-nameNotFound- Team

"JavaChess"

DOCUMENTO DE ARQUITECTURA

Autor(es)
Cáceres, Juan Manuel
Hidalgo, Fabian Nicolás
-
-

Versión de documento
1.0.0

Tabla de contenidos

1 Historial de revisiones

Versión	Fecha	Observación	Autor(es)
1.0.0	02/06/2022	Versión base del documento	Cáceres, Juan Manuel Hidalgo, Fabian Nicolás
1.0.1	19/06/2022	Correcciones en base a Feedback	Cáceres, Juan Manuel Hidalgo, Fabian Nicolás

2 Página de Aprobaciones

A continuación se encuentra la información de las personas cuya aprobación es necesaria para realizar cambios significativos al siguiente plan.

* Aprobación sólo será necesaria en caso de cambios mayores, no será necesaria para cambios menores.
* Aprobación sólo será necesaria en caso de que el cambio sea realizado por personas que no son Change Managers.

Configuration Manager	Fecha	Firma
Hidalgo, Fabian Nicolás	02/06/2022	

Configuration Manager BackUp	Fecha	Firma
Cáceres, Juan Manuel	02/06/2022	

Build Manager	Fecha	Firma
Hidalgo, Fabian Nicolás	02/06/2022	

Release Manager - Issue Coordinator	Fecha	Firma
Cáceres, Juan Manuel	02/06/2022	

3 Introducción

En esta sección introductoria al documento se buscará explicar de manera detallada y clara como es la arquitectura base del diseño del proyecto propuesto, y cómo fueron ingeniados los componentes que buscamos ser implementados en el mismo.
Buscaremos exponer la arquitectura general del sistema mediante gráficos, para lograr visualizar los diferentes componentes y sus relaciones entre las distintas partes y cómo se es el complemento entre el usuario y el sistema del juego mediante las interfaces externas. Usaremos diagramas UML y diagramas de despliegue de componentes.
Explicaremos en palabras cuál es el patrón principal de arquitectura elegido para el proyecto y exponiendo cuál fue el motivo de la elección pensando en requerimientos no funcionales.
Finalmente, detallaremos algunos de los casos de prueba de integración que pensamos son necesarios, y que se buscarán desarrollar para corroborar la correcta interacción entre todos los diferentes módulos que conforman el sistema.

3.1 Referencias / Acrónimos / Glosario

Acrónimo	Descripción
SCM	Software Configuration Management - Administración de las Configuraciones de Software
CI	Continuous Integration - Integración Continua
CD	Continuous Deployment - Implementación Continua
ID	Identification Code - Código de Identificación
AI	Artificial Intelligence - Inteligencia Artificial
UI	User Interface - Interfaz de Usuario
1vsCOM	Jugador vs Computadora
1vs1	Jugador vs Jugador
SRS	Software Requirements Specifications - Especificación de Requerimientos de Software
MVC	Model-View-Controller - Modelo-Vista-Controlador
UT	Unit Test - Test Unitario
IT	Integration Test - Test de Integración

Tabla 1 - Acrónimos/Glosario

4 Descripción General

4.1 Arquitectura Principal del Sistema

Para arrancar, primero definiremos lo que sería directamente la arquitectura del sistema adoptada para la realización del proyecto, donde hemos elegido la implementación de la arquitectura Model-View-Controller, ya que consideramos es la adecuada para un juego de las características que manejaremos nosotros a lo largo del proyecto.

4.2 Características de la Arquitectura MVC

La arquitectura Model-View-Controller es un patrón de arquitectura de software, que separa los datos y principalmente lo que es la lógica de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello el patrón de arquitectura MVC propone la construcción de tres componentes distintos que son la vista, el modelo y el controlador, es decir, por un lado define los componentes para la representación de la información, y por otro lado para la interacción del usuario.
Este patrón de arquitectura de software se basa en las ideas de la reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de la aplicación y su tarea de mantenimiento.

