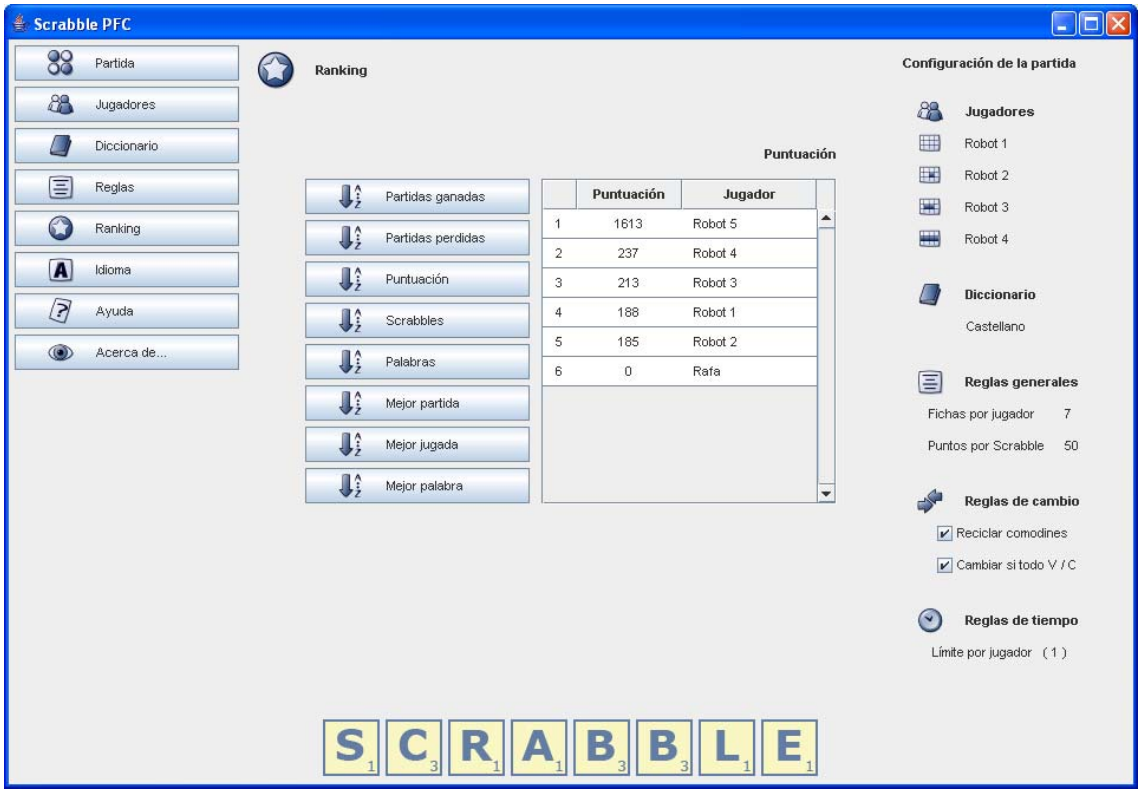


Figura iii.18. Pantalla de rankings según puntuación

iii.7. IDIOMA DE LOCALIZACIÓN

Desde la pantalla de selección del idioma de localización, el usuario podrá seleccionar el idioma en que la aplicación será visualizada, así como definirlo por omisión para futuras ejecuciones de la aplicación.

