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.

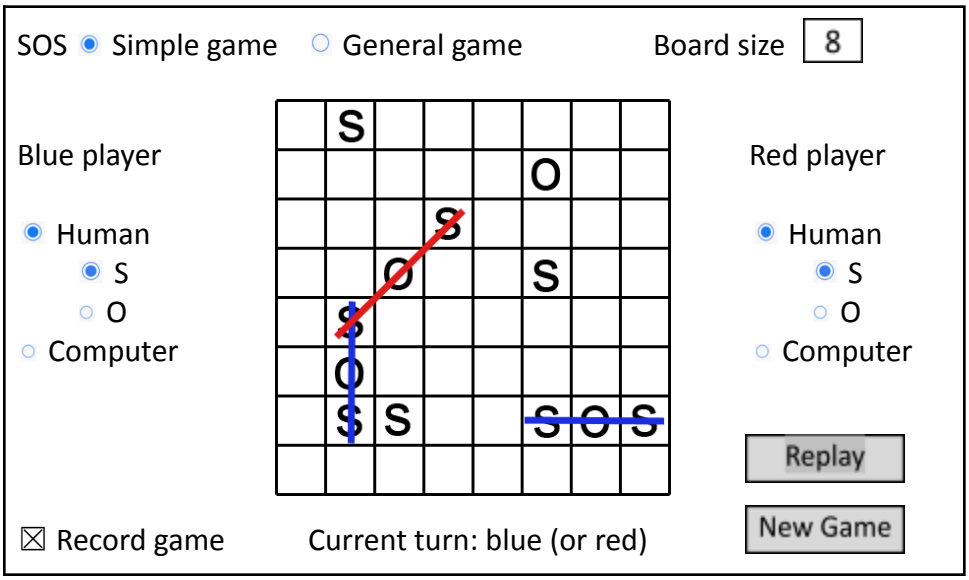

Figura 1. Sample GUI layout of the final product Diseño de GUI del producto final

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.

2. 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)
20	Grabación del último juego en un .txt	Como usuario necesito que todas las jugadas del último juego se graben en un archivo de texto.	5	2
21	Replay del último juego.	Como usuario necesito una opción que me permita ver un replay del último juego terminado.	4	3

ID y nombre de la historia de usuario	AC ID	Descripción del criterio de aceptación	Estado (completado, por hacer, en progreso)
Historia 20	20.1	AC 20.1 <descripción del escenario> Dado un juego a punto de finalizar Cuando el juego finaliza Entonces se graba un archivo de texto con la secuencia de jugadas del juego que acaba de terminar.	Completado
Historia 21	21.1	AC 21.1 <descripción del escenario> Dado un archivo de texto con el último juego guardado Cuando se va a salir de la aplicación Entonces se pregunta al jugador si quiere ver un replay del último juego, si responde sí entonces se lee el archivo **solution.txt** y se muestra el replay. Si elige que no, entonces se cierra la aplicación.	Completado

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: SOSGameBoard, SOSGameConsole

