

Universidad Técnica Nacional

Ingeniería de Software

Curso

Programación I

ISW-211

Proyecto 1

Responsable

Jose Elías Benavides Varela

Correo: [jsbenavidesv@est.utn.ac.cr](mailto:jsbenavidesv@est.utn.ac.cr)

Fecha de entrega 12 de julio de 2014

Contenido

[1 Objetivo general 3](#_Toc392893084)

[2 Objetivo específico 3](#_Toc392893085)

[3 Descripción del problema 4](#_Toc392893086)

[Juego Aliados vs Enemigos 4](#_Toc392893087)

[4 Solución del problema 5](#_Toc392893088)

[5 Análisis de resultados 7](#_Toc392893089)

[6 Conclusiones y recomendaciones 7](#_Toc392893090)

# Objetivo general

Crear soluciones para resolver problemas simples mediante el modelaje, diseño detallado y programación, aplicando técnicas actuales de desarrollo de software orientado a objetos y considerando criterios de calidad apropiados, mediante la utilización de la notación UML.

# Objetivo específico

Resolver problemas mediante el uso de modelos abstractos, para la formación de un esquema lógico conceptual.

Fecha de inicio y Fecha última modificación. Inicio 24 de junio de 2014 Finalización 12 de Julio de 2014

# Descripción del problema

## Juego Aliados vs Enemigos

El proyecto consiste en desarrollar un algoritmo que permita inicializar un juego. Para ello debe pedir los siguientes datos al usuario en que se indiquen la información necesaria.

1. Seleccionar el nombre del jugador número 1.
2. Seleccionar el nombre del jugador número 2.
3. Seleccionar la cantidad elementos a participar dentro la de la partida (mínimo 1 elementos por aliado y mínimo 1 elementos por enemigos).
4. Seleccionar la profundidad del área de juego (Largo y ancho)

Dentro de los métodos que se deberán implementar son:

1. AcertarBlancoAliado();
2. FallarBlancoAliado();
3. AcertarrBlancoEnemigo();
4. FallarBlancoEnemigo();
5. SumarAliadosDestruidos();
6. SumarEnemigosDestruidos();
7. EscogerJugadorAliado();
8. EscogerJugadorEnemigo();
9. EscogerCeldaAliado();
10. EscogerCeldaEnemigo();
11. Ganador();
12. Perdedor();
13. Turno();
14. CantidadDePartidasJugadas();
15. CantidadDePartidasGanadasJugador1();
16. CantidadDePartidasGanadasJugador2();
17. CantidadDePartidasPerdidasJugador1();
18. CantidadDePartidasPerdidasJugador2();
19. Rendirse();
20. TotalDeAliadosDestruidos();
21. TotalDeEnemigosDestruidos();
22. Jugar();
23. IniciarNuevaPartida();
24. VerEstadisticas();

El algoritmo debe crear las posiciones en donde se colocarán los objetivos de manera aleatoria y de forma que no se repita cualquier posición.

Con los métodos descritos el programa desarrollado debe ser capaz de crear el área de los jugadores y seguir el algoritmo para que se identifique cual será el ganador y el perdedor de cada partida según se cumplan los criterios del juego.

Habrá un ganador si:

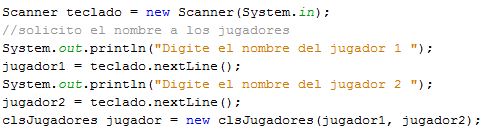
* El oponente se rinde
* Si el jugador acierta con todos los objetivos rivales antes que el contrincante

De la misma forma el programa debe acarrear las estadísticas de cada partida y estar disponibles cuando el usuario las solicite.

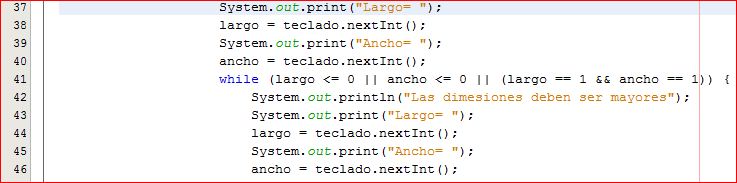
# Solución del problema

Para la resolución del problema se siguió con el siguiente protocolo:

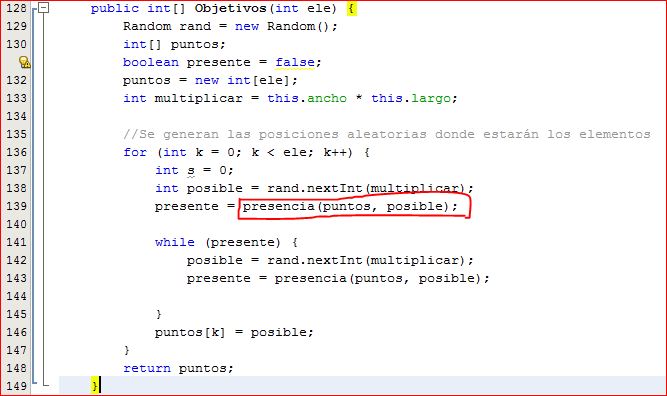
1. Inicialmente se solicita al usuario el nombre de los jugadores y se ingresan por medio del método nextLine()

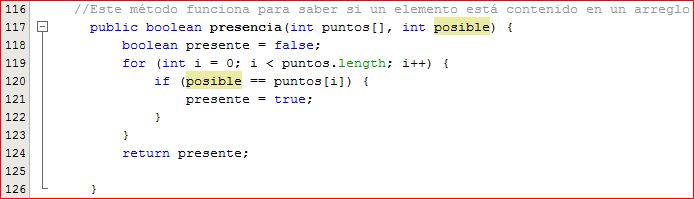


1. Posteriormente se crea el área de juego con las dimensiones que el mismo usuario desea y con la cantidad de objetivos con los que quiere jugar. Debe tenerse en cuenta verificar que el número de objetivos sea menor que el número de espacios posibles



1. Se elabora un arreglo para guardar las posiciones de los objetivos. Se va llenando cada una de sus posiciones hasta que se logre el número de elementos solicitado por el usuario. Al mismo tiempo se verifica que el número generado aleatoriamente no se encuentre en el arreglo para así no repetirlo





1. Se define el bando de cada jugador ya sea aliado o enemigo. Una vez definido el área de juego y el bando se procede a solicitar al jugador si desea: rendirse, atacar alguna posición del oponente, reiniciar otra partida o ver las estadísticas.
   1. Si el jugador decide rendirse: se identifica como perdedor y se le muestra un nuevo menú para que decida si quiere cambiar de jugador o reiniciar la partida
   2. Atacar una posición del oponente: se recorre el área del oponente para revisar si el jugador acertó o falló con la posición que eligió.
   3. Reiniciar una partida: se inicia con una nueva partida
   4. Ver estadísticas: se muestra la cantidad de intentos y el número de aciertos de cada jugador
2. Se van sumando los intentos, tanto los aciertos como los fallos y se direcciona a ejecutar la sección del código correspondiente a cada jugador. Esto es, por ejemplo, si el oponente aliado acertó, se le suma un acierto en las estadísticas y le corresponderá el turno de elegir de nuevo. Si por el contrario, falló en su intento, le corresponderá al jugador oponente la elección de la posición a atacar, rendirse, iniciar una nueva partida o ver las estadísticas.
3. Cada vez que el jugador acierte se confrontará que el número de aciertos sea menor al número de objetivos para verificar si el jugador ha ganado la partida.

Se pensó desarrollar el problema por medio de un algoritmo capaz de inter relacionar los métodos de selección de posición a atacar entre los aliados y los enemigos. Se verifica que la información solicitada cumpla con los requerimientos del programa.

# Análisis de resultados

Después de ejecutar el algoritmo se puede evidenciar un buen funcionamiento, acorde con los requerimientos propuestos. Se puede comprobar que no existen validaciones con las entradas de los datos por medio del Scanner, esto queda evidenciado cuando se trata de ingresar un string en un dato numérico como el largo o ancho del área de juego.

No se realizaron los métodos de las estadísticas totales: TotalDeAliadosDestruidos(); TotalDeEnemigosDestruidos(); ya que considero resultan de más si se toma en cuenta que ya se tienen los métodos SumarAliadosDestruidos(); SumarEnemigosDestruidos().Es importante conocer algunas técnicas para evitar código sobrante e implementar maneras de reutilización de código.

# Conclusiones y recomendaciones

La realización del proyecto sirvió para:

* Comprender con mayor profundad el funcionamiento de un lenguaje orientado a objetos: esto se evidenció en la necesidad de crear varios métodos que modificaran los atributos de la clase jugar en este caso
* Desarrollar un mapa lógico para tratar de reducir la cantidad de código y lograr al mismo tiempo cumplir con los objetivos propuestos
* Aplicar los conceptos desarrollados en clase como matrices, arreglos, string entre otros
* Entender la importancia de clarificar el problema a desarrollar antes de comenzar a escribir código

Recomendaciones

* Detallar con mayor profundidad los enunciados del proyecto. De manera que facilite al estudiante la comprensión del mismo
* Definir un tiempo de la clase para que los estudiantes generen consultas sobre el proyecto. Esto porque no todos los estudiantes pueden asistir al horario de consulta del profesor