

Group Project

(Updated Instructions)

In this group project, we will design the python programme to have a competition, by controlling a robot based on keyboard input to find its way from the START (yellow background) to the GOAL (blue background) through a maze.

Two teams, i.e. Host team and Guest team, will compete together. Host team sets the maze and draws the GUI, while Guest team moves in the maze. **START and GOAL can be located by the Host team anywhere in the map, with constraints that they should be two different positions, and not on boundary.**

You will be trained to collaborate within the team and between teams. Modularize your coding in functions, so that your teammates and your opponents can call.

A sample GUI with demonstration steps is shown as below.

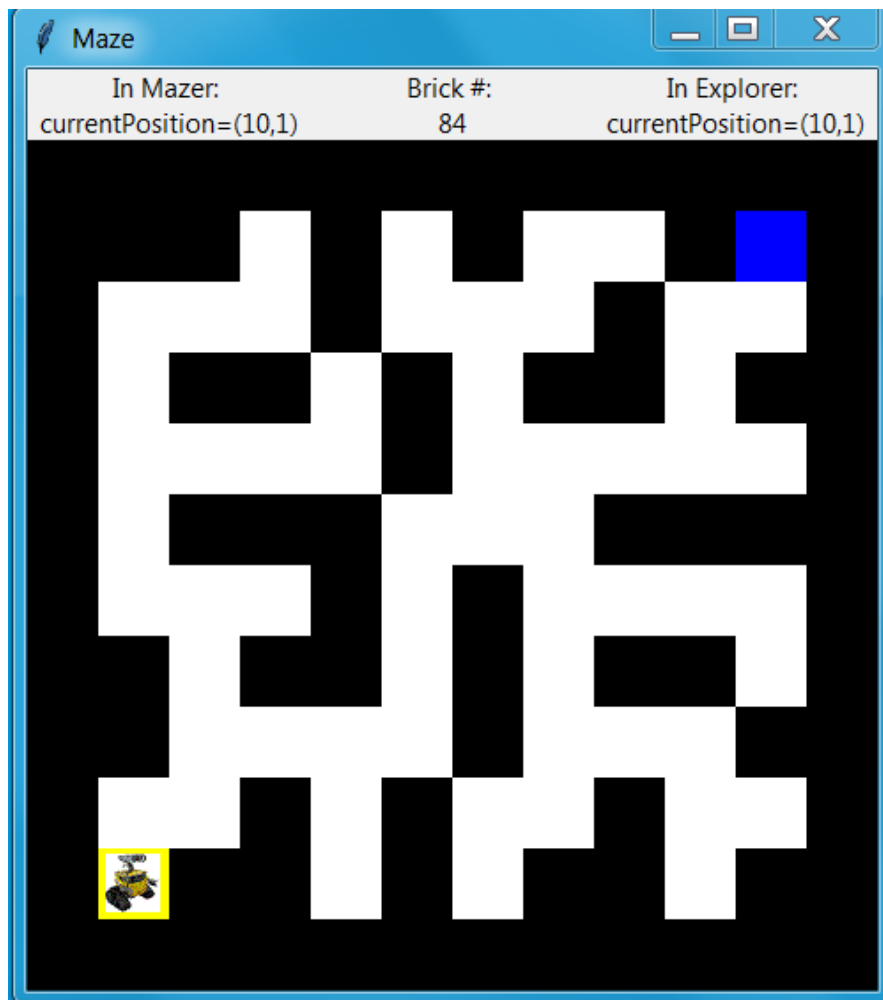


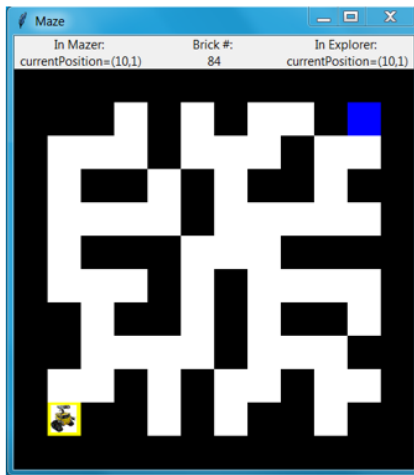
Fig. 1 Sample GUI

The figure below illustrates one sample competition process.

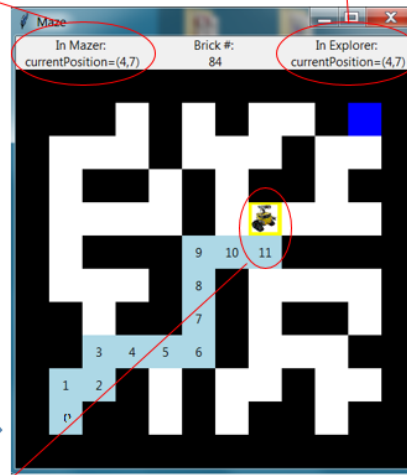
Host: update the current position in Host

Guest: tell Host your current position

Host: update the current position in Guest



- Host team draws the GUI

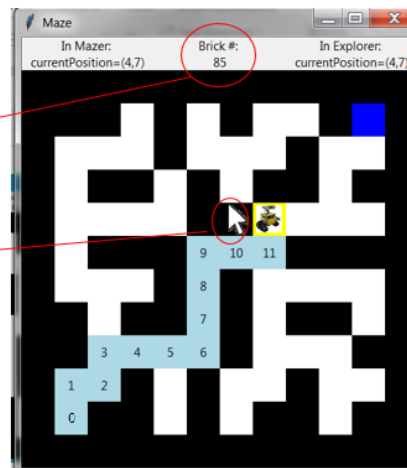


- Guest team controls the robot by 4 keyboard input
- Host team updates the GUI

Host: update the current position, trajectory and show the number of steps.

Host: brick number change

Host: add maximally one brick based on left click



Guest: declare you have reached the GOAL, and show the number of steps

- Host team can add maximally **one** additional brick to the maze by left-click, based on some rules.

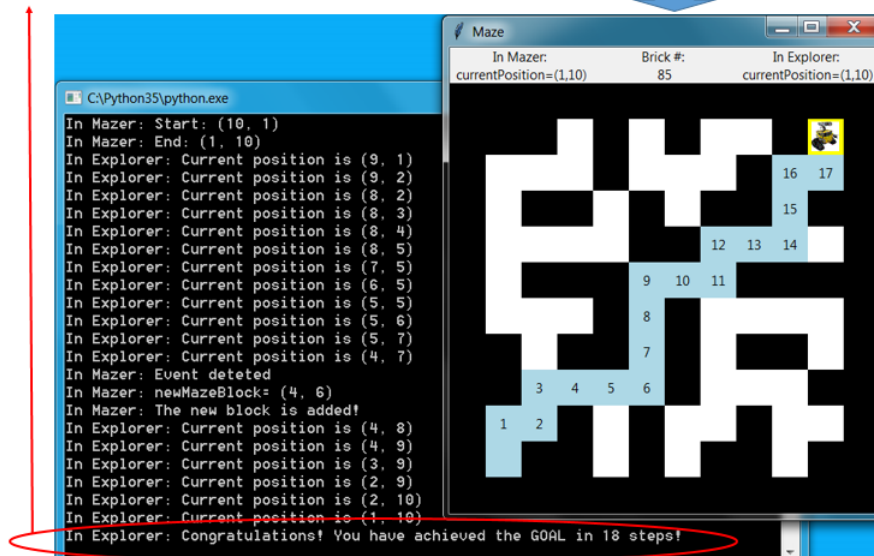


Fig. 2 Sample competition process

Host team and Guest team have roles as below:

- Host team
 - H1. Design maze: The maze dimension is 12x12. Maximal maze number is 85: 44 (fixed number on boundary) + 40 (flexible number for interior part) + 1 (additional)
 - H2. Draw GUI and update it
 - H3. When there a mouse-left-click **on the window**, Host team can add **an** additional brick (black block) to the maze.
- Guest team
 - G1. Give instructions to the robot manually by the four keyboard inputs (up, down, left and right)
 - G2. Check if there is a path from the START to GOAL.
 - G3. Count the number of steps.

