Thinking Recursively

An Interesting Peruse

WSJ: “Best and Worst Jobs of 2012”

http://online.wsj.com/article/SB10001424052702303772904577336230132805276.html

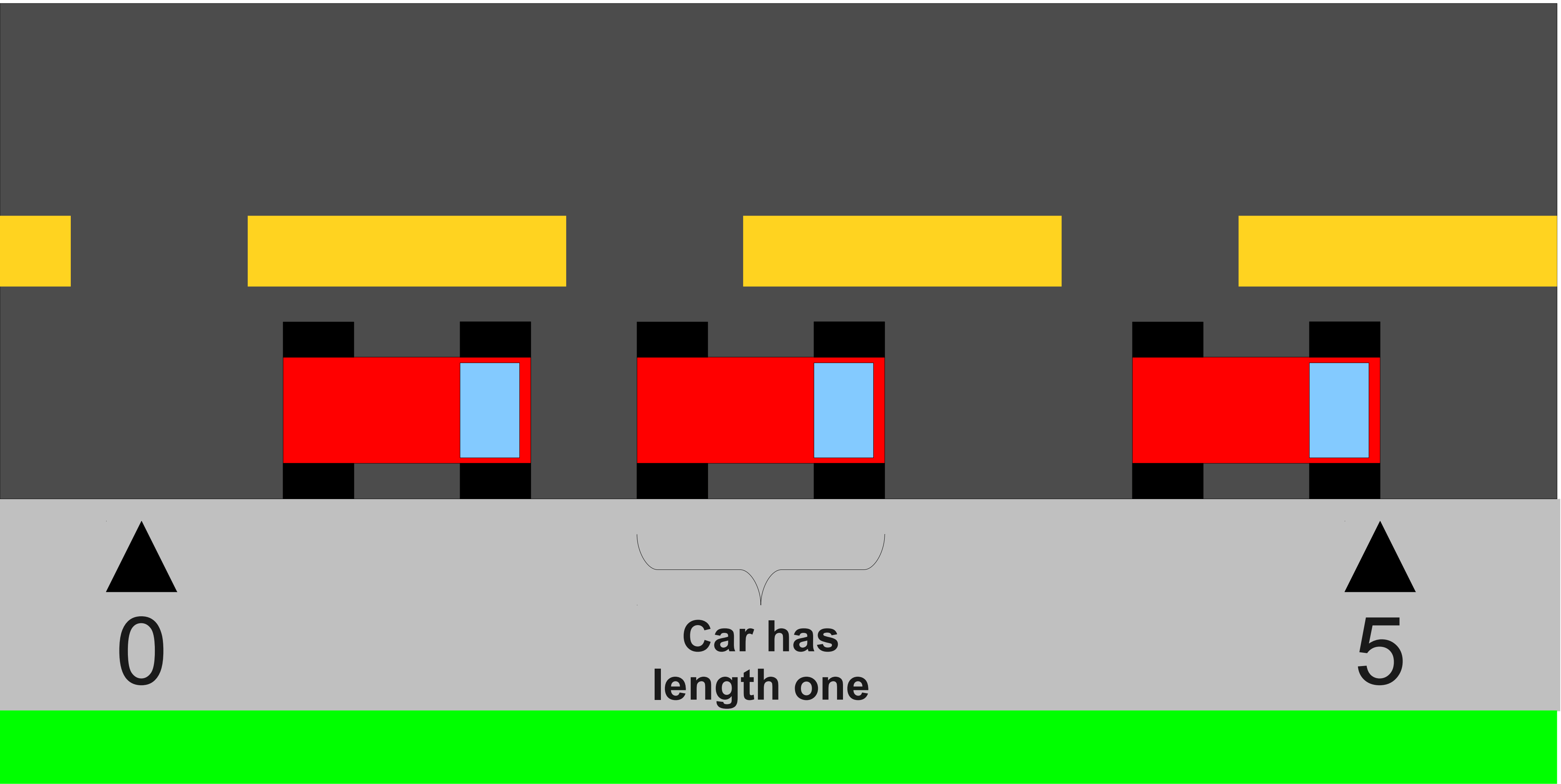
#1 Job: **Software Engineer**

Thinking Recursively, Part II

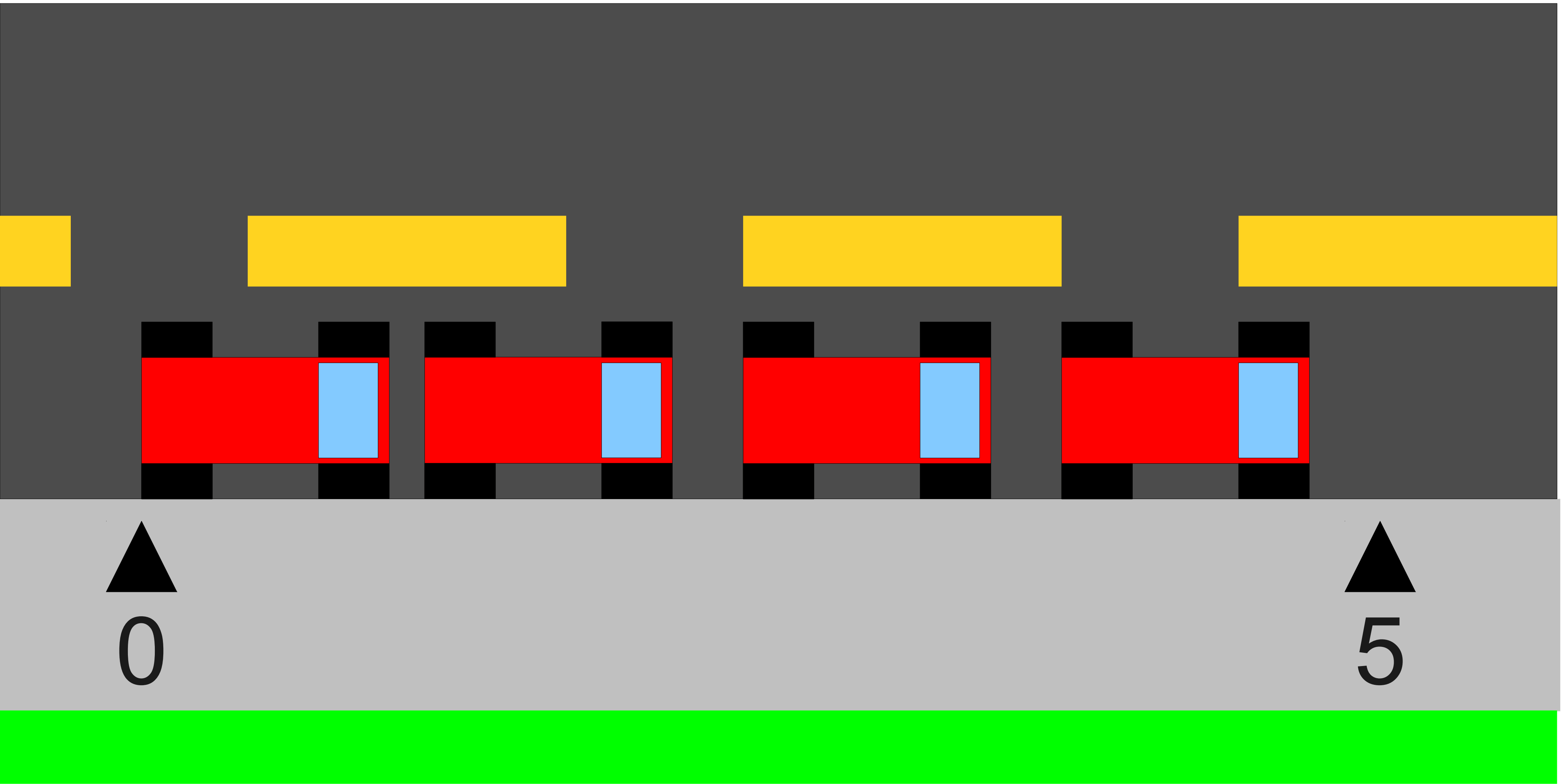
Recursive Problem-Solving

```
if (problem is sufficiently simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem up into one or more smaller
        problems with the same structure as the original.
    Solve each of those smaller problems.
    Combine the results to get the overall solution.
    Return the overall solution.
}
```

