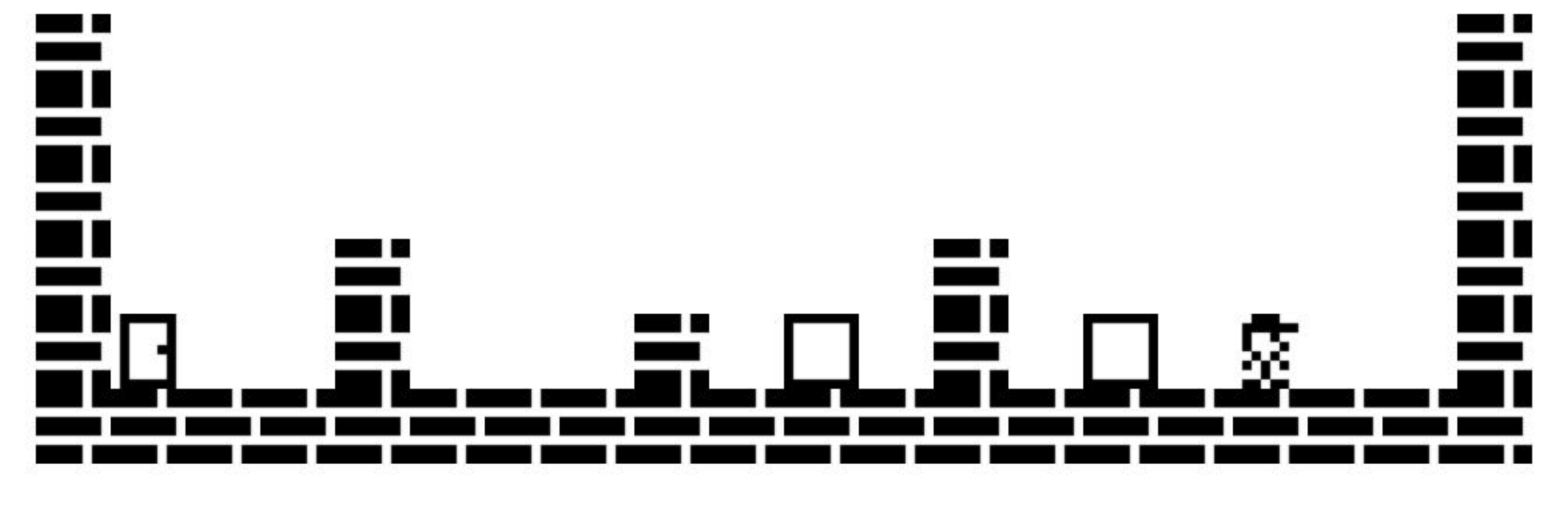
Block Dude Solver

Joseph Yeager

What is *Block Dude*?

- *Block Dude* is a 2d platformer puzzle game originally developed for TI-83/TI-84 calculators. You move blocks to turn obstacles into stairs.
- Game environment:
  - Bricks - Unmoveable 
  - Blocks - Moveable 
  - Door - The goal state 
  - The Dude 
- Some legal moves:
  - Pick up a block - if not already holding one
  - Drop a block - if holding one
  - Move West/East
  - Face West/East
  - Climb stairs
  - Fall

Example Level



My Solution

- Constraint Satisfaction
- Heuristic Search

Constraint Satisfaction

- Moves can be modeled as constraints

0	3
1	1

- The newly modeled constraints are used to:
  - Check what moves are available in the current state
  - Prevent illegal moves from being made

Constraint Satisfaction

- To check what moves are available, a 3x3 matrix surrounding the player is examined:

0	0	0
0	3	0
1	1	1

- Then the quadrants of the matrix are compared against the legal moves:

0	0	0
0	3	0
1	1	1

0	0	0
0	3	0
1	1	1

0	0	0
0	3	0
1	1	1

0	0	0
0	3	0
1	1	1

Heuristic Search

- The search utilized a ternary tree to track state (level, player pos,etc).
- I originally started with a traditional tree with left, right, middle members
  - Algorithms were hard to implement and debug.
  - Very time and space inefficient.
- Moved to an array representation of a ternary tree.
  - Algorithms trivial to implement.
  - Very time and space inefficient. Requires 3^{moves} entries.
- Finally arrived at an ordered dictionary representation of a tree.

Ordered Dictionary Tree

- Uses an ordered dictionary from Python collections library.
  - Key:Value store
  - Indexable by keys and order in dictionary
- Uses the same indexing system as an array representation:

Parent: $\left\lfloor \frac{i-1}{k} \right\rfloor$ Nth Child: $k \cdot i + 1 + c$

- When a node is added, its children's indices are computed and stored in a list in the node. When the node's children are populated, the first index is popped off the front of the list and used as the key for the child.

- Allows for only storing and iterating over what is needed - very space and time efficient

My Heuristic

- Prevent backwards progression and oscillations.
  - Blacklisted move sequences. Ex:[“Pickup”, “Drop”], [“FW”, “FE”]
  - Prune when a blacklisted move sequence is encountered

- Scan for obstacles periodically and use the data in several ways:
  - If there is no obstacle, prioritize moves in the goal direction
  - If there is an obstacle, generate a list of the block goal positions
  - Only drop a block if the position is the the block goals

- If a move results in victory, always choose that.

Demonstration