Competition Arrangement

■ Individual game match (This section is completely updated for fairer score evaluation)

For each *individual game match* played by, for example, Teams #i and #j, there are two half match sessions.

In each half match session, there will be one score for each team, evaluated based on the following flowchart (In this half match session, i is Host, while j is Guest).

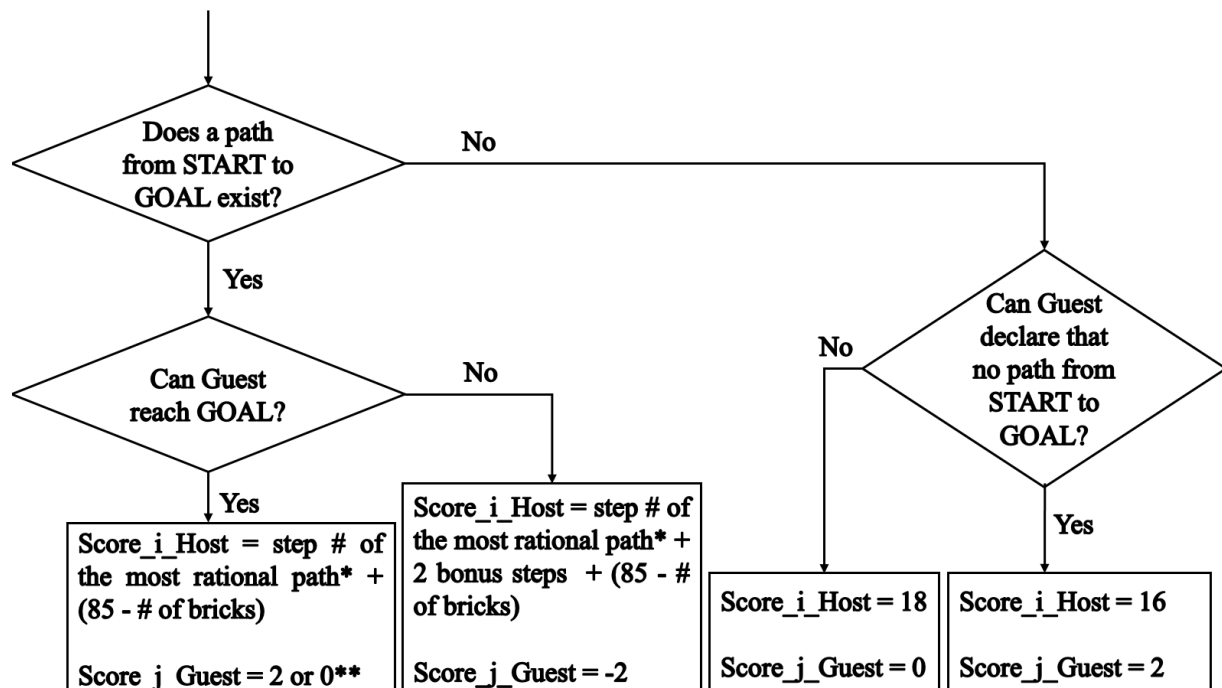


Fig. 3 Score Evaluation Flowchart

* **Step # of the most rational path** should be demonstrated by the Host team (based on Host team's *explore.py*, with keyboard controlled by Host team) and verified by the instructor and TAs, after completion of this half match session.

** If the steps of the Guest's path is identical to the most rational one, Score_j_Guest = 2, otherwise 0.

In the next half match session, j is Host and i is Guest. Consequently, the other 2 scores, i.e. Score_i_Guest and Score_j_Host are obtained.

Finally for this match,

$$\text{Score}_i = \text{Score}_i_{\text{Host}} + \text{Score}_i_{\text{Guest}}$$

and

$$\text{Score}_j = \text{Score}_j_{\text{Host}} + \text{Score}_j_{\text{Guest}}.$$

When the maze GUI appears, Judge starts timing. Guest team has at most 5 seconds to observe, before keyboard input for move.

■ Round Level-1

In general, each team has 3 students. There are all together 103 teams with Team ID #1, #2, ... #103. Every team will compete with 2 other teams, as arranged below. Each color link between 2 teams indicates an *individual game match*.

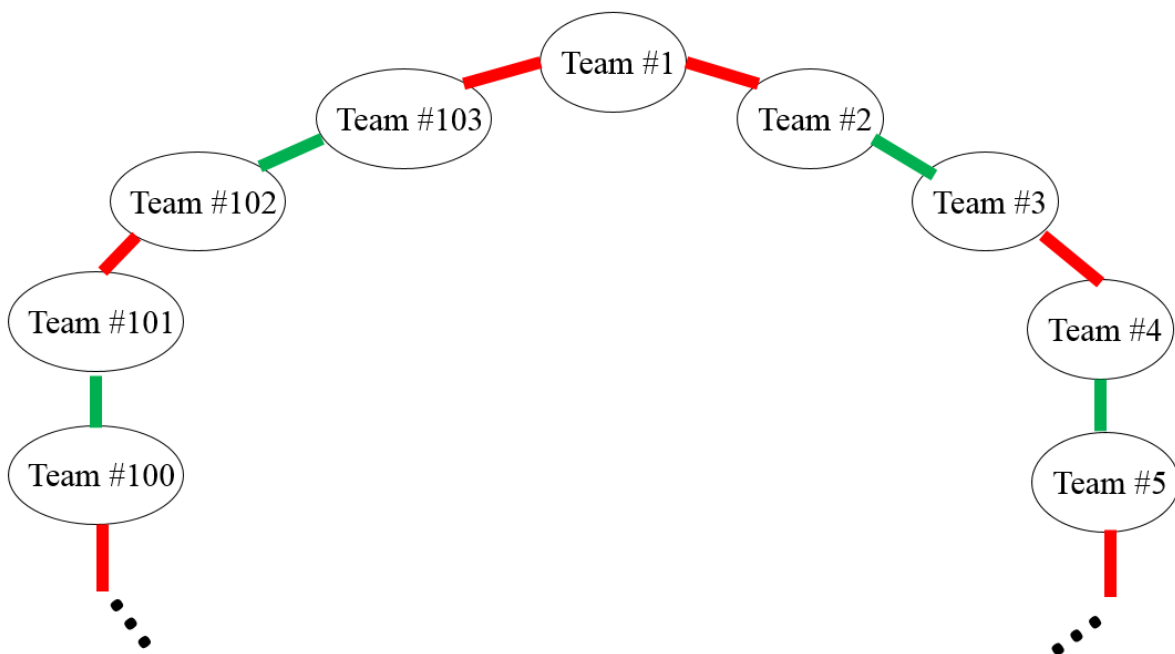


Fig. 4 Round Level-1 Arrangement

- Team # i will play with Team #i-1 and #i+1. (i=2, 3, 4, ...102)
- Team #1 will play with Teams # 2 and #103, while Team #103 will play with Teams #102 and #1.

In general, Team will greater ID number will be the Guest team first, except for the individual game matches between Team #1 and Team #103, where Team #1 will be the Guest Team first.

The sum of the two scores, which Team #i has earned in the two individual game matches, will be its overall score for the Round Level-1.

We will shortlist 16 teams for Round Level-2, based on:

- The teams' scores. Higher scores will be at higher rank.
- If there are equivalent scores preventing from a clear ranking, the one completing the two individual game matches within shorter total time will be ranked higher.
- If there is still any draw preventing from selection of the 16 teams, additional individual matches among the teams with equivalent scores and competition time will be carried out.

