<Dídac Maldonado Parnau 1633861>
<Oscar Melendez Codina 1632380>
<DILLUNS 10:30>
<2048>

Funcionalitat 1: Control del moviment de les fitxes al tauler de joc (esquerra, dreta, amunt, avall)

- **Descripció**: S'ha implementat el comportament per gestionar els moviments del tauler en quatre direccions (esquerra, dreta, amunt, avall), incloent la compressió i fusió de fitxes, així com l'aparició de noves fitxes en posicions buides després d'un moviment vàlid.
- **Localització**:
  - **Arxiu**: `cat.uab.tqs._2048.model.GameBoard`
  - **Mètodes**:
    - `handleSwipeLeft()`
    - `handleSwipeRight()`
    - `handleSwipeUp()`
    - `handleSwipeDown()`
    - `processMovements(int[] row, String movement)`
    - `compress(int[] row)`
    - `merge(int[] row)`
- **Test**:
  - **Arxiu**: `cat.uab.tqs._2048.model.TestGameBoard`
  - **Mètodes de test**:
    - `testSwipeLeft()`
    - `testSwipeRight()`
    - `testSwipeUp()`
    - `testSwipeDown()`
    - `testProcessMovementsLeft()`
    - `testProcessMovementsRight()`
    - `testProcessMovements()`
    - `testCompress()`
    - `testMerge()`
    - **Tipus de test**: Caixa negra i caixa blanca.
    - **Tècniques utilitzades**:
      - Particions equivalents per validar moviments de fitxes segons diferents configuracions inicials.
      - A testSwipeLeft comprovem decision i condition coverage amb els moviments de caselles buides i plenes.
      - Al metode testCompress tenim loop testing simple per provar que el metode comprimeix les files correctament sense importar on estan els zeros.
      - A testProcessMovements hem fet pairwise testing per a poder verificar el comportament del mètode amb diferents combinacions de dues variables que poden influir en el resultat, com els valors de la fila (com es distribueixen els números) i el moviment (a l'esquerra o a la dreta).

Decision i Condition Coverage → testSwipeLeft()

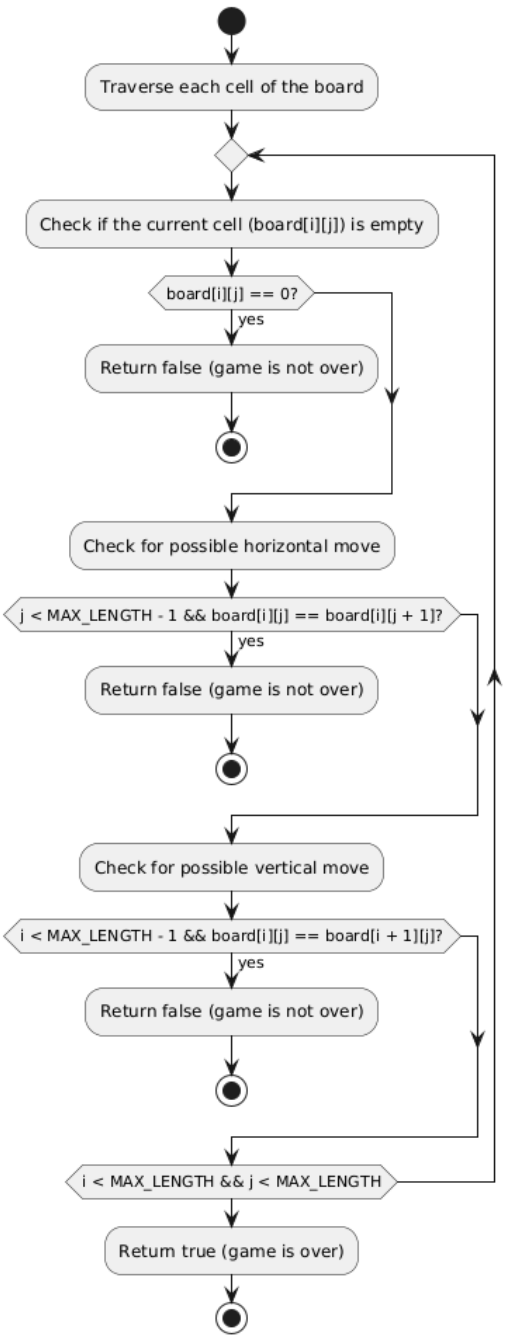
```
33      public void handleSwipeLeft() {
34          movement = "left";
35          boolean hasBoardChanged = false;
36
37          for (int row = 0; row < MAX_LENGTH; row++) {
38              boolean rowChanged = processMovements(board[row], movement);
39              if (rowChanged) {
40                  hasBoardChanged = true;
41              }
42          }
43
44          if (hasBoardChanged) {
45              spawnTile();
46              /* int[][] predefinedTiles = new int[][] {
47                      { 0, 3, 2 } // row, column, value
48              };
49
50              MockSpawnTile mockBoard = new MockSpawnTile(predefinedTiles);
51
52
53              mockBoard.mergePredefinedTiles(getBoard(), predefinedTiles); */
54          }
55
56          isGameOver = isGameOver(board);
57      }
```

