INFORME DEL PROYECTO

Incluye la documentación correspondiente a la implementación del proyecto.

2248 Game.

Tabla de contenido

INTRODUCCIÓN:

El proyecto consiste en implementar una aplicación que permita al usuario jugar al 2248; además de implementar las reglas y la dinámica básicas del juego (movida básica y su efecto).

Fue implementado en HTML, JavaScript, Prolog y CSS; donde el foco fue puesto principalmente en la implementación en Prolog de la funcionalidad básica del juego, combinándolo con componentes en React.

EL JUEGO:

El 2248 tiene como objetivo combinar bloques numerados con potencias de dos, para crear bloques de mayor valor y ganar puntos.

Estos bloques, o squares, son visibles a través de una grilla con la que puede interactuar en todo momento el jugador.

La **movida básica** permite al jugador trazar un camino a partir de un bloque de la grilla. Este camino puede trazarse con bloques adyacentes a este, ya sea horizontal, vertical o diagonal.

El camino arranca conectando 2 números iguales, y luego cada número del camino es igual al anterior o es la potencia de 2 siguiente a dicho número anterior.

El **efecto** que se genera al concretar un camino es el siguiente:

- Se eliminan todos los bloques del camino y en el lugar del último se genera un nuevo bloque cuyo valor V es la menor potencia de 2 mayor o igual que la sumatoria de los números del camino.
- Los bloques que se encuentran por encima de lo eliminados ocupan su lugar, en una especie de “efecto gravedad”, y se generan por encima nuevos bloques generados de manera aleatoria.
- El valor V es sumado posteriormente al puntaje total actual.

REQUERIMIENTOS:

Extendimos la implementación molde (React + Prolog) provista por la cátedra para cumplir con los siguientes requerimientos de funcionalidad:

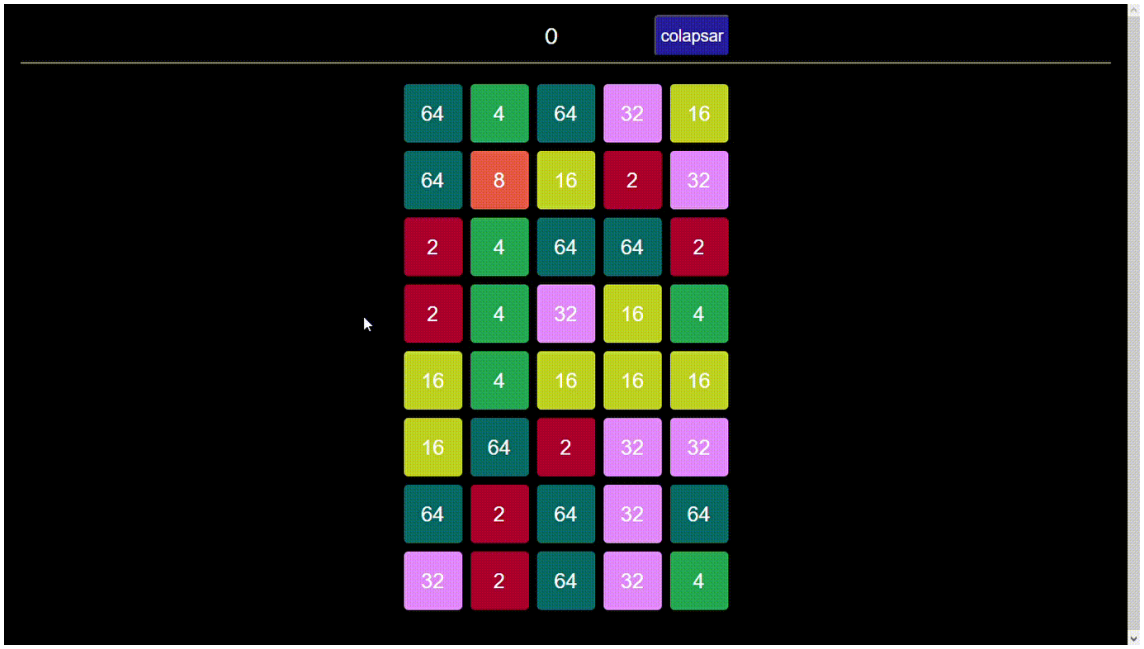
- Mostrar progresivamente, en mínimamente dos etapas. Una etapa donde se muestre la grilla con los bloques eliminados del camino trazado, junto con la aparición del bloque resultante; y otra con el efecto gravedad aplicado, junto con la generación de los nuevos bloques aleatorios.
- Mientras el usuario va trazando el camino, ir mostrando un bloque con el valor que va generando el camino a medida que se va trazando.
- Agregar un booster “Colapsar iguales”, cuyo efecto consta en consiste en colapsar todos los grupos de bloques adyacentes y de igual valor, reemplazándolos por nuevos bloques aleatorios, cuyos valores son calculados como la menor potencia de 2 mayor o igual a la sumatoria de los bloques del grupo, y se ubicado en el lugar más abajo y más a la derecha del grupo.

