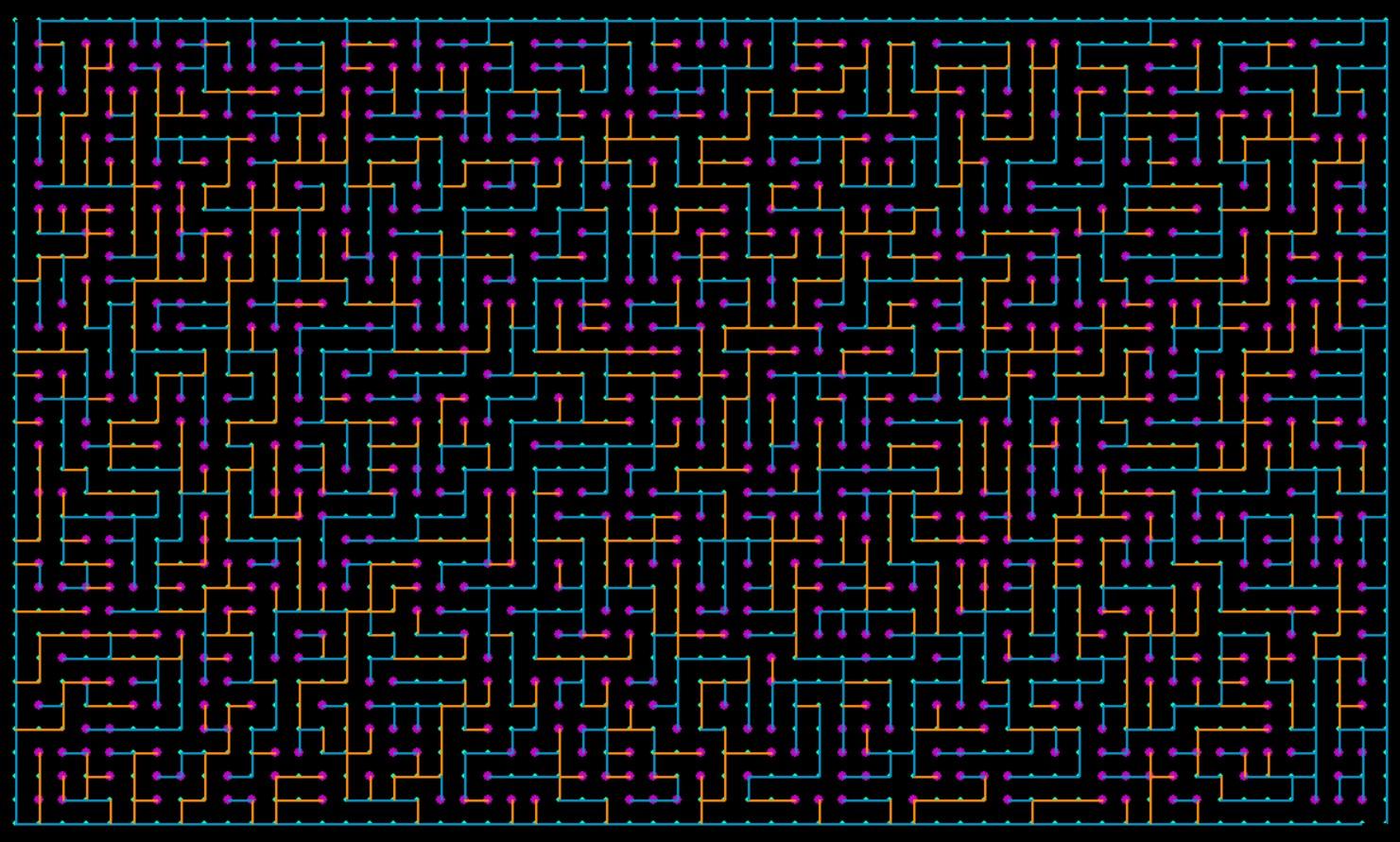
**National Public School 2020-21**

**Computer Science Project**

MAZE GENERATOR

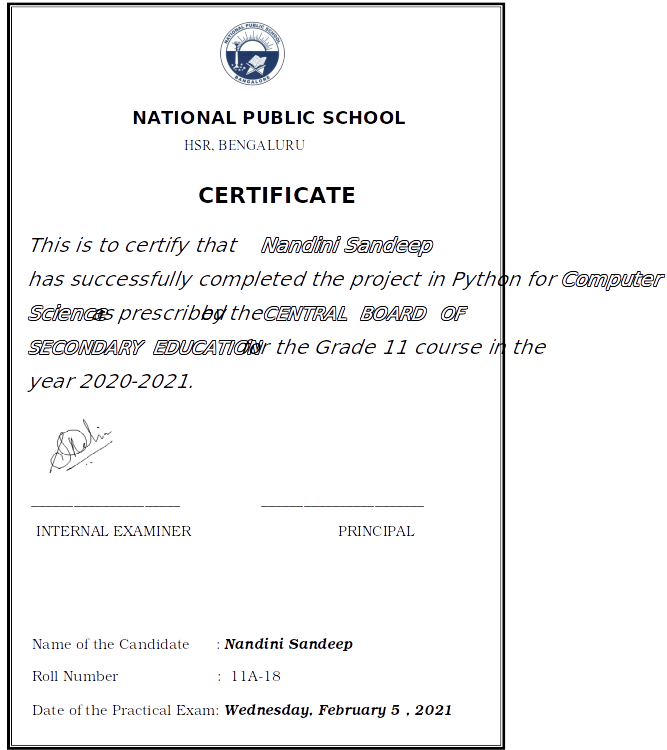
**N**

**NANDINI SANDEEP AND SHREYA RAO**

**CLASS 11A, ROLL 18 and 32.**

**2020-21**

**CERTIFICATE**

****

**ACKNOWLEDGEMENTS**

We are extremely grateful to our teacher, Nalina ma’am for her valuable guidance and support. We would also like to thank Shefali ma’am, who gave us the golden opportunity to do this wonderful project. It helped us learn a lot about python, and programming games. Last but not least, we thank our family and friends, who helped us successfully complete this project within the limited time frame.

Nandini and Shreya**CONTENTS**

| **Sl. no.** | **Topic** | **Pg. no.** |
| --- | --- | --- |
| 1 | Overview of Python | 5 |
| 2 | Project Synopsis | 6 |
| 3 | Algorithm | 6 |
| 4 | System Requirements | 10 |
| 5 | Modules and Functions | 10 |
| 6 | Instruction Manual | 11 |
| 7 | Program Code | 13 |
| 8 | Program Output | 39 |
| 9 | Limitations | 47 |
| 10 | Bibliography | 48 |

1. **OVERVIEW OF PYTHON**

Python is a high-level, interpreted, interactive and object-oriented scripting language. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

***History of Python***

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages. Python is copyrighted. Python is now maintained by a core development team at the institute.

