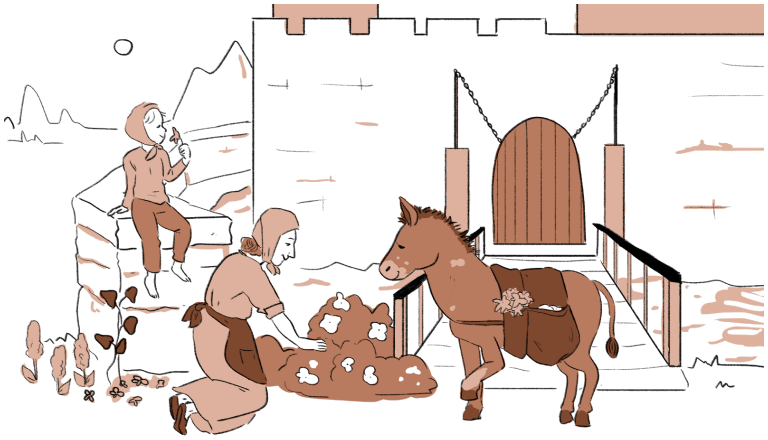
Day 5: Traverse the Maze to Return to the Future

"We have accomplished our missions on the Land of Apple Pi!" The BestFour exclaims as they merge into the gorgeous view.

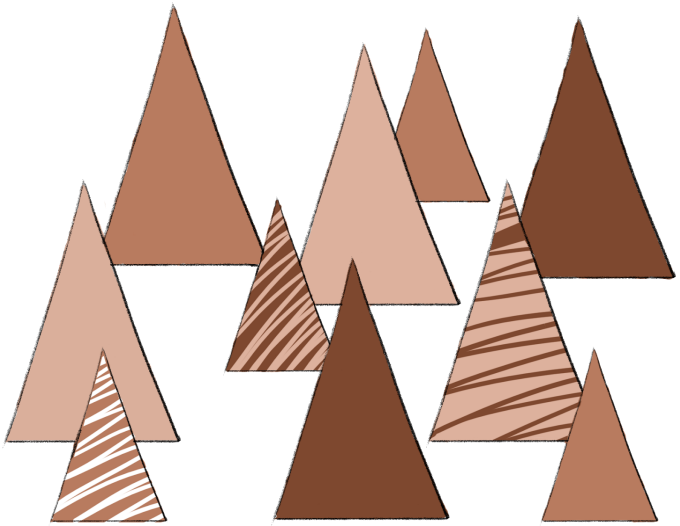


Polymorphic Woods Maze

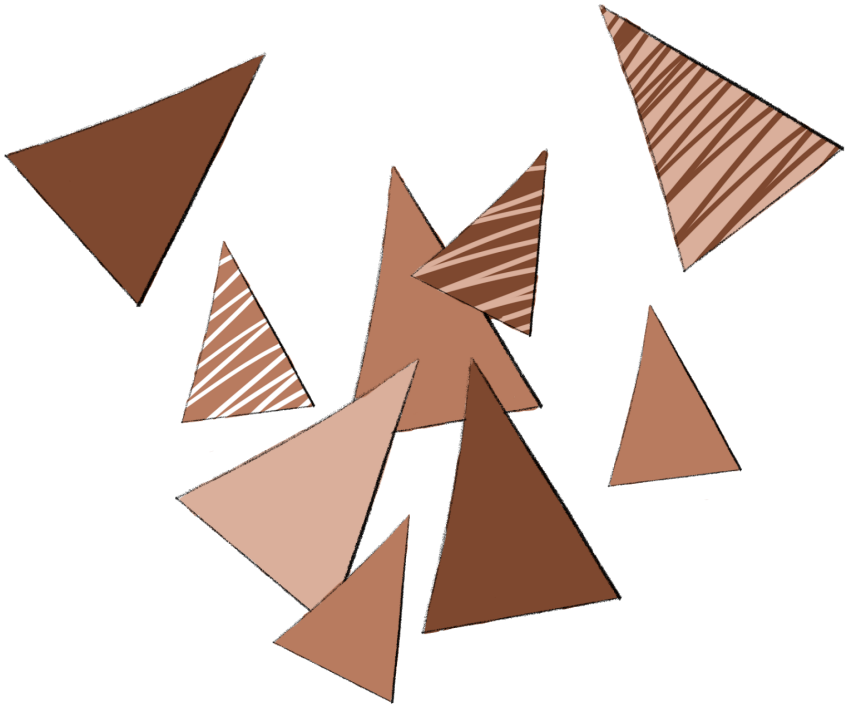
"Now back to the future!" Dark Knight happily shouts, "Before school break, our science teacher told us to enjoy the holiday and that he'll see us in the lab on Monday! We definitely do not want to miss out on phenomenal experiments."

Let’s quickly figure out the entrance to the time tunnel! We already miss our family and friends.

The entrance is hidden in the Polymorphic Woods. In Grandma’s story, the landscape changes every season. When the earth is vibrant, the trees are lined up nicely.



However, when the earth is under weather, the plants will grow in all directions signaling the urgent need for attention.



The BestFour gathers the souvenirs from the Encapsular Metro and takes off to the woods. To their surprise, the Polymorphic Woods is indeed a complicated labyrinth now. Since the Gate of Traceback has been sealed, it'll heal gradually. We can not afford to wait that long. Remember, the time here is a 1000 times S-L-O-W-E-R-R-R!

