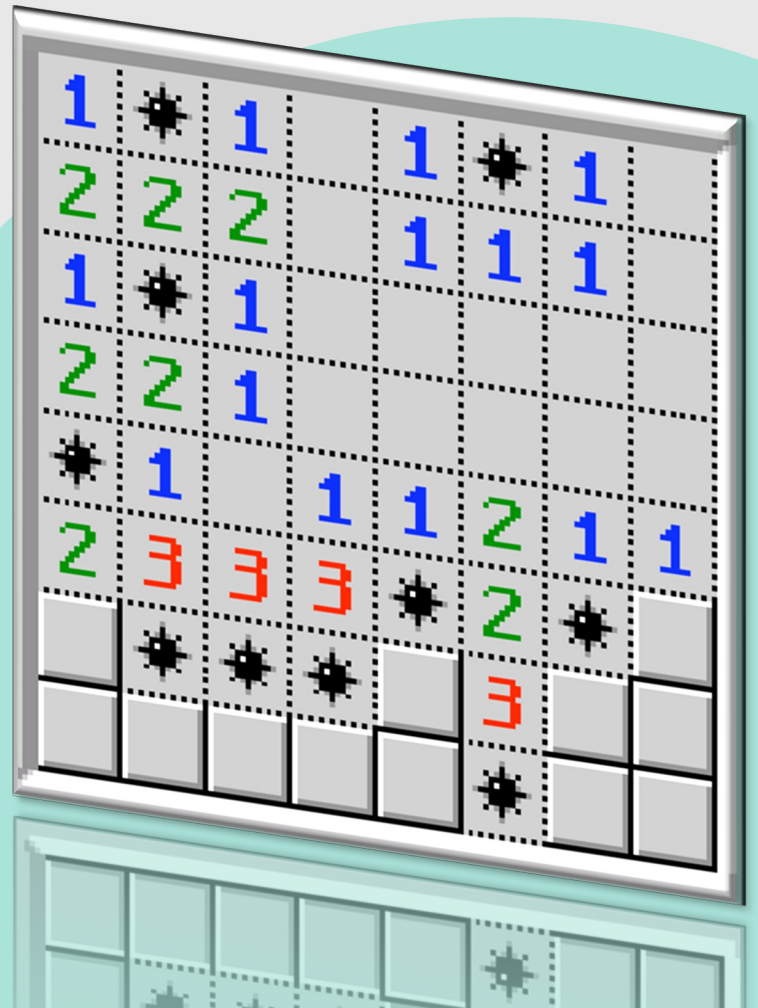
Minesweeper: a SDM project

The steps we took
and how we worked together

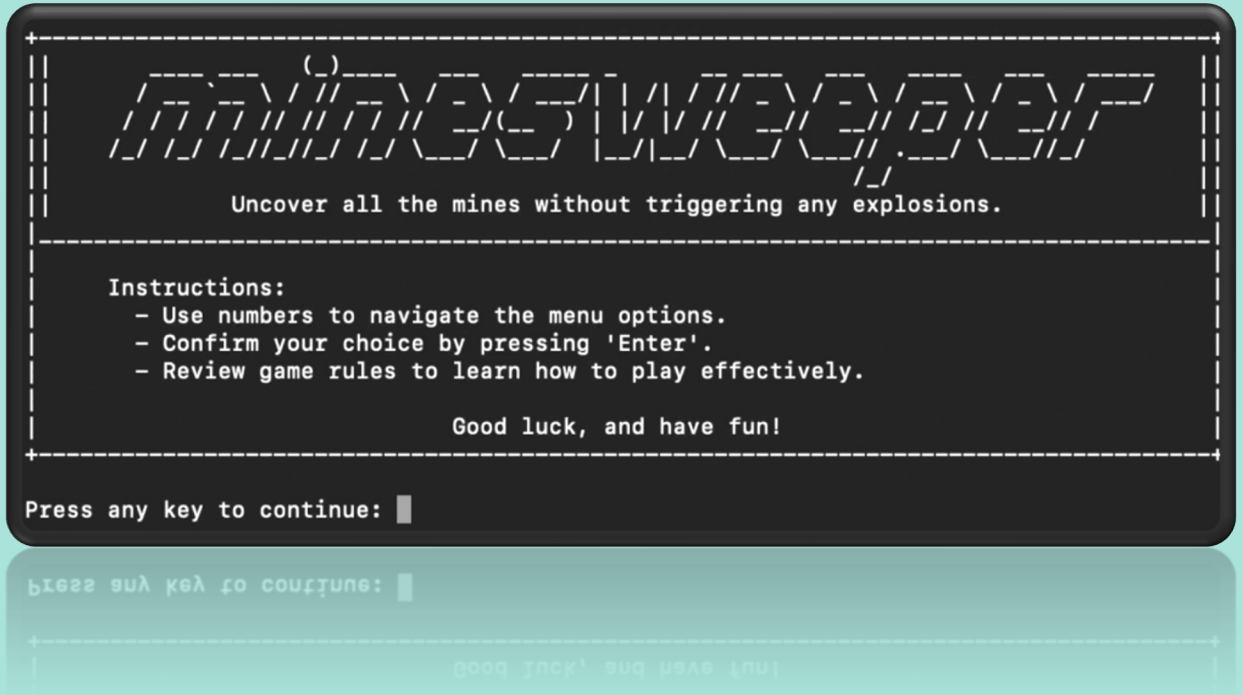


1. Introduction

a. Our team

1. Introduction

b. Minesweeper game



1. Introduction

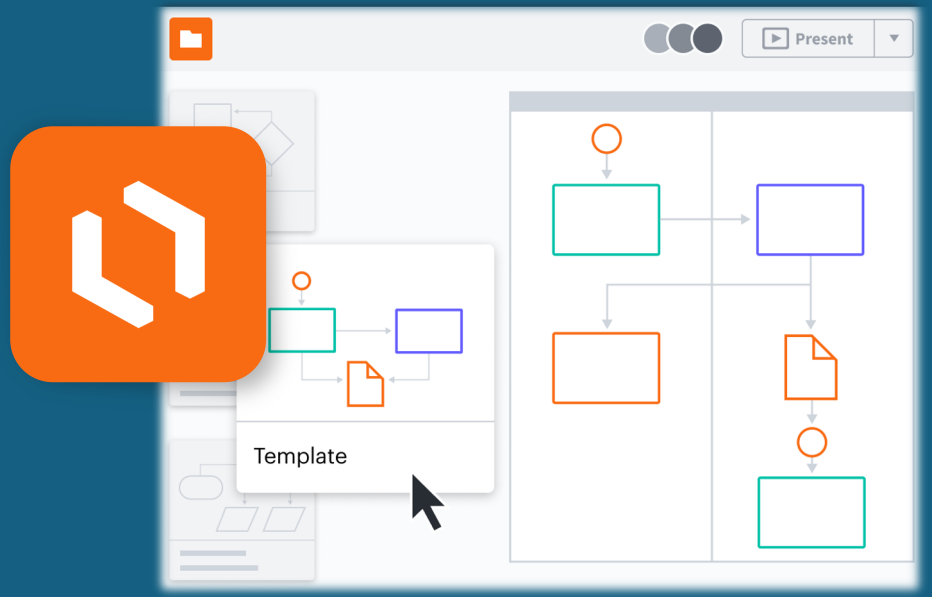
c. Goals of the project

- Agile Development
- Continuous Integration
- Automated tests
- Test Driven Development
- Working software

