For Lesson Three the same code used for Lesson Two will be used. This includes modifications to the original Project code which did two things:

- Turned off the 'backtrack' capability
- Turned off the 'Erase the Walls' capability.

In Lesson Three, we will turn back on the 'Erase the Walls' capability and watch the video explaining how the Walls are eliminated for the DFS path solving algorithm.

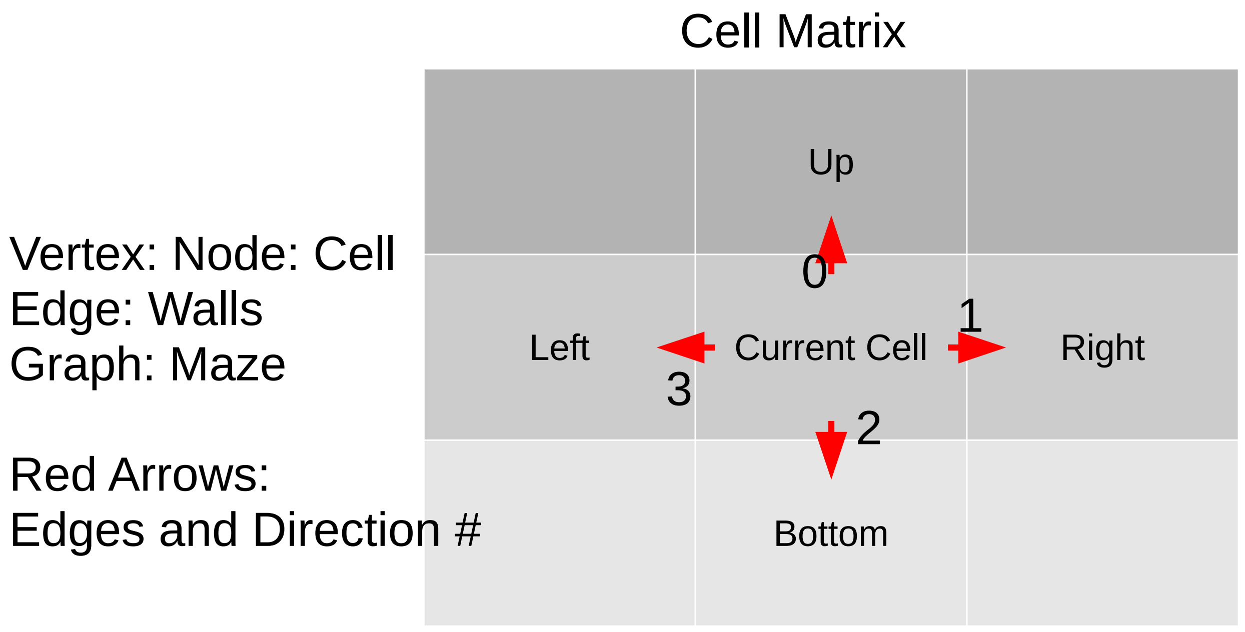


Illustration 1: Cell and its neighbors

Illustration 1 shows a graphic description of the cell layout of the DFS solution. This is the same orientation used in the Coding Train videos. The Matrix is a collection of Cells. Each cell object contains data members which can tell if it is visited or not, and an array indicating which Walls Exist.

The connection between cells is the Edge Direction number. This is unique to the wxDFS3 solution. If movement is from the Current Cell to the Right, then the Right cell would have a 'from' data member indicating a direction of 1 which is the Edge direction the cell was entered from.

The 'from' direction member for each cell creates a linked list. The links in the cells give a list which enables solving the matrix from any chosen destination cell to the DFS origin. The solution is accomplished by following the list of 'from' edges for each cell all the way back to the origin.

In this Lesson, we will be clearing the Walls for the Current Cell and also the 'from' Cell. Both cells are neighbors and each Cell has a wall, so both have to be cleared. This occurs in lines 356-375 where the Current cell and Parent cell Wall values are set to 'false' – so the walls are cleared.

Watch the Video – take the time and watch the video!

Coding Challenge #3: Select Part 3 video
https://thecodingtrain.com/challenges/10-dfs-maze-generator

Video: time Remove Walls	Main.cpp Lines
Part 1 review 00:00 Remov walls and mark current cell as visited and chosen	`Source split into 4 files: wxDFS3App.h, wxDFS3App.cpp, wxDFS3Main.h, wxDFS3Main.cpp`
summary 00:00	
Algorithm review 00:35	`Algorithm is the State Machine in Line 158 OnTimer1Trigger(wxTimerEvent& event)`
01:28	

01:40 Each cell object has array of walls	`Line 48 wxDFS3Main.h`
	`bool WallFlag[4]; //0 top, 1 right, 2 bot, 3 left`

