

INFORME PRÀCTICA - 2048 GAME

Tipus de tests:

Caixa negra:

- **Particions equivalents, valors límit i frontera:** Es divideix el domini en particions, correctes y no correctes. En aquest test, mirem els valors fronteres, els valors limits i valors diferenciats dins dels límits. En el nostre cas, com tenim una matriu 4x4, els valors correctes son entre 0 i 3 tots dos inclòs. Per això fem aquests tests:

Per a que els tests no donin error, fem assertThrows en els tests que donen errors. En aquestes captures es mostren els valors -10, -1, 0, 1, 3, 4, 5, 10.

```
assertEquals(b[0][0].getValue(), actual: 2);
assertEquals(b[0][0].getColor(), InfoGame.Color.GREEN);

assertEquals(b[3][3].getValue(), actual: 2);
assertEquals(b[3][3].getColor(), InfoGame.Color.GREEN);

board.moveBlock( i: 1, j: 1, c: 'a');

assertThrows(AssertionError.class, () -> board.moveBlock( i: -1, j: 0, c: 'w'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: 0, j: 0, c: 'x'));
b[0][0] = null;
assertThrows(AssertionError.class, () -> board.moveBlock( i: 0, j: 0, c: 's'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: -1, j: -1, c: 'w'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: 5, j: 5, c: 'w'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: 4, j: 4, c: 'w'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: 10, j: 0, c: 'w'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: -10, j: -10, c: 'w'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: -10, j: -10, c: 'f'));
assertThrows(AssertionError.class, () -> board.moveBlock( SIZE, j: 0, c: 'w'));
assertThrows(AssertionError.class, () -> board.moveBlock( i: 0, SIZE, c: 'a'));
```

- **Pairwise:** És una manera de fer testing amb combinacions de diferents paràmetres sense tenir la necessitat de fer totes les possibilitats assegurant que no hi hagi cap error. En el nostre cas, hem fet en el canMove utilitzant la pàgina web

<https://pairwise.teremokgames.com/>:

0 0 w

0 1 a

0 2 s
0 3 d
1 1 s
1 2 d
1 3 w
1 0 a
2 2 w
2 3 a
2 0 s
2 1 d
3 3 s
3 0 d
3 1 w
3 2 a

En el nostre cas, com hem inicialitzat un taulell amb 2 valors, hi ha casos on s'intenta fer un canMove d'un bloc buit, és a dir, valor = 0, per tant les precondicions fa que surti error i evitem l'error del test amb assertThrows.

```
//pairwise
assertFalse(board.canMove( i: 0, j: 0, c: 'w'));
assertThrows(AssertionError.class, () -> board.canMove( i: 0, j: 1, c: 'a'));
assertThrows(AssertionError.class, () -> board.canMove( i: 0, j: 2, c: 's'));
assertThrows(AssertionError.class, () -> board.canMove( i: 0, j: 3, c: 'd'));
assertThrows(AssertionError.class, () -> board.canMove( i: 1, j: 1, c: 's'));
assertTrue(board.canMove( i: 1, j: 2, c: 'd'));
assertThrows(AssertionError.class, () -> board.canMove( i: 1, j: 3, c: 'w'));
assertThrows(AssertionError.class, () -> board.canMove( i: 1, j: 0, c: 'a'));
assertThrows(AssertionError.class, () -> board.canMove( i: 2, j: 2, c: 'w'));
assertThrows(AssertionError.class, () -> board.canMove( i: 2, j: 3, c: 'a'));
assertThrows(AssertionError.class, () -> board.canMove( i: 2, j: 0, c: 's'));
assertThrows(AssertionError.class, () -> board.canMove( i: 2, j: 1, c: 'd'));
assertFalse(board.canMove( i: 3, j: 3, c: 's'));
assertThrows(AssertionError.class, () -> board.canMove( i: 3, j: 0, c: 'd'));
assertThrows(AssertionError.class, () -> board.canMove( i: 3, j: 1, c: 'w'));
assertThrows(AssertionError.class, () -> board.canMove( i: 3, j: 2, c: 'a'));
```

Caixa blanca:

- **Statement coverage:** Serveix per veure si els tests passen per totes les línies de codi en forma de percentatge.

En el nostre codi, tenim invariants que eviten que per exemple el bloc mai tingui un valor diferent als que tenim assignats, és a dir tampoc poden ser negatius. Això fa que precondicions de altres classes mai es puguin testejar i que no passi per aquestes línies de codi i que el coverage no surti 100%. Proves:

Class InfoGame Color:

```
Color(int value) { 24 usages  ↗ Adrián Valverde
    //precondition
    assert(value >= 0 && value <= 11);

    this.value = value;

    //postcondition
    assert (value == this.value && this.value >= 0 && this.value <= 11);
}

public class Block 86 usages  ↗ Adrián Valverde +1
{
    private int value;  15 usages
    private InfoGame.Color color;  11 usages
    ●
    private boolean invariant() {  6 usages  ↗ Adrián Valverde
        return ((value >= 0 && value <= 2048) && (color.getValue() == log2(value)));
    }
}
```

