Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica

II ciclo 2021

Proyecto: Juego Clue

IE-0117 Programación Bajo Plataformas Abiertas

Estudiantes:

Jean Paul González Benavides (B83381)

Laura Mata Mata (B84689)

Grupo: La Papa

Profesor: Juan Carlos Coto

3 de diciembre del 2021

Tabla de contenido

Introducción. 1

Diseño General. 2

- Archivo main.c 2
- Archivo cantidad_jugadores.h 2
- Archivo pedir_nombre.h 3
- Archivo dado.h 3
- Archivo juego.h 4
- Archivo puntos.h 5
- Archivo mapa.h 6
- Caso sin mapa. 8

Principales retos. 9

Conclusiones. 10

Introducción.

El objetivo de este proyecto es poner en práctica los conceptos obtenidos en el curso de plataformas, específicamente en la parte de programación en lenguaje C. Para esto se programó el juego CLUE que consiste en que un grupo de jugadores, mínimo 2 y máximo 6 jugadores, se mueven a través de habitaciones para poder averiguar quien cometió un asesinato, sospechando el lugar, objeto y jugador.

Para la implementación se utilizan 21 cartas, donde 6 corresponden a las posibles armas utilizadas para el asesinato, 9 corresponden a los lugares donde pudo haber ocurrido el asesinato y 6 personas sospechosas.

Para la implementación lo primero que se realiza es separar tres cartas, que van a ser la persona que cometió el asesinato, el lugar y el arma utilizada, las demás cartas se reparten a los jugadores, esta repartición depende de la cantidad de jugadores, si son 2 se reparten 9 cartas a cada uno y si son 6 se reparten 3 a cada jugador. Luego, para que el jugador pueda sospechar o culpar debe entrar a una habitación y de ahí se despliegan las opciones para ver sus cartas, sospechar, culpar o salir del juego.

En este juego puede ser que ningún jugador gane, solo va a ganar la persona que adivine las tres cartas del asesinato, si culpa y el jugador está equivocado, el jugador pierde y no puede seguir jugando.

Diseño General.

Para la implementación se realizaron seis funciones, donde main.c es la principal, es la que se encarga de llamar a las demás funciones.

Archivo main.c

En esta función se implementa el menú del juego. Lo primero que se define es un char opción, que con ayuda del scanf va a elegir la opción del menú a la que se quiere acceder. Todo el código, exceptuando el char opción , está dentro de un do while, esto con el fin de que imprima todas las opciones y luego con char se escoja si quiere iniciar juego, ver las puntuaciones, leer instrucciones o salir. Después ejecuta el while que va a evitar que se escoja el entero negativo (EOF) dado, esto para evitar que se encicle el código.

Si se digita ‘1’ se inicia el juego y se llama a cantidad de jugadores, al digitar ‘2’ se pueden observar las puntuaciones más altas, al introducir un ‘3’ se despliegan las instrucciones del jugo, al digitar ‘4’ nos lleva a una versión del juego sin el mapa y, por último, al digitar ‘5’ se cierra el programa. Cabe destacar que si se marca cualquier opción que no sea de ‘1’ a ‘5’, se va a indicar que el valor es erróneo y se debe digitar de nuevo la opción.

Archivo cantidad_jugadores.h

Esta función se comienza a ejecutar cuando el usuario marca la opción uno en el menú. El objetivo de esta es recibir el valor de jugadores que va a tener el juego, este valor lo va a escribir el usuario en la variable valorUsuarioAux gracias al comando scanf. Luego este número (tipo char) se convierte a entero, se guarda en valorUsuario y se llama a la función pedir_nombre, que va a tener como atributo valorUsuario.

Archivo pedir_nombre.h

Esta función se va a encargar de la parte de almacenar los nombres de los jugadores y llamar a la función mapa. Como se puede observar en el código, primero se definen un entero x que tiene un valor de ‘1’ que va a ayudar a definir el número de jugador cuando se le pide el nombre, luego se define un carácter temp que es donde el usuario va a digitar el nombre que quiere tener en el juego. También, se define un matriz de carácteres de dos dimensiones que va a almacenar el número del jugador (la posición en el array) y el nombre respectivo.

En la línea 16 del código se realiza un if para informar al usuario que el valor digitado de cantidad no corresponde al rango de jugadores.

