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.

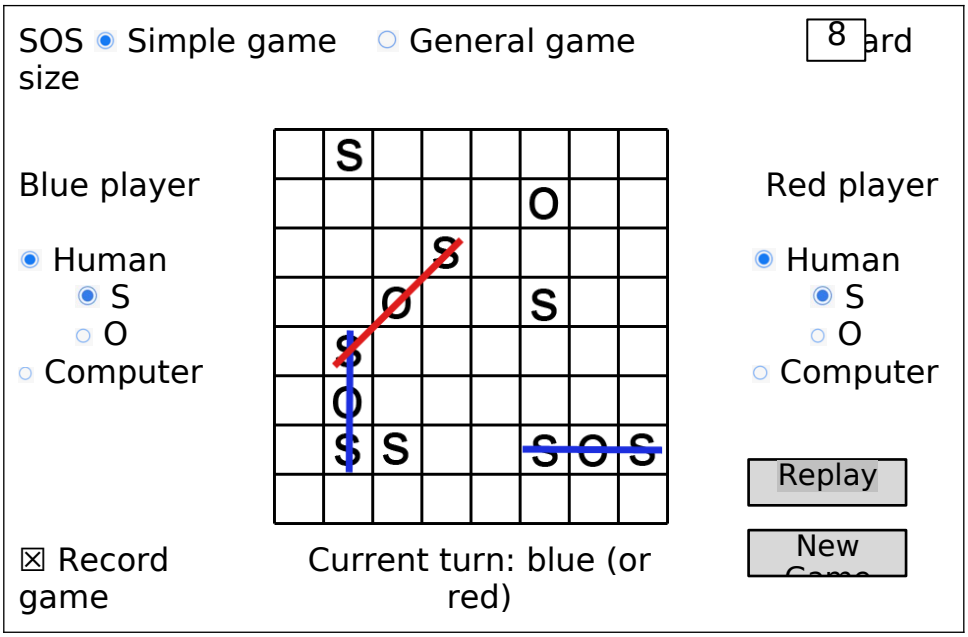


Figura 1. Sample GUI layout of the final product Diseño de GUI del producto final

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)
14	Grabar un juego	Como usuario quiero guardar en un archivo de texto todos los movimientos realizados durante un juego.	"debería tener"	4
15	Reproducir un juego	Como usuario quiero poder reproducir en el tablero el juego guardado	"podría tener"	4

ID y nombre de la historia de usuario	AC ID	Descripción del criterio de aceptación	Estado (completado, por hacer, en progreso)
14. Grabar un juego	14.1	AC 14.1 Guardado de un juego en un archivo de texto Dado que aún no comienza el juego y se ha seleccionado la opción para grabar el juego Cuando se inicie el juego Entonces cada jugada será guardad en un archivo de texto	Completado
15. Repoducir un juego	15.1	AC 15.1 Reproducciñon de un juego guardado Dado que aún no comienza el juego y existiendo un juego guardado en un archivo de texto Cuando se presione el botón reproducir Entonces se ejecutarán los movimientos del juego guardado y se mostrarán en el tablero	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: JuegoSimple, JuegoGeneral, LineaSos
Fecha/hora de duración del ejercicio de revisión del código: 03/06/2023, 04/06/2023

Checklist	Items Checklist	Conclusiones
Estándares de codificación	Convenciones de nombres	Se utilizó la guia de estilo de Java de Google (https://google.github.io/styleguide/javaguide.html) . Por lo cual los nombres de las clases usan UpperCamelCase, por ejemplo la clase JuegoSimple. Los nombres de los métodos, campos no constantes, parámetros y variables locales usan lowerCamelCase, por ejemplo el método guardarJuego. Las constantes usan UPPER_CAMEL_CASE. Además se intenta que los nombres sean significativos para dar una idea de que representan, por ejemplo el método getNumeroCeldasVacias retorna el número de celdas vacías en el tablero en el momento actual.
	Convención de ordenación de argumentos de método	No se siguió alguna convención para el ordemaniento de los argumentos de método.

	Comentarios significativos y válidos.	Se añadieron comentarios a los campos para aclarar a que representan, por ejemplo:
		`private int totalFilas; // número de filas del tablero`
		Se añadieron comentarios a los métodos para aclarar que hacen, sus entradas y salidas. Por ejemplo:
		`/**`
		`* Añade la jugada actual a la cadena de texto juegoGuardado`
		`* @param fila fila de la jugada`
		`* @param columna columna de la jugada`
		`* @param valorCelda valor de la celda de la jugada`
		`*/`
		`public void guardarJugada(int fila, int columna, Celda valorCelda) {`

	Estilo consistente de bloques de código	Se siguió la guia de estilo de Java de Google, la cual indica que para bloques no vacíos se use el estilo de Kernighan y Ritchie.
	Indentación consistente	Se siguió la guia de estilo de Java de Google, la cual indica que para indentación se usen dos espacios.

