



**FACULTAD
DE INGENIERIA**

Universidad de Buenos Aires ://es.overleaf.com/project/5bedb4cf21f7045434f1cafc

Trabajo Práctico 2 - Java

Algo of Empires

[7507/9502] Algoritmos y Programación III

Curso 1

Segundo cuatrimestre de 2018

Alumnos:	BIAUS, Ivo GARCIA, Ailen LOPEZ HIDALGO, Tomas TROTТА, Laura
Grupo:	2

Índice

1. Supuestos	2
2. Diagramas de clase	2
3. Diagramas de secuencia	6
4. Diagrama de paquetes	9
5. Diagramas de estado	9
6. Detalles de implementacion	10
6.1. Patrón MVC.	10
6.2. Patrón State.	11
7. Excepciones	11

1. Supuestos

- 1) Cada Jugador, dentro de un mismo turno, puede realizar múltiples acciones, dado que su turno se compone de la acción de cada Pieza; y cada Pieza solo puede realizar una acción por turno.
- 2) Los Edificios pueden realizar una sola acción por turno, la cual es crear una Unidad.
- 3) Un Aldeano, en un mismo turno, sólo puede mover, reparar, o construir, pero solo si realiza la acción de mover (o ninguna) puede generar oro.
- 4) Si un Aldeano está reparando/construyendo, no puede realizar ninguna otra acción o cancelar la actual tarea hasta que ésta se termine.
- 5) Los edificios sólo pueden tener un aldeano asignado ya sea para construcción o reparación. No más de un Aldeano puede construir/reparar.
- 6) El tamaño mínimo del tablero es de 16x16,

2. Diagramas de clase

Diagrama de clase integral.

