Esta tarea integradora presenta una actividad en la cual se requiere aplicar todos los conocimientos adquiridos en la unidad 1 y la unidad 2 de nuestro curso. Por tanto, esta tarea es un instrumento para verificar el cumplimiento de los objetivos que han sido planteados para las unidades 1 y 2 descritas en el programa del curso. A continuación, encontrará un enunciado que narra de forma detallada la situación problemática que se espera usted solucione.

Enunciado

Desde la Facultad de Negocios y Economía de la Universidad Icesi, los estudiantes de Economía y Negocios Internacionales están realizando estudios sobre *Teoría de juegos*. La teoría de juegos estudia la toma de decisiones estratégicas en situaciones donde los resultados dependen de las acciones de múltiples agentes. Esta teoría, se aplica para modelar competencia, negociación, subastas y mercados, ayudando a entender comportamientos y diseñar incentivos óptimos.

Para poner en práctica los conocimientos adquiridos sobre teoría de juegos, los estudiantes de Economía y Negocios Internacionales han sido enviados a investigar en detalle varios juegos con varias condiciones, donde se pueden explorar distintas posibilidades y estrategias según el escenario, y donde también existe cierto concepto de aleatoriedad. En este proyecto, usted, como estudiante del curso Algoritmos y Programación I, diseñará y creará un software para simular uno (1) de los juegos que deben investigar los estudiantes de Economía, este software les ayudará a estudiarlos mejor y a simular distintos escenarios en cada juego.

Tenga en cuenta que el juego debe desarrollarse en Java. La interfaz de usuario del juego debe ser clara y comprensible, mostrando la información de los menús, el estado del juego y mensajes de error de manera organizada a través de la consola. Finalmente, el programa debe ejecutarse en menos de 1 segundo por turno, asegurando respuestas rápidas para una experiencia de usuario fluida.

Primera fase: Battleship 1D

La intención de esta primera fase es desarrollar un juego de Batalla Naval en una dimensión. El juego consta de dos jugadores: uno humano y uno controlado por la computadora. El objetivo del juego es hundir todos los barcos del oponente antes de que el oponente lo haga contigo. La diferencia, por ahora, con el juego de batalla naval, que se juega en 2 tableros, es que esta vez tendremos solo una porción del tablero, una fila para que cada jugador coloque sus botes.

Reglas del Juego:

- El juego comienza con los jugadores colocando sus barcos en su respectiva línea de mar.
    - La máquina los coloca de manera aleatoria.
    - Al jugador humano se le pide por consola las coordenadas para ubicarlos.
    - La fila es de 10 casillas.
- Luego, el juego alterna entre los turnos del jugador humano y el computador.
- En cada turno, un jugador realiza un disparo en el tablero del oponente y recibe una respuesta (agua, tocado o hundido).

- El juego termina cuando un jugador ha hundido todos los barcos del oponente.

Para la ubicación de los botes en la línea de mar, se deben ubicar los barcos desde su posición inicial hasta su posición final, ambas incluidas. Dicha línea de mar es una contenedora de tamaño fijo de 10 casillas, donde cada casilla puede contener uno de los siguientes valores:

- **0 (AGUA)**: Casilla vacía.
- **1 (BARCO)**: Parte de un barco. Lo ve el respectivo jugador para saber dónde ubicó sus barcos.
- **2 (TOCADO)**: Casilla donde se ha disparado un acierto. Se ve en el tablero del rival, indicando que dicha casilla ha impactado en una parte del barco.
- **3 (HUNDIDO)**: Casilla donde se ha hundido un barco. Cuando todas las casillas de un barco han dado en 2, todas cambian a 3 para indicar que dicho barco no está a flote y ha sido totalmente hundido.

Distribución de Barcos:

- 1 barco de 1 casilla (lancha).
- 1 barco de 2 casillas (barco médico).
- 1 barcos de 3 casillas (barco de munición).

Importante: Los barcos deben colocarse de manera que no se solapen.

Un jugador gana cuando en el tablero del rival se han hundido todos los barcos.

Funcionalidades y restricciones

Usted deberá diseñar y construir un programa en Java que sea capaz de simular un juego de batalla naval (battleship) entre el jugador humano (que digita las instrucciones mediante la consola de comandos) y el jugador máquina (que, recuerde, juega aleatoriamente escogiendo su coordenada a jugar al azar).