***Python Features***

* Easy-to-learn
* Easy-to-read
* Broad standard library
* Interactive Mode
* Portable
* Extendable
* Databases
* GUI Programming

1. **PROJECT SYNOPSIS**

We have created an algorithm that draws complex mazes. These mazes are randomized and there are millions of different possible mazes with our algorithm. There is no bias towards any type of solution in the mazes.

We have developed a game where you can test your maze solving skills against the clock. You can choose from 3 different difficulty levels and even customize your line color. But be warned, a wrong move can result in a loss. However, you can reset your solution and try again until you complete the maze.

To prevent cheating, you cannot retry a maze which you have already solved. You can, however, try a new maze instead. Can you solve a level 3 maze in under 30 seconds?

1. **ALGORITHM**

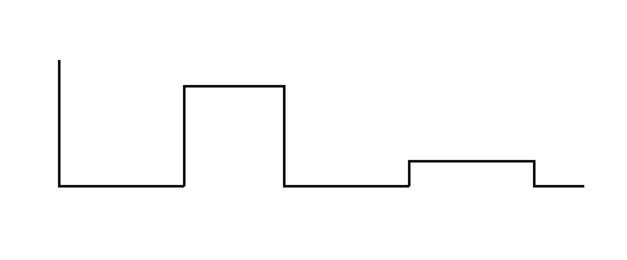
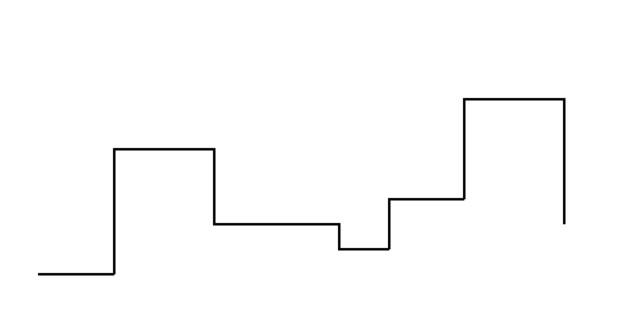
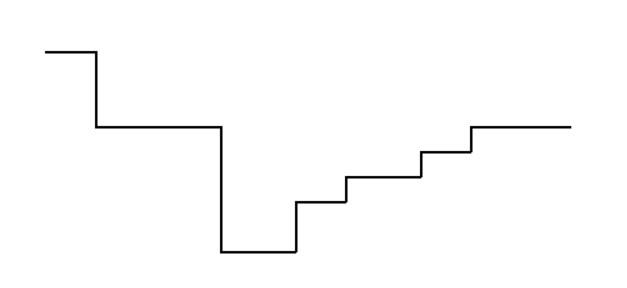
Our algorithm is completely original. It is based on a method I use to draw mazes on paper. It is faster and generates more random mazes than the already existing maze generation algorithms.

We have taken all the factors into consideration to ensure that all the mazes are solvable. These mazes will always have only one solution. However, larger mazes take much longer to generate. Mazes generated by our algorithm are much superior to those drawn by hand, because the solutions are much more random and unpredictable.

This is a detailed description of the working of our algorithm.

1. A 2D list of coordinates is created to keep track of the locations within the maze. These are the points at which a line can begin. All coordinates with lines on them are marked ‘closed’, and the remaining are marked ‘open’.
2. The program begins by drawing a box with two openings- the starting point and the ending point. (the points through which these lines pass through are now marked ‘closed’)
3. A random coordinate is chosen from the list of ‘open’ coordinates.
4. The program randomly generates a list of ‘*maze like snake*’ moves for the walls of the maze.
5. ***How snake moves work-***
   1. Snake moves were created to avoid structures like spirals, excessive stairs and long corridors which are a complete waste of space in a maze. These lines also help us avoid creating completely isolated areas
   2. The main direction of the snake is chosen. For example, let us choose up as the main direction. Now, the line cannot move down
   3. If the line goes left, it can continue to go left or go up. If the line goes right, it results in a useless pointy structure
   4. The number of lefts and rights are regulated to avoid stairs or very diagonal lines that waste space
   5. The line takes a randomly generated number of steps in the chosen direction to avoid long corridors
   6. While stairs and long corridors can be created by the merging of multiple snakes, they are now less probable and contribute more to the difficulty of the maze.
   7. The resultant line usually resembles a snake
6. The snake line is drawn until it joins another line.
7. More snake lines are drawn until all the coordinates have been filled.
8. This results in a maze

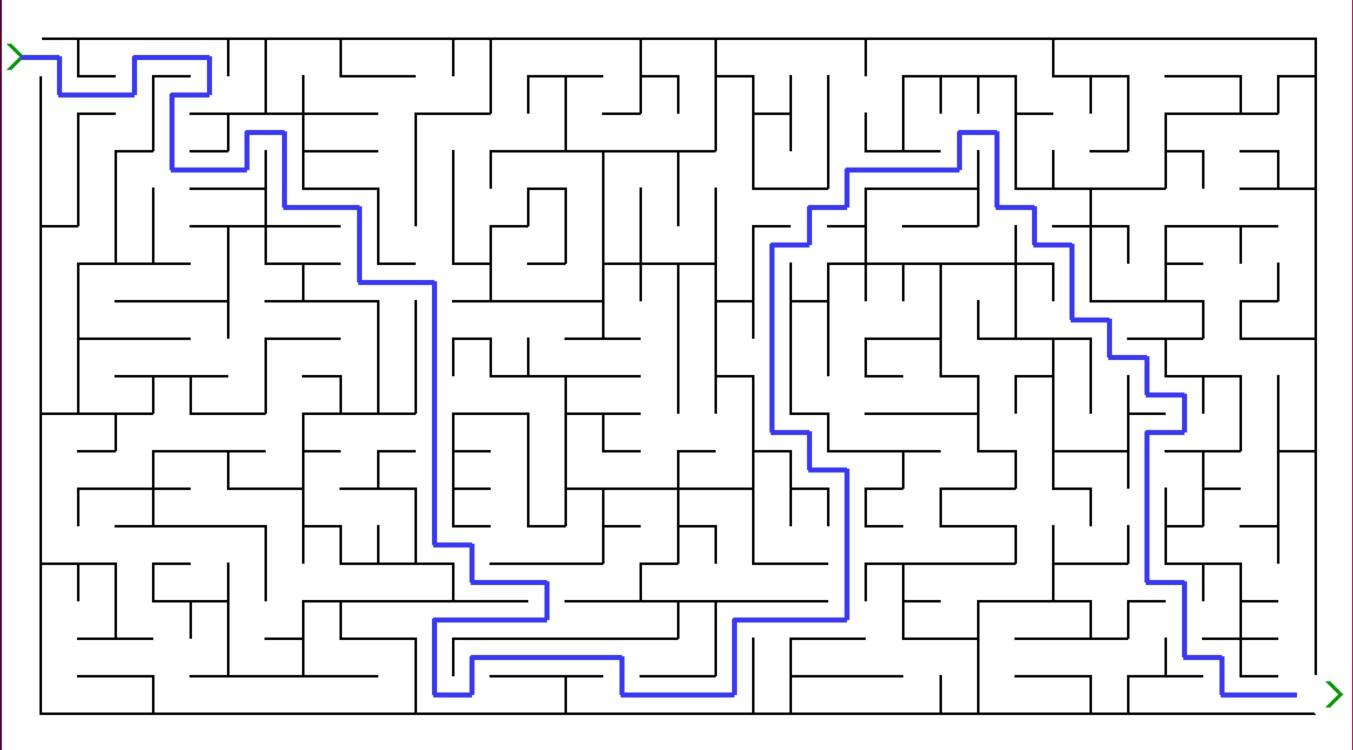
Since each line is connected to only one other line, there is always a solution. Since no line is left not connected, there is only one solution.

