

Chess 2

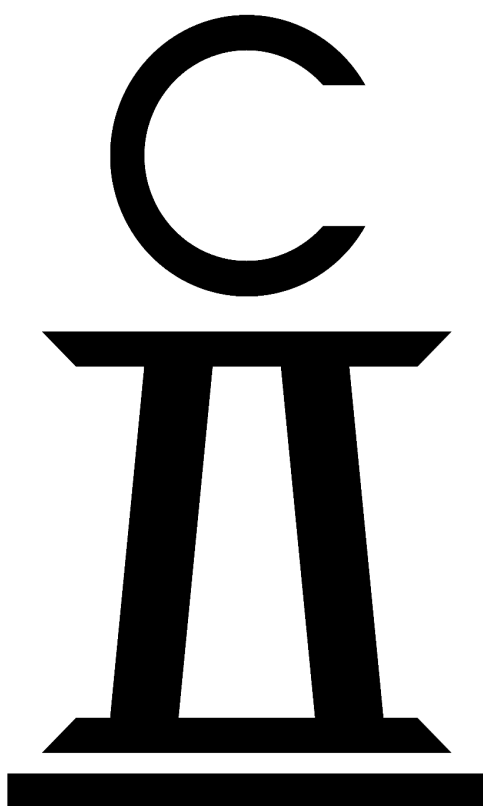


Tabla de contenidos:

Integrantes del equipo

Introducción

Distribución de roles

Descripción de roles

Planificación de riesgos

Planificación

Requisitos

Casos de Uso

Modelo de Dominio

Diagramas de secuencia

Pruebas JUnit

Software usado

Integrantes del equipo:

- | | |
|--------------------------------------|---------------------------------|
| • Víctor Pérez Armenta | vparmenta@uma.es |
| • David Bueno Carmona | davidbuenocarmona@uma.es |
| • Adrián Torremocha Doblas | adriantorremocha@uma.es |
| • Pablo Márquez Benítez | usoftpablomarquezbenitez@uma.es |
| • Jaime Ezequiel Rodriguez Rodriguez | jaezro@uma.es |
| • José Miguel Prieto Páez | jmprietopaez@alu.uma.es |
| • Ezequiel Sánchez García | ezequielsanchezgarcia@uma.es |
| • Gregorio Merchán Merchán | gregomerchan@uma.es |
| • Pablo Miguel Aguilar Blanco | pablo__972@uma.es |
| • José Ángel Bueno Ruiz | jbueno3830@uma.es |

Repositorio de GitHub: <https://github.com/VctPerez/Chess2-UMA>

Introducción:

Desarrollamos nuestra aplicación con el propósito de entretener a nuestros clientes. Nuestro objetivo es crear una versión modificada del clásico juego del ajedrez ofreciendo partidas dinámicas e interactivas con variaciones en el juego. Sobre todo, aprovechamos el reciente incremento de popularidad del ajedrez, ya que cada vez más personas se suman a jugarlo, ya sea de forma casual o profesional.

Distribución de roles:

- **Game Designer:** José Ángel Bueno Ruiz, Pablo Márquez Benítez, David Bueno Carmona
- **Scrum Master:** David Bueno Carmona, Adrian Torremocha Doblas.
- **Product owner:** Pablo Aguilar Blanco, Ezequiel Sánchez García.
- **Analista:** Ezequiel Sánchez García, Jaime Ezequiel Rodriguez Rodriguez.
- **Diseñador:** Adrian Torremocha Doblas, Jaime Ezequiel Rodriguez Rodriguez.
- **Programador:** José Ángel Bueno Ruiz, Víctor Pérez Armenta, Pablo Márquez Benítez.
- **UI / UX Designer:** José Miguel Prieto Páez, Gregorio Merchan Merchan.
- **Tester:** Víctor Pérez Armenta, Gregorio Merchan Merchan.
- **Modelador:** José Miguel Prieto Páez, Pablo Aguilar Blanco.

Descripción de roles:

- **Game Designer:** Se centra en el diseño de mecánicas, ideas para el juego y en hacer el juego lo más interesante posible.
- **Scrum Master:** Hace de coach del equipo y organiza las reuniones.
- **Product Owner:** Aporta la visión de proyecto al equipo de trabajo poniendo una dirección en común en la que trabajar.
- **Analista:** Analiza los riesgos que puedan afectar al proyecto y diseña planes de contingencia.
- **Diseñador:** Encargado del apartado artístico y sonoro. Además, es responsable de la cohesión estética del proyecto en su totalidad.
- **Programador:** Encargados de implementar en código y hacer tangibles las ideas de diseño del juego propuesta por los compañeros.
- **UI / UX Designer:** Diseña interfaces gráficas intuitivas y visualmente atractivas.
- **Tester:** Se encarga de planificar y llevar a cabo pruebas de software para comprobar que su funcionamiento sea el esperado. Detecta errores, comunica y actúa en consecuencia.
- **Modelador:** Encargado de diseñar los modelos del código que será implementado en el proyecto.

Planificación de riesgos:

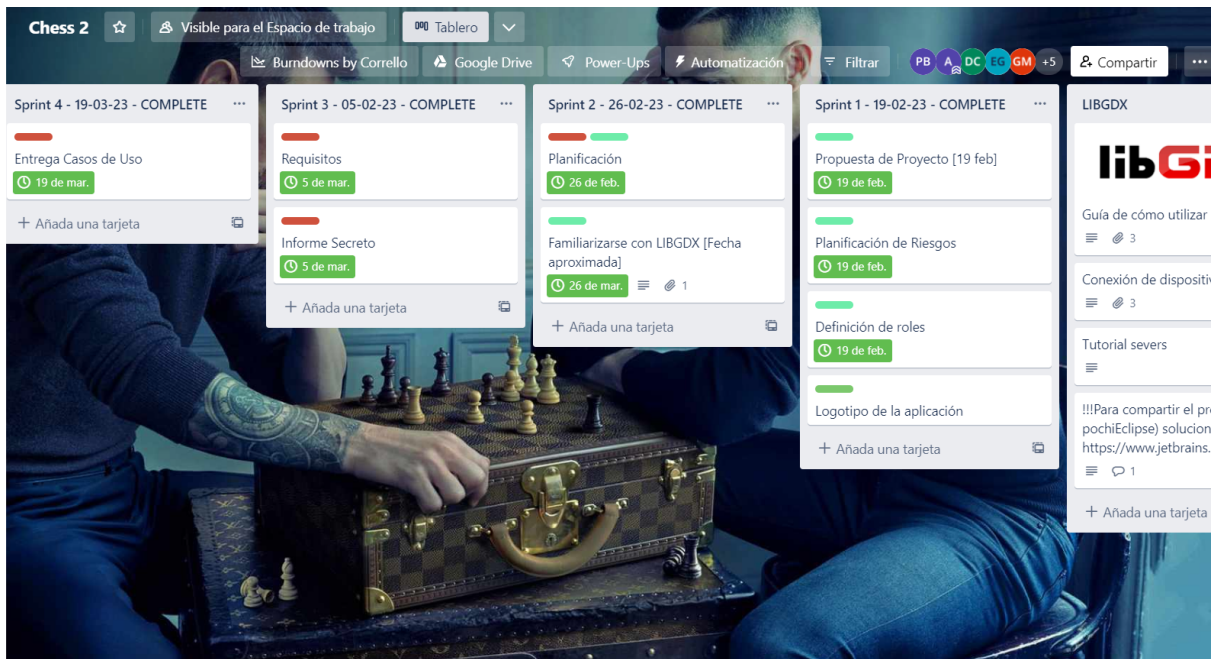
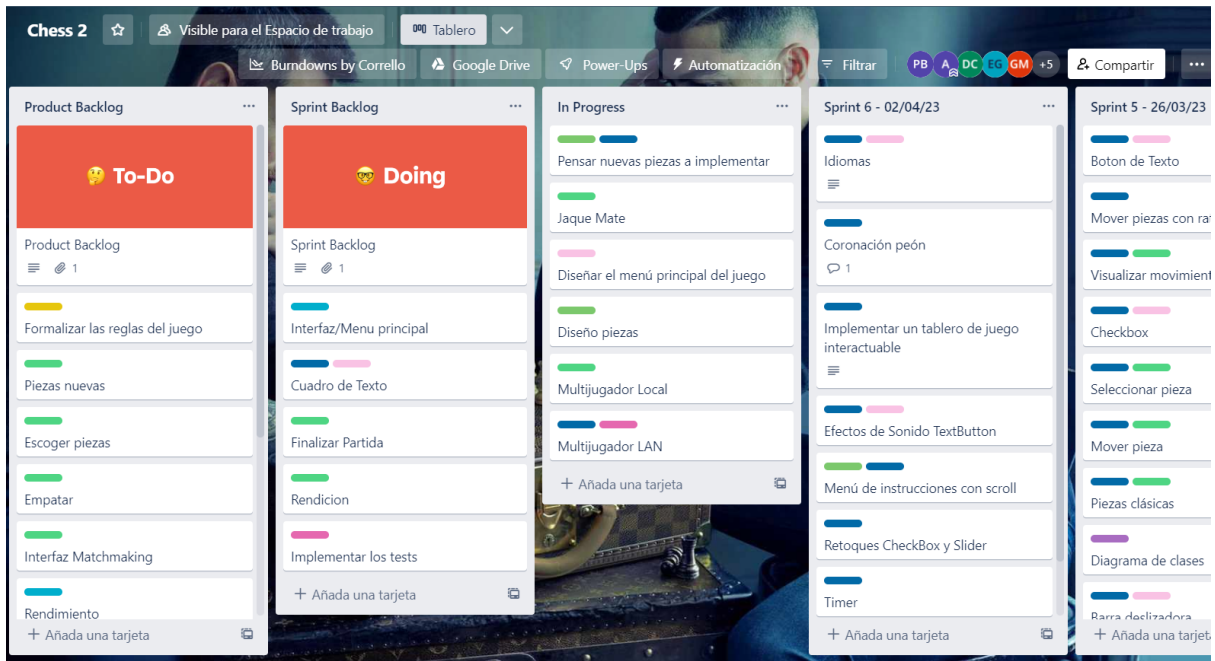
Riesgo	Tipo	Descripción	Probabilidad	Efectos	Estrategia
Problemas con las librerías.	Producto	Incompatibilidad entre las librerías deseadas o que sean de pago.	Moderada	Tolerable: Puede llevar a tener que remodelar la idea del código a otras menos deseables.	Investigar previamente el repertorio de librerías que realmente podemos usar y que nos sean más útiles.
Pérdidas en el progreso.	Proyecto	No realizar un guardado del proyecto.	Baja	Serio: Perdemos una cantidad considerable de progreso	Recurrir a github y hacer copias de seguridad
Subestimación del aprendizaje.	Proyecto	Sobreestimar nuestras capacidades para adquirir los conocimientos necesarios	Alta	Serio: Podría retrasar indefinidamente la planificación del proyecto.	Utilizar recursos de aprendizaje útiles y organizados.
Problemas de organización y planificación.	Proyecto	No planificar adecuadamente las sesiones de trabajos sin tener en cuenta exámenes o diferentes necesidades y mala organización del desarrollo del proyecto.	Alta	Serio: Puede llevar a tener que realizar trabajo bajo presión debido a tener que reorganizarse y por tiempo	Cumplir con la organización propuesta en Trello

