## Patrón: Command

### Tipo: de Diseño

### Contexto de Aplicación

En el caso de la aplicación hacemos uso de este patrón en el caso de la separación de las vistas y los controladores junto con los servicios. Las vistas proporcionan un conjunto de acciones a poderse realizar y los controladores y servicios se encargan de la lógica de negocio correspondiente a estas acciones.

### Clases o paquetes creados

Principalmente todas las entidades que interactúan de forma directa con las vistas, es decir que tienen un controlador asociado:

* AchievementController
* GooseBoardController
* GooseMatchController
* LudoBoardController
* LudoMatchController
* PlayerController
* UserController
* SessionController
* WelcomeController

### Ventajas alcanzadas al aplicar el patrón

Reduce el tamaño del problema separando las responsabilidades en distintas capas como lo son las vistas, los controladores y los servicios.

## Patrón: State

### Tipo: de Diseño

### Contexto de Aplicación

Se usa en ciertas entidades para definir su comportamiento dependiendo del estado en el que se encuentren. En el caso de la aplicación lo hemos usado para las fichas del parchís ya que funcionan de forma distinta dependiendo de si se encuentran en la casa, en el circuito del tablero o en la recta final de su color correspondiente.

### Clases o paquetes creados

* LudoChip: Clase que hace uso de los estados
* GameState: Enumerado que define los estados posibles que puede tener una determinada ficha de parchis.
* LudoChipService y LudoChipController: Clases que controlan la lógica de los cambios de estado

### Ventajas alcanzadas al aplicar el patrón

Simplifica la lógica necesaria a la hora de desplazar las fichas e interactuar entre ellas reduciendo el numero de comprobaciones que hay que hacer para decidir que acciones puede realizar una determinada ficha o no.

## Patrón: Template view

### Tipo: de Diseño

### Contexto de Aplicación

Se usa principalmente en el contexto de las vistas a la hora de generar listados de forma automática. Se ha usado principalmente en la representación de datos globales y personales de los jugadores en listas y tablas.

### Clases o paquetes creados

* AchievementController
* GooseMatchController
* LudoMatchController
* PlayerController

### Ventajas alcanzadas al aplicar el patrón

Agiliza en desarrollo de la aplicación reduciendo el tiempo invertido en diseñar nuevas vistas

## Patrón: Transaction Script

### Tipo: de Diseño

### Contexto de Aplicación

Organiza la lógica de negocio de la aplicación en métodos sencillos que solo hacen una cosa determinada. Usado principalmente en los servicios.

### Clases o paquetes creados

* AchievementService
* GooseBoardService
* LudoBoardService
* GooseMatchService
* LudoMatchService
* GooseChipService
* LudoChipService
* PlayerService
* PlayerGooseStatsService
* PlayerLudoStatsService
* AuthoritiesService
* UserService

### Ventajas alcanzadas al aplicar el patrón

Reduce el tamaño de los problemas mas grandes en un conjunto de pasos más sencillos mejorando así la mantenibilidad del código.

## Patrón: Table Module

### Tipo: de Diseño

### Contexto de Aplicación

Organiza la lógica de la aplicación con una clase por tabla de una de una forma similar a como lo hacen los servicios. Cada entidad tiene su propio servicio que maneja la lógica de los datos entre los atributos y las tablas.

### Clases o paquetes creados

* AchievementService
* GooseBoardService
* LudoBoardService
* GooseMatchService
* LudoMatchService
* GooseChipService
* LudoChipService
* PlayerService
* PlayerGooseStatsService
* PlayerLudoStatsService
* AuthoritiesService
* UserService

### Ventajas alcanzadas al aplicar el patrón

Mejora la mantenibilidad del código y la manejabilidad de los posibles cambios que se puedan hacer sobre este.

## Patrón: Domain Model

### Tipo: de Diseño

### Contexto de Aplicación

Se usa para diferenciar atributos más complejos y sus relaciones como por ejemplo los jugadores y sus logros un logro puede tener unas características al igual que su jugador y estos están claramente relacionados entre ellos.

### Clases o paquetes creados

* Player
* PlayerGooseStats
* PlayerLudoStats
* User
* Authorities

### Ventajas alcanzadas al aplicar el patrón

Distribuye la lógica de la aplicación en distintas clases reduciendo el tamaño del problema y su manejabilidad.

## Patrón: Service layer

### Tipo: Arquitectónico

### Contexto de Aplicación

Se usa en los servicios y se encargan de la comunicación entre los controladores y los repositorios además de manejar la mayor parte de la lógica sobre los datos.

### Clases o paquetes creados

* AchievementService
* GooseBoardService
* LudoBoardService
* GooseMatchService
* LudoMatchService
* GooseChipService
* LudoChipService
* PlayerService
* PlayerGooseStatsService
* PlayerLudoStatsService
* AuthoritiesService
* UserService

### Ventajas alcanzadas al aplicar el patrón

Encapsula claramente las funcionalidades mas importantes dentro de estas clases y mejora la mantenibilidad de la aplicación.

## Patrón: Active Record

### Tipo: Arquitectónico

### Contexto de Aplicación

Se ha aplicado en diferentes contextos, el primero siendo el seguimiento de los movimientos de las fichas, que quedan almacenados en la base de datos. El segundo viene dado por la auditación de algunas tablas y entidades.

### Clases o paquetes creados

Tablas auditadas:

* Achievements
* GooseMatch
* LudoMatch
* Player
* PlayerGooseStats
* PlayerLudoStats
* Authorities
* User

### Ventajas alcanzadas al aplicar el patrón

Nos permite tener un mejor seguimiento de los cambios realizados en las partidas y en las tablas.

## Patrón: Meta-data Mapper

### Tipo: de Diseño

### Contexto de Aplicación

Nos ayuda a establecer un contexto entre las relaciones de los atributos y entidades a la base de datos y es usado generalmente para indicar como se relacionan los atributos de ciertas entidades con otras clases facilitando así el manejo de la base de datos.

### Clases o paquetes creados

* Achievements
* GooseMatch
* LudoMatch
* GooseBoard
* LudoBoard
* Player
* PlayerGooseStats
* PlayerLudoStats
* User
* Authorities
* GooseChip
* LudoChip

### Ventajas alcanzadas al aplicar el patrón

En una correcta aplicación puede mejorar el rendimiento de la aplicación y la disposición de los datos.

## Patrón: Identify field

### Tipo: Arquitectónico | de Diseño

### Contexto de Aplicación

Nos permite identificar una fila de una tabla de forma unívoca, en el contexto de nuestra aplicación se ve claramente con el atributo username de los jugadores que usamos para diferenciar a unos de otros

### Clases o paquetes creados

### Ventajas alcanzadas al aplicar el patrón

Reduce la cantidad de memoria necesaria a la hora de hacer cambios sobre filas específicas en las tablas y facilita la identificación de individuos o datos concretos.

## Patrón: Layer supertype

### Tipo: de Diseño

### Contexto de Aplicación

En este caso hacemos uso de este patrón de diseño en clases que heredan ciertas características de otras clases, como es el caso de BaseEntity que lega alguna de sus características a clases como Achievements.

### Clases o paquetes creados

* BaseEntity
  + Achievement
  + Authorities
  + PlayerLudoStats
  + PlayerGooseStats
  + LudoMatch
  + GooseMatch
  + LudoBoard
  + LudoChip
  + GooseBoard
  + GooseChip
  + Person
    - Player

### Ventajas alcanzadas al aplicar el patrón

Simplifica el desarrollo de la aplicación reduciendo significativamente el código necesario para generar y almacenar algunas de las características de las entidades.