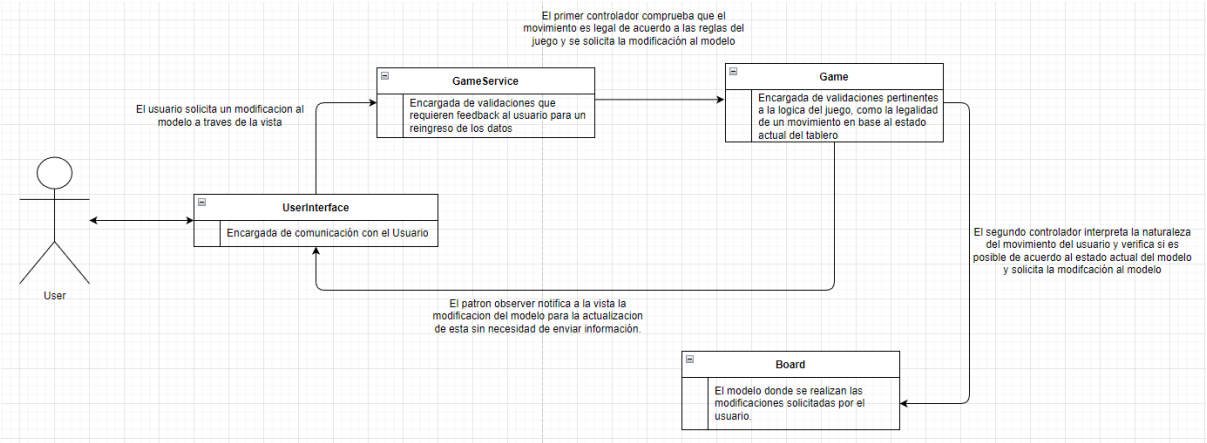
4.3 El Porqué de la Elección del Patrón de Arquitectura MVC

La elección de este patrón de arquitectura de software viene fundamentado con la idea de que tendremos una vista, que será la encargada de presentar al usuario todo lo relacionado al juego, como ser el tablero, las piezas en juego y las piezas tomadas, cronómetros, el menú con sus diferentes apartados y configuraciones de estrategia, entre otras cosas. A su vez el usuario debe ser capaz de interaccionar con la aplicación ya sea para seleccionar un apartado del menú, como para por ejemplo realizar un movimiento de sus piezas, etc, por lo tanto además de visualizar debemos tomar cierta información proveniente del jugador, donde dada la arquitectura de software elegida lo realizará la parte de la View.
Cada objeto representado en la vista es un objeto, como ser el tablero y las piezas, y estos en la arquitectura forman la parte de Model, y a su vez, entre ellos deben ser capaces de interactuar entre sí, sobretodo entre las piezas y el tablero (por cuestiones relacionadas a la naturaleza del juego de ajedrez en sí) y cada movimiento o selección debe cumplir con las reglas del juego, por lo tanto deberemos definir movimientos válidos, posiciones dentro del tablero válidas, o por ejemplo en caso de los menús deberemos definir cómo actuará la interfaz gráfica al seleccionar una opción, entre otras relaciones que se pueden distinguir. Todas estas interacciones y validaciones se llevarán a cabo dentro de la parte del Controller. Cabe detallar adicionalmente que si bien la parte lógica de la aplicación se busca que se desarrolle dentro del Controller, este módulo, dentro del código fuente, se verá dividido en dos clases con el objetivo de separar los tipos de verificaciones que se requiere realizar sobre la input del usuario. Verificaciones que puedan requerir un reingreso de los datos por parte del usuario serán responsabilidad de la clase “GameService” que interactúa de forma

