UNIVERSITÄT
DES
SAARLANDES

Programming 2 - SoSe 23

Project 4 - Ultimate Tic-Tac-Toe

Authors: Ben Sievers, Johannes Schöneberger

07. June 2023

Saarland University

Overview

1. Organisation
2. The Game
3. Project Details
   - Exercise 1 - Writing Tests
   - Exercise 2 - UTTT Implementation
   - Autoplayer

Organisation

Git Project Repository

You can get the project files with `git clone` using the following command:

```
$ git clone ssh://git@git.prog2.de:2222/project4/ $NAME.git $FOLDER
```

$NAME = Your username in the CMS / our course website
$FOLDER = The folder where you want to put the project

Timeline

The project is split into two parts:

1. Writing tests

   `Deadline: Tuesday, 13.06.2023, 23:59h`

2. Implementing the project specification

   `Deadline: Tuesday, 20.06.2023, 23:59h`

(Yes, that *does* mean you have to start earlier.)

The Game

Tic-Tac-Toe

1. Game for two players, where each player has a symbol (X, O)
2. Played on a 3x3 grid
3. The players take turns placing their symbol on the board
4. The first player with 3 marks in a row (also diagonally), wins the game

X	X	O
O	O	X
X	X	O

Figure 1:
Winner: tie

X	X	X
X	O	O
O	X	O

Figure 2:
Winner: X

O	O	X
X	O	X
X	X	O

Figure 3:
Winner: O

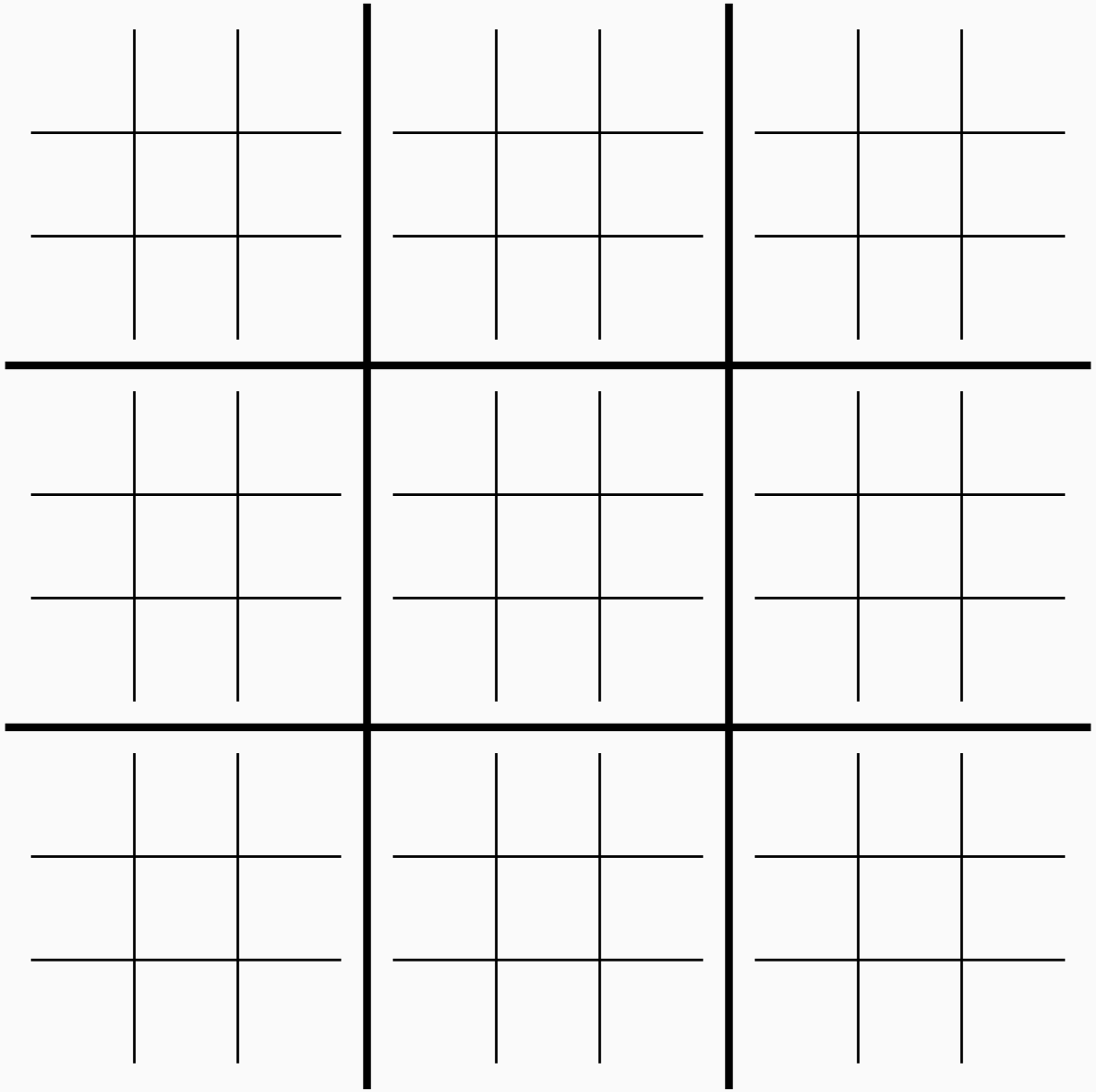
Ultimate Tic-Tac-Toe

X		O						
			X			X	X	X
	O				O			O
	O			O	X		O	
	O		O		X	O		
X		X		X			X	
		X						O

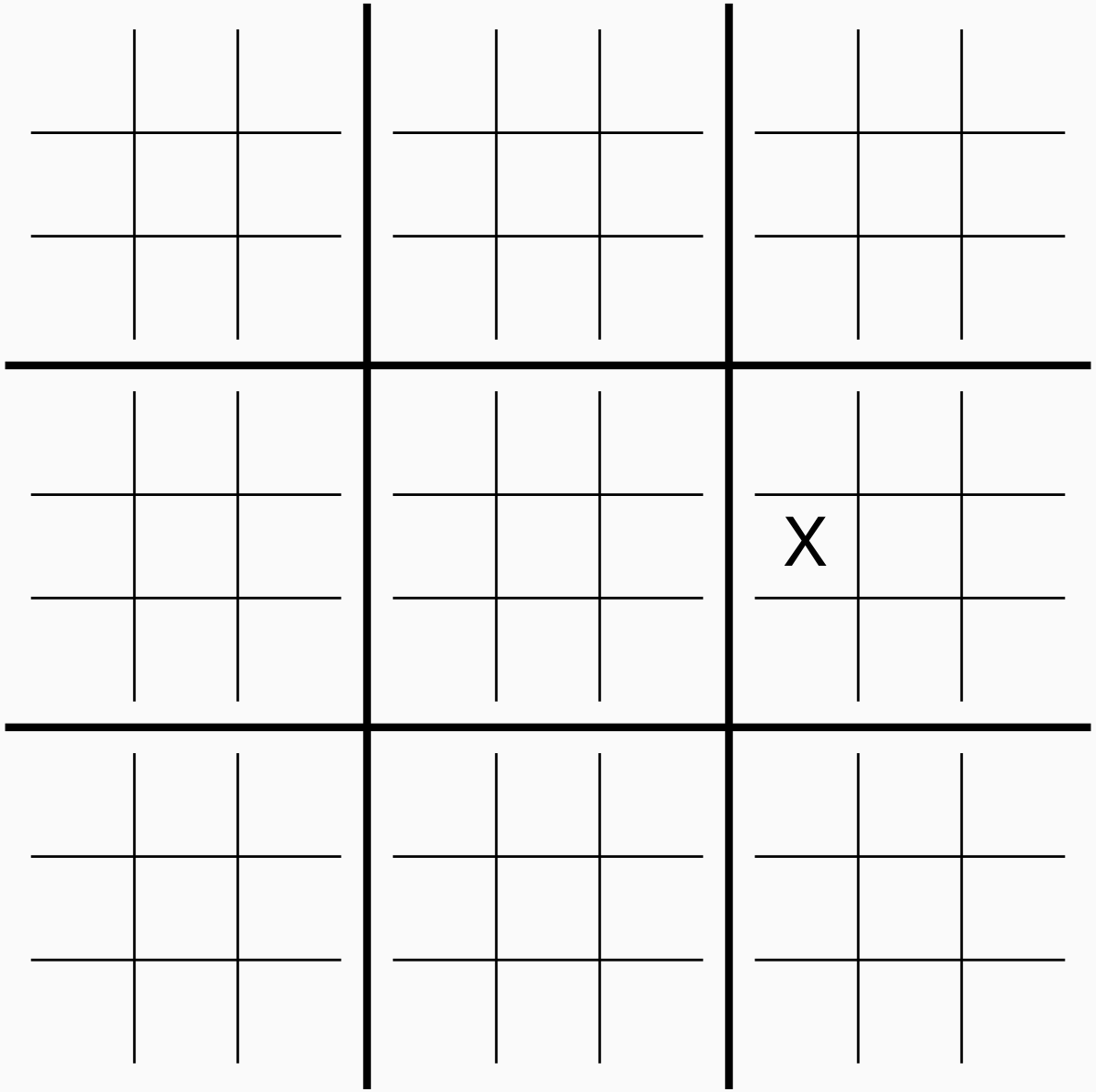
UTTT Rules

1. First player places mark on any board
2. Move of the next player is locked to the board, which position corresponds to the previous move on its board
3. This continues for future moves
4. If next board is won or full, player can choose board for next move
5. Winning a board is done like normal Tic-Tac-Toe
6. To win the game, a player has to win three boards in a row