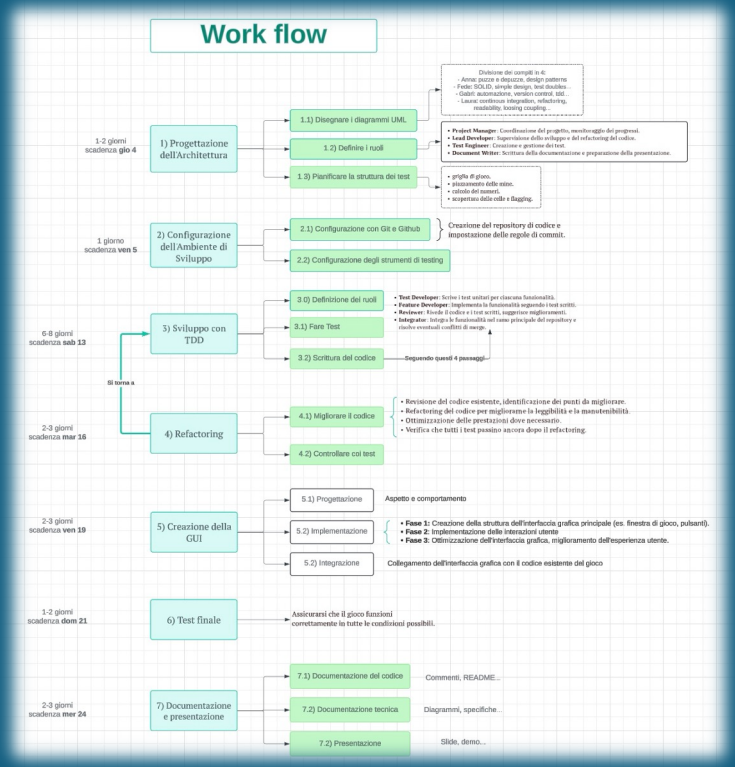
2. Project planning

a. The software we used

Lucidchart



2. Project planning

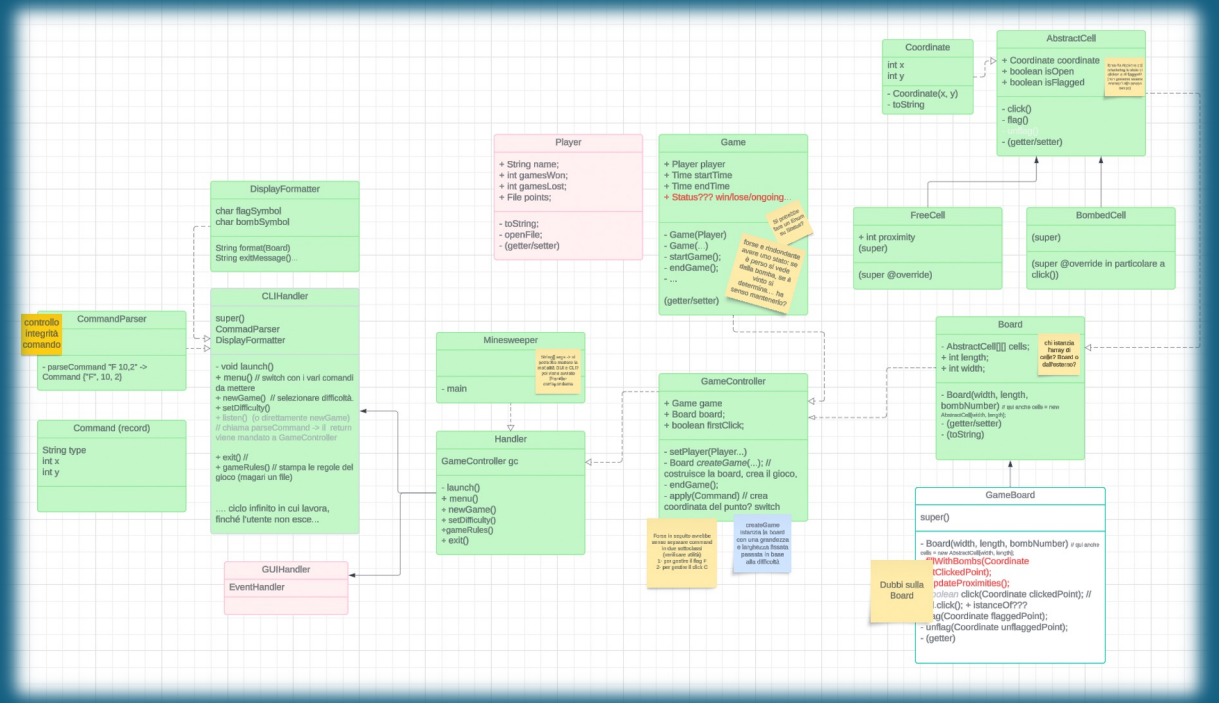


b. Defining the steps

1. Architecture design	2. Github Configuration
3. Test Driven Development	4. Refactoring
5. Final debug	6. Presentation and documentation

2. Project planning

c. Brainstorming a class diagram

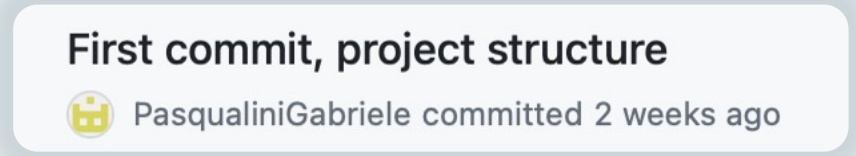


3. Environment setup

a. Tools overview

3. Environment setup

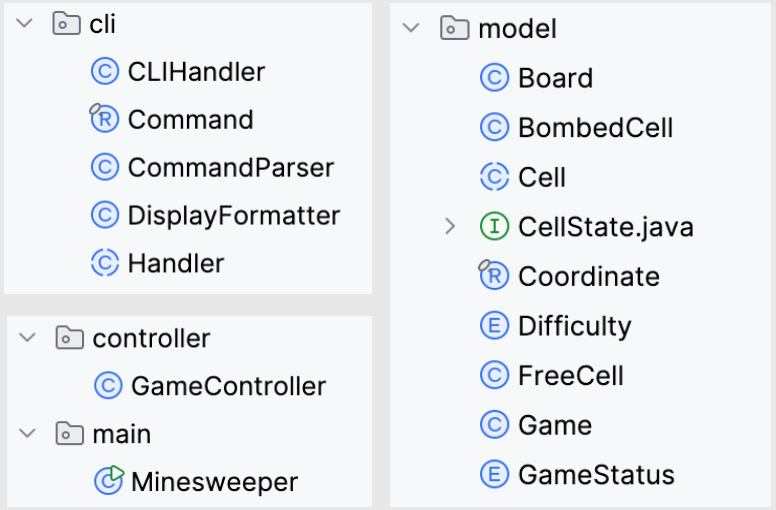
