TP1: Ta-Te-Ti

Alumno: Santino Nicolas Andreatta

Profesor: Ezequiel Barrios

Curso: 5º AO

Asignatura: Laboratorio de Programación Orientada a Objetos

Fecha de entrega: 05 / 09 /2024

Indice

Introducción

Consigna

Se nos solicita poder desarrollar y documentar un juego de Ta-Te-Ti funcional, utilizando python y los distintos conceptos vistos en clase. Se pide que el código tenga calidad funcional (funcione en base a lo pedido), calidad descriptiva (código limpio) y que se haga uso de POO, se respeten los principios SOLID y del uso de patrones de diseño.

Propósito

El objetivo de este trabajo es aplicar los conocimientos del bimestre a un juego simple de Ta-Te-Ti. Esto incluye POO, patrones de diseño, metodologías de desarrollo, y opcionalmente, conceptos de colecciones, iteradores y generadores. Se deben utilizar tests automatizados (preferentemente TDD) y otros tipos de pruebas (manuales, por ejemplo).

Desarrollo

Metodología de Diseño de Software

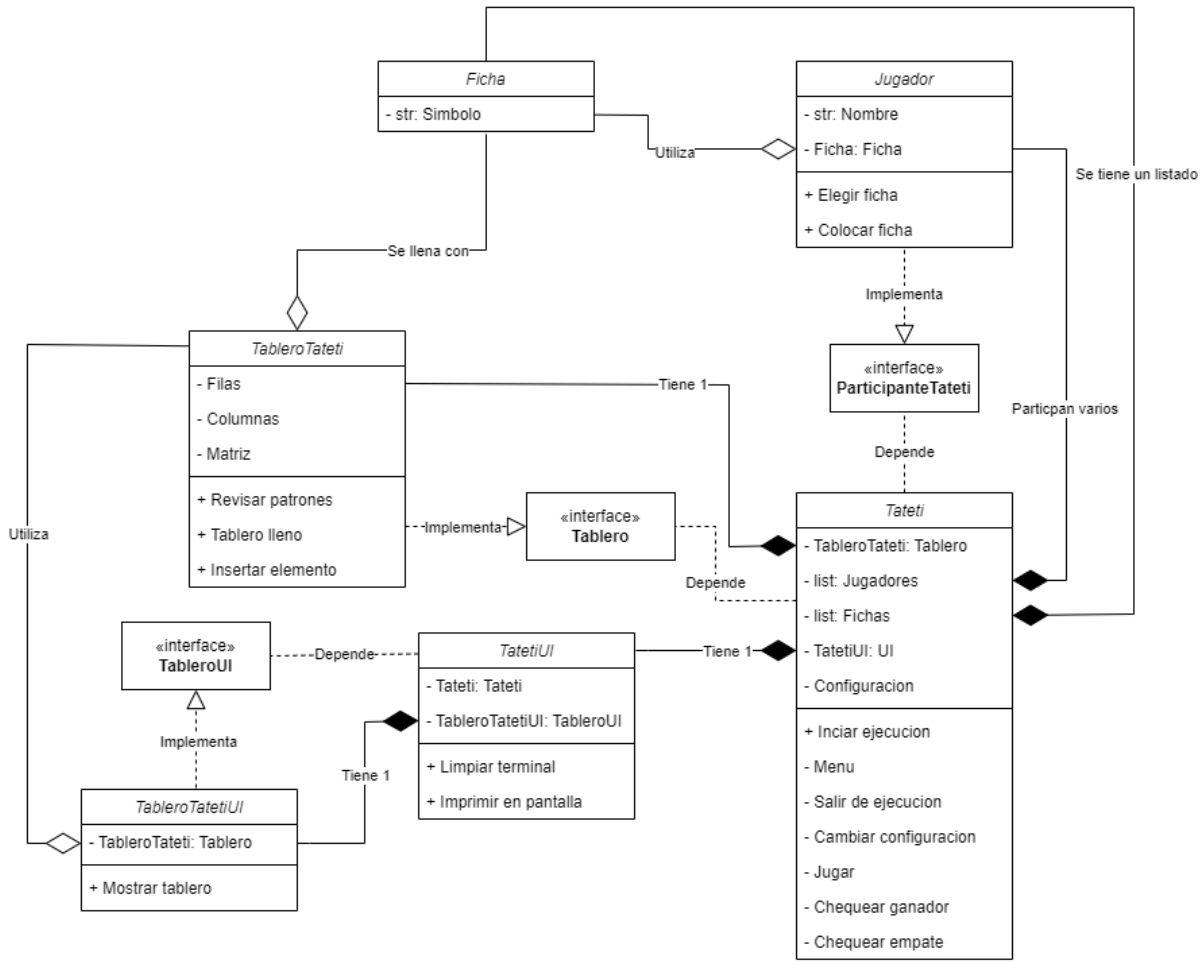
TDD

Utilice esta metodología para generar una base sólida sobre la cual comenzar el desarrollo de la lógica del juego. En primer lugar comencé con las clases Ficha, Jugador y Tablero (siendo tablero la clase con más funcionalidades). Más adelante modifiqué estas clases, utilizando 2 clases abstractas Tablero y ParticipanteTateti para Tablero y Jugador (lo cual permitiría extender las funcionalidades del proyecto en un futuro).

Método Incremental

Para el desarrollo de la interacción con los usuarios mediante la entrada/salida estándar utilicé el método incremental. Esto se debe a que la interacción por terminal con el programa es un poco más compleja que la interacción entre clases y código ejecutable y la realización de tests automáticos para TDD se hace más compleja, y hacer TDD con tests automáticos no me pareció cómodo.
Primero realice una lista de todas las funcionalidades que quería agregar (menú principal, poder jugar una partida, etc.) y las programé con requerimientos mínimos. Luego expandí las funcionalidades que consideré necesarias (agregar más de 2 jugadores, tableros más grandes, adaptación de la condición de victoria, refactorizaciones, etc.).

Modelo de dominio