■ Round Level-2

The 16 teams will be named from A to P, from the highest to the lowest scores in the Round Level-1. Total match completion time is the second ranking parameter (the shorter, the better). If there are still teams with equivalent performance, they will be selected randomly to fit the team position for Round Level-2.

The competition table is shown as below.

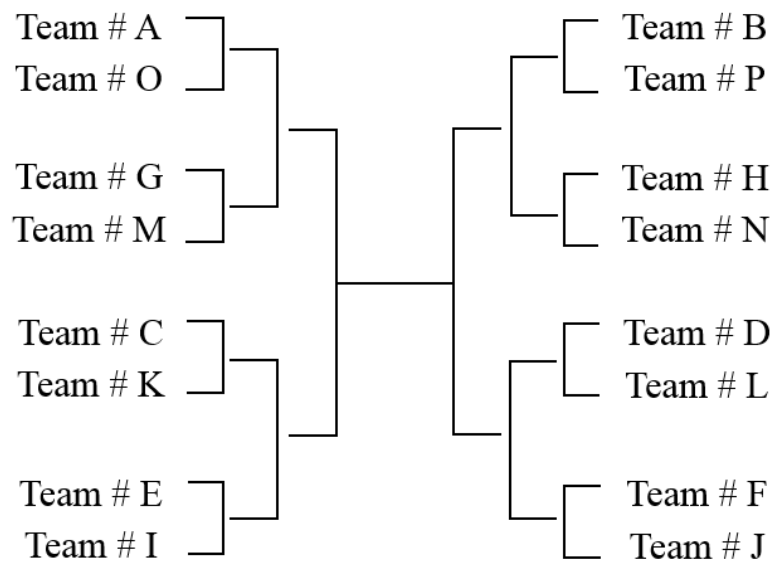


Fig. 5 Round Level-2 Arrangement

For each individual game match in Round Level-2, there should be one winner, based on the rules below.

- The team with higher score wins.
- If the scores are equivalent, the team which reaches GOAL within shorter time wins.
- If there is still a draw, another individual game match will be carried out.

Time Arrangement

Round Level-1

- Time: May 7, 2016 (Saturday)
- Venue: Room 201, Chengdao Building

Each *individual game match* will take around 5 minutes.

The matches shown as the red links will be carried out in the morning from 8:30-12:30.

The matches shown as the green links will be conducted in the afternoon from 14:00-18:00.

Details of the time arrangement will be posted to Moodle later in due course. For each team attending the competition, arrive at least 15 minutes before your time slot.

Note:

- Each team should upload their two .pkl maze files to Moodle by 17:30 in May 6 (Friday). No further change will be allowed.

Round Level-2

- Time: 10:00-12:00, May 8, 2016 (Sunday)
- Venue: Room 201, Chengdao Building

Each *individual game match* will take around 5 minutes.

Details of the time arrangement will be posted to Moodle later in due course. For each team attending the competition, arrive at least 15 minutes before your time slot.

Note:

- Before the beginning of this round, all teams should copy all the maze files into the computer, before 9:45am May 8. They can select from different pkl files, but can not modify them.

Work Distribution

Each team will have 3 students in general, with work distributions as below (The work content is included again for easier reference). Discussion *within the team* is encouraged to build up your team spirit.

- Student A: write coding to generate maze files (.pkl)
 - H1.Design maze: The maze dimension is 12x12. Maximal maze number is 85: 44 (on boundary) + 40 (interior) + 1 (additional)
- Student B: write *mazer.py* coding
 - H2.Draw GUI and update it
 - H3.When there a mouse-left-click **on the window**, Host team can add **an** additional brick (black block) to the maze.
- Student C: write *explorer.py* coding (DO NOT change file name and the pre-provided function names in it.)
 - G1.Give instructions to the robot manually by the four keyboard inputs (up, down, left and right).
 - G2.Check if there is a path from the START to GOAL.
 - G3.Count the number of steps.

Finally, *mazer.py*, *explorer.py*, and the secret maze file, will be included in the same folder (in the Computer Lab witnessed by the class of students). Another image file is included to show the robot picture by the Host team's GUI. (We will learn how to include the image in your GUI later.)

mazer.py, the secret maze file, and the image file are provided by Host team, while *explorer.py* is from Guest team. *mazer.py* will be main programme run.





Name	Date modified	Type	Size
 mazer.py	4/4/2016 10:12 PM	PY File	11 KB
 explorer.py	4/4/2016 9:59 PM	PY File	6 KB
 robot.gif	4/2/2016 11:26 PM	GIF image	2 KB
 mazeFileOriginal.pkl	4/2/2016 7:55 PM	PKL File	1 KB

Fig. 6 Files in One Folder

Information Shared Between Codes

There is significant need to share information among codes written by Host team (student A and student B) and Guest team (student C). There can be various ways. In this group project, we will practice a fundamental approach, i.e. based on shared files. The information sharing between *mazer.py* and *explorer.py* includes:

mazer.py => *explorer.py*: the most updated maze information

explorer.py => *mazer.py*: the most updated current position of the robot

Two .pkl file will be used to exchange information.

■ mazeFile.pkl

This file contains ONE and ONLY ONE *list* object, which is the most updated maze (with values in only 0, 1, 2, and 3). If the new brick is added, this file should be changes.

