Proyecto 2: Implementación del tamagotchi

UNAM, Facultad De Ciencias.

Curso: Modelado y Programación
Profesor.Rosa Victoria Villa Padilla
Ayudante. Arturo Lemus Pablo
Ayudante. Fernando López Balcazar
Ayudante. Itzel Azucena Delgado Díaz

Equipo: Prietos en aprietos

SanMartin Macias Juan Daniel No. Cuenta 318181637
López Diego Gabriela No. Cuenta 318243485
Rivera Zavala Javier Alejandro No. Cuenta 311288876

---

18 de Noviembre de 2022.
Ciudad de México.

1. Problemática a resolver

El proyecto consiste en implementar una simulación de la popular mascota electrónica conocida como Tamagotchi, haciendo uso de algunos patrones vistos (explicados más adelante). Además de agregarle algo más de variedad aparte de sólo estar cambiando entre estados del Tamagotchi.
Se agregó un sistema de comida, que consiste en armar un hot dog para el consumo del Tamagotchi, además de un sistema de minijuegos como *Gato, Piedra Papel o Tijeras, y Ahorcado*, las cuales te ayudan a conseguir monedas para comprar cosméticos o más minijuegos.
Cosas que debes tener en cuenta:

1. Debes de gestionar bien tu gasto de monedas ya que, el comprar la comida hace uso de éstas, sin embargo, si no tienes suficientes monedas y tu opción de jugar llegó a su límite en ese estado, muy probablemente tu Tamagotchi morirá de hambre, y una vez muerto ya no hay forma de activar ninguna opción más.
2. El jugar no siempre te dará monedas, es más como un sistema de apuestas donde si empatas, no se te dará ninguna moneda, si pierdes se te quitará alguna cantidad de monedas, y obviamente si ganas, se te otrogará una cantidad de monedas (estos últimos dos puntos varian dependiendo de la dificultad en la que esté el minijuego).

A continuación se da una breve explicación del por qué se utilizaron lo patrones implementados, además de señalar dónde se están implentando.

2. Patrón de diseño Strategy

Utilizamos el patrón Strategy para la función de los múltiples aspectos disponibles para el tamagotchi. Haciendo uso de la mayor ventaja de este patrón, que es el hacer al código más límpio para su mantenimiento, evitandonos estar considerando (en todas las partes donde se tenga que imprimir el aspecto del Tamagotchi) múltiples casos, usando un alto número de condicionales.
Con este patrón la agregación de un nuevo aspecto a futuro quedaría muy sencillo, puesto que sólo sería agregar una clase correspondiente a ese nuevo aspecto que implemente a la interface Apariencia, sin tener que modificar nada en la parte del patrón state (que es donde se manda a llamar la apariencia del Tamagotchi).