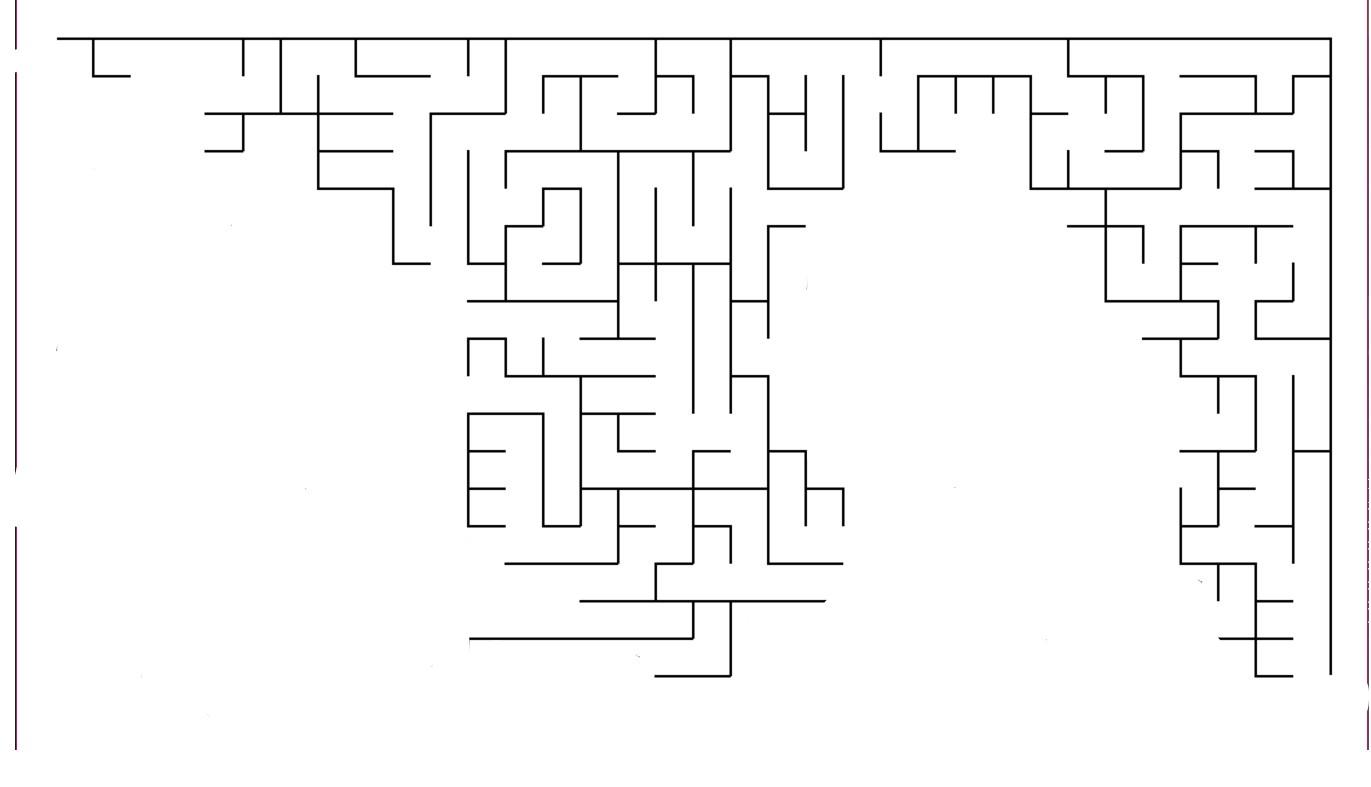
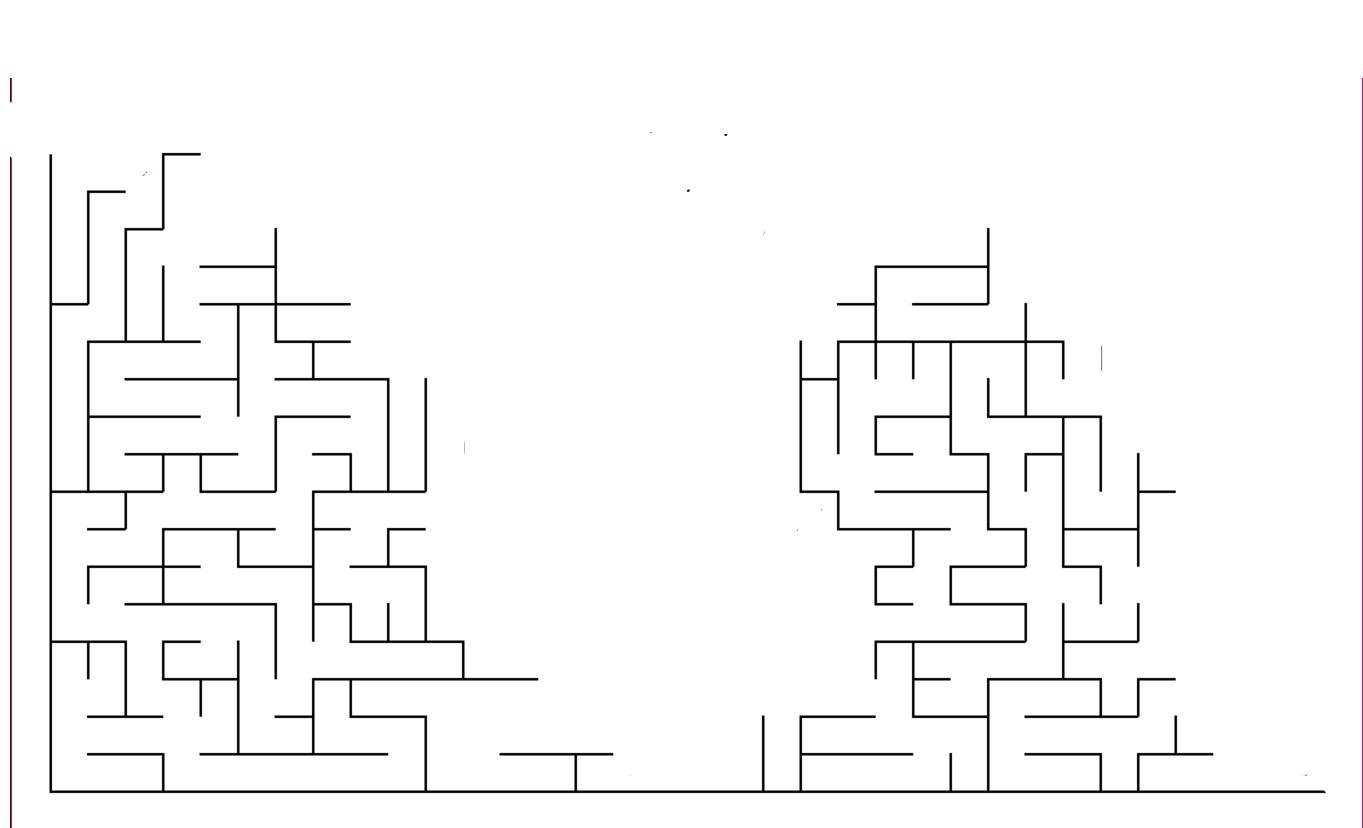
The above are a few examples of snake lines generated for the maze

