Report for the GameSimulations.

What is the GameSimulations project.

Is an Java implementation of well known theoretical graph games, played turn-by-turn by two players.

Why we did it.

Our initial motivation for the GameSimulation project was to implement some theoretical graph games to help recherches to actually play the games and see their token on the screen. Later, we focused on implementing some existing strategies for those games, to allow people to play against "the computer" and try their own strategies.

Our work.

We focused on two players turn-by-turn games in graphs: The “Cops and Robbers” and the “Surveillance” game. In both games, each player has a team of tokens to place or move.

In cops and robber games, a team of cops aims are capturing a robber moving in a graph. It is a turn by turn game with full information: at each step, the cops may move to adjacent neighbors, then the robber moves to one of its neighbor. The robber is captured if it occupies the same node as a cop. The problem is to determine the smallest number of cops that ensures the capture of the robber in a graph. Plenty of strategies have been proposed to allow the cops to capture a robber in particular graph classes or in general (also, is exists few strategies for the robber to escape). There are also plenty of extension of these game. (e.g., see [1] and reference therein, see also Section 2 of [2]).

The surveillance game, is another two-players game similar to Cops and Robber Games in graphs. The first player, a fugitive, starts on a marked vertex of a graph G. The second player, an observer, marks $k \geq 1$ vertices, then the fugitive moves along one edge/arc of G to a new vertex, then the observer marks k vertices, etc. The observer wins if he prevents the fugitive to reach an unmarked vertex. The fugitive wins otherwise, i.e., if he succeed to enter an unmarked vertex. The surveillance number of a graph is the minimum $k \geq 1$ allowing the observer to win against any strategy of the fugitive.

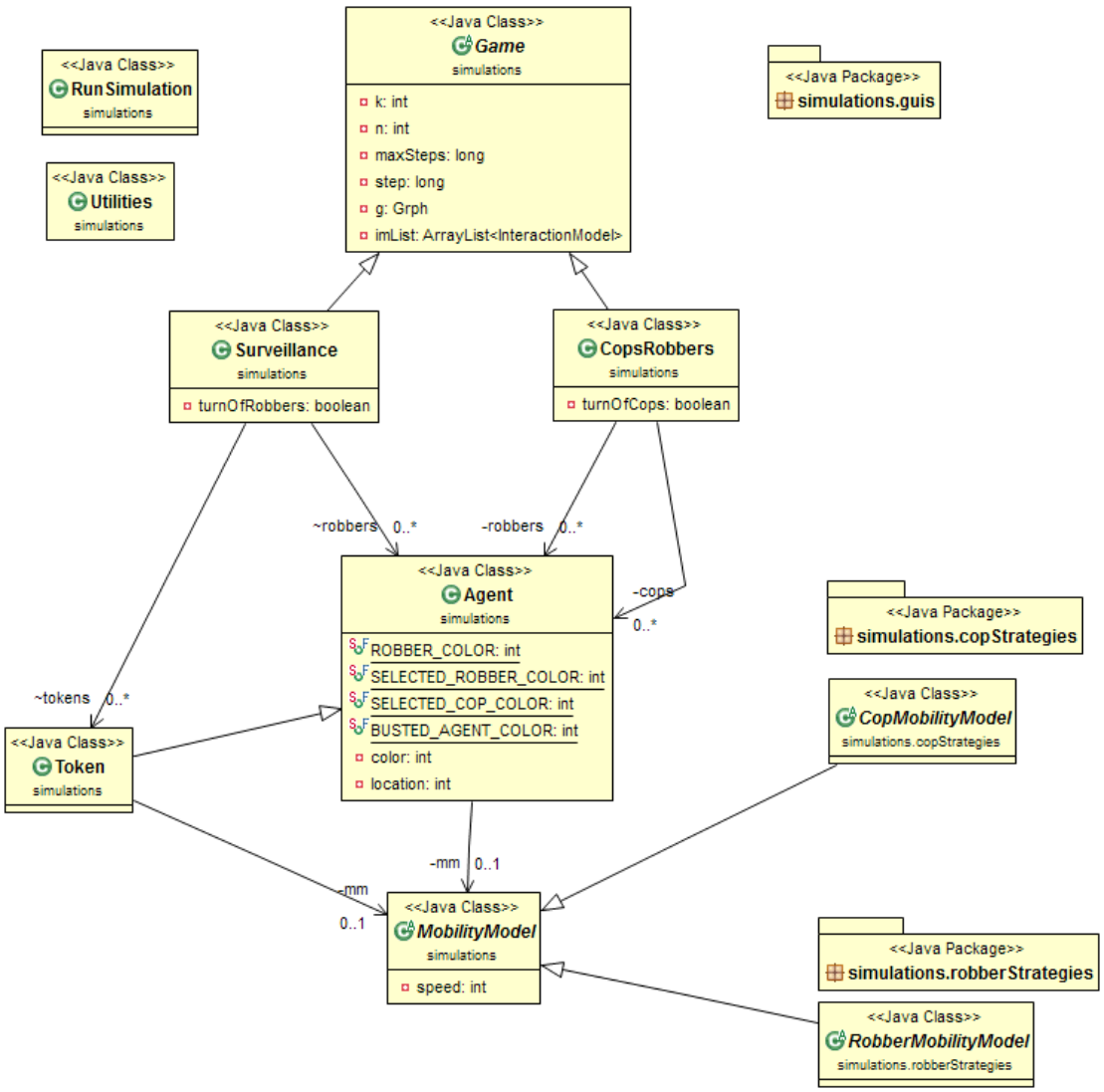
GameSimulations is a Java application, written in an object oriented way, aiming to be flexible for hosting as many games and strategies is possible.

