# FIUBA - 75.07

# Algoritmos y programación III

*Trabajo práctico 2: Dragon AlgoBall*

1er cuatrimestre, 2017

(trabajo grupal de 4 integrantes)

Alumnos:

| Nombre | Padrón | Mail |
|---|---|---|
| Jamilis Netanel | | |
| Scakosky Matias | | |
| Sinisi Fernando | | |
| Varela Agustina | | |


*Fecha de entrega final*: 26 de junio

*Tutor*: Marcio Degiovannini

*Comentarios*:

# Informe

## Supuestos

*Un supuesto que tuvimos en cuenta es que los usuarios van a ir avanzando a los personajes casillero por casillero, en cada paso eligiendo a qué casillero contiguo se quieren trasladar.*

*También supusimos tanto a los personajes como a los consumibles como obstáculos en el tablero que no se puede atravesar, en el caso de los consumibles se puede caer encima para tomarlos.*

## Modelo de dominio

*Para comenzar creamos el objeto tablero, en el se desarrollara todo el juego. Este no es más que un conjunto de casilleros relacionados entre sí, cada uno con su respectiva posición. La posición es un objeto el cual nos permite saber la ubicación de cada casillero en nuestro tablero.*

*Por este motivo nos vimos obligados a crear también un objeto casillero. El objeto casillero tiene dos estados posibles, vacío o ocupado.*

*En el estado ocupado puede contener un único objeto del juego a la vez, este puede ser un personaje o un consumible.*

*Debido a que tanto los personajes como los consumibles son para el casillero una misma cosa, objetos del juego que los contiene, entonces nos dimos cuenta que necesitábamos de un objeto que abarque a ambos objetos. Este objeto sería de la clase que se puede guardar en un casillero.*

*A continuación, ya que el jugador se iba a encontrar objetos del tipo consumible al recorrer el tablero, entonces nos dimos cuenta de que nosotros necesitábamos tener a ese objeto en nuestro juego. Este objeto consumible abarcara a todas las formas en que se puede presentar un consumible.*

*En nuestro juego los consumibles podrán presentarse como semilla del ermitaño, como esfera del dragón o como nube voladora. Estos consumibles tendrán las características propias de cualquier consumible más las propias de la forma en que se presentan.*

*Nuestro otro objeto del juego posible, el cual va a poder ocupar un casillero, es el personaje. Todos los personajes de nuestro juego van a compartir ciertas propiedades, van a poder realizar acciones comunes típicas de un personaje.*

*En nuestro juego, todo personaje va a  ser un GuerreroZ o un Enemigo De La Tierra. Dependiendo a qué grupo de esos pertenezca va a tener va a compartir ciertas cualidades.*

*Los personajes que se van a presentar con la forma de GuerreroZ son Goku, Gohan y Picolo, y los que se van a presentar con la forma de enemigos de la tierra son Cell, Freezer y MajinBoo.*

*Cada personaje va a tener un estado posible que va a estar definido por las características del personaje y el estado de esas. Las características del personaje vienen dadas por la fase de evolución en que se encuentra y los cambios que sufrió este en el transcurso de la partida.*

*Cada personaje tiene tres estados de evolución posible, cada uno de estos viene dados por su respectiva fase de evolución.*

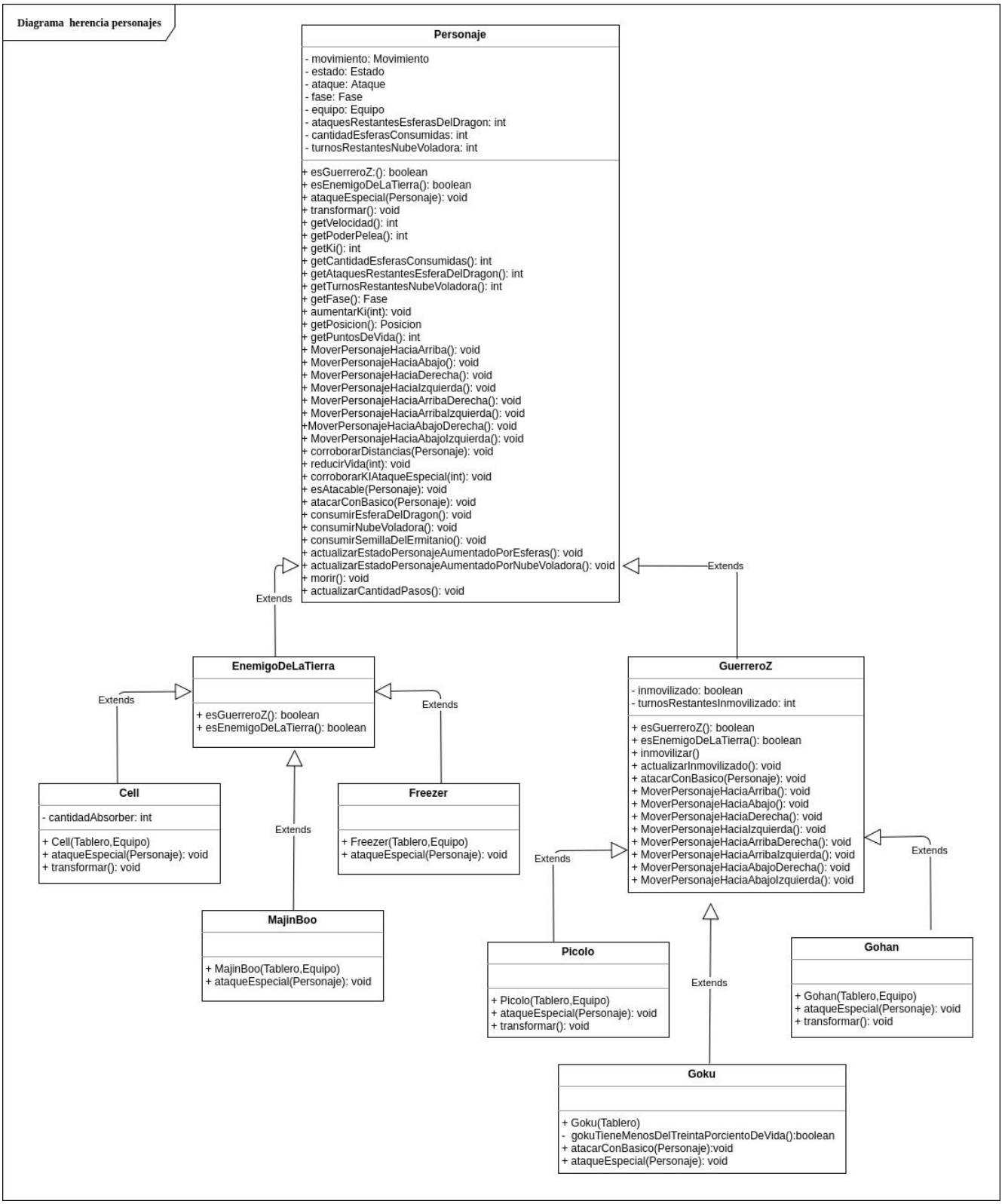
*Los datos que toman las fases para poder realizar las evoluciones están almacenados en una base de datos, que contiene los estados del juego.*