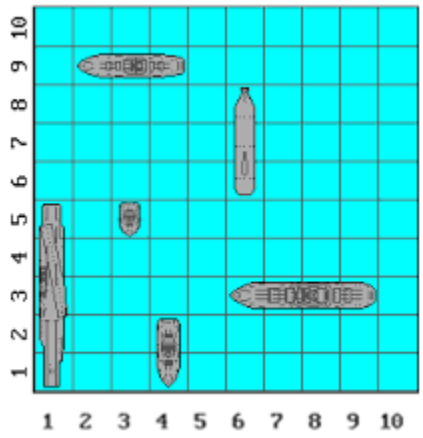
1. Permitir al jugador humano ubicar sus barcos como se muestra en el ejemplo de ejecución.
2. Realizar turno a turno:
    a. La jugada humano, pedirla por consola y modificar la línea de mar de la máquina en base a esta entrada. Verificar que la coordenada brindada sea válida, en caso de no serlo, pedirle al usuario que la vuelva a digitar.
    b. Realizar la jugada máquina aleatoria y modificar la línea de mar del jugador humano según las coordenadas.
    c. Primero juega el humano y luego la máquina, y así sucesivamente.
3. Después de cada turno siempre mostrar el estado actual de ambas líneas de mar.
    a. La línea de mar del humano siempre se mostrará en términos de 0 (agua), 1 (barco perteneciente al humano), 2 (parte tocada de barco humano) y 3 (barco totalmente hundido).
    b. La línea de mar del rival (máquina) siempre se mostrará en términos de 0 (agua), 2 (parte tocada de barco enemigo) y 3 (barco totalmente hundido).
4. Determinar cuándo y quién gana el juego. Esto sucede cuando en cualquier línea solo hay 0 (agua) y 3 (todos los barcos hundidos).
5. Permitir realizar varias iteraciones del juego en la misma ejecución del programa.

Ejemplo de ejecución

A continuación se muestra un ejemplo de varias de las interacciones que podría tener el sistema con el usuario a la hora de ejecutarse (el **#** indica mensaje del sistema y **>** indica entrada del usuario):

```
# Bienvenido a la simulación de Batalla Naval para los estudiantes de Economía de la Universidad ICESI.
# Te presentamos las siguientes opciones, ingresa:
# 1. Para jugar
# 2. Para salir del programa
> 1
# A continuación, se te pedirá que coloques tus barcos en el tablero.
# Los barcos no deben solaparse y deben ajustarse a las reglas de orientación.
# Indica las coordenadas X, Y para cada barco. Recuerda que el tablero es de 10x10.
# Coloca tu Lancha (1 casilla):
# Ingresa la coordenada de la Lancha (1-10):
> 1
# La Lancha se colocará en la casilla 1.
# Coloca el Barco Médico (2 casillas):
# Ingresa la coordenada de la primera casilla del Barco Médico (1-10):
> 3
# Ingresa la coordenada de la última casilla del Barco Médico (1-10):
> 4
# El Barco Médico se colocará en las casillas de la 3 a la 4.
# Coloca el Barco de Munición (3 casillas):
# Ingresa la coordenada de la primera casilla del Barco de Munición (1-10):
> 6
# Ingresa la coordenada de la última casilla del Barco de Munición (1-10):
> 8
# El Barco de Munición se colocará en las casillas de la 6 a la 8.
# Finalmente la disposición de tu línea de mar con tus barcos queda así:
# 1 0 1 1 0 1 1 1 0 0
# ¡Muy bien! ¡Ahora vamos a jugar!
# Dime el punto a atacar (1-10) en la línea de mar rival:
> 2
# Atacarás el punto exacto 2.
# ¡Le has dado a un objetivo enemigo!
# Línea de mar rival:
# 0 2 0 0 0 0 0 0 0 0
# El jugador máquina realiza la jugada de atacar el punto: 1.
# ¡Te han dado!
# ¡Han hundido tu Lancha!
#  Línea de mar del jugador:
# 3 0 1 1 0 1 1 1 0 0
# Dime el punto a atacar (1-10) en la línea de mar rival:
> 3
# ¡Le has dado a un objetivo enemigo!
# ¡Has hundido el Barco Médico enemigo!
# Línea de mar rival:
# 0 3 3 0 0 0 0 0 0 0
# El jugador máquina realiza la jugada de atacar el punto: 9.
# ¡No te han dado! ¡No ha pasado nada aquí!
#  Línea de mar del jugador:
# 3 0 1 1 0 1 1 1 0 0
…
```

Segunda fase: Battleship 2D



En esta segunda fase, el simulador de batalla naval se debe diseñar y construir empleando principios de *POO*, programación orientada a objetos. Al igual que la primera fase, se debe poder permitir realizar varios juegos en una misma ejecución del programa y llevar la cuenta de quién gana cuántas veces. Lo que cambia aquí, es que ya no es una línea de mar, se usarán los clásicos tableros de batalla naval y se añadirán algunas funcionalidades. Recordemos que aquí no hay empates. En el contexto identificamos algunas entidades con sus atributos, por ejemplo:

- Jugador:
  - Nombre del jugador.
  - Tablero del jugador.
  - Cantidad de partidas ganadas.
- Tablero:
  - Matriz correspondiente.
  - Barcos que hay en el agua.
- Barco:
  - Nombre del barco.
  - Lista de coordenadas que ocupa el barco.
  - Estado actual del barco (si está a flote o hundido).
- Coordenada:
  - Componente en el eje X.
  - Componente en el eje Y.

