**Reporte del Sprint #5**

Las principales tareas de esta asignación son:

1. Agrega la función de grabar (record) un juego en un archivo de texto. Se requiere la historia de usuario y los criterios de aceptación tanto de grabación como de reproducción
2. Realización de un ejercicio de revisión de código.
3. Resumir las lecciones aprendidas del Sprint 0 al Sprint 5.

El siguiente es un diseño de GUI de muestra del producto final, donde "Replay" es opcional.

**El trabajo es de caracter individual.**

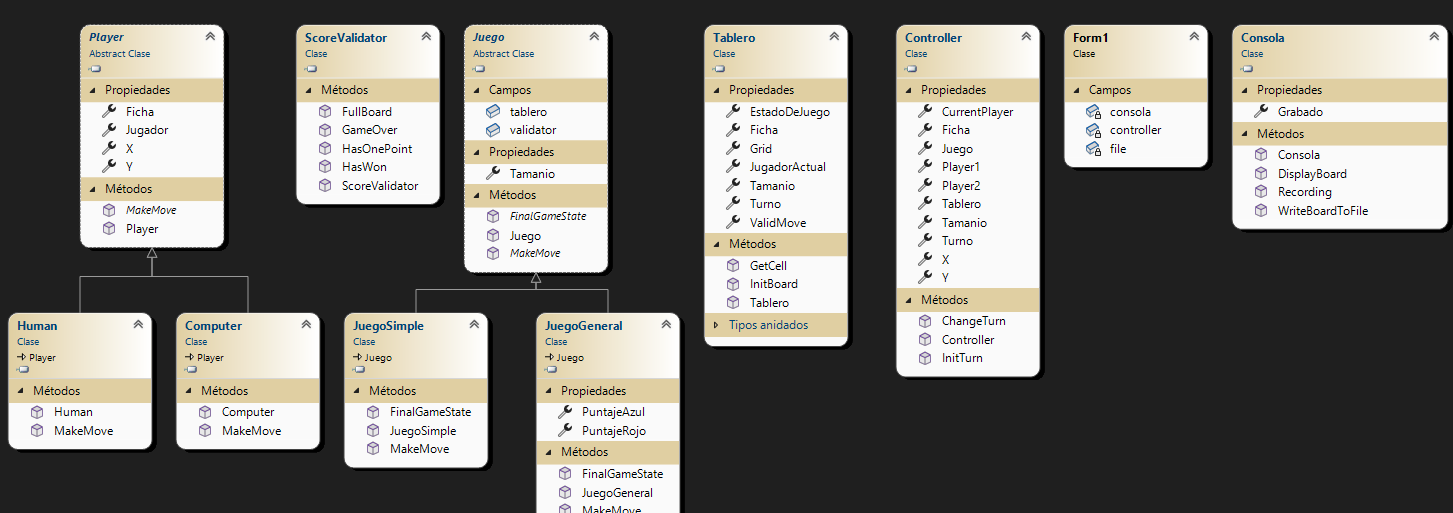
|  |  |  |
| --- | --- | --- |
| SOS Icon  Description automatically generated Simple game Icon  Description automatically generated General game Board size  8 | | |
| Blue player  Icon                                Description automatically generated Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon                                Description automatically generated Computer | Chart, line chart  Description automatically generated | Red player  Icon  Description automatically generated Human  Icon  Description automatically generated S  Icon  Description automatically generated O  Icon  Description automatically generated Computer  Replay |
| ☒ Record game | Current turn: blue (or red) | New Game |

Figura 1. Sample GUI layout of the final product Diseño de GUI del producto final

**Puntos totales**

1. **Demostración (10 puntos)**

Envía un video de no más de 15 minutos, demostrando claramente que has implementado todas las funciones en la siguiente tabla. En el video, debes explicar lo que se está demostrando. **Presenta el diagrama de clases de tu código de producción y describe cómo la jerarquía de clases en su diseño trata con los requisitos del oponente de la computadora.**



|  |  |
| --- | --- |
|  | **Feature** |
| 1 | Se graba un juego simple completo de dos jugadores humanos. |
| 2 | Se graba un juego general completo de dos jugadores humanos |
| 3 | Se graba un juego simple completo de jugadores humano-computadora |
| 4 | Se graba un juego general completo de jugadores humano-computadora |
| 5 | Se graba un juego simple completo de jugadores computadora-computadora |
| 6 | Se graba un juego general completo de jugadores computadora-computadora |

Si has implementado la función de "replay" para obtener crédito adicional, debes incluir tu demostración en el video.

