LAPORAN FINAL PROJECT

GRAFIKA KOMPUTER

Oleh :

Putu Mas Anggita Putra	(1708561007)
I Nengah Aryadi Suputra	(1708561014)
Anak Agung Rama Suryananda Widarsa	(1708561024)
I Wayan Gede Indrayasa	(1708561030)

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS UDAYANA

2018

DAFTAR ISI

DAFTAR GAMBAR

BAB I

PENDAHULUAN

Dunia teknologi pada saat ini berkembang dengan sangat pesat. Perkembangan teknologi mencakup semua bidang kehidupan manusia seperti kesehatan, pangan, industri, dll. Begitupun dengan perkembangan teknologi dalam bidang *game*. Perkembangan teknologi dalam bidang *game* saat ini sudah sangat bagus karena game seperti kondisi nyata nya (Pratama, 2014). Pada awal kemunculan *game* pertama kalinya, *game* masih disajikan secara sederhana dan di prakarsai oleh Steven Russel dan proyek yang bernama *Computer Games* pada tahun 1962 dengan produk andalannya bernama Star Wars. Beberapa puluh tahun kemudian , banyak *game* bermunculan dari *game* 2 dimensi dan *game* 3 dimensi. Serta yang bersifat hiburan saja ataupun sebagai media pembelajaran atau edukatif (Pratama, 2014).

Game bertipe 3D merupakan *game* dengan grafis yang baik dalam penggambaran secara realita, akan tetapi *game* 3D meminta spesifikasi komputer yang lumayan tinggi agar tampilan 3 dimensi *game* tersebut ditampilkan secara sempurna. Basis desktop lebih cocok dibandingkan dengan basis web, karena desktop memiliki tampilan yang baik dengan kemampuan *DirectX* yang telah disediakan *Microsoft*. Maka bisa disebutkan bahwa *game* berkembang beriringan dengan teknologi. Beberapa tahun belakangan ini, dalam industri *game* semakin marak munculnya *game* yang semakin menarik dan berkualitas dari segi visualisasi maupun dari segi cerita. Contohnya, game *flappy birds* yang mempunyai konsep sederhana namun dapat memukau dan banyak dimainkan oleh orang. Game *"Flappy Birds"* itu termasuk buatan asing. Namun dengan perkembangan *game* saat ini yang sangat pesat, sangat disayangkan para pengguna *game* khususnya di Indonesia masih sering menggunakan *game* - *game* buatan asing. *Game* yang secara khusus buatan anak bangsa ini masih minim dan belum terpublikasi sepenuhnya sehingga yang sering dijumpai hanya game-game buatan luar negeri. Alangkah bangganya bila orang Indonesia mencintai produk dalam negeri khususnya game buatan anak Indonesia (Pratama, 2014).

Labirin merupakan salah satu permasalahan yang cukup terkenal dalam sejarah kehidupan manusia, labirin pada masa kini lebih sering dinikmati sebagai sebuah permainan pemecahan masalah. Labirin biasanya berawal dari sebuah jalur yang merupakan jalur buntu, dan hanya satu jalur yang merupakan jalan keluar, namun untuk membuat menarik, labirin dapat dibuat berujung

banyak atau memiliki banyak jalan keluar sehingga misi penyelesaian masalah bertambah yang semula hanya satu jalan keluar menjadi banyak, terlebih lagi bila ada beberapa musuh yang menghalangi jalan keluar. Berdasarkan uraian diatas, peneliti tertarik untuk membuat "Pembuatan *Game Labyrinth Maze*". Diharapkan *game* ini dapat menjadi salah satu bentuk usaha memajukan dan ikut berpartisipasi dalam dunia game khususnya di Indonesia. Dalam *game* " *Game Labyrinth Maze*" disajikan dengan visualisasi 3D dan dengan latar belakang tempat kastil kuno yang diharapkan dapat memberikan hiburan yang lebih menarik.

BAB II

METODE PENELITIAN

2.1 Deskripsi Program

Pada Tugas Akhir matakuliah Grafika Komputer ini, kelompok kami akan membuat sebuah *Game Labyrinth Maze* sederhana menggunakan OpenGL dan C++ pada DevC. *Game Labyrinth Maze* adalah permainan yang terdiri dari tembok dengan labirin. Tujuan dari permainan ini adalah mencoba untuk keluar dari labirin dimana pemain tidak boleh melewati tembok.

2.2 Batasan Program

Dalam pembuatan program ini terdapat beberapa batasan pada hasil program ini, yaitu :

- Program masih bersifat endless game (game yang tidak pernah berhenti).

2.3 Library

Dalam pembuatan program ini memerlukan beberapa library tambahan, yaitu :

- Glut
- Glew

2.4 Pencahayaan

Sebetulnya pemodelan cahaya ini bertujuan tidak hanya berkaitan dengan redup atau terangnya sebuah obyek. Tujuannya adalah membuat obyek terlihat lebih realistik seperti yang ada didunia maya dengan memanfaatkan model-model pencahayaan. Salah satu pemodelan cahaya yang biasa dipakai yaitu model cahaya phong. Dalam model cahaya phong dapat digolongkan kedalam tiga kategori :

- Cahaya sekitar / ambient light

  Cahaya sekitar tidak berasal dari arah yang spesifik. Obyek menerima cahaya tidak langsung dari sumber cahaya tetapi berupa hasil pantulan tidak langsung dari sumber cahaya. Karakteristik obyek yang dikenai cahaya sekitar yaitu akan terang di seluruh permukaan di segala arah.

- Cahaya tersebar / diffuse light

Cahaya tersebar adalah hasil interaksi sumber cahaya dengan permukaan yang menyebarkan cahaya karena permukaannya tidak rata atau kasar.