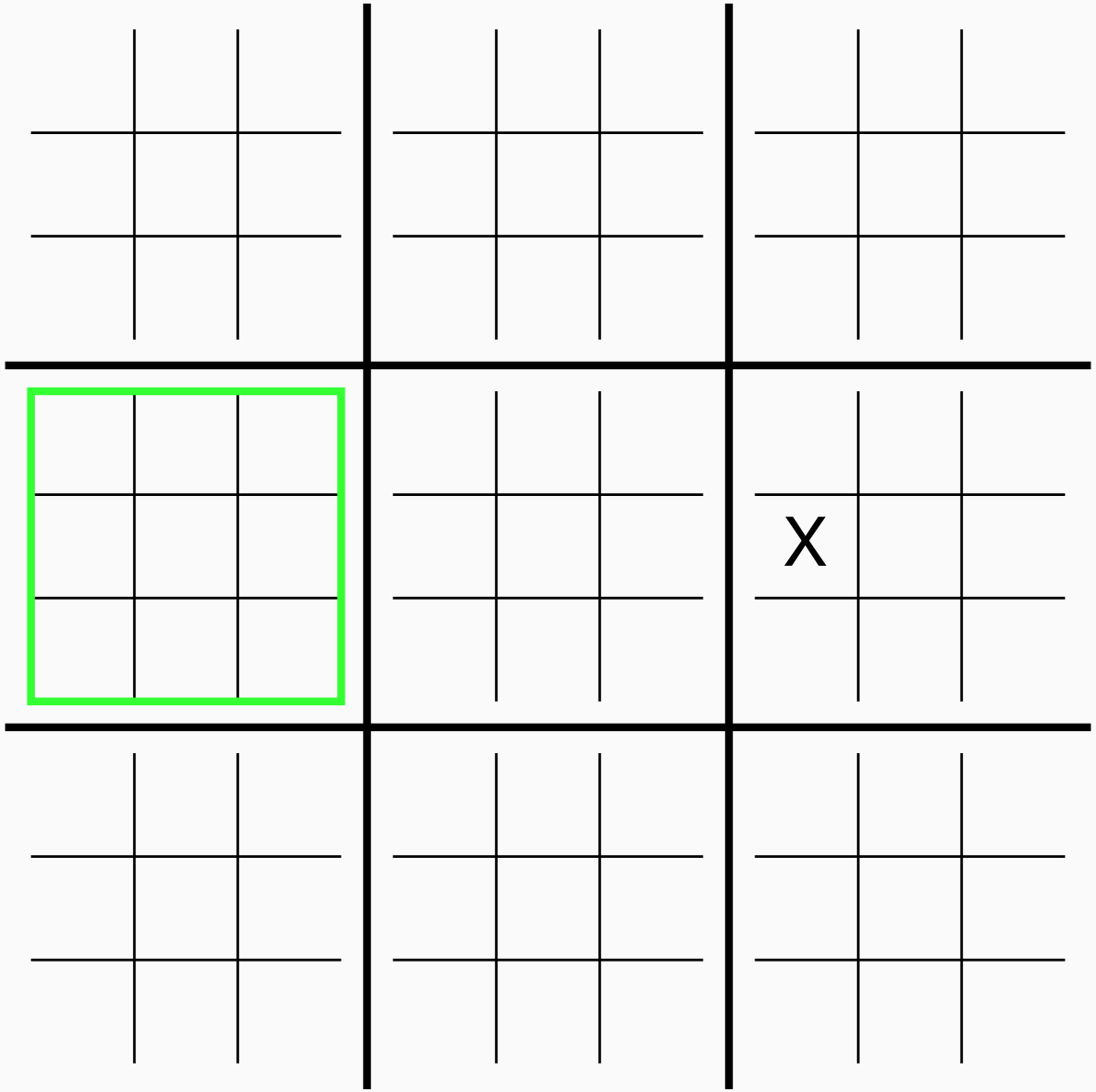
Example First Move



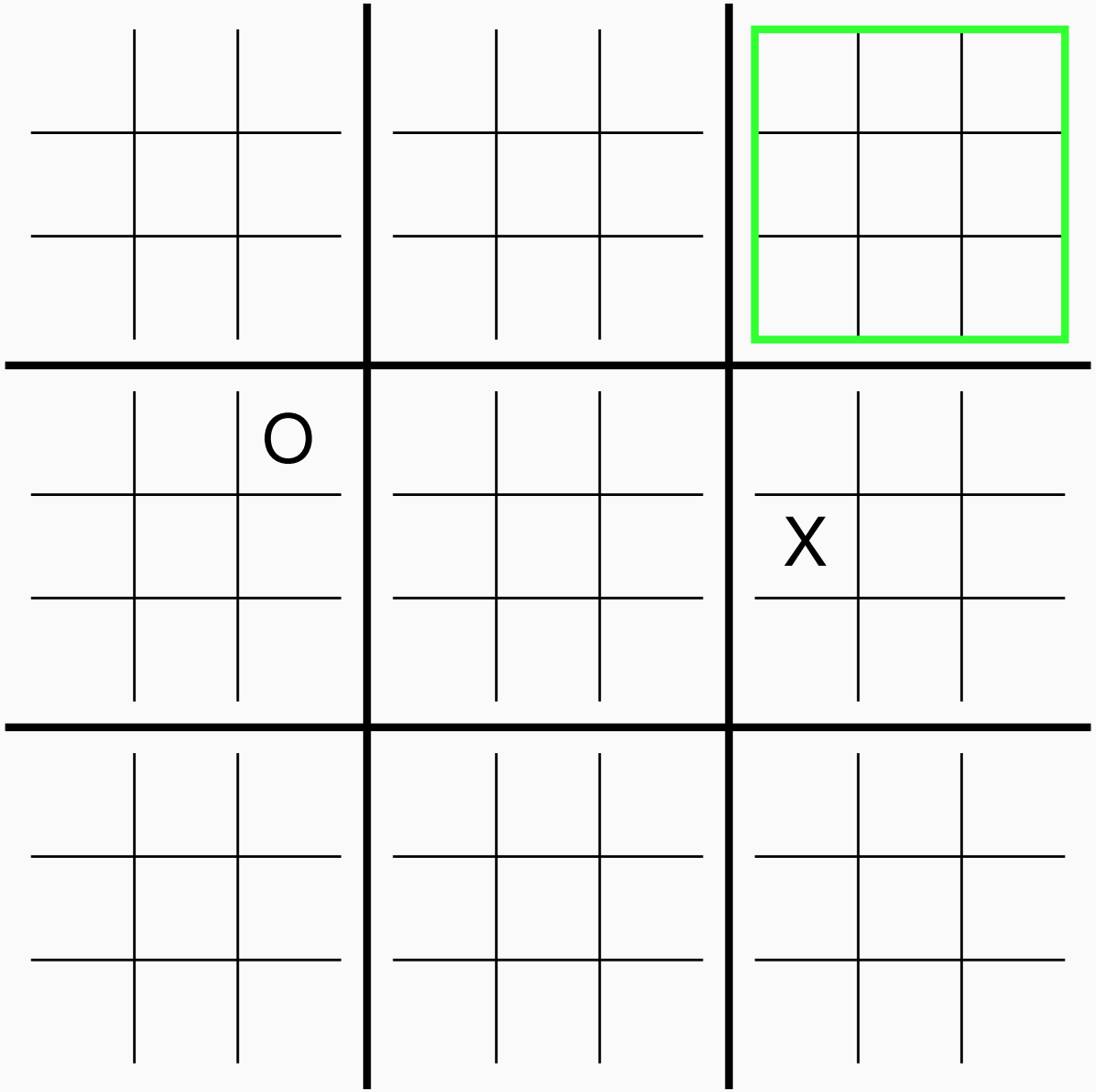
Example First Move



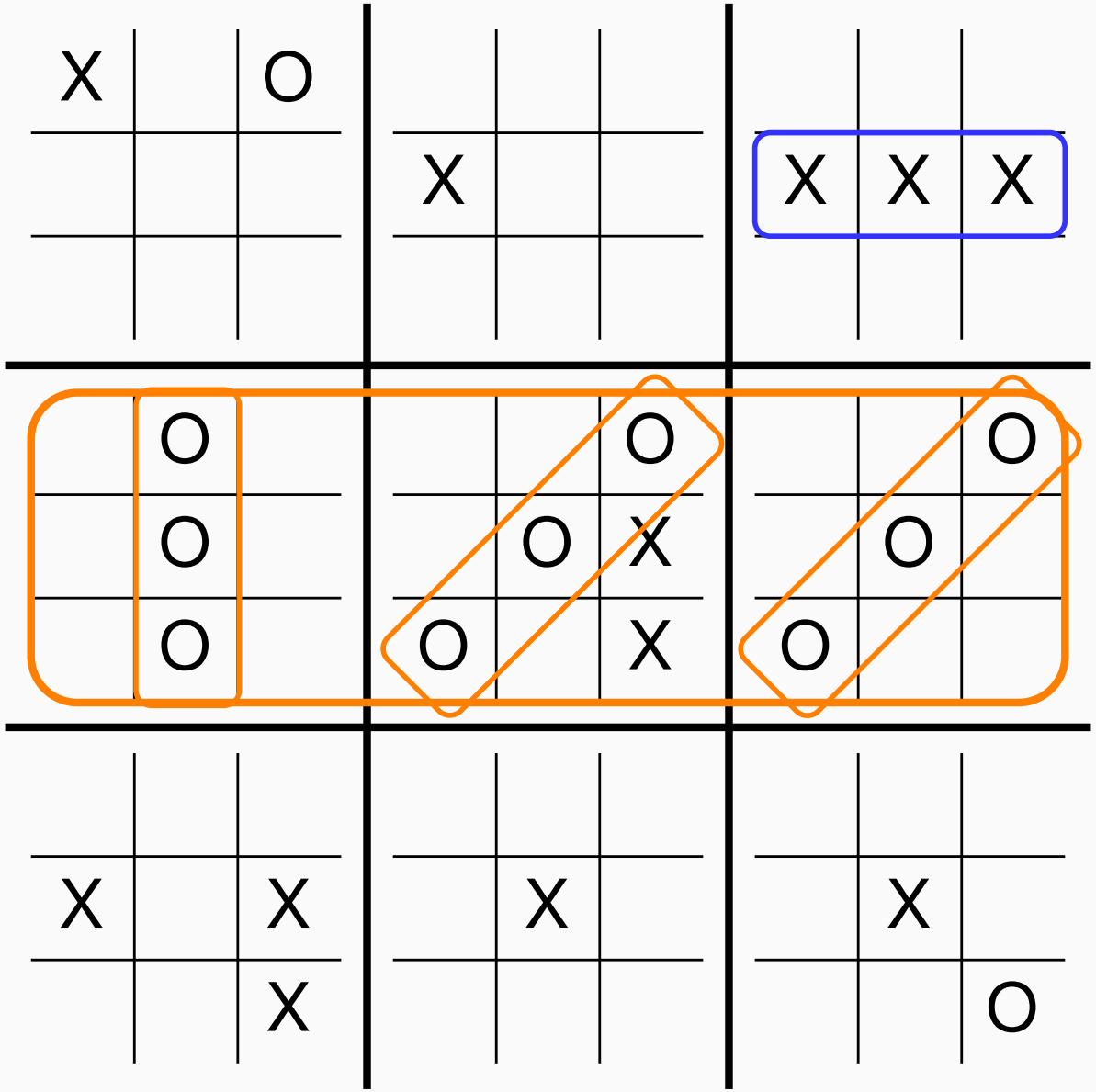
Example First Move



Example First Move



Ultimate Tic-Tac-Toe



Project Details

Project Details

Exercise 1 - Writing Tests

Exercise 1 - Writing Tests

1. **Your task:** Write tests for a game implementation
2. Such tests are used to check that an implementation works as expected
3. Tests can help you in the implementation phase later

```
Deadline:  Tuesday, 13.06.2023, 23:59h
```

Reference Implementations

On the test server:

- **Incorrect implementations:** At least one test must fail
- **Correct implementations:** No test may fail

