Name: Abdullah Yağız Özbek

Project: Blind AI implementation with MATLAB and Torque 3D Game Engine interactions

1. TOOLS
   MATLAB, TORQUE 3D GAME ENGINE, TORQUE SCRIPT, SOCKET PROGRAMMING. MATLAB used in computational purposes and Torque Engine used in visual representation and informing MATLAB about some problems.
2. PURPOSE
   To learns how to construct communication between MATLAB and Torque Engine with use of Socket Programming, to show computational advantages of MATLAB and what can be done with these 2 tools (In this case it is a blind AI pathing.).
3. WHAT IS DONE
   Communication between MATLAB server and Torque Engine client is achieved. Blind AI can get to the intended position.
4. HOW AI WORKS
   First, AI have no idea about map information. He is blind. MATLAB server side starts server connection and waits for input from Torque Engine, where AI resides. Torque engine sends AI position and player position. AI position is used as start point and player position is used as target point. MATLAB server takes these positions and with use of A* pathing algorithm it finds a path and sends it back to the Torque Engine. Torque engine starts to follow the path and in the way if it gets stuck in some place, it informs MATLAB about there is an obstacle interferes the path and MATLAB marks the path element that AI could not achieved as an obstacle and calculates the path and sends it again. This cycle repeats until AI achieves its goal.

Some pictures to show how communication is happening:

Type server to execute server.m in MATLAB:

```
>> server
```

In command window of Torque Engine type start_socket(); to start communication in Torque Engine:

When connection is achieved this will be printed in Torque Engine:

First 3 values are the AI's x y z coordinates and last 3 values are the Player's x y z coordinates.

When this information is read by MATLAB server, this will be printed:

```
starting path calculations
path found
took 0.21875 seconds to find path
path found sending
sending path
sent 15 15 5 33 6 32 7 31 8 30 9 29 10 28 11 27 12 26 13 25 14 24 15 23 16 22 17 21 18 20 19 19
```

5 33 is the start position of AI. However, you see the first 2 elements are 15 15 because as a first element MATLAB server sends size of the path. Read the path as they are a pair. For example: AI will go first to x:6 y:32 then, x:7 y:31 and so on. Sending start position might look like redundant but it is necessary. Sometimes AI tries to pick target position as an obstacle, but it is a wrong action to do since it will cause path to be not found even though there is one! To prevent this problem, MATLAB will pick start position as an obstacle. When start position becomes an obstacle, MATLAB checks start points surroundings to find a new start point. If it finds everything is okay, if it cannot then, it means there is no path since AI is surrounded by obstacles and cannot get out.

When AI is stuck somewhere this is printed in torque engine:

```
to the next path 13 25
stuck at 13.3455 26.1294 -0.0205285
```

You see AI was trying to get to the position x:13 y:25 but it stuck. Torque engine informs MATLAB position x:13 y:25 as a possible obstacle.

When stuck information is read by MATLAB server:

```
new obstacle 13 25
path found
took 0.03125 seconds to find path
sending path
sent 8 8 13 26 14 25 15 24 16 23 17 22 18 21 18 20 19 19
```

Obstacle data is informed, and MATLAB adds new obstacle data to the obstacles array. So, when trying to find a new path, MATLAB will dismiss that position.

When AI achieves its goal:

```
achieved the goal!
finish request sent
```

When finish request read by MATLAB server:

```
Mission Done
Unrecognized function or variable 't'.

Error in server (line 116)
fclose(t);
```

Error message is not important. It means socket is deleted and MATLAB cannot find it.

5. MATLAB DATA STRUCTURES

For more information read comments in the code files.

Node.m class:

```
properties
    x
    y
    distance
    g
    h
    isProcessed
    edge_to
    node_count
    isObstacle
end
```

X: x coordinate
Y: y coordinate
Distance: cost of the node. It is used in A* algorithm to choose node with the lowest cost to process. Distance always equals to g + h.
G: Distance between parent node and current node
H: Distance between current node and target node.
isProcessed: If popped node from the priority queue is already processed isProcessed value is 1, otherwise 0. If it is 1 then popped node is dismissed. Else it is processed.
Edge_to: Holds pointing parent node with lowest cost. It is used to construct path.
Node_count: It is the number of nodes to get to the current node. It is used as a path size. If target node's node_count is equal to 0, it means there is no path.
isObstacle: not used!

pq_handle.m class: Simple min priority queue implementation.

```
properties(SetAccess = public)
    array
    top
    bottom
end
```

Array: Priority queue implemented as a cell array since it holds node objects.

Top: position of next data to be inserted.
Bottom: always equals to 1.

