AI Project 2 Report

By Thalia Kennedy & Audrey Fuller

Painter Report

1) **Audrey Fuller Evaluation Function**

My evaluation function looks for colors within an RGB bounded range between pink and purple. It then creates a mask over the image, looking for only pixels that fall within the color range. The evaluation value is then determined by the number of pixels in that range. More purple-pink images are favored at the end of each generation to survive to the next.

2) **Thalia Kennedy Evaluation Function**

In my evaluation of the image outputs, I find the amount of pixels that are within a range that favors a blue-ish green color. However the RGB value is somewhat randomized, as the R value in the lower range is a random integer between 0 and 20, and on the upper range is between 100 and 120. This allows for there to be some variety each time the program is run. Each pixel in the image is counted and if it is within the pixel range given, it is added to the counter. The images in the pool with the highest counts are the 3 images saved.

Parking Report

1) **Calculate the size of the state space as a function of a, b, c.**

Known Variables:
a attendants (does not contribute to state space)
b non-unique barriers
c unique cars
*c*c* unique spaces

Permutation Formula:
$P(n, r) = \frac{n!}{(n-r)!}$
n = total unique items in the set
r = items taken for the permutation

To put *c* unique cars in *c*c* unique spaces,
$P\left(c^2, c\right) = \frac{(c^2)!}{(c^2-c)!}$

Combination Formula:
$C(n, r) = \frac{n!}{r!(n-r)!}$
n = total items in the set
r = items taken for the combination

To put *b* non-unique barriers in the remaining *c*c-c* unique spaces,

$$C\left(c^2 - c, b\right) = \frac{(c^2-c)!}{b!((c^2-c)-b)!}$$

By multiplying the total possible permutations by the total possible combinations, we get the State Space Size function:

$$S(b,c) = \frac{(c^2)!}{(c^2-c)!} * \frac{(c^2-c)!}{b!((c^2-c)-b)!} =$$

$$S(b,c) = \frac{(c^2)!}{b!(c^2-c-b)!}$$

Checking Work:
c= 2
b=1

$$S(1,2) = \frac{(2^2)!}{1!(2^2-2-1)!}$$

$$= \frac{4!}{1!1!} = 4! = 24$$

This checks out when calculating size manually.

2) Calculate the branching factor as a function of a, b, c.

A "node" of the branching tree in this problem is considered to be a set of actions for *a* attendants to take on a subset *a* of *c* total cars. While this problem statement does include illegal actions (such as moving off the board, into another car's space or into a barrier), this formula takes *all* actions into account in order to find the *maximum branching factor*. In addition, checking for valid actions would add a level of complexity to the formula I was unable to solve.

Known Variables:
a non-unique attendants
b barriers (does not contribute to branching factor)
c unique cars
*c*c* spaces (does not contribute to branching factor)
5 possible actions each car can take (up, down, left, right, nothing)

The Brute Force Method
As a starting point, I manually solved the number of branches for a few *a* and *c* values.
a = 1, *c* = 2, branches = 10
a = 2, *c* = 2, branches = 25
a = 1, *c* = 3, branches = 15
a = 2, *c* = 3, branches = 75
a = 3, *c* = 3, branches = 125
Looking for a correlation between *a, c* and branches, I saw that factors of 5 were showing up quite a bit. This got me the first half of my equation, and the branches were then rewritten as:

$$5^1 * 2 = 5^{c-a} * 2$$

$5^2 * 1 = 5^{c-a} * 1$

$5^1 * 3 = 5^{c-a} * 3$

$5^2 * 3 = 5^{c-a} * 3$

$5^3 * 1 = 5^{c-a} * 1$

When looking at the later part of the equation, I realized that it was correlating to the number of different combinations between attendants *a* and cars *c*. For example, when *a* = 1 and *c* = 2, there were two possible combinations, where the attendant moves either car 1 or 2. When *a* = 2 and *c* = 3, there were three possible combinations, where the attendants move either cars 1 & 2, 1 & 3 or 2 & 3. Therefore, I used the combination formula for the second half.

Combination Formula:

$C(n, r) = \frac{n!}{r!(n-r)!}$

n = total items in the set

r = items taken for the combination

To find the number of combinations of a non-unique attendants choosing from c unique cars,

$C(c, a) = \frac{c!}{a!(c-a)!}$

This is then combined to the first half of the equation to get the branching factor equation:

$B(c, a) = 5^{c-a} * \frac{c!}{a!(c-a)!}$

3) Heuristic_dist implementation.

The heuristic_dist function goes through each car and adds up how many rows and columns away from it's goal state it is. In addition to this sum, the amount of barriers and cars in it's way until that goal state is also added to the cost. This is summed up for ever car in the CxC garage and this cost is returned.