The initial maze in the demonstrated scenario above is:

```
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 3, 1],
 [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1],
 [1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1],
 [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1],
 [1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
 [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1],
 [1, 2, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

■ explorerPath.pkl

This file contains ONE and ONLY ONE **list** object, with information of the path which the robot has already explored. The elements of the **list** are in **tuple** type, i.e. (10,1) indicating the block in row 10 and column 1, which is the START block in the sample.

The final path list in the demonstrated robot motion is:

[(10, 1), (9, 1), (9, 2), (8, 2), (8, 3), (8, 4), (8, 5), (7, 5), (6, 5), (5, 5), (5, 6), (5, 7), (4, 7), (4, 8), (4, 9), (3, 9), (2, 9), (2, 10), (1, 10)]

Another secret .pkl file is used within Host team members to share the information of maze. To prevent from being changed by explorer.py, it is advised to keep the file name confidential within Host team.

A suggested flowchart for the Host and Guest teams are shown as below¹.

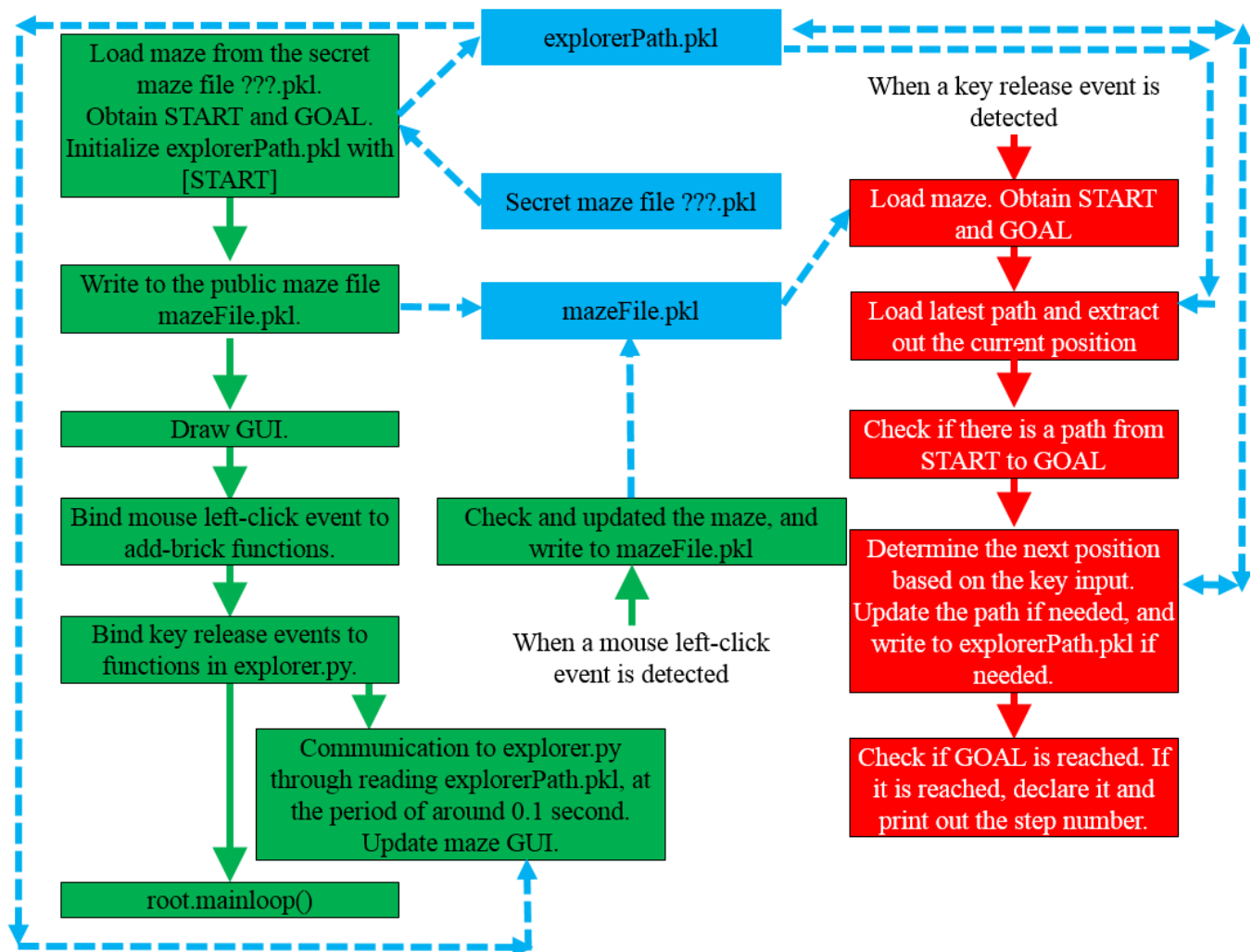


Fig. 7 Suggested Flowchart

- The green blocks are coding sections (**mazer.py**) of Host team T
- The red blocks (**explorer.py**) are for Guest team.
- The blue blocks are three files for information sharing between the two teams.

¹ The flow chart is updated for non-fixed START and GOAL

- The blue dashed lines with arrows indicates file read/write.

Here below, we provide some regulations and/or suggestions for each team member.

Host Team: Student A

Regulations:

- The maze should be at that size of 12x12. The values should be integers 0, 1, 2, and 3.
 - 0 is the vacant space, where robot can step into
 - 1 is the brick, where robot can not move into
 - 2 is the START. You can set it anywhere in the maze, except on the boundary and GOAL.
 - 3 is the GOAL. You can set it anywhere in the maze, except on the boundary and START.

A sample maze object is as below

```
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 3, 1],
 [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1],
 [1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1],
 [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1],
 [1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1],
 [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1],
 [1, 2, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

- The blocks on four boundaries should be all 1s.
- The initial brick number should be fewer or equal to 84, including 44 at the four boundaries, and maximally 40 at the inside. When the additional brick is added, the maximal brick number is 85.
- The brick **CAN NOT** be added to the following positions.
 - START, GOAL
 - The robot's current position
 - The position which is already a brick
- Generally, there should be at least one path from START to GOAL, no matter if it is in the initial map or in the map with one additional brick added. Otherwise, Host team has violated the rule, and will have low score, and Host team provides opportunity for Guest team to earn an additional score.
 - If *explorer.py* discovers and declares that there is no path, Score_i_Host = 16 and Score_j_Guest = 2



Fig. 8. *explorer.py* successfully determines the non-existence of the path from START to GOAL

- If *explorer.py* CAN NOT discover and CAN NOT declare that there is no path, $\text{Score}_i_{\text{Host}} = 18$ and $\text{Score}_j_{\text{Guest}} = 0$
- If the added brick traps the robot and makes it infeasible to reach GOAL, and meanwhile, there is still a path from START to GOAL, $\text{Score}_i_{\text{Host}} = \text{step \# of the most rational path}^2 + 2 \text{ bonus steps} + (85 - \# \text{ of bricks})$, and $\text{Score}_j_{\text{Guest}} = -2$

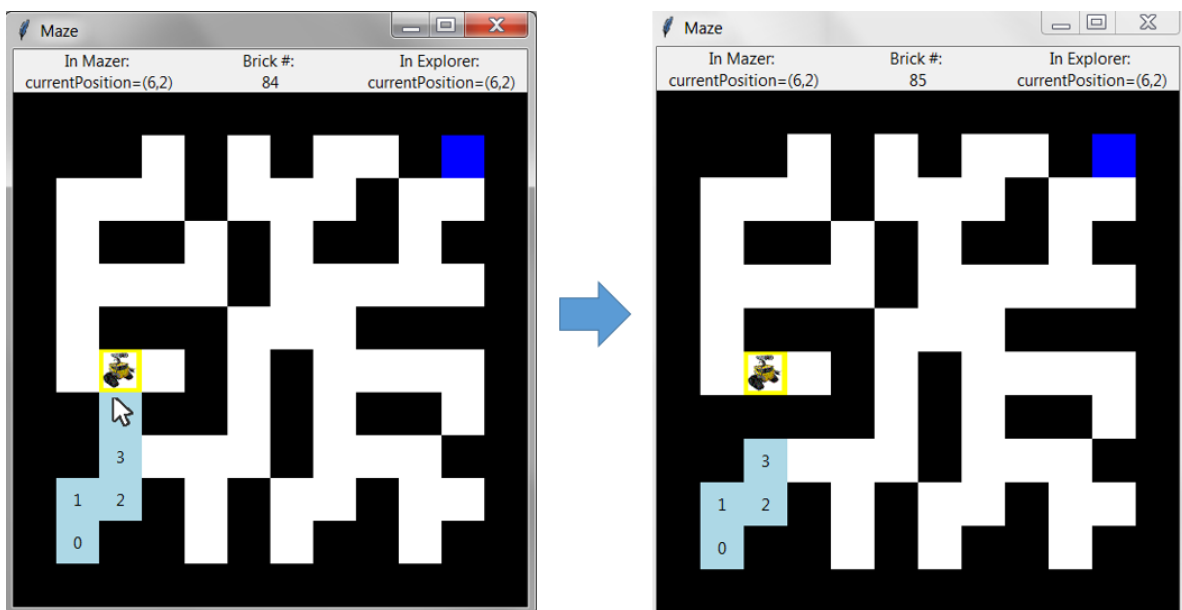


Fig. 9 Trap the robot in a dead-end path

² Step # of the most rational path in this maze should be demonstrated by the Host team and verified by the instructor and TAs, after the half match session.

■ Coordination with other coding

The coordination of Student A's coding with other codings is simple. Give the name of the secret maze file to *mazer.py*. The figure below, with highlighted color blocks, shows the coordination of Student A and B's work.

