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 TROTTA, Laura
Grupo:	2

Índice

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.

2. Diagramas de clase

Diagrama de clase integral.

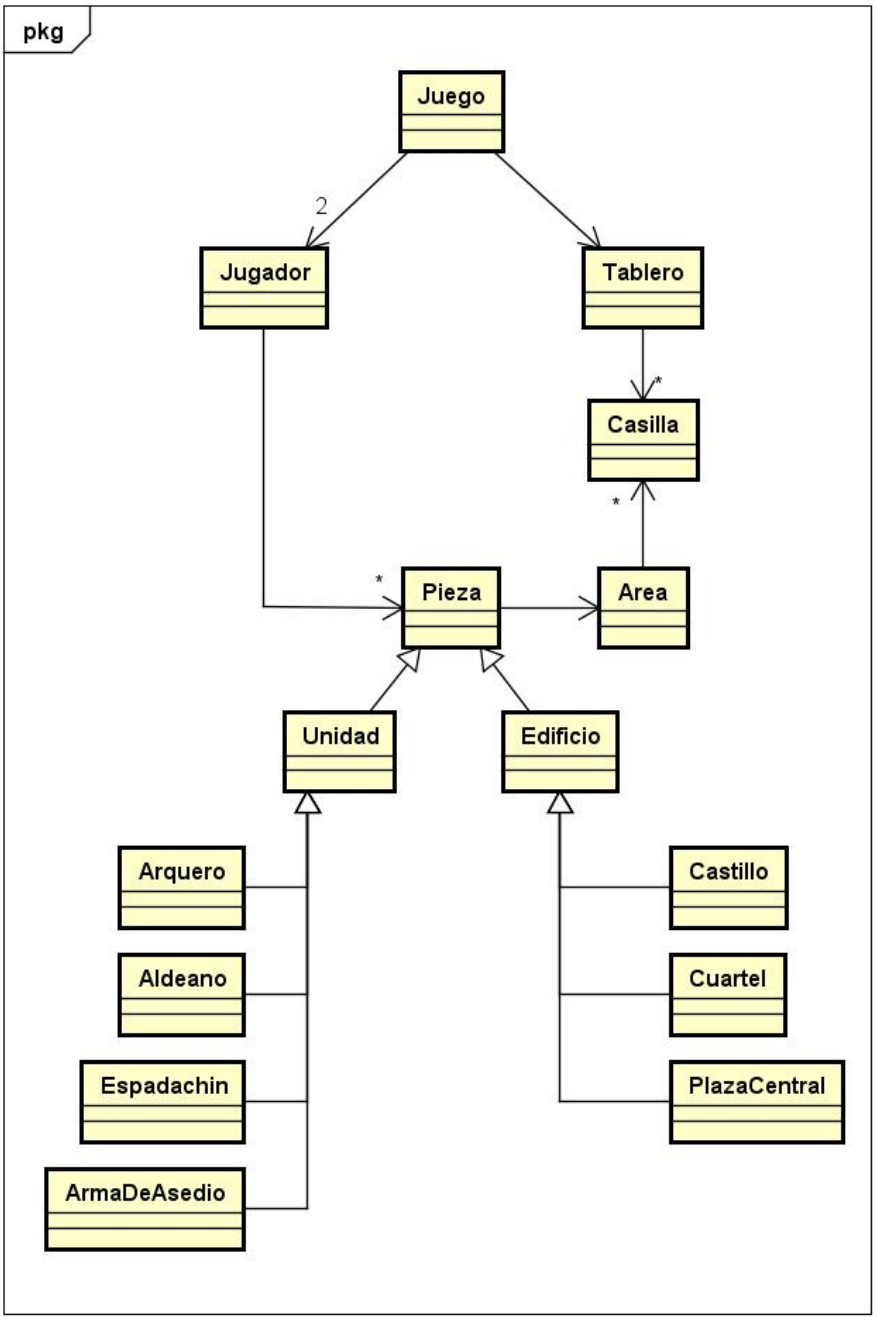

Figura 1: Diagrama del Algo Empires2.