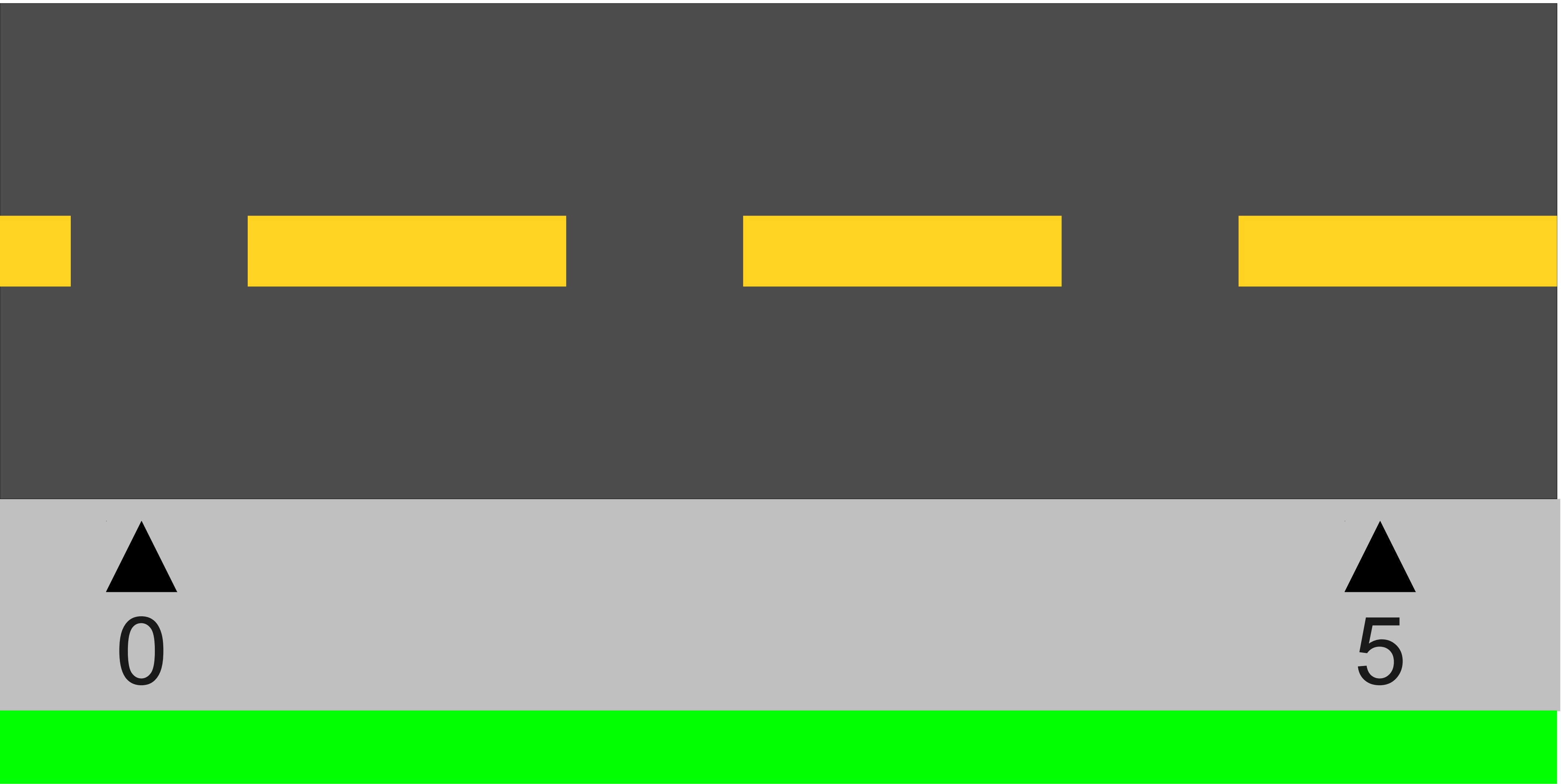
Parking Randomly



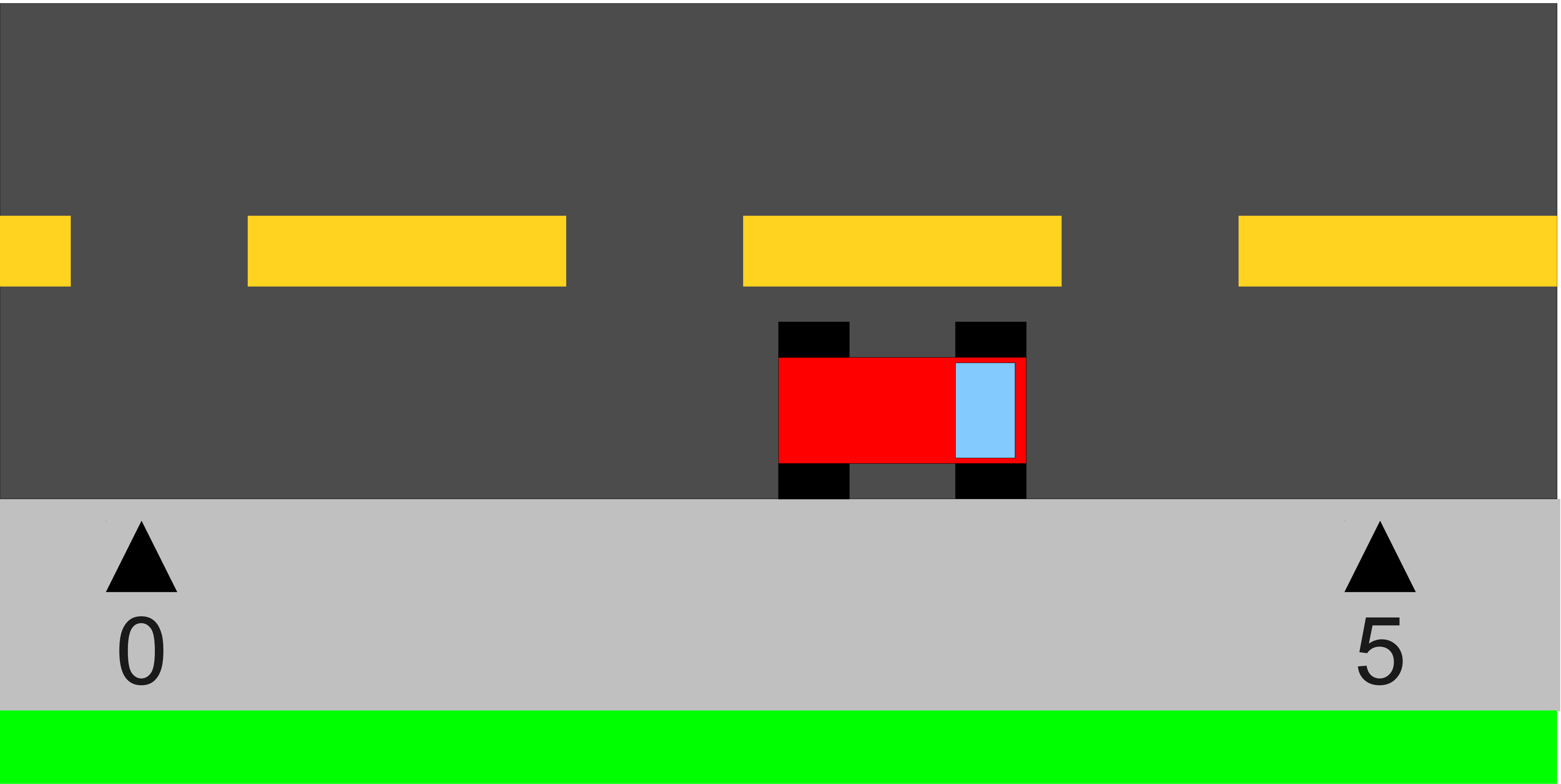
Parking Randomly



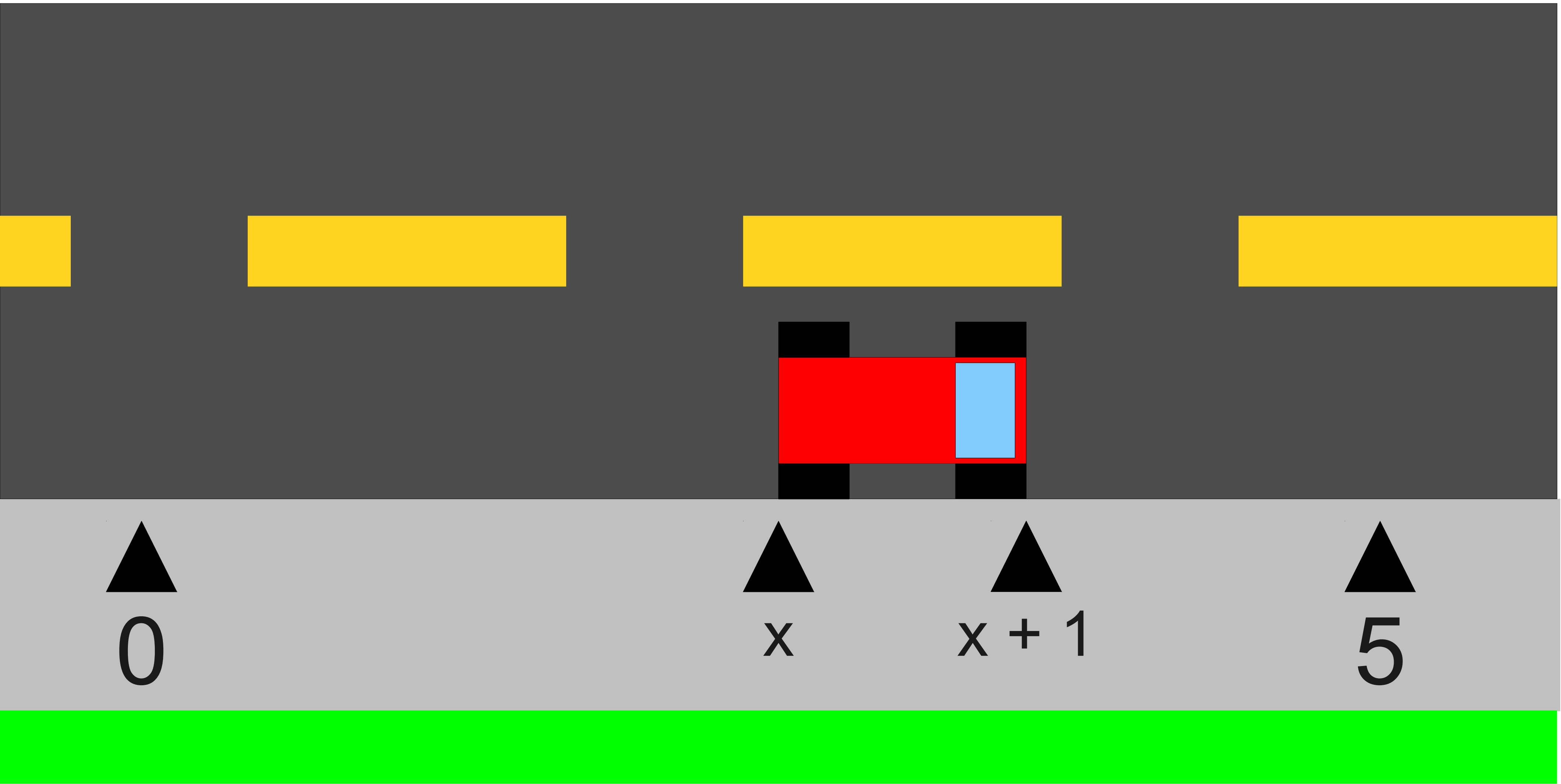
Parking Randomly



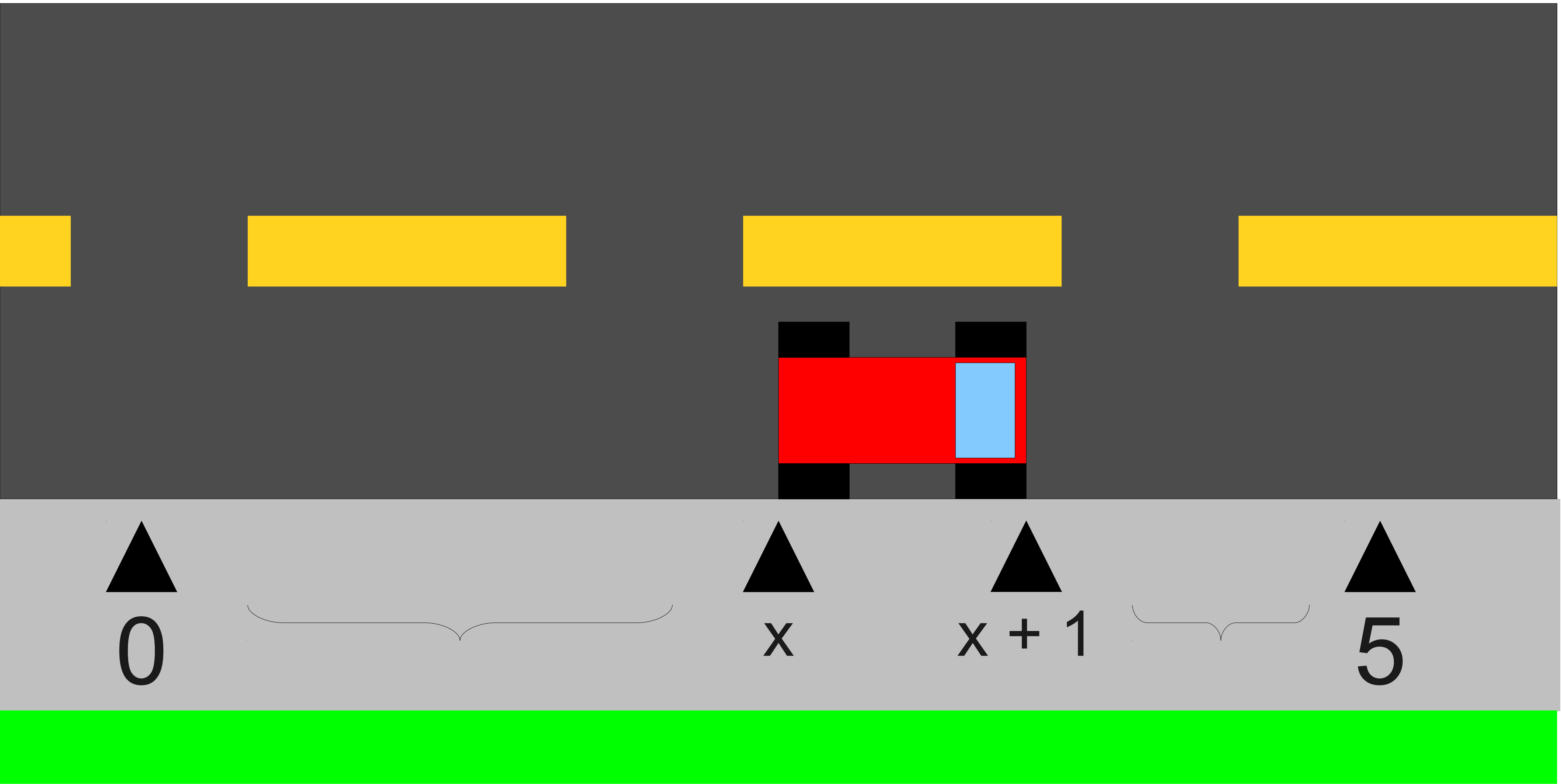
Parking Randomly



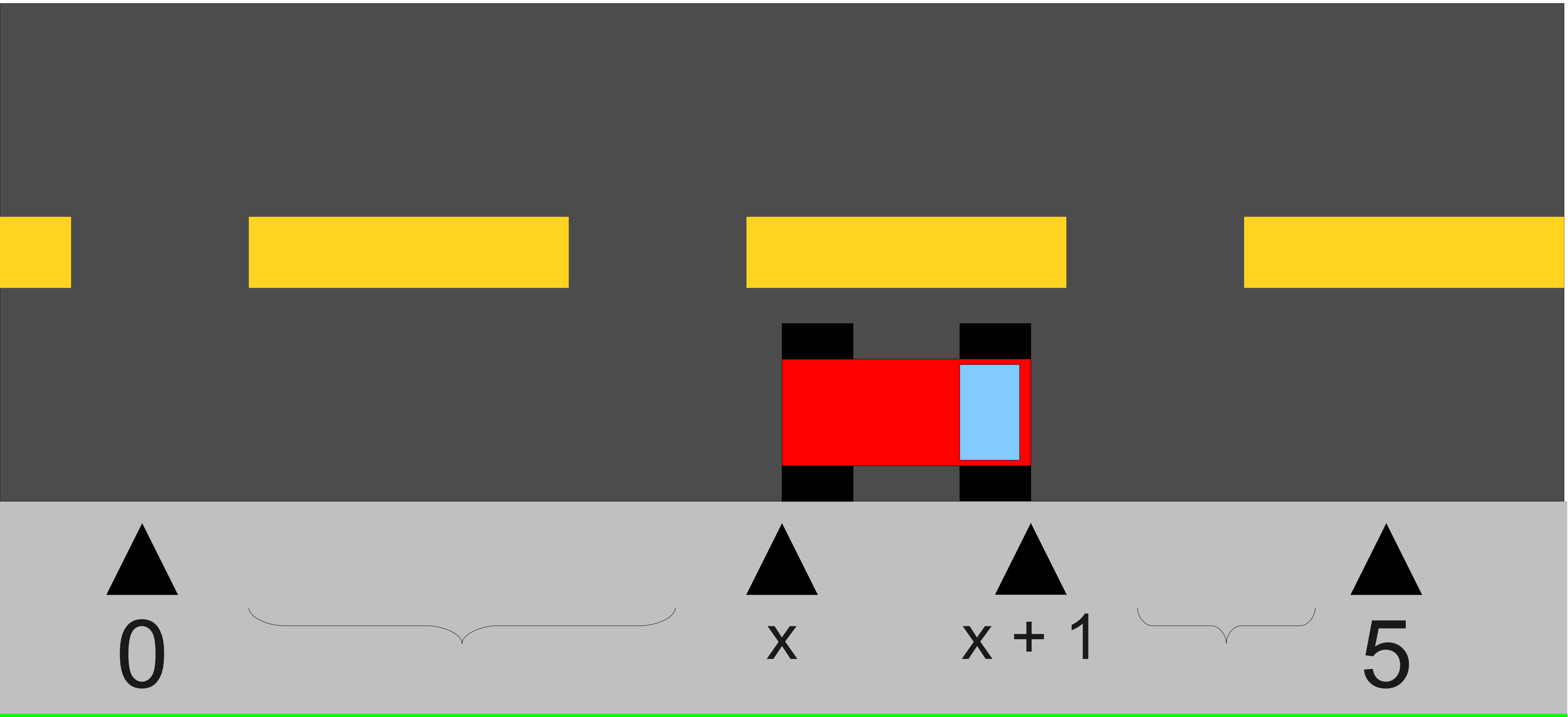
Parking Randomly



Parking Randomly



Parking Randomly



Place cars randomly in these ranges!

Parking Randomly

```
int parkRandomly(double low, double high) {
    if (high - low < 1.0) {
        return 0;
    } else {
        double x = randomReal(low, high - 1.0);
        return 1 + parkRandomly(low, x) +
                   parkRandomly(x + 1, high);
    }
}
```

So What?

- The beauty of our algorithm is the following recursive insight:

Split an area into smaller, independent pieces and solve each piece separately.

- Many problems can be solved this way.