b. Repository



git | GitHub

3. Environment setup

c. First Commit

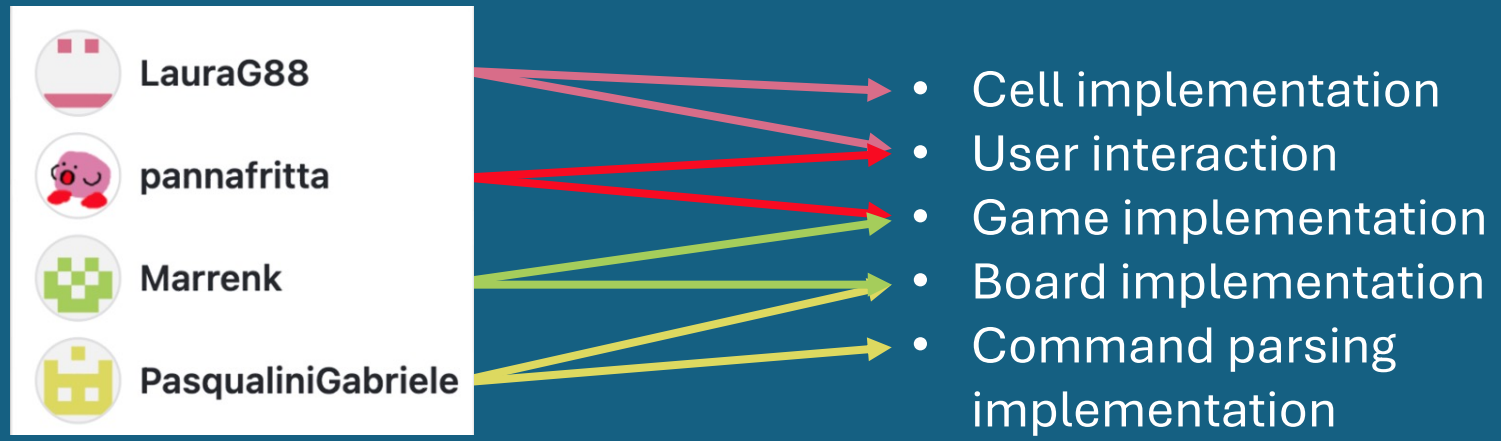


Gradle

```
12
13  dependencies {
14      testImplementation platform('org.junit:junit-bom:5.10.0')
15      testImplementation 'org.junit.jupiter:junit-jupiter'
16      testImplementation 'org.mockito:mockito-core:4.11.0'
17  }
18
19  test {
20      useJUnitPlatform()
21  }
```

4. Teamwork

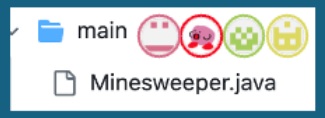
a. Initial distribution of responsibilities