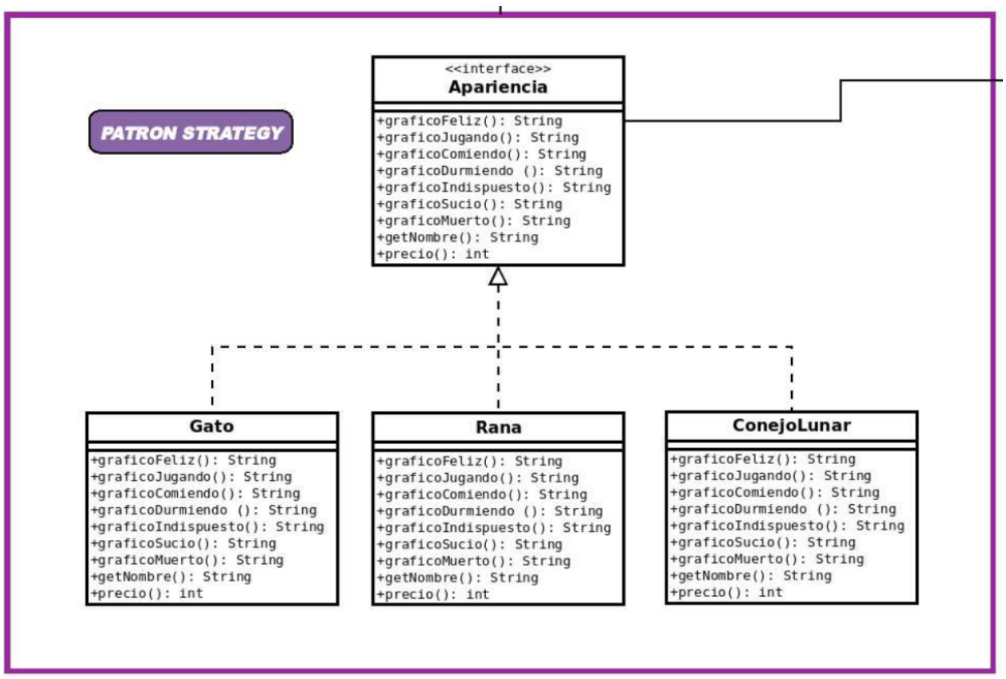

Figura 1: Diagrama UML Strategy

3. Patrón de diseño Facade

Utilizamos este patrón en varias partes del proyecto.

1. Estando en el patrón state, cuando está disponible la acción de jugar, sólo se hace una instancia de la clase MenuDeJuegos, quedando en una sóla línea, sin embargo, detrás de esta clase están pasando muchísimas cosas, por ejemplo, el verificiar los juegos disponibles del jugador, el desplegar el menú de los juegos, el iniciar el juego, terminada la partida guardar y regresar el número de monedas que ganó el jugador, y finalmente asignar estas monedas al puntaje del Jugador.
2. Nuevamente, en el patrón state, cuando está disponible la acción de comer, sólo se hace instancia de la clase TiendaComida, sin embargo por detrás lo que sucede es básicamente la implementación del patrón decorator para el Hot dog a consumir, y finalmente verifica si las monedas son suficientes para pagar, si lo son, se descuentan y el hot dog se consume, permitiendo la continuación del patrón state. Sin embargo, si las monedas no son suficientes, el hot dog no es consumido y el Tamagotchi no puede seguir su recorrido, sólo teniendo la opción de jugar para ganar más monedas. (Sí pasan 10 turnos sin que el tamagotchi no coma, morirá y ya no será posible seguir jugando).
3. Finalmente en el Menu que se despliega en la clase Main, está haciendo uso de facade, ya que la clase main sólo se instancia la clase Menu y se manda a llamar a su método display() que es el encargado de todo.

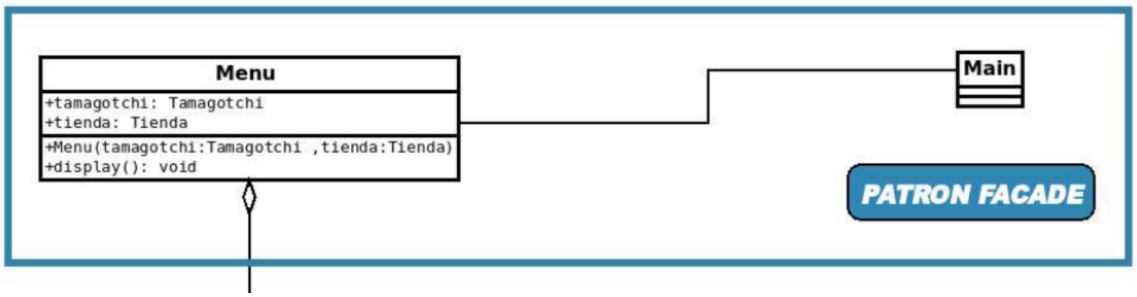

Figura 2: Diagrama UML patrón Facade

Razón de uso
Decidimos utilizar este patrón para hacer un poco más evidente la separación de las clases, para poder identificar el patrón MVC.
Nota: En el UML entregado, hay una presencia más de Facade, sólo que la calidad de imagen bajaba mucho al colocarlo en este documento.