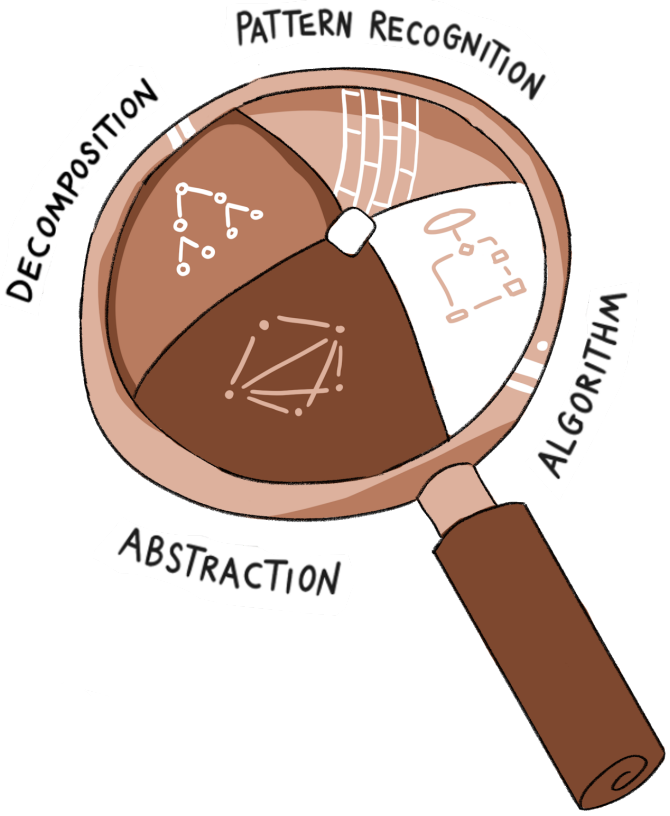
Without further ado, the BestFour begins to figure out the shortest path to traverse through the Polymorphic Woods.

The name labyrinth came from Greek mythology. The Minotaur was a monster with the body of a man and the head and tail of a bull. It was the offspring of the Cretan Queen and a majestic bull. King Minos of Crete had people build a huge maze called Labyrinth to house the monster. Every nine years (several contradictions in the sources regarding this year), they threw seven youths and seven maidens into the Labyrinth to feed the

Minotaur. Eventually, the Athenian hero Theseus went into the maze and slaughtered the monster. He unwound a thread as he went into the Labyrinth and found his way back successfully using the thread [19][20].

Labyrinth did not only exist in Greek mythology. The movie *The Shining* by Stephen King is presented with a labyrinth of hedges. The maze symbolizes the barriers in the family members' connection. Danny, the little boy, has a strong connection with his mother, and they were able to navigate to the center successfully, whereas Danny's father was blocked by the maze. Because of this, he was disoriented and trapped.

Our life is like a maze, a complex branching puzzle with many choices of paths and directions. We constantly face decisions and frequently experience surprises. If we keep an open and growth mindset, many paths will eventually lead to our goals.



The woods have overwhelming details: oak, pine, walnut, cottonwood, buckeye, etc. They all act as obstacles in the maze. CalliLens are able to filter out the unnecessary details and abstract the essential information.

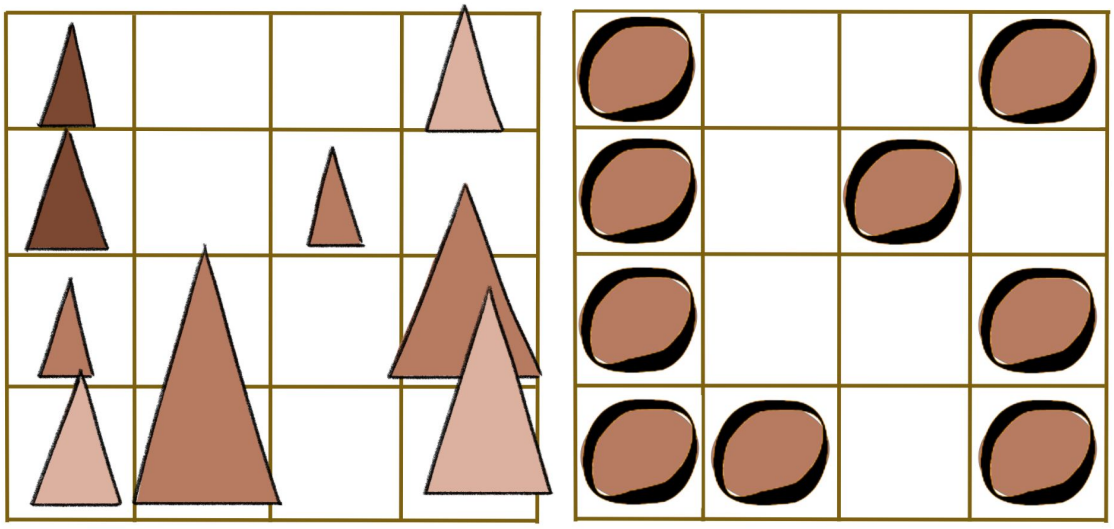


Figure 5.1 Maze: abstraction of the woods

2D Array (list of list) to represent the maze

```
def init_maze():
    maze = []
    maze.append(["#", " ", "O", "#"])
    maze.append(["#", " ", "#", " "])
    maze.append(["#", " ", " ", "#"])
    maze.append(["#", "#", "X", "#"])

    return maze
```

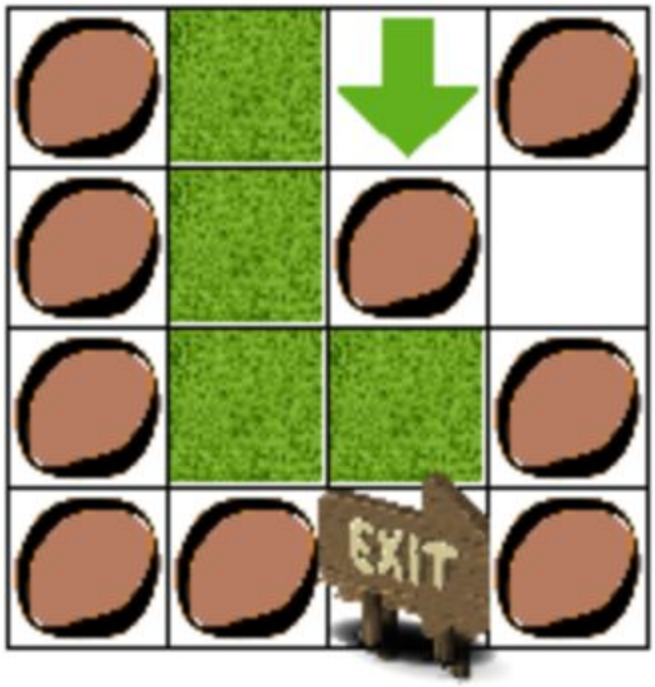


Figure 5.2 Solving the maze

Banana Split finds this question highly intriguing.