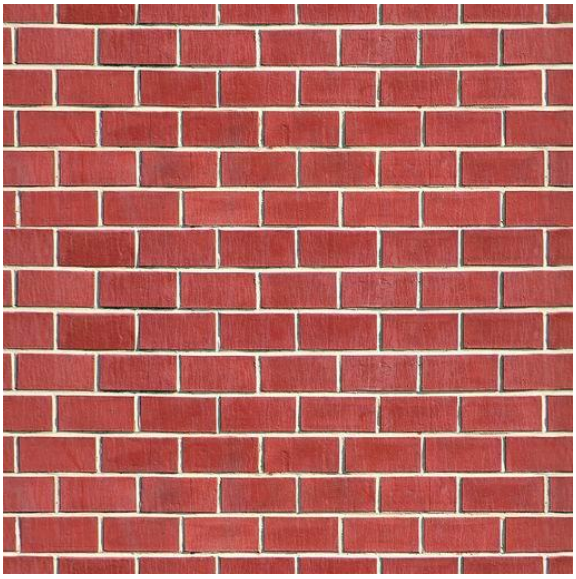
- Cahaya biasa / specular light

Cahaya biasa merupakan hasil dari interaksi sumber cahaya dari arah tertentu terhadap permukaan benda. Permukaan yang terpancar cahaya akan terang dan yang tidak terpancar akan gelap tergantung dari sudut pandang terhadap posisi sumber cahaya dan obyek.

2.5 Texture Yang digunakan

Tekstur biasanya digunakan untuk gambar untuk menghias model 3D, tetapi pada kenyataannya mereka dapat digunakan untuk menyimpan berbagai jenis data. Dimungkinkan untuk memiliki tekstur 1D, 2D dan bahkan 3D, yang dapat digunakan untuk menyimpan data massal pada GPU. Contoh penggunaan lain untuk tekstur adalah menyimpan informasi medan. Artikel ini akan memperhatikan penggunaan tekstur untuk gambar, tetapi prinsip umumnya berlaku untuk semua jenis tekstur.

Tektur yang digunakan :



Gambar 1 Texture

2.6 Pembagian Pekerjaan Personal

Pembagian pekerjaan dalam kelompok kami dibagi menjadi 2 team. Kelompok kami terdiri dari 4 orang, sehingga satu team terdiri dari 2 orang. Team 1 terdiri dari Putu Mas Anggita Putra dan Anak Agung Rama Suryananda Widarsa, sedangkan Team 2 terdiri dari I Nengah Aryadi Suputra dengan I Wayan Gede Indrayasa.

Pembagian pekerjaan :

- Team 1

Dalam team 1 bertugas membuat penetapan nilai konstanta, penetapan tinggi layar, mengatur luas labirin, inisialisasi awal layar, proses loading texture, proses pembatasan tembok dengan kubus, pembuatan latar belakang, dan proses penempatan posisi awal.

- Team 2
  Team 2 bertugas membuat proses pembuatan labirin pada tembok, proses pengecekan apakah labirin dapat diselesaikan atau tidak, kondisi saat menabrak tembok, penentuan pergerakan player, proses pembuatan scene, pengaturan control dengan arrow pad, pengaturan control keluar, dan pengaturan control mouse.

- **Proses including library**

```
#define GLEW_STATIC
#include <glew.h>
#include <windows.h>
#include <GL/glu.h>
#include <glut.h>
#include <stdlib.h>
#include <fstream>
#include <cmath>
#include <ctime>
```

- **Menetapkan nilai konstanta**

```
const GLint CONTROLLER_PLAY=250;
const GLint WINDOW_STARTX=20;
const GLint WINDOW_STARTY=20;
const GLint ESCAPE=27; /* Kode ascii untuk keluar */
const GLint TEXTURE_SIZE=512;
const GLint MAX_APPERROR=64;
const GLint BMP_HEADER_SIZE=54;
const GLint WINDOW_MARGIN=100;
const GLfloat MAZE_EXTREME_LEFT=-5.0f;
const GLfloat MAZE_EXTREME_TOP=-9.0f;
const GLfloat HALF_CUBE=1.25f;
const GLfloat FULL_CUBE=HALF_CUBE+HALF_CUBE;
const GLfloat START_X_AT=-10.0f;
const GLfloat START_Y_AT=0.0f;
const GLfloat START_ROT=270.0f;
const GLfloat START_CAMERA_Y=5.0f;
const GLfloat CAMERA_SINK=0.05f;
const GLfloat VIEW_FIELD=45.0f;
const GLfloat NEAR_Z=0.1f;
const GLfloat FAR_Z=1000.0f;
const GLfloat SKY_DISTANCE=250.0f;
const GLfloat LEFTMOST_CUBE_CENTER=MAZE_EXTREME_LEFT+HALF_CUBE;
const GLfloat COLLIDE_MARGIN=0.15625;
const GLfloat ROTATE_MOUSE_SENSE=0.00004f;
const GLfloat ROTATE_KEY_SENSE=0.08f;
const GLfloat WALK_MOUSE_SENSE=0.00019f;
const GLfloat WALK_KEY_SENSE=0.19;
const GLfloat WALK_MOUSE_REVERSE_SENSE=0.00008f;
```

```
const GLfloat WALK_KEY_REVERSE_SENSE=0.08f;
const GLfloat BOUNCEBACK=5.0f;
const GLfloat SKY_SCALE=6.0f;

const GLint XSIZE=8;
const GLint YSIZE=8;
const GLint DIFFICULTY=2;

const GLint OBFUSCATION_LOOP_RUNS=(XSIZE * YSIZE * 20);

const GLint SOLUTION_PATH=2;
const GLint FALSE_PATH=1;
const GLint NO_PATH=0;

const GLint EAST=0;
const GLint SOUTH=1;
const GLint WEST=2;
const GLint NORTH=3;

static GLfloat x_at=START_X_AT;
static GLfloat y_at=START_Y_AT;
static GLfloat rot=START_ROT;
static GLint xin=0,yin=0;
static GLfloat camera_y=START_CAMERA_Y;
```

- **Penetapan tinggi layar**

```
GLint windowheight()
{
 static int ret=0;
 if(!ret)ret=glutGet(GLUT_SCREEN_HEIGHT)-WINDOW_MARGIN;
 return ret;
}
```

- **Mengatur luas laibirin**

```
GLint ((*(maze_innards()))[YSIZE]){
 static int whole_maze[XSIZE+2][YSIZE+2]={NO_PATH};
 return (int(*)[YSIZE])(&whole_maze[0][1]);
}
```

- **Inisialisasi awal layar**

```
void initgl(GLint width, GLint height)
{
 glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
 glClearDepth(1.0);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_CULL_FACE);
 glFrontFace(GL_CCW);
 glShadeModel(GL_SMOOTH);
```

```
 glMatrixMode(GL_PROJECTION);

gluPerspective(VIEW_FIELD,(GLfloat)width/(GLfloat)height,NEAR_Z,FAR_Z)
;
 glMatrixMode(GL_MODELVIEW);
}
```

```
void resizer(GLint width, GLint height)
{
 if(width!=windowwidth() || height!=windowheight()) exit(0);
}
```

