Programación II

Proyecto final: BATTLESHIP

Sección: 3.03

Nombres y apellidos	Teoría	Participacion
Pablo Eduardo Ghezzi Barton	3	100%
Flavia Ailen Mañuico Quequejana	3	100%
Yamileth Rincón Tejeda	3	100%
Rancé Blondet Borja	3	100%

Profesor: *Contreras Chavez Estanislao*

10/12/2021

LIMA-PERÚ

Índice de contenidos

Antecedentes

El proyecto es crear un programa de c + + capaz de jugar battleship de forma independiente y capaz.

Nuestro proyecto debe:

- Interactuar con los .in, .out y .not files para comunicar con el programa
- Crear dos tableros
    - En el primer tablero, el equipo coloca su flota y registra los tiros del oponente.
    - En el segundo tablero se registran los tiros propios contra el otro equipo, diferenciando los impactos y los que dan al agua. Con esta información se deduce la posición de los barcos del contrincante.
- Crear una flota y posicionarnos
    - Debe contener
        - 1 x Aircraft Carrier (A) → 4 casillas contiguas
        - 2 x Battlecrucier (B) → 3 casillas contiguas
        - 3 x Submarine (S) → 2 casillas contiguas
        - 4 x Torpedo boat (T) → 1 casilla
- Poder detectar ganar o perdido el juego
- Que funciona prácticamente de forma automática, independiente del creador
- El programa debe ser capaz de jugar battleship bien (no puede ser los ataques random)

Este reporte contiene: Fundamento Teórico, Metodología y Desarrollo, Conclusiones.

- El fundamento teórico toca brevemente en los temas que tratamos al programar el código.
- Metodología y Desarrollo describe la estructura y el desarrollo del código mismo.
- Conclusiones presenta cómo funciona el código final y sus cosas buenas y malas.

Fundamento Teórico

Relación De Objetos

POO

Usamos programación orientada a objetos para las clases en sí. Su uso fue más para organizar como funcionaba el codigo para mejorar la legibilidad y organizar variables.

Hererencia

Herencia es usar los atributos de una clase para construir otra, que agrega extra funcionalidad. Es útil en polimorfismo pero acá no se utiliza de esta manera. En nuestro proyecto se utilizó para dividir la clase principal en dos partes. 1) La parte que se encarga de archivos y su lectura y 2) la parte que se encarga de la lógica del juego.

Composicion

Composición es crear una clase que su funcionamiento es dependiente de otras clases. Para contener objetos que requieren muchos procesos mismos, creamos clases y luego los integramos a la clase principal.

fstream

Como iostream, fstream se encarga de procesar información externa. Pero a diferencia de iostream, consigue esta información de archivos. fstream lo utilizamos para interactuar con el programa battleship. Esto incluye lectura y escritura.

Vectores

Vectores son objetos que funcionan como arrays pero con tamaños variables basados en la cantidad de información que requieres guardar. Nosotros lo usamos para contener la información del fstream debido a que desconocemos cuánta información va contener el archivo. Además, lo utilizamos para procesar información de forma rápida y como un buffer temporal de información.