Generating Mondrian Paintings

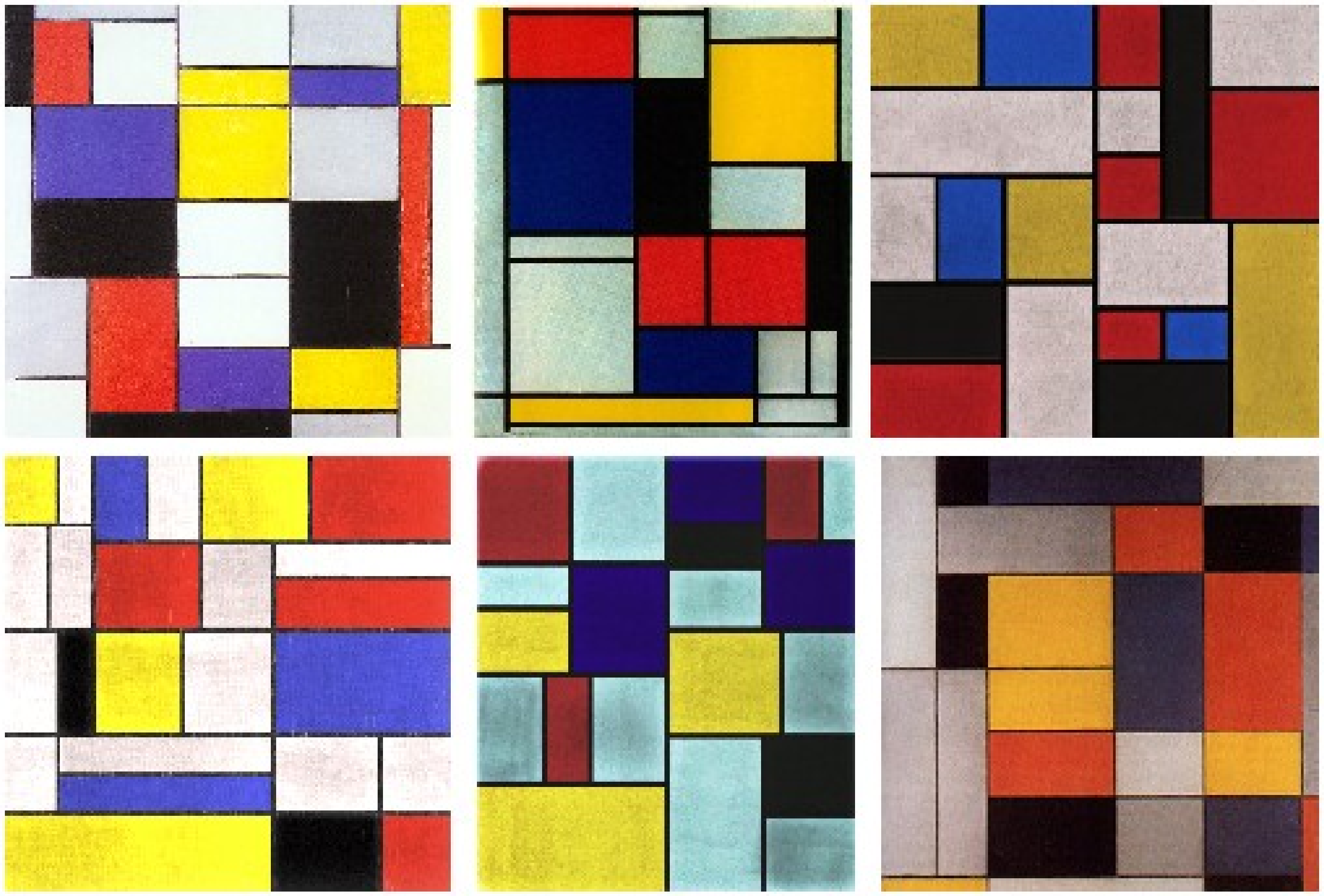
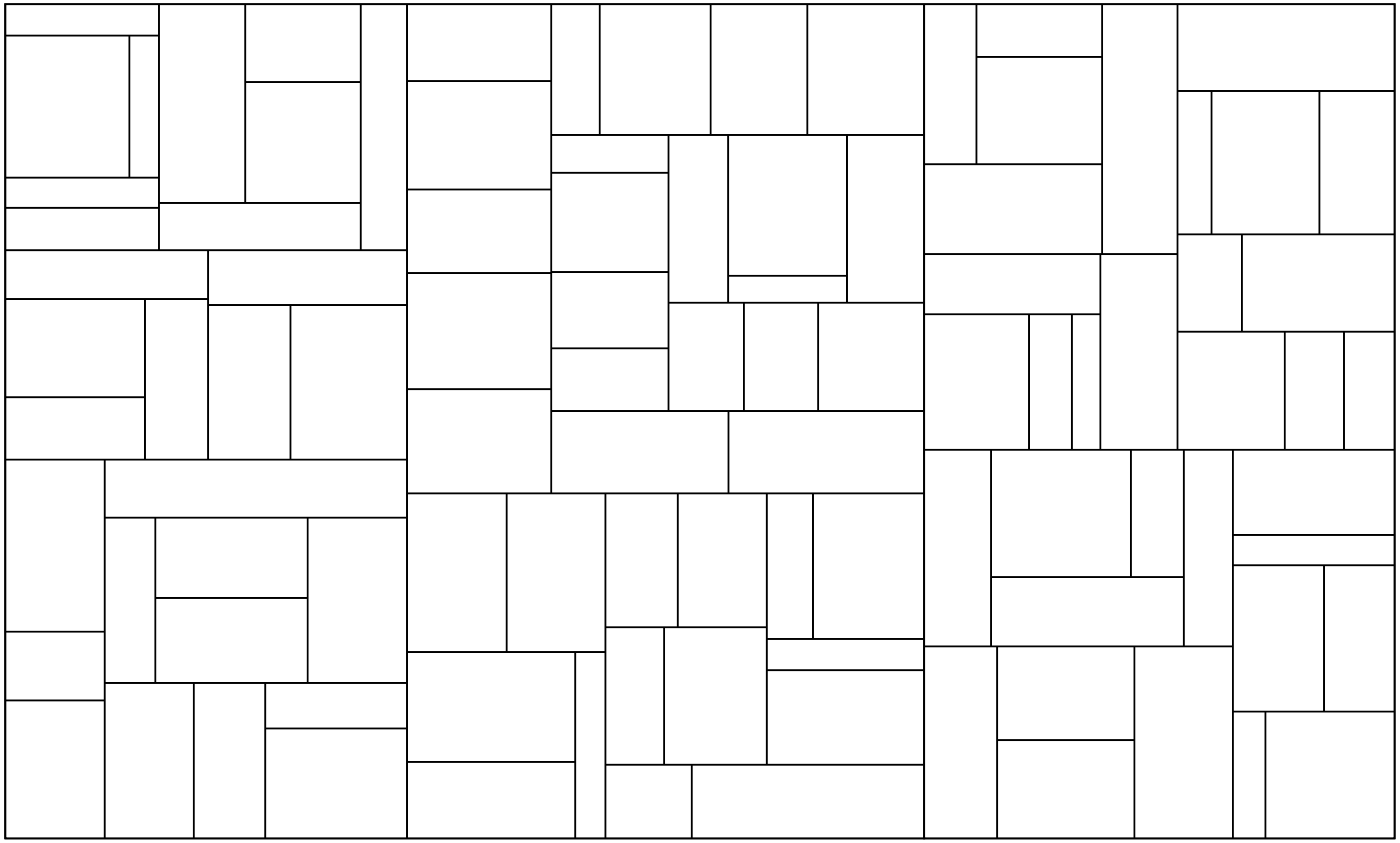


Fig. 11: Three real Mondrian paintings, and three samples from our targeting function. Can you tell which is which?

Source: Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun, "Metropolis Procedural Modeling," *ACM Transactions on Graphics,* April 2011.