- **Proses loading texture**

```
GLuint maketex(const char* tfile,GLint xSize,GLint ySize)
{
 GLuint rmesh;
 FILE * file;
 unsigned char * texdata = (unsigned char*) malloc( xSize * ySize * 3 );


 file = fopen(tfile, "rb" );
 fseek(file,BMP_HEADER_SIZE,SEEK_CUR);
 fread( texdata, xSize * ySize * 3, 1, file );
 fclose( file );
 glEnable( GL_TEXTURE_2D );


 char* colorbits = new char[ xSize * ySize * 3];


 for(GLint a=0; a<xSize * ySize * 3; ++a) colorbits[a]=0xFF;


 glGenTextures(1,&rmesh);
 glBindTexture(GL_TEXTURE_2D,rmesh);


 glTexImage2D(GL_TEXTURE_2D,0 ,3 , xSize,
 ySize, 0 , GL_RGB, GL_UNSIGNED_BYTE, colorbits);
```

```
app_assert_success("post0_image");

glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);

app_assert_success("pre_getview");


GLint viewport[4];
glGetIntegerv(GL_VIEWPORT,(GLint*)viewport);

app_assert_success("pre_view");
glViewport(0,0,xSize,ySize);
app_assert_success("post0_view");

glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

glPushMatrix();
glLoadIdentity();

app_assert_success("ogl_mvx");

glDrawPixels(xSize,ySize,GL_BGR, GL_UNSIGNED_BYTE,texdata);

app_assert_success("pre_copytext");
glPopMatrix();
app_assert_success("copytext2");
glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
0,0, xSize, ySize, 0);
```

```
app_assert_success("post_copy");


glViewport(viewport[0],viewport[1],viewport[2],viewport[3]); //{X,Y,Width,Height}
app_assert_success("ogl_mm1");
delete[] colorbits;
free(texdata);
return rmesh;
}
```

- **Proses pembatan tembok dengan menggunakan kubus**

```
void cube(GLfloat x, GLfloat y, GLfloat z) //Pembuatan Tembok
{
 //Kubus atas
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE, z+HALF_CUBE);

 // Bawah
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);

 // Depan
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE, z+HALF_CUBE);

 // Belakang
 glTexCoord2d(1.0,1.0);
 glVertex3f(x-HALF_CUBE,HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,1.0);
```

```
 glVertex3f(x+HALF_CUBE,HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);

 // Kiri
 glTexCoord2d(1.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);

 // Kanan
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
}
```

- **Pembuatan latar belakang**

```
void sky(GLuint haze)
{
 glBindTexture(GL_TEXTURE_2D,haze);
 glBegin(GL_QUADS);
 glTexCoord2d(1.0,1.0);
 glVertex3f(   (windowwidth()/SKY_SCALE),   (windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(  -(windowwidth()/SKY_SCALE),   (windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(  -(windowwidth()/SKY_SCALE),  -(windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(   (windowwidth()/SKY_SCALE),  -(windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glEnd();
}
```

- **Proses penempatan posisi awal**

```
void make_solution()
```

```
{

 int path_leg_length=3;

 int x=0,y=0;
 int d=EAST;

 bool facing_east_west=true;

 y=rand()%YSIZE;


 while(x<XSIZE)
 {

  while(path_leg_length-- && x<(XSIZE))
  {
   switch(d)
   {
   case EAST:
    (maze_innards())[x++][y]=SOLUTION_PATH;
    break;

   case SOUTH:
    (maze_innards())[x][y++]=SOLUTION_PATH;
    break;

   case WEST:
    (maze_innards())[x--][y]=SOLUTION_PATH;
    break;

   case NORTH:
    (maze_innards())[x][y--]=SOLUTION_PATH;
    break;
   }
  }

  int tempx,tempy;

  do
  {
   tempx=x;
   tempy=y;
   if(facing_east_west)
   {
    d=(rand()%2)?NORTH:SOUTH;
   }else{
    d=EAST;
   }

   if(XSIZE-x<3)
```

```
   {
    d=EAST;
    path_leg_length=XSIZE-x;
   }

   if(facing_east_west)
   {
    path_leg_length=((rand()%(XSIZE/DIFFICULTY)+2));
   }else{
    path_leg_length=((rand()%(YSIZE/DIFFICULTY)+2));
   }

   switch(d)
   {
   case EAST:
    tempx+=path_leg_length;
    break;
   case SOUTH:
    tempy+=path_leg_length;
    break;
   case WEST:
    tempx-=path_leg_length;
    break;
   case NORTH:
    tempy-=path_leg_length;
    break;
   }
  }while(tempx<0||tempy<0||tempy>=YSIZE);


  facing_east_west=!facing_east_west;
 }
}
```

- **Proses pengecekan apakah labirin dapat diselesaikan atau tidak**

```
bool valid_for_obfuscation(int x, int y)
{

 if(x<=0) return false;
 if(y<0) return false;
 if(x>=XSIZE-1) return false;
 if(y>=YSIZE) return false;

 if((maze_innards())[x][y]) return false;

 int ret=0;

 if((maze_innards())[x+1][y]) ++ret;
 if(x-1>=0 && (maze_innards())[x-1][y]) ++ret;
 if(y+1<YSIZE && (maze_innards())[x][y+1]) ++ret;
```

```
 if(y-1>=0 && (maze_innards())[x][y-1]) ++ret;

 if (ret==1)return true;
 else return false;
}

void obfuscate_maze()
{
 int x,y;
 int c=0;

 for(int ob=0; ob < OBFUSCATION_LOOP_RUNS; ++ob)
 {
  x=rand()%XSIZE;
  y=rand()%YSIZE;

  if(valid_for_obfuscation(x,y))
  {
   c++;
   (maze_innards())[x][y]=FALSE_PATH;
  }
 }
}
```