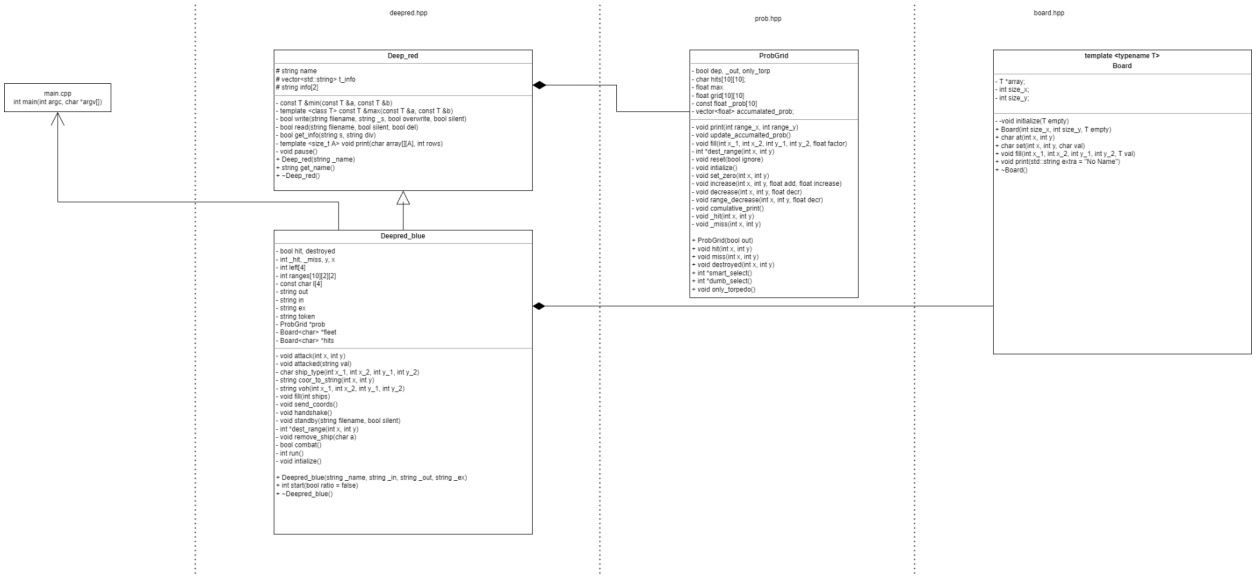
Template

Template ayuda a funciones o clases poder procesar información dependiendo de qué tipo es. Usamos esto para board.cpp para poder utilizarlo de forma clara y precisa.

Punteros/Variable Dinámicos

Variable Dinámicos son variables que son contenidos en el heap, e interactúan con el programa vía punteros dinámicos. Usamos esto para asegurar que variables no constantes como arrays funcionan de forma apropiada, debido a que hay casos donde hay problemas debido a cambios en tamaño.

Métodos y Desarrollo



(Nota: este concepto es presentado y explorado en Battleship, 2011). APA en bibliografía)

Programa (General)

El programa está dividido en 4 partes: main.cpp, deepred.hpp, board.cpp y prob.hpp

main.cpp
El main.cpp contiene el main, donde comienza a correr el programa. Crea una instancia de Deepred_blue y utiliza información del command line para saber si juega como player 1 o player 2. Además, imprime información como turnos cuando termina el programa.

deepred.hpp
Deepred.hpp contiene dos clases: deepred y Deepred_blue
Deepred se encarga de interactuar con los files, leyendo, escribiendo y borrando cuando necesario. También temporalmente contiene la información de un file para poder utilizarlo sin tener que leer constantemente el file.
Deepred_blue (quién hereda de Deepred) es una clase se encarga de la lógica general del juego. Corre un turno y que se debe hacer en caso de las respuestas de Deepred. Consigue y manda información de Probgrid.

board.hpp
Board se encarga de manejar toda la información de tablas, para que no sea necesario que sea contenido en deepred. No es integral al programa, pero simplifica y mejora la legibilidad del código mismo, además asegura división entre variables (problemas que ocurren cuando hay múltiples arrays). Deepred_blue contiene uno para el fleet y otro para los hits. Es

“multidimensional”, pero esto es una abstracción, debido a que el board solo usa un array de solo una dimensión para ahorrar memoria y mejorar velocidad.

prob.hpp
prob.hpp contiene la clase Probgrid
Probgid es cual corre toda la lógica especificada en la primera parte. Usando esta información, manda la mejor elección para hacer un ataque. La forma que corre esta lógica es usando un float array y un vector. El array contiene las probabilidades de cada una, y el vector la probabilidad cumulativa)

Parte 1: Checker y Probabilidad (Aircraft Carrier + Battle Cruisers + Submarines + Torpedo boat)

Primero hay que considerar los más fáciles de encontrar, los más grandes (e.i. Los de 2 o más en tamaño). Considerando esto, podemos ignorar la mitad de las posiciones, ya que todos deben estar al menos en uno de ellos, en los blancos o negros.

Además podemos reducir más usando probabilidad, usando float[1] representando probabilidades/ puntaje (más alto puntaje, más probable es que haya bote). Usando una combinación de esto, es fácil encontrar un bote usando los siguientes parámetros:

- Usando Probabilidad general, asumimos que cualquier combinación es igual de probable, pero con una concentración más alta en el centro
- “Negro” o “Blancos” son 1 de puntaje, para que no elija ninguna de ellas (con test he encontrado que para selecionar 1 es matematicamente improbable)
- Un miss decrece la posibilidad en esa fila, vertical y horizontalmente. Esto es usado para incentivar la variedad en el proceso de elección.