**The solution of a maze**

A perfect maze always has only one solution. There will always be only one path between any two points on the maze. This is how the algorithm always generates a perfect maze.



This maze can be split into two halves as shown below.



Every line in these halves is interconnected. When joined, the halves form a maze. The only solution to the maze is right between the two halves.

1. **SYSTEM REQUIREMENTS**
2. PYTHON - System should be able to run python programs
3. PYGAME - An additional Pygame module should be installed
4. OPERATING SYSTEM – Windows 8/ 9/ 10
5. **MODULES & FUNCTIONS**

* Pygame is used to create the basic gaming interface. It is used both in maze generation and user interface
* Time is the simplest module used to measure how long an event takes, like a stopwatch. (user interface)
* Random is used to randomize the maze each time. (maze generation)

1. **INSTRUCTION MANUAL**
2. This is a project to randomly generate a fun maze. The aim of the game is to complete the maze in the shortest time possible.
3. When you run the game, a small screen will pop up, with the words, ‘Maze Generator’. To begin, press enter on the keyboard, or click on the green button which says ‘Play’.
4. Every time you press a button using the mouse or press a key on the keyboard, you will hear a sound effect. This means the program has accepted your input.
5. You will see an orange screen with all the controls for the game. You can use ‘W A S D’ or arrow keys to move. Press backspace to undo your last move, and M to toggle mute. Press enter on the keyboard if you’re ready to play.
6. You will see a screen with 3 orange buttons. Select a level by clicking on one of the buttons, or press the level number on the keyboard.
7. You will see a screen with 10 colours you can draw with. Click on any of the squares to select that colour.
8. You will see a maze in a few seconds. As soon it appears, the timer will start. The 2 green arrows represent the start point and end point. You can start moving using the keyboard.
9. Remember, you are not allowed to go back the same way you came. If you want to undo the last move, press backspace. If you accidentally go back, or hit a line, you lose.
10. You will see 2 small buttons at the top of the screen. If you want to restart the same maze, click on ‘Try again’ and start moving from the start point. Otherwise, if you’re done playing, click on ‘Done’.
11. If you click on ‘Done’, or if you successfully complete the maze, you will see a new screen, with 2 green buttons.
12. If you want to try a new maze of the same level, click on ‘New maze of same level’. If you want to change your level, click on ‘Select level and colour’.
13. Replay as many times as you want, and enjoy! But if you want to exit anytime, just click on the red cross on the top right. You may also press escape on your keyboard.
14. **Game creator’s special tip**: Instead of mentally following a path to see if you and win, at any turn, select a random direction. If that direction turns out to be wrong, you can always press backspace and try again.
15. **PROJECT CODE**

# Pygame and other modules

import pygame, time, random

from pygame import \*

pygame.init()

# Colours in program

white = (255, 255, 255)

green = (0, 155, 0)

orange = (255,92,0)

usr = (0,25,200)

vio = (0,25,200)

red = (255,0,0)

black=(0,0,0)

# Colours to pick

color1 = (205,0,0)

color2 = (255,55,0)

color3 = (255,255,0)

color4 = (0,155,0)

color5 = (55,55,255)

color6 = (125,0,225)

color7 = (205,0,205)

color8 = (125,75,25)

color9 = (75,205,255)

color10 = (25,225,25)

# Variables

openlocs=[]

turndict={'up':'down','down':'up','left':'right','right':'left'}

danger=[]

clx = 0; cly = 0

r=2

mute = False

t1 = True

# to see if user quit

uquit = False

# Customisation

pygame.display.set\_caption('Maze Generator')

image = pygame.image.load('pic.png')

pygame.display.set\_icon(image)

clock = pygame.time.Clock()

# font(file,size) and screen

font1 = pygame.font.Font('Font1.ttf', 75)

font2 = pygame.font.Font('Font2.ttf', 67)

font3 = pygame.font.Font('Font3.ttf', 60)

screenx=1080; screeny=600

window = pygame.display.set\_mode((screenx, screeny))

window.fill(white)

# Music

mixer.init()

sWin = pygame.mixer.Sound('win.wav')

sMove = pygame.mixer.Sound('move.mp3')

sNext = pygame.mixer.Sound('next.wav')

sLoss = pygame.mixer.Sound('stop.wav')

sounds = [sWin,sMove,sNext,sLoss]

step=40

# Background

ship = pygame.image.load("bg.jpg")

ship = pygame.transform.scale(ship,(screenx,screeny))

rect = ship.get\_rect()

rect = rect.move((0, 0))

window.blit(ship, rect)

# Play button

play = font1.render('PLAY',True,green,black)

Rect2 = Rect((screenx // 2)+100,(screeny // 2)+50,175,85)

pygame.draw.rect(window,black,Rect2)

window.blit(play, Rect2)

# Game Loop 1

running = True

while running:

for event in pygame.event.get():

# Check for red screenx

if event.type == pygame.QUIT:

uquit = True

running = False

break

if event.type== pygame.KEYDOWN:

# escape is as good as the red screenx

if event.key == pygame.K\_ESCAPE:

uquit = True

running = False

break

elif event.key == pygame.K\_m:

for so in sounds:

if not mute: so.set\_volume(0.0)

else: so.set\_volume(1.0)

mute = not(mute)

elif event.key == pygame.K\_RETURN:

sNext.play()

window.fill(white)

hel = pygame.image.load("help.png")

hel = pygame.transform.scale(hel,(screenx,screeny))

rect2 = hel.get\_rect()

rect2 = rect2.move((0, 0))

window.blit(hel, rect2)

pygame.display.update()

running = False

break

#Mouse interaction

if event.type == pygame.MOUSEBUTTONDOWN:

mousex, mousey = event.pos

if Rect2.collidepoint(mousex,mousey):

sNext.play()

window.fill(white)

hel = pygame.image.load("help.png")

hel = pygame.transform.scale(hel,(screenx,screeny))

rect2 = hel.get\_rect()

rect2 = rect2.move((0, 0))

window.blit(hel, rect2)

pygame.display.update()

running=False

break

# update the screen

pygame.display.update()

# Game Loop 2

run4 = True

while run4 and not uquit:

for event in pygame.event.get():

# Check for red screenx and escape

if event.type == pygame.QUIT:

uquit = True

running = False

break

elif event.type== pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

uquit = True

running = False

break

# Mute

elif event.key == pygame.K\_m:

for so in sounds:

if not mute: so.set\_volume(0.0)

else: so.set\_volume(1.0)

mute = not(mute)

elif event.key == pygame.K\_RETURN:

sNext.play()

run4 = False

break

# update the screen

pygame.display.update()

# Level buttons

but1 = font3.render(' Level1 ', True, black, orange)

but1Rect = but1.get\_rect()

but1Rect.center = (screenx // 2, (screeny // 2)-150)

but2 = font3.render(' Level2 ', True, black, orange)

but2Rect = but2.get\_rect()

but2Rect.center = (screenx // 2, (screeny // 2))

but3 = font3.render(' Level3 ', True, black, orange)

but3Rect = but3.get\_rect()

but3Rect.center = (screenx // 2, (screeny // 2)+150)

# Loop to start again from level selection

Play = True

while(Play and not uquit):

# Re - Initialise Variables

openlocs=[]

turndict={'up':'down','down':'up','left':'right','right':'left'}

danger=[]

r=2

winner=False

# Draw the level buttons

window.fill(white)

window.blit(but1, but1Rect)

window.blit(but2, but2Rect)

window.blit(but3, but3Rect)

PlayL=True

# Game loop 3

running2 = True

while running2 and not uquit:

for event in pygame.event.get():

# Check for red screenx and escape

if event.type == pygame.QUIT:

uquit = True

running2 = False

break

# Mouse interaction

elif event.type == pygame.MOUSEBUTTONDOWN:

mousex, mousey = event.pos

if not Rect2.collidepoint(mousex,mousey):

if but1Rect.collidepoint(mousex,mousey): step=60; t1 = False

elif but2Rect.collidepoint(mousex,mousey): step=40; t1 = False

elif but3Rect.collidepoint(mousex,mousey): step=30; t1 = False

elif event.type== pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

uquit = True

running2 = False

break

# Mute

elif event.key == pygame.K\_m:

for so in sounds:

if not mute: so.set\_volume(0.0)

else: so.set\_volume(1.0)

mute = not(mute)

# Select level with keyboard

elif event.key == pygame.K\_1: step=60; t1 = False

elif event.key == pygame.K\_2: step=40; t1 = False

elif event.key == pygame.K\_3: step=30; t1 = False

# If something is selected, continue to next screen

if(not t1):

sNext.play()

running2 = False

break

# update the screen

pygame.display.update()

# Instructions

chip = pygame.image.load("color.png")

chip = pygame.transform.scale(chip,(screenx,screeny))

rect = chip.get\_rect()

rect = rect.move((0, 0))

window.blit(chip, rect)

# The colour options

cRect1 = pygame.Rect(120,0,120,120)

cRect2 = pygame.Rect(360,0,120,120)

cRect3 = pygame.Rect(600,0,120,120)

cRect4 = pygame.Rect(840,0,120,120)

cRect5 = pygame.Rect(120,240,120,120)

cRect6 = pygame.Rect(360,240,120,120)

cRect7 = pygame.Rect(600,240,120,120)

cRect8 = pygame.Rect(840,240,120,120)

cRect9 = pygame.Rect(120,480,120,120)

cRect10 = pygame.Rect(360,480,120,120)

# Printing the colour options

pygame.draw.rect(window,color1,cRect1)

pygame.draw.rect(window,color2,cRect2)

pygame.draw.rect(window,color3,cRect3)

pygame.draw.rect(window,color4,cRect4)

pygame.draw.rect(window,color5,cRect5)

pygame.draw.rect(window,color6,cRect6)

pygame.draw.rect(window,color7,cRect7)

pygame.draw.rect(window,color8,cRect8)

pygame.draw.rect(window,color9,cRect9)

pygame.draw.rect(window,color10,cRect10)

pygame.display.update()

# Game loop 4

t2 = True

runcol = True

while runcol and not uquit:

for event in pygame.event.get():

# Check for red screenx and escape

if event.type == pygame.QUIT:

uquit = True

running = False

break

elif event.type== pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

uquit = True

running = False

break

# Mute

elif event.key == pygame.K\_m:

for so in sounds:

if not mute: so.set\_volume(0.0)

else: so.set\_volume(1.0)

mute = not(mute)

# Mouse - clicked on a button

elif event.type == pygame.MOUSEBUTTONDOWN:

mousex, mousey = event.pos

if cRect1.collidepoint(mousex,mousey): usr = color1; t2 = False

elif cRect2.collidepoint(mousex,mousey): usr = color2; t2 = False

elif cRect3.collidepoint(mousex,mousey): usr = color3; t2 = False

elif cRect4.collidepoint(mousex,mousey): usr = color4; t2 = False

elif cRect5.collidepoint(mousex,mousey): usr = color5; t2 = False

elif cRect6.collidepoint(mousex,mousey): usr = color6; t2 = False

elif cRect7.collidepoint(mousex,mousey): usr = color7; t2 = False

elif cRect8.collidepoint(mousex,mousey): usr = color8; t2 = False

elif cRect9.collidepoint(mousex,mousey): usr = color9; t2 = False

elif cRect10.collidepoint(mousex,mousey): usr = color10; t2 = False

# If something is selected, continue to next screen

if(not t2):

sNext.play()

runcol = False

break

# update the screen

pygame.display.update()

# Loop for new game in the same level

while(PlayL and not uquit):

window.fill(white)

