



DISEÑO UML

INGENIERÍA DEL SOFTWARE II

Grupo 05

Guillermo García Patiño Lenza
Tania Romero Segura
María Cristina Alameda Salas
Alejandro Rivera León
Gema Blanco Núñez
David Cruza Sesmero

ÍNDICE

INTRODUCCIÓN	7
ARQUITECTURA	8
DISEÑO	8
<u>3.1 Generar mazo de fichas</u>	<u>8</u>
Explicación	8
Diagramas	9
<u>3.2 Ver el tablero</u>	<u>12</u>
Explicación	12
Diagramas	13
<u>3.3 Colocar fichas en el tablero</u>	<u>15</u>
Explicación	15
Diagramas	16
<u>3.4 Quitar fichas del tablero</u>	<u>19</u>
Explicación	19
Diagramas	19
<u>3.5 Saber el número de fichas que quedan por robar</u>	<u>22</u>
Explicación	22
Diagramas	22
<u>3.6 Rellenar mano del jugador</u>	<u>22</u>
Explicación	23
Diagramas	23
<u>3.8 Saber mi conjunto de fichas actual</u>	<u>26</u>
Explicación	26
Diagramas	26
<u>3.9 Saber cuántos puntos vale cada ficha</u>	<u>27</u>
Explicación	27
Diagramas	27

3.10 Salir	28
Explicación	28
Diagrama	28
3.11 Pedir New Game o Load Game	28
Explicación	29
Diagrama	29
<u>3.12 Verificar una palabra</u>	<u>29</u>
Explicación	29
Diagramas	29
<u>3.13 Ganar puntos</u>	<u>31</u>
Explicación	31
Diagramas	32
<u>3.14 Conocer turnos de la partida</u>	<u>34</u>
Explicación	34
Diagramas	35
<u>3.15 Pasar de turno</u>	<u>37</u>
Explicación	37
Diagramas	37
3.17 Introducir nick	41
Explicación	41
Diagramas	41
<u>3.18 Colocar más de una ficha por turno</u>	<u>42</u>
Explicación	42
Diagramas	43
<u>3.19 Jugar con otro jugador</u>	<u>45</u>
Explicación	45
Diagramas	46
<u>3.20 Saber quién empieza la partida</u>	<u>48</u>
Explicación	48
Diagramas	48
<u>3.21 Tener en la pila de fichas dos comodines</u>	<u>52</u>

Explicación	53
Diagramas	53
3.22 Canjear comodines de la mano por letras	53
Explicación	54
Diagramas	54
<u>3.23 Disponer de casillas especiales</u>	<u>54</u>
Explicación	55
Diagramas	55
<u>3.24 Poner la primera ficha de la partida en el centro</u>	<u>56</u>
Explicación	56
Diagramas	57
3.25 Guardar partida a medias	58
Explicación	58
Diagramas	59
<u>3.26 Cargar partida a medias</u>	<u>61</u>
Explicación	61
Diagrama	62
<u>3.28 Verificar las palabras automáticamente al finalizar el turno</u>	<u>63</u>
Explicación	63
Diagrama	66
<u>3.29 No pasar de turno si he colocado alguna ficha que no forme palabra</u>	<u>69</u>
Explicación	69
Diagramas	70
<u>3.29 Obtener monedas</u>	<u>72</u>
Explicación	72
Diagramas	73
<u>3.30 Solo colocar fichas al lado de fichas</u>	<u>74</u>
Explicación	75
Diagramas	76
<u>3.31 Si finaliza la partida obtener puntos por monedas sobrantes</u>	<u>77</u>
Descripción	77

Diagramas	77
<u>3.34 Cambiar una ficha por un comodín a cambio de monedas</u>	<u>79</u>
Explicación	79
Diagramas	80
<u>3.35 Saltar a un jugador a cambio de monedas</u>	<u>83</u>
Explicación	83
Diagramas	83
<u>3.36 Invertir el orden de jugadores a cambio de monedas</u>	<u>86</u>
Explicación	86
Diagramas	87
<u>3.37 Poder ver la mano</u>	<u>89</u>
Explicación	89
Diagramas	90
<u>3.38 Poder ver el tablero</u>	<u>92</u>
Explicación	92
Diagramas	93
<u>3.39 Poder ver los turnos en la GUI</u>	<u>97</u>
Explicación	97
Diagramas	98
<u>3.40 Mostrar información</u>	<u>98</u>
Explicación	99
Diagrama	99
<u>3.41 Poder ver los puntos de los Jugadores en la GUI</u>	<u>99</u>
Explicación	100
Diagramas	100
<u>3.42 Poder ver las monedas de los jugadores en la GUI</u>	<u>102</u>
Explicación	102
Diagramas	102
<u>3.43 Poder cargar una partida desde la GUI</u>	<u>104</u>
Explicación	104
<u>3.43 Poder guardar una partida desde la GUI</u>	<u>105</u>

Explicación	105
<u>3.44 Colocar una ficha en el tablero desde la GUI</u>	<u>106</u>
Explicación	106
Diagramas	<u>107</u>
<u>4.45 Poder cambiar una ficha de la mano desde la GUI</u>	<u>109</u>
Explicación	110
Diagramas	<u>110</u>
<u>4.48 Poder jugar con jugadores controlados por la IA</u>	<u>112</u>
Explicación	112
Diagramas	<u>113</u>
<u>4.49 Poder pasar turno desde la GUI</u>	<u>115</u>
Explicación	115
Diagramas	<u>116</u>
<u>4.50 Poder comprar un comodín en la GUI</u>	<u>119</u>
Explicación	119
Diagramas	<u>119</u>
<u>4.51 Poder saltar jugador en la GUI</u>	<u>119</u>
Explicación	120
Diagramas	120
<u>4.52 Poder pasar turno desde la GUI</u>	<u>120</u>
Explicación	121
Diagramas	<u>121</u>
<u>4.54 Poder jugar en red con otros jugadores</u>	<u>121</u>
Explicación	<u>122</u>
<u>4.55 Iniciar una partida desde GUI</u>	<u>122</u>
Explicación	123
Diagramas	123
<u>Diagrama general del modelo</u>	<u>123</u>

1. INTRODUCCIÓN

En este documento vamos a exponer la arquitectura y diseño de nuestro proyecto. En el apartado de diseño, se describirá detalladamente el diseño para cada historia de usuario, así como su posible evolución y modificación a o largo del proyecto.

Para facilitar la explicación del diseño de cada historia de usuario, nos apoyaremos en diagramas UML. Seguida de la explicación del diseño, aparecerán los diagramas de clases y secuencia correspondientes.

2. ARQUITECTURA

A continuación, vamos a describir la arquitectura de nuestra aplicación, destacando el patrón modelo vista controlador y cliente servidor. También aportaremos una serie de diagramas que nos permitan ver a vista de pájaro la arquitectura general.

2.1 Patrón Modelo Vista Controlador

Cuando nos propusimos introducir una vista por GUI en el programa, nos vimos forzados a aplicar el patrón modelo-vista-controlador. En esta sección se explican las funciones de cada una de las tres partes mencionadas, además de la interacción entre éstas.

Por lo general, este patrón divide las clases del programa según sus responsabilidades en tres categorías:

- El modelo contiene los objetos que implementan la lógica del juego y los hace interactuar entre ellos.
- La vista se encarga de representar la información que contiene el modelo y los cambios que se producen en él. Además, en nuestro caso, la vista sirve también como medio de entrada de datos para el usuario.
- El controller toma el papel de intermediario entre la vista y el modelo, y proporciona a la vista un medio para acceder a los datos del modelo. Además, permite al usuario introducir datos en el modelo para provocar cambios en él.

Con el fin de implementar este patrón modelo-vista-controlador, hemos aplicado también el patrón observador. Este patrón nos ha servido para agrupar el comportamiento de los objetos sensibles a cambios en el modelo, y para indicar los objetos del modelo a cuyos cambios puede reaccionar la vista.

De este modo se han creado la siguiente interfaz que implementan los objetos del modelo:

- **Observable:** se trata de una interfaz genérica con métodos para manejar una colección de objetos que extienden al parámetro genérico. La implementan los objetos del modelo en los que los objetos de la vista se pueden registrar.

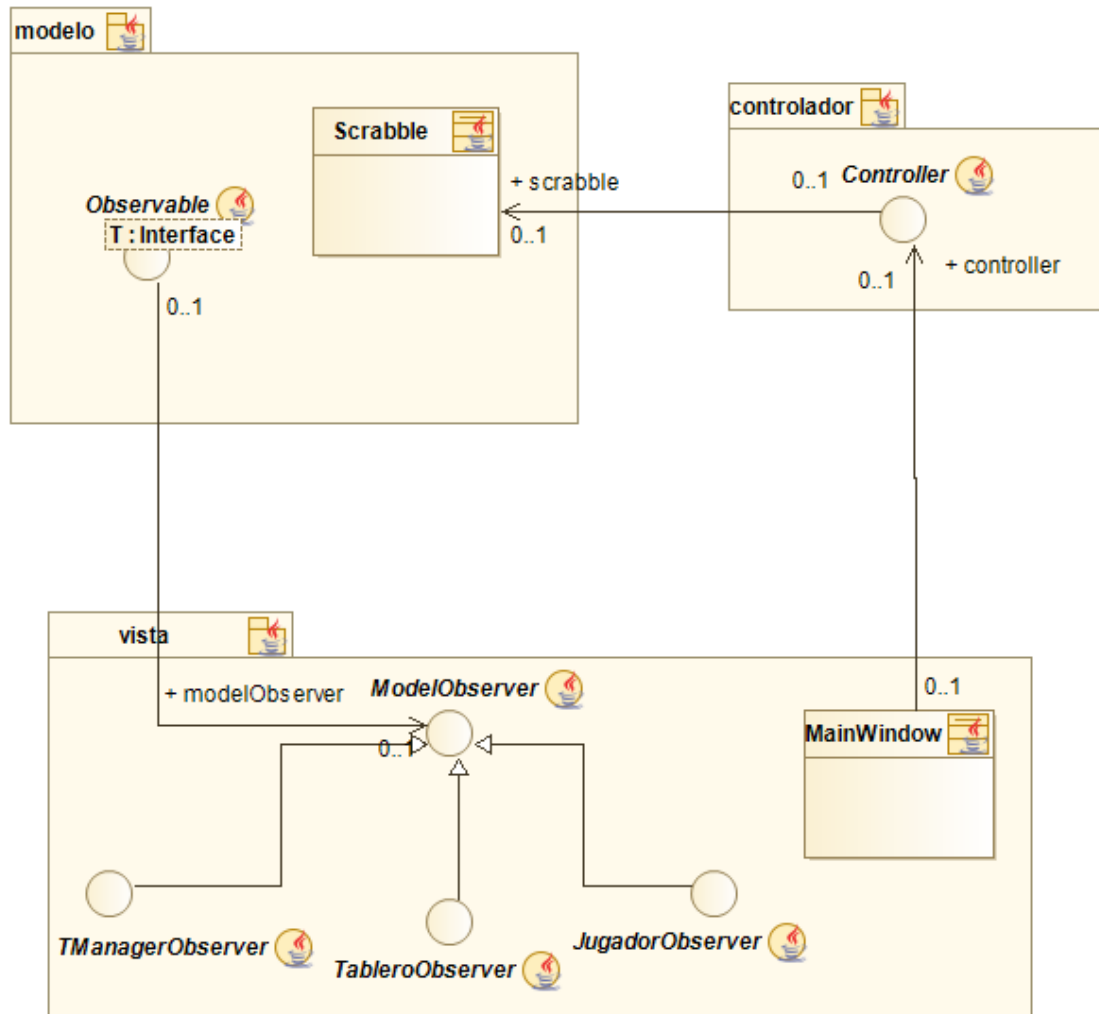
Así, cada vez que se produzca un cambio en una parte del modelo, los objetos que implementen esta interfaz pueden recorrer la colección de observadores y transmitirles los cambios que se han producido para que éstos actualicen su información.

Además, se crearon las siguientes interfaces, que implementan los objetos de la vista para reaccionar a los cambios en las diferentes partes del modelo:

- **ModelObserver:** esta interfaz contiene un método que emplean todos los objetos que reaccionan a cambios en el modelo, este es, registrarse como observadores de una parte del modelo a través del controlador.
- **TableroObserver:** los métodos que contiene esta interfaz sirven a los objetos de la vista para reaccionar a los cambios que se producen en el tablero de juego.

- **JugadorObserver**: contiene los métodos necesarios para que un objeto de la vista actualice su información de acuerdo a las fichas que tiene un jugador en su mano.
- **TManagerObserver**: agrupa métodos para que los objetos de la vista reaccionen a los cambios que suceden en el AdminTurnos.

Finalmente, incluimos un diagrama de clases en el que se representan estas comunicaciones entre los objetos de la vista, el modelo, y el controlador.



2.2 Arquitectura cliente servidor

Para incorporar a nuestro proyecto la parte de multijugador en red, hemos seguido la arquitectura Cliente-Servidor.

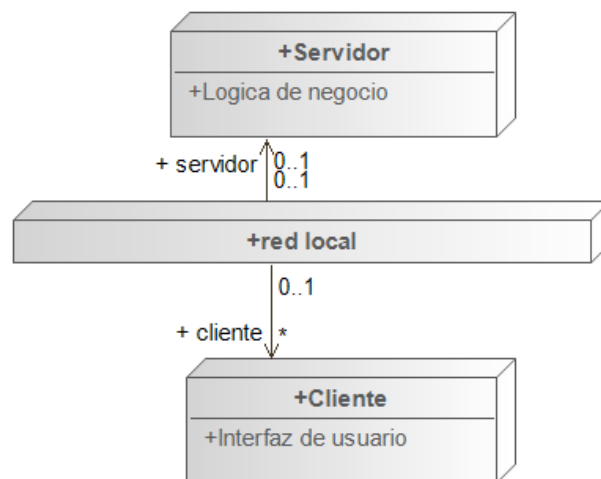
La arquitectura Cliente-Servidor es un modelo de diseño de software en el que las tareas se reparten entre el servidor y el cliente. En nuestro caso, hemos considerado el sistema de la siguiente forma:

El servidor es un proveedor de servicios y atiende a las demandas de los clientes, por tanto en él reside el modelo de nuestro proyecto (la lógica del juego).

El cliente es un demandante de servicios, en él reside la vista (la interfaz del juego).

La comunicación entre cliente y servidor se realiza a través de sockets.

Con respecto a los diagramas que muestren esta arquitectura, destacamos el diagrama de despliegue.



Para consultar los diagramas generales de la parte del cliente y del servidor, consulte el siguiente documento ([Multijugador en red.pdf](#)).

3. DISEÑO

Para comentar el diseño de cada una de las historias que se han desarrollado se han utilizado las historias tal y como se tenían antes de la elaboración del nuevo product backlog. Esto se debe a que el nuevo product backlog se estructura de manera distinta para muchas historias y hemos preferido utilizar las historias antiguas aunque se hayan eliminado o hayan pasado a ser parte

de la evolución de otra en el nuevo backlog. De esta manera queda más claro cómo se han elaborado las funcionalidades, en qué orden y quién las ha realizado.

3.1 Generar mazo de fichas

ID:H_1	Usuario:Jugador
Nombre historia:Generar mazo de fichas	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 3	Iteración asignada: 1
Programador responsable: María Cristina Alameda Salas	
Descripción: Como Jugador quiero que se genere el mazo de fichas para que pueda utilizarlas en mi mano.	

Explicación

Una de las tareas de más prioridad fue “Generar un mazo de fichas”, así constituyó una de las primeras historias de usuario que llevamos a cabo.

Para implementar esta funcionalidad en nuestro proyecto, creamos la clase “Ficha” como una abstracción de una ficha del Scrabble, también creamos una clase “GeneradorMazo”, cuya única responsabilidad era la creación de una lista de fichas mezcladas. Esta clase aislaba la creación de esta lista de fichas del propio mazo, haciendo uso del principio de diseño **Single Responsibility**. Además, quisimos que se generara a partir de un archivo externo, por ello se creó un archivo json que guardaba para cada letra oficial de Scrabble, su puntuación y su frecuencia.

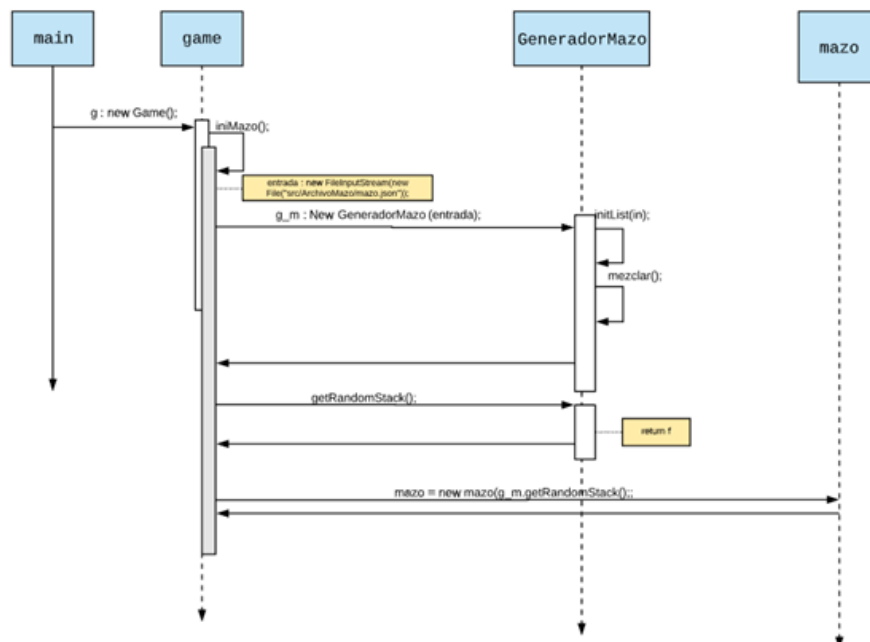
De esta manera el método `-iniList()` es el encargado de crear esta lista (todavía sin mezclar). Según va extrayendo información del archivo .json, creaba una ficha pasándole la letra y los puntos.

Para la creación de una ficha se necesitaba como acabamos de decir los puntos y la letra. En el constructor de esta clase, se le asignaban estos valores pasados por parámetros a sus atributos correspondientes. Además, se incluyó un método `generateID()` para generar un id distinto para cada ficha previendo que nos fuera útil en un futuro como ha resultado ser.

Con respecto a la evolución de esta clase, podemos decir que su estructura apenas se ha visto modificada. El único cambio que se realizó, fue sustituir la implementación del método mezclar, de un algoritmo creado por nosotros que realizaba esta acción, por una llamada a un método de la clase **Collections** que realiza la misma función. Por último, tras la refactorización, esta clase dejó de ser utilizada por la clase “Game” y pasó a ser utilizada por el Administrador de turnos y Scrabble, por ello se creó un método **-generate()-** que inicializaba la lista del generador, las mezclaba y devolvía un Mazo ya creado.

Diagramas

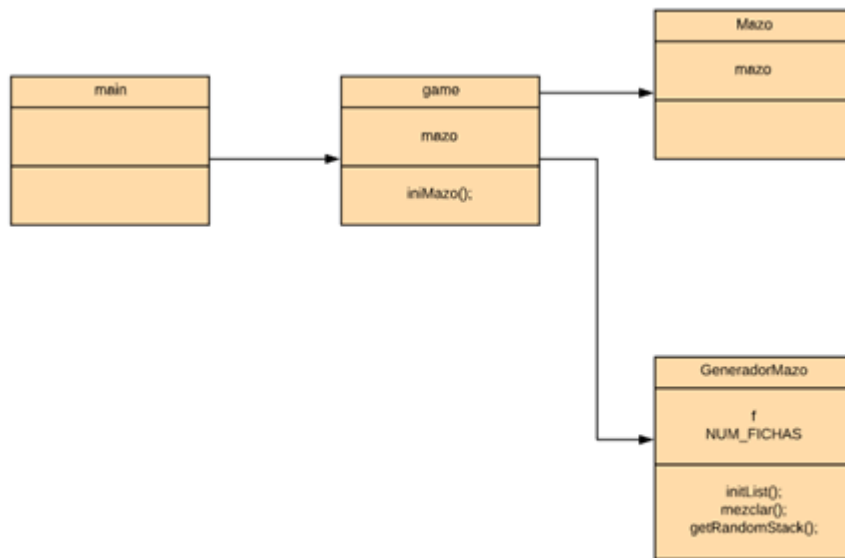
- Antiguos
 - Diagrama de secuencia



[click para ampliar la imagen](#)

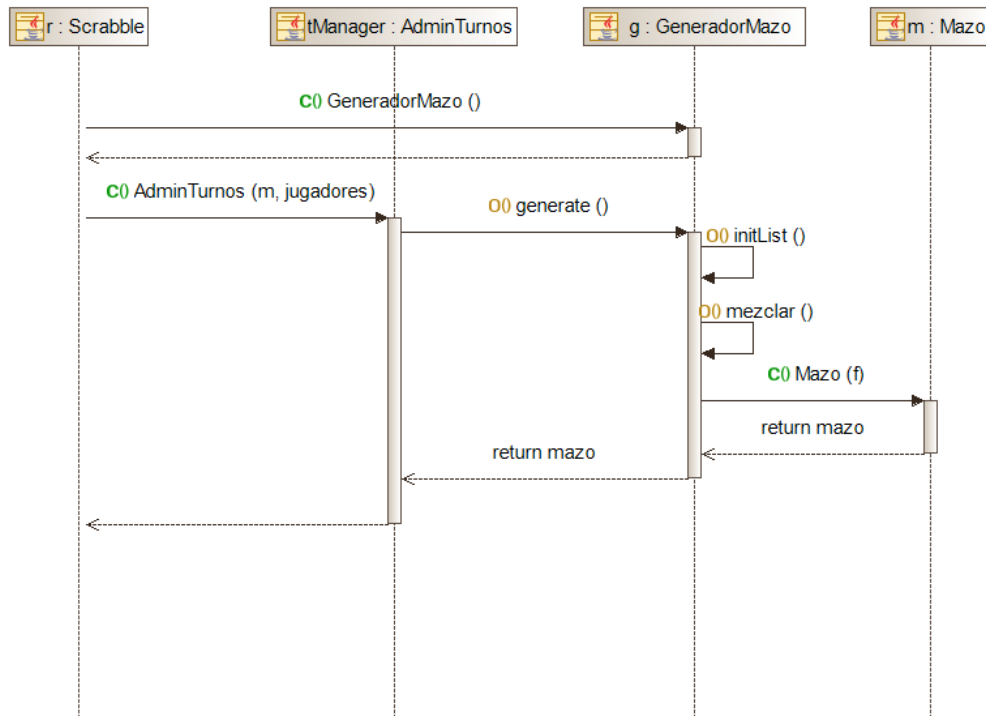
Como se puede apreciar en esta imagen. El game creaba, creaba una instancia del **GeneradorMazo** pasándole la dirección del archivo .json donde se encontraba la información de las fichas. Una vez creado, lo inicializa con el método **iniMazo()** del **Generador**. Este método primero creaba una lista de fichas en el método **iniList()** y luego las mezclaba. Por último el **Game** obtenía esta lista de fichas gracias al **getRandomStack()** que devolvía el conjunto de fichas ya mezcladas. Para construir el **Mazo** final, creaba el objeto pasándole como argumento la lista de fichas mezcladas generada en el **GeneradorMazo**.

- Diagrama de clases



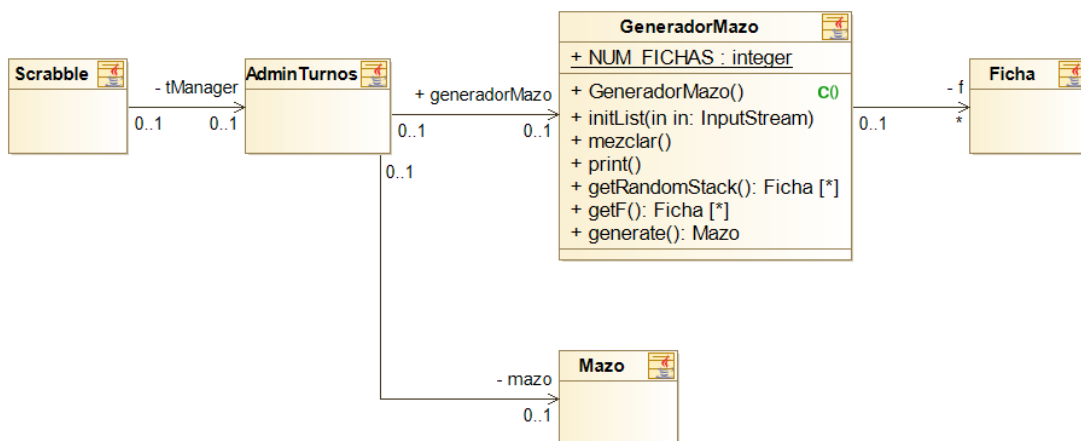
Aquí se muestra el diagrama de clases de la implementación antigua. Se puede apreciar el aislamiento entre el mazo y el generador.

- Nuevos
 - Diagrama de secuencia



[click para ampliar la imagen](#)

Como se puede apreciar, el Scrabble crea el Administrador de Turnos, pero esta clase en su constructor requiere que le pasen como argumentos una instancia de GeneradorMazo, por ello es Scrabble quien lo construye. A continuación, el administrador de turnos en su constructor inicializa sus atributos, entre los que está el mazo. Es por ello por lo que el método generate de GeneradorMazo después de crear esta lista de fichas y mezclarla, devuelve el Mazo ya creado. De esta manera el Administrador de turnos únicamente asigna este mazo su atributo mazo.



○ Diagrama de clases

En este diagrama se puede ver cómo se relacionan las distintas clases que aparecían en el diagrama anterior.

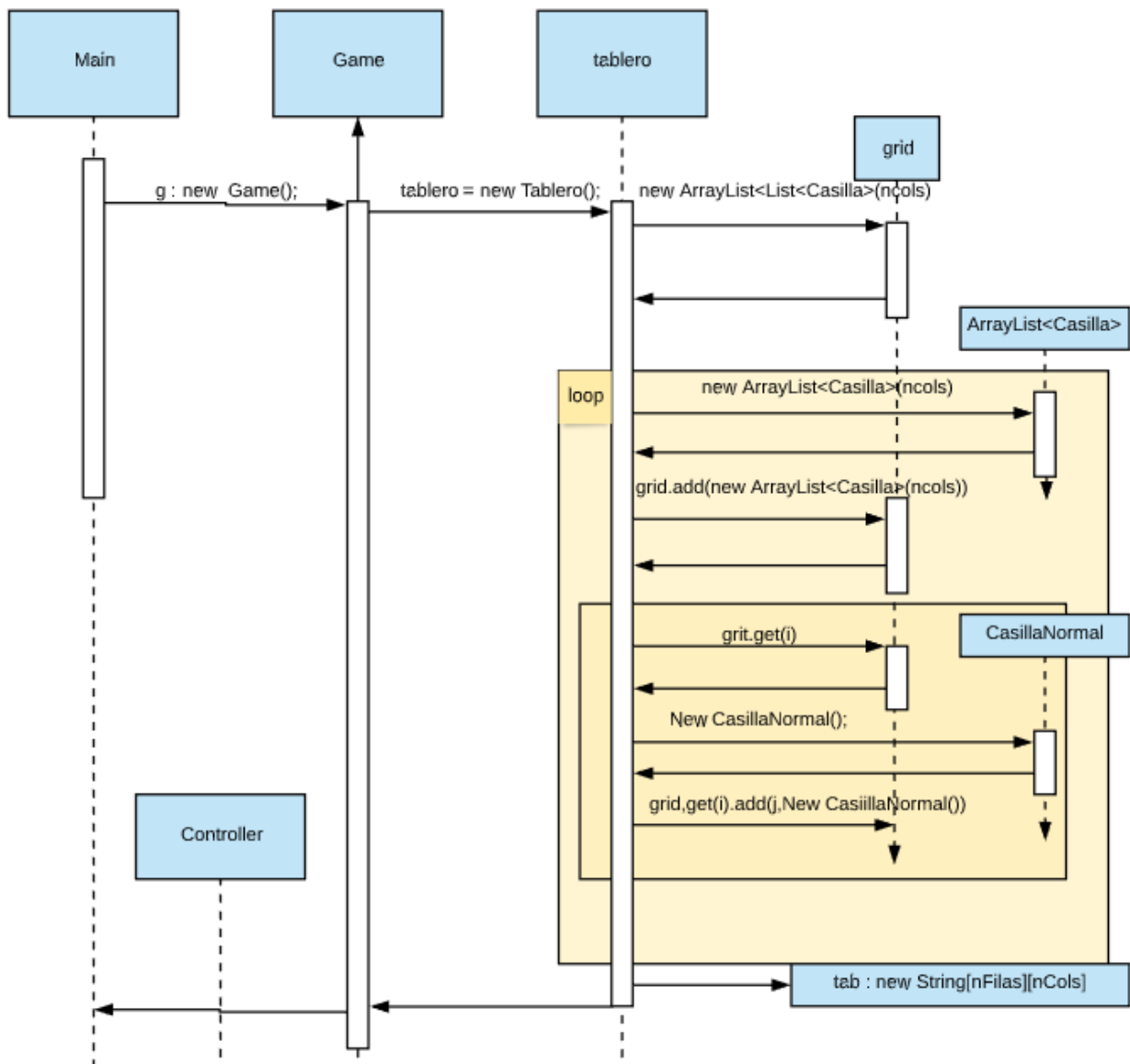
3.2 Ver el tablero

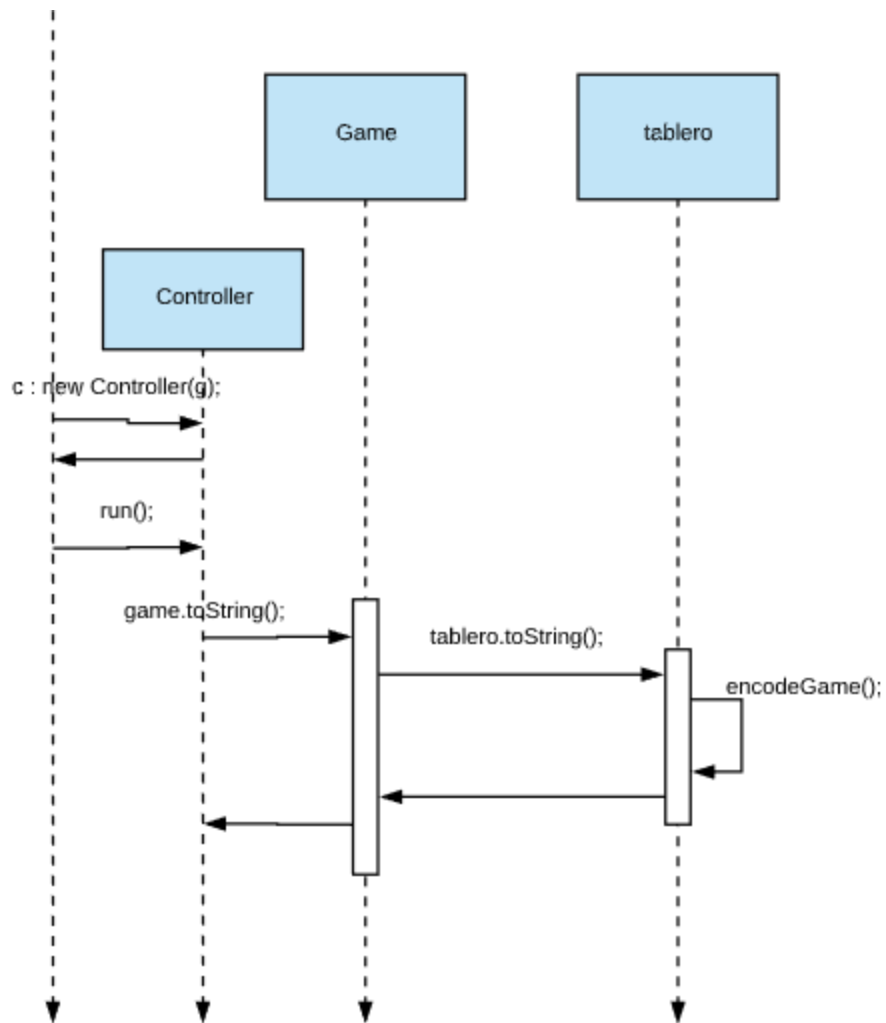
ID:H_2	Usuario:Jugador
Nombre historia:Ver el tablero	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 3	Iteración asignada: 1
Programador responsable:	
Descripción: Como jugador quiero poder ver el tablero para seguir la colocación de las fichas.	

Explicación

Para conseguir mostrar al usuario una **representación del tablero de juego** por la consola, usamos **código que se nos dio hecho en la asignatura de TP1** para este mismo fin. Usamos ese código ya que **permitía codificar un tablero en forma de matriz en una cadena de texto** que mostrar al usuario cómodamente por pantalla. Posteriormente, esta vista se eliminó y fue reemplazada por un tablero en la GUI

Diagramas





3.3 Colocar fichas en el tablero

ID:H_3	Usuario:Jugador		
Nombre historia:Colocar fichas en el tablero			
Prioridad: Alta		Estado: Terminado	
Puntos estimados: 2		Iteración asignada: 1	
Programadores responsables: María Cristina Alameda Salas, Alejandro Rivera León, Guillermo García Patiño Lenza			
Descripción: Como jugador quiero poder colocar fichas al tablero para poder formar palabras.			

Explicación

Durante esta etapa tan temprana del desarrollo **nuestra principal influencia fue** todo lo aprendido en **la asignatura de TP1**.

Esto nos fue de gran ayuda ya que **en la práctica de dicha asignatura empleamos el patrón MVC y el patrón comando** (realmente no sabíamos que eran, pero se nos mencionaron sus nombre) por lo que nos pareció correcto empezar tomando dicha práctica como modelo a seguir dado nuestro desconocimiento en el ámbito del diseño.

En cuanto al diseño, como se ha mencionado anteriormente, utilizamos el **patrón comando** para gestionar todo lo relacionado con los comandos que podría usar el usuario.

Esto se hizo así en parte porque **en TP1 habíamos tenido que usar esta estructura** y nos pareció muy conveniente emularla en el proyecto.

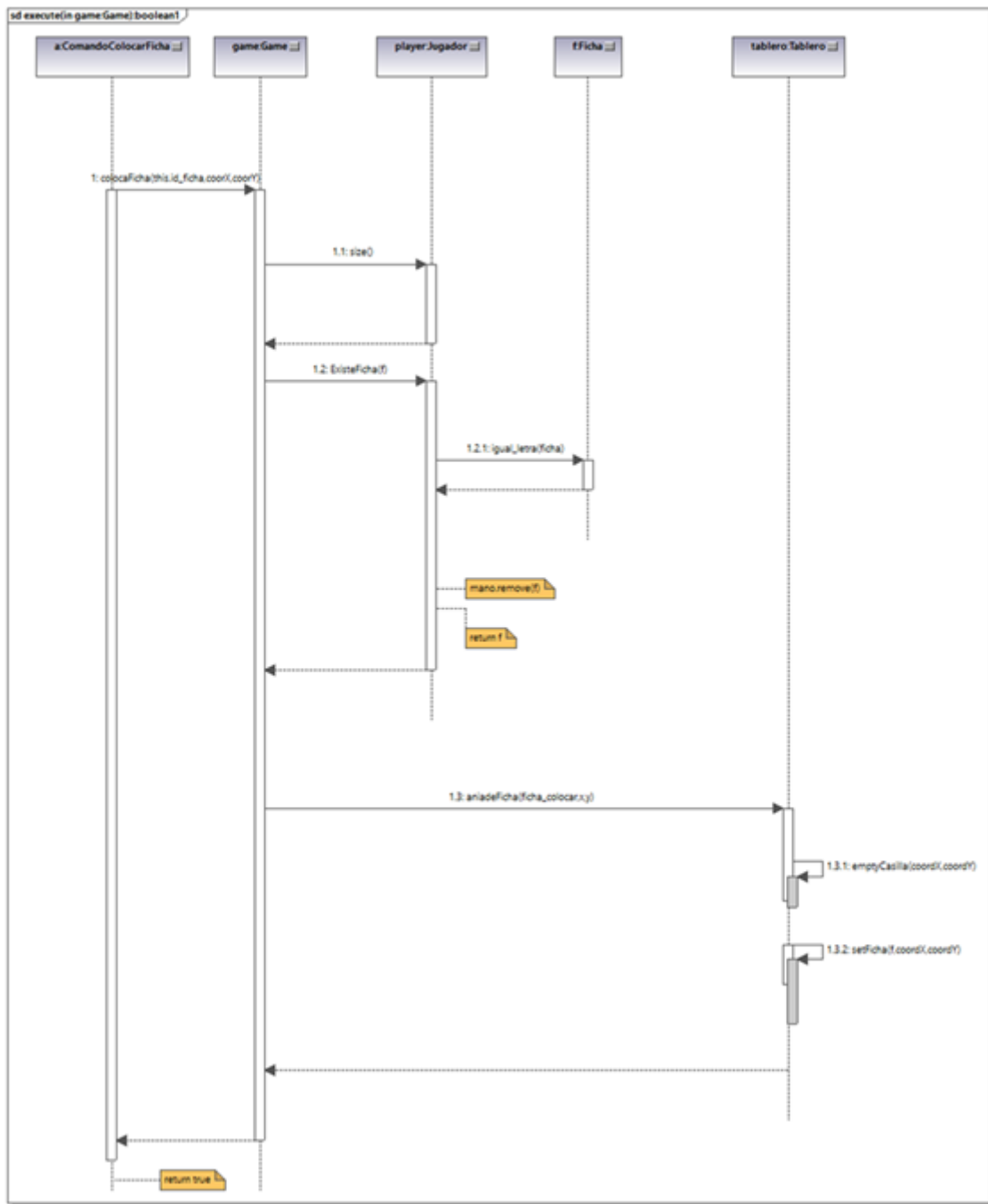
Ahora, **después de todo el conocimiento relativo al diseño** que hemos adquirido gracias a IS 2 **podemos afirmar que fue una gran decisión** debido principalmente al elevado número de comandos que tenemos y **a la gran escalabilidad que este patrón nos ofrece.**

En cuanto a los diagramas, como se puede observar, **la clase Game** (aparte de contener la lógica del juego) **hace uso del patrón de baja cohesión al delegar acciones en la clase Tablero y en la clase Jugador** entre otras.

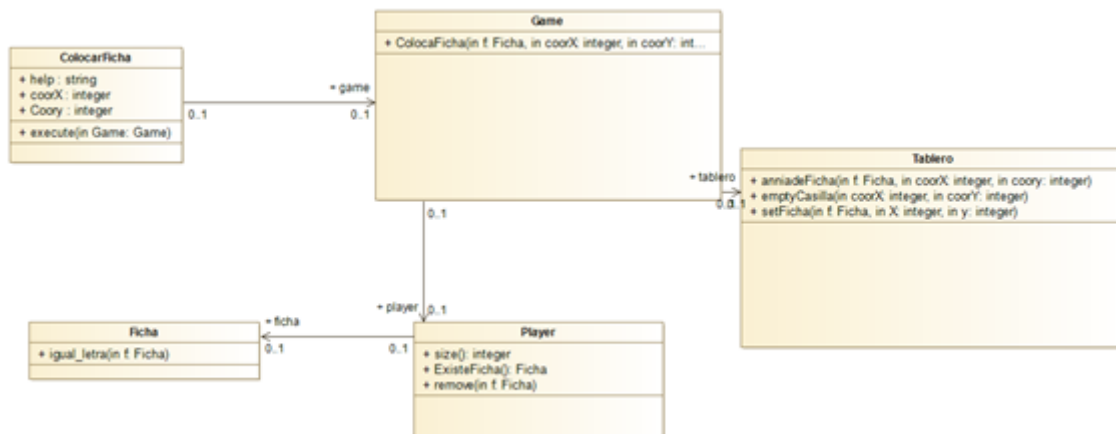
Esto se hizo una vez más **emulando lo aprendido en TP1** y los métodos que utilizábamos en dicha asignatura.

Referente a la clase Jugador, como puede verse en los diagramas, a pesar de que desconocíamos los patrones de diseño decidimos que jugador fuese responsable de comprobar si la ficha seleccionada estaba entre las fichas que conformaban su mano

Diagramas



[Click para verlo más grande.](#)



[Click para verlo más grande.](#)

3.4 Quitar fichas del tablero

ID:H_4	Usuario:Jugador
Nombre historia:Quitar fichas en el tablero	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 2	Iteración asignada: 1
Programador responsable:	
Descripción: Como jugador quiero quitar las fichas que he colocado este turno del tablero para poder rectificar y hacer una mejor jugada.	

Explicación

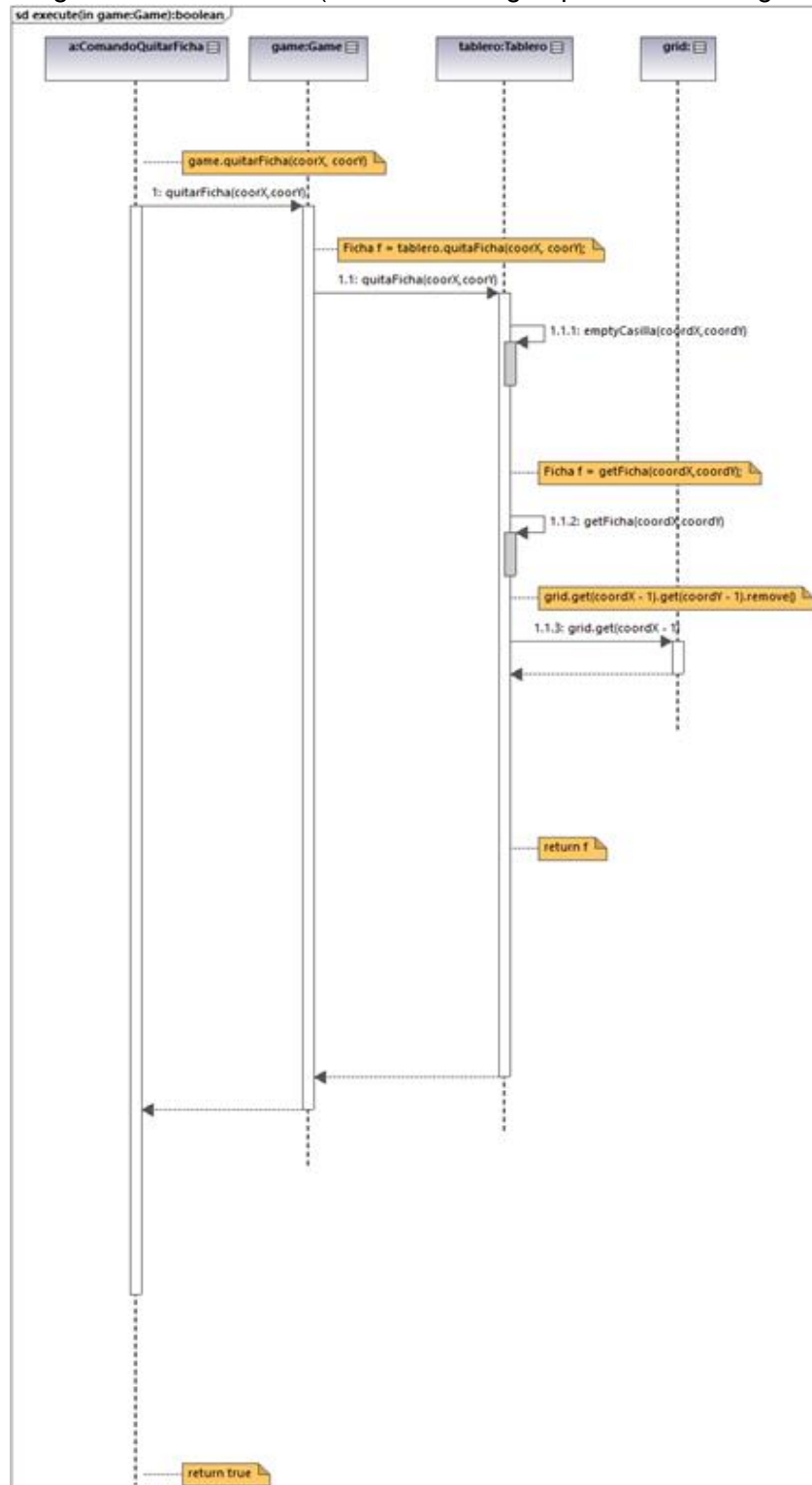
Para quitar fichas se ha empleado **siempre** uno de los **comandos**. Los únicos **cambios** que ha sufrido esta implementación han sido **provocados por la transformación** del juego para **guiarse por los eventos** producidos desde la GUI, y con el **refactor** que sufrió el **modelo** durante las últimas etapas del desarrollo.