4) Problem invocation and output for the largest parameters (number of cars, number of attendants, number of barriers) on which the algorithm can find a solution within 2 minutes.

a) Depth_first_tree_search

Depth first tree search never completes because it does not keep track of the explored nodes (different from depth first graph search). This causes the search to loop within itself indefinitely, and it will never reach a solution, even with the smallest maps.

b) Depth_first_graph_search

python parking.py -s depth_first_graph_search -c 3 -a 3 -b 1

```
+----------+
|  0 1 2 |
|    *    |
|         |
+----------+
```

[{(0, 'down'), (1, 'stay'), (2, 'down')}, {(0, 'down'), (1, 'right'), (2, 'down')}, {(1, 'left'), (2, 'left'), (0, 'up')}, {(0, 'down'), (1, 'right'), (2, 'left')}, {(1, 'left'), (2, 'up'), (0, 'right')}, {(1, 'right'), (2, 'down'), (0, 'right')}, {(1, 'left'), (2, 'right'), (0, 'up')}, {(0, 'down'), (1, 'right'), (2, 'right')}, {(1, 'left'), (2, 'up'), (0, 'stay')}, {(1, 'left'), (2, 'up'), (0, 'left')}, {(1, 'down'), (2, 'left'), (0, 'right')}, {(1, 'down'), (2, 'right'), (0, 'up')}, {(0, 'down'), (1, 'right'), (2, 'left')}, {(1, 'stay'), (2, 'left'), (0, 'up')}, {(0, 'down'), (1, 'right'), (2, 'right')}, {(2, 'right'), (1, 'up'), (0, 'left')}, {(1, 'down'), (2, 'left'), (0, 'left')}, {(1, 'left'), (2, 'right'), (0, 'up')}, {(0, 'down'), (1, 'left'), (2, 'left')}, {(1, 'stay'), (2, 'right'), (0, 'right')}, {(1, 'stay'), (2, 'down'), (0, 'stay')}, {(1, 'up'), (2, 'down'), (0, 'right')}, {(1, 'stay'), (2, 'left'), (0, 'up')}, {(1, 'down'), (2, 'right'), (0, 'up')}, {(2, 'left'), (1, 'up'), (0, 'left')}, {(1, 'down'), (2, 'right'), (0, 'left')}, {(0, 'down'), (2, 'left'), (1, 'up')}, {(0, 'down'), (1, 'stay'), (2, 'left')}, {(1, 'up'), (2, 'stay'), (0, 'right')}, {(1, 'down'), (2, 'up'), (0, 'right')}, {(2, 'stay'), (0, 'up'), (1, 'up')}, {(1, 'right'), (2, 'stay'), (0, 'up')}, {(0, 'down'), (1, 'stay'), (2, 'up')}, {(1, 'right'), (2, 'stay'), (0, 'stay')}, {(0, 'down'), (1, 'stay'), (2, 'right')}, {(1, 'down'), (2, 'left'), (0, 'up')}, {(1, 'stay'), (0, 'up'), (2, 'down')}, {(1, 'down'), (2, 'down'), (0, 'left')}, {(1, 'up'), (2, 'right'), (0, 'right')}, {(1, 'down'), (2, 'right'), (0, 'stay')}, {(0, 'down'), (1, 'stay'), (2, 'left')}, {(1, 'stay'), (2, 'left'), (0, 'stay')}, {(1, 'left'), (2, 'stay'), (0, 'stay')}, {(0, 'down'), (1, 'stay'), (2, 'stay')}]
elapsed time: 0.17249774932861328 seconds

c) Breadth_first_graph_search

python parking.py -s breadth_first_graph_search -c 3 -a 3 -b 1

```
+----------+
|   0 1 2 |
|    *     |
|          |
+----------+
```

[{(1, 'stay'), (0, 'stay'), (2, 'down')}, {(0, 'stay'), (2, 'down'), (1, 'right')}, {(1, 'down'), (2, 'left'), (0, 'right')}, {(1, 'down'), (2, 'left'), (0, 'right')}, {(0, 'down'), (2, 'stay'), (1, 'left')}, {(0, 'down'), (2, 'stay'), (1, 'stay')}]
elapsed time: 0.42432427406311035 seconds

d) Best_first_graph_search

python parking.py -s best_first_graph_search -c 3 -a 3 -b 1

```
+----------+
|   0 1 2 |
|    *     |
|          |
+----------+
```

[{(0, 'stay'), (1, 'stay'), (2, 'down')}, {(2, 'stay'), (1, 'right'), (0, 'stay')}, {(0, 'right'), (1, 'stay'), (2, 'down')}, {(2, 'up'), (1, 'down'), (0, 'stay')}, {(0, 'right'), (1, 'stay'), (2, 'down')}, {(0, 'stay'), (2, 'left'), (1, 'stay')}, {(1, 'down'), (0, 'stay'), (2, 'right')}, {(2, 'up'), (0, 'down'), (1, 'stay')}, {(2, 'up'), (0, 'stay'), (1, 'stay')}, {(0, 'stay'), (2, 'left'), (1, 'stay')}, {(0, 'up'), (2, 'left'), (1, 'stay')}, {(0, 'down'), (1, 'left'), (2, 'down')}, {(0, 'down'), (1, 'right'), (2, 'down')}, {(1, 'left'), (2, 'stay'), (0, 'stay')}]
elapsed time: 0.5924556255340576 seconds

e) Astar_search

```
python parking.py -s astar_search -c 3 -a 3 -b 1
+----------+
|   0 1 2 |
|     *      |
|            |
+----------+

[{(1, 'stay'), (2, 'down'), (0, 'stay')}, {(1, 'right'), (2, 'down'), (0, 'stay')}, {(1, 'down'), (2, 'left'), (0, 'right')}, {(1, 'down'), (2, 'left'), (0, 'right')}, {(1, 'left'), (0, 'down'), (2, 'stay')}, {(1, 'stay'), (0, 'down'), (2, 'stay')}]
elapsed time: 0.8177845478057861 seconds
```