$\hookrightarrow$ Black-Box-Tests

Hint

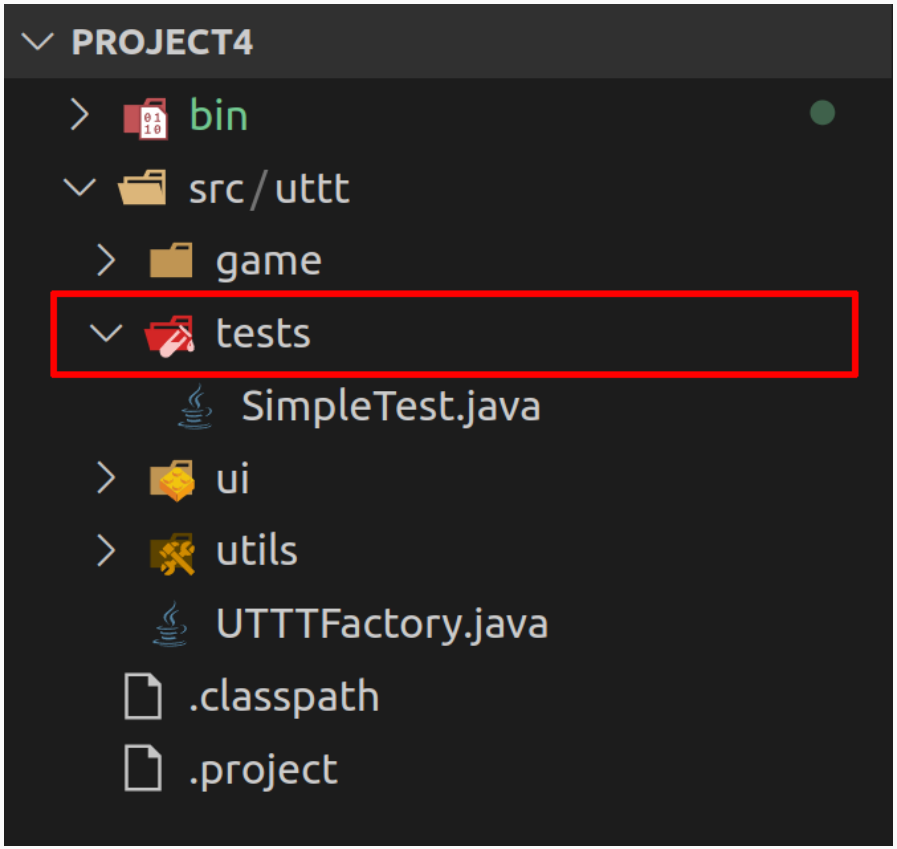
Correct implementations are useful for checking the specification

Where to start?

- What do you **not** want to see?
- What **do** you want to see?

Where to put my tests?

- New test classes should go into the `tests` folder



Requirements

```java
public class BoardTest {

    BoardInterface board;

    @Before
    public void setUp() throws Exception {
        this.board = UTTTFactory.createBoard();
    }

    @Test
    public void simpleBoardTest() {
        assertNotNull(this.board);
    }

}
```

Requirements

- Tests are marked with a `@Test` annotation

```java
public class BoardTest {

    BoardInterface board;

    @Before
    public void setUp() throws Exception {
        this.board = UTTTFactory.createBoard();
    }

    @Test
    public void simpleBoardTest() {
        assertNotNull(this.board);
    }

}
```

Requirements

- Use asserts to check the behavior of the given implementation

```java
public class BoardTest {

    BoardInterface board;

    @Before
    public void setUp() throws Exception {
        this.board = UTTTFactory.createBoard();
    }

    @Test
    public void simpleBoardTest() {
        assertNotNull(this.board);
    }

}
```

Requirements

- `@Before` marks a method that is executed before each test

```java
public class BoardTest {

    BoardInterface board;

    @Before
    public void setUp() throws Exception {
        this.board = UTTTFactory.createBoard();
    }

    @Test
    public void simpleBoardTest() {
        assertNotNull(this.board);
    }

}
```

Requirements

```java
public class BoardTest {

    BoardInterface board;

    @Before
    public void setUp() throws Exception {
        this.board = UTTTFactory.createBoard();
    }

    @Test
    public void simpleBoardTest() {
        assertNotNull(this.board);
    }

}
```

Testing for exceptions

- Testing if a function throws an exception can be done using the `assertThrows` function

```
@Test
public void symbolNullTest() {
    assertThrows(IllegalArgumentException.class, () -> {
        this.mark.setSymbol(symbol:null);
    });
}
```

Java tests vs C0 tests

```c
void test_null(void) {
    unsigned res = roots(0, 0, 0, NULL);
    assert(res == UINT_MAX);
}
```

Figure 4: C0 Test

```java
@Test
public void simpleBoardTest() {
    assertNotNull(this.board);
}
```

Figure 5: Java Test

What do you need for a test?

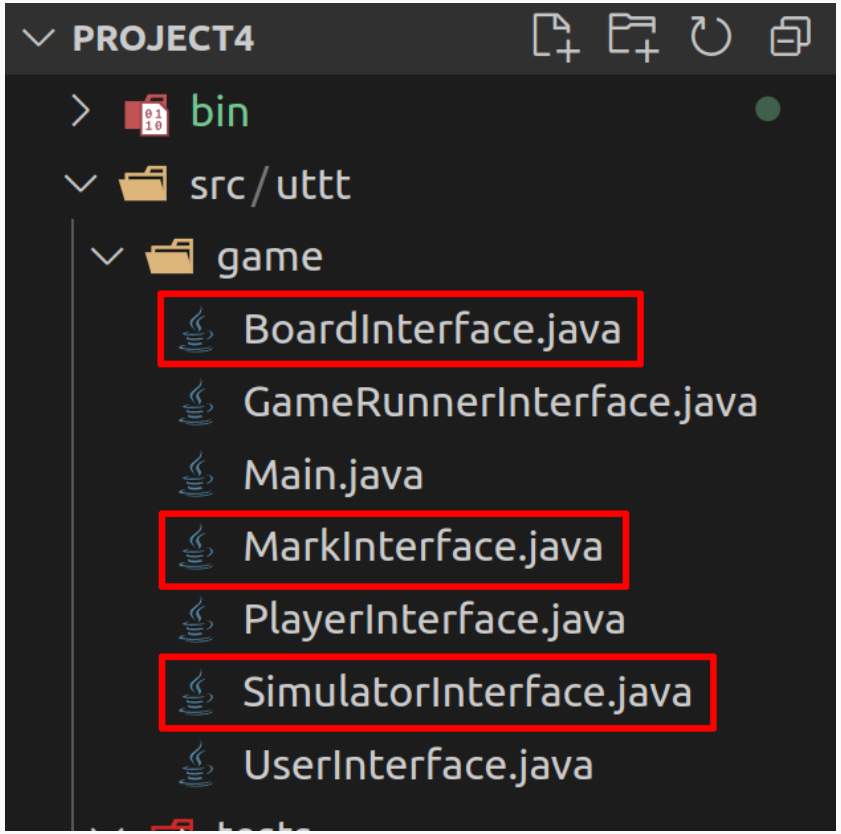
1. Correct JUnit-annotation:
   `@Test`
2. Fitting JUnit-assert-Statement:
   `assertNull()`

