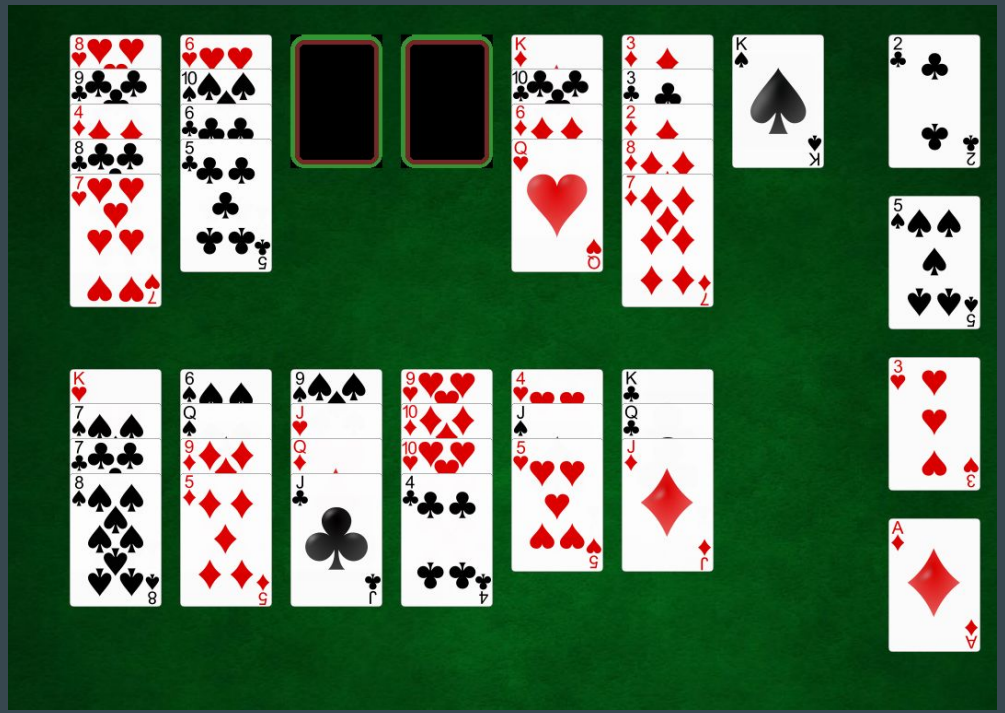
Baker's Dozen Solitaire

Group_A1_50



Made by:
Diogo Miguel Mendes de Faria
Diogo Costa Pinto
Francisco Manuel Carneiro de Moura

Project Specifications

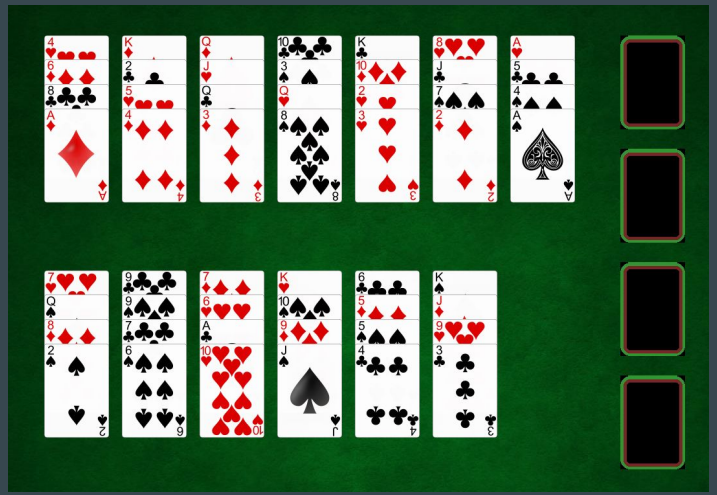
-The game board consists of the 4 foundations and 13 card columns.

-The game starts with the 4 foundations empty and the 52 cards of a standard card deck divided evenly, randomly and face-up between the 13 columns.

-Any kings are moved below all other non-king cards of their columns when the game starts.

-The game ends when the player fills the each foundation with the 13 cards of each suit in ascending order according to their rank.