Loop Testing Simple → testCompress()

```
183         public int[] compress(int[] row) {
184             int[] newRow = new int[MAX_LENGTH];
185             int index = 0;
186
187             for (int i = 0; i < MAX_LENGTH; i++) {
188                 if (row[i] != 0) {
189                     newRow[index] = row[i];
190                     index++;
191                 }
192             }
193             return newRow;
194         }
```

Funcionalitat 2: Configuració i validació del tauler

- **Descripció**: S'ha afegit la funcionalitat per establir un nou tauler, assegurant que tingui la mida adequada (4x4), i verificar si el joc ha acabat.
- **Localització**:
    - **Arxiu**: `cat.uab.tqs._2048.model.GameBoard`
    - **Mètodes**:
        - `setGameBoard(int[][] newBoard)`
        - `isGameOver(int[][] board)`
- **Test**:
    - **Arxiu**: `cat.uab.tqs._2048.model.TestGameBoard`
    - **Mètodes de test**:
        - `testSetGameBoardValidBoard()`
        - `testSetGameBoardInvalidBoard()`
        - `testIsGameOver()`
        - `testGameNotOverWhenEmptyTileExists()`
        - **Tipus de test**: Caixa negra i caixa blanca.
        - **Tècniques utilitzades**:
            - Test de valors límit per comprovar taulers vàlids i invàlids.
            - Decision i condition coverage amb *testSetGameBoardValidBoard* i *testSetGameBoardInvalidBoard* per comprovar que la mida és vàlida o no.
            - Al metode isGameOver fem decision, condition i path coverage.
            - En el test *testIsGameOver()* fem loop testing anidat, assegurant així que recorrem totes les cel·les del tauler. confirmant així també que la variable isGameOver s'actualitza.


Traverse each cell of the board
Check if the current cell (board[i][j]) is empty
board[i][j] == 0?
yes
Return false (game is not over)
Check for possible horizontal move
j < MAX_LENGTH - 1 && board[i][j] == board[i][j + 1]?
yes
Return false (game is not over)
Check for possible vertical move
i < MAX_LENGTH - 1 && board[i][j] == board[i + 1][j]?
yes
Return false (game is not over)
i < MAX_LENGTH && j < MAX_LENGTH
Return true (game is over)