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

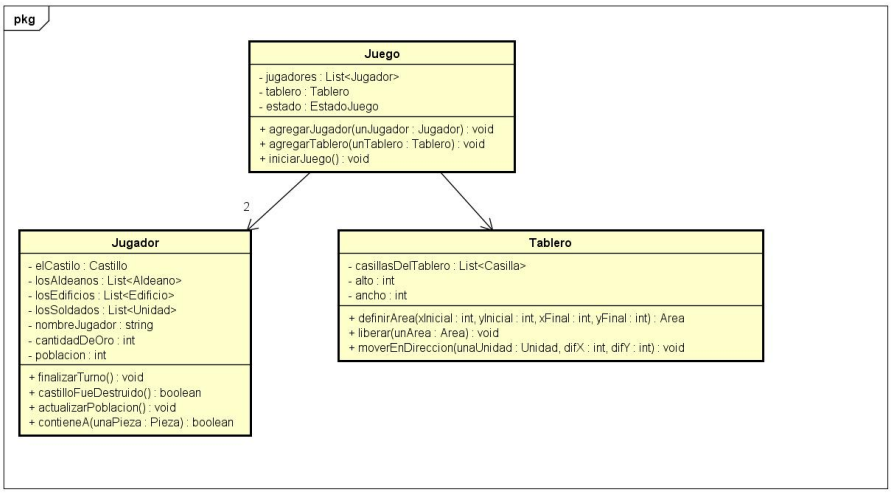

Diagram01.jpg

Figura 2: Diagrama de Clases01.

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

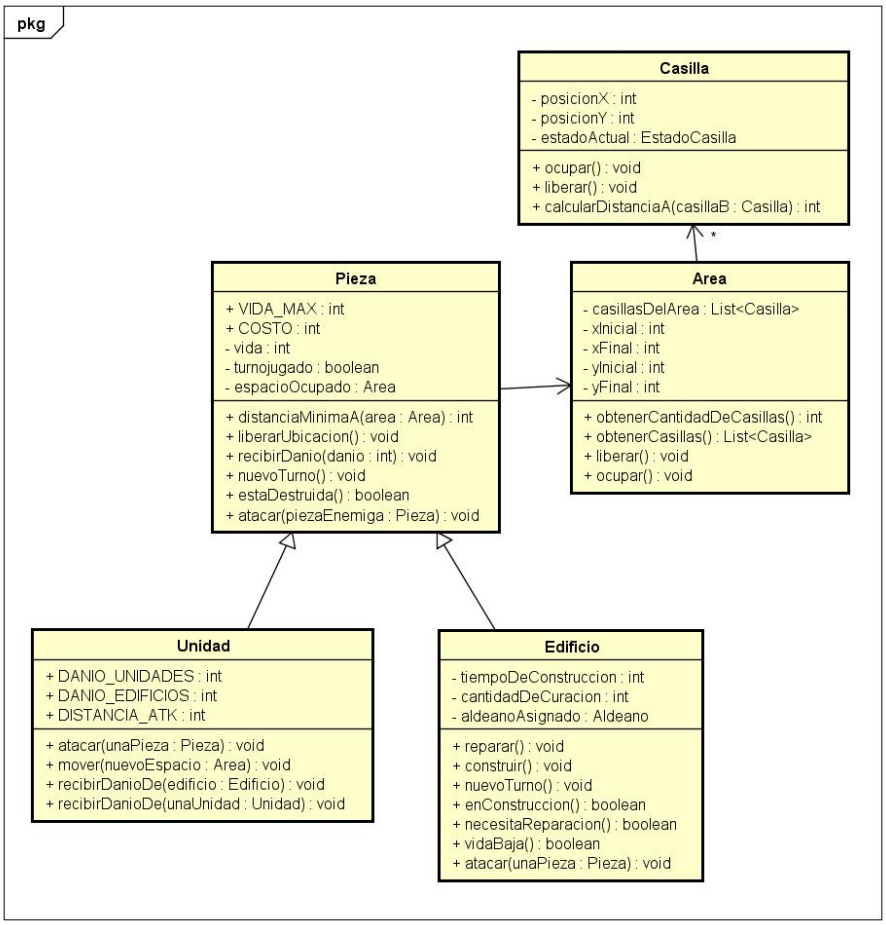

Diagram02.jpg

Figura 3: Diagrama de Clases02.

