Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

**Horari de pràctiques: Dijous 12:30-14:30**

# TEST I QUALITAT DE SOFTWARE. PRÀCTICA 1

## INTRODUCCIÓ

## BUSCAMINES

El joc que hem escollit per a desenvolupar i portar a terme a aquesta pràctica és el **Buscamines**. Aquest joc és un clàssic de lògica on l'objectiu principal és revelar totes les caselles sense bomba d'un tauler de joc sense activar cap bomba. Es tracta d'un joc molt conegut que combina deducció, estratègia i atzar.

En la seva forma original, el tauler està format per una graella rectangular dividida en caselles, algunes de les quals contenen bombes. Quan el jugador revela una casella sense bomba, aquesta pot mostrar un número que indica quantes bombes hi ha en les caselles adjacents. El joc acaba quan es revelen totes les caselles sense bombes o quan el jugador revela una casella amb bomba, perdent la partida.

## LES NORMES DEL NOSTRE LLOC

En el nostre **Buscamines**, hem desenvolupat una versió interactiva amb diverses funcionalitats i nivells de dificultat. El nostre joc inclou:

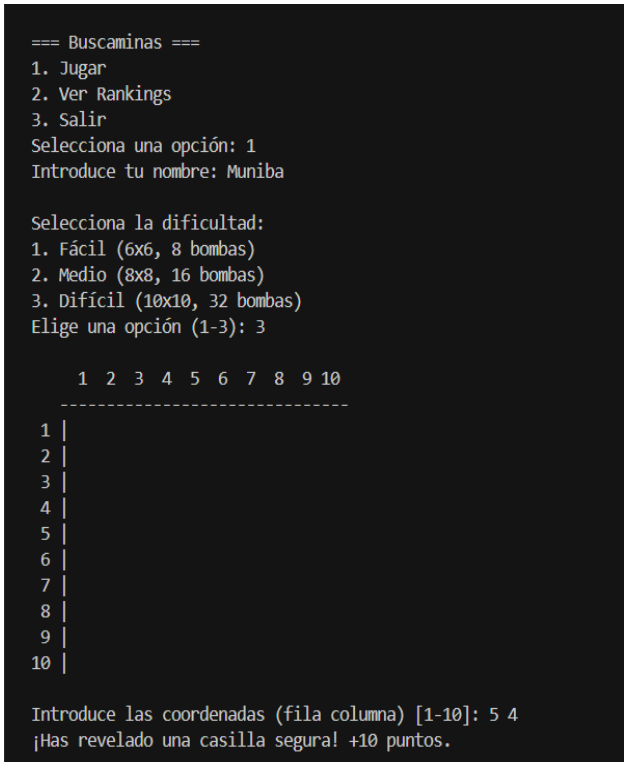
1. **Un menú principal amb tres opcions:**
    - **Jugar:** Permet iniciar una partida seleccionant el nivell de dificultat.
    - **Veure Rankings:** Mostra les puntuacions més altes de jugadors anteriors.
    - **Sortir:** Finalitza el joc.

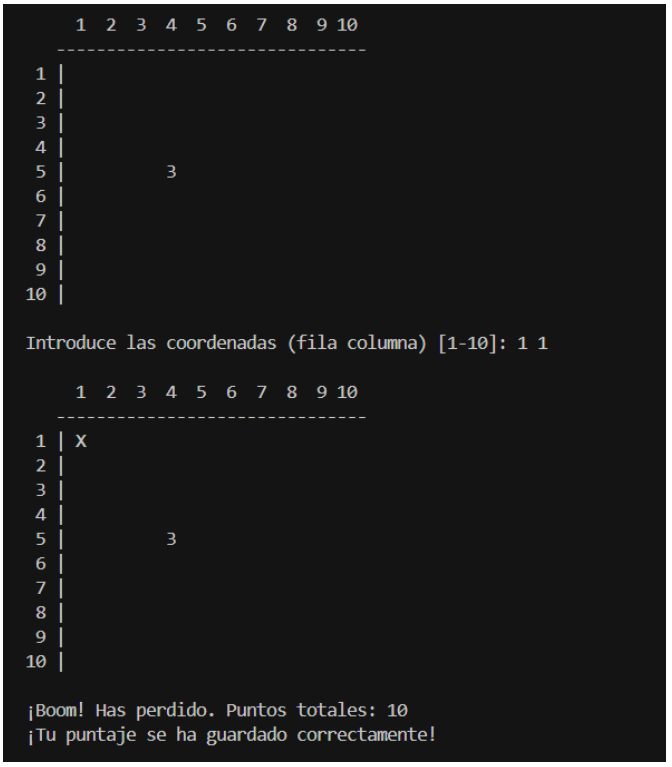
```
=== Buscaminas ===
1. Jugar
2. Ver Rankings
3. Salir
Selecciona una opción: 2

=== Rankings ===
1. Muniba - 10 puntos
```

2. **Introducció del nom del jugador:**
   Si selecciones l'opció 1. Jugar, abans de seleccionar el nivell de dificultat, el joc demana al jugador que introdueixi el seu nom, el qual s'associarà amb la seva puntuació.
3. **Selecció de dificultat:** Quan un jugador decideix jugar, el nostre joc li ofereix la possibilitat d'escollir entre tres nivells de dificultat:
   - **Baix:** Tauler de 6x6 amb 8 bombes.
   - **Mitjà:** Tauler de 8x8 amb 16 bombes.
   - **Difícil:** Tauler de 10x10 amb 32 bombes.
4. **Funcionament dins del joc:**
   - Durant la partida, el jugador introdueix les coordenades (fila i columna) de la casella que vol revelar.
   - Si les coordenades són **invàlides** (fora de límits) o la casella ja ha estat revelada, el joc ho indicarà i demanarà unes noves coordenades.
   - Cada vegada que el jugador revela una casella **sense bomba**, guanya **10 punts**.
   - Si la casella revelada està **buida**, mostrarà un número que indica quantes bombes hi ha al voltant d'aquella casella.
   - El joc continua fins que:
     - **Guanya:** Quan el jugador revela totes les caselles sense bombes.
     - **Perd:** Quan el jugador revela una casella amb una bomba.
5. **Finalització de la partida:**
   - En acabar la partida (sigui per victòria o derrota), es guarda la puntuació del jugador al sistema.
   - El jugador pot veure les seves puntuacions en la segona opció del menú: **Veure Rankings**.
6. **Veure rankings:**
   - Aquesta opció del menú permet accedir a les puntuacions dels millors jugadors.
   - El rànquing mostra els noms dels jugadors i els punts que han obtingut durant les seves partides

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)




## MEMBRES DEL GRUP

- Àngela Obón Soto - NIU 1600359
- Muniba Liaqat Ali Ali - NIU 1635042

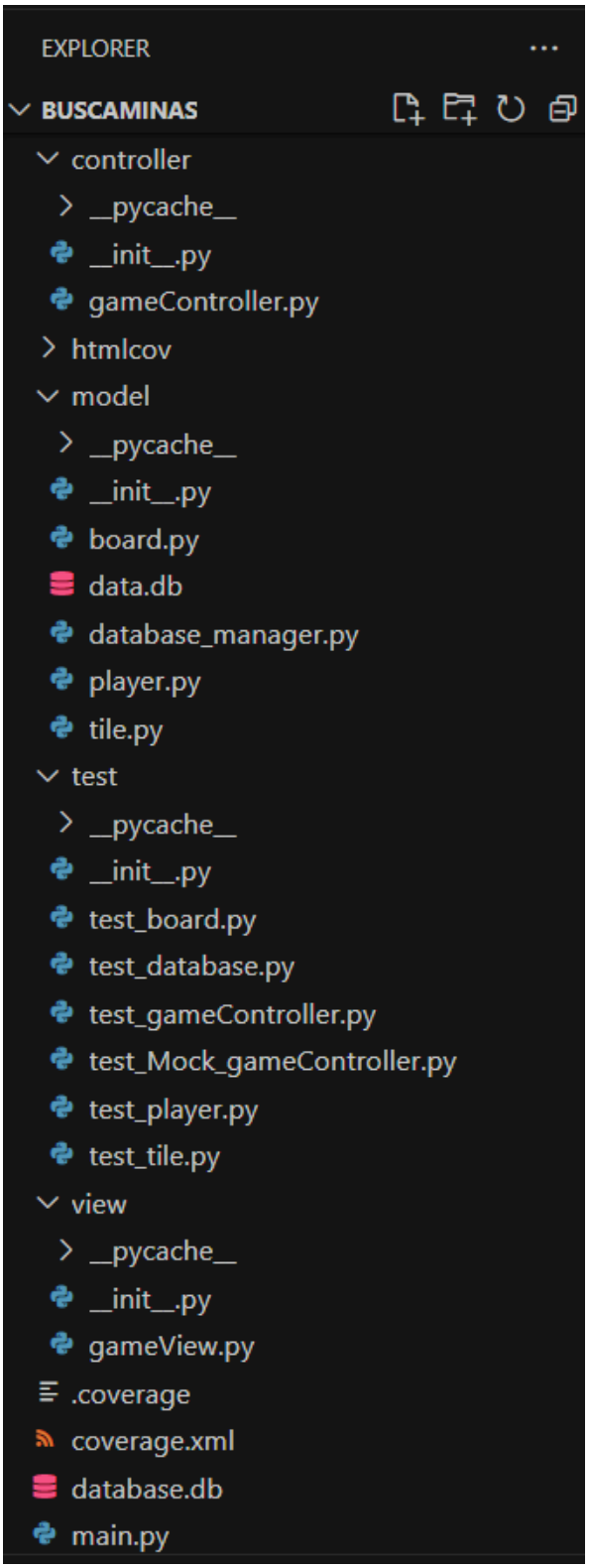
## LINK AL NOSTRE REPOSITORI GITHUB

**Repositori**: https://github.com/angela-uab/Buscaminas

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## ESTRUCTURA DEL NOSTRE PROJECTE: MODEL - VISTA - CONTROLADOR

El nostre projecte segueix l'estructura **MVC (Model-Vista-Controlador)**, amb una clara separació de responsabilitats. A la carpeta **model** es gestiona la lògica del joc, com la casella (**tile.py**), el tauler (**board.py**), els jugadors (player.py) i la base de dades (**database_manager.py**). A la carpeta **view** es troba la interacció amb l'usuari i la visualització del joc (**gameView.py**), i a **controller** hi ha el controlador (**gameController.py**), que coordina les accions de l'usuari amb la lògica del joc. A més, incloem la carpeta **test**, que conté proves automatitzades per validar el funcionament de cada component, com ara el tauler, la base de dades o el controlador, amb tècniques com mocks per simular comportaments.