Izan Caballer Jimenez - 1710282
Adrián Valverde Ambrosio - 1707952

```
17    private boolean invariant() { 20 usages by Adrián Valverde
18        if (board == null || board.length != SIZE) return false;
19
20        for (int i = 0; i < SIZE; i++) {
21            if (board[i] == null || board[i].length != SIZE) return false;
22
23            for (int j = 0; j < SIZE; j++) {
24                Block block = board[i][j];
25                if (block == null) return false;
26
27                int value = block.getValue();
28                InfoGame.Color color = block.getColor();
29
30                if (value < 0 || value > 2048) return false;
31                if ((value == 0 && color != InfoGame.Color.NONE) ||
32                    (value > 0 && color == InfoGame.Color.NONE)) return false;
33            }
34        }
35    }
36
37    public void printBoard() {
38        for (int i = 0; i < SIZE; i++) {
39            for (int j = 0; j < SIZE; j++) {
40                System.out.print(board[i][j]);
41            }
42            System.out.println();
43        }
44    }
45
46    public void rotate() {
47        for (int i = 0; i < SIZE; i++) {
48            for (int j = 0; j < i; j++) {
49                swap(i, j);
50            }
51        }
52    }
53
54    public void swap(int i, int j) {
55        for (int k = 0; k < SIZE; k++) {
56            for (int l = 0; l < k; l++) {
57                swap(i, j, k, l);
58            }
59        }
60    }
61
62    public void swap(int i, int j, int k, int l) {
63        if (i == k && j == l) return;
64
65        if (i == k) {
66            if (j < l) {
67                swap(i, j, k, l - 1);
68                swap(i, j + 1, k, l);
69            } else {
70                swap(i, j, k, l + 1);
71                swap(i, j - 1, k, l);
72            }
73        } else {
74            if (j < l) {
75                swap(i, j, k, l - 1);
76                swap(i, j + 1, k, l);
77            } else {
78                swap(i, j, k, l + 1);
79                swap(i, j - 1, k, l);
80            }
81        }
82    }
83
84    public void moveUp() {
85        for (int i = 0; i < SIZE; i++) {
86            for (int j = 0; j < SIZE; j++) {
87                if (board[i][j] == null) continue;
88
89                for (int k = j; k < SIZE; k++) {
90                    if (board[i][k] == null) continue;
91
92                    if (board[i][j].value == board[i][k].value) {
93                        board[i][j].value *= 2;
94                        board[i][k].value = null;
95                    } else if (board[i][j].value < board[i][k].value) {
96                        board[i][j].value = board[i][k].value;
97                        board[i][k].value = null;
98                    }
99                }
100            }
101        }
102    }
103
104    public void moveDown() {
105        for (int i = 0; i < SIZE; i++) {
106            for (int j = 0; j < SIZE; j++) {
107                if (board[i][j] == null) continue;
108
109                for (int k = j; k >= 0; k--) {
110                    if (board[i][k] == null) continue;
111
112                    if (board[i][j].value == board[i][k].value) {
113                        board[i][j].value *= 2;
114                        board[i][k].value = null;
115                    } else if (board[i][j].value < board[i][k].value) {
116                        board[i][j].value = board[i][k].value;
117                        board[i][k].value = null;
118                    }
119                }
120            }
121        }
122    }
123
124    public void moveLeft() {
125        for (int i = 0; i < SIZE; i++) {
126            for (int j = 0; j < SIZE; j++) {
127                if (board[i][j] == null) continue;
128
129                for (int k = i; k < SIZE; k++) {
130                    if (board[k][j] == null) continue;
131
132                    if (board[i][j].value == board[k][j].value) {
133                        board[i][j].value *= 2;
134                        board[k][j].value = null;
135                    } else if (board[i][j].value < board[k][j].value) {
136                        board[i][j].value = board[k][j].value;
137                        board[k][j].value = null;
138                    }
139                }
140            }
141        }
142    }
143
144    public void moveRight() {
145        for (int i = 0; i < SIZE; i++) {
146            for (int j = 0; j < SIZE; j++) {
147                if (board[i][j] == null) continue;
148
149                for (int k = i; k >= 0; k--) {
150                    if (board[k][j] == null) continue;
151
152                    if (board[i][j].value == board[k][j].value) {
153                        board[i][j].value *= 2;
154                        board[k][j].value = null;
155                    } else if (board[i][j].value < board[k][j].value) {
156                        board[i][j].value = board[k][j].value;
157                        board[k][j].value = null;
158                    }
159                }
160            }
161        }
162    }
163
164    public void printBoard() {
165        for (int i = 0; i < SIZE; i++) {
166            for (int j = 0; j < SIZE; j++) {
167                System.out.print(board[i][j]);
168            }
169            System.out.println();
170        }
171    }
172
173    public void rotate() {
174        for (int i = 0; i < SIZE; i++) {
175            for (int j = 0; j < i; j++) {
176                swap(i, j);
177            }
178        }
179    }
180
181    public void swap(int i, int j) {
182        for (int k = 0; k < SIZE; k++) {
183            for (int l = 0; l < k; l++) {
184                swap(i, j, k, l);
185            }
186        }
187    }
188
189    public void swap(int i, int j, int k, int l) {
190        if (i == k && j == l) return;
191
192        if (i == k) {
193            if (j < l) {
194                swap(i, j, k, l - 1);
195                swap(i, j + 1, k, l);
196            } else {
197                swap(i, j, k, l + 1);
198                swap(i, j - 1, k, l);
199            }
200        } else {
201            if (j < l) {
202                swap(i, j, k, l - 1);
203                swap(i, j + 1, k, l);
204            } else {
205                swap(i, j, k, l + 1);
206                swap(i, j - 1, k, l);
207            }
208        }
209    }
210
211    public void moveUp() {
212        for (int i = 0; i < SIZE; i++) {
213            for (int j = 0; j < SIZE; j++) {
214                if (board[i][j] == null) continue;
215
216                for (int k = j; k < SIZE; k++) {
217                    if (board[i][k] == null) continue;
218
219                    if (board[i][j].value == board[i][k].value) {
220                        board[i][j].value *= 2;
221                        board[i][k].value = null;
222                    } else if (board[i][j].value < board[i][k].value) {
223                        board[i][j].value = board[i][k].value;
224                        board[i][k].value = null;
225                    }
226                }
227            }
228        }
229    }
230
231    public void moveDown() {
232        for (int i = 0; i < SIZE; i++) {
233            for (int j = 0; j < SIZE; j++) {
234                if (board[i][j] == null) continue;
235
236                for (int k = j; k >= 0; k--) {
237                    if (board[i][k] == null) continue;
238
239                    if (board[i][j].value == board[i][k].value) {
240                        board[i][j].value *= 2;
241                        board[i][k].value = null;
242                    } else if (board[i][j].value < board[i][k].value) {
243                        board[i][j].value = board[i][k].value;
244                        board[i][k].value = null;
245                    }
246                }
247            }
248        }
249    }
250
251    public void moveLeft() {
252        for (int i = 0; i < SIZE; i++) {
253            for (int j = 0; j < SIZE; j++) {
254                if (board[i][j] == null) continue;
255
256                for (int k = i; k < SIZE; k++) {
257                    if (board[k][j] == null) continue;
258
259                    if (board[i][j].value == board[k][j].value) {
260                        board[i][j].value *= 2;
261                        board[k][j].value = null;
262                    } else if (board[i][j].value < board[k][j].value) {
263                        board[i][j].value = board[k][j].value;
264                        board[k][j].value = null;
265                    }
266                }
267            }
268        }
269    }
270
271    public void moveRight() {
272        for (int i = 0; i < SIZE; i++) {
273            for (int j = 0; j < SIZE; j++) {
274                if (board[i][j] == null) continue;
275
276                for (int k = i; k >= 0; k--) {
277                    if (board[k][j] == null) continue;
278
279                    if (board[i][j].value == board[k][j].value) {
280                        board[i][j].value *= 2;
281                        board[k][j].value = null;
282                    } else if (board[i][j].value < board[k][j].value) {
283                        board[i][j].value = board[k][j].value;
284                        board[k][j].value = null;
285                    }
286                }
287            }
288        }
289    }
290
291    public void printBoard() {
292        for (int i = 0; i < SIZE; i++) {
293            for (int j = 0; j < SIZE; j++) {
294                System.out.print(board[i][j]);
295            }
296            System.out.println();
297        }
298    }
299
300    public void rotate() {
301        for (int i = 0; i < SIZE; i++) {
302            for (int j = 0; j < i; j++) {
303                swap(i, j);
304            }
305        }
306    }
307
308    public void swap(int i, int j) {
309        for (int k = 0; k < SIZE; k++) {
310            for (int l = 0; l < k; l++) {
311                swap(i, j, k, l);
312            }
313        }
314    }
315
316    public void swap(int i, int j, int k, int l) {
317        if (i == k && j == l) return;
318
319        if (i == k) {
320            if (j < l) {
321                swap(i, j, k, l - 1);
322                swap(i, j + 1, k, l);
323            } else {
324                swap(i, j, k, l + 1);
325                swap(i, j - 1, k, l);
326            }
327        } else {
328            if (j < l) {
329                swap(i, j, k, l - 1);
330                swap(i, j + 1, k, l);
331            } else {
332                swap(i, j, k, l + 1);
333                swap(i, j - 1, k, l);
334            }
335        }
336    }
337
338    public void moveUp() {
339        for (int i = 0; i < SIZE; i++) {
340            for (int j = 0; j < SIZE; j++) {
341                if (board[i][j] == null) continue;
342
343                for (int k = j; k < SIZE; k++) {
344                    if (board[i][k] == null) continue;
345
346                    if (board[i][j].value == board[i][k].value) {
347                        board[i][j].value *= 2;
348                        board[i][k].value = null;
349                    } else if (board[i][j].value < board[i][k].value) {
350                        board[i][j].value = board[i][k].value;
351                        board[i][k].value = null;
352                    }
353                }
354            }
355        }
356    }
357
358    public void moveDown() {
359        for (int i = 0; i < SIZE; i++) {
360            for (int j = 0; j < SIZE; j++) {
361                if (board[i][j] == null) continue;
362
363                for (int k = j; k >= 0; k--) {
364                    if (board[i][k] == null) continue;
365
366                    if (board[i][j].value == board[i][k].value) {
367                        board[i][j].value *= 2;
368                        board[i][k].value = null;