Principio de diseño	Clase o método no bien modularizado	Se observó que tanto en la clase JuegoSimple como JuegoGeneral el método setTurno:
		`public void setTurno(Turno turno) {`
		`this.turno = turno;`
		`}`
		solo se llama dentro de la expresión:
		`setTurno((getTurno() == Turno.ROJO) ? Turno.AZUL : Turno.ROJO);`
		por lo que se decidió eliminar el método setTurno y crear el método cambiarTurno:
		`public void cambiarTurno() {`
		`turno = (getTurno() == Turno.ROJO) ? Turno.AZUL : Turno.ROJO;`
		`}`
		y llamarlo desde donde antes se usaba la expresión anterior.

	Visibilidad adecuada de cada variable, método y clase.	Se cambió la visibilidad del campo juegoDebeGuardarse de protected a private y se creó el método isJuegodebeGuardarse para acceder a su valor. De esta manera todos los campos quedan privados y los métodos públicos.
	Alguna clase con pobre abstracción	No se encontró ninguna clase con pobre abstracción.
	Diseño por contrato (pre/postcondiciones)	Las precondiciones se verifican dentro de cada método, por ejemplo en las clase getCelda, que tiene los parámetros fila y columna, se verifica los valores de fila y columna dentro del método y retorna null en caso la fila o columna no esté dentro

		del rango adecuado: `public Celda getCelda(int fila, int columna) { if (fila >= 0 && fila < totalFilas && columna >= 0 && columna < totalColumnas) { return tablero[fila][columna]; } else { return null; } }`
	¿Se viola el Principio Abierto-Cerrado?	No se observó violaciones al Principio Abierto-Cerrado ya que por ejemplo si se quiere modificar el tamaño del tablero del juego, no es necesario modificar alguna linea de código que defina el tamaño del tablero sino que se debe usar el método setTablero(tamanio).
	¿Se viola el Principio de Responsabilidad Única [1]?	Se observa que la clase JuegoSimple viola el principio de responsabilidad única, ya que algunos de sus métodos como guardarJuego, guardarJugada deberían estar en una clase aparte que maneje el guardado de juegos. Por esto se creó la clase Guardado que sea responsable de estos métodos.

Smells código	Números mágicos	Se observaron números mágicos en el código, por lo que se crearon constantes para mejorar la legibilidad del código:
		`private final int TAMANIO_MINIMO_TABLERO = 2;`
		`private final int TAMANIO_MAXIMO_TABLERO = 20;`

	Variable global /clase innecesaria	En la clase JuegoSimple no se han creado variables globales.
	Código duplicado	En la clase JuegoSimple se observa que el código del método esEmpate es similar al del método getNumeroCeldasVacias y que esEmpate solo se llama una vez. Por lo que para no repetir código se decide eliminar el método es empate y reemplazar su llamado por la expresion getNumeroCeldasVacias() == 0. En la clase JuegoGeneral se observa que el método hizoSos repite casi todo el código del método hizoSos de JuegoSimple por lo que se realizó modificaciones para que JuegoGeneral use el método hizoSos de JuegoSimple.
	Métodos largos	El método hizoSos es demasiado largo por lo que se crearon dos métodos hizoSosConS e hizoSosConO que dividen la funcionalidad de hizoSos.

	Larga lista de parámetros	El método aniadirLineaSos tiene 4 parámetros que son la fila y columna de las celdas que son los extremos de la línea SOS. Para reducir el número de parámetros se pasará un arreglo de 4 elementos. Para esto se modificó el constructor de la clase LineaSos ylos llamados a aniadirLineaSos hechos desde las clases JuegoSimple y JuegoGeneral.
		Antes:
		`public void aniadirLineaSos(int col1, int fil1, int col2, int fil2) { lineasSos.add( new LineaSos(col1, fil1, col2, fil2, getTurno() == Turno.ROJO ? Color.RED : Color.BLUE)); }`
		Después:
		`public void aniadirLineaSos(int[] coord) { Color color = getTurno() == Turno.ROJO ? Color.RED : Color.BLUE; lineasSos.add(new LineaSos(coord, color)); }`

	Expresión demasiado compleja	En la clase JuegoSimple en el método realizarMovimiento se encontró que la expresión dentro del if era demasiado compleja por lo que se decidió reemplazarla por la llamada a un método. Antes: `public void realizarMovimiento(int fila, int columna, Celda valorCelda) {`

