PROXECTO

HUNDIR A FLOTA

Luca Mascani Opazo

1. O proxecto e o porqué.
2. Tecnoloxías e Ferramentas empregadas.
3. Deseño e explicación do código.
4. Diagrama UML.
5. Probas do correcto funcionamento.
6. Conclusión do proxecto.

1.O Proxecto e o Porqué:

O selecionado código foi obtido da plataforma GitHub (a ligazón específica encontraráse na propia biografía).

O ocio sempre foi unha necesidade intrínseca dentro das propias do ser humano, por eso os xogos sempre han representado un apartado mais dentro de algúns ámbitos, sendo a programación un dos máis, por non decir o máis empregado na era moderna, onde a maioría das interacións que levamos a cabo e mediante unha computadora. O seguinte xogo é o coñecido e normalmente adaptado a formato tradicional “Hundir a Flota”. Un tradicional pero entretido xogo basado na intuición fronte a fronte (neste caso contra a propia computadora), no que debes atinar a posición de diferentes barcos que o contrario tratará de ocultar.

É un xogo adaptado para todas as idades e entornos, sendo un bo metodo de ensinar a orientación espacial para os nenos pequenos e ao mismo tempo unha maneira orixinal de manter entretido a todo tipo de individuos.

A orixe exacta do xogo “Hundir a Flota” non é preciso, pero creese que comezou como un xogo de guerra naval no século XVIII. A comezos do XX, na década do 1930 popularizose a sua versión como xogo de mesa, lanzándose así diferentes versions ao largo dos anos, con variados deseños e variacións nas reglas, pero o concepto básico de hundir os barcos do oponente permanece constante.

2.Tecnoloxías e Ferramentas empregadas:

- **Java**: Escollín Java como lenguaxe de programación, aparte de porque é o que estivemos estudando no presente curso, por una serie de características que o fan óptimo, entre elas a portabilidade que presenta entre diferentes sistemas operativos (Java ten a súa propia maquina virtual, jvm, coa que non e complicado pasalo de un sistema a outro sen modificadores), a súa amplia gama de bibliotecas, o seu rendemento, e outros factores máis complexos que fan de Java unha ferramenta óptima para desenvolver iste xogo.

- **IntelliJ IDEA Community Version**: Elixin IntelliJ IDEA como compilador debido ás súas ferramentas de edición, depuración e refactorización de código, ademais do seu soporte para Java e JavaScript. IntelliJ IDEA facilita o traballo en equipo e a colaboración entre desarrolladores, entre outros factores a maiores.

- **GitHub**: GitHub proporciona un control detallado do código fonte, favorece a colaboración entre desenvolvedores e fomenta a transparencia no desenvolvemento da aplicación. Empregouse GitHub como sistema de control de versións para xestionar o código fonte da aplicación. GitHub permite rastrexar as modificacións, colaborar con outros desenvolvedores e compartir o código coa comunidade.

3. Deseño e explicación do código:

O código básease nun diseño simple e pragmático, o cal vou explicar a continuación:

```
public static String[] reglas= new String[11];
public static String[][] tablero= new String[11][11];
public static ArrayList<String> ocupadas=new ArrayList<String>();
public static ArrayList<String> repetidas=new ArrayList<String>();
public static ArrayList<String> posfinal=new ArrayList<String>();
public static String user;
public static int posx;
public static int posy;
```

Comezando polas variables: “**string reglas**”, coa que se define as reglas do xogo para o usuario, contando cun total de 11 reglas. “**string tablero**” no que se define o taboleiro de xogo, o cal conta con 11*11 xanelas.
“**ArrayList ocupadas**” configura unha lista de posiciones ocupadas por barcos, “**ArrayList repetidas**” recolle as posicións repetidas dos barcos (superposición), “**ArrayList posfinal**” recolle as posicións finais dos barcos. “**String user**” é a entrada do usuario. “**int posx posy**” recolle as posicións introducidas polo usuario mediante un sistema coordenado: x,y.