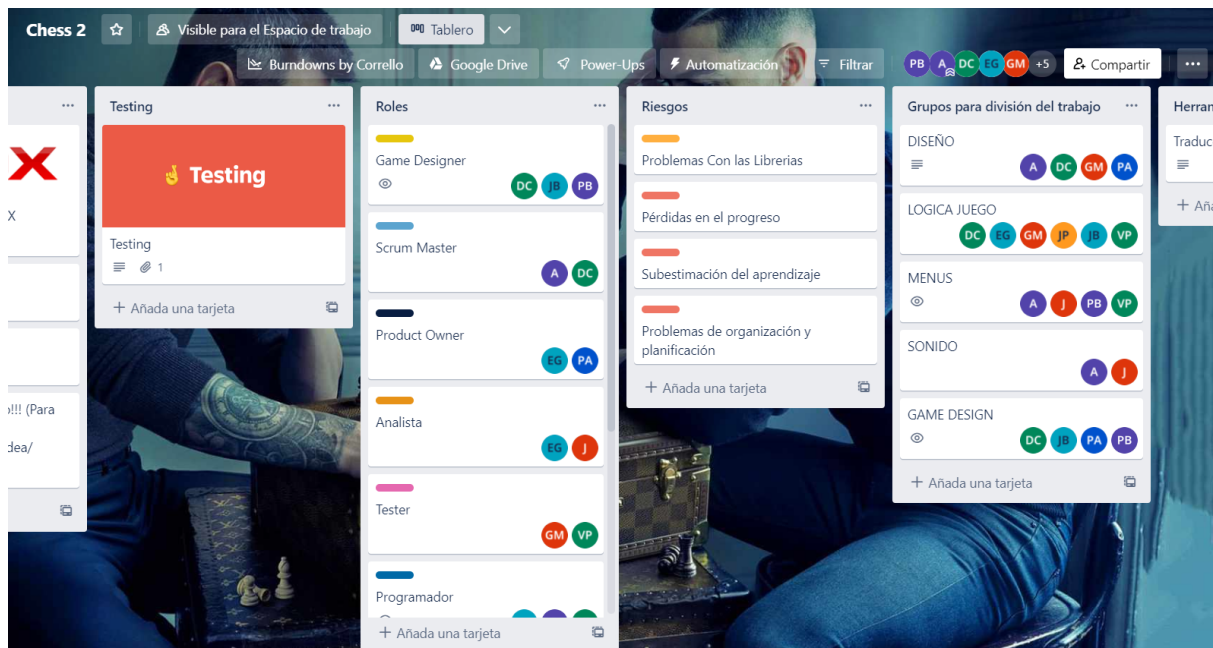
Planificación:

El modelo elegido ha sido una estrategia ágil por varios motivos, el principal es la flexibilidad que nos permite definir las tareas para cada sprint sobre la marcha (el sprint es semanal).

Trello es una herramienta realmente útil, pues nos permite organizar las tareas en el *Product Backlog* e ir realizandolas a conveniencia en cada *Sprint* semanal, es decir, en el *Sprint Backlog*.

Además, mediante *Sprints* mantenemos el registro de todos los avances en el proyecto semana a semana.





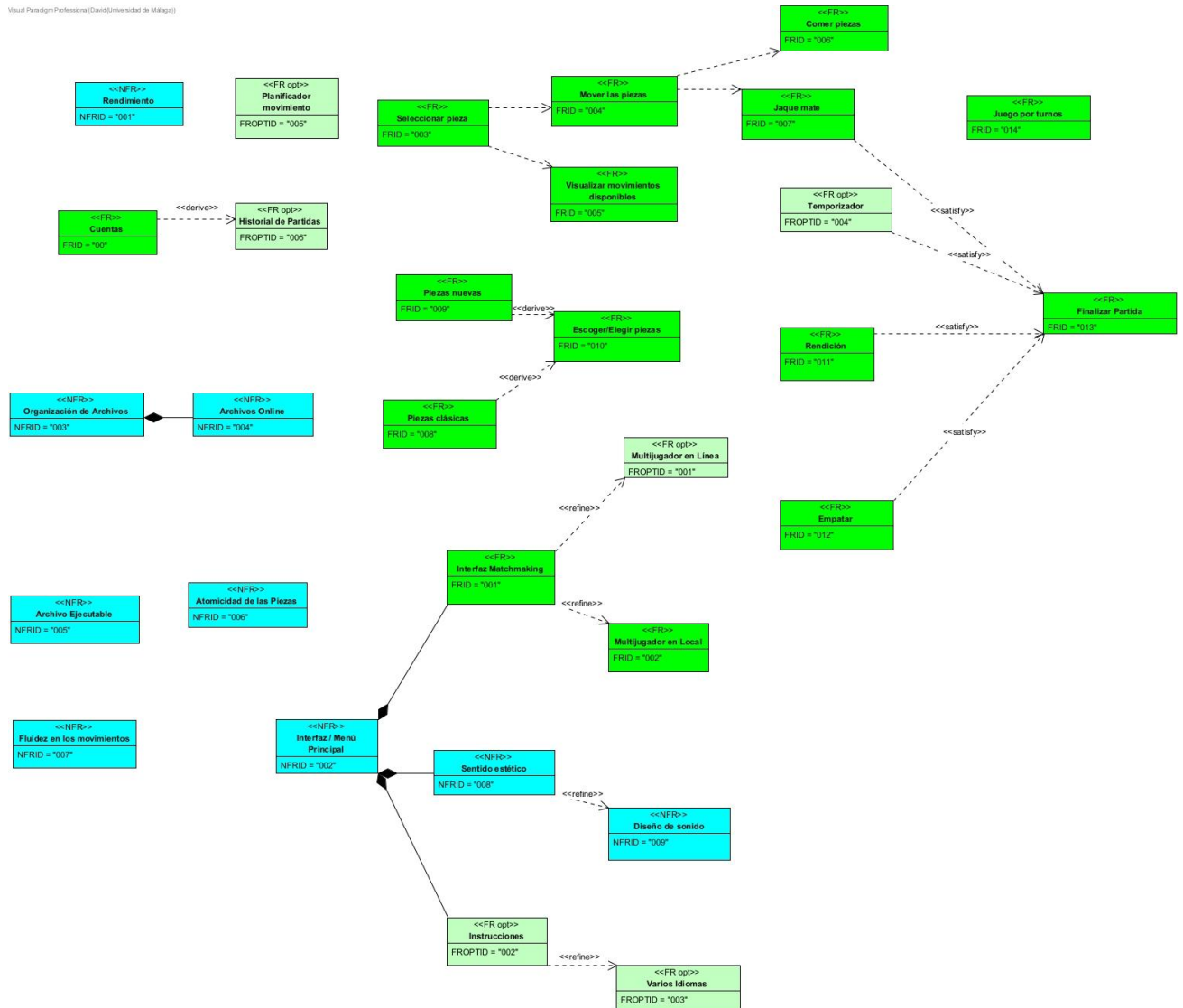
Dentro de Trello organizamos en diferentes listas los *Sprint Backlogs*, el *Product Backlog* e información de utilidad. Dentro del *product backlog* planteamos las tareas a realizar a largo plazo. En los *Sprint Backlogs* planteamos los objetivos y requisitos necesarios a completar para cuando llegue el final del sprint.

