**FIUBA - 75.07**

**Algoritmos y programación III**

*Trabajo práctico 2: Dragon AlgoBall*

1er cuatrimestre, 2017

(trabajo grupal de 4 integrantes)

Alumnos:

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Padrón** | **Mail** |
| ADANALIAN, Juan | 98007 | juan90109@gmail.com |
| CABO, Alexis | 86248 | alexis.cabo.87@gmail.com |
| MACIA, Tomás | 99248 | jalt1996@gmail.com |
| TELIAS, Ezequiel | 98600 | eze-telias@hotmail.com |

***Fecha de entrega final***: 22 de junio

***Tutor***: Diego Sánchez

***Comentarios***:

# 

# 

Introducción

# Objetivo del trabajo

Aplicar los conceptos enseñados en la materia a la resolución de un problema, trabajando en forma grupal y utilizando un lenguaje de tipado estático (Java)

# Consigna general

Desarrollar la aplicación completa, incluyendo el modelo de clases e interfaz gráfica. La aplicación deberá ser acompañada por pruebas unitarias e integrales y documentación de diseño. En la siguiente sección se describe la aplicación a desarrollar.

# Descripción de la aplicación a desarrollar

Se deberá desarrollar una aplicación que implemente un juego relacionado con el clásico anime Dragon Ball Z. Todos los personajes presentan las siguientes características en cualquiera de sus modos:

* Nombre
* Puntos de vida
* Poder de pelea
* Distancia de ataque
* Velocidad de desplazamiento
* Ki

Todos los personajes tienen un modo normal y modos especiales, que serán distinto para cada uno.

### Guerreros Z

1. Son Goku, (Kakarotto hijo de Bardok). Líder de los Guerreros Z en la tierra. Tiene un espíritu inquebrantable, y vive en una constante búsqueda de un oponente más poderoso, por lo que se vuelve más fuerte cuando más difícil está la batalla.

|  |  |  |  |
| --- | --- | --- | --- |
| **Goku (\*\*)** | Normal | Kaio-Ken  [Cuesta 20 unidades de Ki] | Super Sayajin  [Cuesta 50 unidades de Ki] |
| Ptos de Vida | 500 | | |
| Poder de pelea | 20 | 40 | 60 |
| Distancia de ataque | 2 | 4 | 4 |
| Velocidad | 2 | 3 | 5 |
| Ki inicial | 0 | | |
| Ataque especial | Kamehameha [Cuesta 20 unidades de Ki] (\*) | | |

*(\*)causa un 50% más de daño que el ataque básico.*

*(\*\*)cuando Goku tiene menos del 30% de vida, su daño global aumenta 20%*

1. Gohan, Hijo de Goku; tiene un poder oculto el cual solo es revelado cuando realmente se enoja.

|  |  |  |  |
| --- | --- | --- | --- |
| **Gohan** | Normal | Super Sayajin fase 1  [Cuesta 10 unidades de Ki] | Super Sayajin fase 2  ([Cuesta 30 unidades de Ki] + (\*)) |
| Ptos de Vida | 300 | | |
| Poder de pelea | 15 | 30 | 100 |
| Distancia de ataque | 2 | 2 | 4 |
| Velocidad | 2 | 2 | 3 |
| Ki inicial | 0 | | |
| Ataque especial | Masenko [Cuesta 10 unidades de Ki] (\*\*) | | |

*(\*) Gohan puede liberar su poder oculto cuando sus otros dos compañeros tienen menos del 30% de vida.*

*(\*\*)causa un 25% más de daño que el ataque básico.*

1. Piccolo, Luchador del planeta Namek. Tiene poderes regenerativos. Tiene profundo afecto hacia Gohan.

|  |  |  |  |
| --- | --- | --- | --- |
| **Piccolo** | Normal | Fortalecido  [Cuesta 20 unidades de Ki] | Protector  ((\*)) |
| Ptos de Vida | 500 | | |
| Poder de pelea | 20 | 40 | 60 |
| Distancia de ataque | 2 | 4 | 6 |
| Velocidad | 2 | 3 | 4 |
| Ki inicial | 0 | | |
| Ataque especial | Makankosappo [Cuesta 10 unidades de Ki] (\*\*) | | |

*(\*) solo puede entrar en este modo cuando Gohan está por debajo de los 20% de vida.*