- Cuando se **introdujo la GUI** en el juego, se **eliminó la ejecución del comando** vía texto en **consola**, y se le otorgó la funcionalidad a las **casillas** del tablero, que podrían **enviar al modelo la orden** de retirar la ficha que contenían **al hacer click en ellas**.
- Por otro lado, al **refactorizar el modelo** para obtener un **diseño más limpio**, la **manera de ejecutarse de los comandos** cambió, al extraerse muchos

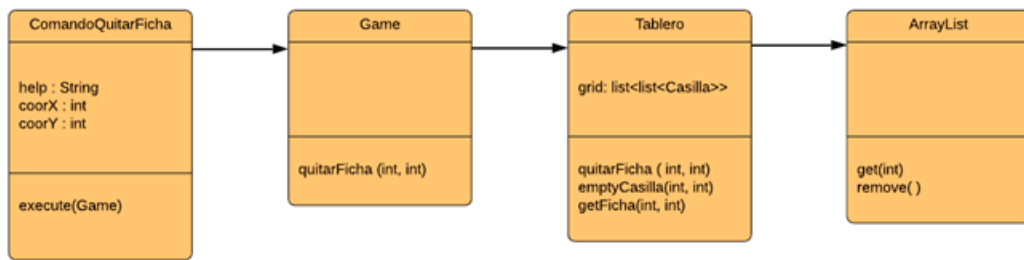
atributos y funcionalidades del Game. Para **más detalles**, ver el documento [‘Refactor Modelo’](#)

Diagramas

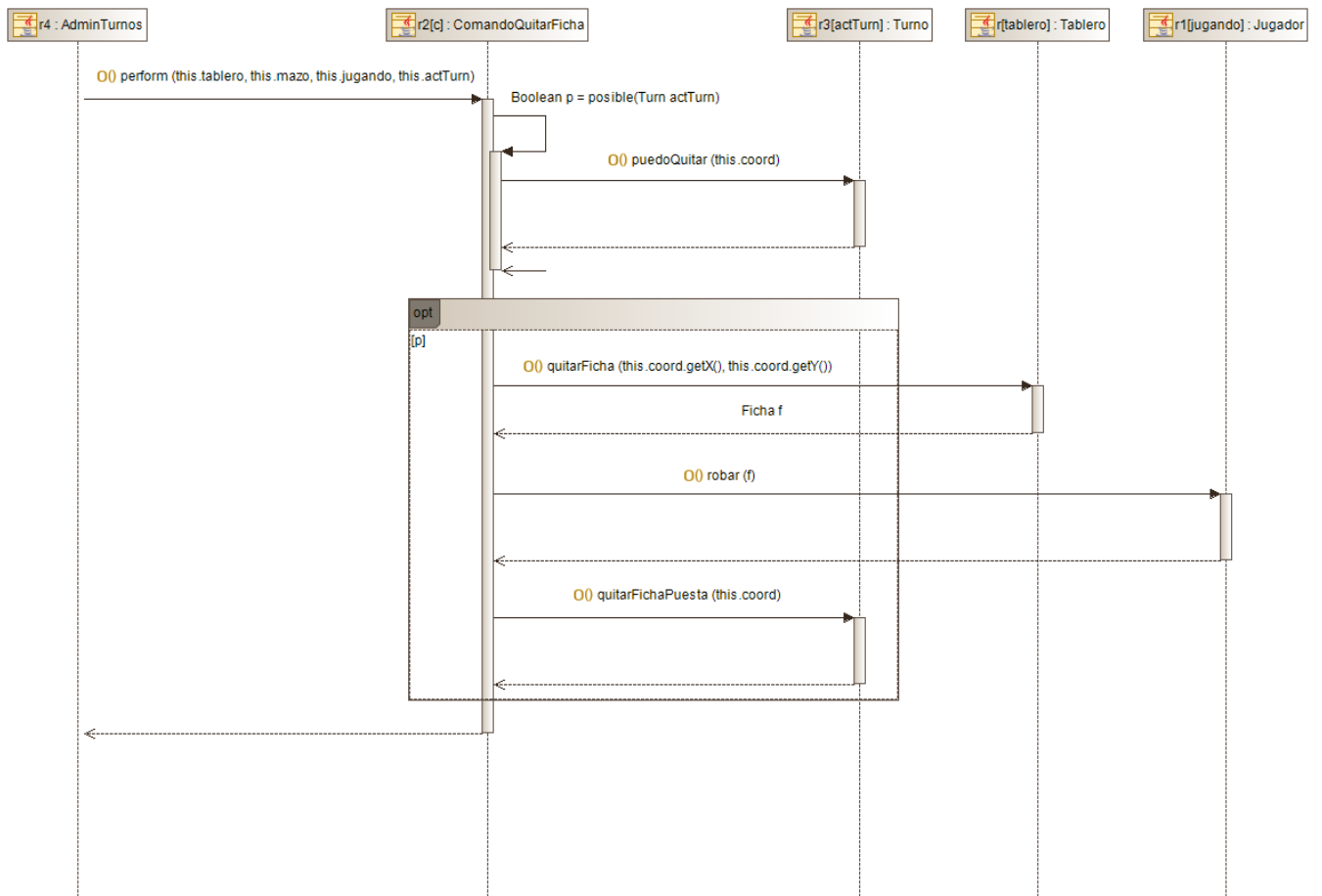
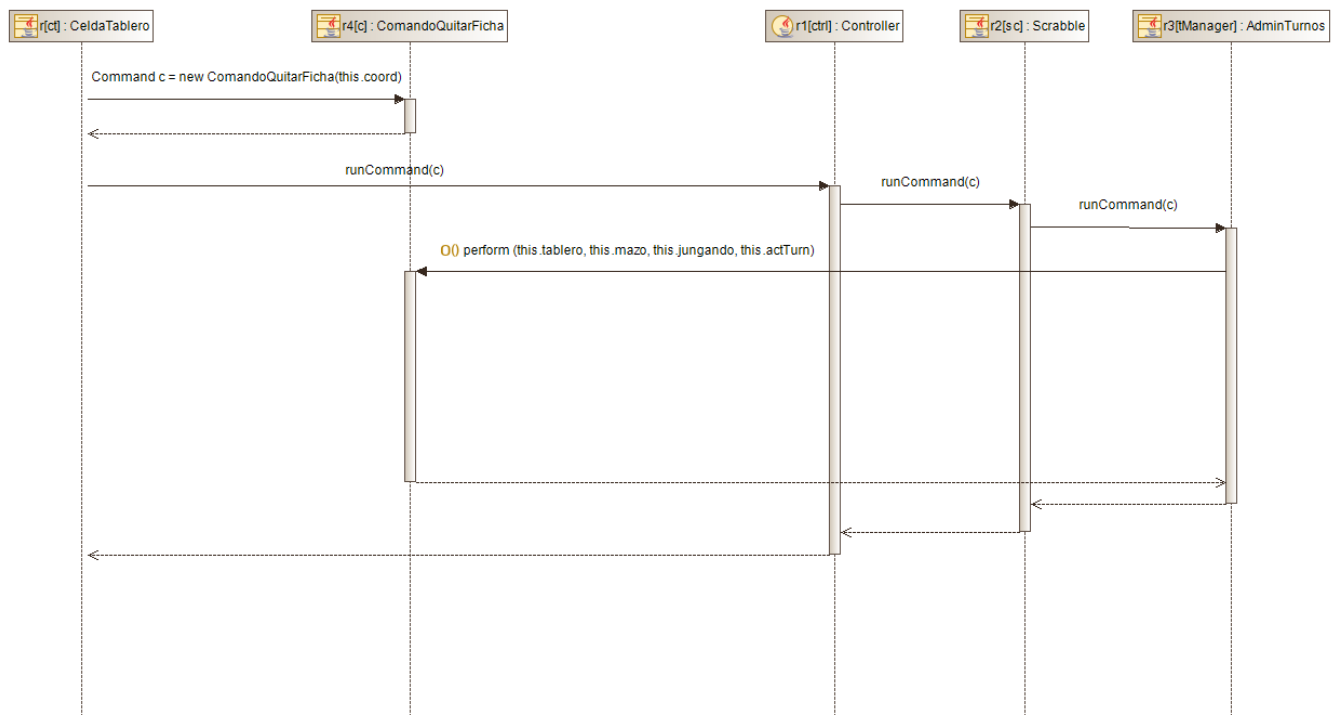
- Antiguos
 - Diagrama de secuencia (click en la imagen para verla más grande)

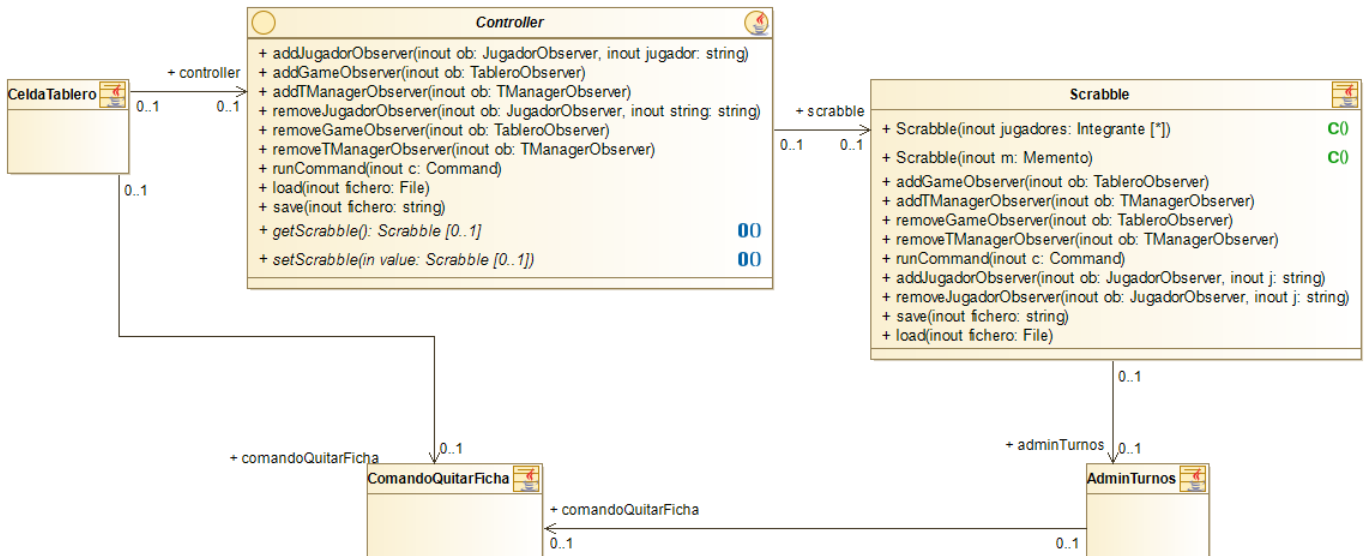


○ Diagrama de clases (click para ver la imagen más grande)



○ Nuevos:





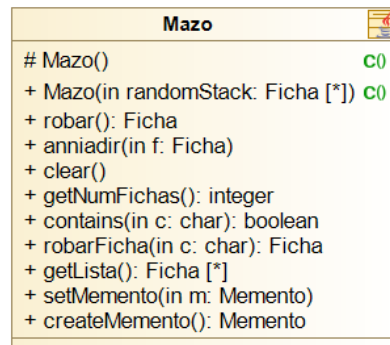
3.5 Saber el número de fichas que quedan por robar

ID:H_5	Usuario:Jugador
Nombre historia:Saber el número de fichas que quedan por robar.	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 1	Iteración asignada: 1
Programador responsable: María Cristina Alameda Salas	
Descripción: Como jugador quiero saber cuántas fichas quedan por robar para poder definirme estrategias y hacer mejores jugadas.	

Explicación

Esta historia de usuario correspondía a una funcionalidad muy pequeña. Comprendía únicamente añadir un método al Mazo que devolviera la longitud de su lista de fichas.

Diagramas



- Diagrama de clase

3.6 Rellenar mano del jugador

ID:H_6	Usuario:Jugador
Nombre historia:Rellenar mano.	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 2	Iteración asignada: 1
Programador responsable: María Cristina Alameda Salas y Tania Romero Segura	
Descripción: Como jugador quiero que mi mano se rellene con fichas nuevas del mazo.	

Explicación

Esta historia de usuario consistía en una vez realizadas una serie de acciones por parte de un jugador, podría verse disminuido el número de fichas de su mano. Así, el número de fichas de esta debería volver a ser 7 para que en otros turnos pudiera continuar sus jugadas.

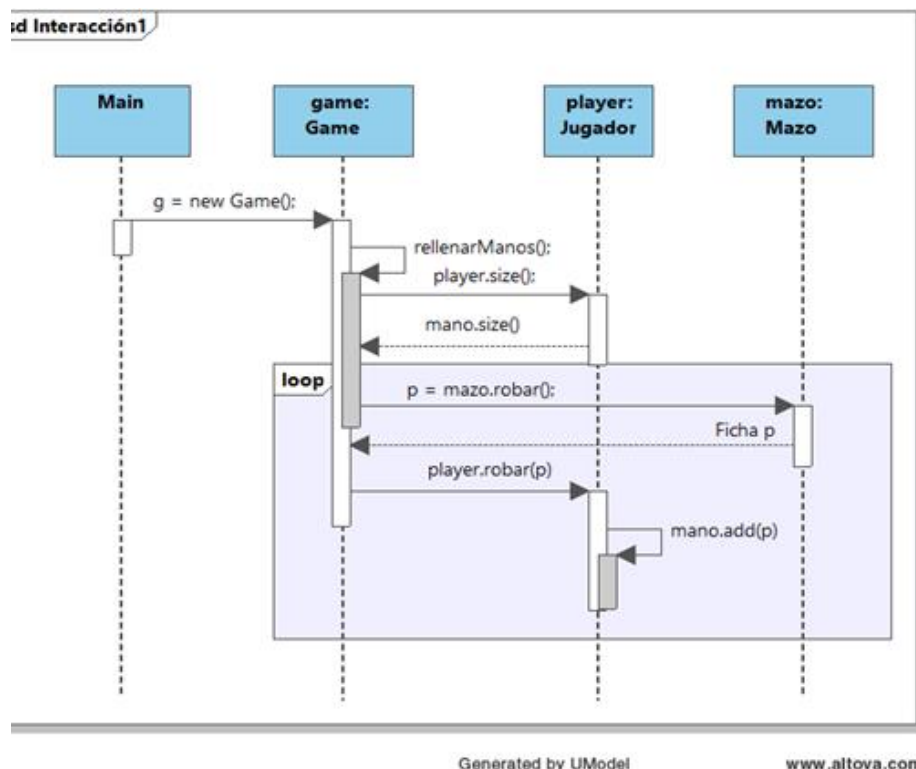
En un principio, la clase Game era quien se encargaba de rellenar tras cada turno en su método rellenarManos la mano del jugador que acababa de terminar el turno. Mientras consultaba el número de fichas de la mano del jugador, si era menor al número máximo de fichas establecido, extraía una ficha del Mazo y se la daba al jugador a través del método -robar(Ficha)-. Este jugador, lo añadía a su mano (atributo que consiste en una lista de fichas).

Entonces, tras un feedback de un sprint, se nos avisó de que el game estaba demasiado cargado y que almacenaba demasiada funcionalidad distinta que no debería tener.

Así, tras la refactorización, la responsabilidad de rellenar la mano tras terminar un turno recayó en el Administrador de turnos. Ahora es esta clase la encargada de rellenar las manos y en su metodo del mismo nombre, llamar a robar de todos los integrantes.

Diagramas

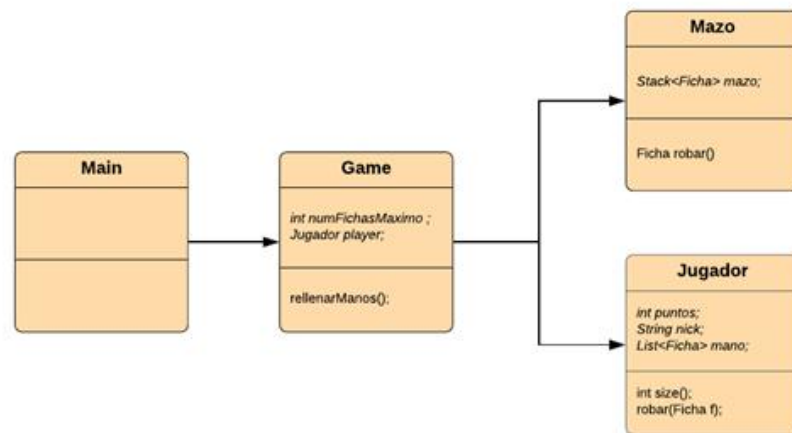
- Antiguos
 - Diagrama de secuencia



[click para ampliar la imagen](#)

En este diagrama se puede apreciar el proceso de rellenar manos del jugador referido a la primera representación de la funcionalidad. Era el propio constructor del Game, el que rellenaba las manos de los jugadores consultando del jugador el número de fichas que faltaría por robar.

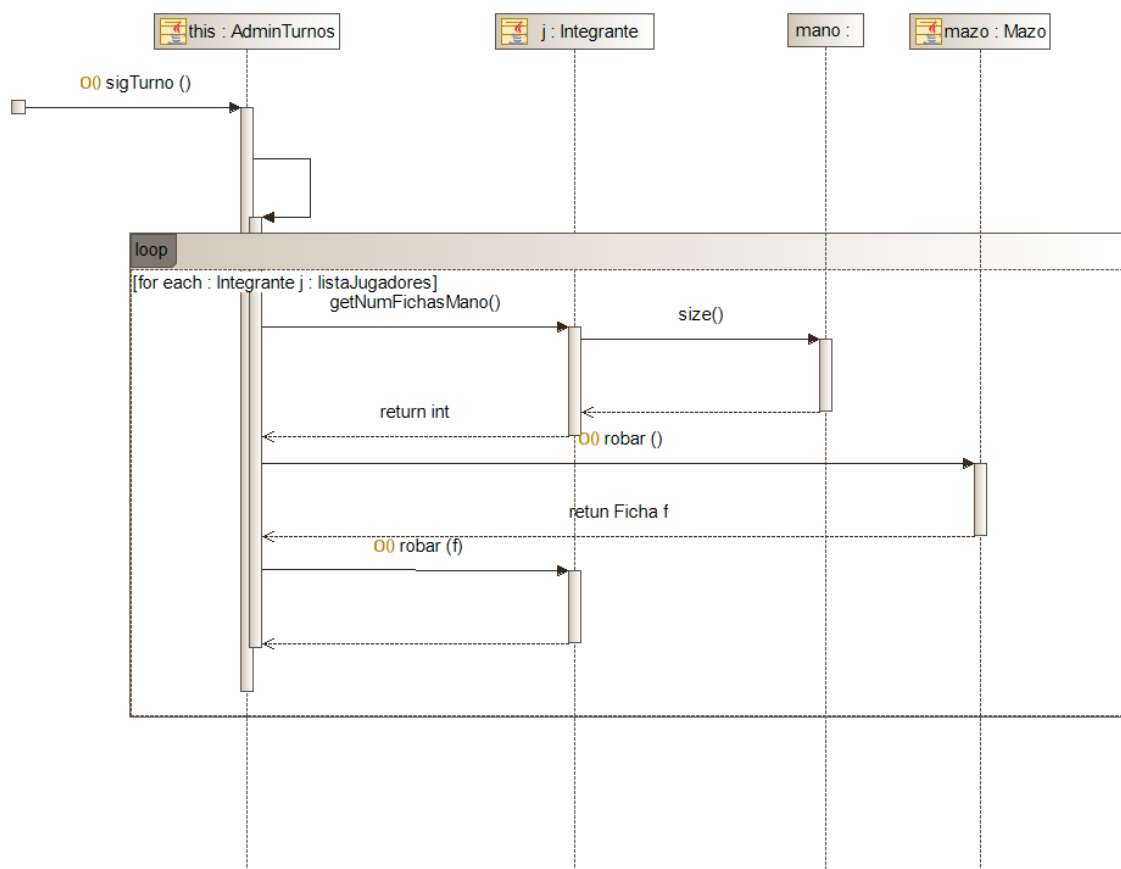
- Diagrama de clases



En este diagrama de clases se puede observar la implicación del Game en rellenar las manos de los jugadores.

- Nuevos

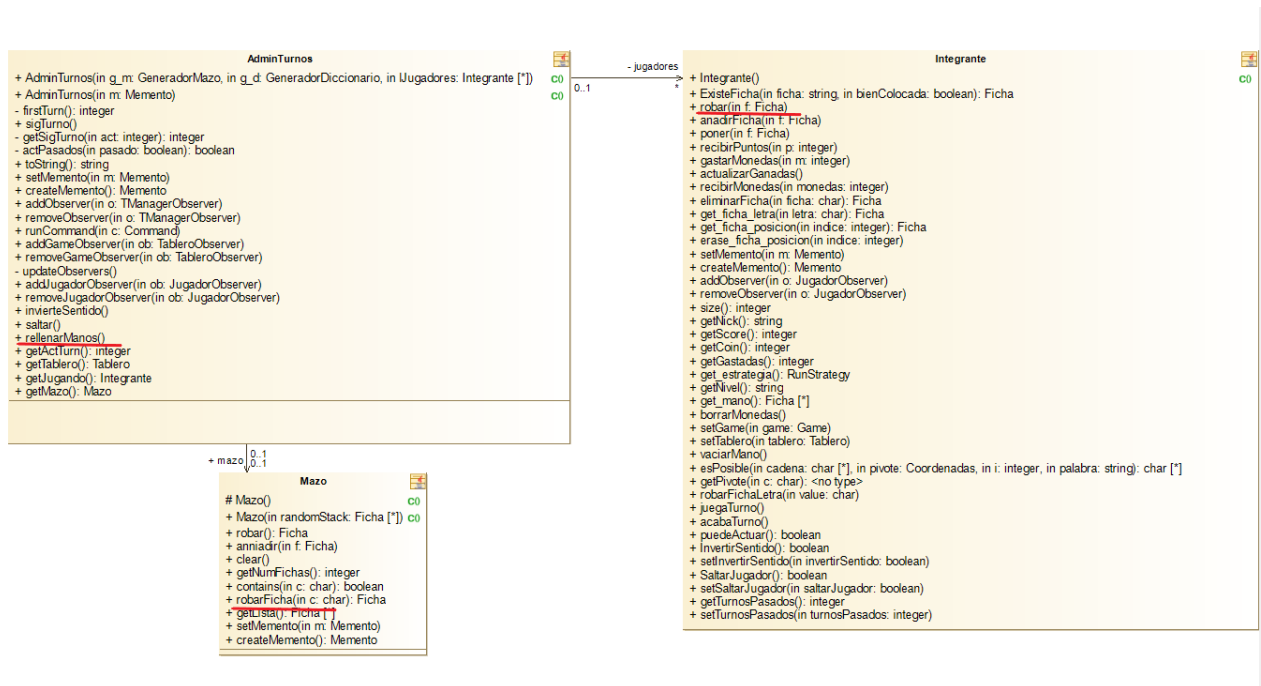
- Diagrama de secuencia



[Click para ampliar la imagen](#)

Se puede apreciar el cambio de responsabilidad de rellenarMano desde el Game al Administrador de turnos, además, ahora para cada uno de los jugadores se rellena la mano, facilitando en multijugador en red que todos los jugadores del juego vean su mano rellena al principio de este.

○ Diagrama de clases



[Click par ampliar la imagen](#)

En este diagrama de clases se puede observar la implicación del Administrador de turno para rellenar las manos de los jugadores.

3.8 Saber mi conjunto de fichas actual

ID: H_8	Usuario: Jugador
Nombre historia: Saber mi conjunto de fichas actual	
Prioridad: Alta	Estado: Completado
Puntos estimados: 2	Iteración asignada: 2

Programador responsable: Jorge Rodríguez
Descripción: Como jugador quiero saber mi conjunto de fichas actual para poder tomar decisiones sobre mi jugada respecto a las fichas de las que dispongo.

Descripción:

Como jugador quiero saber mi conjunto de fichas actual para poder tomar decisiones sobre mi jugada respecto a las fichas de las que dispongo.

Explicación

Inicialmente, el conjunto de fichas de un jugador, es decir, su atril, se guardaba en la clase *Jugador.java* como una lista de fichas. **Posteriormente**, con la inclusión de las máquinas manejadas por IA se incluyó una clase abstracta *Integrante.java* de la cual extendían las clases *Jugador.java* y *Maquina.java*. Actualmente, es la clase *Integrante.java* la que contiene el atril del jugador en el turno actual (puede ser un jugador o una máquina). En lo que concierne a esta historia de usuario, **solo nos interesa mostrar el atril del jugador humano, por lo que nos centraremos en la interacción con la clase *Jugador.java*.**

Actualmente, esta historia de usuario ha evolucionado a la historia “**Poder ver la mano**” para adaptarse a la **introducción de la GUI**.

Diagramas

Respecto a los diagramas, el miembro del equipo encargado de esta historia de usuario no los realizó y por tanto **tuvimos que realizar los diagramas posteriormente, con lo que no se conserva una versión antigua**. De todas formas, al igual que con muchas otras historias de usuario, dichos diagramas se verían sustituidos por los diagramas correspondientes a [“Poder ver la mano”](#).

3.9 Saber cuántos puntos vale cada ficha

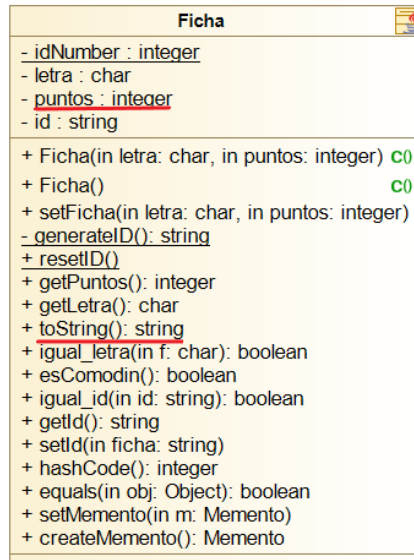
ID: H_9	Usuario: Jugador		
Nombre historia: Saber cuántos puntos vale cada ficha			
Prioridad: Alta		Estado: Completado	
Puntos estimados: 1		Iteración asignada: 2	
Programador responsable: Mª Cristina Alameda			
Descripción: Como jugador quiero saber cuántos puntos vale cada ficha de las que dispongo para poder estimar los puntos que gano con una palabra.			

Explicación

Para conocer los puntos de cada ficha simplemente se añadió el atributo -int puntos- a la clase “Ficha”. También se sobrescribe el método toString de manera que mostrara la ficha y la puntuación de esta, de la forma letra[puntuación] y que fuera más fácil de representar en el tablero y de mostrarse en la mano. No ha sufrido evolución ninguna en a lo largo del proyecto.

Diagramas

- Diagrama de clases



Únicamente vamos a mostrar el diagrama de clases para ver la ampliación de esta con el método toString y el atributo puntos.

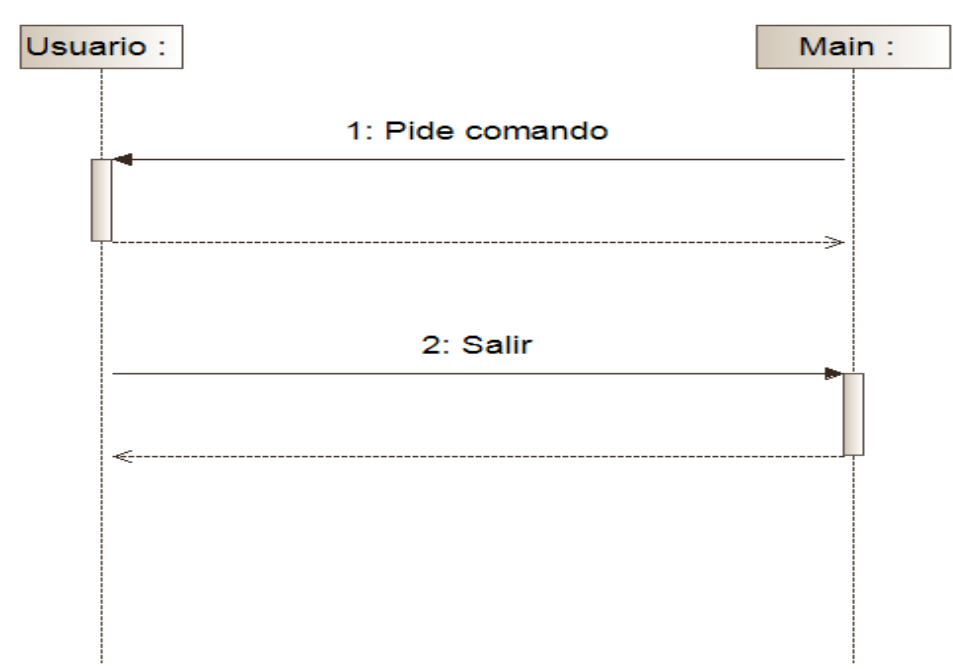
3.10 Salir de la partida

ID:H_5	Usuario:Jugador
Nombre historia:Salir de la partida	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 1	Iteración asignada: 1
Programador responsable:	
Descripción: Como jugador quiero salir de la partida para acabar la partida.	

Descripción detallada:

Por consola se le pedía al jugador que acción quería tomar o bien empezar una partida o bien salir de la aplicación, más tarde esta opción fue implementada como un comando dentro de la aplicación. Básicamente es un booleano que si se cumplía se acababa el juego.

Diagramas:



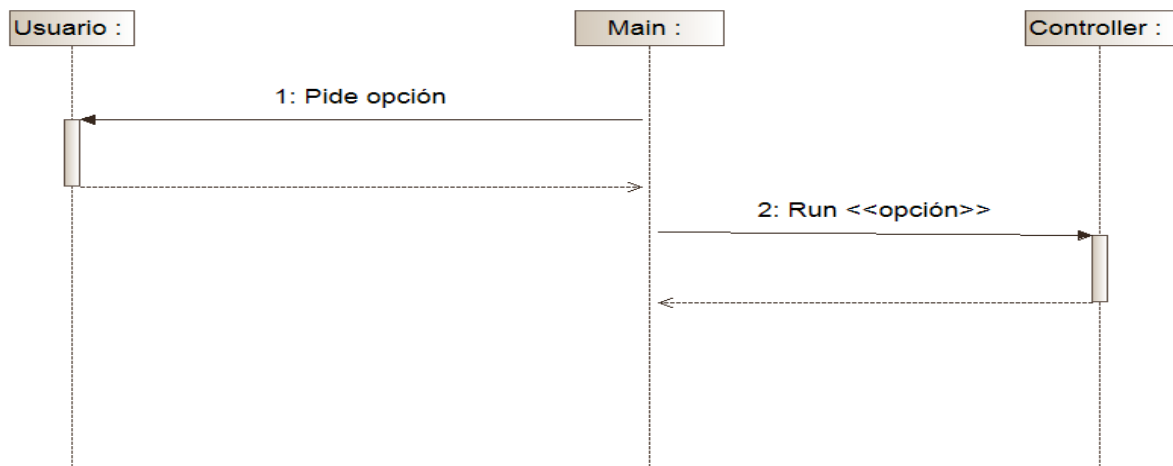
3.11 Pedir New Game o Load Game

ID:H_27	Usuario:Jugador
Nombre historia:Pedir si quieres New Game / Load Game	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 3	Iteración asignada: 3
Programador responsable: David Cruza Sesmero	
Descripción: Como jugador puedo elegir cargar una partida a medias o bien crear una nueva la cual me mostrará un menú secundario en el que tendré como opciones añadir jugadores, eliminar jugadores, mostrar información del juego y comenzar la partida.	

Descripción detallada:

Se pedía desde el main por consola que se eligiese entre comenzar una nueva partida o una ya a medias, si se elegía una nueva partida este menú te llevaba a otro submenú que te permitía empezar la partida o añadir y quitar jugadores.

Diagramas:



3.12 Verificar una palabra

ID:H_12	Usuario:Jugador
Nombre historia:Verificar una palabra	
Prioridad: Alta	Estado: Finalizado
Puntos estimados:	Iteración asignada: 2
Programador responsable: Tania Romero Segura y María Cristina Alameda Salas	
Descripción: Como jugador quiero verificar una palabra para aumentar mi puntuación	

Explicación

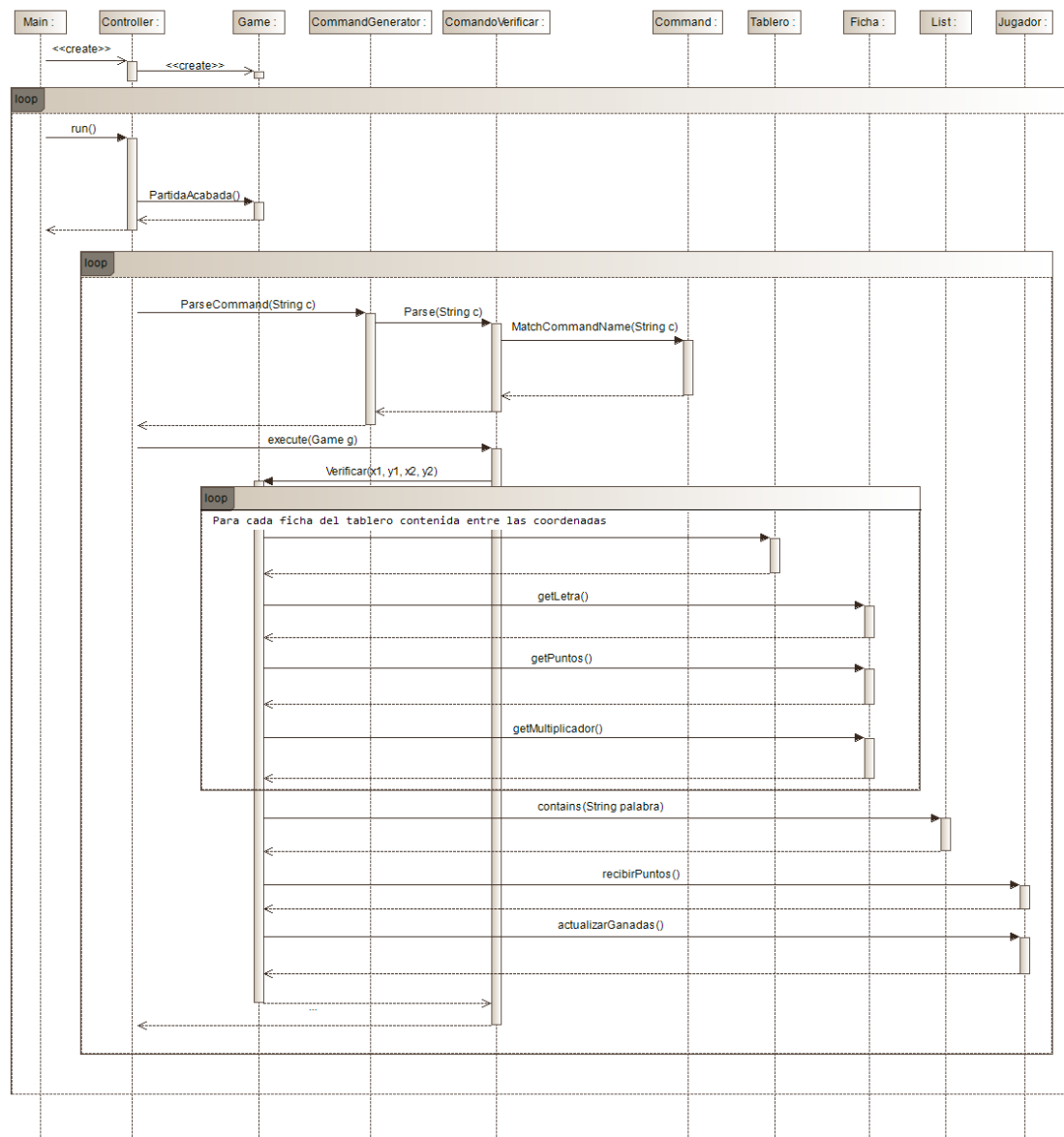
Durante el segundo sprint la distribución de la carga de trabajo no fue del todo adecuada así que con el tiempo que sobraba dos desarrolladores implementaron esta funcionalidad.

El objetivo era que el jugador dispusiera de un comando que permitiera comprobar si una palabra estaba en el diccionario. La parte de obtener puntos si es correcta se implementó para complementar otra Historia de Usuario (Ganar Puntos).

Para cumplir con este objetivo se creó la clase ComandoVerificar que heredaba de la clase Command. Para la ejecución de este comando se le pedía al usuario que diera la posición en coordenadas de la primera letra de la palabra y la posición en coordenadas de la última letra de la palabra. El método Verificar del game al que llama el comando cuando se ejecuta recorre las posiciones del tablero contenidas entre las coordenadas, coge la letra y la añade a la palabra. Al final, cuando tiene la palabra completa, si esa palabra está en el diccionario devuelve true dado que se ha confirmado que la palabra colocada es correcta.

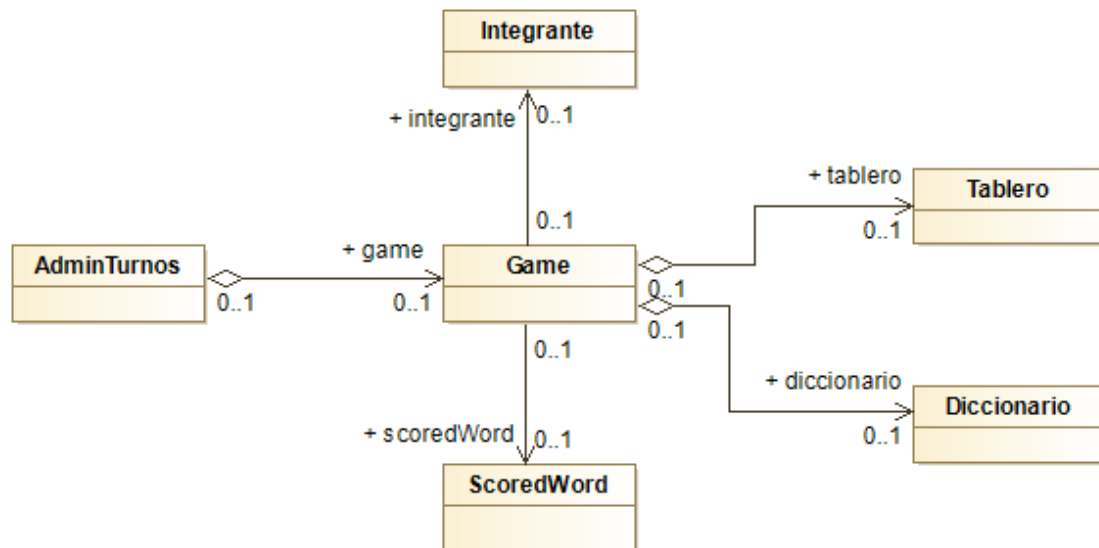
Diagramas

- Diagrama de secuencia



[Click para ampliar](#)

- Diagrama de clases



3.13 Ganar puntos

ID:H_23	Usuario:Jugador
Nombre historia:Obtener puntos por mis jugadas	
Prioridad: Media	Estado: Finalizado
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: Tania Romero Segura	
Descripción: Como jugador quiero poder ganar puntos al poner una palabra en el tablero.	

Explicación

Llegados a este punto se hizo necesario hacer que el jugador pudiera ganar puntos. Si no, no se podría seguir avanzando en el juego ya que el objetivo de todo juego es ganar y si no ganas puntos en este juego no se puede ganar. Así que para ello se implementó un sistema que obtenía la puntuación de una palabra. Al principio se hacía directamente durante la verificación de un apalabra, después se creó un método a parte para poder utilizarla durante el proceso de verificación todas las veces que fuera necesario de forma clara.

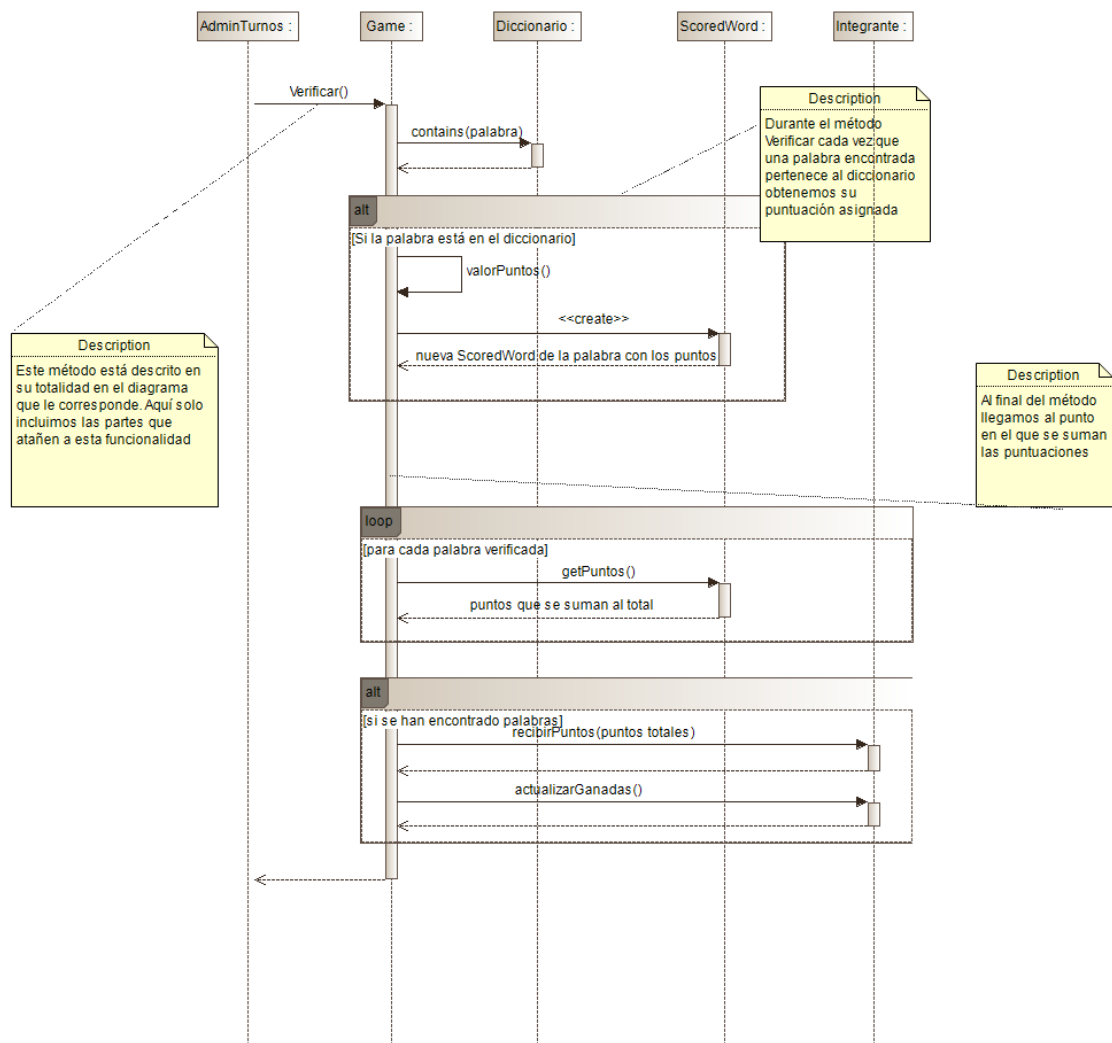
El sistema que obtiene la puntuación que el corresponde comprueba letra por letra qué casilla del tablero ocupa y modifica la puntuación de las siguientes maneras:

- Si es una casilla normal suma a la puntuación de la letra a la puntuación total
- Si es una casilla de doble de letra se suma el doble de la puntuación de la letra a la puntuación total
- Si es una casilla de triple de letra se suma el triple de la puntuación de la letra a la puntuación total
- Si es una casilla de doble de palabra al multiplicador por palabra que se lleva en una variable (inicializado a uno) se le multiplica por 2.
- Si es una casilla de triple de palabra al multiplicador por palabra que se lleva en una variable (inicializado a uno) se le multiplica por 3.

Al finalizar este proceso, se multiplica el valor total de la palabra por el multiplicador de palabra y esta será la puntuación que se le sumará al jugador.

