



Plagiarism detection in game-playing software

Paolo Ciancarini

Gian Piero Favini

University of Bologna, Italy

4th ACM Int. Conf. on the Foundations of Digital Games

Florida, USA, April 2009

The problem

- Handling the issue of plagiarism in computer game tournaments and other agonistic events – for example, a Chess tournament for computers.
- Plagiarism accusations arise quite frequently during or after tournaments.

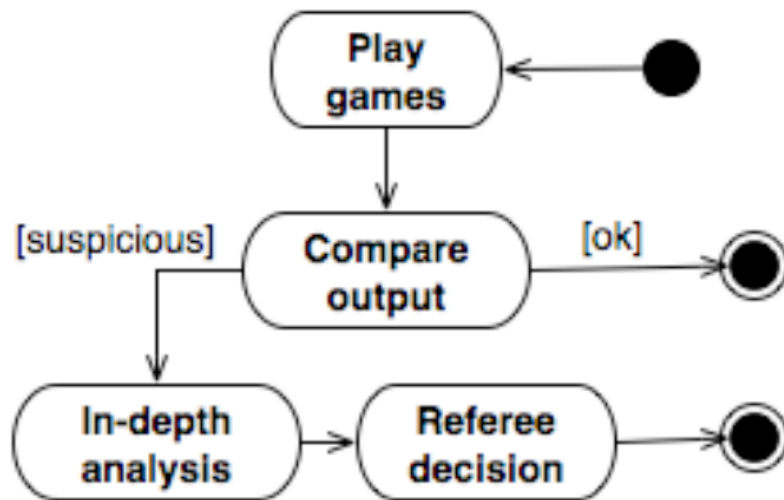
Motivations

- **Ethical reasons:** plagiarism is cheating
- **Legal reasons:** plagiarism may break intellectual property or copyright laws
- **Organizational reasons:** plagiarism accusations create public controversy and disrupt the normal schedule of the event
- **Marketing reasons:** these tournaments and the titles they award are a big factor in determining the popularity and commercial success of a playing program

Issues to consider

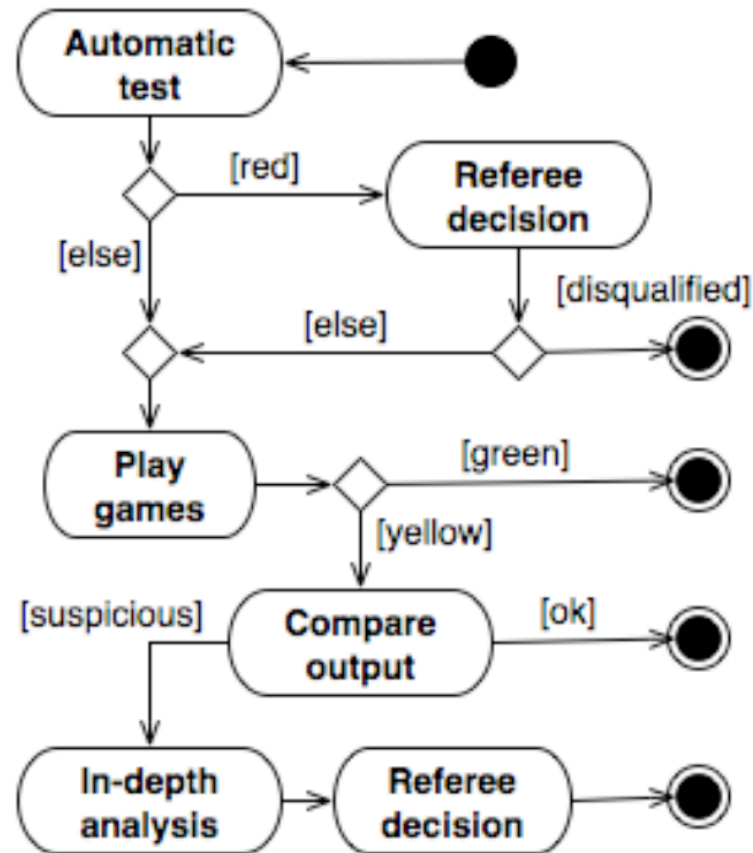
- Basing accusations on output alone may generate false positives and negatives (all programs are playing the same game).
- Manual verification of cloning for all pairs of participating programs is unfeasible
- Organizers want to maintain an atmosphere of friendliness and trust – no witch hunts.

How it usually works for Chess



- Similarities are found by running popular programs on the same positions.
- The incriminated program's competitors are often the ones running these tests.
- If similarities are found, the program is reported and somehow a decision is made.

Our approach (1)



- A possible answer involves using a suite of plagiarism detection tools generally applied to **academic** settings (students copying their assignments)
- This automatic code analysis is the first filter

Our approach (2)

- Programs enter a “pool” consisting of all participants plus a set of “originals” (e.g. open source programs).
- All program pairs are tested automatically.
- Automatic analysis has two thresholds: **red** and **yellow**.
- Red calls for immediate human examination and usually results in disqualification.
- Yellow marks a pair of programs as similar (unknown to operator/s); their outputs will be compared by the staff

Notes

- Automatic analysis should be quantitative (e.g. amount of similar lines of code, tokens, etc.)
- It should be very accurate for “**red zone**” programs - no false positives.
- It should help a human expert to evaluate “**yellow zone**” programs.
- Implemented as a modified version of the SIM plagiarism detection tool www.cs.vu.nl/~dick/sim.html
- Changes are geared towards game software - for example, stripping arrays from chess programs (all chess programs use big arrays).

Evaluation

- “**Red zone**” programs have 40% similar tokens (programming language words).
- “**Yellow zone**” programs have 10% similar tokens.
- The SIM sw helps a judge to evaluate the yellow zone programs by providing a series of potentially similar areas. The judge only has to determine whether the sections are actually similar and establish **where** and **how obviously**.
- Our system computes a **similarity score** based on the size, location, and severity of similar sections.

Similarity score

- Assign categories to similar sections - not all similarities are equally significant.
- Multiple certainty levels, from “suspicion” to “certainty” of similarity.
- Each category has a weight.
- Categories: **Rules, Search, Evaluation, I/O**
- Specific games (e.g. Chess) can define subcategories (such as “opening books”).
- Compare score to threshold, and disqualify if score is higher.