

Informe Sprint 5

Nombre: **Jhonnatan Antonio Espinoza Rojas**

Fecha: **14/06/2023**

1. Demostración

n	Features	Estado
1	Se graba un juego simple completo de dos jugadores humanos.	✓ Completado
2	Se graba un juego general completo de dos jugadores humanos.	✓ Completado
3	Se graba un juego simple completo de jugadores humano-computadora.	✓ Completado
4	Se graba un juego general completo de jugadores humano-computadora.	✓ Completado
5	Se graba un juego simple completo de jugadores computadora-computadora.	✓ Completado
6	Se graba un juego general completo de jugadores computadora-computadora.	✓ Completado
7	Se realiza el uso de la función Record/Replay	✓ Completado

Diagrama de Clases



2. Historias de usuario

Historia de usuario

Como Jugador quiero grabar/reproducir una partida tal que pueda ver mis movimientos una vez terminado el juego.

Nombre de la historia de usuario: **Grabar/Reproducir partida**

Descripción: El jugador puede grabar/reproducir su partida una vez terminado el juego.

Επίσκεψη: 2 ημερ.

Criterios de aceptación

[illegible]

```
Given: El juego ha terminado.
When: El usuario presiona el botón de reproducir.
Then: El sistema muestra los movimientos de la partida.
```

3. Revisión de código

Classe revisada: **GameController.ts**

Fecha/hora de duración del ejercicio de revisión del código

Convenciones de nombre

Convención de ordenación de argumentos de método y setter van después del constructor

Estandares de codificación.	Comentarios significativos y válidos.	Se proporciona comentarios de documentación para describir la funcionalidad y el propósito de los métodos y variables. Los comentarios son significativos y ayudan a comprender el código.	
	Estilo consistente de bloques de código.	Gracias a prettier podemos mantener un estilo consistente en cada bloque de código definiendo previamente las convenciones que usamos en el lenguaje.	
	Indentación consistente.	De igual forma con la ayuda de la herramienta prettier podemos lograr mantener una indentación consistente atraves de todo el codigo (1 Tab = 4 espacios)	
Principio de diseño.	Clase o método no bien modularizado.	Existe una buena modularización en la clase.	
	Visibilidad adecuada de cada variable, método y clase.	Las variables se definen como private, lo que limita su acceso solo dentro de la clase. Los métodos importantes, como getCurrentPlayer(), gameState(), etc., se definen como public para permitir su acceso desde otras partes del programa.	
	Clase o método con pobre abstracción.	El método con una pobre abstracción en la clase gameController es el método de checkSOS	
	Diseño por contrato (pre/postcondiciones).	No se muestra explícitamente un diseño por contrato utilizando precondiciones y postcondiciones en los métodos. Solo el metodo checkSOS y makeMove muestran un intento de diseño por contrato	
	Violación del Principio Abierto-Cerrado.	La clase GameController esta diseñada para ser extensible, ya que permite agregar nuevas funcionalidades y comportamientos sin modificar la propia clase.	
Smells código.	Violación del Principio de Responsabilidad Única.	Cumple con la responsabilidad unica de controlar el juego pero considero que algunos metodos como checkSOS podrian salir de la clase ya que es una responsabilidad de verificar el tablero	
	Números mágicos.	Si, el código tiene números mágicos, para ser específico el currentPlayerIndex se inicializa con un 0, una solución factible seria el uso de constantes, luego de eso si se utilizan constantes para las demas variables.	
	Variable global /clase innecesaria.	No se utilizan variables globales, debido a que en js y ts se utiliza var, let y const para variables, pero se recomienda no usar var para variables debido a que var se utiliza para declarar variables globales, respetando esa pauta no se ah usado var para declarar variables globales solo let y const	
	Código duplicado.	No existe código duplicado en la clase GameController, por suerte se ah tenido en consideración desde un principio para evitar la refactorización constante por este problema	
	Métodos largos.	Si existe un método largo en esta clase el metodo es el checkSOS, como ya menciono antes lo mejor seria extraer una clase y llamarla en el makeMove	
Errores.	Larga lista de parámetros.	No se encontraron listas largas de parametros en los metodos ni en el constructor de la clase	
	Expresión demasiado compleja.	No se utilizaron expresiones complejas, con la intencion que sea facil de entender para las personas que quisieran tomar como referencia el código para sus proyectos	
	Switch o if-then-else que necesita ser reemplazado con polimorfismo.	Si en el checkSOS hay 2 if-else que podrian ser reemplazados usando el polimorfismo	
	Nombre de método o variable cuya intención no está clara.	Considero que checkSOS no esta bien declarada ya que no especifica si verifica el SOS o los puntos o si el juego ah terminado, tampoco considero que la clase reset este bien definida creo que mas claro seria llamarla resetGame especificando que lo que va a reiniciar es el juego completo.	
	¿Algún método similar en otras clases?	Si el metodo reset en Record y en GameController ademas de algunos getters para que sean accedidos directamente desde el gameController	
Errores.	Fragmento de código con errores.	¿Cuál es el error?	¿Por qué es un error?
	No se encontraron errores en la clase.	-	-