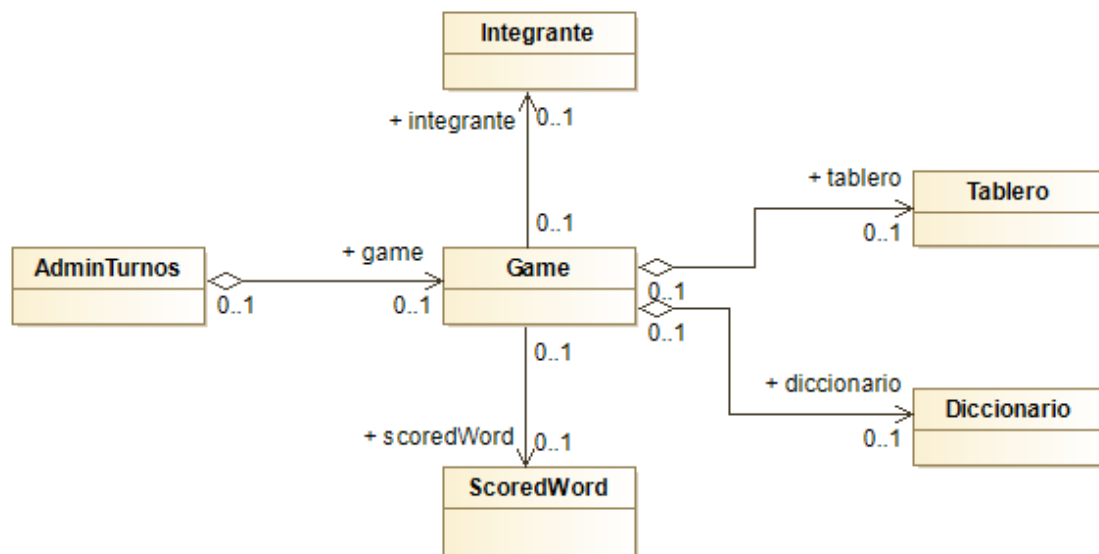
Diagramas

- Diagrama de secuencia



[Click para ampliar](#)

- Diagrama de clases



3.14 Conocer turnos de la partida

ID:H_14	Usuario:Jugador		
Nombre historia:Conocer turnos de la partida			
Prioridad: Alta		Estado: Terminado	
Puntos estimados: 2		Iteración asignada: 2	
Programador responsable: Alejandro Rivera León			
Descripción: Como jugador quiero conocer turnos de la partida para saber cuál es el orden de jugada de los jugadores.			

Explicación

Durante el segundo sprint se introdujeron las clases **AdminTurnos** y **Turno** para **encapsular todo lo relacionado con la gestión de un turno**.

Se decidió darle la **responsabilidad de mostrar el orden de los turnos** a la clase **AdminTurnos** ya que este tenía la lista de jugadores como atributo y en general porque era la clase encargada de crear los turnos cosa que facilitaba mucho el proceso.

El funcionamiento es el siguiente:

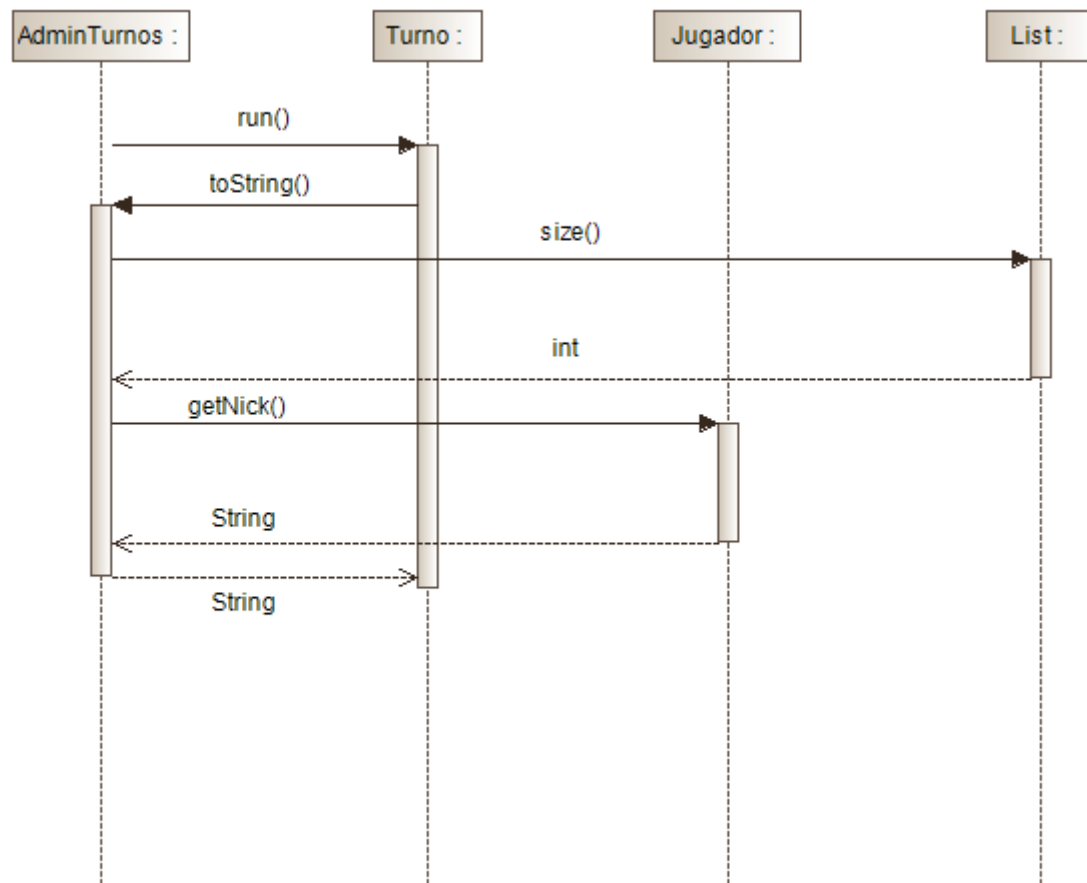
AdminTurnos crea un Turno, dicho turno **recibe el jugador** al que le toca jugar **y una instancia de AdminTurnos**.

Se realiza el turno correctamente y posteriormente, con la instancia pasada a turnos **se llama al método toString () de AdminTurnos**.

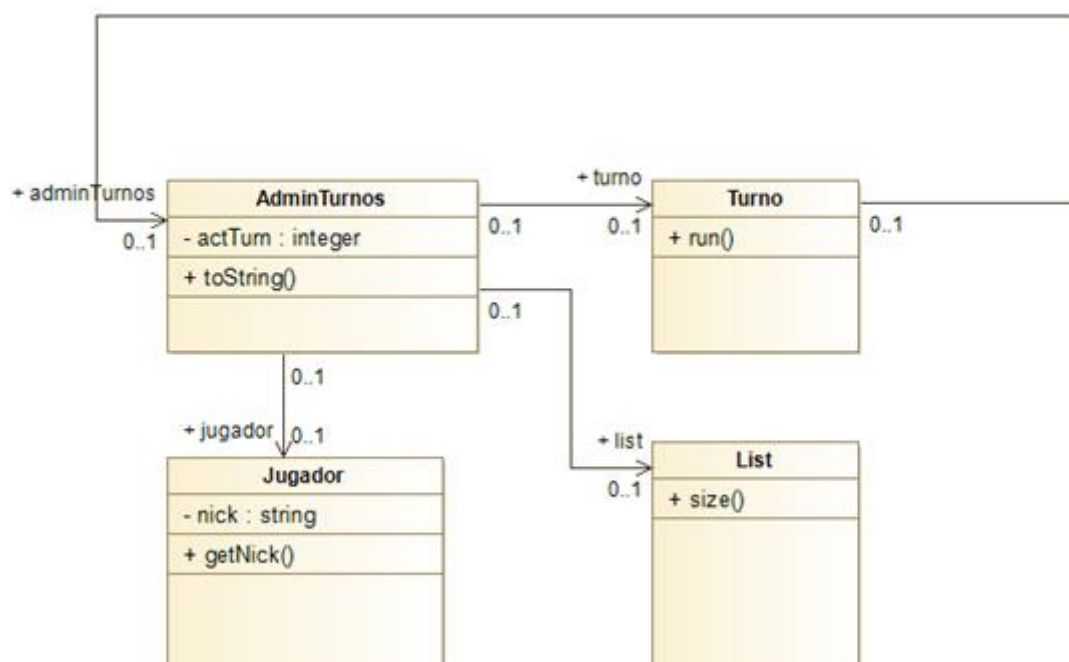
toString () **recorre la lista de jugadores de forma circular** permitiendo así que en el caso de que fuese el turno del último jugador de la lista, el siguiente turno sea el del primero de la lista.

Como se puede apreciar en el diagrama y por la explicación, **esto generaba una gran dependencia entre AdminTurnos y turnos** y realmente qué turnos recibiese un atributo AdminTurnos no era del todo correcto ya que una clase como Turnos no debería conocer la existencia de la clase AdminTurnos.

Diagramas



[Click para verlo más grande.](#)



[Click para verlo más grande.](#)

3.15 Pasar de turno

ID:H_15	Usuario:Jugador
Nombre historia:Pasar de turno	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: Gema Blanco Núñez	
Descripción: Como jugador quiero pasar de turno para terminar mi jugada.	

Explicación

Esta historia de usuario tiene como principal **objetivo** permitir que un jugador ceda el turno al siguiente.

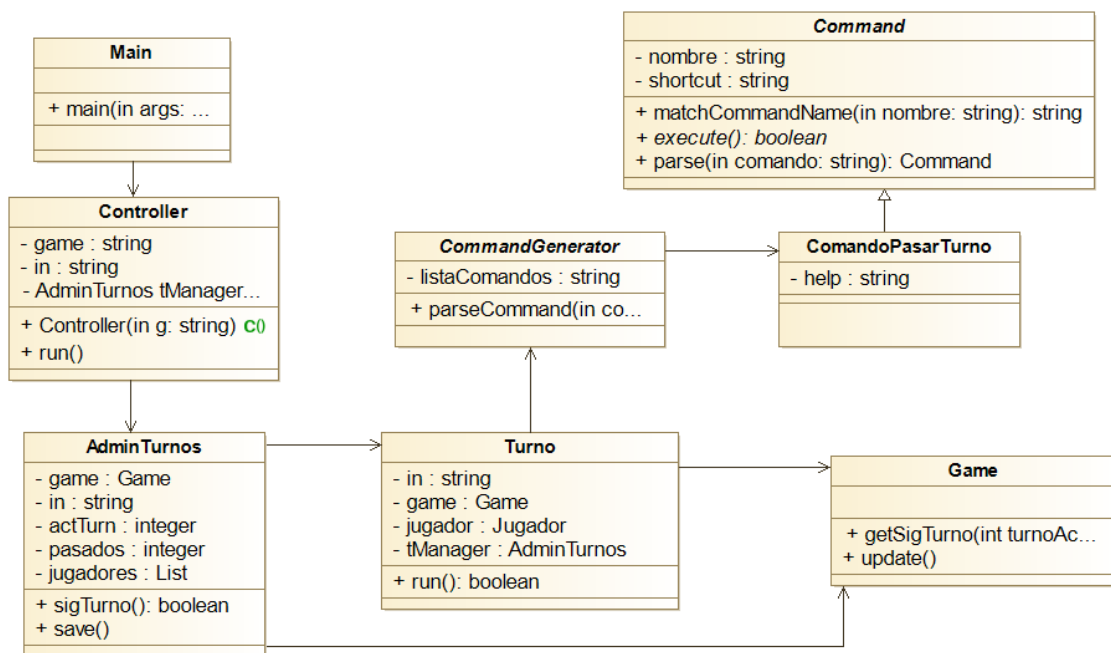
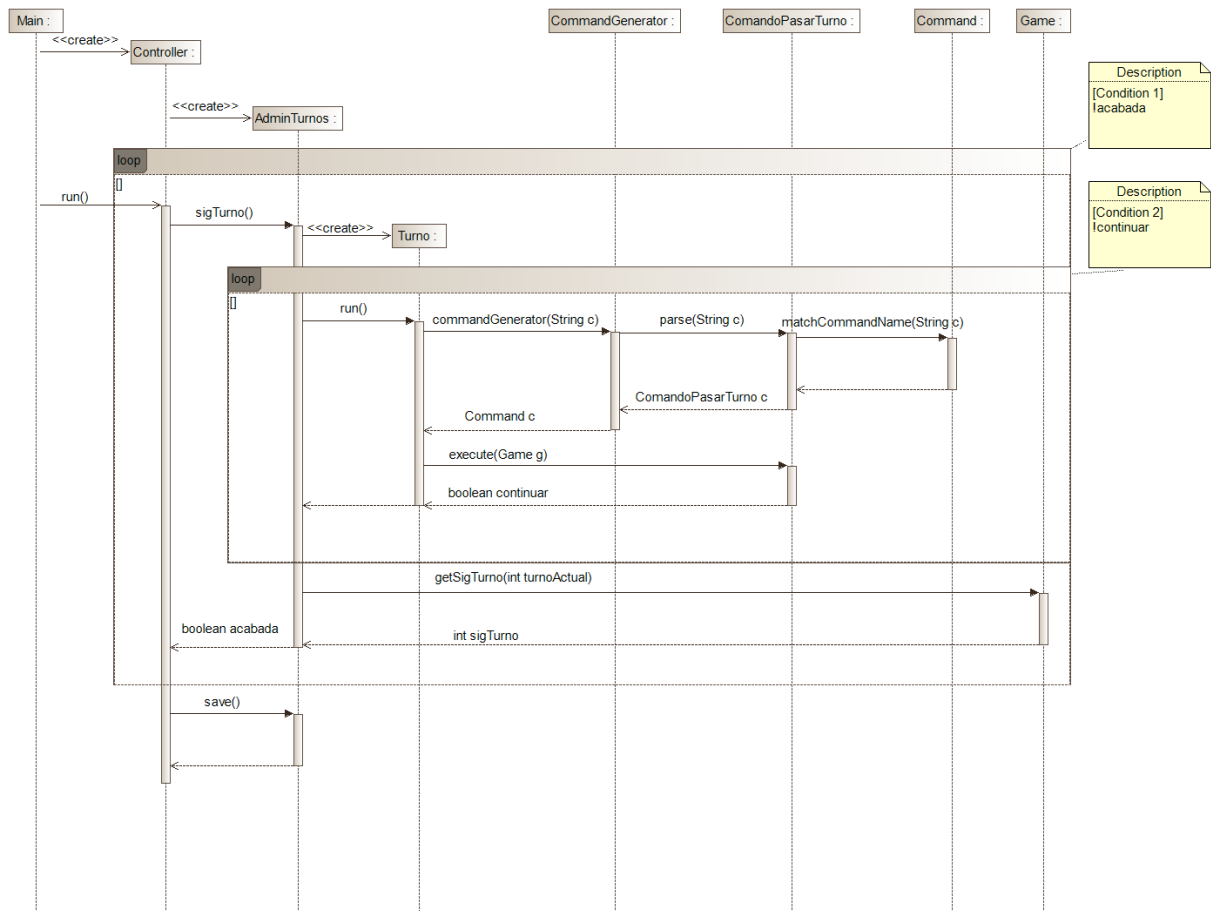
Como se explica en la H_14, se crearon las clases **AdminTurnos** y **Turno** con el objetivo de reducir dependencias y encapsular la información relativa a cada turno.

Anteriormente, la clase Turno era la encargada de llamar al método *execute()* del **ComandoPasarTurno** y este devolvía un booleano "false" que hacía que el bucle principal del juego de la clase AdminTurnos se detuviera y pasara el turno al siguiente jugador. Esto es lo que se refleja en los diagramas antiguos.

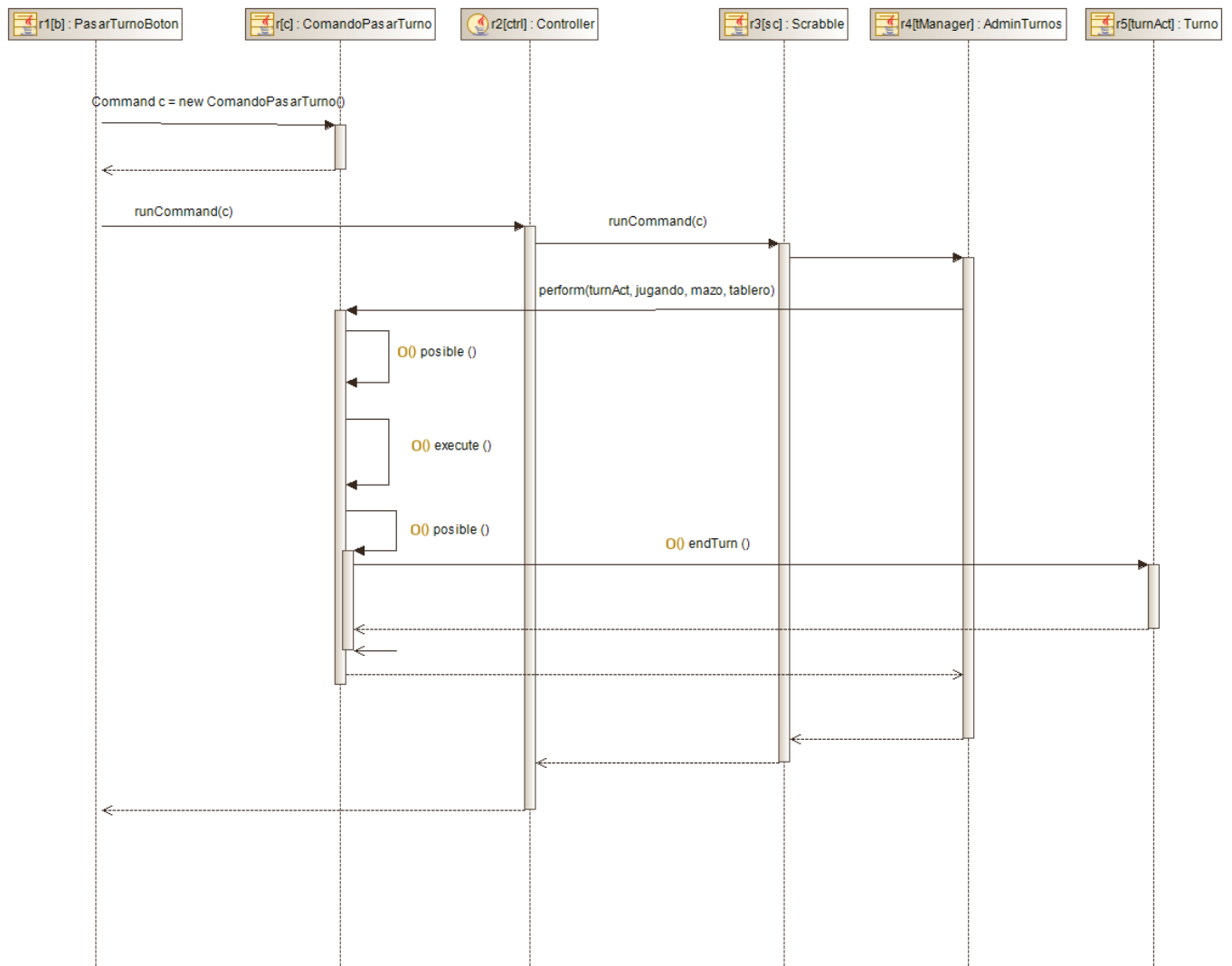
Actualmente, al implementar la GUI y hacer refactorizaciones en el modelo, cada vez que el usuario genera un evento para pasar de turno se continúa llamando al *execute()* del ComandoPasarTurno, pero ahora este no se encarga de finalizar el turno. Es en el método *addCambios()*, llamado tras el *execute()*, cuando se le indica al Turno que el jugador decide pasar el turno.

Diagramas

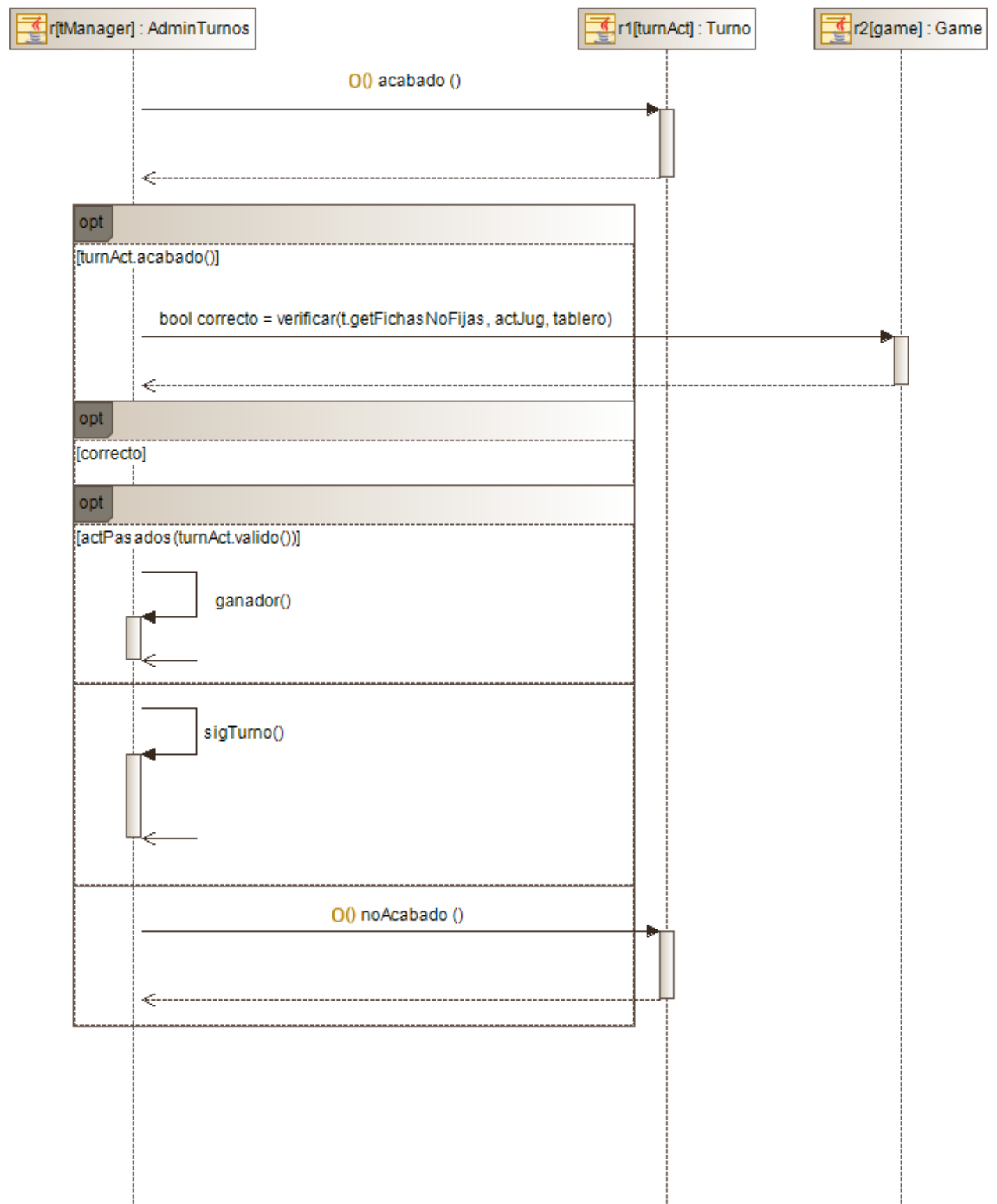
- Antiguos:
 - Diagrama de secuencia

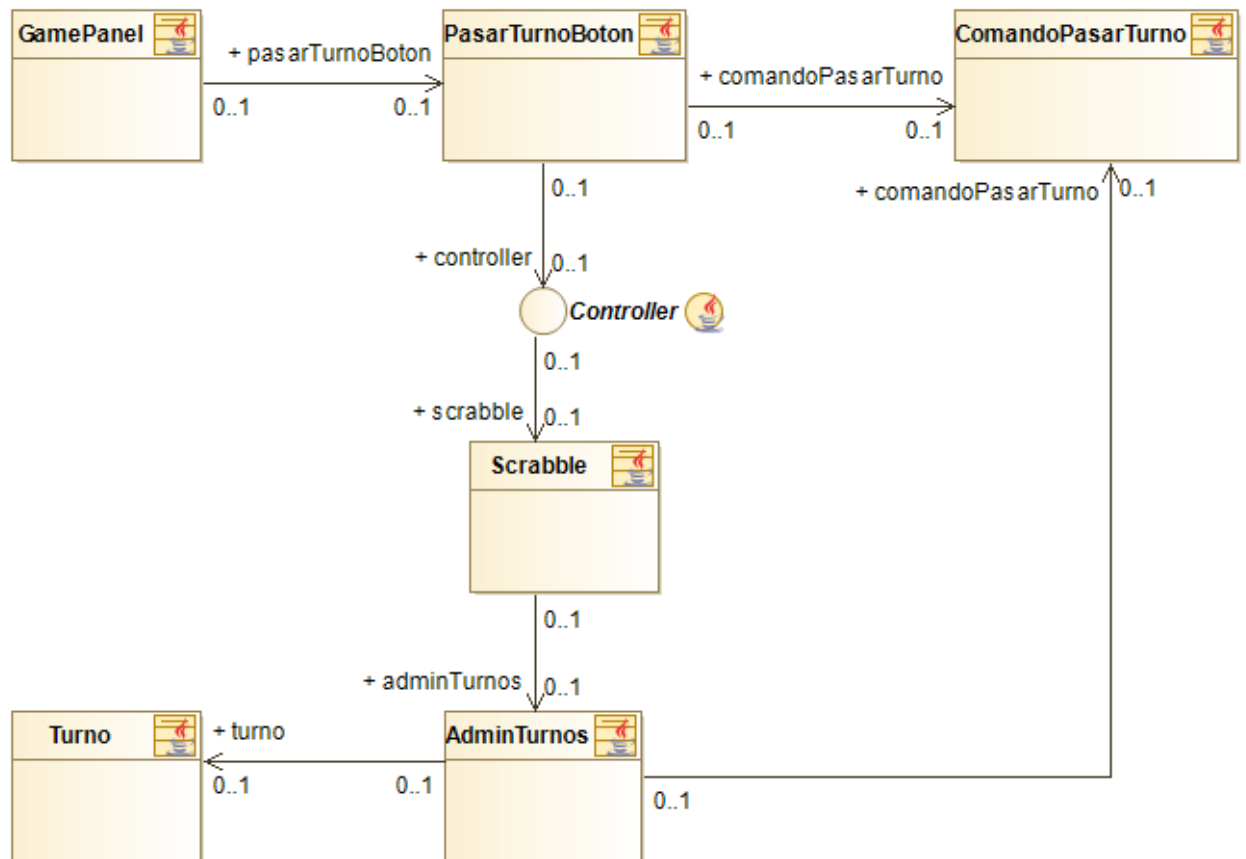


○ Diagrama de clases



- Nuevos:





3.17 Introducir mi nick

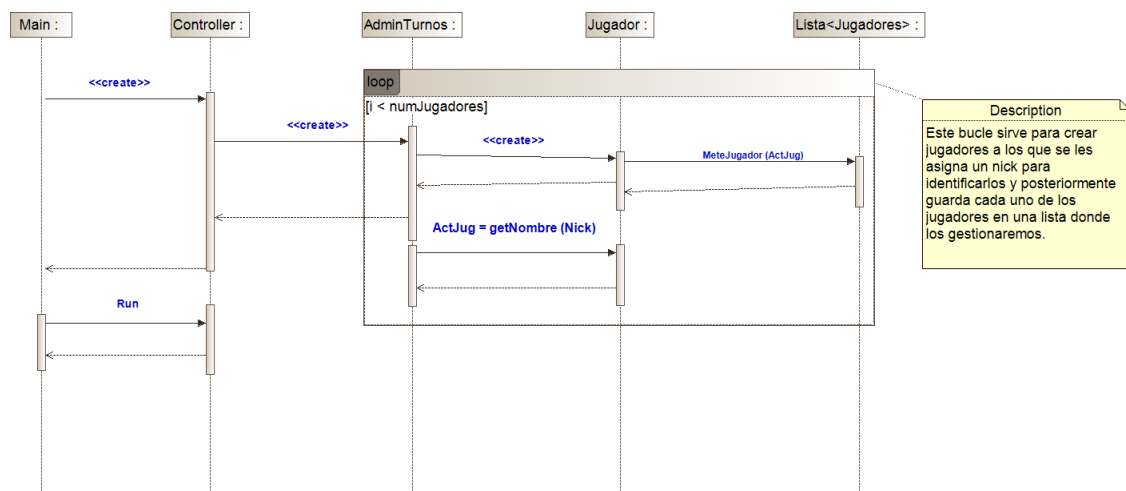
ID:H_13	Usuario:Jugador
Nombre historia:Introducir mi nick	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 1	Iteración asignada: 2
Programador responsable: David Cruza Sesmero	
Descripción: Como jugador quiero introducir mi nick para identificarme en mis partidas.	

Explicación:

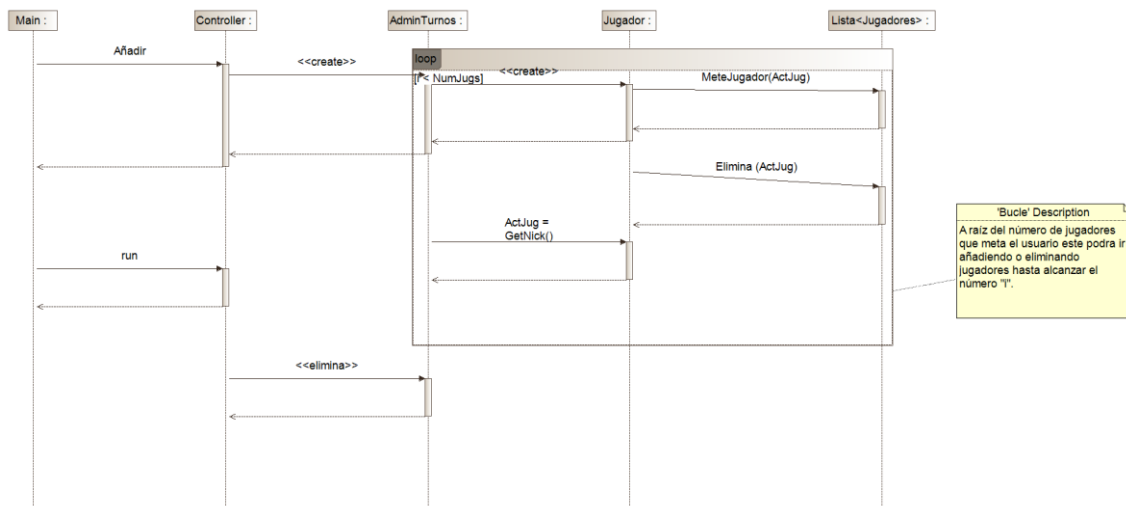
La funcionalidad de introducir un nick por consola se realizaba pidiendo al usuario el número de jugadores que iban a participar desde el menú, una vez se recibía dicho número de pasaba como parámetro a un método del AdminTurnos el cual era InitListaJugs. Este método era un bucle que pedía n veces unos nicks los cuales iban a ser los jugadores.

GUI: En este punto nos encontramos con una tabla en la cual debemos introducir los nicks de los jugadores dando enter para confirmar dicho valor de las celdas. Tras guardar esta tabla se nos mostrará una ventana que nos pide cuántas máquinas meter y el nivel que queremos que tengan.

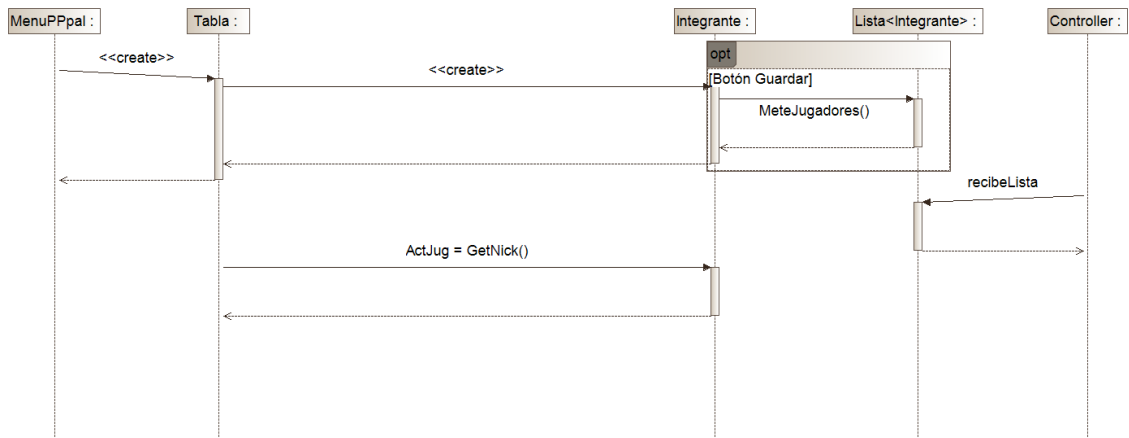
Diagramas antiguos:



En el siguiente diagrama se añadió la opción de eliminar jugadores



Una vez se introdujo la GUI este cambió hasta este punto



3.18 Colocar más de una ficha por turno

ID:H_18	Usuario:Jugador		
Nombre historia:Colocar más de una ficha por turno			
Prioridad: Media		Estado: Terminado	
Puntos estimados: 2		Iteración asignada: 2	
Programador responsable: Alejandro Rivera León			
Descripción: Como jugador quiero poder colocar más de una ficha en el tablero cada turno para formar palabras y acumular puntuación.			

Explicación

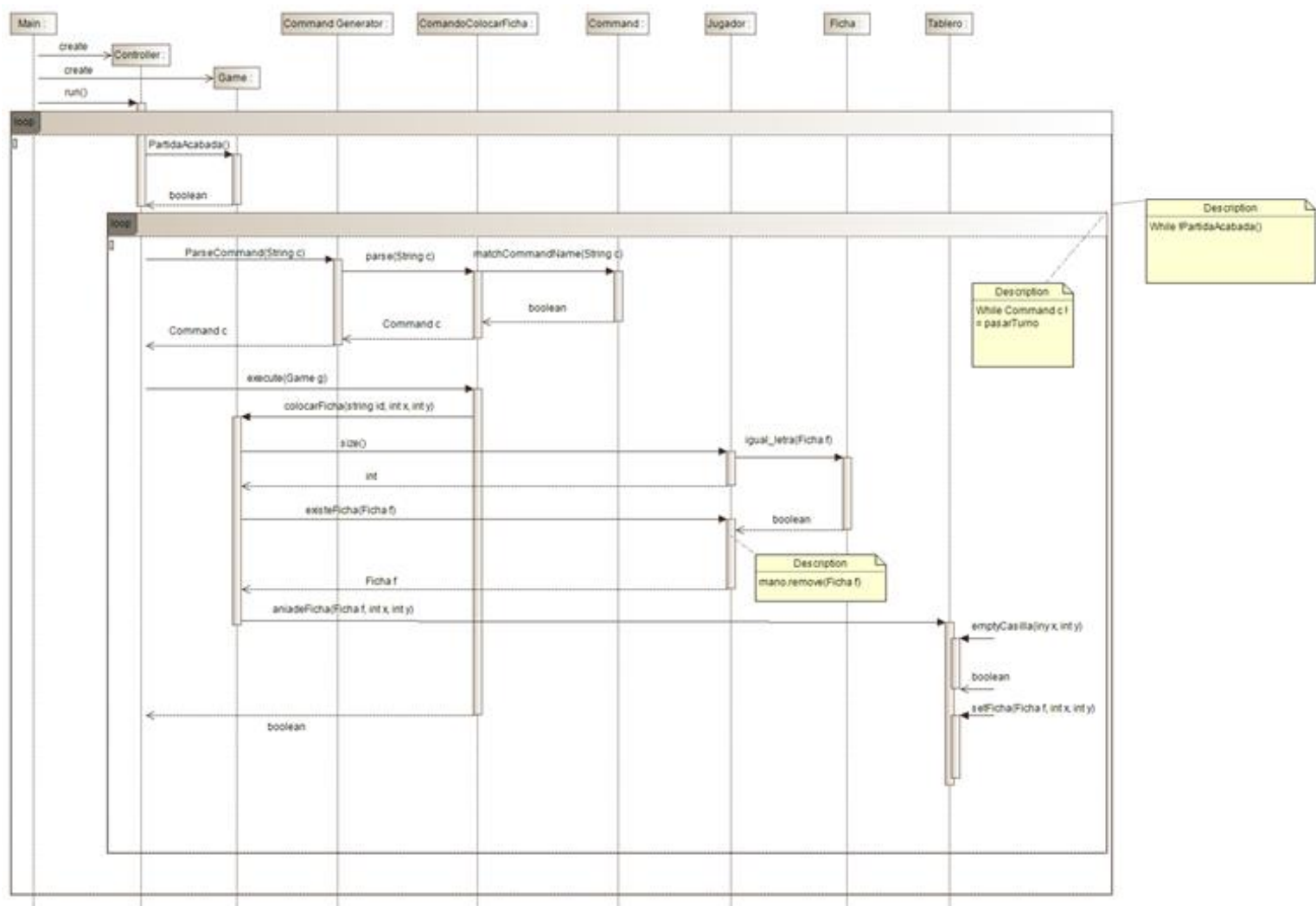
Para diseñar esta funcionalidad **fue de mucha ayuda la implementación de la historia de usuario “Colocar fichas en el tablero”** del sprint anterior ya que realmente **la lógica detrás de la colocación de las fichas se siguió manteniendo igual.**

De cara a esta modificación teníamos que tener en cuenta que **el usuario debía de ser el que decidiese cuando su turno llegaba a su fin** y en resumidas cuentas cuando dejaba de poner fichas.

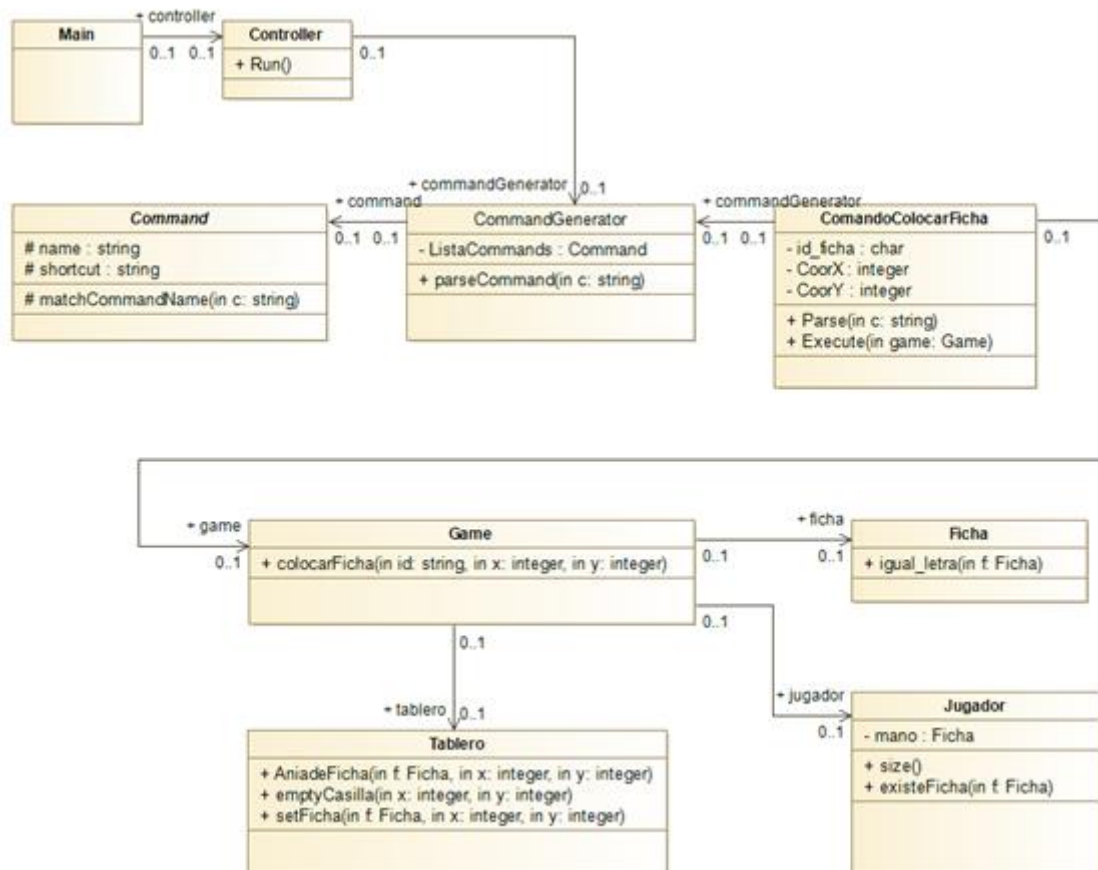
Para ello **se desarrolló un comando pasar turno del cual se profundiza en la siguiente historia de usuario [\(ENLACE\)](#).**

Empleando dicho comando le dimos la capacidad al usuario de que el mismo decidiese cuando dar por terminado su turno, mientras que este no introdujese dicho comando su turno se seguiría ejecutando indefinidamente dándole así la posibilidad de poner varias fichas en un mismo turno (entre otras posibilidades).

Diagramas



[Click para verlo más grande.](#)



[Click para verlo más grande.](#)

3.19 Jugar con otro jugador

ID:H_19	Usuario:Jugador
Nombre historia:Jugar con otro jugador	
Prioridad: Media	Estado: Finalizado
Puntos estimados: 3	Iteración asignada: 2
Programador responsable: Guillermo García Patiño Lenza	
Descripción: Como jugador quiero poder jugar otra partida con otra persona.	

Explicación

Para implementar esta funcionalidad, decidimos crear una nueva clase **AdminTurnos** que fuera encargada de **gestionar la alternancia de jugadores** a la hora de actuar en el juego.

No se ha aplicado ningún patrón de diseño para implementar esta funcionalidad, pero llegado cierto punto en el desarrollo, **se ha tenido que cambiar su implementación** para **adaptar las comunicaciones del modelo** con el resto de elementos del programa (Patrón Fachada, clase Scrabble), y para **adaptar su funcionamiento también a un juego basado en eventos**. Para ver estos cambios, mirar el documento [‘Refactor Modelo’](#).