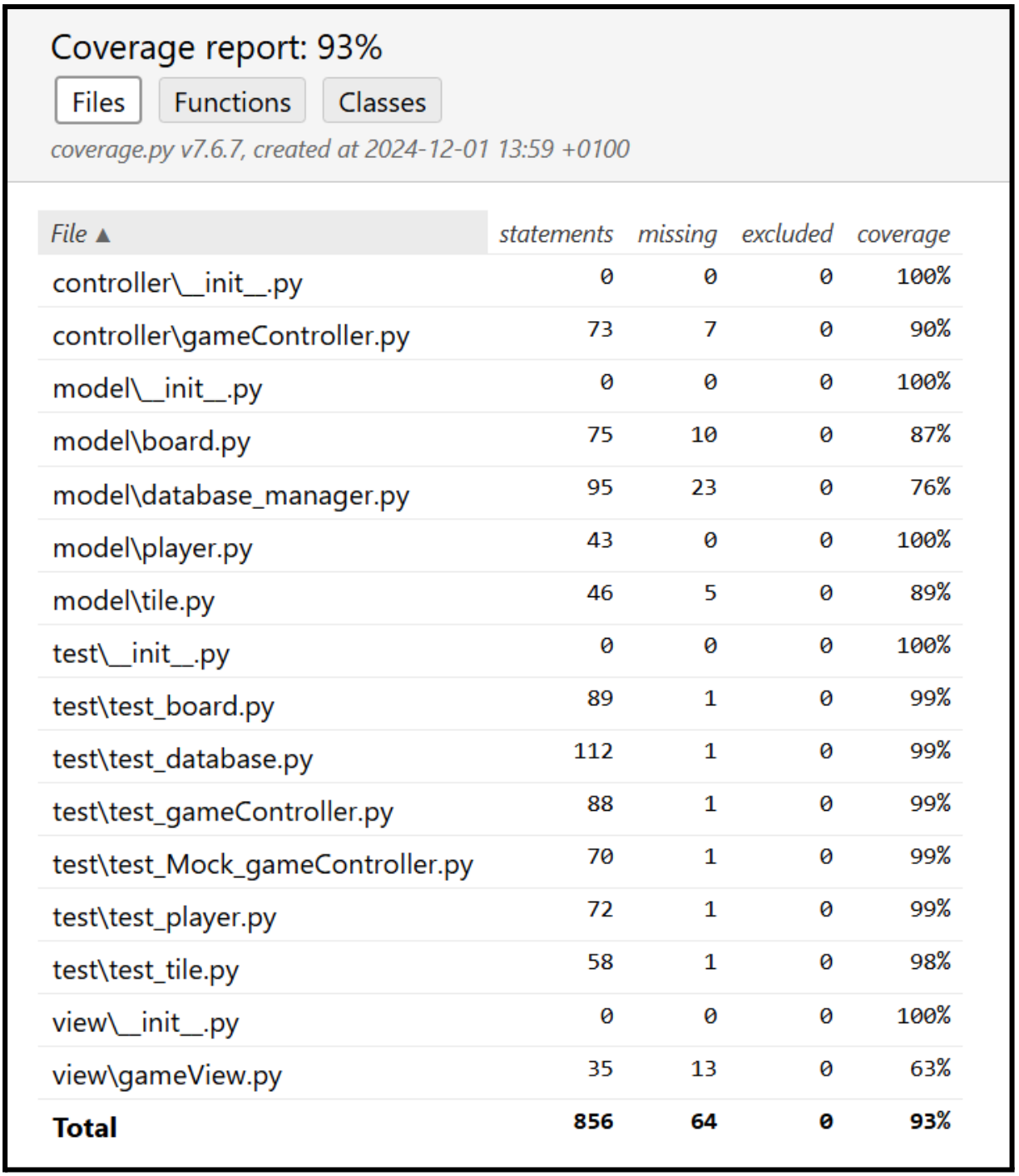



Tot el projecte s'ha desenvolupat seguint la metodologia **Test Driven Development (TDD)**, que consisteix a escriure primer els tests abans de començar a programar qualsevol funcionalitat. D'aquesta manera, s'assegura que el programari sigui comprovat de manera rigorosa durant totes les fases del desenvolupament. Aquest enfocament es pot observar clarament en els commits del repositori de GitHub, ja que abans d'implementar una nova funcionalitat, sempre es creen els tests corresponents. Pel que fa als fitxers, els tests relacionats amb el TDD es troben dins la carpeta de tests, juntament amb altres tipus de proves, com ara les de caixa blanca, caixa negra o pairwise, que es distingeixen per la seva finalitat específica.

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## STATEMENT COVERAGE

En el projecte hem aconseguit un **93% de cobertura de Statement Coverage**, tal com mostra la imatge. Això indica que la gran majoria de les línies de codi han estat executades durant les proves, assegurant una bona qualitat i fiabilitat del sistema.



Coverage report: 93%

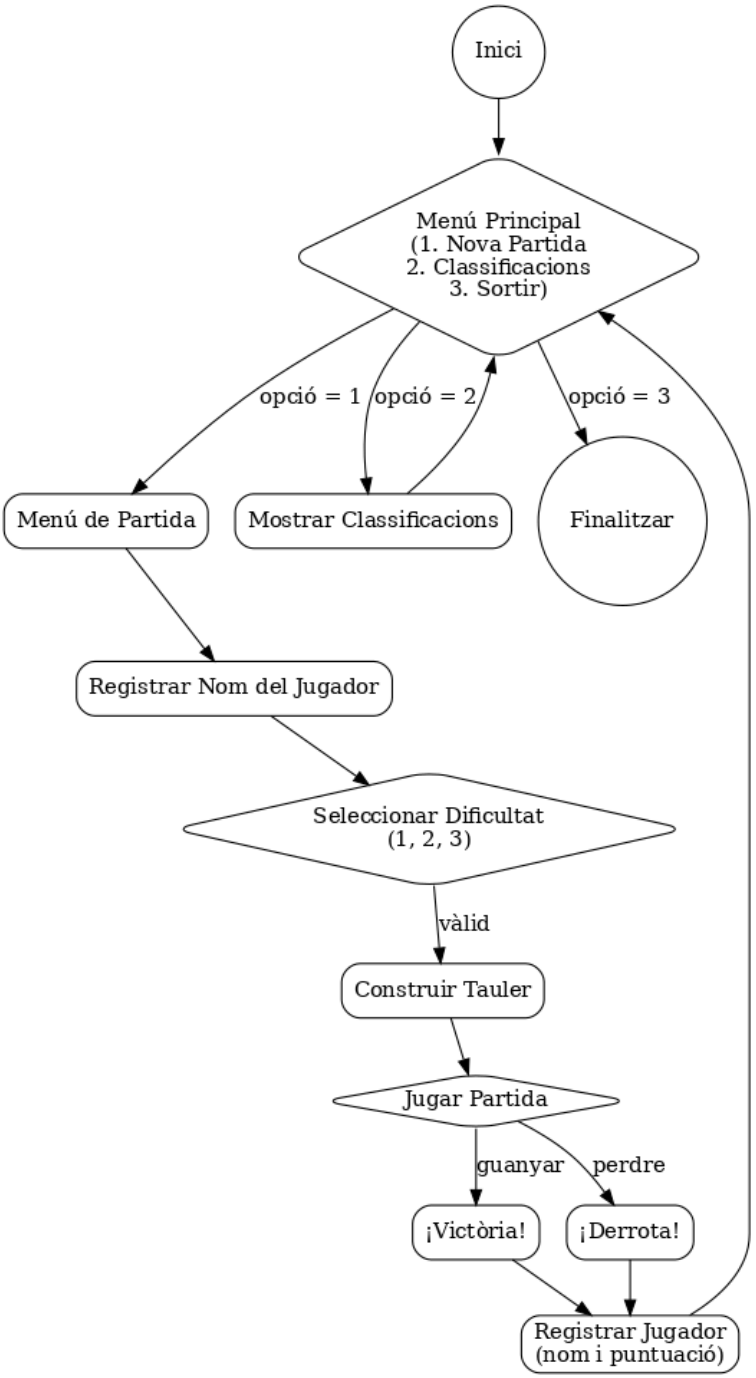
Files | Functions | Classes

coverage.py v7.6.7, created at 2024-12-01 13:59 +0100

| File ▲ | statements | missing | excluded | coverage |
|---|---|---|---|---|
| controller__init__.py | 0 | 0 | 0 | 100% |
| controller\gameController.py | 73 | 7 | 0 | 90% |
| model__init__.py | 0 | 0 | 0 | 100% |
| model\board.py | 75 | 10 | 0 | 87% |
| model\database_manager.py | 95 | 23 | 0 | 76% |
| model\player.py | 43 | 0 | 0 | 100% |
| model\tile.py | 46 | 5 | 0 | 89% |
| test__init__.py | 0 | 0 | 0 | 100% |
| test\test_board.py | 89 | 1 | 0 | 99% |
| test\test_database.py | 112 | 1 | 0 | 99% |
| test\test_gameController.py | 88 | 1 | 0 | 99% |
| test\test_Mock_gameController.py | 70 | 1 | 0 | 99% |
| test\test_player.py | 72 | 1 | 0 | 99% |
| test\test_tile.py | 58 | 1 | 0 | 98% |
| view__init__.py | 0 | 0 | 0 | 100% |
| view\gameView.py | 35 | 13 | 0 | 63% |
| **Total** | **856** | **64** | **0** | **93%** |

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## POSSIBLES PATHS DE LA NOSTRA LÒGICA



Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

# TESTS QUE HEM REALITZAT AL NOSTRE JOC

## Model/tile.py i test_tile.py

**Funcionalitat:** Implementació de la classe **Tile** per representar una casella en un joc de Buscamines. Inclou funcionalitats per establir si és bomba, revelar la casella, establir el nombre de bombes veïnes, i mostrar el seu estat actual.