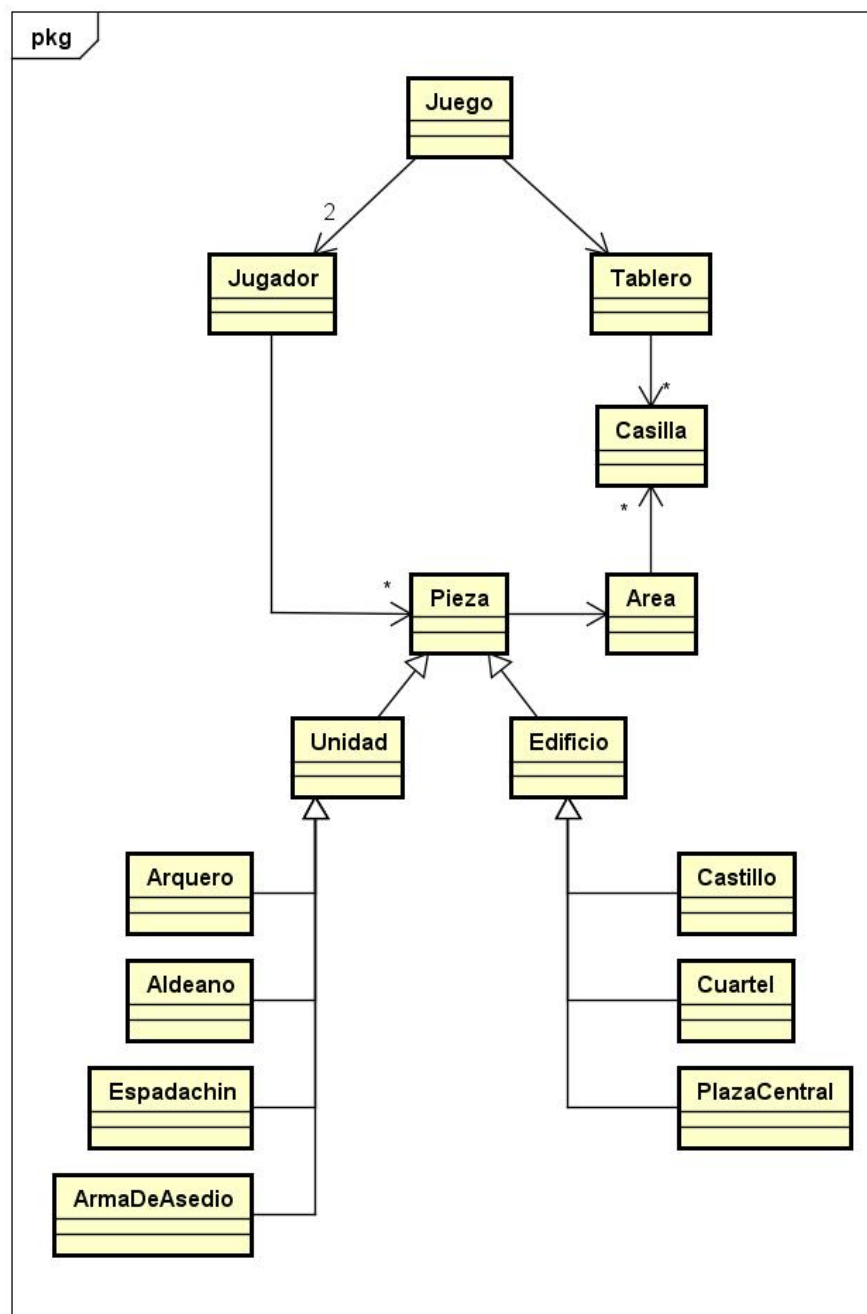


Figura 1: Diagrama del Algo Empires2.

Diagrama de clases 1: Relacion entre Juego, Jugador y Tablero.

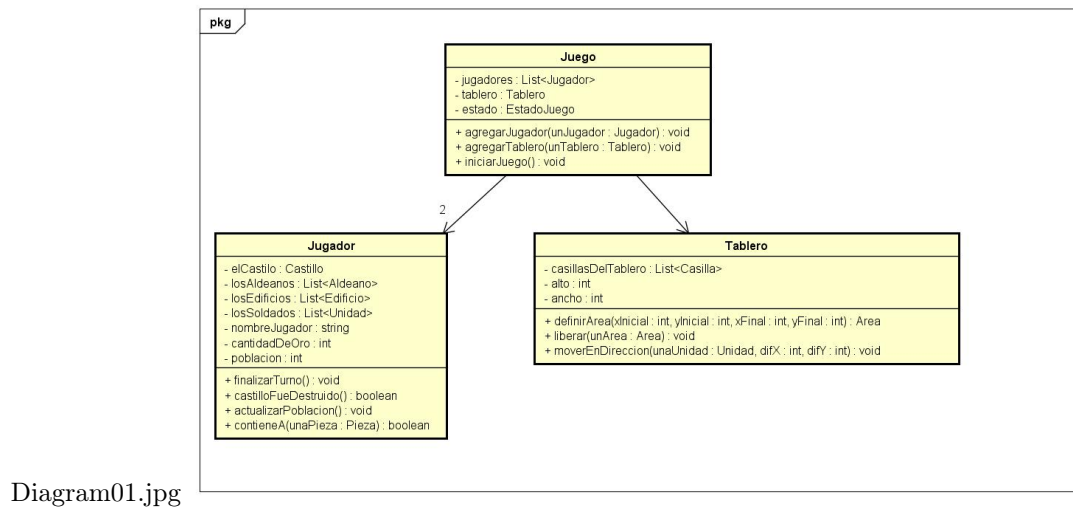


Figura 2: Diagrama de Clases01.

Diagrama de clases 2: Relacion entre Pieza, Area y Casilla. Unidad y Edificio heredan de Pieza.