En la línea 25 del código se realiza un while que se ejecuta la cantidad veces equivalentes a valorUsuario (este va de 2 a 6), luego con scanf y el carácter temp el usuario digita su nombre y este se almacena en la matriz, el while se sigue ejecutando hasta que valor usuario sea 1, ya que dentro hay un if que toma en cuenta cuando valorUsuario vale 1.

Cuando finaliza el while se definen el entero ‘i’, que se utiliza en el for de la línea 43 del código que tiene la función de imprimir el número y nombre de cada jugador; y, por último, se llama a la función mapa que tiene como atributo valorUsuario (cantidad de jugadores) y la matriz nombre.

Archivo dado.h

La función tiene como objetivo simular la acción de un dado real, donde se tiene un array llamado dado con todos los valores posibles para moverse. Se utiliza la función srand() junto con el tiempo para obtener un valor aleatorio a escoger del array mencionado, a este valor se le aplica el módulo para que esté dentro del rango del array. Finalmente, se imprimen las indicaciones de las teclas para moverse en el mapa junto a la cantidad de posibles movimientos que pueden

realizarse para este turno, como son dos dados los valores posibles van de 2 a 12 movimientos.

Archivo juego.h

Esta función es llamada desde mapa, por lo que trae los atributos necesarios para poder realizarse. Lo primero que se pretende es declarar las variables necesarias para llevar a cabo la función, el código es una secuencia de if's que de condición está la cantidad de jugadores (valorUsuario), por lo que sí valorUsuario es 3, ese if va a trabajar con jugador 1, 2 y 3, por lo que se va a explicar la dinámica con dos jugadores ya que la diferencia es que se cambia el código agregando la cantidad de jugadores correspondientes.

Al entrar al if cuando valorUsuario es igual a 2, en la línea 21 del código, lo primero que se realiza es ver si el estado de los jugadores es de perdedores o no, si el jugador no ha perdido se pide que si quiere ver sus cartas, sospechar o culpar. Entonces, si el jugador marca 'r', se le van a realizar un for donde imprime de todas las armas y el jugador elige la que cree que es correcta y se guarda en supos[0]; en estas secciones se encuentran whiles para que le siga preguntando al jugar qué elemento desea seleccionar en caso de digitar cualquier tecla que no sea correspondiente a los elementos mostrados en pantalla. Realiza lo mismo con los lugares y lo guarda en supos[1], así como con los personajes y este valor se guarda en supos[3]. Con este array se compara con el array de asesinato, si es igual, el jugador gana, termina el juego y, si es diferente el jugador pierde y el turno pasa al siguiente jugador.

Si el jugador digita 'c', entra en el if correspondiente e imprime las cartas que le fueron repartidas al inicio y luego vuelve a retornar a juego para poder decidir si quiere sospechar, adivinar, ver cartas o salirse del juego. La otra opción es marcar cualquier tecla, esto te va a dejar sospechar del asesinato, realiza la misma función que la de culpar con la diferecia de que cuando se tiene el array supos[ ] cada valor se compara con las cartas de cada jugador y se despliegan en

la pantalla las que concuerden con supos[ ], en caso de que no aparezca ninguna es porque las tiene el propio jugador que hizo la sospecha o que son las cartas del asesinato; así los jugadores podrán ir descartando posibles culpables y cuando termina retorna a mapa y para que siga el siguiente jugador.

En esta parte se llama a la función puntos, para almacenar el nombre y las puntuaciones de cada jugador, con el fin de que cuando termine la partida los datos del ganador se puedan guardar en las puntuaciones.

Archivo puntos.h

El archivo puntos.h contiene dos funciones: guardar_puntuaciones() e imprimir_puntuaciones(). La primera de estas dos funciones se encarga de tomar el nombre del ganador de una partida, junto con su puntuación y los guarda en la lista de las diez puntuaciones más altas sólo si dicha puntuación supera al menos la puntuación en el puesto diez. Para conseguir esto, la función almacena los nombres de los jugadores y sus puntuaciones en un array bidimensional, el cual se modifica a través de la implementación del algoritmo de re ordenamiento conocido como *bubble sort*. La parte del código en la que se aplica este algoritmo reordena las posiciones en la lista de menor a mayor, ya que entre menos puntos tenga un jugador mejor fue su desempeño en una partida. Posteriormente, la función escribe la lista reordenada en el archivo puntuaciones.txt y también actualiza la lista de nombres y puntos, los cuales se encuentran en los archivos nombres.txt y puntos.txt respectivamente.