Además dentro de Trello tenemos listas en las que agrupamos información relevante de cara al proyecto, por ejemplo, en la lista LIBGDX hay recursos para aprender a utilizar libgdx, que es algo interesante para todo el grupo. Cualquier recurso que pueda ser útil a la hora de aprender sobre el entorno y desarrollar el proyecto quedará reflejado en las listas, de forma que todos tengamos acceso a esa información.

Las tareas tienen asignados los roles que son relevantes, la leyenda de los colores de roles está en la lista roles.

Requisitos:

Visual Paradigm Professional (David Universidad de Málaga)



- **Funcionales:**

RF001: Interfaz Matchmaking

Como cliente,
quiero un sistema y apartado en el juego dedicado al matchmaking
para poder conectarme a una partida con el rival deseado ya sea a nivel
local o multijugador

RF002: Multijugador Local

Como programador, modelador y tester,
quiero multijugador local que permita jugar partidas en una red local
para poder jugar partidas más interactivas con un jugador cara a cara.

RF003: Seleccionar pieza

Como cliente,
quiero poder seleccionar una pieza
para poder mover o visualizar un movimiento

Pruebas de aceptación

- El jugador sólo puede seleccionar piezas en su turno
- El jugador sólo puede interactuar con las piezas de su propio color.
- Una vez seleccionada una pieza un jugador puede visualizar las casillas disponibles.
- El jugador puede optar por mover la pieza seleccionada o no hacerlo y escoger otra.

RF004: Mover las piezas

Como usuario,
quiero poder mover las piezas a las casillas disponibles
para que se pueda jugar

Pruebas de aceptación

- Las piezas podrán ser movidas por el tablero.
- Cada pieza tendrá un movimiento característico.
- El rey no podrá ser movido a una casilla donde esté en peligro.
- Mover encima de una pieza rival provocará "Comer Piezas".

RF005: Visualizar movimientos disponibles

Como cliente,
quiero poder visualizar las casillas a las que puedo mover una pieza
para planear mis movimientos

Pruebas de aceptación

- Si un jugador selecciona una pieza, se resaltan en el tablero las casillas disponibles para moverlas.
- Un jugador sólo puede visualizar los movimientos de las piezas de su color.
- Si selecciona otra pieza de su color, se reajustan los movimientos disponibles para visualizar movimientos de la última pieza seleccionada.

RF006: Comer piezas

Como cliente,
quiero poder eliminar las piezas del rival
para avanzar en la partida

Pruebas de aceptación

- Solo se pueden comer piezas del rival, no de tu propio color.
- Se comerá una pieza si colocas otra en su misma casilla.
- Al comerse una pieza, se sumará el valor de esta a la puntuación del jugador que ha comido.

RF007: Jaque mate

Como cliente,
quiero poder hacer jaque mate
para ganar la partida.

Pruebas de aceptación

- El jaque mate determinará quién gana la partida y quien la pierde.
- El jaque mate ocurrirá exclusivamente cuando el rey sea amenazado y no se pueda hacer nada para evitarlo en 1 ronda.
- Para que el jaque mate sea exitoso se tendrá que decir "Jaque mate con Tomate".

RF008: Piezas clásicas

Como cliente,
quiero tener las piezas clásicas del ajedrez
para poder jugar de forma tradicional.

RF009: Piezas nuevas

Como cliente,
quiero tener nuevas piezas con nuevas funciones
para vivir nuevas experiencias de juego.

RF010: Escoger piezas

Como cliente,
quiero poder escoger el tipo de pieza antes de comenzar la partida
para tener las piezas antes de empezar a jugar.

RF011: Rendición

Como jugador,
quiero tener la opción de rendirme
para poder parar la partida en caso de que por cualquier motivo no pueda seguir jugándose.

Pruebas de aceptación

- La rendición será voluntaria.
- La rendición provocará una finalización inmediata de la partida con un ganador y un perdedor.
- La rendición cederá la victoria al rival.

RF012: Empatar

Como cliente,
quiero poder empatar la partida con mi rival si se cumplen las condiciones o si los dos queremos
para finalizar la partida.

Pruebas de aceptación

- El empate finaliza la partida sin ganador ni perdedor.
- El empate puede ser provocado por un acuerdo mutuo.
- El empate puede ser provocado por un ahogamiento del rey (este está a salvo, pero no puede moverse ya que estaría en peligro).

RF013: Finalizar partida

Como cliente,
quiero poder ganar, perder o empatar
para acabar la partida.

RF014: Juego por turnos

Como cliente,
quiero poder realizar acciones por turno
para mantener las reglas básicas del ajedrez.

- **No funcionales:**

RNF001: Rendimiento

Como cliente,
quiero que el juego cumpla unos estándares de rendimiento y pulidez a nivel de errores y diseño
para que la experiencia de juego no se vea nunca entorpecida por errores propios del software o diseño.

RNF002: Interfaz / Menú Principal

Como diseñador,
quiero que el juego cuente con una interfaz agradable e intuitiva
para que conecte el menú principal con todos los apartados accesibles del software para el usuario.

RNF003: Organización de Archivos

Como product owner, analista y Scrum master,
quiero que el proyecto este adecuadamente organizado en directorios y los archivos se identifiquen adecuadamente por su nombre
para una mayor organización del proyecto y una mayor facilidad a la hora de trabajar en él.

RNF004: Archivos Online

Como product owner, analista y Scrum master,
quiero que todos esos archivos estén subidos en un repositorio online
para que todos los miembros del equipo tengan fácil acceso a estos archivos y para poder tener todas las versiones de estos almacenadas.

RNF005: Archivo Ejecutable

Como desarrollador,
quiero que el juego se pueda abrir desde un archivo ejecutable
para que sea accesible.

RNF006: Atomicidad de las Piezas

Como programador y modelador,
quiero que cada pieza funcione adecuadamente de manera individual y sin que modificaciones en su comportamiento afecten a las demás
para poder seguir desarrollando piezas y funcionalidades de cara al futuro.

RNF007: Fluidez en los movimientos

Como programador,
quiero que cuando un jugador realice una acción se refleje en la partida
para que la experiencia sea fluida.

RNF008: Sentido estético

Como diseñador,
quiero que el juego mantenga una estética coherente y agradable para el jugador
para llegar a un juego más atractivo visualmente.

RNF009: Diseño de sonido

Como diseñador,
quiero un diseño de sonido adecuado y agradable
para una experiencia más inmersiva y atraer al jugador a la partida.

- **Funcionales Opcionales:**

RFO001: Multijugador Online

Como programador, modelador,
quiero un multijugador en línea que permita jugar partidas en línea vía internet
para poder jugar partidas con jugadores a larga distancia.