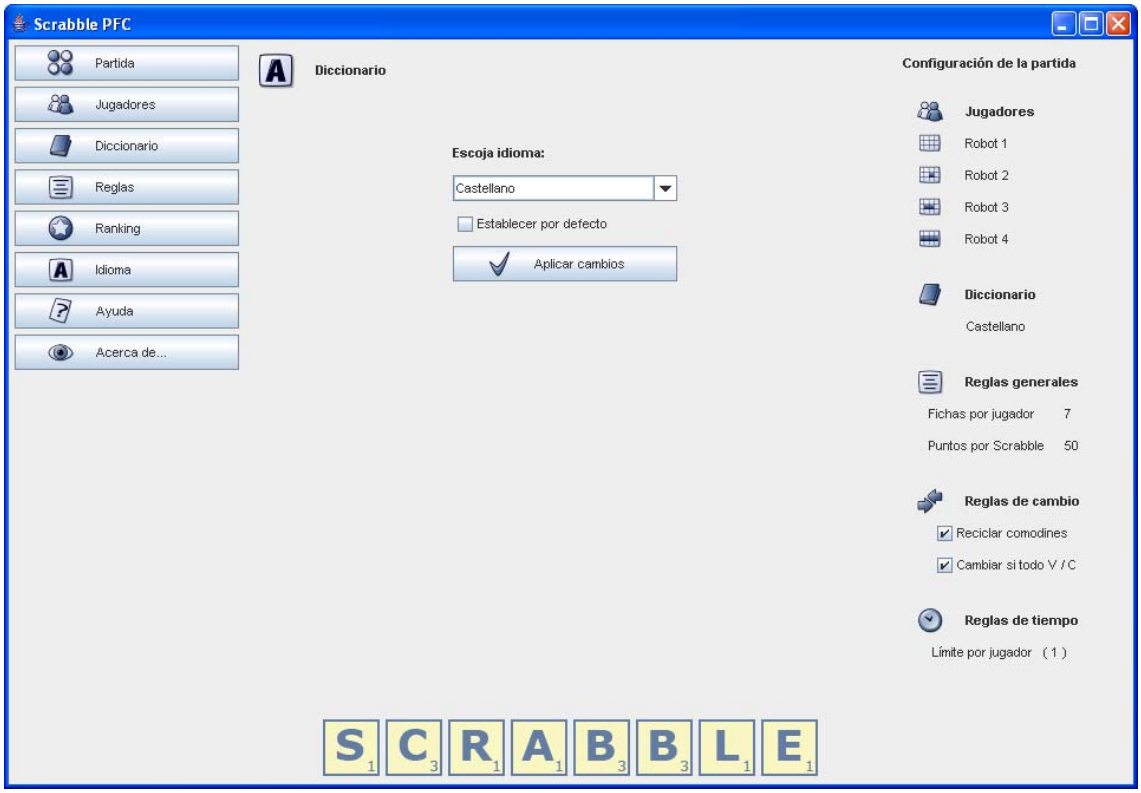


Figura iii.19. Pantalla de selección del idioma de localización

iii.8. AYUDA

Desde la pantalla de ayuda, el usuario podrá consultar las instrucciones de juego del *Scrabble*. Dichas instrucciones corresponden a las detalladas en el *Anexo 2. Instrucciones de juego* de este documento.