Por otra parte, la función imprimir_puntuaciones() se encarga de imprimir en la pantalla la lista de las diez puntuaciones más altas. Para ello, la función abre el archivo puntuaciones.txt en modo de lectura e imprime la lista presente en dicho archivo en la consola de comandos.

Archivo mapa.h

La función mapa realiza dos tareas importantes, la primera es realizar la distribución de las cartas a cada jugador, la segunda es realizar los movimientos y llamar a la función jugador para poder culpar y sospechar el asesinato.

Lo primero que se realiza es crear una matriz que contiene el mapa, se definen las posiciones iniciales de cada jugador, los límites del mapa y un carácter que se utiliza para que el usuario pueda escribir en qué posición se quiere mover y un entero que es el contador de movimientos. Luego se imprime el mapa y se tira el dado.

Para repartir cartas se define un array que se llama cartas con un puntero al igual que los arrays de los estados de los jugadores, los arrays que van a almacenar las cartas de estos y enteros para cada jugador con el fin de que conforme se guarde una carta se cambia la columna para guardar la otra. Hay que tener en cuenta que solo hay 18 cartas que repartir por lo que se reparten dividiendo la cantidad de cartas entre los jugadores, si queda impar un jugador queda con más cartas. Las tres cartas faltantes corresponden a las cartas del asesinato, que se escogen de manera aleatoria con el comando rand(), estas 3 cartas se guardan en las primeras 3 posiciones del array guardar_cartas.

Solo se va a explicar el funcionamiento con dos jugadores, debido a que se cumple el mismo funcionamiento para los demás casos solo que se varía el rango de cartas y la cantidad de jugadores.

Al ingresar un valorUsuario igual a 2 entra al if de la línea 110 del código y pasa directo a un for que va a estar dentro de guardar cartas en la posición 3, ya que las tres primeras ya están ocupadas por el asesitato y se define que complete es igual a cero. Ahora se entra a un while, donde la única forma de salir es que el complete ya no sea cero, en este while se va a almacenar un valor random en guardar cartas y luego este entra a cartas y la carta elegida se almacena en jugador_1, que dentro de jugador_1 está k_1 que es el contador que va a cambiar

la posición del array y complete pasa a 1. Luego, esto ingresa a otro for que se encarga de revisar que el valor que se va a almacenar en jugador_1 no sea igual a los que ya se almacenaron en guardar_cartas, si esta carta corresponde a una que está en guardar cartas el complete pasa a ser cero. Si estas son diferentes se sale del while y se le suma un uno a k_1 y regresa al for hasta cumplir la condición. Cuando termina el for, se realiza la misma técnica para jugador 2.

Ahora, la lógica del mapa comienza en la línea 428, donde se definen los contadores de cada jugador y un carácter que permite al jugador indicar en qué posición se quiere mover. Luego entra a un while del que nunca va a salir debido a que el código no permite que el jugador se salga de los límites del mapa. Esta parte se programa por cada jugador, la dinámica es la misma para los 6 casos lo que cambia es que cuando se terminan los movimientos para el jugador 2, 3, 4 o 5 pregunta la cantidad de jugadores para ver si tiene que regresar a jugador 1 o seguir con el siguiente jugador. Para esta etapa se va a explicar el funcionamiento utilizando al jugador 1:

Cuando el jugador es igual a 1 se ingresa al if que se encuentra en la línea 443 del código, en esta parte entra a una secuencia de if que permite saber si todos los jugadores perdieron el juego para dar por finalizada la partida, si no se cumple, se entra al else y se comienza a pedir al jugador los movimientos que desea realizar, si se quiere mover hacia arriba se digita ‘w’, hacia abajo se digita ‘s’, si se quiere mover a la derecha se digita ‘d’ y a la izquierda ‘a’. Cuando se digita cualquiera de estos carácteres entra a un if que está condicionado a no realizar el movimiento si está al frente de una pared, jugador o letra. Si estas condiciones no se cumplen lo que se realiza es cambiar la coordenada original por un carácter vacío se le suma o resta (dependiendo de la posición) al eje ‘y’ o ‘x’ y se reescribe el valor del jugador que en este caso es el 1 en la nueva posición.