- **Proses pembuatan labirin pada dengan tembok**

```
void print_maze()
{
 int x,y;
 for(x=0; x<XSIZE ; ++x )
 {
  cube(MAZE_EXTREME_LEFT+HALF_CUBE+((GLfloat)x*FULL_CUBE),
  0.0,
  MAZE_EXTREME_TOP+HALF_CUBE);

  cube(MAZE_EXTREME_LEFT+HALF_CUBE+((GLfloat)x*FULL_CUBE),
  0.0,
  MAZE_EXTREME_TOP+HALF_CUBE+FULL_CUBE+(YSIZE*(FULL_CUBE)) );
 }
 for(y=0; y<YSIZE ; ++y )
 {
  for(x=0; x<XSIZE ; ++x )
  {
   if((maze_innards())[x][y]==NO_PATH)
   {
    cube(LEFTMOST_CUBE_CENTER+((GLfloat)x*FULL_CUBE),
    0.0,
    MAZE_EXTREME_TOP+HALF_CUBE+FULL_CUBE+((GLfloat)y*FULL_CUBE));
   }
  }
 }
```

```
}
```

- **Kondisi saat menabrak tembok**

```
bool collide()
{
 int x,y;

 if(x_at>=MAZE_EXTREME_LEFT-COLLIDE_MARGIN &&
   x_at<=MAZE_EXTREME_LEFT+XSIZE*FULL_CUBE+COLLIDE_MARGIN)
 {
  if( y_at<=(MAZE_EXTREME_TOP+FULL_CUBE)+COLLIDE_MARGIN &&
    y_at>=MAZE_EXTREME_TOP-COLLIDE_MARGIN)
  {
   return 1;
  }


if(y_at<=(MAZE_EXTREME_TOP+FULL_CUBE)+FULL_CUBE+(YSIZE*FULL_CUBE)+COLL
IDE_MARGIN &&
     y_at>=               MAZE_EXTREME_TOP+FULL_CUBE+(YSIZE*FULL_CUBE)-
COLLIDE_MARGIN)
  {
   return 1;
  }
 }

 for(y=0; y<YSIZE ; ++y )
 {
  for(x=0; x<XSIZE ; ++x )
  {
   if((maze_innards())[x][y]==NO_PATH)
   {
    if( x_at>=MAZE_EXTREME_LEFT+x*FULL_CUBE-COLLIDE_MARGIN &&
      x_at<=MAZE_EXTREME_LEFT+FULL_CUBE+x*FULL_CUBE+COLLIDE_MARGIN &&
      y_at>=MAZE_EXTREME_TOP+(y+1)*FULL_CUBE-COLLIDE_MARGIN &&
      y_at<=MAZE_EXTREME_TOP+(y+2)*FULL_CUBE+COLLIDE_MARGIN )
    {
     return 1;
    }
   }
  }
 }
 return 0;
}
```

- **Penentuan pergerakan player**

```
void move(GLfloat amt)
{
  x_at+=cos(rot)*amt;
```

```
  y_at+=sin(rot)*amt;
  if(collide())
  {
   x_at-=BOUNCEBACK*cos(rot)*amt;
   y_at-=BOUNCEBACK*sin(rot)*amt;
  }
  if(collide())
  {
   x_at+=BOUNCEBACK*cos(rot)*amt;
   y_at+=BOUNCEBACK*sin(rot)*amt;
   x_at-=cos(rot)*amt;
   y_at-=sin(rot)*amt;
  }
 }
```

- **Proses pembuatan scene**

```
void drawscene()
{
 static bool init=0;
 static GLuint mesh;
 static GLuint haze;

 if(!init)
 {
  init=1;
  mesh=maketex(TEXTURE_FILE,TEXTURE_SIZE,TEXTURE_SIZE);
  haze=maketex(SKY_FILE,TEXTURE_SIZE,TEXTURE_SIZE);
 }

 if(camera_y<=0.0f && xin && yin)
 {
  if(yin<CONTROLLER_PLAY)
   move((yin-windowheight()/2.0f)*-WALK_MOUSE_SENSE);
  if(yin>(windowheight()-CONTROLLER_PLAY))
   move(((windowheight()/2.0f)-yin)*WALK_MOUSE_REVERSE_SENSE);
  if(xin<CONTROLLER_PLAY || xin>(windowwidth()-CONTROLLER_PLAY))
   rot+=(xin-(windowwidth()/2.0f))*ROTATE_MOUSE_SENSE;
 }

 glLoadIdentity();
 glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
 sky(haze);
 glBindTexture(GL_TEXTURE_2D,mesh);

gluLookAt(x_at,camera_y,y_at,x_at+cos(rot),camera_y,y_at+sin(rot),0.0,
1.0,0.0);
 if(camera_y>0.0) camera_y-=CAMERA_SINK;
 glBegin(GL_QUADS);
 print_maze();
 glEnd();
```

```
 glutSwapBuffers();
}
```

- **Pengaturan kontrol dengan arrow pad**

```
void arrows(GLint key, GLint x, GLint y)
{
 if(key == GLUT_KEY_UP)
  move(WALK_KEY_SENSE);
 if(key == GLUT_KEY_DOWN)
  move(-WALK_KEY_REVERSE_SENSE);

 if(camera_y<=0.0f && xin && yin)
 {
  if(key == GLUT_KEY_RIGHT)
   rot+=ROTATE_KEY_SENSE;
  if(key == GLUT_KEY_LEFT)
   rot-=ROTATE_KEY_SENSE;
  }
}
```

- **Pengaturan kontrol keluar**

```
void keypress(unsigned char key, GLint x, GLint y)
{
 if(key==ESCAPE)exit(0);
}
```

- **Pengaturan kontrol mouse**

```
void mouse(int x, int y)
{
 static int mouses=0;
 if(mouses<=1)
 {
  ++mouses;
  xin=0; yin=0;
  return;
 }
 xin=x; yin=y;
}
```