Breadth-First Search Buddies With Queue to Find the Shortest Path

No matter how far away the time tunnel is, we can follow the same sequence of steps (algorithm)

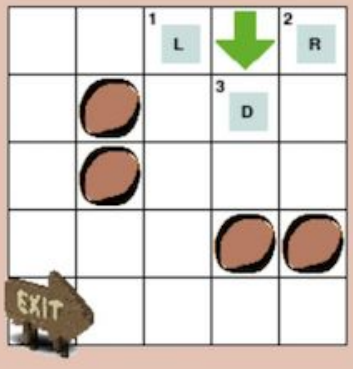
A. Begin at the starting point, explore its adjacent neighbors (left, right, front, back) which is one step away.
B. If the time tunnel is not found, we move to the neighbor locations one by one and further explore its neighbors which are two steps away.
C. Repeat this recognized pattern until the time tunnel is found.
D. Since every time we move one step further from the starting point, it guarantees that the first path found is the shortest path.

Our algorithm here is to search wide first rather than deep. This search algorithm is called **Breadth-First Search**, or **BFS**.

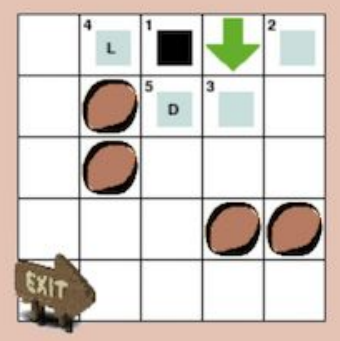
In order to keep track of the neighbors for further exploration, we need to store them in sequence. This requirement reminds us of some daily experience, e.g. waiting in a supermarket checkout line. We walk to the tail of the queue and get served when we move to the front of the queue.

Here is the illustration of a BFS diagram:

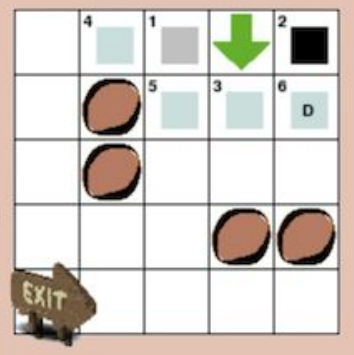
- Number: location exploring/visiting sequence. Fun Fact: computers can teleport across non-adjacent locations using a queue. It remembers the earlier visited locations in a queue, and at some later point, resume from those locations.
- Black: represent the location oriented on which the neighborhood (breadth) is being explored/visited. It's retrieved from the beginning of the queue. We start from the green arrow and only explore the valid neighbors (within boundary, no block, and not visited yet). We check whether the neighbors contain exit or not. If yes, the game ends. Otherwise, the neighbors are added to the end of the queue.
- Blue: represent locations in the queue. These locations have been explored/visited, but their neighbors (breadth) are not explored yet.
- Grey: represent the locations removed from the queue and their neighbors have already been explored



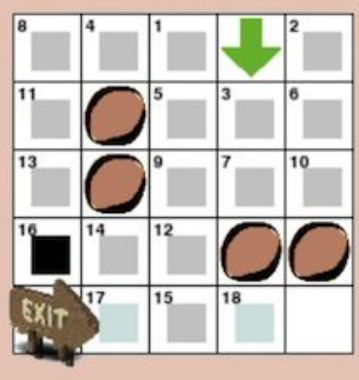
1. Start exploring neighbors (LRUD)
Queue: (0,2) (0,4) (1,3)
Therefore called *Breadth-First Search*



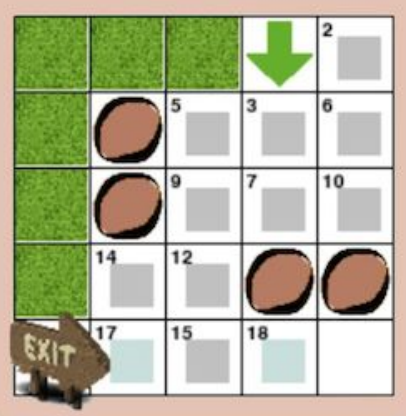
2. Retrieve from Queue
Explore its neighbors
Queue: (0,4) (1,3) (0,1) (1,2)



3. Repeat step 2
Queue: (1,3) (0,1) (1,2) (1,4)



4. Find EXIT



Mission accomplished!

Figure 5.3 Breadth-First Search Steps Illustration

Queue

In the computer, when we need to maintain the order of several data items and retrieve them one by one based on their arrival order, we use a **queue**: **first in first out (FIFO)**!

Voilà, the problem is now solved! We store the already searched locations into a queue. The newly searched location is added to the tail of the queue. Each time we retrieve one location from the head of the queue and expand to its surrounding area, we then add the expanded area to the queue until we find the target.

[queue diagram - (queue diagram, head, tail, FIFO)]

Queue operations

- CreateQueue(): creates an empty queue
- Enqueue(queue, item): adds an item to the tail of the queue
- Dequeue(queue): removes an item from the head of the queue and makes the next item the new head. If the queue is empty, Dequeue() generates an error.
- IsQueueEmpty(queue): returns True if the queue is empty, False otherwise.