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

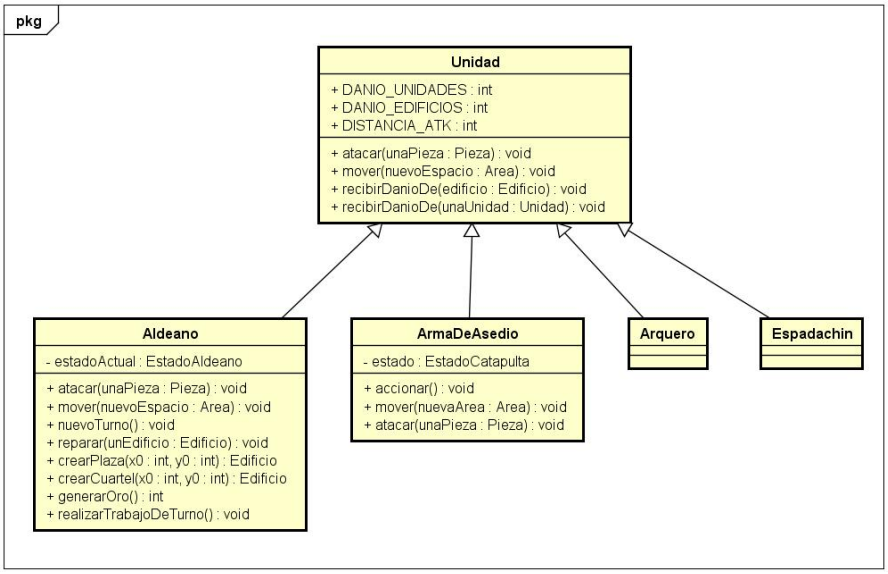

Diagram03.jpg

Figura 4: Diagrama de Clases03.

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

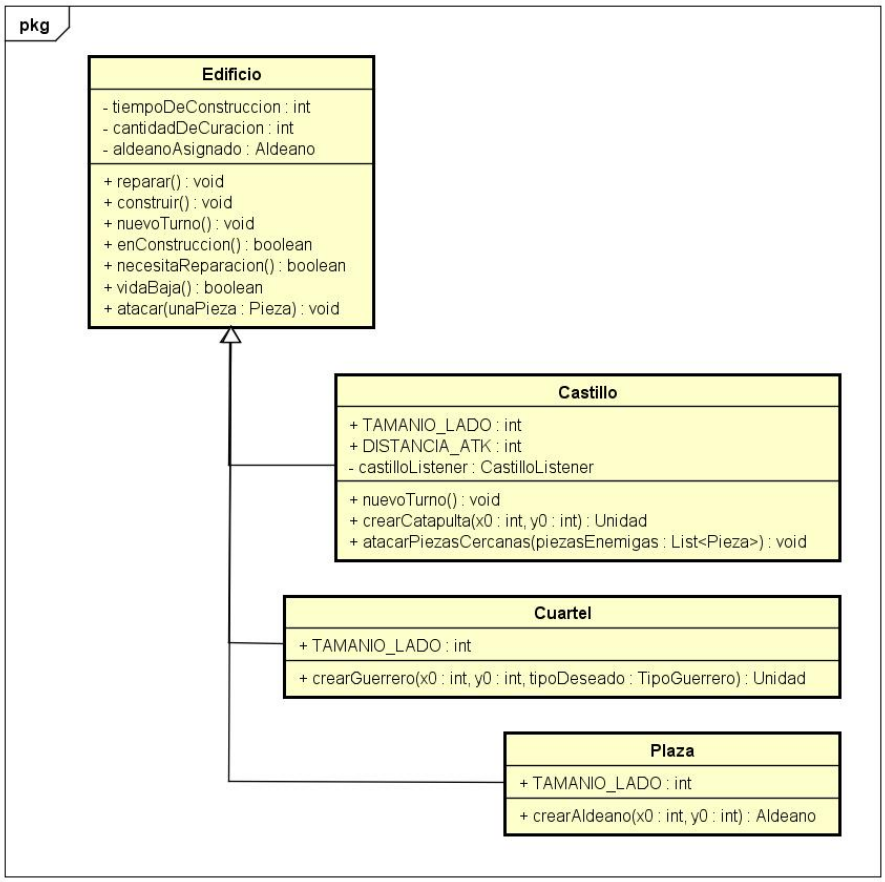

Diagram04.jpg

Figura 5: Diagrama de Clases04.

3. Diagramas de secuencia

Diagrama de secuencia 1: Crear y Colocar Edificio.