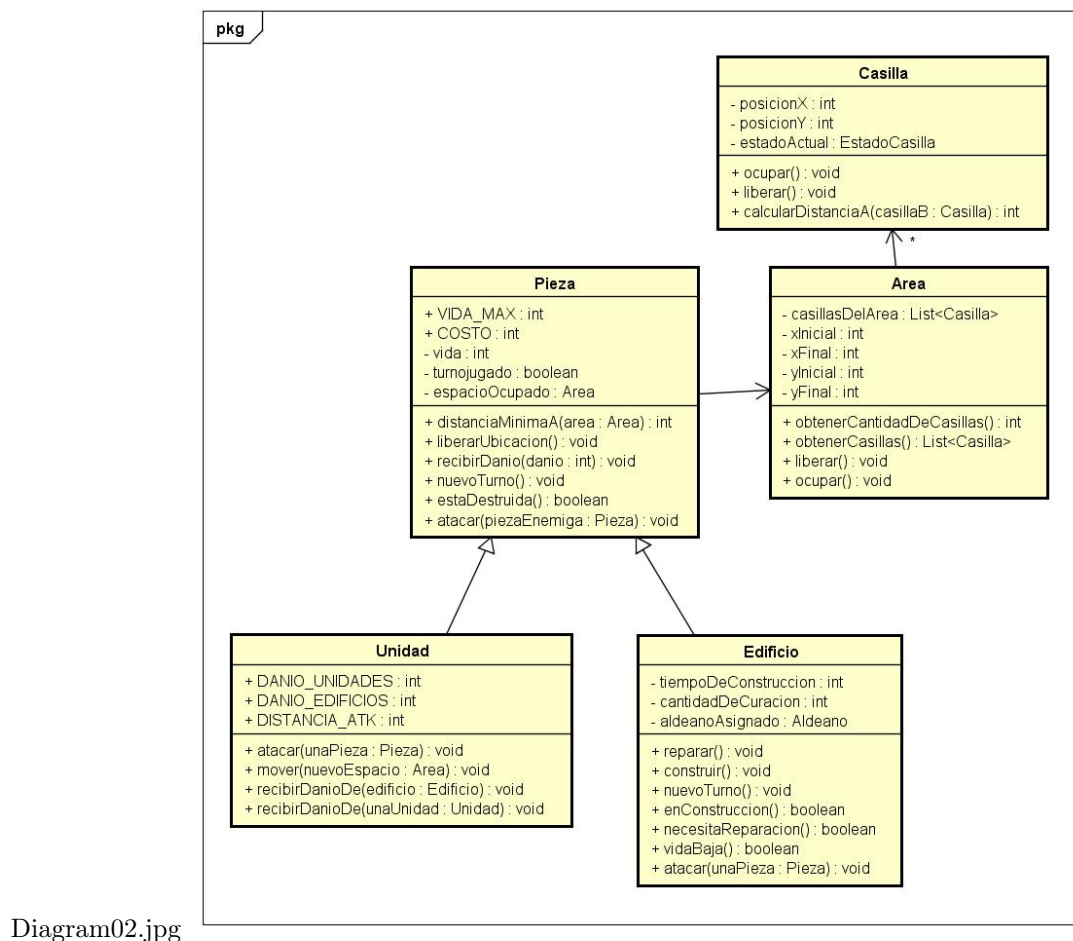


Figura 3: Diagrama de Clases02.

Diagrama de clases 3: Unidad y sus hijos: Aldeano, Arquero, Espadachin y Arma de Asedio.

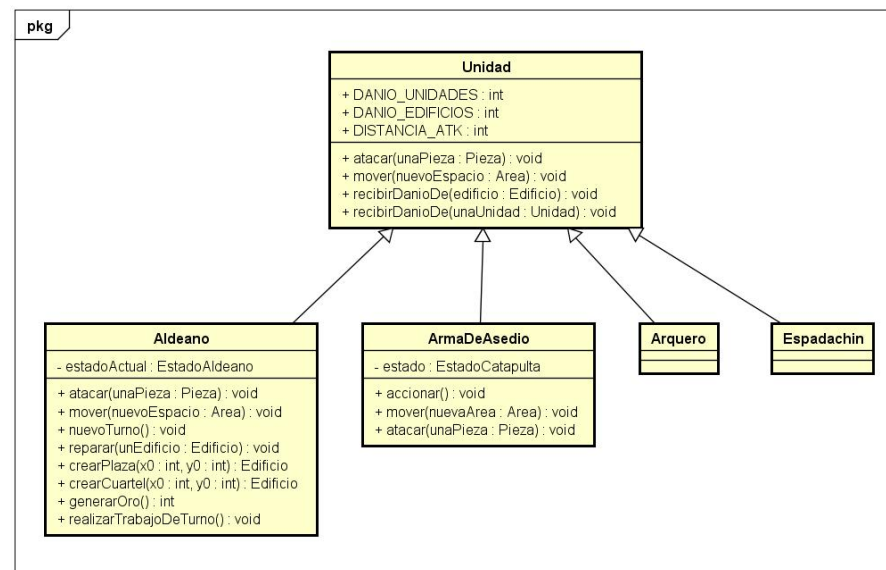


Diagram03.jpg

Figura 4: Diagrama de Clases03.

Diagrama de clases 4: Edificio y sus hijos: Castillo, Cuartel y Plaza.

