Project No. 1: Normal Season

Artificial Intelligence - Setúbal School of Technology 2023/2024

Prof Joaquim Filipe Eng
Filipe Mariano

1. Horse Game: General Description

The Horse Game is a variant of the mathematical problem known as the Horse Walk, the aim of which is to visit all the squares on a chessboard using the horse's movements. This version will take place on a board with 10 rows and 10 columns (10x10), where each square has a score. This section aims to give a general overview of what this game is, leaving the next section to explain what is intended to be developed in the Artificial Intelligence project concerning solving a problem in this context by searching in State Space.

	A	B	C	D	E	F	G	H	I	J	
1	92	52	04	33	15	20	30	23	82	94	1
2	18	51	22	08	35	59	69	61	36	49	2
3	91	12	74	93	11	05	19	75	28	21	3
4	26	88	50	99	37	64	47	97	98	62	4
5	44	83	54	41	43	76	67	09	96	79	5
6	32	06	34	48	27	16	85	68	60	07	6
7	55	87	72	58	81	40	17	73	38	95	7
8	00	29	70	42	13	10	80	78	14	24	8
9	56	01	71	39	84	77	03	25	63	66	9
10	65	90	89	45	46	02	57	31	86	53	10
	A	B	C	D	E	F	G	H	I	J	

Figure 1: Example of a starter board.

1.1. Board

The game has the following features:

- The board is 10x10 and the values of each square are between 00 and 99, without repetition.
- Each time a game is started, a new board is built with the value of the squares distributed randomly.

- The aim of the game is to accumulate more points than your opponent using a chess knight. Each player has a knight of their own colour (white or black).

1.2. Gameplay

- The game begins by placing the white knight on a square in the **1st row (A1-J1 on the board)**. This square is chosen by the player with the white knight.
- If the chosen box has a number with two different digits, for example 57, then in
  As a result, the symmetrical number 75 is erased from the board, making this square inaccessible for the rest of the game. In other words, no knight can finish another move on that square.
- If a knight is placed on a square with a "double" number, for example 66, then any other double number can be removed and the player must choose which one according to their strategy (by *default* remove the one with the highest value). Once a player has left a square to move to another, the square they were on also becomes inaccessible to the game, and the square number is deleted.
- After the first move (placing the white knight), the opponent's move is followed by placing the black knight on a square on the **10th row (A10-J10)** of the board, a square chosen by the 2nd player. player. The symmetrical number of the corresponding square is also erased.
- After both knights have been placed, all subsequent moves are made by a knight move (using the traditional chess rules for the knight). **A knight does not**
  can jump to an empty square (without a number) and cannot do so to a square that is threatened by the opponent's knight.
- Each player's move repeats the symmetrical or double rule.
- A player earns points for each square visited by his horse (equal to the value of the square). Points are only counted for the squares visited, not for the symmetrical or double numbers removed.

1.3. Determining the Winner

The game ends when it is not possible to move any of the horses on the board, and the winner is the player who has gained **the most** points.

2. Project Goal: Single Player

For the purposes of this project, we're going to consider the Horse Problem as a simplified version of the game mentioned above, in which the main objective is to achieve a given score for the problem in as few moves as possible, i.e. horse jumps. To do this, you have to move the white horse along the board from the starting square in successive moves until you can't make any moves or until you reach the goal.

The transformation of the game into a problem, for this phase of the project, presupposes the following differences in the rules:

- There is only one player (white horse);
- The player starts by placing the knight on a square on the first row of the board;
- The end state is reached when the horse reaches a square that allows it to obtain a score equal to or greater than the defined objective;
- If the objective cannot be achieved, the programme should inform the user that the problem has no solution;

- The objectives for problems A-F, which are provided in the annex and have to be solved as part of this project, are: A: 70, B: 60, C:270, D:600, E: 300, F:2000;
- The initialisation of the problem-solving process consists of applying a special operator to place the horse in a box on the first row that has a numerical score. This operator makes it possible to generate the successors of level 1 from the root node of the graph that represents each of the problems mentioned above. From there, the horse movement operators apply.

The aim is for students to write and test a programme, in LISP, to display the sequence of states or moves that lead from the initial position of the problem to the final position, using the search algorithms taught in class, as explained in detail below. The optimal solution is the path with the fewest moves between the initial state and the final state.