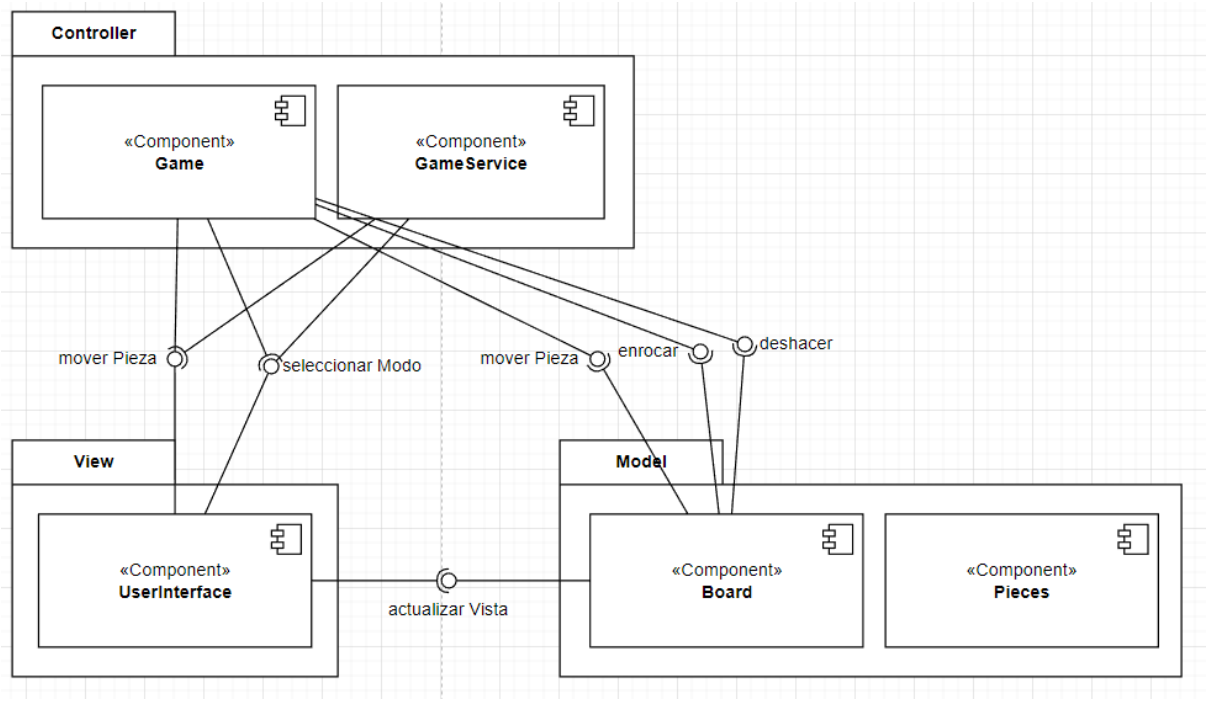
directa con la vista. Una vez verificado la legalidad del movimiento que pretende realizar el jugador se envían los datos a la clase “Game” quien es la encargada de interpretar la naturaleza del movimiento realizado por el usuario y la posibilidad del mismo de acuerdo al estado actual del tablero y las piezas, nuestro Model.
En términos de requerimientos no funcionales previamente establecidos, buscábamos una facilidad de mantenibilidad y reparación, así como un proceso libre de dificultades a la hora de agregar nuevas funcionalidades. La utilización de una arquitectura MVC permite una separación de las distintas partes del programa, por lo que a la hora de identificar errores o fuentes de bugs en el programa será más fácil poder encontrar la misma si se encuentran separadas según funcionalidades. Además esta arquitectura permite la modificación de la lógica, vista, o partes del juego sin requerir una modificación del resto, por lo que a la hora de reparar algo o añadir una nueva funcionalidad solo se deberá modificar una sección del código. Por último buscábamos una rapida actualizacion de la vista del usuario para que en el modo multijugador se pueda observar rápido las acciones enemigas y tener un buen ritmo del juego, por lo que una arquitectura MVC que utiliza el patrón observer permite una aceleración de la actualización de la vista al no requerir el reenvío de información a esta.