- **Main program**

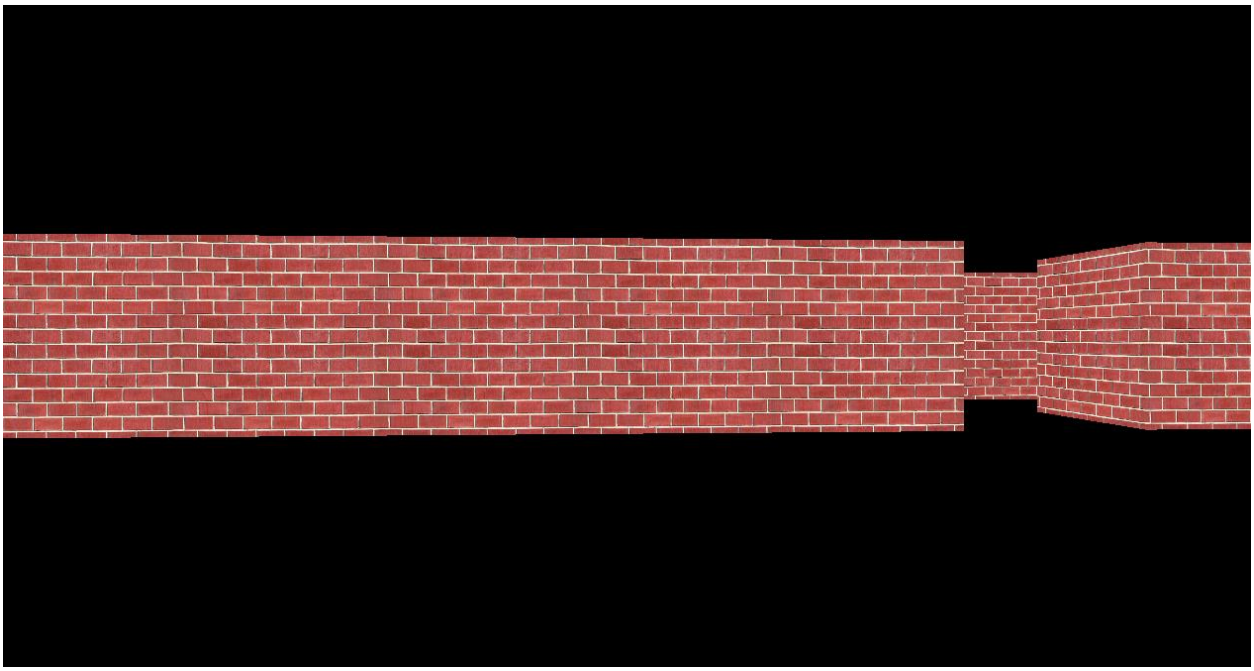
```
int main(int argc, char **argv)
{
 GLuint window;

 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
 glDisable(GLUT_ALPHA);
```

```
 glutInitWindowSize(windowwidth(),windowheight());
 glutInitWindowPosition(WINDOW_STARTX, WINDOW_STARTY);

 window = glutCreateWindow("openmaze");

 glutDisplayFunc(&drawscene);
 glutIdleFunc(&drawscene);
 glutReshapeFunc(&resizer);
 glutSpecialFunc(&arrows);
 glutKeyboardFunc(&keypress);
 glutPassiveMotionFunc(&mouse);
 initgl(windowwidth(),windowheight());
 glewInit();

 srand(time(0));

 make_solution();
 obfuscate_maze();

 glutMainLoop();

 return 0;
}
```

