FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2

Algo Empires

[7507/9502] Algoritmos y Programación III
Curso 1
Segundo cuatrimestre de 2018
Grupo T17

Alumno	Padrón	E-mail
Diaz, Nicolas Andres	101369	diaz.nicolasandres@gmail.com
Mastricchio, Facundo Rodrigo	100874	facundomastricchio@gmail.com
Balladares, Alejandro	101118	alejandroballadaresparisi@gmail.com
Galvan Perez, Jonathan	101250	jonathangalvanperez@gmail.com

Índice

1. Introducción

El presente informe reúne la documentación de la solución del segundo trabajo práctico de la materia Algoritmos y Programación III que consiste en desarrollar una aplicación de un sistema que simula una versión simplificada del juego Age of Empires, junto a su interfaz visual, utilizando los conceptos del paradigma de la orientación a objetos y patrones de diseño vistos hasta ahora en el curso.

2. Supuestos

Debido a que ciertas especificaciones no fueron provistas por la cátedra y en ciertas ocasiones el criterio de cómo abordar determinadas situaciones quedaba a cargo del grupo, se tuvieron en cuenta los siguientes supuestos:

- Un edificio en construcción no recibe daño.
- Una unidad puede moverse o atacar pero no ambas en un mismo turno.
- Las unidades se crean instantáneamente, es decir, no ocupan turnos en crearse.
- Las acciones automáticas como recolectar oro, el ataque del castillo, las reparaciones y construcciones se hacen al final de turno.
- Los edificios solo pueden generar una unidad a la vez.
- Los aldeanos no pueden moverse hasta terminar la actividad que estén haciendo.
- Un edificio que se encuentre en construcción no podrá realizar ninguna acción.
- Un edificio que se encuentre siendo reparado podrá realizar todas sus acciones normalmente.
- Un castillo atacará solo una vez a un edificio sin importar cuantos casilleros del edificio estén dentro del rango de ataque del castillo.

3. Diagramas de clase

En los siguientes diagramas de clases se pueden observar las relaciones entre las mismas.

