Lab 7

[valid 2020-2021]

Concurrency
Write a program that simulates the following positional game between a given number of players (TSP Game).

At the beginning of the game the *board* contains n *tokens*, each token containing a distinct pair of numbers between 1 and n, and a specific positive value.
Each player extracts tokens successively from the board and must create with them a *a closed sequence*, that is *t1=(i1,i2), t2=(i2,i3),...,tk=(ik,i1)*, where *ti* are tokens.
The value of a sequences may be evaluated in various modes: the sum of the token values, the number of tokens, etc. A special bonus may be given if the length of the sequence is n.
The game ends when all tokens have been removed from the board and each player receives a number of points equal to the the value of its largest sequence, or equal to the sum of its sequences, etc.
The players might take turns (or not...) and a time limit might be imposed (or not...).

The main specifications of the application are:

Compulsory (1p)

- Create classes in order to model the problem. You may assume that all possible tokens are initially available, having random values.
- Each player will have a name and they must perform in a concurrent manner, extracting repeatedly tokens from the board.
- **Simulate the game using a thread for each player**.
  Pay attention to the *synchronization* of the threads when extracting tokens from the board.

Optional (2p)

- Implement the scoring and determine who the winner is at the end of the game.
- Make sure that players wait their turns, using a *wait-notify* approach.
- Consider the situation when each player might have a different strategy for extracting a number: automated (random) or manual.

A manual player will use the keyboard, while the bot will extract a random token. Simulate bot contests on large graphs.

- Implement a *timekeeper* thread that runs concurrently with the player threads, as a *daemon*. This thread will display the running time of the game and it will stop the game if it exceeds a certain time limit.

Bonus (2p+)

- Implement a "smart" player. This should try to create valuable sequences, while not allowing others to extend theirs.
- Implement various scoring strategies (in a flexible manner), for example the winner could be the one that determines the most valuable hamiltonian circuit.
- Create a simple graphical user interface for the game, using JavaFX.

References

- Slides
- The Java Tutorials: Concurrency
- Java Language Specification: Threads and Locks
- Java Concurrency / Multithreading Tutorial

Objectives

- Understand the basic principles of *concurrent programming*.
- Create and run threads using the *Thread* class and the *Runnable* interface.
- Understand *resource contention* and *thread interference*.
- Create *synchronized* methods or statements.
- Implement the *wait-notify* mechanism.
- Understand the *thread pool* and *fork/join* concepts.