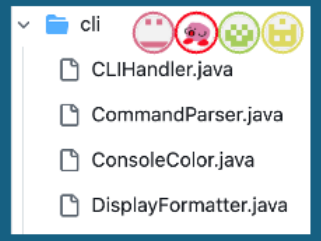


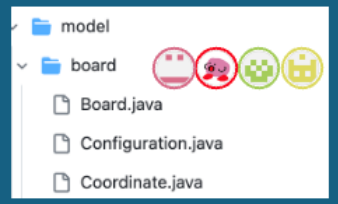
- Cell implementation
- User interaction
- Game implementation
- Board implementation
- Command parsing implementation

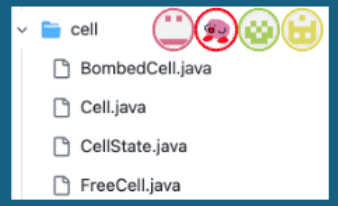
4. Teamwork

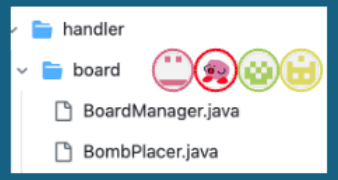
b. How it actually went:

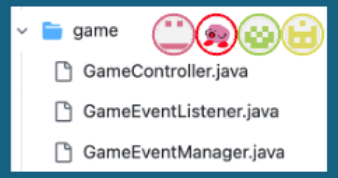










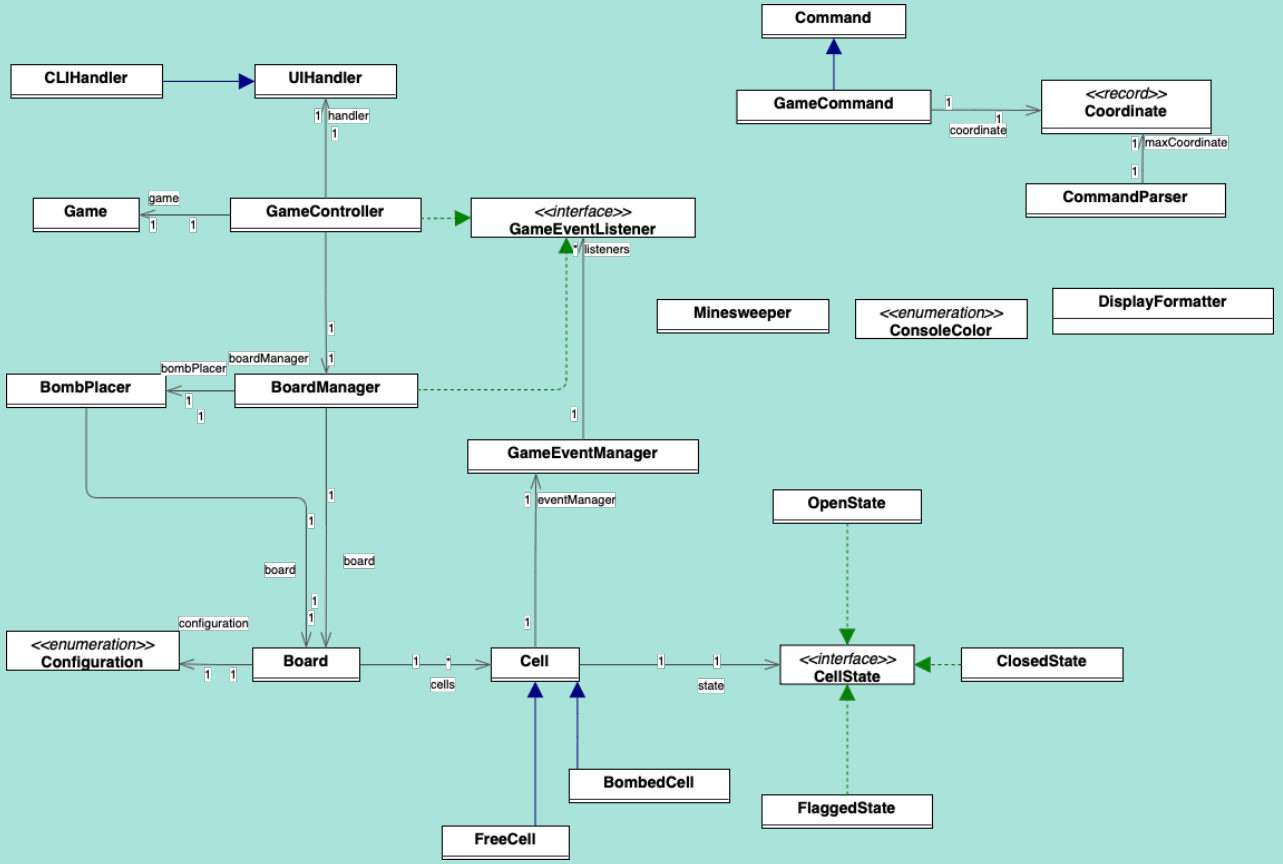