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

Checklist	Items Checklist	Conclusiones	
Estándares de codificación	Convenciones de nombres	Todos los nombres están en inglés. Las clases empiezan con letra mayúscula. Los métodos empiezan con letra minúscula y a partir de la segunda palabra se escriben con mayúscula.	
	Convención de ordenación de argumentos de método	Para la ordenación de los argumentos de método se ha decidido empezar con los ints, luego arrays de ser necesario y por último el enum Box.	
	Comentarios significativos y válidos.	Hay pocos comentarios. Sin embargo, la mayoría de los métodos de la clase **SOSGameBoard** son autoexplanatorios por los que no son necesarios. La mayoría de comentarios están el loop del gameplay, es decir en la clase main de **SOSGameConsole**, y clarifican que hace cada parte dentro del loop.	
	Estilo consistente de bloques de código	Se deja una línea de espacio entre métodos, así como el loops. En cuanto a las declaraciones de variables, se agrupa las relacionadas.	
	Indentación consistente	La indentación es consistente en todo el programa. Se aumenta la sangría en: los cuerpos de los métodos, los cuerpos de los loops(for, while), y los cuerpos de las clases.	
	…		
Principio de diseño	Clase o método no bien modularizado	El método play() de **SOSGameConsole** podría ser más corto y separarse en varios módulos.	
	Visibilidad adecuada de cada variable, método y clase.	La clase **Player** está “oculta” en la clase **SOSGameBoard**. En cuanto a las variables, siempre están bien definidas e inicializadas, además que al inicio de cada método/clase donde pertenezcan.	
	Alguna clase con pobre abstracción	Las clases presentan buena abstracción pues separan bien el comportamiento del objeto de los parámetros del mismo,	
	Diseño por contrato (pre/postcondiciones)		
	¿Se viola el Principio Abierto-Cerrado?		
	¿Se viola el Principio de Responsabilidad Única [1]?	La clase **SOSGameBoard** tiene muchas funciones. Esto es un indicador de que se podría estar violando el principio.	
Smells código	Números mágicos		
	Variable global /clase innecesaria	El vector bidimensional en el método **howManySOS()** probablemente se puede eliminar si se redefine la lógica de otra manera.	
	Código duplicado	El método **positionSOS()** y el método **howManySOS()** usan la misma lógica, solo que almacenan datos diferentes.	
	Métodos largos	El método play de la clase **SOSGameConsole** es muy largo.	
	Larga lista de parámetros		
	Expresión demasiado compleja	La función **detectSOSWhenS(), detectSOSWhenO()**	
	Switch o if-then-else que necesita ser reemplazado con polimorfismo		
	Nombre de método o variable cuya intención no está clara		
	¿Algún método similar en otras clases?		
	…		
Errores	**Fragmento de código con errores**	**¿Cuál es el error?**	**¿Por qué es un error?**

[1] Revisa: Violation solution for single responsibility principle

4. Resumen de todo el código (1 points)

Nombre del archivo de código fuente	Código de producción o prueba?	# lineas de código
SavedPlays	Producción	25
SOSGameBoard	Producción	275
SOSGameConsole	Producción	197
SOSGameGUI	Producción	678
TestSOSGameBoard	Prueba	81
TestSOSGameConsole	Prueba	43
	Total de líneas de código	1299

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?
  A manejar mejor el tiempo de entrega de proyectos, o mejor dicho las "deadlines". A empezar a idear, idear el trabajo con antelación. En el caso específico de este juego. Tener una idea clara de como quería estructurar el método **play()** de **SOSGameConsole,** me ayudó a diseñar el resto.

A estar constantemente en comunicación con tus compañeros de equipos. Es importante comunicar problemas y/o necesidades que uno requiera de otras clases a los encargados de esta, así como pedir ayuda en caso sea necesario.

- ¿Qué hace bien tu proyecto y qué podría hacer mejor tu proyecto?
  Lo que hace bien
  La lógica de como se detectan los SOSs es bastante simple y elegante. El uso de la función makePlay() en las funciones que definen un turno controlado por humano y computadora(humanPlay y computerPlay() también es un buen ejemplo de reutilización de código.

  Lo que podría hacer mejor
  Al juego en general le falta refactorización. El metodo Play() es demasiado largo y hay partes que se podrías pasar a otros métodos como la leída de inputs y la parte donde se cuentan los SOS y se cambia el turno. El juego podría ayudarse de una variable que informe del estado del juego(en curso, empate, gana jugador 1 o gana jugador 2). Así mismo la parte de la GUI tiene demasiadas clases dentro de sí misma, las cuales podrían ir en archivos separados.

- ¿Cómo podrías mejorar tu proceso de desarrollo si desarrollas un juego similar desde cero?
  Creo que ahora tengo una mejor idea de como estructurar las clases y lógica desde un principio.
  Por ejemplo, para una más sencilla implementación de algunas partes de la GUI fue necesario redefinir algunas funciones e incluso agregar clases extra.

  Otra cosa que podría mejorar es la implementación de las pruebas unitarias. En parte

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