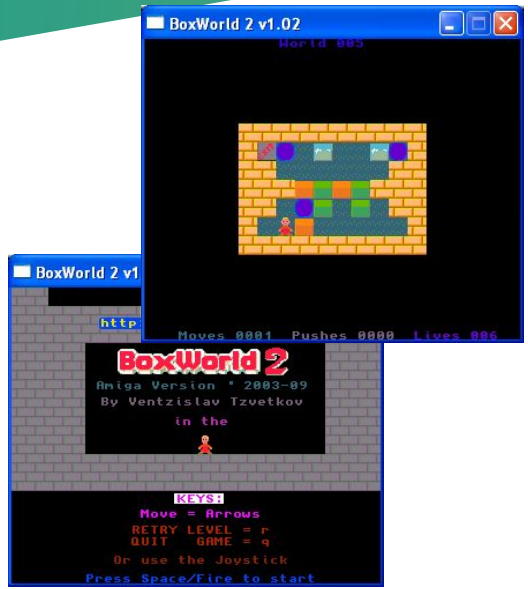
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

IART

Turma 5 - 1A
Box World 2

António Dantas - up201703878@fe.up.pt
Eduardo Macedo - up201703658@fe.up.pt
Gonçalo Pereira - up201705971@fe.up.pt

Specification

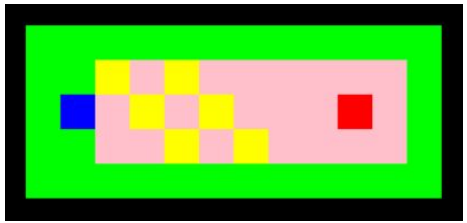


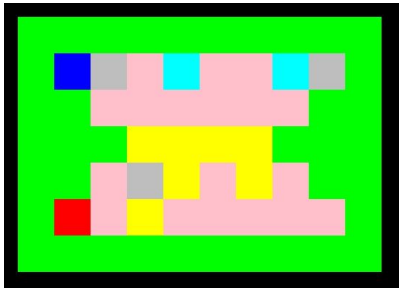
Our project consists in replicating the famous *Box Word 2* game, as well as developing an *AI* that can easily beat it. The game consists in a series of puzzles, where the player has to reach the exit of the stage in order to advance to the next level. Various obstacles are placed around, making the goal harder than it seems at first sight. At some stages, there are holes, which can be filled with the boxes placed around. There are several types of boxes - the orange and green are normal ones, which can be pushed around. The grey boxes, called *Ice Boxes*, when pushed will slide away if there is no wall, hole or another box to stop them. If the player gets stuck, he can retry the level.

The Problem

Here we try to organize the project with a search problem:
The representation of the **game state** consists of a navigable arena delimited by some borders where the player can move around. There will be Boxes that occupate those tiles and can be pushed around. Holes, that can be filled up with Boxes and then become navigable. An Exit tile that when reached by the player ends the game, and finally, a Player that roams around trying to find the Exit.
The **goal test** needs to check if the player collides with the exit tile (if their position is the same).
Our **operators** will be the player's 4 possible movements (*up*, *down*, *left* and *right*). The preconditions for each operator will depend on the movement's direction, but will always be the same. They will need to test whether the player will collide with an immovable object (game border or a hole) or if he will try to push a box. In that case it needs to test whether the box is colliding with the game border or another box. Shall these preconditions be met, the result is the movement of the player and/or the box, to the desired location. Hence altering the game state. The cost is just +1 to the move counter.
As for now, we believe the main ***heuristics*** the *AI* should comply to are the closer the player gets to the exit in each move, if there are obstacles in the way that prevent him from getting there and the minimum moves possible to beat the level.

Our Implementation





We have decided to build a very simplistic representation of the game from the ground up using Python with *JetBrains*' IDE *PyCharm*. We are also using the Pygame library for an easier implementation.

We have five working levels, some of them are displayed in the images. The red square represents the player that is trying to reach the blue square (exit). The yellow squares represent boxes that can be pushed around to achieve our goal. The light blue squares represent the Ice Boxes and the grey squares represent holes. These holes can be filled up with any type of box making it possible for the player to move to this position.

Heuristics and Operators

The heuristics that we found most interesting and effective for our project were the **Manhattan Distance between player and objective** and the **Euclidean Distance between player and finnish**.

There are four possible actions, left, right, up and down. Each of this actions occurs in the player. To make any movement, the position immediately following in the direction chosen for the movement must not be obstructed, by blocks, walls or holes.