4. Teamwork

c. Our approach at continuous integration

5. Design choices

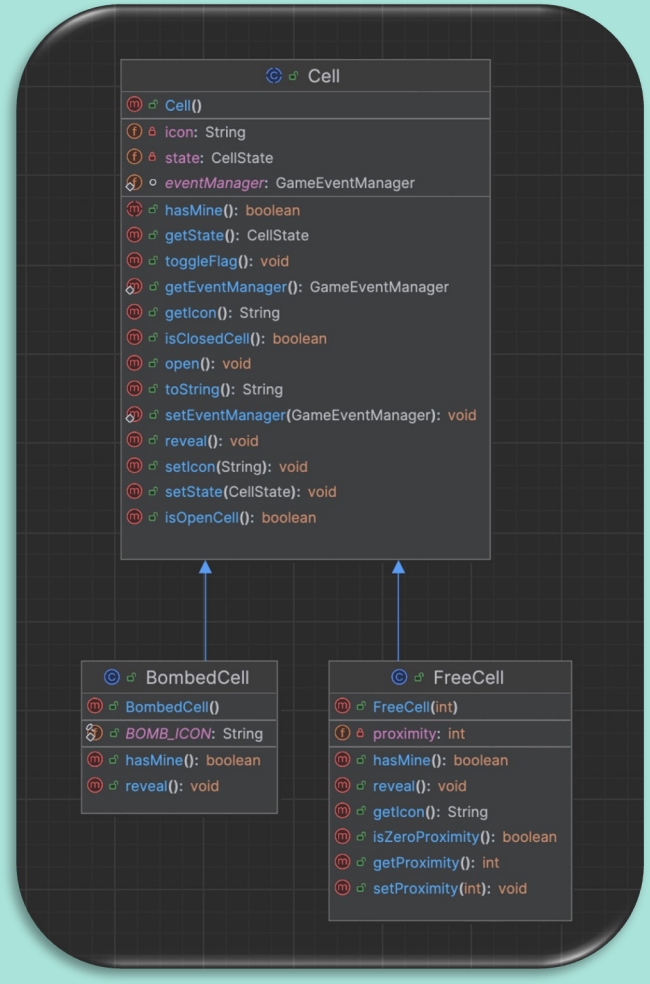
a. The UML diagram



5. Design choices

b. Cell design

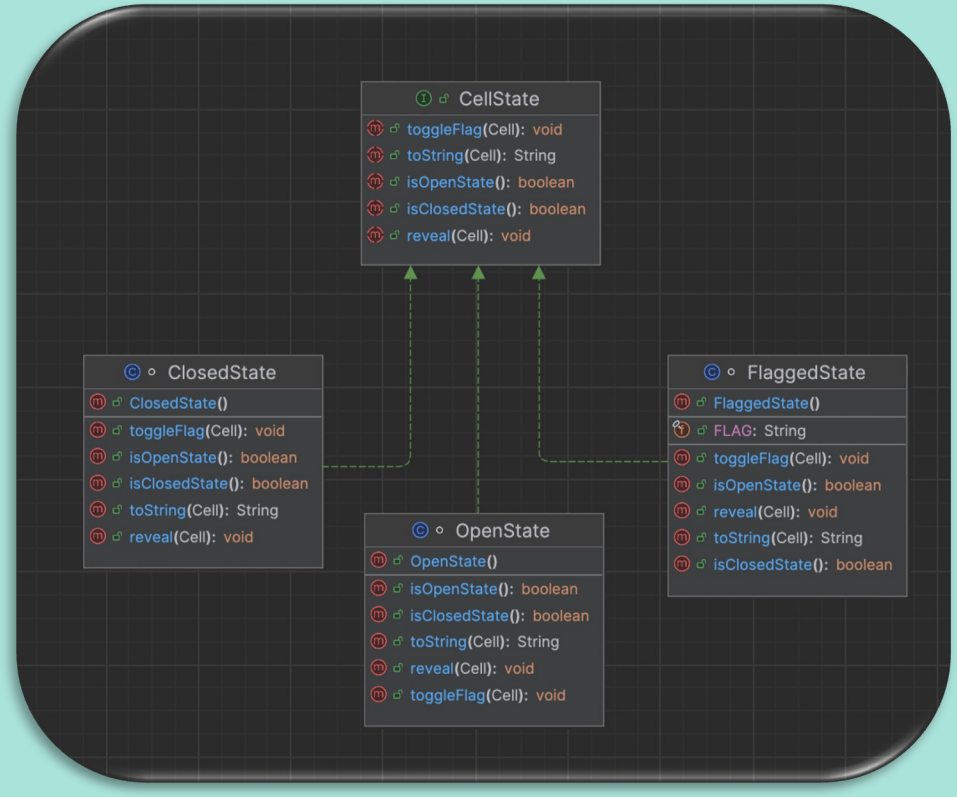
- Distinguishing cell behaviour through hierarchy



5. Design choices

b. Cell design

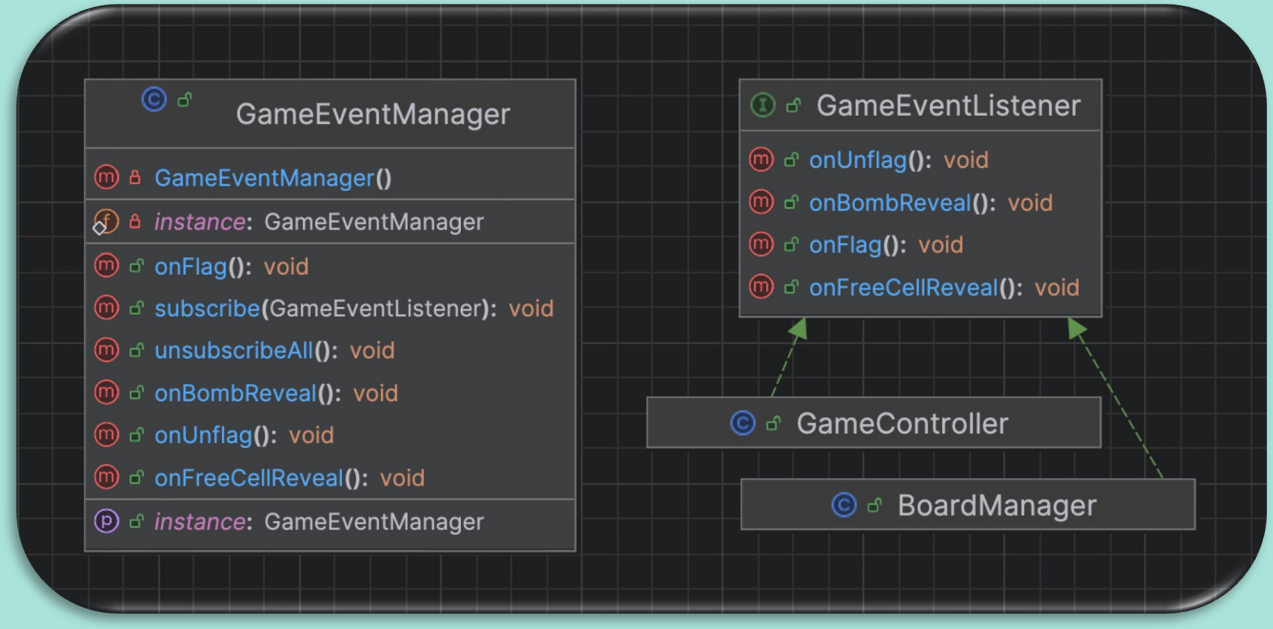
- Distinguishing cell behaviour through hierarchy
- Implemented state classes



