# Comp 424 - Project Description
# **Saboteur**

*Course Instructor:* Jackie Cheung (jcheung@cs.mcgill.ca)
*Course Instructor:* Adam Trischler (Adam.Trischler@gmail.com)
*Project TA:* Pierre Orhan (pierre.orhan@mail.mcgill.ca)

**Code repository:** https://github.com/PierreOrhan/SaboteurComp424

**Code due**: 9 April 2020
**Report due**: 9 April 2020

## 1 Overview

An implementation of Saboteur is provided. Playing games requires: a server (the real board) to be launched, and then two clients (agents) to connect to the server via TCP sockets; this allows for clients to be run on separate computers. We also provide a GUI which displays the game in a nice and intuitive way for learning the strategy behind Saboteur. Note that games between agents can be played without the GUI to increase speed.

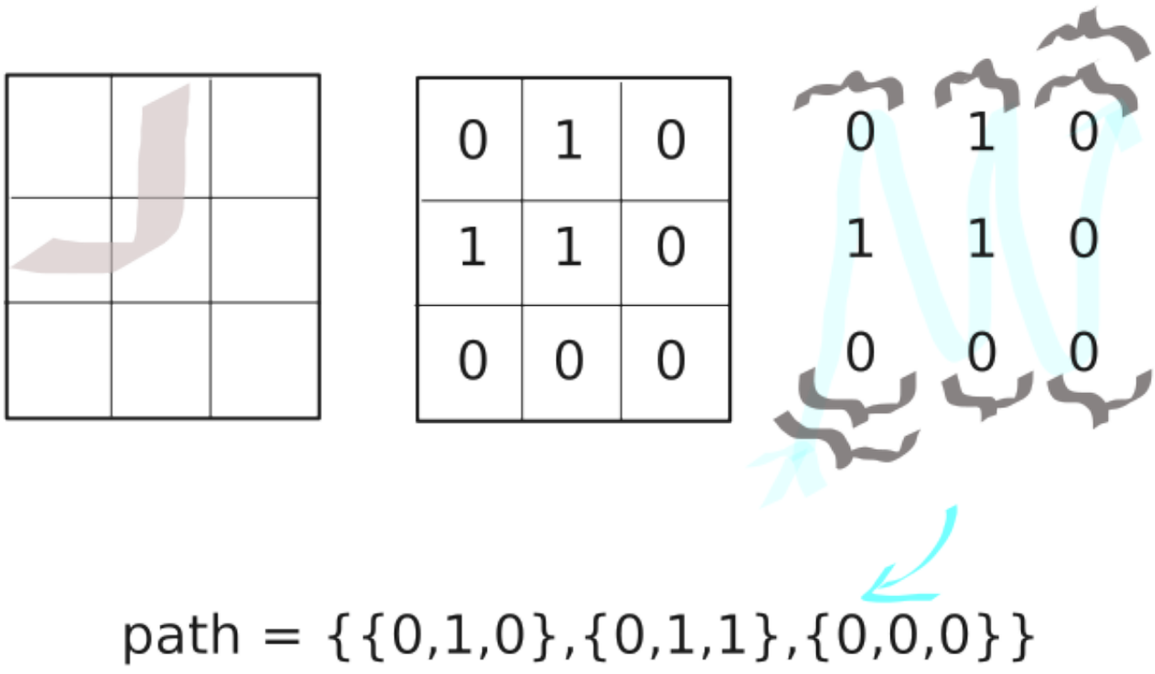
All source code is found in the **src** directory. **You must NOT edit any packages**, other than student_player, which can be modified as much as you want so long as it (1) compiles, and, (2) that you do not use any external libraries, but you can use default internal Java libraries (like Math and Util). If you edit other packages you must be aware that your edits will not be used during the tournament; e.g., so if you know what you're doing, you may want to edit **Autoplay.java** to run many test games of your agent against another agent (or perhaps against itself).

We encourage you to compare your agent with other groups prior to the submission. Set-up the server with the agents of the two group and try a few games!

