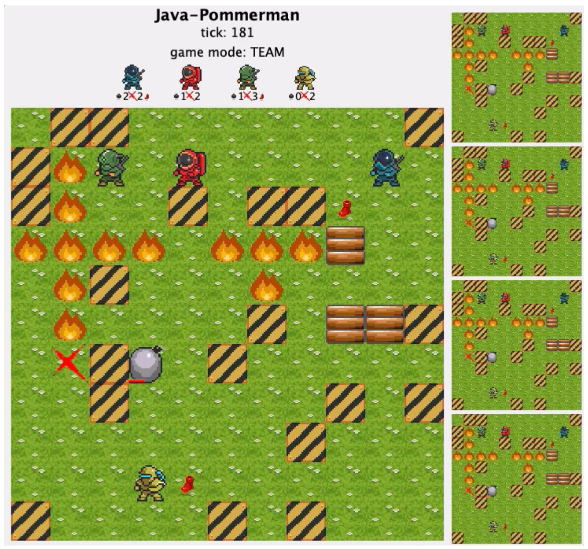
ECS7002P - Lab Exercises - Week 1
Pommerman

Brief Game Description

Pommerman is a complex multi-player and partially observable game where agents try to be the last standing to win. This game poses very interesting challenges to AI, such as collaboration, learning and planning.

The game takes place in a randomly drawn 11 × 11 grid between 4 players. Players are placed in the 4 corners of the grid and the level is scattered with obstacles of two types: wooden and rigid boxes. The board is symmetric along the main diagonal. The game can be played with full or partial observability. The objective of the game is to be the last player alive at the end, or the last standing team. Players can place bombs that explode after 10 game ticks. These bombs, when exploded, generate flames in a cross-shape with size of (initially) 1 cell around the explosion point. These flames eliminate players, items and wooden boxes on collision, as well as explode other bombs. By default, players can't drop more than one bomb at a time. Bombs, as well as obstacles, can't be traversed. Wooden obstacles, when destroyed, may reveal pick-ups, initially hidden to the players. These items are *extra bombs* (adds 1 to the bombs ammo), *blast strength* (adds one to the explosion size) and *kick* (allows kicking a bomb in a straight line). To make sure games are finite, a limit of 800 game ticks is set. The play area is generated automatically using a random `seed`.



The objective of these exercises is to prepare you for the first assignment of the module. In this exercises, you will familiarise yourself with *Pommerman* and how to process the information that is given to you in the different methods that you can implement.

Exercise 0: Getting started

In order to get the framework installed and running, follow these instructions:

1. Clone the following repository using git in a Unix or Windows terminal:

   git clone `https://github.com/GAIGResearch/java-pommerman`

   You can also download the repository or check it out with SVN.

2. Set up your IDE to use the project. These instructions allow you to set it up with IntelliJ IDEA, but you can also use any another IDE you feel more comfortable with.

   (a) Open IntelliJ IDEA

   (b) Select 'Create New Project' and select 'Java'. Progress clicking on 'Next'.

   (c) Type in any project name you want and select, as project location, the 'java-pommerman' folder (the one containing a *src/* directory). Click on Finish (say 'yes' to overwrite the project folder if prompted).

3. Verify that *Pommerman* can execute in the IDE:

   (a) Click on File → Project Structure, and go to Libraries.

   (b) Click on the '+' symbol and select Java. A directory browsing window should appear.

   (c) Select the all the jar files in the directory 'lib/' and click on Open.

   (d) Click then on 'Ok' first to add the libraries to the project and then again to close the Project Structure window.

   (e) In the IDE, open the file src/Test.java (you can navigate files on the left panel - click on *1:Project* if the folder hierarchy is not displayed).

   (f) Click on menu Run → Run, and select *Test* in the pop up window. Alternatively, right click the file Test.java and select the option 'Run'.

   (g) It'll take a few seconds for the project to be compiled and then it will run a game with four AIs playing.

1 Exercise 1 - Understanding and running the game

1.1 Understanding Pommerman

The first step is to inform yourself about the game - how does it work, what kind of objects and actions are available, rules, etc. For this, the first exercise is for you to read the documentation about the framework (introduction, game definition, framework structure, etc) in the framework's Wiki:

`https://github.com/GAIGResearch/java-pommerman/wiki`

Note: it's very important that you read the game rules so you understand the game. Take a look at the rules described here: `https://github.com/GAIGResearch/java-pommerman/wiki/Pommerman-Game-Rules`.

To get a better understanding of the framework, have a look at the code structure described here:

`https://github.com/GAIGResearch/java-pommerman/wiki/Framework-Structure`

The *Pommerman* wiki is generally a good place to find information about the framework.