5. Design choices

c. GameEventManager + GameEventListener

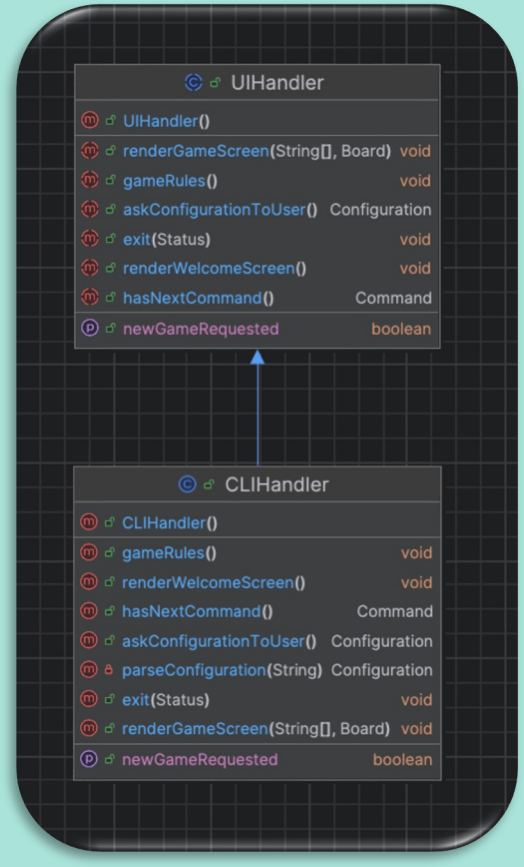
- Observer pattern
- Singleton pattern



5. Design choices

d. UIHandler

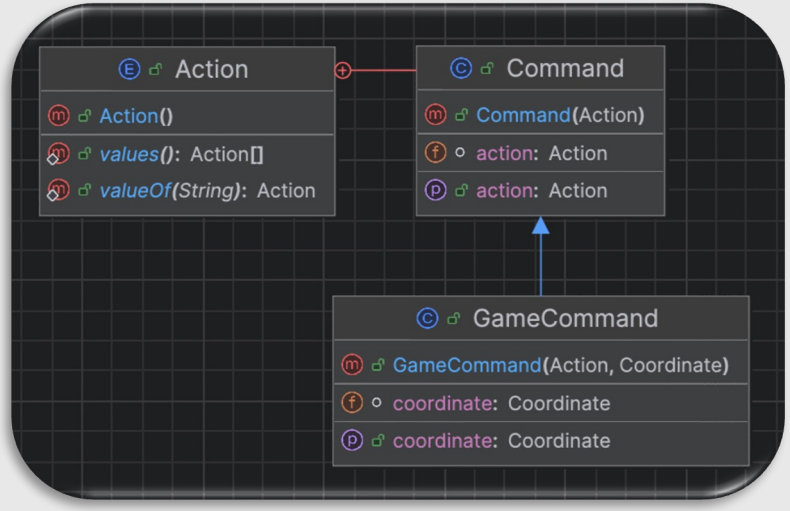
- For a future implementation of a GUI



6. Refactoring choices

a. Command

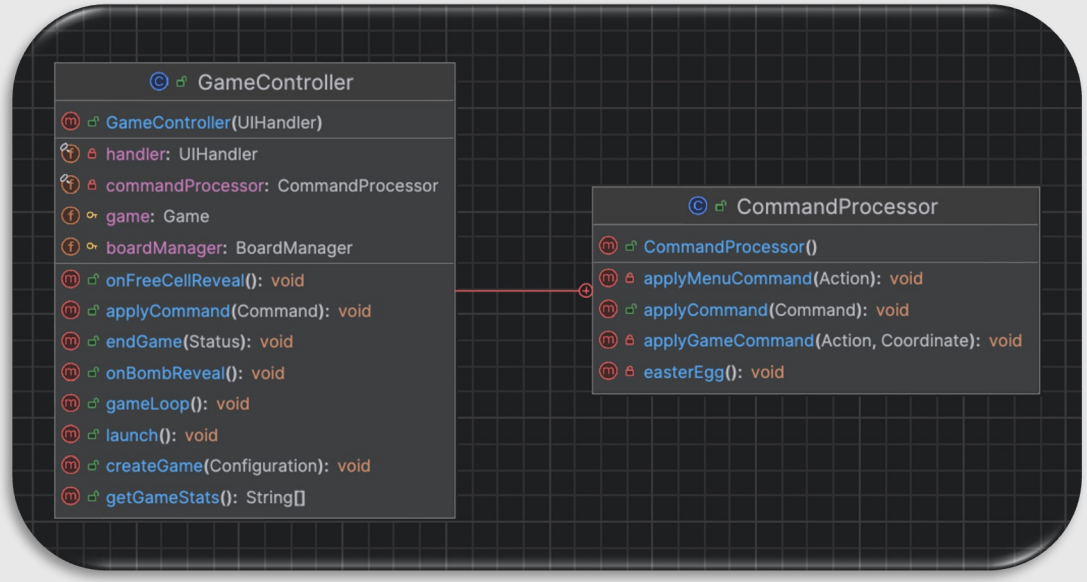
- Command: from record to class
- Subclass GameCommand
- Action: from String to Enum