- **Método principal:**

O programa segue o seguinte orde lóxico: Primeiro crea unha lista de barcos para ser enchida, despois inicializa o teclado para empregalo como metodo de entrada para o usuario, unha variable para contar os intentos e chama métodos para enchir as reglas, e inicializar e mostrar o taboleiro.

Posteriormente xenera barcos de diferentes tamaños dentro duns parámetros establecidos en (barcos.java) e os engade á lista anteriormente definida.

A lóxica para o correcto funcionamento do xogo consta dun bucle “do-while”, que remata cando o usuario completa 25 intentos ou cando todos os barcos

son afundidos: Iste bucle consta de tres pasos resumidos en: 1. Pedir ao usuario unha posición (x,y). 2. Verificar e actualizar o estado desa xanela do taboleiro (auga, tocado, ou afundido)..
3. Mostrar o novo estado do taboleiro tras o último intento.

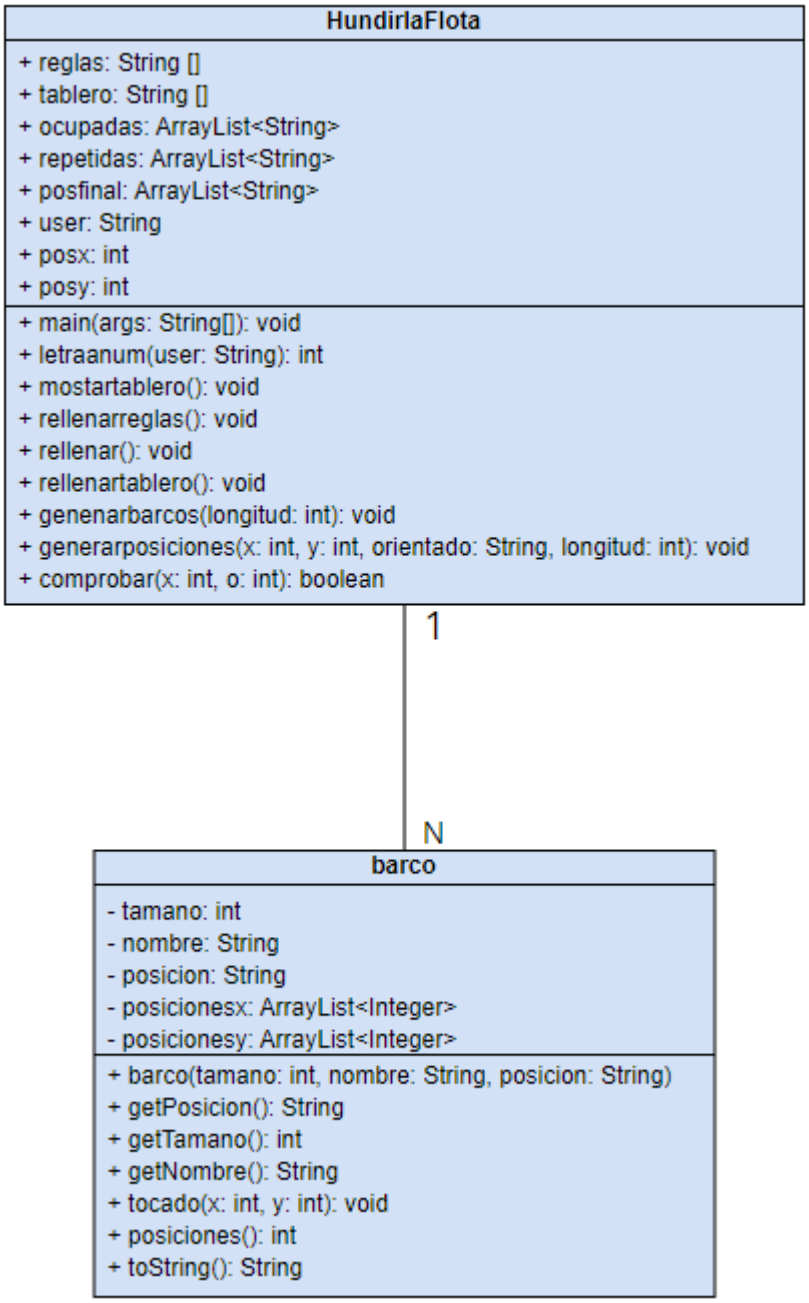
- **Métodos Auxiliares:**

- **“letraanum(String user)”:** Transforma a letra introducida polo usuario para marcar a fila nun número para o seu cómodo procesamento.