1.2 Running Pommerman

There are two classes, Test.java and Run.java, in the src/ directory that can be used to run your bots and also to play the game yourself against the built-in AIs.

1.2.1 Single Games

For now, open the class **Test.java**. You'll see that, from line 40 onward, lines like the following are repeated:

```java
players.add(new ...);
```

players is a Java *ArrayList* that contains all players for a single game. Before a game is started, this array must contain four players - no less, no more. The order of the players in the array matters: from first to fourth, the each player is put in a different corner of the map following this order: top-left, bottom-left, bottom-right, top-right.

By default, the game has two "SimplePlayer" and two MCTS playing. The different players you can use at the moment in this framework are:

Bot	Java construction code
DoNothing	new DoNothingPlayer (playerID++)
Random	new RandomPlayer (seed, playerID++)
OSLA	new OSLAPlayer (seed, playerID++)
SimplePlayer	new SimplePlayer (seed, playerID++)
RHEA	new RHEAPlayer(seed, playerID++)
MCTS	new MCTSPlayer(seed, playerID++)

All the player classes are in src/players/. The seed is used to generate a random starting board for the game, which determines the location of the obstacles and power-ups. Each seed number genreates a completely different map to play in.

By changing the players that are added to the *players* array, try now different combinations of agents playing the game and see how well they do, by running the class **Test.java**. Also take a quick look now at `https://github.com/GAIGResearch/java-pommerman/wiki/AI-Players` and read about the different AI players available. We'll be back to analysing these players in Exercise 3 (next week), but it's good for you to try to understand them on your own for now.

Playing as a human You can also include a human player in the groups of agents that play. A human player is created as follows:

```java
new HumanPlayer(ki1, playerID++)
```

where *ki1* is a *KeyController* object (created in line 26). Substitute the first agent in the array of players for a new human player. Run the Test class and play the game using the arrow keys to move and the Space key to drop a bomb.

You can actually play the game with up to two human agents:

Human Agent	**Key Controller**	**Keys**
Human player 1	KeyController ki1 = new KeyController(true);	Cursor keys + Space
Human player 2	KeyController ki2 = new KeyController(false);	WASD + Left Shift

Configure the game now to play with 2 humans and 2 bots. This would allow you to play with someone else in the same computer - try it out to see if it works.

Different game modes The game can be played in two different game modes, with the following rules:

- Free for all (FFA): All four players are competing against each other.
  - Last player standing **wins**.
  - All players that die **lose** the game.
  - Players that are alive at the end **tie**.
  - If the last players alive die on the same tick, they also **tie**.
- Team: The four players are grouped in teams of 2 vs 2.
  - They're always paired 1^{st}-3^{rd} vs 2^{nd}-4^{th}, sited in opposite corners.
  - Team with no agent **loses**, other team **wins**.
  - If there are players of each team alive at the end, teams **tie**.
  - If the last alive players of each time die on the same tick, teams **tie**.

The team mode is decided when the *Game* object is created. See Test.java, line 23, which by default sets FFA as the game mode:

```java
Game game = new Game(seed, boardSize, Types.GAME_MODE.FFA, "");
```

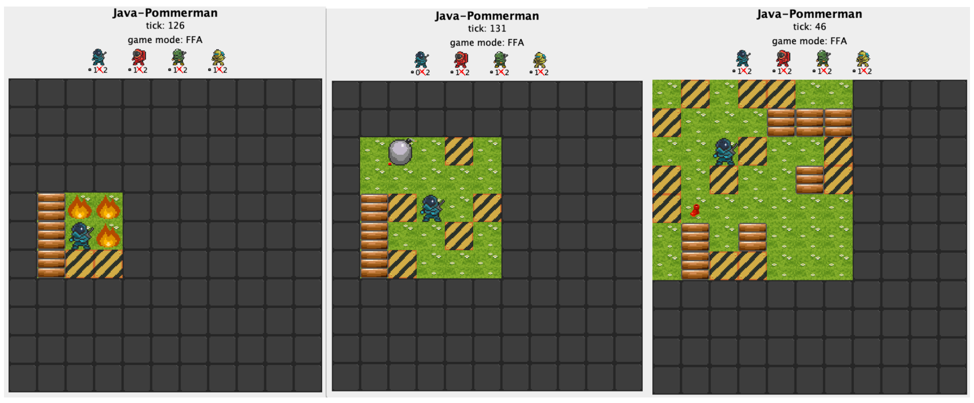
In order to change the mode to TEAM, substitute *Types.GAME_MODE.FFA* for *Types.GAME_MODE.TEAM* and play the game.