369                    } else if (board[i][j].value < board[i][k].value) {
370                        board[i][j].value = board[i][k].value;
371                        board[i][k].value = null;
372                    }
373                }
374            }
375        }
376    }
377
378    public void moveLeft() {
379        for (int i = 0; i < SIZE; i++) {
380            for (int j = 0; j < SIZE; j++) {
381                if (board[i][j] == null) continue;
382
383                for (int k = i; k < SIZE; k++) {
384                    if (board[k][j] == null) continue;
385
386                    if (board[i][j].value == board[k][j].value) {
387                        board[i][j].value *= 2;
388                        board[k][j].value = null;
389                    } else if (board[i][j].value < board[k][j].value) {
390                        board[i][j].value = board[k][j].value;
391                        board[k][j].value = null;
392                    }
393                }
394            }
395        }
396    }
397
398    public void moveRight() {
399        for (int i = 0; i < SIZE; i++) {
400            for (int j = 0; j < SIZE; j++) {
401                if (board[i][j] == null) continue;
402
403                for (int k = i; k >= 0; k--) {
404                    if (board[k][j] == null) continue;
405
406                    if (board[i][j].value == board[k][j].value) {
407                        board[i][j].value *= 2;
408                        board[k][j].value = null;
409                    } else if (board[i][j].value < board[k][j].value) {
410                        board[i][j].value = board[k][j].value;
411                        board[k][j].value = null;
412                    }
413                }
414            }
415        }
416    }
417
418    public void printBoard() {
419        for (int i = 0; i < SIZE; i++) {
420            for (int j = 0; j < SIZE; j++) {
421                System.out.print(board[i][j]);
422            }
423            System.out.println();
424        }
425    }
426
427    public void rotate() {
428        for (int i = 0; i < SIZE; i++) {
429            for (int j = 0; j < i; j++) {
430                swap(i, j);
431            }
432        }
433    }
434
435    public void swap(int i, int j) {
436        for (int k = 0; k < SIZE; k++) {
437            for (int l = 0; l < k; l++) {
438                swap(i, j, k, l);
439            }
440        }
441    }
442
443    public void swap(int i, int j, int k, int l) {
444        if (i == k && j == l) return;
445
446        if (i == k) {
447            if (j < l) {
448                swap(i, j, k, l - 1);
449                swap(i, j + 1, k, l);
450            } else {
451                swap(i, j, k, l + 1);
452                swap(i, j - 1, k, l);
453            }
454        } else {
455            if (j < l) {
456                swap(i, j, k, l - 1);
457                swap(i, j + 1, k, l);
458            } else {
459                swap(i, j, k, l + 1);
460                swap(i, j - 1, k, l);
461            }
462        }
463    }
464
465    public void moveUp() {
466        for (int i = 0; i < SIZE; i++) {
467            for (int j = 0; j < SIZE; j++) {
468                if (board[i][j] == null) continue;
469
470                for (int k = j; k < SIZE; k++) {
471                    if (board[i][k] == null) continue;
472
473                    if (board[i][j].value == board[i][k].value) {
474                        board[i][j].value *= 2;
475                        board[i][k].value = null;
476                    } else if (board[i][j].value < board[i][k].value) {
477                        board[i][j].value = board[i][k].value;
478                        board[i][k].value = null;
479                    }
480                }
481            }
482        }
483    }
484
485    public void moveDown() {
486        for (int i = 0; i < SIZE; i++) {
487            for (int j = 0; j < SIZE; j++) {
488                if (board[i][j] == null) continue;
489
490                for (int k = j; k >= 0; k--) {
491                    if (board[i][k] == null) continue;
492
493                    if (board[i][j].value == board[i][k].value) {
494                        board[i][j].value *= 2;
495                        board[i][k].value = null;
496                    } else if (board[i][j].value < board[i][k].value) {
497                        board[i][j].value = board[i][k].value;
498                        board[i][k].value = null;
499                    }
500                }
501            }
502        }
503    }
504
505    public void moveLeft() {
506        for (int i = 0; i < SIZE; i++) {
507            for (int j = 0; j < SIZE; j++) {
508                if (board[i][j] == null) continue;
509
510                for (int k = i; k < SIZE; k++) {
511                    if (board[k][j] == null) continue;
512
513                    if (board[i][j].value == board[k][j].value) {
514                        board[i][j].value *= 2;
515                        board[k][j].value = null;
516                    } else if (board[i][j].value < board[k][j].value) {
517                        board[i][j].value = board[k][j].value;
518                        board[k][j].value = null;
519                    }
520                }
521            }
522        }
523    }
524
525    public void moveRight() {
526        for (int i = 0; i < SIZE; i++) {
527            for (int j = 0; j < SIZE; j++) {
528                if (board[i][j] == null) continue;
529
530                for (int k = i; k >= 0; k--) {
531                    if (board[k][j] == null) continue;
532
533                    if (board[i][j].value == board[k][j].value) {
534                        board[i][j].value *= 2;
535                        board[k][j].value = null;
536                    } else if (board[i][j].value < board[k][j].value) {
537                        board[i][j].value = board[k][j].value;
538                        board[k][j].value = null;
539                    }
540                }
541            }
542        }
543    }
544
545    public void printBoard() {
546        for (int i = 0; i < SIZE; i++) {
547            for (int j = 0; j < SIZE; j++) {
548                System.out.print(board[i][j]);
549            }
550            System.out.println();
551        }
552    }
553
554    public void rotate() {
555        for (int i = 0; i < SIZE; i++) {
556            for (int j = 0; j < i; j++) {
557                swap(i, j);
558            }
559        }
560    }
561
562    public void swap(int i, int j) {
563        for (int k = 0; k < SIZE; k++) {
564            for (int l = 0; l < k; l++) {
565                swap(i, j, k, l);
566            }
567        }
568    }
569
570    public void swap(int i, int j, int k, int l) {
571        if (i == k && j == l) return;
572
573        if (i == k) {
574            if (j < l) {
575                swap(i, j, k, l - 1);
576                swap(i, j + 1, k, l);
577            } else {
578                swap(i, j, k, l + 1);
579                swap(i, j - 1, k, l);
580            }
581        } else {
582            if (j < l) {
583                swap(i, j, k, l - 1);
584                swap(i, j + 1, k, l);
585            } else {
586                swap(i, j, k, l + 1);
587                swap(i, j - 1, k, l);
588            }
589        }
590    }
591
592    public void moveUp() {
593        for (int i = 0; i < SIZE; i++) {
594            for (int j = 0; j < SIZE; j++) {
595                if (board[i][j] == null) continue;
596
597                for (int k = j; k < SIZE; k++) {
598                    if (board[i][k] == null) continue;
599
600                    if (board[i][j].value == board[i][k].value) {
601                        board[i][j].value *= 2;
602                        board[i][k].value = null;
603                    } else if (board[i][j].value < board[i][k].value) {
604                        board[i][j].value = board[i][k].value;
605                        board[i][k].value = null;
606                    }
607                }
608            }
609        }
610    }
611
612    public void moveDown() {
613        for (int i = 0; i < SIZE; i++) {
614            for (int j = 0; j < SIZE; j++) {
615                if (board[i][j] == null) continue;
616
617                for (int k = j; k >= 0; k--) {
618                    if (board[i][k] == null) continue;
619
620                    if (board[i][j].value == board[i][k].value) {
621                        board[i][j].value *= 2;
622                        board[i][k].value = null;
623                    } else if (board[i][j].value < board[i][k].value) {
624                        board[i][j].value = board[i][k].value;
625                        board[i][k].value = null;
626                    }
627                }
628            }
629        }
630    }
631
632    public void moveLeft() {
633        for (int i = 0; i < SIZE; i++) {
634            for (int j = 0; j < SIZE; j++) {
635                if (board[i][j] == null) continue;
636
637                for (int k = i; k < SIZE; k++) {
638                    if (board[k][j] == null) continue;
639
640                    if (board[i][j].value == board[k][j].value) {
641                        board[i][j].value *= 2;
642                        board[k][j].value = null;
643                    } else if (board[i][j].value < board[k][j].value) {
644                        board[i][j].value = board[k][j].value;
645                        board[k][j].value = null;
646                    }
647                }
648            }
649        }
650    }
651
652    public void moveRight() {
653        for (int i = 0; i < SIZE; i++) {
654            for (int j = 0; j < SIZE; j++) {
655                if (board[i][j] == null) continue;
656
657                for (int k = i; k >= 0; k--) {
658                    if (board[k][j] == null) continue;
659
660                    if (board[i][j].value == board[k][j].value) {
661                        board[i][j].value *= 2;
662                        board[k][j].value = null;
663                    } else if (board[i][j].value < board[k][j].value) {
664                        board[i][j].value = board[k][j].value;
665                        board[k][j].value = null;
666                    }
667                }
668            }
669        }
670    }
671
672    public void printBoard() {
673        for (int i = 0; i < SIZE; i++) {
674            for (int j = 0; j < SIZE; j++) {
675                System.out.print(board[i][j]);
676            }
677            System.out.println();
678        }
679    }
680
681    public void rotate() {
682        for (int i = 0; i < SIZE; i++) {
683            for (int j = 0; j < i; j++) {
684                swap(i, j);
685            }
686        }
687    }
688
689    public void swap(int i, int j) {
690        for (int k = 0; k < SIZE; k++) {
691            for (int l = 0; l < k; l++) {
692                swap(i, j, k, l);
693            }
694        }
695    }
696
697    public void swap(int i, int j, int k, int l) {
698        if (i == k && j == l) return;
699
700        if (i == k) {
701            if (j < l) {
702                swap(i, j, k, l - 1);
703                swap(i, j + 1, k, l);
704            } else {
705                swap(i, j, k, l + 1);
706                swap(i, j - 1, k, l);
707            }
708        } else {
709            if (j < l) {
710                swap(i, j, k, l - 1);
711                swap(i, j + 1, k, l);
712            } else {
713                swap(i, j, k, l + 1);
714                swap(i, j - 1, k, l);
715            }
716        }
717    }
718
719    public void moveUp() {
720        for (int i = 0; i < SIZE; i++) {
721            for (int j = 0; j < SIZE; j++) {
722                if (board[i][j] == null) continue;
723