¿CÓMO JUGAR?

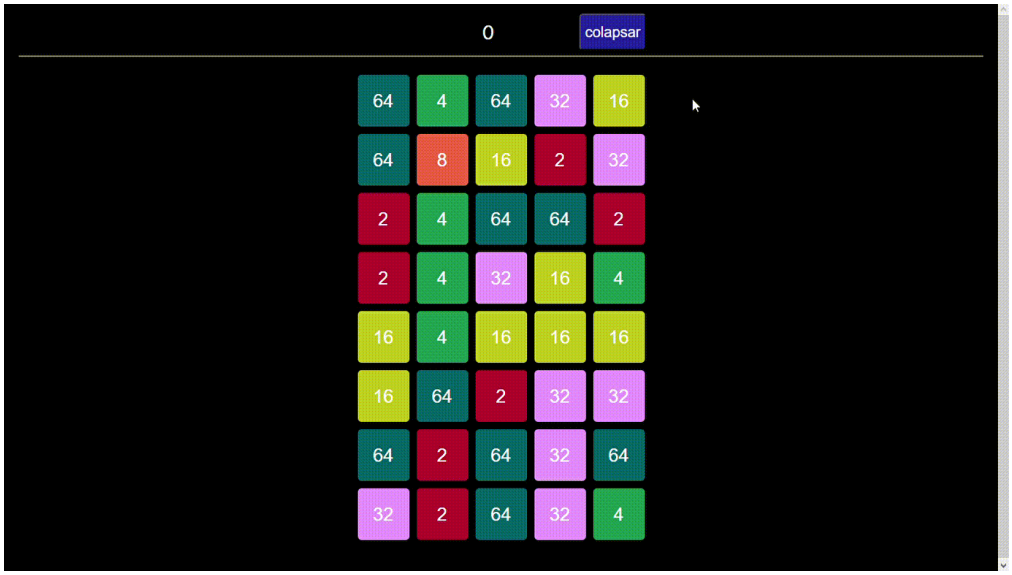
Para trazar un camino, hacer clic sobre el bloque donde se quiera comenzar a trazar el camino. Luego, al pasar el cursor del mouse sobre los bloques adyacentes, se generará un camino visible para el jugador en caso de que el adyacente sea válido.

Podrá ver reflejado el puntaje del camino actual en la parte superior central de la pantalla. Para volver atrás un bloque, deslizar el cursor sobre el bloque anterior al actual. Para cancelar el trazado de un camino, apretar la tecla “Esc” en el teclado.

Para confirmar un camino, hacer clic sobre el bloque en el que se encuentra el cursor. Una vez hecho esto, se verá por pantalla el efecto generado al eliminar el camino, y se actualizará el puntaje, también visible en la parte central superior de la pantalla.



Se puede reiniciar la página para volver a empezar desde cero. Para activar el booster “colapsar iguales”, presionar el botón “colapsar” ubicado en la parte superior derecha de la pantalla. Una vez activado, el botón estará deshabilitado hasta que se termine de realizar el efecto.



IMPLEMENTACIÓN EN PROLOG:

Todas las implementaciones de las funcionalidades en Prolog se encuentran en el archivo proylcc.pl.

Movida básica:

Predicado Join/4:

Join(+Grid, +NumOfColumns, +Path, -Rgrids).

Es el predicado que se encarga de la **movida básica**. Recibe la grilla Grid actual (la que se muestra por pantalla), el número de columnas de la misma, y el camino Path que acaba de trazar el jugador.

Este predicado se encarga de, en base a los índices de los elementos del camino, ir cambiando su valor a 0, para poder eliminarlos de la grilla y luego aplicar el efecto gravedad.

- ✓ **Efecto gravedad:**

  La estrategia que empleamos para implementar la gravedad consta de dividir la grilla en columnas, con el predicado gridToColumns/4, el cual devuelve una lista con las listas correspondientes a las columnas, que va a ser usada por el predicado gravityFalls/6 para aplicar la gravedad.

Predicado gravityFalls/6:

gravityFalls(+Grid, +ColumnsList, +AuxGrids, -ReturnGrids, +Min, +Max).