**Localització:**

- **Arxiu:** tile.py
- **Classe:** Tile
- **Mètodes desenvolupats:**
  - __init__: Inicialització dels atributs de la casella.
  - is_bomb: Getter i setter per l'atribut _is_bomb.
  - is_revealed: Getter i setter per l'atribut _is_revealed.
  - neighboring_bombs: Getter i setter per l'atribut _neighboring_bombs.
  - reveal: Revela la casella.
  - display: Retorna una representació textual de l'estat de la casella.
  - _check_invariants: Verificació d'invariants.

**Test:**

- **Arxiu:** test_tile.py
- **Classe:** TestTile

**Mètodes de test associats:**

**test_tile_initialization:** Prova la inicialització de l'estat del Tile per verificar que tots els atributs comencin en els valors esperats.

➔ **Tipus de test:** Caixa negra.
➔ **Tècnica utilitzada:** Particions equivalents.

```
def test_tile_initialization(self):
    tile = Tile()
    self.assertFalse(tile.is_bomb)  # No debe ser bomba
    self.assertFalse(tile.is_revealed)  # No debe estar revelada
    self.assertEqual(tile.neighboring_bombs, 0)  # Vecinas deben ser 0
```

**test_tile_set_bomb:** Prova el setter de is_bomb assegurant que es pot establir correctament l'estat de bomba.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Hem decidit fer un **statement coverage** en aquest mètode.
  - ◆ Per fer-ho, hem assegurat:
    - Assignar un valor vàlid.
    - Assignar un valor invàlid per capturar l'error. D'aquesta manera, totes les instruccions s'han executat almenys un cop.

```
def test_tile_set_bomb(self):
    tile = Tile()
    tile.is_bomb = True
    self.assertTrue(tile.is_bomb)
```

**test_tile_reveal:** Prova de revelació de la casella. Comprova que després de revelar-la, l'estat is_revealed és correcte.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada**: Decision coverage.

```
def test_tile_reveal(self):
    tile = Tile()
    tile.reveal()
    self.assertTrue(tile.is_revealed)
```

**test_tile_display**: Prova les combinacions possibles de is_bomb i is_revealed per assegurar-se que el mètode display retorna la sortida correcta en cada cas.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Pairwise testing.
  - ◆ Hem fet testos:
    - Amb is_bomb=False i is_revealed=False.
    - Amb is_bomb=False i is_revealed=True.
    - Amb is_bomb=True i is_revealed=False.
    - Amb is_bomb=True i is_revealed=True. Així executem totes les línies de codi del mètode.

```python
def test_tile_display(self):
    tile = Tile()
    self.assertEqual(tile.display(), " ")  # Sin revelar, muestra espacio

    tile.reveal()
    self.assertEqual(tile.display(), "0")  # Revelada sin bomba muestra 0

    # Crear un nuevo tile o reiniciar el estado
    tile = Tile()
    tile.is_bomb = True
    tile.reveal()
    self.assertEqual(tile.display(), "X")  # Revelada con bomba muestra X
```

**test_neighboring_bombs_zero:** Prova per **valor límit**: el nombre de bombes veïnes és 0.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Valors límit.

```python
def test_neighboring_bombs_zero(self):
    tile = Tile()
    tile.neighboring_bombs = 0
    tile.reveal()
    self.assertEqual(tile.display(), "0")  # Sin bombas vecinas muestra 0
```

**test_neighboring_bombs_one:** Prova per **valor límit**: el nombre de bombes veïnes és 1.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Valors límit.

**test_neighboring_bombs_eight:** Prova per **valor límit:** el nombre de bombes veïnes és 8.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Valors límit.

**test_pairwise_combinations:** Prova de combinacions de diferents estats de la casella (is_bomb, is_revealed, neighboring_bombs).

- ➔ Tipus de test: Caixa negra.
- ➔ Tècnica utilitzada: Pairwise testing.
  - ◆ Hem fet testos per cobrir escenaris com:
    - is_bomb=False,is_revealed=False, neighboring_bombs=0.
    - is_bomb=False,is_revealed=True, neighboring_bombs=3.
    - is_bomb=True, is_revealed=False.
    - is_bomb=True, is_revealed=True.

```python
def test_pairwise_combinations(self):
    tile = Tile()

    # No bomba, no revelada
    tile.is_bomb = False
    tile.is_revealed = False
    tile.neighboring_bombs = 0
    self.assertEqual(tile.display(), " ")

    # No bomba, revelada
    tile.is_bomb = False
    tile.is_revealed = True
    tile.neighboring_bombs = 3
    self.assertEqual(tile.display(), "3")

    # Bomba, no revelada
    tile.is_bomb = True
    tile.is_revealed = False
    self.assertEqual(tile.display(), " ")

    # Bomba, revelada
    tile.is_bomb = True
    tile.is_revealed = True
    self.assertEqual(tile.display(), "X")
```

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

---

**Captures d'imatge:**

- Cobertura de línies per als tests (Coverage):

Coverage for **model\tile.py**: 89%

46 statements | 41 run | 5 missing | 0 excluded

« prev ^ index » next coverage.py v7.6.7, created at 2024-11-30 13:03 +0100

```
1  class Tile:
2      def __init__(self):
3          self._is_bomb = False
4          self._is_revealed = False
5          self._neighboring_bombs = 0
6
7          # Invariante: neighboring_bombs no puede ser negativo
8          self._check_invariants()
9
10     @property
11     def is_bomb(self):
12         return self._is_bomb
13
14     @is_bomb.setter
15     def is_bomb(self, value):
16         # Precondición: value debe ser un booleano
17         if not isinstance(value, bool):
18             raise ValueError("is_bomb debe ser un booleano")
19         self._is_bomb = value
20         self._check_invariants()
21
22     @property
23     def is_revealed(self):
24         return self._is_revealed
25
26     @is_revealed.setter
27     def is_revealed(self, value):
28         # Precondición: value debe ser un booleano
29         if not isinstance(value, bool):
30             raise ValueError("is_revealed debe ser un booleano")
31         self._is_revealed = value
32         self._check_invariants()
33
34     def reveal(self):
35         # Precondición: El Tile no debe estar ya revelado
36         if self._is_revealed:
37             raise ValueError("El Tile ya está revelado")
38         self._is_revealed = True
39         # Postcondición: Después de revelar, is_revealed debe ser True
40         assert self._is_revealed, "Error: El Tile no se marcó como revelado"
42     @property
43     def neighboring_bombs(self):
44         return self._neighboring_bombs
45
46     @neighboring_bombs.setter
47     def neighboring_bombs(self, value):
48         # Precondición: value debe ser un entero no negativo
49         if not isinstance(value, int) or value < 0:
50             raise ValueError("neighboring_bombs debe ser un entero no negativo")
51         self._neighboring_bombs = value
52         self._check_invariants()
53
54     def display(self):
55         # Invariante: neighboring_bombs debe estar definido
56         assert self._neighboring_bombs >= 0, "Error: neighboring_bombs no válido"
57         if self.is_revealed:
58             return "X" if self.is_bomb else str(self.neighboring_bombs)
59         return " "
60
61     def _check_invariants(self):
62         # Invariante: neighboring_bombs no puede ser negativo
63         if self._neighboring_bombs < 0:
64             raise ValueError("neighboring_bombs no puede ser negativo")
```

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## Model/board.py i test_board.py

**Funcionalitat:** Implementació de la classe **Board** per gestionar la lògica principal d'un tauler de Buscamines. Inclou funcionalitats com la inicialització del tauler, el càlcul de bombes veïnes, el revelatge de caselles, la verificació de condicions de victòria o derrota, i la representació visual del tauler.

**Localització:**

- **Arxiu:** board.py
- **Classe:** Board
- **Mètodes desenvolupats:**
  - __init__: Inicialització del tauler amb configuracions basades en la dificultat.
  - _initialize_board: Configura les caselles i col·loca les bombes.
  - _count_neighboring_bombs: Calcula les bombes adjacents a una casella.
  - reveal_tile: Revela una casella específica i actualitza l'estat del joc.
  - get_valid_coordinates: Obté coordenades vàlides des de l'entrada de l'usuari.
  - is_game_won: Verifica si s'ha guanyat el joc.
  - display_board: Mostra el tauler actual.
  - _check_invariants: Comprova els invariants del tauler.

