# Background and context for the project

From the close relationship between IBM and the Columbia University, one of the first academic-credit courses in computing emerged in 1946.

[https://www.ibm.com/ibm/history/ibm100/us/en/icons/compsci/#:~:text=The%20first%20computer%20science%20departments,the%20pioneers%20in%20the%20field.]

From then, it had been about 15 years to the establishment of the first Department of Computer Sciences in the United States at Purdue University in October 1962.

[https://www.cs.purdue.edu/history/#:~:text=INTRODUCTION,natural%20phases%20in%20its%20history.]

The number of students taking Computer Science modules has become a growing trend and, according to TopUniversities.com,

[https://www.topuniversities.com/university-rankings-articles/university-subject-rankings/university-subject-rankings-top-ten-most-viewed-subjects]

computer science was the second most-loved subject in 2019; in spite of this form of appreciation, this subject has often seen itself on the highest ranks of the most dropped university degrees. A study conducted from the Higher Education Statistics Agency (HESA) in 2016/2017 [https://www.hesa.ac.uk/data-and-analysis/performance-indicators/non-continuation-1819] identified computer science as the most dropped of the year with a 9.8% rate of dropping.

Among the factors contributing to this high rate, it should be mentioned the difficulty of many students in learning concepts and notions of mathematics, data structures and algorithms, which together constitute a large portion of the subject foundation.

# Review of any relevant literature, including web sites

# Requirements

2. Requirements

The program comes with a total of three pages, intended to be used by two types of users:

* **Student**: the only page dedicated to this user is the one to play and interact with the visualisation of the algorithm or data structure;
* **Teacher**: this is the person who will configure the algorithm or the data structure. Two pages are dedicated to this user, the instructions page and the one to create the configurations.

Student-User requirements

This user should be able to interact with the program via a basic graphic interface and do the following:

- provide inputs which depend on the algorithm or data structure specified by the configuration file to efficiently interact with the program;

- be able to run a configuration importing a JSON configuration file – JSON understanding is not necessary;

- depending on teacher-user who provided the configuration file, the student may need to understand a specific programming language or pseudocode.

Teacher-User requirements

This user is provided with exhaustive information on how to interact with the software, however, they should be capable of:

* Javascript knowledge is required as the language used to specify the code to run by the program.
* The user should understand Javascript asynchronicity to grasp how the program will produce the animations. The user is expected to use the generator functions keyword “yield”, so being able to code using yield syntax notation is required, but generators knowledge is not.
* Solid control structures understanding is recommended to generate better visual output.
* Solid knowledge of data structures and algorithms is required, but very depending on the type of code the user intend to generate.

1. The user should be able to interact with the simulator using a basic graphic interface and do the following: (a) The user must be able to run and stop dialogue simulations. (b) The user must be able to set the properties for each dialogue, namely: proponent type, opponent type, number of runs, and a problem file; a problem file contains the argumentation framework to be used in the dialogue, the arguments that the proponent may use, and the opponent model of the proponent, equivalent to that in Planning for Persuasion [6] (c) The user must be able to save and load dialogue settings to and from a json file. 15 (d) The user must be able to create random problem files that can be used in dialogues. (e) The use must be able to select random opponent model for an existing problem file. 2. The user should be able to run dialogues form the command line without having to use the graphic interface. The settings from the run should be provided as a json file, that can be easily generated using the graphic interface.

3.2 System Requirements Agents

1. There must be proponent agents and opponent agents, with at least two different strategies for each. 2. There must be at least one type of opponent agent that can derive arguments given a closure function. A closure function is a relation from a set of arguments asserted by the agents to a subset of the arguments in the argumentation framework that have not been asserted yet. It means that if there is a closure function from a to b, once a is asserted, the opponent may be able to derive b, depending on their type. 3. Agents must be able to participate in dialogues and assert arguments when needed. 4. Opponent agents must be able to reset their state so they can compete against several proponents. As the state of some agents is randomly initialised, having the ability to reset their state is essential to ensure a fair comparison how do different proponents types perform against different opponent types, as it guarantees they all competed against the same agents. Simulator 1. The simulator must be able to receive as input a problem file in a certain format that it will be able to parse, and extract an argumentation framework (arguments and attacks), a set of arguments for the proponent, opponent models and their probabilities, dialogue goal, a and a closure operation if exists. 2. The simulator must be able to run a planner as a sub-process, provide it with the required input to generate an optimal simple strategy, and parse the results. 3. The simulator must be able to edit the planner input files for two different cases: 16 (a) To generate a new strategy that takes into account arguments that have already been asserted, by both the proponent and the opponent, at a certain state in a dialogue (b) To change the dialogue goal and allow to generate contradicting strategies. In other words, to generate a strategy to contradict a certain argument, rather than to prove it. 4. The simulator must run dialogues according to a given input, indicate the progress in the graphical interface, and save the results to a report. The results must be clear and capture all information that can be used for analysis. 5. For each dialogue, the report must record: (a) The proponent and opponent types. (b) The arguments asserted by the proponent. (c) The arguments asserted by the opponent. (d) The strategy or strategies of the proponent if applicable, i.e., the list of arguments found by the planner to be the optimal simple plan. (e) The search time from the planner. (f) The probability for guaranteed success if applicable (g) Whether or not the proponent has achieved their dialogue goal. 6. The simulator must be able to use a planner to search for strategies, and as such, must be able to create and edit problems files that can be used as input to the planner.

# Specification

# Design