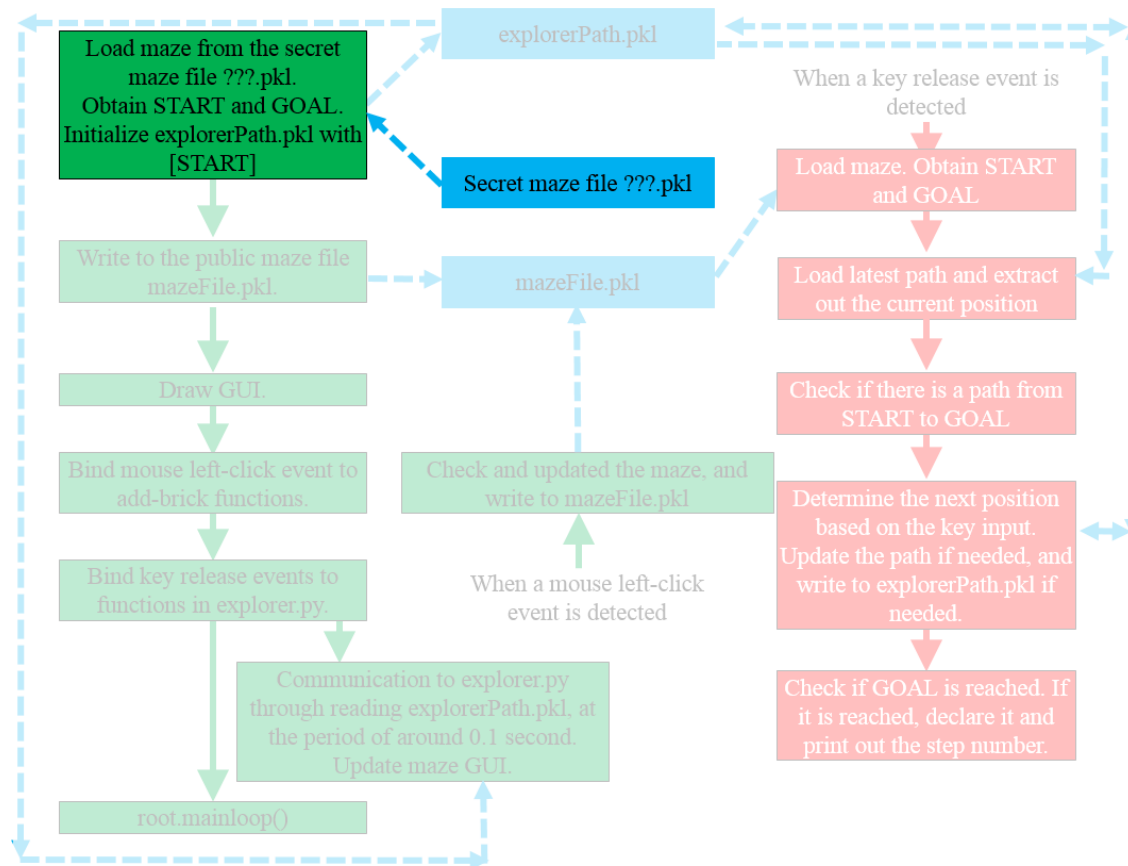


Fig. 10 Coordination between *mazer.py* and maze file (updated)

Suggestions:

- Design more than one maze, and put the additional brick in appropriate place and at appropriate time. This will increase the step number of the Guest team.
- If you find that the robot is in the dead-end path, put the additional brick in appropriate place and at appropriate time to trap the robot.

Host Team: Student B³**Suggestion:**

Student B is in charge of the coding, named *mazer.py*, for the GUI of the Host team. *mazer.py* is the main programme, run in the competition. At appropriate positions, it calls functions in *explorer.py*, written by the Guest team.

- A suggested flowchart is illustrated as below:

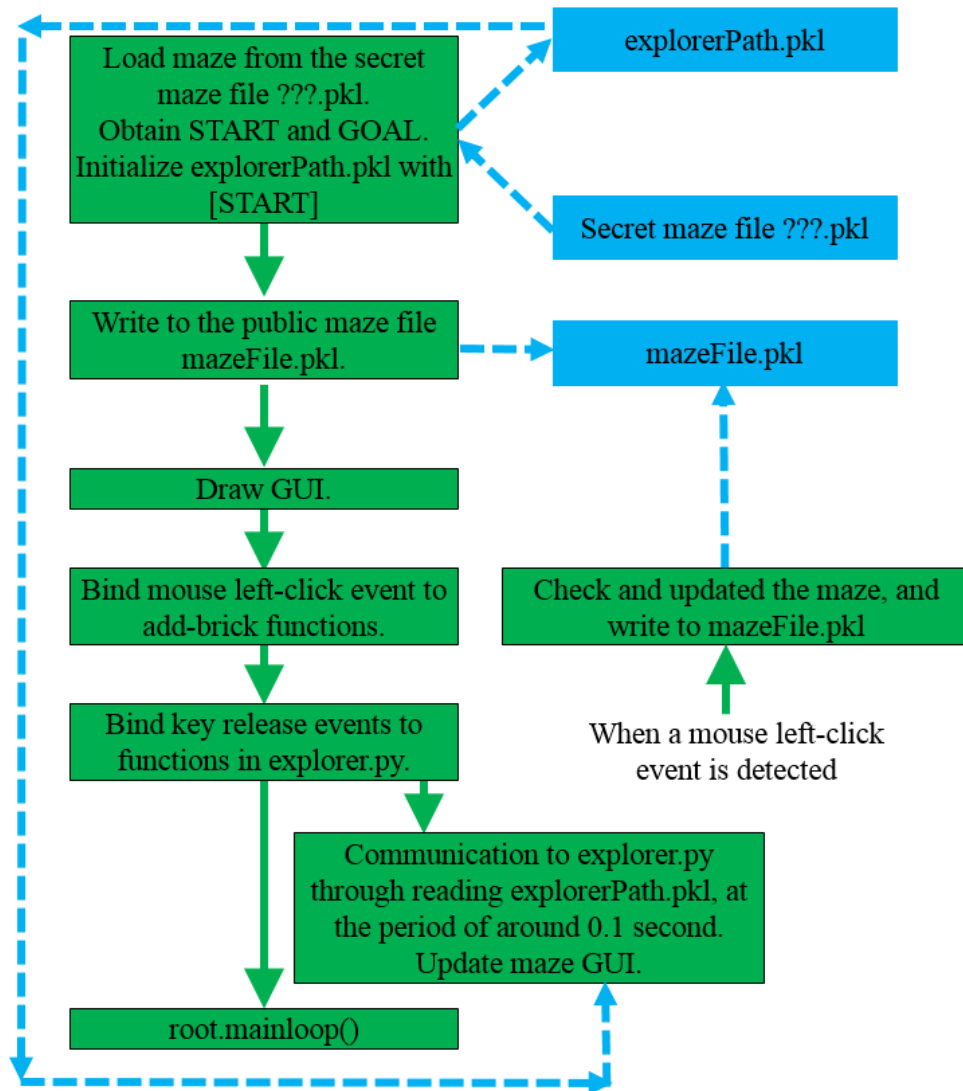


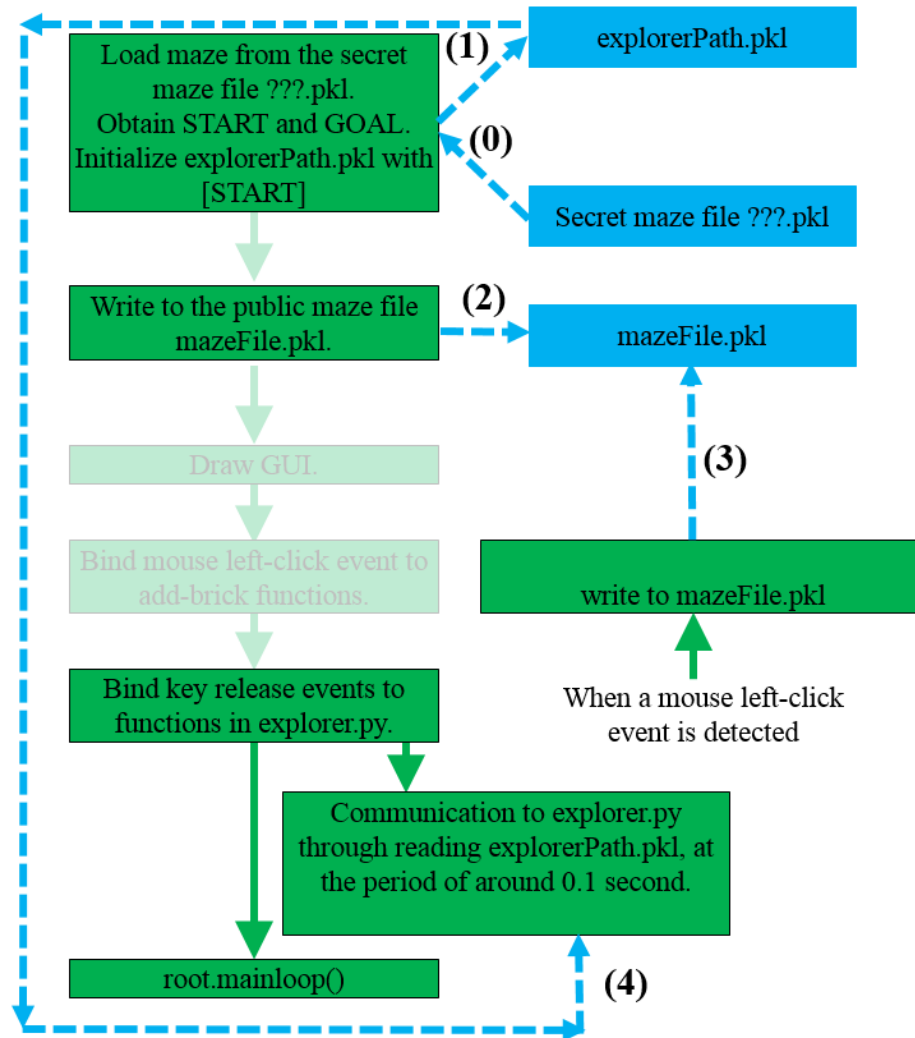
Fig. 11 Suggested flowchart of *mazer.py*

Regulations:

- In the GUI, show an image of robot to the current position. We will learn how to include an image in the label later.

³ Updated for non-fixed START and GOAL.

- Appropriate coordination between *mazer.py* and *explorer.py* is crucial. We have provided a template of *mazer.py* (to be completed by Host team). The figure below highlights the provided codes. You should add your codes to the indicated parts of the *mazer.py* file. **DO NOT modify the pre-provide coding!**
- (0), (1), (2), (3) and (4) indicate **five** parts of file read/write codes. To prevent from inappropriate file operations, which will result in communication failure with *explorer.py*, DO NOT write other file read/write functions.

Fig. 12 Pre-provided modularized coding in *mazer.py*

- Here below we elaborate the pre-provided coding.
 - At the beginning of *mazer.py*
 - Import modules
Guest team's code file name is *explorer.py*, so import it to call its functions. There might be more modules to import.

```

from tkinter import *
import pickle
import explorer #Import explorer as a module and call its functions later.

```

- Global variables

The global variables below are used in the pre-provided functions.

```
# Use these global variables to enable the pre-provided functions.

'''Actually, you do not need to declare the global variables below here. You can either declare
them in the functions where it is assigned, or directly use these variables when assigning
them values in your main script.
The declaration is presented here only for better understanding of the coding.
...

global currentPosition #This is the current position
global root #Define your Tk object as root
global maze #maze contains the maze information loaded from your original secret maze file,
            #identical to the maze information written to mazeFile.pkl, and changed when
            #the additional brick is added. It contains values:
            #0: empty block which is not explored
            #1: brick
            #2: START or the current position
            #3: GOAL
global tempMaze #tempMaze contains the maze information with explored blocks. It will not be
               #written to mazeFile.pkl for read by explorer.py. Its difference with maze is
               #only in the new value below:
               #4: the block is previously empty and has been explored.
global labelCurrentPositionExplorer #It is the Label object showing the current position in
                                    #explore.py. CommunicationMazerExplorer() will update it
                                    #and thus we make it global variable
global START, GOAL #The START and GOAL positions. They can be located in any two distinct positions
                   #of the maze, except for the positions
```

- Pre-provided functions

To standardize the communication among the codes of Host and Guest teams, and prevent from in-compatibility, the codes for are provided as below.

LoadMazeFile(): load from the initial secret maze file, and extract START and GOAL.

```
#(0) File read: only in initialization
#The function below read from the secret path .pkl file.
#START and GOAL can be located in any two distinct positions of the maze, except for the positions
#on boundary.
def LoadMazeFile():
    'Load from the initial secret maze file, and extract START and GOAL.'
    global maze, START, GOAL
    file=open('mazeFileForTest.pkl','rb')#For Host team: You can change to your .pkl file name.
    maze=pickle.load(file)
    file.close()
    for i in range(len(maze)):
        if 2 in maze[i]:
            START=(i, maze[i].index(2))
        if 3 in maze[i]:
            GOAL=(i, maze[i].index(3))
```

InitializeExplorerPath(): initialize explorerPath.pkl for explorer.py. It should be only called once at the initialization stage.

```

#(1) File write: only in initialization
#The functions below write the explorerPath.pkl for communication initialization with explorer.py.
#Just include it in the coding. You do not need to call it.

def InitializeExplorerPath():
    'Initialize explorerPath.pkl for explorer.py. mazer.py write explorerPath.pkl only this time.'
    file=open('explorerPath.pkl','wb') #Overwrite the previous path information.
    pickle.dump([START], file)
    print('In Mazer: explorerPath.pkl initialized.')
    file.close()

```

Initialization(): Load maze and initialize the explorePath.pkl. It should only be called once at the beginning. DO NOT call it again.

```

#Initialize by calling LoadMazeFile() and InitializeExplorerPath()
def Initialization():
    LoadMazeFile() #Call LoadMazeFile() to read from the original secret path file.
    InitializeExplorerPath() #Call InitializeExplorerPath()
                        #to initialize explorerPath.pkl at the begining of your file.

Initialization()

#The function above read from the original secret path file, and the write to the explorerPath.pkl for
#communication initialization with explorer.py.
#Just include it in the coding. You do not need to call it again.

```

WriteMazeFile() will write the global variable *maze* to mazeFile.pkl, so that explorer.py can read. Call it at appropriate positions in *mazer.py*.

```

#(2) File write: in initialization
#(3) File write: when a new brick is added
#The function below is pre-provided for writing mazeFile.pkl.
#Call it at appropriate place to coordinate mazer.py to explorer.py communication.

def WriteMazeFile():
    'Write the global variable maze to mazeFile.pkl for explorer.py to read.'
    file=open('mazeFile.pkl','wb')
    pickle.dump(maze, file)
    file.close()

#The function above is pre-provided for writing mazeFile.pkl.
#Call it at appropriate place to coordinate mazer.py to explorer.py communication.

```

CommunicationMazerExplorer() reads explorerPath.pkl around every 0.1 second, and extract out the current position. If there is any move, it updates the GUI. It calls two functions, ReadCurrentPosition and UpdateGUI

ReadCurrentPosition() read explorerPath.pkl and extract out the current position.