*(\*\*) causa un 25% más que el ataque básico.*

### Enemigos de la Tierra.

1. Cell, Bio-androide creado por el Dr. Gero. Puede absorber vida

|  |  |  |  |
| --- | --- | --- | --- |
| **Cell** | Normal | Semi-perfecto  ((\*)) | Perfecto  ((\*\*)) |
| Ptos de Vida | 500 | | |
| Poder de pelea | 20 | 40 | 80 |
| Distancia de ataque | 3 | 4 | 4 |
| Velocidad | 2 | 3 | 4 |
| Ki inicial | 0 | | |
| Ataque especial | Absorber [Cuesta 5 unidades de Ki] (\*\*\*) | | |

*(\*) debe absorber vida 4 veces*

*(\*\*) debe absorber vida 8 veces*

*(\*\*\*) causa la misma cantidad de daño que el ataque básico, pero Cell aumenta su vida en la misma cantidad.*

1. Freezer, malvado alienígena responsable de la destrucción del planeta de la raza Sayajin.

|  |  |  |  |
| --- | --- | --- | --- |
| **Freezer** | Normal | Segunda forma  [Cuesta 20 unidades de Ki] | Definitivo  [Cuesta 50 unidades de Ki] |
| Ptos de Vida | 400 | | |
| Poder de pelea | 20 | 40 | 50 |
| Distancia de ataque | 2 | 3 | 3 |
| Velocidad | 4 | 4 | 6 |
| Ki inicial | 0 | | |
| Ataque especial | Rayo mortal [Cuesta 20 unidades de Ki] (\*) | | |

*(\*)causa un 50% más de daño que el ataque básico.*

1. Majin Boo, criatura creada por Bibidi, con la habilidad de convertir a cualquiera en chocolate.

|  |  |  |  |
| --- | --- | --- | --- |
| **Majin Boo** | Normal | Boo Malo  [Cuesta 20 unidades de Ki] | Boo Original  [Cuesta 50 unidades de Ki] |
| Ptos de Vida | 300 | | |
| Poder de pelea | 30 | 50 | 60 |
| Distancia de ataque | 2 | 2 | 3 |
| Velocidad | 2 | 3 | 4 |
| Ki inicial | 0 | | |
| Ataque especial | Conviértete en chocolate [Cuesta 30 unidades de Ki] (\*) | | |

*(\*) inutiliza a un personaje enemigo durante 3 turnos, (este tampoco gana Ki en ese tiempo).*

Los personajes de un mismo equipo no pueden atacarse entre ellos.

La distancia de ataque se mide en casilleros. Por ejemplo GOKU en modo Normal posee distancia de ataque = 2 significa que podrá atacar a cualquier enemigo que se encuentre en un casillero verde, no así en los celestes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | **GOKU** |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Consumibles

Una vez que el personaje (en cualquier modo) captura un Consumible el mismo es consumido por el personaje y desaparece del mapa.

Estos consumibles aparecen en cualquier lugar vacío del mapa, aleatoriamente en cualquier momento del juego.

1. **Semilla del ermitaño**: regenera 100 puntos de vida al que la consuma.
2. **Esfera del dragon**: los próximos 2 ataques (de el personaje que agarre la esfera) causan 25% más de daño.
3. **Nube voladora**: en los próximos dos turnos, el personaje que agarre la nube, tiene el doble de velocidad.

Jugabilidad

Es un juego por turnos. Hay 2 jugadores, cada uno debe elegir un equipo antes de iniciar una partida. Cada jugador comienza la partida con sus 3 personajes.

Sistema de Ataque:

* Cada personaje tiene un ataque básico (no cuesta Ki) y un ataque especial (tiene costo) que puede aplicar un efecto.
* Ataque básico: el daño del ataque básico es igual al poder de pelea actual del personaje (el poder de pelea no cambia más que en transformaciones, pero *el ataque puede variar por algún efecto*).
* ***[IMPORTANTE]* *Al atacar a un enemigo de mayor poder de pelea, el daño de dicho ataque se reduce un 20%.***

Ki:

* El Ki se aumenta solo en cada personaje, en una cantidad de 5 por turno (cuando empieza el turno se aumenta).

Acciones en turno:

* En un mismo turno, se puede realizar una acción de ataque y una acción de movilidad, no necesariamente del mismo personaje.

El juego elige al azar qué jugador comienza. Cada jugador inicia en el extremo opuesto al otro con sus 3 personajes juntos.