3. Problem formulation

3.1. Board

The board is represented as a list made up of 10 other lists, each containing 10 atoms. Each of the lists represents a row on the board, while each of the atoms has the value of its respective square. In other words, the board is represented by a list of lists in LISP, with each atom representing a square where the row corresponds to the index of the sub-list and the column to the index of the atom within the row. **The squares that have already been visited will have the value NIL, while the square where the horse is will have the value T.**

Below is a representation of the 1st board and the 2nd board in figure 2, respectively.

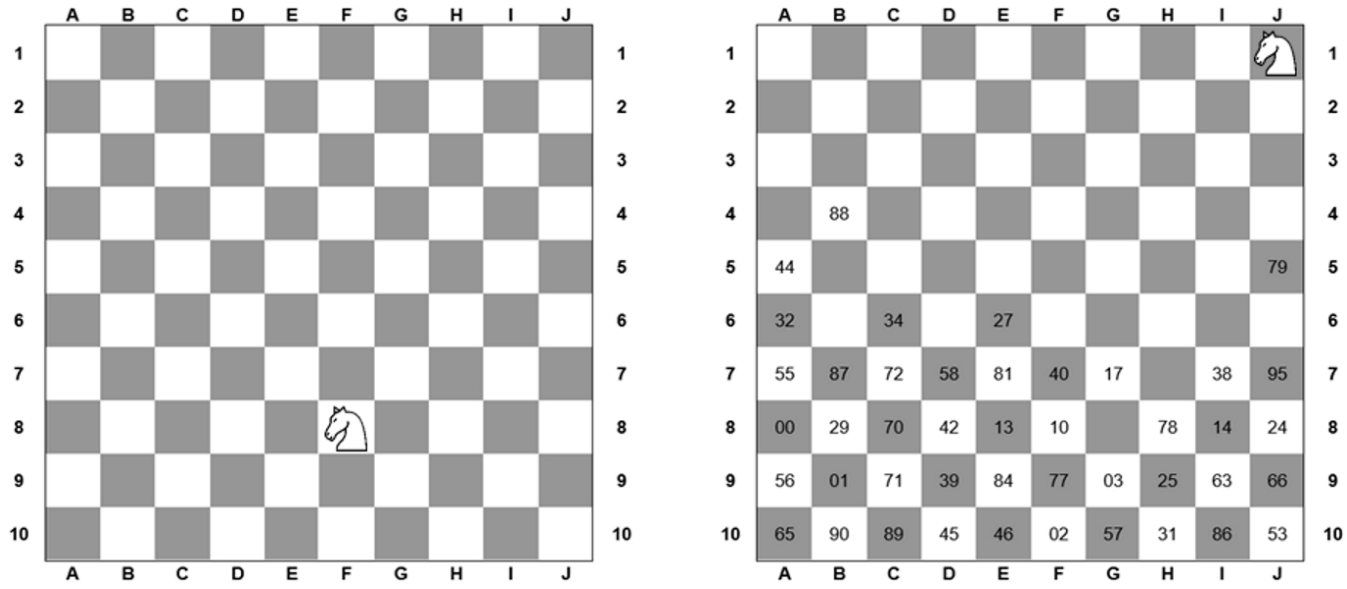


Figure 2: Examples of board representations (both positions are terminal).

```
(
 (NIL NIL NIL NIL NIL NIL NIL NIL NIL) (NIL
 NIL NIL NIL NIL NIL NIL NIL NIL) (NIL NIL
 NIL NIL NIL NIL NIL NIL NIL NIL )
```

```
 (NIL NIL NIL NIL NIL NIL NIL NIL) (NIL NIL NIL
 NIL NIL NIL NIL NIL NIL) (NIL NIL NIL NIL
 NIL NIL NIL NIL NIL NIL) (NIL NIL NIL NIL
 NIL NIL NIL NIL NIL) (NIL NIL NIL NIL NIL T
 NIL NIL NIL NIL)
 (NIL NIL NIL NIL NIL NIL NIL NIL NIL) (NIL
 NIL NIL NIL NIL NIL NIL NIL NIL)
)
```

Figure 3: Representation of the 1st board in figure 2.

```
(
 (NIL NIL NIL NIL NIL NIL NIL T)
 (NIL NIL NIL NIL NIL NIL NIL NIL) (NIL NIL
 NIL NIL NIL NIL NIL NIL NIL NIL) (NIL 88
 NIL NIL NIL NIL NIL NIL NIL NIL) (44 NIL
 NIL NIL NIL NIL NIL NIL NIL 79)
 (32 NIL 34 NIL 27 NIL NIL NIL NIL NIL) (55
 87 72 58 81 40 17 NIL 38 95)
 (0 29 70 42 13 10 NIL 78 14 24)
 (56 1 71 39 84 77 3 25 63 66)
 (65 90 89 45 46 2 57 31 86 53)
)
```

Figure 4: Representation of the 2nd board in figure 3.

