**Memento Pattern**

Link del repositorio analizado: <https://github.com/kan01234/design-patterns/tree/master/memento-pattern>

**Información general del proyecto**

* **¿Para qué sirve el proyecto?**

El proyecto sirve para implementar un sistema de guardado y restauración de estados de un jugador en un videojuego. En este sentido, permite capturar el estado actual del jugador y almacenarlo en un objeto Save. Seguidamente, se puede cargar un objeto Save previamente guardado para restaurar el estado del jugador a ese punto específico de la ejecución.

* **¿Cuál es la estructura general del proyecto?**

El proyecto se compone de tres clases:

* **Player:** Representa al jugador y contiene información sobre su estado actual (hp, mp y state). Proporciona métodos getters y setters para obtener y establecer valores, así como para guardar y cargar el estado.
* **Save:** Representa un objeto de guardado que almacena el estado del jugador en un momento específico. Contiene atributos para los puntos de vida (hp), los puntos de magia (mp) y el estado general. Además, proporciona métodos getters y setters.
* **SaveController:** Actúa como un controlador para administrar los objetos de guardado. Tiene una capacidad máxima predefinida para el número de guardados permitidos. Además, proporciona métodos getters y setters.

Recapitulando, el proyecto sigue la siguiente estructura: el Player contiene su estado y puede guardar y cargar ese estado utilizando objetos Save. El SaveController se encarga de administrar múltiples objetos Save y de las operaciones de guardar y cargar para el player.

* **¿Qué grandes retos de diseño enfrenta el proyecto?**

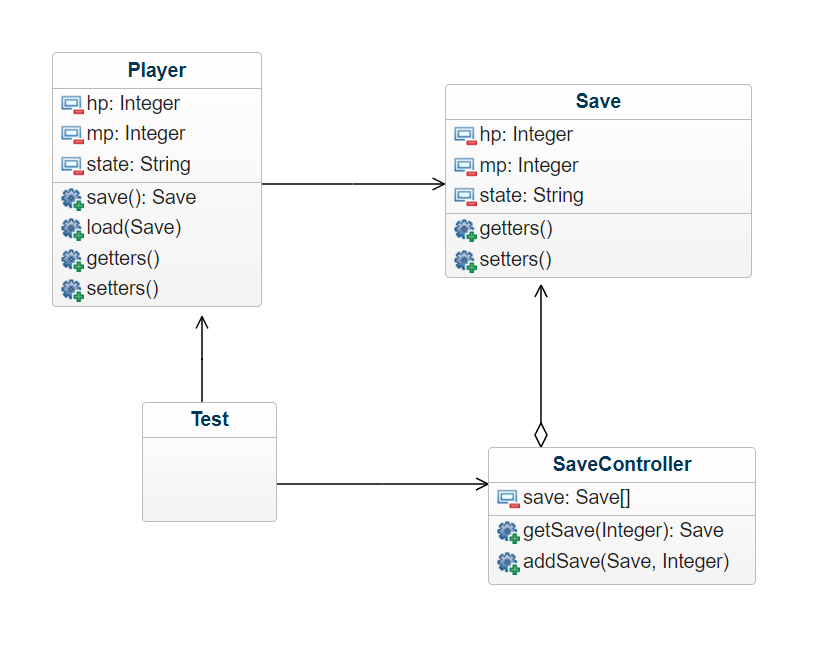
**Encapsulación del estado del jugador:** Un desafío importante es asegurar que el estado del jugador (puntos de vida, puntos de magia y estado general) se mantenga encapsulado y no se exponga directamente. Es decir, manejar métodos privados y acceder por medio de getters y setters para mantener la integridad del estado y evitar modificaciones no autorizadas.

**Gestión eficiente de los guardados:** El proyecto requiere una forma eficiente de gestionar los guardados del jugador. En este caso, se utiliza un arreglo saves en la clase SaveController para almacenar los objetos Save. Sin embargo, un problema es determinar cómo manejar la gestión de la memoria cuando se alcanza el límite máximo de guardados.