- **“mostrartablero()”:** Mostra o taboleiro na consola.
- **“rellenarreglas()”:** Enche as reglas do xogo.
- **“rellenar()”:** Inicializa o taboleiro e representalo con “*”
- **“rellenartablero()”:** Engade números e letras ao taboleiro.
- **“generarbarcos(int longitud)”:** Xenera unha posición e unha orientación para os barcos.
- **“generarposiciones(int x, int y, String orientado, int longitud)”:** Calcula e verifica as posicións para os barcos.
- **“comprobar(int x, int o):** Verifica se unha posición xa está ocupada.

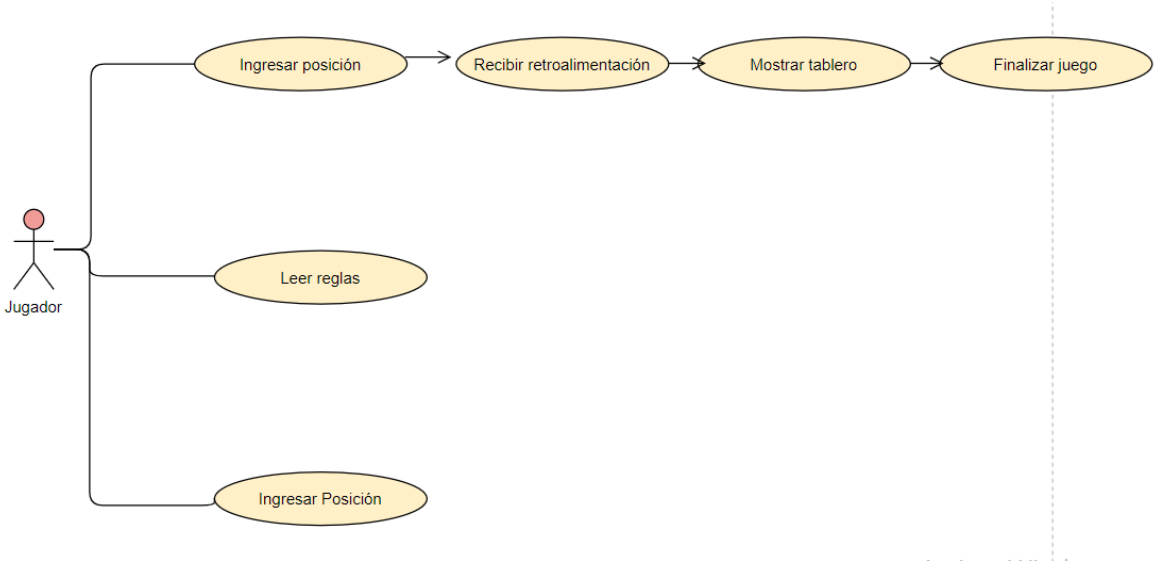
Outros datos acerca do deseño do programa e o código son os barcos que poden ser xerados (dous acorazados = 3 espazos), (tres buques = 2 espazos), (catro submarinos = 1 espazo), (e un portaavións = 4 espazos) (estes parámetros veñen definidos no archivo *barcos.java*.
O taboleiro será enchido con tres diferentes símbolos dependendo do estado de cada xanela: Auga = ~, Tocado = +, Afundido = X.