**Test:**

- **Arxiu:** test_board.py
- **Classe:** TestBoard

**Mètodes de test associats:**

**test_board_initialization:** Prova la inicialització del tauler per cada dificultat.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage i valor límit
- ➔ **Hem decidit fer un statement coverage en aquest mètode.** Per fer-ho, hem considerat:
  - ◆ Inicialitzar el tauler amb cada nivell de dificultat (1, 2, 3).
  - ◆ Verificar que els atributs size i total_bombs coincideixen amb la configuració.
- ➔ D'aquesta manera, totes les instruccions de la inicialització s'han executat almenys un cop.

```python
def test_board_initialization(self):
    board = Board(1)
    self.assertEqual(board.size, 6)
    self.assertEqual(board.total_bombs, 8)

    board = Board(2)
    self.assertEqual(board.size, 8)
    self.assertEqual(board.total_bombs, 16)

    board = Board(3)
    self.assertEqual(board.size, 10)
    self.assertEqual(board.total_bombs, 32)
```

**test_invalid_difficulty:** Prova que es llança un error per dificultats fora de rang (0 o 4).

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Valors límit.

```python
def test_invalid_difficulty(self):
    with self.assertRaises(ValueError):  # Frontera inferior
        Board(0)

    with self.assertRaises(ValueError):  # Frontera superior
        Board(4)
```

**test_bomb_counting:** Comprova que el nombre de bombes assignades al tauler coincideix amb la configuració.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Loop testing.
- ➔ **Hem decidit fer un statement coverage en aquest mètode.** Per fer-ho, hem de:
  - ◆ No recorrer el bucle de col·locació de bombes (tauler buit, 0 bombes).
  - ◆ Recórrer el bucle almenys un cop (tauler amb bombes).
- ➔ D'aquesta manera, totes les instruccions s'hauran executat almenys un cop.

```python
def test_bomb_counting(self):
    board = Board(1)
    total_bombs = sum(tile.is_bomb for row in board.tiles for tile in row)
    self.assertEqual(total_bombs, board.total_bombs)
```

**test_reveal_tile:** Prova que una casella es revela correctament i retorna el resultat esperat.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.
- ➔ Hem decidit provar:
  - ◆ Revelar una casella no bomba.
  - ◆ Revelar una casella bomba.
  - ◆ Intentar revelar una casella ja revelada.
  - ◆ D'aquesta manera, totes les possibles línies de codi s'han executat.

```
def test_reveal_tile(self):
    board = Board(1)
    success = board.reveal_tile(0, 0)
    self.assertTrue(board.tiles[0][0].is_revealed)
    self.assertTrue(success)
```

**test_get_valid_coordinates:** Simula entrades de l'usuari per obtenir coordenades vàlides.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Mocking amb patch.
- ➔ **Hem decidit fer un statement coverage en aquest mètode.**
  - ◆ Per fer-ho, hem fet proves amb les següents entrades:
    - Entrar un nombre vàlid.
    - Entrar un altre nombre vàlid.
    - Entrar un valor no numèric i després un valor vàlid.
    - Entrar un valor no numèric, després un valor no vàlid i finalment un valor vàlid.
    - Entrar un valor no vàlid i després un valor vàlid.
  - ➔ Així, executem totes les línies de codi del mètode en qüestió.

```
def test_get_valid_coordinates(self):
    board = Board(1)

    def mock_input(prompt):
        return "1 1"  # Mock para simular coordenadas válidas

    with patch("builtins.input", mock_input):
        x, y = board.get_valid_coordinates()
        self.assertEqual((x, y), (0, 0))  # Ajustado por índice basado en 0
```

**test_game_lost_condition:** Comprova que es perd el joc després de revelar una casella amb bomba.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Decision coverage.

```
def test_game_lost_condition(self):
    board = Board(1)
    for x in range(board.size):
        for y in range(board.size):
            if board.tiles[x][y].is_bomb:
                board.reveal_tile(x, y)
                self.assertTrue(board.is_game_lost)
                return
```

**test_game_win_condition:** Simula revelar totes les caselles no bomba per verificar la condició de victòria.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Loop testing + Path coverage.

```
def test_game_win_condition(self):
    board = Board(1)
    for row in board.tiles:
        for tile in row:
            if not tile.is_bomb:
                tile.reveal()
    self.assertTrue(board.is_game_won())
```

**test_pairwise_board_configurations:** Prova combinacions de diferents mides de tauler i nombre de bombes.

- ➔ **Tipus de test:** Caixa negra**.**
- ➔ **Tècnica utilitzada:** Pairwise testing.

```
def test_pairwise_board_configurations(self):
    configurations = [
        (6, 8),   # Fácil
        (8, 16),  # Medio
        (10, 32), # Difícil
    ]
    for size, bombs in configurations:
        board = Board(1)
        board.size = size
        board.total_bombs = bombs
        self.assertEqual(board.size, size)
        self.assertEqual(board.total_bombs, bombs)
```

**test_reveal_already_revealed_tile:** Prova que intentar revelar una casella ja revelada retorna el resultat esperat.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.

```
def test_reveal_already_revealed_tile(self):
    board = Board(1)
    board.tiles[0][0].reveal()  # Revelar la casilla primero
    result = board.reveal_tile(0, 0)
    self.assertFalse(result)  # Intentar revelar de nuevo debe devolver False
```

**test_empty_board:** Simula un tauler sense bombes i verifica que es guanya automàticament.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Decision coverage.

```
def test_empty_board(self):
    board = Board(1)
    for row in board.tiles:
        for tile in row:
            tile.is_bomb = False
            tile.reveal()  # Revelar todas las casillas
    self.assertTrue(board.is_game_won())  # Todas las casillas son seguras, debería ganar
```

**test_bomb_count_within_bounds:** Verifica que el nombre de bombes assignades al tauler està dins dels límits (no supera el nombre de caselles disponibles).

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Hem decidit fer un **statement coverage** en aquest mètode. Per fer-ho, hem verificat:
  - ◆ Que el nombre total de bombes sigui menor o igual que el nombre de caselles (size * size).
- ➔ D'aquesta manera, totes les instruccions del mètode _initialize_board relacionades amb la col·locació de bombes són executades almenys un cop.
- ➔ Aquesta prova assegura que no es generen més bombes de les que el tauler pot contenir.

```
def test_bomb_count_within_bounds(self):
    board = Board(1)
    bomb_count = sum(tile.is_bomb for row in board.tiles for tile in row)
    self.assertLessEqual(bomb_count, board.size * board.size)  # Bombas no deben exceder casillas totales
```

**test_board_edge_sizes:** Prova el comportament del tauler en condicions límit de mida:

- ➔ **Tauler mínim:** 2x2 amb 1 bomba.
- ➔ **Tauler gran:** 50x50 amb 100 bombes.
- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Valors límit.
- ➔ Hem decidit provar aquests casos límit per assegurar-nos que el sistema funciona correctament en escenaris poc comuns.
    - ◆ Casos coberts:
        - Inicialitzar un tauler petit i verificar les dimensions i el nombre de bombes.
        - Inicialitzar un tauler gran i assegurar que també manté la configuració esperada.

```python
def test_board_edge_sizes(self):
    # Tablero mínimo
    small_board = Board(1)
    small_board.size = 2
    small_board.total_bombs = 1
    self.assertEqual(small_board.size, 2)
    self.assertEqual(small_board.total_bombs, 1)

    # Tablero grande
    large_board = Board(1)
    large_board.size = 50
    large_board.total_bombs = 100
    self.assertEqual(large_board.size, 50)
    self.assertEqual(large_board.total_bombs, 100)
```

**test_validate_coordinates:** Prova la validació de coordenades segons diferents condicions.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Condition Coverage
- ➔ Hem decidit provar:
    - ◆ Ambdues condicions són certes: Quan x i y estan dins dels límits del tauler.
    - ◆ Primera condició falsa: Quan x està fora dels límits (per sota de 0)
    - ◆ Segona condició falsa: Quan y està fora dels límits (igual o superior a la mida del tauler).
    - ◆ Ambdues condicions falses: Quan tant x com y estan fora dels límits.

Aquest test assegura que el mètode de validació de coordenades verifica correctament les dues condicions per determinar si una posició és vàlida o no dins del tauler, abastant tots els possibles valors combinatoris de les condicions.

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

```python
def test_validate_coordinates(self):
        # Ambas condiciones verdaderas
        x, y = 0, 0
        self.assertTrue(0 <= x < self.board.size and 0 <= y < self.board.size)

        # Primera condición falsa
        x, y = -1, 0
        self.assertFalse(0 <= x < self.board.size and 0 <= y < self.board.size)

        # Segunda condición falsa
        x, y = 0, self.board.size
        self.assertFalse(0 <= x < self.board.size and 0 <= y < self.board.size)

        # Ambas condiciones falsas
        x, y = -1, self.board.size
        self.assertFalse(0 <= x < self.board.size and 0 <= y < self.board.size)
```

**Captures de pantalla:**

- Cobertura de línies per als tests (Coverage):

Coverage for **model\board.py**: 87%

75 statements | 65 run | 10 missing | 0 excluded

« prev ^ index » next coverage.py v7.6.7, created at 2024-11-30 13:03 +0100

```python
import random
from model.tile import Tile

class Board:
    DIFFICULTY_SETTINGS = {
        1: {"size": 6, "bombs": 8},
        2: {"size": 8, "bombs": 16},
        3: {"size": 10, "bombs": 32},
    }

    def __init__(self, difficulty):
        # Precondición: La dificultad debe ser un valor válido (1, 2 o 3)
        if difficulty not in self.DIFFICULTY_SETTINGS:
            raise ValueError("La dificultad debe ser 1, 2 o 3.")

        # Inicialización
        self.size = self.DIFFICULTY_SETTINGS[difficulty]["size"]
        self.total_bombs = self.DIFFICULTY_SETTINGS[difficulty]["bombs"]
        self.tiles = self._initialize_board()
        self.is_game_lost = False

        # Invariante: El número total de bombas debe coincidir con la configuración
        self._check_invariants()

    def _initialize_board(self):
        # Inicializar el tablero y colocar las bombas
        tiles = [[Tile() for _ in range(self.size)] for _ in range(self.size)]
        bomb_positions = random.sample(range(self.size * self.size), self.total_bombs)

        for pos in bomb_positions:
            row, col = divmod(pos, self.size)
            tiles[row][col].is_bomb = True

        # Calcular las bombas vecinas para cada casilla
        for x in range(self.size):
            for y in range(self.size):
                tiles[x][y].neighboring_bombs = self._count_neighboring_bombs(tiles, x, y)

        # Postcondición: Asegurar que el número total de bombas en el tablero sea correcto
        total_bombs_counted = sum(tile.is_bomb for row in tiles for tile in row)
        assert total_bombs_counted == self.total_bombs, "Número de bombas en el tablero no coincide con la configuración."

        return tiles
```