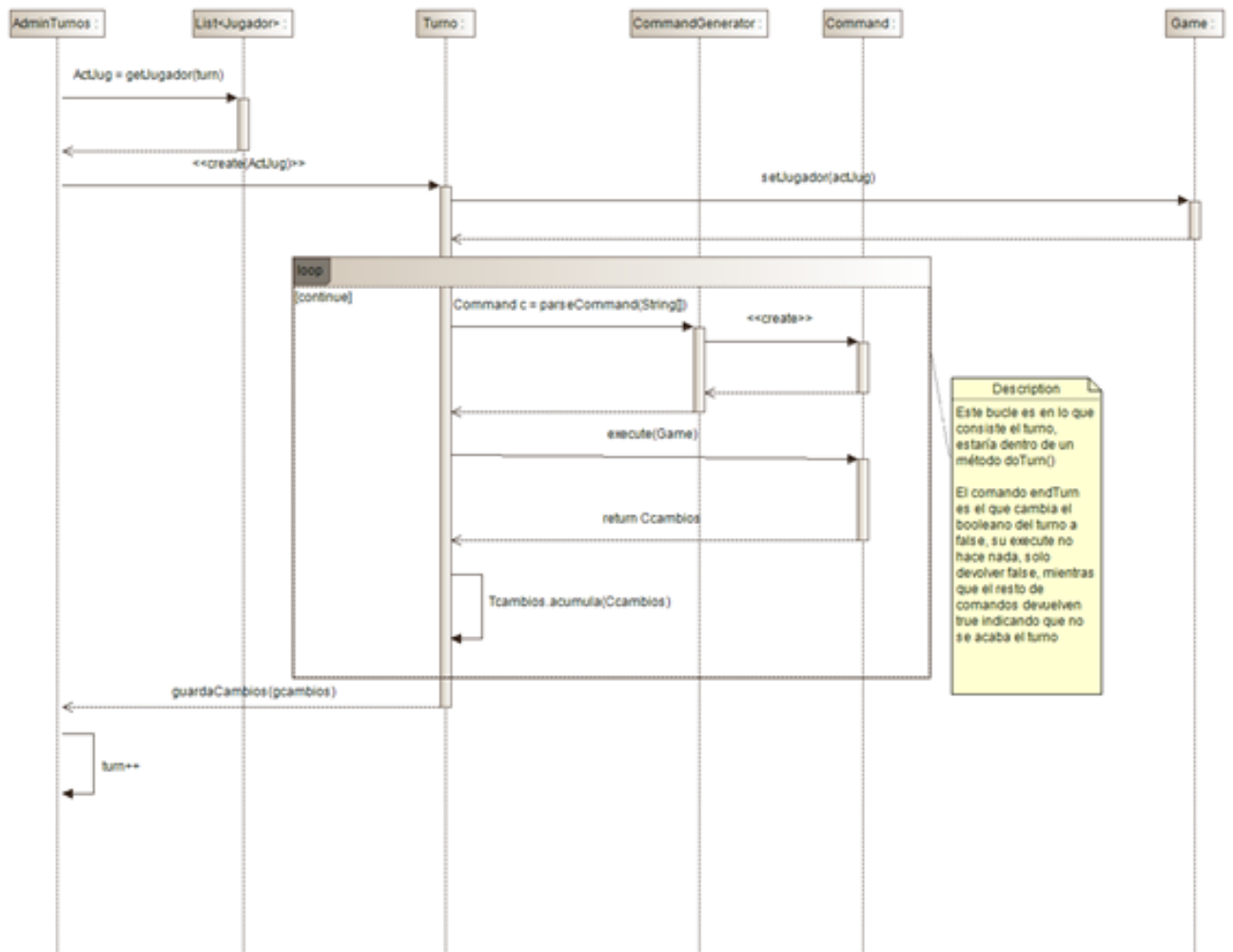
Por otro lado, esta clase tiene una serie de **mejoras importantes** que podrían realizarse en ella:

- **Mover** los atributos **Tablero, Mazo y Game** a la clase **Scrabble**. **No tiene mucho sentido** que una entidad que se dedica a señalar qué jugador puede realizar acciones en la partida en cada momento conozca el tablero del juego, el mazo y el sistema que verifica las palabras.
- Como consecuencia de lo anterior, **mover los métodos que añaden observadores** a las diferentes partes del modelo a la clase scrabble.
- En el constructor de AdminTurnos **se le da a las máquinas las instancias del Tablero y del Game** para que puedan comprobar qué palabras tienen disponibles y si las pueden poner en el tablero. Esto se podría **subir también a la clase Scrabble**. Por otro lado, para construir los jugadores máquina, podrían pasarse el número de máquinas y el

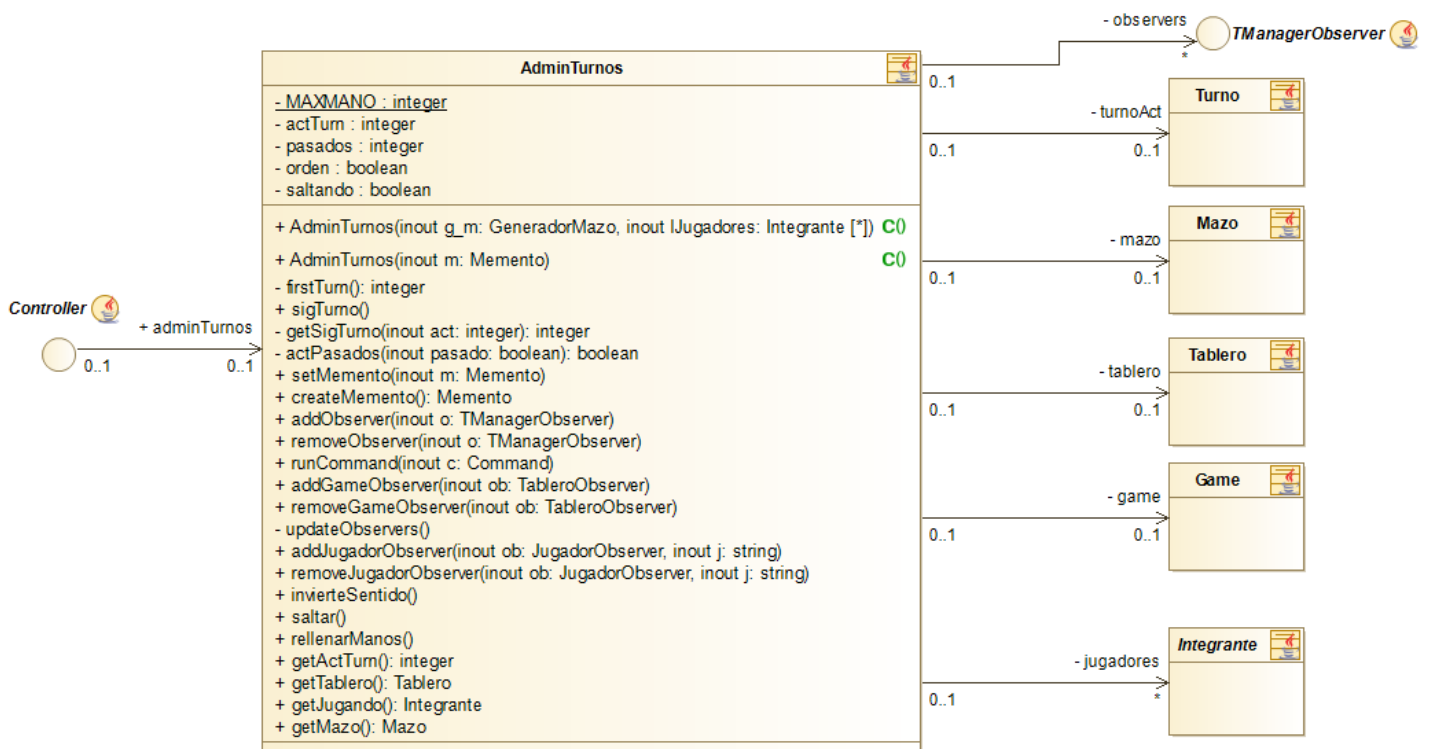
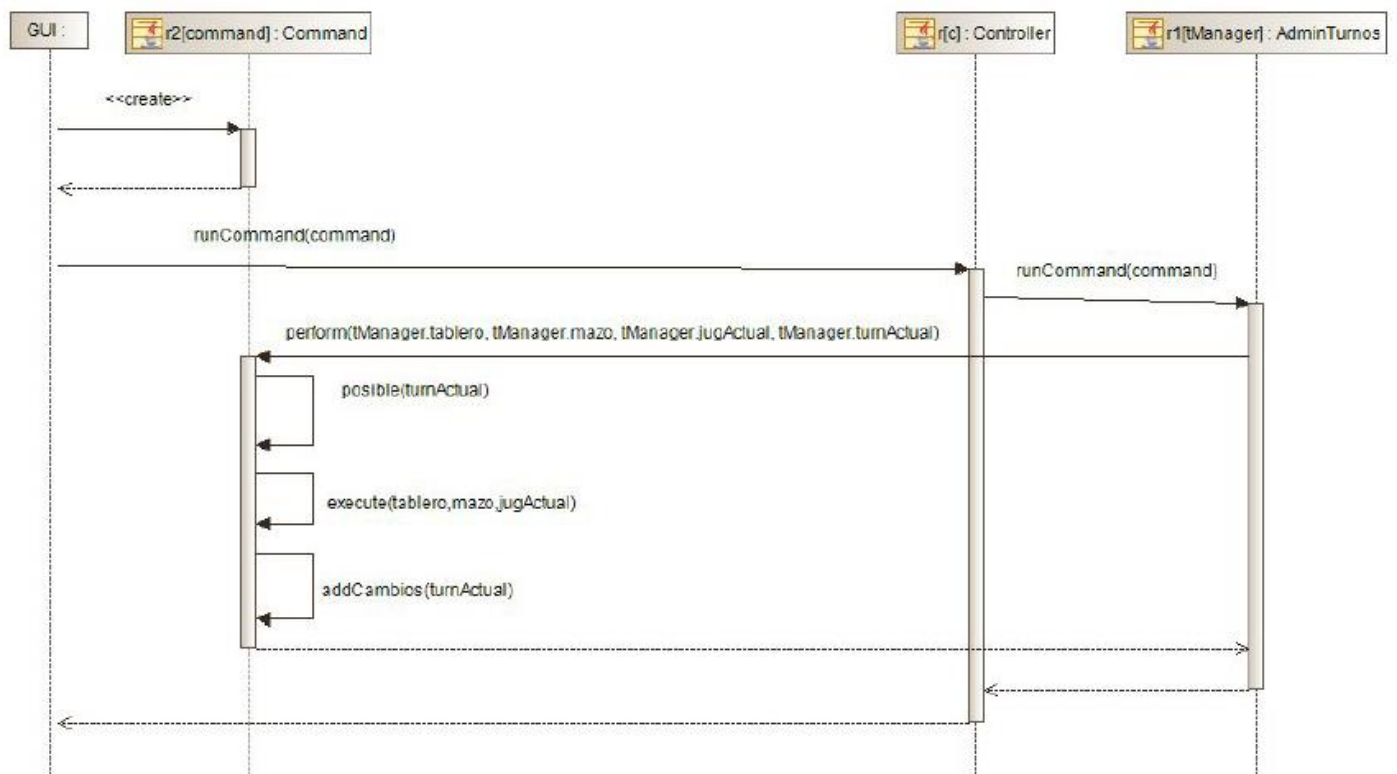
nivel de dificultad y **construirse en el constructor de Scrabble** una vez estuvieran creadas las instancias de Tablero y Game.

Diagramas

- Antiguos: (hacer click en la imagen para verlo más grande)



- Nuevos:(hacer click en la imagen para verlo más grande)



3.20 Saber quién empieza la partida

ID: H_20	Usuario: Jugador
Nombre historia: Saber quién empieza la partida	
Prioridad: Media	Estado: Finalizado
Puntos estimados: 2	Iteración asignada: 2
Programador responsable: Tania Romero Segura	
Descripción: Como jugador quiero poder saber quién tiene el primer turno de la partida.	

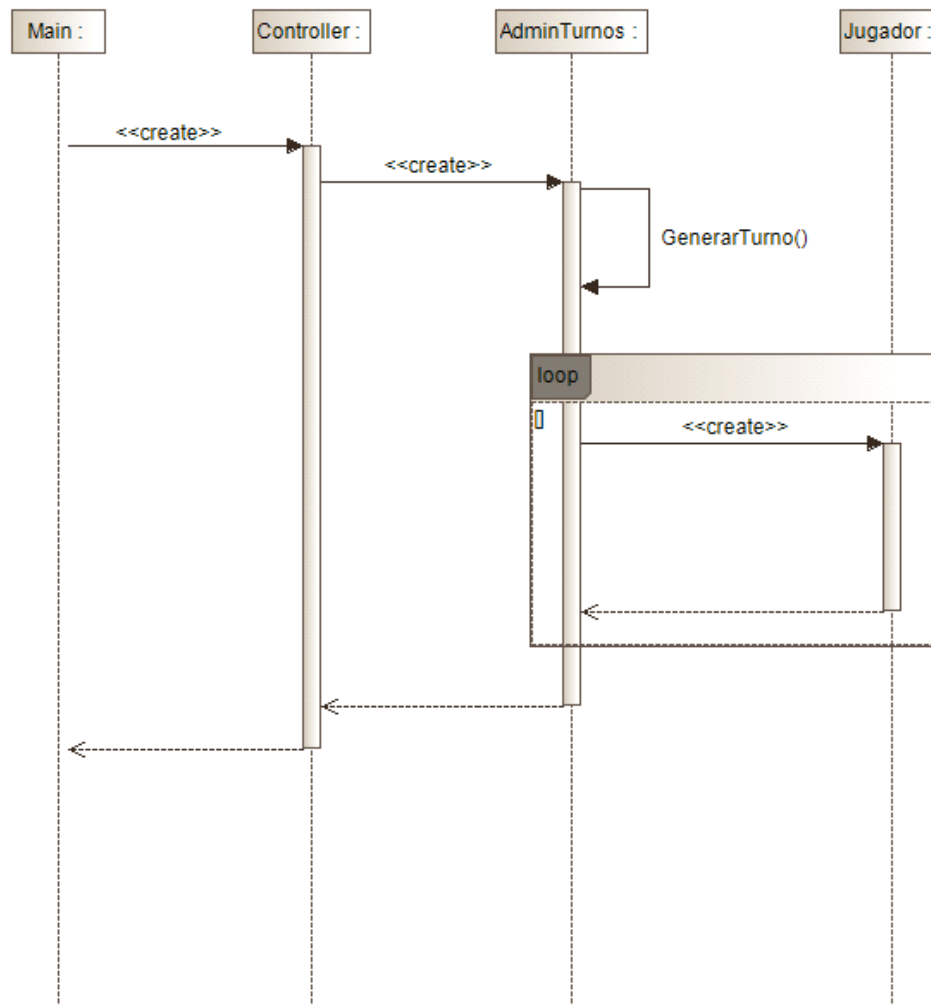
Explicación

Para poder saber quién empieza la partida se creó un método generarTurno en el administrador de Turnos que utilizaba la función random de Math para obtener un número al azar entre 0 y el número de jugadores - 1. De esta manera, este número representaría la posición en la lista de jugadores del jugador que iría primero.

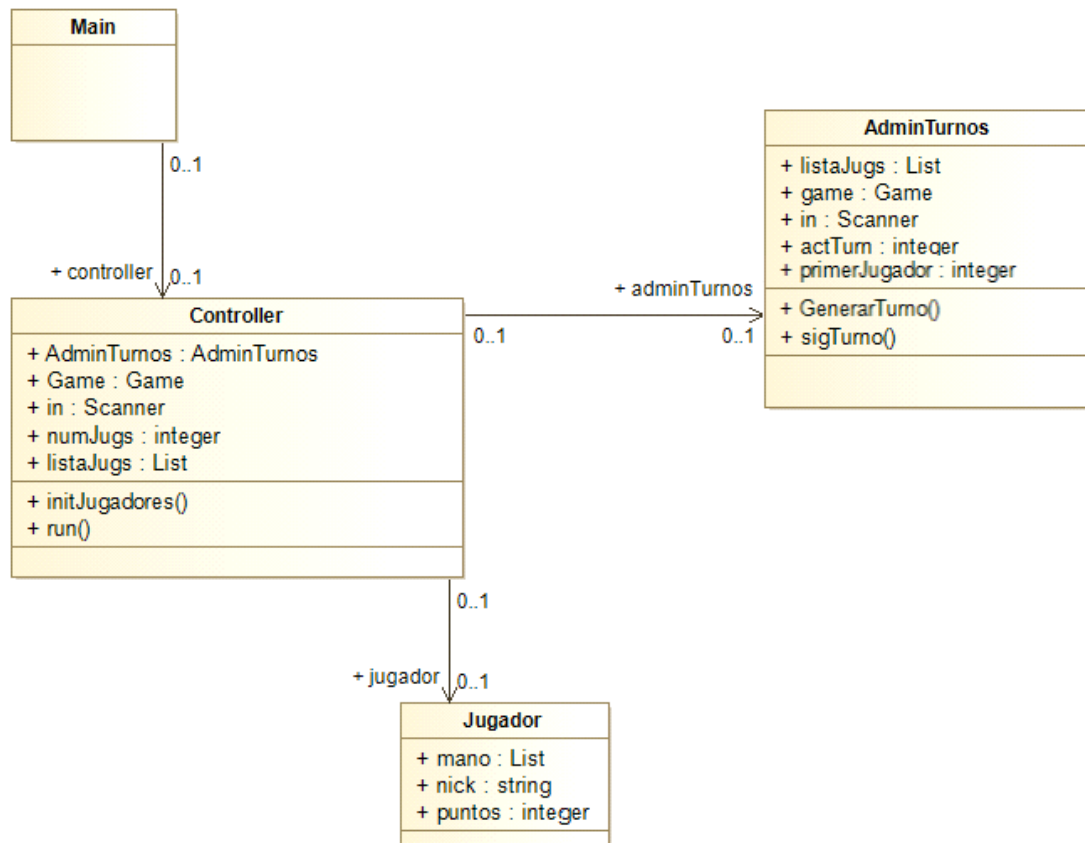
Después de la refactorización general del modelo, este método pasó a llamarse firstTurn() y su índice resultante se almacenaba en el entero del administrador que lleva el índice del jugador actual en la lista al construir el administrador. De esta manera, cuando creamos el administrador se inicializa este índice a un valor al azar entre todos los posibles jugadores.

Diagramas

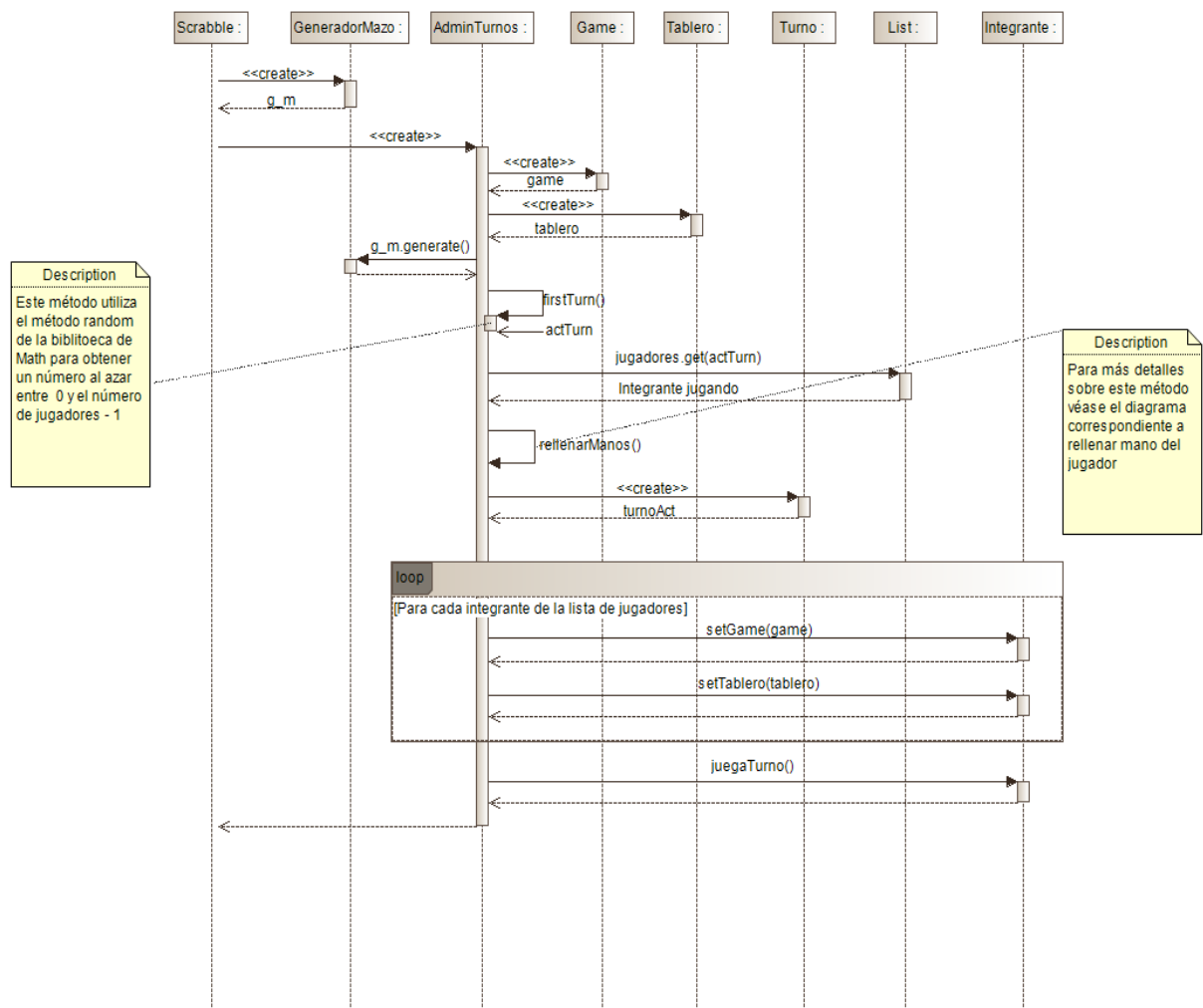
- Diagrama de secuencia previo a la refactorización



- Diagrama de clases anterior

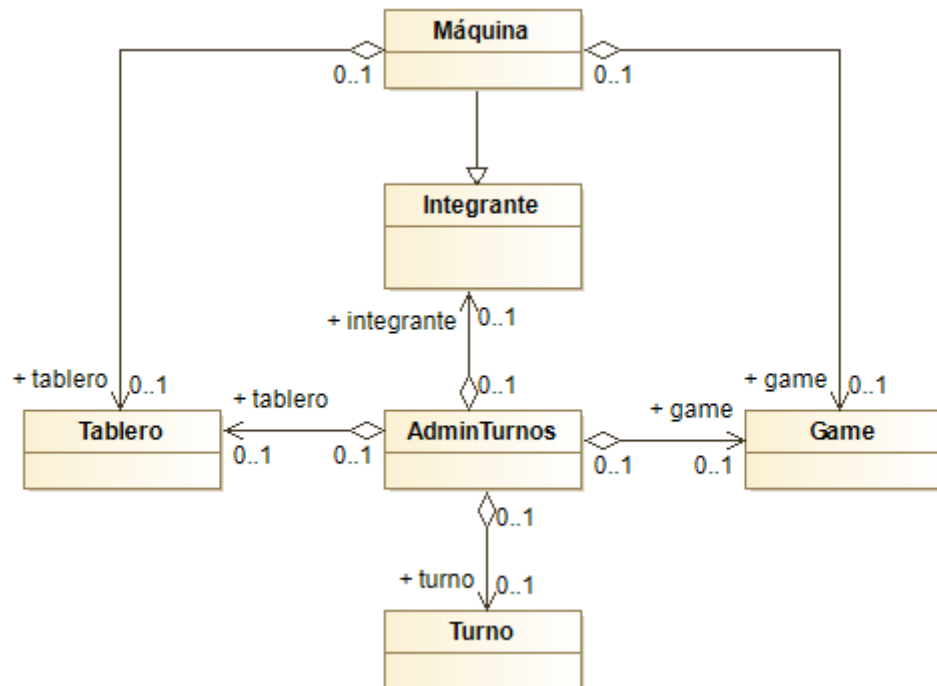


- Diagrama de secuencia actual



[Click para ampliar](#)

- Diagrama de clases actual



3.21 Tener en la pila de fichas dos comodines

ID:H_21	Usuario:Jugador		
Nombre historia: Tener en la pila de fichas dos comodines.			
Prioridad: Media		Estado: Completado	
Puntos estimados: 1		Iteración asignada: 2	
Programador responsable: María Cristina Alameda Salas			
Descripción: Como jugador quiero poder tener dos comodines disponibles para jugarlo.			

Explicación

Llegado un punto quisimos añadir la funcionalidad al juego de poder jugar con comodines. Así, esta historia de usuario solo requiere modificar el archivo de carga de fichas y añadir la ficha de letra '*' y puntuación 0.

Diagramas

No hemos considerado diagramas ya que únicamente las modificaciones han sido en el archivo de texto.

3.22 Canjear comodines de la mano por letras

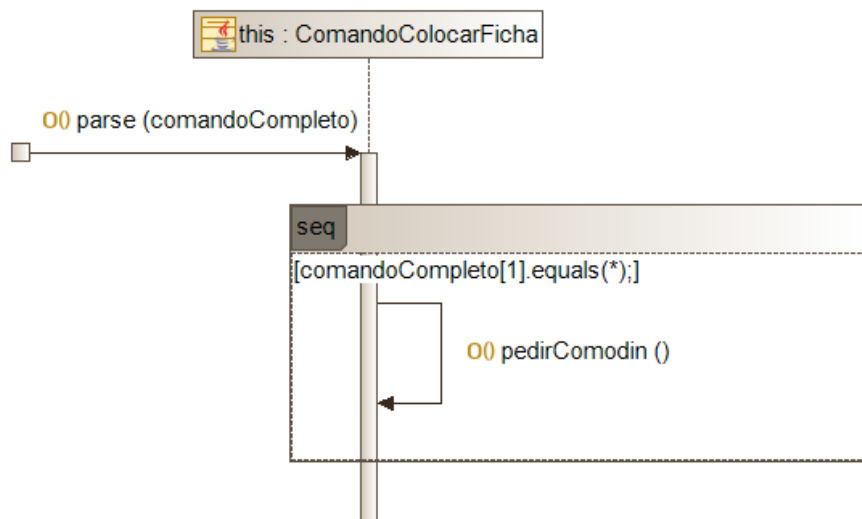
ID:H_22	Usuario:Jugador		
Nombre historia:Canjear comodines de la mano por letras.			
Prioridad: Media		Estado: Completado	
Puntos estimados: 3		Iteración asignada: 2	
Programador responsable: Mª Cristina Alameda			
Descripción: Como jugador quiero poder canjear los comodines de mi mano por letras para poder formar palabras.			

Explicación

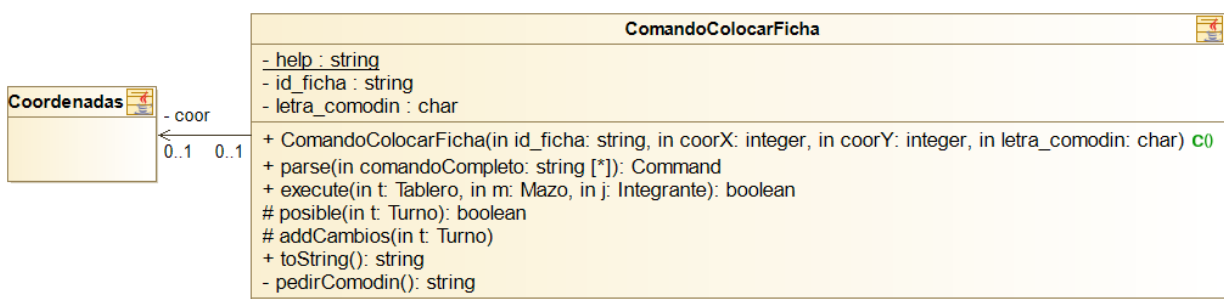
Para poder utilizar los comodines, se modificó el método colocarFicha del game. En él, cuando se quería colocar un comodín, se le pedía al usuario por consola que letra deseaba colocar en su lugar.

Más tarde, con la implementación de la GUI, se pasó a pedir esta letra en el comando colocar ficha. De esta manera en su método parse() cuando comprobaba que una letra era un comodín (*) le pedía al usuario la letra correspondiente.

Diagramas



- Diagrama de secuencia
- Diagrama de clases



3.23 Disponer de casillas especiales

ID:H_23	Usuario:Jugador
Nombre historia:Disponer de casillas especiales	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 5	Iteración asignada: 2
Programador responsable: Tania Romero Segura	
Descripción: Como jugador quiero poder aprovechar las casillas especiales para obtener más puntos por mis jugadas	

Explicación

Para este sprint decidimos que ya deberíamos incluir las casillas especiales típicas del juego de Scrabble.

Para ello, era necesario distinguir entre las casillas de alguna manera. La primera implementación constaba de una clase Casilla que ya teníamos previamente y de una serie de clases que heredan de Casilla: Casilla Roja, CasillaAzul, CasillaMorada y CasillaNaranja. Estas nuevas clases eran iguales que la primera y lo único que cambiaba era un nuevo atributo entero al que llamamos multiplicador. Para distinguir qué efecto tenía cada casilla sobre la palabra, utilizábamos el atributo multiplicador cuyo valor podía ser:

- 1 : casillas normales
- 2 : casillas de doble puntuación por letra
- 3 : casillas de triple puntuación por letra
- 4 : casillas de doble puntuación por palabra
- 5 : casillas de triple puntuación por palabra

Más adelante se decidió que no tenía sentido tener tantas clases con un simple número de diferencia y se eliminaron todas estas clases para quedarnos con una única clase con el atributo multiplicador para distinguir su efecto.

Para incorporar estas nuevas casillas se modificó el método que inicializa el tablero. A la hora de inicializar el tablero se utilizan dos bucles for de 0 a 14 para poder crear el cuadrado de casillas. Por lo tanto, dependiendo del valor de la i (fila del tablero) y la j (columna del tablero) podíamos saber basándonos en el tablero original si le corresponde el tipo de casilla x. Por ejemplo, si la casilla es la (0, 0), la (0,14), la (14, 0) o la (14, 14) se corresponde con las esquinas del tablero y estas esquinas deben tener multiplicador 5 (triple puntuación por palabra).

Diagramas

Para esta funcionalidad no se crearon diagramas ya que no es más que incorporar muchas condiciones a un método ya existente lo que hace que el diagrama no aporte mucha claridad.

3.24 Poner la primera ficha de la partida en el centro

ID:H_24	Usuario:Jugador		
Nombre historia:Poner la primera ficha de la partida en el centro			
Prioridad:Media		Estado:Terminado	
Puntos estimados: 3		Iteración asignada: 3	
Programador responsable: Guillermo García Patiño Lenza			
Descripción: Como Jugador quiero que la primera ficha de la partida sea colocada en el centro del tablero			


Explicación

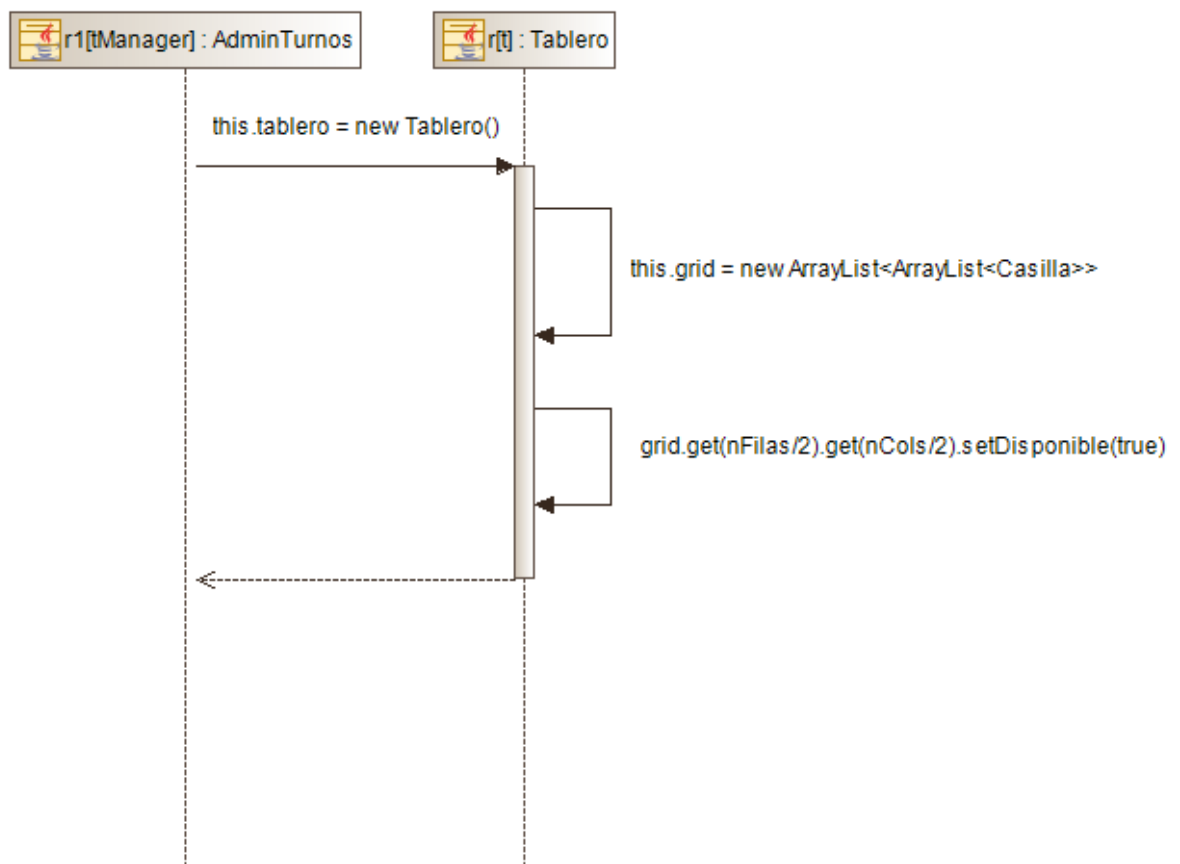
Cuando quisimos implementar en el juego las **reglas de negocio** correspondientes a la **colocación de fichas sobre el tablero**, pensamos en añadir un **atributo booleano**, **‘disponible’** a las casillas.

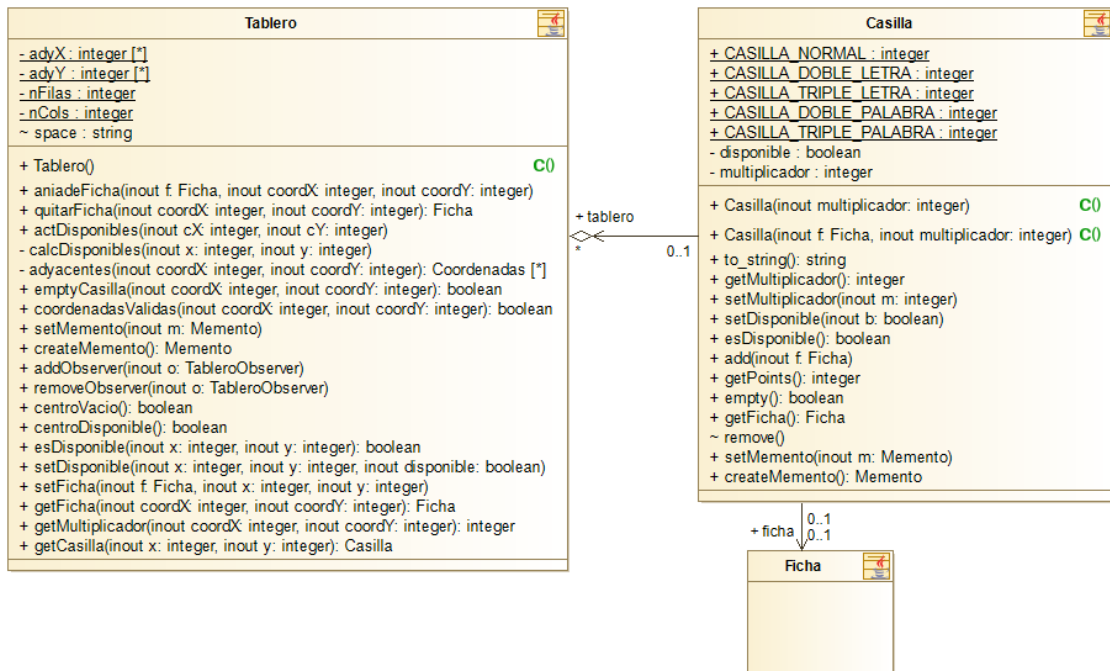
Además, **añadimos al tablero un atributo ‘Center’** que servía como forma **sencilla** de **acceder a la casilla central** y consultar su información. En cuanto al atributo ‘disponible’ de la casilla central, se pone a ‘true’ en el constructor del tablero y cada vez que se altera, su valor se pone a ‘true’ si la casilla tiene una ficha puesta en ella, o a ‘false’ si no.

Diagramas

(hacer click para ver los diagramas más grandes)

Tablero	
<ul style="list-style-type: none"> - <u>adyX</u> : integer [*] - <u>adyY</u> : integer [*] - <u>nFilas</u> : integer - <u>nCols</u> : integer ~ space : string 	
<ul style="list-style-type: none"> + Tablero() + aniadeFicha(inout f: Ficha, inout coordX: integer, inout coordY: integer) + quitarFicha(inout coordX: integer, inout coordY: integer): Ficha + actDisponibles(inout cX: integer, inout cY: integer) - calcDisponibles(inout x: integer, inout y: integer) - adyacentes(inout coordX: integer, inout coordY: integer): Coordenadas [*] + emptyCasilla(inout coordX: integer, inout coordY: integer): boolean + coordenadasValidas(inout coordX: integer, inout coordY: integer): boolean + setMemento(inout m: Memento) + createMemento(): Memento + addObserver(inout o: TableroObserver) + removeObserver(inout o: TableroObserver) + centroVacio(): boolean + centroDisponible(): boolean + esDisponible(inout x: integer, inout y: integer): boolean + setDisponible(inout x: integer, inout y: integer, inout disponible: boolean) + setFicha(inout f: Ficha, inout x: integer, inout y: integer) + getFicha(inout coordX: integer, inout coordY: integer): Ficha + getMultiplicador(inout coordX: integer, inout coordY: integer): integer + getCasilla(inout x: integer, inout y: integer): Casilla 	C0





3.25 Guardar partida a medias

ID:H_25	Usuario:Jugador		
Nombre historia:Guardar partida a medias			
Prioridad: Alta		Estado: Terminado	
Puntos estimados: 3		Iteración asignada: 3	
Programador responsable: Alejandro Rivera			
Descripción: Como Jugador quiero poder guardar la partida en mitad de una partida			

Explicación

Durante esta etapa del desarrollo del proyecto fue cuando **se nos empezaron a introducir realmente los principios de diseño y los diversos patrones** que hemos ido viendo durante el curso por lo que realmente el diseño no destaca especialmente.

También cabe destacar que **en la asignatura de TP estábamos aprendiendo a manejar por primera vez los JSON** por lo que no conocíamos realmente las posibilidades que estos ofrecen.

La idea de esta funcionalidad era la siguiente:

Una vez que se terminase la partida se guardarían los datos relevantes del jugador (nombre, puntuación y monedas) **para posteriormente cargarlo en una tabla de puntuaciones** cada vez que se empezara una nueva partida, de modo que se pudiese llevarse un ranking de los mejores jugadores.

Centrándonos en el diseño, como se puede apreciar en los diagramas incluidos más abajo, **la función de guardar se fundamenta en la interacción de 3 clases distintas, Controller, AdminTurnos y Jugador.**

Durante esta temprana etapa del desarrollo, el juego se ejecutaba mediante un bucle que se encontraba **en Controller** por lo que **la llamada al método save ()** (el encargado de guardar la partida) **se tenía que realizar una vez ese bucle finalizase** (es decir, hubiese terminado la partida)

La clase AdminTurnos fue la elegida para encargarse del manejo de ficheros por el simple motivo de que **Controller tenía acceso a estay esta a su vez a la lista de jugadores** por lo que podría obtener de forma fácil los datos requeridos.

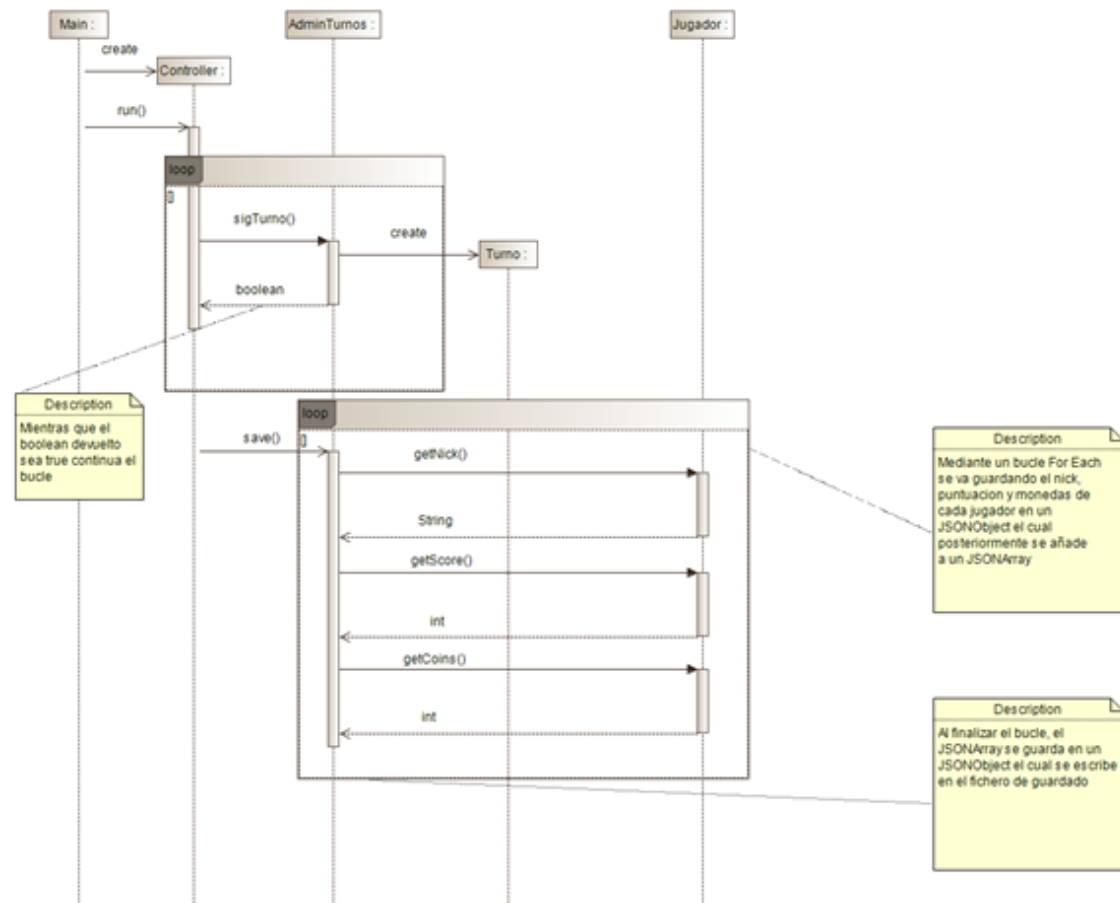
En la clase Jugador se añadieron Gets para obtener la información a través de AdminTurnos, esto obviamente hacía que ambas clases estuviesen demasiados acopladas y daba a AdminTurnos responsabilidades que le correspondían a Jugador, todo esto lo vimos claramente una vez se nos explicaron los **patrones GRASP** y en concreto el **patrón de alta cohesión** y el **patrón de bajo acoplamiento.**

Esta historia de usuario al final fue eliminada en Sprints posteriores por varios motivos:

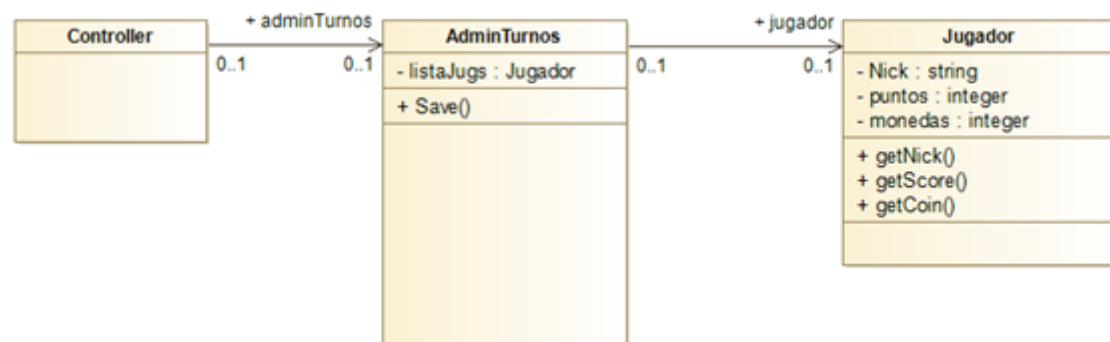
- Si quisiéramos preservar más datos el sistema se volvería inviable debido a su **poca escalabilidad.**
- **La tabla de puntuaciones se desechó al no ser algo prioritario** dentro del desarrollo del juego

- Con la explicación de los **patrones GOF** vimos que **esta forma** de llevar a cabo este tipo de funcionalidades **no era la más óptima**.

Diagramas



[Click para verlo más grande.](#)



[Click para verlo más grande.](#)

3.26 Cargar partida a medias

ID:H_26	Usuario:Jugador
Nombre historia:Cargar una partida a medias	
Prioridad: Alta	Estado: Finalizado
Puntos estimados: 3	Iteración asignada: 3
Programador responsable: Gema Blanco Núñez	
Descripción: Como jugador quiero que la primera ficha de la partida sea colocada en el centro del tablero	

Explicación

El **objetivo** de esta historia de usuario **nunca se llegó a concretar** y por tanto fue **eliminada** posteriormente. Además, en el momento en el que se elimina la historia de usuario guarda una partida a medias, carece de sentido por sí misma el cargar una partida a medias.

Al igual que para guardar una partida, seguimos la misma dinámica **inicialmente** para cargarla usando el patrón Memento.

Tenemos una **clase Memento** que implementa la **interfaz IMemento** y permite crear Mementos guardando la información en formato JSONObject. El resto de clases implementan la **interfaz Originator**, que contiene dos métodos: *setMemento(Memento)* para cargar datos de una partida y *createMemento()*, que devuelve un Memento con los datos para guardar una partida.