Tablero

El juego tiene lugar en un tablero compuesto de casilleros. El tamaño, forma y cantidad de casilleros del tablero queda a definir por cada grupo y acordado con su ayudante.

Todos los personajes ocupan 1 casillero en cualquiera de sus modos. No puede haber más de 1 personaje en un casillero. Hay 1 o 0, nunca 2, 3, etc. independientemente del equipo al que pertenezcan.

Los personaje se desplazan por el tablero de casillero en casillero. Cada punto de su velocidad de desplazamiento representa 1 casillero.

Fin del juego

Para ganar se debe destruir a todos los personajes enemigos.

*[OPCIONAL]* Fin del juego alternativo (suma puntaje extra)

Para ganar el juego sin tener que matar a todos los enemigos, se pueden capturar 7 esferas del dragón entre cualquiera de los 3 personajes que tenga. (es decir el conteo de esferas es global, no de cada personaje particular.).

## Interfaz gráfica

Se debe desarrollar una interfaz visual para la interacción entre los jugadores. En la misma se pondrá mucho énfasis y se evaluará como parte de la consigna la **USABILIDAD** de la misma.

**Cada vez que le toque el turno a cada jugador la vista del mapa debe centrarse en el lugar donde utilizó a su último personaje.**

Entregables

1. Código fuente de la aplicación completa, incluyendo también: código de la pruebas, archivos de recursos
2. Script para compilación y ejecución (ant)
3. Informe, acorde a lo especificado en este documento

# Formas de entrega

Habrá **3 entregas formales** que tendrán una calificación de **APROBADO o NO APROBADO** en el momento de la entrega.

Aquél grupo que acumule 2 no aprobados, quedará automáticamente desaprobado con la consiguiente pérdida de regularidad en la materia. *En cada entrega se debe traer el informe* ***actualizado*** *impreso*.

# Evaluación

El día del vencimiento de cada entrega, cada ayudante convocará a los integrantes de su grupo, solicitará el informe correspondiente e iniciará la corrección mediante una entrevista grupal.

**Es imprescindible la presencia de todos los integrantes del grupo el día de cada corrección.**

Se evaluará el trabajo grupal y a cada integrante en forma individual. El objetivo de esto es comprender la dinámica de trabajo del equipo y los roles que ha desempeñado cada integrante del grupo. Para que el alumno apruebe el trabajo práctico debe estar aprobado en los dos aspectos: grupal e individual.

Dentro de los ítems a chequear el ayudante evaluará aspectos formales (como ser la forma de presentación del informe), aspectos funcionales: que se resuelva el problema planteado y aspectos operativos: que el TP funcione integrado.

# Casos de prueba para cada entrega

1ra Entrega (5 de junio - 8 de junio)

Turnos, Jugadores, Unidades, Tablero, casillero, transformaciones con costo de Ki únicamente

1. Se ubica un personaje en un casillero, se pide que se mueva, se verifica nueva posición acorde a su modo.
2. Se verifica que no puede haber 2 personajes en el mismo casillero
3. Se ubican 2 personajes y se verifica de que no puede pasar uno encima del otro.
4. Se ubica un personaje se lo transforma, se verifica que se pueda transformar.
5. Se ubica un personaje, realiza una transformación y se pide que se mueva y se verifica que su nueva posición sea acorde.
6. Crear una prueba de integración en la cual se pueda crear un juego, con 2 jugadores cada uno de ellos con sus 3 personajes distribuidos en el tablero según el enunciado.
7. Combinaciones en modos de: Ubicar un Guerrero Z, ubicar un Enemigo de la Tierra, pedir que se ataquen respetando ( y no ) las distancias verificando los daños ( o no daños ).

2da Entrega (12 de junio - 15 de junio)

1ra entrega + transformaciones con requisitos especiales + ataques especiales con efectos

1. Posicionar a los Guerreros Z, llevar a Gohan a su primera transformación, comprobar que no puede llegar a la segunda.
2. Posicionar a los Guerreros Z, dejar a Goku y Piccolo con la vida necesaria para la transformación de Gohan, y comprobar que se realiza correctamente.
3. Posicionar a Gohan y Piccolo y llevar a este último a su primera transformación, luego comprobar que no puede llegar a la segunda.
4. Posicionar a Gohan y Piccolo, dejar a Gohan con la vida necesaria para que Piccolo pueda realizar su segunda transformación, comprobar que los cambios se hicieron correctamente.
5. Posicionar a Cell y comprobar que no puede realizar sus transformaciones.
6. Posicionar a Cell y a un Guerrero Z, comprobar el correcto funcionamiento de “absorber vida”.
7. Posicionar a Cell y a un Guerrero Z, efectuar “absorber vida” las veces necesarias para que pueda transformarse y comprobar que los cambios se hicieron correctamente.
8. Posicionar a Majin Boo, y a un Guerrero Z, comprobar el correcto funcionamiento de “Conviértete en chocolate”
9. Posicionar a Goku y verificar que su daño aumenta, al reducir su vida por debajo del 20%.