3.2. Solution found

The solution can be represented by a sequence of states, from the initial state to the final state, or - for reasons of readability - by the list of operations (i.e. moves) carried out on the pieces of the board. Each move is identified by the horse's target square (a letter and a number).

3.3. Operators

The operators represent the possible moves in a given state. For the Horse Problem, the maximum number of possible moves will be 8, as long as these houses have not yet been visited or removed by the symmetric or double rule.

The operators' application costs are considered to be constant and unitary.

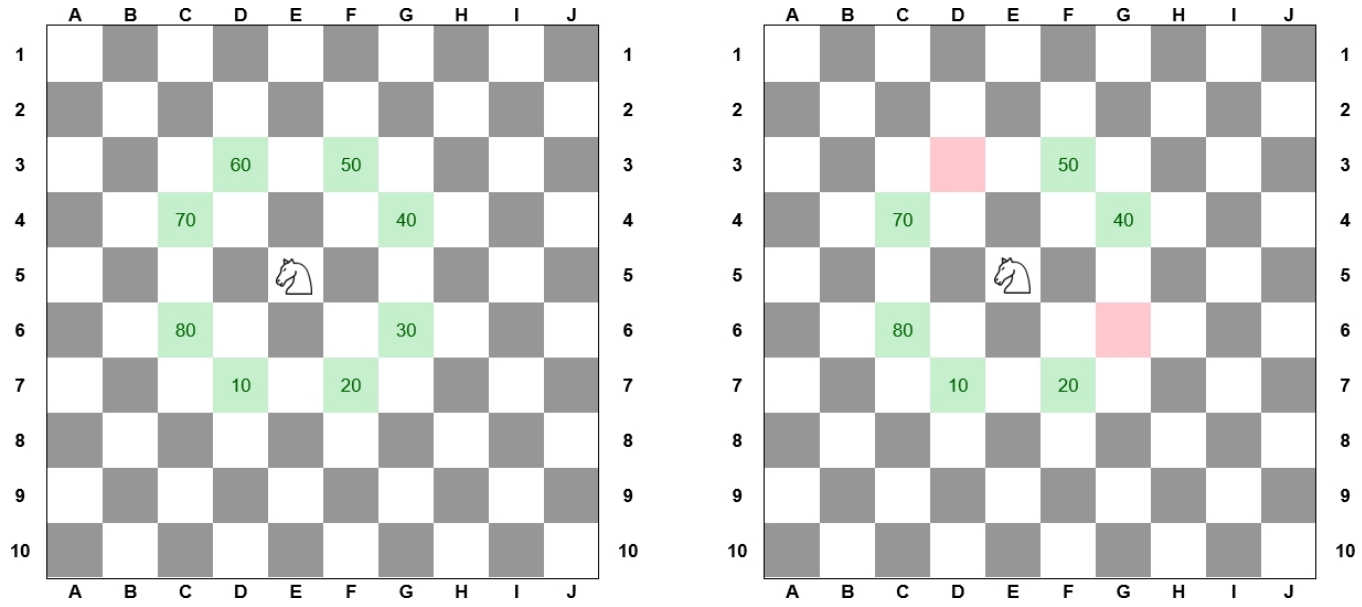


Figure 5: On the 1st board there is a situation in which all 8 possible moves can be made because the houses have not yet been visited (they have dots). On the 2nd board it's only possible to make 6 movements because two of the houses have already been visited.

3.4. Programme structure

The programme should be divided into three parts, each in a different file:

1. A part with the implementation of the search methods and the implementation of the efficiency analysis metrics, i.e. the part of the programme that is independent of the application domain;
2. Another to implement everything involved in solving the specific problem, including defining the operators and heuristics specific to the application domain;
3. And the third part for interacting with the user and writing and reading files.

The project must present a comparative study of the behaviour of the three methods: breadth-first search (BFS), depth-first search (DFS) and A*. Students will also be allowed to optionally develop other algorithms as indicated in section 7.