# Coordinates

corners=(screenx,screeny,0)

coordinates = []

# Edit here

for y in range(step,screeny,step):

l=[]

for x in range(step,screenx,step):

m = []

m.append((x,y))

m.append(0)

l.append(m)

coordinates.append(l)

# So that the edges of the box are closed

for i in coordinates[0]: i[1]=1

for i in coordinates[len(coordinates)-1]: i[1]=1

for i in coordinates: i[len(i)-1][1]=1

for i in coordinates: i[0][1]=1

for i in coordinates:

for j in i:

if j[1]==0:

openlocs.append(j)

# Drawing the outside box

cornerpts = [(step,step),(screenx-step,step),(screenx-step,screeny-step),(step,screeny-step)]

for i in range(4): pygame.draw.polygon(window, (0,0,0),cornerpts, width=2)

# Randomizing the start and end points

alist=[1,2]

ran=random.choice(alist)

# Start opening

if ran==1: pygame.draw.line(window,(255,255,255),(step,step),(step,2\*step),2)

else: pygame.draw.line(window,(255,255,255),(step,step),(2\*step,step),2)

# End opening

if ran==1: pygame.draw.line(window,(255,255,255),(screenx-step,screeny-step),(screenx-step,screeny-(2\*step)),2)

else: pygame.draw.line(window,(255,255,255),(screenx-step,screeny-step),(screenx-(2\*step),screeny-step),2)

# Updating open coordinates

while len(openlocs):

turns=['up','down','left','right']

moves=[]

# Randomly chose an empty coordinate

point=random.choice(openlocs)

x=int(point[0][0])

y=int(point[0][1])

# Remove selected point from list of open coordinates

openlocs.remove(point)

# Randomly pick a main direction for squiggly maze line

maindir=random.choice(turns)

# Setting maximum length in the chosen direction

# This helps increase efficiency

if maindir =='up' or maindir=='down': mainlimit =(len(coordinates)-1)

else: mainlimit=len(coordinates[0])-1

# Finds the exact location of random point

for ii in range(0,len(coordinates)):

for jj in range(0,len(coordinates[1])):

if coordinates[ii][jj]==point:

exactloc=[ii,jj]

# Changing the list of the point

danger.append(point)

if point in openlocs: openlocs.remove(point)

# Avoid going in the opposite direction

ultadir=turndict[maindir]

possibleturns=turns

possibleturns.remove(ultadir)

# Avoid going too straight

most=3

# So that the line doesn't become longer than the maze

if mainlimit<3: most=mainlimit

a=random.choice(possibleturns)

for i in range(random.randint(1,most)): moves.append(a)

# Don't go back the same way

possibleturns.remove(maindir)

# moves is a list of moves used to create the line

for i in moves:

if i==maindir:

mainlimit-=1

# Assorted bug fixes

while mainlimit!=0:

if most>mainlimit: most=mainlimit

if moves[len(moves)-1]==maindir:

a=random.choice(possibleturns)

t1=0

t2=0

# Avoid patterns like stairs, which are a waste of space

for i in moves:

if i==possibleturns[0]: t1+=1

if i==possibleturns[1]: t2+=1

if t1-t2>2: a=possibleturns[1]

if t2-t1>2: a=possibleturns[0]

# Updating the list of moves

for i in range(random.randint(1,most)): moves.append(a)

else:

# Random bug fix

if mainlimit<5: most=mainlimit

for i in range(random.randint(1,most)):

moves.append(maindir)

mainlimit-=1

# Update the danger list

# This will be useful later

for i in range(0,len(danger)):

for p in coordinates:

for q in p:

if danger[i]==q:

# The coordinate is no longer open

q[1]=1

danger[i]=q

checkvar=1

# Time to draw

for spin in range(0,len(moves)):

# Moving up

if moves[spin]=='up':

if (checkvar and coordinates[exactloc[0]][exactloc[1]] not in danger) or not spin :

pygame.draw.line(window,black,(x,y),(x,y-step),2)

exactloc[0]-=1

if coordinates[exactloc[0]][exactloc[1]] in openlocs :

coordinates[exactloc[0]][exactloc[1]][1]='closed'

openlocs.remove(coordinates[exactloc[0]][exactloc[1]])

if y-step>0: y-=step

else: checkvar=0

# Draws lines according to the randomly generated moves

# Draws until it hits another line

# Same for all directions

if moves[spin]=='down':

if (checkvar and coordinates[exactloc[0]][exactloc[1]] not in danger) or not spin:

pygame.draw.line(window,black,(x,y),(x,y+step),2)

exactloc[0]+=1

if coordinates[exactloc[0]][exactloc[1]] in openlocs :

coordinates[exactloc[0]][exactloc[1]][1]=1

openlocs.remove(coordinates[exactloc[0]][exactloc[1]])

if y+step<screeny+1: y+=step

else: checkvar=0

if moves[spin]=='left':

if checkvar and coordinates[exactloc[0]][exactloc[1]] not in danger or not spin:

pygame.draw.line(window,black,(x,y),(x-step,y),2)

exactloc[1]-=1

if coordinates[exactloc[0]][exactloc[1]] in openlocs :

coordinates[exactloc[0]][exactloc[1]][1]=1

openlocs.remove(coordinates[exactloc[0]][exactloc[1]])

if x-step>0: x-=step

else: checkvar=0

if moves[spin]=='right':

if checkvar and coordinates[exactloc[0]][exactloc[1]] not in danger or not spin:

pygame.draw.line(window,black,(x,y),(x+step,y),2)

exactloc[1]+=1

if coordinates[exactloc[0]][exactloc[1]] in openlocs :

coordinates[exactloc[0]][exactloc[1]][1]=1

openlocs.remove(coordinates[exactloc[0]][exactloc[1]])

if x+step<screenx+1: x+=step

else: checkvar=0

# Removing the newly closed coordinate

removables=[]

for i in range(0,len(openlocs)):

if openlocs[i][1]!=0:

removables.append(openlocs[i])

for i in removables:

openlocs.remove(i)

#therefore, when a new point is chosen, it will be empty

pygame.display.update()

# Loop for retrying same maze

playm=True

while(playm and not uquit):

# Re - Initialise Variables

openlocs=[]

turndict={'up':'down','down':'up','left':'right','right':'left'}

danger=[]

r=2

running3 = True

# Game loop 5

lost=False

i=1

t2=0

moo=''

draws=[]