Implemented Algorithms

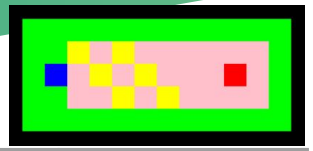
In this project we implemented some search algorithms, such as Depth-First Search, Breadth-First Search, Iterative Depth-First Search, A-Star and Greedy Algorithms.

In the Iterative Depth-First Search we start with a limit of 20 nodes to expand, following with 10 more nodes per node.

In the A-Star algorithm we used Manhattan Distance between player and objective as an heuristic, we first check the distance between the player and the box that is closest to the player and then the distance between the block and the finish line.

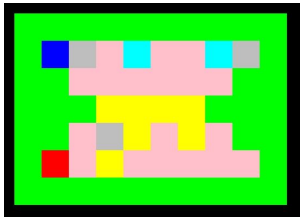
In the Greedy algorithm we used Euclidean Distance between player and finnish as an heuristic

Experimental Results



Level 1

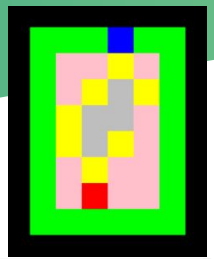
Algorithm	Visited Nodes	Moves	Time(s)
Breadth-First Search	257	**10**	0.1566
Depth-First Search	3259	16	6.7290
Iterative Depth	40	**10**	0.0349
Greedy	12	**10**	**0.0150**
A-Star	6895	12	32.8644



Level 2

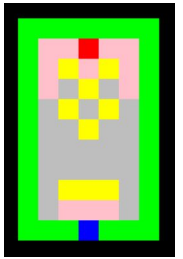
Algorithm	Visited Nodes	Moves	Time(s)
Breadth-First Search	1950	**15**	3.3151
Depth-First Search	---	---	---
Iterative Depth	1162	21	1.3743
Greedy	1021	27	1.2616
A-Star	268	35	**0.2882**

Experimental Results (cont).



Level 3

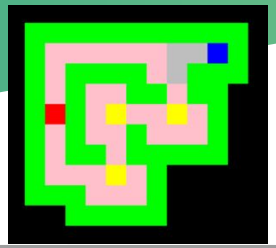
Algorithm	Visited Nodes	Moves	Time(s)
Breadth-First Search	1959	**19**	2.7746
Depth-First Search	1165	31	1.1619
Iterative Depth	617	31	0.9804
Greedy	1051	**19**	1.0223
A-Star	534	**19**	**0.6164**



Level 4

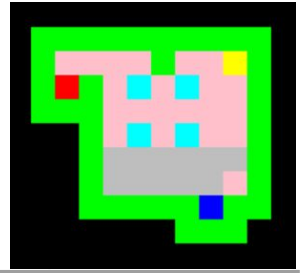
Algorithm	Visited Nodes	Moves	Time(s)
Breadth-First Search	---	---	---
Depth-First Search	2930	99	7.9776
Iterative Depth	13134	31	130.8134
Greedy	4435	39	**17.6609**
A-Star	7780	**19**	47.2529

Experimental Results (cont.)



Level 5

Algorithm	Visited Nodes	Moves	Time(s)
Breadth-First Search	13872	**91**	102.9507
Depth-First Search	11872	197	78.3463
Iterative Depth	7039	105	165.0983
Greedy	10423	133	66.9110
A-Star	4201	93	**10.7476**



Level 6 (No results returned in less than 15 min.)

Algorithm	Visited Nodes	Moves	Time(s)
Breadth-First Search	---	---	---
Depth-First Search	---	---	---
Iterative Depth	---	---	---
Greedy	---	---	---
A-Star	---	---	---

Conclusion

For different levels we obtained best times and solutions with differents algorithms (we don't have an algorithm that is "the most effective"). BFS finds always the minimal cost solution, therefore, the time to find a solution grows exponentially with the number of movements in the solution.
The bigger the arena, the more movements and states are possible, so the longer the search for the solution.
What we learned at the end of the work is that the game does not favor a single algorithm (different algorithms shine in different arenas). the greedy algorithm shone in arenas that did not imply many movements that involved moving away from the exit. A* was good in levels that imply making maneuvers with the boxes to unlock new paths.
In addition, in most cases, the iterative dfs proved to be better than the normal dfs because by setting a limit we prevented the ai from losing time in almost infinite loops of states where the solution was already impossible to obtain.
In general, the greedy's heuristic granted a good performance.
Even though we couldn't find a solution for level 4, the results were satisfactory with a variety of levels and algorithms that our AI can be tested on successfully.

References

- http://hirudov.com/others/BoxWorld2.php
- http://u.cs.biu.ac.il/~shechory/AI/lec3.PDF