*Cómo el juego se va a realizar por equipos, vamos a necesitar tener un equipo. Este equipo va a estar conformado por los guerrerosZ o por los enemigos de la tierra. A este equipo se le va a poder consultar la cantidad de esferas que capturaron sus miembros, entre otras cosas.*

*El movimiento del personaje se va a poder realizar en una de las ocho direcciones posibles, al casillero contiguo del que se encuentra actualmente.*

*Mediante el turno el usuario va a ir avanzando en el juego. Va a poder controlar todas las acciones, manejar con que personaje desea realizar cada acción, verificar si se termino el juego, etc. El turno se va a personalizar dependiendo de a qué equipo pertenece el usuario.*

# Diagramas de clases



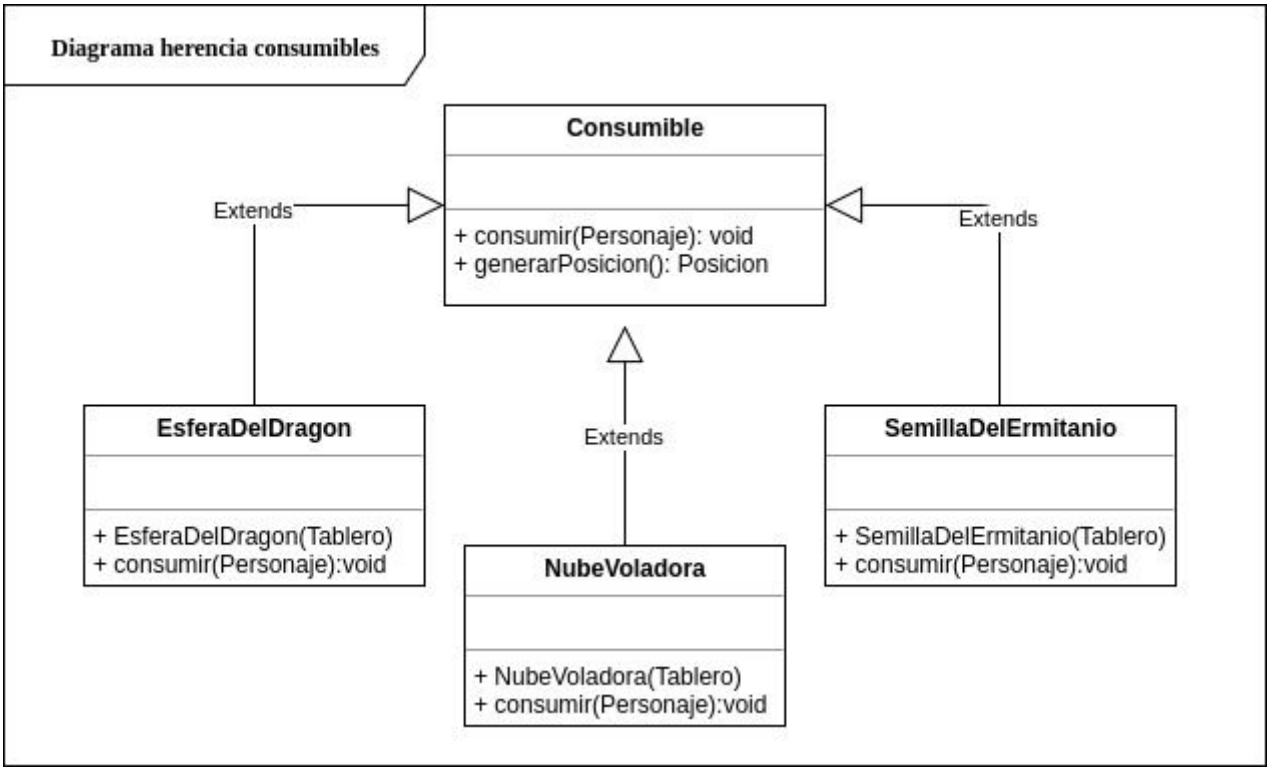


Diagrama herencia consumibles
Consumible
+ consumir(Personaje): void
+ generarPosicion(): Posicion
Extends
Extends
Extends
EsferaDelDragon
+ EsferaDelDragon(Tablero)
+ consumir(Personaje):void
NubeVoladora
+ NubeVoladora(Tablero)
+ consumir(Personaje):void
SemillaDelErmitanio
+ SemillaDelErmitanio(Tablero)
+ consumir(Personaje):void

