#include<ctype.h>

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

#define M\_PI 3.14159265358979323846264338327950288419716939937510

#define false 0

#define true 1

const int BOARD\_X = 31;

const int BOARD\_Y = 28;

int board\_array[BOARD\_X][BOARD\_Y] =

{ {8,5,5,5,5,5,5,5,5,5,5,5,5,1,1,5,5,5,5,5,5,5,5,5,5,5,5,7},

{6,0,0,0,0,0,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,0,0,0,0,0,0,6},

{6,0,8,1,1,7,0,8,1,1,1,7,0,2,4,0,8,1,1,1,7,0,8,1,1,7,0,6},

{6,0,2,11,11,4,0,2,11,11,11,4,0,2,4,0,2,11,11,11,4,0,2,11,11,4,0,6},

{6,0,9,3,3,10, 0,9,3,3,3,10,0,9,10,0,9,3,3,3,10,0,9,3,3,10,0,6},

{6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6},

{6,0,8,1,1,7,0,8,7,0,8,1,1,1,1,1,1,7,0,8,7,0,8,1,1,7,0,6},

{6,0,9,3,3,10,0,2,4,0,9,3,3,11,11,3,3,10,0,2,4,0,9,3,3,10,0,6},

{6,0,0,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,0,0,6},

{9,5,5,5,5,7,0,2,11,1,1,7,0,2,4,0,8,1,1,11,4,0,8,5,5,5,5,10},

{0,0,0,0,0,6,0,2,11,3,3,10,0,9,10,0,9,3,3,11,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,0,0,0,0,0,0,0,0,0,2,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,8,5,5,1,1,5,5,7,0,2,4,0,6,0,0,0,0,0},

{5,5,5,5,5,10,0,9,10,0,6,0,0,0,0,0,0,6,0,9,10,0,9,5,5,5,5,5},

{0,0,0,0,0,0,0,0,0,0,6,0,0,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0},

{5,5,5,5,5,7,0,8,7,0,6,0,0,0,0,0,0,6,0,8,7,0,8,5,5,5,5,5},

{0,0,0,0,0,6,0,2,4,0,9,5,5,5,5,5,5,10,0,2,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,0,0,0,0,0,0,0,0,0,2,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,8,1,1,1,1,1,1,7,0,2,4,0,6,0,0,0,0,0},

{8,5,5,5,5,10,0,9,10,0,9,3,3,11,11,3,3,10,0,9,10,0,9,5,5,5,5,7},

{6,0,0,0,0,0,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,0,0,0,0,0,0,6},

{6,0,8,1,1,7,0,8,1,1,1,7,0,2,4,0,8,1,1,1,7,0,8,1,1,7,0,6},

{6,0,9,3,11,4,0,9,3,3,3,10,0,9,10,0,9,3,3,3,10,0,2,11,3,10,0,6},

{6,0,0,0,2,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,4,0,0,0,6},

{2,1,7,0,2,4,0,8,7,0,8,1,1,1,1,1,1,7,0,8,7,0,2,4,0,8,1,4},

{2,3,10,0,9,10,0,2,4,0,9,3,3,11,11,3,3,10,0,2,4,0,9,10,0,9,3,4},

{6,0,0,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,0,0,6},

{6,0,8,1,1,1,1,11,11,1,1,7,0,2,4,0,8,1,1,11,11,1,1,1,1,7,0,6},

{6,0,9,3,3,3,3,3,3,3,3,10,0,9,10,0,9,3,3,3,3,3,3,3,3,10,0,6},

{6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6},

{9,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,10} };

int pebble\_array[BOARD\_X][BOARD\_Y] =

{ {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,3,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,3,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0},

{0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0},

{0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,3,1,1,0,0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,0,0,1,1,3,0},

{0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0},

{0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0},

{0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,0},

{0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0},

{0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0} };

GLubyte list[5];

int tp\_array[31][28];

int pebbles\_left;

double speed1 = 0.1;

double angle1 = 90;

double a = 13.5, b = 23;

bool animate = false;

int lives = 3;

int points = 0;

void keys();

unsigned char ckey = 'w';

void mykey(unsigned char key, int x, int y);

bool Open(int a, int b);

void Move()

{

a += speed1 \* cos(M\_PI / 180 \* angle1);

b += speed1 \* sin(M\_PI / 180 \* angle1);

if (animate && ckey == GLUT\_KEY\_UP && (int)a - a > -0.1 && angle1 != 270) //w

{

if (Open(a, b - 1))

{

animate = true;

angle1 = 270;

}

}

else if (animate && ckey == GLUT\_KEY\_DOWN && (int)a - a > -0.1 && angle1 != 90)// s