-Each turn the player may move a card from the top of a column and place it on top of another column if it's placed on top of a card whose rank is immediately above the moved card's or on top of its foundation if it's placed on top of the card from the same suit whose rank is immediately below the moved card's or if the moved card is an Ace.



Formulation of the problem

In its initial state there are 52 cards evenly distributed among the 13 columns. The algorithm recognizes a game state as an objective when all 4 foundations have a King on top.

Each turn a card may be removed from its column and added to a foundation or column if it is immediately higher or lower, respectively, than the card on top of that pile.

A cost-attributing function will be called every move to evaluate the new board state's to define heuristics to help guide the informed search algorithms. Every time a play is made by the player, solver starts again taking the actual state of the board as the initial state.

Our approach to the heuristic

We have an heuristic that attributes the cost to all nodes explored based on the preview of how many moves are left to the solution. This includes the number of cards left in the foundation and the difference between the depth on a column and the and the remaining cards to go to the foundation before that card goes to the foundation. Also we attribute a random number between 0 and 1 to break ties between nodes with the same score

We use a min-heap to get the node with less estimated moves.

We also attempted to create another heuristic, which attributed a cost to the node by checking every card's distance to the last ordered card in its column and giving it a cost equal to that distance. It also gave an extra penalty to cards that were above (directly or not) another card of the same suit but smaller rank.

This heuristic presented worse consistency, times and moves than the previous one so it was discarded.

Implemented algorithms

Heuristic search algorithms

- Best first search

  Only uses the values attributed by the heuristic to determine the best nodes to explore

- A*

  Add to the heuristic estimate the distance already made to get the most optimal solutions

Uninformed search algorithms

- BFS
- DFS

We used a board with four columns and four cards for each suite to reduce complexity in order to be able to “brute force” until we found a solution

Statistics

Best-First Search (single core)

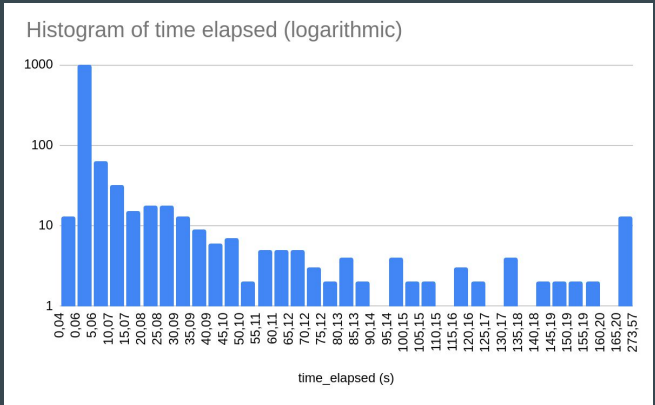
It has 83.13% of solvability for a maximum of 300 000 processed states.

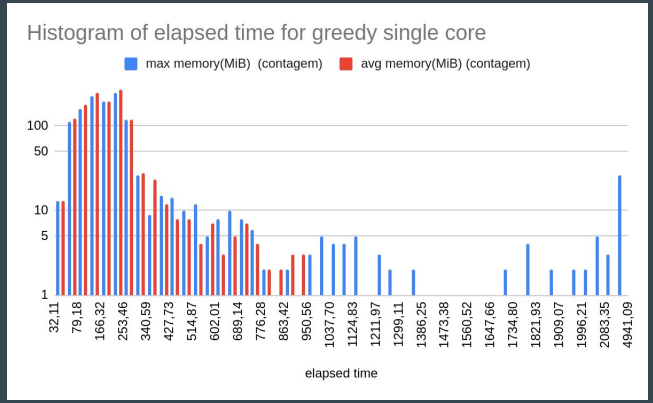
On average it takes:

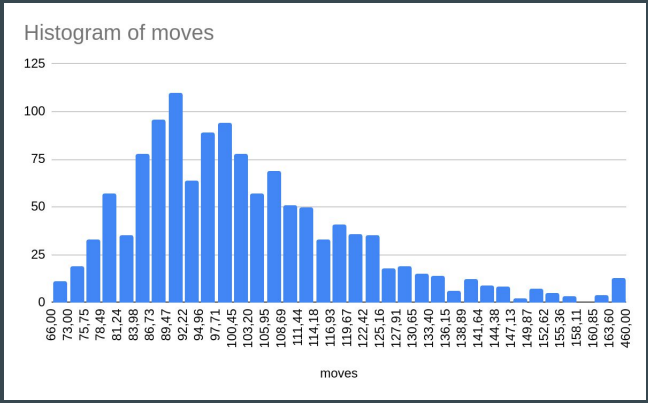
- 9.81s to find a solution
- Up to 317 MiB of memory
- 207 MiB on average during the execution of the solver
- 13516 states explored