https://drive.google.com/file/d/1mgv9fy_jWO9yJdZ8LgO6jaS6PCMdp8wj/view?usp=drive_link (Acceso a draw.io en caso de que la calidad de imagen no sea suficiente)

Interacción entre clases y justificación del diseño

Ficha (jugador.py)

La ficha es utilizada por los **jugadores** y es puesta en un **tablero**. Los símbolos de ficha disponibles para una partida de Ta-Te-Ti están explícitos en un atributo del **tateti** (Tateti.__fichas_simbolos).

Me pareció importante tener una clase que maneje cómo es la interacción entre fichas, además de que puede ser necesaria para extender la funcionalidad del juego. La ficha tiene la única responsabilidad de manejar la interacción entre fichas.

Jugador y ParticipanteTateti (jugador.py)

El jugador utiliza una **ficha** y puede colocarla en un **tablero**. Los jugadores se almacenan en un atributo del **tateti** (Tateti.__jugadores). El **jugador** implementa la clase abstracta **ParticipanteTateti**.
La interfaz ParticipanteTateti me parece necesaria para la implementación de futuros tipos de participantes como una inteligencia artificial contra la cual poder jugar una partida. El jugador tiene la única responsabilidad de manejar la realización de movimientos (una IA calcularía su movimiento y luego lo realizaría).

TableroTateti (tablero_tateti.py) - Tablero (tablero.py)

El tablero de tateti es el responsable de generar la matriz donde se colocarán las **fichas**. Tiene métodos para detectar patrones (3 elementos consecutivos en diagonal, tablero lleno, etc.) que son utilizados por la lógica del **tateti** para detectar estados de victoria o empate dentro del juego. Implementa la interfaz de Tablero.
La interfaz tablero no se si es 100% necesaria para futuras implementaciones, pero me pareció correcto ya que distintos tableros deberían buscar distintos patrones o manejar distinta lógica dependiendo de su uso, por lo que puede tener sentido. El tablero tiene la responsabilidad de manejar que fichas se ponen y donde, por lo que también puede detectar patrones de fichas para indicarle al juego si hay algún patrón que corresponda con un estado de juego.

TableroUI (tateti_ui.py) - TableroUI (tablero.py)

Esta clase implementa TableroUI. Es la clase encargada de mostrar en pantalla el **tablero de tateti.** Es usado por **TatetiUI** para mostrar en terminal el tablero.