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

```
44
45      def _count_neighboring_bombs(self, tiles, x, y):
46          # Precondiciones: Coordenadas válidas
47          assert 0 <= x < self.size, f"x está fuera de rango: {x}"
48          assert 0 <= y < self.size, f"y está fuera de rango: {y}"
49
50          # Contar las bombas vecinas
51          count = 0
52          for i in range(max(0, x - 1), min(self.size, x + 2)):
53              for j in range(max(0, y - 1), min(self.size, y + 2)):
54                  if tiles[i][j].is_bomb:
55                      count += 1
56
57          # Postcondición: El conteo de bombas debe ser un número no negativo
58          assert count >= 0, "El conteo de bombas vecinas no puede ser negativo."
59          return count
60
61      def reveal_tile(self, x, y):
62          # Precondiciones: Coordenadas válidas
63          assert 0 <= x < self.size, f"x está fuera de rango: {x}"
64          assert 0 <= y < self.size, f"y está fuera de rango: {y}"
65
66          # Comprobar si la casilla ya está revelada
67          if self.tiles[x][y].is_revealed:
68              print("La casilla ya está revelada. Por favor, selecciona otra.")
69              return False  # Indica que la casilla no se pudo revelar
70
71          if self.tiles[x][y].is_bomb:
72              self.is_game_lost = True
73          self.tiles[x][y].reveal()
74
75          # Postcondición: La casilla debe estar marcada como revelada
76          assert self.tiles[x][y].is_revealed, "La casilla no se marcó como revelada correctamente."
77          return True  # Indica que la casilla se reveló con éxito
78
79      def get_valid_coordinates(self):
80          while True:
81              try:
82                  input_data = input(f"Introduce las coordenadas (fila columna) [1-{self.size}]: ")
83                  x, y = map(int, input_data.split())
84                  x -= 1  # Ajustar a índice basado en 0
85                  y -= 1  # Ajustar a índice basado en 0
86
87                  if not (0 <= x < self.size and 0 <= y < self.size):
88                      print(f"Coordenadas fuera de rango. Introduce valores entre 1 y {self.size}.")
89                      continue
90
91                  if not self.reveal_tile(x, y):
92                      continue
94                  return x, y
95              except ValueError:
96                  print("Entrada inválida. Introduce dos números separados por un espacio.")
97
98      def is_game_won(self):
99          # Verificar si todas las casillas no bomba están reveladas
100         for row in self.tiles:
101             for tile in row:
102                 # Invariante: Una casilla que no es bomba y no está revelada indica que el juego no está ganado
103                 if not tile.is_bomb and not tile.is_revealed:
104                     return False
105         return True
106
107     def display_board(self):
108         # Invariante: La longitud de cada fila debe coincidir con el tamaño del tablero
109         for row in self.tiles:
110             assert len(row) == self.size, "El tamaño de la fila no coincide con el tamaño del tablero."
111         # Mostrar el tablero
112         for row in self.tiles:
113             print(" ".join(tile.display() for tile in row))
114
115     def _check_invariants(self):
116         # Invariante: El tamaño debe ser consistente con las configuraciones
117         assert self.size > 0, "El tamaño del tablero debe ser mayor que 0."
118         assert self.total_bombs >= 0, "El número de bombas debe ser no negativo."
119
120         # Invariante: Asegurar que el número total de bombas no excede el número total de casillas
121         assert self.total_bombs <= self.size * self.size, "Número de bombas excede el número total de casillas."
```

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## Model/database_manager.py i test_database.py

**Funcionalitat:** Implementació de la classe DatabaseManager per gestionar una base de dades SQLite. Inclou funcionalitats per inicialitzar la base de dades, afegir, eliminar, actualitzar i recuperar informació de jugadors

**Localització:**

- **Arxiu:** database_manager.py
- **Classe:** DatabaseManager
- **Mètodes desenvolupats:**
  - __init__: Inicialitza el gestor i la connexió a la base de dades.
  - _connect: Estableix la connexió amb la base de dades.
  - _initialize_database: Crea la taula players si no existeix.
  - insert_player: Insereix un jugador a la base de dades.
  - get_all_players: Retorna tots els jugadors.
  - get_player_by_id: Retorna un jugador pel seu ID.
  - delete_player: Elimina un jugador pel seu ID.
  - update_player_score: Actualitza el puntatge d'un jugador pel seu ID.
  - get_top_players: Retorna els jugadors amb millor puntuació.
  - close: Tanca la connexió amb la base de dades.

**Test:**

- **Arxiu:** test_database.py
- **Classe:** TestDatabaseManager

**Mètodes de test associats:**

**test_invalid_insert_player:** Verifica que es gestionen correctament les entrades no vàlides per inserir un jugador.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Particions equivalents.
- ➔ Hem decidit provar:
  - ◆ Inserir un nom buit.
  - ◆ Inserir un nom None.
  - ◆ Inserir un puntatge no numèric.
  - ◆ Inserir un puntatge negatiu.

```python
def test_invalid_insert_player(self):
    with self.assertRaises(ValueError):
        self.db.insert_player("", 10)  # Nombre vacío no permitido
    with self.assertRaises(ValueError):
        self.db.insert_player(None, 10)  # Nombre None no permitido
    with self.assertRaises(ValueError):
        self.db.insert_player("Alice", "invalid_score")  # Puntaje debe ser un número
    with self.assertRaises(ValueError):
        self.db.insert_player("Alice", -1)  # Puntaje negativo no permitido
```

**test_invalid_get_player_by_id:** Verifica el maneig d'errors al buscar un jugador amb un ID no vàlid.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Hem decidit provar:
  - ◆ Buscar un jugador amb un ID no numèric.
  - ◆ Buscar un jugador amb un ID inexistent.

```python
def test_invalid_get_player_by_id(self):
    player_id = self.db.insert_player("Alice", 10)  # Insertamos un jugador válido
    self.assertIsNotNone(player_id, "El jugador no fue insertado correctamente.")

    with self.assertRaises(ValueError):
        self.db.get_player_by_id("invalid_id")  # ID no es un entero
    player = self.db.get_player_by_id(999)  # ID inexistente
    self.assertIsNone(player)
```

**test_invalid_delete_player:** Verifica el maneig d'errors a l'eliminar un jugador amb un ID no vàlid.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Hem decidit provar:
  - ◆ Eliminar un jugador amb un ID no numèric.
  - ◆ Eliminar un jugador amb un ID inexistent.

```python
def test_invalid_delete_player(self):
    player_id = self.db.insert_player("Alice", 10)  # Insertamos un jugador válido
    self.assertIsNotNone(player_id, "El jugador no fue insertado correctamente.")

    with self.assertRaises(ValueError):
        self.db.delete_player("invalid_id")  # ID no es un entero
    result = self.db.delete_player(999)  # ID inexistente
    self.assertFalse(result, "Se esperaba que no se eliminara ninguna fila para un ID inexistente.")
```

**test_insert_two_players:** Prova la inserció de múltiples jugadors i la seva recuperació.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.
- ➔ Hem decidit provar:
  - ◆ Inserir un jugador amb un nom i puntatge vàlids.
  - ◆ Inserir un segon jugador amb un nom i puntatge vàlids.
  - ◆ Recuperar tots els jugadors de la base de dades.

```python
def test_insert_two_players(self):
    result_alice = self.db.insert_player("Alice", 10)
    result_bob = self.db.insert_player("Bob", 25)

    self.assertIsNotNone(result_alice, "El jugador 'Alice' no fue insertado correctamente.")
    self.assertIsNotNone(result_bob, "El jugador 'Bob' no fue insertado correctamente.")

    players = self.db.get_all_players()
    self.assertEqual(len(players), 2)
    self.assertEqual(players[0][1], "Alice")
    self.assertEqual(players[0][2], 10)
    self.assertEqual(players[1][1], "Bob")
    self.assertEqual(players[1][2], 25)
```

```python
    def insert_player(self, name, score):
        """Inserta un jugador en la base de datos y devuelve su ID."""
        # Precondiciones
        if not isinstance(name, str) or not name.strip():
            raise ValueError("El nombre debe ser una cadena no vacía.")
        if not isinstance(score, int) or score < 0:
            raise ValueError("El puntaje debe ser un entero no negativo.")

        try:
            cursor = self.connection.execute(
                "INSERT INTO players (name, score) VALUES (?, ?)", (name, score)
            )
            self.connection.commit()

            # Postcondición: El ID del jugador insertado debe ser mayor a 0.
            assert cursor.lastrowid > 0, "El jugador no fue insertado correctamente."
            return cursor.lastrowid
```

**test_delete_player:** Prova el procés d'eliminació d'un jugador per ID.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.
- ➔ Hem decidit provar:
  - ◆ Inserir dos jugadors.
  - ◆ Eliminar un jugador per ID.
  - ◆ Comprovar que només queda el segon jugador a la base de dades.

```python
def test_delete_player(self):
    player_id_alice = self.db.insert_player("Alice", 10)
    player_id_bob = self.db.insert_player("Bob", 25)

    self.assertIsNotNone(player_id_alice, "El jugador 'Alice' no fue insertado correctamente.")
    self.assertIsNotNone(player_id_bob, "El jugador 'Bob' no fue insertado correctamente.")

    result = self.db.delete_player(player_id_alice)
    self.assertTrue(result, "El jugador 'Alice' no fue eliminado correctamente.")

    players = self.db.get_all_players()
    self.assertEqual(len(players), 1)
    self.assertEqual(players[0][1], "Bob")
```

```python
    def delete_player(self, player_id):
        """Elimina un jugador por su ID."""
        # Precondición: player_id debe ser un entero positivo.
        if not isinstance(player_id, int) or player_id <= 0:
            raise ValueError("El ID del jugador debe ser un entero positivo.")

        try:
            cursor = self.connection.execute("DELETE FROM players WHERE id = ?", (player_id,))
            self.connection.commit()

            # Postcondición: El número de filas afectadas debe ser 0 o 1.
            assert cursor.rowcount in (0, 1), "Número inesperado de filas eliminadas."
            return cursor.rowcount > 0
```