In the case of the reported methods, the programme should use modular heuristic functions, i.e. ones that can be added to or removed from the search programme as modules.

The heuristics should not be rigidly embedded in the search programme. The use of two heuristics is required, one provided at the end of this document and one developed by the students.

The project should include the implementation of each of the algorithms, in a modular way, allowing the user to choose any one of them, together with the respective parameters (heuristics, depth, etc.) for solving a given problem.

4. Experiences

The aim of the project is to study the performance of each algorithm and, in the case of A*, each of the proposed heuristics, for each of the problems listed in the annex, presenting the solution found and statistical data on its efficiency,
namely the number of nodes generated, the number of nodes expanded, the penetration, the average branching factor and the execution time.

Projects must submit the above data in a file produced automatically by the programme, and 0.5 points will be deducted for each problem not solved. If the
solution, but not the performance study of the heuristics, the discount is only 0.2 for each case.

5. Heuristics

We suggest using as a basic heuristic a heuristic that favours visiting the squares with the highest number of points. For a given board x:

h(x) = o(x)/m(x)

in which:
m(x) is the average per square of the points on the x board,
o(x) is the number of points left to reach the value defined as the objective.

This heuristic can be improved to more adequately reflect knowledge about the domain and thus contribute to greater efficiency of informed search algorithms. In addition to the heuristics suggested above, at least a second heuristic should be defined which should improve the performance of the search algorithms.
informed search algorithms.

6. Groups

The projects must be carried out in groups of no more than two people, but are always subject to individual oral assessment to confirm the ability to understand the algorithms and develop code in LISP.

The group can be made up of students from different classes or shifts.

7. Bonus

The project includes an optional part that allows a bonus to be awarded to students who manage to implement it.

In addition to the three search methods mentioned above, each group has the option of programming, applying and studying one or more of the following strategies: *Simplified Memory-Bounded A** (SMA*), *Iterative Deepening A** (IDA*) or *Recursive Best First Search* (RBFS).

The programming and study of optional methods is worth 1 value each, to be added to the total value of the project, with the final grade of the project being limited to a maximum of 20 values.

8. Dates

Project submission: 18 December 2023, by 23:00.
Discussion of the project: Beginning of February 2024, in conjunction with the discussion of the second project.

9. Documentation to be submitted

The project and its documentation must be submitted via Moodle, in the "1st Project Submission" event area. All the files to be handed in must be duly archived in a compressed file (ZIP with a maximum size of 5Mb), by the date indicated above. The name of the file should follow the structure <startingStudent1>_<student1 number>_<startingStudent2>_<student2 number>_P1.

9.1. Source code

The code files must be properly commented and organised as follows:

project.lisp Loads the other code files, writes and reads files, and handles interaction with the user.

puzzle.lisp Code related to the problem.

search.lisp Must contain the implementation of:

1. Width First Search Algorithm (BFS)
2. Depth First Search Algorithm (DFS)
3. Best First Search Algorithm (A*)
4. SMA*, IDA* and/or RBFS algorithms (if you choose to implement the bonus)

9.2. Exercises

There should be an exercise file, called problems.dat, containing all the board examples you want to give the user, organised sequentially, which the user must choose using a number entered into the user interface.

This number represents the order number in the sequence of examples. The file should have several lists, separated from each other by a legal separator, not a list of lists. There will be as many lists as there are problems.

In the oral, the teachers will ask to add another example, at a given position in the file, which should immediately become selectable via the user interface and be solved
normally.

9.3. Manuals

As part of the Artificial Intelligence course, the aim is for students to practise writing documents using the ***Markdown markup*** language, which is widely used for ***ReadMe*** files on ***GitHub***. In Section 4 of Lab Guide 2, you'll find all the information on the recommended structure and suggested editing tools for ***Markdown***.

In addition to delivering the code and problem files, 2 manuals (the user manual and the technical manual) must be drawn up and delivered in PDF format along with the MD sources included in the aforementioned archive.

Technical Manual:

The Technical Manual should contain the general algorithm, in parts and duly commented; a description of the objects that make up the project, including data and procedures; identification of limitations and options techniques. A critical analysis of the results of the programme's implementation should be presented, showing an understanding of the project's limitations. They should use a comparative analysis of all the

runs of the programme for each algorithm and each problem, making it possible to check the performance of each algorithm and heuristic. Finally, it should present a list of the project requirements (listed in this document) that have not been implemented.