6. Refactoring choices

b. GameController

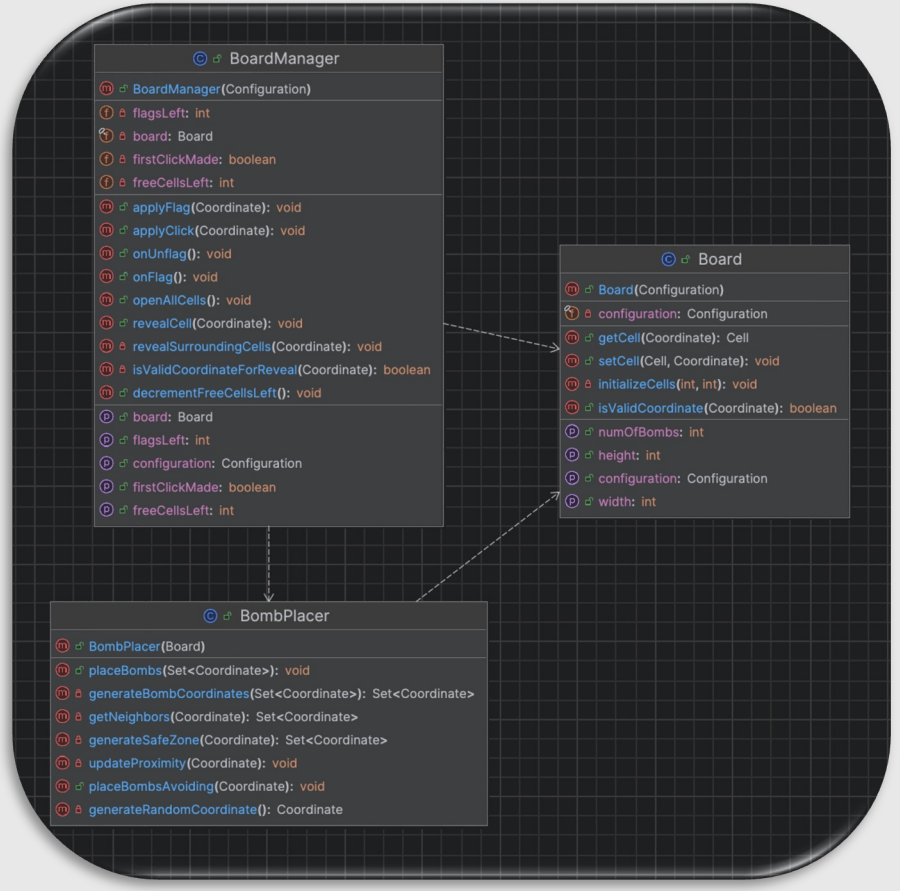
- Moving game loop
- Big class



6. Refactoring choices

c. Board

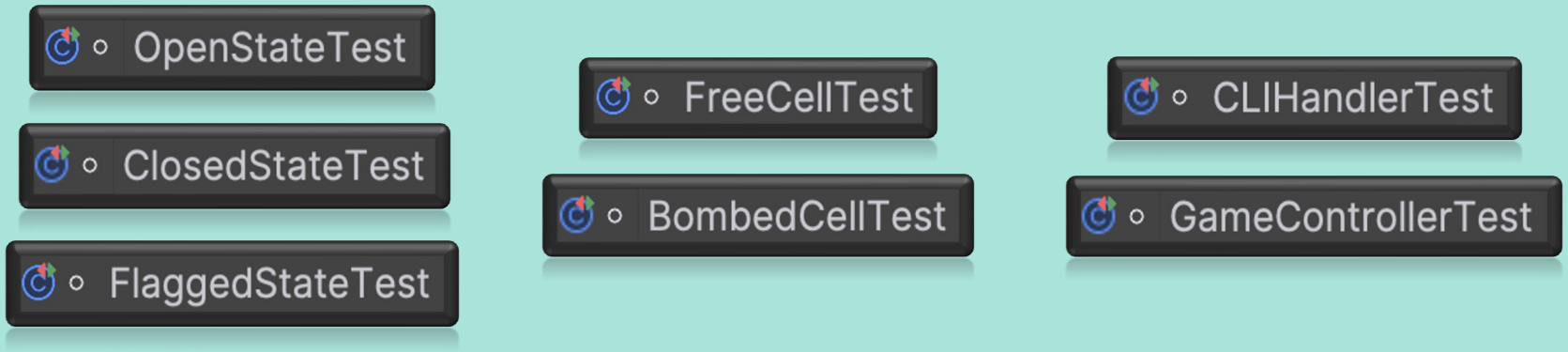
- From Board
  to BoardManager
- From BoardManager
  to BombPlacer



7. Testing

a. TDD

- Basic classes → fully tested
- Game Control Classes → partially tested
- User input → only a few tests



7. Testing

b. Unit tests


```
@Test  lauragallo
void testStateAfterReveal() {
    cell.reveal();
    assertInstanceOf(OpenState.class, cell.getState());
}
```