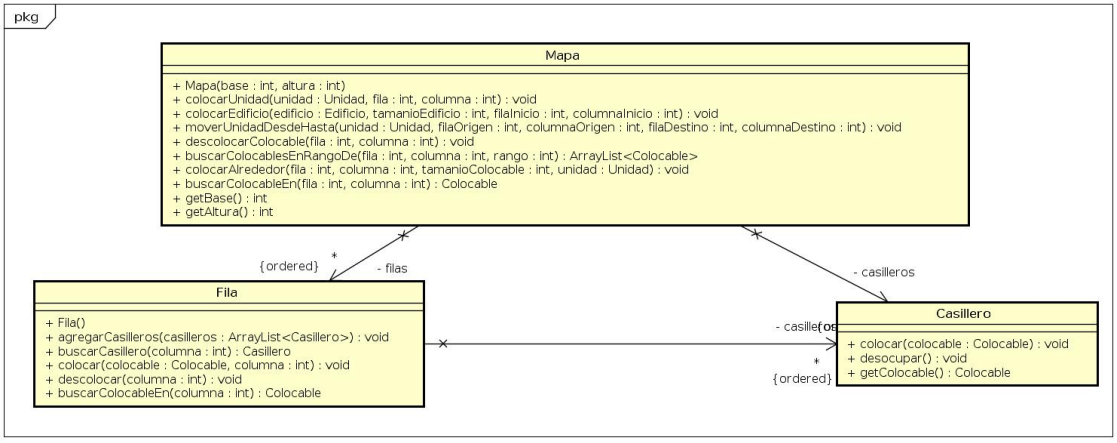

Figura 1: Composición de la clase Mapa

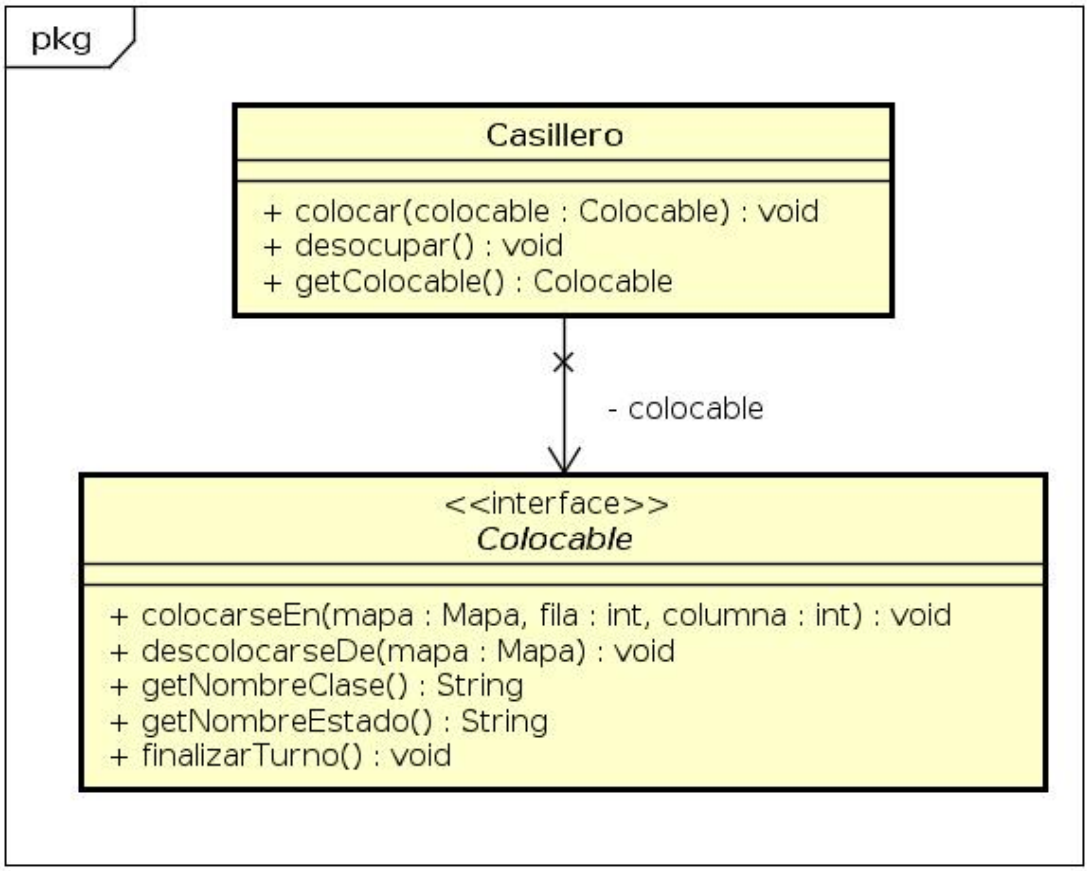

Figura 2: Composición de la clase Casillero

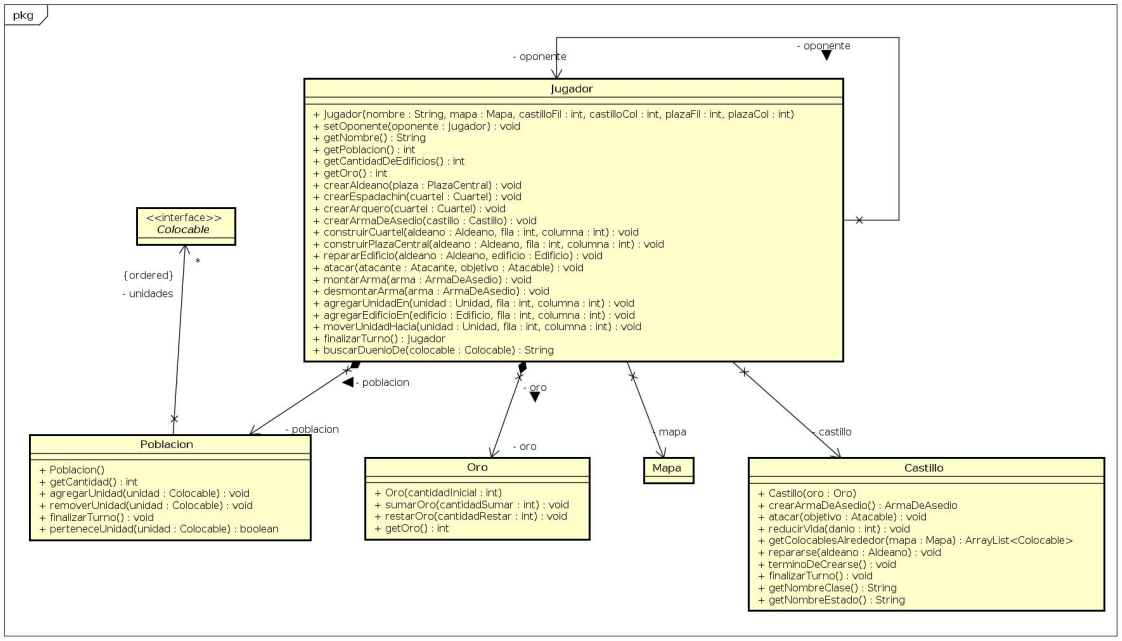

Figura 3: Composición de la clase Jugador

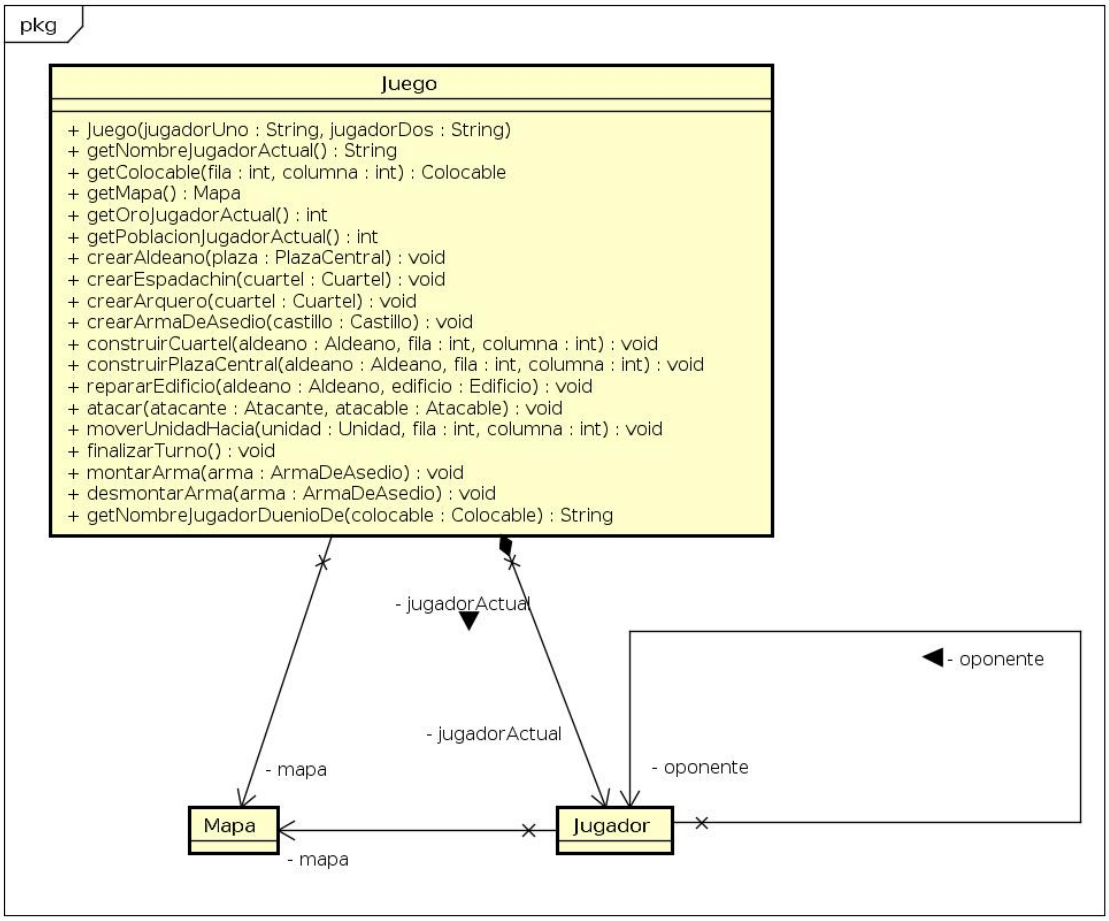

Figura 4: Composición de la clase Juego

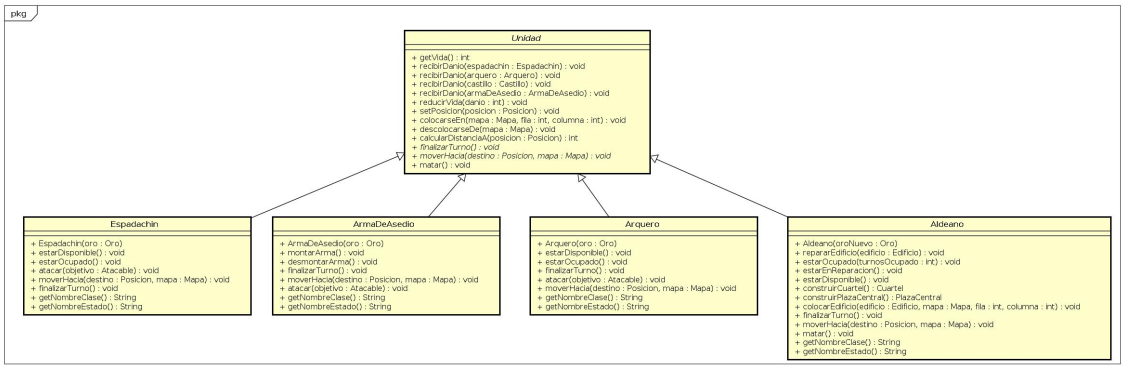

Figura 5: Diagrama de unidades

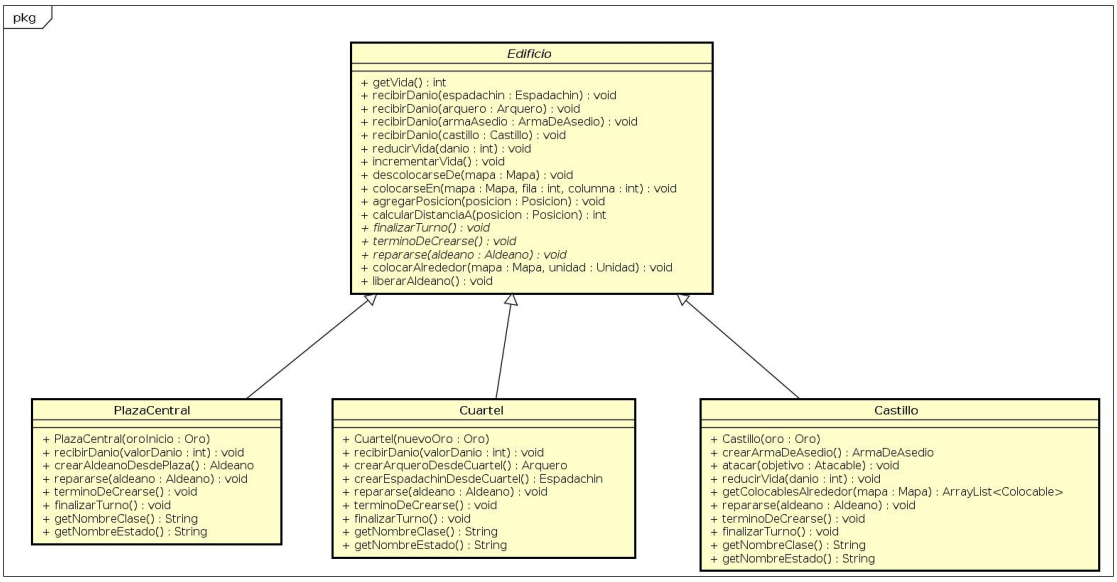

Figura 6: Diagrama de edificios

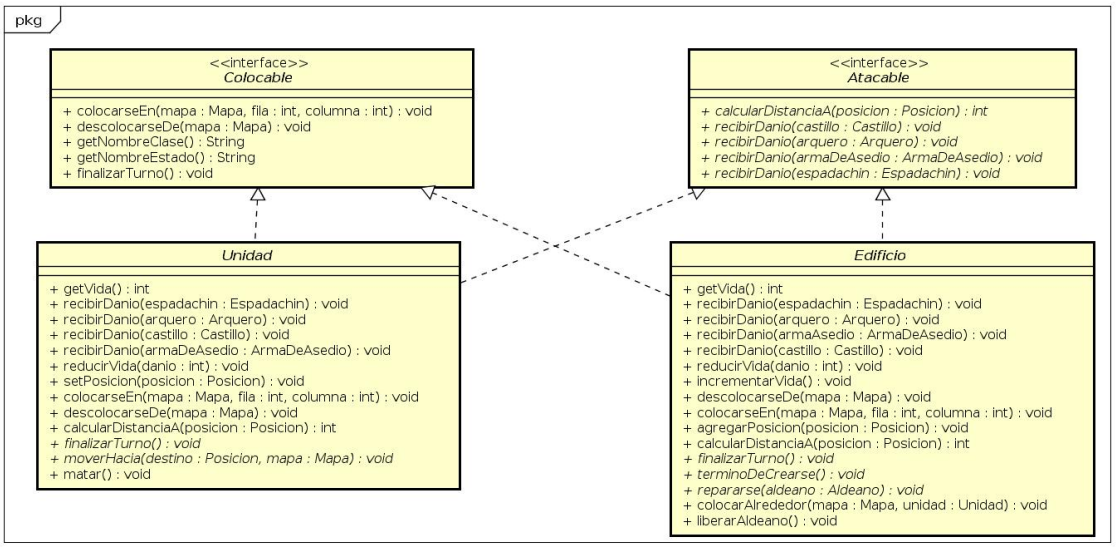

Figura 7: Diagrama de los edificios, unidades y sus interfaces

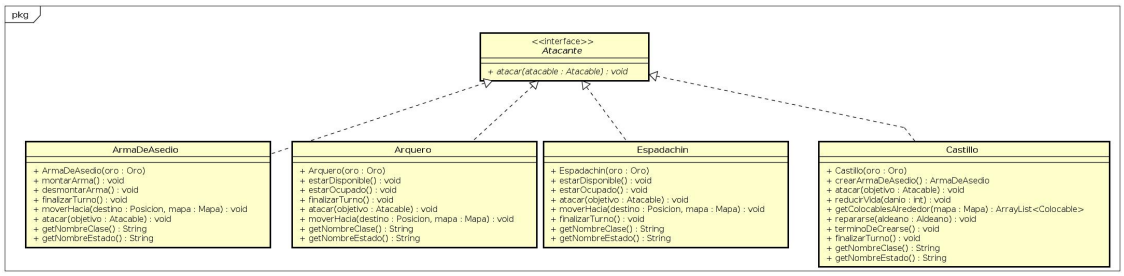