# set starting position, ending position, and draw arrows

# Also fix the first move

if(ran==1):

first='r'

x1st,y1st=step//2,step\*3//2

xdraw,ydraw=step//2,step\*3//2

pygame.draw.line(window,green,(xdraw-10,ydraw-10),(xdraw,ydraw),4)

pygame.draw.line(window,green,(xdraw-10,ydraw+10),(xdraw,ydraw),4)

pygame.draw.line(window,green,(screenx-20,screeny-step\*3//2+10),(screenx-10,screeny-step\*3//2),4)

pygame.draw.line(window,green,(screenx-20,screeny-step\*3//2-10),(screenx-10,screeny-step\*3//2),4)

xend,yend=1080-step//2,600-step\*3//2

else:

first='d'

x1st,y1st=step\*3//2,step//2

xdraw,ydraw=step\*3//2,step//2

xend,yend=1080-step\*3//2,600-step//2

pygame.draw.line(window,green,(xdraw-10,ydraw-10),(xdraw,ydraw),4)

pygame.draw.line(window,green,(xdraw+10,ydraw-10),(xdraw,ydraw),4)

pygame.draw.line(window,green,(screenx-step\*3//2-10,screeny-20),(screenx-step\*3//2,screeny-10),4)

pygame.draw.line(window,green,(screenx-step\*3//2+10,screeny-20),(screenx-step\*3//2,screeny-10),4)

# Start timer

start\_time = pygame.time.get\_ticks()

while running3 and not uquit:

for event in pygame.event.get():

# Check for red screenx and escape

if event.type == pygame.QUIT:

uquit = True

running3 = False

break

elif event.type== pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

uquit = True

running3 = False

break

# Mute

elif event.key == pygame.K\_m:

for so in sounds:

if not mute: so.set\_volume(0.0)

else: so.set\_volume(1.0)

mute = not(mute)

t2+=2

# Undo last move

if event.key == pygame.K\_BACKSPACE and i>1:

if draws[i-2]=='r':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw-step,ydraw),4)

xdraw-=step

elif draws[i-2]=='l':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw+step,ydraw),4)

xdraw+=step

elif draws[i-2]=='d':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw,ydraw-step),4)

ydraw-=step

elif draws[i-2]=='u':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw,ydraw+step),4)

ydraw+=step

draws.pop()

i-=1

if(xdraw==x1st and ydraw==y1st): t2 = 0

sMove.play()

moo=''

# Move Right

if event.key == pygame.K\_RIGHT or event.key == pygame.K\_d:

moo='r'

i+=1

clr=window.get\_at((xdraw+step//2,ydraw))

pygame.draw.line(window,usr,(xdraw,ydraw),(xdraw+step,ydraw),4)

xdraw+=step

sMove.play()

if(clr==black or clr==usr or(t2==2 and moo!=first)): lost = True

# Move Left

elif event.key == pygame.K\_LEFT or event.key == pygame.K\_a:

moo='l'

i+=1

clr=window.get\_at((xdraw-step//2,ydraw))

pygame.draw.line(window,usr,(xdraw,ydraw),(xdraw-step,ydraw),4)

xdraw-=step

sMove.play()

if(clr==black or clr==usr or(t2==2 and moo!=first)): lost = True

# Move Down

elif event.key == pygame.K\_DOWN or event.key == pygame.K\_s:

moo='d'

i+=1

clr=window.get\_at((xdraw,ydraw+step//2))

pygame.draw.line(window,usr,(xdraw,ydraw),(xdraw,ydraw+step),4)

ydraw+=step

sMove.play()

if(clr==black or clr==usr or(t2==2 and moo!=first)): lost = True

# Move Up

elif event.key == pygame.K\_UP or event.key == pygame.K\_w:

moo='u'

i+=1

clr=window.get\_at((xdraw,ydraw-step//2))

pygame.draw.line(window,usr,(xdraw,ydraw),(xdraw,ydraw-step),4)

ydraw-=step

sMove.play()

if(clr==black or clr==usr or(t2==2 and moo!=first)): lost = True

if(moo):draws.append(moo)

# Check if user lost

if lost:

if(step==30):font4 = pygame.font.Font('Font2.ttf', 20)

else: font4 = pygame.font.Font('Font2.ttf', 25)

sLoss.play()

lbut = font4.render(' You Lost! ', True, green, white)

TRec = lbut.get\_rect()

TRec.center = (screenx//2 - 150, 15)

window.blit(lbut, TRec)

lbut = font4.render('Try again', True, vio, orange)

TRec = lbut.get\_rect()

TRec.center = (screenx//2 + 50, 15)

window.blit(lbut, TRec)

lbut = font4.render('Done', True, vio, orange)

DRect = lbut.get\_rect()

DRect.center = (screenx//2 + 200, 15)

window.blit(lbut, DRect)

pygame.display.update()

running3=False

break

# Winning Screen

if xdraw==xend and ydraw==yend:

winner=True

sWin.play()

end\_time=pygame.time.get\_ticks()

t = (end\_time-start\_time) //1000

window.fill((255,255,255))

wpic = pygame.image.load("cup.jfif")

wpic = pygame.transform.scale(wpic,(screenx//2,screeny//2))

wrect = wpic.get\_rect()

wrect = rect.move((0, screeny//2))

window.blit(wpic, wrect)

wbut = font1.render(' You Win ', True, black, orange)

Rec = wbut.get\_rect()

Rec.center = (screenx\*3//4, (screeny//3))

window.blit(wbut, Rec)

wbut = font3.render(' you took '+str(t)+' seconds! ', True, red, white)

Rec = wbut.get\_rect()

Rec.center = (screenx//3, (screeny//3))

window.blit(wbut, Rec)

wbut = font3.render(' select level and colour ', True, black, green)

Rec = wbut.get\_rect()

Rec.center = (screenx\*2//3, (screeny\*2//3))

window.blit(wbut, Rec)

Lbut = font3.render(' new maze of same level ', True, black, green)

RecL = Lbut.get\_rect()

RecL.center = ((screenx\*2//3), (screeny\*2//3)+150)

window.blit(Lbut, RecL)

pygame.display.update()

running3=False

break

pygame.display.update()

clock.tick(60)

# Game Loop 6

if(lost): run4 = True

else: run4 = False

l=False

while run4 and not uquit:

for event in pygame.event.get():

# Check for red screenx and escape

if event.type == pygame.QUIT:

uquit = True

break

elif event.type== pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

uquit = True

break

# Mute

elif event.key == pygame.K\_m:

for so in sounds:

if not mute: so.set\_volume(0.0)

else: so.set\_volume(1.0)

mute = not(mute)

# If user wants to end game

elif event.type == pygame.MOUSEBUTTONDOWN:

mousex, mousey = event.pos

if DRect.collidepoint(mousex,mousey):

window.fill((255,255,255))

lbut = font2.render('Don\'t Give Up!', True, orange, white)

lRect = lbut.get\_rect()

lRect.center = (screenx//2 - 10, (screeny//3))

window.blit(lbut, lRect)

font2 = pygame.font.Font('Font2.ttf', 56)

lbut = font2.render(' select level and colour ', True, black, green)