724                for (int k = j; k < SIZE; k++) {
725                    if (board[i][k] == null) continue;
726
727                    if (board[i][j].value == board[i][k].value) {
728                        board[i][j].value *= 2;
729                        board[i][k].value = null;
730                    } else if (board[i][j].value < board[i][k].value) {
731                        board[i][j].value = board[i][k].value;
732                        board[i][k].value = null;
733                    }
734                }
735            }
736        }
737    }
738
739    public void moveDown() {
740        for (int i = 0; i < SIZE; i++) {
741            for (int j = 0; j < SIZE; j++) {
742                if (board[i][j] == null) continue;
743
744                for (int k = j; k >= 0; k--) {
745                    if (board[i][k] == null) continue;
746
747                    if (board[i][j].value == board[i][k].value) {
748                        board[i][j].value *= 2;
749                        board[i][k].value = null;
750                    } else if (board[i][j].value < board[i][k].value) {
751                        board[i][j].value = board[i][k].value;
752                        board[i][k].value = null;
753                    }
754                }
755            }
756        }
757    }
758
759    public void moveLeft() {
760        for (int i = 0; i < SIZE; i++) {
761            for (int j = 0; j < SIZE; j++) {
762                if (board[i][j] == null) continue;
763
764                for (int k = i; k < SIZE; k++) {
765                    if (board[k][j] == null) continue;
766
767                    if (board[i][j].value == board[k][j].value) {
768                        board[i][j].value *= 2;
769                        board[k][j].value = null;
770                    } else if (board[i][j].value < board[k][j].value) {
771                        board[i][j].value = board[k][j].value;
772                        board[k][j].value = null;
773                    }
774                }
775            }
776        }
777    }
778
779    public void moveRight() {
780        for (int i = 0; i < SIZE; i++) {
781            for (int j = 0; j < SIZE; j++) {
782                if (board[i][j] == null) continue;
783
784                for (int k = i; k >= 0; k--) {
785                    if (board[k][j] == null) continue;
786
787                    if (board[i][j].value == board[k][j].value) {
788                        board[i][j].value *= 2;
789                        board[k][j].value = null;
790                    } else if (board[i][j].value < board[k][j].value) {
791                        board[i][j].value = board[k][j].value;
792                        board[k][j].value = null;
793                    }
794                }
795            }
796        }
797    }
798
799    public void printBoard() {
800        for (int i = 0; i < SIZE; i++) {
801            for (int j = 0; j < SIZE; j++) {
802                System.out.print(board[i][j]);
803            }
804            System.out.println();
805        }
806    }
807
808    public void rotate() {
809        for (int i = 0; i < SIZE; i++) {
810            for (int j = 0; j < i; j++) {
811                swap(i, j);
812            }
813        }
814    }
815
816    public void swap(int i, int j) {
817        for (int k = 0; k < SIZE; k++) {
818            for (int l = 0; l < k; l++) {
819                swap(i, j, k, l);
820            }
821        }
822    }
823
824    public void swap(int i, int j, int k, int l) {
825        if (i == k && j == l) return;
826
827        if (i == k) {
828            if (j < l) {
829                swap(i, j, k, l - 1);
830                swap(i, j + 1, k, l);
831            } else {
832                swap(i, j, k, l + 1);
833                swap(i, j - 1, k, l);
834            }
835        } else {
836            if (j < l) {
837                swap(i, j, k, l - 1);
838                swap(i, j + 1, k, l);
839            } else {
840                swap(i, j, k, l + 1);
841                swap(i, j - 1, k, l);
842            }
843        }
844    }
845
846    public void moveUp() {
847        for (int i = 0; i < SIZE; i++) {
848            for (int j = 0; j < SIZE; j++) {
849                if (board[i][j] == null) continue;
850
851                for (int k = j; k < SIZE; k++) {
852                    if (board[i][k] == null) continue;
853
854                    if (board[i][j].value == board[i][k].value) {
855                        board[i][j].value *= 2;
856                        board[i][k].value = null;
857                    } else if (board[i][j].value < board[i][k].value) {
858                        board[i][j].value = board[i][k].value;
859                        board[i][k].value = null;
860                    }
861                }
862            }
863        }
864    }
865
866    public void moveDown() {
867        for (int i = 0; i < SIZE; i++) {
868            for (int j = 0; j < SIZE; j++) {
869                if (board[i][j] == null) continue;
870
871                for (int k = j; k >= 0; k--) {
872                    if (board[i][k] == null) continue;
873
874                    if (board[i][j].value == board[i][k].value) {
875                        board[i][j].value *= 2;
876                        board[i][k].value = null;
877                    } else if (board[i][j].value < board[i][k].value) {
878                        board[i][j].value = board[i][k].value;
879                        board[i][k].value = null;
880                    }
881                }
882            }
883        }
884    }
885
886    public void moveLeft() {
887        for (int i = 0; i < SIZE; i++) {
888            for (int j = 0; j < SIZE; j++) {
889                if (board[i][j] == null) continue;
890
891                for (int k = i; k < SIZE; k++) {
892                    if (board[k][j] == null) continue;
893
894                    if (board[i][j].value == board[k][j].value) {
895                        board[i][j].value *= 2;
896                        board[k][j].value = null;
897                    } else if (board[i][j].value < board[k][j].value) {
898                        board[i][j].value = board[k][j].value;
899                        board[k][j].value = null;
900                    }
901                }
902            }
903        }
904    }
905
906    public void moveRight() {
907        for (int i = 0; i < SIZE; i++) {
908            for (int j = 0; j < SIZE; j++) {
909                if (board[i][j] == null) continue;
910
911                for (int k = i; k >= 0; k--) {
912                    if (board[k][j] == null) continue;
913
914                    if (board[i][j].value == board[k][j].value) {
915                        board[i][j].value *= 2;
916                        board[k][j].value = null;
917                    } else if (board[i][j].value < board[k][j].value) {
918                        board[i][j].value = board[k][j].value;
919                        board[k][j].value = null;
920                    }
921                }
922            }
923        }
924    }
925
926    public void printBoard() {
927        for (int i = 0; i < SIZE; i++) {
928            for (int j = 0; j < SIZE; j++) {
929                System.out.print(board[i][j]);
930            }
931            System.out.println();
932        }
933    }
934
935    public void rotate() {
936        for (int i = 0; i < SIZE; i++) {
937            for (int j = 0; j < i; j++) {
938                swap(i, j);
939            }
940        }
941    }
942
943    public void swap(int i, int j) {
944        for (int k = 0; k < SIZE; k++) {
945            for (int l = 0; l < k; l++) {
946                swap(i, j, k, l);
947            }
948        }
949    }
950
951    public void swap(int i, int j, int k, int l) {
952        if (i == k && j == l) return;
953
954        if (i == k) {
955            if (j < l) {
956                swap(i, j, k, l - 1);
957                swap(i, j + 1, k, l);
958            } else {
959                swap(i, j, k, l + 1);
960                swap(i, j - 1, k, l);
961            }
962        } else {
963            if (j < l) {
964                swap(i, j, k, l - 1);
965                swap(i, j + 1, k, l);
966            } else {
967                swap(i, j, k, l + 1);
968                swap(i, j - 1, k, l);
969            }
970        }
971    }
972
973    public void moveUp() {
974        for (int i = 0; i < SIZE; i++) {
975            for (int j = 0; j < SIZE; j++) {
976                if (board[i][j] == null) continue;
977
978                for (int k = j; k < SIZE; k++) {
979                    if (board[i][k] == null) continue;
980
981                    if (board[i][j].value == board[i][k].value) {
982                        board[i][j].value *= 2;
983                        board[i][k].value = null;
984                    } else if (board[i][j].value < board[i][k].value) {
985                        board[i][j].value = board[i][k].value;
986                        board[i][k].value = null;
987                    }
988                }
989            }
990        }
991    }
992
993    public void moveDown() {
994        for (int i = 0; i < SIZE; i++) {
995            for (int j = 0; j < SIZE; j++) {
996                if (board[i][j] == null) continue;
997
998                for (int k = j; k >= 0; k--) {
999                    if (board[i][k] == null) continue;
1000
1001                    if (board[i][j].value == board[i][k].value) {
1002                        board[i][j].value *= 2;
1003                        board[i][k].value = null;
1004                    } else if (board[i][j].value < board[i][k].value) {
1005                        board[i][j].value = board[i][k].value;
1006                        board[i][k].value = null;
1007                    }
1008                }
1009            }
1010        }
1011    }
1012
1013    public void moveLeft() {
1014        for (int i = 0; i < SIZE; i++) {
1015            for (int j = 0; j < SIZE; j++) {
1016                if (board[i][j] == null) continue;
1017
1018                for (int k = i; k < SIZE; k++) {
1019                    if (board[k][j] == null) continue;
1020
1021                    if (board[i][j].value == board[k][j].value) {
1022                        board[i][j].value *= 2;
1023                        board[k][j].value = null;
1024                    } else if (board[i][j].value < board[k][j].value) {
1025                        board[i][j].value = board[k][j].value;
1026                        board[k][j].value = null;
1027                    }
1028                }
1029            }
1030        }
1031    }
1032
1033    public void moveRight() {
1034        for (int i = 0; i < SIZE; i++) {
1035            for (int j = 0; j < SIZE; j++) {
1036                if (board[i][j] == null) continue;
1037
1038                for (int k = i; k >= 0; k--) {
1039                    if (board[k][j] == null) continue;
1040
1041                    if (board[i][j].value == board[k][j].value) {
1042                        board[i][j].value *= 2;
1043                        board[k][j].value = null;
1044                    } else if (board[i][j].value < board[k][j].value) {
1045                        board[i][j].value = board[k][j].value;
1046                        board[k][j].value = null;
1047                    }
1048                }
1049            }
1050        }
1051    }
1052
1053    public void printBoard() {
1054        for (int i = 0; i < SIZE; i++) {
1055            for (int j = 0; j < SIZE; j++) {
1056                System.out.print(board[i][j]);
1057            }
1058            System.out.println();
1059        }
1060    }
1061
1062    public void rotate() {
1063        for (int i = 0; i < SIZE; i++) {
1064            for (int j = 0; j < i; j++) {
10
```