Figura 8: Diagrama de atacantes

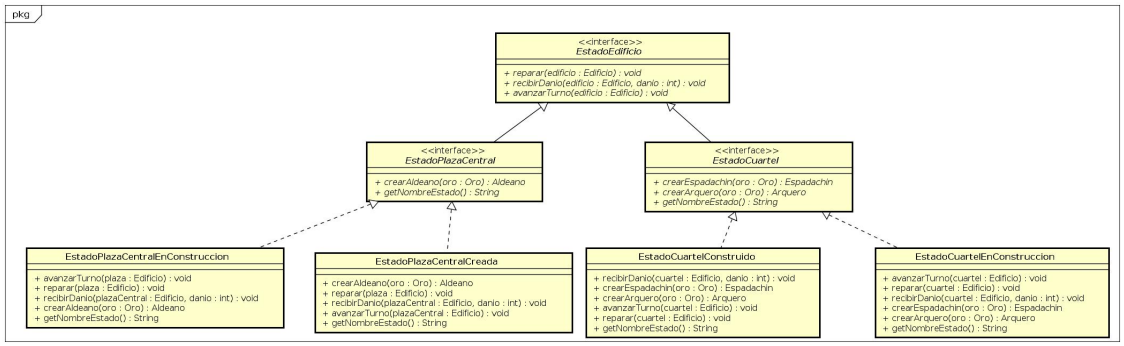

Figura 9: Diagrama de los estados de los edificios

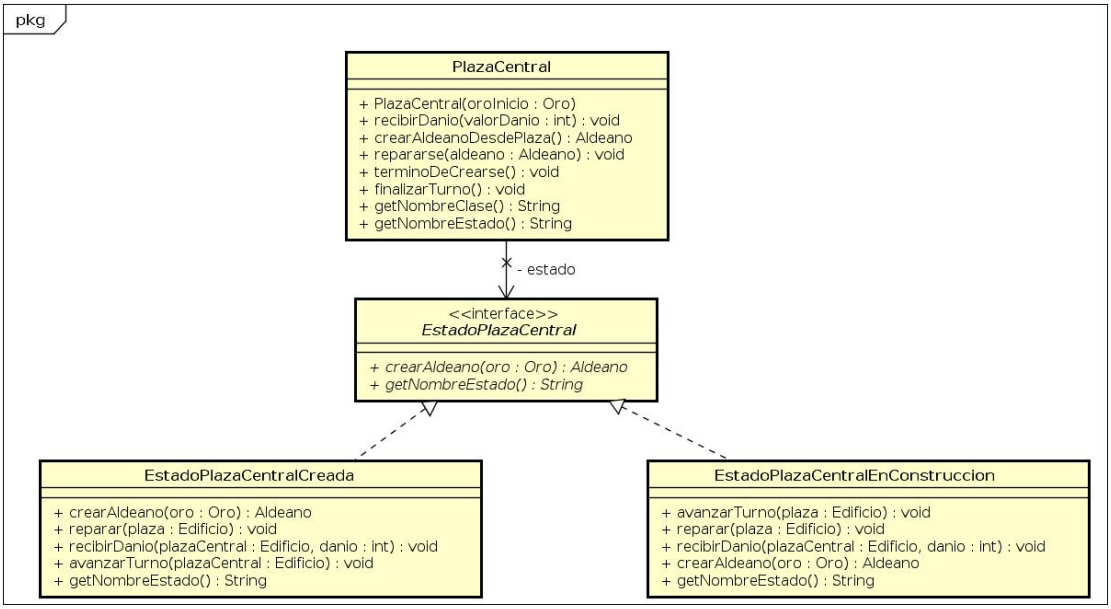

Figura 10: Diagrama de los estados de una plaza central

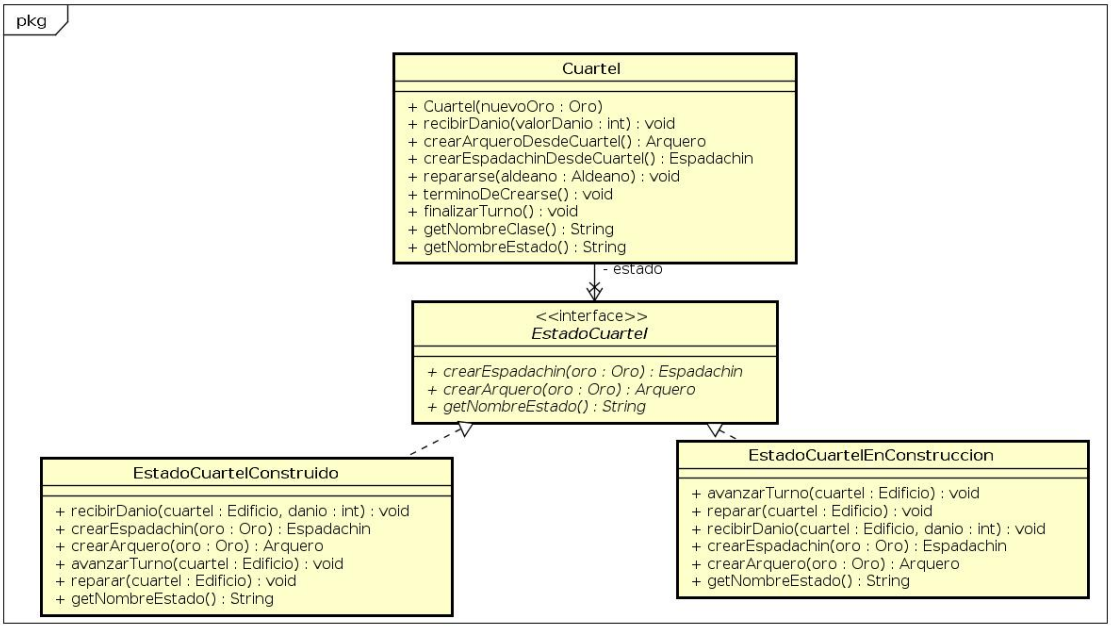

Figura 11: Diagrama de los estados de un cuartel

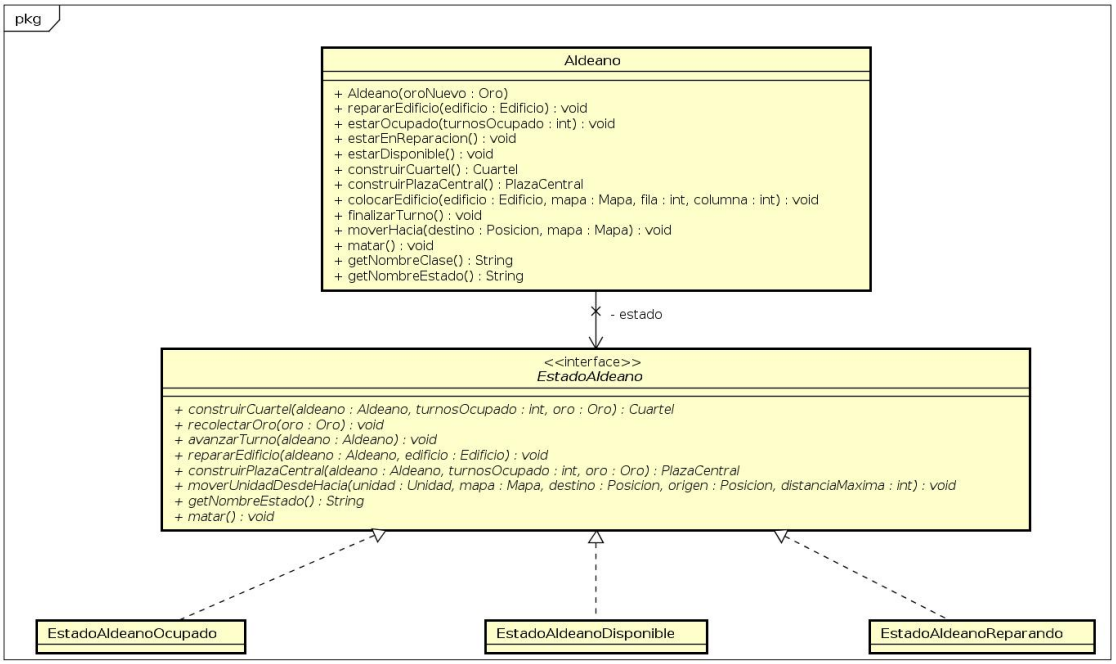

Figura 12: Diagrama de los estados de un aldeano

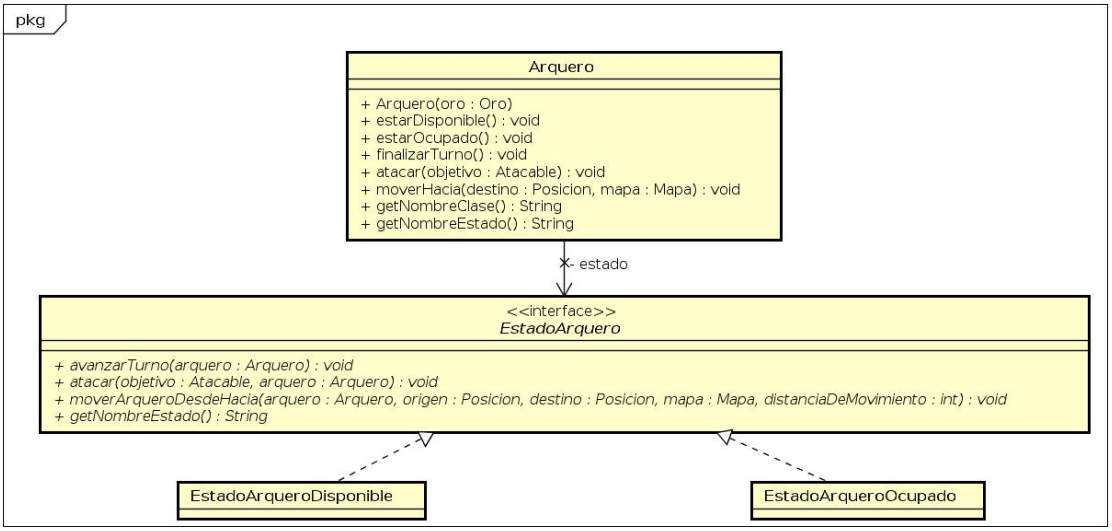

Figura 13: Diagrama de los estados de un arquero

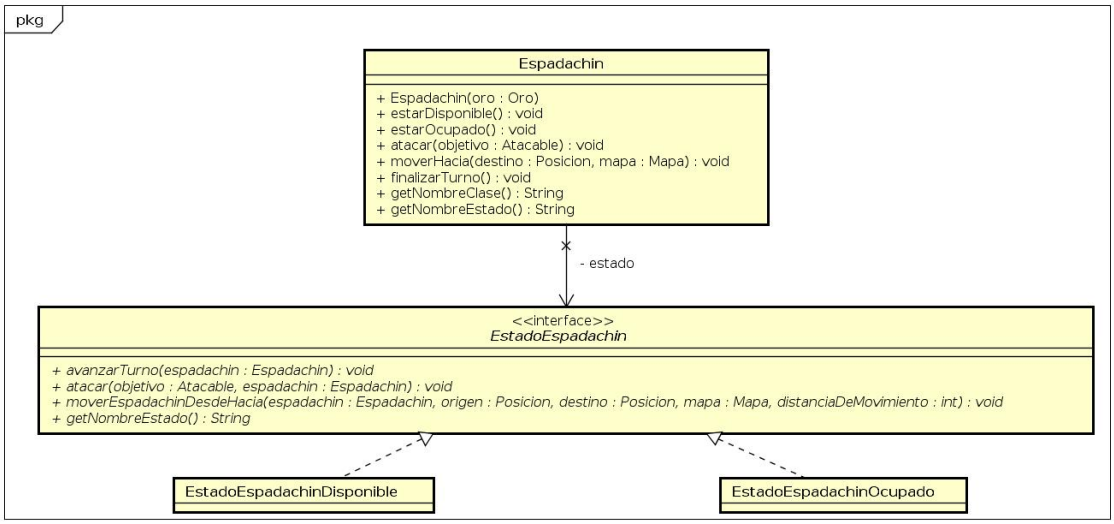

Figura 14: Diagrama de los estados de un espadachín

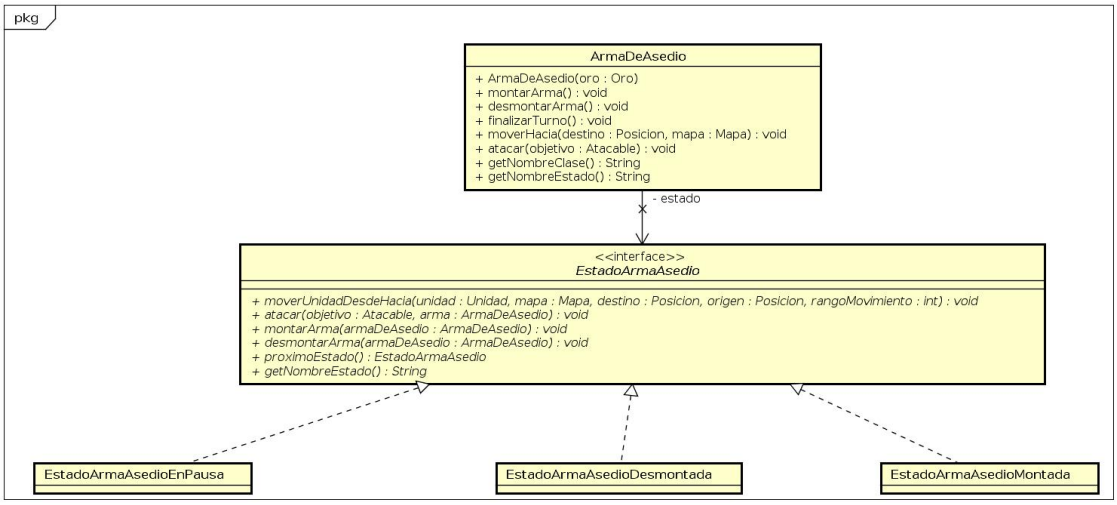