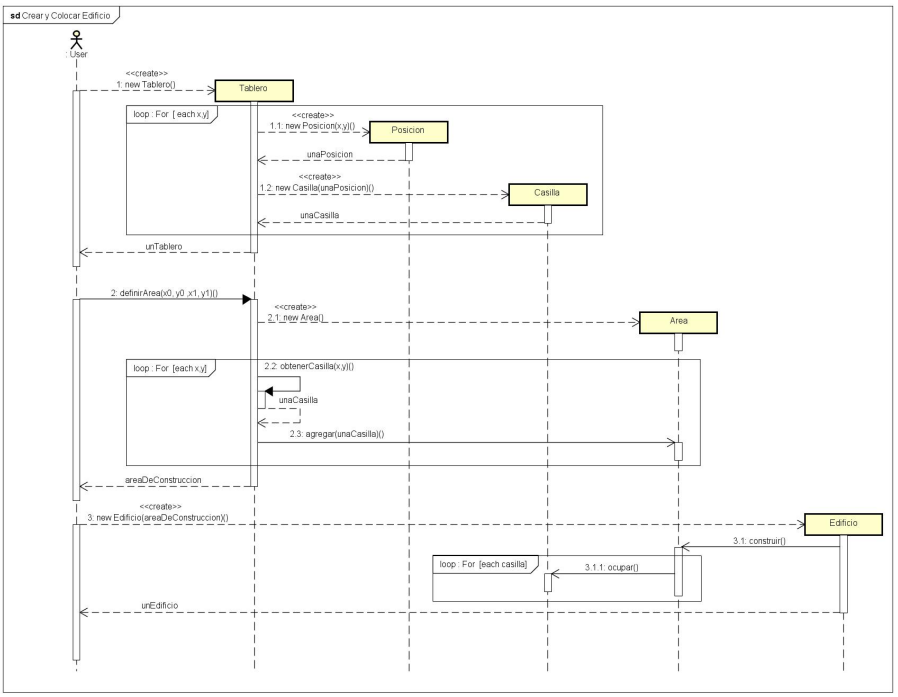

Figura 6: Diagrama Crear y Colocar Edificio

Diagrama de secuencia 2: Crear y Colocar Unidad.

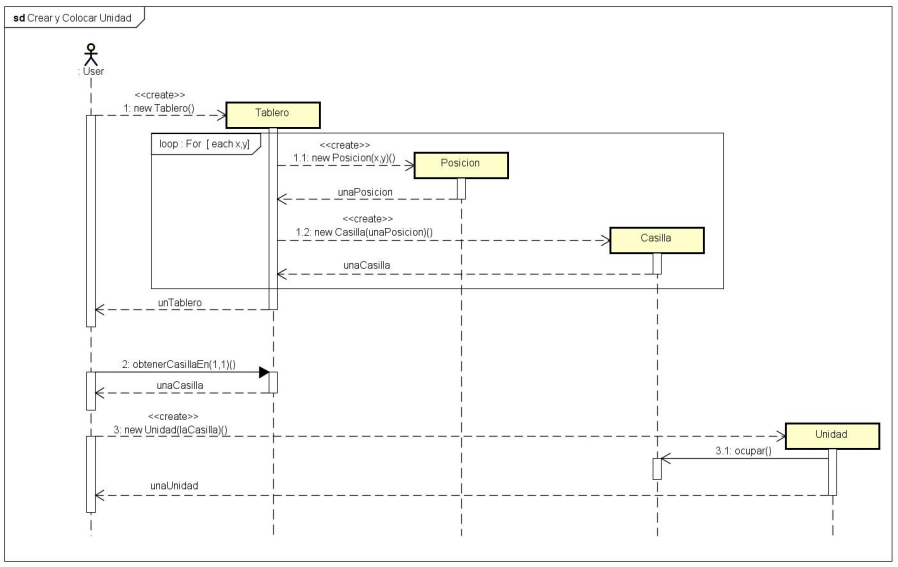

Figura 7: Diagrama Crear y Colocar Unidad

Diagrama de secuencia 3: Aldeano crea Edificio.