1. **Historias de usuario y criterios de aceptación para los requisitos para los requerimientos Record/Replay (1 punto)**

**Plantilla de historia de usuario**: Como <rol>, quiero <objetivo> [tal que <beneficio>]

Agrega o elimina filas si es necesario

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Nombre de historia de usuario** | **Descripción de historia de usuario** | **Prioridad** | **Esfuerzo estimado (horas)** |
| 10 | Grabar Juego | Como Jugador  Quiero tener la capacidad de grabar un juego  Tal que pueda ver el desarrollo de la partida del juego grabado | INTERMEDIA | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID y nombre de la historia de usuario** | **AC**  **ID** | **Descripción del criterio de aceptación** | **Estado (completado, por hacer, en progreso)** |
| Historia 10 | 10.1 | AC 20.1 <El juego se está grabando>  Dado que soy un jugador y he iniciado la grabación de un juego  Cuando la grabación está en curso  Entonces se muestra un indicador visual que se está grabando | HECHO |
| 10.2 | AC 20.2 <Se ha creado el archivo de texto>  Dado que soy un jugador y he iniciado la grabación  Cuando la grabación está en curso  Entonces el archivo de texto se ha creado | HECHO |
| 10.3 | AC 20.3 <Se ha impreso el desarrollo del juego en el archivo>  Dado que soy un jugador y he finalizado la grabación de un juego,  Cuando la grabación ha terminado y se ha creado el archivo de texto  Entonces el archivo de texto contiene el desarrollo del juego | HECHO |

**3. Revisión de código (4 puntos)**

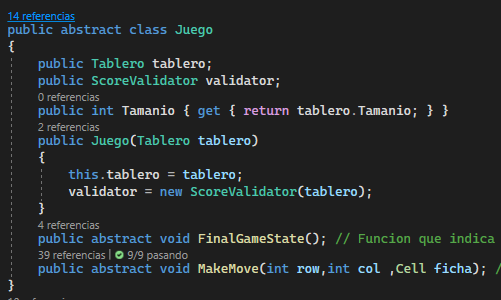
Aplica la revisión del código fuente a una o dos de las clases más importantes (y a otras clases si el tiempo te permite) e informa de los resultados. Además de buscar errores, la revisión debe verificar: (1) si todo el proyecto ha seguido el estándar de codificación de manera consistente, (2) si el proyecto ha seguido los principios de diseño presentados en clase y (3) si hay olores de código que indican la necesidad de refactorización.

Las siguientes listas de verificación proporcionan pautas básicas. Puedes agregar nuevos elementos a cada una de las listas de verificación. Asegúrate de que tus respuestas sean el resultado del ejercicio de revisión del código. Si no hay hallazgos para una entrada, debes proporcionar una explicación. Por ejemplo, si tu respuesta a "¿Se violan las convenciones de nomenclatura?" es no, debes describir una convención de nomenclatura y presentar un ejemplo. No recibirás puntaje por si tus respuestas son simplemente sí o no sin información adicional.

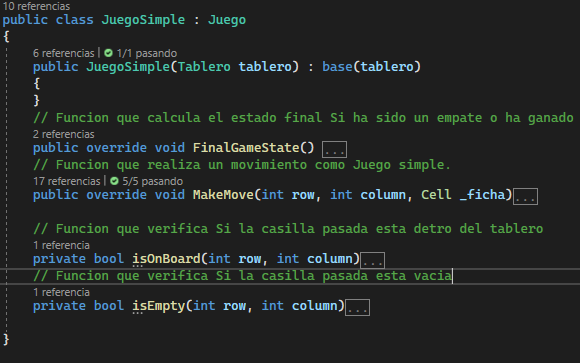
Clases que han sido revisadas:

Fecha/hora de duración del ejercicio de revisión del código:

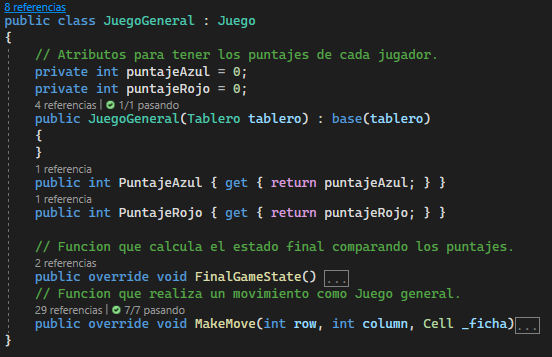
Clase Juego



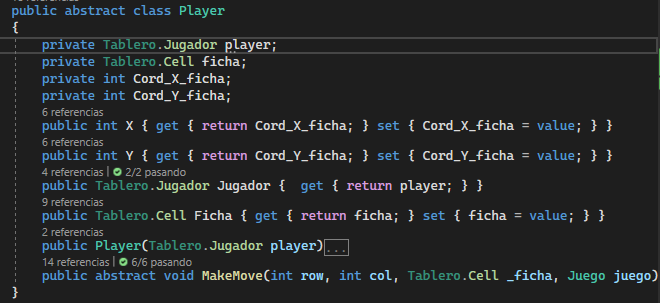
Clase Juego Simple (HIJO)



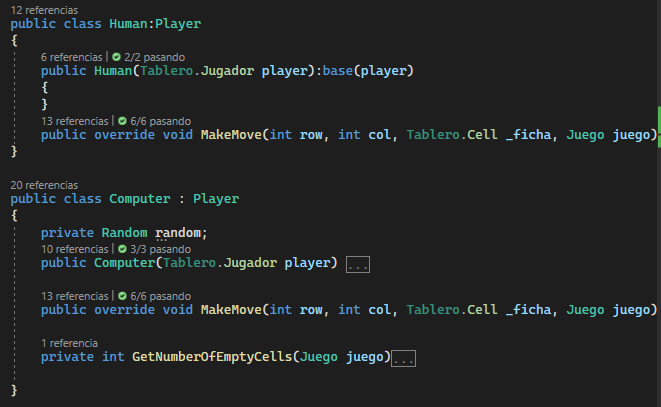
Clase Juego General (HIJO)