Figura 15: Diagrama de los estados de un arma de asedio

4. Diagramas de secuencia

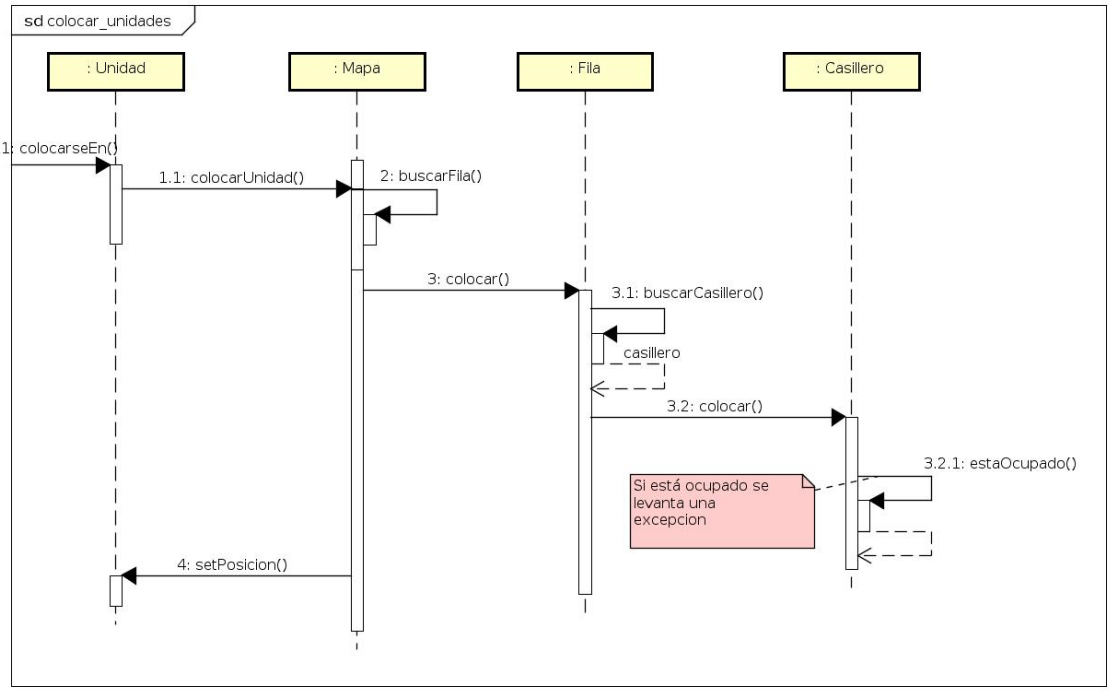

Figura 16: Colocar unidad en mapa

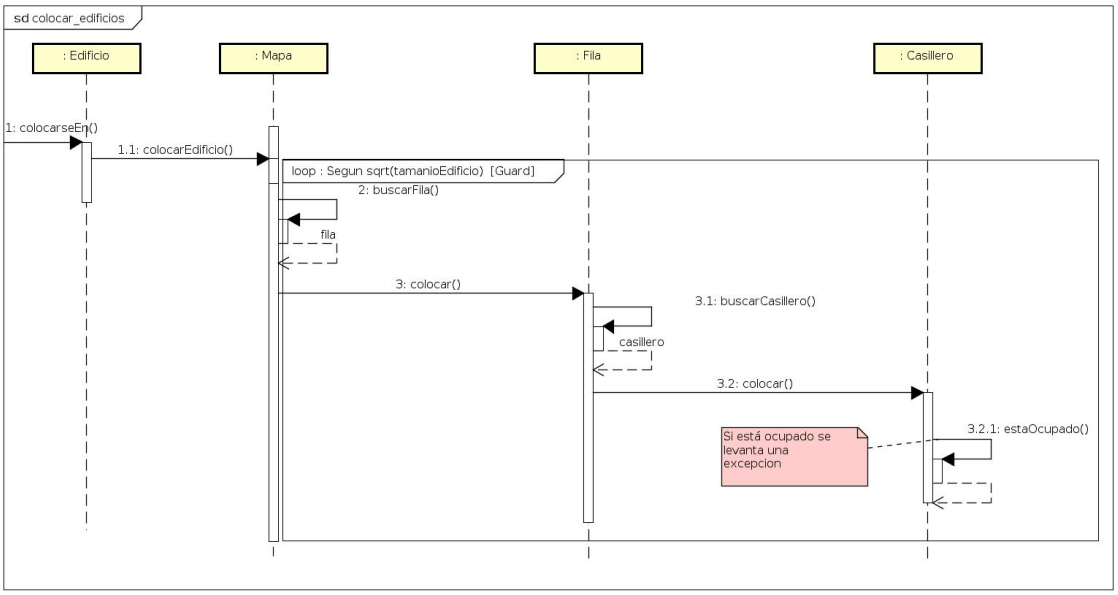

Figura 17: Colocar edificio en mapa

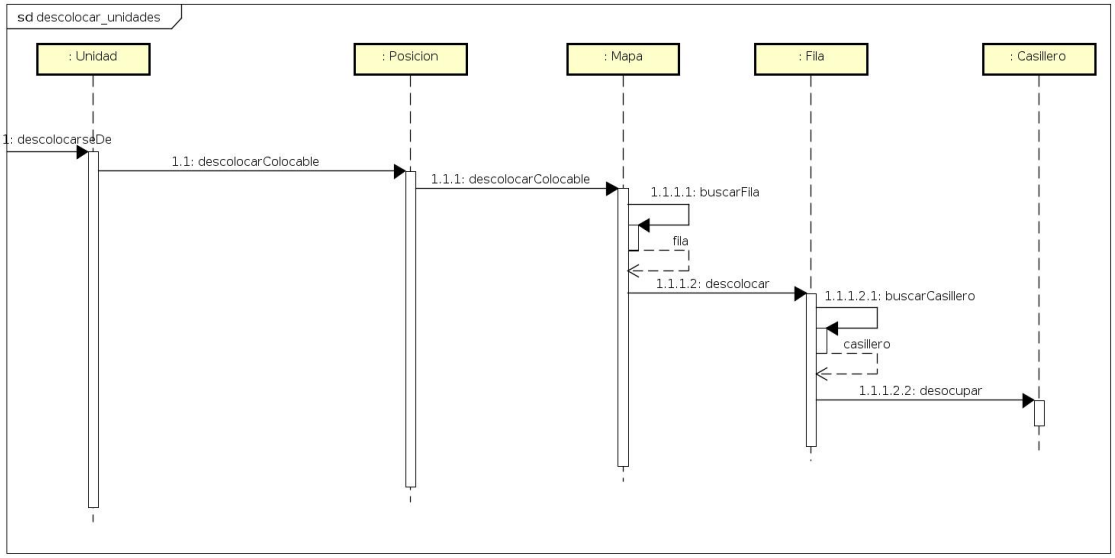

Figura 18: Descolocar una unidad del mapa

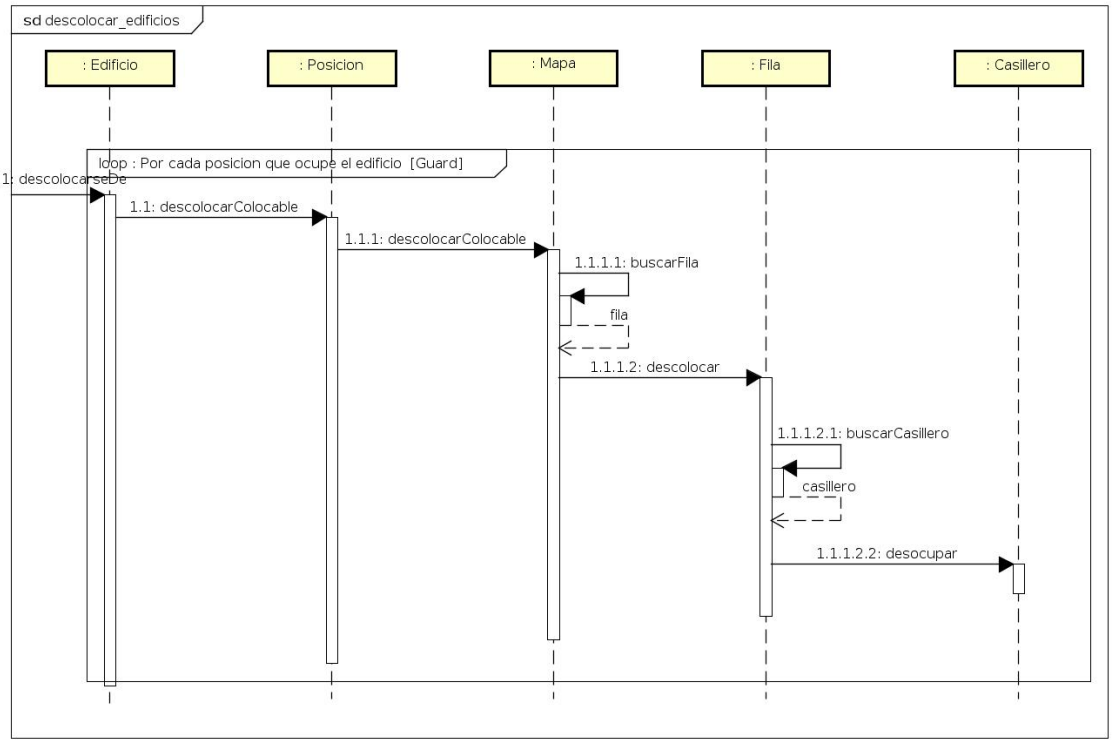

Figura 19: Descolocar un edificio del mapa

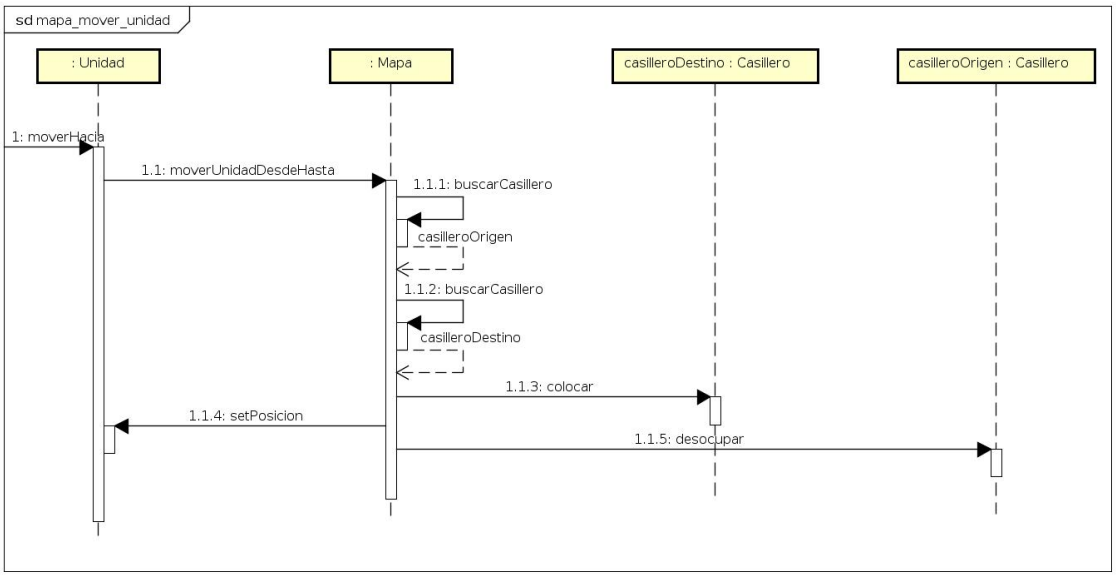

Figura 20: Mover unidad en el mapa

A continuación, se muestra un ejemplo mover unidad disponible según su estado actual. En el resto de unidades la lógica que se presenta es la misma.