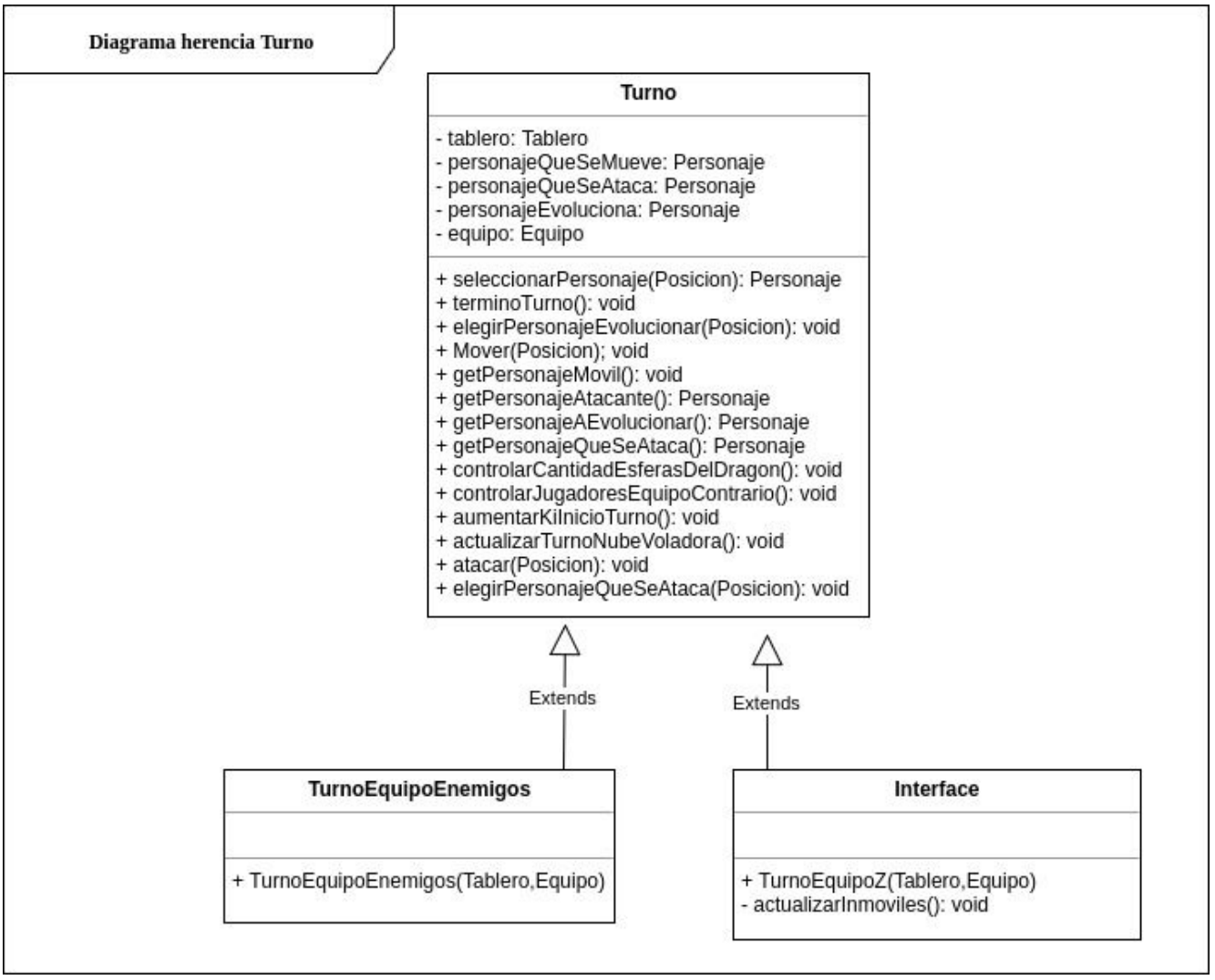


Diagrama herencia Turno
Turno
- tablero: Tablero
- personajeQueSeMueve: Personaje
- personajeQueSeAtaca: Personaje
- personajeEvoluciona: Personaje
- equipo: Equipo
+ seleccionarPersonaje(Posicion): Personaje
+ terminoTurno(): void
+ elegirPersonajeEvolucionar(Posicion): void
+ Mover(Posicion); void
+ getPersonajeMovil(): void
+ getPersonajeAtacante(): Personaje
+ getPersonajeAEvolucionar(): Personaje
+ getPersonajeQueSeAtaca(): Personaje
+ controlarCantidadEsferasDelDragon(): void
+ controlarJugadoresEquipoContrario(): void
+ aumentarKiInicioTurno(): void
+ actualizarTurnoNubeVoladora(): void
+ atacar(Posicion): void
+ elegirPersonajeQueSeAtaca(Posicion): void
Extends
Extends
TurnoEquipoEnemigos
+ TurnoEquipoEnemigos(Tablero,Equipo)
Interface
+ TurnoEquipoZ(Tablero,Equipo)
- actualizarInmoviles(): void