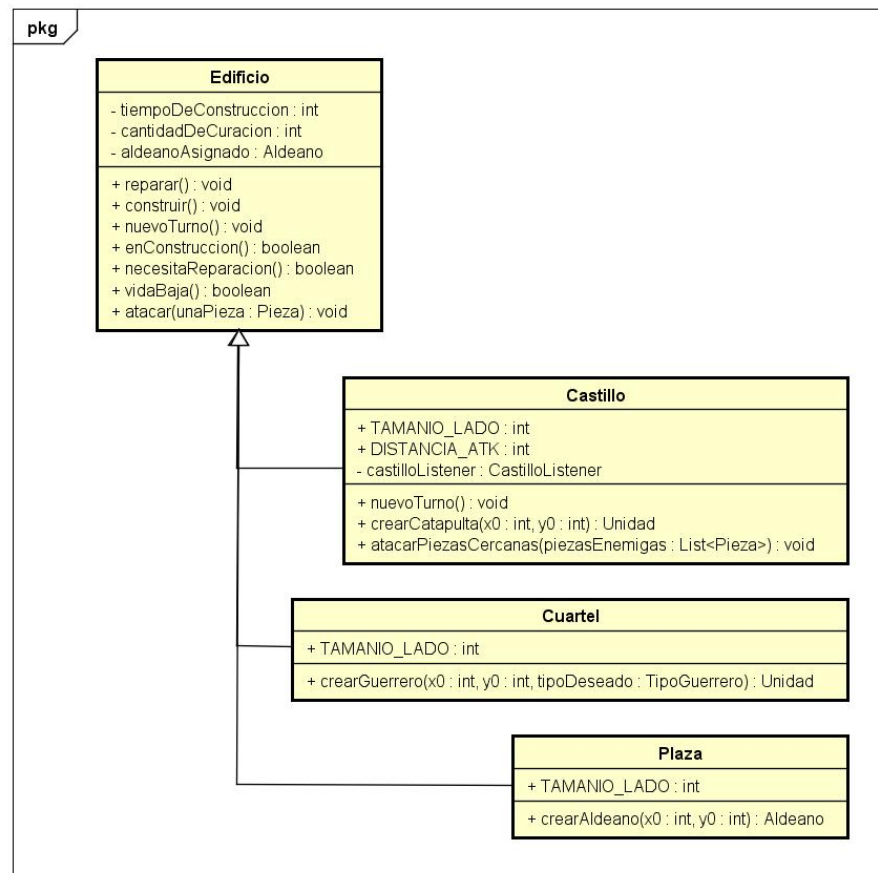


Diagram04.jpg

Figura 5: Diagrama de Clases04.

3. Diagramas de secuencia

Diagrama de secuencia 1: Crear y Colocar Edificio.

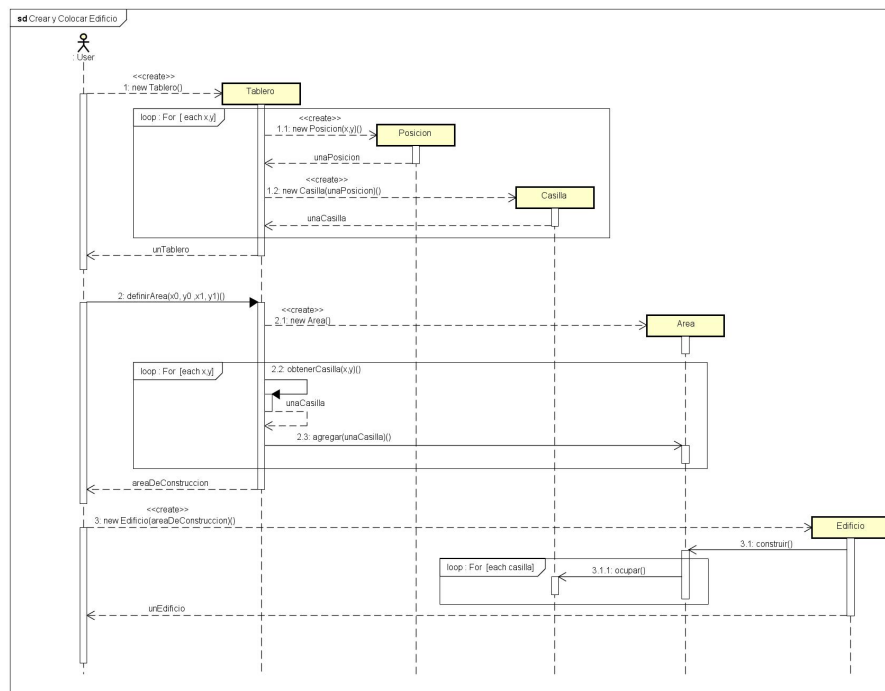


Figura 6: Diagrama Crear y Colocar Edificio

Diagrama de secuencia 2: Crear y Colocar Unidad.