4.Diagrama UML:

Un diagrama UML (Unified Modeling Language) é unha representación gráfica que describe a estrutura e comportamento dun sistema de software. Neste caso o videoxogo de Hundir a Flota. Facilita a visualización a nivel global dos componentes e a súa interacción. Aquí un diagrama de clases:


HundirlaFlota
+ reglas: String []
+ tablero: String []
+ ocupadas: ArrayList<String>
+ repetidas: ArrayList<String>
+ posfinal: ArrayList<String>
+ user: String
+ posx: int
+ posy: int
+ main(args: String[]): void
+ letraanum(user: String): int
+ mostartablero(): void
+ rellenarreglas(): void
+ rellenar(): void
+ rellenartablero(): void
+ genenarbarcos(longitud: int): void
+ generarposiciones(x: int, y: int, orientado: String, longitud: int): void
+ comprobar(x: int, o: int): boolean
1
N
barco
- tamano: int
- nombre: String
- posicion: String
- posicionesx: ArrayList<Integer>
- posicionesy: ArrayList<Integer>
+ barco(tamano: int, nombre: String, posicion: String)
+ getPosicion(): String
+ getTamano(): int
+ getNombre(): String
+ tocado(x: int, y: int): void
+ posiciones(): int
+ toString(): String

Empregar un diagrama de casos de uso para “Hundir la Flota” proporciona unha forma clara e organizada de entender e comunicar os requisitos e funcionalidades do xogo. Facilita o desarrollo e as probas a desglosar o sistema en interaccións específicas e manejables.



O actor nos 3 casos de uso e o “Jugador”.

5.Probas do correcto funcionamento:

Son indispensables para o correcto funcionamento dun programa as súas pertinentes probas de funcionamento, entre as cales, para este xogo podemos empregar as seguintes:

- **Inicialización das reglas:** Chamando a “rellenarreglas()” e comprobando que “reglas” conten as 11 entradas esperadas. Isto ten como obxetivo verificar que as reglas do xogo inicianse correctamente

- **Inicialización do taboleiro:** Chamando a “rellenar()” e “rellenartablero()”, e verificando que “tablero”, e dicir o taboleiro ten o formato esperado, o obxectivo dista proba é asegurar o correcto funcionamento do taboleiro, enchéndose con asteriscos e números/letras.

- **Xeración correcta dos barcos:** Chamando o método “generarbarcos()” varias veces e comprobando que as posicións chamadas non se sobrepoñen e están dentro dos límites do taboleiro.

- **Correcta conversión das coordenadas:** Chamando a “letraanum()” con todas as letras válidas (despóis da J- non é valido, calquera valor non válido pechara o programa) e observando como devolve os valores esperados (conversión letra-número).

- **Correcta entrada do usuario:** Comprobar que os inputs introducidos polo usuario son válidos e que o taboleiro actualízase correctamente.

- **Proba lóxica de golpear un barco:** Simular golpes en posicions coñecidas de barcos e verificar que o estado de “tocado”-“hundido” actualízase correctamente no taboleiro.

- **Proba lóxica de hundir un barco:** Simular golpes en todas as posicións onde encóntrase un barco e verificar que o barco elimínase da lista e que o taboleiro mostra unha X nas posicións onde atópase iste barco.

- **Probas de finalización:** Comprobar como o bucle do-while rómpese en calquera dos dous casos previamente aceptados: No intento 26 (tras 25 intentos) ou cando todos os barcos son hundidos.

6. Conclusión do proxecto:

O xogo é simple e pouco atrativo a nivel gráfico, comparado coa gran cantidad de xogos contemporáneos, de feito se o trasladamos a calquera interfaz gráfica perde moito atrativo visual (se acaso é que o tiña).

Sen embargo elexín este programa de internet, precisamente de GitHub (GITHUB HUNDIR LA FLOTA) para aforrarme a labor de progamar todo e cometer erros innecesarios, e polo mesmo motivo, non adxunto as probas de funcionamento porque foron todas positivas, e dicir, non presenta ningún error involuntario.

Aínda así o usuario podería disfrutar máis do xogo se se implementara unha funcionalidad 1v1 en lugar de un 1vPC, sendo de todas formas moi primitivo o multixogador nun mesmo aparato.