Generating Mondrians



The CS106B Graphics Library

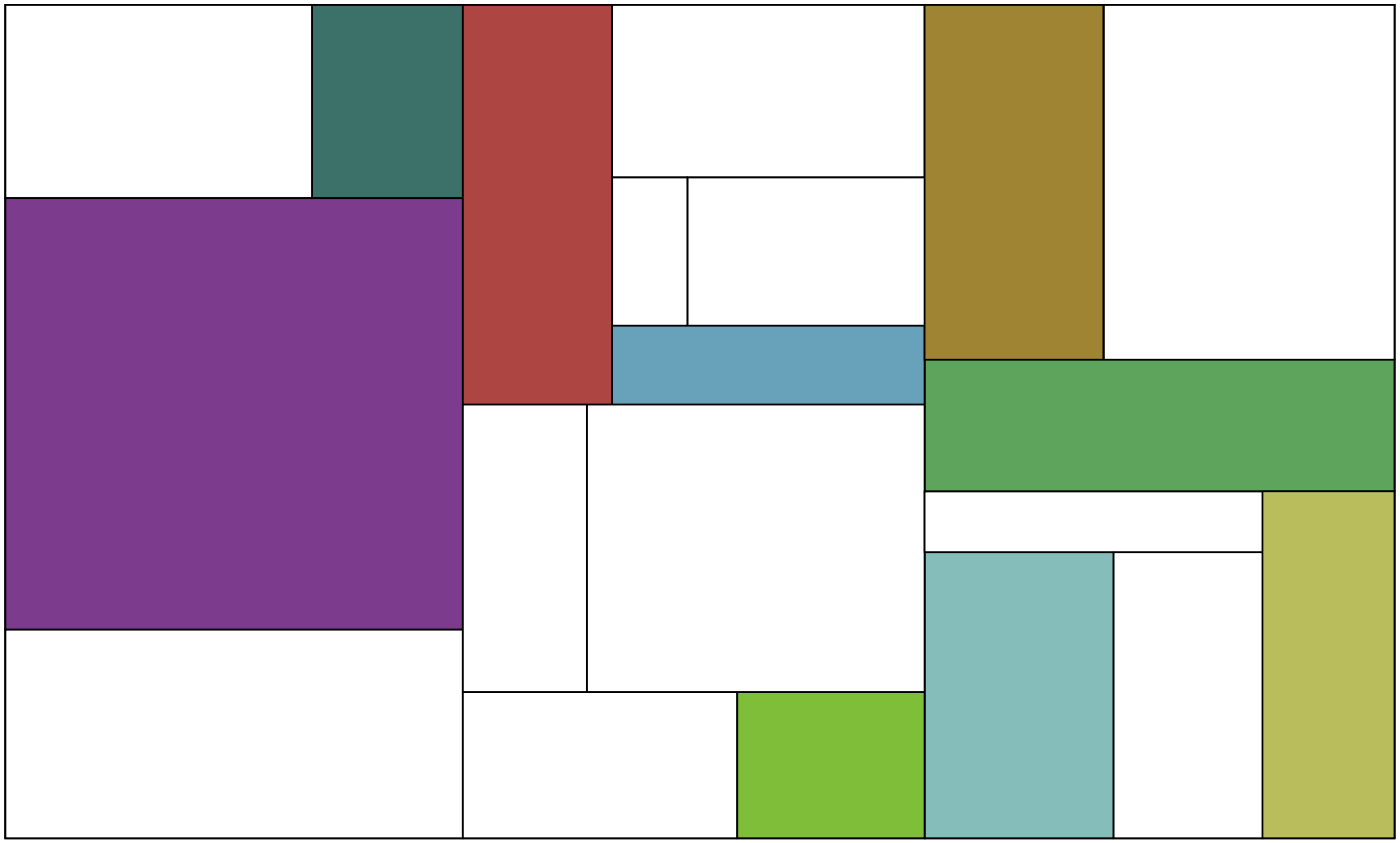
`initGraphics(`*width*`, `*height*`)` Creates a graphics window with the specified dimensions.
`drawLine(`*x*`0, `*y*`0, `*x*`1, `*y*`1)` Draws a line connecting the points (*x*0, *y*0) and (*x*1, *y*1).
`drawPolarLine(`*x*`0, `*y*`0, `*r*`, `*theta*`)` Draws a line *r* pixels long in direction *theta* from (*x*0, *y*0). To make chaining line segments easier, this function returns the ending coordinates as a **`GPoint`**.
`getWindowWidth()` Returns the width of the graphics window.
`getWindowHeight()` Returns the height of the graphics window.

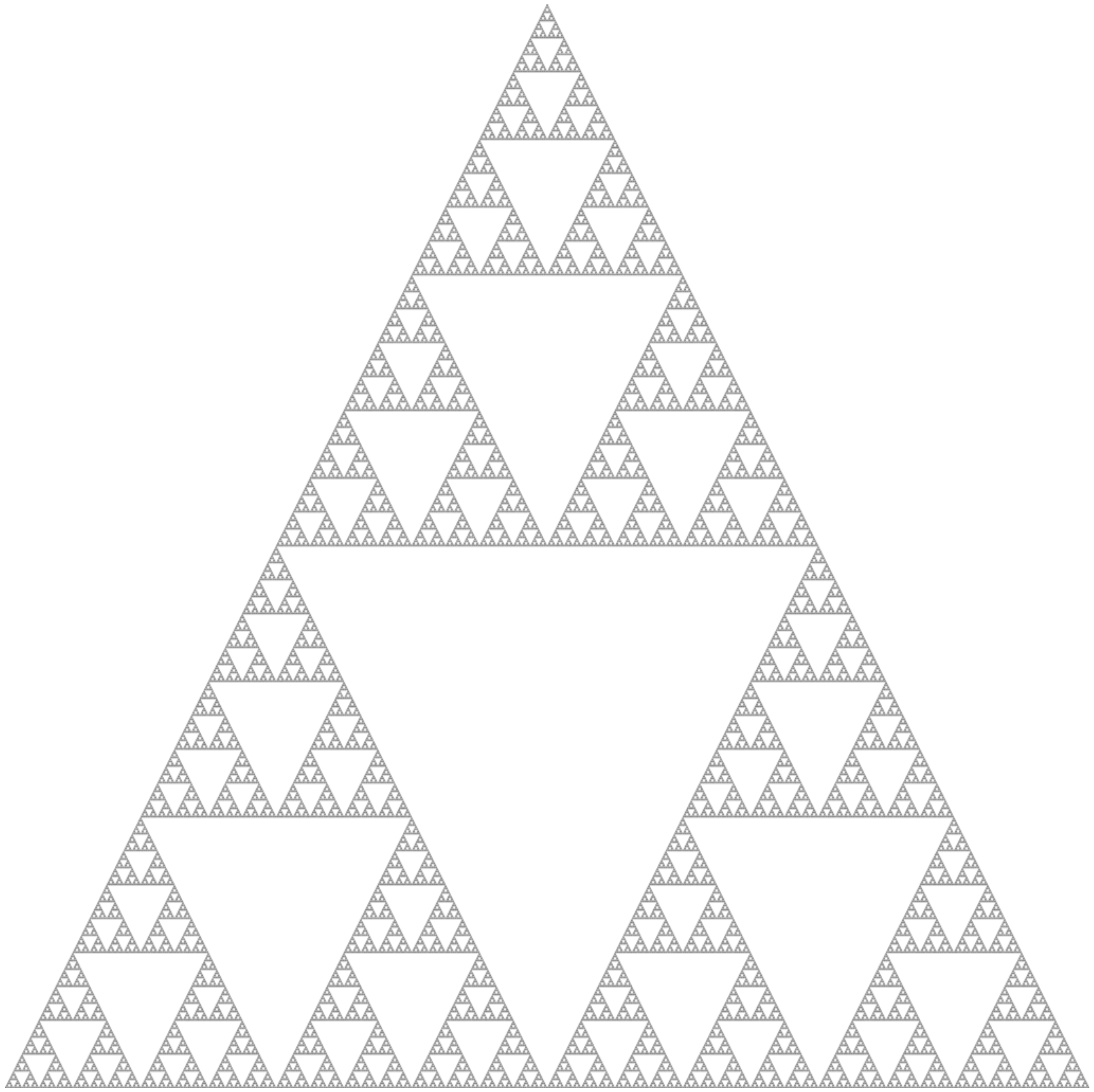
Many more functions exist in the **`graphics.h`** interface, which is described on the web site.