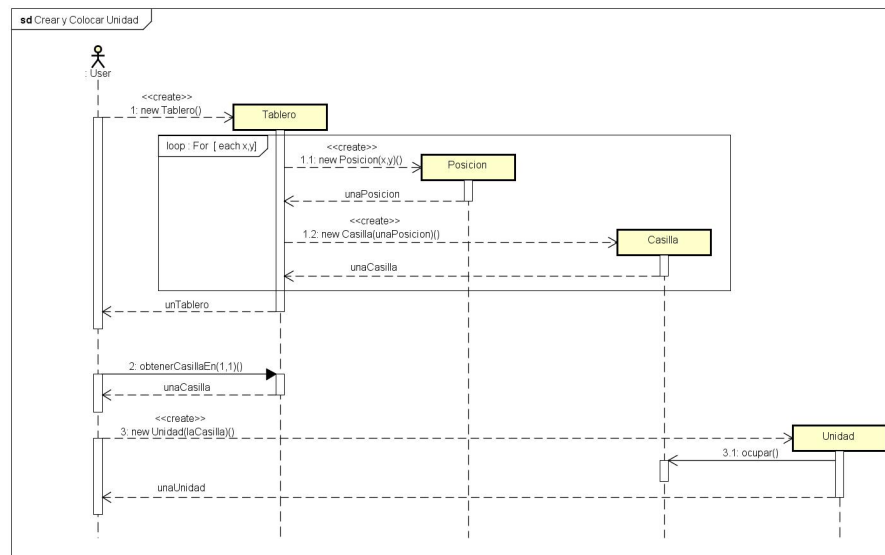


Figura 7: Diagrama Crear y Colocar Unidad

Diagrama de secuencia 3: Aldeano crea Edificio.

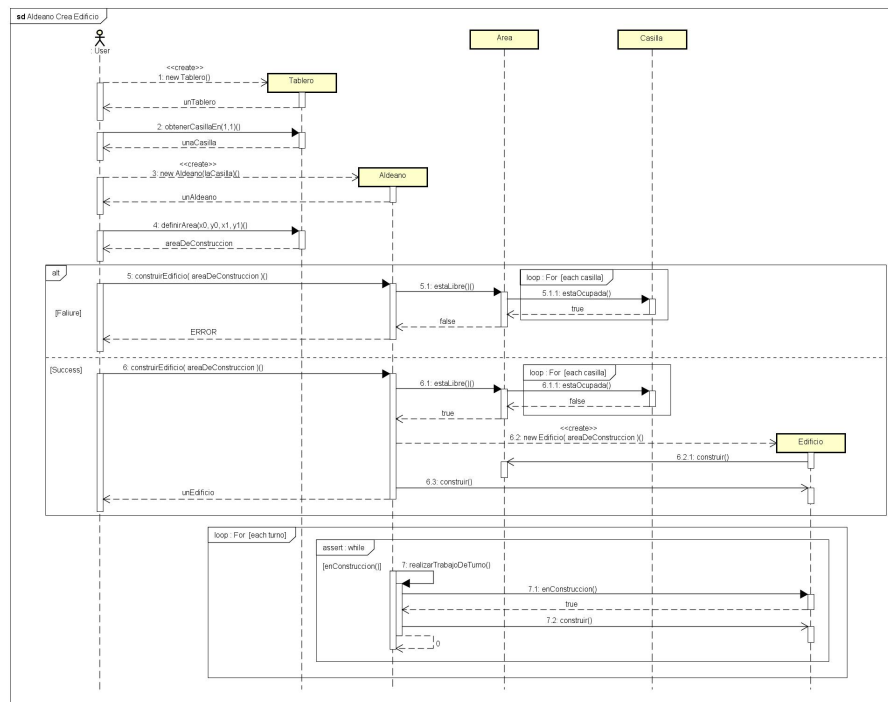


Figura 8: Diagrama Aldeano Crea Un Edificio

Diagrama de secuencia 4: un Espadachin ataca a un Aldeano.

