I Aravind Rajadurai certify that I have carefully read the instructions for the final. Where, Aravind Rajadurai is your full name.

1. a. void glLookAt( GLdouble eyeX,
2. GLdouble eyeY,
3. GLdouble eyeZ,
4. GLdouble centerX,
5. GLdouble centerY,
6. GLdouble centerZ,
7. GLdouble upX,
8. GLdouble upY,
9. GLdouble upZ);
11. glMultMatrixf(M);
12. glTranslated(-eyex, -eyey, -eyez);
13. Vcector3 Eye, At, Up;
14. Vector3 zaxis = At - Eye; zaxis.Normalize();
15. up.Normalize();
16. Vector3 xaxis = Vector3::Cross(Up, zaxis); xaxis.Normalize();
17. Vector3 yaxis = Vector3::Cross(zaxis, xaxis); yaxis.Normalize();
18. float r[16] =
19. {
20. xaxis.x, yaxis.x, - zaxis.x, 0,
21. xaxis.y, yaxis.y, - zaxis.y, 0,
22. xaxis.z, yaxis.z, - zaxis.z, 0,
23. 0, 0, 0, 1,
24. }
25. class Camera
26. {
27. glm::vec3 position;
28. glm::vec3 viewDirection;
29. }
30. Camera::camera():
31. viewDirection(0.0f, 0.0f, 0.0f, up),
32. {
33. }
34. GLfloat ambient[] = {0.3f, 0.3f, 0.3f, 1.0f};
35. GLfloat diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
36. GLfloat specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
37. GLfloat position[] = {-3.0f, -3.4f, -8.8f, 1.0f};
38. glLightfv(GL\_LIGHT0, GL\_POSITION, position);
39. glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambient);
40. glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuse);
41. glLightfv(GL\_LIGHT0, GL\_SPECULAR, specular);
42. GLfloat direction[] = {-3.0f, -3.4f, -8.8f, 0.0f};
43. glLightfv(GL\_LIGHT1, GL\_POSITION, direction);
44. glLightfv(GL\_LIGHT1, GL\_AMBIENT, ambient);
45. glLightfv(GL\_LIGHT1, GL\_DIFFUSE, diffuse);
46. glLightfv(GL\_LIGHT1, GL\_SPECULAR, specular);
47. GLfloat spot\_direction[] = {-3.0f, -3.4f, -8.8f};
48. glLightfv(GL\_LIGHT2, GL\_POSITION, position);
49. glLightfv(GL\_LIGHT2, GL\_SPOT\_DIRECTION, spot\_direction);
50. glLightfv(GL\_LIGHT2, GL\_SPOT\_CUTOFF, 45.0);
51. glLightfv(GL\_LIGHT2, GL\_AMBIENT, ambient);
52. glLightfv(GL\_LIGHT2, GL\_DIFFUSE, diffuse);
53. glLightfv(GL\_LIGHT2, GL\_SPECULAR, specular);
54. mat4 projectionMatrix = glm:: prespective(((float)with()),height(), length());
55. glMultMatrixf(M);
56. glTranslated(-eyex, -eyey, -eyez);
57. glm::mat4 Camera::getWorldToViewMatrix() const
58. {
59. projectionMatrix 8 camera.getWorldToViewMatrix() \*glm::translate(vec4(0.0f, 0.0f, 0.0f, 1.0)))
60. return glm::lookAT(position, postition, position, position + viewDirection);
61. }
62. a. Functional requirement: Login attempt Test case ID: A unique identifier for the test scrip. User sets up and enters a valid username and password that is maximum length. Unique ID for the test scenario. Priority risk level.. whether if is critical, high, medium, or low. Cases: If the user is not already signed up for zoom, then they can sign up with any valid character combination for their username and password. If the user is logged on, then they use one computer to access it and not login in from multiple places. Expected result: success of logging in and accessing the web chat. Status of the test.