Useful assert-Statements:

- `assertNull(message, statement)`
- `assertNotNull(message, statement)`
- `assertTrue(message, boolean-statement)`
- `assertEquals(message, expected, actual)`
- (Example)
  `assertThrows(message, Exception.class, () -> {...})`

What to test?

- You need to test all the methods of the interfaces `MarkInterface`, `BoardInterface` and `SimulatorInterface`
- **Exception:** You don't need to test the `run` methods of `BoardInterface` and `SimulatorInterface`



Interfaces

- Interfaces represent the methods of an actual implementation
- They don't contain any code for the methods and thus can't be instantiated
- If a method is expected to throw an exception at some point, it is marked using the `throws` keyword

```java
/**
 * Set the symbol of the mark.
 *
 * @param symbol The symbol to which the mark shall be assigned.
 */
public void setSymbol(Symbol symbol) throws IllegalArgumentException;
```

UTTTFactory

Q: We have no implementation yet, how do we get objects for the tests?

A: We use the `UTTTFactory` class

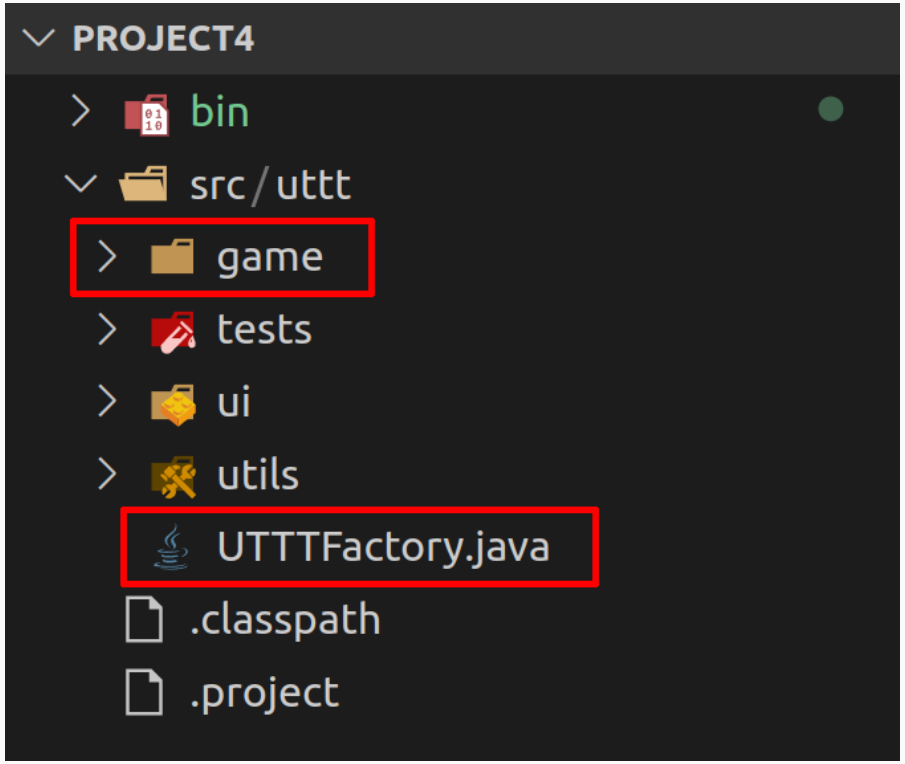
- The `UTTTFactory` class provides static methods to create the objects we need in the tests
- Example:
  `public static BoardInterface createBoard()` creates a new Tic-Tac-Toe board and returns it

- The test server provides implementation available through the methods in `UTTTFactory`