3ra y última Entrega: (19 de junio - 22 de junio)

2da entrega + Consumibles. Trabajo Práctico completo funcionando, con interfaz gráfica final, sonidos e informe completo.

1. Ubicar un consumible y un personaje en casilleros diferentes, y verificar que el consumible desaparece, al mover el personaje al mismo casillero.
2. Repetir 1 y verificar que se aplica correctamente el efecto de cada consumible (un test por cada consumible).

|  |
| --- |
| **Tiempo total de desarrollo del trabajo práctico**: 4 semanas. |

Informe

## **Supuestos**

* Los personajes no pueden regenerar su vida por encima de su vida inicial.
* Cell no puede aumentar su vida al realizar ataque especial absorber si al sumar el daño infligido supera su vida máxima. Sin embargo, el ataque especial se ejecuta como tal.

## **Modelo de dominio**

El modelo consta de entidades que buscan adaptar de la mejor manera posible el problema del trabajo práctico. Para ello se planteó el problema en tres partes: la primera con los personajes y sus distintos modos o transformaciones, una segunda parte que incluye a los casilleros en los que se deben mover y atacar los personajes, y una última parte con la lógica del juego a desarrollar, con los turnos, la partida y los dos equipos involucrados, que manejaran a los Guerreros Z y a los Enemigos de la Tierra.

Para los personajes, se creó una clase abstracta Personaje, de la cual todos los personajes heredan, y en ella están todas las acciones que pueden realizar quienes hereden de ella. Luego cada personaje en particular tiene su modo, y su vida (común a todos los modos). Los modos o transformaciones están implementados con el patrón State (ver detalles de implementación en el informe).

La segunda parte consta de un paquete en el que se incluye a las clases que conforman la base sobre la cual los personajes se mueven. Los personajes se posicionan sobre casilleros, y a estos se accede mediante coordenadas determinadas en la clase Posición (ver explicación detallada de implementación en el informe). Además, existe una clase Camino que es un conjunto de casilleros continuos que modela la trayectoria que debe seguir un personaje para moverse por los casilleros. También consta de una clase Tablero que mantiene referencia a todos los casilleros que se crean una única vez al inicio de una partida.

Una tercera y última parte que contiene a aquellas clases que son menos visibles a primera instancia, pero no por ellos menos importantes. Entre ellas están la clase Equipo, que representa a los dos equipos presentes en nuestro juego: Equipo de Guerreros Z y Equipo de Enemigos de la Tierra. En estas clases se guardan todos los personajes relacionados a estos dos equipos. Luego se encuentra la clase Partida, en la que se inicializa el conjunto de casilleros en un Tablero, con los personajes situados en extremos opuestos, íntimamente relacionada a la clase Turno, donde se maneja el turno actual del juego, correspondiente a uno de los dos equipos.

## **Diagramas de clases**

*[Varios diagramas de clases, mostrando la relación estática entre las clases, pueden agregar todo el texto necesario para aclarar y explicar su diseño, recuerden que la idea de todo el documento es que quede documentado y entendible como está hecho el TP]*

## **Diagramas de secuencia**

*[Varios diagramas de secuencia, mostrando la relación dinámica entre las clases planteando una gran cantidad de escenarios que contemplen las situaciones del trabajo práctico]*

## **Diagrama de paquetes**

*[incluir un diagrama de paquetes para mostrar el acoplamiento de su trabajo ]*

## **Diagramas de estado**

*[Incluir diagramas de estados, mostrando tanto los estados como las distintas transiciones de los mismos para varias entidades del trabajo práctico ]*

