Project Overview

This project is worth **20% of your mark**. You must work on this **individually**. You may seek help from lab tutors, but they are limited to clarifying your understanding of the project or pointing you to relevant course content. They are **not allowed** to write code for you or explain specific solution steps.

AI Code Generation Policy

AI code generation is **expressly forbidden** for this project. Tools like GPT (ChatGPT, Copilot), Bard, or Code Whisperer often produce similar code for different users, which can trigger plagiarism checkers. To avoid confusion between conventional and AI plagiarism, **submitting AI-generated code will be treated as plagiarism and academic misconduct**.

However, you may use AI tools to **understand the project** or **research solutions**, similar to using Google or textbooks. Refer to the Using AI Tools at UWA guidelines for proper usage.

Submission Instructions

- Submit your solutions via the **"Upload Files"** section of the LMS assignment.
- **Deadline:** 23:59 (end of day) on **Thursday, May 22**.
  - Unit staff will not assist with submission issues outside working hours. Aim to submit by **15:00** to avoid last-minute issues.
- Submit a `.zip` file containing only a `src` directory with the same structure as the project folder:

```
src
├── ai
└── game
```

  - Include only `.java` files. **Do not submit** `.class`, `.jar`, test files, or documents like PDFs.
  - Ensure the directory structure is preserved. Incorrect submissions may result in a **mark of 0**.
  - Your code must compile and run with the provided test programs. Failure to do so may also result in a **mark of 0**.

Project Details

Overview

You will add missing classes to an existing codebase. The codebase has two top-level packages: `game` and `ai`.

- The `game` package contains classes for a simple game and a `game.tests` subpackage with tests.
- The `ai` package implements an AI to play the game and a terminal-based program for user vs. AI gameplay.

Your task:

1. Complete the `game` package by implementing the `Move`, `Grid`, and `Game` interfaces.
2. Complete the TODO in the `GameTest` class in the `game.tests` subpackage.

The Game Rules

- The game is played on a square grid (e.g., 5x5).
- Two players alternate turns, placing black or white pieces on unoccupied squares.
- The **white player** starts first.
- A player wins by creating a **4-connected path**:
  - From the **top row to the bottom row**, or
  - From the **left column to the right column**.
- A **4-connected path** moves only in cardinal directions (up, down, left, right), not diagonally.
- The game ends in a **draw** if no empty squares remain and no player has won.

Example Boards

White Wins (Left to Right)

```
WWWWB
...WB
..BWW
..BBB
..WB.
```

Path: `1234B ...5B ..B67 ..BBB ..WB.`

Black Wins (Top to Bottom)

```
WWWWB
..BWB
...WB
.BBBB
.BWWW
```

Path: `WWWW1 ..BW2 ...W3 .7654 .8WWW`

Instructions

1. **Download and Unzip the Project Code**
   - Directory structure:

```
src
├── ai
│   ├── AI.java
│   ├── Heuristic.java
│   ├── Minimax.java
│   ├── MinPiecesHeuristic.java
│   └── PlayVsAI.java
└── game
        ├── Game.java
        ├── Grid.java
        ├── Move.java
        ├── PathFinder.java
        ├── PieceColour.java
        └── tests
                ├── Test.java
                ├── MoveTest.java
                ├── GridTest.java
                └── GameTest.java
```

2. **Implement the Interfaces**
   - **Move Interface**:
     - Create `MoveImpl.java` in the `game` package.
     - Ensure it passes the `MoveTest` program.
     - Override the `toString()` method as per `MoveTest`.
   - **Grid Interface**:
     - Create `GridImpl.java` in the `game` package.
     - Ensure it passes the `GridTest` program.

    - Override the `toString()` method as per `GridTest`.
  - **Game Interface**:
    - Create `GameImpl.java` in the `game` package.
    - Use the `PathFinder` class to implement the game logic.
    - Ensure it passes the `GameTest` program.
3. **Extend the GameTest Class**
  - Write comprehensive tests for all methods of the `Game` interface.
  - Test against various incorrect and correct versions of `GameImpl`.
4. **Play Against the AI**
  - Run `ai/PlayVsAI.java` to play against the AI.

Notes on Compilation

- Use `javac` to compile your code. Remember:
  - `javac` does not recompile dependencies automatically. If you modify a class, recompile it manually.
  - Alternatively, delete `.class` files to force recompilation.

Marking Rubric

Component	Basic	Satisfactory	Proficient	Marks	Learning Outcomes
MoveImpl.java	Passes most tests (+1)	Passes all tests (+1)		/2	1, 2
GridImpl.java	Compiles (+1)	Passes most tests (+1)	Passes all tests (+1)	/3	1, 2
GameImpl.java	Compiles (+1)	Passes most secret tests (+1)	Passes all secret tests (+1)	/3	1, 2
GameTest.java	Catches one bug (+1)	Catches most bugs (+1)	Catches all bugs (+1)	/3	1, 2, 3
Style	Proper indentation (+1)	Follows best practices (+1)	Elegant and concise (+1)	/4	2, 4