TatetiUI (tateti_ui.py)

Se encarga de mostrar en pantalla lo que necesite el **tateti**, todo lo relacionado con los menús del Tateti, estados de juego, mensajes de error, el tablero, etc.
Me pareció necesario separar la responsabilidad de impresión en la terminal y manejo de tiempos de tablero y de tateti, ya que ambos requieren de esto en múltiples ocasiones y separar esta responsabilidad no solo es sencillo, sino que facilita la comprensión del código al revisar la clase Tateti.

Tateti (juego.py)

Esta clase contiene la lógica del juego, la configuración de juego y la lógica de recorrido de menús e interacción con el usuario. A través de la lógica de juego determina cuando debe poner una ficha determinado jugador y cuando finaliza la partida. Hace uso de todas las clases mencionadas anteriormente.

Si bien la clase tateti tiene múltiples responsabilidades (por lo que no cumpliría con Principio de Responsabilidad Única), no me pareció necesario separarlas de esta única clase ya que esto, a mi parecer, haría más compleja algunas interacciones entre métodos y clases. Además de que el proyecto es lo suficientemente simple como para que se entienda y tenga sentido la existencia de esta única clase.

Testeo y Ejecución

Tests automáticos

Este tipo de tests fueron utilizados para analizar los métodos de las clases **Ficha, Jugador y TableroTateti** que no requieran de un input por terminal. Los tests automáticos fueron acompañados del uso de TDD para el desarrollo de la base del proyecto. Los tests están repartidos en 2 archivos: **test_jugador.py** y **test_tablero_tateti.py**. Con estos tests busqué:

- Que se detecten errores correctamente (en constructores y en varios métodos).
- Que las fichas se inserten correctamente en el tablero.
- Que las condiciones de victoria (el chequeo de patrones) sea funcional.

Tests manuales y Ejecución

El juego se ejecuta importando el módulo de juego y la clase Tateti. Luego, simplemente poniendo Tateti.iniciar(), se ejecuta el juego.
Para realizar los testeos manuales, creé un archivo de ejecución del juego (ejecutable.py) y siguiendo la metodología incremental que mencioné anteriormente, fuí testeando las distintas funcionalidades.
Estos son algunos de los tests que utilice para probar la funcionalidad del juego:

- Menú principal (jugar, salir)
    - Ejecutar el juego y luego del mensaje ver el menú con las posibles opciones.
- Poder jugar y colocar una ficha
    - Ejecutar el juego → Elegir el número de la opción que indique jugar → Ver el tablero → Poner coordenadas → Ver tablero con ficha en la coordenada exacta
- Poder terminar una partida
    - **Empate**: Lleno el tablero sin formar patrones → Sale mensaje de empate → Vuelve al menú
    - **Victoria**: Formo alguno de los 4 patrones posibles (3 fichas horizontales, 3 fichas verticales, 3 fichas en diagonal (izquierda → derecha), 3 fichas en diagonal (derecha → izquierda)) → Ver mensaje de victoria → Vuelve al menú

(Hice muchos más tests pero se entiende la idea, si sigo poniendo se me hace muy largo el informe)

Conclusión

Conclusión

A mi parecer, pude cumplir con la consigna y en términos de funcionalidad, el programa funciona correctamente. Por otro lado, creo que el trabajo. en cuanto a cumplir con los principios SOLID, no fue el mejor ya que tengo una clase con más responsabilidades de las que debería tener (Tateti). Este sería un aspecto a mejorar en lo posible.

Trabajo futuro

Si bien gran parte del posible trabajo futuro que tenía en mente ya lo realicé, voy a enumerar todo el contenido extra que se me ocurrió que podría agregarse al juego:

- Tableros más grandes
- Más jugadores → Más fichas
- Nuevas condiciones de victoria (n fichas seguidas en tableros más grandes)
- Jugar contra una IA
- Configuraciones de juego preestablecidas (como un selector de modos: Original (3x3, 2 jugadores y 3 fichas seguidas), etc.)

Algunas mejoras serían separar las responsabilidades de la clase tateti, y capaz también sacarle alguna responsabilidad a la clase TableroTateti. Es posible que las condiciones de victoria no estén 100% relacionadas con el tablero, por lo que habría que modificar la forma en que se chequean las condiciones de victoria.