**Proyecto Final. Redes y Videojuegos en Red**

|  |  |
| --- | --- |
| **Título** | MasterMind |
| **Alumno** | Luis Cano, Sergio Alberto |
| **Alumno** | Docampo Prieto-Puga, Marcos |
| **Breve Descripción: MasterMind clásico por turnos** | |
| * *Funcionalidad básica: El MasterMind que proponemos consiste en que dos jugadores tengan que adivinar la combinación propuesta por la máquina antes de que el otro contendiente llegue a la solución.* * *Arquitectura y requisitos de red: El juego necesita de 2 clientes (1vs1) y un servidor para manejar los datos que los jugadores aportan cada turno.* * *Librerías gráficas: X11/XLib.h* | |
| **Qué es MasterMind** | |
| *MasterMind es un juego de razonamiento, con una edad +8, de menos de 20 min. de duración, y para dos jugadores. El objetivo del juego es adivinar una combinación de colores secreta que plantea el contrario. Esto significa que en cada ronda hay un jugador que codifica un patrón de colores y hay otro que debe adivinarlo, y si lo hace antes de que acaben todos los turnos, es el ganador.*  *El código se compone de una combinación de 4 elementos con posibles repeticiones de entre 9 colores (Rojo, Naranja, Amarillo, Verde, Azul, Morado, Rosa, Blanco y Negro).*  *Para hacer este juego competitivo para dos jugadores, vamos a hacer que el código lo proponga la máquina y los dos jugadores deban adivinarlo. El que adivine antes el código ganará la partida. El Player 1 jugará primero (y todos los turnos impares) y viceversa.*  *El juego contiene un tablero, que es más largo que ancho, teniendo una fila por cada turno que se haya jugado (normalmente 10 como máximo), más otra fila, oculta, que es donde está la clave que el adivinador debe sacar.*  *Cada fila del turno se compone de cuatro agujeros grandes y cuatro pequeños, y un número que representa el turno en el que estamos, de la siguiente forma:*  *Nº Turno    Color 1-Col 2-Col 3-Col 4    Acierto 1-Aci 2-Aci 3-Aci 4*  *Cada “Color” es un intento por resolver la posición en la combinación original, y cada “Acierto” es la respuesta a si se ha conseguido sacar algo de esa combinación. Los aciertos se pueden marcar de tres formas: Chincheta negra si se ha acertado el color y la posición del color puesto, chincheta blanca si se ha acertado el color, pero no la posición, y sin chincheta si el color propuesto no coincide con ninguno de la combinación a adivinar. De esta forma, si el código propuesto es:*  *Rojo-Amarillo-Verde-Azul*  *Y el jugador propone en el turno 1:*  *1    Rojo-Verde-Rosa-Naranja*  *Los marcadores de acierto serán:*  *Negro-Blanco-Nada-Nada*  *De tal forma que la línea del turno quedaría:*  *1      Rojo-Verde-Rosa-Naranja      Negro-Blanco-Nada-Nada*  *Nota 1: Chincheta y Ficha significan lo mismo en este ámbito.*  *Nota 2: El orden de colocación es crucial en este juego.* | |
| **Arquitectura del Juego** | |
| 1.1 Modelo de Juego  *El servidor se inicia y espera a los dos clientes. Cuando los clientes se conectan el servidor les informa de cuál es su turno y espera mientras los clientes interactúen por turnos, jugando primero el que antes se conectó. El servidor al iniciarse crea un código de cuatro colores aleatoriamente, el cual tendrán que adivinar los jugadores.*  *El juego se compone de fichas medianas de colores, y otras pequeñas de dos tipos, blancas y negras. Las de colores las ponen los jugadores para adivinar la contraseña y las blancas y negras dan información (aportada por el servidor) sobre cuáles son correctas, incorrectas o son correctas, pero están en un lugar incorrecto.*  *En cada turno la jugada de cada cliente se enviará al servidor, después el servidor realiza la comprobación para saber si alguien ha ganado. Si nadie lo ha hecho, envía la jugada a los dos clientes para renderizar el tablero, y espera la jugada del siguiente jugador.*  *El que antes lo descubra gana la partida. Se habrá de tener en cuenta que los dos jugadores podrán ver la jugada del contrario, haciendo más fácil la partida. Dada esa información, los jugadores tendrán las pistas que devuelve el servidor a cada jugada, por lo que habrá que no solo adivinar la contraseña oculta, sino también dificultar la partida al contrario intentando dar las menos pistas posibles al oponente.