## **Detalles de implementación**

-Para resolver el problema de los distintos modos se usa el patrón State, entonces cada personaje tiene un atributo transformación que es una clase abstracta, de la cual extienden los distintos modos que admite cada personaje. En cada una de estas implementaciones de los modos, se agregan aquellos atributos que varían de acuerdo al modo en que se encuentre el personaje. La razón por la cual elegimos implementar las transformaciones con una clase abstracta en lugar de una interfaz, es la cantidad de métodos que habría que implementar en cada modo, que son métodos compartidos por todos los modos. De esta forma reutilizamos los métodos comunes a todos, y aquellos que varían de acuerdo a la transformación, se reemplaza el método de la clase abstracta Transformación por la condición específica para dicha transformación. Por ejemplo: la mayoría de las transformaciones solo requieren una cantidad fija de ki, entonces en la clase abstracta Transformación el método transformar hace la transformación consumiendo el ki. En cambio, un personaje específico como Cell, las transformaciones no dependen del ki sino de haber realizado otra acción previamente, entonces en dichas transformaciones el método transformar se fija si se cumple la condición y transforma al personaje en cuestión.

-La clase Tablero está implementada como un mapa de Hash, donde la clave es la clase Posición, y el valor almacenado es la clase Casillero. Esto permite tener un sistema de coordenadas de los casilleros y poder acceder a ellos de una forma más cómoda y simple. Gracias al uso del sistema de coordenadas (la clase Posición), podemos calcular las distancias entre dos puntos más fácilmente.

-Para los ataques especiales en primera instancia se pensó implementar con double dispatch, pero requería de completar el ataque especial en cada personaje que existe, y ante un posible nuevo personaje que se agregue en un futuro, habría que crear nuevos métodos en cada clase. Por ello se decidió implementar el ataque especial con Table-Driven Methods, que consiste en una tabla de hash cuya clave es la clase a la que se quiere atacar y el valor es el método que se ejecuta para atacar a dicho personaje. De esta forma, ante la llegada de un nuevo personaje, simplemente agregamos una nueva entrada al hash si es necesario. O bien si un personaje ya no recibe daño ante el ataque especial de un personaje, simplemente bastará con reemplazar el valor de la clave (de la clase).

-Para los ataques básicos, como todos los personajes comparten el mismo ataque básico, sólo difieren en el daño producido por la transformación en la que se encuentren, se implementó una clase Ataque que representa a la acción del ataque en el juego, y que implementa una interfaz Command, con el método execute. De esta forma cada vez que se quiere atacar, se instancia un Ataque y para ejecutar el ataque al atacado se le envía el mensaje execute.

-Para el ataque especial de MajinBoo (inutilizar a un personaje durante sus siguientes 2 turnos), se intentó una especie de Proxy, siendo que toda acción que es sometida al ataque especial de MajinBoo (ataqueBasico, ataqueEspecial, Mover y Transformar) son delegadas a la clase transformación de dicho personaje, se crea una nueva transformación "chocolate" que es cambiada por la actual transformación y la cual implementa esos 4 métodos lanzando las correspondientes excepciones sin hacer nada más. Además, cuenta los turnos restantes y guarda en si misma la transformación original del personaje para poder devolverla al finalizar el efecto.

(Cuando desde la partida se pasa de turno, el equipo que termino su turno revisa sus personajes para reducir el tiempo restante o eliminar el efecto del chocolate si ya pasaron los turnos correspondientes)

-Para la segunda transformación de Gohan (SuperSayajin1 a SuperSayajin2) es necesario que sus compañeros (Goku y Piccolo) estén en menos del 30% de vida. Así que al querer transformar a Gohan, Se revisa si existen estos aliados, y de ser así le pregunta a cada uno si cumplen la condición de vida necesaria para que él se transforme. Si cumplen todo esto se permite transformar a Gohan. (como cada personaje tiene guardado su equipo, y el equipo tiene referencia a todos los personajes integrantes, se pueden relacionar)

-Lo que sería la parte de la vista del juego, se tiene la clase Aplicacion que engloba todo lo necesario, como ser:

CaminoController: Es quien se encarga de ir guardando los casilleros seleccionados para generar el camino cuando sea necesario (al pedir mover un personaje)

PersonajeController: Misma idea de caminoController, pero para guardar los personajes seleccionados. Los mismos se utilizan para comunicarle a los handler de Mover, Atacar (básico y especial) y Transformar. Por ejemplo: cuando pulse el botón de atacar, los participantes de ese ataque son los previamente seleccionados, siendo el principal -atacante- el que se muestra en el panel lateral de información.