Astar.m function: Calculates path and returns it.

```
function [path] = Astar(mapsize,start,target,obstacles,obsize)
```

Path: returned path array. If there is no path returns -1.
Mapsize: size of the map. For example, if mapsize is equals to 200. It creates a grid with diagonals x:200 y:200, x:-200 y:200, x:-200 y:-200, x:200 y:-200. Choose map size wisely since it effects the performance. To decide map size, find or decide max coordinate possible. For example, max coordinate possible is -1234,450. Then, choose map size as 1250 or 1300.
Start: start position.
Target: target position.
Obstacles: obstacle array
Obsize: size of obstacle array

Astar.m helper function cal_index(mapsize,x,y):

```
function [row,col] = cal_index(mapsize,x,y)
    %calculates index of the node according to its world coordinates
    row = mapsize + x + 1;
    col = mapsize + y + 1;
end
```

According to given coordinate information, it finds index of that coordinate in Nodes array. Coordinates must be indexed!

6. MATLAB SERVER SIDE

Server.m executable: Creates TCPIP object and server connection. It is responsible from all the communications.

Create socket:

```
t = tcpip('123.123.12.1', 12345, 'NetworkRole', 'server');
fopen(t);
data = 0;
```

First parameter is ip of the server. In this case you can use your own computers ip(if you are using windows at command window execute command ipconfig to learn your computers ip.).
Second parameter is port. You can use 12345.
Do not touch other parameters. It just defines MATLAB as a server.
In second line we just open the file to be read from that ip address and port.

Listen loop:

```
%start listening
while true
    %wait until there is data to be read
    while t.BytesAvailable <= 0
        pause(0.1);
    end
```

While there is no data to be read it waits. In other words, listens. While listening, pause by 0.1 seconds at every iteration. Otherwise, it will overwhelm the CPU.

When there is data to be read, meaning Torque Engine sent some data:

```
if t.BytesAvailable > 0
    %read data
    data = fread(t, t.BytesAvailable);
```

Read the data into data variable and processs it. Note that fread, returns a numeric array but there is a problem. The data sent was not a numeric but string! So, we need to cast the data into char array and decode the original data sent by turning data char array into string.

```
data = char(data);
```

Turn data to its original shape:

```
str = "";
for i = 1:length(data)
    str = str + data(i);
end
```

Then we split str by whitespace delimiter:

```
data = split(str);
```

Torque Engine can send “done”, “start” and “stuck” as a first word.
If it is “done”, means AI achieved its goal and MATLAB stops server.
Of it is “start”, Torque Engine sends “start” command when connection is set with the server. Torque engine also sends AI start position and players position so that MATLAB can start to compute path.
If it is “stuck”, means AI stuck somewhere and with given information to the MATLAB.

At the end of every “start” and “stuck” command MATLAB (re)calculates path:

```
path = Astar(mapsize,start,target,obstacles,obs_index-1);
```

After path is calculated:

IF path does exist:

```
if length(path) ~= 1 && (strcmp(data(1),"start") || strcmp("stuck",data(1)))
    str = "";
        disp("sending path");
        for i = 1:length(path)
            str = str + path(i,1) + " " + path(i,2) + " ";
        end
    fprintf(t,str);
    disp("sent "+str);
else
```

Then, returned path have size greater than 1 and we can send the path. Path must be sent as a string so, we reshape the numeric path array to the string.

IF path does not exist:

```
else
    fprintf(t,"nopath");
    disp("no path found. Disconnecting...");
    fclose(t);
    clear;
end
```

Means, path is not an array but numeric value equals to -1. Note that no path information also sends when start point and target point are the same. “nopath” information is sent to inform Torque engine that there is no path.

After MATLAB reads the data and sends it to the Torque engine, it will pause by 1 seconds to achieve synchronized communication. You can try to delete that part but probably it will cause errors. Or not since code changed heavily to prevent that.

```
pause(1);
```

7. WHERE TO WRITE YOUR CODES IN TORQUE ENGINE

It is hard to find where to write your codes in Torque engine and I covered it for you in here.

To start the game, go to:

Torque3D-master/My Projects/Client/game/Client.exe

To see Torque engine code file I wrote, go to:

Torque3D-master/My Projects/Client/game/scripts/client/client_socket.cs

Note: Do not confuse yourselves with .cs as a C sharp code file. It is torque script file. To open it you can simply download Notepad++.

To find where my codes are executed, go to:

Torque3D-master/My Projects/Client/game/scripts/client/missionDownload.cs

and find the function:

```
//--------------------------------------------------------------------------
// Mission loading done!
//--------------------------------------------------------------------------

function onMissionDownloadComplete()
{
   echo("GAME STARTED!");
   exec("./client_socket.cs");

}
```

You can see I executed my code file with exec() function.
onMissionDownloadComplete function is called when you enter the game and start playing. So, it is perfect place for executing your codes here.