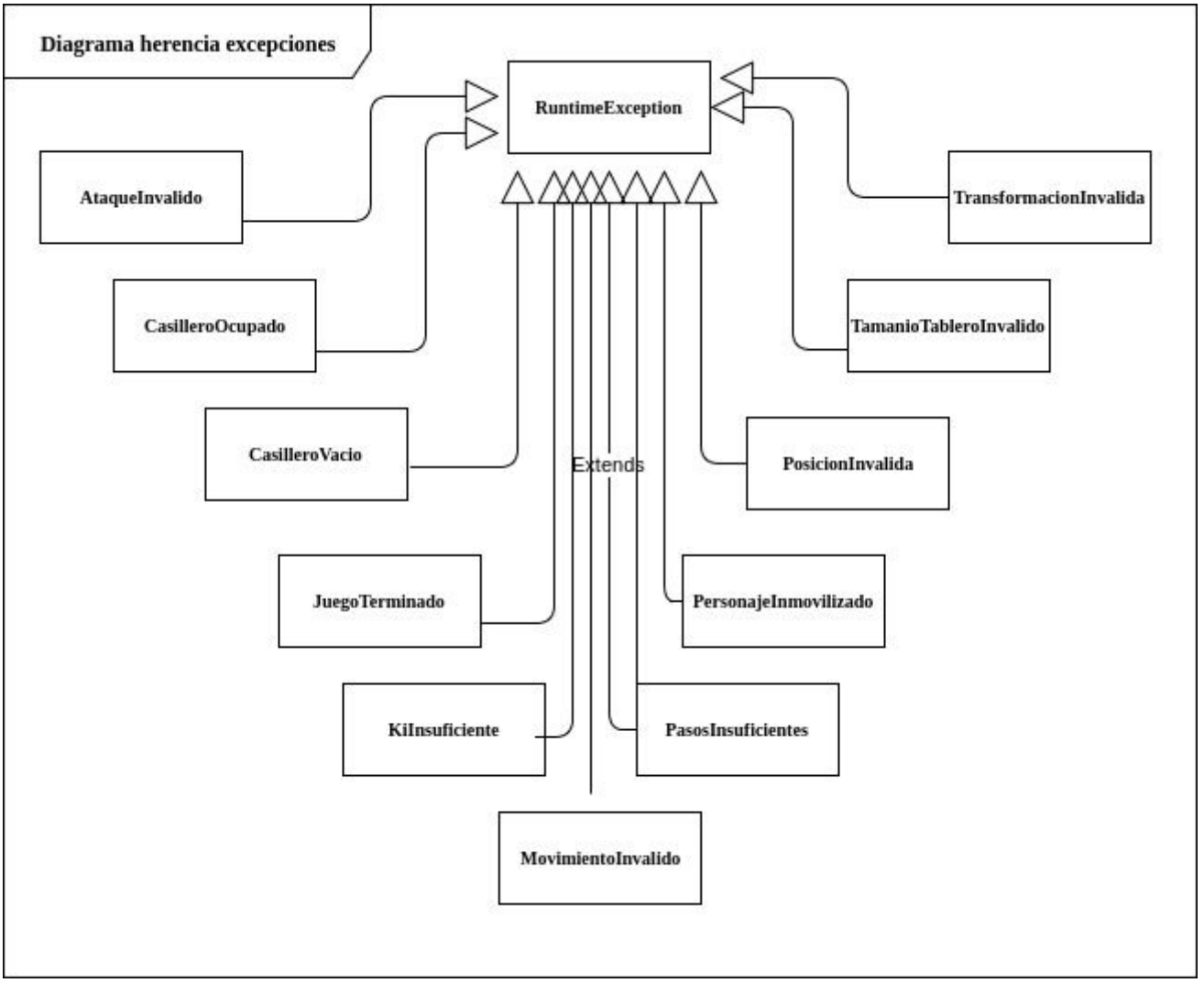


Diagrama herencia excepciones
RuntimeException
AtaqueInvalido
TransformacionInvalida
CasilleroOcupado
TamanioTableroInvalido
CasilleroVacio
Extends
PosicionInvalida
JuegoTerminado
PersonajeInmovilizado
KiInsuficiente
PasosInsuficientes
MovimientoInvalido