User Manual:

The User Manual should contain an identification of the programme's objectives, together with a general description of how it works; an explanation of how to use the programme (accompanied by examples); a description of the information required and the information produced (screen/keyboard and files); the programme's limitations (from the user's point of view, of a non-technical nature).

10. Evaluation

Functionality	Values
State representation and operators	2.5
Puzzle functions	2.5
Depth and width search	2.5
Search with A* and given heuristics	2.5
Implementation of new heuristics	1.5
Solving problems a) to f)	3
Solving problem g) (given in the oral assessment)	0.5
Code quality	2
User interaction	1
Manuals (user and technical)	2
Total	20
Bonus	3

The maximum mark is 20, including the bonus.

Problems a) to f) are described in the Experiences Section, while problem g) will be provided during the oral assessment. The assessment of the project will take into account the following aspects:

- Final delivery date - There is a tolerance of 3 days in relation to the delivery deadline, with a penalty of 1 value for each day of delay. After this period, the project grade will be 0.
- Procedural correction of the project submission - (Moodle; manuals in the correct format). Procedural anomalies will result in a penalty of up to 3 marks.
- Technical quality - Objectives achieved; Correct code; Ease of reading and maintaining the programme; Correct technical options.
- Quality of documentation - Structure and content of the manuals accompanying the project.

- Oral assessment - Effectiveness and efficiency of the presentation; Understanding of the limitations and possibilities for developing the programme. At this stage there may be a total revision of the project grade.

11. Final recommendations

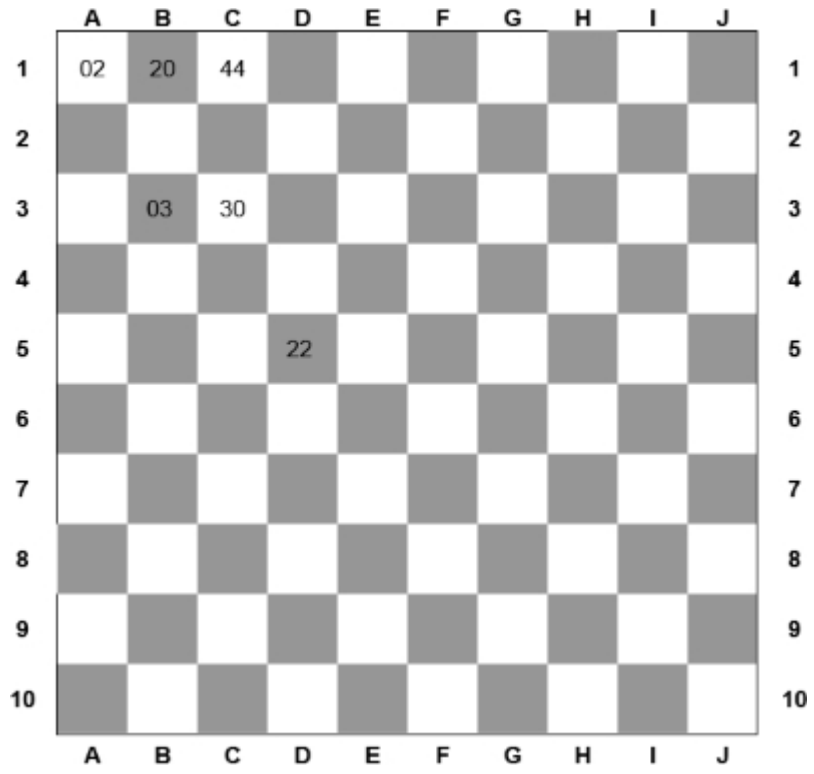
The aim of this project is to motivate the functional programming paradigm. The use of instructions of the type set, setq, setf, cycles, destructive functions or any functions with side effects is strongly discouraged since it usually denotes a low technical quality. Sequencing will only be allowed in the context of read/write functions.

The only exceptions allowed to these rules could be the use of the loop instruction to implement the main cycles of the implemented algorithms (as an alternative to a purely recursive solution), in conjunction with the global variables Open and Closed to maintain the lists of nodes.

ATTENTION: Confirmed suspicions of plagiarism will be penalised by the cancellation of both projects involved (source and target), and those responsible will be subject to disciplinary proceedings.

