PROYECTO 01 - Modelado y Programación

Equipo "Better Code Saul"

Integrantes

Nombre	No. de cuenta
Álcantara Estrada Kevin Isaac	319073799
Cureño Sánchez Misael	418002485
Hernández Páramo Elizabeth	319143209

Problematica a resolver

La división Fciencias.game.inc de la facultad de ciencias desea desarrollar un videojuego para que los alumnos puedan disfrutar y desestresarse. Tal videojuego es uno con un sistema de combate por turnos (determinados por caracteristicas propias del barco) en el que los usuarios puedan tomar decisiones en y estrategias de juego durante el combate. Se solicita que el usuario pueda crear un barco de acuerdo a los distintos componentes disponibles, además, éstos componentes deben afectar las estadísticas del barco. Aunque por el momento se cuenta con pocos componentes, se piensa agregar más en el futuro.

Otra característica que se quiere implementar al videojuego es que haya poderes o power ups que afecten la forma de jugar segun el tipo de nave, además, se espera que posteriormente éstos poderes puedan tener más comportamientos y acciones.

Un profesor de la carrera de biología propuso utilizar personajes de otros videojuegos de la división para ésta entrega, por ejemplo, los monstruos marinos de la famosa saga "Pulpos and Masters", sin embargo, tales personajes no son barcos como se espera para éste videojuego y no se comportan como tal.

Como la Fciencias.game.inc tuvo una reducción de su espacio ilimitado de Jmail, se han vuelto obsesivos con el uso de recursos, incluso en éstos videojuegos, así que desean utilizar la menor cantidad de instancias posibles de una clase, el problema es que necesitan alguna estructura para guardar los datos del juego, como los componentes disponibles, los enemigos a combatir, etc; pero no sólo eso, desean que se genere una variedad de enemigos para cada enfrentamiento. Vaya dilema.

El profesor de biología, además, se ha vuelto receloso con sus personajes de "Pulpos and Masters" y condiciono que si se utilizan sus diseños, los jugadores no deben de poder acceder directamente a la información de éstos.

Por último, la división espera que más adelante se puedan agregar más funcionalidades al videojuego, quizás ver un ranking o un foro con consejos para ganar, por ello desean que agregar o quitar secciones del juego, especialmente en el aspecto visual no sea complejo.

Instrucciones de compilacion

Linux

Forma 1

1. Dirigirse al directorio raiz del proyecto
2. Compilar usando `./mvnw package`
3. Ejecutar usando `java -jar ./target/proyecto_2-0.0.1-executable.jar`

Forma 2

1. Dirigirse al directorio raiz del proyecto
2. Ejecuta el script haciendo `./run.sh` desde la terminal (en caso de no tener permiso de ejecución ejecutar `chmod +x run.sh`).

Windows

Forma 1

1. Dirigirse al directorio raiz del proyecto
2. Compilar usando `.\mvnw.cmd package`
3. Ejecutar usando `java -jar .\target\proyecto_2-0.0.1-executable.jar`

Patrones

Para esta práctica utilizamos los siguientes patrones:

1. **Strategy**: Como solución para la problemática, decidimos usar Strategy en el aspecto de los poderes, pues de ésta manera podemos dar los power ups que se decida a los distintos tipos de barcos para que su comportamiento sea distinto. Además, como se espera que los power ups hagan más cosas en la posterioridad, usar una interfaz para encapsular el comportamiento de éstos poderes nos permite modificar o agregar cosas a los power ups sin tener que modificar el resto del código.

2. **Prototype**: Cómo la división quiere ahorrarse espacio, para los enemigos no crearemos miles y miles de instancias distintas para que cada enfrentamiento sea distinto al menos en las estadísticas de los enemigos, sino que utilizaremos Prototype, pues al usar la interfaz Cloneable podemos generar copias con estadísticas modificadas a partir de una sola instancia de la clase, ahorrándonos mucho espacio y tiempo.

3. **Adapter**: Decidimos utilizar Adapter porque podemos meter otro tipo de enemigos, tal como los monstruos marinos de "Pulpos and Masters", de tal forma que aunque no sean de la clase Nave (los barcos que constituyen nuestro juego) podemos usar un adaptador para que se comporten como deseamos. Además, si se desea agregar otro tipo de enemigos con otras caracteristicas será posible hacerlo sin tocar el codigo de la clase Nave.

4. **Iterator**: Haciendo caso al profesor de biología, para evitar que los usuarios tengan acceso directo a la clase de sus monstruos marinos, podemos almacenar las instancias de ésta clase (y otras) en una estructura de datos privada y brindar al usuario un iterador con el cual podrá ejecutar de forma controlada y tal como se planea un partida.

5. **Builder**: Implementamos éste patrón para que el usuario pueda construir su Nave (barco) en tiempo real, actualmente hay 3 componentes de cada tipo y hay 3 tipos de componentes, así que hay 9 posibles combinaciones de componentes para disfrutar, pero se dice que se desea agregar más componentes a futuro, así que unos cuantos más y tendríamos cientos de combinacioes, es por eso que al utilizar Builder relegamos todas éstas posibles combinaciones a la perspicacia del usuario, ahorrándonos mucho código a futuro.

6. **MVC**: Decidimos utilizar éste patrón de diseño ya que podemos separar las partes de nuestro código bastante bien y ésto facilita la realización de cambios a futuro para toda la parte visual y de control, los modelos también resultan más sencillos de modificar y esto evita rigidez y viscosidad.

7. **Singleton**: Utilizamos el patrón singleton para la parte de los repositorios en vista de que están simulando un acceso a datos y por lo tanto no necesitamos crear diferentes instancias de la clase, pues con una sola podríamos acceder a ellos desde cualquier parte del codigo, ahorrando de esta forma recursos.

Notas

- Los diagramas están ubicados en la carpeta `docs` del proyecto.

Diagramas

- Adapter
- Prototype
- Iterator
- MVC

- Singleton
- Strategy
- Builder