Loop Testing, condition, path, decision → isGameOver()

```java
public boolean isGameOver(int[][] board) {
    // Traverse each cell of the board
    for (int i = 0; i < MAX_LENGTH; i++) {
        for (int j = 0; j < MAX_LENGTH; j++) {
            // If there is an empty cell, the game is not over
            if (board[i][j] == 0) {
                return false;
            }

            // Check possible moves horizontally and vertically
            if (j < MAX_LENGTH - 1 && board[i][j] == board[i][j + 1]) { // Right
                return false;
            }
            if (i < MAX_LENGTH - 1 && board[i][j] == board[i + 1][j]) { // Down
                return false;
            }
        }
    }
    // If there are no empty cells and no possible moves, the game is over
    return true;
}
```

Decision i condition (Tests *testSetGameBoardValidBoard* i *testSetGameBoardInvalidBoard)*

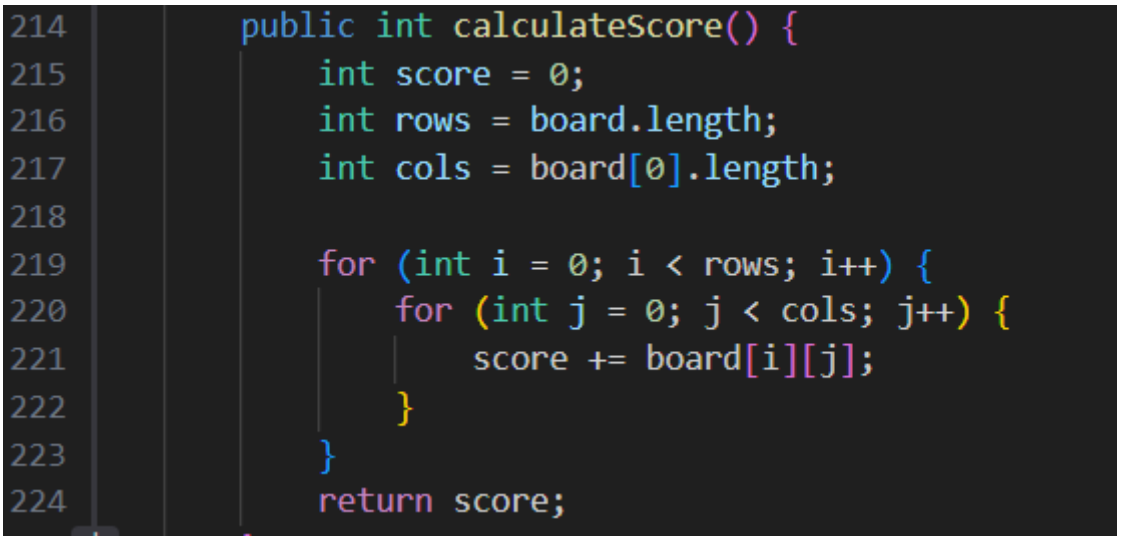
```java
public void setGameBoard(int[][] newBoard) {
    if (newBoard.length == MAX_LENGTH && newBoard[0].length == MAX_LENGTH) {
        board = newBoard;
    } else {
        throw new IllegalArgumentException("Invalid board size.");
    }
}
```

Funcionalitat 3: Càlcul de la puntuació de la partida.

1. **Descripció:** S'ha afegit aquesta funcionalitat per tal de poder tenir a la vista després de cada moviment la puntuació que portem.
2. Localització:
    - Arxiu: cat.uab.tqs._2048.model.GameBoard
    - Mètodes:
        - calculateScore()

