PROYECTOS DE PRORGAMACIÓN

MASTERMIND

Alejandro de Haro Ruíz.....alejandro.de.haro
Rafael Ramírez Pozo .....rafael.ramirez.pozo
Álex Sánchez Gil.....alex.sanchez.gil

Grupo 23.63
Versión 2.0

ÍNDICE

MANUAL DE JUEGO

Mastermind es un juego de mesa, de ingenio y reflexión, para dos jugadores.

Se juega en un tablero con fichas blancas y negras pequeñas y de otros colores, de un tamaño algo superior. Uno de los jugadores escoge un número de fichas de colores, 4 en el juego original, y pone un código secreto oculto del otro jugador. Este, tomando fichas de colores del mismo conjunto, aventura una posibilidad contestada con negras (fichas de color bien colocadas) o blancas (fichas de color con el color correcto, pero mal colocadas).

Termina al averiguarse la combinación (es decir, se consigue una combinación con cuatro negras), o bien se agota el tablero

Nada más ejecutar, la aplicación nos dará a elegir entre tres opciones:

- **Exit ->** Acaba la ejecución de la aplicación.
- **Log In ->** Inicia sesión con un usuario existente mediante un nombre de usuario y una contraseña.
- **Register->** Registra un nuevo usuario no existente mediante un nombre de usuario y una contraseña.

Una vez se ha iniciado sesión o registrados, tenemos más opciones entre las que elegir:

- **Log out ->** Cierra la sesión actual y vuelve al menú de inicio de la aplicación.
- **New Game ->** Crea una nueva partida, escogiendo entre diferentes modos de juego, dificultades y roles, e inicia la partida.
- **Load Game ->** Muestra una lista de partidas guardadas anteriormente y da la opción de cargar una de ellas y seguir jugando. En caso de que no haya ninguna partida guardada por el usuario, mostrará un mensaje informativo.
- **Ranking ->** Muestra una lista con las 10 mejores puntuaciones obtenidas por diferentes usuarios en el papel de Codebreaker.
- **Info ->** Muestra información sobre las diferentes opciones que da la aplicación para crear una partida nueva y sobre el método de corrección de los códigos del juego.
- **Edit User ->** Permite cambiar el nombre de usuario y/o contraseña del usuario actual.

Modos de juego

Según los jugadores:

- **Player vs. Player ->** El jugador juega contra otra persona en la misma máquina.
- **Player vs. CPU ->** El jugador juega contra la aplicación. La aplicación utiliza el algoritmo five-guess para jugar.
- **CPU vs. CPU ->** El jugador hace de espectador en una partida en la que juega la aplicación contra su propio algoritmo.

Según el rol del jugador:

- **Codemaker ->** El jugador crea un código que la máquina deberá resolver utilizando el algoritmo five-guess.
- **Codebreaker ->** La máquina crea un código para que lo el jugador lo resuelva.

Según la dificultad:

- **Easy ->** El código a resolver tiene tamaño 4 y 6 posibles colores. Los colores dentro del código no pueden repetirse. El máximo de intentos es 30.
- **Medium ->** El código a resolver tiene tamaño 4 y 6 posibles colores. El máximo de intentos es 25.
- **Hard ->** El código a resolver tiene tamaño 5 y 8 posibles colores. El máximo de intentos es 20.

Dentro del juego

Las acciones posibles durante la partida son:

Jugar turno: Para generar un turno, se hace uso de un sistema de drag & drop desde unos selectores de color hasta unas plantillas que actúan de receptores de dicho sistema para que el usuario pueda ir generando su solución color a color. En caso que el usuario quiera eliminar un color ya colocado, con un solo click sobre el color deseado se elimina de la selección. Según el rol del jugador, el código que escriba durante el turno será diferente:

- **Codemaker ->** En el primer turno, escribirá un código de colores siguiendo las normas según la dificultad. El resto de turnos corregirá el código propuesto por el codebreaker generando un código de corrección.
- **Codebreaker ->** Cada turno escribirá un código propuesta para resolver el código escrito por el codemaker.

Pedir pista: Al clicar sobre el botón *Help*, la aplicación genera una pista sobre el código solución del juego. Las pistas generadas siguen uno de los siguientes patrones:

- Token in position *positionnumber* is *tokencolor*.
- There is/are *numberoftokens* *tokencolor* token.

Consultar código solución: En caso de ser code maker, puede ser que se nos haya olvidado el código introducido, por lo que aparecerá el botón *Show Code*, el cual abrirá un popup con el código solución introducido.