ataca a Aldeano.png

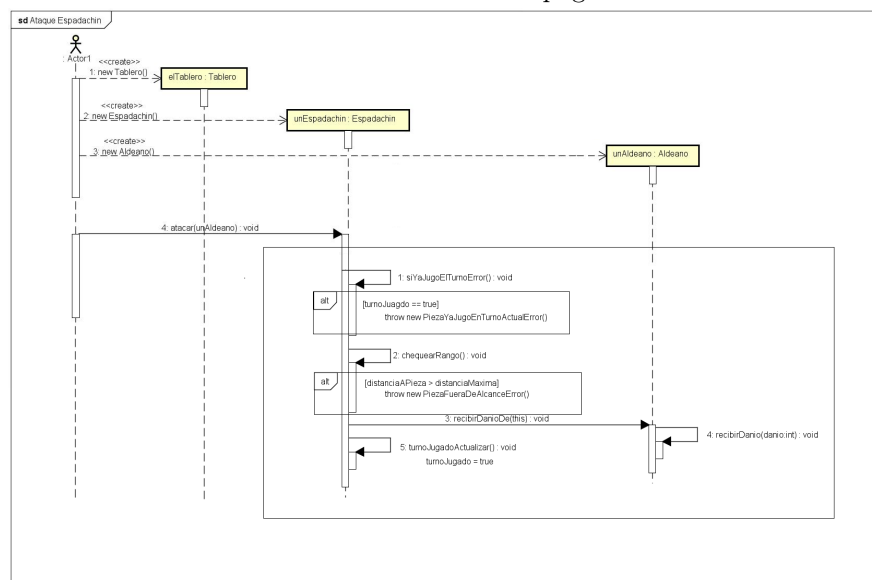


Figura 9: Diagrama Espadachin ataca a Aldeano

4. Diagrama de paquetes

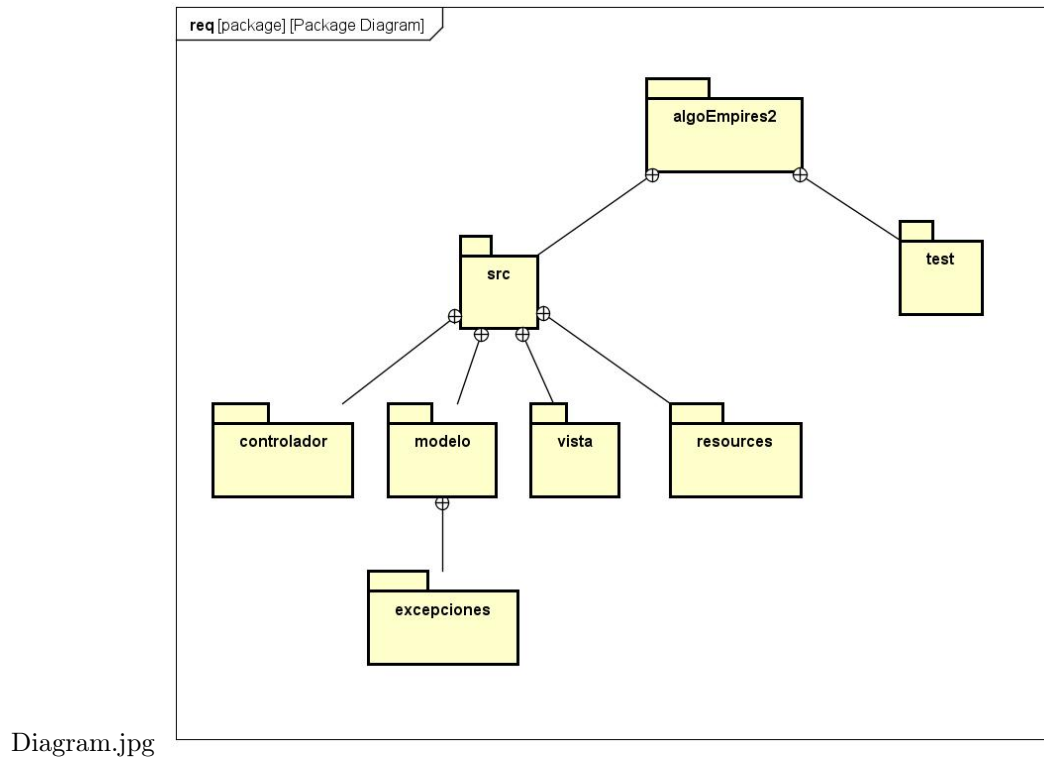


Figura 10: Diagrama de paquetes

5. Diagramas de estado

Diagrama de Estado: Juego

de Estado Juego.jpg

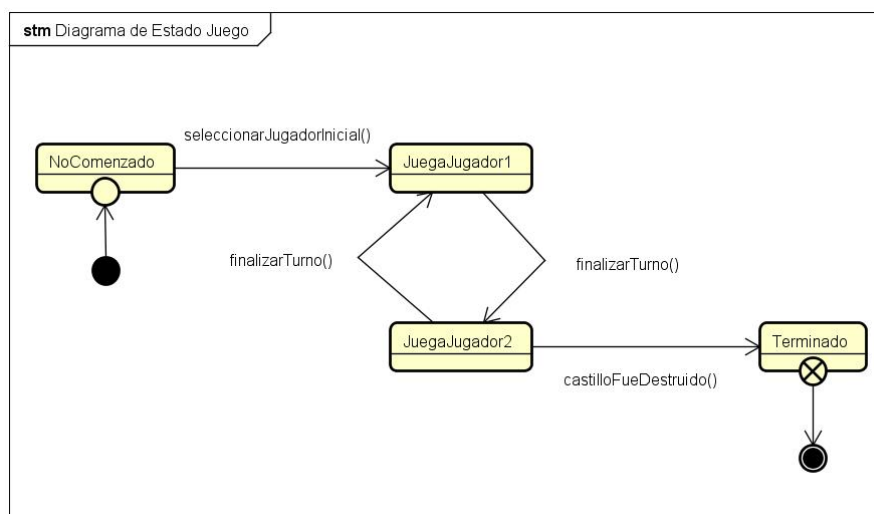


Figura 11: Diagrama de Estado Juego

Diagrama de Estado: Aldeano

de Estado Aldeano.jpg

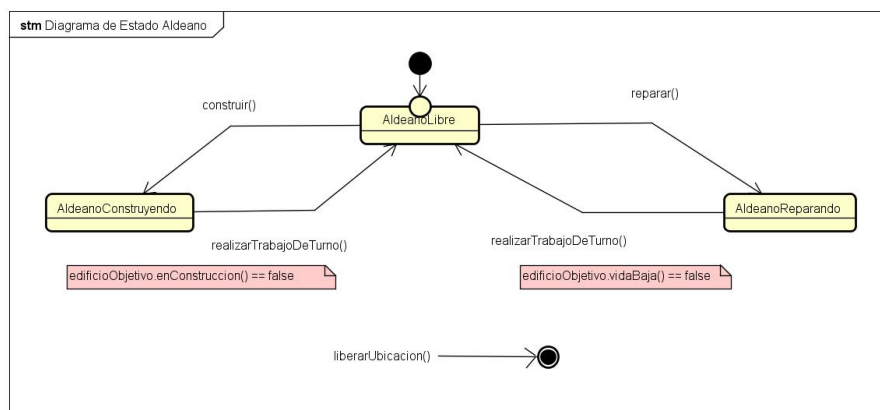


Figura 12: Diagrama de Estado Aldeano

6. Detalles de implementacion

Para la implementación de este Trabajo Práctico hicimos uso de dos patrones: MVC y State.

6.1. Patrón MVC.

Modelo-vista-controlador (MVC) es un patrón que separa la lógica de una aplicación de su representación e interfaz, por medio de un encargado de gestionar los eventos y las comunicaciones. Para ello, se propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

De manera genérica, los componentes de MVC se podrían definir de la siguiente manera:

El Modelo: Es la parte lógica del sistema. Contiene la información con la cual el sistema opera. Envía a la "Vista.^a aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al "Modelo.^a a través del Controlador".

El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al "Modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a su "Vista.^a asociada si se solicita un cambio en la forma en que se presenta el "Modelo", por tanto se podría decir que el Controlador hace de intermediario entre la "Vista y el "Modelo".

La Vista: Presenta el "Modelo.^{en} un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho "Modelo" la información que debe representar como salida.

Usamos este patrón para englobar el juego entero, y así separar las tareas de la interfaz con las de la lógica, usando como intermediario a un controlador, logrando así que cada clase tenga responsabilidades únicas y acordes al contexto.

6.2. Patrón State.

El patrón de diseño State se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo. Propone una solución al manejo de los cambios en un objeto, creando básicamente, un objeto por cada estado posible de dicho objeto que lo llama. El objeto de la clase a la que le pertenecen dichos estados resuelve los distintos comportamientos según su estado, con instancias de dichas clases de estado. Así, siempre tiene presente en un objeto el estado actual y se comunica con este para resolver sus responsabilidades.

Usamos este patrón en las clases Juego, Casilla, Aldeano y Arma de Asedio.

Los estados del Juego son: NoComenzado, JuegaJugador1, JuegaJugador2 y Terminado.

Los estados de la Casilla son: CasillaLibre y CasillaOcupada.

Los estados del Aldeano son: AldeanoLibre, AldeanoConstruyendo y AldeanoReparando.

Los estados del Arma de Asedio son: CatapultaArmada y CatapultaDesarmada.