8. TORQUE ENGINE AI and CALLBACK FUNTIONS

Inside client_socket.cs:

AI side:

Create AI player:

```
54  new AiPlayer(Bob){
55      dataBlock = DefaultPlayerData;
56      position = "5 33 1";
57      moveStuckTolerance = 20;
58      mMoveTolerance = 1;
59  };
```

dataBlock: is the same data block with our player. It defines its animations, textures and etc.
position: Spawn position.
moveStuckTolerance: When AI gets stuck somewhere, engine starts to count ticks and if it is equals to that variable onMoveStuck() call back function is called to inform us that AI is stuck somewhere.
mMoveTolerance: Think about some target position as a center of a sphere and mMoveTolerance as a radius of that sphere. If AI enters that sphere, it is accepted as AI achieved its goal and onReachDestination call back is called.

onMoveStuck call back function:

```
8  function PlayerData::onMoveStuck(%this,%obj){
9      /*
10     When AI stucks at some point it means that AI could not arrived to the next decided path. So, we mark this positi
11     obstacle and send it to the matlab server. Matlab will try to search path from the current point of AI. Target pc
12     */
13     if($once){
14         echo("stuck at "@ %obj.getPosition());
15         echo("box features " @ %obj.getWorldBox());
16         Bob.stop();
17         %start = Bob.getPosition();
18         %end = $save_end;
19         MATLABSocket.send("stuck "@ $path[$next_path,0]@" "@$path[$next_path,1]@ " a" @ " " @ %start @ " " @ %end );
20         //MATLABSocket.send("stuck " @ %obj.getWorldBox() @ " " @ %start @ " " @ %end @ " " @ Bob.getForwardVector())
21         $next_path = 0;
22         $previous_path = -1;
23         $path = null;
24         $once = false;
25     }
26 }
```

onMoveStuck call back function is designed for AI so, it won't work in players.

%this: It is the AI object that is stuck.
%obj: It is the object that AI is stuck into(Think it as a Collision).
Since I used only 1 AI, instead of using %this variable I used Bob in every function. So, if you want to add more than 1 AI you need change and add too many things. Even MATLAB side probably.

When this function is called it means that AI is got stuck somewhere and it will send new obstacle information to the MATLAB server.
When onMoveStuck is called you will see this line:

```
to the next path 13 25
stuck at 13.3455 26.1294 -0.0205285
```

%start: Is the new start position of the AI.
%end: target point is didn't change. Note that to observe AI, we as a player move around a lot so target position is changing. To prevent that, when communication is started, we save player location in $save_end global variable so that we can send same target position when AI get stuck(or you can just save it in MATLAB server so that you do not have to send target position every time.).

Sending obstacle data, start position and target position to the MATLAB server:

```
MATLABSocket.send("stuck "@ $path[$next_path,0]@" "@$path[$next_path,1]@ " a" @ " " @ %start @ " " @ %end );
```

Remember MATLAB side? If AI is stuck somewhere first element of data is "stuck" command. Then, obstacle information (x and y pair). After, I gave unnecessary character "a". It has no meaning. It is just supposed to be the z coordinate of the obstacle, but it is unnecessary in my project. Lastly, start and target positions. MATLAB server will split this string to string arrays.

onReachDestination call back function:

```
27 function PlayerData::onReachDestination(%this,%obj){
28     /*
29         This is called when AI reaches its destination. If current path did not finish,
30         mission done message to the matlab server and socket stops.
31     */
32     if(strcmp($target_comp,$path[$next_path,0]@" "@$path[$next_path,1]) != 0){
33             $previous_path = $next_path;
34             $next_path++;
35             Bob.setMoveDestination($path[$next_path,0]@" "@$path[$next_path,1],false);
36             echo("to the next path " @ $path[$next_path,0]@" "@$path[$next_path,1]);
37     }
38     else{
39     echo("achieved the goal!");
40     bob.stop();
41     MATLABSocket.missionDone();
42     }
43 }
```

We keep path array in $path global variable. AI Bob must achieve these path elements one by one and every time it achieves some path position in the array, onReachDestination call back function is called to inform us AI achieved its goal. Then, we set next position AI needs to go. If AI achieved its final goal which is getting

to the target position, Torque engine sends “done” command to the MATLAB server by using missionDone() function. Problem with $path variable is that it overwrites its data everytime new path is found. So, using last element of path array becomes useless. So, in order to detect whether AI is achieved its final goal or not we keep target position in $target_comp variable and do string comparison with achieved path element.

9. TORQUE ENGINE CLIENT SIDE and CALLBACK FUNCTIONS

Inside client_socket.cs:

Start_socket() function:

```
function start_socket(){
    /*
        First function to call. It cr
        remember that ip of matlab se
        as a port you can use 12345.
    */
%t = new TCPObject(MATLABSocket);
%t.lastState = "None";
%addr = "123.123.12.1";
%port = "12345";
%t.connect(%addr@":"@ %port);
}
```

It is where we create our socket object with name MATLABSocket so that we can use this object and its methods in game with using this name.