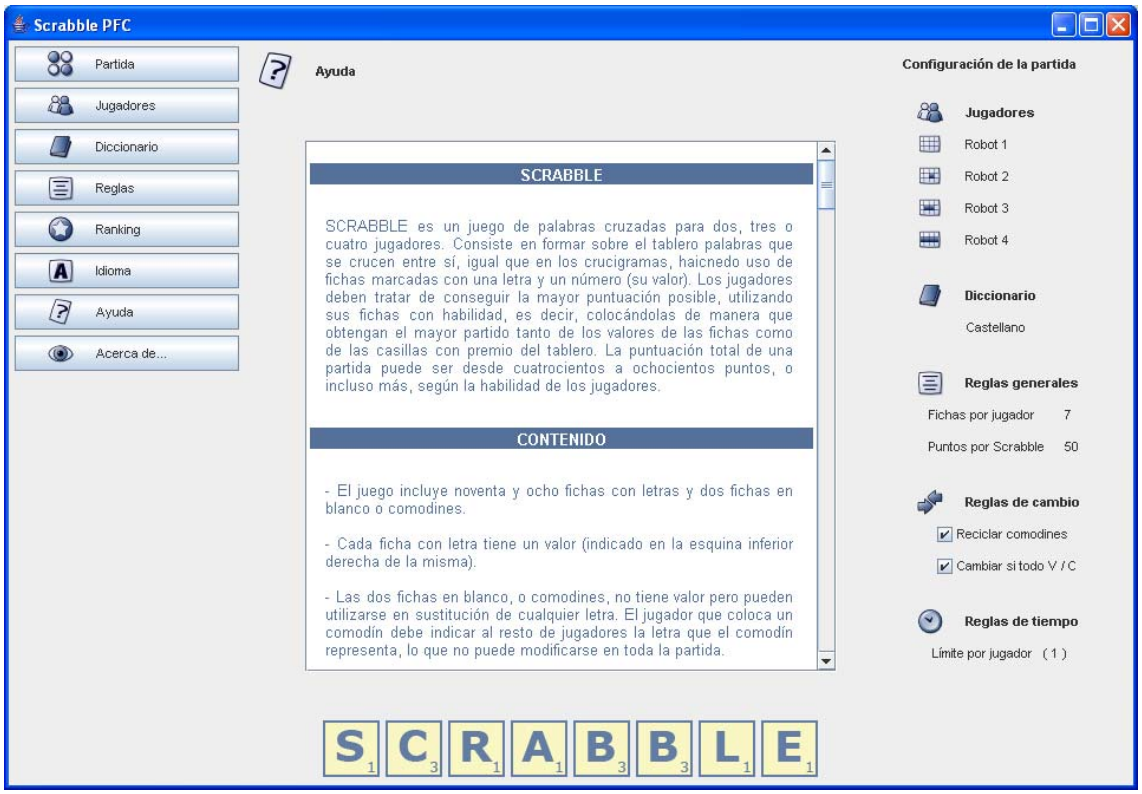


Figura iii.20. Pantalla de ayuda

iii.9. ACERCA DE...

Desde la pantalla de *Acerca de...* el usuario podrá consultar toda la información referente al proyecto:

- Proyecto
- Alumno
- Titulación
- Departamento
- Créditos
- Modalidad
- Director
- Presidente
- Vocal

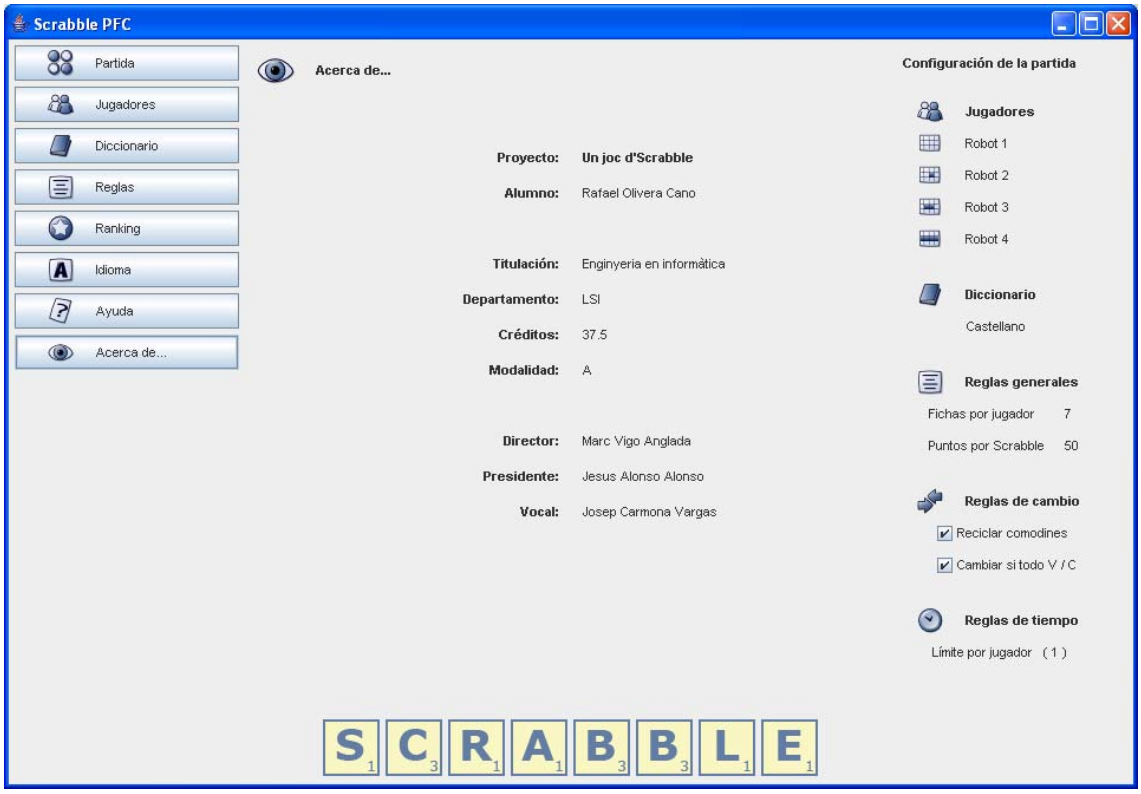


Figura iii.21. Pantalla de Acerca de...