Refactor Sprint 6

1. Introducción:

Durante este sprint, debido a la necesidad de adaptar el funcionamiento del juego a los eventos que produce la GUI, hemos refactorizado el modelo, eliminando la mayoría de los bucles que guiaban la ejecución del juego.

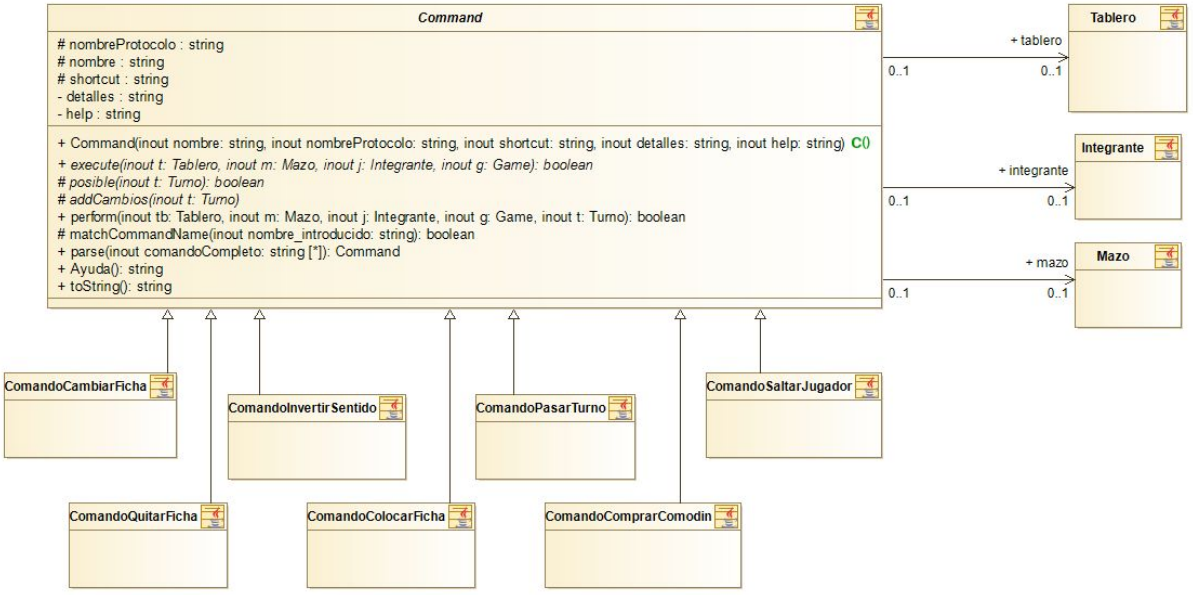
De paso, tras leer el feedback proporcionado por Gonzalo, hemos tratado de separar y delegar responsabilidades dentro del modelo para obtener un diseño más limpio y claro.

2. Principales Cambios:

Durante la refactorización, se han realizado los siguientes cambios:

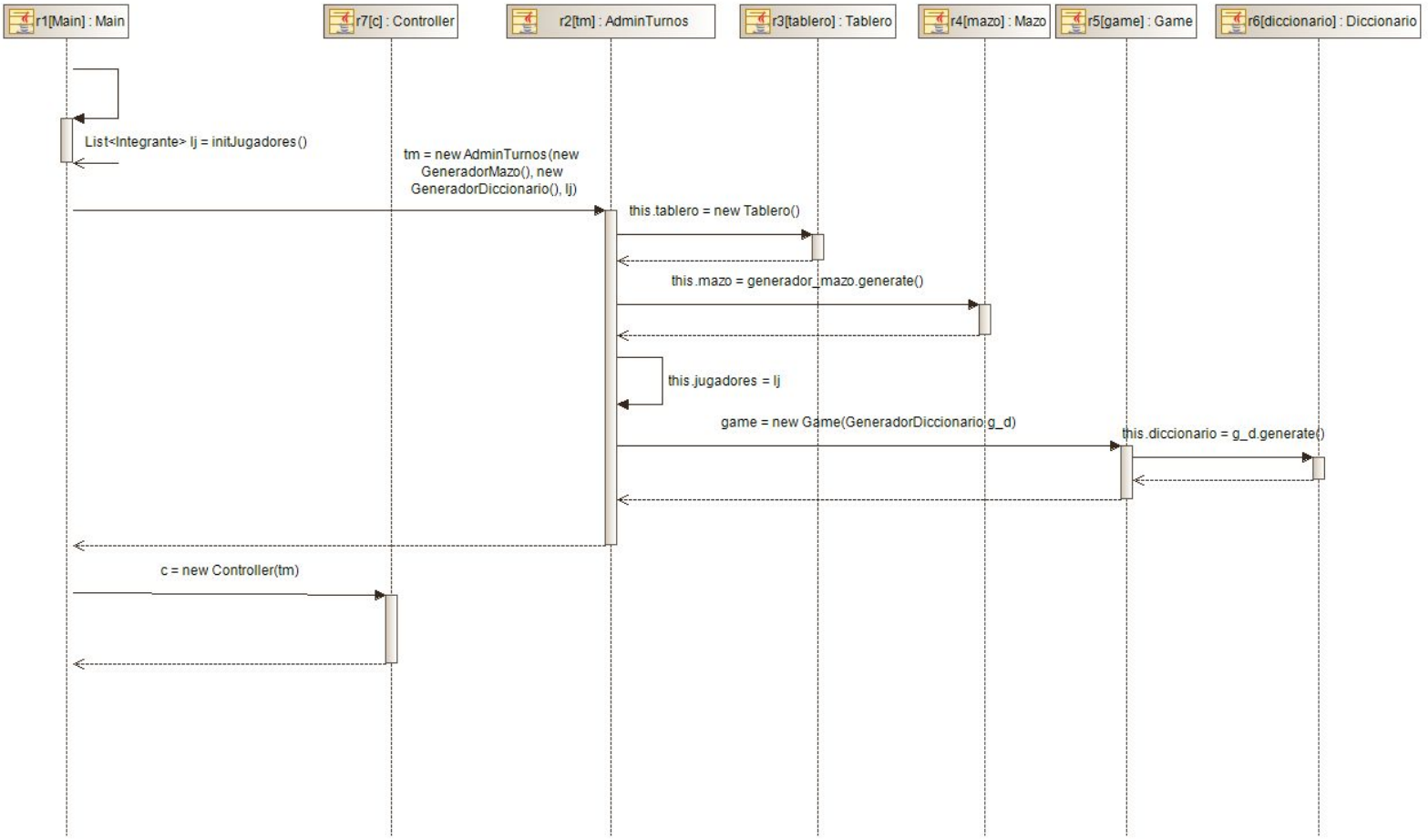
a. Comandos:

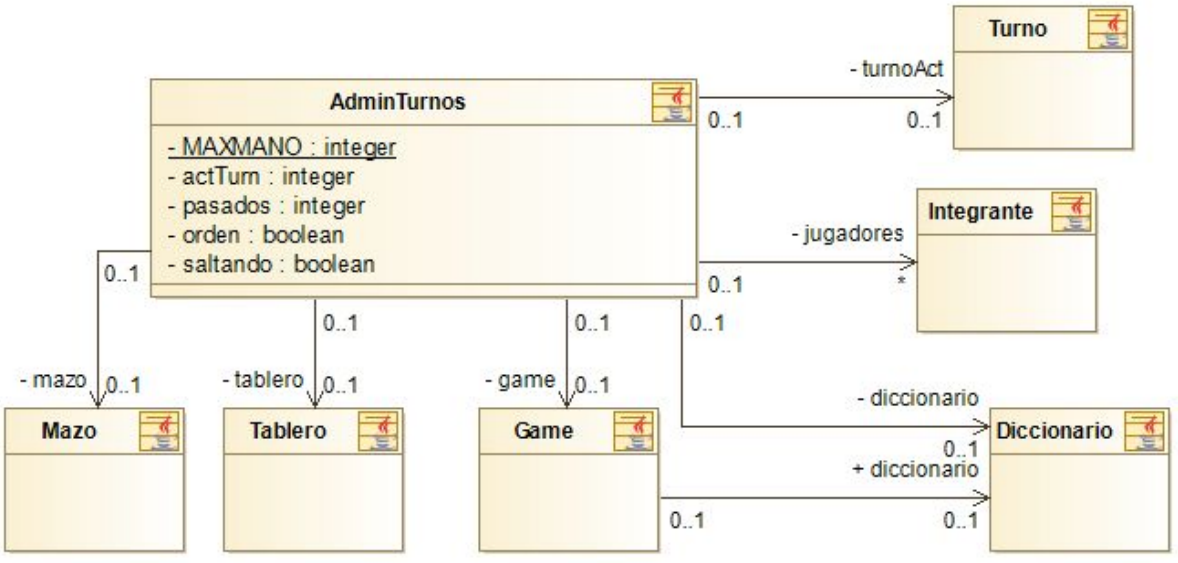
En el diseño anterior, los comandos en su ejecución, llamaban a un método del game que hacía de mediador en la interacción de los diferentes elementos del modelo (Tablero, Jugador, Mazo). Ahora, esa interacción se produce directamente dentro del método ‘execute’ de los comandos y el Game únicamente tiene la responsabilidad de verificar palabras usando el diccionario, y de calcular el valor de dichas palabras empleando el tablero.



