Candy Crisis Puzzle Solver

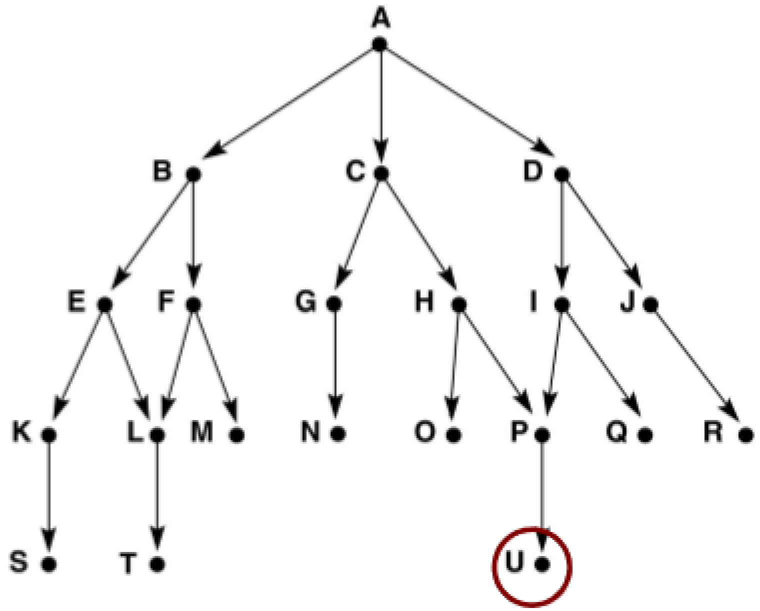
TEAM: CAPTAIN CRYPTO

Data Structures

- Puzzle Object:
  - Holds current state
  - Functions to perform moves and search
- Node Struct:
  - Used in open list and closed list stack
  - Holds a state of a puzzle
  - And current path cost

Search w/o Heuristic

- Depth First
- Depth-Cutoff K = 40
  - Maximum of 40 moves to be considered



Search w/o Heuristic cont'd

- Generates children of a given parent
- Discards children if they are more than 40 levels deep
- Checks each child generated for success
- Places children at top of open list if no success

```
begin

stack openList = first parent
stack closedList = empty

while !openList.empty do
    parent = top.stack;
    children = generateChildren(parent);

    if childrenDepth < 40
        foreach child in children:
            if child !exsists in openList or closedList
                openList.push(child);

    closedList.push(parent);

end
```

Search w/o Heuristic cont'd

- **Pros:**
  - Uses less memory, therefore quicker due to smaller open and closed lists

- **Cons:**
  - If solution >40 moves, will not succeed
  - Will take long if solution path is not in one of the first branches

Heuristic

- Similar to A*: f(n) = g(n) + h(n)
  - Except our g(n) = all path costs to this state
- h(n) = Manhattan distance
  - Goes through top row
  - Finds a matching candy that's not being used
  - Counts the distance to the bottom cell of same column
  - Adds count to total move count for that state

Example State:

b	r	r	b	w
r	b	r	w	
r	r	r	r	b

h(n) = 2 + 1 + 2 = 5

Search and Heuristic

- Open list sorted by increasing f(n)
- Will pick lowest f(n) to generate children and test for success

- **Pros:**
  - Fast
  - Not memory intensive
- **Cons:**
  - Will not always find shortest path (solution path might be a few more than the actual shortest path)

Conclusion

- Our result in Challenge
- Result compared to hypothesis
- The end!