**Limitación del número de guardados:** Otro reto consiste en establecer y mantener un límite para la cantidad de guardados permitidos. En el caso del proyecto actual, se utiliza una constante MAX\_NUM\_OF\_SAVE para definir el número máximo de guardados. Sin embargo, la gestión de este límite puede complicarse en diferentes aspectos, como qué sucede cuando se intenta agregar un guardado más allá del límite establecido o cómo notificar al usuario cuando se alcance el límite de guardados.

**Información del fragmento del proyecto donde aparece el patrón.**

El proyecto sigue el patrón de diseño Memento, que se utiliza para capturar y almacenar el estado interno de un objeto sin perder encapsulación, permitiendo que el objeto pueda ser restaurado a ese estado en un momento posterior. El patrón se aplica de la siguiente manera:



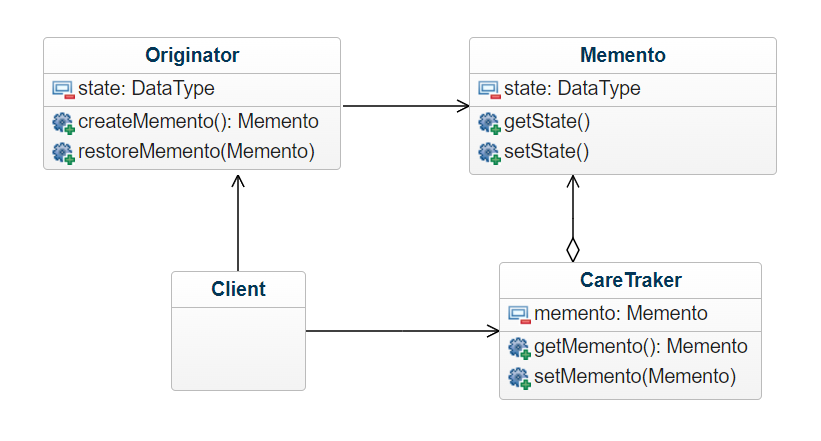
El objeto Player es el origen del cual se desea guardar y restaurar el estado. Utiliza la clase Save como un memento para capturar su estado actual y almacenarlo. El objeto SaveController actúa como un caretaker y se encarga de gestionar múltiples objetos Save. Permite guardar y recuperar los estados capturados del Player. Estos ítems se explicarán a continuación:

**Información general sobre el patrón**

* **¿Qué patrón es? Memento.**

El patrón Memento es un patrón de diseño de software que se utiliza para capturar y almacenar el estado interno de un objeto en un momento determinado, sin violar su encapsulación. Luego, este estado almacenado, llamado memento, se puede restaurar más tarde, devolviendo al objeto a ese estado previo.

El patrón Memento se compone de tres partes principales:



**Origen (Originator):** Es el objeto del cual se desea guardar y restaurar el estado. Este objeto crea un memento que representa su estado actual.

**Memento:** Es el objeto que almacena el estado capturado del objeto origen. Contiene una representación del estado interno del origen, pero no expone detalles internos. Solo el origen puede acceder y modificar el estado contenido en el memento.

**Caretaker:** Es el objeto responsable de almacenar y gestionar los mementos creados por el origen. No debe modificar ni acceder directamente al estado contenido en el memento, sino que se encarga de guardarlos, restaurarlos o entregarlos al origen cuando sea se lo soliciten.

* **¿Para qué se usa usualmente el patrón?**

El patrón Memento se utiliza cuando se requiere capturar y restaurar estados anteriores de un objeto, preservando la encapsulación. Algunas situaciones en las que se aplica el patrón Memento son:

**Implementación de funcionalidades de deshacer/rehacer:** El patrón Memento permite guardar diferentes estados de un objeto, lo que facilita la implementación de operaciones de deshacer/rehacer. Los mementos actúan como puntos de restauración a los que se puede regresar para revertir cambios o avanzar en un historial de acciones.

**Guardado y restauración de estados en aplicaciones:** Cuando se desea guardar el estado de una aplicación en un momento dado y restaurarlo posteriormente. Esto es importante en aplicaciones como editores de texto, editores gráficos o videojuegos, donde los usuarios pueden guardar su progreso y regresar a estados anteriores.