Rec = lbut.get\_rect()

Rec.center = (screenx//2, (screeny\*2//3)-50)

window.blit(lbut, Rec)

Lbut = font2.render(' new maze of same level ', True, black, green)

RecL = Lbut.get\_rect()

RecL.center = ((screenx//2), (screeny\*2//3)+100)

window.blit(Lbut, RecL)

sNext.play()

playm=False

run4=False

break

# If user wants to try again

elif TRec.collidepoint(mousex,mousey):

sNext.play()

if(step==30):pygame.draw.rect(window,white,pygame.Rect(screenx//2 - 300, 0,700,27))

else: pygame.draw.rect(window,white,pygame.Rect(screenx//2 - 300, 0,700,30))

draws.reverse()

L=1

l=True

for d in draws:

if d=='r':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw-step,ydraw),4)

if(L==1 and len(draws)!=1):pygame.draw.rect(window,black,pygame.Rect(xdraw-(step//2),ydraw-1,2,4))

xdraw-=step

elif d=='l':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw+step,ydraw),4)

if(L==1 and len(draws)!=1):pygame.draw.rect(window,black,pygame.Rect(xdraw+(step//2),ydraw-1,2,4))

xdraw+=step

elif d=='d':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw,ydraw-step),4)

if(L==1 and len(draws)!=1):pygame.draw.rect(window,black,pygame.Rect(xdraw-1,ydraw-(step//2),4,2))

ydraw-=step

elif d=='u':

pygame.draw.line(window,white,(xdraw,ydraw),(xdraw,ydraw+step),4)

if(L==1 and len(draws)!=1):pygame.draw.rect(window,black,pygame.Rect(xdraw-1,ydraw+(step//2),4,2))

ydraw+=step

L+=1

run4=False

break

# update the screen

pygame.display.update()

# Game Loop 7

if(not l or winner):run5 = True

else: run5 = False

while run5 and not uquit:

for event in pygame.event.get():

# Check for red screenx and escape

if event.type == pygame.QUIT:

uquit = True

break

elif event.type== pygame.KEYDOWN:

# escape is as good as the red screenx

if event.key == pygame.K\_ESCAPE:

uquit = True

break

# Mute

elif event.key == pygame.K\_m:

for so in sounds:

if not mute: so.set\_volume(0.0)

else: so.set\_volume(1.0)

mute = not(mute)

# Mouse interaction

elif event.type == pygame.MOUSEBUTTONDOWN:

mousex, mousey = event.pos

if Rec.collidepoint(mousex,mousey):

sNext.play()

PlayL=False

run5=False

playm=False

uquit = False

break

elif RecL.collidepoint(mousex,mousey):

sNext.play()

run5=False

uquit = False

playm=False

break

# update the screen

pygame.display.update()

# when it breaks out of the loop, the whole thing closes

pygame.quit()

1. **PROJECT OUTPUT**

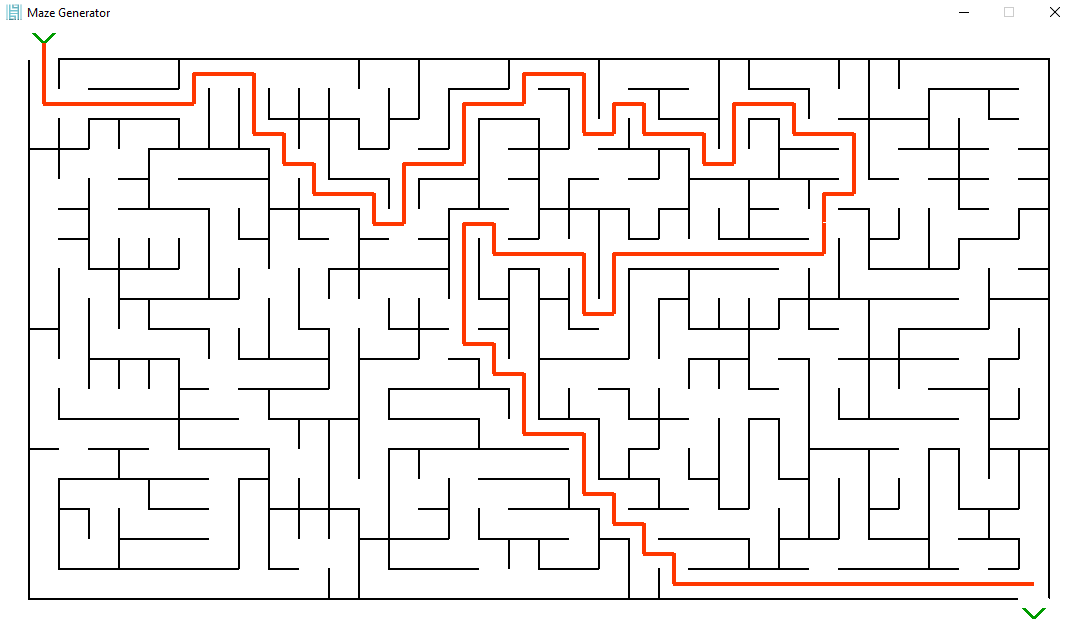


**Screen 1**

****

**Game Icon/ Logo**

|  | **Screen 2**  **Instructions** |
| --- | --- |
|  | **Screen 3**  **Level selection** |
|  | **Screen 4**  **Color selection** |
|  | **Level 1 Maze**  **example** |
|  | **Attempting a**  **level 1 maze** |
|  | **Losing after**  **hitting a line** |
|  | **Screen displayed after choosing ‘Try Again’** |
|  | **After choosing**  **‘Done’** |
|  | **Moving in a Level 2 Maze after selecting blue colour** |
|  | **Winning Screen** |
|  | **Moving in a Level 3 Maze after choosing Green colour** |
|  | **After using backspace to undo moves** |



**One Last Example!**

1. **LIMITATIONS**

Right now, there’s a lot of scope for improvement in our project. For example, complicated mazes take a lot of time to load. There are many additional features which may be added in later versions of the game. Here are a few:

* Collecting coins/ points on the way
* Other shapes of Mazes like triangle, circle
* Multiplayer game
* A character that moves around, while drawing
* Converting this game into an executable file
* Other puzzles/ mini-games
* 3D mazes

1. **BIBLIOGRAPHY**
2. These are the websites that helped us complete this project:

* [https://www.youtube.com](https://www.youtube.com/)
* [https://www.geeksforgeeks.org](https://www.geeksforgeeks.org/)
* [https://www.w3schools.com](https://www.w3schools.com/)
* [https://stackoverflow.com](https://stackoverflow.com/)
* <https://www.pygame.org/docs>

Thank you!