Este predicado recorre cada bloque de la grilla, por columnas; y mientras haya ceros en la grilla (elementos eliminados), les aplica **gravedad** llamando al predicado gravityOneSquare/6, el cuál realiza un efecto gravedad de un único bloque por cada columna.

Cuando ya no hay mas elementos eliminados, unifica las columnas en una única lista, con el predicado ColumnsToGrid/5, que devuelve la grilla GridGravity, la cual pasará a reemplazar a la grilla original.

Para más detalles acerca de los predicados auxiliares utilizados, revisar la documentación presente en los comentarios del código Prolog.

Colapsar iguales:

Predicado collapse/3:

collapse(Grid, NumOfColumns, RGrids)

Este predicado es el encargado de recorrer la grilla, buscar los grupos de adyacentes de mismo valor, eliminarlos, generar el bloque en la posición más abajo y a la derecha del grupo de a acuerdo a la funcionalidad ya mencionada en la sección de requerimientos, y generar los nuevos bloques.

Lo primero que hace es llamar al predicado **shellAdyacents/6**, el cual recorre la grilla elemento por elemento, chequea que no esté visitado y, en caso de no estarlo, se llama a **findGroups/5** que busca recursivamente todo el grupo adyacente al elemento que se pasa como parámetro.

El control de visitados se realiza mediante una lista Visitados. Una vez encontrado un grupo, se concatena completo a la lista de visitados.

Además, una vez conseguidos todos los grupos, se utiliza **deleteAllPaths/3**, el cual se encarga primeramente de eliminar el elemento de cada grupo con índice más alto, para luego reemplazarlo por el bloque con el valor actualizado. Luego, remueve ese Path de la grilla con **deletePathInGrid/4**, y realiza una llamada recursiva hasta que no haya más grupos a eliminar.

Luego, se utiliza la misma estrategia que en la movida básica para realizar el efecto de gravedad y generar los bloques nuevos con el resto de bloques eliminados.

Remover valores bajos:

Además de los requerimientos mínimos del proyecto, agregamos una funcionalidad pasiva que se encarga de ir removiendo del tablero, luego de cada jugada o luego de accionar el “colapsar iguales”, aquellos bloques cuyo valor es igual a 2 elevado a la mínima potencia de la grilla. Estos se remueven si la diferencia entre la potencia mínima y la potencia máxima de la grilla es mayor a 11.

La finalidad de esta función es evitar que queden sueltos en la grilla bloques de valores muy distantes, para así mejorar la jugabilidad.

Para ello, implementamos el predicado **removeLowValues/5,** el cual recibe la grilla, las potencias mínima y máxima, y la cantidad de columnas; para devolver la grilla actualizada en caso de tener que remover algún valor.

Primero se calcula la diferencia entre las potencias y se controla que la diferencia sea mayor a 11. Caso contrario, se mantiene la grilla ingresada originalmente.

Luego, se buscan en la grilla todos los bloques con valor 2**MinPower y se llama al predicado **deletePathInGrid/4** para setear todos esos valores en 0. Posteriormente se hace uso del predicado **gravityFalls/6** que aplica la gravedad correspondiente. Finalmente, se actualiza la mínima potencia a la potencia siguiente.

Consultar los comentarios del código Prolog para más detalles acerca de los predicados auxiliares utilizados.

IMPLEMENTACIÓN EN REACT:

El código React se comunica con el servidor en Prolog a través de Queries.

El archivo Game.js contiene queries para inicializar el servidor, para mostrar el trazado de un camino, para actualizar el score una vez que el usuario confirma un camino; etc.

En base al código suministrado por la cátedra, nos encargamos de agregar funciones (componentes de React) para poder realizar ciertas funcionalidades adicionales.

La función collapse() que envía la query al presionar el botón “colapsar”. Esta función pone en espera al botón para que no pueda ser utilizado hasta que finalice toda la animación, y anima el efecto correspondiente a la funcionalidad de Colapsar Iguales.

La const scoreOrSquare es la utilizada para determinar si en la parte superior de la pantalla se debe mostrar el puntaje o el square con el valor actual del camino. Si no se está trazando un camino, se muestra el puntaje actual y, en caso contrario, se muestra un square con el valor del camino trazado hasta el momento.

En el archivo util.js modificamos la función numberToColor(num) que recibe el valor de un square y, en base a dicho valor, le asigna un color de una lista de colores preestablecida. Ésta contiene 12 colores, ya que la funcionalidad de remover valores bajos no permite que hayan más de 11 squares diferentes en la grilla, por lo que se asegura que no van a haber colores repetidos en la grilla, mejorando la visualización de la misma y la jugabilidad.