After selecting the metrics to use and analyzing the metrics from the results then see the improvements needed. The iid tasks derived from zoom that zoom system is using is with user interphase. They have added features that have improved the interactions like adding reaction emoji’s. Ways that put an individual in a spot light like in a aprivate break out room. To test the duration. The potential users are everyone, big companys, doctors, lawyers, students, and any interaction that requires face to face interaction. The zoom service is free and is constantly bring improved by its users by their support and feedback. The individuals that interact and use this zoom service have an easy learning curve. Zoom meeting users find it easy just to hop on the call because their number one task is to access the system and join the meeting with the appropriate group. In that sense the learnability metric is pretty easy to measure. All we need to gather is the number of users and data for their first-time encounter with the interface and learn from the points from the metric curve. Operability on zoom can be measured through the ability and the operation of the software. Zoom’s operability can be tested on the how well it’s users are able to connect to their meetings, their mean time to recovery, having continuous flow of customers without having failures and loss of existing users. Having a continuous delivery of high level quality of time in the duration the meeting is in progress. Stability can be synthesized in methods and design, having error metrics. The way it can be tested is evaluate on random simulations and error estimation. Boolean functions that efficiently determine error behaviors in a sequential circuits.

2 b. To test learnability for zoom. The test procedures needs to be tested time on task. First use learnability. Organize a large sample size of individuals who haven’t used zoom before. Have them use the product with a check list of things to do. Then test the individuals on the steepness of the learning curve. How quickly do the users get better with repeated use of the design. Address and record the progression of using the product a set number of times. Then test on efficiency of the interaction. How high is an individual’s productivity of the user once they have learned fully how to use the product. The depth of the compliance testing could range from a high level audit to a simple detailed scrutiny.

2 c. To test the pin point compliance testing. Zoom’s operability can be tested on the how well it’s users are able to connect to their meetings, their mean time to recovery, having continuous flow of customers without having failures and loss of existing users. Test procedure can be how well does the program trouble shoot the issue of connection. Having a set required time to hold a meeting, the band width needed, the user and client reached Id. Have a supervised algorithm that monitors the saccades per segment, like the measurement of the mouse click on join button. Then having a threshold number of saccades per segment. After the measurement then comparing the threshold in the pin point analysis group to make an excessive or non excessive decision.

3.a DFS

class Graph

{

int V; // No. of vertices

// Pointer to an array containing

// adjacency lists

list<int> \*adj;

// A recursive function used by DFS

void DFSUtil(int v, bool visited[]);

public:

Graph(int V); // Constructor

// function to add an edge to graph

void addEdge(int v, int w);

// DFS traversal of the vertices

// reachable from v

void DFS(int v);

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::DFSUtil(int v, bool visited[])

{

// Mark the current node as visited and

// print it

visited[v] = true;

cout << v << " ";

// Recur for all the vertices adjacent

// to this vertex

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFSUtil(\*i, visited);

}

// DFS traversal of the vertices reachable from v.

// It uses recursive DFSUtil()

void Graph::DFS(int v)

{

// Mark all the vertices as not visited

bool \*visited = new bool[V];

for (int i = 0; i < V; i++)

visited[i] = false;

// Call the recursive helper function

// to print DFS traversal

DFSUtil(v, visited);

}

BFS

class Graph

{

    int V;    // No. of vertices

    // Pointer to an array containing adjacency

    // lists

    list<int> \*adj;

public:

    Graph(int V);  // Constructor

    // function to add an edge to graph

    void addEdge(int v, int w);

    // prints BFS traversal from a given source s

    void BFS(int s);

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::BFS(int s)

{

    // Mark all the vertices as not visited

    bool \*visited = new bool[V];

    for(int i = 0; i < V; i++)

        visited[i] = false;

    // Create a queue for BFS

    list<int> queue;

    // Mark the current node as visited and enqueue it

    visited[s] = true;

    queue.push\_back(s);

    // 'i' will be used to get all adjacent

    // vertices of a vertex

    list<int>::iterator i;

    while(!queue.empty())

    {

        // Dequeue a vertex from queue and print it

        s = queue.front();

        cout << s << " ";

        queue.pop\_front();

        // Get all adjacent vertices of the dequeued

        // vertex s. If a adjacent has not been visited,

        // then mark it visited and enqueue it

        for (i = adj[s].begin(); i != adj[s].end(); ++i)

        {

            if (!visited[\*i])

            {

                visited[\*i] = true;

                queue.push\_back(\*i);

            }

        }

    }

}