Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

Laboratorio de Algoritmos y Estructuras I (CI2691)

**Informe de**

**Proyecto**

**(Entrega Final)**

Autores:

Guillermo Betancourt, carnet # 11-10103, grupo # 3

Gabriel Giménez, carnet # 12-11006, grupo #16

**Introducción**

Resumen del juego:

El juego se trata de una versión simplificada del juego de mesa Master Mind, que requiere dos jugadores, en nuestro caso la máquina y el usuario. Uno de los dos jugadores será el Codemaker, quien se encargará de crear una clave secreta de cinco colores con una cantidad de siete colores a escoger. Luego, el otro jugador será el Codebreaker, quien se encargará de descubrir dicha combinación de colores en la menor cantidad de turnos posibles. Los roles de Codebreaker y Codemaker se alternarán en cada nueva ronda.

Al final de cada ronda, el Codebreaker recibirá tres puntos si logra descifrar el código secreto en menor cantidad de turnos que su contrincante en la ronda anterior; recibirá 1 punto si lo hizo en la misma cantidad de turnos; y finalmente no recibirá puntos si lo hizo en mayor cantidad de turnos.

El ganador del juego será el jugador que tenga más puntos al finalizar la cantidad de rondas establecidas al inicio del juego.

**El juego**

Objetivo del juego:

Obtener una puntuación más alta que el contrincante al finalizar la partida.

Contenido del juego:

El juego constará de los siguientes elementos:

* 1 tablero.
* 1 grupo de fichas pequeñas (blancas y negras).
* 1 grupo de fichas grandes (azul, amarillo, rojo, anaranjado, verde, morado, marrón).

Preparación:

En el juego participan solo dos jugadores: El Usuario y La Máquina

El juego se desarrolla por rondas y fases: Cada ronda consta de 2 fases, donde se alternan los roles de CodeBreaker y CodeMaker, y en la primera fase  los roles son escogidos aleatoriamente.

En el tablero se desarrollará el juego Master Mind, donde se colocarán cinco fichas grandes ocultas en la parte superior donde el Codemaker guardará el código a descubrir por el Codebreaker. Luego, se dispondrá de otras cinco fichas grandes donde el Codebreaker, en cada turno, intentará descubrir el código secreto.

Cuando te toque jugar, dependiendo de tu rol, tendrás 2 modos de juego:

* Si eres CodeBreaker, deberás colocar fichas desde la parte inferior del tablero, intentando adivinar la clave secreta que la máquina te ha generado. Luego en base a las respuestas indicadas por la misma jugaras tus próximos turnos.
* Si eres CodeMaker, deberas colocar una clave secreta que la máquina intentará adivinar.

**Pistas**

La lectura de las pistas de la máquina son leídas de la siguiente manera: un grupo de cinco fichas pequeñas, donde se expresa cuál es la relación entre el código secreto y el código enviado: Si en una cierta posición, la ficha es de color negro, significa que el color está en el código secreto y está en la posición correcta; si la ficha es de color blanco, significa que el color está en el código secreto pero que no se encuentra en la posición correcta; y finalmente si la ficha es de color gris, significa que el color no está en el código secreto (lo que implica que tampoco posee posición correcta alguna dentro del código secreto) o se repite un mayor número de veces.

**Rondas**

Una ronda está compuesta por 2 fases

**Fases**

Una fase es jugada cada vez que un código secreto es generado, las fases dentro de las rondas se alternan.

**Turnos**

Un turno es jugado, cada vez que una combinación es comparada con la clave secreta.

**Dificultad**

Dependiendo de la dificultad, se tendrá una mayor cantidad de turnos para adivinar la clave secreta. Fácil = 15 Turnos, medio = 10 Turnos, y difícil = 5 turnos.

**Final del Juego**

Cuando todas las rondas establecidas al inicio de la partida son jugadas, el jugador con mayor número de puntos gana el juego.

**Problema a resolver**

El programa constaría de tres (3) pantallas principales, la primera donde se seleccionaría que se quiere hacer, “Juego nuevo” o “Cargar una partida”. En la segunda pantalla, en caso de ser seleccionado el juego nuevo se debe poder escoger el número de rondas y la dificultad donde se quiere jugar. Luego, la tercera pantalla sería el tablero de juego donde se jugaría en ambos casos, si la máquina es el Codebreaker o si el usuario es el Codebreaker. Finalmente, si se selecciona el botón “Cargar partida”, el programa debe generar el tablero nuevamente, pero con las rondas y puntaje adecuados.