Pausar juego: Al pausar el juego, la aplicación nos ofrece diferentes opciones.

- **Save ->** Guarda la partida que se está jugando y vuelve al menú de pausa. **Continue ->** La aplicación vuelve al estado de jugar turno.
- **Main Menu ->** Sale al menú principal del juego, previamente preguntando por confirmación ya que la partida en curso no será guardada al salir.

Método de corrección

Los códigos de corrección utilizan una codificación de colores diferente y tienen un uso específico:

NEGRO -> Informa al jugador que una de las fichas de su código coincide en color y posición con una de las fichas del código solución.
BLANCO -> Informa al jugador que una de las fichas de su código coincide en color con una de las fichas del código solución.
VACÍO -> Los huecos del código corrección que no sean negros o blancos. Informa al usuario que no hay mas fichas blancas o negras, así que no coinciden mas fichas de su código con el código corrección.

Cargado de partidas

En la vista de cargado de partidas, aparecerá una lista con las diversas partidas almacenadas por el usuario. En caso que no haya ninguna, el sistema avisará de ese echo.

- **Nombre de partida guardada**: el nombre de partida guardada aparecerá como etiqueta de un botón selector de partida. Una vez uno de estos botones se pulsa, se activa la petición de carga de partida y aparece el botón *Load Game* para confirmar el cargado de la partida seleccionada.
- **Botón *Delete*:** este botón aparece al lado de cada partida guardada, dando al usuario la posibilidad de borrar esa partida guardada permanentemente.
- **Botón *Load Game*:** aparece únicamente cuando se ha seleccionado alguna partida para cargar, y permite proceder a la carga de la partida seleccionada.

Guardado de partidas

El usuario puede elegir el nombre con el que desea guardar una partida. Una vez se ha pulsado el botón de *Save* en el menú de pausa, aparecerá un popup con un campo de texto en el que se puede indicar el nombre de la partida a guardar, que más adelante aparecerá en la pantalla de carga de partidas.

Edición de datos de usuario

El usuario puede decidir en un momento dado cambiar sus datos (nombre de usuario y contraseña). Para eso está habilitada la opción en el menú principal *Edit User*. Una vez pulsada, aparecerán dos zonas diferenciadas:

- **Editar nombre de usuario:** en pantalla aparecerá el nombre de usuario actual. En caso que se pulse el botón *Edit*, aparecerá un cuadro de texto sobre el que editar el nombre. No se permiten nombres vacíos. Cuando se ha acabado de editar, se clica al botón *Done* y automáticamente se realiza el renombramiento del usuario sin que éste pierda acceso a sus partidas guardadas.
- **Editar contraseña:** en la zona de edición de contraseña existen tres campos diferenciados:

    - **Contraseña actual:** obligatoria para autorizar el cambio de contraseña.
    - **Nueva contraseña:** contraseña a la que se desea cambiar (no se admiten contraseñas vacías).
    - **Confirmación de contraseña:** mediante la confirmación de contraseña se evita que el usuario introduzca equivocadamente la contraseña que quería poner y, por lo tanto, reduce las posibilidades que no sepa cuál es.

CASOS DE USO

Salir de la aplicación:
El cliente debe poder salir de la aplicación desde el menú inicial de la aplicación. El sistema le pide una confirmación para poder salir de la aplicación. En caso afirmativo, la ejecución de la aplicación se finaliza.

Registrar usuario:
El cliente debe poder registrarse en la aplicación usando un nombre y una contraseña de acceso. El sistema pide al cliente primero un nombre y una contraseña, así como una confirmación de la misma para evitar que pueda equivocarse en su introducción. En caso de que el nombre escogido ya se haya usado para registrar otro usuario, el sistema avisará al cliente de que no puede usar dicho nombre y el usuario no será registrado. En caso contrario, el usuario será registrado con el nombre y contraseña introducidos sin problemas.

Iniciar sesión:
El cliente debe poder iniciar sesión en la aplicación usando un nombre y una contraseña previamente usadas para registrar un usuario. El sistema pide al cliente un nombre y una contraseña. En caso de que el nombre y la contraseña no coincidan con ningún usuario registrado, el sistema avisará al cliente de lo sucedido y no se iniciará sesión. En caso contrario, el usuario iniciará sesión con el nombre y contraseña introducidos sin problemas.