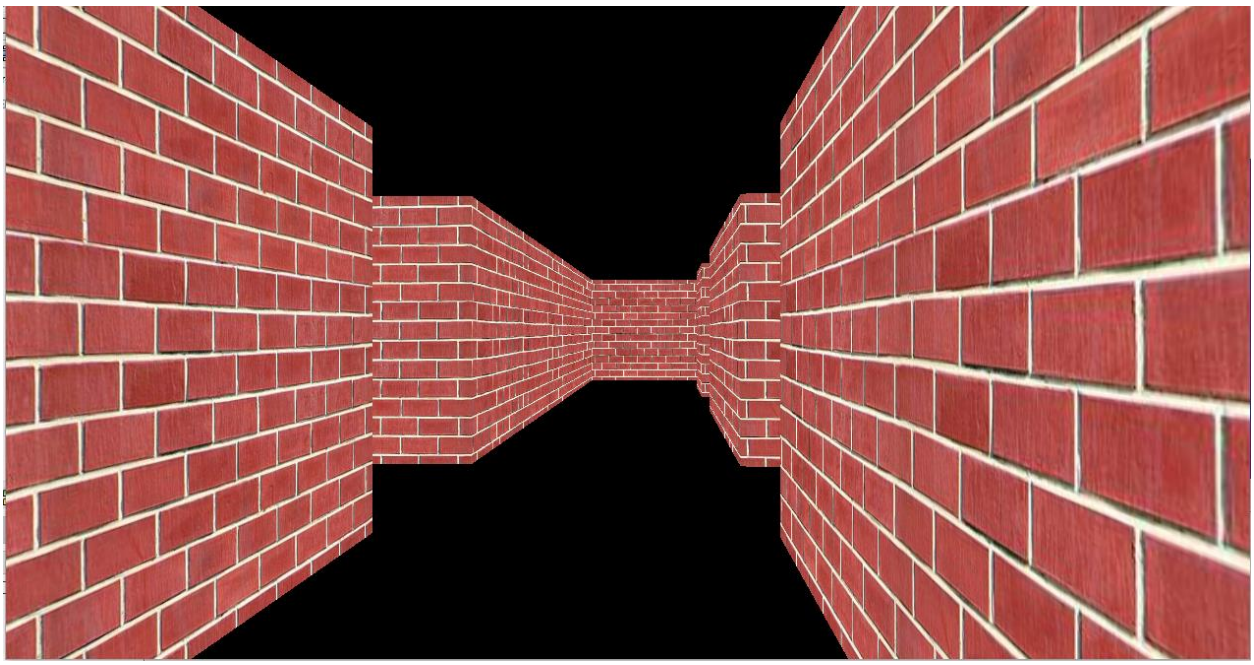
BAB III

HASIL DAN PEMBAHASAN

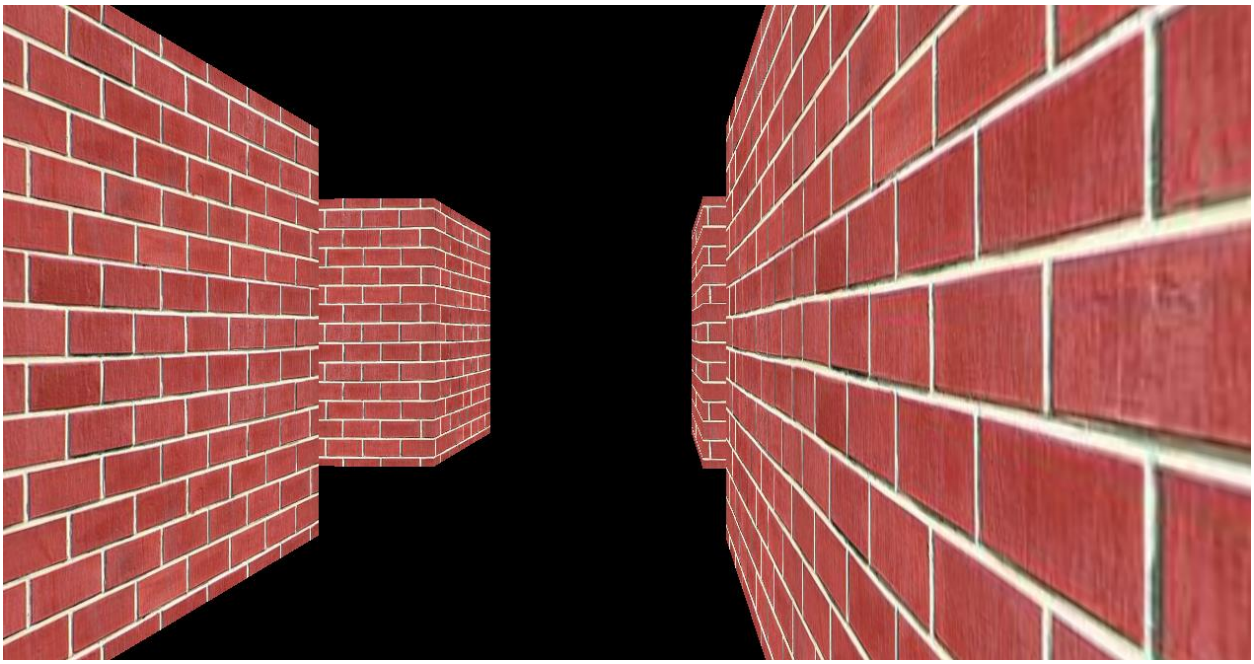
3.1 Screenshoot Output Program



Gambar 2 Screenshoot Output 1



Gambar 3 Screenshoot Output 2



Gambar 4 Screenshoot Output 3

BAB IV

LAMPIRAN SCRIPT CODE

4.1 Lampiran Script Code

```
#define GLEW_STATIC
#include <glew.h>
#include <windows.h>
#include <GL/glu.h>
#include <glut.h>
#include <stdlib.h>
#include <fstream>
#include <cmath>
#include <ctime>

//Constants

const GLint CONTROLLER_PLAY=250;
const GLint WINDOW_STARTX=20;
const GLint WINDOW_STARTY=20;
const GLint ESCAPE=27; /* Kode ascii untuk keluar */
const GLint TEXTURE_SIZE=512;
const GLint MAX_APPERROR=64;
const GLint BMP_HEADER_SIZE=54;
const GLint WINDOW_MARGIN=100;
const GLfloat MAZE_EXTREME_LEFT=-5.0f;
const GLfloat MAZE_EXTREME_TOP=-9.0f;
const GLfloat HALF_CUBE=1.25f;
const GLfloat FULL_CUBE=HALF_CUBE+HALF_CUBE;
const GLfloat START_X_AT=-10.0f;
const GLfloat START_Y_AT=0.0f;
const GLfloat START_ROT=270.0f;
const GLfloat START_CAMERA_Y=5.0f;
const GLfloat CAMERA_SINK=0.05f;
const GLfloat VIEW_FIELD=45.0f;
const GLfloat NEAR_Z=0.1f;
const GLfloat FAR_Z=1000.0f;
const GLfloat SKY_DISTANCE=250.0f;
const GLfloat LEFTMOST_CUBE_CENTER=MAZE_EXTREME_LEFT+HALF_CUBE;
const GLfloat COLLIDE_MARGIN=0.15625;
const GLfloat ROTATE_MOUSE_SENSE=0.00004f;
const GLfloat ROTATE_KEY_SENSE=0.08f;
const GLfloat WALK_MOUSE_SENSE=0.00019f;
const GLfloat WALK_KEY_SENSE=0.19;
const GLfloat WALK_MOUSE_REVERSE_SENSE=0.00008f;
const GLfloat WALK_KEY_REVERSE_SENSE=0.08f;
const GLfloat BOUNCEBACK=5.0f;
const GLfloat SKY_SCALE=6.0f;

#define TEXTURE_FILE "wall2.bmp"
#define SKY_FILE "sky.bmp"

const GLint XSIZE=8;
const GLint YSIZE=8;
const GLint DIFFICULTY=2;
```

```
const GLint OBFUSCATION_LOOP_RUNS=(XSIZE * YSIZE * 20);

const GLint SOLUTION_PATH=2;
const GLint FALSE_PATH=1;
const GLint NO_PATH=0;

const GLint EAST=0;
const GLint SOUTH=1;
const GLint WEST=2;
const GLint NORTH=3;

static GLfloat x_at=START_X_AT;
static GLfloat y_at=START_Y_AT;
static GLfloat rot=START_ROT;
static GLint xin=0,yin=0;
static GLfloat camera_y=START_CAMERA_Y;

// Functions

GLint windowwidth()
{
 static int ret=0;
 if(!ret)ret=glutGet(GLUT_SCREEN_WIDTH)-WINDOW_MARGIN;
 return ret;
}

GLint windowheight()
{
 static int ret=0;
 if(!ret)ret=glutGet(GLUT_SCREEN_HEIGHT)-WINDOW_MARGIN;
 return ret;
}

GLint ((*(maze_innards()))[YSIZE]){
 static int whole_maze[XSIZE+2][YSIZE+2]={NO_PATH};
 return (int(*)[YSIZE])(&whole_maze[0][1]);
}

void initgl(GLint width, GLint height)
{
 glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
 glClearDepth(1.0);
 glEnable(GL_DEPTH_TEST);
 glEnable(GL_CULL_FACE);
 glFrontFace(GL_CCW);
 glShadeModel(GL_SMOOTH);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(VIEW_FIELD,(GLfloat)width/(GLfloat)height,NEAR_Z,FAR_Z);
 glMatrixMode(GL_MODELVIEW);
}

void resizer(GLint width, GLint height)
{
 if(width!=windowwidth() || height!=windowheight()) exit(0);
}
```

```
void app_assert_success(const char* szz)
{
 if(GLint xerr= glGetError())
 {
  char szerr[MAX_APPERROR];
  sprintf(szerr,"%s , %d",szz,xerr);
  fprintf(stderr,"%s",szerr);
  exit(1);
 }
}

GLuint maketex(const char* tfile,GLint xSize,GLint ySize)
{
 GLuint rmesh;
 FILE * file;
 unsigned char * texdata = (unsigned char*) malloc( xSize * ySize * 3 );

 file = fopen(tfile, "rb" );
 fseek(file,BMP_HEADER_SIZE,SEEK_CUR);
 fread( texdata, xSize * ySize * 3, 1, file );
 fclose( file );
 glEnable( GL_TEXTURE_2D );

 char* colorbits = new char[ xSize * ySize * 3];

 for(GLint a=0; a<xSize * ySize * 3; ++a) colorbits[a]=0xFF;

 glGenTextures(1,&rmesh);
 glBindTexture(GL_TEXTURE_2D,rmesh);

 glTexImage2D(GL_TEXTURE_2D,0 ,3 , xSize,
 ySize, 0 , GL_RGB, GL_UNSIGNED_BYTE, colorbits);

 app_assert_success("post0_image");

 glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER, GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER, GL_LINEAR);

 app_assert_success("pre_getview");


 GLint viewport[4];
 glGetIntegerv(GL_VIEWPORT,(GLint*)viewport);

 app_assert_success("pre_view");
 glViewport(0,0,xSize,ySize);
 app_assert_success("post0_view");

 glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