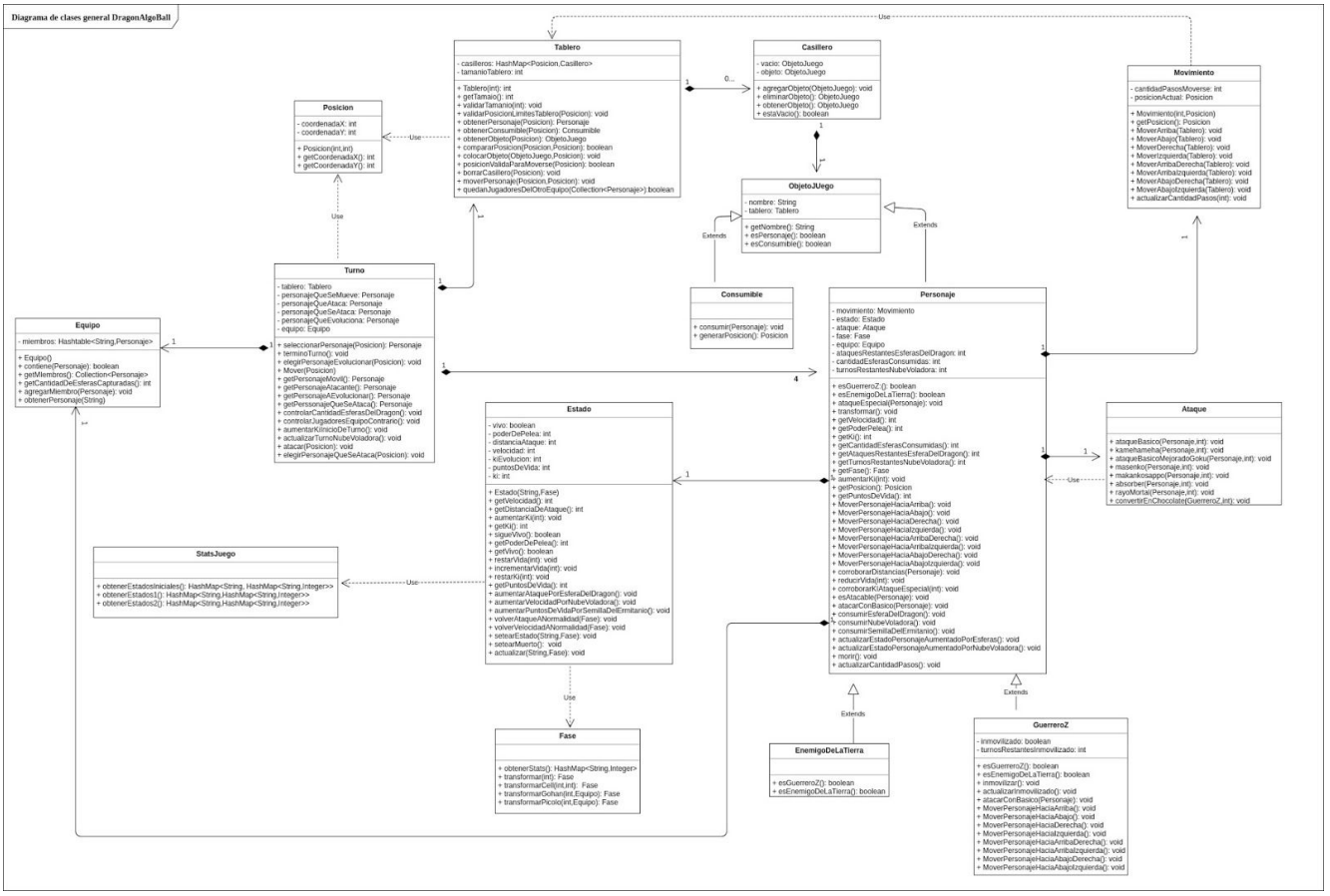
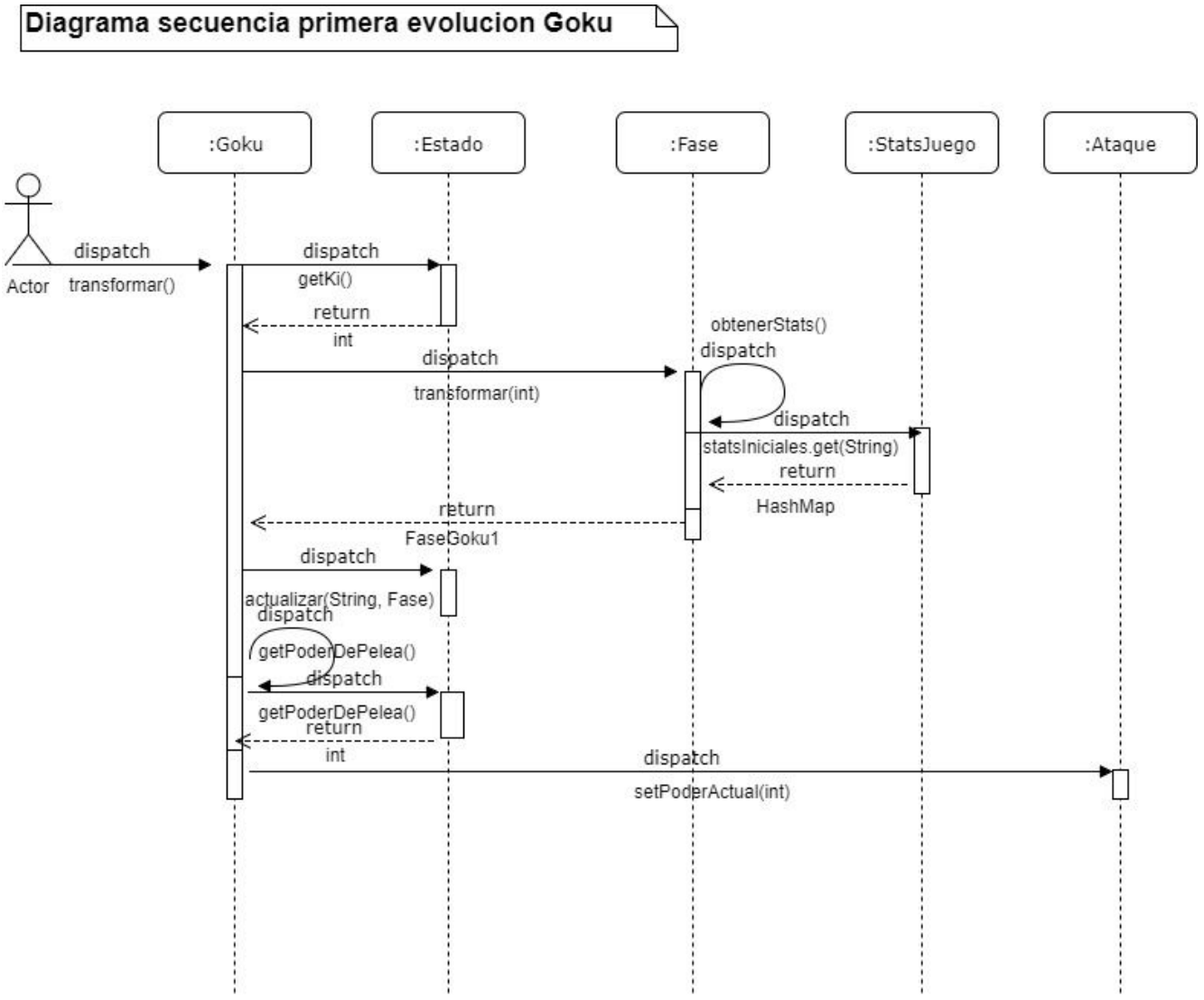