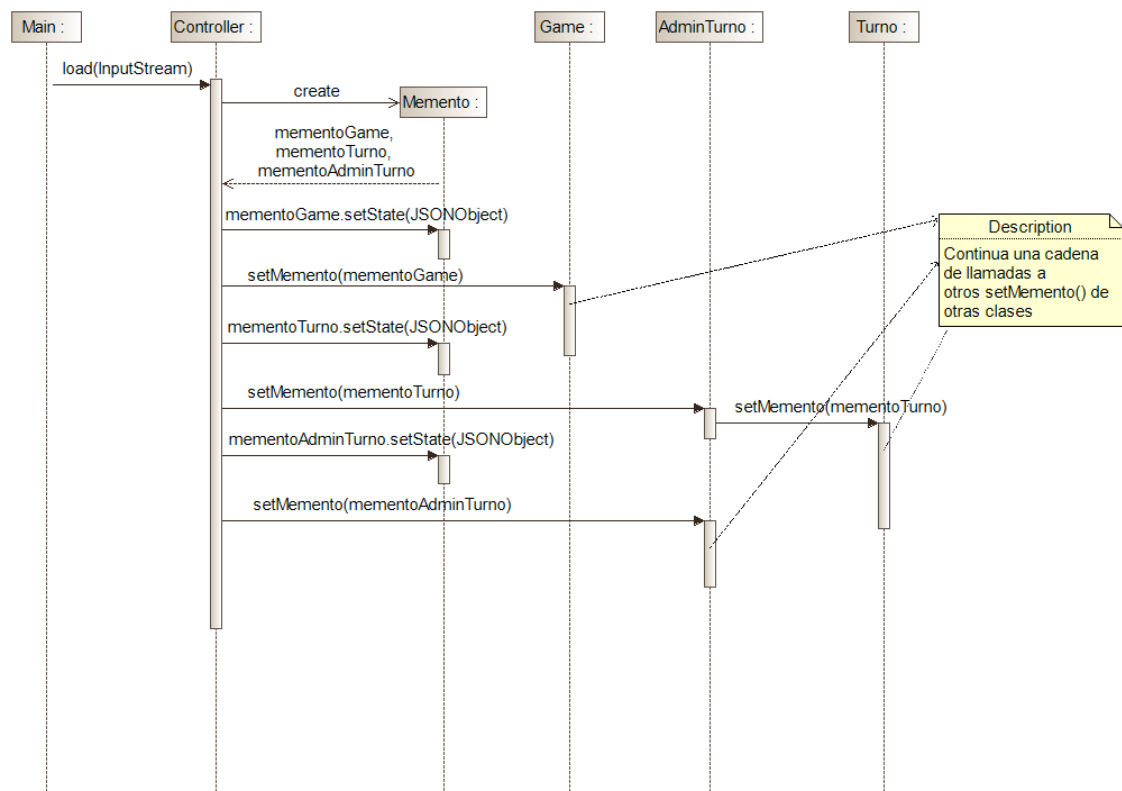
Como se explica para guardar una partida, inicialmente el Controller creaba **tres mementos principales** con la información relativa a las clases Game, AdminTurnos y Turnos. Tras guardar la información (JSONObject) en los mementos se llamaba a los métodos *setMemento()* de la clase Game para guardar el memento del Game, y al

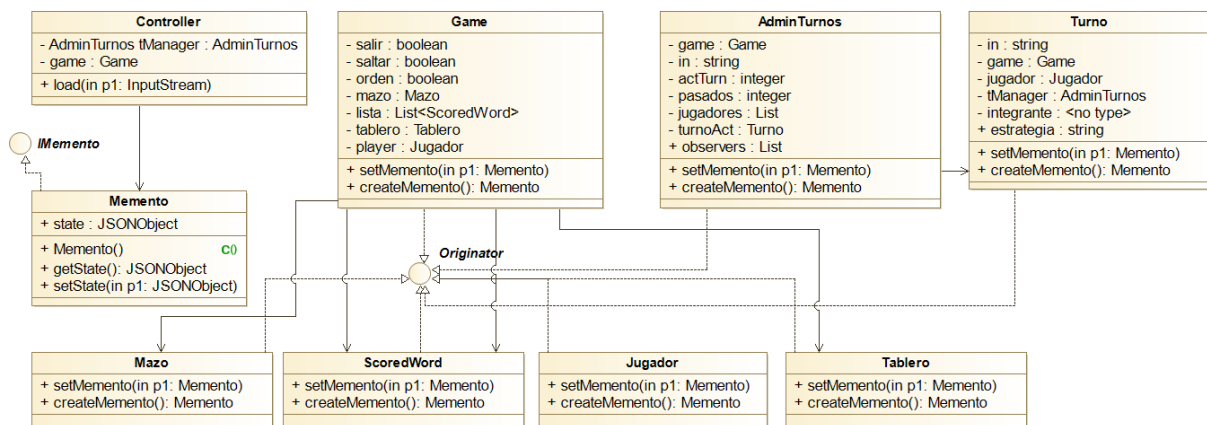
setMemento() de la clase AdminTurnos para guardar tanto el memento del AdminTurnos como el de Turnos, ya que el Controller no disponía de acceso directo al turno.

Tanto el **diagrama de clases** como el **diagrama de secuencia** pretendían ser generales para visualizar el funcionamiento del patrón memento tal y como estaba implementado al inicio.

Diagrama

- Diagrama de secuencia
- Antiguos





● Diagrama de clases

3.28 Verificar las palabras automáticamente al finalizar el turno

ID:H_28	Usuario:Jugador		
Nombre historia:Verificar las palabras correctas automáticamente al finalizar el turno			
Prioridad: Alta		Estado:	
Puntos estimados: 3		Iteración asignada: 3	
Programador responsable: María Cristina Alameda Salas			
Descripción: Como Jugador quiero que se verifiquen las palabras correctas automáticamente al finalizar mi turno para que se sumen a mis puntos todas			

las correctas.

Explicación

Durante este sprint pensamos que teníamos la suficiente funcionalidad como para introducir que las palabras colocadas se verifican automáticamente.

Para introducir este requisito, únicamente se añadió al game un nuevo método llamado `-verificar-`. Este método sustituye a un anterior `verificar` que comprobaba si una palabra entre unas coordenadas proporcionadas directamente por el usuario era correcta. La diferencia con este anterior es que ya no se realiza entre unas coordenadas dadas sino que se realiza sobre las fichas colocadas por el jugador en su turno.

Métodos, atributos y clases añadidas:

1. Se añadió a la clase `Turno` una lista de coordenadas que simboliza las fichas colocadas por el jugador durante su turno.
2. Para poder modificar esta lista desde los comandos una vez se que ejecutaran, se añadió a los mismos el método `-addCambios(Turno)`. Así, cuando se llama al método `execute` de los comandos, se chequea que sea posible ejecutarlo, se ejecuta y se añaden los cambios pertinentes a turno.

Este método `-addCambios-` supone que la ejecución de los comandos devuelva un boolean que indique si se ejecuto correctamente. De esta manera, al colocar una ficha de manera correcta (una ficha que esté en tu mano y en una casilla correcta), el `addCambios` añade a la lista de fichas del turno las coordenadas donde se colocó la ficha. En el caso de quitar la ficha del tablero, el `addCambios` la elimina.

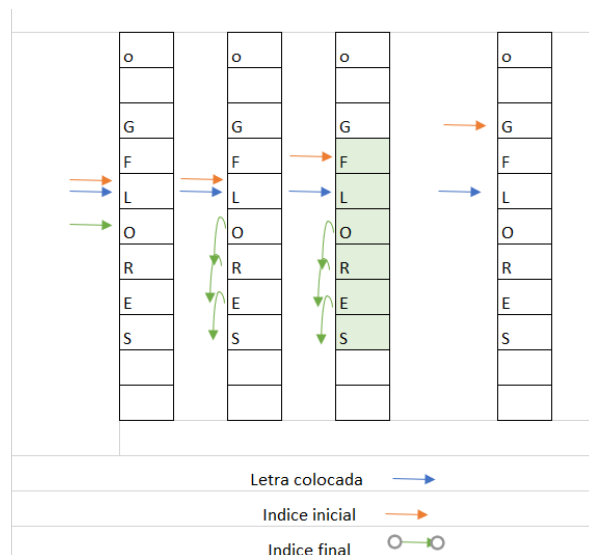
3. Se creó una nueva clase `Coordenadas`. Sus atributos comprenden dos enteros que indican la fila y la columna correspondiente del tablero. Entre sus métodos se añadió un método para saber si unas coordenadas son menor que otra. Este método es utilizado en `WordScored` para saber si una palabra está contenida en otra.
4. También se creó una clase `ScoredWord`. Esta clase abstrae una palabra que se ha verificado. Entre sus atributos encontramos un string correspondiente a la palabra verificada, un entero que informa de su

puntuación total y las coordenadas iniciales y finales de la palabra (donde comienza y dónde acaba en el tablero). Entre sus métodos encontramos el método `-thisContenidaEnOther-`, en el cual dada `this` y otra `ScoredWord` devuelve un boolean indicando si está contenida (Ej: perro está contenida en perros, punta está contenida en sacapuntas....). Su método `equals` devuelve true sólo cuando dos palabras tienen igual palabra, puntuación y coordenadas iniciales y finales.

5. Se añadió al game una lista de `ScoredWord` en el que se almacenan la lista de todas las palabras verificadas durante una partida.

Proceso de verificación del tablero:

1. Cuando un jugador solicita pasar el turno, se llama al método `verificar`. Este método recibe la **lista de coordenadas** del turno, el tablero para comprobar y un integrante al que sumar los puntos.
2. Lo primero que hace es crear un **HashSet** de `ScoredWord`, de esta manera no se repetirán palabras verificadas dos veces y solo se añadirán los puntos una vez.
3. Entonces comienza a recorrer la lista de coordenadas en un **bucleforeach**. Para cada coordenada de la lista mirará si ha formado alguna palabra en vertical y en horizontal.
 - a. **Verificación vertical:** Me **guardo** en un entero la **fila de la coordenada** que estoy verificando. Mientras las coordenadas formadas sean **válidas** y no me encuentre **una casilla vacía** se analiza si se forman palabra en vertical. El proceso consiste en avanzar sobre la columna hacia abajo desde un **índice inicial hasta un índice final**. Este índice inicial comienza siendo la fila guardada y posteriormente se va **decrementando** hasta que encuentra una casilla vacía o se sale del tablero. El índice final comienza siendo el mismo de la fila guardada (para que incluya siempre la casilla que se quiere verificar) y se va **aumentando** bajo la misma condición que el anteriormente dicho. En la siguiente imagen se puede apreciar el proceso gráficamente.



Para cada palabra obtenida del tablero, se comprueba si se encuentra en el diccionario. Si se encuentra se analiza su puntuación y construyo una ScoredWord. Compruebo si está contenida en la lista de palabras o si hay alguna palabra que esté contenida en ella, añadiendo de esta manera o no la ScoredWord al HashSet.

- b. Verificación horizontal: Se realiza de la misma manera que en vertical pero sobre las filas del tablero.
4. Por último, se calcula la puntuación total de las palabras del HashSet recorriendolo y se añaden al jugador.
5. Si se ha verificado alguna palabra o no se colocó ninguna ficha en el tablero devuelve true, de esta manera solo puede un jugador pasar turno si ha colocado una palabra correcta o si no ha colocado ninguna letra.

Mejoras

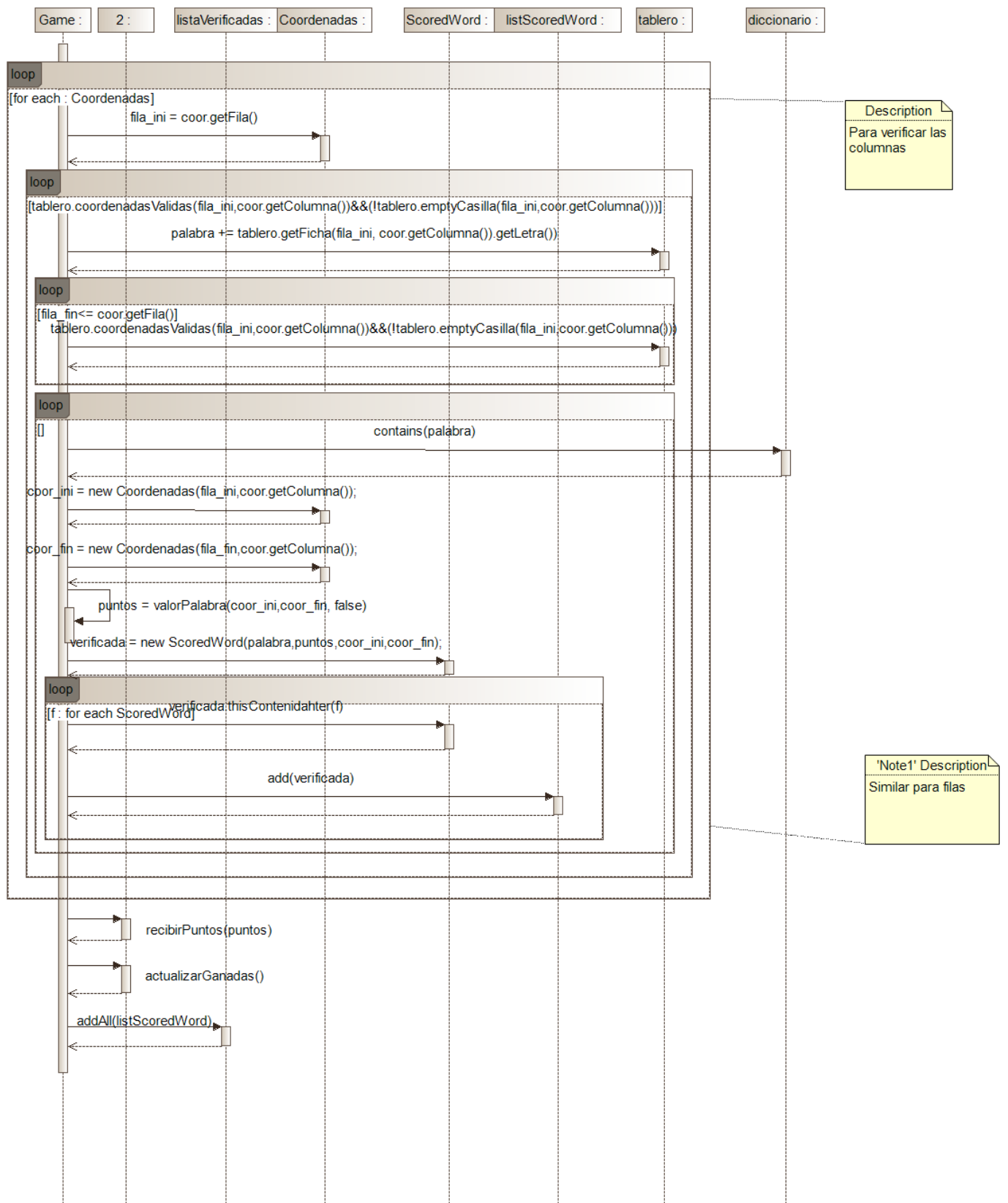
Con respecto al algoritmo: Se podría realizar el proceso la inversa. Actualmente, en cada iteración del bucle se comienza con una palabra pequeña (concretamente de tamaño dos letras) y según se van modificando los índices iniciales y finales, se va aumentando la longitud de la palabra. La mejora consistiría en comenzar con una palabra de la máxima longitud posible en esa fila/columna a partir del índice inicial e ir decrementando la longitud de la palabra (el índice final). Como siempre vamos buscando la palabra de máxima puntuación posible, este algoritmo sería más eficiente.

Con respecto al diseño, se podría refactorizar el método verificar ya que las partes del verificar columna y verificar filas son prácticamente iguales. Así se podría crear un método que dado un índice inicial, un índice final y un índice sobre el que realizarlo (una fila o columna) realizara la acción de verificar una columna o fila. Este método devolvería una lista de las palabras correctas. De esta manera, el método verificar iteraría sobre la lista de coordenadas colocadas y para cada una llamaría a este método dos veces, una para verificar sobre esta coordenada de manera horizontal y otra para vertical.

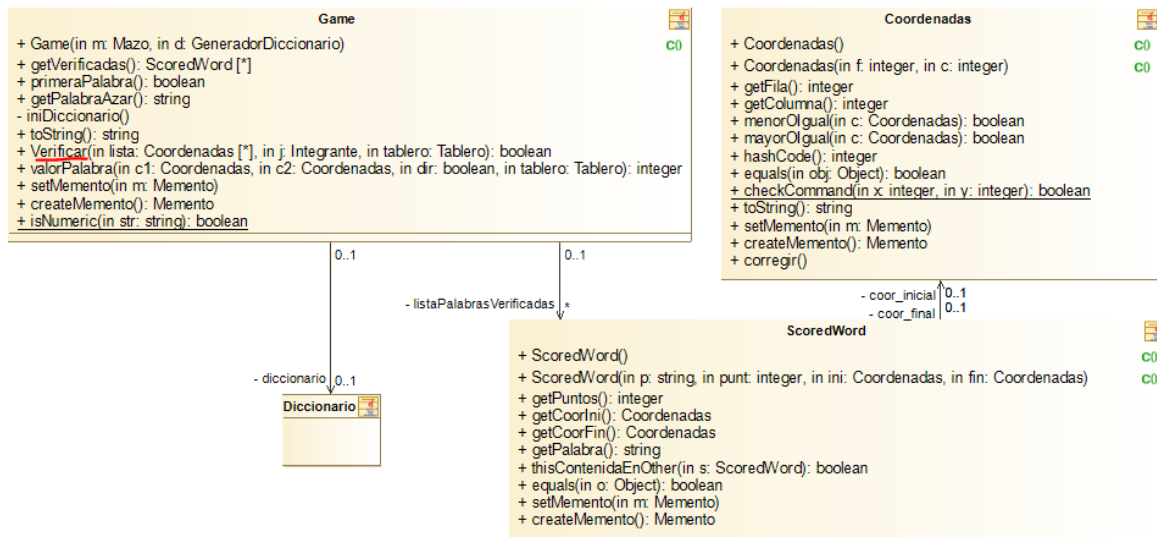
Estas mejoras se plantearon realizarlas en unos de los últimos sprint pero ya que esta versión funcionaba correctamente y que nos encontrábamos casi al final del proyecto no quisimos implementarla.

Diagrama

- Diagrama de secuencia



- Diagrama de clases



3.29 No pasar de turno si he colocado alguna ficha que no forme palabra

ID:H_28	Usuario:Jugador	
Nombre historia: No pasar de turno si he colocado alguna ficha que no forme palabra		
Prioridad: Alta		Estado: Terminado
Puntos estimados: 3		Iteración asignada: 3
Programador responsable: María Cristina Alameda Salas		

Descripción:

Como Jugador quiero que no pueda pasar de turno si he colocado alguna ficha que no forme palabra para que sean las partidas de acuerdo a las normas.

Explicación

Para poder llevar a cabo esta funcionalidad, hubo varias versiones. Además, podemos hablar de que esta historia de usuario supuso una evolución de la primera.

La primera versión consistía en verificar las palabras con el método verificar del Game cuando el jugador pasaba turno y aprovechando que este devolvía un boolean que indicaba si alguna palabra se había verificado o no, en el bucle del turno si no había verificado ninguna palabra se comprobaba que la lista de coordenadas de fichas colocadas durante el turno estuviera vacía. Si lo estaba se podía pasar de turno, si no lo estuviera, se mostraba un mensaje de aviso y no se realizaba dicha acción.

La segunda versión, realizaba lo mismo pero la condición de comprobar si la lista de coordenadas era vacía se relegaba al final del método verificar.

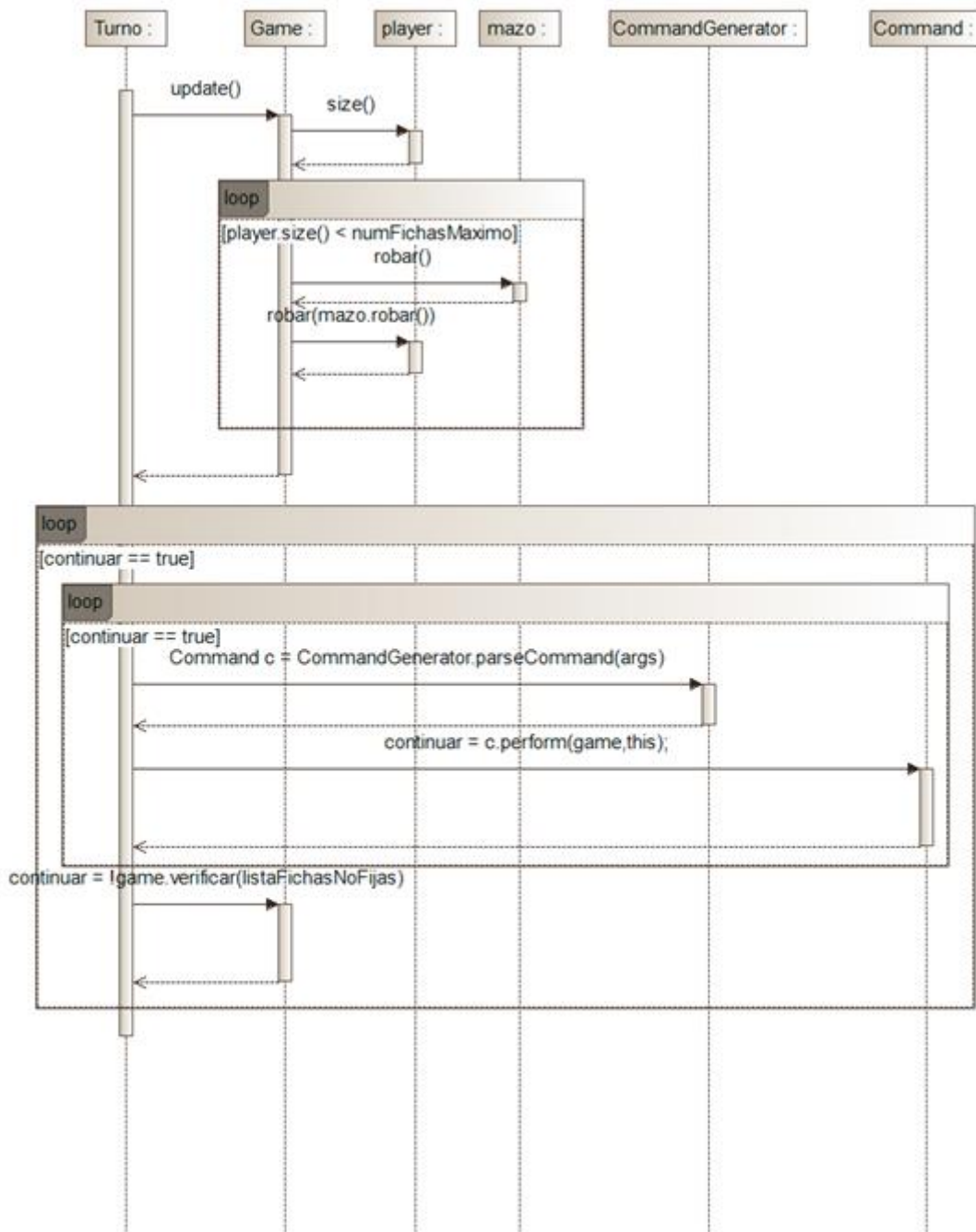
La tercera versión se realizó tras la refactorización. Ahora no se verifican las palabras desde el turno sino desde el administrador de turnos.

Diagramas

- Antiguos : segunda versión

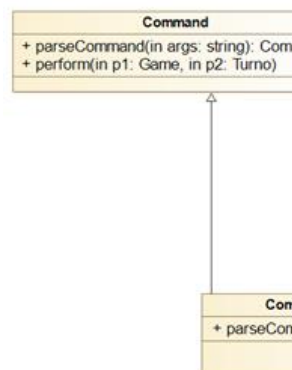
- Diagrama de secuencia

Como se puede apreciar al final del siguiente diagrama, se comprobaba si se había verificado bien. Si no, el turno no se acababa y por tanto continuaba.

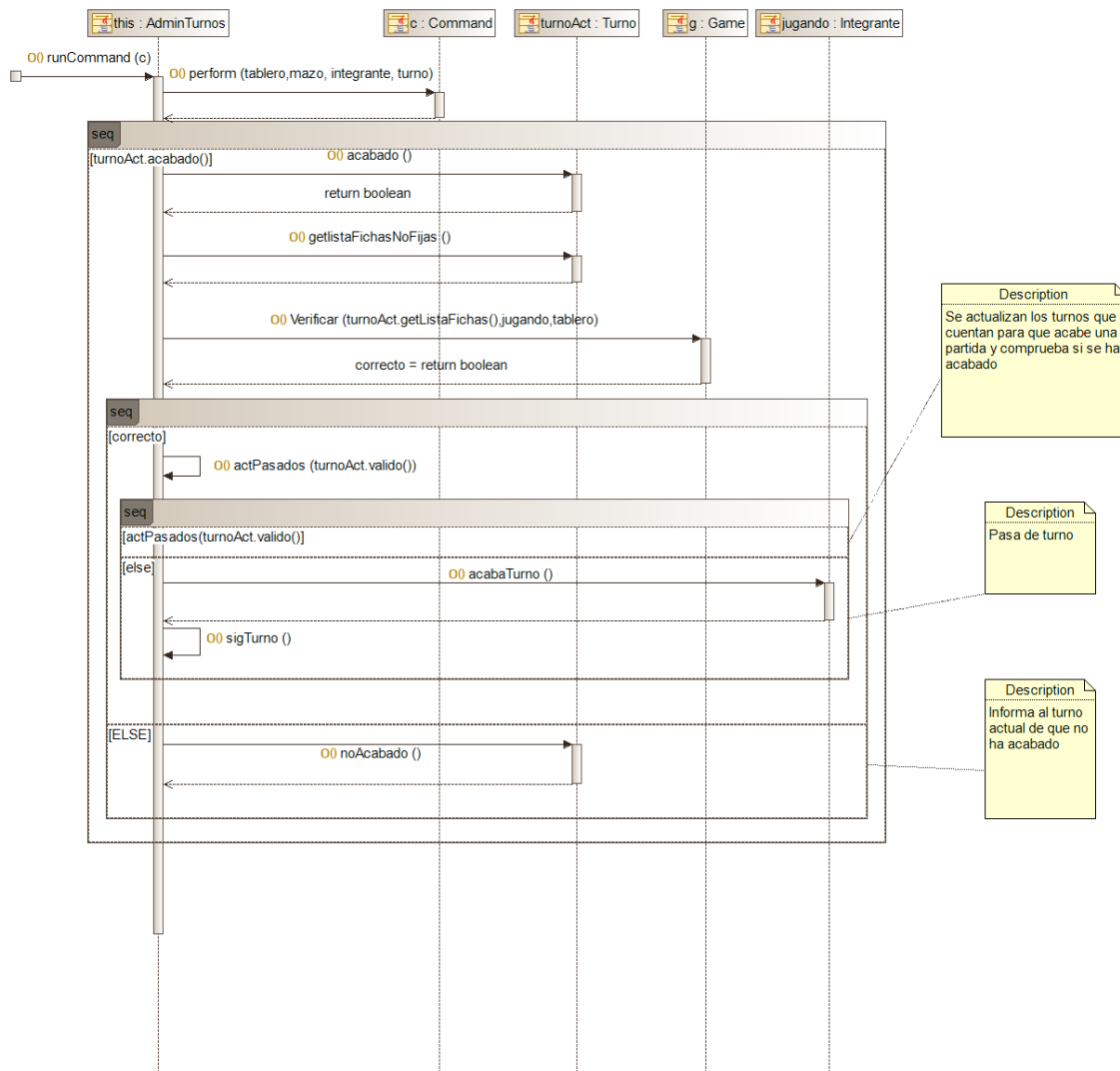


○ D

iagr
am
a
de
clas
es



- Nuevos: tercera versión



○ Diagrama de secuencia

[Click para ampliar](#)

3.29 Obtener monedas

ID:H_30	Usuario:Jugador
Nombre historia:Obtener monedas	
Prioridad: Alta	Estado: Finalizado

Puntos estimados: 3	Iteración asignada: 3
Programador responsable: Tania Romero Segura	
Descripción: Como jugador quiero poder obtener monedas cada vez que obtenga un número determinado de puntos	

Explicación

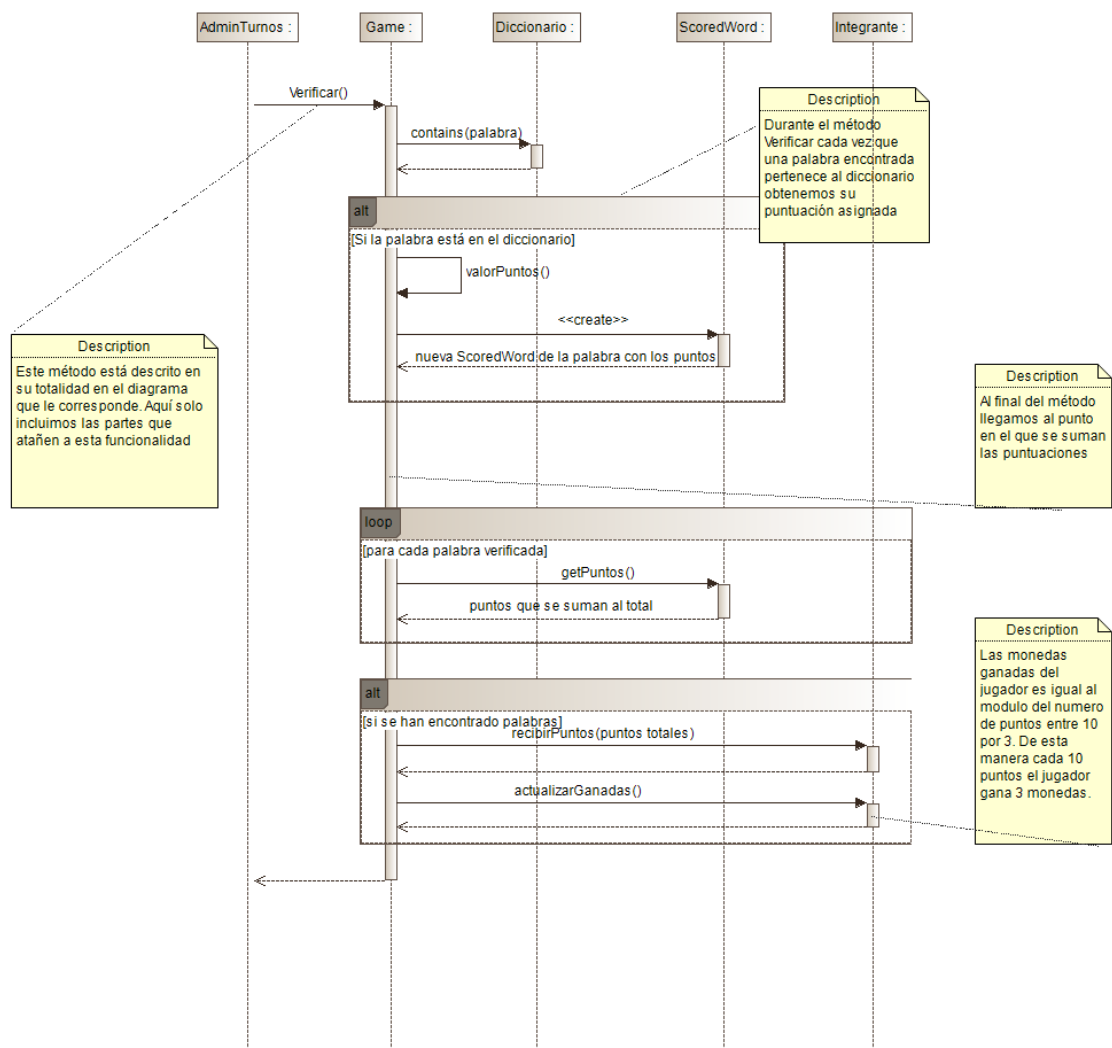
Como ampliación del juego, se planteó la posibilidad de ofrecerle al jugador la posibilidad de comprar lo que hemos llamado ventajas durante el juego para mejorar sus jugadas o entorpecer el avance de otro jugador. Para poder ofrecer esta posibilidad es necesario que exista una moneda de cambio para que el jugador tenga que sacrificar algo a cambio de esa ventaja. Al principio se pensó en gastar puntos, al final se decidió hacer que el jugador gane 3 monedas cada 10 puntos y que tenga que pagar monedas para comprar una ventaja.

Para que el jugador pueda ganar monedas, se decidió crear los atributos de la clase jugador (actualmente la clase integrante de la que hereda jugador): `monedas_ganadas` y `monedas_gastadas`. Esta es una manera sencilla de saber cuantas monedas tiene que haber ganado el jugador en función de sus puntos actuales en vez de tener que llevar la cuenta de los puntos que se van ganando con respecto a turnos anteriores. Así, para saber de cuántas monedas dispone el jugador basta con restar las gastadas a las ganadas y para gastar monedas basta con incrementar el número de monedas gastadas.

Para ganar estas monedas, se creó el método en la clase Jugador (ahora integrante) llamado `actualizarGanadas` que hace que `monedas_ganadas` sea igual al módulo de los puntos entre 10 y los multiplica por 3. De esta manera conseguimos que haya ganado 3 monedas por cada 10 puntos.

Cuando se verifica una palabra, después de sumar los puntos se llama al método `actualizarGanadas` y así las monedas se actualizan para el nuevo número de puntos.

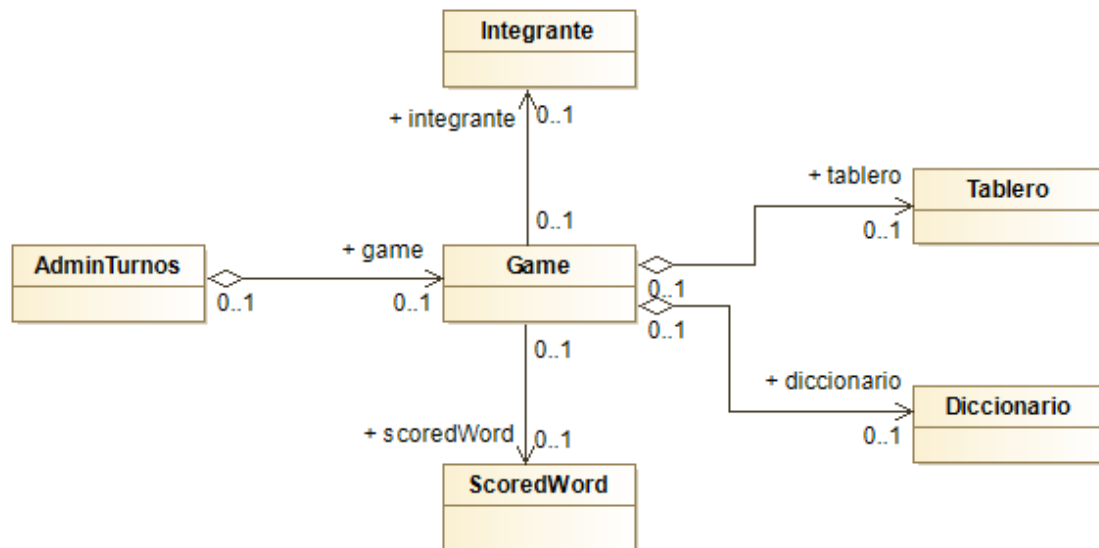
Diagramas



- Diagrama de secuencia

[Click para ampliar](#)

- Diagrama de clases



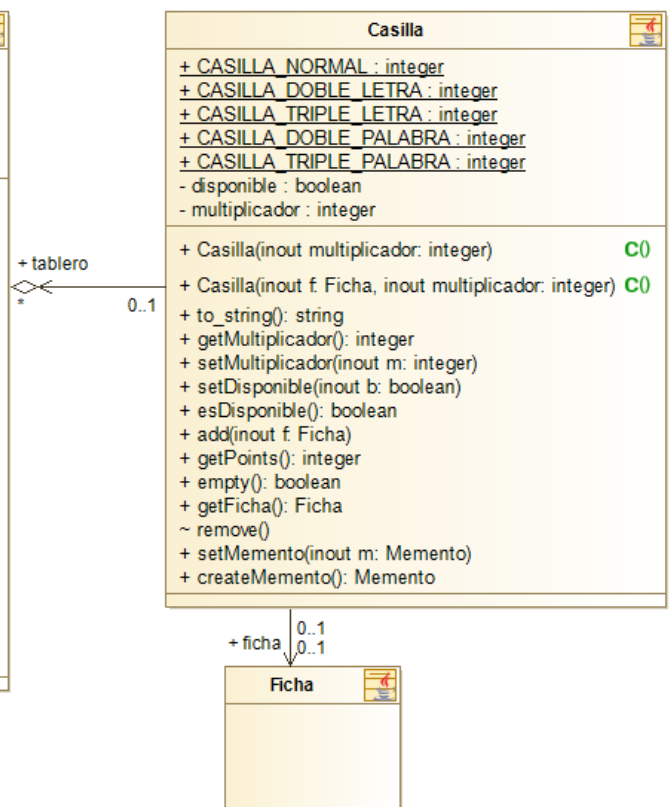
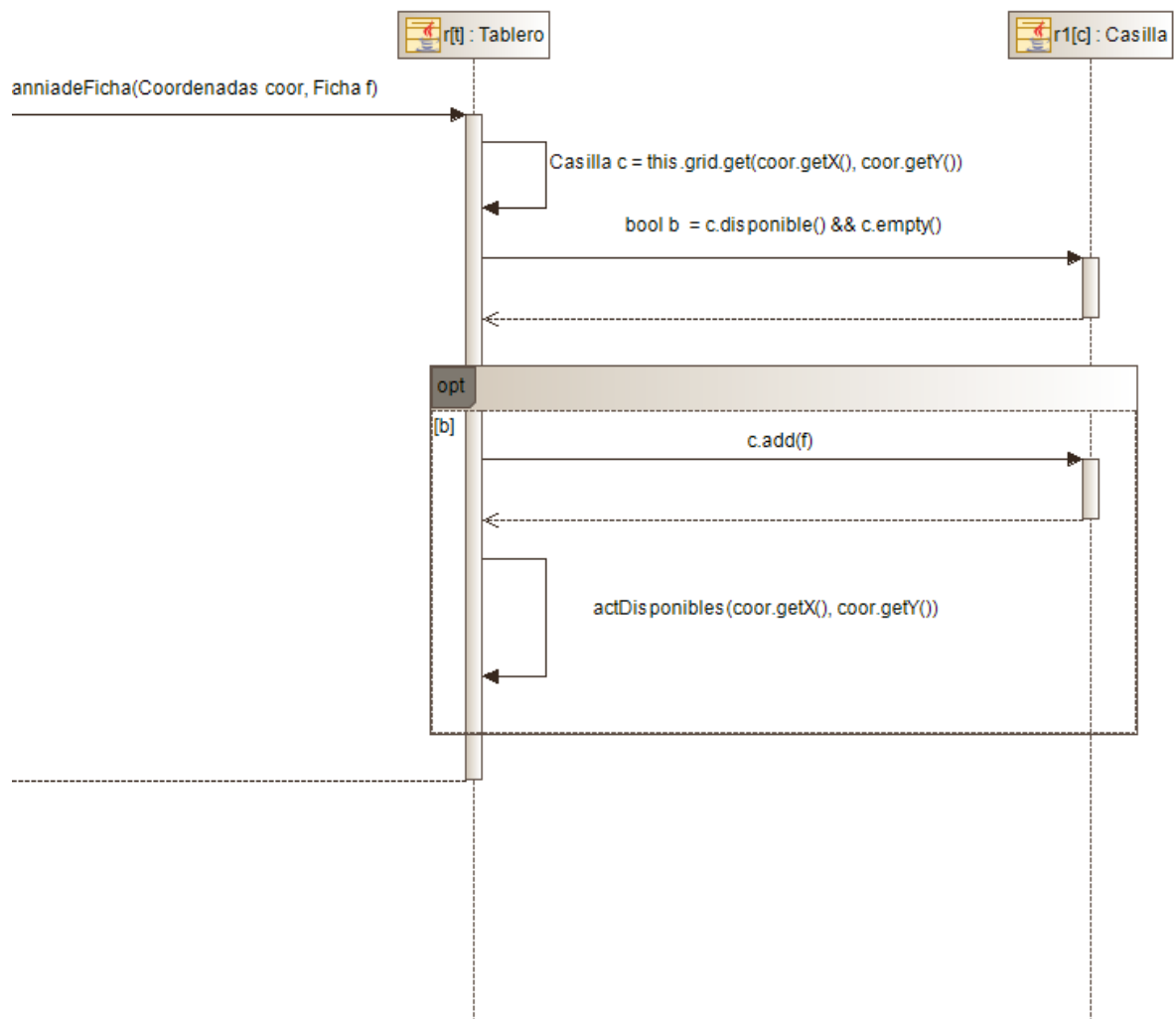
3.30 Solo colocar fichas al lado de fichas

ID: H_31	Usuario: Jugador
Nombre historia: Solo colocar fichas al lado de fichas	
Prioridad: Alta	Estado: Terminado
Puntos estimados: 2	Iteración asignada: 3
Programador responsable: Guillermo García Patiño Lenza	
Descripción: Como Jugador quiero que solamente se permita colocar una ficha en una casilla si la casilla tiene otra adyacente con una ficha.	

Explicación

Esta es otra historia de usuario destinada a **implementar reglas de negocio** acerca de la colocación de fichas en el tablero. Se implementó **manipulando el atributo 'disponible'** de las casillas **cada vez que se coloca una ficha** en el tablero.

Cada vez que se coloca una ficha en una casilla del tablero, **se recorren las casillas adyacentes** poniendo su atributo disponible a 'true' **empleando el método 'calcDisponibles'**. Después, **para cada una de esas casillas adyacentes**, se obtienen sus casillas adyacentes y se modifica el atributo disponible **empleando el mismo método 'calcDisponibles'**.



3.31 Si finaliza la partida obtener puntos por monedas sobrantes

ID:H_32	Usuario:Jugador
Nombre historia:Si finaliza la partida obtener puntos por las monedas sobrantes	
Prioridad: Baja	Estado: Finalizado
Puntos estimados: 1	Iteración asignada: 3
Programador responsable: Tania Romero Segura	
Descripción: Como jugador quiero poder obtener puntos como recompensa por no haber gastado todas mis monedas al finalizar la partida.	

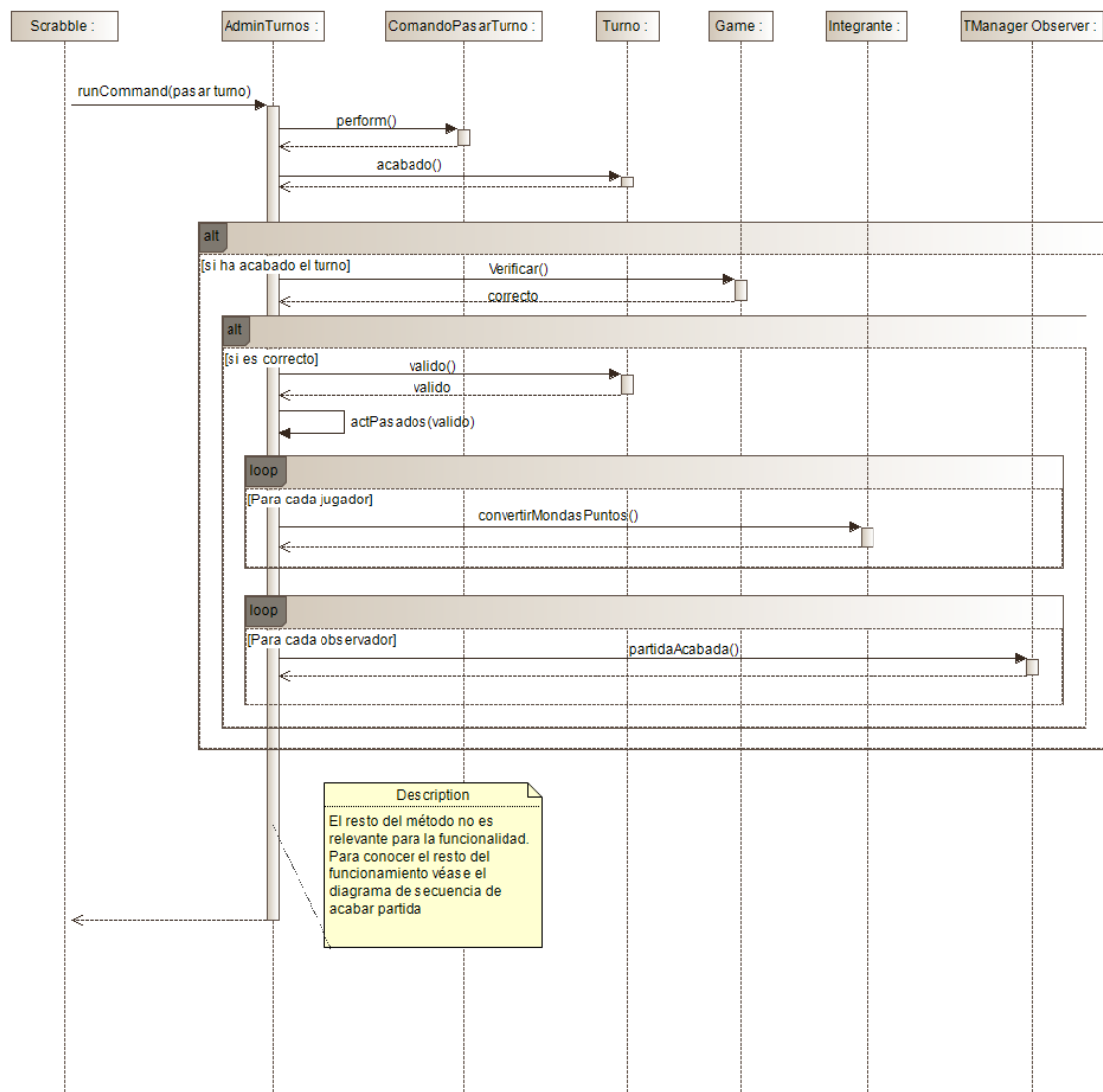
Descripción

Esta funcionalidad surgió por la idea de que si un jugador no gasta sus monedas debería ser recompensado de alguna manera. Por eso, se decidió que por cada moneda que no se gaste se deberían ganar 2 puntos al final de la partida.