b. AdminTurnos:

Anteriormente, el AdminTurnos contenía al Game, que a su vez contenía al Tablero, al Mazo y al Diccionario. Ahora, debido a la modificación anterior del patrón comando, el Game debería dar acceso al Tablero y al Mazo cada vez que se ejecute un comando. Por ello, hemos considerado más sencillo extraer el Tablero y el Mazo y dejarlos como atributos del AdminTurnos.

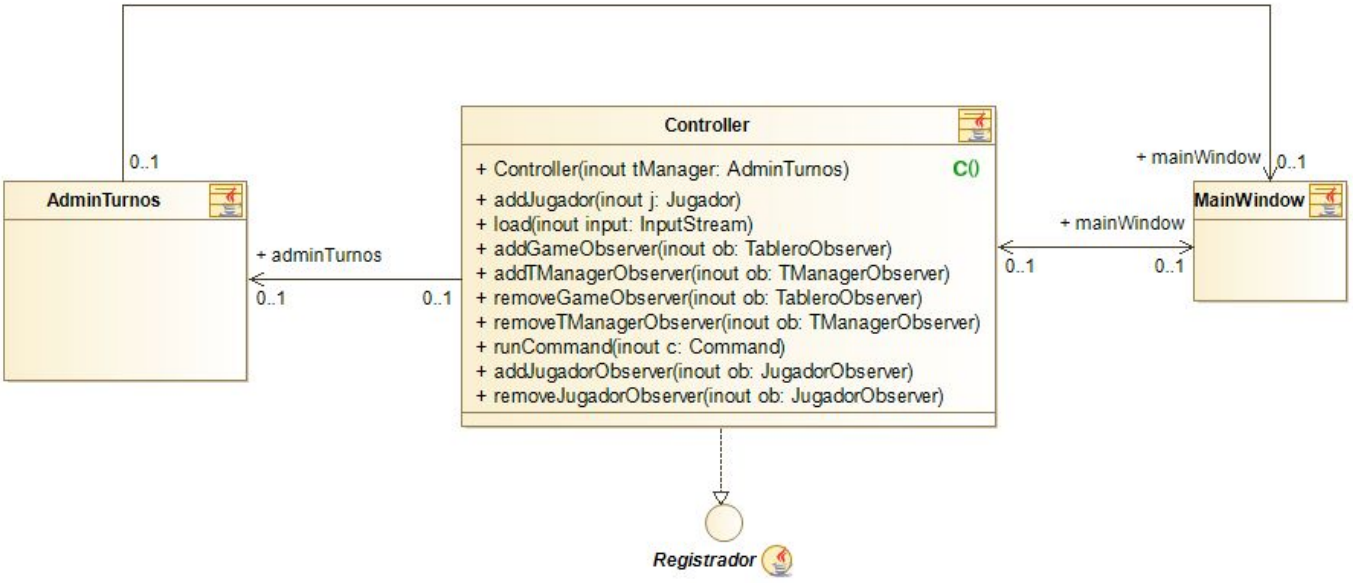




c. Controller

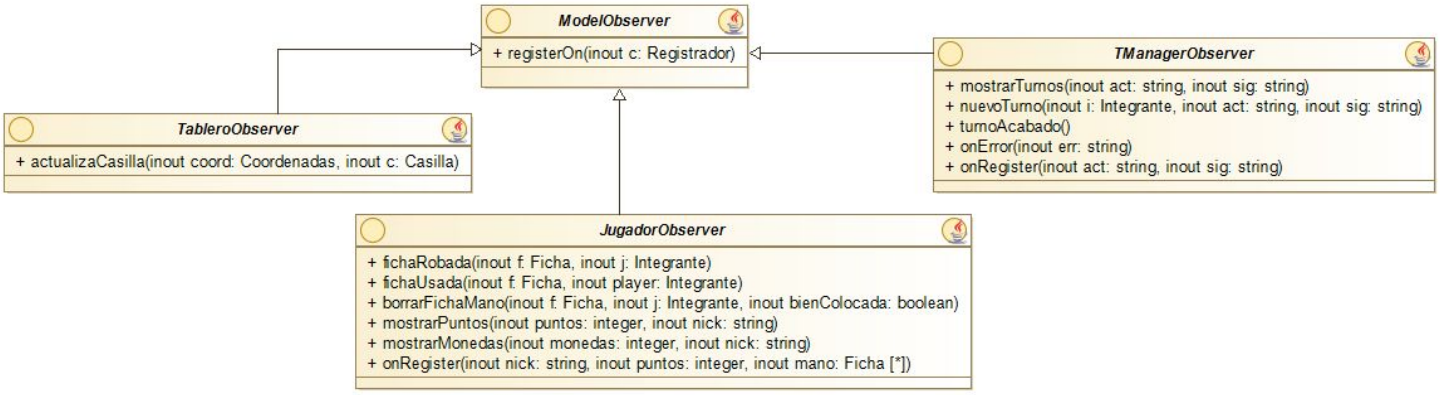
En el sprint anterior, la responsabilidad de crear el modelo pertenecía al Controller. En esta versión, hemos delegado esa responsabilidad al Main, pudiendo así liberar al Controller de eso, y asignarle únicamente la función de intermediario entre GUI y Modelo. En concreto, es encargado de transmitir las órdenes que envía la GUI al Modelo, y de registrar en el Modelo los elementos de la GUI que observan los cambios en sus diferentes elementos.

Además, el Controller implementa ahora una interfaz registrador, que engloba el comportamiento referente al registro de los observadores en el modelo. Esto nos posibilita un diseño más limpio a la hora de implementar la arquitectura cliente-servidor, pudiendo asignar este comportamiento a otro objeto (Sustituto, ver documento Cliente-Servidor).