**test_get_player_by_id:** Prova la cerca d'un jugador pel seu ID.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Decision coverage.
- ➔ Hem decidit provar:
  - ◆ Cercar un jugador existent pel seu ID.
  - ◆ Cercar un jugador inexistent pel seu ID.

```python
def test_get_player_by_id(self):
    player_id = self.db.insert_player("Alice", 10)
    self.assertIsNotNone(player_id, "El jugador no fue insertado correctamente.")

    player = self.db.get_player_by_id(player_id)
    self.assertIsNotNone(player, f"No se encontró el jugador con ID {player_id}.")
    self.assertEqual(player[1], "Alice")
    self.assertEqual(player[2], 10)
```

```python
    def get_player_by_id(self, player_id):
        """Devuelve un jugador por su ID."""
        # Precondición: player_id debe ser un entero positivo.
        if not isinstance(player_id, int) or player_id <= 0:
            raise ValueError("El ID del jugador debe ser un entero positivo.")

        try:
            cursor = self.connection.execute("SELECT * FROM players WHERE id = ?", (player_id,))
            player = cursor.fetchone()

            # Postcondición: El resultado puede ser None o una tupla.
            assert player is None or isinstance(player, tuple), "El resultado no es válido."
            return player
```

**test_update_player_score:** Prova l'actualització del puntatge d'un jugador existent.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.
- ➔ Hem decidit provar:
  - ◆ Inserir un jugador amb un puntatge inicial.
  - ◆ Actualitzar el puntatge del jugador.
  - ◆ Verificar que el puntatge actualitzat es reflecteix correctament.

```python
def test_update_player_score(self):
    player_id = self.db.insert_player("Alice", 10)
    self.assertIsNotNone(player_id, "El jugador no fue insertado correctamente.")

    result = self.db.update_player_score(player_id, 50)
    self.assertTrue(result, f"No se pudo actualizar el puntaje del jugador con ID {player_id}.")

    player = self.db.get_player_by_id(player_id)
    self.assertIsNotNone(player, f"No se encontró el jugador con ID {player_id} tras actualizar la puntuación.")
    self.assertEqual(player[2], 50)
```

```python
    def update_player_score(self, player_id, new_score):
        """Actualiza el puntaje de un jugador por su ID."""
        # Precondiciones
        if not isinstance(player_id, int) or player_id <= 0:
            raise ValueError("El ID del jugador debe ser un entero positivo.")
        if not isinstance(new_score, int) or new_score < 0:
            raise ValueError("El puntaje debe ser un entero no negativo.")

        try:
            cursor = self.connection.execute(
                "UPDATE players SET score = ? WHERE id = ?", (new_score, player_id)
            )
            self.connection.commit()

            # Postcondición: El número de filas afectadas debe ser 0 o 1.
            assert cursor.rowcount in (0, 1), "Número inesperado de filas actualizadas."
            return cursor.rowcount > 0
```

**test_get_top_players:** Prova la recuperació dels jugadors amb els millors puntatges.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Particions equivalent
- ➔ Hem decidit provar:
  - ◆ Inserir diversos jugadors amb diferents puntatges.
  - ◆ Recuperar els dos millors jugadors de la base de dades.

```python
def test_get_top_players(self):
    self.db.insert_player("Alice", 10)
    self.db.insert_player("Bob", 25)
    self.db.insert_player("Charlie", 15)

    top_players = self.db.get_top_players(2)
    self.assertEqual(len(top_players), 2)
    self.assertEqual(top_players[0][1], "Bob")  # Mejor puntuación
    self.assertEqual(top_players[1][1], "Charlie")  # Segunda mejor puntuación
```

**test_get_top_players_limit_cases:** Prova la recuperació dels millors jugadors en casos límit.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Valors límit.
- ➔ Hem decidit provar:
  - ◆ Recuperar jugadors quan la base de dades està buida.
  - ◆ Gestionar un límit de 0 jugadors, que no és vàlid.
  - ◆ Recuperar més jugadors dels que hi ha registrats a la base de dades.
  - ◆ Recuperar exactament el mateix nombre de jugadors que hi ha registrats.
  - ◆ Recuperar un sol jugador (límit igual a 1).
  - ◆ Gestionar una base de dades amb només un jugador.
  - ◆ Gestionar jugadors amb puntuacions iguals i verificar que es poden recuperar correctament.

```python
def test_get_top_players_limit_cases(self):
    # Caso 1: Base de datos vacía
    top_players = self.db.get_top_players(1)
    self.assertEqual(len(top_players), 0)

    # Caso 2: Límite igual a 0
    with self.assertRaises(ValueError):  # Límite no puede ser 0
        self.db.get_top_players(0)

    # Caso 3: Límite mayor al número total de jugadores
    self.db.insert_player("Alice", 50)
    self.db.insert_player("Bob", 100)
    top_players = self.db.get_top_players(10)  # Más de los jugadores registrados
    self.assertEqual(len(top_players), 2)
    self.assertEqual(top_players[0][1], "Bob")  # Mayor puntaje
    self.assertEqual(top_players[1][1], "Alice")
```

Aquest test verifica que el mètode get_top_players maneja correctament tant els casos límit com situacions especials, assegurant-se que el comportament sigui coherent i robust en tots els escenaris.

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## Model/player.py i test_player.py

**Funcionalitat:** Implementació de la classe **Player** per representar un jugador amb atributs com ID, nom i puntuació. La classe inclou funcionalitats per inicialitzar, validar i actualitzar aquests atributs, assegurant que es mantenen les precondicions i invariants.

**Localització:**

- **Arxiu:** player.py
- **Classe:** Player
- **Mètodes desenvolupats:**
  - __init__: Inicialitza un jugador amb ID, nom i puntuació, validant les precondicions.
  - get_id: Retorna l'ID del jugador assegurant que és un enter no negatiu.
  - set_id: Assigna un nou ID al jugador després de validar-lo.
  - get_name: Retorna el nom del jugador, assegurant que és una cadena no buida.
  - set_name: Assigna un nou nom al jugador després de validar-lo.
  - get_score: Retorna la puntuació del jugador, assegurant que és un número no negatiu.
  - set_score: Assigna una nova puntuació al jugador després de validar-la.
  - _check_invariants: Comprova que es compleixen els invariants de la classe (ID, nom i puntuació són vàlids).

**Test:**

- **Arxiu:** test_player.py
- **Classe:** TestPlayer

**Mètodes de test associats:**

**test_initialization:** Verifica que totes les sentències del constructor es completen correctament.

➔ **Tipus de test:** Caixa blanca.
➔ **Tècnica utilitzada:** Statement coverage.
➔ Hem decidit provar:
  ◆ Crear un jugador amb ID, nom i puntuació vàlids.
  ◆ Recuperar els valors per comprovar que es van assignar correctament.

```
def test_initialization(self):
    self.assertEqual(self.player.get_id(), 1)
    self.assertEqual(self.player.get_name(), "Alice")
    self.assertEqual(self.player.get_score(), 10)
```

**test_set_valid_id:** Prova assignar un ID vàlid al jugador.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Particions equivalents i valors límit.
- ➔ Hem decidit provar:
  - ◆ Assignar l'ID 0 (límit inferior vàlid).
  - ◆ Assignar un ID positiu aleatori (partició vàlida).

```
def test_set_valid_id(self):
    self.player.set_id(0)  # Límite inferior válido
    self.assertEqual(self.player.get_id(), 0)
    self.player.set_id(100)  # Valor aleatorio dentro del rango válido
    self.assertEqual(self.player.get_id(), 100)
```

**test_set_invalid_id:** Verifica que es gestionen correctament els ID no vàlids.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Particions equivalents.
- ➔ Hem decidit provar:
  - ◆ Assignar un ID de tipus no vàlid (cadena).
  - ◆ Assignar un ID negatiu.

```
def test_set_invalid_id(self):
    with self.assertRaises(ValueError):  # Tipo inválido
        self.player.set_id("invalid_id")
    with self.assertRaises(ValueError):  # Valor negativo
        self.player.set_id(-1)
```

**test_set_valid_name:** Prova assignar noms vàlids al jugador.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Particions equivalents.
- ➔ Hem decidit provar:
  - ◆ Assignar un nom vàlid.
  - ◆ Assignar un nom amb espais al voltant.

```
def test_set_valid_name(self):
    self.player.set_name("Bob")  # Caso válido
    self.assertEqual(self.player.get_name(), "Bob")
    self.player.set_name("   John   ")  # Nombre con espacios
    self.assertEqual(self.player.get_name(), "   John   ")
```

**test_set_invalid_name:** Verifica que es gestionen correctament els noms no vàlids.

- **Tipus de test:** Caixa negra.
- **Tècnica utilitzada**: Particions equivalents.
- Hem decidit provar:
  - Assignar un nom buit.
  - Assignar un nom de tipus no vàlid (número).
  - Assignar un nom amb només espais.

```
def test_set_invalid_name(self):
    with self.assertRaises(ValueError):  # Nombre vacío
        self.player.set_name("")
    with self.assertRaises(ValueError):  # Tipo inválido
        self.player.set_name(123)
    with self.assertRaises(ValueError):  # Nombre solo espacios
        self.player.set_name("   ")
```

**test_set_valid_score:** Prova assignar punts vàlids al jugador.

- **Tipus de test:** Caixa negra.
- **Tècnica utilitzada:** Particions equivalents i valors límit.
- Hem decidit provar:
  - Assignar el puntatge 0 (límit inferior vàlid).
  - Assignar un puntatge flotant positiu.

```
def test_set_valid_score(self):
    self.player.set_score(0)  # Límite inferior válido
    self.assertEqual(self.player.get_score(), 0)
    self.player.set_score(1000)  # Límite superior razonable
    self.assertEqual(self.player.get_score(), 1000)
    self.player.set_score(50.5)  # Puntaje como flotante válido
    self.assertEqual(self.player.get_score(), 50.5)
```

