Curso 2019 / 2020

Práctica 3:

Implementación de la funcionalidad del sistema, dado su diseño dinámico

Incorporando código suministrado por los profesores a tu proyecto

En esta práctica también os proporcionamos código que tenéis que incorporar a vuestro proyecto, en concreto se trata de clases que incorporan una interfaz de texto que, con el programa principal proporcionado, permite jugar, ¡por fin!, a DeepSpace.

Averigua la ruta de tu proyecto. En el guion de la práctica 2 se indica cómo hacerlo.

- En Java: Si la ruta de tu proyecto es R, en R/src verás una carpeta por cada paquete de tu proyecto. Ahora mismo solo debes tener la carpeta deepspace correspondiente al paquete con el mismo nombre. Tras descomprimir el archivo zip suministrado copia las carpetas View, *controller* y *main* junto con su contenido dentro de tu carpeta R/src/
  Debe quedar una estructura de carpetas como la siguiente:

```
R/src
├ controller
├ deepspace
├ main
└ View
  └ UI
```

- En Ruby, copia los ficheros fuente suministrados en R/lib/

- En ambos lenguajes, modifica las propiedades ("Properties") del proyecto para que el nuevo programa principal utilizado sea el suministrado por los profesores.
  - Haz clic con el botón derecho sobre el nombre del proyecto en Netbeans y selecciona la opción "Properties".
  - Haz clic sobre la categoría "Run"
  - En el campo "Main Class" o "Main Script" haz clic sobre el botón "Browse...", localiza y elige el archivo "PlayWithUI.*" correspondiente.

No debes modificar ninguna de las clases suministradas por los profesores. Si hay un problema de concordancia repasa en primer lugar las clases que ya has creado para ver si siguen totalmente las especificaciones dadas. Si después de ese repaso persiste el problema, consulta con tu profesor.

Desarrollo de esta práctica

En esta práctica se implementan los métodos que faltan finalizando la implementación de la primera versión del juego. Esta, junto con las clases suministradas por los profesores que implementan una interfaz de texto, hace posible jugar al mismo.

A pesar de haber probado todos los métodos implementados en esta práctica y en las anteriores, tal como se recomendaba en la práctica 1 para todas las prácticas; es posible que, jugando partidas, te

des cuenta de nuevos errores en el código. En estos casos se debe crear un pequeño programa principal de prueba que *fuerce* la situación que ha provocado el error como ayuda a localizarlo y comprobación de que se ha corregido. Debido a la aleatoriedad del juego será muy poco probable que se vuelva a dar el mismo error en un tiempo razonable.

Completando los mazos de cartas

Modifica los métodos que sean necesarios para que en cada mazo de cartas haya una carta más. Crea en cada caso la carta con los valores de tu elección. Todos los mazos de cartas se crean en la clase CardDealer.

Implementando el resto de métodos

Para la implementación de los métodos que faltan se proporcionan diagramas de secuencia. La implementación se debe ajustar a lo especificado en dichos diagramas. Forma parte del aprendizaje, y de la evaluación, saber interpretar los diagramas y traducirlos a código.

SpaceStation

float fire(). Realiza un disparo y se devuelve la energía o potencia del mismo. Para ello se multiplica la potencia de disparo por los factores potenciadores proporcionados por todas las armas. Su diagrama de secuencia se encuentra en el archivo SpaceStation-fire.pdf

float protection(). Se usa el escudo de protección y se devuelve la energía del mismo. Para ello se multiplica la potencia del escudo por los factores potenciadores proporcionados por todos los potenciadores de escudos de los que se dispone.
Su diagrama de secuencia se encuentra en el archivo SpaceStation-protection.pdf

ShotResult receiveShot(float shot). Realiza las operaciones relacionadas con la recepción del impacto de un disparo enemigo. Ello implica decrementar la potencia del escudo en función de la energía del disparo recibido como parámetro y devolver el resultado de si se ha resistido el disparo o no.
Su diagrama de secuencia se encuentra en el archivo SpaceStation-reciveShot.pdf

void setLoot(Loot loot). Recepción de un botín. Por cada elemento que indique el botín (pasado como parámetro) se le pide a CardDealer un elemento de ese tipo y se intenta almacenar con el método *receive*()* correspondiente. Para las medallas, simplemente se incrementa su número según lo que indique el botín.
Su diagrama de secuencia se encuentra en el archivo SpaceStation-setLoot.pdf

void discardWeapon(int i). Se intenta descartar el arma con índice i de la colección de armas en uso. Además de perder el arma, se debe actualizar el daño pendiente (pendingDamage) si es que se tiene alguno.
Su diagrama de secuencia se encuentra en el archivo SpaceStation-discardWeapon.pdf

void discardShieldBooster(int i). Se intenta descartar el potenciador de escudo con índice i de la colección de potenciadores de escudo en uso. Además de perder el potenciador de escudo, se debe actualizar el daño pendiente (pendingDamage) si es que se tiene alguno.
Te puedes basar en el diagrama anterior para implementar este método.

GameUniverse

void init(ArrayList<String> names). Este método inicia una partida. Recibe una colección con los nombres de los jugadores. Para cada jugador, se crea una estación espacial y se equipa con suministros, hangares, armas y potenciadores de escudos tomados de los mazos de cartas correspondientes. Se sortea qué jugador comienza la partida, se establece el primer enemigo y comienza el primer turno.
Su diagrama de secuencia se encuentra en el archivo GameUniverse-init.pdf

boolean nextTurn(). Se comprueba que el jugador actual no tiene ningún daño pendiente de cumplir, en cuyo caso se realiza un cambio de turno al siguiente jugador con un nuevo enemigo con quien combatir, devolviendo true. Se devuelve false en otro caso.
Su diagrama de secuencia se encuentra en el archivo GameUniverse-nextTurn.pdf

CombatResult combat(). Si la aplicación se encuentra en un estado en donde el combatir está permitido, se realiza un combate entre la estación espacial que tiene el turno y el enemigo actual. Se devuelve el resultado del combate.
Su diagrama de secuencia se encuentra en el archivo GameUniverse-combat.pdf

CombatResult combat(SpaceStation station,EnemyStarShip enemy). Se realiza un combate entre la estación espacial y el enemigo que se reciben como parámetros. Se sigue el procedimiento descrito en las reglas del juego: sorteo de quién dispara primero, posibilidad de escapar, asignación del botín, anotación del daño pendiente, etc. Se devuelve el resultado del combate.
Su diagrama de secuencia se encuentra en el archivo GameUniverse-combat.pdf

Para finalizar

Ya se ha completado el juego. Solo queda crear pequeños programas principales para probar los últimos métodos implementados en esta práctica en búsqueda de posibles errores y ya se puede pasar a jugar unas partidas.

Comprobando lo aprendido hasta ahora

Una vez finalizada esta práctica y las anteriores deberías saber y entender los siguientes conceptos:

- Los indicados en el guion anterior.
- Saber interpretar e implementar un diagrama de secuencia.
- Saber diseñar e implementar pequeños programas para probar métodos concretos.
- Saber localizar un error y corregirlo. Saber usar el depurador como ayuda a la localización del error.