En la linea 32, mai podrem crear un bloc amb value > 0 i color NONE.

Com aquest cas en tenim tots els que fan que el percentatge no sigui 100% de les classes. També hem pensat en no fer testing de les classes que només fan funcions de JavaUtil o altres. Un exemple seria la classe ScannerGame, que només conté un objecte tipus Scanner i les seves funcions només són JavaUtil per a poder fer el mock mitjançant IOC. Per tant, els asserts dels invariant, sempre son true, per tant sempre baixarà el percentatge.

Izan Caballer Jimenez - 1710282
Adrián Valverde Ambrosio - 1707952

```
package eng.uab.tqs.game2048.model;

import java.util.Scanner;

public class ScannerGame { 5 usages  Adrián Valverde Ambrosio
    Scanner scanner; 4 usages

    public ScannerGame() { this.scanner = new Scanner(System.in); }

    public String scanName() { no usages 1 override  Adrián Valverde Ambrosio
        String input = scanner.nextLine().trim().toLowerCase();
        return input;
    }

    public Character scanOptions() { 1 usage 1 override  Adrián Valverde Ambrosio
        String input = scanner.nextLine().trim().toLowerCase();
        return input.charAt(0);
    }

    public String nextLine() { return scanner.nextLine(); }
}
```

Block	Block	75,1 %	187	62	249
	Block()	64,7 %	22	12	34
	Block(int, Color)	84,9 %	45	8	53
	getColor()	100,0 %	3	0	3
	getValue()	100,0 %	3	0	3
	invariant()	89,5 %	17	2	19
	log2(int)	100,0 %	12	0	12
	mix()	71,4 %	60	24	84
	resetBlock()	60,0 %	24	16	40
Board.java	Board	88,3 %	1.450	192	1.642
	Board	88,3 %	1.450	192	1.642
	Board()	83,7 %	123	24	147
	Board(Board)	84,6 %	154	28	182
	canMove(int, int, char)	94,4 %	134	8	142
	drawBoard()	100,0 %	27	0	27
	getBoard()	100,0 %	3	0	3
	getGenerator()	100,0 %	3	0	3
	getRestart()	100,0 %	3	0	3
	getScore()	100,0 %	3	0	3
	invariant()	83,9 %	104	20	124
	isGameOver()	86,0 %	74	12	86
	isGameWon()	78,2 %	43	12	55
	join(int, int, char)	92,7 %	255	20	275
	moveBlock(int, int, char)	85,5 %	142	24	166
	moveBoard(char)	96,0 %	291	12	303
	random()	73,9 %	68	24	92
	randomInitialize()	63,6 %	14	8	22
	setGenerator(Generator)	100,0 %	4	0	4
	setRestart(boolean)	100,0 %	4	0	4

		Class, %	Method, %	Line, %	Branch, %
Game.java	Game	71,4 %	432	173	605
	Game()	71,4 %	432	173	605
	getExit()	100,0 %	24	0	24
	getGameMatch()	63,6 %	3	0	3
	getScores()	58,6 %	7	4	11
	invariant()	68,0 %	17	12	29
	menu()	82,6 %	57	12	69
	playGame()	60,0 %	24	16	40
	playGameFX()	56,8 %	21	16	37
	saveScore(String)	81,9 %	77	17	94
	setManager(FileManager)	60,0 %	12	8	20
	setScanner(ScannerGame)	60,0 %	12	8	20
	showScores()	80,3 %	114	28	142
	showScoresFX()	74,2 %	46	16	62
GameMatch.java	GameMatch	70,2 %	179	76	255
	getBoard()	70,2 %	179	76	255
	invariant()	63,6 %	7	4	11
	setBoard(Board)	66,7 %	8	4	12
	setScanner(ScannerMovement)	60,0 %	12	8	20
	startGame()	57,1 %	16	12	28
	startGameFX()	74,6 %	88	30	118
	65,4 %	34	18	52	
Generator.java	Generator	100,0 %	50	0	50
	genRandom()	100,0 %	50	0	50
	randomIncialize(Block[][])	100,0 %	8	0	8
			39	0	39
InfoGame.java	InfoGame	93,8 %	165	11	176
	Color	0,0 %	0	3	3
	getValue()	95,4 %	165	8	173
		100,0 %	3	0	3