Optimization to Prevent Repeated Paths

In the busy maze traversing process, we may forget the visited locations, and end up repeating the same routes. Why don't we leverage the superb memory of our friend, computer? We just need to create a **visited** data storage to mark the visited locations. This saves tremendous time!

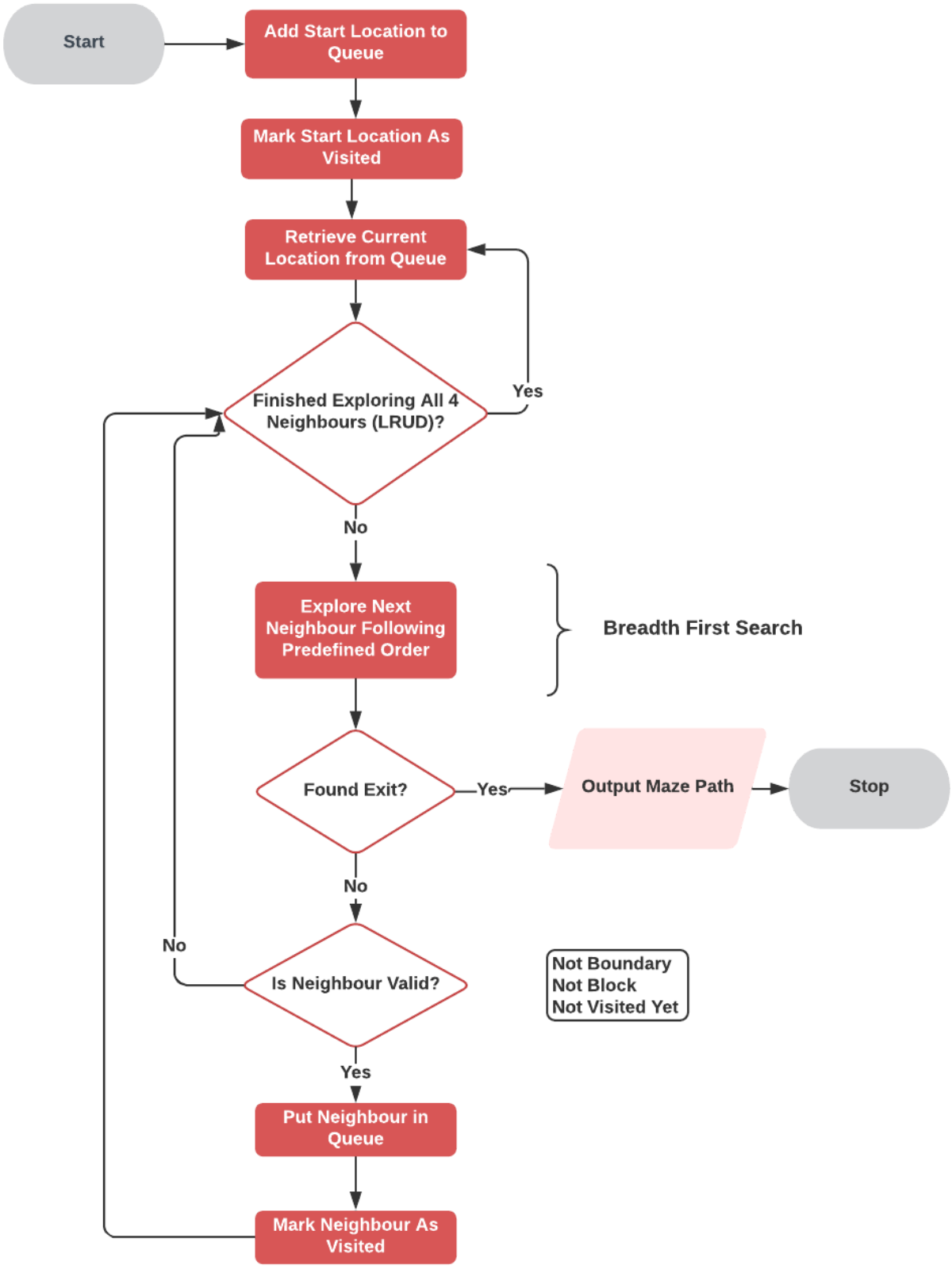


Figure 5.4 Flowchart: Breadth-First Search to solve Maze Problem

Mighty Python

Maze Representation
Maze.py at https://github.com/applepiinc/arithmaticthinking/tree/main/maze/2Darray

BFS Implementation
mazesolver_BFS.py at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Maze Pygame
pygame_mazesolver_BFS.py and related 5 images (start, end, wall, grass, path) at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

DIY Maze Game

Maze Demo
pygame_mazesolver_BFS_DemoPlayModes.py at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

We invite you to play this maze game. You can easily customize the maze with your favorite scene by changing the blocks and path icons in pygame_mazesolver_BFS_DemoPlayModes.py.