- src
  - autoplay
    - Autoplay.java *Autoplays games; can be ignored.*
  - boardgame *Package for implementing boardgames, provides useful software infrastructure for logging, GUI, and server TCP protocol. Primarily, it abstracts game logic, but can be ignored for this project.*
  - student_player *Package containing your agent.*
    - SutendPlayer.java *The class you will implement your AI within.*
    - MyTools.java *Placeholder for any extra code you may need.*
  - Saboteur
    - cardClasses *Package with the different types of card, all extending the SaboteurCard class*
      - SaboteurTile *take a look in these classes and understand them to manipulate them.*
    - tiles *directory with images files.*
    - SaboteurBoardPanel.java *Implements the GUI, IN THE CONSTRUCTOR YOU CHANGE THE SCALE DOUBLE (DECREASE IT WILL MAKE THE BOARD BIGGER) IF YOUR SCREEN IS LARGE ENOUGH!*
    - SaboteurBoard.java *Used for server logic, can be ignored.*
    - SaboteurMove.java *A move object for Saboteur. All your moves should be given as instance of this class.*
    - SaboteurBoardState.java *Implements all game logic, most important.*
      - `getHiddenIntBoard()` *Returns the board as 0,1 or -1 where 1 are tunnel, 0 walls and -1 empty position. Hidden objectives will be displayed as empty.*
      - getHiddenBoard() *Returns the board as SaboteurTile, where the remaining hidden objectives for the current playing player are displayed as tile number 8 (the cross).*
      - getCurrentPlayerCards() *Returns current player hand*
      - clone() *Returns a cloned SaboteurState, you are not allowed to use this method (made for server purpose)*
      - getTurnPlayer() *get the current player number (1 for player 1, 0 for player 2), it should always be you when you are requested a move*
      - getRandomMove() *A random move*
      - `getAllLegalMoves`() *returns all legal moves the current player could do.*
      - `possiblePositions()` *returns all possible position for a tile to be put on the board. Be careful: it does not test for the flipped version of the tile*
      - `verifyLegit()` *Given a tile's path (it's 0-1 version) and a position ({ int x, int y }) indicates if the card can be put at this position legally. (does not test for player status)*

- `processMove()` *Process a move, if the move is illegal, throws an IllegalArgumentException and the winner is the other player.*
- `printBoard()` *A 0-1 print of the board*

## 2 Attention on the implementation of the path




The SaboteurTile instance has its path written as int[3][3] arrays. The convention is described above: left the tile, middle: its representation inside the int[][] board, right: its representation as a path in a SaboteurTile instance. It does not seem intuitive at first but you will get used to it. A close look at how the int[][] board is created in the SaboteurBoardState class can also make things clearer for you. Note: one can get a flipped version of a tile with the appropriate method: getFlipped().

## 3 SaboteurMove

To send a move, you must generate a SaboteurMove object. For an agent it should be constructed with:

1. a Saboteur card (an instance of the card object you wish to play).
2. A x integer:
   (a) if the card is a tile: the x position (row) in the SaboteurCard[] board.
   (b) If the card is a map: the x position (row) in the SaboteurCard[] board of the objective you want to look.
   (c) If the card is a destroy: the x position (row) in the SaboteurCard[] board of the tile you want to destroy (can't be the entrance or an objective).
   (d) If the card is a SaboteurDrop: the index of the card in your hand, which you are willing to drop.
   (e) If the card is a bonus or a malus: 0
3. A y integer:
   (a) if the card is a tile: the y position (column) in the SaboteurCard[] board.
   (b) If the card is a map: the y position (column) in the SaboteurCard[] board of the objective you want to look.
   (c) If the card is a destroy: the y position (column) in the SaboteurCard[] board of the tile you want to destroy (can't be the entrance or an objective).
   (d) If the card is a SaboteurDrop: 0.
   (e) If the card is a bonus or a malus: 0
4. You player id (int).

Some examples:

- SaboteurMove((new SaboteurTile("5")).getFlipped(),10,10,id)
- SaboteurMove(new SaboteurTile("8"),7,8,id)

- SaboteurMove((new SaboteurBonus()),0,0,id)
- SaboteurMove(new SaboteurMap(),12,3,id)
- SaboteurMove(new SaboteurDrop(),4,0,id)

If you use an illegal move, you will directly loose the game, so make sure that your moves are legal!

# 4 Workflow: Eclipse

We strongly recommend that you develop your agent using Eclipse. If you use something else, you will need to figure out how to compile and run the project.

The root directory of the project package is a valid Eclipse project. There are two ways to get it running, based on the following clicks: - From GitHub: **File → Import → Git → Projects from Git → Clone URI** → put in field URI https://github. con/PierreOrhan/SaboteurComp424 → keep pressing **Next** until the end!

From source: **File → Import → General → Existing projects into workspace** and then select the root of the project package.

You may need to set the launch configurations manually as follows: File → Import → Run/Debug → Launch configurations and then navigate to the eclipse directory and select those launches.

Note: GitHub is a great tool for managing your project. However, if you choose to use it, **please set the project access to private** - this is free for students.
Another great IDE freely available for student is Intellij-Idea, which we also recommend.