Iniciar nueva partida:
El cliente, una vez ha iniciado sesión, debe poder iniciar una partida nueva. El sistema, a través de una vista con botones radiales, pide al usuario el modo de juego, el rol dentro del mismo y la dificultad de la partida. El usuario puede volver hacia atrás en cualquier momento pulsando el botón correspondiente del menú. Una vez introducidos los valores necesarios, el sistema creará una partida nueva y la iniciará.

Cargar partida guardada:
El cliente, una vez ha iniciado sesión, debe poder acceder a una lista de partidas comenzadas y guardadas por él anteriormente. El sistema mostrará por pantalla una lista de partidas y el cliente puede restaurar una de ellas pulsando sobre la misma, o bien volver al menú anterior. En caso de que no exista ninguna partida guardada, el sistema avisará de esto al usuario.

Jugar partida:
Después de iniciar nueva partida o cargar partida guardada, el cliente debe poder jugar la partida que se está ejecutando el sistema.

Consultar Ranking:
El cliente, una vez ha iniciado sesión, debe poder acceder a una lista de las mejores puntuaciones obtenidas al finalizar partidas y los respectivos usuarios que las han obtenido. El sistema mostrará dicha lista.

Consultar información de la aplicación:

El cliente, una vez ha iniciado sesión, debe poder acceder a información sobre el juego y los diferentes modos de juego. El sistema mostrará dicha información.

Cerrar sesión:
El cliente, una vez ha iniciado sesión, debe poder cerrar sesión con su usuario. El sistema pedirá confirmación al cliente para cerrar su sesión y en caso afirmativo, volverá al menú inicial de la aplicación. En caso contrario, volverá al menú en el que se encontraba.

Insertar código:
Mientras está jugando una partida, el cliente debe poder introducir una lista de valores que forma un código. El sistema muestra por pantalla una serie de selectores de colores y una serie de contenedores sobre los que arrastrar y soltar los colores de los selectores para que el cliente pueda generar un código con ellos, de un tamaño y siguiendo unas normas según el modo y la dificultad de la partida, también especificadas por el sistema. Si el código no corresponde con las normas establecidas por el sistema, éste avisará al usuario y volverá a pedir un código. En caso contrario, el sistema recogerá el código y lo usara según convenga, haciendo que continúe la partida.

Pausar partida:
Mientras está jugando una partida, el cliente debe poder acceder a un menú de pausa. El sistema dirigirá al jugador a un menú de pausa con diversas opciones, entre ellas el poder continuar la partida.

Cerrar partida:
Mientras está la partida en pausa, el cliente debe poder cerrar la partida. El sistema pedirá una confirmación y en caso afirmativo, devolverá al cliente al menú principal del usuario. En caso contrario, lo devolverá al menú de pausa de la partida.

Guardar partida:
Mientras está la partida en pausa, el cliente debe poder guardar una partida ya iniciada. El sistema guarda los datos de la partida para que el cliente pueda volver a acceder a ella en otro momento.

Pedir pista:
Mientras está la partida en juego, el cliente debe poder pedir una pista que le ayude a encontrar la solución en mitad de una partida. El sistema generará una pista aleatoria teniendo en cuenta el código que debe solucionar el cliente y reducirá la puntuación del cliente a modo de penalización. La pista se mostrará por pantalla y devolverá al usuario al menú de escritura de código.

Editar datos de usuario:
El cliente, una vez ha iniciado sesión, debe poder editar los datos de su usuario. En caso del nombre, no debe coincidir con ningún usuario existente ni debe ser nulo. En caso de la contraseña, debe introducir la contraseña actual para verificar que no está suplantando al usuario.

ESTRUCTURAS DE DATOS Y ALGORITMOS

En el presente apartado se describen las estructuras de datos y algoritmos utilizados de más relevancia para el proyecto.