{

if (Open(a, b + 1))

{

animate = true;

angle1 = 90;

}

}

else if (animate && ckey == GLUT\_KEY\_LEFT && (int)b - b > -0.1 && angle1 != 180)//a

{

if (Open(a - 1, b))

{

animate = true;

angle1 = 180;

}

}

else if (animate && ckey == GLUT\_KEY\_RIGHT && (int)b - b > -0.1 && angle1 != 0)//d

{

if (Open(a + 1, b))

{

animate = true;

angle1 = 0;

}

}

}

void Pac(void)

{

//Draw Pacman

glColor3f(0, 1, 1);

glPushMatrix();

glTranslatef(a, -b, 0);

glTranslatef(0.5, 0.6, 0);

glTranslatef((float)BOARD\_X / -2.0f, (float)BOARD\_Y / 2.0f, 0.5);

glutSolidSphere(0.5, 15, 10);

glPopMatrix();

}

//Monster Drawing And Moving Begins

bool open\_move[4];

bool gameover = false;

int num\_ghosts = 4;

int start\_timer = 3;

class Ghost

{

private:

public:

bool edible;

int edible\_max\_time;

int edible\_timer;

bool eaten;

bool transporting;

float color[3];

double speed;

double max\_speed;

bool in\_jail;

int jail\_timer;

double angle;

double x, y;

Ghost(double, double);

~Ghost(void);

void Move(); //Move the Monster

void Update(void); //Update Monster State

void Chase(double, double, bool\*); //Chase Pacman

bool Catch(double, double); //collision detection

void Reinit(void);

void Vulnerable(void);

void Draw(void); //Draw the Monster

void game\_over(void);

};

Ghost\* ghost[4];

Ghost::~Ghost(void) {}

Ghost::Ghost(double tx, double ty)

{

tx = x;

ty = y;

angle = 90;

speed = max\_speed = 1;

color[0] = 1;

color[1] = 0;

color[2] = 0;

eaten = false;

edible\_max\_time = 300;

edible = false;

in\_jail = true;

jail\_timer = 30;

}

void Ghost::Reinit(void)

{

edible = false;

in\_jail = true;

angle = 90;

}

//Move Monster

void Ghost::Move()

{

x += speed \* cos(M\_PI / 180 \* angle);

y += speed \* sin(M\_PI / 180 \* angle);

}

void Ghost::game\_over()

{

}

void Ghost::Update(void)

{

if ((int)x == 0 && (int)y == 14 && (!(transporting)))

{

angle = 180;

}

if (x < 0.1 && (int)y == 14)

{

x = 26.9;

transporting = true;

}

if ((int)x == 27 && (int)y == 14 && (!(transporting)))

{

angle = 0;

}

if (x > 26.9 && (int)y == 14)

{

x = 0.1;

transporting = true;

}

if ((int)x == 2 || (int)x == 25)

transporting = false;

if (((int)x < 5 || (int)x > 21) && (int)y == 14 && !edible && !eaten)

speed = max\_speed / 2;

speed = max\_speed;

//edibility

if (edible\_timer == 0 && edible && !eaten)

{

edible = false;

speed = max\_speed;

}

if (edible)

edible\_timer--;

//JAIL

if (in\_jail && (int)(y + 0.9) == 11)

{

in\_jail = false;

angle = 180;

}

if (in\_jail && ((int)x == 13 || (int)x == 14))

{

angle = 270;

}

//if time in jail is up, position for exit

if (jail\_timer == 0 && in\_jail)

{

//move right to exit

if (x < 13)

angle = 0;

if (x > 14)

angle = 180;

}

//decrement time in jail counter

if (jail\_timer > 0)

jail\_timer--;

//EATEN GHOST SEND TO JAIL

if (eaten && ((int)x == 13 || (int)(x + 0.9) == 14) && ((int)y > 10 && (int)y < 15))

{

in\_jail = true;

angle = 90;

if ((int)y == 14)

{

eaten = false;

speed = max\_speed;

jail\_timer = 66;

x = 11;

}

}

}

bool Ghost::Catch(double px, double py)

{

// Collision Detection

if (px - x < 0.2 && px - x > -0.2 && py - y < 0.2 && py - y > -0.2)

{

return true;

}

return false;

}

//called when pacman eats a super pebble

void Ghost::Vulnerable(void)

{

if (!(edible))

{

angle = ((int)angle + 180) % 360;

speed = max\_speed;

}

edible = true;

edible\_timer = edible\_max\_time;

//speed1=0.15;

}

void Ghost::Chase(double px, double py, bool\* open\_move)