RFO002: Instrucciones

Como cliente,
quiero que el juego cuente con un apartado de instrucciones
para poder entender de forma concisa todas las reglas y características de este.

RFO003: Varios Idiomas

Como director,
quiero que el juego incluya el inglés
para una mayor accesibilidad.

RFO004: Temporizador

Como cliente,
quiero poder ver el tiempo que me queda de turno
para planear mis movimientos

RFO005: Planificador de Movimientos

Como jugador,
quiero poder marcar casillas y dibujar flechas a casillas
para planificar mis movimientos con más facilidad.

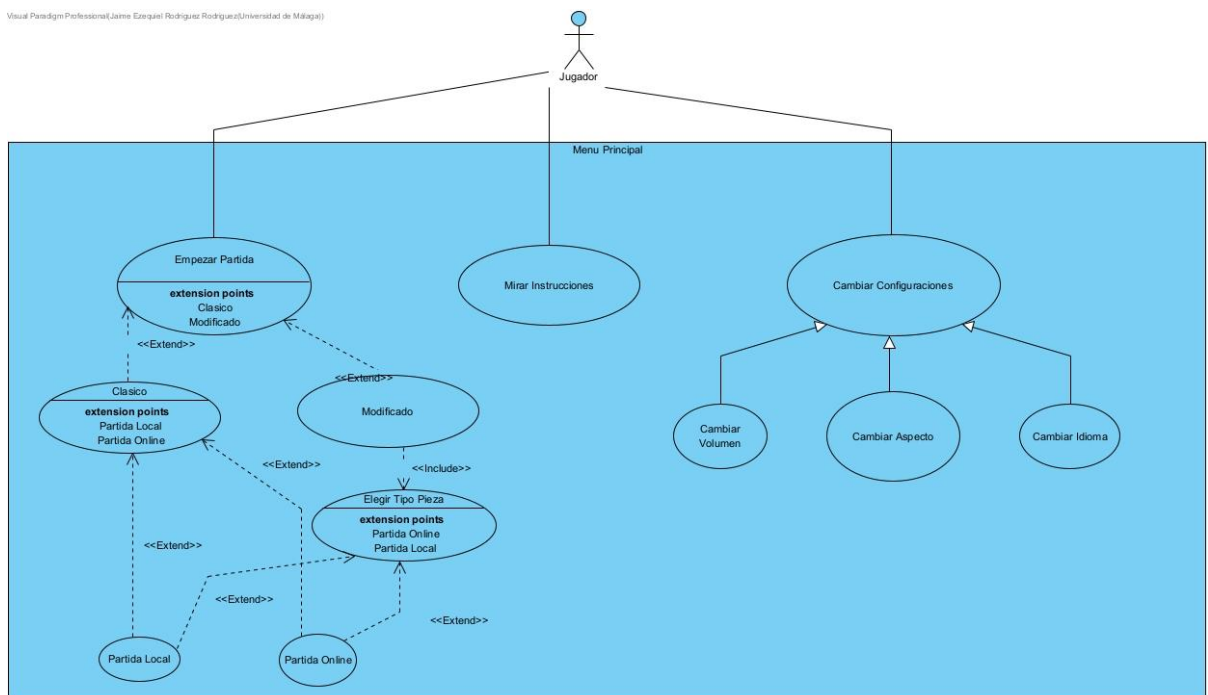
RFO006: Historial de Partidas

Como cliente,
quiero que en la pantalla de inicio haya una opción de historial de partidas
para que el jugador pueda ver sus rivales y resultados de sus anteriores partidas.

Casos de Uso:

- **PrePartida:**

Visual Paradigm Professional (Jaime Ezequiel Rodríguez Rodríguez (Universidad de Málaga))



Identificador único

· CU1 - Empezar Partida

Contexto de uso

· El jugador puede empezar una partida cuando está en el menú principal

Precondiciones y activación

· El jugador debe estar en la pantalla inicial

Garantías de éxito/Postcondición

· El jugador inicia una partida online o local

Escenario principal

1. El jugador selecciona "Empezar Partida"
2. El jugador elige modo de juego
3. El jugador elige tipo de partida
4. La partida comienza

<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador selecciona “Volver al menú principal” El juego vuelve a la pantalla inicial
--

<p>Identificador único</p> <ul style="list-style-type: none"> · CU2 - Mirar Instrucciones
<p>Contexto de uso</p> <ul style="list-style-type: none"> · El jugador puede mirar las instrucciones del juego para aprender a jugar y empezar un tutorial
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en la pantalla inicial
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador puede ver el menú de instrucciones
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. El jugador selecciona “Instrucciones” en el menú principal 2. El jugador puede decidir si ve las reglas del juego o inicia un tutorial
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador selecciona “Volver al menú principal” para volver a la pantalla inicial

<p>Identificador único</p> <ul style="list-style-type: none"> · CU3 - Cambiar Aspecto
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador esté en el menú de opciones, podrá cambiar el aspecto del juego

<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en el menú de opciones · El jugador debe pulsar el botón “Cambiar Aspecto”
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador abre el menú de aspecto, pudiendo cambiar diversos valores
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. El jugador está en la pantalla de opciones 2. El jugador pulsa “Configuración de Aspecto” 3. El jugador puede cambiar tablero, fichas y activar el modo daltónico
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · Regresar al menú de Opciones

<p>Identificador único</p> <ul style="list-style-type: none"> · CU4 - Cambiar configuraciones
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador esté en el menú, podrá cambiar configuraciones
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en el menú · El jugador debe pulsar el botón opciones
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador abre el menú de opciones pudiendo cambiar diversos valores
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. El jugador está en el menú 2. El jugador pulsa “Opciones” 3. El jugador puede cambiar volumen, aspecto e idioma

<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador vuelve al menú principal · El jugador cierra la aplicación
--

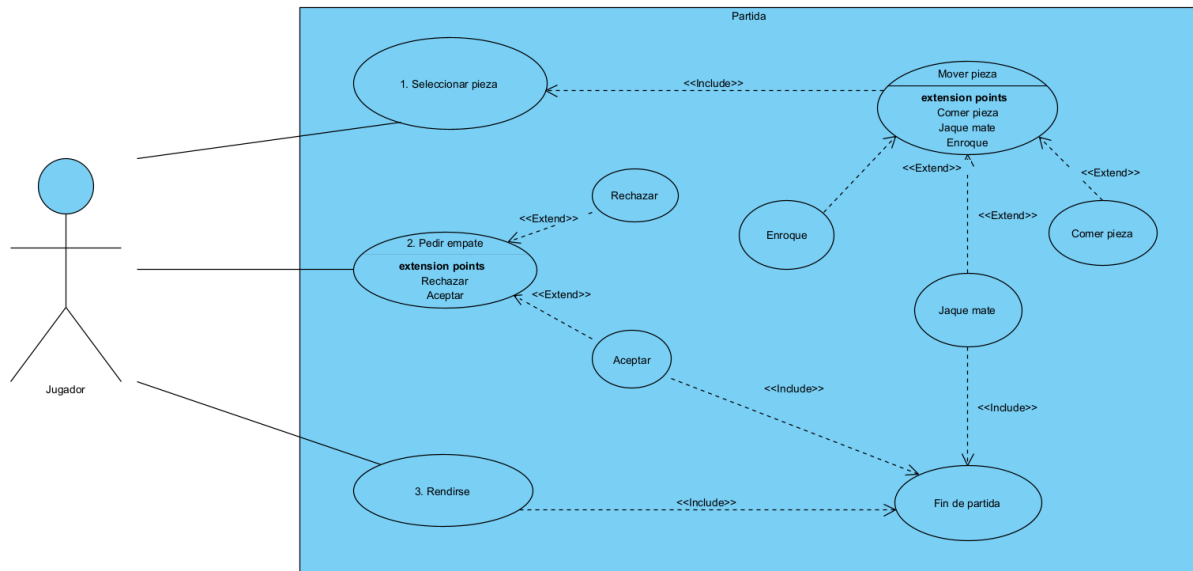
<p>Identificador único</p> <ul style="list-style-type: none"> · CU5 - Clásico
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador esté en el menú de empezar partida, puede elegir el modo clásico.
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en el menú de empezar partida · El jugador debe pulsar el botón de modo clásico
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador puede empezar una partida clásica en modo local/online
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. El jugador está en el menú de empezar partida 2. El jugador pulsa "Modo clásico" 3. El jugador puede elegir entre online o local
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador puede volver al menú de opciones.