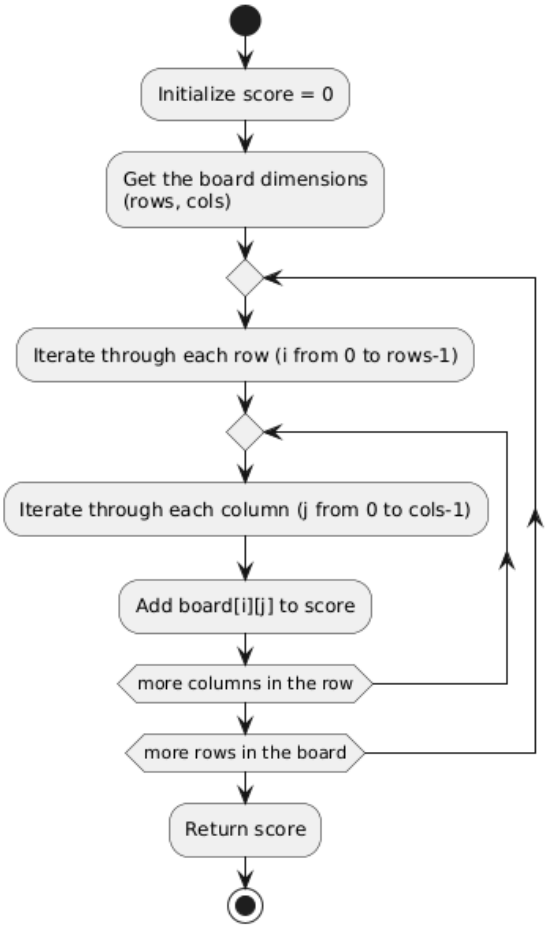
- Test:
    - Arxiu: cat.uab.tqs._2048.model.TestGameBoard
    - Mètodes de Test:
        - testCalculateScore()
    - Tipus de test: Caixa negra i caixa blanca
    - Tècniques utilitzades:
        - Particions equivalents: Es realitzen proves per diferents casos representatius del càlcul de puntuació: tauler buit (totes les cel·les amb valor 0), tauler amb valors grans.
        - Al mètode calculateScore fem path coverage.
        - Al mètode calculateScore fem loop testing anidat per assegurar que es comproven totes les caselles per fer la puntuació.

Path Coverage, Loop Testing. → calculateScore()



```
214     public int calculateScore() {
215         int score = 0;
216         int rows = board.length;
217         int cols = board[0].length;
218 
219         for (int i = 0; i < rows; i++) {
220             for (int j = 0; j < cols; j++) {
221                 score += board[i][j];
222             }
223         }
224         return score;
```

Diagrama UML Test Calculate Score:



Funcionalitat 4: Generació de noves fitxes al tauler

- **Descripció**: Implementació del mètode `spawnTile()` per generar fitxes noves en posicions buides del tauler. Per a poder testejar això hem hagut de fer un mock object (***mockSpawnTile***) per a poder controlar la generació de noves fitxes de manera no aleatòria, així podent testejar el seu comportament.
- **Localització**:
  - **Arxiu**: `cat.uab.tqs._2048.model.GameBoard`
  - **Mètode**: `spawnTile()`
- **Test**:
  - **Arxiu**: `cat.uab.tqs._2048.model.GameBoardTest`
  - **Mètodes de test**:
    - `testSpawnTileControlled()`
    - `testInitialBoardHasTwoTiles()`
    - **Tipus de test**: Mock objects
    - **Tècniques utilitzades**:

- *Mockups* per controlar la generació de fitxes en posicions predefinides. (CLASSE MOCKSPAWNTILE)
- Verificació de propietats post-condició (nombre de fitxes generades).

Funcionalitat 5: Inversió d'arrays per moviments de dreta i avall

- **Descripció**: Implementació del mètode `reverseArray()` per invertir l'ordre de les fitxes quan es processen moviments cap a la dreta o cap avall. D'aquesta manera ens estalviem fer noves funcions pels moviments, podent així reutilitzar la lògica del processMovements només girant els arrays.
- **Localització**:
  - **Arxiu**: `cat.uab.tqs._2048.model.GameBoard`
  - **Mètode**: `reverseArray(int[] row)`
- **Test**:
  - **Arxiu**: `cat.uab.tqs._2048.model.GameBoardTest`
  - **Mètodes de test**:
    - `testReverseArray()`
    - **Tipus de test**: Caixa negra i caixa blanca.
    - **Tècniques utilitzades**:
      - Particions equivalents perquè donat un array com a input es retorna l'array del revés.
      - Loop testing simple per comprovar que el metode fa l'invers de l'array amb totes les mides possibles

Loop Testing → **testReverseArray()**

```
150     public void reverseArray(int[] row) {
151         int start = 0;
152         int end = row.length - 1;
153 
154         while (start < end) {
155             int temp = row[start];
156             row[start] = row[end];
157             row[end] = temp;
158             start++;
159             end--;
160         }
161     }
```

.