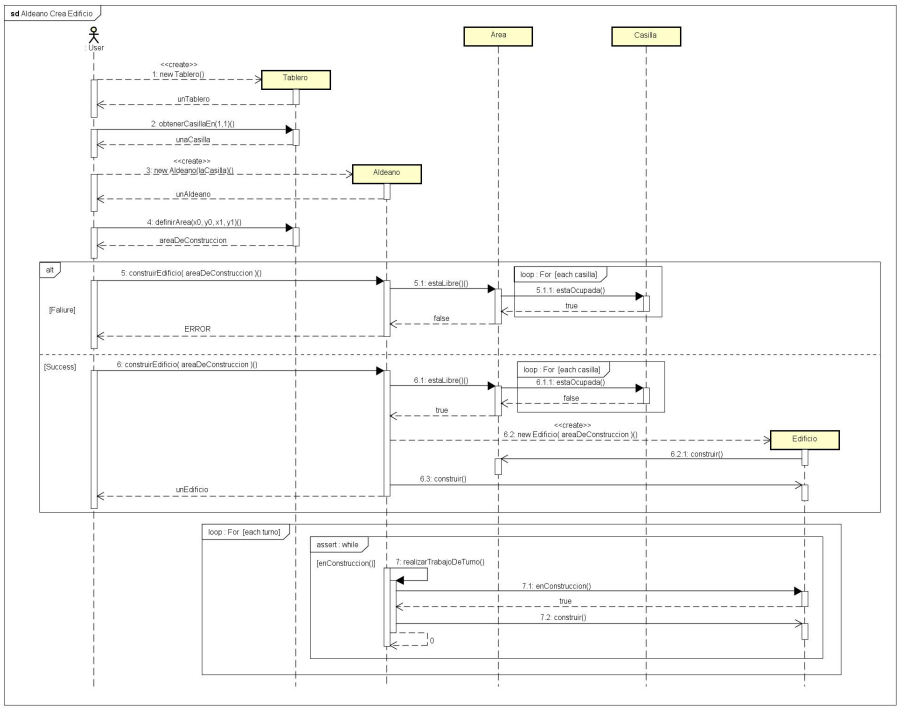

Figura 8: Diagrama Aldeano Crea Un Edificio

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

ataca a Aldeanoc.png

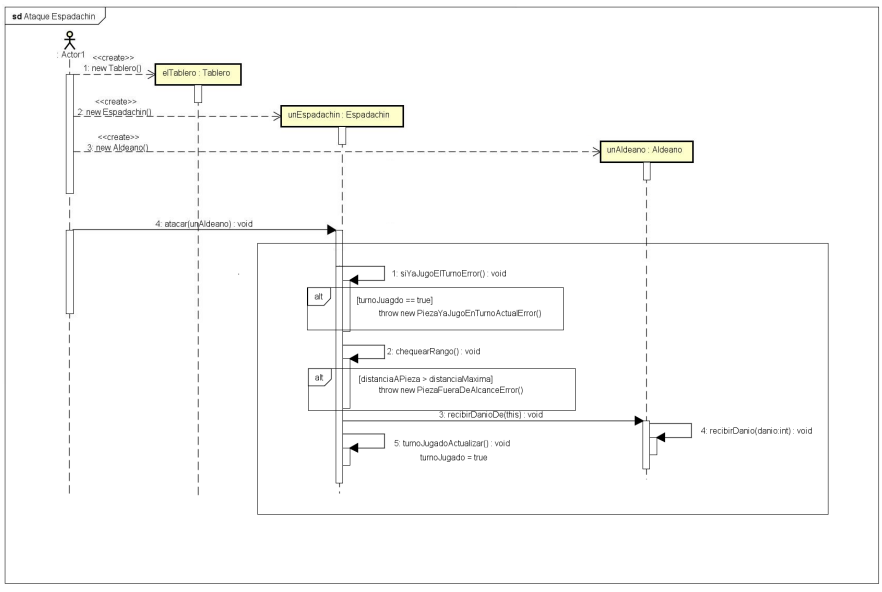

Figura 9: Diagrama Espadachin ataca a Aldeano

4. Diagrama de paquetes

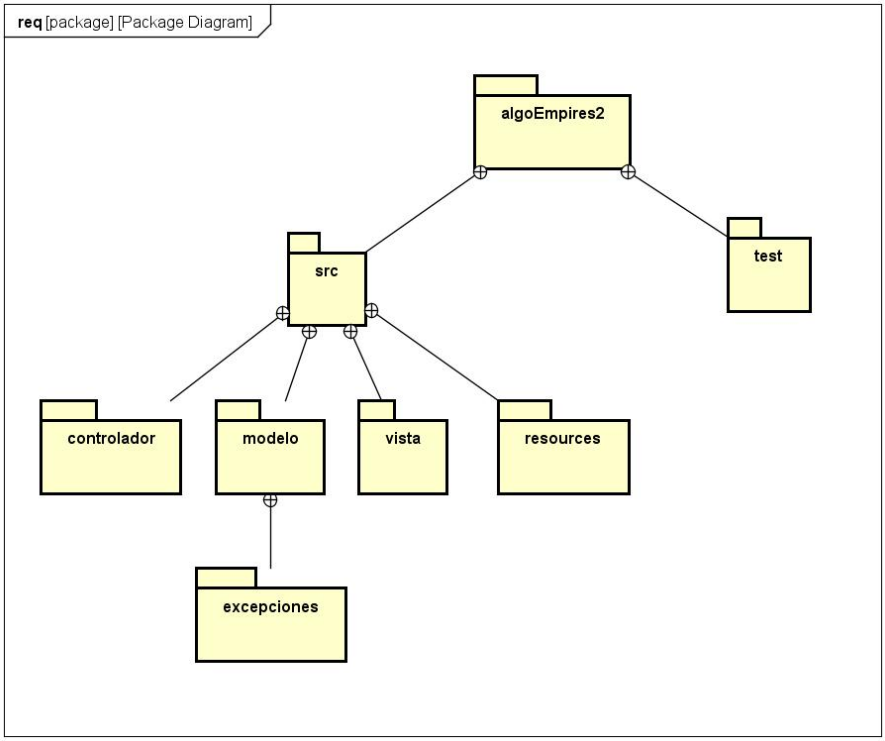

Diagram.jpg

Figura 10: Diagrama de paquetes

5. Diagramas de estado

Diagrama de Estado: Juego