 glPushMatrix();
 glLoadIdentity();

 app_assert_success("ogl_mvx");

 glDrawPixels(xSize,ySize,GL_BGR, GL_UNSIGNED_BYTE,texdata);
```

```
 app_assert_success("pre_copytext");
 glPopMatrix();
 app_assert_success("copytext2");
 glCopyTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA,
 0,0, xSize, ySize, 0);
 app_assert_success("post_copy");

 glViewport(viewport[0],viewport[1],viewport[2],viewport[3]);
//{X,Y,Width,Height}
 app_assert_success("ogl_mm1");
 delete[] colorbits;
 free(texdata);
 return rmesh;
}

void cube(GLfloat x, GLfloat y, GLfloat z) //Pembuatan Tembok
{
 //Kubus atas
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE, z+HALF_CUBE);

 // Bawah
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);

 // Depan
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE, z+HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE, z+HALF_CUBE);

 // Belakang
 glTexCoord2d(1.0,1.0);
 glVertex3f(x-HALF_CUBE,HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x+HALF_CUBE,HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
```

```
 // Kiri
 glTexCoord2d(1.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x-HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x-HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);

 // Kanan
 glTexCoord2d(1.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE,z-HALF_CUBE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(x+HALF_CUBE, HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z+HALF_CUBE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(x+HALF_CUBE,-HALF_CUBE,z-HALF_CUBE);
}

void sky(GLuint haze)
{
 glBindTexture(GL_TEXTURE_2D,haze);
 glBegin(GL_QUADS);
 glTexCoord2d(1.0,1.0);
 glVertex3f(    (windowwidth()/SKY_SCALE),    (windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glTexCoord2d(0.0,1.0);
 glVertex3f(   -(windowwidth()/SKY_SCALE),    (windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glTexCoord2d(0.0,0.0);
 glVertex3f(   -(windowwidth()/SKY_SCALE),   -(windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glTexCoord2d(1.0,0.0);
 glVertex3f(    (windowwidth()/SKY_SCALE),   -(windowheight()/SKY_SCALE),-
SKY_DISTANCE);
 glEnd();
}

void make_solution()
{

 int path_leg_length=3;

 int x=0,y=0;
 int d=EAST;

 bool facing_east_west=true;

 y=rand()%YSIZE;


 while(x<XSIZE)
 {

  while(path_leg_length-- && x<(XSIZE))
```

```
{
 switch(d)
 {
 case EAST:
  (maze_innards())[x++][y]=SOLUTION_PATH;
  break;

 case SOUTH:
  (maze_innards())[x][y++]=SOLUTION_PATH;
  break;

 case WEST:
  (maze_innards())[x--][y]=SOLUTION_PATH;
  break;

 case NORTH:
  (maze_innards())[x][y--]=SOLUTION_PATH;
  break;
 }
}

int tempx,tempy;

do
{
 tempx=x;
 tempy=y;
 if(facing_east_west)
 {
  d=(rand()%2)?NORTH:SOUTH;
 }else{
  d=EAST;
 }

 if(XSIZE-x<3)
 {
  d=EAST;
  path_leg_length=XSIZE-x;
 }

 if(facing_east_west)
 {
  path_leg_length=((rand()%(XSIZE/DIFFICULTY)+2));
 }else{
  path_leg_length=((rand()%(YSIZE/DIFFICULTY)+2));
 }

 switch(d)
 {
 case EAST:
  tempx+=path_leg_length;
  break;
 case SOUTH:
  tempy+=path_leg_length;
  break;
 case WEST:
  tempx-=path_leg_length;
```

```
    break;
   case NORTH:
    tempy-=path_leg_length;
    break;
   }
  }while(tempx<0||tempy<0||tempy>=YSIZE);


  facing_east_west=!facing_east_west;
 }
}

bool valid_for_obfuscation(int x, int y)
{

 if(x<=0) return false;
 if(y<0) return false;
 if(x>=XSIZE-1) return false;
 if(y>=YSIZE) return false;

 if((maze_innards())[x][y]) return false;

 int ret=0;

 if((maze_innards())[x+1][y]) ++ret;
 if(x-1>=0 && (maze_innards())[x-1][y]) ++ret;
 if(y+1<YSIZE && (maze_innards())[x][y+1]) ++ret;
 if(y-1>=0 && (maze_innards())[x][y-1]) ++ret;

 if (ret==1)return true;
 else return false;
}

void obfuscate_maze()
{
 int x,y;
 int c=0;

 for(int ob=0; ob < OBFUSCATION_LOOP_RUNS; ++ob)
 {
  x=rand()%XSIZE;
  y=rand()%YSIZE;

  if(valid_for_obfuscation(x,y))
  {
   c++;
   (maze_innards())[x][y]=FALSE_PATH;
  }
 }
}

void print_maze()
{
 int x,y;
 for(x=0; x<XSIZE ; ++x )
 {
  cube(MAZE_EXTREME_LEFT+HALF_CUBE+((GLfloat)x*FULL_CUBE),
```

```
   0.0,
   MAZE_EXTREME_TOP+HALF_CUBE);

   cube(MAZE_EXTREME_LEFT+HALF_CUBE+((GLfloat)x*FULL_CUBE),