Project Details

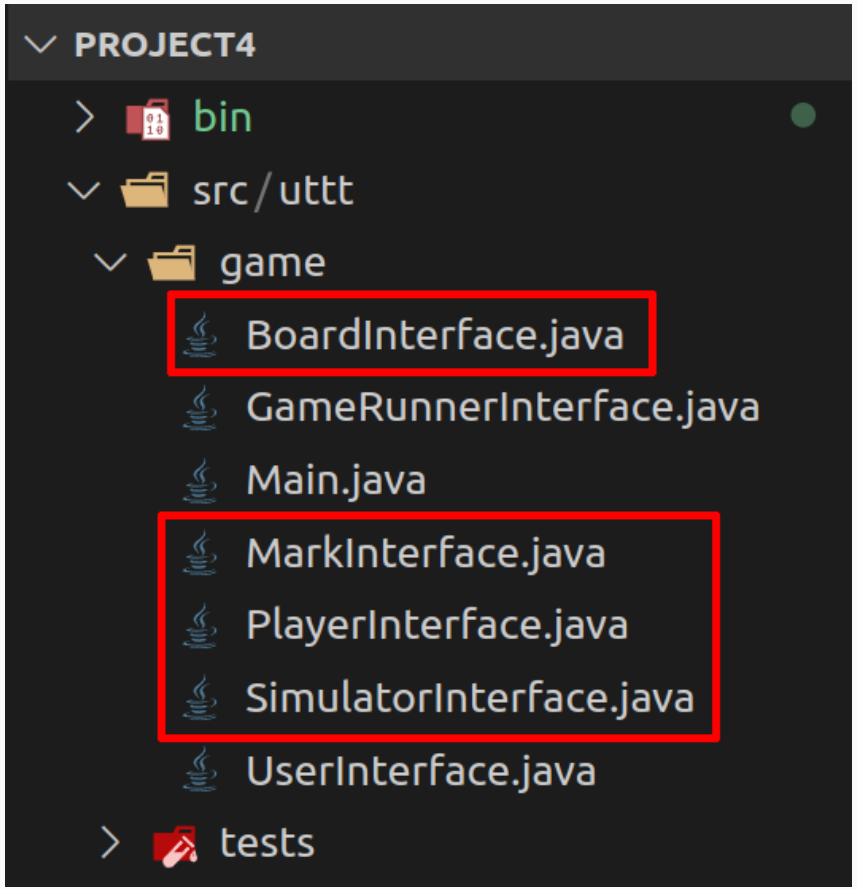
Exercise 2 - UTTT Implementation

Implementation folder



Implementation folder

- Implement the marked Interfaces
- Check the comments in the interfaces
- Do not change the given files, except for `UTTTFactory`



How to implement an Interface

- Create a new .java file in the game folder
- Give it a reasonable name (e.g. XImplementation.java)
- Use the `implements` keyword (see 8.7 Inheritance)
- Implement all methods specified in the interface

```
                    View Problem (Alt+F8)   Quick Fix... (Ctrl+.)
public class MarkImplementation implements MarkInterface {
```

Hint

Use Quick Fix... → Add unimplemented methods

UTTTFactory Implementation

UTTTFactory.java - project4 - VSCodium

Datei Bearbeiten Auswahl Anzeigen Gehe zu Ausführen Terminal Hilfe

GEÖFFNETE EDITOREN
PROJECT4
- bin
- src / uttt
  - game
    - BoardInterface.java
    - GameRunnerInterface.java
    - Main.java
    - MarkInterface.java
    - PlayerInterface.java
    - SimulatorInterface.java
    - UserInterface.java
  - tests
    - SimpleTest.java
  - ui
  - utils
  - UTTTFactory.java
- .classpath
- .project

ZEITACHSE
JAVA PROJECTS

SimpleTest.java | UTTTFactory.java

src > uttt > UTTTFactory.java > UTTTFactory > createBoard()

Administrator, letzte Woche | 1 author (Administrator)

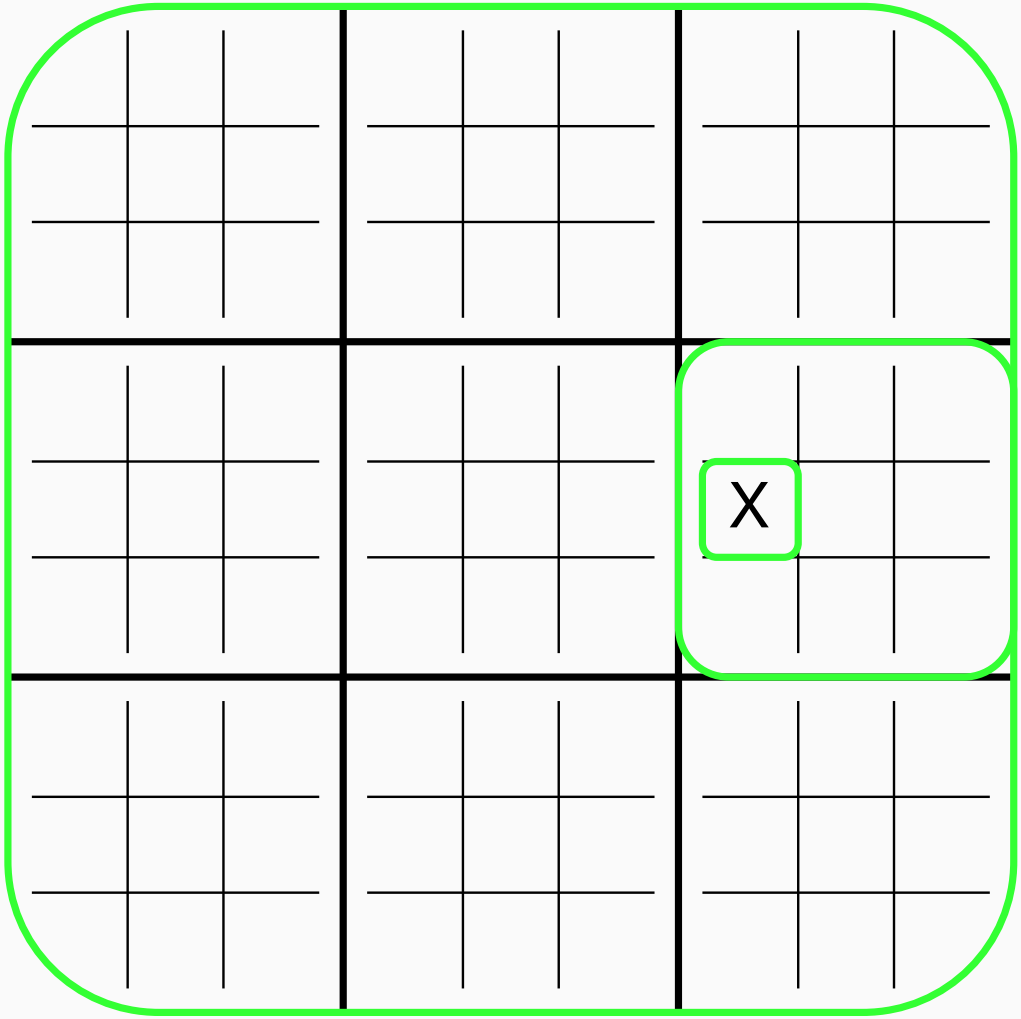
```java
package uttt;

import uttt.game.BoardInterface;
import uttt.game.MarkInterface;
import uttt.game.PlayerInterface;
import uttt.game.SimulatorInterface;
import uttt.game.UserInterface;
import uttt.ui.GUI;
import uttt.utils.Symbol;

public class UTTTFactory {

    /**
     * Create a Ultimate TicTacToe mark on board index j.
     *
     * @param symbol The symbol for the new mark.
     * @param j      The index of the piece in its board.
     *
     * @return A Ultimate TicTacToe mark.
     */
    public static MarkInterface createMark(Symbol symbol, int j) {
        throw new UnsupportedOperationException("Not yet implemented");
    }

    /**
     * Create a Ultimate TicTacToe board.
     *
     * @return A Ultimate TicTacToe board.
     */
    public static BoardInterface createBoard() {
        throw new UnsupportedOperationException("Not yet implemented");
    }

    /**
     * Create a Ultimate TicTacToe simulator.
     *
     * @return A Ultimate TicTacToe simulator.
     */
    public static SimulatorInterface createSimulator() {
        throw new UnsupportedOperationException("Not yet implemented");
    }

    /**
     * Create a user interface for a Ultimate TicTacToe simulator.
     *
```