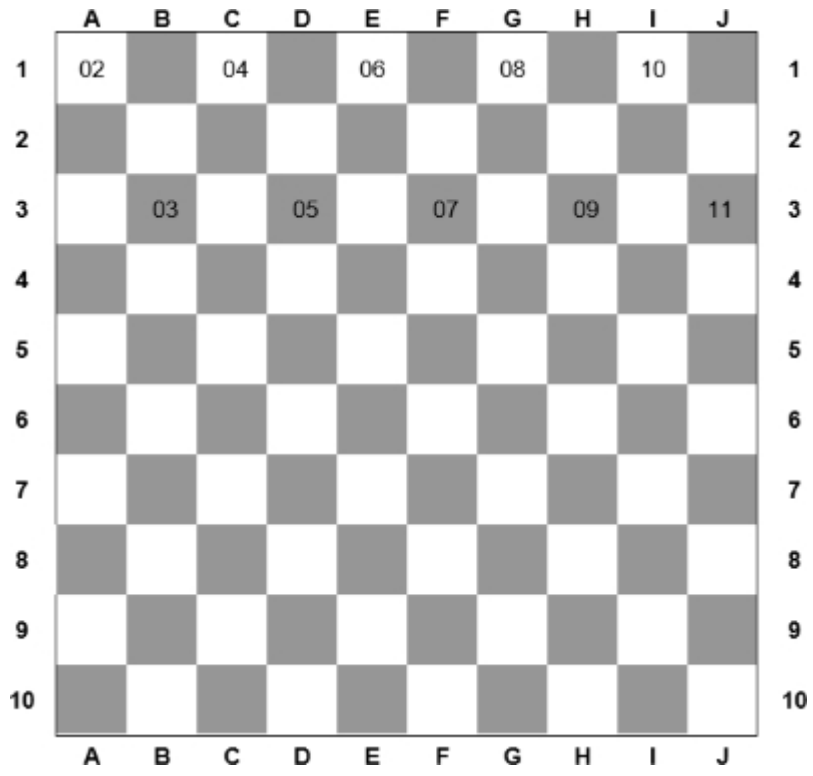
Annexes - Problems

Problem A:



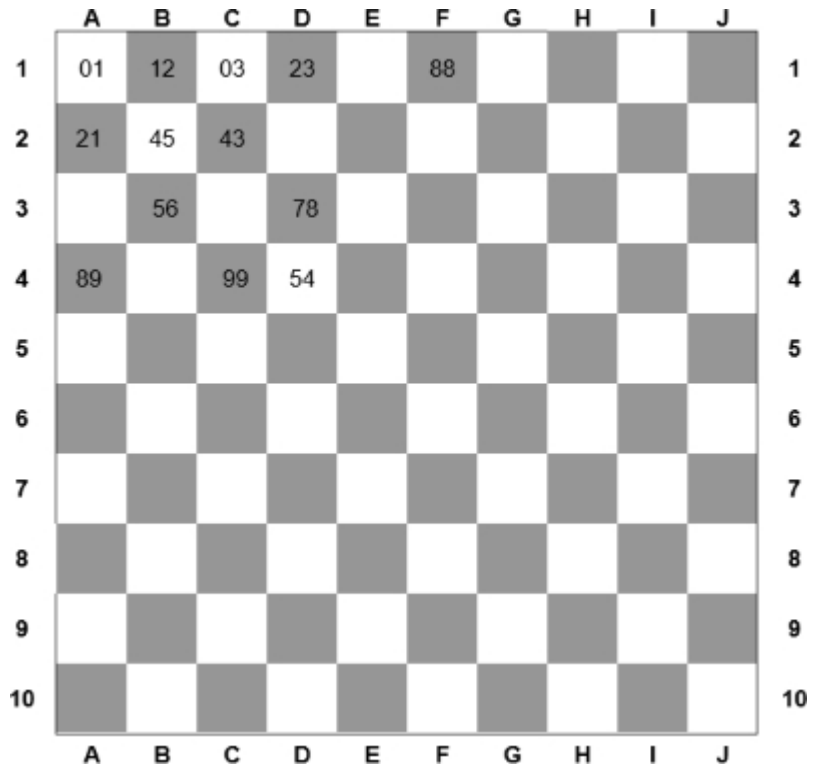
Target: 70 points

Problem B:



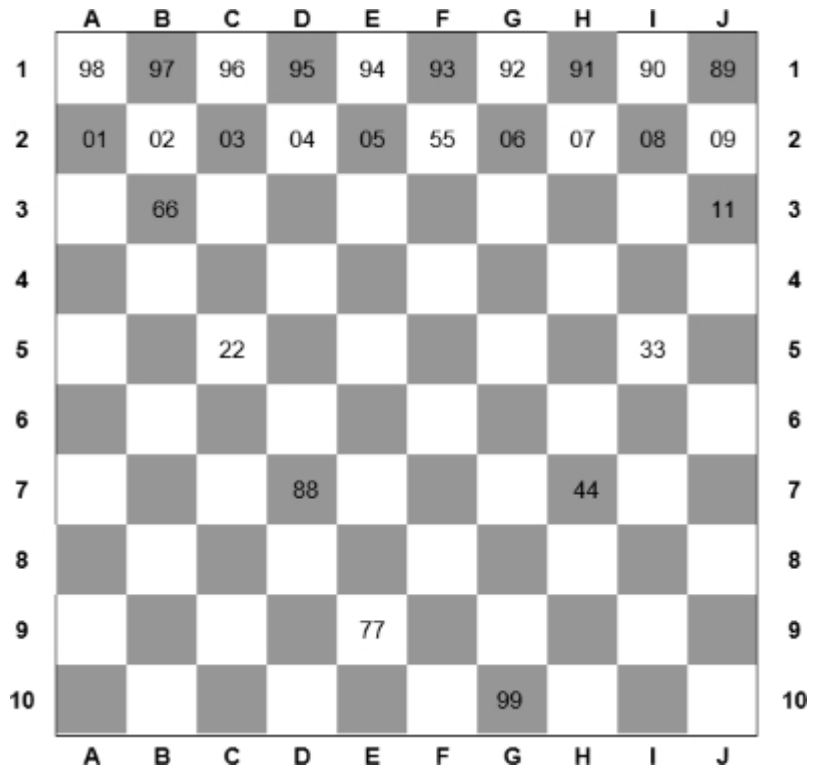
Objective: 60 points

Problem C:



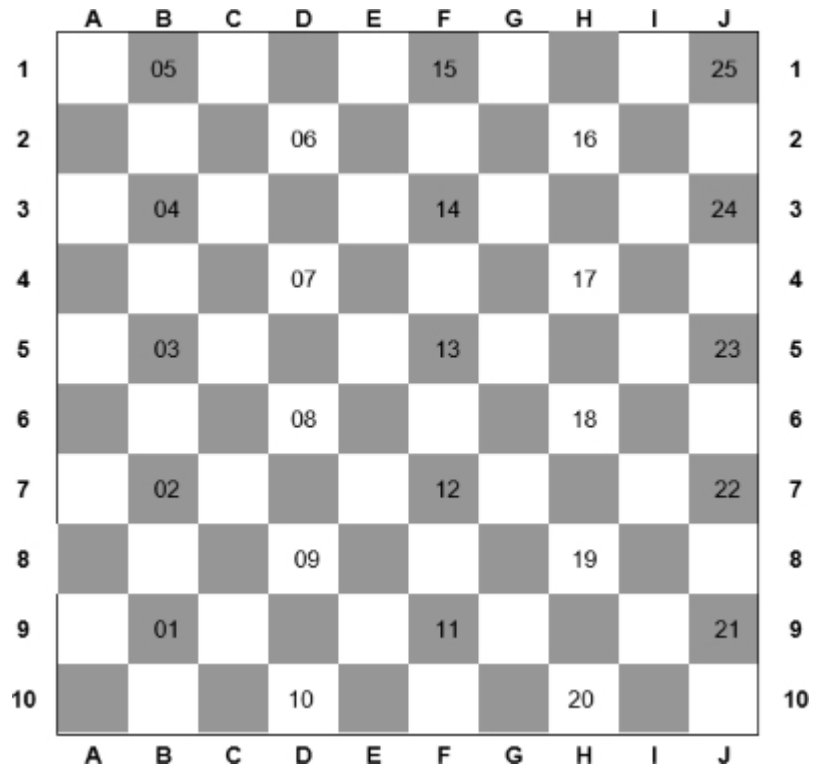
Target: 270 points

Problem D:



Target: 600 points

Problem E:



Target: 300 points

Problem F:

It should be studied for the various algorithms from a complete board generated at random and saved in the problem file.

Target: 2000 points

Problem G:

It will be made available during the discussion of the project.