<p>Identificador único</p> <ul style="list-style-type: none"> · CU6 - Modificado
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador esté en el menú de empezar partida, puede elegir el modo modificado.

<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en el menú de empezar partida · El jugador debe pulsar el botón de modo modificado
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador accede al menú de elegir tipo de pieza/draft.
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. El jugador está en el menú de empezar partida 2. El jugador pulsa "Modo modificado" 3. El jugador puede elegir entre online o local
<p>Escenarios alternativos</p> <p>.El jugador puede volver al menú principal</p>

<p>Identificador único</p> <ul style="list-style-type: none"> · CU7 - Elegir Tipo pieza
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador selecciona jugar al modo modificado, puede elegir las piezas de juego
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe haber seleccionado · El jugador debe pulsar el botón de modo clásico
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador puede empezar una partida clásica en modo local/online
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. El jugador está en el menú de empezar partida 2. El jugador pulsa "Modo clásico" 3. El jugador puede elegir entre online o local

- **Partida:**



Identificador único <ul style="list-style-type: none"> · CU8 - Mover Pieza
Contexto de uso <ul style="list-style-type: none"> · El jugador puede mover la pieza seleccionada a una posición válida del tablero
Precondiciones y activación <ul style="list-style-type: none"> · El jugador debe haber seleccionado una pieza válida
Garantías de éxito/Postcondición <ul style="list-style-type: none"> · El jugador puede mover una pieza · El jugador puede comer otra pieza · El jugador puede hacer enroque · El jugador puede hacer jaque mate
Escenario principal <ol style="list-style-type: none"> 1. Un jugador selecciona una pieza 2. Si el movimiento es válido, se mueve la pieza 3. Puede comerse una pieza, hacer enroque, jaque mate o no hacer nada

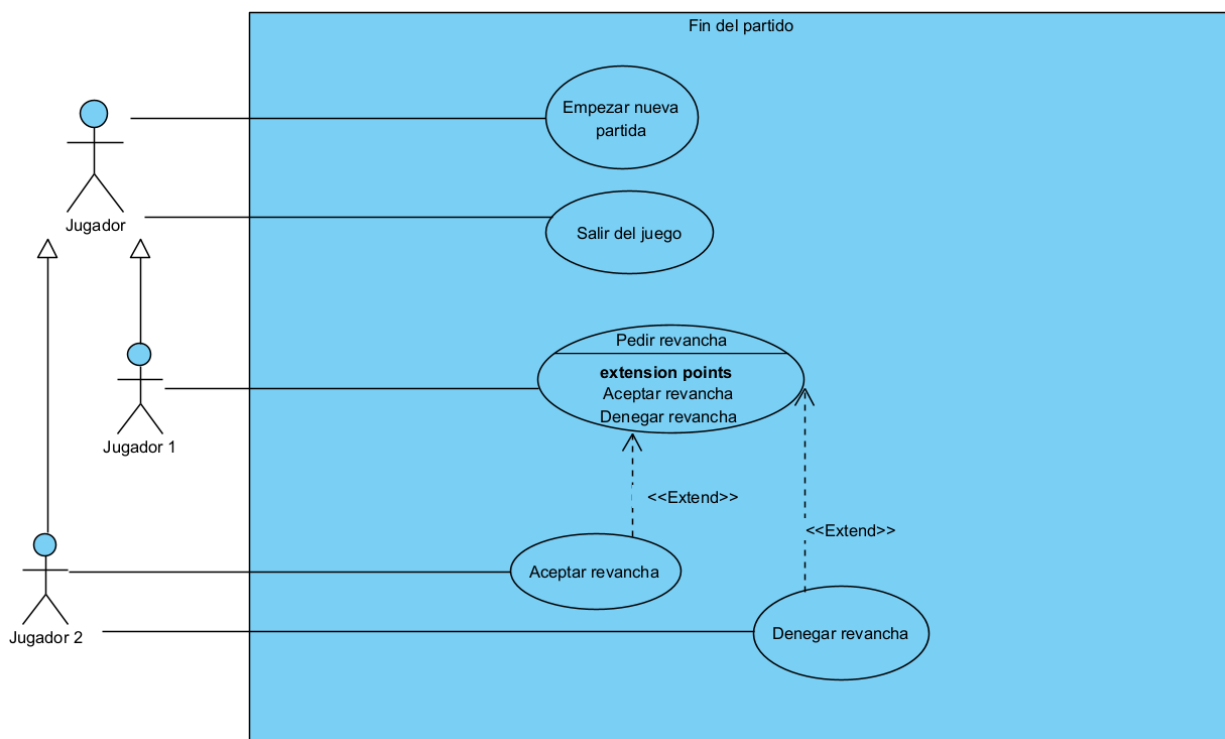
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador cancela la selección de la pieza

<p>Identificador único</p> <ul style="list-style-type: none"> · CU9 - Pedir Empate
<p>Contexto de uso</p> <ul style="list-style-type: none"> · El jugador puede solicitar un empate a su oponente
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · Debe estar en una partida
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El oponente acepta el empate · El oponente rechaza el empate
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. El jugador pulsa empate 2. El otro jugador puede aceptarlo o rechazarlo 3. Se termina la partida
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador cancela la petición de revancha

<p>Identificador único</p> <ul style="list-style-type: none"> · CU10 - Fin de Partida
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Se acaba la partida (Diagrama de usa "Final de Partida")

<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · Se hace un jaque mate · Un jugador solicita empate y el otro acepta · Un jugador se rinde
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · Se acaba la partida y se pasa al diagrama de “Final de Partida”
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. Se ha producido alguna de las precondiciones 2. Se termina la partida y se pasa al diagrama “Fin de Partida”
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · No hay

● **Final de Partida:**



<p>Identificador único</p> <ul style="list-style-type: none"> · CU11 - Empezar nueva partida
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador termine una partida, podrá empezar una nueva
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en una partida · La partida debe haber finalizado
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador inicia una partida exitosamente
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. La partida finaliza 2. El jugador vuelve al menú principal 3. El jugador pulsa “Empezar nueva partida” 4. El jugador empieza una nueva partida
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador sale del juego

<p>Identificador único</p> <ul style="list-style-type: none"> · CU12 - Salir del juego
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador termine una partida, podrá salir del juego
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en una partida. · La partida debe haber finalizado.

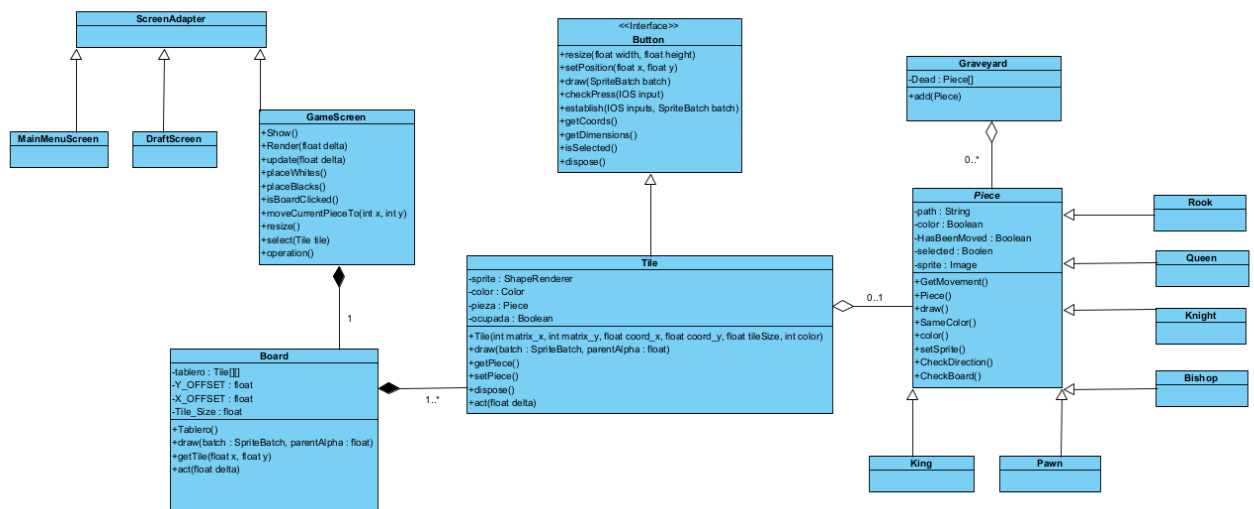
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador consigue salir de la aplicación
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. La partida finaliza 2. Un jugador vuelve al menú principal 3. El jugador pulsa “Salir del juego”
<p>Escenarios alternativos</p> <ul style="list-style-type: none"> · El jugador empieza una nueva partida