**Implementación de checkpoints:** En aplicaciones complejas, el patrón puede utilizarse para implementar checkpoints que permitan guardar el estado actual y restaurarlo en caso de fallos o un reinicio. Esto asegura la continuidad de la aplicación desde el último guardado.

* **¿Por qué tiene sentido haber utilizado el patrón en ese punto del proyecto? ¿Qué ventajas tiene?**

El uso del patrón Memento en este proyecto tiene sentido por diferentes razones, y ofrece diversas ventajas, a continuación:

**Preservación de la encapsulación:** ya que, permite capturar y almacenar el estado interno de un objeto (Player) sin exponer su estructura interna, ni perder su encapsulación. Al utilizar el objeto Save como memento, el estado del jugador se encxapsula y se mantiene oculto al exterior. Esto promueve una mayor cohesión y modularidad en el diseño, evitando que el estado del jugador sea accesible o modificado directamente por otros objetos.

**Funcionalidad de deshacer/rehacer:** el patrón facilita la implementación de funcionalidades de deshacer/rehacer en el proyecto. Al permitir que el jugador guarde y cargue diferentes estados a través del objeto Save.

**Gestión flexible de los guardados:** se facilita la gestión de los guardados del jugador. El SaveController actúa como un intermediario que almacena y gestiona los objetos Save. Esto permite tener un control más flexible sobre los guardados, como establecer un límite máximo de guardados MAX\_NUM\_OF\_SAVE, acceder a guardados específicos según su posición en el arreglo y agregar nuevos guardados. Además, esta estructura permite mantener una separación clara de responsabilidades, donde el Player se encarga de guardar y cargar su estado, mientras que el SaveController se encarga de la gestión y almacenamiento de los mementos.

**Facilidad de mantenimiento y escalabilidad:** Al utilizar el patrón Memento, el proyecto se beneficia de una mayor modularidad y separación de responsabilidades. Esto facilita el mantenimiento y evolución del código, ya que los cambios en la implementación del Player o del SaveController pueden realizarse de forma independiente sin afectar el funcionamiento general del sistema.

* **¿Qué desventajas tiene haber utilizado el patrón en ese punto del proyecto?**

**Sobrecarga de memoria:** El patrón implica un consumo adicional de memoria, especialmente si se guardan y almacenan varios estados del objeto Player. Cada objeto Save contiene una copia del estado completo del jugador, lo que puede resultar en un aumento en el uso de memoria. Si el proyecto maneja una gran cantidad de guardados o si los estados del jugador son muy grandes, esto puede impactar negativamente en el rendimiento y los recursos del sistema.

**Complejidad adicional:** La implementación del patrón Memento agrega una capa adicional de complejidad al proyecto. Requiere la creación de nuevas clases (como Save y SaveController) y la gestión de la interacción entre ellas. Esto aumenta la complejidad del código, especialmente si se trata de un proyecto pequeño o si los requerimientos de guardado y carga son simples.

**¿De qué otras formas se le ocurre que se podrían haber solucionado, en este caso particular, los problemas que resuelve el patrón?**

Algunas alternativas que podrían considerarse para abordar los problemas que resuelve el patrón podrían ser:

* **Gestión manual del estado:** En lugar de utilizar el patrón Memento, se podría implementar una gestión manual del estado del jugador sin la necesidad de crear objetos Save. Esto implica definir métodos específicos en la clase Player para guardar y cargar el estado en estructuras de datos. Aunque esto requiere más código y gestión directa del estado, puede ser una opción más simple.
* **Serialización/deserialización del estado:** En lugar de almacenar el estado en objetos Save, se podría utilizar la serialización/deserialización para guardar y cargar el estado del jugador. Esto requiere convertir el estado en una representación serializada (como un archivo JSON o una cadena de texto) y almacenarlo de manera persistente. La desventaja de esto es que se perdería la capacidad de acceder directamente a propiedades del estado, ya que se trataría como un bloque que no se puede dividir.

**Bibliografía:**  
  
Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.