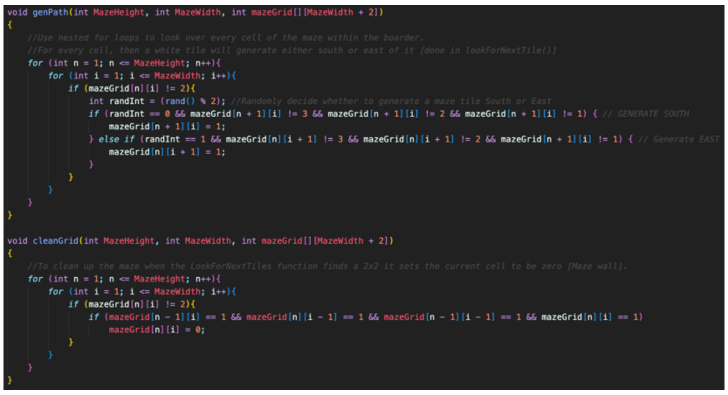
COMP0002 – C Coursework – 2022

The programme is split into 3 separate files – MazeGen.c, MoveRobot.c and MazeMain.c

MazeGen.c is used to randomly generate a maze. The maze generator begins by taking the empty 2D array mazeGrid[][MazeWidth + 2] and initialising it with a boarder of 3's, with the rest filled with 0's. The size of the array is defined as MazeWidth + 2 to account for this boarder which is not part of the maze. The genEnding function then generates the finish point of the maze at a random point in the final column and assigns it the value 2 in the array.

To create the paths of the maze there are two functions - genPath and cleanGrid:



```c
void genPath(int MazeHeight, int MazeWidth, int mazeGrid[][MazeWidth + 2])
{
    //Use nested for loops to look over every cell of the maze within the boarder.
    //For every cell, then a white tile will generate either south or east of it [done in lookForNextTile()]
    for (int n = 1; n <= MazeHeight; n++){
        for (int i = 1; i <= MazeWidth; i++){
            if (mazeGrid[n][i] != 2){
                int randInt = (rand() % 2); //Randomly decide whether to generate a maze tile South or East
                if (randInt == 0 && mazeGrid[n + 1][i] != 3 && mazeGrid[n + 1][i] != 2 && mazeGrid[n + 1][i] != 1) { // GENERATE SOUTH
                    mazeGrid[n + 1][i] = 1;
                } else if (randInt == 1 && mazeGrid[n][i + 1] != 3 && mazeGrid[n][i + 1] != 2 && mazeGrid[n + 1][i] != 1) { // Generate EAST
                    mazeGrid[n][i + 1] = 1;
                }
            }
        }
    }
}

void cleanGrid(int MazeHeight, int MazeWidth, int mazeGrid[][MazeWidth + 2])
{
    //To clean up the maze when the LookForNextTiles function finds a 2x2 it sets the current cell to be zero [Maze wall].
    for (int n = 1; n <= MazeHeight; n++){
        for (int i = 1; i <= MazeWidth; i++){
            if (mazeGrid[n][i] != 2){
                if (mazeGrid[n - 1][i] == 1 && mazeGrid[n][i - 1] == 1 && mazeGrid[n - 1][i - 1] == 1 && mazeGrid[n][i] == 1)
                    mazeGrid[n][i] = 0;
            }
        }
    }
}
```

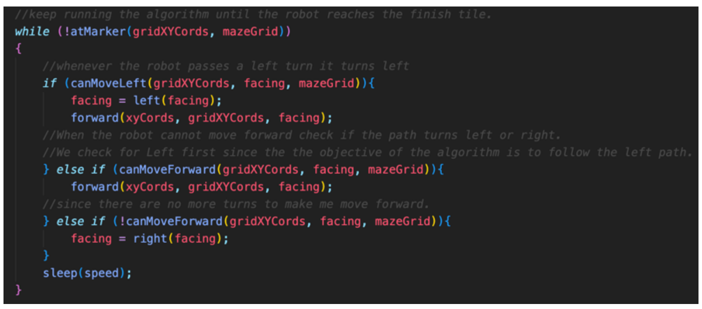
genPath loops through every cell in the grid, randomly generating a path tile (denoted by 1 in the array) either south or east for each cell.

cleanGrid loops through every cell looking for 2x2 squares of path tiles – if one is found the path tile from that cell is removed, clearing large areas of white space. This ensures the path is only one tile wide.

Finally, the testMaze function is used to check that the maze generated has a complete path to the finish, although this is called separately in MazeMain.c. If the maze doesn't have a complete path a new maze is generated. This function works by running the robot in the background (with no delay) until the maze finish is found or until the robot loops back-round to the start, hence showing there is no complete path to the finish.

Functions are then defined in MoveRobot.c to allow the robot to navigate the maze: forward, left and right as well as canMoveForward and canMoveLeft to check for the robot's next move. The function atMarker allows the programme to recognise when the robot is at the finish and displayRobot displays the robot on the screen at its current coordinates and orientation.

MazeMain.c first calls testMaze until a working maze is generated, then the drawMaze function to display the random maze from the array using the drawapp library. The robot Function then uses a loop to solve the maze:



```c
//keep running the algorithm until the robot reaches the finish tile.
while (!atMarker(gridXYCords, mazeGrid))
{
    //whenever the robot passes a left turn it turns left
    if (canMoveLeft(gridXYCords, facing, mazeGrid)){
        facing = left(facing);
        forward(xyCords, gridXYCords, facing);
    //When the robot cannot move forward check if the path turns left or right.
    //We check for Left first since the the objective of the algorithm is to follow the left path.
    } else if (canMoveForward(gridXYCords, facing, mazeGrid)){
        forward(xyCords, gridXYCords, facing);
    //since there are no more turns to make me move forward.
    } else if (!canMoveForward(gridXYCords, facing, mazeGrid)){
        facing = right(facing);
    }
    sleep(speed);
}
```

This loop makes use of the functions in MoveRobot.c to solve the maze using the left-turn rule. At any point in the maze the robot checks whether it can move left, doing so if it can, otherwise it will move forward. If the robot reaches a point where it cannot move left or forward it will check whether it can turn right since there might just be a right hand turn in the path. Finally, if it cannot do this, it must be at a dead-end and therefore it turns to comeback the way it came, while continuing to look for left turns before making any move. This while loop will continue until robot finds the finish line, then ending the programme.

To compile the programme use the command: **gcc -o Maze MazeMain.c MoveRobot.c MazeGen.c graphics.c** then to run the programme use the command: **./Maze | java -jar drawapp-2.0.jar**