4. Patrón de diseño Decorator

Como se mencionó antes, este patrón se utilizó en la preparación de hotdogs para alimentar al tamagotchi. Usando la salchica del hot dog como núcleo y decorandolo con múltiples ingredientes, además de subir el precio, entre más alto sea el número de ingredientes en el hot dog, el número de turnos que se pueden pasar sin comer aumenta, en relación uno a uno, es decir, si el hot dog cuesta 5 monedas, el número de turnos que puedes pasar aumenta en 5.

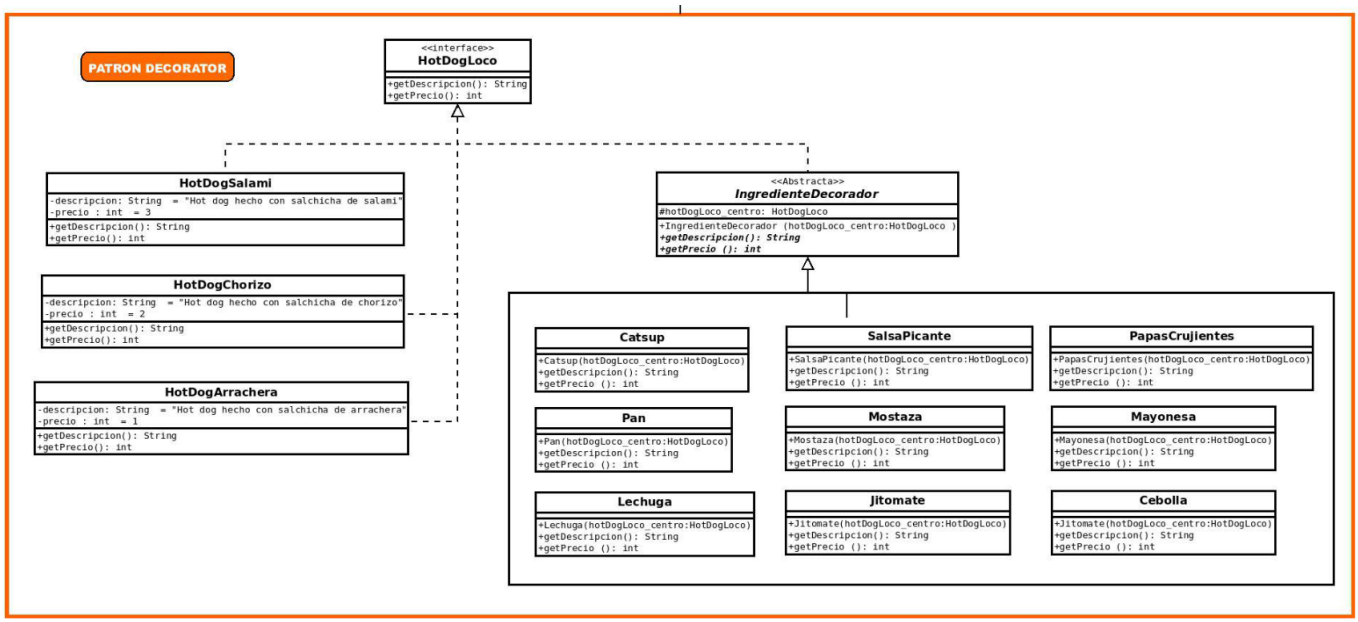

Figura 3: Diagrama UML patron Decorator

Razón de uso
Decidimos utilizar este Patrón para simular la práctica 3 del curso, sólo que esta vez se prepararón Hot Dogs en vez de Baguettes.

5. Patrón de diseño State

Utilizamos este patrón para las transiciones del tamagotchi, dejando así más limpia la implementación del Tamagotchi.