CPU

La clase CPU utiliza las siguientes estructuras de datos:

- *solutions: HashSet*
  - Permite mantener un Set no necesariamente ordenado de todas las posibles combinaciones de colores que son potenciales soluciones durante una partida concreta.
  - Su acceso, adición y eliminación de elementos constante permite mantener un rendimiento elevado a la hora de hacer una gran cantidad de operaciones en la ejecución del algoritmo
- *guesses: HashSet*
  - Permite mantener un Set no necesariamente ordenado de todas las posibles combinaciones de colores que son potenciales jugadas durante una partida concreta.
  - Su acceso, adición y eliminación de elementos constante permite mantener un rendimiento elevado a la hora de hacer una gran cantidad de operaciones en la ejecución del algoritmo
- *coincidencesByCorrection: HashMap*
  - Permite mantener una relación entre una corrección concreta y la cantidad de códigos dentro de guesses para una determinada solución dentro de solutions que comparten esa misma solución. El código que disponga del máximo número de coincidencias es aquel que eliminará menos códigos de la solución, requisito básico en la implementación del 5 guesses algorithm.
  - Su acceso, adición y eliminación de elementos constante permite mantener un rendimiento elevado a la hora de hacer una gran cantidad de operaciones en la ejecución del algoritmo
- *maxNotEliminatedByGuess: HashMap*
  - Permite mantener una relación entre una jugada y el número máximo de soluciones que no elimina, es decir, entre una jugada y el número mínimo de soluciones que elimina. La jugada que disponga del mínimo máximo es aquella que no elimina el mínimo máximo de soluciones, es decir, es aquella que elimina el máximo mínimo de ellas.
  - Su acceso, adición y eliminación de elementos constante permite mantener un rendimiento elevado a la hora de hacer una gran cantidad de operaciones en la ejecución del algoritmo

Los algoritmos utilizados en la clase CPU son:

- *permute*: algoritmo de backtracking que realiza las permutaciones (no combinaciones) de un set de colores dado.

    - Coste: O(n^k), donde k es el tamaño de cada código (número de colores) y n es el número de colores que se pueden utilizar
- *combinate*: algoritmo de backtracking que realiza las combinaciones (no permutaciones) de un set de colores dado.
    - Coste: O(n^min{k, n-k}), donde k es el tamaño de cada código (número de colores) y n es el número de colores que se pueden utilizar.
- *getCodeCorrect*: algoritmo de fuerza bruta que genera la corrección de un código comparando con coste O(k) primero todos los colores en las posiciones correspondientes entre la jugada y la solución (para obtener los pines negros de corrección) y luego comparando con coste O(k^2) cada elemento de la jugada con el de la solución (para obtener los pines blancos de corrección).
    - Coste: O(k^2), donde k es el tamaño de cada código (número de colores).
- *getCodeBreak*: algoritmo de búsqueda exhaustiva que genera la jugada con más probabilidades de eliminar el máximo de soluciones posibles del set de soluciones, de forma que asegura ganar en cerca de 5 turnos.
    - Coste: O(g·s·k^2), donde k es el tamaño de un código (número de colores), s es el tamaño del set de soluciones, g es el tamaño del set de jugadas potenciales. En la primera iteración, el coste es el de la función permute o combine, según la dificultad de juego.
- *getCodeMake*: algoritmo voraz de generación de códigos según la dificultad de la partida.
    - Coste: O(k), donde k es el tamaño de un código (número de colores).
- *getInitialGuess*: algoritmo voraz de generación del código de la primera jugada siguiendo criterios para maximizar la cantidad de soluciones que se eliminan en una única jugada al azar.
    - Coste: O(k), donde k es el tamaño de un código (número de colores).

Ranking

La clase Ranking utiliza las siguientes estructuras de datos:

- *topTen: LinkedList*
    - Permite mantener una lista con los diez jugadores que han hecho mejores puntuaciones en sus distintas partidas.
    - La LinkedList permite recorrer la lista con coste lineal (igual que una ArrayList), pero además permite inserciones y eliminaciones en tiempo constante (necesario para insertar jugadores en posiciones aleatorias del ranking).

Los algoritmos utilizados en la clase ranking son:

- *addToTopTen*: algoritmo de inserción dicotómica que inserta un jugador en la posición que le corresponde del ranking según su puntuación.
    - Coste: O(log r), donde r es el tamaño del ranking.

Code

La clase Code implementa dos atributos especiales:

- *unorderedHash: long*
  - Hash calculado a partir de los valores del código pero no a partir de su orden, de forma que se puede hacer una comparación de dos códigos con los mismos valores en posiciones distintas de forma constante en vez de cuadrática. Su principal función es acelerar las comparaciones entre códigos de corrección.
- *orderedHash: long*
  - Hash calculado a partir de los valores del código en función de su ordenación, de forma que se puede hacer una comparación de dos códigos de forma constante en vez de lineal. Su principal función es acelerar las comparaciones entre códigos de jugadas.