%addr: It your MATLAB server’s ip address.

%port: Port where your MATLAB server uses.

With using this address and port we call connect() method to connect to the MATLAB server.

onConnected() call back function:

```
function MATLABSocket::onConnected(%this) {
/*
    After connection is set this callback is called
    basically, it just sends AI position as start p
    target point.
*/
// Output red text to show we connected
echo("Connected to MATLAB Computing Server");
// Send the request for the file to the server
%start =Bob.getPosition();
$save_start = %start;
%end = LocalClientConnection.player.getPosition();
$save_end = %end;
        %temp = %end;
        %temp = nextToken(%temp,"token"," ");
        %a = %token;
        %temp = nextToken(%temp,"token"," ");
        %b = %token;
        $target_comp = %a @ " " @ %b;
%all ="start " @ %start @ " " @ %end;
%this.send(%all);
echo("sent ",%all);
}
```

This callback function is called after connection is achieved via start_socket() function. We get AI Bob's and player's position information. Note that LocalClientConnection.player.getPosition() can be called only if you are the host. We save target position in $save_end and $target_comp variables here. Difference between the two variables is that $target_comp does not have the z coordinate information since we use it in comparing with path pairs which are x and y pairs. Remember that "start" command is used in first data transfer with MATLAB server. It sends AI position and player positions to the MATLAB server.

onLine() call back function:

```
function MATLABSocket::onLine(%this, %line)
{
    /*
    this function is called when matlab server resp
    Response is a path array or path not found mes:
    If it is an array, the first 2 tokens are size
    path array and set AI's first destination. Reme
    array. It comes to be usefull when start point
    If its not a path but no path message, first t
    */
    $next_path = 0;
    $previous_path = -1;
%line = nextToken(%line,"token"," ");
%nopath = %token;
```

This function is called when MATLAB server responds. %line variable hold respond message. In every call of onLine() function, it means new path is returned so, $next_path variable is reassigned to 0. It first takes the first token the figure out if MATLAB found a path or not. If first token is equal to the string “nopath”, it means there is no path and it simply disconnects from the MATLAB server.

No path found:

```
else{
    echo("no path found!! Disconnecting...");
    Bob.stop();
    MATLABSocket.missionDone();
}
```

Path found:

```
if(strcmp(%nopath,"nopath") != 0){
%line = nextToken(%line,"token"," ");
$path_size =  %token;
echo("path size " @ $path_size);
$path = null;
$path = new Array($path_size,2);
echo("path start");
for(%i = 0; %i < $path_size ; %i++){
    if(%i != $path_size-1){
    %line = nextToken(%line,"token"," ");
    $path[%i,0] = %token;
    %line = nextToken(%line,"token"," ");
    $path[%i,1] = %token;
    }
    else{
        %temp = $save_end;
        %temp = nextToken(%temp,"token"," ");
        $path[%i,0] = %token;
        %temp = nextToken(%temp,"token"," ");
        $path[%i,1] = %token;
    }
    echo("path " @%i @ ": x " @ $path[%i,0] @ " y " @ $path[%i,1]);
}
echo("path end");
//echo("array length" @ length($path));
Bob.setMoveDestination($path[$next_path,0]@" "@$path[$next_path,1],false);
echo("to the next path " @ $path[$next_path,0]@" "@$path[$next_path,1]);
$previous_path = $next_path;
$next_path++;
$once = true;
}
```

Note that we already consumed the first token and if there is path the first 2 tokens are the size of the path so, it again reads another token to leave path information only and save path size in $path_size variable.

$path global variable is 2-dimensional array. It has $path_size number of rows and 2 columns. First colums is the x coordinate and second is the y coordinate. It simply reads tokens one by one and assign them into $path variable's columns. Note that on last iteration of the loop instead of reading last tokens from the %line variable, it reads from the $save_end variable since when we re-declare $path array, it overwrites the previous one instead of creating new empty array so, to decide whether AI Bob reached its destination or not we will compare it with $save_end variable that we previously assigned.

After $path array is assigned it's values we give the first path to go to AI Bob and $once variable re-assigned to true for onMoveStuck call back function.

missionDone() function:

```
function MATLABSocket::missionDone(%this){
    /*
        Called when AI achieves its mission. Then, connection is deleted.
    */
    %this.send("done");
    echo("finish request sent");
    %this.delete();
}
```

Called when AI achieved its goal. Also, use this function when MATLAB server shutdown unexpectedly otherwise you can not use start_socket() function since Torque Engine does not allow re-declaration of socket object.