Para implementar esta funcionalidad, se ha creado un método convertirMonedasPuntos en la clase Integrante de manera que las monedas del jugador se multiplican por dos y se suman a los puntos. Cuando se ha acabado la partida, antes de decidir el ganador se llama a este método para cada jugador de manera que las puntuaciones se actualizan con las monedas y después ya podemos decidir quién tiene más puntuación.

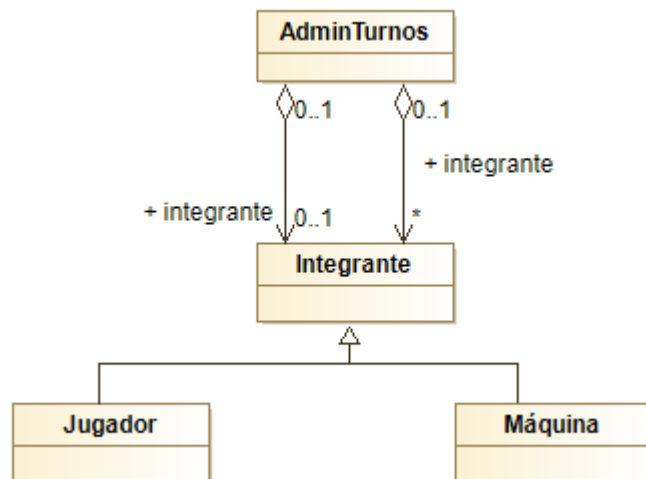
Diagramas

- Diagrama de secuencia



[Click para ampliar](#)

- Diagrama de clases



3.34 Cambiar una ficha por un comodín a cambio de monedas

ID:H_34	Usuario:Jugador
Nombre historia:Cambiar una ficha por un comodín a cambio de monedas	
Prioridad: Baja	Estado: Finalizado
Puntos estimados: 1	Iteración asignada: 3
Programador responsable: Tania Romero Segura	
Descripción: Como jugador quiero poder convertir una de mis fichas en un comodín a cambio de monedas	

Explicación

Una de las ventajas que pensamos que un jugador podría querer comprar es la de cambiar una ficha de la mano por un comodín. Pensamos que un precio medianamente decente para esta ventaja son 5 monedas.

Para implementar esta funcionalidad era necesario crear un nuevo comando al que llamamos ComandoComprarComodín al que el jugador debería aportarle qué ficha de su mano quiere sustituir. De esa manera, si el jugador no tiene esa ficha se lanzaba una excepción (esto ya no es así porque con la GUI es

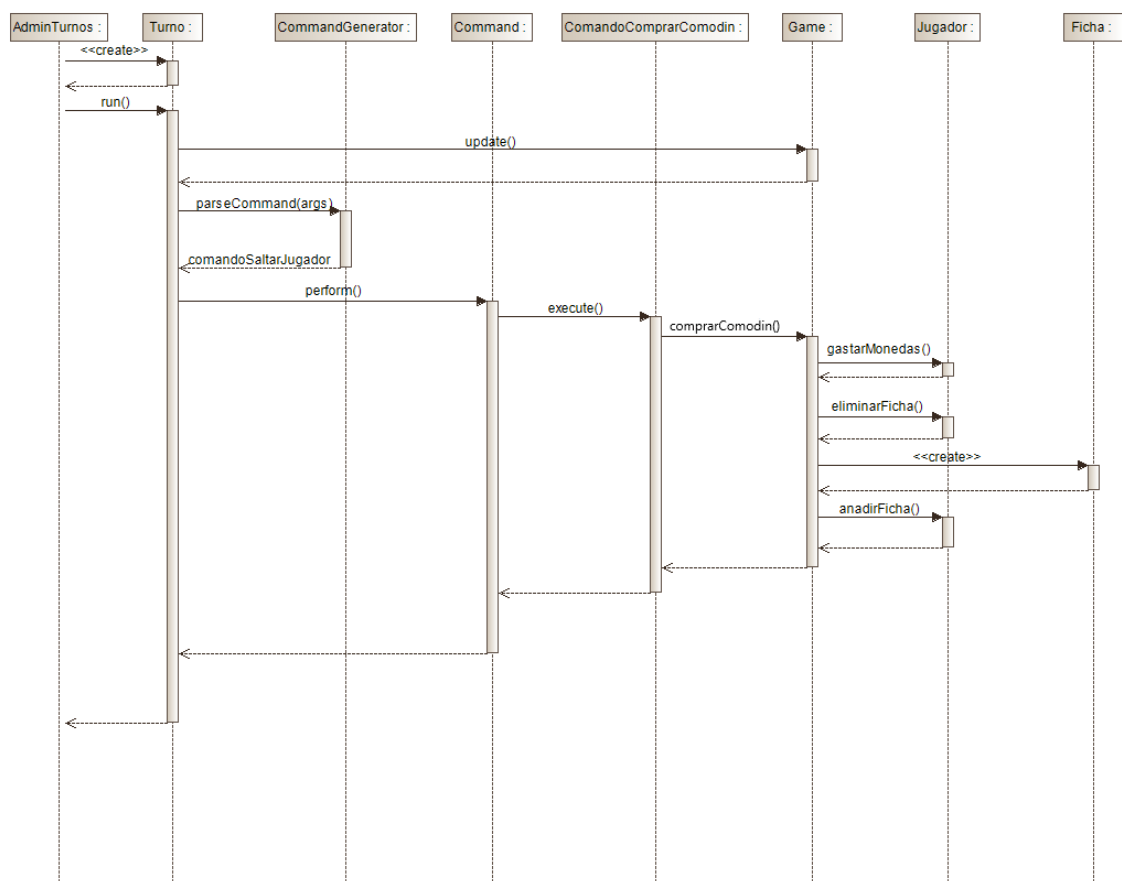
imposible que el jugador seleccione una ficha que no tiene). Para ejecutar este comando ha habido dos sistemas: uno antes del refactor del modelo y la actual.

Antes de la refactorización del modelo el execute del comando llamaba al método del gamecomprarComodín que llamaba al método gastarMonedas del jugador (si el jugador no tiene suficientes monedas se lanza excepción), eliminaba la ficha seleccionada por el jugador de su mano, creaba una nueva Ficha comodín y se la pasaba al jugador para que la robase.

Ahora, cuando se ejecuta un comando pasa por una serie de pasos previos. Cuando se quiere ejecutar un comando se llama al runCommand del controller, este llama al runCommand de la clase Scrabble y después ésta ejecuta el runCommand del AdminTurnos. En este último método runCommand se llama al execute del comando. El execute después hace esencialmente lo mismo que hacía el antiguo método del gamecomprarComodin.

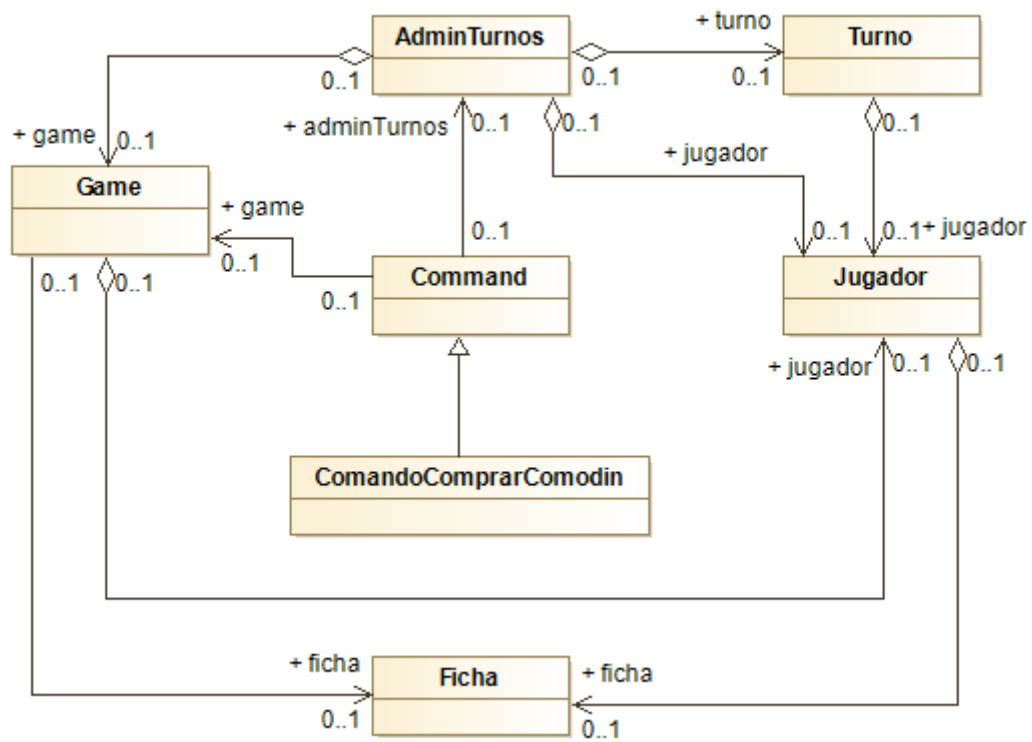
Diagramas

- Diagrama de secuencia anterior



[Click para ampliar](#)

- Diagrama de clases anterior



- Diagrama de secuencia actual

-
- ```

classDiagram
 class Jugador
 class Integrante
 class Game
 class Ficha
 class ComprarVentajasDialog
 class ComandoComprarComodin
 class Command
 class AdminTurnos
 class Scrabble
 class Controller
 class ControllerLocal

 Jugador --> Integrante : + utiliza 0..1
 Integrante --> Game : + integrante 0..1
 Game --> Ficha : + ficha 0..1
 Integrante --> ComprarVentajasDialog : + jugador 0..1
 ComprarVentajasDialog --> ComandoComprarComodin : + comandoSaltarJugador 0..1
 ComandoComprarComodin --|> Command
 Integrante --> AdminTurnos : + utiliza 0..1
 AdminTurnos --> Scrabble : + ejecuta 0..1
 AdminTurnos --> Scrabble : + adminTurnos 0..1
 Scrabble --> Integrante : + ficha 0..1
 Scrabble --> ControllerLocal : + scrabble 0..1
 ControllerLocal --|> Controller
 ControllerLocal --> ComprarVentajasDialog : + controllerLocal 0..1

```
- The diagram illustrates the relationships between various components of a Scrabble game system. The classes are represented as rectangles, with some having multiple compartments for attributes, operations, and associations. The relationships are as follows:
- Jugador** (Player) is associated with **Integrante** (Member) via a directed association labeled `+ utiliza` with multiplicity `0..1` at the Jugador end.
  - Integrante** is associated with **Game** via a directed association labeled `+ integrante` with multiplicity `0..1` at the Integrante end.
  - Game** is associated with **Ficha** (Token) via a directed association labeled `+ ficha` with multiplicity `0..1` at the Game end.
  - Integrante** is associated with **ComprarVentajasDialog** (BuyAdvantagesDialog) via a directed association labeled `+ jugador` with multiplicity `0..1` at the Integrante end.
  - ComprarVentajasDialog** is associated with **ComandoComprarComodin** (BuyAdvantagesCommand) via a directed association labeled `+ comandoSaltarJugador` with multiplicity `0..1` at the ComprarVentajasDialog end.
  - ComandoComprarComodin** is a specialization (generalization) of the **Command** class, indicated by a hollow triangle arrow pointing from the command to the base class.
  - Integrante** is associated with **AdminTurnos** (AdminTurns) via a directed association labeled `+ utiliza` with multiplicity `0..1` at the Integrante end.
  - AdminTurnos** is associated with **Scrabble** (Game) via a directed association labeled `+ ejecuta` with multiplicity `0..1` at the AdminTurnos end.
  - AdminTurnos** is also associated with **Scrabble** via a directed association labeled `+ adminTurnos` with multiplicity `0..1` at the AdminTurnos end.
  - Scrabble** is associated with **Integrante** via a directed association labeled `+ ficha` with multiplicity `0..1` at the Scrabble end.
  - Scrabble** is associated with **ControllerLocal** via a directed association labeled `+ scrabble` with multiplicity `0..1` at the Scrabble end.
  - ControllerLocal** is a specialization (generalization) of the **Controller** class, indicated by a hollow triangle arrow pointing from the controller to the base class.
  - ControllerLocal** is associated with **ComprarVentajasDialog** via a directed association labeled `+ controllerLocal` with multiplicity `0..1` at the ControllerLocal end.

### 3.35 Saltar a un jugador a cambio de monedas

|                                                                                                |                       |
|------------------------------------------------------------------------------------------------|-----------------------|
| ID:H_35                                                                                        | Usuario:Jugador       |
| Nombre historia:Saltar al jugador siguiente a cambio de monedas                                |                       |
| Prioridad: Baja                                                                                | Estado: Finalizado    |
| Puntos estimados: 1                                                                            | Iteración asignada: 3 |
| Programador responsable: Tania Romero Segura                                                   |                       |
| Descripción:<br><br>Como jugador quiero poder saltar al jugador siguiente a cambio de monedas. |                       |

#### Explicación

Una de las ventajas que pensamos que un jugador podría querer comprar es la de cambiar saltar el turno del jugador siguiente. Pensamos que un precio medianamente decente para esta ventaja son 3 monedas.

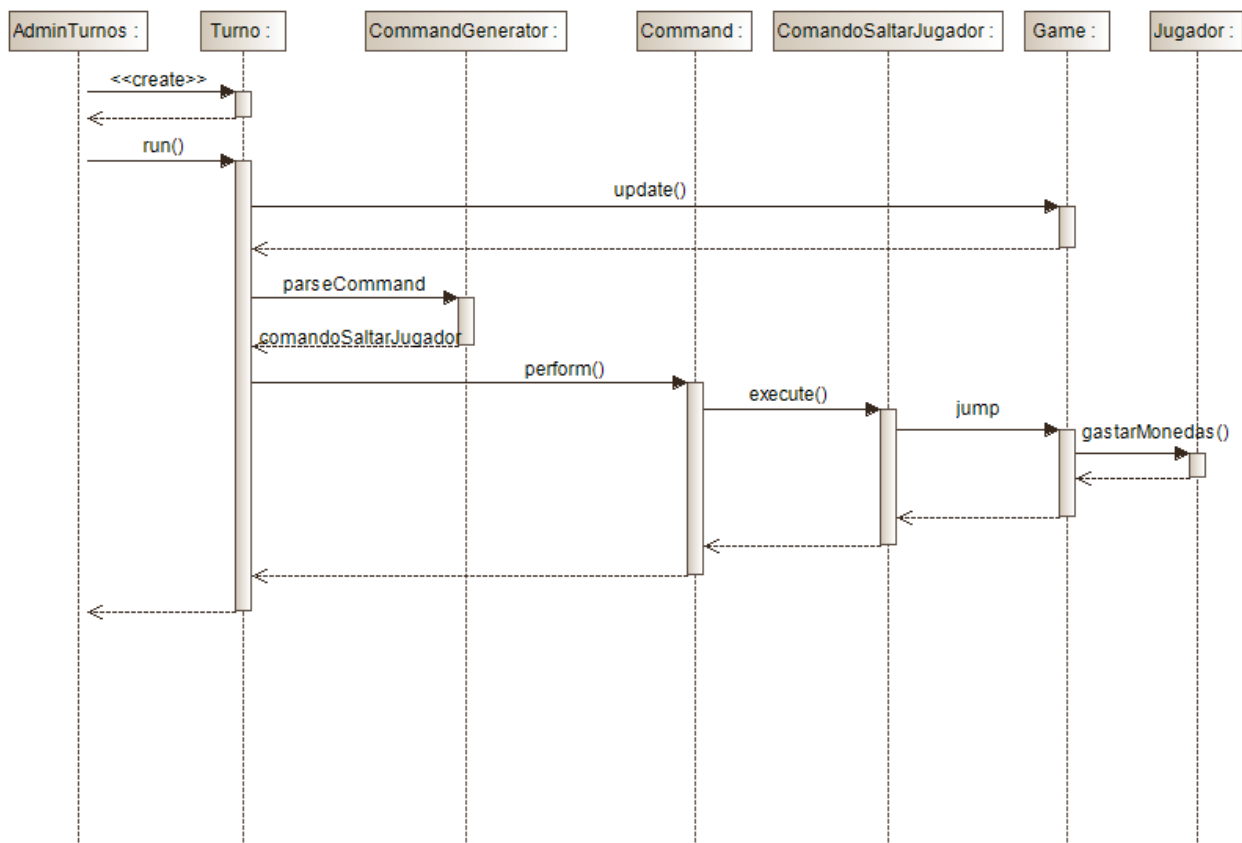
Para implementar esta funcionalidad era necesario crear un nuevo comando al que llamamos ComandoSaltarJugador. Para ejecutar este comando ha habido dos sistemas: uno antes del refactor del modelo y la actual.

Antes de la refactorización del modelo el execute del comando llamaba al método del gamejump() que llamaba al método gastarMonedas del jugador (si el jugador no tiene suficientes monedas se lanza excepción) y después cambiaba el valor del atributo booleano saltar a true.

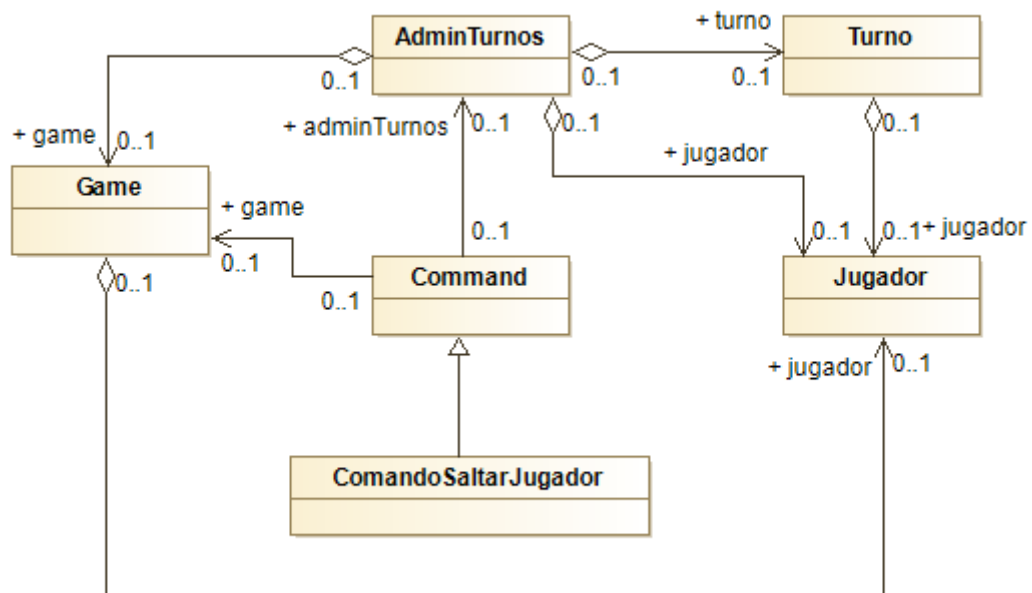
Ahora, cuando se ejecuta un comando pasa por una serie de pasos previos. Cuando se quiere ejecutar un comando se llama al runCommand del controller, este llama al runCommand de la clase Scrabble y después ésta ejecuta el runCommand del AdminTurnos. En este último método runCommand se llama al execute del comando. El execute después hace esencialmente lo mismo que hacía el antiguo método del gamejump.

#### Diagramas

- Diagrama de secuencia anterior

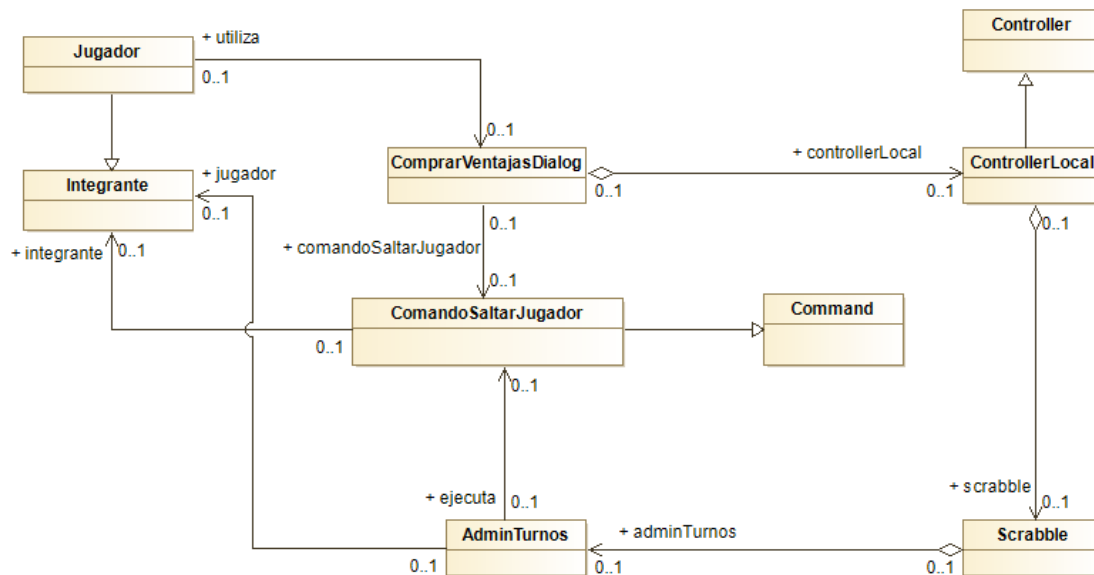


- Diagrama de clases anterior



- Diagrama de secuencia actual

- Diagrama de clases actual



### 3.36 Invertir el orden de jugadores a cambio de monedas

|                                                                                                 |                       |
|-------------------------------------------------------------------------------------------------|-----------------------|
| ID:H_36                                                                                         | Usuario:Jugador       |
| Nombre historia:Revertir el orden de jugadores a cambio de monedas                              |                       |
| Prioridad: Baja                                                                                 | Estado: Finalizado    |
| Puntos estimados: 2                                                                             | Iteración asignada: 3 |
| Programador responsable: Tania Romero Segura                                                    |                       |
| Descripción:<br><br>Como jugador quiero poder cambiar el orden de jugadores a cambio de monedas |                       |

#### Explicación

Una de las ventajas que pensamos que un jugador podría querer comprar es la de invertir el orden de turnos. Pensamos que un precio medianamente decente para esta ventaja son 5 monedas.

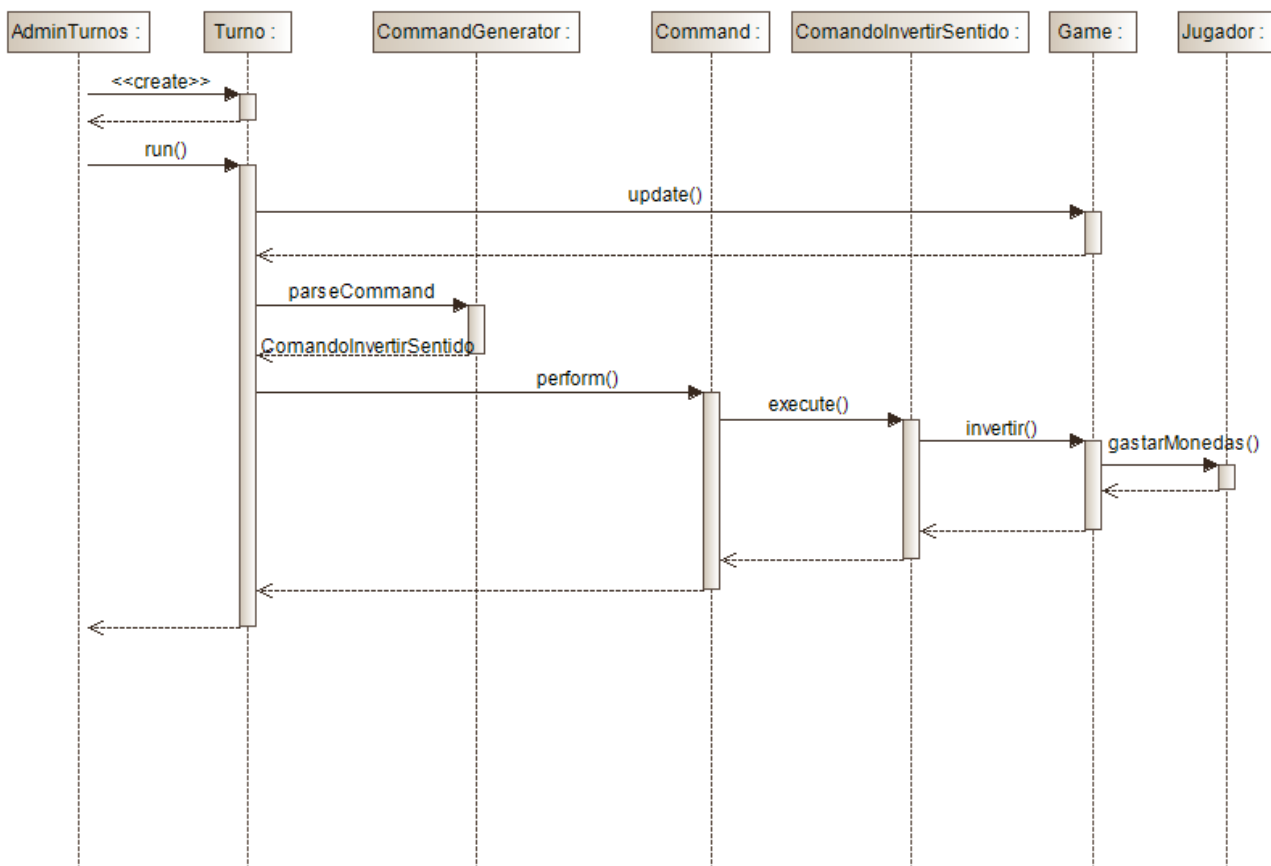
Para implementar esta funcionalidad era necesario crear un nuevo comando al que llamamos ComandoInvertirSentido. Para ejecutar este comando ha habido dos sistemas: uno antes del refactor del modelo y la actual.

Antes de la refactorización del modelo el execute del comando llamaba al método del gamejump() que llamaba al método gastarMonedas del jugador (si el jugador no tiene suficientes monedas se lanza excepción) y después cambiaba el valor del atributo booleano invertir al opuesto al que estaba.

Ahora, cuando se ejecuta un comando pasa por una serie de pasos previos. Cuando se quiere ejecutar un comando se llama al runCommand del controller, este llama al runCommand de la clase Scrabble y después ésta ejecuta el runCommand del AdminTurnos. En este último método runCommand se llama al execute del comando. El execute después hace esencialmente lo mismo que hacía el antiguo método del game invertir.

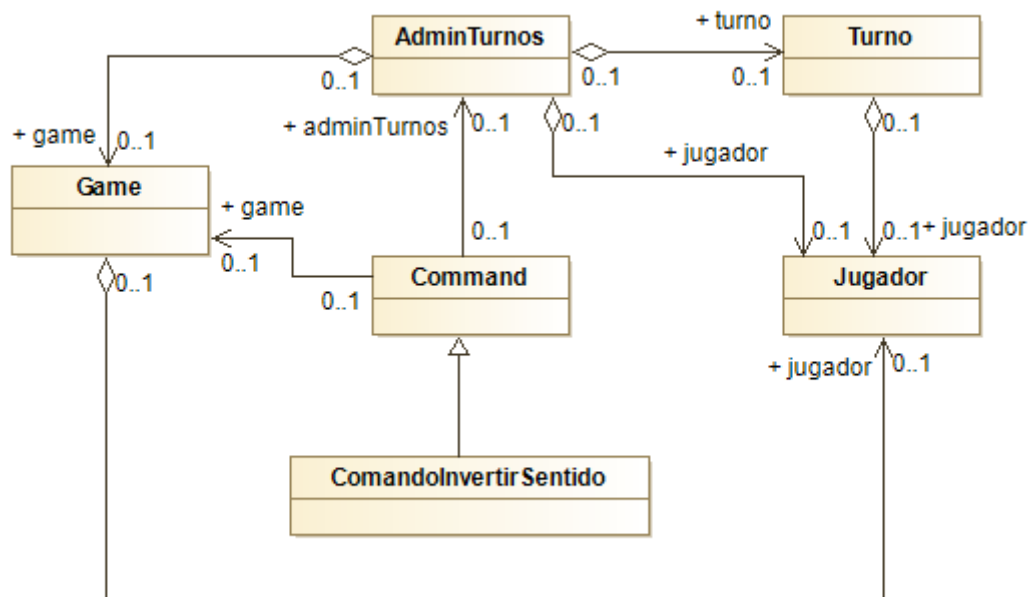
## Diagramas

- Diagrama de secuencia anterior

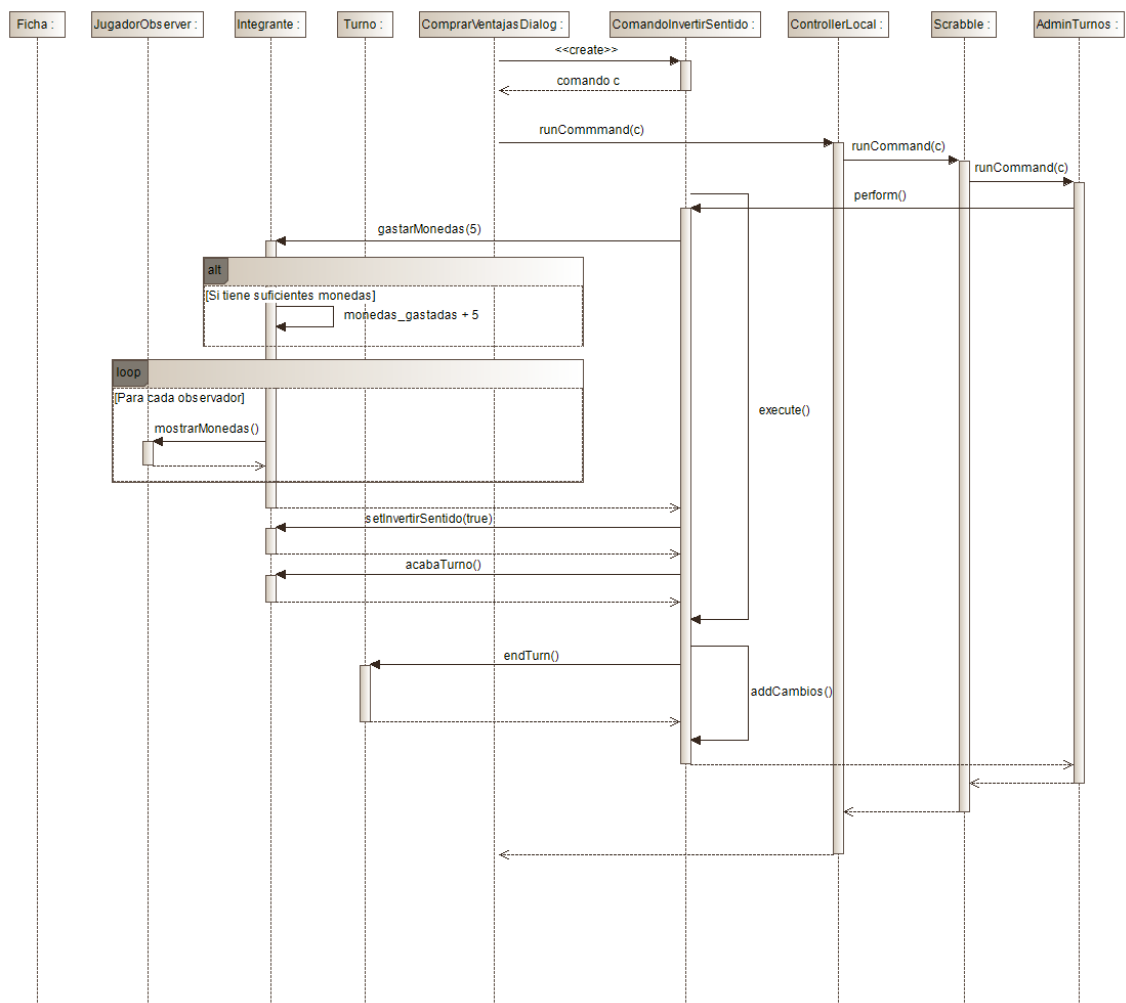


- Diagrama de clases anterior



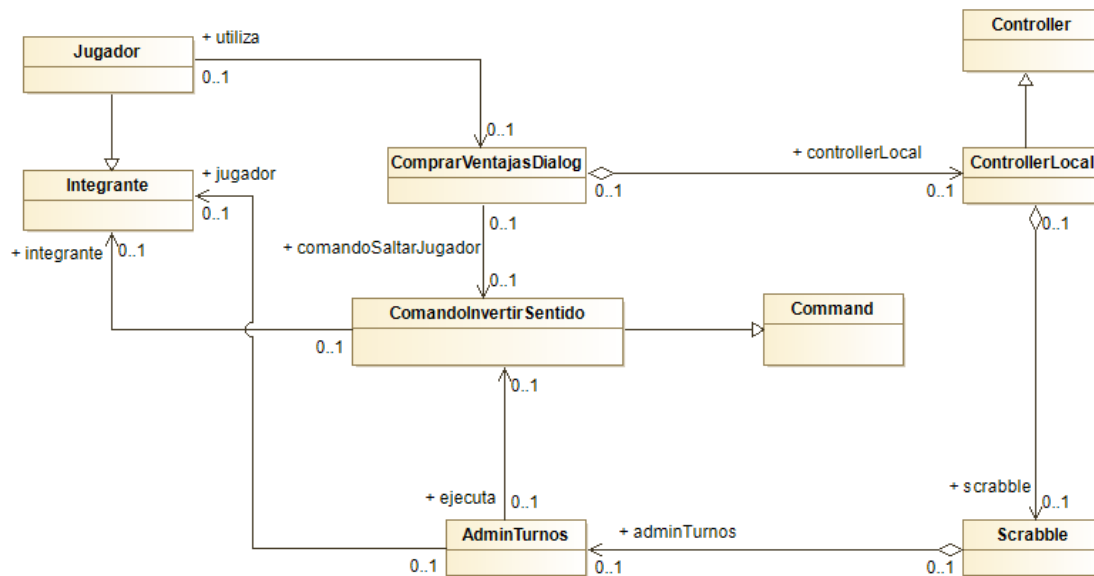


- Diagrama de secuencia actual



[Click para ampliar](#)

- Diagrama de clases actual



### 3.37 Poder ver la mano

|                                                                                                |                 |                       |  |
|------------------------------------------------------------------------------------------------|-----------------|-----------------------|--|
| ID:H_37                                                                                        | Usuario:Jugador |                       |  |
| Nombre historia:Poder ver mi conjunto de fichas en la GUI                                      |                 |                       |  |
| Prioridad: Alta                                                                                |                 | Estado:               |  |
| Puntos estimados: 3                                                                            |                 | Iteración asignada: 4 |  |
| Programador responsable: María Cristina Alameda Salas                                          |                 |                       |  |
| Descripción:<br><br>Como Jugador quiero poder ver las fichas que tengo en mi mano desde la GUI |                 |                       |  |

## Explicación

Esta historia fue una de las primeras que se realizó con respecto a la GUI. El diseño de esta historia no ha cambiado a penas a lo largo del proyecto.

Para llevar a cabo esta funcionalidad, se creó la clase “**PanelMano**” que extiende a JPanel. En este panel se mostraría las fichas del jugador.

En la primera versión, esta clase implementaba la interfaz de observadores del modelo “GameObserver”. Después, nos fuimos dando cuenta de que esta interfaz tenía demasiados métodos, además de que todas las clases que fueran observadoras del modelo (bastantes) tenían que implementarlos cuando la mayoría solo necesitaba unos pocos métodos para funcionar. Así, siguiendo el principio de diseño software “**Principio de segregación de Interfaces**”, dividimos las funcionalidades de la interfaz GameObserver en varias interfaces.

Actualmente, esta clase implementa la interfaz de observadores “JugadorObserver” y por tanto deberá implementar sus métodos, estos están relacionados con acciones realizadas sobre el jugador, entre ellos el añadir una ficha a la mano o borrar una ficha de esta.

Siguiendo el patrón modelo vista controlador, esta clase se registra como observadora del modelo a través del controller que recibe como parámetro.

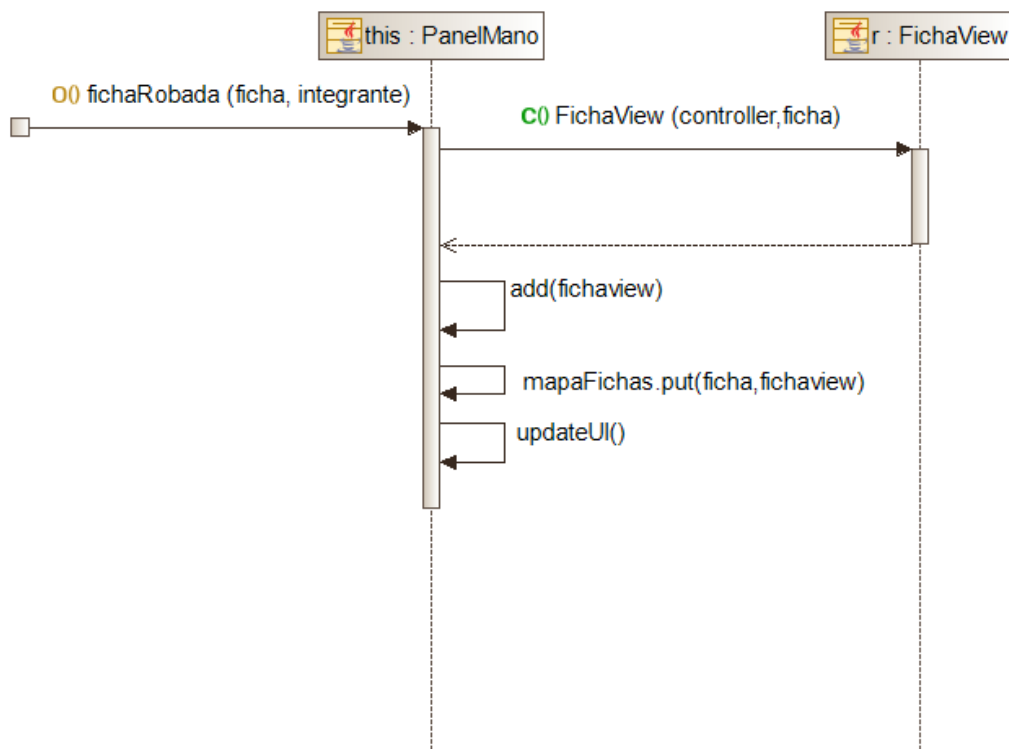
También tiene como atributos el nombre del jugador (requerido para añadirse como observador de jugadores), y un HashMap de clave una Ficha y de valor una FichaView. De esta manera, al asociar una ficha a una FichaView contenida en el JPanel, facilita la búsqueda y eliminación de una FichaView.

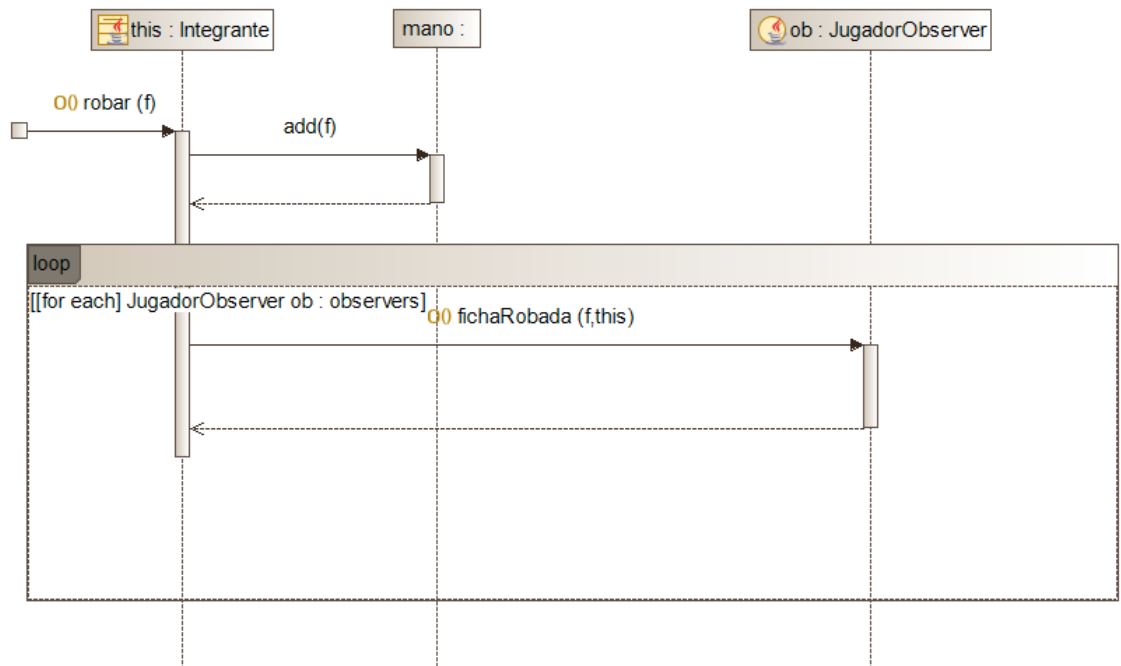
La segunda clase que se creó fue “**FichaView**”. Esta clase representa la interfaz de una ficha y extiende a JLabel. En la constructora recibe el controlador para poder ejecutar los comandos pertinentes y a una ficha, a partir de su letra y su puntuación genera una imagen para mostrar la representación gráfica de la ficha.

## Diagramas

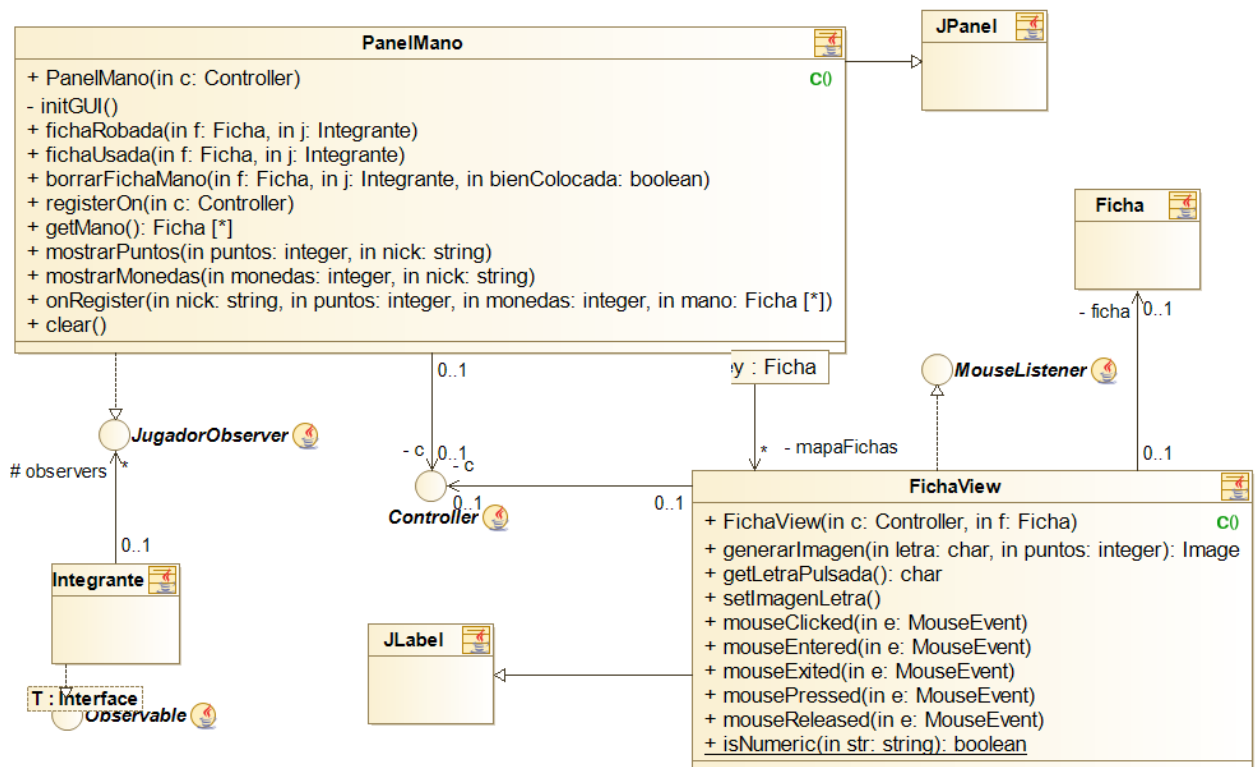
- Diagrama de secuencia

\*El segundo diagrama concreta el método fichaRobada del PanelMano.