Diagrama de clases general DragonAlgoBall
Tablero
- casilleros: HashMap<Posicion,Casillero>
- tamanioTablero: int
+ Tablero(int): int
+ getTamaio(): int
+ validarTamanio(int): void
+ validarPosicionLimitesTablero(Posicion): void
+ obtenerPersonaje(Posicion): Personaje
+ obtenerConsumible(Posicion): Consumible
+ obtenerObjeto(Posicion): ObjetoJuego
+ compararPosicion(Posicion,Posicion): boolean
+ colocarObjeto(ObjetoJuego,Posicion): void
+ posicionValidaParaMoverse(Posicion): boolean
+ borrarCasillero(Posicion): void
+ moverPersonaje(Posicion,Posicion): void
+ quedanJugadoresDelOtroEquipo(Collection<Personaje>): boolean
Casillero
- vacio: ObjetoJuego
- objeto: ObjetoJuego
+ agregarObjeto(ObjetoJuego): void
+ eliminarObjeto(): ObjetoJuego
+ obtenerObjeto(): ObjetoJuego
+ estaVacio(): boolean
Posicion
- coordenadaX: int
- coordenadaY: int
+ Posicion(int,int)
+ getCoordenadaX(): int
+ getCoordenadaY(): int
Movimiento
- cantidadPasosMoverse: int
- posicionActual: Posicion
+ Movimiento(int,Posicion)
+ getPosicion(): Posicion
+ MoverArriba(Tablero): void
+ MoverAbajo(Tablero): void
+ MoverDerecha(Tablero): void
+ MoverIzquierda(Tablero): void
+ MoverArribaDerecha(Tablero): void
+ MoverArribaIzquierda(Tablero): void
+ MoverAbajoDerecha(Tablero): void
+ MoverAbajoIzquierda(Tablero): void
+ actualizarCantidadPasos(int): void
ObjetoJUego
- nombre: String
- tablero: Tablero
+ getNombre(): String
+ esPersonaje(): boolean
+ esConsumible(): boolean
Turno
- tablero: Tablero
- personajeQueSeMueve: Personaje
- personajeQueAtaca: Personaje
- personajeQueSeAtaca: Personaje
- personajeQueEvoluciona: Personaje
- equipo: Equipo
+ seleccionarPersonaje(Posicion): Personaje
+ terminoTurno(): void
+ elegirPersonajeEvolucionar(Posicion): void
+ Mover(Posicion)
+ getPersonajeMovil(): Personaje
+ getPersonajeAtacante(): Personaje
+ getPersonajeAEvolucionar(): Personaje
+ getPerssonajeQueSeAtaca(): Personaje
+ controlarCantidadEsferasDelDragon(): void
+ controlarJugadoresEquipoContrario(): void
+ aumentarKiInicioDeTurno(): void
+ actualizarTurnoNubeVoladora(): void
+ atacar(Posicion): void
+ elegirPersonajeQueSeAtaca(Posicion): void
Consumible
+ consumir(Personaje): void
+ generarPosicion(): Posicion
Personaje
- movimiento: Movimiento
- estado: Estado
- ataque: Ataque
- fase: Fase
- equipo: Equipo
- ataquesRestantesEsferasDelDragon: int
- cantidadEsferasConsumidas: int
- turnosRestantesNubeVoladora: int
+ esGuerreroZ(): boolean
+ esEnemigoDeLaTierra(): boolean
+ ataqueEspecial(Personaje): void
+ transformar(): void
+ getVelocidad(): int
+ getPoderPelea(): int
+ getKi(): int
+ getCantidadEsferasConsumidas(): int
+ getAtaquesRestantesEsferaDelDragon(): int
+ getTurnosRestantesNubeVoladora(): int
+ getFase(): Fase
+ aumentarKi(int): void
+ getPosicion(): Posicion
+ getPuntosDeVida(): int
+ MoverPersonajeHaciaArriba(): void
+ MoverPersonajeHaciaAbajo(): void
+ MoverPersonajeHaciaDerecha(): void
+ MoverPersonajeHaciaIzquierda(): void
+ MoverPersonajeHaciaArribaDerecha(): void
+ MoverPersonajeHaciaArribaIzquierda(): void
+ MoverPersonajeHaciaAbajoDerecha(): void
+ MoverPersonajeHaciaAbajoIzquierda(): void
+ corroborarDistancias(Personaje): void
+ reducirVida(int): void
+ corroborarKiAtaqueEspecial(int): void
+ esAtacable(Personaje): void
+ atacarConBasico(Personaje): void
+ consumirEsferaDelDragon(): void
+ consumirNubeVoladora(): void
+ consumirSemillaDelErmitanio(): void
+ actualizarEstadoPersonajeAumentadoPorEsferas(): void
+ actualizarEstadoPersonajeAumentadoPorNubeVoladora(): void
+ morir(): void
+ actualizarCantidadPasos(): void
Ataque
+ ataqueBasico(Personaje,int): void
+ kamehameha(Personaje,int): void
+ ataqueBasicoMejoradoGoku(Personaje,int): void
+ masenko(Personaje,int): void
+ makankosappo(Personaje,int): void
+ absorber(Personaje,int): void
+ rayoMortal(Personaje,int): void
+ convertirEnChocolate(Guerrero/_int): void
Equipo
- miembros: Hashtable<String,Personaje>
+ Equipo()
+ contiene(Personaje): boolean
+ getMiembros(): Collection<Personaje>
+ getCantidadDeEsferasCapturadas(): int
+ agregarMiembro(Personaje): void
+ obtenerPersonaje(String)
Estado
- vivo: boolean
- poderDePelea: int
- distanciaAtaque: int
- velocidad: int
- kiEvolucion: int
- puntosDeVida: int
- ki: int
+ Estado(String,Fase)
+ getVelocidad(): int
+ getDistanciaDeAtaque(): int
+ aumentarKi(int): void
+ getKi(): int
+ sigueVivo(): boolean
+ getPoderDePelea(): int
+ getVivo(): boolean
+ restarVida(int): void
+ incrementarVida(int): void
+ restarKi(int): void
+ getPuntosDeVida(): int
+ aumentarAtaqueEnPorEsferaDelDragon(): void
+ aumentarVelocidadPorNubeVoladora(): void
+ aumentarPuntosDeVidaPorSemillaDelErmitanio(): void
+ volverAtaqueANormalidad(Fase): void
+ volverVelocidadANormalidad(Fase): void
+ setearEstado(String,Fase): void
+ setearMuerto(): void
+ actualizar(String,Fase): void
StatsJuego
+ obtenerEstadosIniciales(): HashMap<String, HashMap<String,Integer>>
+ obtenerEstados1(): HashMap<String,HashMap<String,Integer>>
+ obtenerEstados2(): HashMap<String,HashMap<String,Integer>>
Fase
+ obtenerStats(): HashMap<String,Integer>
+ transformar(int): Fase
+ transformarCell(int,int): Fase
+ transformarGohan(int,Equipo): Fase
+ transformarPicolo(int,Equipo): Fase
EnemigoDeLaTierra
+ esGuerreroZ(): boolean
+ esEnemigoDeLaTierra(): boolean
GuerreroZ
- inmovilizado: boolean
- turnosRestantesInmovilizado: int
+ esGuerreroZ(): boolean
+ esEnemigoDeLaTierra(): boolean
+ inmovilizar(): void
+ actualizarInmovilizado(): void
+ atacarConBasico(Personaje): void
+ MoverPersonajeHaciaArriba(): void
+ MoverPersonajeHaciaAbajo(): void
+ MoverPersonajeHaciaDerecha(): void
+ MoverPersonajeHaciaIzquierda(): void
+ MoverPersonajeHaciaArribaDerecha(): void
+ MoverPersonajeHaciaArribaIzquierda(): void
+ MoverPersonajeHaciaAbajoDerecha(): void
+ MoverPersonajeHaciaAbajoIzquierda(): void
Use
Extends
1
0..
4