Resum final:

Element ^	Class, %	Method, %	Line, %	Branch, %
eng.uab.tqs.game2048.model	68% (13/19)	82% (87/106)	92% (756/871)	77% (549/712)
mock	71% (5/7)	77% (31/40)	88% (287/314)	89% (162/181)
Block	100% (1/1)	100% (9/9)	100% (36/36)	62% (34/54)
Board	100% (1/1)	100% (19/19)	99% (263/263)	80% (256/318)
FileManager	0% (0/1)	0% (0/4)	0% (0/4)	100% (0/0)
Game	100% (1/1)	93% (14/15)	95% (97/102)	60% (60/99)
GameMatch	100% (1/1)	100% (6/6)	95% (42/44)	63% (29/46)
Generator	100% (1/1)	100% (2/2)	100% (9/9)	75% (3/4)
InfoGame	50% (1/2)	100% (4/4)	100% (18/18)	50% (5/10)
main	0% (0/1)	0% (0/1)	0% (0/2)	100% (0/0)
Movement	0% (0/1)	100% (0/0)	100% (0/0)	100% (0/0)
ScannerGame	100% (1/1)	25% (1/4)	28% (2/7)	100% (0/0)
ScannerMovement	100% (1/1)	50% (1/2)	50% (2/4)	100% (0/0)

- Decision coverage, condition coverage i path coverage:

Decision i condition coverage: es basa en el fet que el codi pren totes les decisions possibles, és a dir, totes les condicions passen per false i per true. En el nostre cas veiem codi on s'executa cada cas del switch, i a més a més, en la funció de canMove, trobem un if després del switch, això crea molts camins.

Path coverage: el path coverage en ajuda a que tots els camins que es poden executar en el codi, s'executin. Amb ajuda d'un diagrama de flux pots veure tots els paths que pot tenir el teu mètode .

De la classe board:

- public int[] moveBlock(int i, int j, char c)

```
148  @  public int[] moveBlock(int i, int j, char c) { 31 usages 1 override & Adrián Valverde Ambrosio
149    assert invariant();
150    //preconditions
151    assert(i >= 0);
152    assert(i < SIZE);
153    assert(j >= 0 && j < SIZE);
154    assert(board[i][j].getValue() > 0);
155    assert(c == 'w' || c == 'a' || c == 's' || c == 'd');
156
157    int i2 = i;
158    int j2 = j;
159    switch (c) {
160      case 'a':
161        j2 = j-1;
162        break;
163
164      case 'w':
165        i2 = i-1;
166        break;
167
168      case 'd':
169        j2 = j+1;
170        break;
171    }
```

Izan Caballer Jimenez - 1710282
Adrián Valverde Ambrosio - 1707952

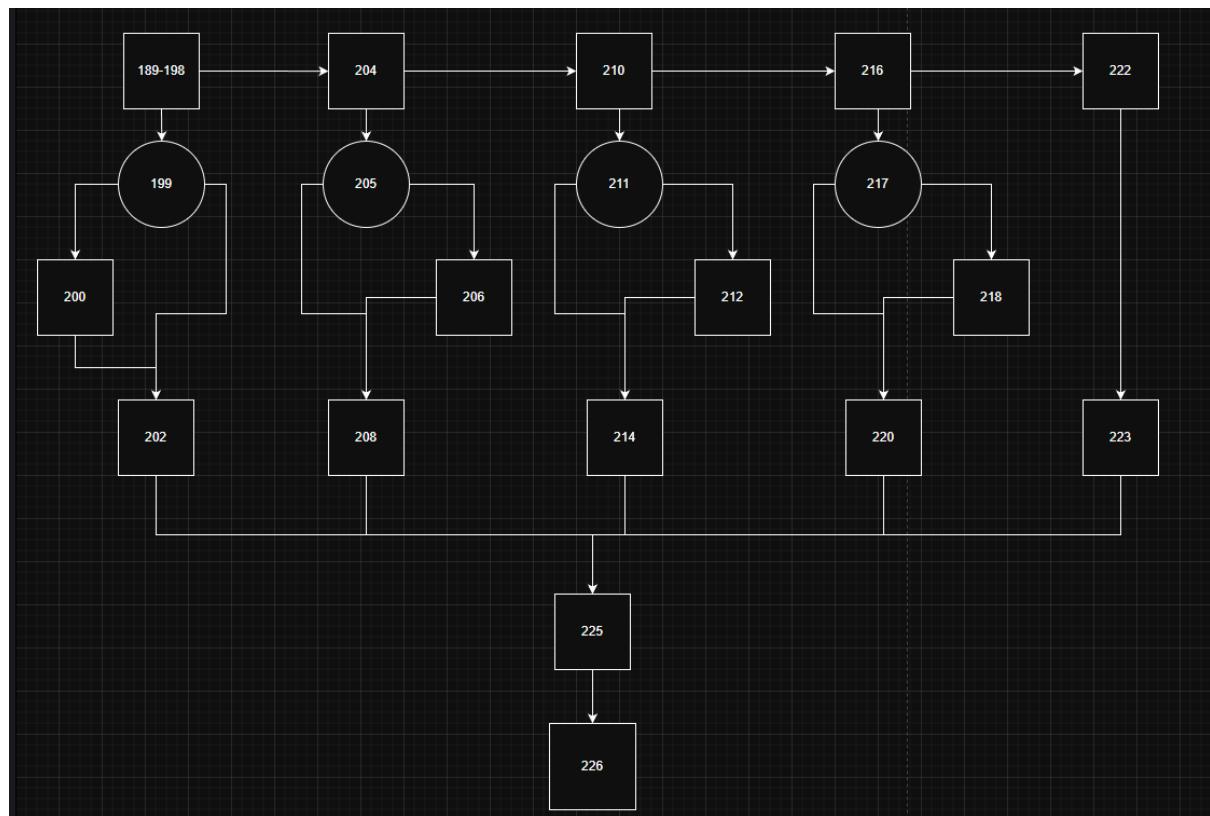
```
172
173     case 's':
174         i2 = i+1;
175         break;
176
177     default:
178         break;
179     }
180     board[i2][j2] = new Block(board[i][j].getValue(), board[i][j].getColor());
181     board[i][j].resetBlock();
182     int[] coords = {i2,j2};
183     assert invariant();
184     assert(board[i][j].getValue() == 0);
185     assert(board[i2][j2].getValue() != 0);
186     return coords;
187 }
```

- public boolean canMove(int i, int j, char c)

```
188     public boolean canMove(int i, int j, char c) { 41 usages 1 override & Adrián Valverde Ambrosio +1
189         assert invariant();
190         //preconditions
191         assert(i >= 0 && i < SIZE);
192         assert(j >= 0 && j < SIZE);
193         assert(board[i][j].getValue() > 0);
194         assert(c == 'w' || c == 'a' || c == 's' || c == 'd');
195
196         boolean result = false;
197         switch (c) {
198             case 'a':
199                 if (j > 0) {
200                     result = (0 == board[i][j - 1].getValue());
201                 }
202                 break;
203
204             case 'w':
205                 if (i > 0) {
206                     result = (0 == board[i - 1][j].getValue());
207                 }
208                 break;
209
210             case 'd':
211                 if (j < SIZE - 1) {
212                     result = (0 == board[i][j + 1].getValue());
213                 }
214                 break;
215 }
```

```
216     case 's':
217         if (i < SIZE - 1) {
218             result = (0 == board[i + 1][j].getValue());
219         }
220         break;
221
222     default:
223         break;
224     }
225     assert invariant();
226     return result;
227 }
```