# 5 Quick Start Saboteur with Eclipse

If using Eclipse and you've set it up as described above, you can easily start a game. Simply click **Run** in Eclipse, then click on GUI. This will launch the GUI for you. Then, click the launch tab in the GUI, and select **Launch Server**. Then, the first client you launch will be the first player; for example, select Launch Human Player next. Then, to set the second player, select the next client class; for example, select **Launch Client (Saboteur.RandomSaboteurPlayer)**. You can now play against a random player in the GUI to get a feel for the game. Changing the Clients you select will determine who/what plays who/what - so you can run two humans against each other too, and similarly two agents against each other.

# 6 Playing Games

Here we provide documentation for the server and client programs. The commands outlined in this section are the commands that are called by the provided build.xml file (if using ant) and launch configurations (if using Eclipse). The details provided in this section are especially useful for advanced use of the provided code, such as playing games where the clients are located on different machines than the server. All of these commands assume that you have compiled the code and stored the class files in a directory named bin located inside the root directory of the project package (Eclipse will automatically do these things).

In a nutshell, if you want to run from the command line, you will need three terminal shells open and simultaneously running: one to launch the server, one to launch a Client for the first player, and one to launch a Client for the second player.

## 6.1 Launching the Server

To start the server from the root folder of the project package, run the command: java -cp bin boardgame. Server [-p port] [-ng] [-q] [-t n] [-ft n] [-k] where:

- (-p) port sets the TCP port to listen on. (default=8123)
- (-n g) suppresses display of the GUI
- (-q) indicates not to dump log to console.
- (-t n) sets the timeout to n milliseconds. (default=2000)
- (-ft n) sets the first move timeout to n milliseconds. (default=30000)
- (-k) launch a new server every time a game ends (used to run multiple games without the GUI)

For example, assuming the current directory is the root directory of the project package, the command: java -cp bin boardgame. Server -p 8123 - - 300000
launches a server on port 8123 (the default TCP port) with the GUI displayed and a timeout of 300 seconds. The server waits for two clients to connect. Closing the GUI window will not terminate the server; the server exits once the game is finished. If the -k arg was passed, then a new server starts up and waits for connections as soon as the previous one exits. Log files for each game are automatically written to the logs subdirectory. The log file for a game contains a list of all moves, names of the two players that participated, and other parameters. The server also maintains a file, outcomes.txt, which stores a summary of all game results. At present this consists of the integer game sequence number, the name of each player, the name of the winning player, the number of moves, and the name of the log file.

### 6.2 Launching a Client

As stated previously, the server waits for two client players to connect before starting the game. If using the GUI, one can launch clients (which will run on the same machine as the server) from the Launch menu. This starts a regular client running in a background thread, which plays using the selected player class. In order to play a game of Saboteur using the GUI, choose Launch human player from the Launch menu (as described in Section ]4 ).

Clients can also be launched from the command line. From the root directory of the project package, run the command:
java -cp bin boardgame. cılient [playerclass [serverName [serverPort]]]
where:

- playerClass is the player to be run (default=Saboteur.RandomSaboteurPlayer)
- serverName is the server address (default=localhost)
- serverPort is the port number (default=8123)

For example, the command:
Java - cp bin boardgame.Client Saboteur.RandomSaboteurPlayer localhost 8123
launches a client containing the random Saboteur player, connecting to a server on the local machine using the default TCP port. The game starts immediately once two clients are connected.

The two approaches of launching players from the GUI and launching players from the command line can also be combined. For instance, one can use the GUI to manually play against the random player (or any other agent) by first launching a human player using the GUI, and subsequently launching the random player, either by selecting it in the GUI or running the appropriate command from the command line. The order can also be switched; the player that connects to the server first will move first.

## 7 Autoplay

The provided Autoplay script can be used to play a large batch of games between two agents automatically. It launches a Server with the options - **k** - **ng**, and then repeatedly launches pairs of agents to play against one another. To use Autoplay, run the following command from the root directory of the project package:
Java -cp bin autoplay.Autoplay n_games
where **n_games** is a positive integer number of games to play. The default behaviour of Autoplay is to play the student agent against the random player, with the random player agent going first every second game. You can modify this behaviour by editing **Autoplay.java.**.

## 8 Implementing a Player

New agents are created by extending the class Saboteur.SaboteurPlayer. The skeleton for a new agent is provided by the class student_player.Student Player, and you should proceed by directly modifying that file. In implementing your agent, you have two primary responsibilities:

1. Change the constructor of the **StudentPlayer** class so that it calls **super** with your student number (use the ID of one student of your group).
2. Change the code in the method **chooseMove** to implement your agent's strategy for choosing moves (the real work).

To launch your agent from the command line, run the command:
Java -cp bin boardgame.Client student_player. StudentPlayer
You can also launch your agent by selecting it from the Launch menu in the GUI. The project specification has further details on implementing your player, and, in particular, what you need to submit.