Administrator, letzte Woche • Initial commit

master* | 0 0 | Connected to Discord | Administrator, letzte Woche | Zeile 28, Spalte 43 | Tabulatorgröße: 4 | UTF-8 | LF | Java

```
UnsupportedOperationException("Not yet implemented");
```

Game Hierarchy



Simulator

...

Board

...

Mark

Inner workings

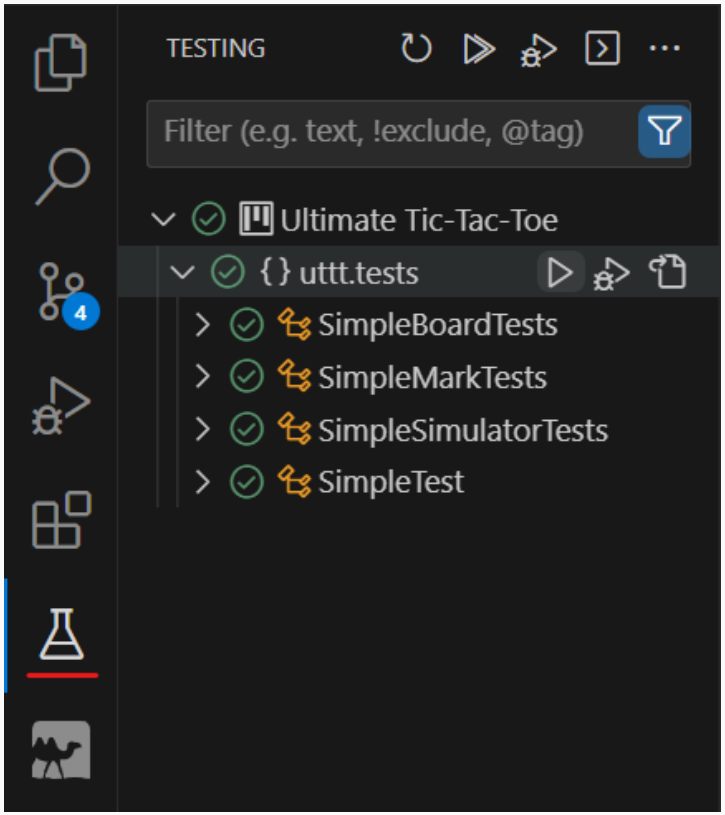
- Actual game logic will be implemented in the `run()` method of the simulator
- Runs the game and handles communication with the UI (`UserInterface`)
- Needs a working implementation of `PlayerInterface`

UserInterface methods

- `ui.showGameOverScreen(...)` to announce a winner,
- `ui.updateScreen(...)` to visualize changes,
- `ui.getUserMove(...)` to request a move from the player

Running your tests

- To run your own tests on your implementation, navigate to the Erlenmeyer Flask in VSCode
- There you can choose which of your tests to run (**or debug**)



Project Details

Autoplayer

Bonus Task - General

- You can implement an Autoplayer for Bonus Points (**Deadline: 27.06.2023**)
- There are up to 3 bonus points available
- Two `getPlayerMove()` methods for Tic-Tac-Toe and UTTT autoplayer respectively

Bonus Task - Implementation

```
Move getPlayerMove(SimulatorInterface game, UserInterface ui)
throws IllegalArgumentException;
```

Figure 6: For UTTT (human or autoplayer)

```
default int getPlayerMove(BoardInterface game, UserInterface ui) {
    return -1; // invalid move
}
```

Figure 7: For Tic-Tac-Toe (human or autoplayer)

Bonus Task - Implementation

- Implement `PlayerInterface` again. Put autoplayer logic in `getPlayerMove()` methods
- It's useful to implement the `run()` method of `BoardInterface` to test your TicTacToe autoplayer implementation
- For further details: See Project Description!
- There will be an UTTT autoplayer tournament: See Project Description!