Las primeras dos pantallas se manejarían con botones importados de Tkinter, que permitirían llamar a funciones para tomar las decisiones deseadas. Una vez ya escogido eso, pasamos a resolver los siguientes problemas, necesitamos ambos modos de juego:

En el caso de que el usuario sea el Codebreaker, necesitamos que la máquina genere automáticamente un código secreto a descifrar, luego se procede a recibir un *input* del usuario y a verificarlo. La verificación del input debe ser uno por uno, puesto que se debe decidir la condición de cada ficha: Si el color está, no está, o está en otra posición. Este *input* debe ser posible recibirlo sólo un número limitado de veces puesto que está restringido por dificultad de la partida.

En el otro modo de juego, cuando el Codebreaker es la máquina, necesitamos recibir un *input* del Usuario, y así mismo una orden para que la máquina empiece a generar intentos. Luego la máquina generará un primer intento aleatorio de números, y en base a nuestra función ya creada de evaluación de fichas (utilizada en el otro caso), la computadora deberá tomar decisiones para así jugar las siguientes fichas. Si alguna ficha tiene el color y la posición correcta, la máquina no debe cambiarlo; en caso contrario debe intentar con otro color. A medida que se juega, debe ser visible tanto el puntaje que tienen ambos jugadores, como el número de intentos en el que descifró su última partida.

Finalmente al finalizar cada ronda la máquina debe ser capaz de calcular el puntaje a sumar a cada jugador: Si los dos utilizaron la misma cantidad de turnos en su fase, el programa debe otorgar 1 punto a cada uno. Si alguno de los dos utiliza una menor cantidad de turnos que el otro, la Máquina debe asignar 3 puntos al que necesito menor cantidad de intentos, dejar el puntaje del otro usuario igual.

**Descripción de la Carta Estructurada**

Para cada módulo se usa el siguiente esquema:

Nombre en la Carta Estructurada -> Nombre del Subprograma en el esqueleto

Explicación del Subprograma

A continuación la descripción de los módulos:

* Cambiar color -> changeColor(event)

Procedimiento que permite cambiar los colores al momento de ingresar un turno.

* Crear botones -> crearBotones(largo, TURNO, tablero, lista = [1,1,1,1,1])

Procedimiento que crea una fila de botones de la clase Botones.

* Crear color -> crearColor(coorx, coory, spacex, numero, i, tablero)

Procedimiento que permite crear una ficha especificando número y posición.

* Revelar secreto ->  revelarSecreto(x, y, spacex, lista, tablero)

Procedimiento que crea una lista ordenada de cinco botones.

* Inicializar -> Inicializar()

Procedimiento para inicializar, que solicita el nombre del Usuario, la dificultad a jugar y la cantidad de rondas.

* Crear juego -> crearJuego(juego, dificultad, rondas, nombre, seleccion)

Procedimiento que ejecuta las funciones establecerParametro e iniciar, con los parámetros anteriores.

* Establecer parámetros -> establecerParametros(juego, dificultad, rondas, nombre)

Procedimiento que asigna a ciertas variables de la clase juego, los parámetros dificultad, rondas y nombres.

* Iniciar -> iniciar

Procedimiento que oculta una pantalla y crea un tablero.

* Generar lista -> generarLista(numero)

Procedimiento que genera una lista aleatoriamente de cinco elementos enteros del 1 al 7

* Desactivar botones -> desactivarBotones(botones)

Procedimiento que permite desactivar los botones de los intentos anteriores.

* Verificar -> verificar(usuario, tablero, boton, juego)

Procedimiento intermedio que permite a la Máquina comparar la clave con el intento y toma una decisión sobre si ya descubrió la clave o aun tiene turnos por jugar

* Ganador -> ganador (juego, tablero)

Procedimiento que indica quien es el ganador según la sumatoria final de los puntos

* Fin de ronda -> finRonda(juego, tablero)

Procedimiento que compara la cantidad de turnos en las que se encontró la clave secreta para asignar los puntos.

* Probar -> probar(intento, jefe)

Procedimiento que genera el resultado al comparar la clave con la clave secreta.

