Ball Sort Puzzle

Heuristic Search Methods for One Player Solitaire Games

Artificial Intelligence

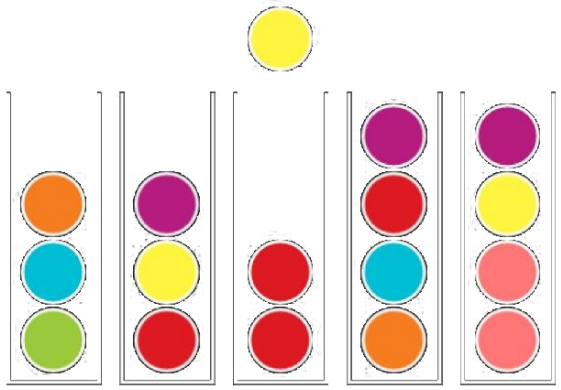
Master in Informatics and Computing Engineering

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Gonçalo Alves – up201806451
Gustavo Mendes – up201806078
Pedro Seixas – up201806227

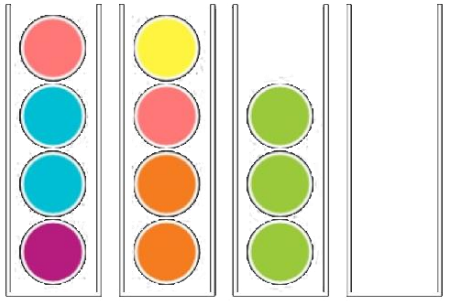
Specification

The objective of the Ball Sort Puzzle is to sort the colored balls in the tubes until all balls with the same color stay in the same tube.



The player can only move a ball at the top of a tube to:

- an empty tube;
- another tube that has a ball with the same color on top and has enough space;



The player can also undo his moves if he finds himself without moves.

Related Work

- Link to the Google Play page;
- Link of all existing levels;

We used the curricular unit's presentations to guide our implementation of the various algorithms.

Formulation of the problem as a search problem

State Representation:

- Ball: [1..∞]
- Tube: [Ball, Ball, Ball, Ball] or []
- Game: [Tube,Tube,…]

Initial State:

- Game: [Tube,Tube,…], where every Tube doesn't contain 4 equal numbers

Objective State:

- Game: [Tube,Tube,…], where every Tube either contains 4 equal numbers or is empty

Operators:

- Move(X,Y):
  - PreCond: Tube Y must be empty or have a Ball, on top, just like the one that is moved
  - Effect: move Ball from Tube X to Tube Y
  - Cost: 1 is the general case, it represents a move; C2, evaluation of the number of wrongly placed Balls, based on the Ball at the bottom of a Tube

Implemented Heuristics

We have implemented BFS, DFS, IDS, Greedy Search and A*.

Our evaluation function, used in A*, calculates the number of wrongly placed Balls (Balls with diferent color of the Ball at the bottom) in a Tube.

Code:

```python
def greedy(state: Node, a_star: bool = False, max_depth: int = 5000):
    graph = Graph()
    state.setDist(0)
    stack = [state]
    graph.new_depth()
    graph.add_node(state, 1)

    depth = 1
    while depth != max_depth and len(stack) != 0:
        if a_star:
            stack.sort()
        else:
            stack.sort(key = lambda x: x.cost)
        graph.new_depth()
        node = stack.pop(0)
        if node.gamestate.finished():
            print("Found Goal. Depth:", node.dist)
            return graph, node
        else:
            graph.visit(node)
            expanded = new_states(node, a_star)
            [graph.add_node(x, depth + 1) for x in expanded]
            for children in expanded:
                if children in graph.visited:
                    continue
                if children in stack:
                    if (children.cost + node.dist + 1 < children.cost + children.dist) and a_star:
                        children.setParent(node)
                        children.setDist(node.dist + 1)
                else:
                    stack.append(children)

        print(depth)
        depth += 1
```

```python
def better_nWrong_heuristics(self):
    cost = 0

    for tube in self.gamestate.tubes:
        balls = tube.balls.copy()
        idx = next((i for i, v in enumerate(balls) if v != balls[0]), -1)
        if idx == -1:
            continue

        balls = balls[idx:]
        cost += len(balls) * 2

    return cost
```

Implementation work already carried out

Programming language: Python, with visualization using pygame package

Development environment: VSCode/IntelliJ

Data Structures:

- Lists, for representing the Tubes and the Game
- Nodes and Graphs

File Structure: The Project's Repository is available at Github

Implemented Work: All algorithms lectured; Graphical Interface, playable game with hints