4. Resumen de todo el código

Modulos	Tipos	Nombre del archivo de código fuente	Código de producción o prueba?	# líneas de código
Lógica de negocio	Controllers	GameController.ts	Producción	208
	Models	Board.ts	Producción	103
		Game.ts	Producción	77
		Player.ts	Producción	50
		Record.ts	Producción	44
		Score.ts	Producción	49
	Helpers	Checker.ts	Producción	299
		MoveGenerator.ts	Producción	171
	Interfaces	Essentials.ts	Producción	27
		MakeMove.ts	Producción	15
		Movement.ts	Producción	22
		Winline.ts	Producción	32
	Constants	Difficulty.ts	Producción	21
		GameMode.ts	Producción	21
		GamePlayers.ts	Producción	16
GameState.ts		Producción	16	
GameType.ts		Producción	16	
GameWinner.ts		Producción	26	
	Letter.ts	Producción	21	
Interfaz Grafica	Utils	useContextGame.tsx	Producción	40
	Pages	Home/page.tsx	Producción	40
		Setting/page.tsx	Producción	137
		Game/page.tsx	Producción	291
	Components	GameBoard.tsx	Producción	56
		GameControls.tsx	Producción	33
		Lines.tsx	Producción	57
		ModalWinner.tsx	Producción	33
		ReplayButton.tsx	Producción	21
ScoreBoard.tsx		Producción	20	
	TurnIndicator.tsx	Producción	22	
Pruebas Unitarias	__Test__	ComputerPlayingGame.test.ts	Test	141
		GeneralGameFinished.test.ts	Test	90
		GeneralGameWrite.test.ts	Test	28
		PreGame.test.ts	Test	34
		RecordGame.test.ts	Test	31
		SimpleGameFinished.test.ts	Test	66
		SimpleGameWrite.test.ts	Test	28

proyecto. 📖

Aprendí a utilizar React, TypeScript y Jest. También

... a utilizar el sistema de control de versiones Git y la plataforma de desarrollo colaborativo GitHub.

¿Qué ganaste personalmente con el proyecto?

Desde mi perspectiva personal, disfruté mucho trabajando en esta propuesta. Considero

extremadamente interesante y me proporcionó una gran cantidad de aprendizaje en el ámbito del desarrollo de software. En cuanto a la codificación, me encontré con varios desafíos de lógica, pero gracias a las pruebas unitarias pude identificar y solucionarlos a tiempo, evitando así perder largas horas en la búsqueda de errores. Esto aceleró significativamente mi proceso de desarrollo. En lo que respecta al diseño, pude aprovechar al máximo el uso de componentes y ganchos (hooks), lo que resultó en un código más limpio y fácil de comprender.

A lo largo de los sprints, tuve que refactorizar el código en varias ocasiones, ya que constantemente surgían nuevas

funciones o comentarios que modificaban mi perspectiva sobre cómo abordar la programación. Por ejemplo, me pareció sumamente interesante el uso de Stubs, dado que los movimientos de la computadora debían ser aleatorios. Por lo tanto, sentí la necesidad de implementar esta lógica en mi código, lo que me llevó a realizar cambios significativos en el GameController y el MoveGenerator de mi aplicación.

proyecto?

En mi opinión, considero que la aplicación cumple con los criterios establecidos y funciona correctamente

No logré alcanzar una nivel de abstracción de código muy bueno, lo que me impide considerarlo como un código limpio. Además, el uso de tailwind CSS directamente en las páginas, sin hacer uso de componentes personalizados, tampoco me agrada del todo, ya que dificulta la lectura y comprensión de la lógica de la aplicación.

Un aspecto que debo mejorar es los movimientos generados aleatoriamente en las dificultades medium y hard

En la dificultad hard, la computadora no siempre elige el movimiento que le permite ganar la partida, sino que en ocasiones elige un movimiento que le permite al jugador ganar la partida. Esto se debe a que la computadora elige un movimiento aleatorio, sin tener en cuenta si el movimiento es ganador o no. Por lo tanto, considero que debo mejorar la lógica de la computadora para que elija un movimiento ganador en lugar de uno aleatorio.

trabaja, no le óptimo utilizar en su totalidad características como `getServerSideProps` para realizar llamadas al backend.

En lugar de eso, opté por renderizar la mayor parte de la lógica desde el cliente, lo cual desde mi punto de vista puede afectar el rendimiento. Debido a limitaciones de tiempo y a mi nivel de experiencia en el desarrollo, siento que no pude aprovechar al máximo las funcionalidades de Next.js en este proyecto.

Sin embargo, considero que adquirí nuevos conocimientos valiosos que me ayudarán a seguir mejorando en el