Drawing Rectangles

`drawRect(`*`x`*`, `*`y`*`, `*`width`*`, `*`height`*`)` Draws the outline of a rectangle with the specified bounds.
`fillRect(`*`x`*`, `*`y`*`, `*`width`*`, `*`height`*`)` Fills the outline of the specified rectangle using the current color.
`setColor(`*`color`*`)` Sets the pen color to the specified color string (such as `"BLACK" or "RED"`)
`setColor("#`*rrggbb*`")` Sets the red/green/blue components to the specified hexadecimal values.

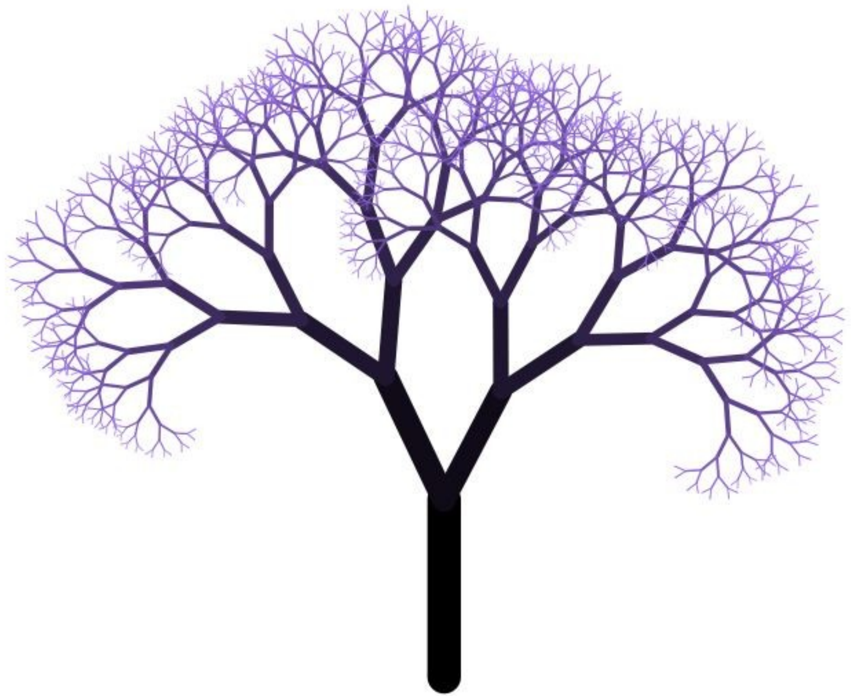
Once More with **Color**





A **fractal image** is an image that is defined in terms of smaller versions of itself.

Fractal Trees



- We can generate a fractal tree as follows:
  - Grow in some direction for a period of time.
  - Then, split and grow two smaller trees outward at some angle.

More Trees

- What if we change the amount of branching?
- What if we make the lines thicker?
- What if we allow the tree to keep growing after it branches?
- Stanford **Dryad** program uses a combination of recursion, machine learning, and human feedback to design aesthetically pleasing trees.
  - Check it out at **http://dryad.stanford.edu/**

Exhaustive Recursion

Generating All Possibilities

- Commonly, you will need to generate all objects matching some criteria.
  - Word Ladders: Generate all words that differ by exactly one letter.
- Often, structures can be generated iteratively.
- In many cases, however, it is best to think about generating all options recursively.

Subsets

- Given a set S, a **subset** of S is a set T composed of elements of S.
- Examples:
  - {0, 1, 2} ⊆ {0, 1, 2, 3, 4, 5}
  - {dikdik, ibex} ⊆ {dikdik, ibex}
  - { A, G, C, T } ⊆ { A, B, C, D, E, …, Z }
  - { } ⊆ {a, b, c}
  - { } ⊆ { }

Generating Subsets

- Many important problems in computer science can be solved by listing all the subsets of a set *S* and finding the “best” one out of every option.
- Example:
  - You have a collection of sensors on an autonomous vehicle, each of which has data coming in.
  - Which **subset** of the sensors do you choose to listen to, given that each takes a different amount of time to read?

Generating Subsets

$$\{0, 1, 2\}$$

$$
\begin{array}{ll}
\{\quad\quad\quad\} & \{0\quad\quad\} \\
\{\quad\quad 2\} & \{0,\quad 2\} \\
\{\quad 1\quad\} & \{0, 1\quad\} \\
\{\quad 1, 2\} & \{0, 1, 2\}
\end{array}
$$

Generating Subsets

$$\{0, 1, 2\}$$

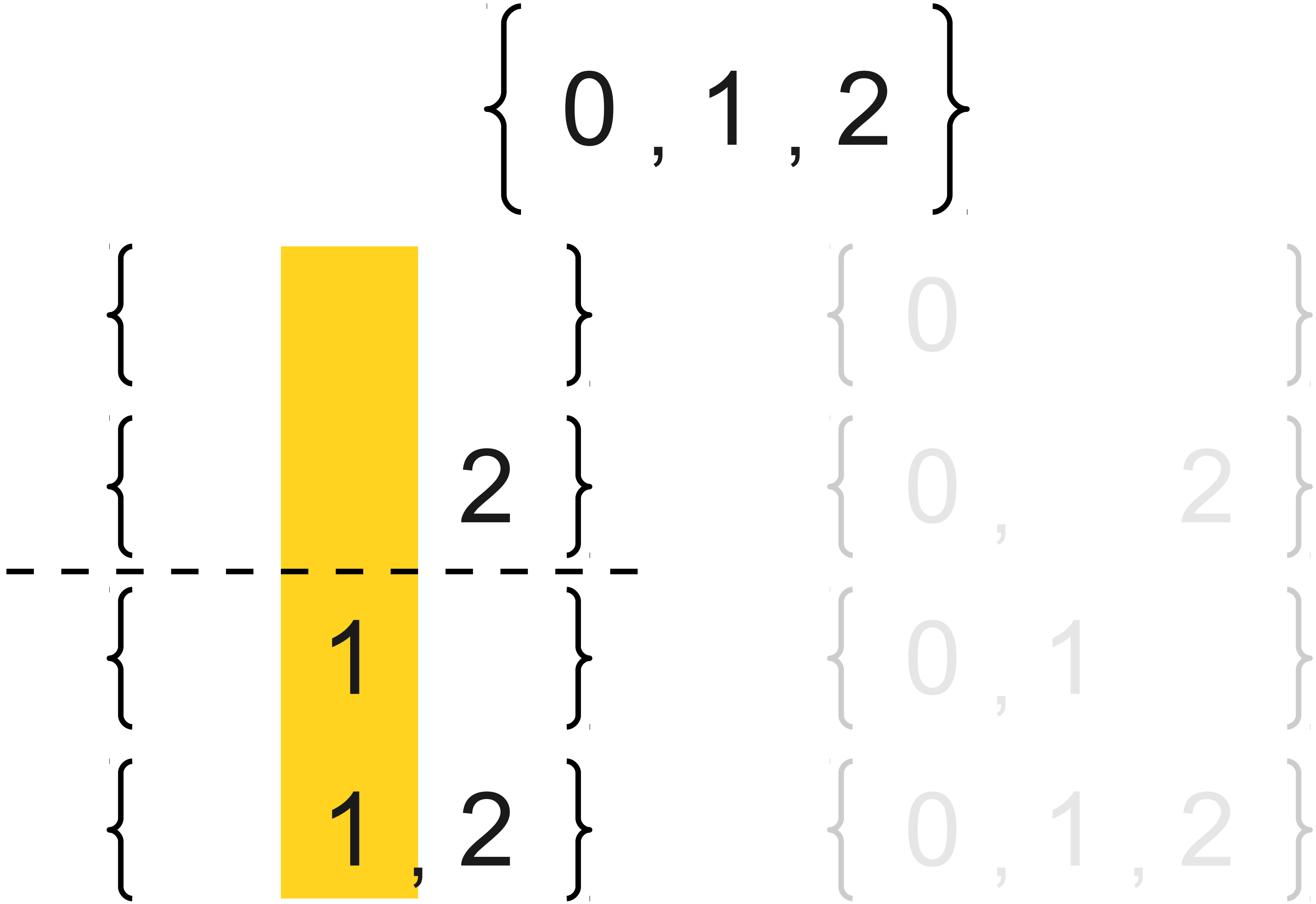
$\{\ \}$	$\{0\}$
$\{2\}$	$\{0, 2\}$
$\{1\}$	$\{0, 1\}$
$\{1, 2\}$	$\{0, 1, 2\}$