**Clase Player**



**Clases HIJAS : Human y Computer**



|  |  |  |  |
| --- | --- | --- | --- |
| **Checklist** | **Items Checklist** | **Conclusiones** | |
| Estándares de codificación | Convenciones de nombres | Los nombres de las variables tanto de las clases indican indican lo que representan. Ejmplo : El nombre de las clase Padre “Juego” como padre y sus HIJAS : “JuegoSimple”,”JuegoGeneral” dan idea de los objetos que se van a instanciar y del polimorfismo que se pretende usar. Además los nombres de métodos están en CamelCase. | |
| Convención de ordenación de argumentos de método | Siguiendo la convenciones de C# de Microsoft no aborda convenciones de argumentos de los métodos. Sin embargo podríamos mejorarlo añadiéndole alguno.En este caso no se realizo convenciones con este tipo. | |
| Comentarios significativos y válidos. | Seguimos las convención de empesar con mayúscula y terminar con un punto según Microsoft C# | |
| Colocación de directivas using fuera de la declaración del espacio de nombres | Las directivas se usa fuera de declaración siguiendo la convencion | |
| Convenciones de diseño | Escribimos solo una instruccion por linea,declaracion por linea y agregamos linea en blanco entre las definiciones de metodo y las propeidades. | |
| … |  | |
| Principio de diseño | Clase o método no bien modularizado | Considero que las dos clases padres y sus hijas están bien modularizadas Siguen el principio de abierto cerrado , pero también opino que el método de makemove de Computer puede mejorar debido a que usa parámetros que no le son necesarios solo para adecuar el poslimorfismo , en realidad al final no los utiliza porque randomea la fila y columna | |
| Visibilidad adecuada de cada variable, método y clase. | Contiene una visibilidad adecuada poniendo al inicio los atributos y luego los métodos, identificados cada uno con nombres que ayudan a saber que funciones va realizar, | |
| Alguna clase con pobre abstracción | Se identifica clases con buenas abstracciones que representan la idea abstracta de un juego SOS. | |
| Diseño por contrato ( pre/postcondiciones) | No se uso un diseño con precondiciones y pos condiciones | |
| ¿Se viola el Principio Abierto-Cerrado? | No se viola porque se pueden crear más clases que hereden de estos padres hasta podríamos crear un nuevo tipo de juego en vez de juego simple y general y añadir otros tipos de players quizás uno personalizado etc. | |
| ¿Se viola el Principio de Responsabilidad Única [[1]](#footnote-1)? | No se viola, más bien se añadieron clases para poder dividir esta responsabilidad y lo tenga cada clase en particular y no estar amontonando responsabilidades a una clase en particular. | |
| Smells código | Números mágicos |  | |
| Variable global /clase innecesaria | No tenemos clases innecesarias, todas se utilizan. | |
| Código duplicado | Creo que si, ahora que lo veo se repite 2 veces una parte :  PaintGrid();  ShowGameStatus();  ShowTurn();  controller.ChangeTurn(); | |
| Métodos largos | No existen métodos largos porque se ha tratado de refactorizar ayudándolo con otros métodos, los métodos más largos estarían en el form1 , pero con no más de 40 líneas | |
| Larga lista de parámetros | En el metodo makemove de la clase hija Computer tiene parámetros que no utiliza | |
| Expresión demasiado compleja | Existe expresiones complejas para simplificar los if, pero de ahi no mucho. | |
| Switch o if-then-else que necesita ser reemplazado con polimorfismo | Ahora que lo noto puede ser que los casos de computadora vs Computer pueda ser reemplazado con polimorfismo añadiendo una clase que identifique este caso. Pero como es solo uno opino que no es necesario. | |
| Nombre de método o variable cuya intención no está clara | Hay una variable booleana que es actualzargrid que esta en la clase form1 que se puede quitar | |
| ¿Algún método similar en otras clases? | No todos cumplen diferentes funciones y se ayudan entre si para armar la clase. | |
| … |  | |
| **Errores** | **Fragmento de código con errores** | **¿Cuál es el error?** | **¿Por qué es un error?** |
| Existe una parte que lo deje en comentario. Que don las funciones : StartComputerVsComputerGame() y EndComputerVsComputerGame(), en la clase form1. | Cuando quería desactivar el numeric updown salía una excepción. | En realidad, es una excepción y no un error. |
| Existía un error en los sprints4 y 3 que ya fueron corregidos | El juego no te mandaba ganador para un SOS de una de las diagonales | Sería considerado un error en la lógica del funcionamiento del juego o un mal manejo de los casos específicos |
|  |  |  |

**4. Resumen de todo el código (1 points)**

|  |  |  |
| --- | --- | --- |
| Nombre del archivo de código fuente | ¿Código de producción o de prueba? | # lineas de código |
| Juego.cs | Producción | 140 |
| Tablero | Producción | 60 |
| ScoreValidator | Producción | 61 |
| **UnitTest1.cs** | **Prueba** | **423** |
| **Consola.cs** | **Producción** | **79** |
| **Form1.cs** | **Produccion** | **313** |
| Player | Produccion | 106 |
| Controller.cs | Producción | 36 |
| Total | | 1001 |

Mejorando solo alteramos los valores de Form1, Consola para agregar lo del recording, UnitTest para agregar las pruebas de la historia de usuario 10

**No recibirás puntaje por esta tarea a menos que envíes tu código fuente completo.**

5. Resume las lecciones aprendidas de todo el proyecto respondiendo las siguientes preguntas desde la perspectiva de los procesos de desarrollo, codificación, diseño, refactorización y prueba (**4 puntos**):

* ¿Qué ganaste personalmente con el proyecto?
* ¿Qué hace bien tu proyecto y qué podría hacer mejor tu proyecto?
* ¿Cómo podrías mejorar tu proceso de desarrollo si desarrollas un juego similar desde cero?