- path coverage:



- `public boolean join(int i, int j, char c)`

```
229  @. public boolean join(int i, int j, char c) { 20 usages 1 override & Adrián Valverde Ambrosio +2
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
```

```
    assert invariant();
    //preconditions
    assert(i >= 0 && i < SIZE);
    assert(j >= 0 && j < SIZE);
    assert(board[i][j].getValue() > 0);
    assert(c == 'w' || c == 'a' || c == 's' || c == 'd');

    boolean result = false;
    int scoreAdd = 0;
    switch (c) {
        case 'w':
            if (i > 0) {
                if (board[i][j].getValue() == board[i-1][j].getValue()) {
                    board[i-1][j].mix();
                    result = true;
                    scoreAdd += board[i-1][j].getValue();
                }
            }
            break;

        case 'a':
            if (j > 0) {
                if (board[i][j].getValue() == board[i][j-1].getValue()) {
                    board[i][j-1].mix();
                    result = true;
                    scoreAdd += board[i][j-1].getValue();
                }
            }
            break;

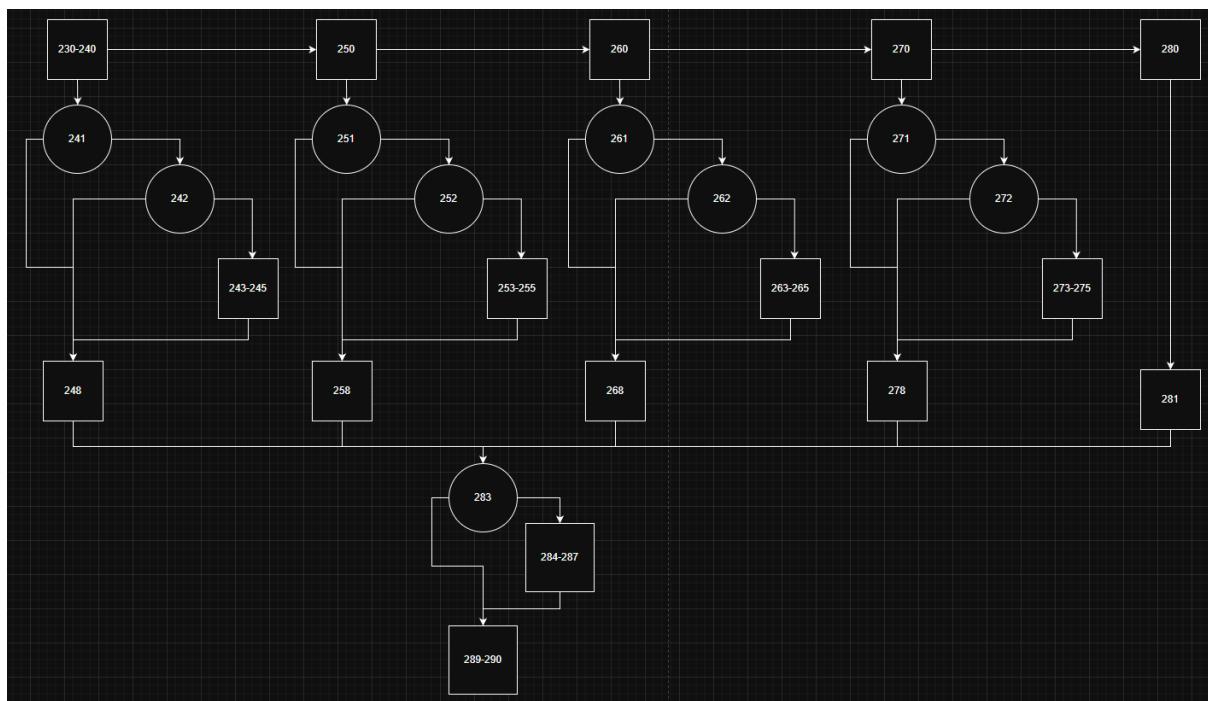
        case 's':
            if (i < SIZE - 1) {
                if (board[i][j].getValue() == board[i+1][j].getValue()) {
                    board[i+1][j].mix();
                    result = true;
                    scoreAdd += board[i+1][j].getValue();
                }
            }
            break;

        case 'd':
            if (j < SIZE - 1) {
                if (board[i][j].getValue() == board[i][j+1].getValue()) {
                    board[i][j+1].mix();
                    result = true;
                    scoreAdd += board[i][j+1].getValue();
                }
            }
            break;

        default:
            break;
    }
```

```
283     if (result) {  
284         board[i][j].resetBlock();  
285         //postcondition  
286         assert score < scoreAdd+score;  
287         score += scoreAdd;  
288     }  
289     assert invariant();  
290     return result;  
291 }
```

- path coverage:



Loop testing:

- simple loop → showScores() de la classe Game

Izan Caballer Jimenez - 1710282
Adrián Valverde Ambrosio - 1707952

```
93     @Test  ♫ Adrián Valverde
94     void loopSimpleTest()
95     {
96         //avoid loop
97         fm.setConfig("EMPTY_FILE");
98         assertDoesNotThrow(() -> game.showScores());
99         List<String> s = game.getScores();
100        assertNotNull(s);
101        assertEquals(expected: 0, s.size());
102
103        //one iteration
104        fm.setConfig("ONE_SCORE");
105        game.showScores();
106        s = game.getScores();
107        assertEquals(expected: 1, s.size());
108        assertEquals(expected: "TestUser:400", s.get(0));
109
110        //two iterations
111        fm.setConfig("two");
112        game.showScores();
113        List<String> s2 = game.getScores();
114        assertEquals(expected: 2, s2.size());
115
116        //Some iterations
117        fm.setConfig("some");
118        game.showScores();
119        List<String> s3 = game.getScores();
120        assertEquals(expected: 4, s3.size());
121
122        //ten iterations
123        fm.setConfig("ten");
124        game.showScores();
125        List<String> s4 = game.getScores();
126        assertEquals(expected: 10, s4.size());
127
128    }
129
84
85
86
87
88
for (String s : scores) {
    String[] parts = s.split(regex: ":");
    System.out.printf("%d\t%s\t%s\n", rank, parts[0].trim(), parts[1].trim());
    rank++;
}
```

- Loops aniuats →isGameOver() de la classe Board

```
414     @Test  ➔ Adrián Valverde
415     void loopNestedGameOver()
416     {
417         //interior simple loop
418         Board board = new Board();
419         assertFalse(board.isGameOver());
420         Block[][] b = board.getBoard();
421
422         //some iterations
423         for(int j = 0; j < SIZE; j++)
424             {b[0][j] = new Block( value: 2, InfoGame.Color.GREEN);}
425         assertFalse(board.isGameOver());
426
427         //half iterations
428         for (int i = 0; i < 2; i++) {
429             for (int j = 0; j < 4; j++) {
430                 b[i][j] = new Block( value: 2, InfoGame.Color.GREEN);
431             }
432         }
433
434         assertFalse(board.isGameOver());
435     |
436
437         //full iterations
438         for (int i = 0; i < SIZE; i++ )
439         {
440             for (int j = 0; j < SIZE; j++ )
441                 {b[0][j] = new Block( value: 2, InfoGame.Color.GREEN);}
442         }
443
444         boolean result = board.isGameOver();
445         //we just want to run through all the board
446         assertTrue( condition: result || !result);
447
448     }
449 }
```

```
445  @Override public boolean isGameOver() { 15 usages 1 override Adrián Valverde
446    assert invariant();
447    boolean movementDone = false;
448    int i = 0;
449    for (i = 0; i < SIZE; i++) {
450      for (int j = 0; j < SIZE; j++) {
451        if (board[i][j].getValue() == 0) {
452          assert invariant();
453          return false;
454        }
455      }
456    }
}
```