Funcionalitat 6: Processar els inputs de l'usuari

S'ha implementat la lògica per processar les entrades de l'usuari i actualitzar l'estat del tauler en el joc 2048. Aquesta funcionalitat permet gestionar moviments del jugador en les quatre direccions (`w`, `a`, `s`, `d`) i actualitzar la vista en conseqüència.

Localització:

- **Arxiu:** `GameController.java`
- **Classe:** `GameController`
- **Mètode:** `processInput(String input)`

Test:

- **Arxiu:** `GameControllerTest.java`
- **Classe:** `GameControllerTest`
- **Mètodes de test associats:**
  - `testProcessInputSwipeUp()`
  - `testProcessInputSwipeLeft()`
  - `testProcessInputSwipeDown()`
  - `testProcessInputSwipeRight()`
- **Tipus de test:**
  - **Caixa negra:**
  - **Tècniques utilitzades:**
    - Particions equivalents per validar diferents escenaris de moviment i comprovació detallada de l'estat del tauler després de cada acció.

Funcionalitat 7: Gestionar Múltiples Entrades (PROVA DE JOC)

S'ha verificat que el controlador pot gestionar múltiples entrades seqüencials i que la vista s'actualitza correctament després de cada moviment.

Localització:

- **Arxiu:** `GameControllerTest.java`
- **Classe:** `GameControllerTest`
- **Mètodes:**
  - `testProcessMultipleInputs()`
  - `testMockGameview()`

Test:

- **Tipus de test:**
  - *Caixa negra*: Validació dels resultats finals després d'una seqüència d'entrades, sense considerar la implementació interna.
  - *Mockups*: Verificació que el mètode `update` de la vista s'invoca el nombre esperat de vegades.

Comentaris que volem fer notar:

- En els tests amb mocks, s'han capturat correctament els arguments per comparar-los amb els resultats esperats, assegurant la coherència entre model i vista.

Funcionalitat 8: Visualització del taulell

Funcionalitat:
S'ha implementat la visualització del tauler de joc a la consola amb format tabulat. El mètode update mostra el tauler, substituint els espais buits amb punts (.), i proporciona indicacions per al moviment del jugador. A més, el mètode showScore permet mostrar la puntuació actual del joc.

Localització:

- **Arxiu:** GameView.java
- **Classe:** GameView
- **Mètodes desenvolupats:**
  - update(int[][] board)
  - showScore(int score)

Test:

- **Arxiu:** GameViewTest.java
- **Classe:** GameViewTest
- **Mètodes de test associats:**
  - testUpdateDisplaysBoardCorrectly()
  - testUpdateDisplaysEmptyBoard()
  - testUpdateDisplaysFullBoard()
- **Descripció del tipus de test:**
  - **Caixa negra:** Validació de la sortida generada en la consola segons l'entrada proporcionada, sense considerar la implementació interna dels mètodes.
  - **Tècniques utilitzades:**
    - *Particions equivalents*: Proves amb un tauler parcialment ple, completament buit i completament ple per cobrir diversos escenaris.

Comentaris que volem fer notar:

- El format de la sortida generada està dissenyat per ser clar i fàcilment llegible en un entorn de consola, amb punts per indicar caselles buides.
- Les proves cobreixen un ampli rang d'escenaris del joc, però es podria considerar la inclusió de tests amb inputs irregulars (p.ex., taulers no quadrats) per augmentar la robustesa.