Generating Subsets

$$\{0, 1, 2\}$$

$\{\ \}$	$\{0\}$
$\{2\}$	$\{0, 2\}$
$\{1\}$	$\{0, 1\}$
$\{1, 2\}$	$\{0, 1, 2\}$

Generating Subsets



Generating Subsets

$$\{0, 1, 2\}$$

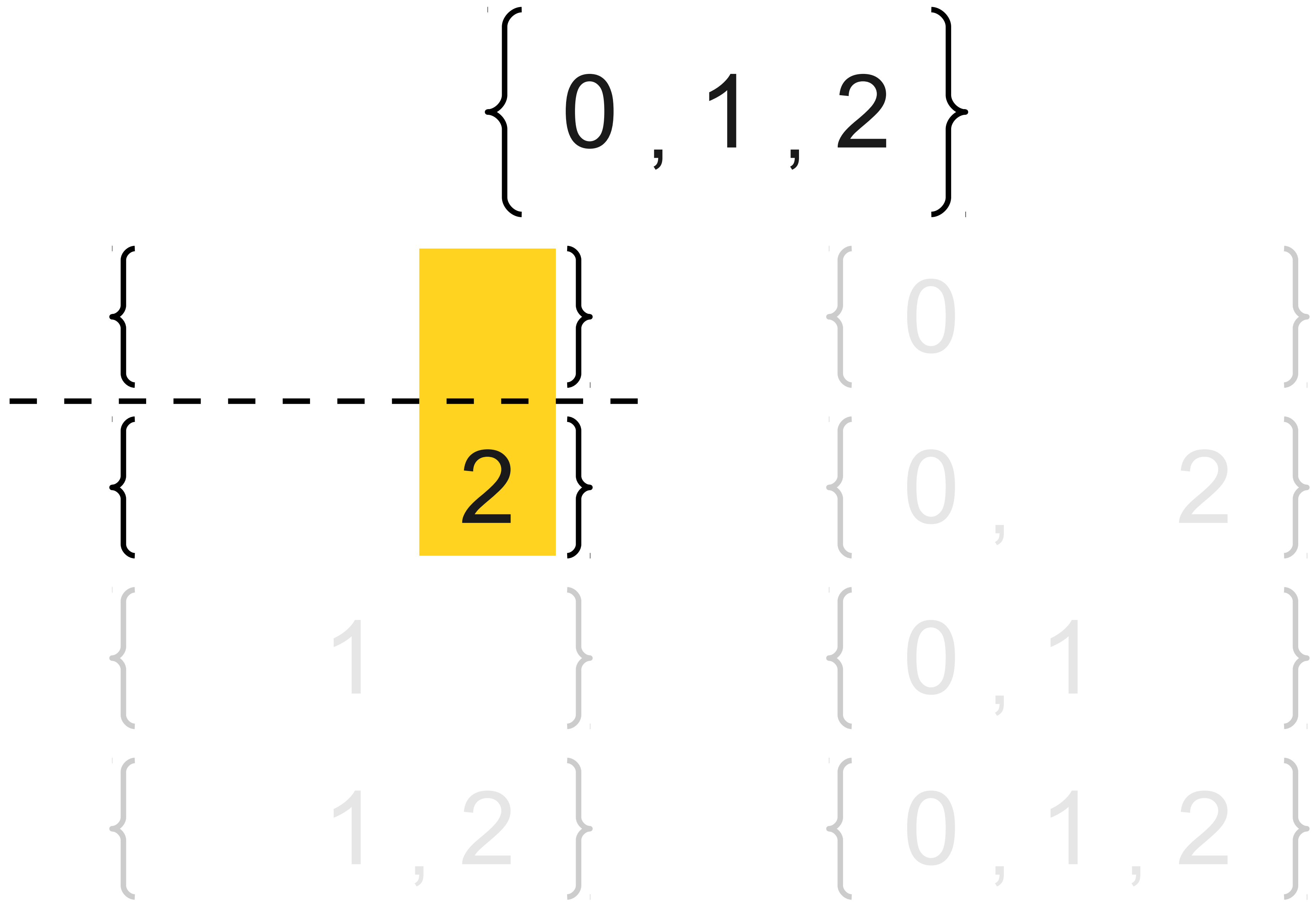
$$\{\ \} \qquad \{0\}$$

$$\{2\} \qquad \{0, 2\}$$

$$\{1\} \qquad \{0, 1\}$$

$$\{1, 2\} \qquad \{0, 1, 2\}$$

Generating Subsets



Generating Subsets

- The only subset of an empty set is the empty set itself.
- Otherwise:
  - Fix some element *x* of the set.
  - Generate all subsets of the set formed by removing *x* from the main set.
  - These subsets are subsets of the original set.
  - All of the sets formed by adding *x* into those subsets are subsets of the original set.

Tracing the Recursion

Tracing the Recursion

{ A, H, I }

Tracing the Recursion

{ A, H, I }

{ H, I }

Tracing the Recursion

{ A, H, I }

{ H, I }

{ I }

Tracing the Recursion

{ A, H, I }

{ H, I }

{ I }

{ }

Tracing the Recursion

{ A, H, I }

{ H, I }

{ I }

{ }

{ }

Tracing the Recursion

{ A, H, I }

{ H, I }

{ I }

`{I}, { }`

{ }

`{ }`

Tracing the Recursion

{ A, H, I }	
{ H, I }	`{H, I}, {H}, {I}, { }`
{ I }	`{I}, { }`
{ }	`{ }`

Tracing the Recursion

{ A, H, I }	`{A, H, I}, {A, H}, {A, I}, {A}`
	`{H, I}, {H}, {I}, { }`

{ H, I }	`{H, I}, {H}, {I}, { }`
{ I }	`{I}, { }`
{ }	`{ }`