Development of an autonomous agent for the game **Centipede**

Group Assignment

Inteligência Artificial

Ano Lectivo de 2025/2026

Diogo Gomes
Luís Seabra Lopes

October 5, 2025

I Important notes

1. This work must be carried out in groups of 2/3 students. In each Python module submitted, you must include a comment with the authors' name and mechanographic number.
2. A first version of the program must be submitted by 19th of November 2025. The final version must be submitted by 17th of December 2025. In both submissions, the work can be submitted beyond the deadline, but will be penalized in 5% for each additional day.
3. Each group must submit its code through the *GitHub Classroom* platform. In the final submission, include a presentation (Powerpoint type) in `.pdf` format and named `presentation.pdf`, with a maximum of five pages, where you should summarize the architecture of the developed agent.
4. The code should run at least on Python 3.11 (minimum requirement). The main module should be named `student.py`.
5. If you discuss this work with colleagues from other groups, include a comment with the name and mechanographic number of these colleagues. If you use other sources, cite them as well.
6. All code submitted must be original; though trusting that most groups will comply with this requirement, tools will be used to detect plagiarism. Students involved in plagiarism cases will have the assignment canceled.
7. The project will be evaluated taking into account: performance; quality of architecture and implementation; and originality.

II Overview

This work involves the application of concepts and techniques from three main chapters of the AI course, namely: Python programming; agent architectures; and search techniques for automated problem solving.

Within the scope of this work, you should develop an agent capable of playing intelligently the game Centipede, as popularized in the Arcades of the 80's.

Based on the original Atari arcade game Centipede from 1980, we propose the development of an AI agent capable of high scoring in this popular game. In Centipede, the player controls a small shooter (often called the "bug blaster") that moves horizontally along the bottom of the screen. The goal is to destroy a centipede that winds its way down through a field of mushrooms. Each time the centipede is shot, it breaks into two smaller centipedes and a mushroom appears in the position where the centipede was hit. Other insects like spiders, fleas, and scorpions appear, each with different movement patterns and effects on the playfield. Players must dodge and shoot to survive as the centipede and other creatures increase in speed and number, making the game progressively more challenging. The game ends when the player's shooter is hit.

In this version of Centipede we introduced some modifications, that will increasingly rise the difficulty of the game. Scoring is also handled differently. Hitting a body segment of the centipedes will score higher when the centipede is in the higher rows and will proportionaly score less in the bottom rows. Enemies such as spiders and fleas might show up during game play. By killing a spider or a flea, the agent will score a certain number of points (see below).

The game will stop when: all centipedes are dead; or the player dies; or the game reaches 3600 steps.

1 Objectives

- To obtain a positive mark, the agent must be able to reach at least 1000 points.
- Score as much as possible. (see below details on marking)

III Game Rules

- *Bug Blaster* starts at the bottom of the screen while at least 1 centipede starts at the top of the screen.
- The agent controls a cursor, through which he can move the Bug Blaster either vertically or horizontally (diagonally is not allowed). The bug blaster can shoot blast towards the top of the screen only.
- In *Centipede*, you can use the commands **"w"** (*move up*), **"s"** (*move down*), **"a"** (*move left*) and **"d"** (*move right*). **"A"** is used to shoot blasts.
- A Centipede game is composed of a single map. Each game can have a different map.
  - Mushroom destroyed: 1 point.
  - Centipede shot: 100 points, if at the top most row; in other rows, the agent will score 100 points minus the distance to the top row.
  - Spider killed: 1000 points.
  - Flea killed: 200 points.

IV Code and Development Support

A *Centipede* game engine written in Python is available at `https://github.com/dgomes/ia-centipede`.

All game entities are represented by classes.

Each group develops an agent creating a client that implements the protocol exemplified in the `client.py` file. No modifications to other files are necessary (you can't change `game.py`), but you can create new files, folders, etc.

If you implement a new feature or implement any improvement to the game engine and/or viewer, you can create a "Pull Request"(PR) on the GitHub platform. If your change is accepted, you will be credited with a bonus on the final evaluation up to a maximum of 1 point (in 20).

The developed agent must be delivered in a module named `student.py` and the agent should connect to the local game server, using as *username* the mechanographic number of one of the elements of the group (any). You should not hardcode this information, and should follow the example in `client.py` file, that uses environment variables.

There is a support channel in `https://detiuaveiro.slack.com/messages/ai/` where students can ask questions and receive notifications of changes.

Given the novelty of the game engine, it is expected that there are some bugs and tweaks during the course of work. Be on the watch out for server modifications (configure the professor repository as an upstream repository and fetch/merge often) and notifications in *e-mail*, *Slack* and *e-learning*.

To start the work, you must form a group with colleagues and access the link `https://classroom.github.com/a/63eybip2` which will *fork* the code for the group. Only one *fork* should be done per group. One of the elements of the group creates the group's repository and associates the other elements. After this step, the *fork* will be created automatically (do not create a new *fork* without all elements being registered).

V Recommendations

1. Start by studying `client.py`. The code is very basic and simple, so start by *refactoring* the client to something more oriented to an autonomous AI Agent.
2. Periodically `git fetch` from the original repository to update your code.
3. Run `git log` to keep track of small changes that have been made.
4. Follow the channel #ai on Slack

VI Clarification of doubts

Clarifications on the main doubts that may arise during the performance of the work will be placed here.

1. **Question**: How will the performance of agents be evaluated?

   Answer: Agents will be evaluated on their performance in a set of games based on the sum of the final scores in these games. The final score in a game is the one the agent has at the time of *gameover*.

2. **Question**: How will the practical work be evaluated?

   Answer: Total game scores for each submitted agent are mapped to marks taking into acount the distribution of total scores. A total score of 1000 is mapped to 10/20. The median of the total scores is mapped to 16/20. The maximum total score is mapped to 20/20. Other total scores are mapped linearly. The games will be played in a computer with 2 Cores at 2.0Ghz and 4Gb RAM without any viewer running.

   1st delivery evaluation

   - In this delivery, only the agent code must be submitted.
   - Each agent will play 10 games and the average of these 10 games will be obtained.

   2nd delivery evaluation

   - Each agent will play 10 games and the average of these 10 games will be obtained.
   - In this delivery it is necessary to submit a presentation (see point I.3 above).
   - The evaluation of the second delivery reflects the agent's performance (90%), the design quality (according to the delivered presentation, 7.5%) and the quality of the implementation (2.5%).