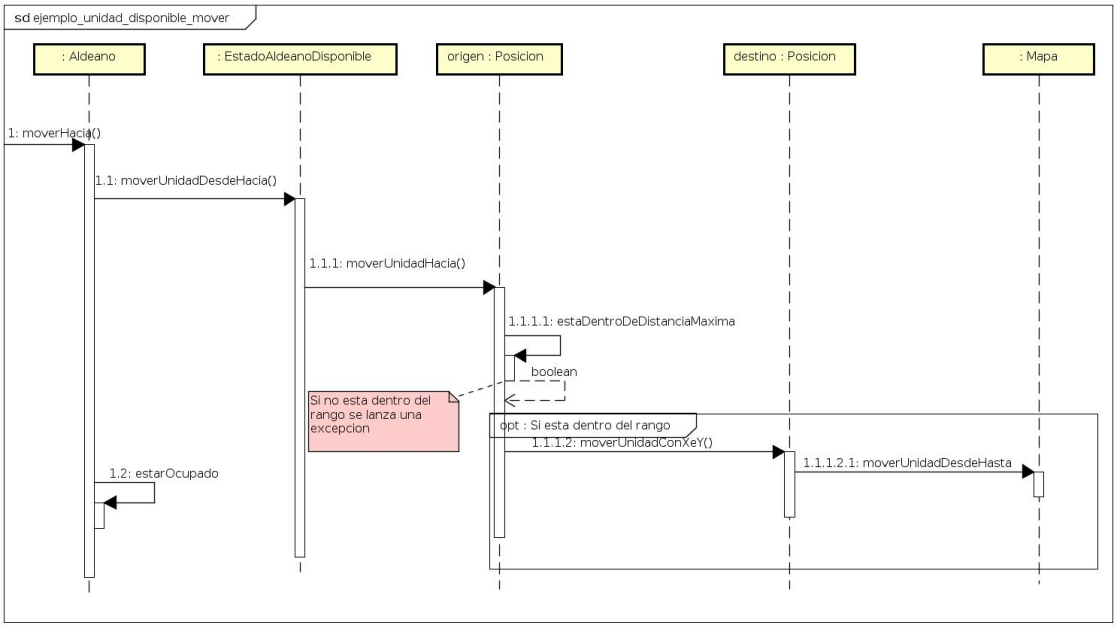

Figura 21: Movimiento de unidad

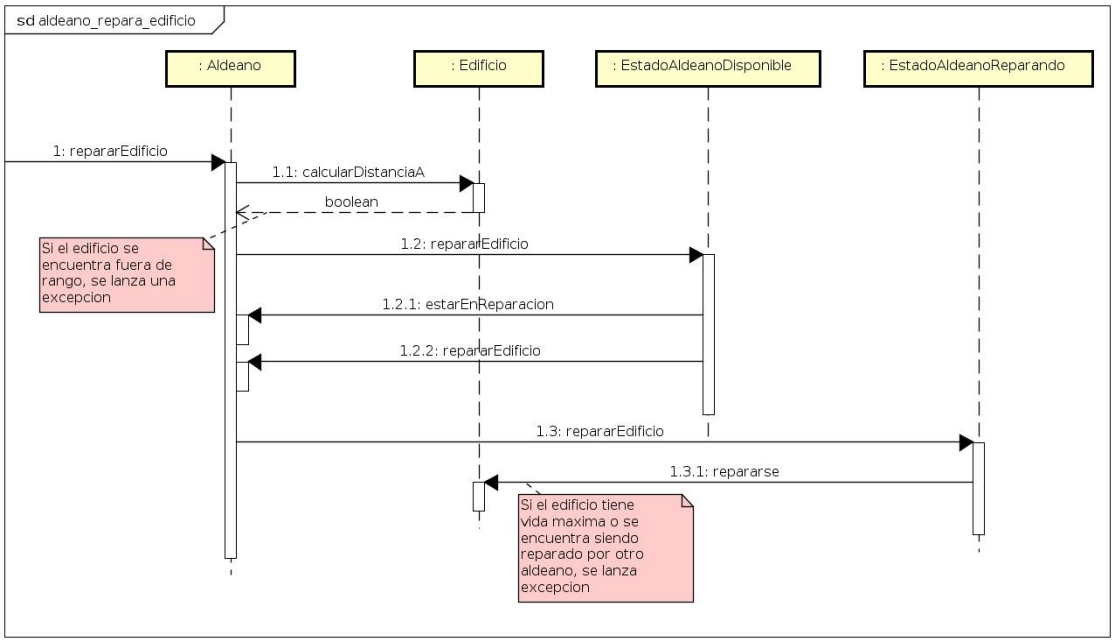

Figura 22: Aldeano repara edificio

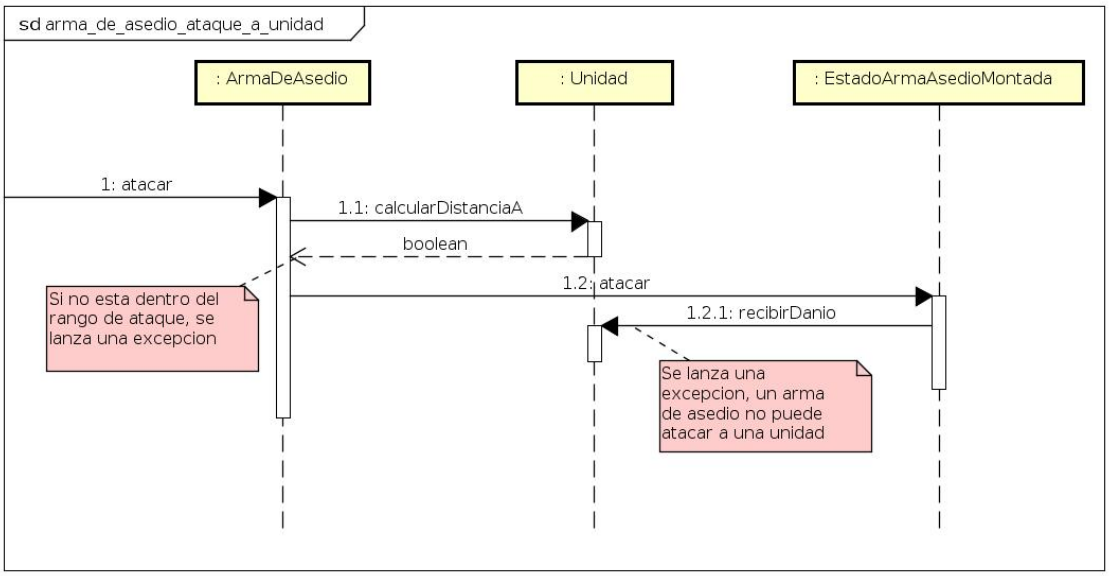

Figura 23: Arma de asedio ataque a unidad

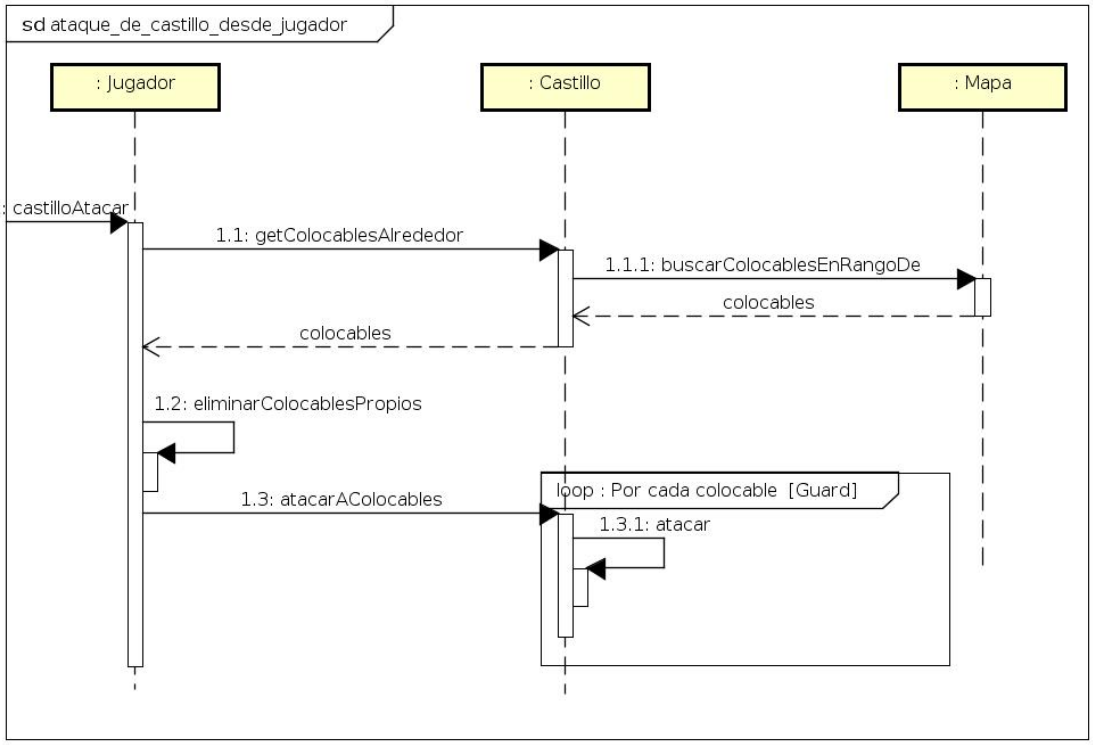

Figura 24: Ataque de castillo desde jugador

A continuación, se muestra un ejemplo de ataque de unidad disponible según su estado actual. En el resto de unidades la lógica que se presenta es la misma.

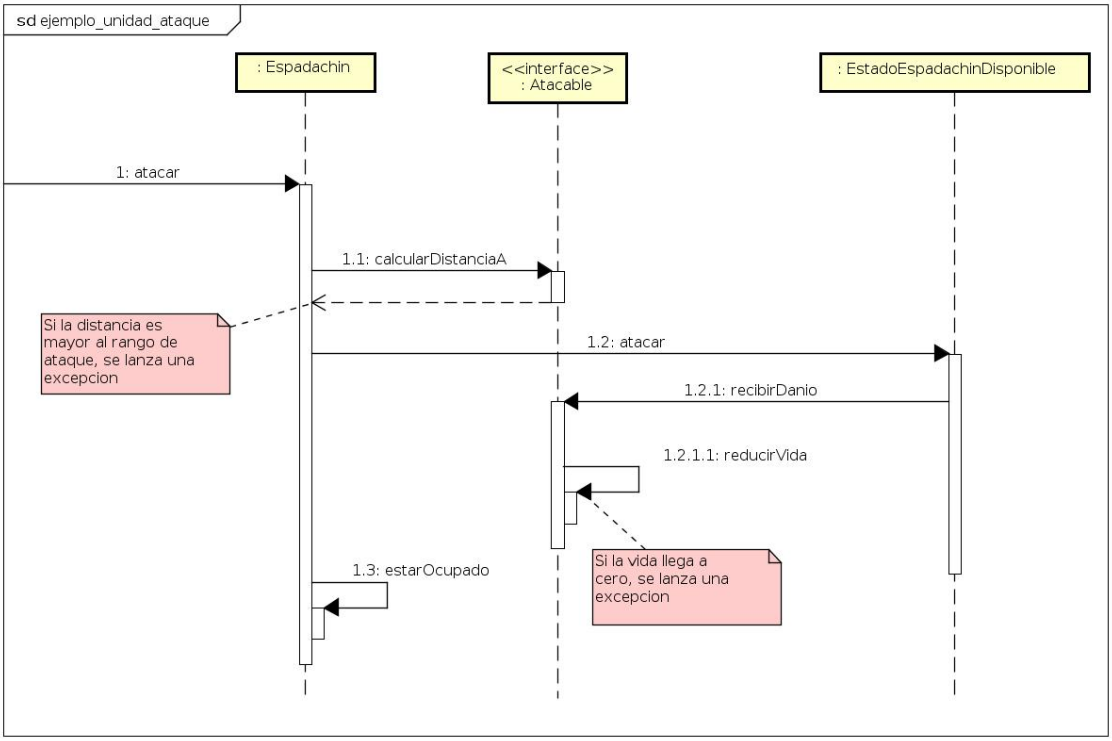

Figura 25: Ataque de unidad disponible

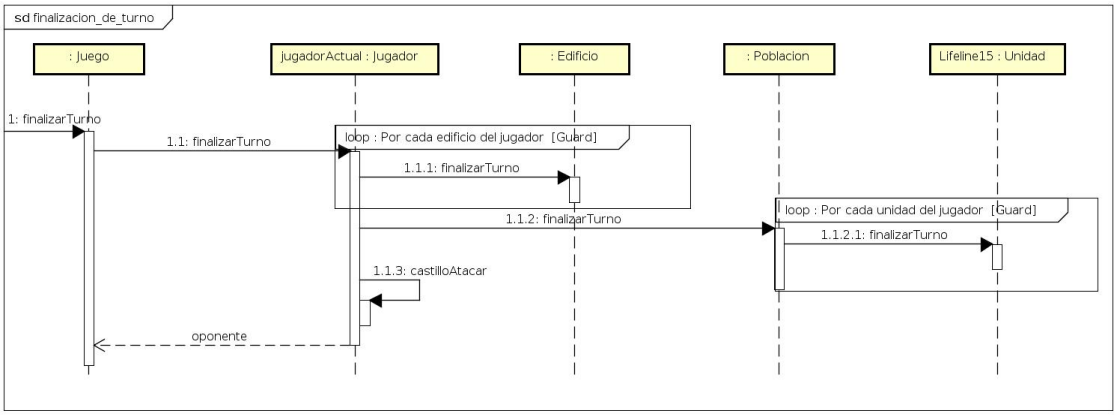

Figura 26: Finalización de turno de juego

A continuación, se muestra un ejemplo de acción (crear aldeano desde plaza) desde la clase Juego. El resto de acciones disponibles siguen la misma lógica: preguntan si las entidades seleccionadas pertenecen al jugador correspondiente y luego se continua con la secuencia producida por los métodos de las clases de más bajo nivel.