**test_set_invalid_score:** Verifica que es gestionen correctament els punts no vàlids.

- **Tipus de test:** Caixa negra.
- **Tècnica utilitzada:** Particions equivalents.
- Hem decidit provar:
  - Assignar un puntatge de tipus no vàlid (cadena).
  - Assignar un puntatge negatiu.

```
def test_set_invalid_score(self):
    with self.assertRaises(ValueError):  # Tipo inválido
        self.player.set_score("invalid_score")
    with self.assertRaises(ValueError):  # Valor negativo
        self.player.set_score(-1)
```

**test_get_name_after_invalid_set:** Prova el flux complet en gestionar un intent d'assignar un nom no vàlid.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.
- ➔ Hem decidit provar:
  - ◆ Intentar assignar un nom no vàlid.
  - ◆ Comprovar que el nom anterior es manté sense canvis.

```python
def test_get_name_after_invalid_set(self):
    with self.assertRaises(ValueError):
        self.player.set_name("")
    self.assertEqual(self.player.get_name(), "Alice")
```

```python
    def set_name(self, name):
        # Precondición: El nombre debe ser una cadena no vacía
        if not isinstance(name, str) or not name.strip():
            raise ValueError("El nombre debe ser una cadena no vacía.")

        # Asignación después de validar
        self._name = name

        # Postcondición: El nuevo nombre debe coincidir con el valor asignado
        assert self._name == name, "El nombre no fue correctamente asignado."

        # Invariantes
        self._check_invariants()
```

**test_set_id_combinations:** Prova combinacions vàlides i no vàlides per als ID.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Decision coverage.
- ➔ Hem decidit provar:
  - ◆ Assignar un ID vàlid.
  - ◆ Intentar assignar un ID no vàlid (cadena).

```python
def test_set_id_combinations(self):
    self.player.set_id(5)  # ID vàlid
    self.assertEqual(self.player.get_id(), 5)
    with self.assertRaises(ValueError):  # Tipus no vàlid
        self.player.set_id("invalid_id")
    with self.assertRaises(ValueError):  # Valor negatiu
        self.player.set_id(-100)
```

```python
    def set_id(self, player_id):
        # Precondición: El ID debe ser un entero no negativo
        if not isinstance(player_id, int) or player_id < 0:
            raise ValueError("Player ID debe ser un entero no negativo.")

        # Asignación después de validar
        self._id = player_id

        # Postcondición: El nuevo ID debe coincidir con el valor asignado
        assert self._id == player_id, "El ID no fue correctamente asignado."

        # Invariantes
        self._check_invariants()
```

**test_constructor_valid:** Prova la inicialització amb valors vàlids.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Hem decidit provar:
  - ◆ Crear un jugador amb ID, nom i puntuació vàlids.

```
def test_constructor_valid(self):
    player = Player(0, "Bob", 0)  # Valores válidos mínimos
    self.assertEqual(player.get_id(), 0)
    self.assertEqual(player.get_name(), "Bob")
    self.assertEqual(player.get_score(), 0)

    player = Player(9999, "Charlie", 100.5)  # Valores válidos altos
    self.assertEqual(player.get_id(), 9999)
    self.assertEqual(player.get_name(), "Charlie")
    self.assertEqual(player.get_score(), 100.5)
```

**test_constructor_invalid:** Prova la inicialització amb valors no vàlids.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Hem decidit provar:
  - ◆ Assignar un ID no vàlid.
  - ◆ Assignar un nom no vàlid.
  - ◆ Assignar un puntatge no vàlid.

```
def test_constructor_invalid(self):
    with self.assertRaises(ValueError):  # ID inválido
        Player(-1, "Alice", 10)
    with self.assertRaises(ValueError):  # Nombre inválido
        Player(1, "", 10)
    with self.assertRaises(ValueError):  # Puntaje inválido
        Player(1, "Alice", -10)
```

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## Controller/gameController.py i test_GameController.py

**Funcionalitat:** Implementació del controlador principal GameController per gestionar la lògica del joc Buscamines. Inclou funcionalitats com mostrar el menú principal, gestionar la inicialització del joc, revelar caselles, calcular punts, guardar els resultats en una base de dades i mostrar els rankings.

**Localització:**

- **Arxiu:** gameController.py
- **Classe:** gameController
- **Mètodes desenvolupats:**
  - __init__: Inicialitza la vista i la connexió amb la base de dades.
  - run: Controla el flux principal del menú.
  - start_game: Configura el jugador i el tauler segons la dificultat seleccionada.
  - play_game: Gestiona el flux del joc fins a la victòria o derrota.
  - save_player_to_rankings: Desa els resultats del jugador en la base de dades.
  - show_rankings: Mostra els millors jugadors des de la base de dades.

**Test:**

- **Arxiu:** test_gameController.py
- **Classe:** TestGameControllerTDD

**Mètodes de test associats:**

**test_initialize_game_controller:** Verifica la correcta inicialització del controlador, incloent la vista i la base de dades.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ **Casos coberts:**
  - ◆ Inicialització amb connexió vàlida a la base de dades.
  - ◆ Comprovació que la vista està inicialitzada correctament.

```
def test_initialize_game_controller(self):
    self.assertIsInstance(self.controller.view, View)
    self.assertIsInstance(self.controller.database, DatabaseManager)
```

**test_show_menu:** Simula la selecció d'una opció del menú principal i verifica que el resultat coincideix amb l'entrada.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Mocking i particions equivalents.
- ➔ Casos coberts:
  - ◆ Entrada vàlida (opció "1").
  - ◆ Entrada no vàlida (opció no existent).

```python
def test_show_menu(self):
    with unittest.mock.patch('builtins.input', return_value="1"):
        option = self.controller.view.display_menu()
        self.assertEqual(option, "1")
```

**test_start_game_initialization:** Verifica que el joc es configura correctament segons la dificultat seleccionada.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.
- ➔ Casos coberts:
  - ◆ Selecció de dificultat fàcil (6x6).
  - ◆ Inicialització correcta del tauler i el jugador.

```python
def test_start_game_initialization(self):
    with unittest.mock.patch('builtins.input', side_effect=["TestPlayer", "1"]):
        player_name = self.controller.view.get_player_name()
        difficulty = self.controller.view.display_difficulty_menu()
        board = Board(difficulty)

        self.assertEqual(player_name, "TestPlayer")
        self.assertEqual(difficulty, 1)
        self.assertEqual(board.size, 6)  # Dificultad 1 equivale a 6x6
```

```python
    def start_game(self):
        # Precondición: La vista debe estar inicializada para recibir datos del jugador.
        assert self.view is not None, "La vista no está inicializada."
        player_name = self.view.get_player_name()
        difficulty = self.view.display_difficulty_menu()

        try:
            # Precondición: La dificultad debe ser válida (1, 2 o 3).
            board = Board(difficulty)
            player = Player(0, player_name, 0)

            # Postcondición: El jugador y el tablero deben estar inicializados correctamente.
            assert board is not None, "El tablero no se inicializó correctamente."
            assert player is not None, "El jugador no se inicializó correctamente."

            self.play game(board, player)
```

**test_reveal_bomb_tile:** Verifica que el joc es perd quan es revela una casella amb bomba.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Decision coverage.
- ➔ Casos coberts:
  - ◆ Revelar una casella amb bomba.

```python
def test_reveal_bomb_tile(self):
    board = Board(1)
    player = Player(0, "TestPlayer", 0)
    x, y = 0, 0
    board.tiles[x][y].is_bomb = True  # Simulamos una bomba en (0, 0)

    board.reveal_tile(x, y)
    self.assertTrue(board.is_game_lost)
    self.assertEqual(player.get_score(), 0)
```

```python
    def play_game(self, board, player):
        # Precondiciones
        assert board is not None, "El tablero no está inicializado."
        assert player is not None, "El jugador no está inicializado."

        while not board.is_game_won() and not board.is_game_lost:
            self.view.display_board(board)

            try:
                x, y = self.view.get_coordinates(board.size)

                # Precondición: Las coordenadas deben estar dentro del rango del tablero.
                assert 0 <= x < board.size, f"La coordenada x={x} está fuera de rango."
                assert 0 <= y < board.size, f"La coordenada y={y} está fuera de rango."

                # Revela la casilla seleccionada
                is_bomb = board.tiles[x][y].is_bomb
                successfully_revealed = board.reveal_tile(x, y)

                # Postcondición: La casilla debe estar revelada.
                if successfully_revealed:
                    assert board.tiles[x][y].is_revealed, "La casilla no se reveló correctamente."

                    # Suma puntos si se descubre una casilla sin bomba
                    if not is_bomb:
                        player.set_score(player.get_score() + 10)
                        self.view.display_message("¡Has revelado una casilla segura! +10 puntos.")
```

**test_add_score_on_bomb:** Comprova que es suma puntació en revelar una casella segura.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Casos coberts:
  - ◆ Sumar 10 punts quan es revela una casella segura.

```
def test_add_score_on_bomb(self):
    board = Board(1)
    player = Player(0, "TestPlayer", 0)
    x, y = 0, 0
    board.tiles[x][y].is_bomb = True

    board.reveal_tile(x, y)
    player.set_score(player.get_score() + 10)
    self.assertEqual(player.get_score(), 10)
```

**test_game_won:** Simula revelar totes les caselles no bomba i verifica que es declara la victòria.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Loop testing.
- ➔ Casos coberts:
  - ◆ Revelar totes les caselles no bomba.

```
def test_game_won(self):
    board = Board(1)
    player = Player(0, "TestPlayer", 0)

    # Revelamos todas las casillas no bomba
    for row in board.tiles:
        for tile in row:
            if not tile.is_bomb:
                tile.reveal()

    self.assertTrue(board.is_game_won())
```

```
79          self.view.display_board(board)
80
81          if board.is_game_lost:
82              self.view.display_message(f"¡Boom! Has perdido. Puntos totales: {player.get_score()}")
83              self.save_player_to_rankings(player)
84          else:
85              # Añade puntaje adicional al ganar
86              player.set_score(player.get_score() + (100 - board.total_bombs))
87              self.view.display_message(f"¡Felicidades! Has ganado con {player.get_score()} puntos.")
88              self.save_player_to_rankings(player)
```