   0.0,
   MAZE_EXTREME_TOP+HALF_CUBE+FULL_CUBE+(YSIZE*(FULL_CUBE)) );
  }
  for(y=0; y<YSIZE ; ++y )
  {
   for(x=0; x<XSIZE ; ++x )
   {
    if((maze_innards())[x][y]==NO_PATH)
    {
     cube(LEFTMOST_CUBE_CENTER+((GLfloat)x*FULL_CUBE),
     0.0,
     MAZE_EXTREME_TOP+HALF_CUBE+FULL_CUBE+((GLfloat)y*FULL_CUBE));
    }
   }
  }
}

bool collide()
{
 int x,y;

 if(x_at>=MAZE_EXTREME_LEFT-COLLIDE_MARGIN &&
   x_at<=MAZE_EXTREME_LEFT+XSIZE*FULL_CUBE+COLLIDE_MARGIN)
 {
  if( y_at<=(MAZE_EXTREME_TOP+FULL_CUBE)+COLLIDE_MARGIN &&
    y_at>=MAZE_EXTREME_TOP-COLLIDE_MARGIN)
  {
   return 1;
  }


if(y_at<=(MAZE_EXTREME_TOP+FULL_CUBE)+FULL_CUBE+(YSIZE*FULL_CUBE)+COLLIDE
_MARGIN &&
     y_at>= MAZE_EXTREME_TOP+FULL_CUBE+(YSIZE*FULL_CUBE)-COLLIDE_MARGIN)
  {
   return 1;
  }
 }

 for(y=0; y<YSIZE ; ++y )
 {
  for(x=0; x<XSIZE ; ++x )
  {
   if((maze_innards())[x][y]==NO_PATH)
   {
    if( x_at>=MAZE_EXTREME_LEFT+x*FULL_CUBE-COLLIDE_MARGIN &&
      x_at<=MAZE_EXTREME_LEFT+FULL_CUBE+x*FULL_CUBE+COLLIDE_MARGIN &&
      y_at>=MAZE_EXTREME_TOP+(y+1)*FULL_CUBE-COLLIDE_MARGIN &&
      y_at<=MAZE_EXTREME_TOP+(y+2)*FULL_CUBE+COLLIDE_MARGIN )
    {
     return 1;
    }
   }
```

```
   }
  }
  return 0;
}

void move(GLfloat amt)
{
   x_at+=cos(rot)*amt;
   y_at+=sin(rot)*amt;
   if(collide())
   {
    x_at-=BOUNCEBACK*cos(rot)*amt;
    y_at-=BOUNCEBACK*sin(rot)*amt;
   }
   if(collide())
   {
    x_at+=BOUNCEBACK*cos(rot)*amt;
    y_at+=BOUNCEBACK*sin(rot)*amt;
    x_at-=cos(rot)*amt;
    y_at-=sin(rot)*amt;
   }
  }

void drawscene()
{
 static bool init=0;
 static GLuint mesh;
 static GLuint haze;

 if(!init)
 {
  init=1;
  mesh=maketex(TEXTURE_FILE,TEXTURE_SIZE,TEXTURE_SIZE);
  haze=maketex(SKY_FILE,TEXTURE_SIZE,TEXTURE_SIZE);
 }

 if(camera_y<=0.0f && xin && yin)
 {
  if(yin<CONTROLLER_PLAY)
   move((yin-windowheight()/2.0f)*-WALK_MOUSE_SENSE);
  if(yin>(windowheight()-CONTROLLER_PLAY))
   move(((windowheight()/2.0f)-yin)*WALK_MOUSE_REVERSE_SENSE);
  if(xin<CONTROLLER_PLAY || xin>(windowwidth()-CONTROLLER_PLAY))
   rot+=(xin-(windowwidth()/2.0f))*ROTATE_MOUSE_SENSE;
 }

 glLoadIdentity();
 glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
 sky(haze);
 glBindTexture(GL_TEXTURE_2D,mesh);

gluLookAt(x_at,camera_y,y_at,x_at+cos(rot),camera_y,y_at+sin(rot),0.0,1.0
,0.0);
 if(camera_y>0.0) camera_y-=CAMERA_SINK;
 glBegin(GL_QUADS);
 print_maze();
 glEnd();
```

```
 glutSwapBuffers();
}

void arrows(GLint key, GLint x, GLint y)
{
 if(key == GLUT_KEY_UP)
  move(WALK_KEY_SENSE);
 if(key == GLUT_KEY_DOWN)
  move(-WALK_KEY_REVERSE_SENSE);

 if(camera_y<=0.0f && xin && yin)
 {
  if(key == GLUT_KEY_RIGHT)
   rot+=ROTATE_KEY_SENSE;
  if(key == GLUT_KEY_LEFT)
   rot-=ROTATE_KEY_SENSE;
  }
}

void keypress(unsigned char key, GLint x, GLint y)
{
 if(key==ESCAPE)exit(0);
}

void mouse(int x, int y)
{
 static int mouses=0;
 if(mouses<=1)
 {
  ++mouses;
  xin=0; yin=0;
  return;
 }
 xin=x; yin=y;
}

int main(int argc, char **argv)
{
 GLuint window;

 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
 glDisable(GLUT_ALPHA);
 glutInitWindowSize(windowwidth(),windowheight());
 glutInitWindowPosition(WINDOW_STARTX, WINDOW_STARTY);

 window = glutCreateWindow("openmaze");

 glutDisplayFunc(&drawscene);
 glutIdleFunc(&drawscene);
 glutReshapeFunc(&resizer);
 glutSpecialFunc(&arrows);
 glutKeyboardFunc(&keypress);
 glutPassiveMotionFunc(&mouse);
 initgl(windowwidth(),windowheight());
 glewInit();
```

```
 srand(time(0));

 make_solution();
 obfuscate_maze();

 glutMainLoop();

 return 0;
}
```

BAB V

PENUTUP

5.1 Kesimpulan

OpenGL adalah sebuah program aplikasi interface yang digunakan untuk mendefinisikan komputer grafis 2D dan 3D. Dengan tambahan beberapa library, OpenGL dapat membentuk sebuah program dengan interface dan fungsi yang beragam. Library yang dapat digunakan antara lain Glut dan Glew. Glut merupakan pengembangan dari OpenGL yang didesain untuk aplikasi dengan level kecil hingga menengah dan menggunkan callback function untuk menambahkan interaksi dari user. Sedangkan Glew atau OpenGL Wrangler Library Extension (GLEW) adalah library C / C ++ lintas platform yang membantu dalam meminta dan memuat ekstensi OpenGL.