UpdateGUI() updates the global variable tempMaze (NOT maze), and update the GUI. You need to write the function ReDraw() to update the GUI.


```

#(4) File read: every 0.1s approximately
#The functions below are pre-provided for read from explorerPath.pkl around every 0.1 second and update
#the global variable tempMaze and the GUI.
# -- ReadCurrentPosition(): read from explorerPath.pkl, and assign current position to the global variable
#    currentPosition. It is called in CommunicationMazerExplorer().
# -- CommunicationMazerExplorer(): read from explorerPath.pkl around every 0.1 second, and if there is move
#    to a new position, update the the global variable tempMaze and show it in the GUI.
# -- UpdateTempMaze(previousPosition, currentPosition): update the global variable tempMaze and show it in
#    the GUI. !!! There is still need to write a function ReDraw to update the GUI.

def CommunicationMazerExplorer():# Use this function before the root.mainloop()
    ''' This function checks the new current position from Explorer, updates the labelCurrentPositionExplorer in GUI,
    stores the new current position in path, and updates the global variable tempMaze (ReDraw the GUI in function UpdateMaze()).
    CommunicationMazerExplorer() will be called around every 100 mini seconds.
    ...
    global currentPosition, path
    previousPosition=currentPosition
    ReadCurrentPosition()#The global currentPosition will be changed by the position information in explorerPath.pkl

    labelCurrentPositionExplorer.configure(text='In Explorer: \ncurrentPosition=(%s,%s)'%(currentPosition[0],currentPosition[1]),
                                          justify=CENTER)
    #labelCurrentPositionExplorer shows the current position feedback from Explorer.

    if currentPosition!=previousPosition:
        UpdateGUI(previousPosition,currentPosition)
        #This function only updates tempMaze, rather than maze, which is changed only when you add a brick.

    root.after(100, CommunicationMazerExplorer)#CommunicationMazerExplorer() will be called every 100 ms.

def ReadCurrentPosition():
    '''Read the current position stored in currentPosition.pkl and assign to the global variable currentPosition.
    After InitializeCommunicationFile(), only explorer.py can write currentPosition.pkl, and thus it is the latest current
    position information after keyboard input.
    ...
    global currentPosition, path
    file=open('explorerPath.pkl','rb')
    path=pickle.load(file)
    currentPosition=path[len(path)-1]
    file.close()

def UpdateGUI(previousPosition, currentPosition):
    '''Update the global variable tempMaze, and then update GUI by ReDraw().
    This function is called in CommunicationMazerExplorer().
    You may not need to call it in your coding.'''
    tempMaze[previousPosition[0]][previousPosition[1]]=4
    tempMaze[currentPosition[0]][currentPosition[1]]=2
    ReDraw() #For Host team: ReDraw will update the maze GUI. You need to write it. It does not need to return anything.

```

CountBricks() returns the brick number in the maze.

#The function below counts the number of bricks in the maze. Call it when needed.

```

def CountBricks():
    'It counts the brick number in the global variable maze, and return it.'
    brickNumber=0
    for mazeRow in maze:
        brickNumber+=mazeRow.count(1)# Add the number of 1s, i.e. bricks.
    return brickNumber

```

#The function above counts the number of bricks in the maze. Call it when needed.

- At the end of *mazer.py*
 - Bind the events of key press to the functions written in explorer.py. When the direction keys (up, down, left and right) are pressed, the corresponding functions in *explore.py* will be activated to write the new current position of the robot into the completed latest path in explorerPath.pkl.

For more details in the way to bind keyboard events (and also the **mouse click events on Labels**), we will introduce in the future.

- Thereafter, run the periodically called function `CommunicationMazerExplorer()` mentioned above to check information from `explorer.py` through the file `explorerPath.pkl`.

```
####Include the code below and use them in mazer.py. Do not modify them, unless instructed to do so.###
root.bind("<KeyRelease-Up>", explorer.KeyUp)#To Host Team: We change the KeyPress event to KeyRelease event,
                                             #to prevent from continuous KeyPress. When the key is released from
                                             #the pressed state, the event will be activated.
root.bind("<KeyRelease-Down>", explorer.KeyDown)
root.bind("<KeyRelease-Left>", explorer.KeyLeft)
root.bind("<KeyRelease-Right>", explorer.KeyRight)
CommunicationMazerExplorer()
root.mainloop()
```

Guest Team: Student C⁴

Suggestion:

Student C is in charge of the coding, named *explorer.py*, for the robot to explore the maze.

- The flowchart of the coding is suggested as below.

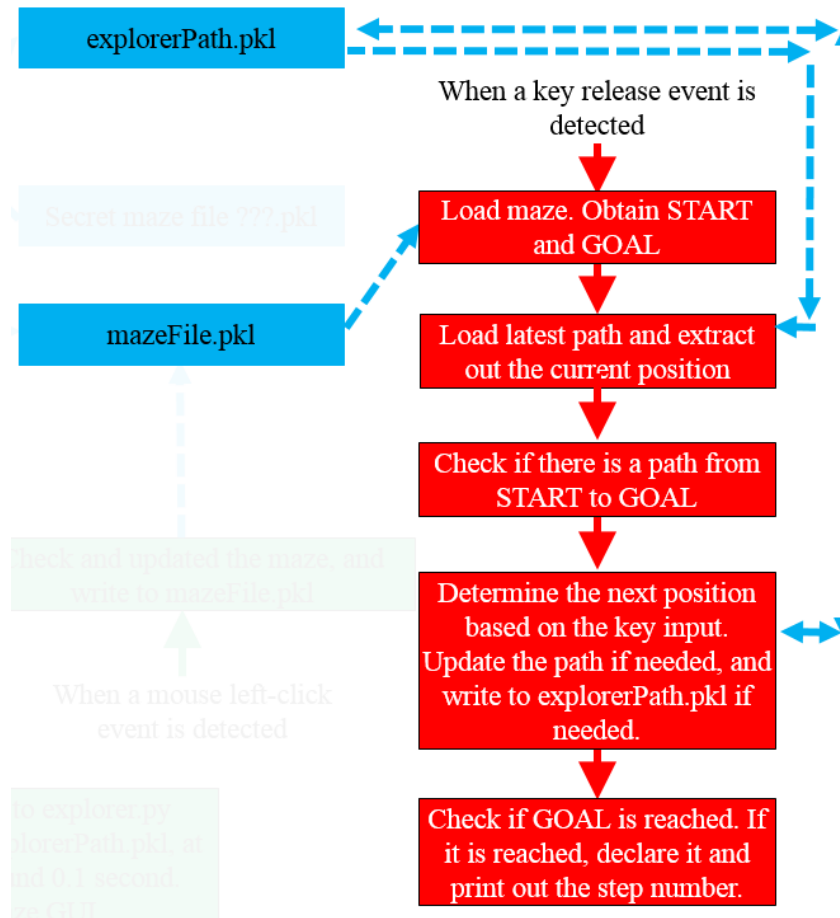


Fig. 13 Suggested flowchart of *explorer.py*

- On how to detect whether there is a path from START to GOAL, there will be different ways. We will teach on recursion-based approach in the future.

Regulations:

- When there is one key release event (i.e. hit the key and release it once), only one additional path node can be added to *explorerPath.pkl*.
- Appropriate coordination between *explorer.py* and *mazer.py* is crucial. We have provided a template of *explorer.py* (to be completed by Guest team). The figure below highlights the provided codes. You should add your codes to the indicated parts of the *explorer.py* file. **DO NOT** modify the pre-provide coding!

⁴ Updated for non-fixed START and GOAL.

- (1), (2), and (3) indicate three parts of file read/write codes. To prevent from inappropriate file operations, which will result in failure in communication with *mazer.py*, DO NOT write other file read/write functions.