- Diagrama de clases



### 3.38 Poder ver el tablero

|                                                                                                    |                       |
|----------------------------------------------------------------------------------------------------|-----------------------|
| ID:H_38                                                                                            | Usuario:Jugador       |
| Nombre historia:Poder ver el tablero en la GUI                                                     |                       |
| Prioridad: Alta                                                                                    | Estado:               |
| Puntos estimados: 5                                                                                | Iteración asignada: 4 |
| Programador responsable: Guillermo García Patiño Lenza                                             |                       |
| Descripción:<br><br>Como Jugador quiero poder ver el tablero, y las fichas que contiene, en la GUI |                       |

### Explicación

Desde el primer momento, el tablero se **implementó como una matriz de casillas** que contenían fichas. Esta representación ha sufrido **pocos cambios** durante el desarrollo del proyecto, y se ha mantenido hasta la release final.

Lo que sí **ha cambiado** durante el desarrollo es la manera en la que se representa la información **de cara al usuario**. **Primero** consistía en una **cadena de texto** que formaba una **cuadrícula** en la que cada celda se mostraba la ficha que había en la correspondiente celda de la matriz del programa. **Ahora**, en la versión final, el tablero se muestra al jugador a través de una **ventana** que también **contiene una cuadrícula** que también muestra en cada celda la ficha que contiene, y en caso de estar vacía, muestra qué modificador de puntos tiene esa casilla.

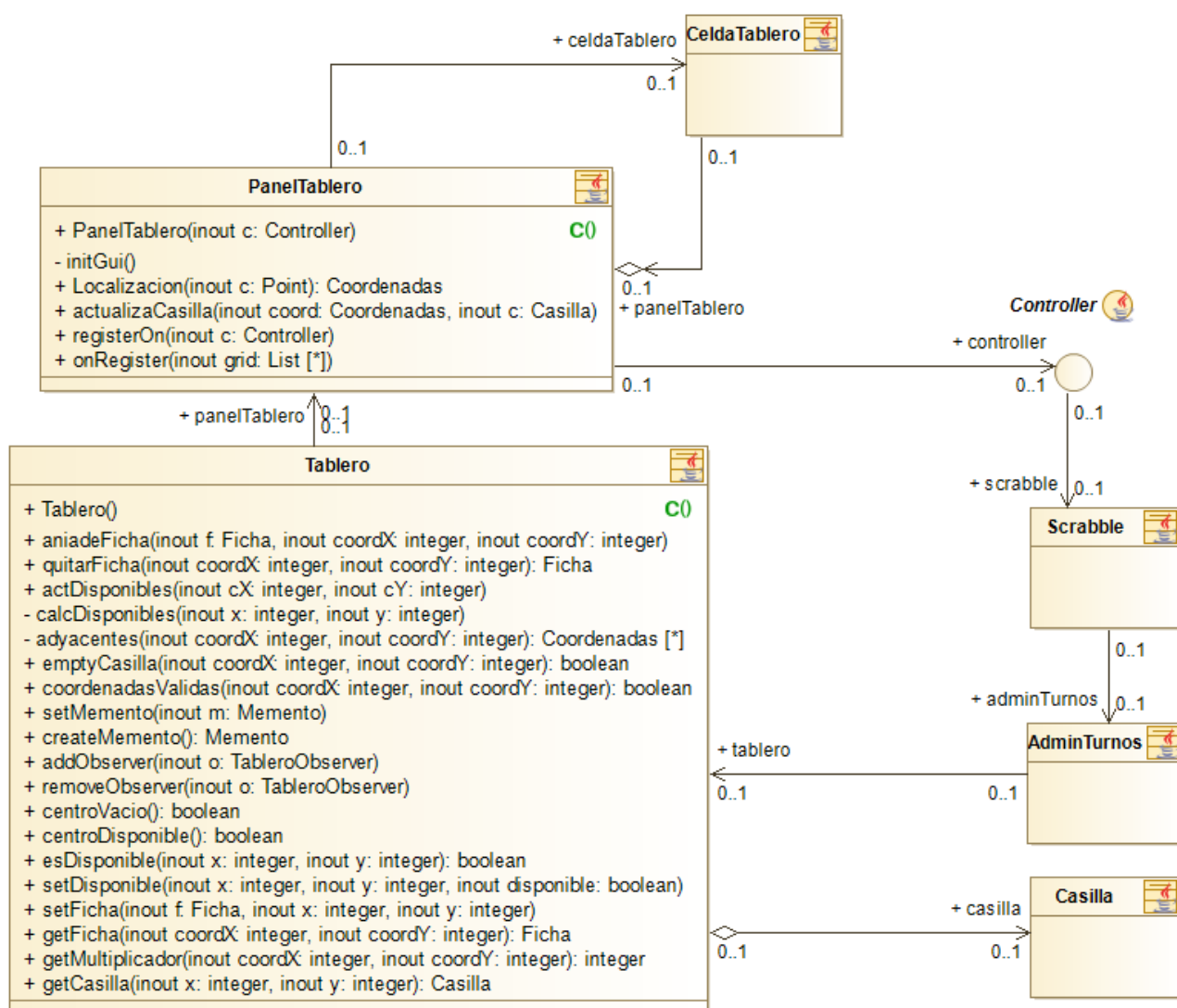
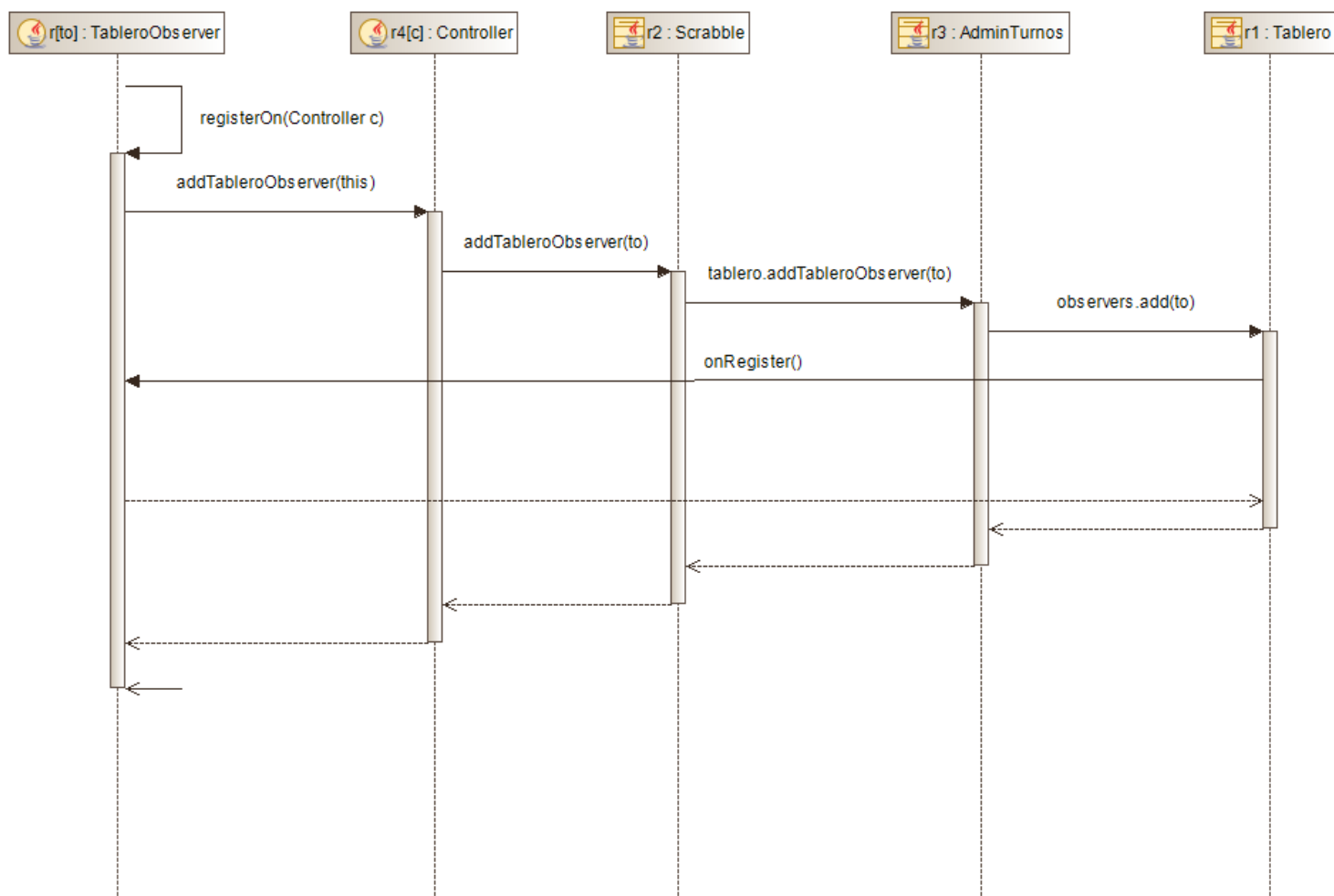
También tenemos claro qué **mejoras** podríamos aplicar a esta parte del proyecto:

- **Abstraer el cálculo de los puntos** que proporciona una casilla con un patrón polimorfismo para hacerlo más sencillo.
- **Empaquetar el Tablero y las Casillas en un mismo componente** y darle un patrón fachada para simplificar las comunicaciones de estos elementos con el resto del programa, facilitando así además la reutilización de esta parte del código.
- También podríamos mejorar las **transferencias de información** entre el tablero y sus observadores **empleando transfer objects**.

### Diagramas

(hacer click en las imágenes para verlas más grandes)

- Nuevos:



### 3.38 Cambiar Ficha

|                                                                                                                                                             |                              |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| <b>ID:</b> H_38                                                                                                                                             | <b>Usuario:</b> Jugador      |
| <b>Nombre historia:</b> Cambiar Ficha                                                                                                                       |                              |
| <b>Prioridad:</b> Alta                                                                                                                                      | <b>Estado:</b>               |
| <b>Puntos estimados:</b> 4                                                                                                                                  | <b>Iteración asignada:</b> 3 |
| <b>Programador responsable:</b> Guillermo García Patiño Lenza                                                                                               |                              |
| <b>Descripción:</b><br><br>Como Jugador quiero poder cambiar una de las fichas de mi mano por la siguiente ficha del mazo para poder hacer una mejor jugada |                              |

#### Explicación

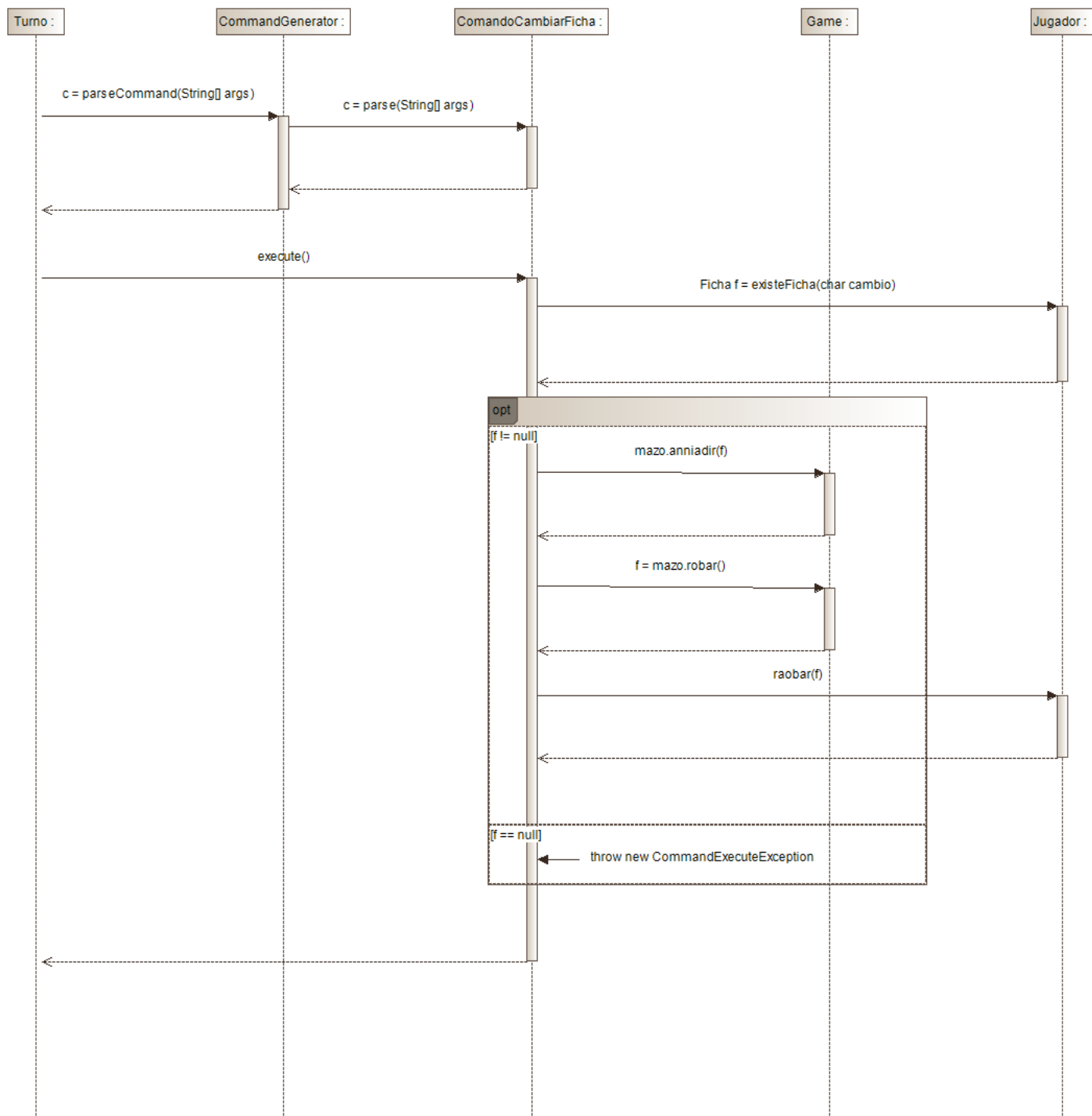
Esta es otra de las funcionalidades que se han implementado mediante comandos. Esta funcionalidad no ha sufrido casi cambios durante el desarrollo. Únicamente se han producido cambios debido al refactor que sufrió el modelo para obtener un diseño más limpio. A continuación se muestran los diagramas anteriores al refactor, los posteriores a él se muestran en la historia homónima que refiere a la GUI.

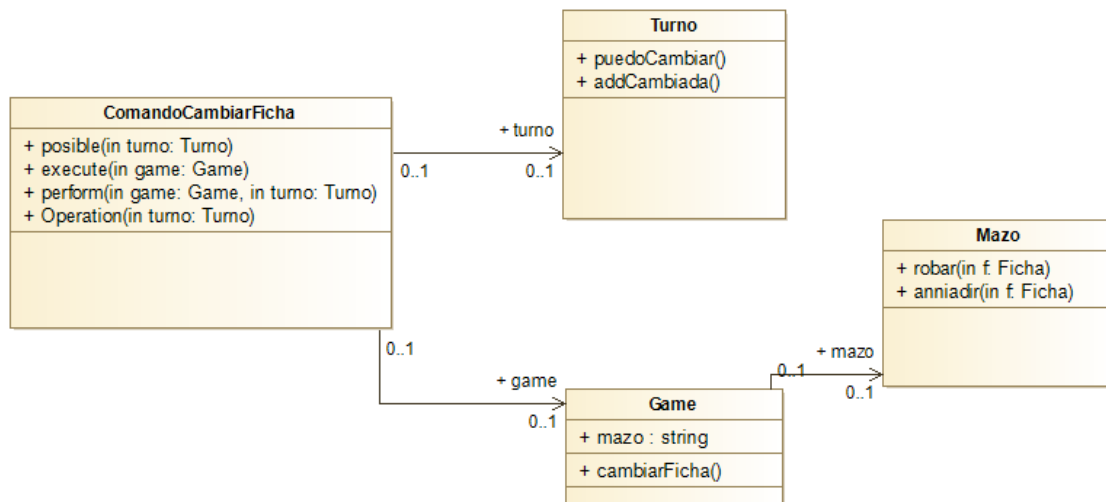
#### Diagramas

- Antiguos:

(hacer click en las imágenes para verlas más grandes)







### 3.39 Poder ver los turnos en la GUI

|                                                                                         |                 |                       |  |
|-----------------------------------------------------------------------------------------|-----------------|-----------------------|--|
| ID:H_39                                                                                 | Usuario:Jugador |                       |  |
| Nombre historia:Poder ver los turnos en la GUI                                          |                 |                       |  |
| Prioridad: Media                                                                        |                 | Estado: Terminado     |  |
| Puntos estimados: 2                                                                     |                 | Iteración asignada: 4 |  |
| Programador responsable: Alejandro Rivera León                                          |                 |                       |  |
| Descripción:<br><br>Como Jugador quiero saber cuál es el estado de los turnos en la GUI |                 |                       |  |

### Explicación

Debido a la inclusión de la GUI se decidió aplicar el **patrón Observador** para **comunicar el modelo con la vista**.

Este patrón es fundamental para comunicar el modelo con las diversas vistas que observan a este.

**Una de los factores que ayudó a que implementar dicho patrón no fuese un problema fue haber optado por una estructura Modelo Vista Controlador** permitiéndonos así utilizar el **controlador como interfaz entre la vista y el modelo**.

**Para mostrar el orden de los turnos** en la vista concretamente **se usaron dos interfaces distintas**.

La primera de ellas, **ModelObserver**, engloba el método que comparten el **resto de interfaces** que forman parte del patrón **Observador**, dicho método es el método **registerOn ()** que sirve para registrar a la vista como observador del modelo a través del controlador que se recibe como parámetro.

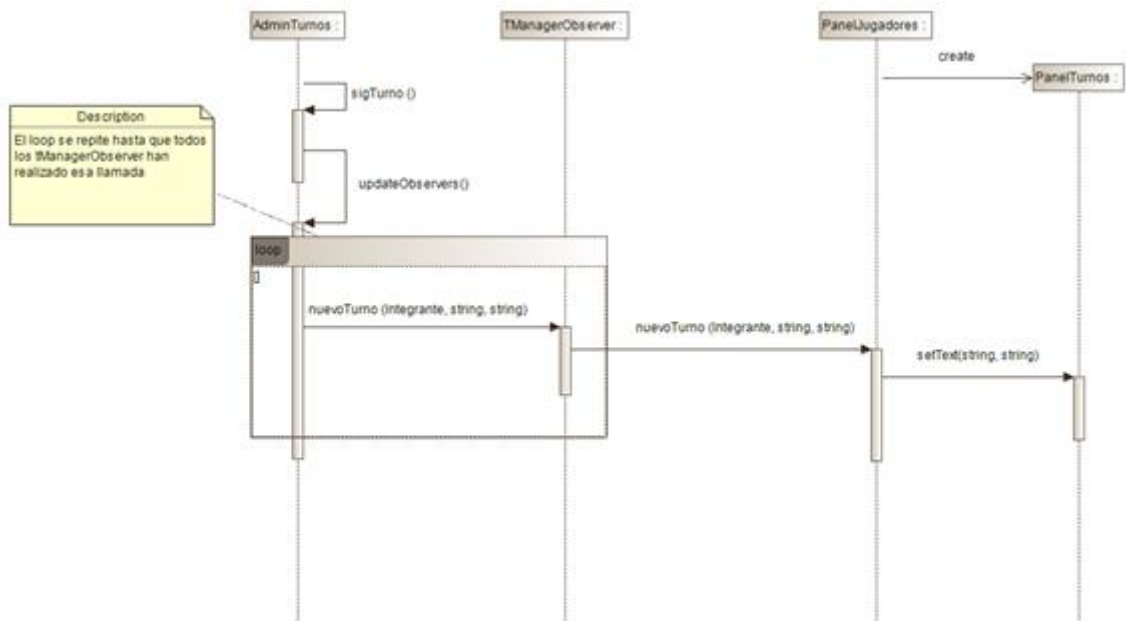
La otra interfaz utilizada para llevar a cabo esta funcionalidad es la interfaz **TManagerObserver** la cual **extiende a ModelObserver** y **es implementada por AdminTurnos**.

Esta interfaz **cuenta con métodos para notificar a la vista cambios en los turnos** y en el jugador que está llevando a cabo su turno.

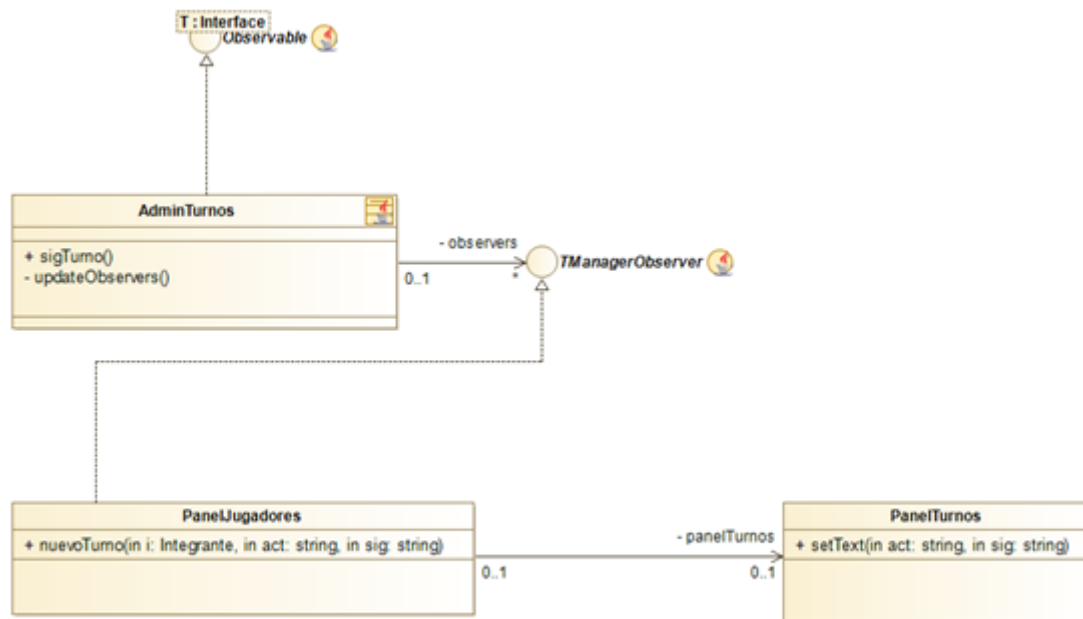
Como se puede apreciar en los diagramas, el funcionamiento es muy sencillo.

Una vez que se termina el turno y AdminTurnos está generando el siguiente lo que se hace es, **empleando** el método **updateObservers ()**, **indicar a todos los observadores** que implementan tManagerObserver **que los turnos han cambiado** y por lo tanto que deben actualizarse en consecuencia mostrando los nuevos valores válidos.

## Diagramas



[Click para verlo más grande.](#)



[Click para verlo más grande.](#)

### 3.40 Mostrar Información

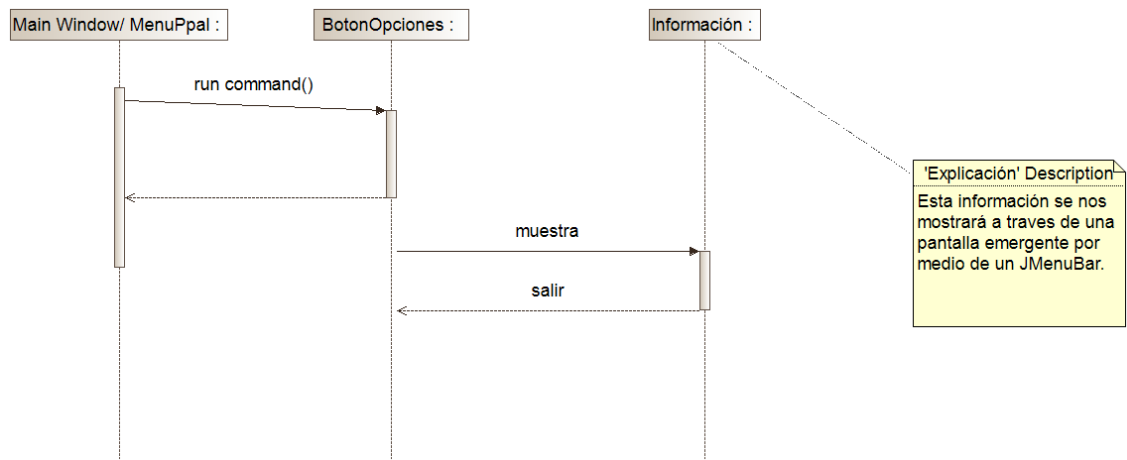
|         |                 |
|---------|-----------------|
| ID:H_53 | Usuario:Jugador |
|---------|-----------------|

|                                                                                                    |                              |
|----------------------------------------------------------------------------------------------------|------------------------------|
| <b>Nombre historia:</b> Poder acceder a las instrucciones de la GUI                                |                              |
| <b>Prioridad:</b> Alta                                                                             | <b>Estado:</b> Finalizado    |
| <b>Puntos estimados:</b> 3                                                                         | <b>Iteración asignada:</b> 5 |
| <b>Programador responsable:</b> David Cruza Sesmero                                                |                              |
| <b>Descripción:</b><br><br>Como Jugador quiero poder ver las instrucciones del juego desde la GUI. |                              |

**Descripción detallada:**

He decidido aunar ambos diagramas puesto que solo se trata de un JMenuBar que se encuentra en dos clases distintas, una de ellas es MenuPPal y la otra MainWindow. Tras elegir la opción de instrucciones te sale un diálogo donde te muestra dicha información.

**Diagrama:**



**3.41 Poder ver los puntos de los Jugadores en la GUI**

|                                                                         |  |
|-------------------------------------------------------------------------|--|
| <b>ID:</b> H_41                                                         |  |
| <b>Nombre historia:</b> Poder ver los puntos de los Jugadores en la GUI |  |
| <b>Prioridad:</b> Media                                                 |  |

|                                                                              |
|------------------------------------------------------------------------------|
| Puntos estimados: 2                                                          |
| Programador responsable: Gema Blanco Núñez                                   |
| Descripción:<br><br>Como Jugador quiero saber los puntos que tengo en la GUI |

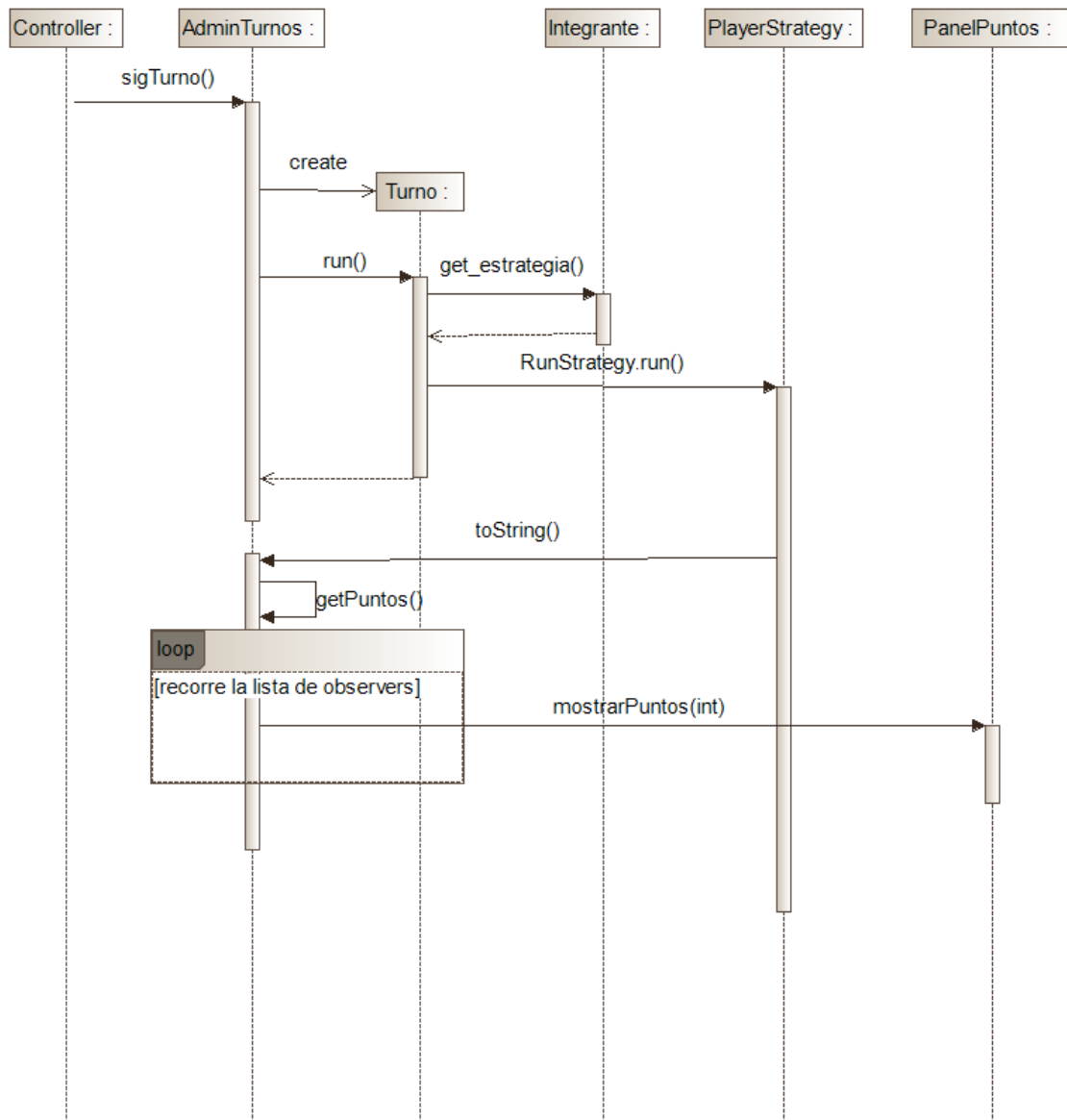
## Explicación

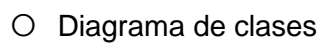
El **objetivo** de esta historia de usuario es mostrar por la GUI los puntos del jugador, que antes se mostraban por consola.

Tomando como base la **explicación para la historia “poder ver los turnos en la GUI” el funcionamiento es el mismo**. La clase AdminTurnos es la que se encarga de notificar a los observadores enviando la información actualizada respecto a las monedas y el **observador que muestra los puntos por la GUI** es el que muestra dicha información.

## Diagramas

- Diagrama de secuencia





|                                                                                                              |                              |
|--------------------------------------------------------------------------------------------------------------|------------------------------|
| <b>ID:</b> H_42                                                                                              | <b>Usuario:</b> Jugador      |
| <b>Nombre historia:</b> Poder ver las monedas de los jugadores en la GUI                                     |                              |
| <b>Prioridad:</b> Media                                                                                      | <b>Estado:</b>               |
| <b>Puntos estimados:</b> 2                                                                                   | <b>Iteración asignada:</b> 4 |
| <b>Programador responsable:</b> Gema Blanco Núñez                                                            |                              |
| <b>Descripción:</b><br><br>Como Jugador quiero saber las monedas que tengo en cualquier momento desde la GUI |                              |

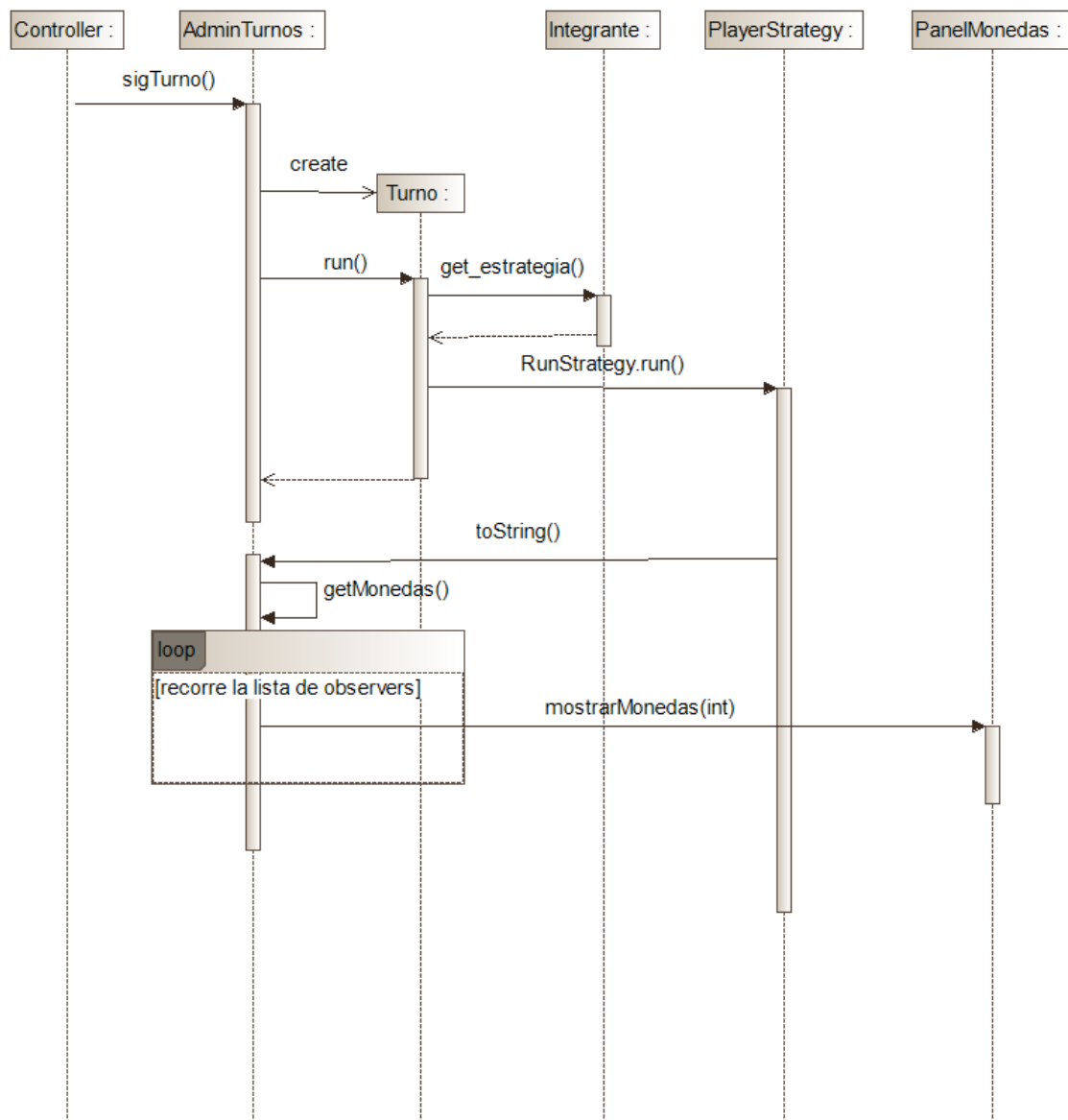
Tomando como base la **explicación para la historia “poder ver los turnos en la GUI” el funcionamiento es el mismo**. La clase AdminTurnos es la que se encarga de notificar a los observadores enviando la información actualizada respecto a los



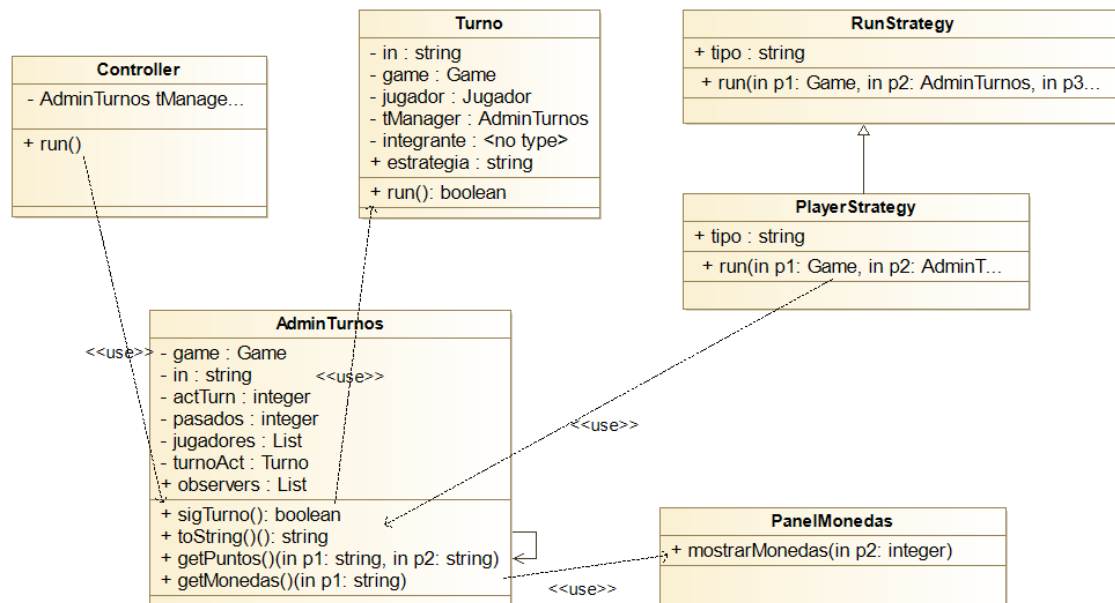
puntos y el **observador que muestra las monedas por la GUI** es el que muestra dicha información.

## Diagramas

- ☐ Diagrama de secuencia



○ Diagrama de clases



### 3.43 Poder cargar una partida desde la GUI

|                                                           |                       |
|-----------------------------------------------------------|-----------------------|
| ID:H_43                                                   | Usuario:Jugador       |
| Nombre historia:Poder cargar una partida desde la GUI     |                       |
| Prioridad: Alta                                           | Estado:               |
| Puntos estimados: 4                                       | Iteración asignada: 4 |
| Programador responsable: Gema Blanco Núñez                |                       |
| Descripción:                                              |                       |
| Como Jugador quiero poder cargar una partida desde la GUI |                       |

### Explicación

El **objetivo** de esta historia de usuario es la **adaptación de la historia de usuario anterior “Cargar partida a medias” al uso de la GUI**, dejando atrás la consola.

Esta historia de usuario **sigue la misma dinámica que la siguiente, “Poder guardar una partida desde la GUI”**, ya que cargar y guardar son dos historias

que van acopladas la una a la otra. Por tanto, para más información consultar la explicación perteneciente a [“Poder guardar una partida desde la GUI”](#).

### 3.43 Poder guardar una partida desde la GUI

|                                                                             |                 |                       |  |
|-----------------------------------------------------------------------------|-----------------|-----------------------|--|
| ID:H_43                                                                     | Usuario:Jugador |                       |  |
| Nombre historia:Poder guardar una partida desde la GUI                      |                 |                       |  |
| Prioridad: Alta                                                             |                 | Estado: Terminado     |  |
| Puntos estimados: 2                                                         |                 | Iteración asignada: 4 |  |
| Programador responsable: Alejandro Rivera León                              |                 |                       |  |
| Descripción:<br><br>Como Jugador quiero poder guardar partida desde la GUI. |                 |                       |  |

### Explicación

Durante este sprint **se llevó a cabo la adaptación del juego a GUI** para **mejorar la experiencia del usuario** y permitirle **interactuar de forma más cómoda con el sistema** por lo que tuvimos que tener esto en cuenta a la hora de realizar la funcionalidad de guardar partida.

Durante el desarrollo del proyecto esta funcionalidad ha sido implementada de varias formas.

**Durante el sprint 4**, para poder guardar y cargar partidas de forma cómoda **optamos por emplear el patrón Memento** ya que de todas las opciones que se nos dieron era la más adecuada.

**A finales del sprint 6** se nos dio a conocer la existencia de **arquitecturas multicapa** proporcionándonos así herramientas para poder mejorar esta funcionalidad **durante el sprint 7**, para ello **se decidió utilizar el patrón DAO**

en lugar del patrón Memento ya que nos permitía obtener los mismos resultados pero además logrando desacoplar ciertas clases y quitándoles responsabilidades durante el proceso.

Toda la información relacionada con la toma de decisiones, evolución, cambios, diagramas (mostrando la evolución de la funcionalidad) y posibles mejoras está explicada con sumo detalle en el siguiente documento ([Documento explicativo](#))

### 3.44 Colocar una ficha en el tablero desde la GUI

|                                                                                            |                 |                       |  |
|--------------------------------------------------------------------------------------------|-----------------|-----------------------|--|
| ID:H_44                                                                                    | Usuario:Jugador |                       |  |
| Nombre historia:Poder colocar una ficha en el tablero desde la GUI                         |                 |                       |  |
| Prioridad: Alta                                                                            |                 | Estado: Terminado     |  |
| Puntos estimados: 3                                                                        |                 | Iteración asignada: 4 |  |
| Programador responsable: María Cristina Alameda Salas                                      |                 |                       |  |
| Descripción:<br><br>Como Jugador quiero poder colocar una ficha en el tablero desde la GUI |                 |                       |  |

### Explicación

Durante este sprint se puso en funcionamiento la creación de la GUI. Así, nos basamos en las historias más básicas, las cuales fuimos implementando poco a poco en la GUI.

Esta historia de usuario está referida a la colocación de una ficha en el tablero.

La implementación de esta funcionalidad fue un reto ya que se nos ocurrió que funcionara de manera que arrastraras la ficha desde el panelMano al panelTablero y se colocara en su sitio.

Primero, miramos información acerca del Drag and Drop, pero como cuando quisimos empezar con la GUI a penas teníamos conocimientos acerca de Swing preferimos hacerlo a nuestra manera.

Se idearon varias maneras de realizar esta acción de arrastrar aunque finalmente se implementó únicamente la última manera pensada.

La idea era sencilla, la clase FichaView debía responder a acciones del mouse, por tanto debía implementar la interfaz MouseListener. En el método mousePressed, el cursor del pc toma la imagen de la ficha a la que se presionó (esto supone que la imagen siempre tomará el mismo tamaño del cursor -> máximo 45x45 pixeles). De esta manera al mantener el botón presionado del mouse, la imagen del cursor cambia a la de la imagen de la fichaView.

El método mouseReleased (referido a soltar el botón del mouse), activa de nuevo la imagen por defecto del cursor. Además, crea un comando colocarFicha (...) con los argumentos necesarios para crearlos (el id\_ficha, y las coordenadas). El obtener las coordenadas correctas del tablero fue bastante complicado.