5 Diagramas del sistema

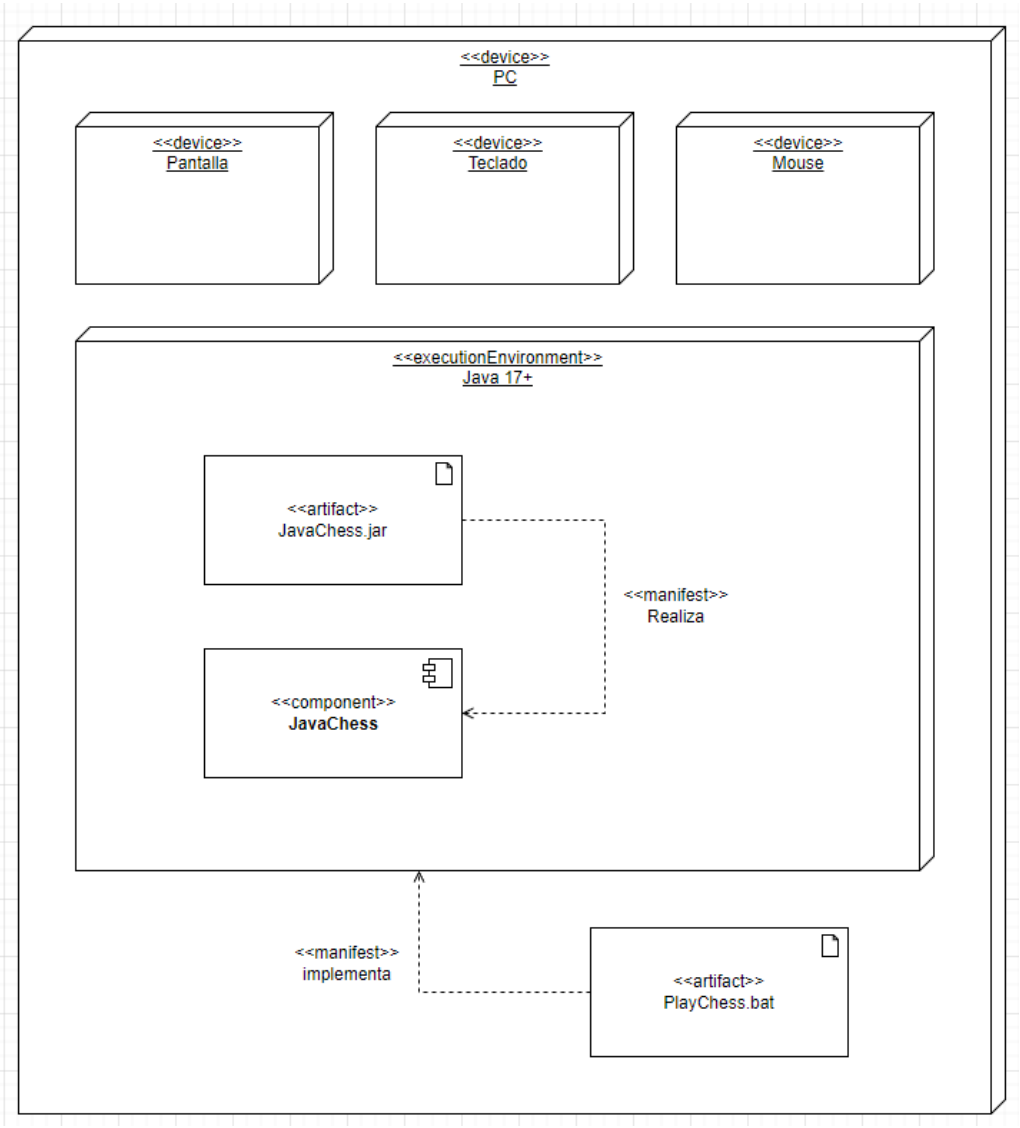
5.1 Diagrama Básico del Funcionamiento MVC



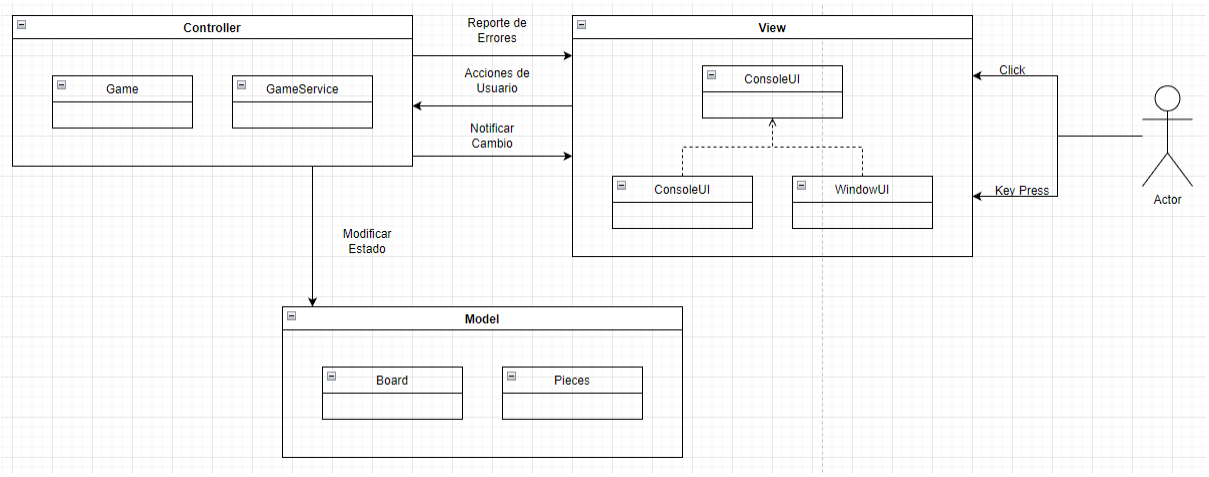
5.2 Diagrama UML de los Componentes del Sistema