1. Choose your maze blocks

   [ToDo: add screenshot of pygame]

   Figure 5.5 Pygame screenshot: choosing maze blocks

2. Design the layout of your maze

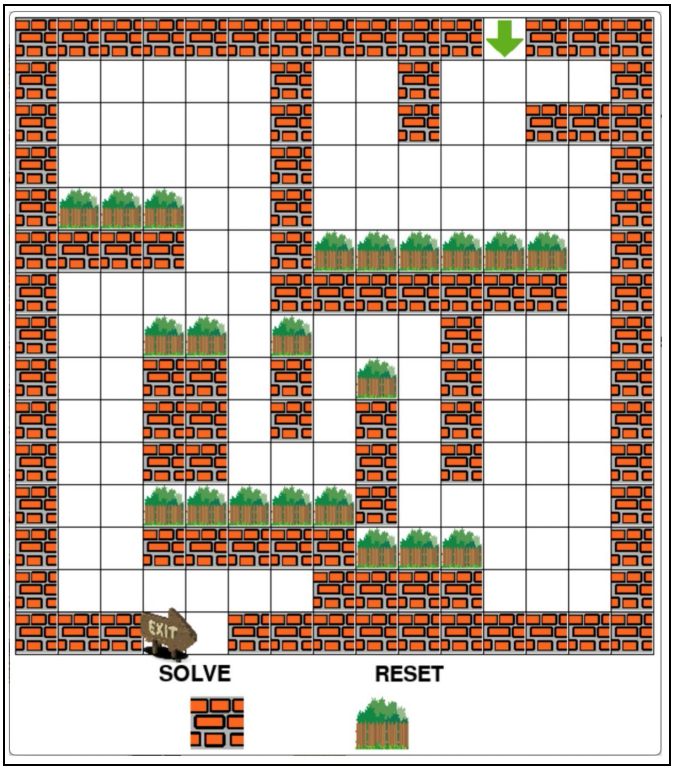


Figure 5.6 Pygame screenshot: designing maze layout

3. Select Demo Mode, and then click Solve button

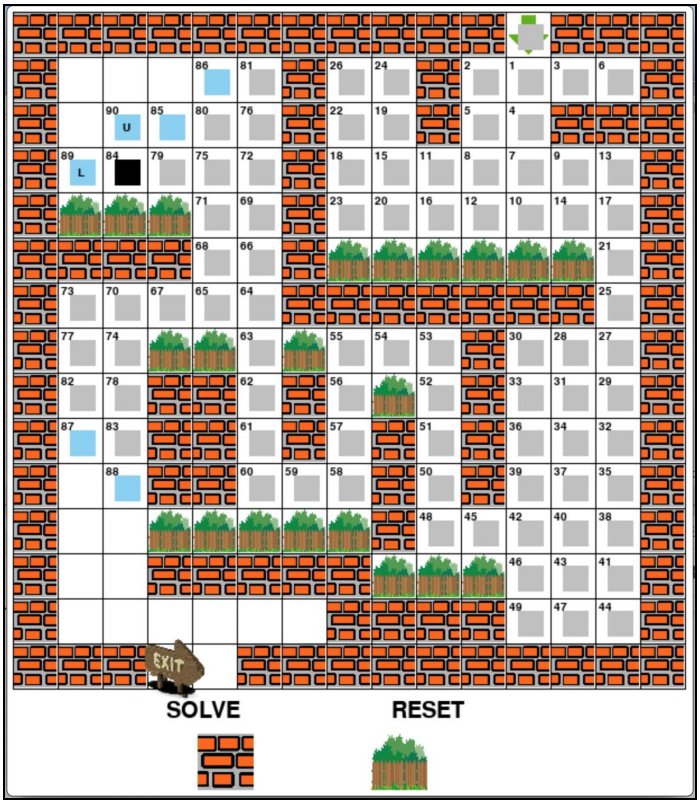


Figure 5.7 Pygame screenshot: displaying BFS steps in demo mode

4. Your maze solved!

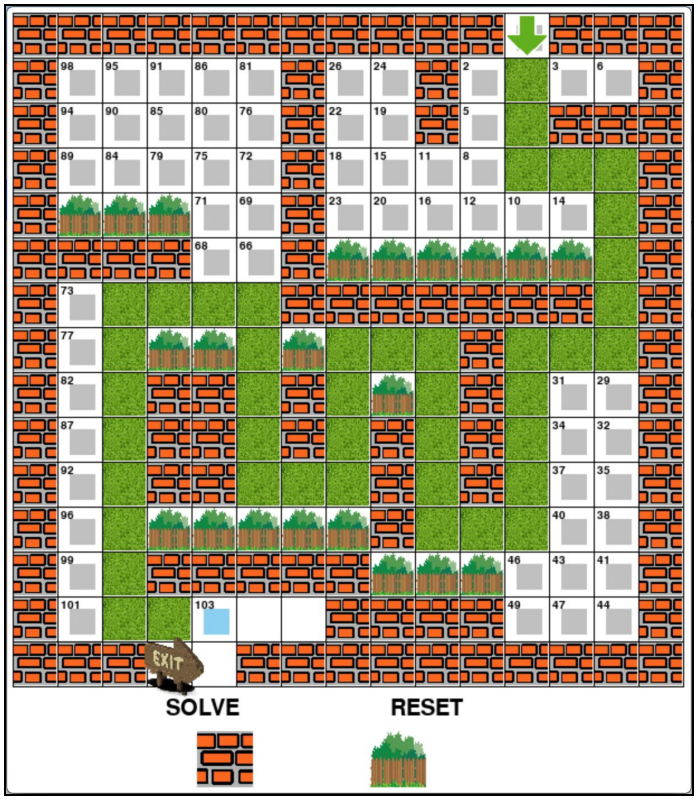


Figure 5.8 Pygame screenshot: showing path to exit

Depth-First Search Friends With Stack to Locate All the Paths

Well, if in BFS we go breadth first, is there an algorithm to go depth first?

Yes, there is an algorithm that prioritizes depth.

Let's use the same maze like the one solved by BFS. However, only change the block to a wall so that we can differentiate the two illustrations.