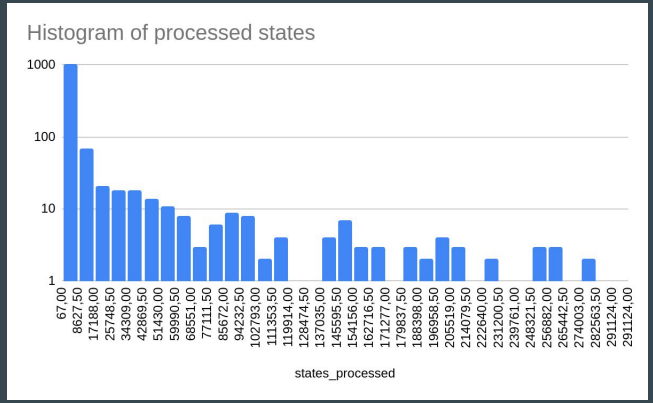
And each solution has on average 103 moves

Statistics


Histogram of time elapsed (logarithmic)
1000
100
10
1
0.04
0.06
5.06
10.07
15.07
20.08
25.08
30.09
35.09
40.09
45.10
50.10
55.11
60.11
65.12
70.12
75.12
80.13
85.13
90.14
95.14
100.15
105.15
110.15
115.16
120.16
125.17
130.17
135.18
140.18
145.19
150.19
155.19
160.20
165.20
273.57
time_elapsed (s)


Histogram of elapsed time for greedy single core
max memory(MiB) (contagem)
avg memory(MiB) (contagem)
100
50
10
5
1
32.11
79.18
166.32
253.46
340.59
427.73
514.87
602.01
689.14
776.28
863.42
950.56
1037.70
1124.83
1211.97
1299.11
1386.25
1473.38
1560.52
1647.66
1734.80
1821.93
1909.07
1996.21
2083.35
4941.09
elapsed time


Histogram of moves
125
100
75
50
25
0
66.00
73.00
75.75
78.49
81.24
83.98
86.73
89.47
92.22
94.96
97.71
100.45
103.20
105.95
108.69
111.44
114.18
116.93
119.67
122.42
125.16
127.91
130.65
133.40
136.15
138.89
141.64
144.38
147.13
149.87
152.62
155.36
158.11
160.85
163.60
460.00
moves


Histogram of processed states
1000
100
10
1
67.00
8627.50
17188.00
25748.50
34309.00
42869.50
51430.00
59990.50
68551.00
77111.50
85672.00
94232.50
102793.00
111353.50
119914.00
128474.50
137035.00
145595.50
154156.00
162716.50
171277.00
179837.50
188398.00
196958.50
205519.00
214079.50
222640.00
231200.50
239761.00
248321.50
256882.00
265442.50
274003.00
282563.50
291124.00
291124.00
states_processed

Project Progress

The project is being coded in Python using pygame.

The gamestate is being organized in its own class which contains two tuples for Columns and Foundations. Both of these contain a tuple of Cards.

The algorithms employ a tree with distinct board states as nodes.

The Best-First Search algorithm is our best algorithm, since it is capable of solving almost any board when using the multicore implementation. We tried to implement A* and an IDA* but since it retrieves optimal (or near), it needs a more refined heuristic in order to work properly. Both Best-First and A* have a multicore implementation in order to improve user experience.

Related Works

To help guide our project, we used the Baker's Dozen Solitaire wikipedia page as well as the page given to us in the assignment document.

https://www.solitaire.org/bakers-dozen/

https://en.wikipedia.org/wiki/Baker%27s_Dozen_(card_game)

https://ai.dmi.unibas.ch/papers/paul-helmert-icaps2016wshsdip.pdf