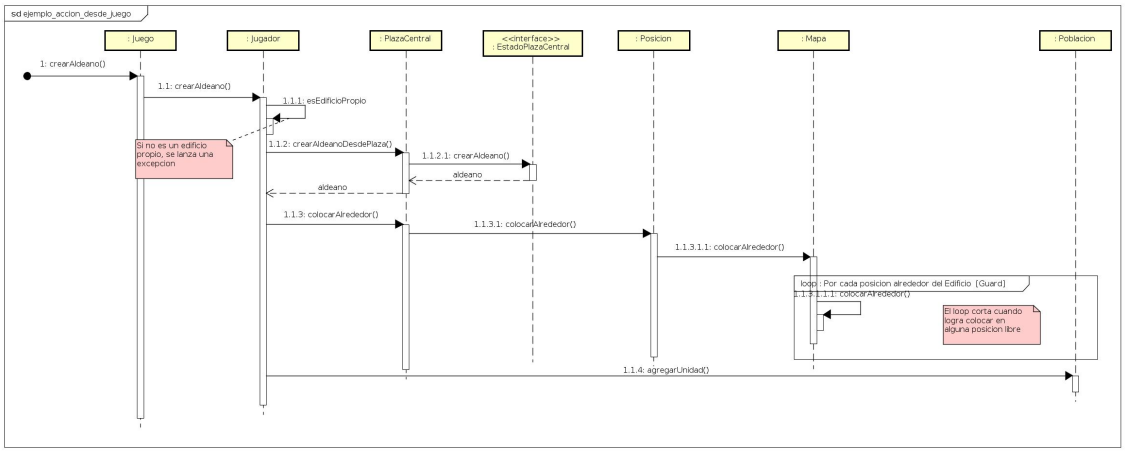

Figura 27: Ejemplo de acción (crear aldeano) desde juego

5. Diagramas de paquetes

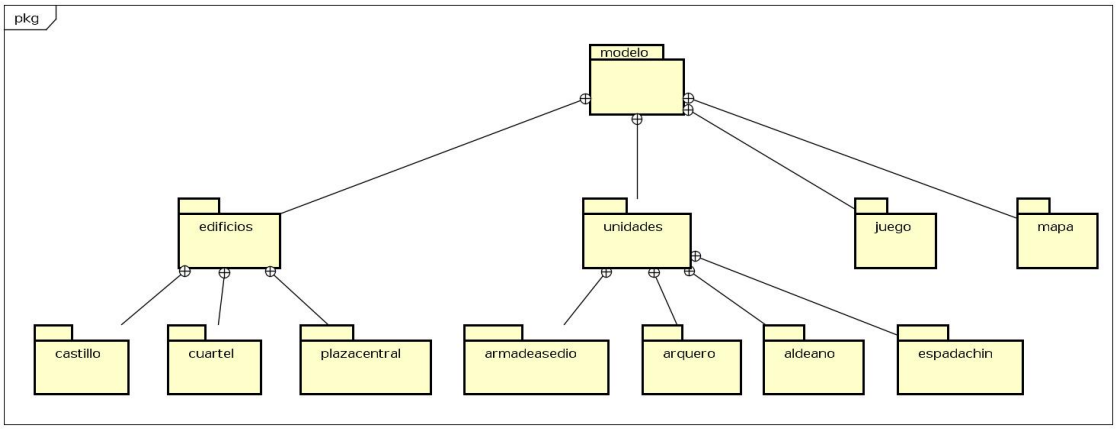

Figura 28: Paquete modelo en general

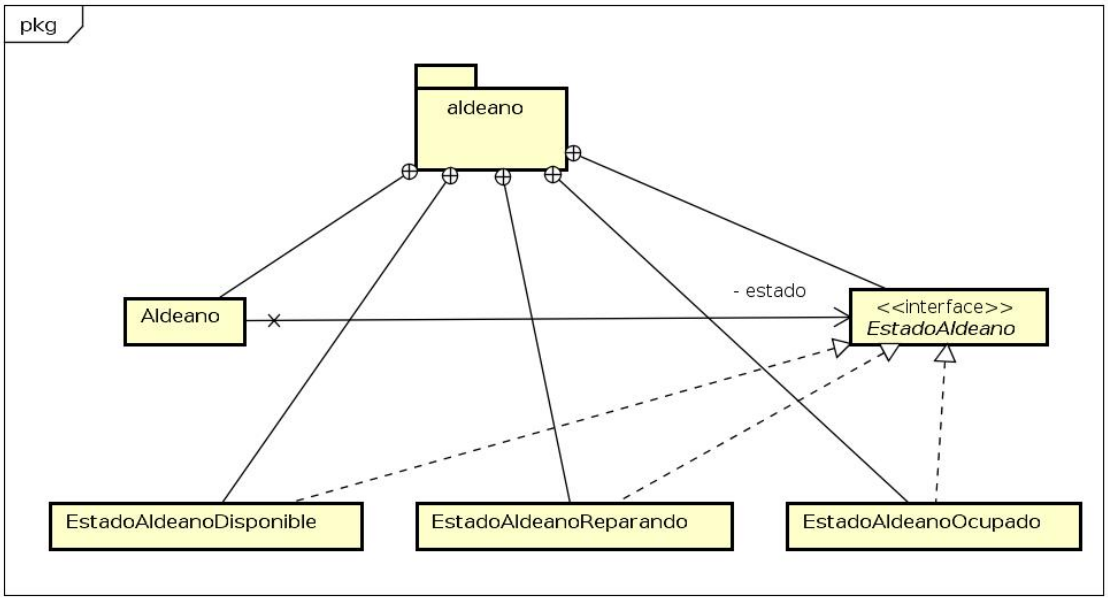

Figura 29: Paquete aldeano

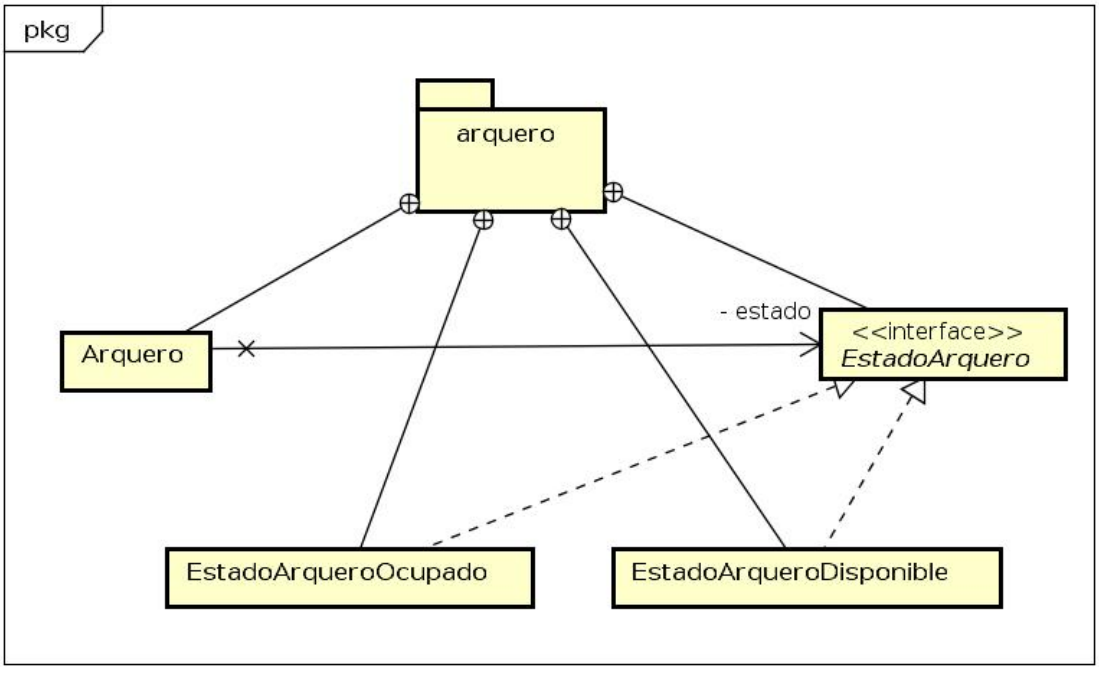

Figura 30: Paquete arquero

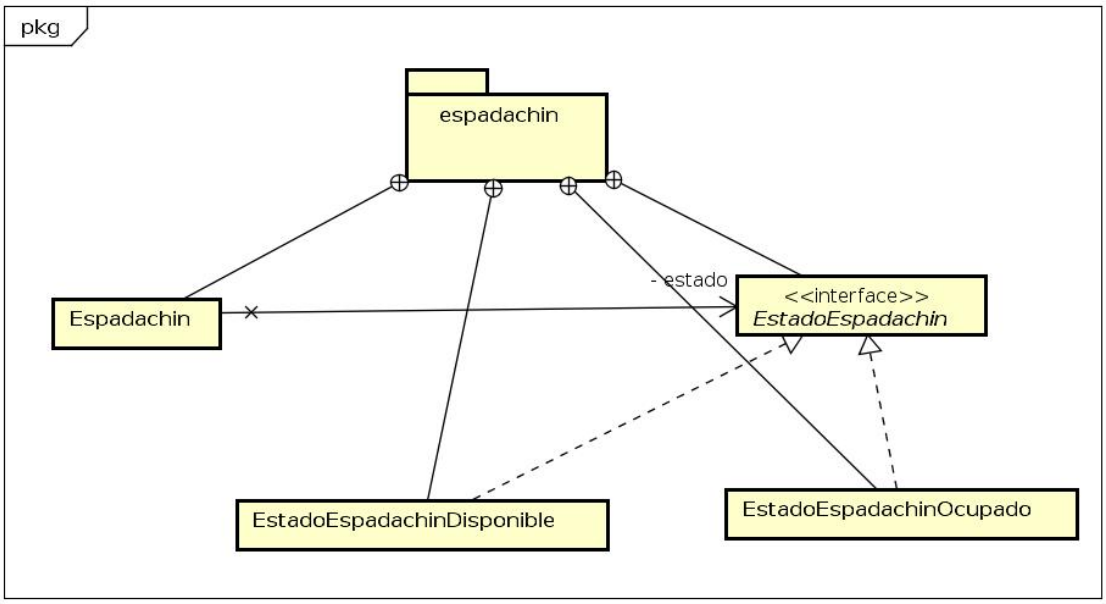

Figura 31: Paquete espadachín

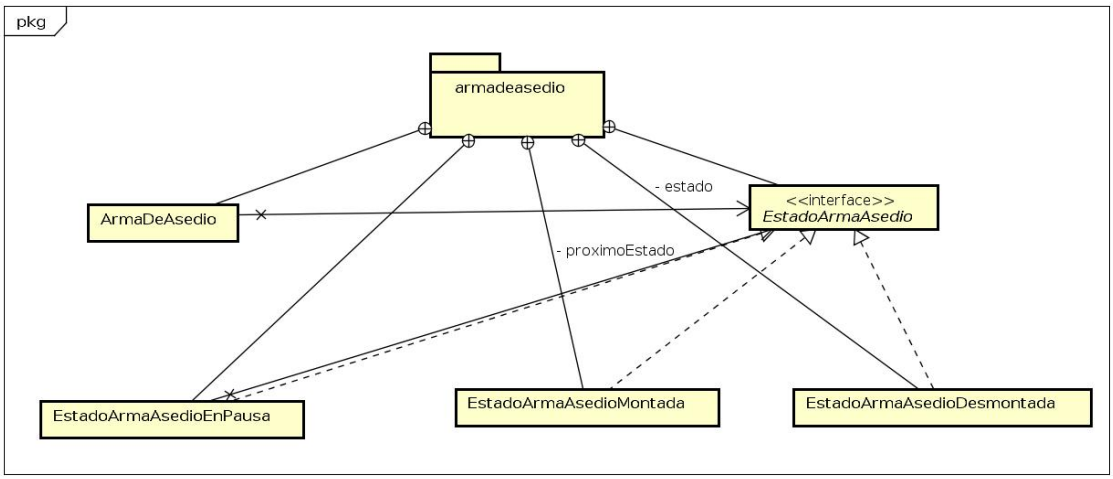

Figura 32: Paquete arma de asedio

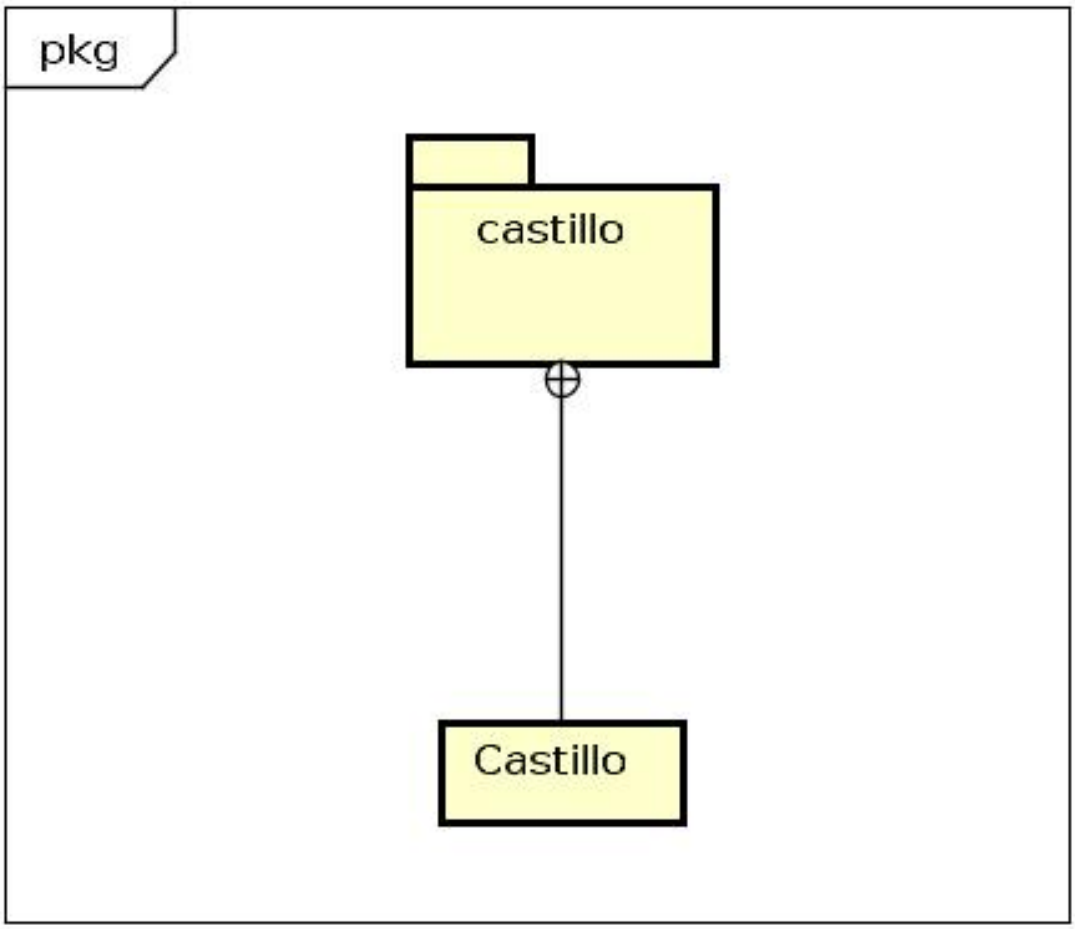