La clase Code utiliza los siguientes algoritmos:

- *calcUnorderedHash*: calcula el *unorderedHash* utilizando una identificación entre colores y enteros a partir de una distribución de hashes que evita completamente la colisión para valores dentro de los utilizados en el desarrollo del proyecto.
  - Coste: O(k), donde k es el tamaño de un código (número de colores)
- *calcOrderedHash*: calcula el *orderedHash* utilizando una identificación colores y enteros, así como el índice de sus posiciones en el código, utilizando una distribución que evita completamente la colisión para valores dentro de los utilizados en el desarrollo del proyecto.
  - Coste: O(k), donde k es el tamaño de un código (número de colores)

DIVISIÓN DEL TRABAJO

Inicialmente, decidimos dividirnos el trabajo por capas. Por un lado, Alejandro de Haro, puesto que se está especializando en computación, decidimos que empezara con la capa de dominio y, por el otro, Alex Sánchez hiciese capa de persistencia y Rafael Ramírez capa de presentación que en esta primera entrega consta de aquellas operaciones relacionadas con la interacción con la terminal. Debido a que obviamente la carga de trabajo de la capa dominio era bastante más elevada, en cuanto Alex Sánchez y Rafael Ramírez acabaron sus respectivas partes, ambos se pusieron a ayudar con la capa de dominio. Además, teníamos claro des del principio que las funcionalidades más complejas las teníamos que implementar juntos. Este es el caso de, por ejemplo, el algoritmo.

Cabe decir que, en referencia a la etapa de diseño, la realizamos conjuntamente con diferentes reuniones combinándolas con trabajo individual. Durante la etapa de implementación, también nos hemos ido reuniendo, aparte de las clases de laboratorio, para ir comentando los avances de cada uno y para repartir las nuevas tareas que aparecían. A continuación, vamos a comentar más concretamente quien ha hecho cada parte:

Alejandro de Haro

- CPU
- Board
- Code
- Turn
- BoardController
- DomainController
- PlayerController
- GameController
- AbstractPersistence

Alex Sánchez

- CPU
- Action
- CodeMake
- CodeBreak
- CodeCorrect
- Ranking
- PlayerController
- AbstractPersistence
- PlayerPersistence
- GamePersistence
- RankingPersistence

Rafael Ramírez

- CPU
- Game
- Player
- Human
- DomainController
- PresentationController
- GameController

En relación a la segunda entrega, la dinámica de trabajo ha sido la misma. Tal como ya habíamos realizado con anterioridad, realizamos diversas reuniones para dejar establecidas diferentes ideas en relación a las vistas y, principalmente, para realizar la división del trabajo. Inicialmente, decidimos que Rafael Ramírez y Alex Sánchez se encargaran de las vistas y Alejandro de los controladores. Obviamente, estos últimos conllevan más faena y, por ello, en cuanto Rafael Ramírez y Alex Sánchez acabaran las vistas, se pondrían a ayudar con los controladores. Finalmente, el resultado más concreto de esta repartición es el siguiente:

Alejandro de Haro

VISTAS

- ExitCurrentGameWarningView
- GameInProgressView
- LoadGameView
- LoadingView
- LogOutWarningView
- MainMenuView
- ShowCodeView

CONTROLADORES DE VISTA

- ExitCurrentGameWarningViewController
- GameInProgressViewController
- InitSessionViewController
- LoadGameViewController
- LoadingViewController
- LogOutWarningViewController
- MainMenuViewController
- NonRegisteringPresentationController
- PopUpController
- PresentationController
- RegisteringPresentationController
- ShowCodeViewController

Alex Sánchez

VISTAS

- EditUserView
- ErrorMessageView
- InitSessionView
- RankingView
- RegisterView
- SaveGameOverwriteView
- SaveGameViewController

CONTROLADORES DE VISTA

- EditUserViewController
- ErrorMessageViewController
- RankingViewController
- RegisterViewController
- SaveGameOverwriteViewController
- SaveGameViewController

Rafael Ramírez

VISTAS

- CloseProgramWarningView
- GameOverView
- HintView
- InfoView
- InitSessionView
- LogInView
- NewGameView
- PauseView

CONTROLADORES DE VISTA

- CloseProgramWarningViewController
- GameOverViewController
- HintViewController
- InfoViewController
- LogInViewController
- NewGameViewController
- PauseViewController

En conclusión, finalmente todos hemos cooperado para la consecución del proyecto ayudando en los diferentes controladores, en las diferentes vistas y en la documentación requerida de forma equitativa y acordadamente repartida.