The basic components of this applications are:

- The classes defining a game and its rules (e.g.: `Game`, `CopsRobbers, Surveillance`).
- The classes describing the rules of a team or a strategies. (e.g.: `CopMobilityModel, RobberMobilityModel, Agent, Token)`
- The classes responsible for the interaction between the program and the user (interaction models). (e.g.: `InteractionBasedOnConfigFile, InteractionBasedOnGUI)`

UML Diagrams... small java doc what the basic classes does...

The following picture, shows the relation between some of the most basic classes.



The Game class contains the common data and methods that a turn by turn game could have. Such informations are, the graph, the number of the agents or the tokens and the current round.

An other piece of information the Game class has, is a list with the Interaction Models associated to the current game.

An interaction model describes how the required information for the initialization of the game, will be collected and what processes has to be done when important phases of the game occur (e.g.: when the game is started and some information has to be displayed to the user).

For example the InteractionBasedOnGUI initializes the game asking the user to chose between the available games, the number of agents and the strategies each agent should have. On the other hand, the InteractionBasedOnConfigFile gets the above information through a configuration file.

The classes CopsRobbers and Surveillance are the implementation of the two games introduced earlier at this document. Those two classes extends the Game class and in addition, contain the agent teams for each player. The agents teams: 'cops' and 'robber' for the first class and the 'tokens' and the 'robber', for the other.

The Agent class contains all the properties for an agent (such as the location of the agent in the graph)

alongside with the Mobility Model that is assigned to him. Each agent has a mobility model, allowing him to move in the graph. The mobility model, is actually the strategy of the agent (a full list of the strategies follows later). Notice that a token is an unmovable agent, thus a token does not have a mobility model.

Both games are turn by turn games, so at any turn (or 'step' according to the step method at the above two classes) the simulation handles the agent teams according to the strategy assigned to them.

Robber Strategies

Name of strategy	**Cops & Robber**	**Surveillance**	**Description**
MoveRobberByUser	✔	✔	Displays some dialog boxes asking the user to move each agent at the desired location or pass.
MoveRobberRandomly	✔ Avoiding Cop locations	✔ Looking for unmarked vertices	Moves the agent randomly to his valid neighborhood.
MoveRobberSimpeStrategy	✔	✖	Searches for a vertex that is not adjacent to a cop. If such a vertex exists, he move there, otherwise he stays at the same position.