5.3 Diagrama de Despliegue del Sistema



5.4 Gráfico de Arquitectura General



6 Pruebas de integración del sistema

6.1 Qué son las Pruebas de Integración del Sistema

Las pruebas de integración de sistema son una fase de desarrollo y prueba en la ingeniería de software en la que se reúnen varias unidades o módulos dentro de un programa o configuración de hardware. Por lo general, estas diversas unidades se han probado individualmente para garantizar que cada unidad funcione correctamente por sí sola y en esta fase de prueba se integran estas unidades dispares en un sistema más grande. Esto se puede hacer en el desarrollo de software para garantizar que diferentes aspectos de un programa más grande puedan trabajar juntos o en pruebas de hardware para garantizar la funcionalidad adecuada entre diferentes unidades.
Las pruebas de integración de sistema son una fase clave de las pruebas, ya que pueden encontrar muchos errores y se pueden desarrollar versiones iniciales del software o hardware en general.
Las pruebas de integración de sistema son conocidas también como Integration Test, y se consideran típicamente como la segunda fase principal de las pruebas. La primera fase se denomina Unit Test o pruebas unitarias y se centra principalmente en probar unidades individuales o partes de un programa o sistema más grande para garantizar que cada unidad funcione por sí sola. Estas unidades separadas se reúnen en pruebas de integración, una vez que han pasado las pruebas unitarias, para luego asegurarse de que pueden trabajar juntas en grupos más grandes. Posteriormente, normalmente se agruparán en el sistema o programa de ese sistema completo, lo que se denomina prueba del sistema.

6.2 Qué Buscamos con Nuestras Pruebas de Integración del Sistema

Con la realización de las pruebas de integración de sistema que efectuamos, buscamos principalmente que las clases definidas puedan interaccionar correctamente y evitar malfuncionamientos sobretodo entre lo que consideramos la parte más esencial del proyecto que son los movimientos de las piezas sobre el tablero, ya que de acuerdo a las reglas del ajedrez, existen muchas validaciones necesarias antes de realizar ciertos movimientos, un posible caso de ejemplo es el enroque de Rey y Torre, donde primero necesitamos capturar las inputs del usuario mediante la View de la aplicación, corroborar que esas entradas sean válidas dentro del tablero y que también efectivamente se relacionan a piezas propias. Luego esas posiciones se trasladan a la sección del Controller, donde se llevan a cabo validaciones como verificar que el usuario haya seleccionado ambas piezas (Rey y Torre, en cualquier orden), ninguna de esas piezas deben haber sido movidas previamente, ninguna posición por las que se moverá el Rey debe encontrarse atacada por piezas rivales ni debe existir Jaque en el momento de realizar el movimiento, y por último no deben haber piezas entre la Torre y el Rey. Todas estas validaciones requieren que en caso de no cumplirse, el usuario debe indicar un movimiento nuevo, o en caso de ser posible la selección, realizar el movimiento y eso indica que la Sección de View con la de Controller y la del Model se ven continuamente interaccionando entre sí, pudiendo llegar a ocasionar conflictos.

Si bien tenemos más clases y objetos, acá solo resumimos un caso concreto en estos 3 grandes grupos (Model, View, Controller) cuando en la realidad, en la etapa de Controller, nosotros tendremos la clase Game y la clase GameService que serán todas parte del grupo que se encarga de validar y controlar las acciones requeridas, como también poseeremos patrones de diseño que requieren una interacción entre clases, como por ejemplo, el uso del patrón Observer, Strategy, Factory, entre algunos que se implementan en el sistema. Esos patrones debido a su forma de operar requieren una constante y sólida implementación donde en caso de haber malfuncionamiento entre objetos, puede quedar completamente inútil su funcionalidad, causando que por ejemplo, no se cargue la estrategia deseada y la computadora no reaccione con algún movimiento, o que las vistas no se actualicen quedándose estáticas y el usuario no tenga ningún tipo de retroalimentación, o que los objetos que se deban generar mediante Factory no se generen, y nunca se creen piezas, o en el caso del Singleton que puedan llegarse a generar múltiples tableros y partidas paralelas al elegir múltiples veces la opción del menú para partida “Un Jugador”.