Map of walls (edges) and vertices 0,1,2,3	
02:00 Diagram of edges (walls)	
03:00 Right/Left or Up/Down	
03:23 Steps 1 and 4 of algorithm	

03:50 Setp 3 of algorithm and RemoveWalls(current, next)	`lines 356-375 Pop New Current cell, remove edge Walls between Parent and Current cells`
	`Cases 0-3`

04:50 Generic function to remove walls left/right	
06:42 Same for top/bottom	
07:50 Try to run Remove Walls code	`line 518-538 for drawing the Maze and Cell walls`
08:23 Debug code operation print to log	
09:27 Rectangle drawing edges too	
09:48 Proper wall removal	
10:05 Current color highlight	
10:50 Gets hung needs backtrack	`No neighbors, part 4 to complete solution`
Go to Part 4	

Great short alternative explanation for Recursive Backtracker (less than 9 minutes)
https://www.youtube.com/watch?v=elMXlO28Q1U

Recursive backtracker

Depth-first search
Easy to implement
Uses a stack
Few dead-ends

Illustration 2: Great Youtube algorithm explanation

Assignment Setup:

Use the code from Lesson Two.

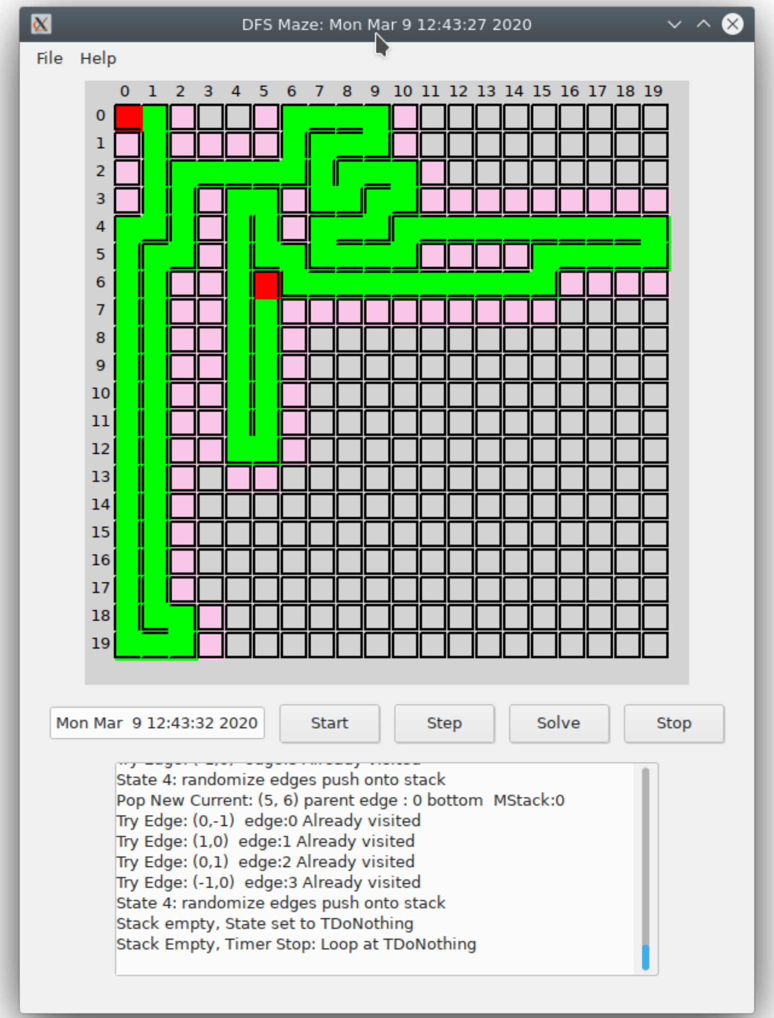
1. In Lesson Two, we turned off erasing Walls by commenting out line 347 and replace with
   ```
   //switch(current.from)
   switch(5)
   ```

2. Now turn back on erasing Walls by uncomment line 347 while comment out the line added in Lesson Two.
   ```
   switch(current.from)
   //switch(5)
   ```

3. Compile and run, you will see the walls of each cell in the are path erased:

4. Run this new version, screen copy, and turn in as your assignment submission.

5. Obviously, your run will be different than this one. See below Illustration 3.
   The program runs until it gets trapped with no neighbors are available but the walls of the path have been removed.



```
State 4: randomize edges push onto stack
Pop New Current: (5, 6) parent edge : 0 bottom  MStack:0
Try Edge: (0,-1)  edge:0 Already visited
Try Edge: (1,0)  edge:1 Already visited
Try Edge: (0,1)  edge:2 Already visited
Try Edge: (-1,0)  edge:3 Already visited
State 4: randomize edges push onto stack
Stack empty, State set to TDoNothing
Stack Empty, Timer Stop: Loop at TDoNothing
```

Illustration 3: Lesson Three output without walls