Data driven test:

La tècnica de Data-Driven Testing consisteix a separar les dades de prova de la lògica del test, de manera que un mateix cas de prova s'executa múltiples vegades utilitzant diferents conjunts de dades externes. En el nostre cas, hem creat 2 arxius CSV on hem deixat les dades a llegir i dins dels mètodes de test, només cal llegir l'arxiu

Hem provat aquesta tècnica pels mètodes de moveBoard() i showScored()

```
453
454
455  @Test new *
456  void moveBoard_dataDriven() throws Exception {
457    List<String> lines = Files.readAllLines(
458      Path.of(first: "src/test/java/eng/uab/tqs/game2048/model/datadriven/board.csv"));
459
460    for (String line : lines) {
461      char dir = line.trim().charAt(0);
462
463      Board board = new Board();
464      board.setGenerator(gen);
465      board.randomInitialize();
466      assertDoesNotThrow(() -> board.moveBoard(dir));
467    }
468  }
469
470 }
```

```
192 @Test new *
193 void saveScore_dataDriven() throws Exception {
194
195     List<String> names = Files.readAllLines(
196         Path.of(first:"src/test/java/eng/uab/tqs/game2048/model/datadriven/score.csv"));
197
198     for (String name : names) {
199
200         Game game = new Game();
201         FileManagerMock fm = new FileManagerMock();
202         game.setManager(fm);
203
204         fm.setConfig("EMPTY_FILE");
205
206         BoardMock board = new BoardMock();
207         board.setScore(100);
208
209         game.getGameMatch().setBoard(board);
210
211         game.saveScore(name);
212
213         assertNotNull(game.getScores());
214     }
215 }
216 }
```

CI test:

S'ha configurat un flux de CI mitjançant GitHub Actions que executa els tests automàticament a cada pull request. El merge es bloqueja si els tests no passen, o si no segueixen l'estruccura que té el checkstyle. A la imatge, podem veure un merge que ha fallat, ja que no complia cap de la premisses establerta de no existir errors de tests.

Izan Caballer Jimenez - 1710282
Adrián Valverde Ambrosio - 1707952

Ci cd test error #3

[Open](#) AdrianValverdeA wants to merge 3 commits into `main` from `CI-CD-test-error`

Conversation 0 Commits 3 Checks 0 Files changed 2

AdrianValverdeA commented now

No description provided.

AdrianValverdeA added 3 commits 10 minutes ago

- cicd test 8d85829 ✓ aaad0e8
- ciCd test try ✘ 0eb0414
- ciCd test try assertFalse- error ✘ 0eb0414

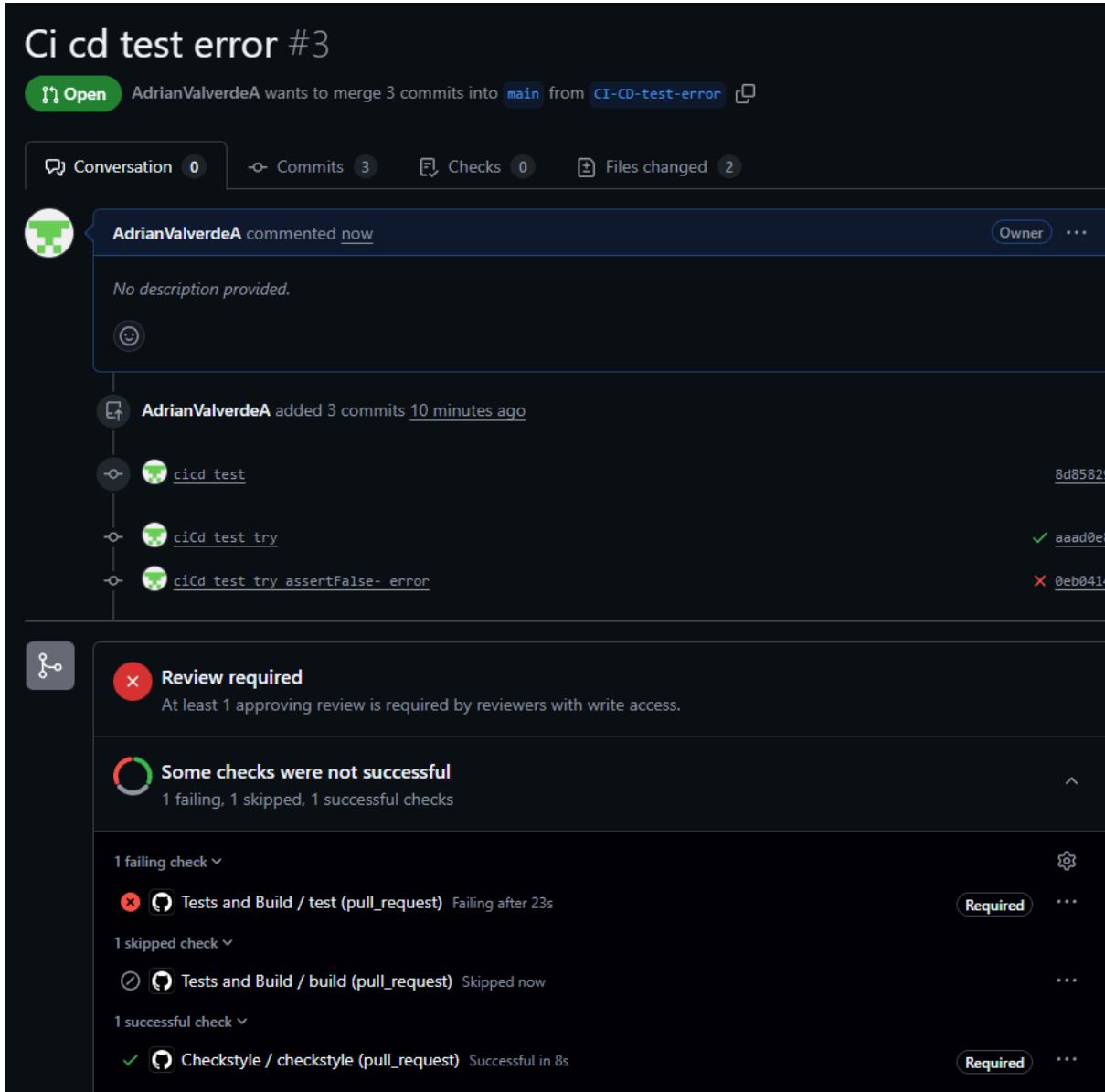
Review required
At least 1 approving review is required by reviewers with write access.

Some checks were not successful
1 failing, 1 skipped, 1 successful checks

1 failing check ▾
✗ Tests and Build / test (pull_request) Failing after 23s Required ⋮

1 skipped check ▾
⌚ Tests and Build / build (pull_request) Skipped now ⋮

1 successful check ▾
✓ Checkstyle / checkstyle (pull_request) Successful in 8s Required ⋮



Izan Caballer Jimenez - 1710282
Adrián Valverde Ambrosio - 1707952

Falta aprovació de persona amb permisos, pero passa correctament:

Ci cd test #2

Open AdrianValverdeA wants to merge 2 commits into `main` from `CI-CD-test` ↗

Conversation 0 · Commits 2 · Checks 3 · Files changed 2 · Owner · ...

AdrianValverdeA commented 1 minute ago · Owner · ...
No description provided.
😊

AdrianValverdeA added 2 commits 4 minutes ago
-o- cicd test · 8d85829
-o- ciCd test try · ✓ aaad0e8

Review required
At least 1 approving review is required by reviewers with write access.

All checks have passed
3 successful checks

✓ Checkstyle / checkstyle (pull_request) Successful in 10s · Required · ...
✓ Tests and Build / build (pull_request) Successful in 23s · ...
✓ Tests and Build / test (pull_request) Successful in 24s · Required · ...

Merging is blocked
At least 1 approving review is required by reviewers with write access.

Merge without waiting for requirements to be met (bypass rules)