Figura 33: Paquete castillo

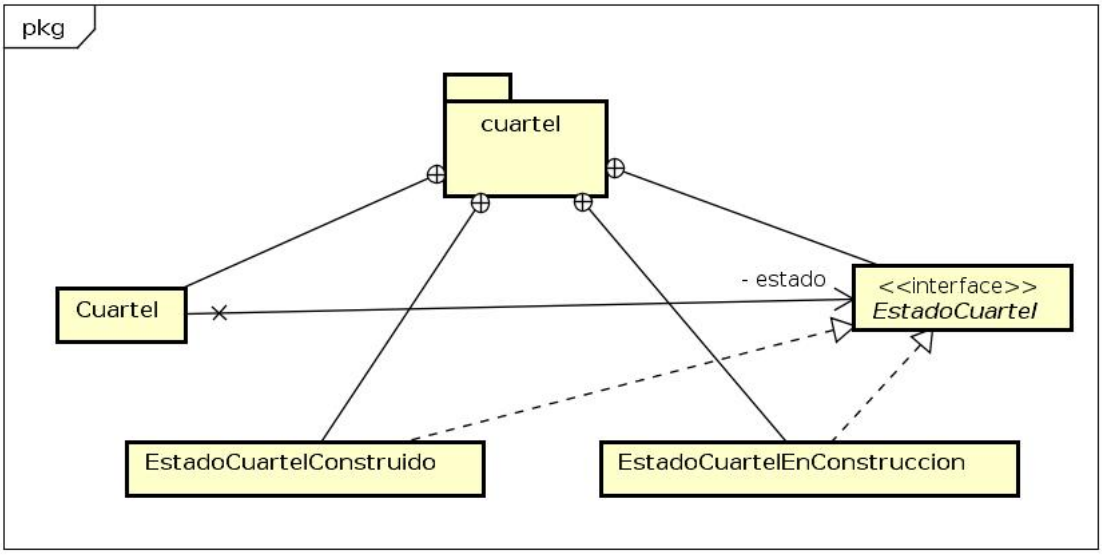

Figura 34: Paquete cuartel

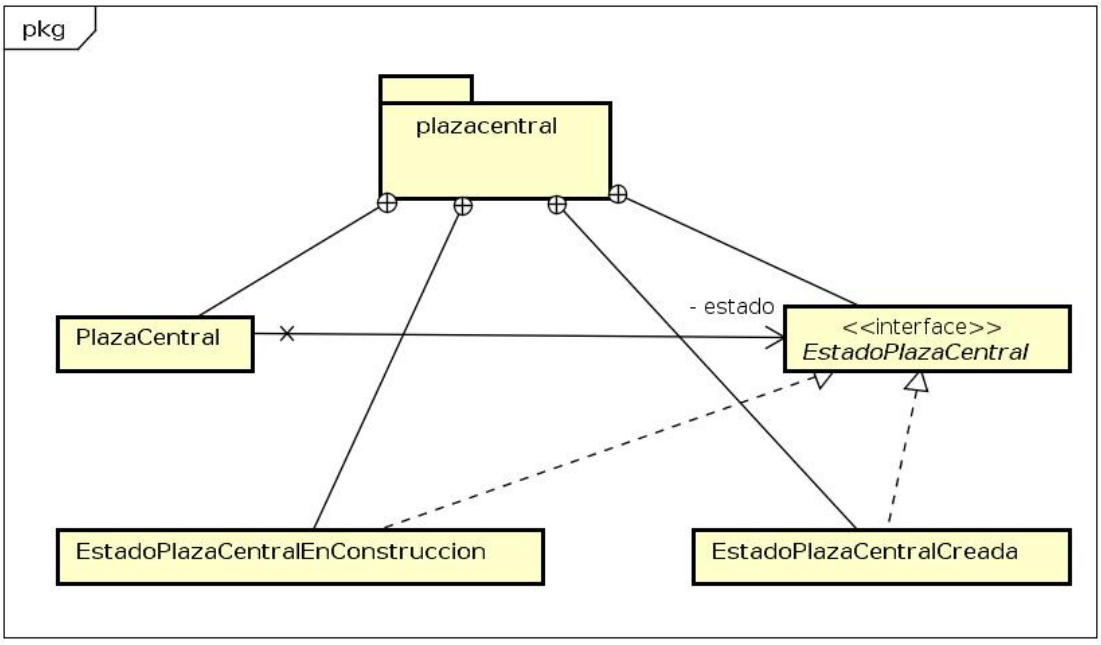

Figura 35: Paquete plaza central

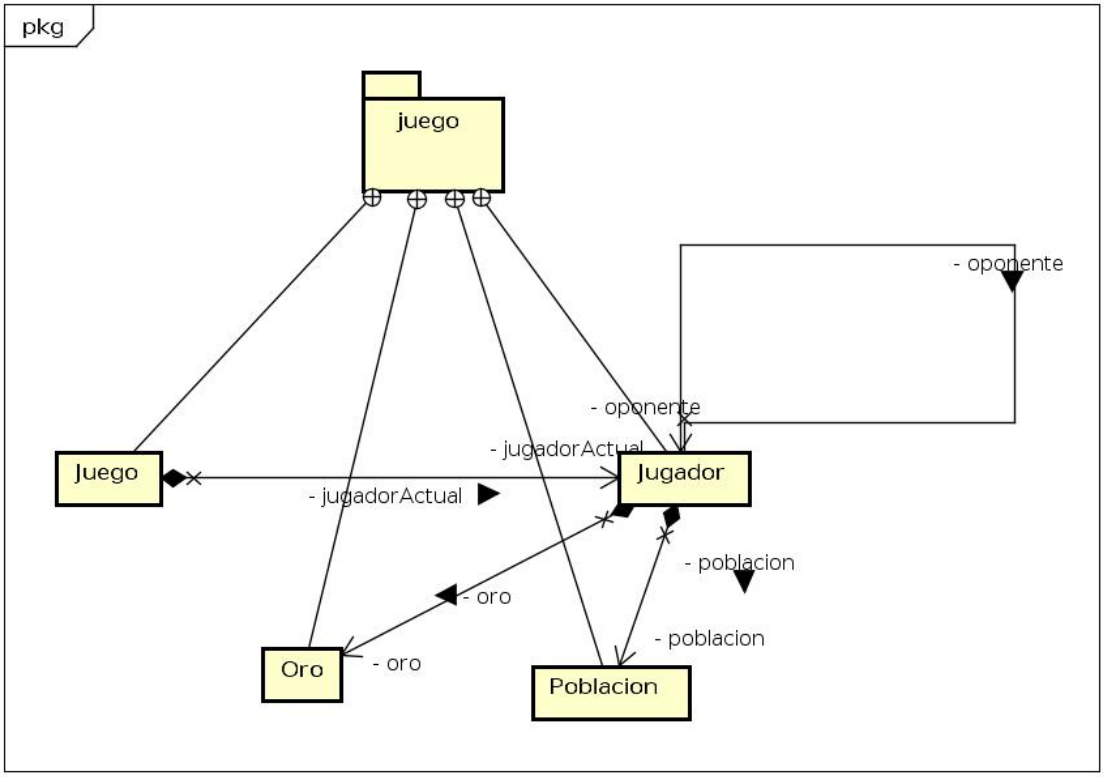

Figura 36: Paquete juego

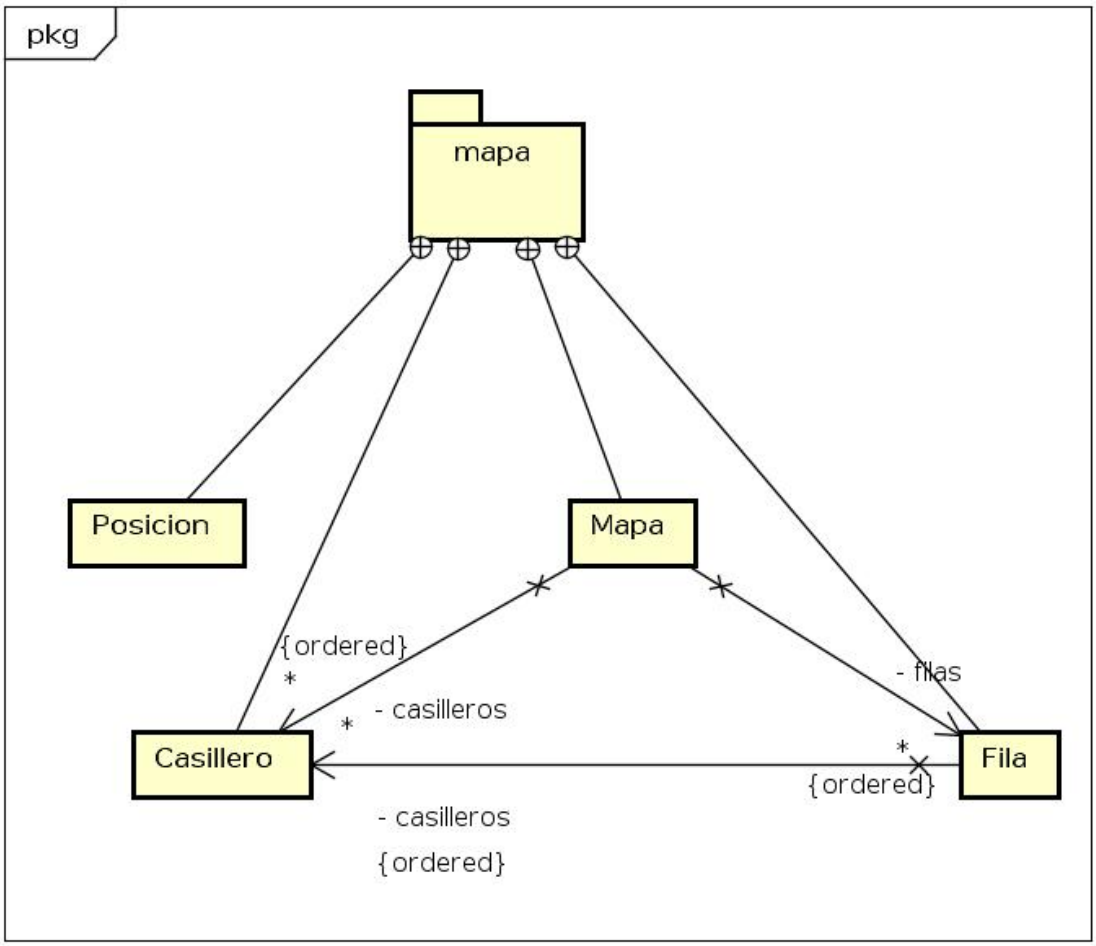

Figura 37: Paquete mapa

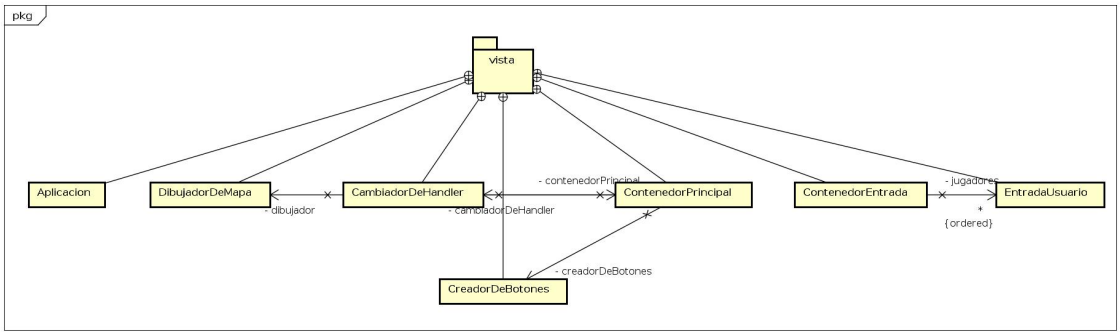

Figura 38: Paquete vista

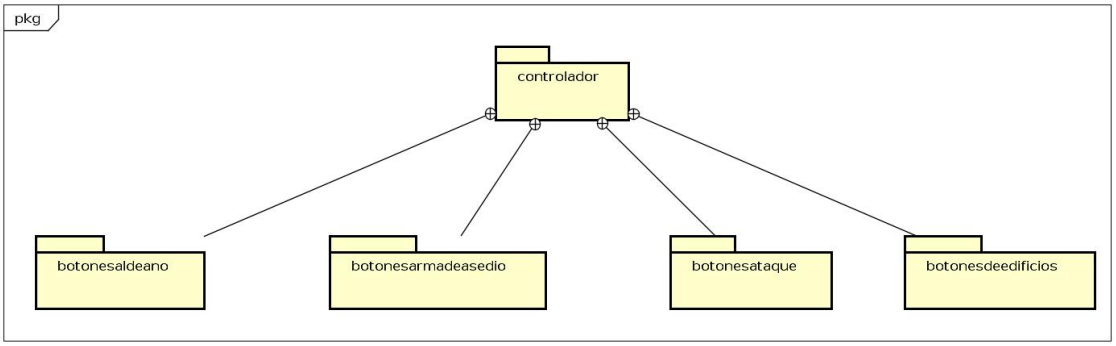

Figura 39: Paquete controlador (general)

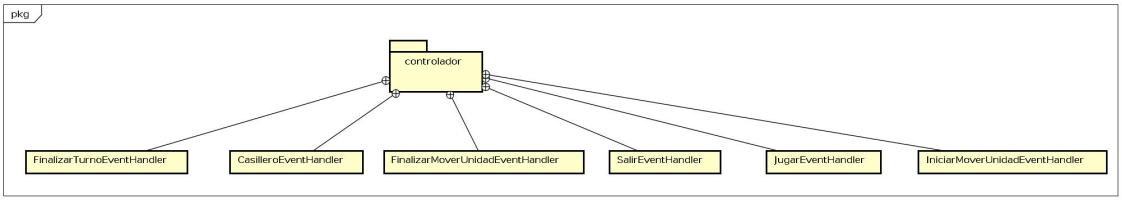

Figura 40: Paquete controlador (inicial)