ContenedorPrincipal: Aquí se guarda la barra de menú superior, la vista del tablero, la consola, y el panel lateral con los botones y el panel de información.

Cada botón tiene asignado un handler especifico que se encarga de ejecutar la acción esperada, siendo ellos quien interactúan con el modelo.

-Para la generación aleatoria de consumibles, existe una clase GeneradorDeConsumiblesRandom que se ejecuta al pasar de turno en la partida, y tiene un 25% de probabilidad de generar uno de los 3 posibles consumibles en alguna posición aleatoria del tablero que no esté ocupada por algún personaje.

-Los efectos en el tiempo que generan los consumibles (doble de velocidad o aumento de daño) son guardados en la transformación del personaje que los consume, y ese "efecto" es usado a la hora de que la transformación calcule el daño o velocidad (para saber la velocidad, la transformación le pregunta a su efecto por lo que, si no se consumió nada, la transformación tiene el efecto "SinEfectoEspecial" que simplemente devuelve el numero original esperado. En el caso de haber consumido, por ejemplo, una NubeVoladora, el efecto pasa a ser éste, y al preguntarle por la velocidad devuelve el doble de la original).

## **Excepciones**

Las excepciones creadas son todas excepciones chequeadas que heredan de Exception. Se eligió de esta forma porque son todas excepciones que son provocadas por el usuario y no por el programador. Es el usuario quien no proporciona las condiciones deseadas de ejecución y no una falla del programador.

Las excepciones creadas son las siguientes:

-***DimensionDeTableroInvalidoException***: se lanza cuando se quiere crear un tablero con dimensiones negativas.

-***CasilleroOcupadoException***: se lanza cuando se quiere posicionar un personaje en un casillero ya ocupado.

-***NoPuedeMoverAEsaDistanciaException***: se lanza cuando se quiere mover un personaje más allá de su capacidad en velocidad.

-***NoPuedeMoverCaminoObstruidoException***: se lanza cuando se quiere mover un personaje y en el camino entre el casillero origen y el destino hay otro personaje posicionado.

-***NoPuedeAtacarAEsaDistanciaException***: se lanza cuando se quiere atacar a un oponente que se encuentra a una distancia superior a la distancia de ataque del personaje agresor.

-***NoPuedeTransformarException:*** se lanza cuando no se cumple alguna condición especial de transformación para los personajes.

***-KiInsuficienteException***: se lanza cuando se quiere transformar un personaje a otro modo, o realizar un ataque especial de algún personaje que requiera Ki, pero el personaje no posee el Ki suficiente para hacerlo.

-***NoHayProximaTransformacionException***: se lanza cuando se quiere transformar un personaje a un modo superior pero dicho modo no existe (el personaje está en su última transformación).

-JuegoTerminadoException

-NoPuedeAtacarMismoEquipoException

-NoPuedeAtacarSiendoChocolateException

-NoPuedeMoverseSiendoChocolateException

-NoPuedeTransformarseSiendoChocolateException

-NoPuedeUsarAtaqueEspecialSiendoChocolate

-YaAtacasteEsteTurnoException

-YaMovisteEsteTurnoException

-NoEsSuTurnoException

-NoPuedeAtacarAEsaDistanciaException

-NoPuedeMoverAEsaDistanciaException

-NoPuedeMoverCaminoObstruidoException

-NoSePuedeSeleccionarDosPersonajesDelMismoEquipoException

-NoSePuedeSeleccionarMasDeDosPersonajesException

-CasillaSinPersonajeException

-NoSeSeleccionoNingunPersonajeException

-CaminoInvalidoException

-CasilleroNoContiguoException

-HuecoEntreCaminoYPersonajeException

-CasilleroOcupadoException

-DimensionDeTableroInvalidoException

-KiInsuficienteException

-NoHayProximaTransformacionException

-NoPuedeTransformarException

AZUL: MERGE DE EXCEPCIONES EN UNA (EJ:ESCHOCOLATEEXCEPTION)

VERDE: OK

ROJO: ELIMINAR (si son del mismo equipo del turno, el ataque lanza NoPuedeAtacarMismoEquipoException. Si son del otro equipo, debería lanzar NoEsSuTurnoException)

NEGRO: No estoy seguro. Analizar si son necesarias

TODO: (Lo que nos recomendó Tomás el ayudante)Hacer todas las excepciones extender de RuntimeException así los métodos no se llenan de excepciones en sus firmas.