## Diagramas de secuencia

---



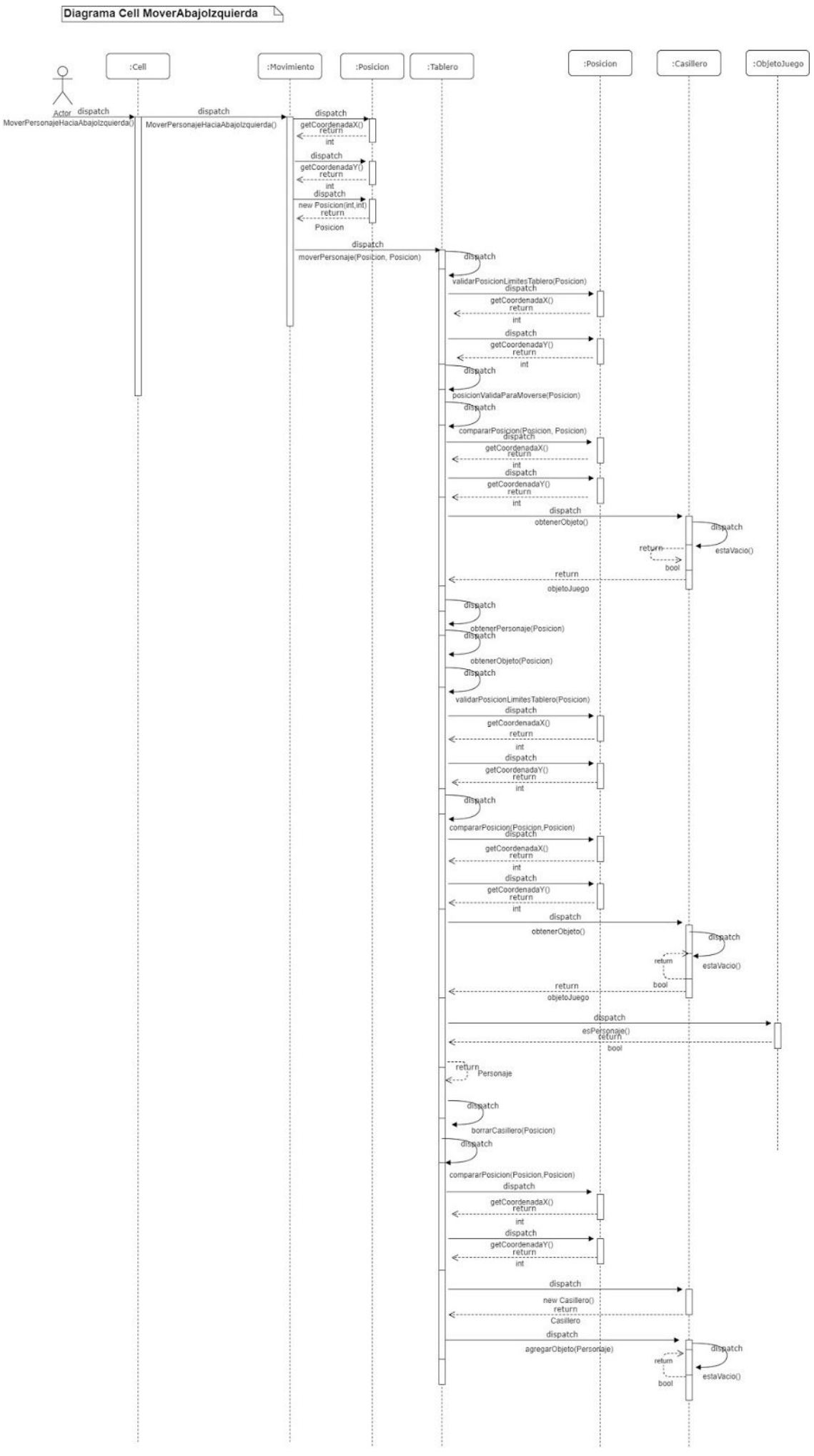
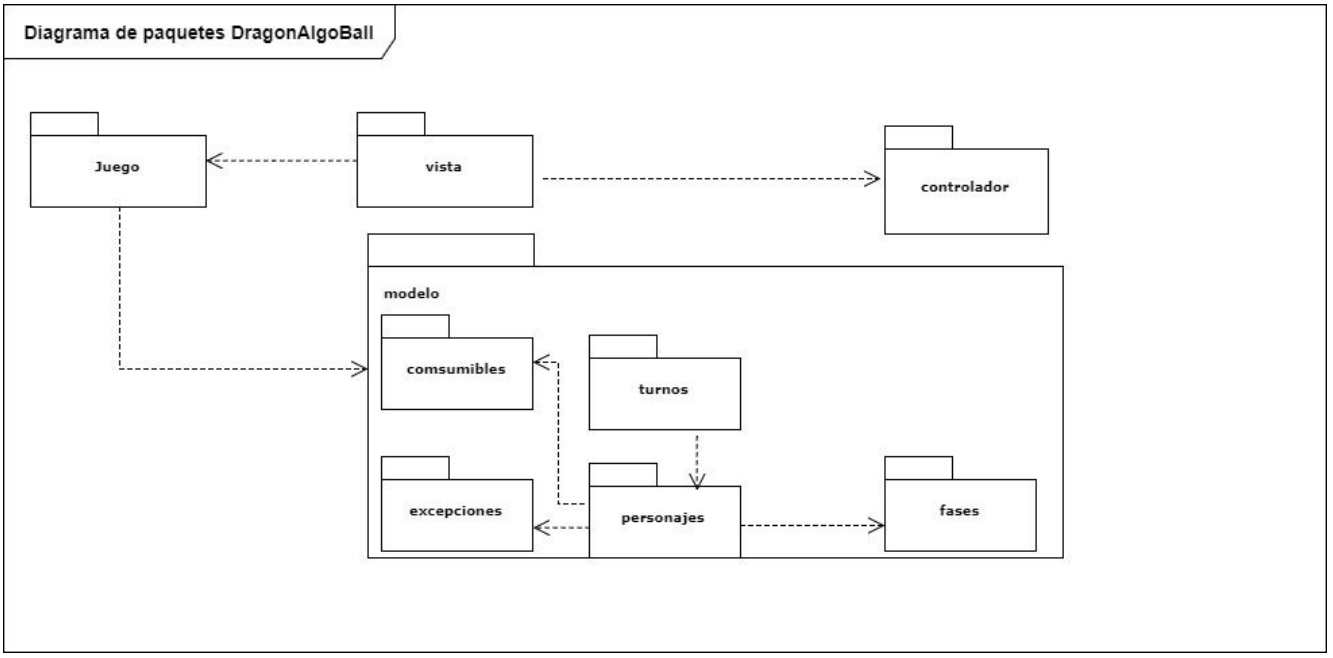
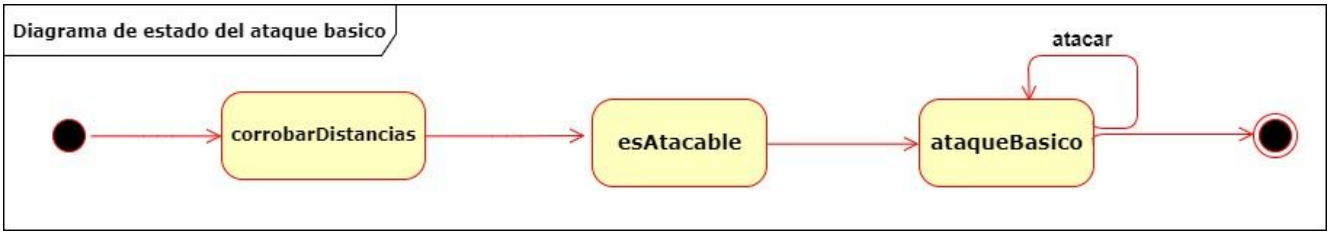