1 Revisa: [Violation solution for single responsibility principle]

		<pre>if (fila >= 0
		&& fila < totalFilas
		&& columna >= 0
		&& columna < totalColumnas
		&& getCelda(fila, columna) == Celda.VACIA) {
		setCelda(fila, columna, valorCelda);</pre> Después: <pre>if (isCeldaValida(fila, columna)) {
		setCelda(fila, columna, valorCelda);</pre> Creándose para ello el método: <pre>public boolean isCeldaValida(int fila, int columna) {
		return fila >= 0
		&& fila < totalFilas
		&& columna >= 0
		&& columna < totalColumnas
		&& getCelda(fila, columna) == Celda.VACIA;
		}</pre>	

	Switch o if-then-else que necesita ser reemplazado con polimorfismo	No se encontró algún switch o if-then-else que necesite ser reemplazado por polimorfismo	
	Nombre de método o variable cuya intención no está clara	En la clase JuegoGeneral se cambió los nombres de las variables numeroSosAzul y numeroSosRojo por puntosJugadorAzul y puntosJugadorRojo ya que facilita su entendimiento. Por la misma razón se cambiaron el nombre del parámetro tamanio a tamanioTablero en los contructores de las clases JuegoSimple y JuegoGeneral.	
	¿Algún método similar en otras clases?	Como se explicó en la sección de código duplicado se encontraron que los métodos hizoSos de la clase JuegoSimple y JuegoGeneral eran muy similares por lo que se realizaron modificaciones para eliminar dicho método de la clase JuegoGeneral y solo utilizar la versión de JuegoSimple.	
Errores	**Fragmento de código con errores**	**¿Cuál es el error?**	**¿Por qué es un error?**
	Los errores que se fueron encontrando se fueron corrigiendo pero en el momento no se encuentra ningún error.	-	-

4. Resumen de todo el código (1 points)

Nombre del archivo de código fuente	Código de producción o prueba?	# lineas de código
AutoJuego	producción	18
AutoJuegoGeneral	producción	187
AutoJuegoSimple	producción	152
Celda	producción	8
EstadoJuego	producción	8
Guardado	producción	31
JuegoGeneral	producción	77
JuegoSimple	producción	325
LineaSos	producción	42
SosGui	producción	622
TipoJugador	producción	3
Turno	producción	5
TestAutoJuegoGeneral	prueba	99
TestAutoJuegoSimple	prueba	102
TestComienzaJuego	prueba	30
TestJuegoGeneral	prueba	120
TestJuegoSimple	prueba	84

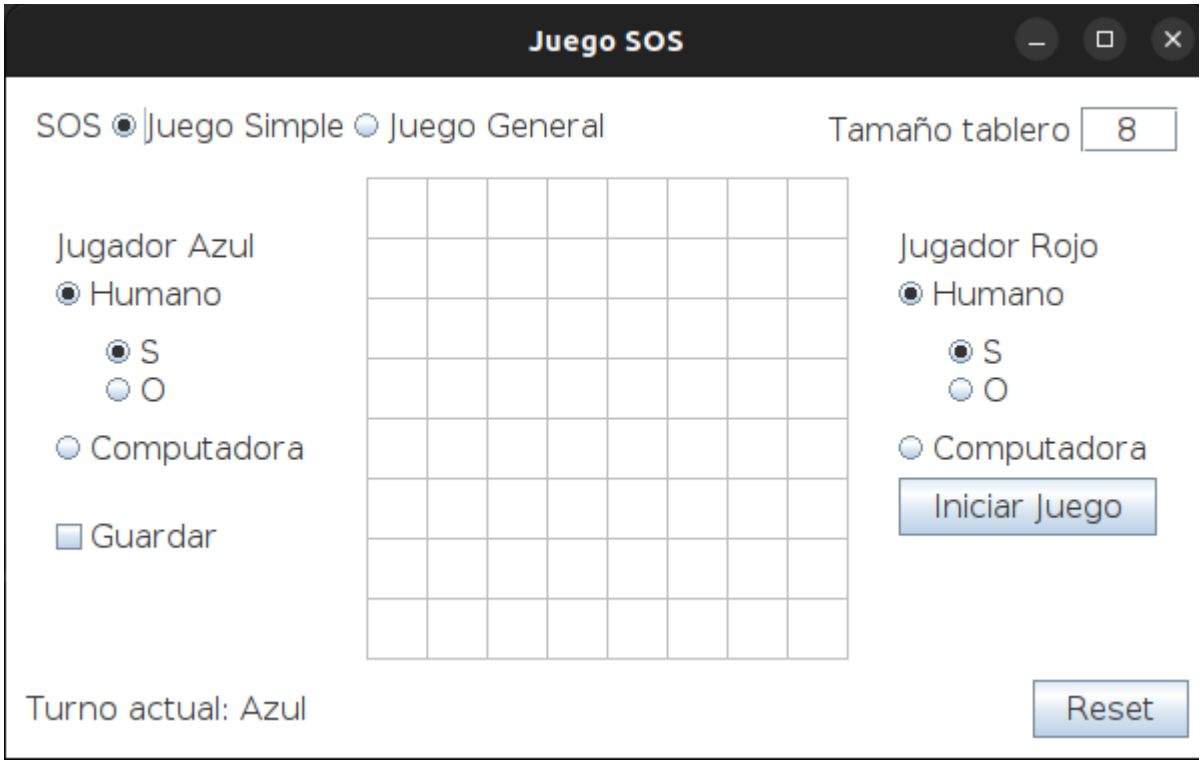
TestSeleccionaTamanio	prueba	26
TestTipoJugador	prueba	31
	prueba	82
	Total de líneas de código	2052