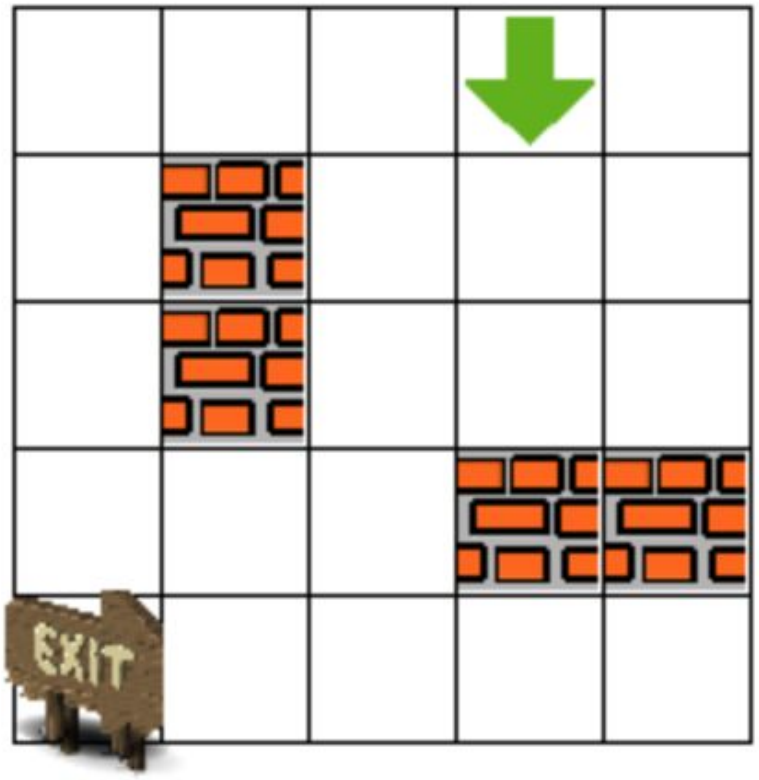
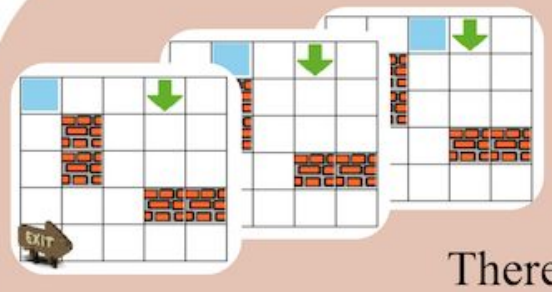


Figure 5.9 Maze to be solved using Depth-First Search

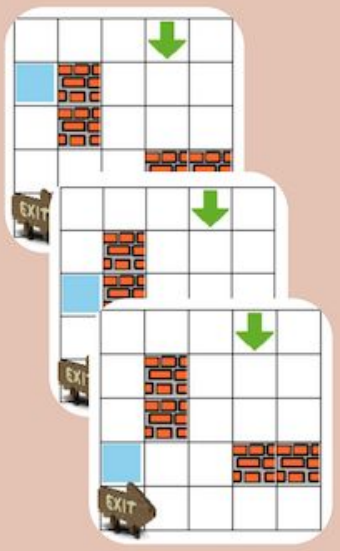
The program will explore the left neighbor first, and then the left neighbor's left, and then the left's left's left… It goes down one direction until the end of the route before exploring other directions. Therefore, this algorithm is called **Depth-First Search**, or **DFS**.



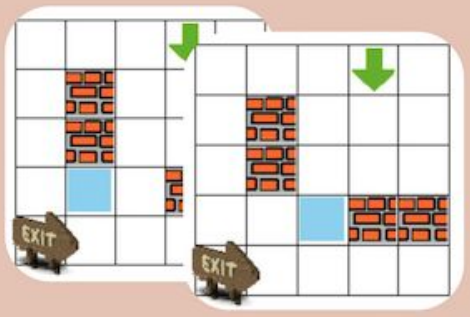
1. Start exploring Left

2. Continue Left until boundary

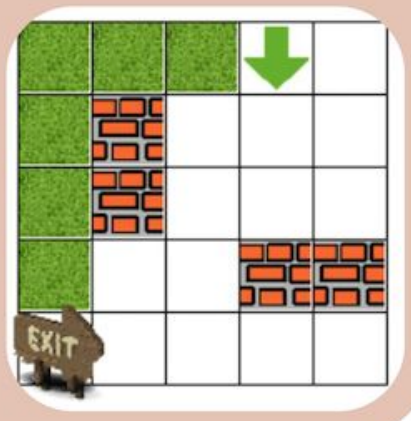
Therefore called *Depth-First Search*



3. Continue Down
Left? invalid
Right? invalid
Up? invalid (visited)
Down? valid



4. Continue Right
Left? invalid
Right? valid



After exploring all the possible paths, select the shortest one.

Mission accomplished!

Figure 5.10 Flowchart: Depth-First Search to solve Maze Problem

DFS is not a good fit in this case because the program has to explore all possible paths to find the shortest one; but also because it constantly revisits the same sub-paths. Overlapping subproblem issue? Ha! We are so familiar with using a Dynamic Programming memoization table to solve this problem! Remember our Day 2 School Encounter Adventure? We trust that you can optimize the duplicated sub-paths in DFS with DP.

Can you believe it? DFS needs to exhaust 64 paths to determine the shortest one!