<p>Identificador único</p> <ul style="list-style-type: none"> · CU13 - Pedir revancha.
<p>Contexto de uso</p> <ul style="list-style-type: none"> · Cuando un jugador termine una partida, podrá solicitar revancha al contrincante.
<p>Precondiciones y activación</p> <ul style="list-style-type: none"> · El jugador debe estar en una partida. · La partida debe haber finalizado.
<p>Garantías de éxito/Postcondición</p> <ul style="list-style-type: none"> · El jugador envía su petición de revancha exitosamente.
<p>Escenario principal</p> <ol style="list-style-type: none"> 1. La partida finaliza 2. Un jugador pulsa “Pedir revancha” 3. El jugador espera la decisión del otro jugador

Escenarios alternativos

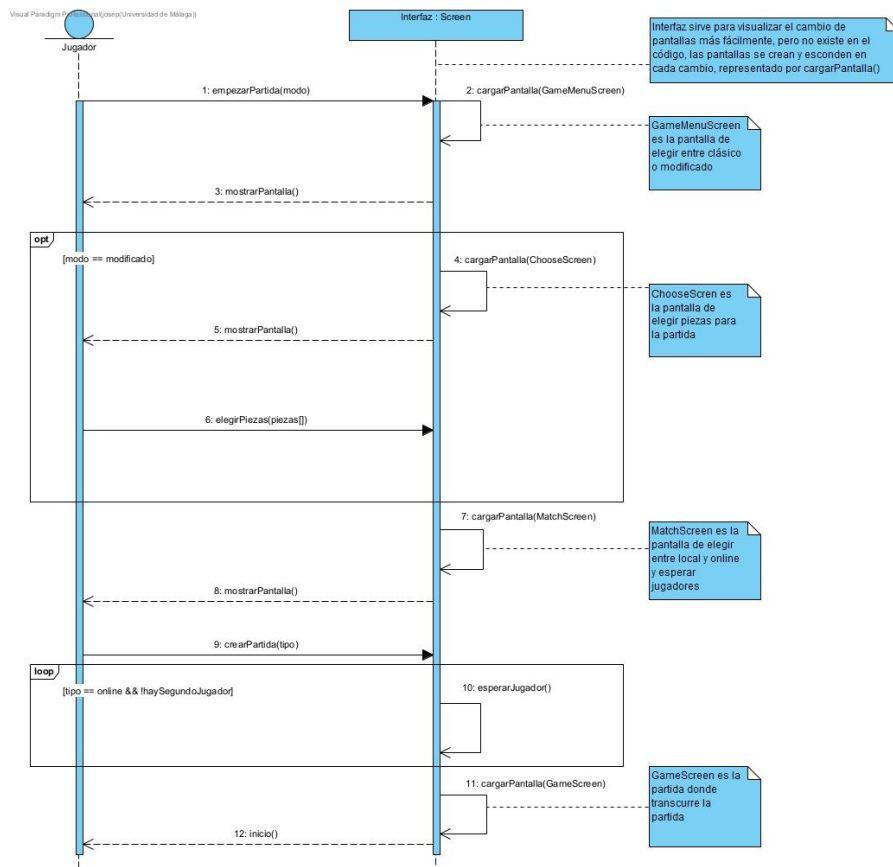
- El jugador vuelve al menú principal.
- El jugador sale del juego.

Modelo de Dominio:



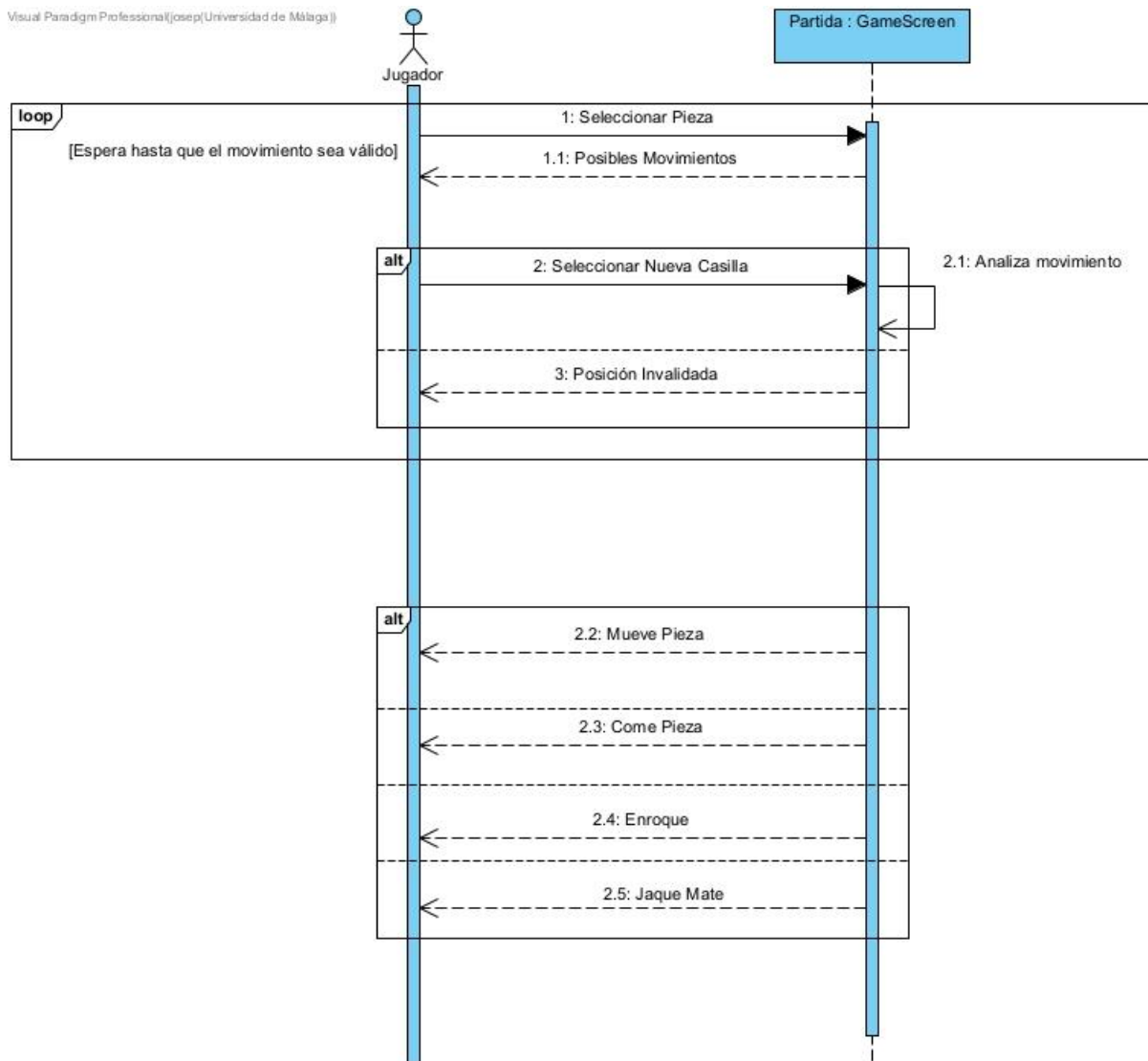
Diagramas de secuencia:

- Diagrama para empezar partida:



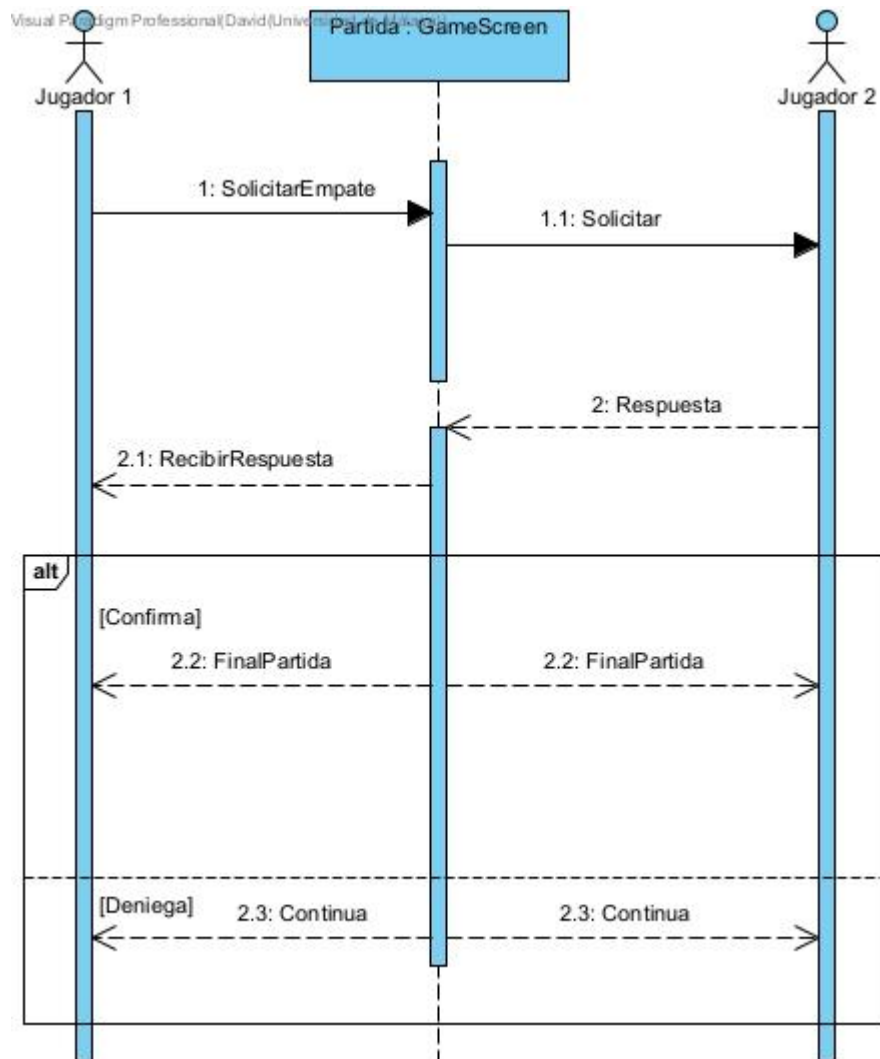
En el diagrama se indica la serie de pasos que el jugador puede tomar para crear una partida, puede elegir el modo y el tipo de partida, cada paso genera una pantalla nueva.

- **Diagrama para mover pieza:**



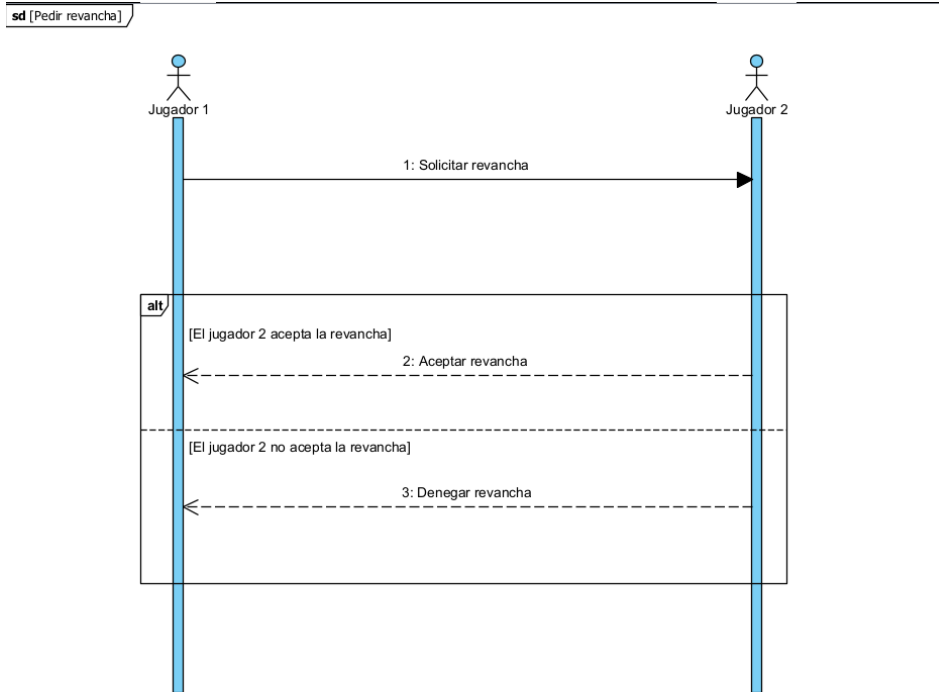
En este diagrama el Jugador selecciona una pieza y recibe como respuesta los posibles movimientos a realizar, después selecciona la nueva casilla destino de la pieza pudiendo recibir como respuesta que su nueva casilla es una posición inválida o que es una posición válida y realiza el movimiento.

- Diagrama para pedir empate:



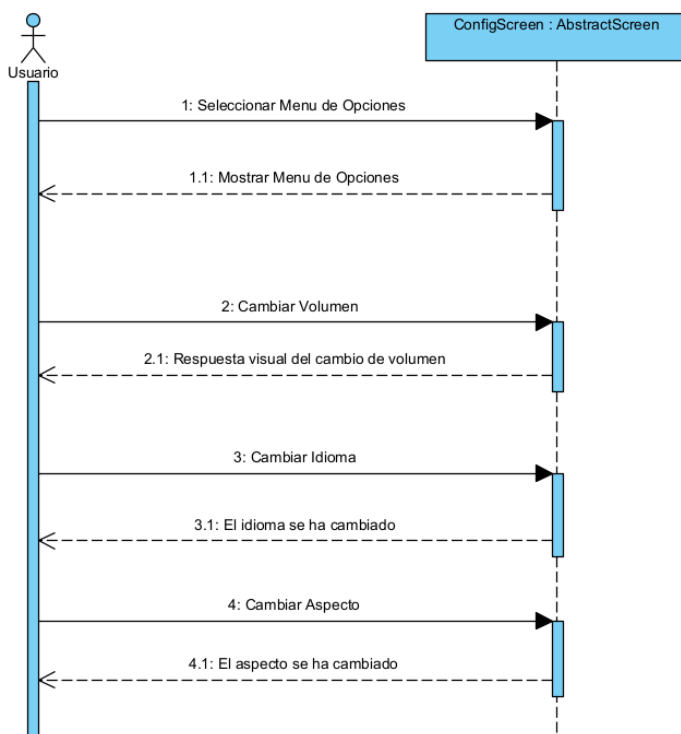
En este diagrama el Jugador 1 solicita terminar la partida y dejarla en empate al Jugador 2 y este último la acepta o la deniega recibiendo el Jugador 1 la respuesta.

- Diagrama para pedir revancha:



En este diagrama el Jugador 1 solicita revancha al Jugador 2 y este último la acepta o la deniega, recibiendo el Jugador 1 la respuesta.

- Diagrama del Cambio de Configuraciones:



En este diagrama el jugador puede acceder al menú de opciones desde el menú principal. Una vez dentro, puede cambiar el volumen, el idioma y el aspecto.

Pruebas JUnit:

Las pruebas se encuentran en el paquete test.java de core.