Cop Strategies

Name of strategy	**Description**	**Special Info**
MoveCop_ByUser	Displays some dialog boxes asking the user to move each agent at the desired location or pass.	-
MoveCop_Randomly	Moves the cop randomly to his valid neighborhood. If the robber is inside the cop's neighborhood he captures the robber.	-
MoveCop_CopWinStrategy	Computes an tree for the cop win graph and moves the agent toward the robber choosing vertices that are ancestors of the robber's position.	Only if the graph is a cop win graph.
MoveCop_CopWinGeneralGraphs	This is the Cop win strategy generalized for	-

	all graphs.	
MoveCop_ControlPathWithOneAgent	Uses one agent to control a shortest path (preventing Robber to visit the path).	MoveCop_CopWinGeneralGraphs
MoveCop_FollowAgentOnPath	Uses at most 5 agents to control a shortest path.	Extends the capabilities of the MoveCop_ControlPathWithOneAgent strategy

An other important class is the `Utilities` class, which contains methods that are, responsible for generating a graph topology and general graph methods, like finding an ancestor of a vertex that belongs to a tree.

Finally, the classes under the 'simulations.gui' package are responsible for the graphical interface between the user and the application. The `UInputs` class contains most of the dialog boxes displayed to the user waiting for an input.

How you can run it.

First make sure you have Java installed on your system. You can verify that Java is installed by typing into the command terminal:

```
java -version
```

The application requires a Java Virtual Machine library of version 1.6 or higher (You can download one from http://java.com/en/download/index.jsp).

Then you can run the application, as any runnable jar file, by executing the file or through a terminal by the command:

```
java -jar GameSimulationsName.jar
```

By default the application runs in gui mode, and displays dialog boxes asking the user to chose the game and it's parameters. However, the application contains a configuration mode too, where the user can input a configuration file with the parameters of the game. For full control of the application it is advised to use the command line mode.

```
Usage: simulation_file_name.jar [options] [input-file]
Options:
        -s      Silent Configuration Mode (for command terminal).
        -h      Displays this help message.
```

Full example:

```
java -jar Simulations.jar -s inputConfig1.txt
```

Inside the project folder there are some input configurations files. You can find a description for them

at the files “input_Cops_N_Robbers.txt” and “input_Surveillance.txt” along side with some other example input-files.

How to Import/Export a project to Eclipse.

There is a tutorial there:

http://agile.csc.ncsu.edu/SEMaterials/tutorials/import_export/

It is advised to have a Java Virtual Machine library of version 1.7 or higher to build correctly this project.

After importing the project from the archive file, if there are some errors, maybe you have to update the required libraries.

To do this: Right click on the project and go to 'Properties'. At the new window, go to “Java Build Path” then go to “Libraries” tab. There will be some libraries with a 'red x icon'. Remove all the libraries except the JRE System Library and go to “Add External JARs” Then add all the libraries under the “required_libraries” folder inside the project's folder (it should be at .../EclipseWorkspaceFolder/Gamesimulations/required_libraries). Then click 'ok' and 'Refresh' the project (Right click on the project and refresh or F5).

Bibliography

[1] Jérémie Chalopin, Victor Chepoi, Nicolas Nisse and Yann Vaxès, Cop and robber games when the robber can hide and ride.
SIAM Journal of Discrete Maths, Volume 25(1), pages 333-359, 2011.
http://hal.archives-ouvertes.fr/inria-00448243/fr/

[2] Adrian Kosowski, Bi Li, Nicolas Nisse and Karol Suchan. k-Chordal Graphs: from Cops and Robber to Compact Routing via Treewidth.
In Proceedings of 39th International Colloquium on Automata, Languages and Programming (ICALP), track C, Springer LNCS 7392, pages 610-622, 2012.
http://hal.inria.fr/hal-00671861

[3] Surveillance... To Satisfy Impatient Web Surfers Is Hard