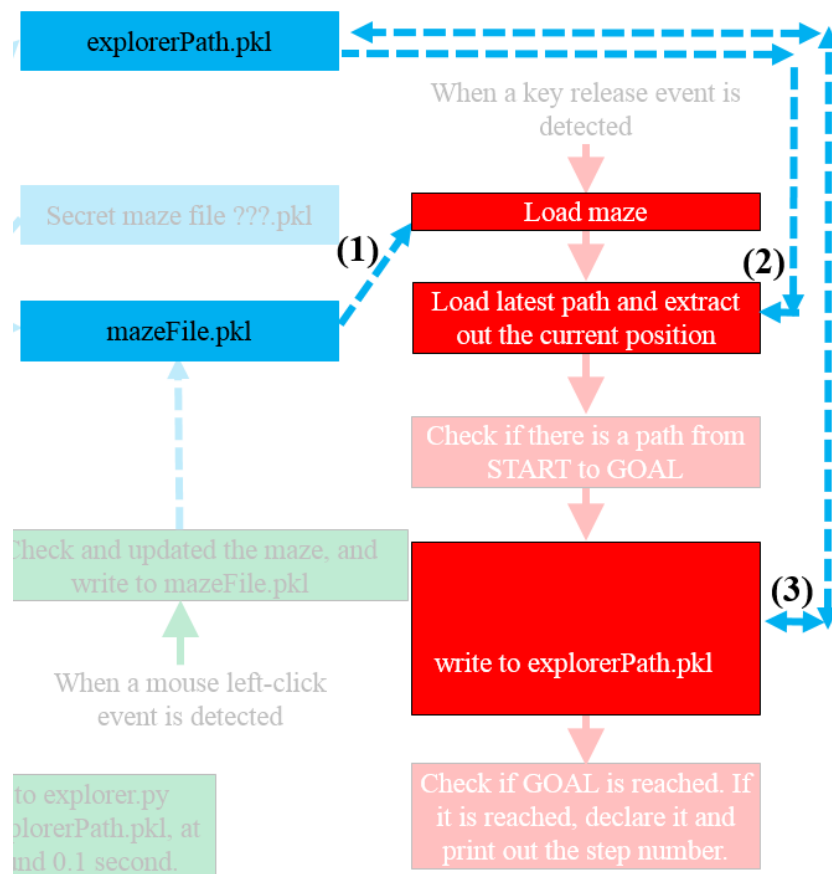


Fig. 14 Pre-provided modularized coding in *explorer.py*

- Here below we elaborate the pre-provided coding.
 - Import module and set global variables. The global variables below are used in the pre-provided functions.

```
import pickle

#Use the following global variables to enable the pre-provided functions
'''Actually, you do not need to declare the global variables below here. You can either declare
them in the functions where it is assigned, or directly use these variables when assigning
them values in your main script.
The declaration is presented here only for better understanding of the coding.
...

global maze #maze contains the maze information loaded from your original secret maze file,
            #identical to the maze information written to mazeFile.pkl, and changed when
            #the additional brick is added. It contains values:
            #0: empty block which is not explored
            #1: brick
            #2: START or the current position
            #3: GOAL
global currentPosition #This is the current position
global path #The previous path trajectory stored in explorerPath.pkl
global START, GOAL #The START and GOAL positions
```

- LoadMaze() reads from mazeFile.pkl and put into the global variable maze. This prevents from loss of the maze information.

```

#(1) File read: when keyboard event is activated, load maze
def LoadMaze():
    'Load the maze information into the global variable maze.'
    global maze, START, GOAL
    file=open('mazeFile.pkl','rb')
    maze=pickle.load(file)
    file.close()
    for i in range(len(maze)):
        if 2 in maze[i]:
            START=(i, maze[i].index(2))
        if 3 in maze[i]:
            GOAL=(i, maze[i].index(3))

```

- ReadCurrentPosition() read explorerPath.pkl and extract out the current position.

```

#(2) File read: load path and get current position
def ReadCurrentPosition():
    '''Get the previous stored path from explorerPath.pkl and put it to the global variable path.
    Extract out the current position and put it in the global variable currentPosition.
    Call this function when needed.'''
    global path, currentPosition
    file=open('explorerPath.pkl','rb')
    path=pickle.load(file)
    currentPosition=path[len(path)-1]
    file.close()

```

- WriteExplorerPath() stores the most complete path (with the latest move) for *mazer.py* and *explorer.py* to read. This prevents from the loss of the current position information. To prevent from your mistakes in operation to the variable path, we re-load it from explorerPath.pkl, append the new position, and write it back to explorerPath.pkl.

#(3) File read & write: load last path and store the new path

```
def WriteExplorerPath():
    file=open('explorerPath.pkl','rb')
    path=pickle.load(file)
    file.close()
    file=open('explorerPath.pkl','wb')
    path.append(currentPosition)
    pickle.dump(path,file)
    file.close()
```

- *mazer.py* will call the functions named KeyUp, KeyDown, KeyLeft, and KeyRight, when the four key events are activated. Complete them.

#To Guest team: complete these functions. DO NOT change the function names. Otherwise, mazer.py can not call them.

```
def KeyUp(self):
    #To Guest team: complete this function. DO NOT change the function name, so that mazer.py can call it.

def KeyDown(self):
    #To Guest team: complete this function. DO NOT change the function name, so that mazer.py can call it.

def KeyLeft(self):
    #To Guest team: complete this function. DO NOT change the function name, so that mazer.py can call it.

def KeyRight(self):
    #To Guest team: complete this function. DO NOT change the function name, so that mazer.py can call it.
```

Grading Policy and Submission Method

This group project takes up 50 points of your final grade, including

- 20 points in coding (for the group)
Assessed based on the final coding (*mazer.py* and *explorer.py*) submitted at the end of the competition.
- 10 points in competition result (for the group)
Assessed based on the result.
 - 2 points, if your *mazer.py* can run successfully as Host.
 - 2 points, if your *explorer.py* can run successfully as Guest.
 - 2 more points, if your team get into Round Level-2.
 - 1 more point for final 8 teams.
 - 1 more point for final 4 teams.
 - 1 more point for final 2 teams.
 - 1 more point for the champion.
- 20 points in report (for individual students)
In your report, include:
 - Your weekly progress. Make an itemized weekly diary, recording what you have completed that week, what are the problems you have discovered, what are comments given by your teammates, and what is expected to do in the next week.

For the completed part, include some images, data, or source code, either in the document, or as an attachment.

E.g.

Completed:

1. Tested the pre-provided functions in *mazer.py*.
2. Designed and tested the GUI.

Problems discovered:

1. Cannot draw the path in GUI

Comments from the team:

1. Try to print out the path in IDLE shell to check if the path information is successfully obtained.

Next week plan:

1. Solve the path draw problem.
2. Test together with maze file
3. Design the add brick function.

Submission method:

1. Your programs (*mazer.py*, *explorer.py*, and the *.pkl* maze files) copied to the competition computer is deemed as final submissions. Hence, you do not need to submit again in Moodle.
2. Each student: submit through Moodle the individual report in a MicroSoft Word file **by 17:30, May 11, 2016 (Wednesday)**.

Each one-day late in submission will result in 2 point deduction. If submitted after 5 days, the 20 points will be all deducted.

Final Words

Team spirit is very important for research and big project, because no one can do the whole work by him/herself.

This project is designed to train you in 2 levels of collaborations, i.e. *within team* and *between teams*.

- Collaboration within team

Weekly discussion is expected for your team to make consistent progress. DO NOT postpone the effort to the final week.

- Collaboration between teams

For the coding of both Host and Guest teams to function appropriately, each team should comply with the regulations set above. You can also play with other teams with a test maze file (a simple and not the one your team will use in the real competition), so that you can guarantee that the coding can work together.

You may approach instructor and TAs when needed in office hours and tutorial time, if there is any question.