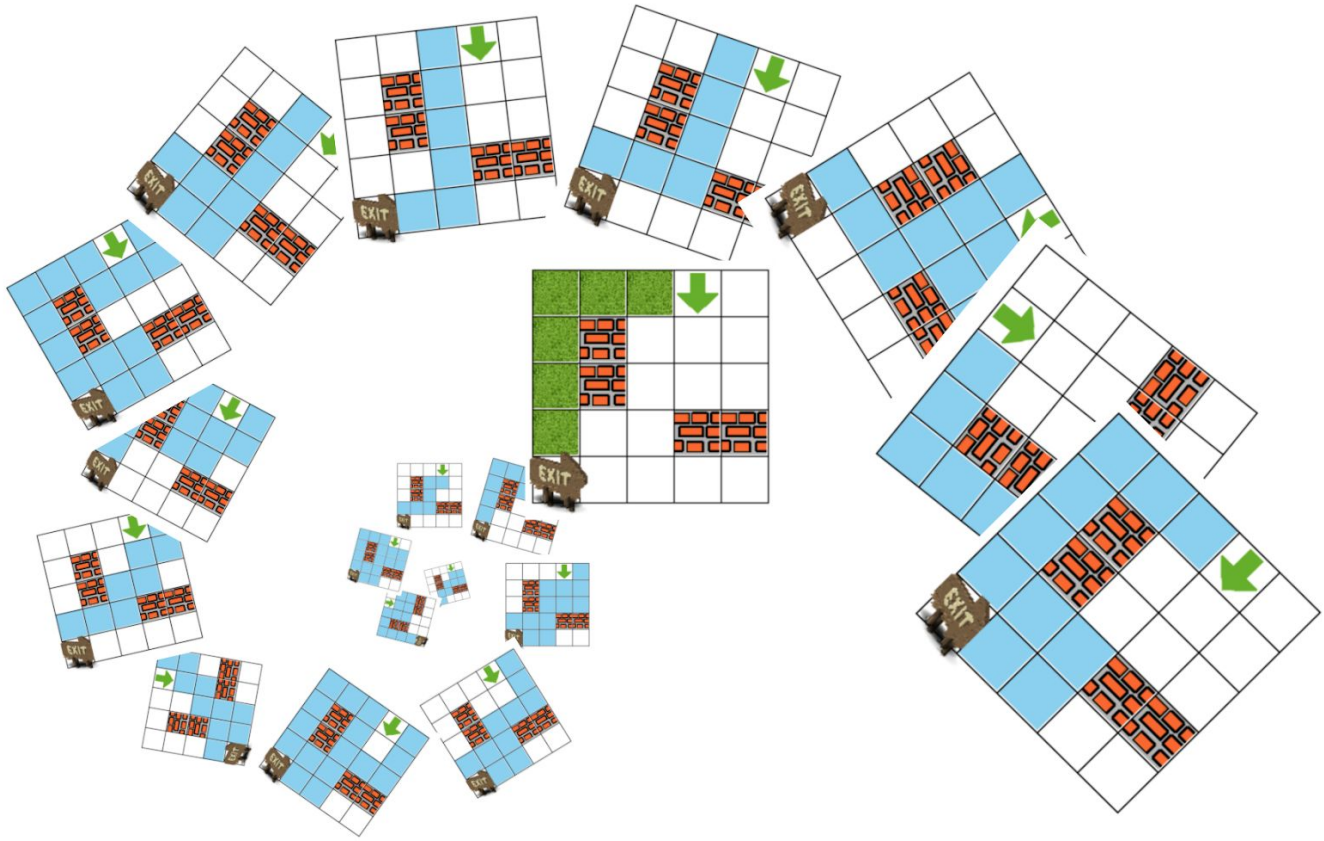


Figure 5.11 Flowchart: exploring all paths in Depth-First Search

Stack and Recursion

BFS needs to keep all the current nodes in the memory, so this algorithm is not preferred in a high branching graph.Otherwise, the computer will run out of memory very quickly.

Since DFS is a recursive algorithm based on the concept of stack, this algorithm is not preferred on super deep graphs, or the computer will use up its memory quickly.