En esta fase, para la ubicación de las coordenadas en el tablero, la componente X demarca la columna (columnas de la 1 a la 10) y la componente Y demarca la fila (de la 1 a la 10 también). Dicho tablero es una matriz de **10x10** donde cada casilla puede contener la misma nomenclatura de la fase 1:

- **0 (AGUA)**: Casilla vacía.
- **1 (BARCO)**: Parte de un barco. Lo ve el respectivo jugador para saber dónde ubicó sus barcos.
- **2 (TOCADO)**: Casilla donde se ha disparado un acierto. Se ve en el tablero del rival, indicando que dicha casilla ha impactado en una parte del barco.
- **3 (HUNDIDO)**: Casilla donde se ha hundido un barco. Cuando todas las casillas de un barco han dado en 2, todas cambian a 3 para indicar que dicho barco no está a flote y ha sido totalmente hundido.

También, adicionalmente se tendrán 2 tipos de partida: partida estándar y partida personalizada. En la partida estándar se tiene una cantidad de barcos fija y la longitud de dichos barcos también ya está predeterminada. En una partida personalizada, el jugador humano elige las condiciones de los barcos (cuántos barcos serán) y cada uno de qué longitud será, además de qué sentido tendría (si se ubica de manera horizontal o vertical).

Partida estándar

Distribución de Barcos:

- 1 barco de 1 casilla (lancha).
- 1 barco de 2 casillas (barco médico), que se ubica de forma vertical
- 2 barcos de 3 casillas
    - 1 debe ser colocado de manera horizontal (barco de provisiones)
    - 1 debe ser colocado de manera vertical (barco de munición)
- 1 barco de 4 casillas (buque de guerra), que se ubica de forma horizontal.
- 1 barco de 5 casillas (portaaviones), que se ubica de forma vertical.

Partida personalizada

En este tipo de partida, el jugador humano elige:

- Cuántos barcos habrá en los tableros (máximo 10 barcos).
- Cada barco, qué longitud (cuántas casillas ocupa) tiene (máximo 5 casillas).
- La orientación de ubicación del barco (si es vertical u horizontal).

Funcionalidades y restricciones

Los requerimientos funcionales son similares a los de la primera fase, solo aplicando el mismo concepto a los tableros humano y tablero máquina, e igualmente, ampliando el concepto de partida estándar y personalizada.

Debe diseñar y construir un programa en Java que sea capaz de simular un juego de batalla naval (battleship) utilizando POO, entre el jugador humano (que digita las instrucciones mediante la consola de comandos) y el jugador máquina (que, recuerde, juega aleatoriamente escogiendo su coordenada a hundir al azar).

1. Permitir al jugador humano ubicar sus barcos como se muestra en el ejemplo de ejecución.
2. Realizar turno a turno:
    a. La jugada humano, pedirla por consola y modificar el tablero de la máquina en base a esta entrada. Verificar que la coordenada brindada sea válida, en caso de no serlo, pedirle al usuario que la vuelva a digitar.
    b. Realizar la jugada máquina aleatoria y modificar el tablero del jugador humano según las coordenadas.
    c. Primero juega el humano y luego la máquina, y así sucesivamente.

3. Después de cada turno siempre mostrar el estado actual de ambos tableros.

- El tablero humano siempre se mostrará en términos de 0 (agua), 1 (barco perteneciente al humano), 2 (parte tocada de barco humano) y 3 (barco totalmente hundido).
- El tablero del rival (máquina) siempre se mostrará en términos de 0 (agua), 2 (parte tocada de barco enemigo) y 3 (barco totalmente hundido).

4. Determinar cuándo y quién gana el juego. Esto sucede cuando en cualquier tablero solo hay 0 (agua) y 3 (todos los barcos hundidos).
5. Permitir realizar varias iteraciones del juego en la misma ejecución del programa.
6. Tener un recuento de las veces que ha ganado el jugador y las veces que ha ganado la máquina en una ejecución del programa. Este recuento debe ser comunicado cada vez que un juego termina. También debe existir una opción en el menú para consultarlo cuando se está en el menú principal, fuera de un juego en curso. Tenga en cuenta que se deben presentar estas estadísticas por cada tipo de partida: estándar y personalizada.
7. Permitir los dos modos de juego: partida estándar y partida personalizada.