Si el jugador entra a una habitación, entra al if de la línea 555, en este caso se realizan diferentes condiciones dependiendo de qué movimiento se debe utilizar para entrar a la puerta. Cuando esté en esta parte se va a llamar a la función juego que es la que va a realizar la dinámica de que el jugador pueda elegir si quiere ver sus cartas, sospechar o culpar, también se define el valor de 'l' que indica si el jugador ya perdió o no. Después, se imprime el mapa con el valor modificado.

Luego de realizar esta función el contador de movimientos se iguala al valor de movimientos del dado, esta condición también sucede cuando el jugador se queda sin movimientos, por lo que va a pasar al if de línea 597 del código, este if permite pasar al siguiente jugador. Esta parte es la que varía con respecto a los siguientes jugadores ya que en este caso se debe tomar en cuenta la cantidad de jugadores para saber si debe regresar a jugador 1 o seguir con el siguiente jugador.

Por último en la línea 1326 del código, se realiza el for que tiene los valores del entero 'l', este entero permite pasar el estado del jugador a perdedor.

Caso sin mapa.

A la hora de implementar el mapa se obtuvieron problemas, ya que habían jugadores que al entrar a una habitación en el turno siguiente no podían salir de ahí ya que realizaba un movimiento automático que volvía a llamar a juego. Por lo que se implementó una función llamada unión jugadores, donde se realiza una implementación conjunta de las funciones cantidad_jugadores y pedir_nombre. Luego de pedir el nombre, esta verifica el entero 'valorUsuario' para saber la cantidad de jugadores y revisa los condicionales para visualizar cual caso tenemos, si es de 2, 3, 4, 5 o 6 jugadores. Para el caso ocurrente de valorUsuario, se va a una función juego específica para esa cantidad de jugadores y se procede a jugar sin el mapa. Aquí el juego funciona bien.

Principales retos.

- ➔ Al implementar el mapa la idea original era realizar la lectura y escritura del mapa en un archivo .txt, el problema fue que en el momento de implementarlo al indicar la coordenada del jugador, en el archivo impreso no lo localizaba. Como esto es una parte esencial del juego se decidió replantear cómo se iba a trabajar el mapa, entonces, se empezó a trabajar el mapa como un objeto estático y no guardarlo en memoria, como se planteó al inicio.
- ➔ Otro reto importante que se tuvo con el mapa fue que antes este trabajaba con tres funciones para realizar el mismo comportamiento que tiene en este momento, entonces para una mayor facilidad se tuvo que reestructurar para que quedara en una sola función.
- ➔ Se presentaron problemas a la hora de unir la sección de juego con el mapa, ya que los dos archivos se hicieron por aparte, entonces se tuvo que reestructurar las funciones para que recibiera los parámetros necesarios para hacer funcionar la conexión entre el mapa y la llamada al juego dentro de la misma.
- ➔ Otro reto fue con respecto al entrar a las habitaciones, debido a que realizaba el cambio pero el programa enciclaba al jugador para que no pudiera salir de esa habitación, este problema no se logró modificar.
- ➔ Con respeto a las puntuaciones, se tuvo problemas a la hora de leer los archivos de texto de las puntuaciones y también a la hora de guardarlos, debido que para guardarlos se quería realizar por medio de un for, pero al final se tuvo que hacer de forma serial.

Conclusiones.

Para concluir, se logró realizar parcialmente el código del juego que se implementó, debido a que no se localizó de donde provenía el error cuando un jugador entraba a una habitación por lo que se optó por realizar una versión donde se implementara el juego sin tablero.

Gracias a este proyecto se logró aprender más sobre los comandos que se pueden realizar en C, también se observó lo positivo que es trabajar en diferentes archivos para que el código sea más ordenado de trabajar. Esto debido a que todos los archivos tienen bastantes líneas de código por el hecho de que se debía de repetir el código ya sea para los diferentes jugadores o la cantidad de jugadores.

Para terminar, fue un proyecto tedioso debido a que C es un lenguaje de programación muy reducido, por lo que al querer realizar ciertas funciones hay que re acomodarlas a los comandos que C nos brindaba, al igual que se tuvo que investigar ciertos comandos para poder llevarlo a cabo.