7. Testing

c. Service tests

- BoardManagerTest
- BombPlacerTest
- GameControllerTest

```java
@Test  Gabriele Pasqualini +1
void testFreeCellsLeft() {
    bombPlacer.placeBombsAvoiding(new Coordinate( x: 4,  y: 4));
    Command command = new GameCommand(CLICK_ACTION, new Coordinate( x: 4,  y: 4));
    gameController.createGame(Configuration.EASY);
    gameController.applyCommand(command);

    int expectedFreeCellsLeft = boardManager.getFreeCellsLeft();
    int actualFreeCellsLeft = 0;

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (board.getCell(new Coordinate(i, j)) instanceof FreeCell freeCell && freeCell.isClosedCell()) {
                actualFreeCellsLeft++;
            }
        }
    }
    assertEquals(expectedFreeCellsLeft, actualFreeCellsLeft);
}
```

8. Conclusion

a. Challenges

- github
- continuous integration
- gradle
- test
- lavoro di gruppo
- responsabilità delle classi

8. Conclusion

b. Our results

Element	Class, %	Method, %	Line, %	Branch, %
all	96% (26/27)	91% (141/154)	86% (313/360)	80% (109/135)
cli	100% (4/4)	76% (26/34)	72% (78/107)	77% (24/31)
handler	100% (11/11)	94% (55/58)	94% (138/146)	88% (59/67)
board	100% (2/2)	100% (23/23)	100% (68/68)	95% (40/42)
game	100% (5/5)	92% (25/27)	89% (57/64)	76% (19/25)
input	100% (4/4)	87% (7/8)	92% (13/14)	100% (0/0)
main	0% (0/1)	0% (0/1)	0% (0/2)	100% (0/0)
model	100% (11/11)	98% (60/61)	92% (97/105)	70% (26/37)
board	100% (3/3)	100% (16/16)	100% (28/28)	100% (12/12)
cell	100% (6/6)	97% (38/39)	88% (54/61)	52% (11/21)
game	100% (2/2)	100% (6/6)	93% (15/16)	75% (3/4)

8. Conclusion

b. Our results

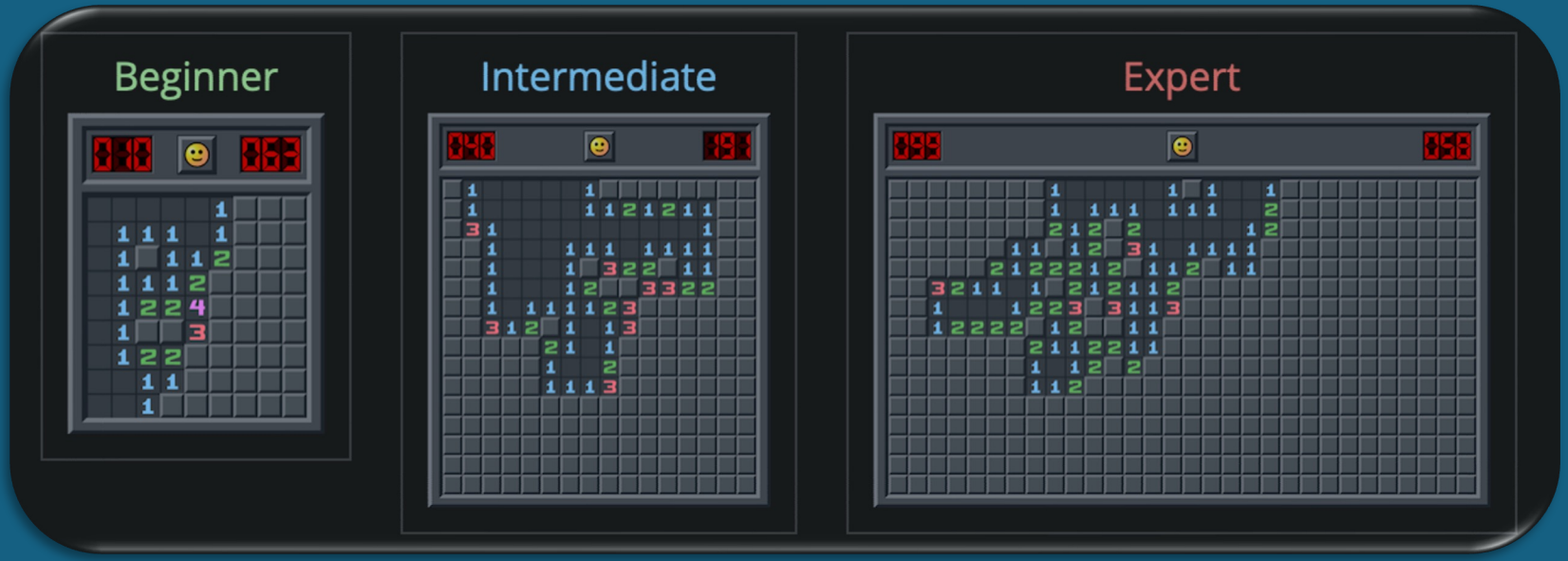
275 Commits

Packages

Package	Description
cli	This package contains classes for handling the command-line interface (CLI) for the Minesweeper game.
handler.board	This package contains classes that manage the board of the Minesweeper game, including placing bombs and handling board configurations.
handler.game	This package contains classes that manage the game flow and the events for the Minesweeper game.
handler.input	This package contains classes for handling user input and commands in the Minesweeper game.
main	This package contains the entry point for the Minesweeper game application.
model.board	This package contains classes representing the board and its configurations for the Minesweeper game.
model.cell	This package contains classes representing the cells and their various states.
model.game	This package contains classes representing the game state and its status for the Minesweeper game.

8. Conclusion

c. Future Implementations: GUI



9. Demo Live


Level: EASY Flags: 10 Time: 4
I Game Rules
Q Exit
Command: c 2 6


Level: EASY Flags: 9 Time: 18
I Game Rules
Q Exit
Command: f 5 4


Level: EASY Flags: 9 Time: 46
Want to play again? (y/n)


YOU LOST
YOU WON!