Ejemplo de ejecución

A continuación se muestra un ejemplo de varias de las interacciones que podría tener el sistema con el usuario a la hora de ejecutarse (el **#** indica mensaje del sistema y **>** indica entrada del usuario):

```
# Bienvenido a la simulación de Batalla Naval para los estudiantes de Economía de la Universidad ICESI.
# Te presentamos las siguientes opciones, ingresa:
# 1. Para jugar
# 2. Para conocer cuántas veces ha ganado el jugador y cuántas veces la máquina
# 3. Para salir del programa
> 1
# A continuación, se te pedirá que coloques tus barcos en el tablero.
# Los barcos no deben solaparse y deben ajustarse a las reglas de orientación.
# Indica las coordenadas X, Y para cada barco. Recuerda que el tablero es de 10x10.
# Coloca tu Lancha (1 casilla):
# Ingresa la coordenada X de la Lancha (1-10):
> 3
# Ingresa la coordenada Y de la Lancha (1-10):
> 5
# Coloca el Barco Médico (2 casillas, vertical):
# Ingresa la coordenada X de la primera casilla del Barco Médico (1-10):
> 7
# Ingresa la coordenada Y de la primera casilla del Barco Médico (1-10):
> 2
# El Barco Médico se colocará verticalmente en las coordenadas (7, 2) y (7, 3).
# Coloca el Barco de Provisiones (3 casillas, horizontal):
# Ingresa la coordenada X de la primera casilla del Barco de Provisiones (1-10):
> 4
# Ingresa la coordenada Y de la primera casilla del Barco de Provisiones (1-10):
> 8
# El Barco de Provisiones se colocará horizontalmente en las coordenadas (4, 8), (5, 8) y (6, 8).
# Coloca el Barco de Munición (3 casillas, vertical):
# Ingresa la coordenada X de la primera casilla del Barco de Munición (1-10):
> 9
# Ingresa la coordenada Y de la primera casilla del Barco de Munición (1-10):
> 4
# El Barco de Munición se colocará verticalmente en las coordenadas (9, 4), (9, 5) y (9, 6).
# Coloca el Buque de Guerra (4 casillas, horizontal):
# Ingresa la coordenada X de la primera casilla del Buque de Guerra (1-10):
> 1
# Ingresa la coordenada Y de la primera casilla del Buque de Guerra (1-10):
> 9
```

```
# El Buque de Guerra se colocará horizontalmente en las coordenadas (1, 9), (2, 9), (3, 9) y (4, 9).
# Coloca el Portaaviones (5 casillas, vertical):
# Ingresa la coordenada X de la primera casilla del Portaaviones (1-10):
> 6
# Ingresa la coordenada Y de la primera casilla del Portaaviones (1-10):
> 1
# El Portaaviones se colocará verticalmente en las coordenadas (6, 1), (6, 2), (6, 3), (6, 4) y (6, 5).
# Finalmente la disposición de tu tablero con tus barcos queda así:
# 0 0 0 0 0 1 0 0 0 0
# 0 0 0 0 0 1 1 0 0 0
# 0 0 0 0 0 1 1 0 0 0
# 0 0 0 0 0 1 0 0 1 0
# 0 0 1 0 0 1 0 0 1 0
# 0 0 0 0 0 0 0 0 1 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 1 1 1 0 0 0 0
# 1 1 1 1 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# ¡Muy bien! ¡Ahora vamos a jugar!
# Dime la coordenada X a atacar en el tablero rival:
> 2
# Dime la coordenada Y a atacar en el tablero rival:
> 3
# Atacarás el punto exacto de fila 3, columna 2.
# ¡Le has dado a un objetivo enemigo!
# Tablero rival:
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 2 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# El jugador máquina realiza la jugada X: 7, Y: 3
# ¡Te han dado!
#  Tablero del jugador:
# 0 0 0 0 0 1 0 0 0 0
# 0 0 0 0 0 1 1 0 0 0
# 0 0 0 0 0 1 2 0 0 0
# 0 0 0 0 0 1 0 0 1 0
# 0 0 1 0 0 1 0 0 1 0
# 0 0 0 0 0 0 0 0 1 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 1 1 1 0 0 0 0
# 1 1 1 1 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# Dime la coordenada X a atacar en el tablero rival:
> 7
# Dime la coordenada Y a atacar en el tablero rival:
> 5
```

```
# Atacarás el punto exacto de fila 5, columna 7.
# ¡No hay nada allí!
Tablero rival:
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 2 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
# El jugador máquina realiza la jugada X: 3, Y: 5
# ¡Te han dado! ¡Y parece que han hundido un barco!
Tablero del jugador:
# 0 0 0 0 0 1 0 0 0 0
# 0 0 0 0 0 1 1 0 0 0
# 0 0 0 0 0 1 2 0 0 0
# 0 0 0 0 0 1 0 0 1 0
# 0 0 3 0 0 1 0 0 1 0
# 0 0 0 0 0 0 0 0 1 0
# 0 0 0 0 0 0 0 0 0 0
# 0 0 0 1 1 1 0 0 0 0
# 1 1 1 1 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0
…
```