También fue necesario usar este patrón en la clase ArmaDeAsedioVista, con sus dos estados CatapultaArmadaVista y CatapultaDesarmadaVista.

7. Excepciones

Exception `JuegoNoTerminadoError` Esta excepción se lanza cuando se le pide al Juego que me muestre el ganador, si aún el juego no ha finalizado.

Exception `NoExistenJugadoresActualesError` Esta excepción fue creada para que, en el caso de que se le pida al Juego el jugador actual, y todavía la partida no ha comenzado.

Exception `NoHayJuegoEnProcesoError` Esta excepción se lanza en el caso de que se quiera terminar un juego que aún no ha comenzado.

Exception `CasillaOcupadaError` Esta excepción fue creada para evitar que un jugador ocupe una casilla que ya está siendo ocupada por una pieza.

Exception `NullPointerExceptionCasillaError` Esta excepción se lanza si, en algún caso, se pasa por parámetro una Casilla que no apunta a nada.

Exception `PiezaOcupadaNoPuedeAccionarError` Esta excepción se lanza en el caso de que, a una Pieza que está ocupada, le pido que realice una acción.

Exception `PiezaYaJugoEnTurnoActualError` Esta excepción se lanza en el caso de que una Pieza quiera jugar en un turno en donde ya había jugado.

Exception `CasillaInvalidaError` Esta excepción fue creada para evitar que un jugador quiera acceder a una Casilla que está fuera del mapa, es decir, que posea coordenadas inválidas.

Exception `PoblacionLimiteSuperadaError` Esta excepción se lanza en el caso de que a un jugador que ya posee la cantidad de población límite, se le quiere agregar más población.

Exception `AldeanoConstruyendoNoPuedeRepararError` Esta excepcion se lanza si un Aldeano está ocupado construyendo un Edificio y se le manda la instrucción de reparar otro Edificio.

Exception `AldeanoOcupadoNoPuedeMoverseError` Esta excepcion se usa para indicar que un Aldeano no puede moverse si está ocupado con otra tarea, ya sea reparando o construyendo.

Exception `AldeanoOcupadoConOtroEdificioError` Si a un Aldeano se lo manda a construir/reparar un edificio, y ya esta ocupado construyendo/reparando otro, se lanza esta excepción.

Exception `AldeanoReparandoNoPuedeConstruirError` Si un Aldeano está reparando no puede construir.

Exception `CasillaYaEstaLibreError` Esta excepción se lanza si se quiere liberar una casilla que ya está libre.

Exception `CatapultaArmadaNoPuedeMoverseError` Esta excepción se lanza si la catapulta está armada, por lo tanto no puede moverse, se debe esperar a que la catapulta este desarmada.

Exception `CatapultaDesarmadaNoPuedeAtacarError` Si la catapulta no esta montada, no puede atacar, por ende se lanza esta excepción.

Exception `EdificioTieneOtroAldeanoAsignadoError` Los edificios solo pueden tener un aldeano asignado ya sea para construcción o reparación. Si se intenta asignar otro aldeano se lanzará esta excepción.

Exception `EdificioYaEstaSiendoReparadoError` Si se le manda la instrucción de reparar al edificio y ya está siendo reparado, se tira esta excepción.

Exception `NoSePuedeConstruirTanLejosError` Solo se puede construir a una casilla de distancia. Si esta mas lejos se lanzara este mensaje para notificar que fue un error.

Exception `NoSePuedeCrearUnidadesDuranteConstruccionError` Si el edificio esta en construccion, no puede crear unidades de ningun tipo. Se debe esperar a que se termine de construir por completo.

Exception `OroInsuficienteError` Si el Jugador no tiene suficiente Oro para crear una Unidad o Edificio, se lanza esta excepción.

Exception `PiezaFueraDeAlcanceError` Se usa cuando se chequea el rango de ataque de una pieza a otra. Si intenta atacar y no esta dentro del alcance, se lanza este error.

Exception `PiezaNoEstaEnEquipoAliadoError` Si un Jugador selecciona una pieza del Jugador oponente, para intentar moverla o realizar cualquier acción, se lanzará esta excepción.

Exception `PiezaNoEstaEnEquipoEnemigoError` Si un Jugador pretende atacar a una pieza que no es del equipo contrincante, se lanza este error.

Exception `TableroEsDeTamanoInvalidoError` Si el Tablero mide menor a 16x16, se lanzara esta excepción.

Exception `TipoDeGuerreroInvalidoError` Si se intenta crear un Guerrero indicando otro tipo de pieza que no corresponde, se lanza esta excepción.