Una vez encontrado el bote, programa cambia como calcula la probabilidad:

- Los de al costado incrementan en probabilidad de tener botes
- Un miss de arriba y al costado de cualquier casilla da casi 100% de probabilidad que el bote esté abajo del otro.

Parte 2: 100% Random
Para los torpedos, no existen muchas estrategias viables, entonces en estos casos será completamente random. (Mientras que no haya hit or misses en cualquiera). Usamos esto con una combinación de el decremento de probabilidades y imposibilidad de dar a espacios ya usados para crear un sistema random más variado

Usando la combinación de estas dos estrategias, se crea el programa mismo.

[1] Porque float? Float son MUCHO mejores con división y multiplicación, la base de todo el concepto.

Detalles técnicos extra

- Uno de los problemas más grandes del programa no era la probabilidad de jugar el juego. Era la interacción con archivos. Algunos de estos problemas incluyen
    - Tiempos no determinados entre input y respuesta
        - Solucionado con un delay (standby)
    - Velocidad de cambio
        - Se tuvo que mejorar la velocidad de del read para poder leer las respuestas
    - Colapso del programa
        - Hay varios safeguards en caso de colapso del programa, incluso remendando la última jugada del jugador. Aunque responda con incorrecto, recomenzara el proceso de in and out.
- La interacción con G: presentaba problemas que no había en ninguno de los test anteriores. Battleship colapsaba con más frecuencia. Esto inspiró varios de los safeguards.
- Se agregó un CMDline interacción para poder jugar como PL1 o PL2, entonces se puede agregar un argumento para seleccionar como que jugador va a jugar.
- La separación de board de deep red blue fue creado más tarde en development debido a variedad de funciones de repetidas para exactamente la misma función. El board es de una sola dimensión pero crea funcionalmente como uno multidimensional. No solo funciona de forma más rápida, sino también simplifica conseguir y manipular información.
- Puse la mayoría del procesamiento cuando es el turno del programa, poniendo “buffers” para poder procesar con tranquilidad la información. De esta forma, no es necesario tener que constantemente procesar durante la lectura de información.

Conclusiones

Resumen de funcionamiento:
Cuando el programa comienza, provee alguna información de inicio, imprimiendo lo que escribe al .in file. Luego espera hasta que el oponente se posicione para comenzar. Inmediatamente corre, manda sus golpes, usando la información para calcular el mejor siguiente y espera su turno, imprimiendo los dos tableros en momentos apropiados (e.j justo después de un ataque). Cuando termina el juego (en victoria o derrota), se cierra automáticamente después de imprimir su hits y miss con su ratio de jugadas victoriosas.

El presente trabajo es un programa que puede jugar de forma completamente independiente de algún jugador.

- Se puede comunicar de forma clara y eficiente con el programa battleship para jugar usando los .in y .out
- Crea dos tableros
    - Puede enseñar la su propia tabla de golpes y la de su oponente*
- Puede jugar el juego, respetando turnos y termina cuando gana o llega a 100 golpes.
- Puede posicionar la flota basado en un preconstruido array
- Funciona de forma 100% independiente
- Basado en las calculaciones, la clase probgrid puede calcular y retornar el mejor valor posible para mandar un ataque.
- Puede fácilmente dar a todo un barco de forma seguida y puede ganar una partida en alrededor de 80 tiros (El juego promedio de battleship es típicamente 96 tiros (Battleship, 2011)).
- El Deepred_blue se encarga de todo lo que involucra la comunicación con el programa de battleship y provee probgrid con la información necesaria para hacer los cálculos.

*Mirar problemas

El proyecto no existe sin problemas:

- Para algunos casos se tuvo que sacrificar velocidad para mejorar la interacción de la aplicación y prevenir colapso si hay un error del programa (Battleship.exe)
- El G: drive causa que ya no mande files hasta que tenga que procesar archivos una vez más. Hasta el momento no se que causa el incremento de errores
- Mientras no causó problemas al inicio, cuando se implementó el miss y hits del oponente al programa, el failed hit no procesaba debido a que el programa inmediatamente actualizaba el archivo para presentar la información que era el turno del jugador, información que se debía priorizar sobre esto. En consecuencia, el programa solo puede enseñar los hits del oponente, pero no sus misses

Bibliografía

- Battleship. (2011, December 3). Data Genetics. Retrieved November 28, 2021, from https://www.datagenetics.com/blog/december32011/