de Estado Juego.jpg

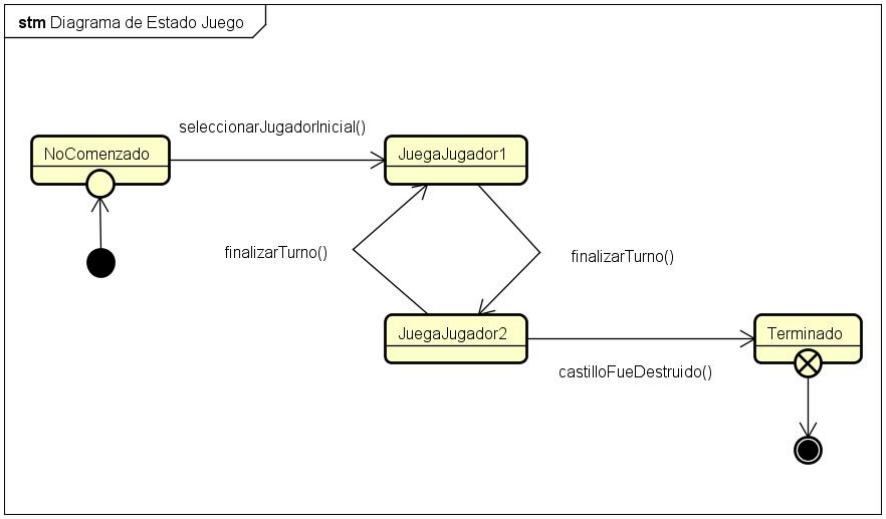

Figura 11: Diagrama de Estado Juego

Diagrama de Estado: Aldeano

de Estado Aldeano.jpg

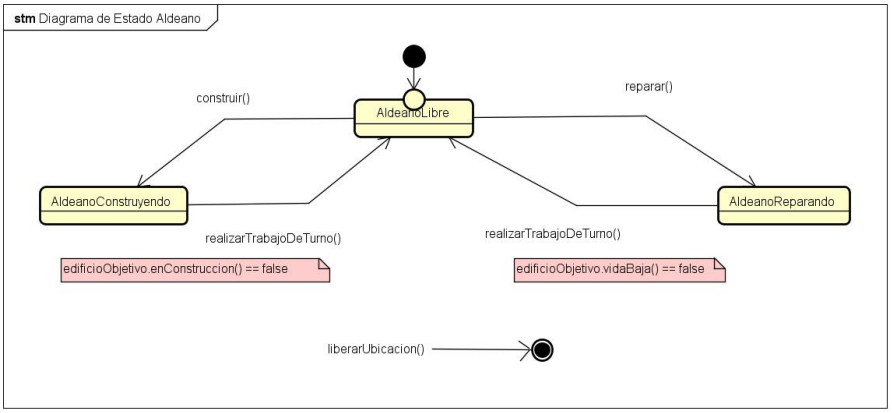

Figura 12: Diagrama de Estado Aldeano

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.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 través del Çontrolador".

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