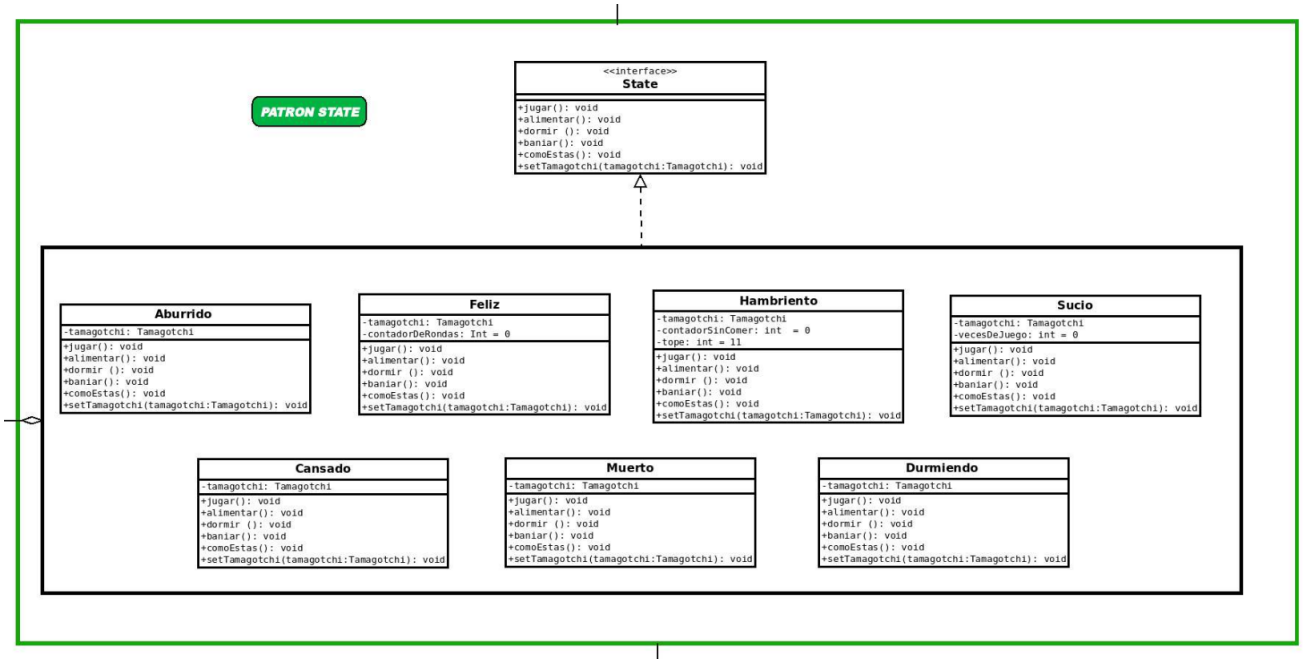

Figura 4: Diagrama UML patron State

Razón de uso
Consideramos que este patrón es el *Esqueleto* del proyecto ya que sin él, la implementación de un tamagotchi sería muy confusa y con muchísimos condificionales, haciendo muy complicado su mantenimiento, así como el agregar nuevas funciones o elementos al Juego.

6. Patrón de diseño MVC

Este patrón era el único obligatorio para este proyecto, para la fácil división de trabajo sin tener problemas de rigidez si queires cambair algo de Frontend y Backend.

Nota: En esta ocasión en el UML no se encuentra presente la división de MVC, porque resultaba complicado separar las clases en estas tres secciones mantiendo la subdivisión de los patrónes antes presentados.

Por último.
Algunos puntos a aclarar para este proyecto son los siguientes:

1. En la carpeta entregada se incluye el diagrama de estados derivado del patrón *State*, además de un archivo *PDF* donde se presenta el UML del proyecto, debido a que sólo en ese formato no disminuía la resolución.
2. Para compilar el proyecto escribimos lo siguiente en terminal

   javac Vista/Main.java

   Y para ejecutarlo

   java Vista.Main
3. Finalmente, cabe aclarar que los minijuegos **Gato** y **Ahorcado**, están basados en soluciones publicadas en internet, aunque están modificados y adaptados para un buen funcionamiento acorde a lo buscado en este proyecto. No obstante consideramos oportuno mencionarlo y darles crédito.

   a) Solución del minijuego Gato:
   https://parzibyte.me/blog/2021/06/21/tres-linea-java-programacion-juego/

   b) Solución del minijuego Ahorcado:
   https://www.lawebdelprogramador.com/codigo/Java/5660-Juego-del-ahorcado-en-consola.html