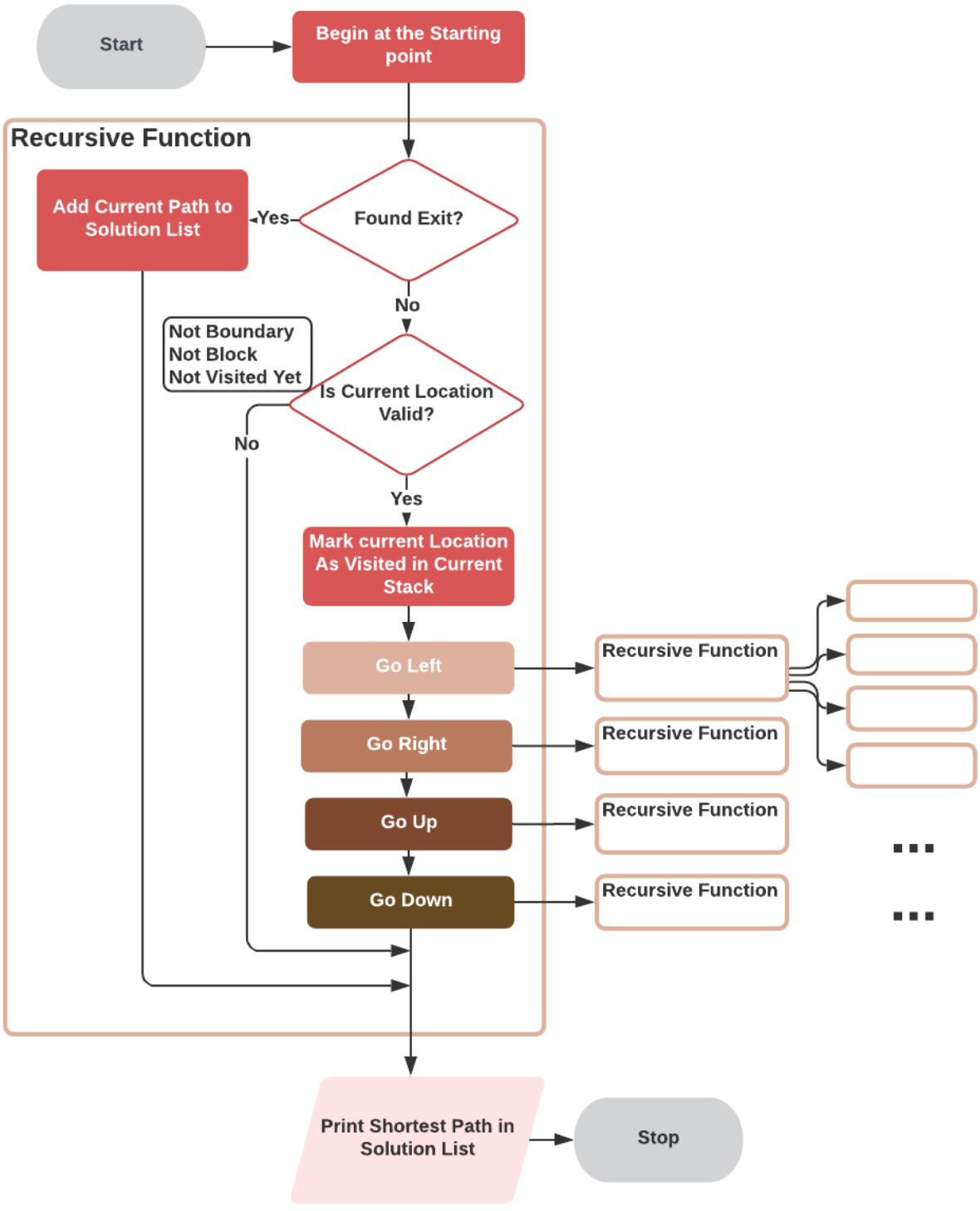


Figure 5.12 Flowchart: Depth-First Search to solve Maze Problem

Mighty Python

DFS Implementation
mazesolver_DFS.py at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Maze Pygame
pygame_mazesolver_DFS.py and related 5 images (start, end, wall, grass, path) at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Maze Demo
pygame_mazesolver_DFS_DemoPlayModes.py
path) at
https://github.com/applepiinc/arithmaticthinking/tree/main/maze

Breadth, Depth
Graph Search Twins!

B buddies with queue

D friends with stack

Graph is our playground

As we explore through the paths

B: The shortest is my first path

D: Whereas the largest is my number

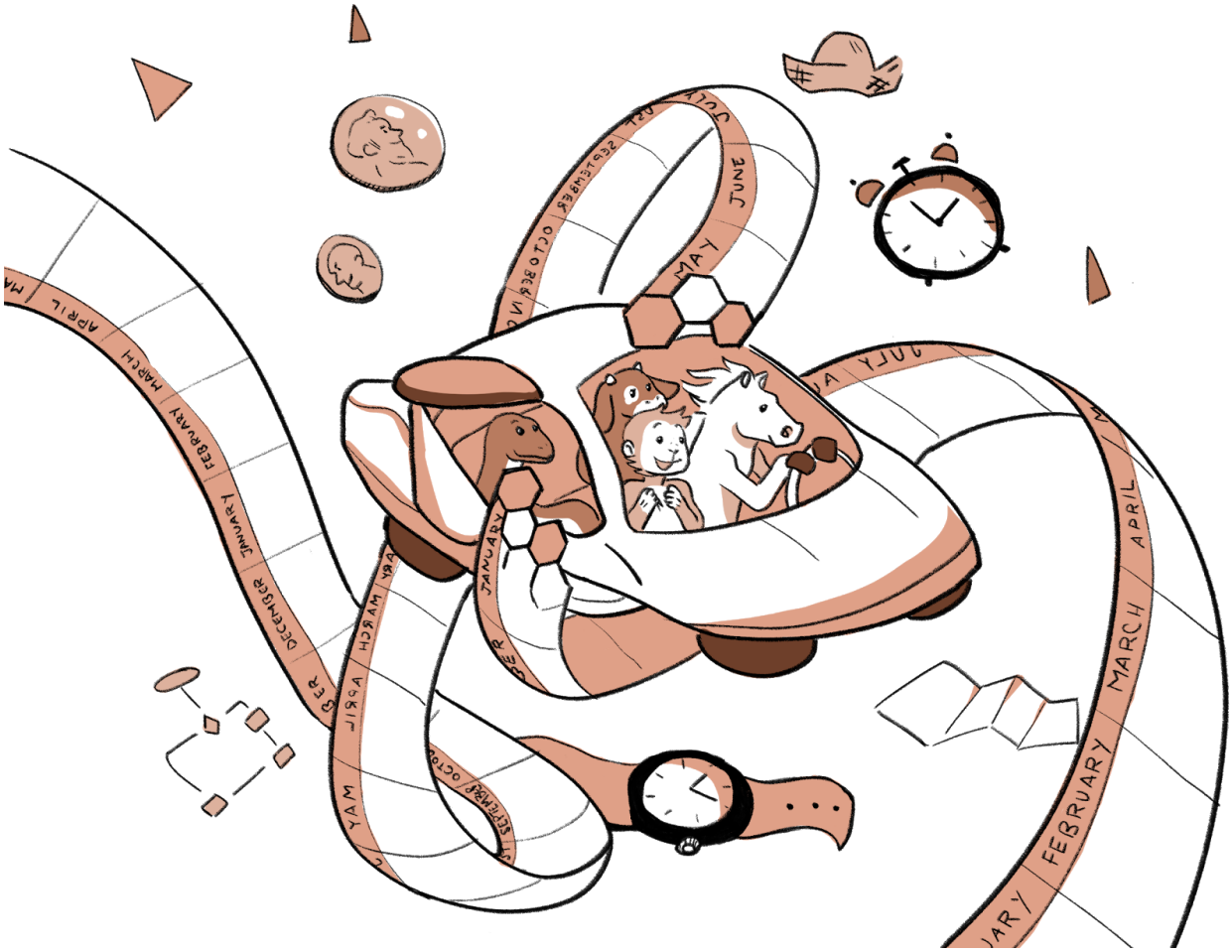
How should the twins compare?

You got to be fair

Each has a role for what you care!

Here we go, the hidden entrance to the time tunnel!

The BestFour's time vessel sputters… flickers… and turns on! ...Rumbling to the future—



What an adventure!