Author: Sandra Garcia Rodriguez. <sandragrodriguez@gmail.com>

The package contains an eclipse project that implements an artificial intelligence module applied to a strategy game with some specific rules. However, this module can be adapted to different games with other rules by modifying some methods of the JAVA files. You are free to reuse the code for your own interest.

A Java documentation in “pathfinding/doc” folder is also provided to help to understand better the code.

Components:

- com.uc3m.gade4all/Principal.java: Part I of the AI module. This is the main code to generate different scenario implementations. Regarding the model game rules (showed below), the AI procedure decides the best actions for each enemy regarding the players, obstacles, life points, specials cells and other factors that can influence on the choosing of the right decision. This class also implements a first approach to test the AI module (set parameters in method *test*()).
- com.uc3m.gade4all/AI: Path finding algorithm implementation. A* can be replaced by any other.
- com.uc3m.gade4all/game: Game elements.

Instructions to execute:

1) Open the project in eclipse.
2) Run as an Android application
3) Use the screen of your android mobile and the shell of the computer to play

Specific game rules for the implementation:

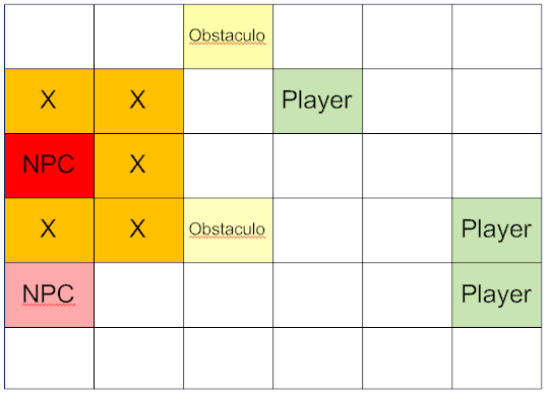
The movement area is a fixed matrix of NxZ. For instance 8x8, 10x8, 13x13, etc.

0,0	1,0	2,0	3,0	4,0	5,0
0,1	1,1	2,1	3,1	4,1	5,1
0,2	1,2	2,2	3,2	4,2	5,2
0,3	1,3	2,3	3,3	4,3	5,3
0,4	1,4	2,4	3,4	4,4	5,4
0,5	1,5	2,5	3,5	4,5	5,5

There are several elements that can be situated on the area. The AI module affects to the non-controlled elements **NPC**, deciding their next action to perform.

The **NPC** can move around the movement area and/or attack to a **Player**:

- The movement is set by the capacity of movement of the element (number of cells that it can move in any direction) and it is specific for each element. For instance, with *movement*=1 the NPC can go to:



- The attack radius is specific for each element. For instance, if *attackRadius*=2, the NPC can attack to any player situated at n*2+1 (n=2) cells from the NPC position:

An optimum NPC must move and/or attack to:

- Eliminate the **Players** as soon as possible.
- Stay alive, avoids any dangerous situation.

- Optionally, defend a special cell **S**.
- Optionally, move to a special cell **S**.

If **S** exists, it must be the goal of the NPC:

- The **NPC** may situate on it.
- The **NPC** may attack to the **Player** situated on it.
- The **NPC** may protect the **NPC** that is in this cell.

The IA module must decide the best actions that the current NPC must perform. It returns an *IAResult* object per action. Each turn, the current enemy can:

1) Move.
2) Attack.
3) Move & attack.

There must also be some difficulty levels (from 0-10) to lead the quality of the decisions taken by the NPC. For instance, in level 0 the NPCs avoids to take the best decision (don't attack some close Players, don't defend a special cell, etc.), in level 10 the NPC should play as best as it can.