- **SettingsTest:**

```
package test.java;

import exceptions.SettingsException;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import utils.Settings;

import static org.junit.jupiter.api.Assertions.*;

public class SettingsTest {
    float sfx, music, language;

    @BeforeEach
    public void setUp(){
        sfx = 50;
        music = 50;
        language = 0;
    }

    @Test
    public void turnUpEffects(){
        Settings.setSfxVolume(sfx+1);
        assertEquals(sfx+1,
Settings.reconvertAudioToText(Settings.sfxVolume), " El volumen de efectos
deberia haber subido.");
    }

    @Test
    public void turnDownEffects(){
        Settings.setSfxVolume(sfx-1);
        assertEquals(sfx-1,
Settings.reconvertAudioToText(Settings.sfxVolume), " El volumen de efectos
deberia haber bajado.");
    }

    @Test
    public void turnUpMusic(){
        Settings.setMusicVolume(music + 1);
        assertEquals(music + 1,
Settings.reconvertAudioToText(Settings.musicVolume), "El volumen de la
musica deberia haber subido.");
    }
}
```

```

    }

    @Test
    public void turnDownMusic(){
        Settings.setMusicVolume(sfx-1);
        assertEquals(sfx-1,
Settings.reconvertAudioToText(Settings.musicVolume), " El volumen de la
musica deberia haber bajado.");
    }

    @Test
    public void changeLanguage(){
        Settings.setLanguage((int) (language+1)%2);
        assertNotEquals(language, Settings.language, "El idioma deberia ser
distinto");
    }

    @Test
    public void sfxException(){
        Exception exception = assertThrows(SettingsException.class, () ->
Settings.setSfxVolume(-1));
        assertEquals("El volumen que desea establecer no esta permitido.",
exception.getMessage());
        Exception exception2 = assertThrows(SettingsException.class, () ->
Settings.setSfxVolume(105));
        assertEquals("El volumen que desea establecer no esta permitido.",
exception2.getMessage());
    }

    @Test
    public void musicException(){
        Exception exception = assertThrows(SettingsException.class, () ->
Settings.setMusicVolume(-1f));
        assertEquals("El volumen que desea establecer no esta permitido.",
exception.getMessage());
        Exception exception2 = assertThrows(SettingsException.class, () ->
Settings.setMusicVolume(150f));
        assertEquals("El volumen que desea establecer no esta permitido.",
exception2.getMessage());
    }

    @Test
    public void languageException(){
        Exception exception = assertThrows(SettingsException.class, () ->
Settings.setLanguage(-1));
        assertEquals("El lenguaje seleccionado no esta disponible.",
exception.getMessage());
        Exception exception2 = assertThrows(SettingsException.class, () ->
Settings.setLanguage(2));
        assertEquals("El lenguaje seleccionado no esta disponible.",
exception2.getMessage());
    }

```

```
}
```

- **PieceTest:**

```
package test.java;

import com.badlogic.gdx.graphics.Texture;
import elements.Board;
import elements.Piece;
import elements.Tile;
import elements.pieces.Rook;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import static org.junit.Assert.assertNull;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.Mockito.*;

public class PieceTest {
    Piece pieza;

    @BeforeEach
    public void init() {
        pieza = mock(Piece.class, Mockito.CALLS_REAL_METHODS);
    }

    @AfterEach
    public void terminate() {
        pieza = null;
    }

    @Test
    public void colorTest(){
        Tile tileMock = mock(Tile.class);
        Board boardMock = mock(Board.class);
        when(boardMock.getTile(0, 0)).thenReturn(tileMock);
        when(tileMock.getX()).thenReturn(0f);
        when(tileMock.getY()).thenReturn(0f);

        pieza = new Rook(true,0,0,boardMock, mock(Texture.class));

        assertEquals(pieza.color(),true,"Si se construye una pieza que deriva
de Piece se describe el atributo color correctamente");
    }

    @Test
    public void SameColor() {
        Tile tileMock = mock(Tile.class);
        Board boardMock = mock(Board.class);
```



```

        when(boardMock.getTile(0, 0)).thenReturn(tileMock);
        when(boardMock.getTile(0, 0)).thenReturn(tileMock);
        when(tileMock.getX()).thenReturn(0f);
        when(tileMock.getY()).thenReturn(0f);

        Piece auxT = new Rook(true,0,0,boardMock,mock(Texture.class));
        Piece auxF = new Rook(false,0,0,boardMock,mock(Texture.class));
        pieza = new Rook(true,0,0,boardMock,mock(Texture.class));

        assertEquals(pieza.sameColor(auxT),true);
        assertEquals(pieza.sameColor(auxF),false);
    }

    @Test
    public void correctDispose() {
        Tile tileMock = mock(Tile.class);
        Board boardMock = mock(Board.class);

        when(boardMock.getTile(0, 0)).thenReturn(tileMock);
        when(boardMock.getTile(0, 0)).thenReturn(tileMock);
        when(tileMock.getX()).thenReturn(0f);
        when(tileMock.getY()).thenReturn(0f);

        pieza = new Rook(true,0,0,boardMock,mock(Texture.class));
        pieza.dispose();
        assertNull(pieza.getImage());
    }

    @Test
    public void String() {
        Tile tileMock = mock(Tile.class);
        Board boardMock = mock(Board.class);
        when(boardMock.getTile(1, 1)).thenReturn(tileMock);
        when(tileMock.getX()).thenReturn(0f);
        when(tileMock.getY()).thenReturn(0f);

        pieza = new Rook(true,1,1,boardMock, mock(Texture.class));

        assertEquals(pieza.toString().equalsIgnoreCase("{Rook,(1,1)}"),true,"La salida en
        consola de la posicion de la pieza es correcta");
    }

    @Test
    public void aliveTest()
    {
        Tile tileMock = mock(Tile.class);
        Board boardMock = mock(Board.class);
        when(boardMock.getTile(0, 0)).thenReturn(tileMock);
        when(tileMock.getX()).thenReturn(0f);
        when(tileMock.getY()).thenReturn(0f);
    }

```

```

        pieza = new Rook(true,0,0,boardMock, mock(Texture.class));

        assertEquals(pieza.alive,true,"En un principio debe estar viva");
    }

    @Test
    public void equals() {
        Tile tileMock = mock(Tile.class);
        Board boardMock = mock(Board.class);
        when(boardMock.getTile(1, 1)).thenReturn(tileMock);
        when(tileMock.getX()).thenReturn(0f);
        when(tileMock.getY()).thenReturn(0f);

        pieza = new Rook(true,1,1,boardMock, mock(Texture.class));
        Piece aux = new Rook(true,1,1,boardMock, mock(Texture.class));

        assertTrue(pieza.equals(aux),"Las piezas son iguales");
    }

    @Test
    public void getPosTest()
    {
        Tile tileMock = mock(Tile.class);
        Board boardMock = mock(Board.class);
        when(boardMock.getTile(1, 1)).thenReturn(tileMock);
        when(tileMock.getX()).thenReturn(0f);
        when(tileMock.getY()).thenReturn(0f);

        pieza = new Rook(true,1,1,boardMock, mock(Texture.class));

        assertEquals(pieza.getPos().x,1,"El valor de x es el adecuado al del constructor");
        assertEquals(pieza.getPos().y,1,"El valor de y es el adecuado al del constructor");
    }
}

```

- TileTest:

```

package test.java;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import com.badlogic.gdx.math.Vector2;

import elements.Board;
import elements.Piece;
import elements.Tile;
import elements.pieces.Pawn;

```

```

import utils.Settings;

import static org.junit.Assert.assertTrue;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;

public class TileTest {
    Tile tile;

    @BeforeEach
    public void init() {
        tile=mock(Tile.class, Mockito.CALLS_REAL_METHODS);
    }

    @AfterEach
    public void terminate() {
        tile=null;
    }

    @Test
    public void setPieceTest(){
        tile.setPiece(mock(Pawn.class));
        boolean cumple=(tile.piece instanceof Pawn);
        assertEquals(cumple,true,"Se debe guardar la pieza en los atributos
de Tile");
    }

    @Test
    public void moveToTest(){
        Piece mockPiece = mock(Pawn.class);
        Tile mockTile = mock(Tile.class, Mockito.CALLS_REAL_METHODS);
        tile.setPiece(mockPiece);
        tile.moveTo(mockTile);
        assertEquals(mockTile.getPiece() instanceof Pawn,true,"Se debe
cambiar la pieza de una casilla a otra");
    }

    @Test // Reemplazable, lo puse por probar con otra forma de acceder a la
pieza
    public void getPieceTest(){
        tile.setPiece(mock(Pawn.class));
        boolean cumple=(tile.getPiece() instanceof Pawn);
        assertEquals(cumple,true,"Se debe guardar la pieza en los atributos
de Tile");
    }
}

```

```








@Test
public void equalsTest(){
    Tile tile2=mock(Tile.class);
    Vector2 pos = new Vector2(0, 0);
    when(tile2.getPos()).thenReturn(pos);
    when(tile.getPos()).thenReturn(pos);
    assertTrue(tile2 instanceof Tile &&
tile2.getPos().equals(tile.getPos()),"Deben ser iguales por sus posiciones");

}

}

```

Software usado:

- **Google Docs**, para crear la documentación. 
- **GitHub**, para el control de versiones. 
- **Discord y WhatsApp**, para la comunicación entre los miembros del equipo.  
- **Trello**, para la organización del trabajo. 
- **Adobe Photoshop**, para la creación del logo. 
- **Visual Paradigm**, para crear los modelos de los requisitos, los casos de uso y los diagramas de clase. 
- **Entornos de desarrollo**, para realizar el código del proyecto hemos usado *Eclipse IDE* e *IntelliJ IDEA*. 