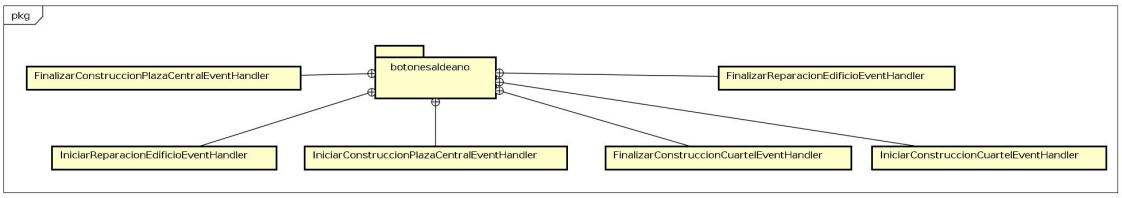

Figura 41: Paquete controlador aldeano

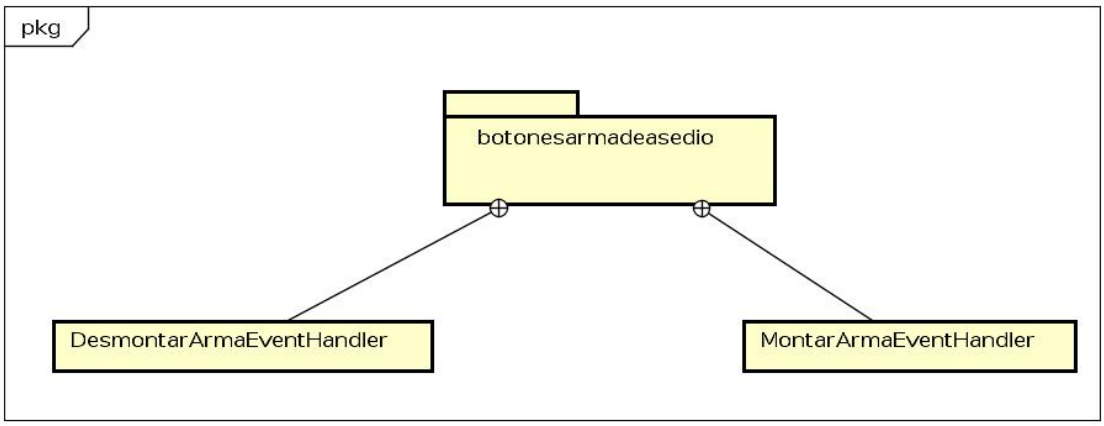

Figura 42: Paquete controlador arma de asedio

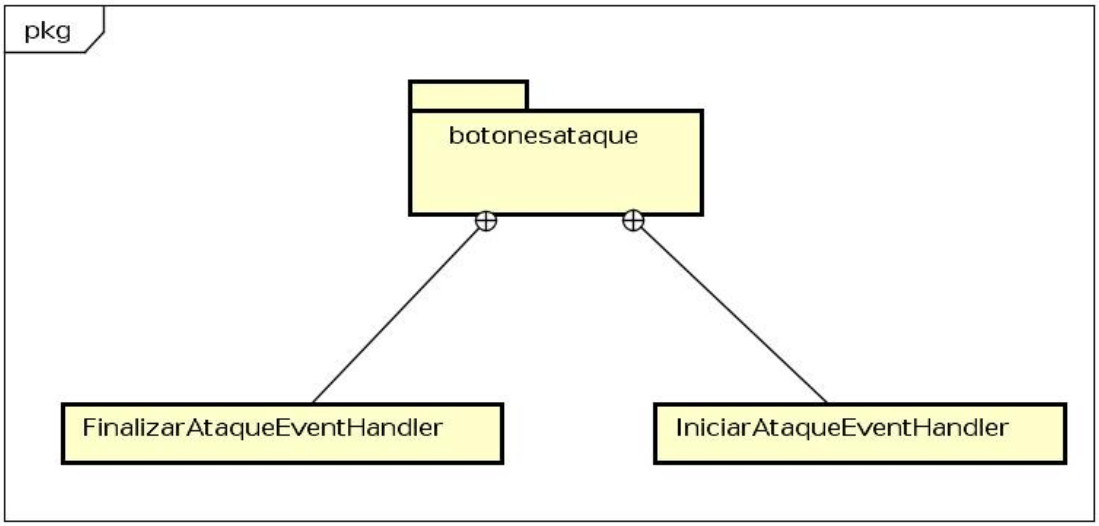

Figura 43: Paquete controlador ataque

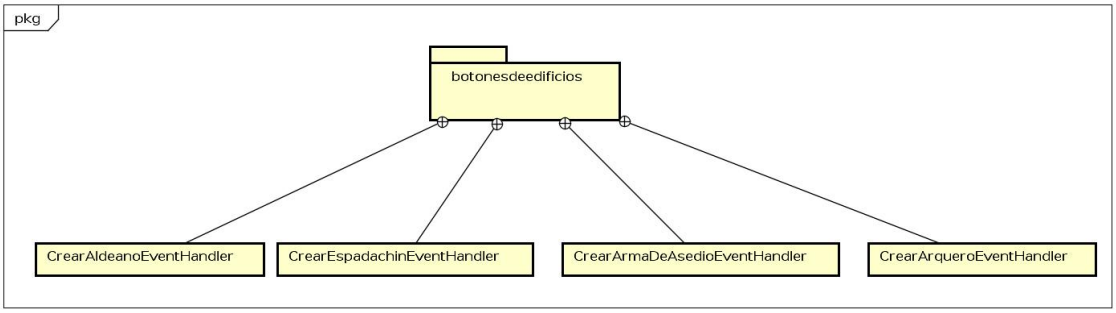

Figura 44: Paquete controlador edificios

6. Diagramas de estado

7. Detalles de implementación

7.1. Implementación de clases Edificio y Unidad

Tanto Edificio como Unidad son dos clases abstractas de las cuales heredan su comportamiento las distintas entidades que tiene el juego. Tienen implementados dos métodos, uno es utilizado para disminuir su vida y otro para aumentarla. A su vez, mediante la utilización de la interfaz Atacable y el patrón de diseño Double Dispatch, permite realizar los ataques. Esto se detallara en la sección 7.4.

7.2. Implementación de clase Aldeano

Aldeano extiende de la clase abstracta Unidad y a su vez implementa el método repararEdificio y construirPlazaCentral. Se utilizo el patrón de diseño State para poder modelar los estados del mismo: ocupado, reparando o libre. Esto permite que el Aldeano modifique su comportamiento cada vez que se cambie de estado interno pareciendo que cambia la clase del objeto. No es el aldeano quien realizar la acción, sino su estado.

7.3. Implementación de clases Espadachín y Arquero

De la misma manera que Aldeano, Espadachín y Arquero heredan de Unidad y utilizan el patrón State, pero además implementa la interfaz Atacante. Gracias a este patrón, se puede modelar que solo se pueda realizar un ataque o movimiento por turno. Esto permite que el Espadachín/Arquero modifique su comportamiento cada vez que se cambie de estado interno pareciendo que cambia la clase del objeto. No es el Espadachín/Arquero quien realiza el ataque sino el estado en el que se encuentra el mismo. Si se intenta atacar o mover cuando el estado es ocupado, se lanzará una excepción.

7.4. Implementación de interfaces Atacante y Atacable

Ante la variedad de ataques disponibles y sus distinciones utilizamos el patrón de Double Dispatch. Si bien, según algunos autores, no está considerado como un patrón de diseño, consideramos apropiado mencionarlo. Al realizar el ataque, el atacante se pasa a si mismo (this) como parámetro y es el atacable quien sabe como restarse la vida según quien sea el atacante. Por ejemplo, Espadachín ataca a Arquero de la siguiente manera: espadachín.atacar(arquero, this). Luego, internamente Arquero reduce su vida en función que Espadachín fue quien lo ataco ya que utiliza la interfaz Atacable que en este caso tendría la firma del método arquero.recibirDanio(espadachín).

7.5. Implementación de clase ArmaDeAsedio

Utiliza la interfaz Atacante ya explicada anteriormente para realizar sus ataques y el patrón State para poder modelar cuando el arma se encuentra montada o desmontada. Además, tiene un tercer estado el cual es en pausa debido a que al montar o desmontar, en ese mismo turno no se puede realizar otra acción. Este ultimo estado es el que modela dicha situación. Gracias a este patrón (State), se puede modelar que solo se pueda realizar un ataque, movimiento o montar/desmontar por turno. Esto permite que el ArmaDeAsedio modifique su comportamiento cada vez que se cambie de estado interno pareciendo que cambia la clase del objeto.

7.6. Implementación de clases PlazaCentral y Cuartel

Las clases PlazaCentral y Cuartel heredan de la clase abstracta Edificio, y a su vez volvemos a utilizar el patrón State, ya que éstos tienen dos estados. Uno de ellos es en construcción, en el cual el edificio todavía no puede realizar acciones. Una vez construido, se cambia de estado interno y se pueden realizar las acciones pertinentes correctamente. Esto permite que la PlazaCentral y el Cuartel modifiquen su comportamiento cada vez que se cambie de estado interno pareciendo que cambia la clase del objeto.

7.7. Implementación de clase Juego

La clase Juego proporciona una interfaz unificada para un conjunto de interfaces (métodos de Jugador) implementando el patrón Facade. La idea era definir una interfaz de alto nivel que haga que el subsistema sea mas fácil de usar minimizando la comunicación de dependencias entre subsistemas. Además, esto nos permite que en el caso de que haya cambios de implementaciones en las clases de más bajo nivel, no se vea el uso de dichos subsistemas.

8. Excepciones

La mayoría de las excepciones son atrapadas en los métodos de los controladores, y dependiendo de la excepción, se mostrará por pantalla un mensaje diferente informando al usuario de la situación.

AldeanoEstaOcupadoException se lanza cuando ordenan a un aldeano reparar, moverse o construir pero ya está ocupado.

ArmaDeAsedioYaSeEncuentraDesmontadaException se lanza cuando se intenta desmontar un arma de asedio que ya se encuentra desmontada.

ArmaDeAsedioYaSeEncuentraMontadaException se lanza cuando se intenta montar un arma de asedio que ya esta montada.

ArqueroYaFueUtilizadoEnEsteTurnoException se lanza cuando se intenta mover o atacar con un arquero que ya fue utilizado en este turno.

CasilleroOcupadoException se lanza cuando se quiere ubicar una unidad o edificio en un casillero ocupado.

CastilloFueDestruidoException se lanza cuando el castillo es destruido. Además, se usa para determinar el final del juego.

ColocableFueraDeRangoDeAtaqueException se lanza cuando se intenta atacar a algún colócale que se encuentra a una distancia superior al rango de ataque del atacante.

ConstruccionFueraDeRangoException se lanza cuando una aldeano intenta construir o reparar un edificio en una posición no adyacente a él.

CoordenadasInvalidasException se lanza cuando el usuario quiera ubicar algo fuera del mapa.

CuartelCreandoseException se lanza cuando se le pide al cuartel realizar acciones cuando todavía no termino de construirse.

DistanciaInvalidaException se lanza cuando se intenta utilizar una distancia negativa a la hora de mover una unidad.

EdificioFueDestruidoException se lanza cuando un edificio es destruido. Sirve para avisarle al Jugador atacado que ya no posee dicho edificio y debe ser removido del mapa.

EdificioObjetivoEsPropioException se lanza cuando una unidad intenta atacar a un edificio del mismo jugador.

EdificioSeleccionadoNoPerteneceAJugadorException se lanza cuando un jugador intenta realizar una acción (crear, mover, atacar, etc.) con un edificio que no le pertenece.

EdificioSiendoReparadoException se lanza cuando un aldeano intenta reparar un edificio que ya se encuentra siendo reparado.

EdificioTieneVidaMaximaException se lanza cuando se intenta reparar un edificio que posee vida máxima. Además, se utiliza para liberar al aldeano que se encuentre reparando cuando el edificio alcanza su vida máxima.

EspadachinYaFueUtilizadoEsteTurnoException se lanza cuando se intenta mover o atacar con un espadachín que ya fue utilizado en este turno.

LimiteDePoblacionAlcanzadoException se lanza cuando un jugador intenta crear una nueva unidad pero ya alcanzó el limite de población.

NoHayLugarSuficenteParaColocarEdificioException se lanza cuando se intenta colocar un edificio en una zona donde no se encuentran disponible los casilleros para dicha acción.

NoSePuedeAtacarArmaAsedioDesmontadaException se lanza cuando se intenta atacar con un arma de asedio desmontada.

NoSePuedeAtacarConArmaDeAsedioEnPausaException se lanza cuando se intenta atacar con un arma de asedio en pausa.

NoSePuedeDesmontarArmaDeAsedioEnPausaException se lanza cuando se intenta desmontar un arma de asedio en pausa.

NoSePuedeMontarArmaDeAsedioEnPausaException se lanza cuando se intenta montar el arma de asedio en pausa.

NoSePuedeMoverArmaAsedioMontadaException se lanza cuando se intenta mover el arma de asedio montada.

NoSePuedeMoverArmaDeAsedioEnPausaException se lanza cuando se intenta mover el arma de asedio en pausa.

OroInsuficienteException se lanza cuando se quiera crear unidades o edificios sin tener oro el suficiente.

PlazaCentralEnConstruccionException se lanza cuando se le pide a una plaza central crear aldeanos cuando todavía no fue terminada de construir.

PosicionFueraDeRangoException se lanza cuando se intenta mover una unidad hacia una posición fuera del rango de movimiento de dicha unidad.

TamanioInvalidoException se lanza cuando se intenta crear un mapa con menores medidas que el mínimo necesario para cumplir las consignas del Trabajo Práctico.

UnidadFueDestruidaException se lanza cuando una unidad muere. Sirve para avisarle al Jugador atacado que ya no posee dicha unidad y debe ser removida del mapa.

UnidadNoPuedeSerAtacadaPorArmaDeAsedioException se lanza cuando un arma de asedio intenta atacar a unidad.

UnidadObjetivoEsPropiaException se lanza cuando una unidad intenta atacar a otra unidad propia del jugador.

UnidadSeleccionadaNoPerteneceAJugadorException se lanza cuando una jugador intenta realizar una acción (atacar, construir, reparar o moverse) con una unidad que no le pertenece.

UnidadYaFueUtilizadaEnEsteTurnoException se lanza cuando el jugador intenta realizar una acción con una unidad que ya fue utilizada en este turno.