Different observabilities All players receive information (presence and location of the different tiles) about their surroundings. By default, the agents receive information about the **full** board (full observability setting). This is indicated by the variable *DEFAULT_VISION_RANGE* in the class *Types.java* (src/utils/Types.java).

utils.Types.DEFAULT_VISION_RANGE value	**Observability**
-1	Full observability
N (> 0)	Partial observability (N tiles from location)

When a value $N > 0$ is set for the Vision Range, the agent will only receive information about the tiles at a distance $\leq N$ from the location of the agent:

Execute now the game with 4 different **bot** agents and a PO setting of 2. Observe how the game takes place. Note that, with Partial Observability and no human players, you see the vision of an independent observer, while you can see the visibility of every player on the right part of the screen. You can press 1, 2, 3 and 4 keys to toggle which agent's view you display on the main left panel. Press 0 to come back to the independent observer's view.



PO $N = 1$ PO $N = 2$ PO $N = 4$

Figure 1: *Pommerman* executed with different Partial Observability (PO) settings.

Game settings *Pommerman* is an easily parameterizable game. This means that there are many ways you can tweak the rules and the game parameters. Take a look at `https://github.com/GAIGResearch/java-pommerman/wiki/Game-Parameters` and modify some of these parameters to see the effect they take on the game and the agents that play it.

1.2.2 Multiple Games

When trying to determine if an AI is better at *Pommerman* (or any other environment, really), it is never enough to play a single game. Because of this, the framework includes another class (Run.java) that allows you to run multiple games and get comprehensive statistics at the end of the execution.

Open Run.java (in src/). This class is executed passing 7 parameters as arguments (although passing none executes a default mode). The usage instructions are as follows:

```
Usage: java Run [args]
    [arg index = 0] Game Mode. 0: FFA; 1: TEAM
    [arg index = 1] Number of level generation seeds (i.e. different levels).
    [arg index = 2] Repetitions per seed [N]. "1" for one game only with visuals.
    [arg index = 3] Vision Range [VR]. (0, 1, 2 for PO; -1 for Full Observability)
    [arg index = 4-7] Agents. When in TEAM, agents are mates as indices 4-6, 5-7:
      0 DoNothing
      1 Random
      2 OSLA
      3 SimplePlayer
      4 RHEA 200 itereations, shift buffer, pop size 1, random init, length: 12
      5 MCTS 200 iterations, length: 12
      6 Human Player (controls: cursor keys + space bar).
```

The total number of games played will be the number of seeds (arg index 1) multiplied by the number of repetitions by seed (arg index 2). For instance, running:

```
java Run 0 1 1 -1 2 3 4 5
```

Executes one FFA game with 1 as random seed, full observability and the players (in this order): OSLA, SimplePlayer, RHEA and MCTS. This is also the default setting when no parameters are passed to the class.

Run now 'Run.java' to see what happens (Menu 'Run' → 'Run' and select 'Run').

In order to set your own parameters in IntellijIDEA, click on 'Run' → 'Edit Configurations...' and introduce them (separated by spaces) in the field 'Program arguments:'. Try different arguments, setting different number of seeds (i.e. different levels) and repetitions per seed. You should obtain an output with this format:

```
0-2-2--1-2-3-4-5 [OSLA,RuleBased,RHEA,MCTS]
40665, 0/4, [LOSS, LOSS, LOSS, WIN]
40665, 1/4, [LOSS, LOSS, LOSS, WIN]
```

```
83757, 2/4, [LOSS, LOSS, LOSS, WIN]
83757, 3/4, [LOSS, LOSS, WIN, LOSS]
N   Win   Tie   Loss  Player
4 0.0%  0.0%  100.0%  players.OSLAPlayer
4 0.0%  0.0%  100.0%  players.SimplePlayer
4 25.0% 0.0%  75.0% players.rhea.RHEAPlayer
4 75.0% 0.0%  25.0% players.mcts.MCTSPlayer
```

The lines of the output are described as follows:

- Line 1: execution string (with the parameters specified) and agents that played the games.
- Lines 2 onward: seed for the level, game number, result per player (same order as in the first line).
- Last 5 lines: table of results, where each column indicates, from left to right, number of games played, percentage of wins, ties and losses, and the player.

Run this class a few times and try to determine which is the strongest agent of the ones available by pitching them against each other.

2 Conclusion Exercise - Sharing your results

Run a tournament between 4 agents of your liking, in at least 10 different levels with no less than 5 repetitions for each level. Add a comment to the relevant post in the forum indicating:

- The settings you have chosen for the tournament.
- The results of the contest.