*  *El límite de turnos será de diez, o sea, cinco por cada jugador. Una vez la partida finaliza, se anuncia el ganador, o perdedores por pantalla, se cierran los clientes y el servidor.*  1.2 Estado del Juego: Objetos y Replicación  *En el juego existen dos tipos de objetos, los enviados y los recibidos. Son distintos en el caso de servidor y cliente.*  *El servidor recibe objetos de 4 caracteres y envía objetos de 9 caracteres. Los clientes funcionan a la inversa, envían objetos de 4 caracteres y reciben objetos de 9.*    1.3 Protocolo de Aplicación y Serialización  *El servidor debe tener un sistema en el que acepta clientes, organiza turnos entre ellos y recibe combinaciones de 4 elementos que posteriormente analiza y compara con la combinación que él mismo genera aleatoriamente. Luego genera una respuesta a clientes con la combinación del turno del jugador correspondiente y añade 4 elementos más con las pistas y un extra de fin de partida.*  1.4 Arquitectura del programa.    *Clases:*  *-Game.cc -Game.h -Makefile -MMClient.cc -MMClient.h -MMServer.cc -MMServer.h*  *-Serializable.h -Socket.cc -Socket.h -XLDisplay.cc -XLDisplay.h*  *1.4.1 Makefile*  *Es un archivo que sirve para compilar en la terminal de Linux.*  *En él se especifican las características necesarias para la compilación de* | |
| **Diseño del Servidor** | |
| 2.1 Características del Servidor  *Se trata de un servidor TCP adaptado para aceptar únicamente dos clientes por thread, los cuales serán el Player 1 y Player 2 en el orden en el que se conecten.*  *El servidor creará un thread para cada dos clientes.*  2.2 Lógica que ejecuta  *El servidor se inicia con IP y puerto como argumentos del programa. Genera una combinación aleatoria de 4 números que representan los colores y la guarda en un array de la siguiente manera:*  *enum COLOR { Rojo, Naranja, Amarillo, Verde,*  *Azul, Morado, Rosa, Blanco, Negro };*  *COLOR sol[4] = { Rojo, Azul, Azul, Amarillo };     // 0-4-4-2*  *Después espera a la conexión de dos clientes. Informa a cada jugador de quien es cada uno, es decir, quien es el jugador 1 y quien el 2.  Inicia la recepción de mensajes del primer conectado. El jugador 1 elige su combinación y la manda al servidor. El cliente serializa la información antes de enviarla, y el servidor recoge el mensaje, lo deserializa y lo procesa.*  *Si la combinación es acertada:*   * *Envía la respuesta de la combinación elegida por el jugador, las cuatro pistas correctas y un bool = true de fin de partida a los dos jugadores. Después sale de su bucle principal y el programa acaba.*   *Los clientes salen de su bucle y dependiendo de quien haya ganado, aparecerá en pantalla un mensaje con el jugador ganador y perdedor, y el programa termina en los clientes.*  *El ganador se define en función de quién fue el jugador que tuvo el último turno.*  *Si la combinación no es correcta:*   * *El mutex se activa para la ejecución de código del jugador 2 que debe enviar su combinación y se repite el proceso del servidor.*   2.3 Lógica de clientes  *Los clientes ejecutan su programa introduciendo como argumentos la IP y puerto del servidor. Renderizarán en pantalla el tablero de juego y serán informados por el servidor de si son el Player 1 o el Player 2.*  *Si es su turno, deben enviar una combinación de 4 caracteres al servidor y se quedarán a la espera de que llegue su turno nuevamente.*  *Cada vez que los clientes reciben un mensaje se deserializa la información, se procesa en el render y se pinta en pantalla. Si es su turno y el juego no ha finalizado, el jugador debe seleccionar una combinación de cuatro colores y mandarla al servidor.* | |
| **Conclusiones** | |
| 3.1 Posibles Mejoras:   * *En vez de usar vector<vector<int>> usar un array de arrays, pero da más problemas a la hora de serializar.* | |
| **Referencias** | |
| *https://mathworld.wolfram.com/Mastermind.html* | |