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.



Lecciones Aprendidas

¿Qué ganaste personalmente con el proyecto?

Aprendí a escribir código que sea más fácil de entender, usando para ello por ejemplo las guías de estilo de Java de Google, el utilizar nombres de variables que den una mejor idea de para que se usan, a colocar comentarios antes de las clases para indicar que abstraen, antes de métodos para indicar que hace el método, que argumentos requiere y cual es su salida, y antes de variables para aclarar su uso.

Aprendí a utilizar los principios SOLID como guía para escribir programas que sean más fáciles de entender, de modificar sin alterar mucho el código fuente en caso se requiera añadir alguna funcionalidad o realizar cambios en la implementación de algún método.

Practiqué la refactorización, lo cual ví que ayuda a escribir código más legible y también puede llegar a eliminar varias líneas de código repetido, lo cual facilita la comprensión del programa en general.

Pude practicar más las pruebas unitarias que había visto muy poco y que ahora veo más su importancia para comprobar cada vez que se realizan cambios que el programa sigue funcionando y no se ha introducido algún error.

También aprendí a como trabajar en sprints, en donde en cada spring se tiene una aplicación que implementa ciertas funcionalidades, dadas en un inicio a través de historias de usuario y comprobadas mediante criterios de aceptación, y luego en cada spring subsiguiente se van añadiendo nuevas funcionalidades que mejoran el proyecto.

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

El proyecto ejecuta de manera correcta el juego SOS implementado, permitiendo jugar los dos tipos de juego: simple y general, entre humanos, entre humano-computador y computador-computador en distintos tamaño de tablero. Permite resetear el juego para volver al mismo estado inicial. Permite guardar el juego en archivo de texto, el cual indica las jugadas realizadas durante la partida. Permite reproducir el juego guardado, el cual se muestra en el tablero de manera fluida de tal manera que cada jugada se realiza luego de un pequeño tiempo para simular un juego real. También se puede ver el turno actual de cada jugador durante la partida.

Lo que se podría hacer mejor es que el juego no solo guarde el último juego terminado sino que pueda guardar varios juegos y que se pueda seleccionar cual reproducir y que también permita borrar juegos guardados. También sería mejor que cualquier color de jugador pudiera iniciar ya que actualmente siempre inicia el jugador azul. Asimismo, una posible mejora sería que el juego pudiera ejecutarse desde un navegador web ya que de esa manera podría ejecutarlo cualquiera así no tenga instalado el JRE en su computadora. En ese caso, se podría implementar que los jugadores jueguen cada uno desde su computadora, ya que actualmente ambos jugadores deben jugar desde la misma computadora.

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

Si empezace a desarrollar un juego similar desde cero mi proceso de desarrollo mejoraría comparado con lo que haría si no hubiera hecho este proyecto ya que ahora sé que para organizarlo en un inicio lo mejor sería empezar realizando historias de usuario y definiendo los criterios de aceptación de dichas historias de usuario. Inclusive colocarles prioridades. Esto ayudaría mucho a estar más enfocado al momento de programar en realizar una tarea en específico y no hacerlo de manera desordenada. Adicionalmente, el realizar pruebas unitarias desde el inicio ayuda a comprobar el código a lo largo del proyecto, ya que anteriormente escribía todo el código y lo ejecutaba al final, resultando muy dificil depurar cualquier error en el programa o estar seguro de que el programa funciona para distintos valores de entrada. Finalmente, la revisión de código mejoraría el programa para que sea más facil de entender y de modificar posiblemente en un futuro.