**test_game_lost**: Verifica que el joc es declara perdut quan es revela una bomba.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Decision coverage.
- ➔ Casos coberts:
  - ◆ Revelar una bomba en qualsevol moment del joc.

```
def test_game_lost(self):
    board = Board(1)
    player = Player(0, "TestPlayer", 0)
    x, y = 0, 0
    board.tiles[x][y].is_bomb = True

    board.reveal_tile(x, y)
    self.assertTrue(board.is_game_lost)
```

**test_rankings_no_data:** Verifica que els rankings estan buits si no hi ha jugadors a la base de dades.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Casos coberts:
  - ◆ Base de dades buida (cap jugador).

```
def test_rankings_no_data(self):
    rankings = self.controller.database.get_top_players()
    self.assertEqual(len(rankings), 0)
```

**test_insert_and_retrieve_rankings**: Inserta un jugador en els rankings i verifica que es recupera correctament.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Path coverage.
- ➔ Casos coberts:
  - ◆ Inserir un jugador.
  - ◆ Recuperar el jugador des de la base de dades.

```
def test_insert_and_retrieve_rankings(self):
    self.controller.database.insert_player("TestPlayer", 50)
    rankings = self.controller.database.get_top_players()
    self.assertEqual(len(rankings), 1)
    self.assertEqual(rankings[0][1], "TestPlayer")
    self.assertEqual(rankings[0][2], 50)
```

```
90      def save_player_to_rankings(self, player):
91          # Precondición: El jugador debe tener un nombre y un puntaje válido.
92          assert player.get_name(), "El jugador no tiene un nombre válido."
93          assert player.get_score() >= 0, "El puntaje del jugador no es válido."
94
95          success = self.database.insert_player(player.get_name(), player.get_score())
96
97          # Postcondición: La operación debe ser exitosa o manejar el error adecuadamente.
98          if success:
99              self.view.display_message("¡Tu puntaje se ha guardado correctamente!")
```

**test_show_rankings_with_data**: Comprova que es poden recuperar i mostrar rankings amb dades.

- ➔ **Tipus de test:** Caixa blanca.
- ➔ **Tècnica utilitzada:** Statement coverage.
- ➔ Casos coberts:
  - ◆ Recuperar rankings amb múltiples jugadors.

```
def test_show_rankings_with_data(self):
    self.controller.database.insert_player("Player1", 100)
    self.controller.database.insert_player("Player2", 80)
    rankings = self.controller.database.get_top_players()

    self.assertEqual(len(rankings), 2)
    self.assertEqual(rankings[0][1], "Player1")
    self.assertEqual(rankings[1][1], "Player2")
```

**test_show_rankings_message_no_data**: Verifica que es mostra el missatge adequat quan no hi ha dades als rankings.

- ➔ **Tipus de test:** Caixa negra.
- ➔ **Tècnica utilitzada:** Mocking.
- ➔ Casos coberts:
  - ◆ Base de dades sense dades de rankings.

```
def test_show_rankings_message_no_data(self):
    with unittest.mock.patch('builtins.print') as mock_print:
        self.controller.show_rankings()
        mock_print.assert_called_once_with("No hay rankings disponibles. Juega una partida para aparecer aquí.")
```

**test_multiple_games**: Simula múltiples partides i comprova que els resultats es desen correctament.

- ➔ **Tipus de test:** Caixa blanca.

- ➔ **Tècnica utilitzada:** Loop testing.
- ➔ Casos coberts:
  - ◆ Inserir múltiples jugadors amb diferents punts.
  - ◆ Recuperar els rankings.

```
def test_multiple_games(self):
    for i in range(5):
        self.controller.database.insert_player(f"Player{i}", i * 10)
    rankings = self.controller.database.get_top_players()
    self.assertEqual(len(rankings), 5)
```

## test_Mocks_gameController.py

**Mocks y su función en las pruebas**

**mock_view**: (Simulación de la clase View)

- ➔ **Uso**: Simula la interacción del usuario con la interfaz.
- ➔ **Métodos simulados:**
  - ◆ **display_menu:** Simula la selección de opciones del menú principal.
    - **Casos probados:** Selección de las opciones "1" (Iniciar juego) y "3" (Salir del juego).
  - ◆ **get_player_name:** Devuelve un nombre de jugador simulado, como "TestPlayer".
    - **Casos probados:** Introducir un nombre válido para el jugador.
  - ◆ **display_difficulty_menu:** Simula la selección de la dificultad del juego.
    - **Casos probados:** Selección de la dificultad 1 (Fácil).
  - ◆ **get_coordinates:** Devuelve coordenadas simuladas para jugar.
    - **Casos probados:** Coordenadas válidas como (0, 0) y (1, 1).
  - ◆ **display_message:** Captura mensajes mostrados al usuario.
    - **Casos probados:** Mensajes de victoria, derrota y rankings vacíos.

**mock_board** (Simulación de la clase Board)

- ➔ **Uso**: Simula el comportamiento del tablero de juego.
- ➔ **Atributos y métodos simulados:**
  - ◆ **size**: Tamaño del tablero (ejemplo: 6x6).
    - **Casos probados:** Tamaño del tablero según la dificultad seleccionada.
  - ◆ **tiles**: Matriz de casillas simuladas, configuradas con o sin bombas.
    - **Casos probados:** Tablero con bombas (is_bomb=True) o sin bombas (is_bomb=False).

- **is_game_won:** Indica si el juego ha sido ganado.
  - **Casos probados:** Flujo en el que el jugador gana después de revelar todas las casillas seguras.
- **is_game_lost:** Indica si el juego ha sido perdido.
  - **Casos probados:** Flujo en el que el jugador pierde al revelar una bomba.
- **reveal_tile:** Revela una casilla del tablero.
  - **Casos probados:** Revelar casillas seguras y bombas.

**mock_database_manager** (Simulación de la clase DatabaseManager)

➔ **Uso**: Simula la interacción con la base de datos.
➔ **Métodos simulados:**
- **insert_player**: Simula la inserción de un jugador en los rankings.
  - **Casos probados:** Inserción exitosa de puntajes del jugador.
- **get_top_players**: Recupera rankings de la base de datos.
  - **Casos probados**: Rankings vacíos y rankings con datos.

**Detalles por Test**

**test_start_game**

➔ **Mocks utilizados:**
- mock_view: Simula el flujo de entrada de usuario para iniciar un juego y elegir una dificultad.
- mock_board: Configura un tablero vacío (sin bombas) y asegura que el flujo de juego se realiza correctamente.
- mock_database_manager: Comprueba que se inserta el puntaje correctamente al finalizar el juego.

```python
class TestGameController(unittest.TestCase):
    @patch("controller.gameController.View")
    @patch("controller.gameController.Board")
    @patch("controller.gameController.DatabaseManager")
    def test_start_game(self, mock_database_manager, mock_board, mock_view):
        mock_view.return_value.display_menu.side_effect = ["1", "3"]
        mock_view.return_value.get_player_name.return_value = "TestPlayer"
        mock_view.return_value.display_difficulty_menu.return_value = 1
        mock_view.return_value.get_coordinates.side_effect = [(0, 0), (1, 1)]
        mock_board.return_value.size = 6
        mock_board.return_value.tiles = [[MagicMock(is_bomb=False) for _ in range(6)] for _ in range(6)]
        mock_board.return_value.is_game_won.side_effect = [False, True]
        mock_board.return_value.is_game_lost = False
        mock_board.return_value.total_bombs = 0
        mock_database_manager.return_value.insert_player.return_value = True

        controller = GameController()
        controller.run()

        mock_database_manager.return_value.insert_player.assert_called_once_with("TestPlayer", 110)
```

**test_show_rankings_no_data**

- ➔ **Mocks utilizados:**
  - mock_view: Verifica que se muestra el mensaje correcto cuando no hay rankings disponibles.
  - mock_database_manager: Simula una base de datos sin datos.

**test_play_game_discover_bomb**

- ➔ **Mocks utilizados:**
  - mock_view: Controla la entrada de coordenadas del jugador.
  - mock_board: Configura un tablero con bombas y asegura que el juego detecte la derrota.
  - mock_database_manager: Verifica que se guarda el puntaje de 0 puntos tras perder el juego.

**test_play_game_win**

- ➔ **Mocks utilizados:**
  - mock_view: Simula el flujo de entrada de usuario, incluyendo las coordenadas que conducen a la victoria.
  - mock_board: Configura un tablero seguro (sin bombas) y asegura que el flujo de victoria se ejecuta correctamente.
  - mock_database_manager: Comprueba que se inserta el puntaje correcto al ganar el juego.

Cada test utiliza mocks estratégicamente para aislar el comportamiento del GameController y validar su lógica de manera precisa y controlada.

Àngela Obón Soto (1600359)
Muniba Liaqat Ali Ali (1635042)

## CONCLUSIONS

Gràcies a aquest projecte, hem après la importància de fer tests per garantir la qualitat del nostre codi. Els tests ens han fet més crítics i ens han ajudat a identificar errors que d'altra manera podrien haver passat desapercebuts. Especialment, en fer tests de coverage i caixa blanca, hem comprès millor el funcionament del nostre codi i hem detectat problemes en condicions i bucles que podrien haver trigat molt més a solucionar-se sense aquests tests.

Amb la metodologia TDD, hem desenvolupat funcionalitats com el rànking de manera estructurada, començant per escriure els tests i després adaptant el codi perquè passin. Això ens ha donat una visió clara del que necessitava el nostre programa i ha fet que el desenvolupament fos més eficient i organitzat. Hem trobat que TDD ens força a pensar millor en l'estructura del nostre codi abans d'escriure'l, assegurant que sigui més coherent i fàcil de mantenir.

Tot i això, hem vist que els tests inicials no sempre cobreixen tots els casos possibles. Per això, creiem que és important combinar TDD amb una revisió contínua durant el desenvolupament per afegir nous tests que abordin funcionalitats o errors que no havíem previst inicialment. Així, podem garantir que el programa sigui més robust i complet.