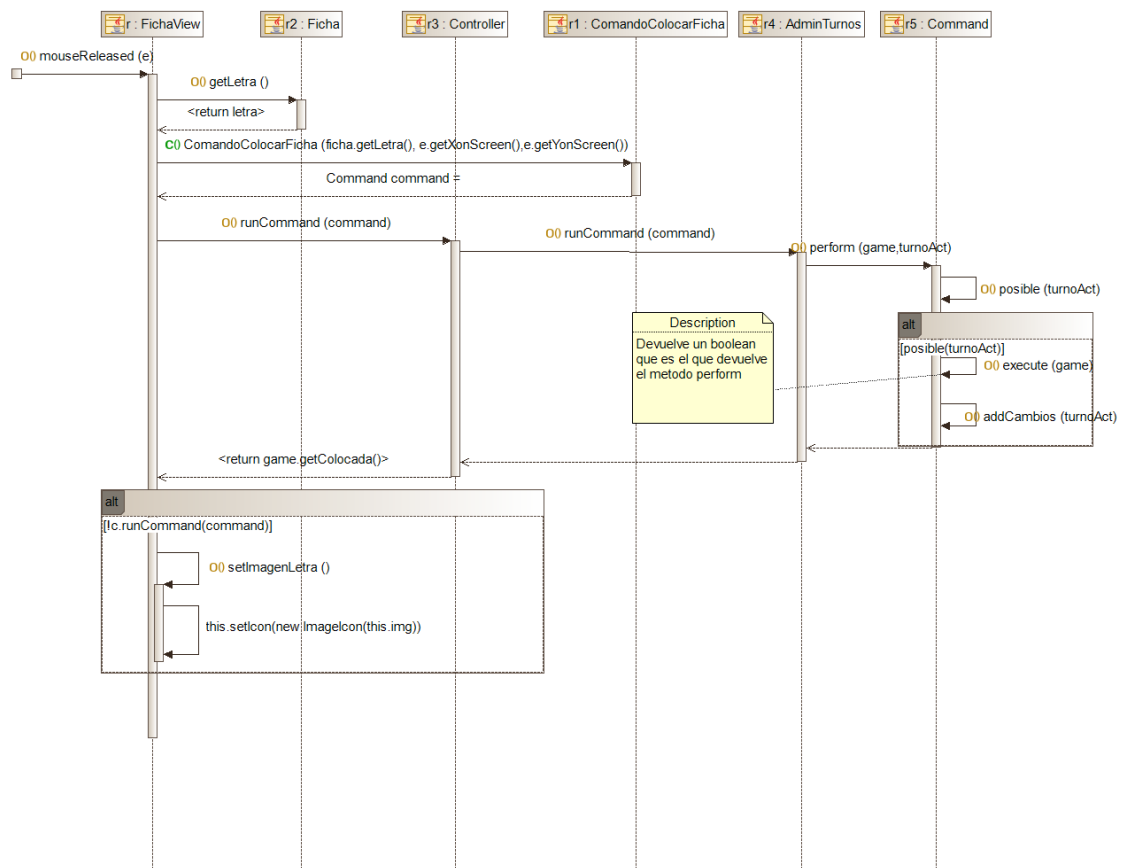
Primeramente, se pasaban al comando colocar ficha las coordenadas del puntero del cursor y en el método ColocarFicha del game se pedía a los observadores las coordenadas a través del método getCoordenadas(). Este método tenía especial importancia en el panel del tablero, que comprobaba para cada una de sus celdas si corresponden a esas coordenadas del puntero. Si no lo fuera devolvía unas coordenadas incorrectas (-1,-1).

Más tarde, se eliminó este método de la interfaz GameObserver y se cambió por una llamada a un método estático del tablero, igual al de getCoordenadas, que se realizaba antes de crear el comando colocarFicha.

Por último, gracias a que la "FichaView" tiene como atributo el controller, se ejecuta el comando a través del método executeCommand del controller.

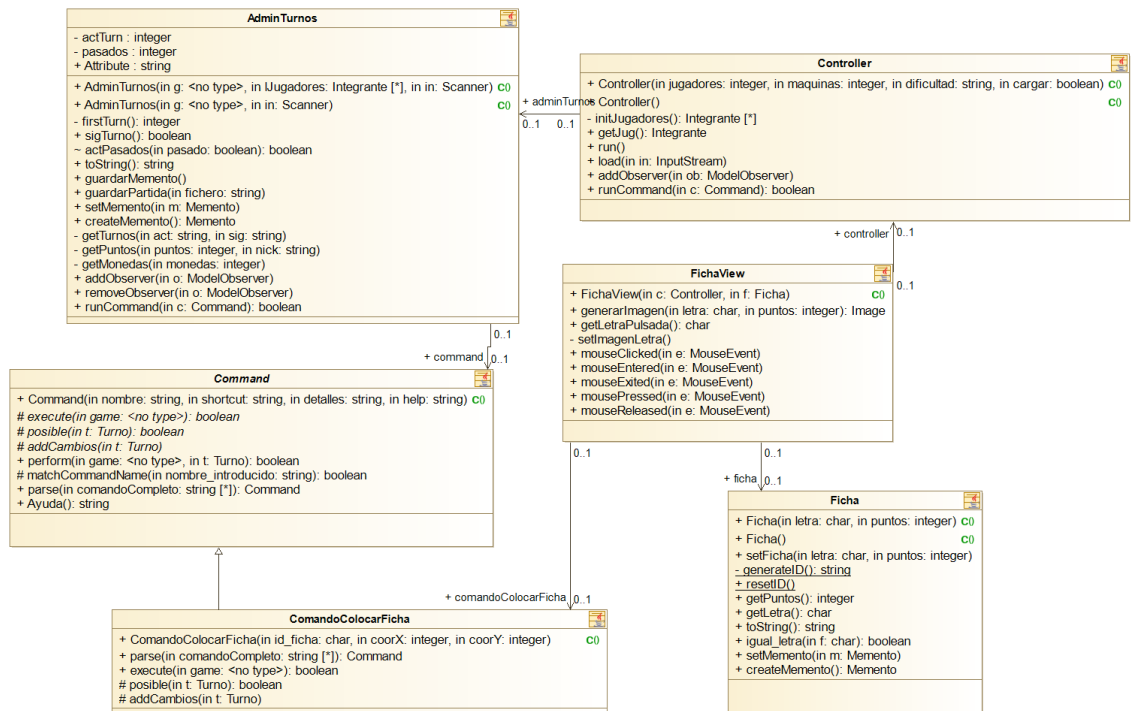
## Diagramas

- Antiguos: cuando pasábamos al comando las coordenadas del cursor
  - Diagrama de secuencia



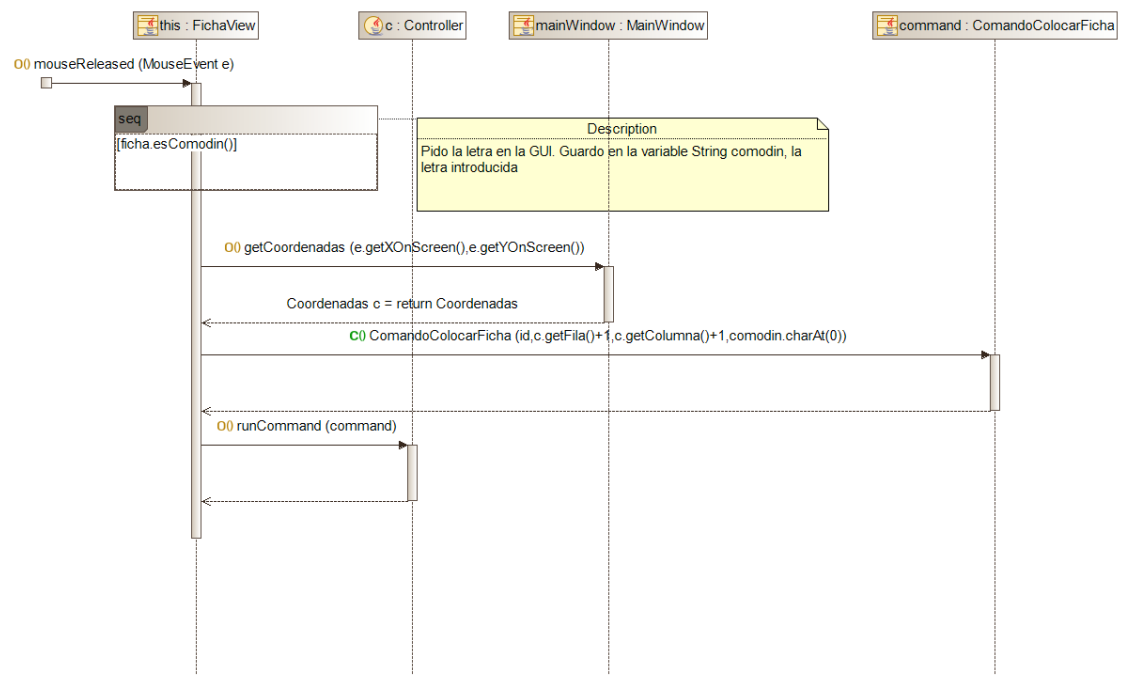
[click para ampliar](#)

○ Diagrama de clases



[Click para ampliar](#)

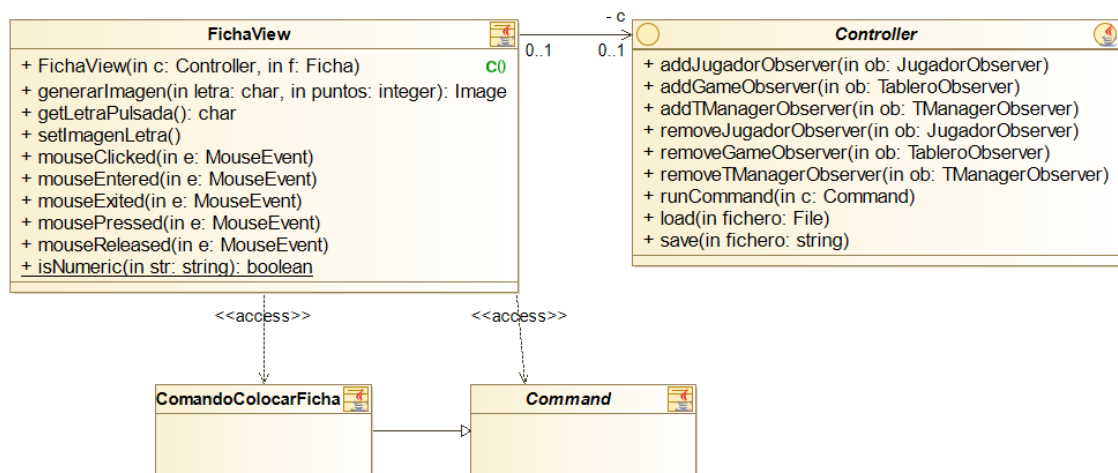
- Nuevos
  - Diagrama de secuencia



[Click para ampliar](#)

- Diagrama de clases





#### 4.45 Poder cambiar una ficha de la mano desde la GUI

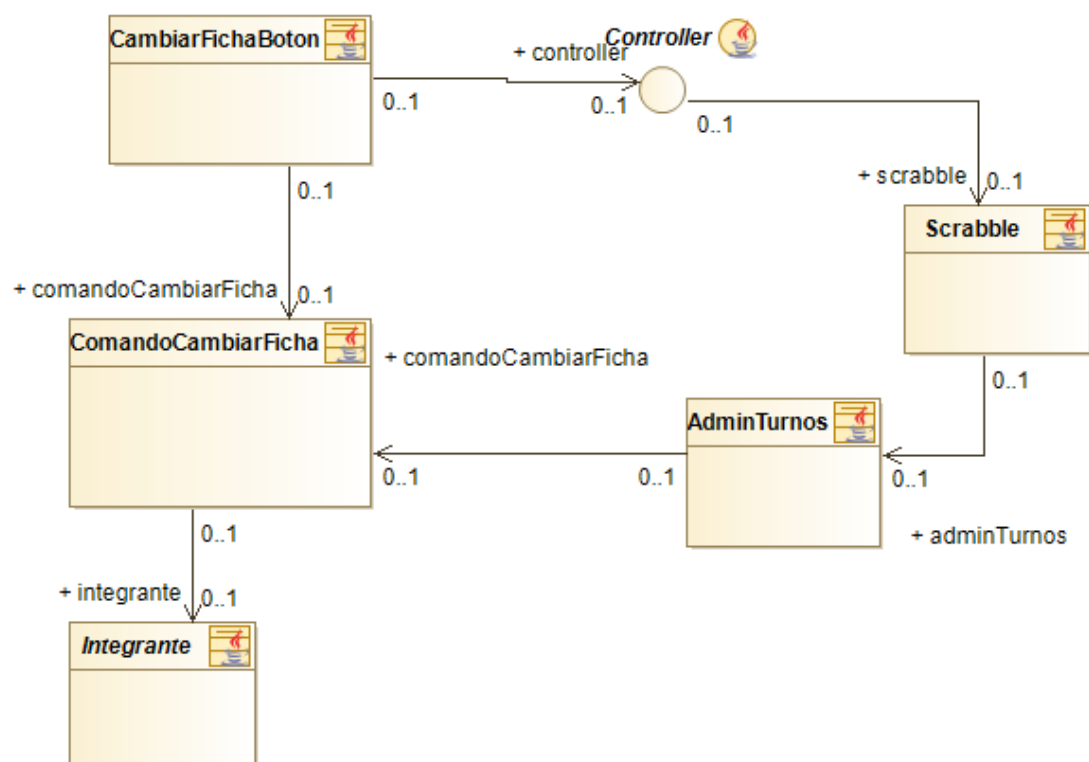
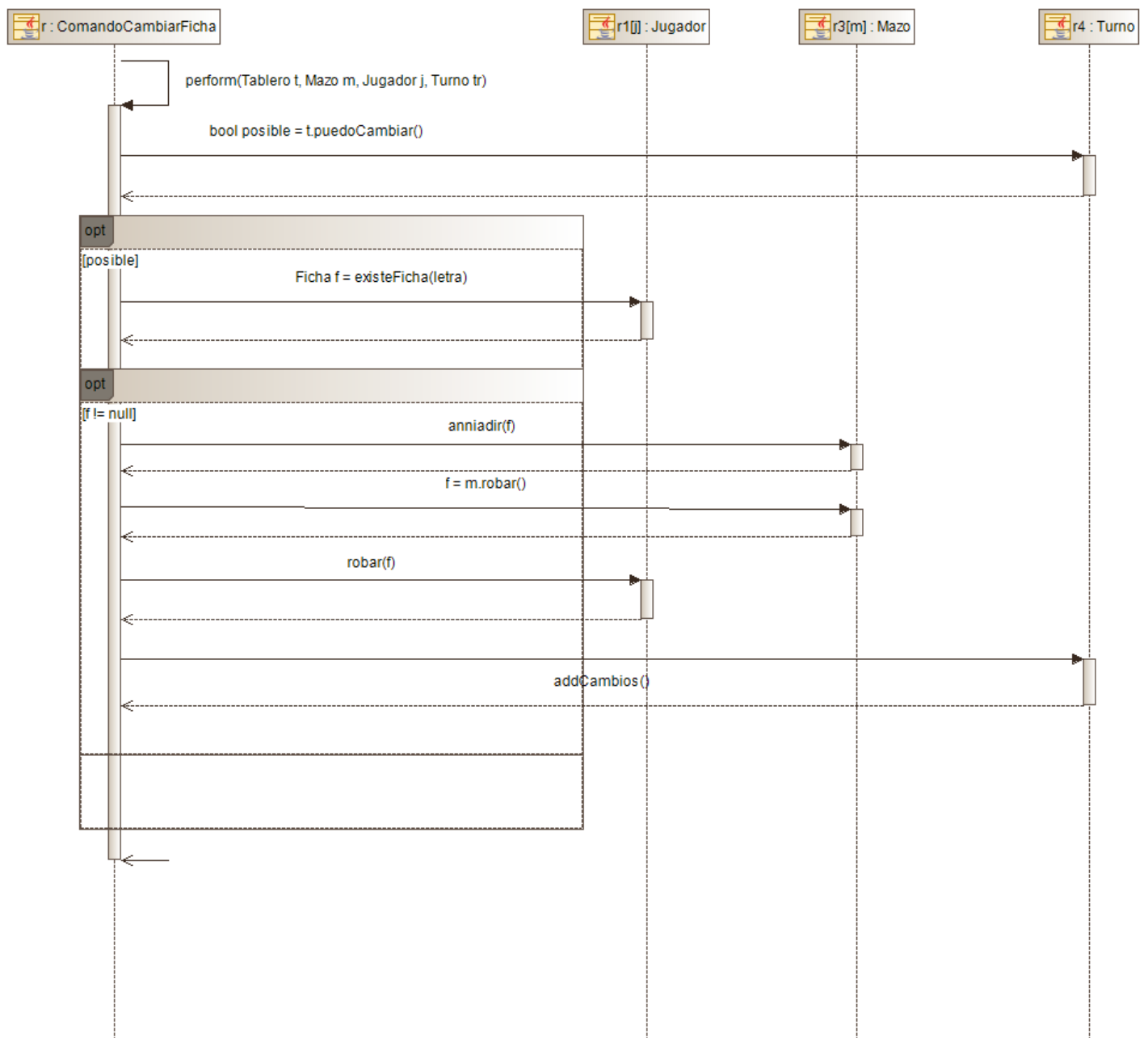
|                                                                                     |                       |
|-------------------------------------------------------------------------------------|-----------------------|
| ID:                                                                                 | Usuario:Jugador       |
| Nombre historia:Poder cambiar una ficha de mi mano desde la GUI                     |                       |
| Prioridad: Alta                                                                     | Estado:               |
| Puntos estimados: 3                                                                 | Iteración asignada: 4 |
| Programador responsable: Guillermo García Patiño Lenza                              |                       |
| Descripción:<br>Como Jugador quiero poder cambiar una ficha de mi mano desde la GUI |                       |

#### Explicación

**Antes**, este comando **se generaba a través del CommandGenerator**, que recibía el texto introducido por consola y lo transformaba en un comando que se ejecutaba sobre el modelo.

**Ahora**, para cambiar una ficha de la mano empleando la GUI, **creamos un botón** que se añade a la ventana y que, al pulsarlo, **pide un carácter empleando un JDialog**, y **emplea ese carácter para crear el comando** correspondiente y enviarlo a través del Controller.

En lo referente a la **ejecución del comando**, esta ha sufrido los **mismos cambios que el resto de comandos**. (Ver ['Refactor Modelo'](#))Diagramas



#### 4.48 Poder jugar con jugadores controlados por la IA

|                                                                                                                                                              |                              |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| <b>ID:</b> H_37                                                                                                                                              | <b>Usuario:</b> Jugador      |
| <b>Nombre historia:</b> Poder jugar con jugadores controlados por IA                                                                                         |                              |
| <b>Prioridad:</b> Alta                                                                                                                                       | <b>Estado:</b>               |
| <b>Puntos estimados:</b> 4                                                                                                                                   | <b>Iteración asignada:</b> 4 |
| <b>Programador responsable:</b> Tania Romero Segura                                                                                                          |                              |
| <b>Descripción:</b><br><br>Como Jugador quiero poder jugar contra un jugador controlado por la máquina en tres niveles de dificultad (fácil, medio, difícil) |                              |

#### Explicación

El profesor nos pidió que incluyéramos jugadores automáticos así que creamos esta historia de usuario para cubrir esa petición. Para la implementación de esta funcionalidad ha habido dos etapas.

En el sprint 4 se hizo una primera versión. Para esta versión se ideó un sistema que obtenía palabras al azar del diccionario y las colocaba en el tablero si era posible concatenar con alguna de las palabras puestas (o directamente si es la primera palabra del juego). Para diferenciar entre niveles se incluyó la posibilidad de colocar más de una palabra y comprar ventajas.

Sin embargo, se decidió hacer una nueva versión para el sprint siguiente para intentar conseguir un funcionamiento que se aproximase más a la forma de actuar de un jugador humano.

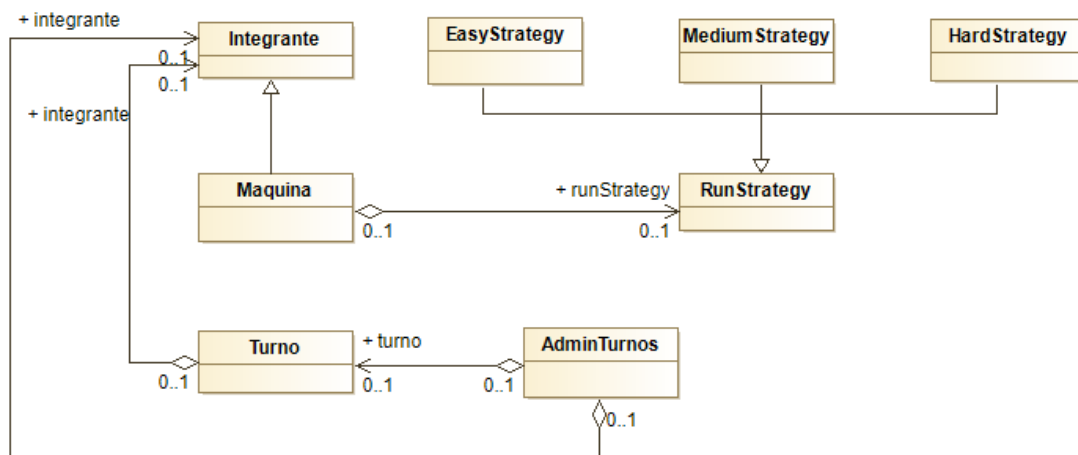
Para ver más detalles sobre la implementación véase el documento RefactorizaciónIA.

#### Diagramas

- Diagrama de secuencia de la segunda fase



- Diagrama de clases de la segunda fase



[Click para ampliar](#)

#### 4.49 Poder pasar turno desde la GUI

|                                                                       |                       |
|-----------------------------------------------------------------------|-----------------------|
| ID:H_49                                                               | Usuario:Jugador       |
| Nombre historia:Poder pasar el turno en la GUI                        |                       |
| Prioridad: Alta                                                       | Estado:               |
| Puntos estimados: 3                                                   | Iteración asignada: 5 |
| Programador responsable: Guillermo García Patiño Lenza                |                       |
| Descripción:<br>Como Jugador quiero poder pasar mi turno desde la GUI |                       |

#### Explicación

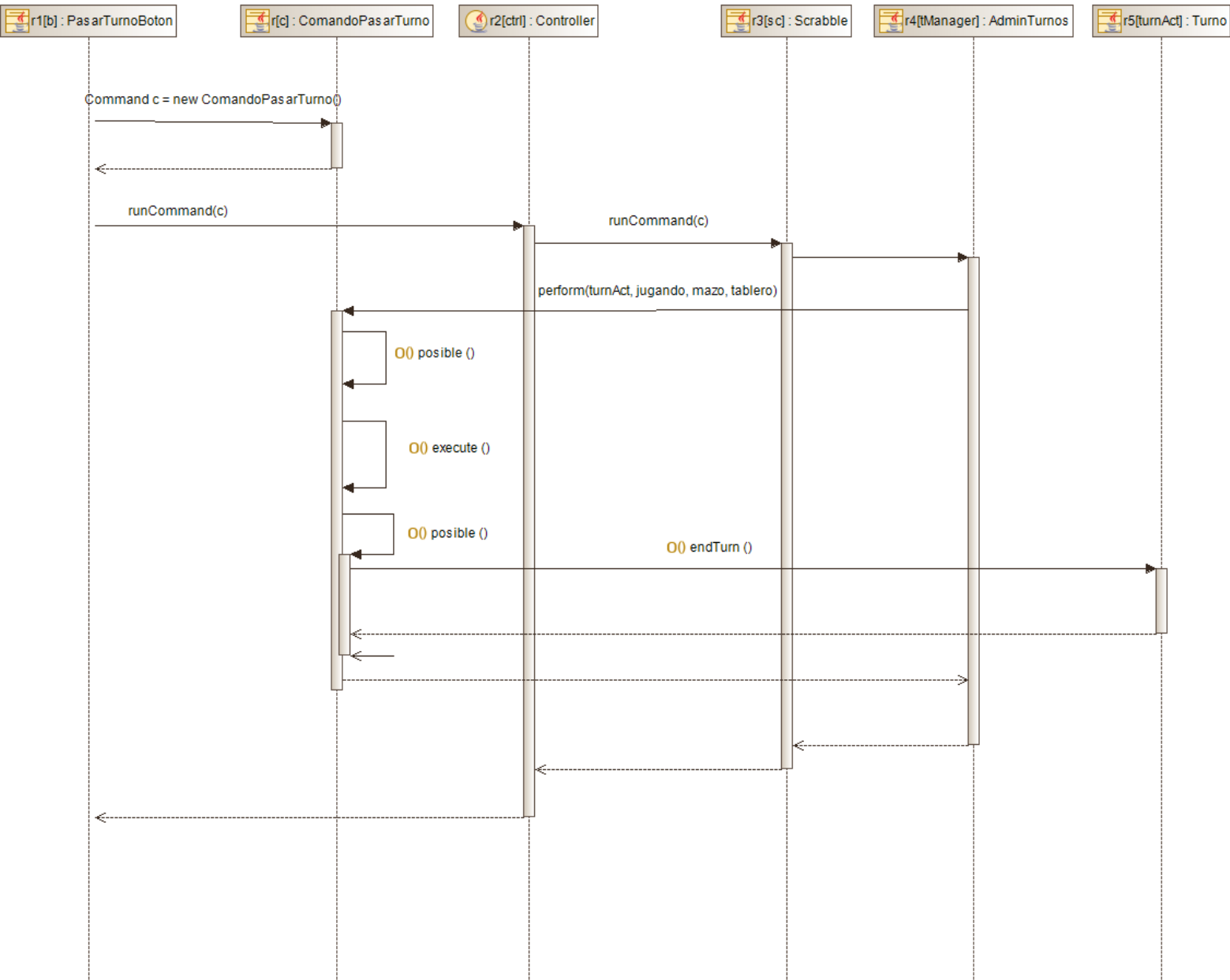
Para poder pasar turnos desde la GUI, una vez teníamos el modelo adaptado al funcionamiento guiado por eventos, no se nos ocurrió otra cosa que **introducir un atributo 'acabado' en el turno** que indicara **si el turno se ha acabado o no**.

Este **atributo** es únicamente **alterado por el comando pasar turno**, que **cambia su valor booleano** a 'true'. Además, el **AdminTurnos consulta este atributo de los turnos tras la ejecución de cada comando** para saber si debe terminar el turno y generar el siguiente. Si al consultarlo, el

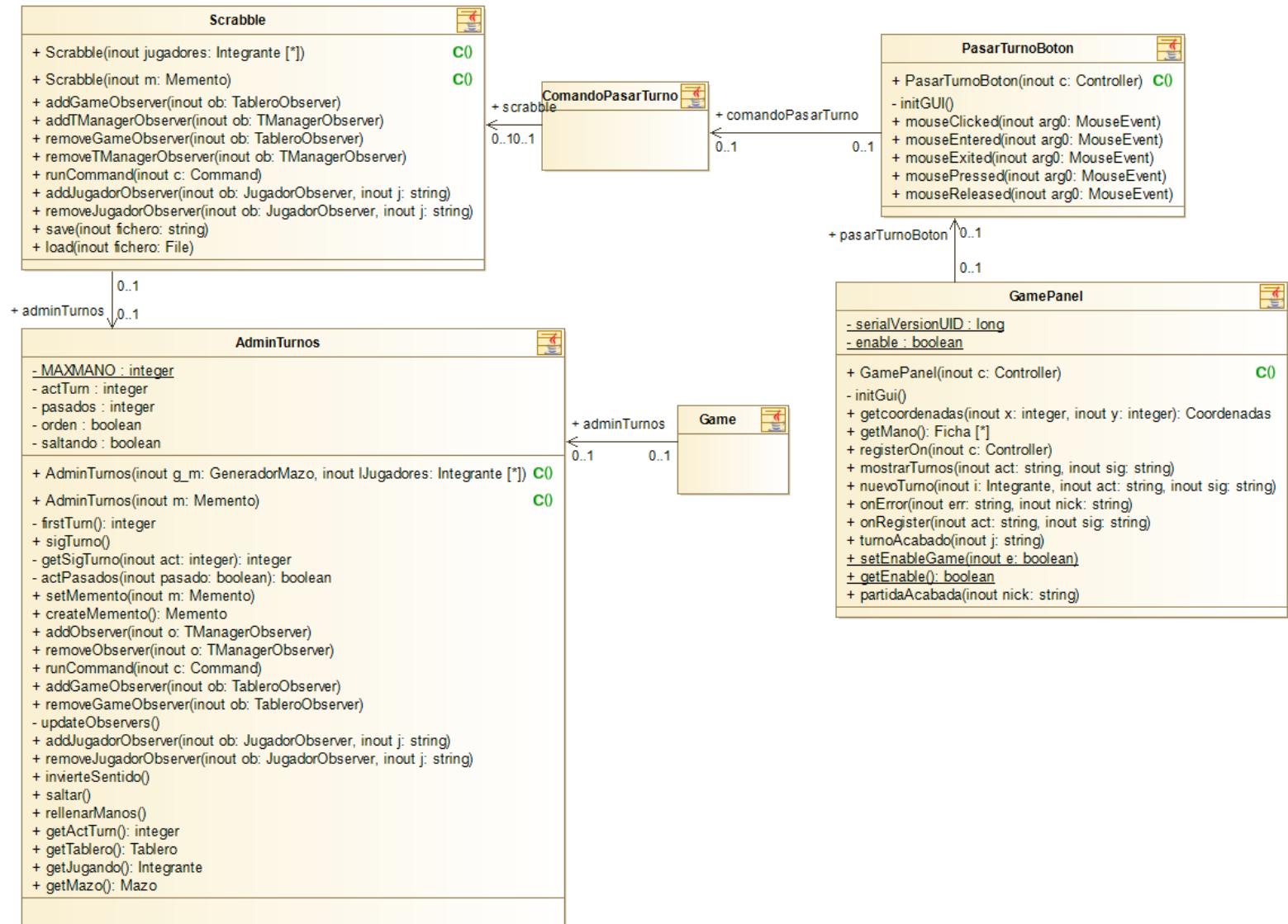
AdminTurnos observa que el **turno se ha acabado** y **dirá al Game que verifique las palabras** que ha puesto el jugador. **Si esa verificación resulta exitosa**, el AdminTurnos **generará un turno para el siguiente jugador**, y si no, pondrá el **atributo 'acabado' a false** y permitirá al **jugador seguir ejecutando comandos**.

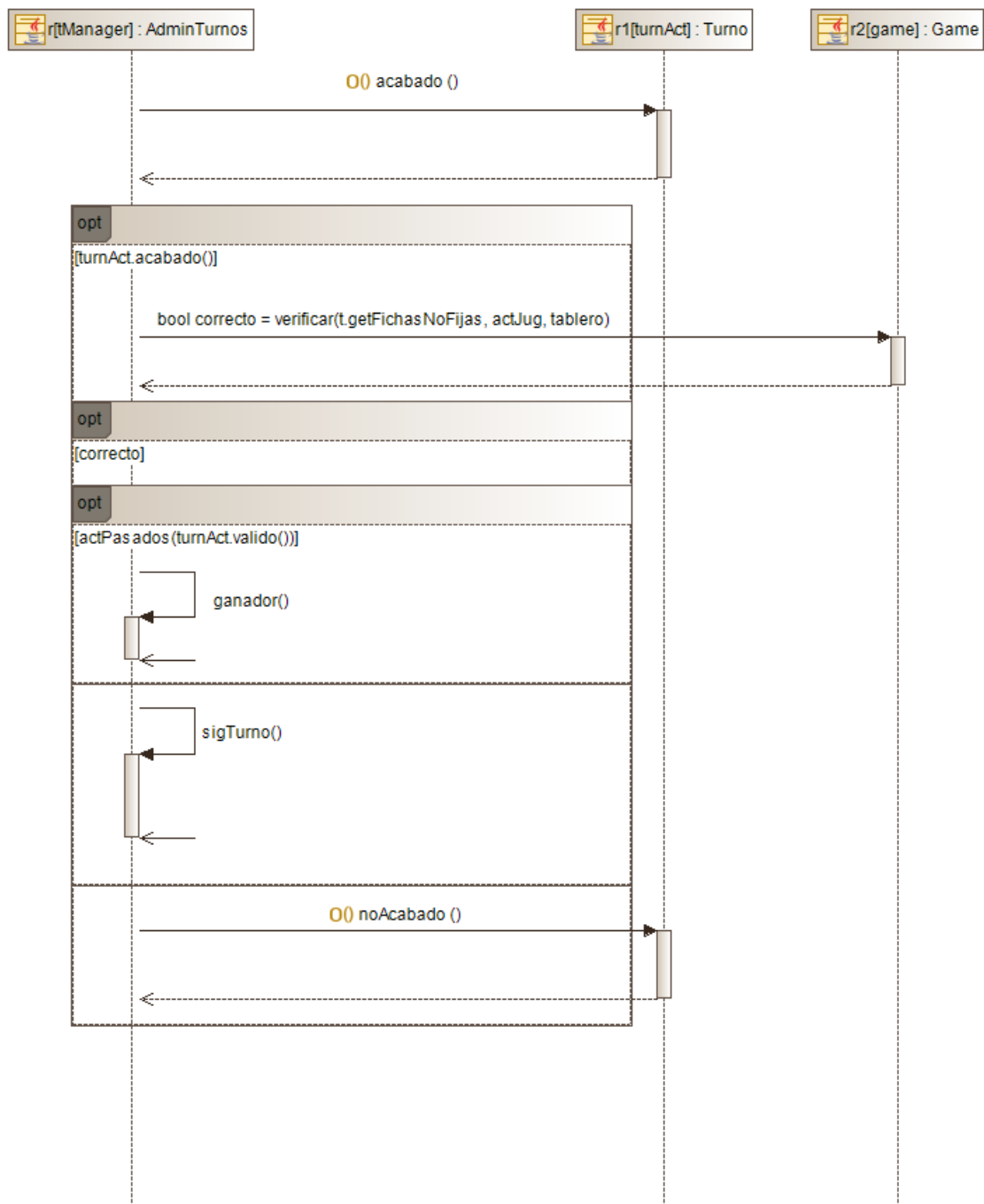
**En vez de comprobar** si el turno se ha finalizado, podríamos haber introducido un **comando especial**, que **no siguiera la estructura del resto** de comandos, para **transmitir la información directamente al AdminTurnos** de que el turno se ha acabado y que éste **entonces** pudiera enviar al **Game** la orden de **verificar las palabras que ha puesto el jugador**. Así **evitaríamos** estar realizando **comprobaciones a cada ejecución** del comando, y sería **menos costoso** para el programa.

## Diagramas









#### 4.50 Poder comprar un comodín en la GUI

|                                                                                                |                       |
|------------------------------------------------------------------------------------------------|-----------------------|
| ID:H_52                                                                                        | Usuario:Jugador       |
| Nombre historia:Poder comprar un comodín en la GUI                                             |                       |
| Prioridad: Alta                                                                                | Estado: Finalizado    |
| Puntos estimados: 3                                                                            | Iteración asignada: 5 |
| Programador responsable: Tania Romero Segura                                                   |                       |
| Descripción:<br><br>Como Jugador quiero poder ver las fichas que tengo en mi mano desde la GUI |                       |

#### Explicación

Esta Historia de Usuario surgió por la necesidad de incluir la funcionalidad de 'p comprar esta ventaja a la nueva interfaz gráfica. Para poder implementarla se creó un botón en la ventana del juego. Al pulsar el botón se abre un diálogo que permite elegir qué ventaja quieres comprar. Una vez se selecciona la opción comprar un comodín se abre una ventana que permite seleccionar la ficha que se quiere sustituir por el comodín. Después se pide la confirmación del jugador y si éste da su aprobación se procede a ejecutar un nuevo comando ComandoComprarComodín llamando al método runCommand() del controller que tiene la ventana como atributo. Después el funcionamiento es el mismo que antes de haber incluido la GUI.

#### Diagramas

Dado que la funcionalidad es la misma una vez se ha decidido ejecutar el comando los diagramas son los mismos que se ven en los diagramas actuales de la Historia de Usuario "Cambiar una ficha por un comodín a cambio de monedas".

#### 4.51 Poder saltar jugador en la GUI

|         |                 |
|---------|-----------------|
| ID:H_50 | Usuario:Jugador |
|---------|-----------------|

|                                                                                                            |                              |
|------------------------------------------------------------------------------------------------------------|------------------------------|
| <b>Nombre historia:</b> Poder saltar jugador en la GUI                                                     |                              |
| <b>Prioridad:</b> Alta                                                                                     | <b>Estado:</b> Finalizado    |
| <b>Puntos estimados:</b> 3                                                                                 | <b>Iteración asignada:</b> 5 |
| <b>Programador responsable:</b> Tania Romero Segura                                                        |                              |
| <b>Descripción:</b><br><br>Como Jugador quiero poder comprar una ventaja para saltar el turno a un jugador |                              |

## Explicación

Esta Historia de Usuario surgió por la necesidad de incluir la funcionalidad de comprar esta ventaja a la nueva interfaz gráfica. Para poder implementarla se creó un botón en la ventana del juego. Al pulsar el botón se abre un diálogo que permite elegir qué ventaja quieres comprar. Una vez se selecciona la opción saltar jugador se pide la confirmación del jugador y si éste da su aprobación se procede a ejecutar un nuevo comando ComandoSaltarJugador llamando al método runCommand() del controller que tiene la ventana como atributo. Después el funcionamiento es el mismo que antes de haber incluido la GUI.

## Diagramas

Dado que la funcionalidad es la misma una vez se ha decidido ejecutar el comando los diagramas son los mismos que se ven en los diagramas actuales de la Historia de Usuario “Saltar al jugador siguiente a cambio de monedas”.

### 4.52 Poder pasar turno desde la GUI

|                                                          |                              |
|----------------------------------------------------------|------------------------------|
| <b>ID:</b> H_51                                          | <b>Usuario:</b> Jugador      |
| <b>Nombre historia:</b> Poder invertir sentido en la GUI |                              |
| <b>Prioridad:</b> Alta                                   | <b>Estado:</b> Finalizado    |
| <b>Puntos estimados:</b> 3                               | <b>Iteración asignada:</b> 5 |
| <b>Programador responsable:</b> Tania Romero Segura      |                              |

**Descripción:**

Como Jugador quiero poder comprar una ventaja para poder invertir el sentido desde la GUI

**Explicación**

Esta Historia de Usuario surgió por la necesidad de incluir la funcionalidad de comprar esta ventaja a la nueva interfaz gráfica. Para poder implementarla se creó un botón en la ventana del juego. Al pulsar el botón se abre un diálogo que permite elegir qué ventaja quieres comprar. Una vez se selecciona la opción invertir sentido se pide la confirmación del jugador y si éste da su aprobación se procede a ejecutar un nuevo comando `ComandoInvertirSentido` llamando al método `runCommand()` del controller que tiene la ventana como atributo. Después el funcionamiento es el mismo que antes de haber incluido la GUI.

**Diagramas**

Dado que la funcionalidad es la misma una vez se ha decidido ejecutar el comando los diagramas son los mismos que se ven en los diagramas actuales de la Historia de Usuario “Invertir el orden de jugadores a cambio de monedas”.

**4.54 Poder jugar en red con otros jugadores**

|                                                         |                  |                       |  |
|---------------------------------------------------------|------------------|-----------------------|--|
| ID: H_54                                                | Usuario: Jugador |                       |  |
| Nombre historia: Poder jugar en red con otros jugadores |                  |                       |  |
| Prioridad: Alta                                         |                  | Estado: Finalizada    |  |
| Puntos estimados: 5                                     |                  | Iteración asignada: 6 |  |
| Programador responsable: María Cristina Alameda Salas   |                  |                       |  |

**Descripción:**

Como Jugador quiero poder jugar una partida con otros jugadores en red.

**Explicación**

Esta funcionalidad supuso un gran reto para nosotros ya que en ninguna asignatura habíamos estudiado nada parecido. Además, fue un gran desafío de diseño software.

En un sprint se realizó un modelo provisional básico de implementación del servidor y del cliente.

En el sprint siguiente se llevó a cabo una refactorización en el cual se crearon numerosas clases nuevas más pequeñas a partir de aquellas que acumulaban demasiada funcionalidad y resultaban muy grandes. También, se delegó la responsabilidad de parsear la información en una jerarquía de intérpretes.

Ya que supone una gran parte del proyecto se realizó un documento a parte con la explicación de su diseño y evolución. En el siguiente enlace puede encontrar la información ([multijugador en red.pdf](#)).

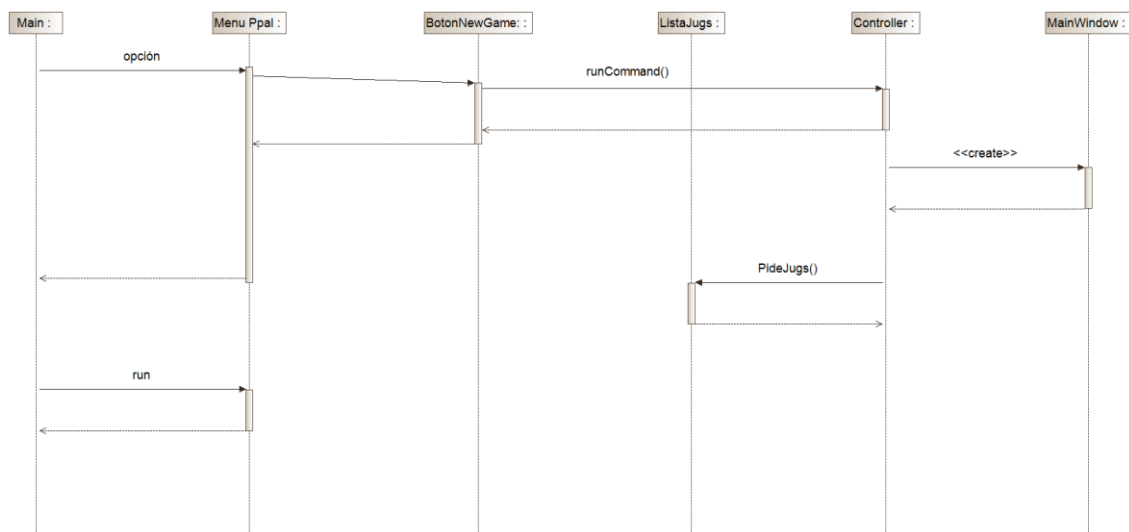
**4.55 Iniciar una partida desde la GUI**

|                                                                                      |                       |
|--------------------------------------------------------------------------------------|-----------------------|
| ID:H_47                                                                              | Usuario:Jugador       |
| Nombre historia:Iniciar una nueva partida desde la GUI                               |                       |
| Prioridad: Alta                                                                      | Estado: Finalizado    |
| Puntos estimados: 3                                                                  | Iteración asignada: 4 |
| Programador responsable: David Cruza Sesmero                                         |                       |
| Descripción:<br><br>Como Jugador quiero poder iniciar una partida nueva desde la GUI |                       |

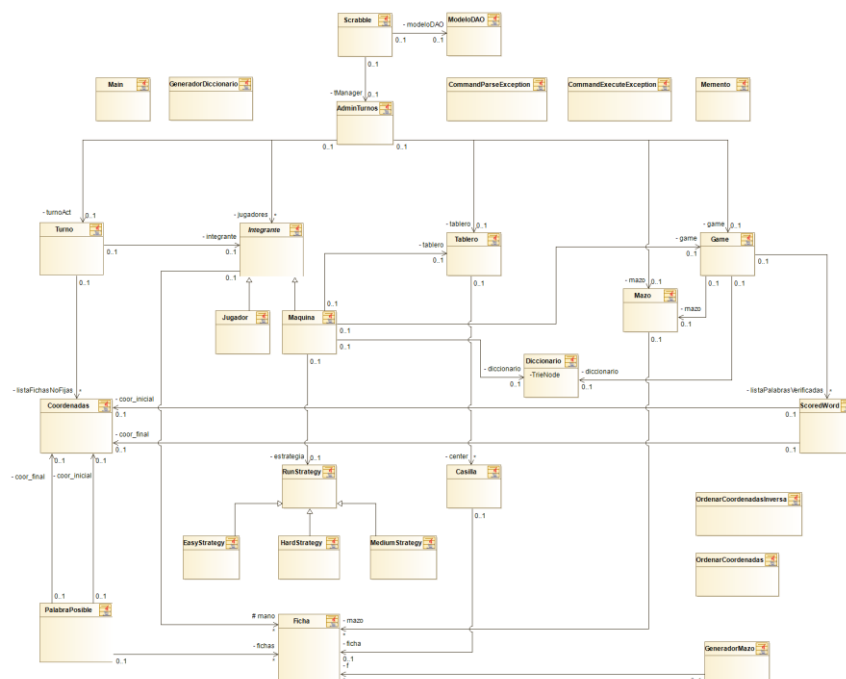
## Explicación detallada:

Puesto que la consola había sido desechada y tras generar todos los componentes de la interfaz gráfica debíamos unificar estos para poder comenzar una partida desde la GUI. En un principio este arrancaba directamente desde el MenuPpal hasta que se centró en el main el cual llamaba al MenuPpal y tras crear la lista de jugadores la cual se le pasaba al Controller se llamaba a MainWindow que es la clase contenedora de nuestro panel, etc...

## Diagrama:



## Diagrama general del modelo



(click en la imagen para verla más grande) (ENLACE)