d. Observadores

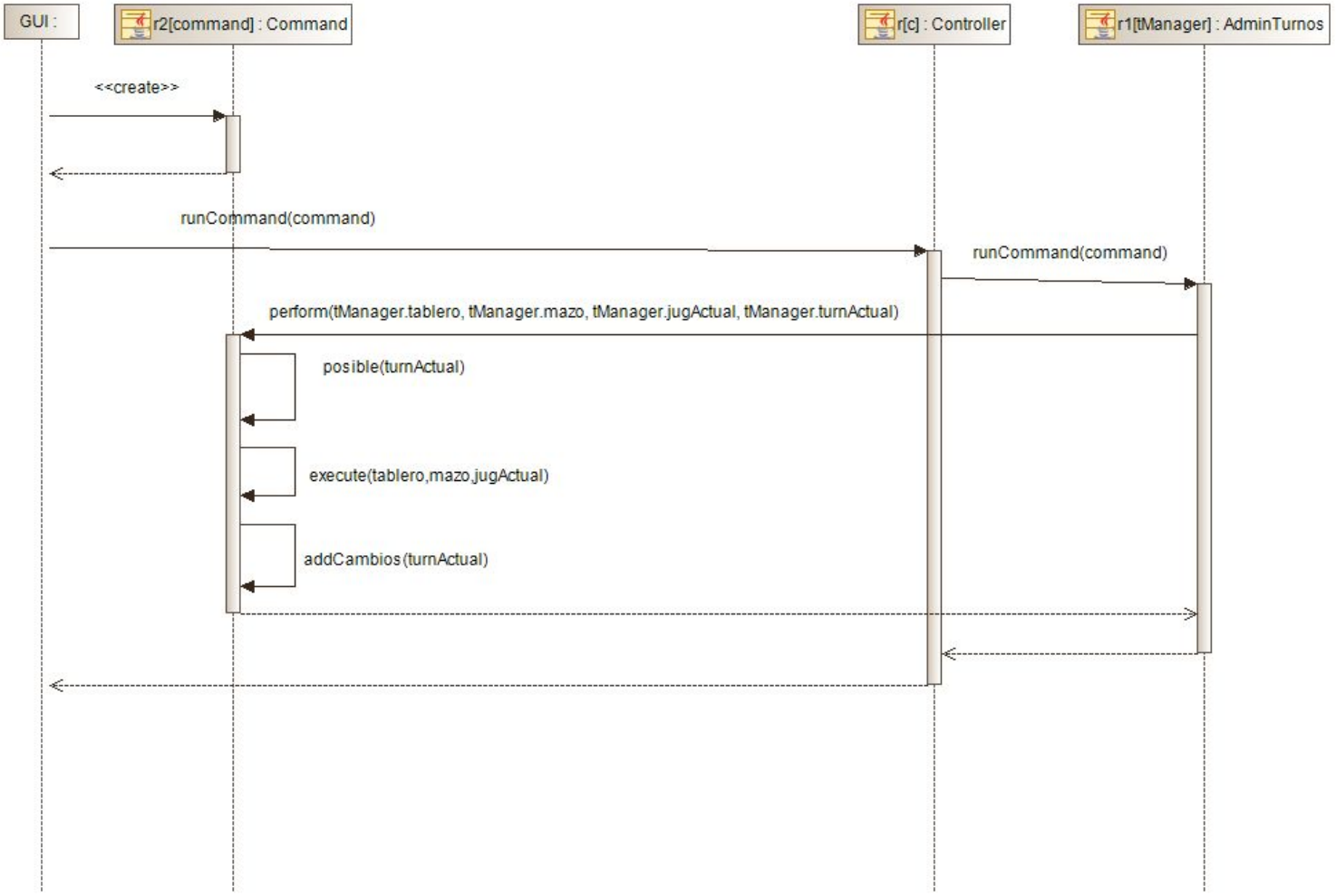
Anteriormente existían dos tipos de observadores TManagerObserver, que se encargaba de observar todos los cambios referentes a los turnos y GameObserver, que observaba los cambios en el Tablero y la Mano del jugador en activo.
Como hemos extraído los Integrantes, el Tablero y el Mazo del Game, ahora hemos tenido que añadir un nuevo tipo de observador más, el JugadorObserver, que se encarga de observar cambios en la mano del jugador.
Además, dados los cambios en los atributos del Game, los GameObservers han pasado a ser TableroObserver.



Model Observer queda como una interfaz general que contiene el comportamiento de cualquier observador, registrarse en un elemento observable del modelo.
De ella, heredan otras 3 interfaces que engloban los métodos que necesita cualquier elemento observador de las diferentes partes del modelo para reaccionar a los diferentes cambios que se produzcan dentro de estas partes.
JugadorObserver se encarga de los cambios producidas en cada uno de los integrantes de la partida, TManagerObserver reacciona a los cambios producidos en la secuencia de turnos, y TableroObserver agrupa el comportamiento de los objetos que observan los cambios en el tablero.

e. Turno

Esta clase ha pasado de ser la que permitía ejecutar comandos a los jugadores sobre el modelo (empleando un bucle que los pedía por consola), a convertirse en un almacén temporal de información para las acciones. De esta manera, todos los comandos cuya ejecución sea posible o no dependiendo de las acciones que haya realizado el jugador durante el turno, pueden acceder al turno para consultar la información necesaria (por ejemplo, no se pueden hacer más de 7 cambios de una ficha de la mano por otra del mazo durante un turno, es ese tipo información la que se guarda en el turno, y la que es consultada por el comando antes de ejecutarse).
(Se pueden observar estos cambios)


GUI :
r2[command] : Command
r[c] : Controller
r1[tManager] : AdminTurnos
<<create>>
runCommand(command)
runCommand(command)
perform(tManager.tablero, tManager.mazo, tManager.jugActual, tManager.turnActual)
posible(turnActual)
execute(tablero,mazo,jugActual)
addCambios(turnActual)