Requisito mínimo para (5): Una página completa a espacio simple, tamaño de fuente no mayor a 12 puntos.

**Resolución:**

En el proyecto del juego SOS, aprendí muchas cosas importantes. Fueron cinco sprints en los que construí el proyecto poco a poco, como si estuviera armando una casa. La parte más complicada fue implementar la funcionalidad de jugar contra la computadora, eso me dio algunos dolores de cabeza. Aprendí a hacer pruebas y seguir la metodología RGB (Requerimientos, Diseño, Codificación) para desarrollar el proyecto de manera más organizada.

En cuanto a las lecciones aprendidas en el proyecto, desde la perspectiva de los procesos de desarrollo, codificación, diseño, refactorización y prueba, puedo responder a las preguntas de la siguiente manera:

¿Qué ganaste personalmente con el proyecto?

Personalmente, gané experiencia en el desarrollo de proyectos de software. Aprendí a planificar y organizar mis tareas en sprints, lo cual me ayudó a mantenerme enfocado y cumplir con los plazos establecidos. También adquirí conocimientos prácticos sobre el desarrollo de juegos y cómo aplicar diferentes técnicas de codificación y diseño en este contexto. Además, pude aprender lógicas interesantes, en este caso la idea de hacer que una computadora juegue vs un humano y se juegue computadora vs computadora. Al inicio no tenía idea de cómo hacerlo, pero viendo cómo funcionaba el TicTactoe y el Autotictactoe me dio una idea para continuar y arreglar el sprint 4 que estaba incompleto, ahora si funciona bien.

Luego implementando el recording también me ayudo a practicar el tema de archivos que ya lo habidas dejado oxidado y más en C# que es un lenguaje que me tuve que adaptar.

¿Qué hace bien tu proyecto y qué podría hacer mejor tu proyecto?

Mi proyecto se destaca por su funcionalidad principal, que es el juego SOS. La mecánica del juego funciona correctamente y ofrece una experiencia de juego satisfactoria. Sin embargo, hay áreas en las que podría mejorar. Algunas de las funciones no funcionaron como se esperaba y tuve que dejarlas como comentarios en el código. Esto podría mejorarse mediante una depuración más exhaustiva y pruebas más rigurosas para identificar y corregir posibles errores. En varias oportunidades se presentaron errores de excepción y entre otras cosas que me hacían dar cuenta de que no había restringido estos casos, y los fui restringiendo con puros "if" para arreglarlos. A veces estaba usando un objeto nulo o saliéndome del tablero, entre otros casos. Tenía que hacer un debug mental para poder solucionarlo y estructurar mi diagrama para ver de manera más general cómo funcionaba el proyecto.

¿Cómo podrías mejorar tu proceso de desarrollo si desarrollas un juego similar desde cero?

Si desarrollara un juego similar desde cero, podría mejorar mi proceso de desarrollo de varias formas. En primer lugar, podría dedicar más tiempo a la etapa de planificación y definición de requisitos, asegurándome de tener una comprensión clara de los objetivos y funcionalidades del juego antes de comenzar a codificar. Además, podría implementar una estrategia de pruebas más robusta, incluyendo pruebas unitarias y de integración para verificar la funcionalidad de manera más exhaustiva. También consideraría la posibilidad de realizar refactorizaciones periódicas para mejorar la estructura y legibilidad del código a medida que avanza el desarrollo.

En resumen, el proyecto del juego SOS me brindó aprendizajes valiosos en términos de desarrollo de software. Gané experiencia en la planificación, codificación, diseño, refactorización y prueba. Mi proyecto tiene una funcionalidad principal sólida, pero también hay áreas en las que podría mejorar.

1. Revisa: [Violation solution for single responsibility principle](https://softwareengineering.stackexchange.com/questions/342051/violation-solution-for-single-responsibility-principle) [↑](#footnote-ref-1)