Diagrama Cell MoverAbajoIzquierda
:Cell
:Movimiento
:Posicion
:Tablero
:Posicion
:Casillero
:ObjetoJuego
Actor
dispatch
MoverPersonajeHaciaAbajoIzquierda()
dispatch
MoverPersonajeHaciaAbajoIzquierda()
dispatch
getCoordenadaX()
return
int
dispatch
getCoordenadaY()
return
int
dispatch
new Posicion(int,int)
return
Posicion
dispatch
moverPersonaje(Posicion, Posicion)
dispatch
validarPosicionLimitesTablero(Posicion)
dispatch
getCoordenadaX()
return
int
dispatch
getCoordenadaY()
return
int
dispatch
posicionValidaParaMoverse(Posicion)
dispatch
compararPosicion(Posicion, Posicion)
dispatch
getCoordenadaX()
return
int
dispatch
getCoordenadaY()
return
int
dispatch
obtenerObjeto()
dispatch
estaVacio()
return
bool
return
objetoJuego
dispatch
obtenerPersonaje(Posicion)
dispatch
obtenerObjeto(Posicion)
dispatch
validarPosicionLimitesTablero(Posicion)
dispatch
getCoordenadaX()
return
int
dispatch
getCoordenadaY()
return
int
dispatch
compararPosicion(Posicion,Posicion)
dispatch
getCoordenadaX()
return
int
dispatch
getCoordenadaY()
return
int
dispatch
obtenerObjeto()
dispatch
estaVacio()
return
bool
return
objetoJuego
dispatch
esPersonaje()
return
bool
return
Personaje
dispatch
borrarCasillero(Posicion)
dispatch
compararPosicion(Posicion,Posicion)
dispatch
getCoordenadaX()
return
int
dispatch
getCoordenadaY()
return
int
dispatch
new Casillero()
return
Casillero
dispatch
agregarObjeto(Personaje)
dispatch
estaVacio()
return
bool

## Diagrama de paquetes




## Diagramas de estado




## Detalles de implementación

---

*Como base de datos para almacenar la información relativa a la cantidad de vida, velocidad, poder, etc de cada personaje creamos una clase la cual contiene como constantes aquellos valores.*

*El tablero se implementó mediante un diccionario que cómo claves posee las posiciones en el tablero y como objeto un objeto de juego.*

## Excepciones

---

*Las excepciones que creamos fueron movimiento invalido para alertar al usuario que no se puede mover a esa posición, ya sea que esta ocupada, que hay un obstáculo en el camino hacia esa posición.*

*También creamos la excepción posición invalida, para alertar al usuario que esa posición no existe en el tablero.*

*La excepción tamaño de tablero invalido se utiliza para alertar al usuario que ese tamaño de tablero no es permitido, excede al tamaño maximo estipulado.*

*La otra excepción que implementamos es la de ki insuficiente, esta sirve para alertar al usuario que no le alcanza el ki para realizar la evolucion o adquirir el poder especial.*

*También creamos la excepcion AtaqueInvalido para informarle al usuario que no puede realizar ese ataque, por ejemplo, porque está muy lejos de su oponente.*

*Las excepciones CasilleroOcupado y CasilleroVacio se utilizan para informar el estado del casillero, de no ser este el ideal para realizar la operacion deseada.*

*La excepcio EvolucionInvalida / Transformacion invalida se utiliza para informarle al usuario que no puede realizar la evolucion deseada. Está excepcion es porque falta un requisito para poder realizar la evolucion. Por ejemplo, para el caso de MaijinBoo por falta de absorciones.*

*La excepcion JuegoTerminado es para informar que el juego se termino.*

*La excepcion PasosInsuficientes es porque el usuario desea avanzar más lejos de lo que le alcanza segun su velocidad.*

*La excepcion PersonajeInmovilizado es para informar que el personaje fue inmovilizado y, por ejemplo, no puede moverse.*