* Jugar -> jugar (jefe, tablero, maquina, clave, intentoAnterior=[])

Procedimiento que grafica el resultado del turno jugado por la Máquina y genera un nuevo turno partiendo del turno jugado anteriormente

* Generar -> generar (resultado, intentoAnterior, permutar, noesta)

Procedimiento que genera un nuevo turno a partir del intento anterior.

* Descifrar -> descifrar (tablero, maquina, go, juego)

Procedimiento que genera un primer turno aleatorio cuando la Máquina es Codebreaker.

* Procedimiento cargar -> ProcedimientoCargar (nombreArchivo, master)

Procedimiento intermedio que llama a la función CargarJuego

* Guardar juego ventana -> GuardarJuegoVentana(juego, usuario)

Procedimiento que crea una ventana para guardar la partida

* Guardar Juego -> GuardarJuego(nombreArchivo, estructura, ventana)

Procedimiento que crea un archivo .txt que guarda todos los datos de la partida actual

* Cargar juego -> cargarJuego (nombreArchivo)

Procedimiento que lee un archivo .txt con todos los datos de un juego guardado anteriormente.

* Crear tablero -> crearTablero(juego,opcion="nuevo")

Procedimiento que crea un tablero de acuerdo a la dificultad establecida

* Ventana cargar -> ventanaCargar(master)

Procedimiento que permite solicitar el nombre del archivo .txt que contiene los datos de la partida guardada.

**Estructura de Datos**

* Clase Botones(): clase posee una función que llama a cada una de las imágenes de los colores y los convierte en botones
* Clase CodeBreaker()
  + turno: variable que asigna el turno que es jugado por el Codebreaker.
  + intento: arreglo que almacena el intento del Codebreaker.
* Clase Juego()
  + rondasMaximas: variable que asigna la cantidad máxima de rondas a jugar.
  + faseActual: indica la fase que está siendo jugada.
  + empiezaFase: variable que indica cual jugador inicia la fase.
  + dificultad: variable que indica la dificultad del juego.
  + nombreUsuario: variable string que indica el nombre del usuario.
  + codemaker: entero que indica quien será el Codemaker.
  + ganador: string que indica quien es el ganador al final del juego.
  + puntosUsu: cantidad de puntos que posee el usuario.
  + puntosMaq: cantidad de puntos que posee la máquina.
  + claveActual: variable que almacena la clave secreta.
  + intentosAnteriores: variable que guarda los intentos anteriores.
  + ultimaResolucionUsuario: variable que guarda la cantidad de intentos en los que el usuario descubrió la clave secreta
  + ultimaResolucionMaquina: variable que guarda la cantidad de intentos en los que el usuario descubrió la clave secreta

**Conclusión**

Luego de la realización del análisis descendente, logramos concluir que dicho análisis nos permite tener una visión mejor estructurada del problema a resolver. En nuestro caso, Master Mind constaba de muchos procedimientos los cuales, si no se abordaban de una manera adecuada, podían llevarnos a tener una resolución del problema mucho menos eficiente. Mediante el análisis descendente, se logró ordenar las actividades principales a realizar en nuestro código, y de ahí en adelante tener una esquematización más cómoda para acometer cada uno de los problemas que iban surgiendo a lo largo del proceso de programación.

Cabe destacar que los procedimientos anteriormente mencionados engloban todas las actividades menores que el código habrá de realizar, es decir, las funciones que en el código permiten dibujar el tablero, contar la puntuación, generar los botones, entre otras, no son explícitamente mencionadas en este informe, pues la finalidad del mismo resulta ser principalmente el análisis descendente del problema más global; sin embargo, dichas funciones también resultan de un análisis descendente para poder esquematizar todos los pasos necesarios a implementar en el código para que el juego funcione de manera esperada.

Referencias bibliográficas:

* Oscar Meza y Jesús Ravelo, "Introducción a la Programación". Universidad Simón Bolívar, 2012. Disponible en la web: [http://ldc.usb.ve/~meza/ci2611/IntroduccionA LaProgramacion.pdf](http://ldc.usb.ve/~meza/ci2611/IntroduccionA%20LaProgramacion.pdf)
* Documentación oficial de Python. Disponible en la web: <https://docs.python.org>
* Guías de laboratorio y prelaboratorio de Laboratorio de Algoritmos y Estructuras I (CI2691)