Entregables

La entrega de la tarea integradora se encontrará dividida en dos partes:

Primera entrega: Hasta el sábado 1 de marzo de 2025 a las 21:59 (8 de marzo grupos IEI y DMI).

Lleve a cabo las siguientes actividades de cada una de las etapas de desarrollo de software:

1. **Análisis del problema.** Identificación del problema y análisis de requerimientos para satisfacer la Primera Fase de implementación del aplicativo (en el formato visto en la clase de Ingeniería de Software I, descárguelo aquí).
2. **Implementación en Java**. Incluya en la implementación, los comentarios descriptivos sobre el código java escrito por usted.
3. **Documentación.** Se deben incluir los contratos de los métodos implementados: descripción, parámetros y retorno.
4. Usar **GitHub como repositorio de código fuente y documentación**. Recuerde que se debe evidenciar su avance a través de los días en el desarrollo de su tarea.

Recuerde que puede encontrar la Rúbrica de la tarea integradora en el siguiente enlace.

Segunda entrega: Hasta el sábado 5 de abril de 2025 a las 21:59 (12 de abril grupos IEI y DMI).

1. Análisis del problema:
    a. **Identificación del problema y análisis de requerimientos actualizado para satisfacer la Segunda Fase** de implementación del aplicativo (en el formato visto en la clase de Ingeniería de Software I, descárguelo aquí).
    b. **Tabla de trazabilidad entre el análisis y el diseño** (Enlace).
2. Diseño de la solución.
    - Elabore un **diagrama de clases** que modele la solución del problema de acuerdo con las buenas prácticas y los patrones de diseño revisados hasta el momento en el curso. Su diagrama debe incluir el paquete modelo y el de interfaz de usuario.
3. **Implementación en Java**. Incluya en la implementación, los comentarios descriptivos sobre los atributos y métodos de cada clase.
4. **Documentación en JavaDoc** (Debe entregarse el JavaDoc generado y ubicarlo en la carpeta doc). **Incluya los contratos de los métodos implementados: descripción, parámetros y retorno.**
5. Usar **GitHub como repositorio de código fuente y documentación** utilizando la estructura de carpetas aprendida en clase. Recuerde que se debe evidenciar su avance a través de los días en el desarrollo de su tarea.

Nota:

- Tenga en cuenta que su repositorio GitHub debe presentar una estructura base como por ejemplo:

  Battleship/
  src/
  bin/
  doc/

- Dentro de los directorios **src/** y **bin/** estarán presentes estos directorios(representando cada uno de sus paquetes):

  ui/
  model/

- El directorio src (source code) contiene sus clases .java dentro del directorio ui/ y model/. Por otro lado, el directorio bin (binary files) contiene los archivos .class en el directorio ui/ y model/. El directorio doc tendrá toda la documentación de análisis y diseño
- Su código debería compilar de acuerdo con lo explicado en la diapositiva 15 de esta presentación: http://tinyurl.com/y3bd9bg2

Anexo

Manejo de Random

Se recomienda utilizar la clase Random (ubicada en java.util) para generar valores aleatorios en Java.

Para usar los métodos de esta clase, primero debe importarlo desde el paquete:

```
import java.util.Random;
```

Para obtener una instancia de Random:

Random randomGenerator = new Random();

A partir de la instancia de Random, se pueden generar valores aleatorios de distintos tipos:

- **Número entero aleatorio en un rango específico**

  int numeroAleatorio = randomGenerator.nextInt(100); // Número entre 0 y 99

- **Número decimal aleatorio entre 0.0 y 1.0**

  ```
  double numeroDecimal = randomGenerator.nextDouble();
  ```

- **Número entero aleatorio en un rango definido [min, max]**

  ```
  int min = 5, max = 15;
  int numeroEnRango = randomGenerator.nextInt(max - min + 1) + min;
  ```