6.3 Listado de Pruebas de Integración del Sistema

Estas pruebas son ejecutadas sobre el programa compilado.

I001: **Seleccionar Vista:** Iniciar el programa y responder a la pregunta del sistema sobre qué vista utilizar. Verificar que al ingresar “window” se muestre el juego por ventana, y que al ingresar cualquier otra cosa se muestre el juego por consola.

I002: **Mover Pieza Propia a Casilla Vacía con Movimiento Legal:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza de acuerdo a sus movimientos definidos en las reglas de ajedrez. Realizar esto por cada pieza. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que cada pieza efectúe su movimiento.

I003: **Mover Pieza Propia a Casilla Vacía con Movimiento Ilegal:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza de una forma no listada en sus movimientos definidos en las reglas de ajedrez. Realizar esto por cada pieza. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se prevenga el movimiento de la pieza.

I004: **Mover Pieza Con Camino Bloqueado:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza propia que no pueda moverse por encima de otras piezas, cuyo movimiento esté bloqueado de acuerdo a sus movimientos definidos en las reglas de ajedrez. Realizar esto por cada pieza. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se prevenga el movimiento de la pieza.

I005: **Mover Pieza Con Camino Bloqueado que Puede Saltar:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza propia que pueda moverse por encima de otras piezas, cuyo movimiento esté bloqueado de acuerdo a sus

movimientos definidos en las reglas de ajedrez. Realizar esto por cada pieza. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que cada pieza efectúe su movimiento.

I006: **Mover Pieza Enemiga:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza enemiga. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se prevenga el movimiento de la pieza.

I007: **Tomar Pieza Enemiga:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza de acuerdo a sus movimientos definidos en las reglas de ajedrez para tomar una pieza enemiga dentro de sus posibilidades. Realizar esto por cada pieza. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que pueda tomarse la pieza.

I008: **Tomar Pieza Enemiga Fuera de Alcance:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza de acuerdo a sus movimientos definidos en las reglas de ajedrez para tomar una pieza enemiga fuera de sus posibilidades. Realizar esto por cada pieza. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que no pueda tomarse la pieza.

I009: **Tomar Pieza Propia:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y mover una pieza de acuerdo a sus movimientos definidos en las reglas de ajedrez para tomar una pieza propia dentro de sus posibilidades. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que no pueda tomarse la pieza.

I010: **Realizar Enroque:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y seleccionar el Rey y Alfil propios para realizar un enroque, cumpliendo las condiciones definidas en las reglas de ajedrez. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se produzca el enroque visualizado en la vista.

I011: **Intentar Enrocar con Inputs Incorrectos:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y seleccionar el Alfil y Rey propios para realizar un enroque. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se informe al usuario el procedimiento correcto de enroque.

I012: **Intentar Enrocar fuera de Condición de Enroque:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y seleccionar el Rey y Alfil propios para realizar un enroque, fuera de las condiciones definidas en las reglas de ajedrez. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se informe al usuario que no está en condiciones de enroque.

I013: **Visualizar Piezas Tomadas:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y tomar una pieza enemiga. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se visualicen las piezas tomadas en la vista.

I014: **Informe de Jaque:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y provocar una posición de Jaque propio o enemigo. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se informe en la vista la posición de Jaque.

I015: **Prevención Propio Jaque:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y provocar una posición de Jaque propio moviendo una pieza propia que estaba protegiendo el Rey propio. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se prevenga el movimiento.

I016: **Informe Jaque Mate:** Iniciar el programa, seleccionar cualquier vista, cualquier modo de juego, y provocar una posición de Jaque Mate propio o enemigo. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista. Verificar que se informe en la vista la posición de Jaque Mate.

I017: **Seleccionar Modo de AI:** Iniciar el programa, seleccionar cualquier vista, seleccionar modo de juego Singleplayer y seleccionar un modo de AI. Realizar movimientos y comprobar que el programa responda de acuerdo a lo seleccionado. Cambiar la selección en ejecución y verificar que el programa cambie su respuesta a lo seleccionado. Comprueba la interacción entre la selección desde la vista, la validación de los controladores, y la modificación del tablero mostrado en la vista.