{

int c;

if (edible)

c = -1;

else

c = 1;

bool moved = false;

if ((int)angle == 0 || (int)angle == 180)

{

if ((int)c \* py > (int)c \* y && open\_move[1])

angle = 90;

else if ((int)c \* py < (int)c \* y && open\_move[3])

angle = 270;

}

else if ((int)angle == 90 || (int)angle == 270)

{

if ((int)c \* px > (int)c \* x && open\_move[0])

angle = 0;

else if ((int)c \* px < (int)c \* x && open\_move[2])

angle = 180;

}

//Random Moves Of Monsters

if ((int)angle == 0 && !open\_move[0])

angle = 90;

if ((int)angle == 90 && !open\_move[1])

angle = 180;

if ((int)angle == 180 && !open\_move[2])

angle = 270;

if ((int)angle == 270 && !open\_move[3])

angle = 0;

if ((int)angle == 0 && !open\_move[0])

angle = 90;

}

void Ghost::Draw(void)

{

if (!edible)

glColor3f(color[0], color[1], color[2]);

else

{

if (edible\_timer < 150)

glColor3f((edible\_timer / 10) % 2, (edible\_timer / 10) % 2, 1);

if (edible\_timer >= 150)

glColor3f(0, 0, 1);

}

if (eaten)

glColor3f(1, 1, 0); //When Eaten By PacMan Change Color To Yellow

glPushMatrix();

glTranslatef(x, -y, 0);

glTranslatef(0.5, 0.6, 0);

glTranslatef((float)BOARD\_X / -2.0f, (float)BOARD\_Y / 2.0f, 0.5);

glutSolidSphere(.5, 10, 10);

glPopMatrix();

}

void tp\_restore(void)

{

for (int ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = 0; j < BOARD\_Y; j++)

{

tp\_array[ISO][j] = pebble\_array[ISO][j];

}

}

pebbles\_left = 244;

}

void Draw(void)

{

glColor3f(1, 0, 1);

//split board drawing in half to avoid issues with depth

for (int ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = 0; j < BOARD\_Y / 2; j++)

{

glColor3f(0, 0, 1);

int call\_this = 0;

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glPushMatrix();

glTranslatef(0.5, 0.5, 0);

switch (board\_array[ISO][j])

{

case 4:

glRotatef(90.0, 0, 0, 1);

case 3:

glRotatef(90.0, 0, 0, 1);

case 2:

glRotatef(90.0, 0, 0, 1);

case 1:

call\_this = 1;

break;

case 6:

glRotatef(90.0, 0, 0, 1);

case 5:

call\_this = 2;

break;

case 10:

glRotatef(90.0, 0, 0, 1);

case 9:

glRotatef(90.0, 0, 0, 1);

case 8:

glRotatef(90.0, 0, 0, 1);

case 7:

call\_this = 3;

break;

}

glScalef(1, 1, 0.5);

glTranslatef(-0.5, -0.5, 0);

glCallList(list[call\_this]);

glPopMatrix();

//now put on the top of the cell

if (call\_this != 0 || board\_array[ISO][j] == 11)

{

glTranslatef(0, 0, -0.5);

glCallList(list[4]);

}

glPopMatrix();

if (tp\_array[ISO][j] > 0)

{

glColor3f(0, 300, 1 / (float)tp\_array[ISO][j]);

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glTranslatef(0.5, 0.5, 0.5);

glutSolidSphere(0.1f \* ((float)tp\_array[ISO][j]), 6, 6);

glPopMatrix();

}

}

}

int ISO;

for (ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = BOARD\_Y - 1; j >= BOARD\_Y / 2; j--)

{

glColor3f(0, 0, 1);

int call\_this = 0;

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glPushMatrix();

glTranslatef(0.5, 0.5, 0);

switch (board\_array[ISO][j])

{

case 4:

glRotatef(90.0, 0, 0, 1);

case 3:

glRotatef(90.0, 0, 0, 1);

case 2:

glRotatef(90.0, 0, 0, 1);

case 1:

call\_this = 1;

break;

case 6:

glRotatef(90.0, 0, 0, 1);

case 5:

call\_this = 2;

break;

case 10:

glRotatef(90.0, 0, 0, 1);

case 9:

glRotatef(90.0, 0, 0, 1);

case 8:

glRotatef(90.0, 0, 0, 1);

case 7:

call\_this = 3;

break;

}

glScalef(1, 1, 0.5);

glTranslatef(-0.5, -0.5, 0);

glCallList(list[call\_this]);

glPopMatrix();

//now put on top

if (call\_this != 0 || board\_array[ISO][j] == 11)

{

glTranslatef(0, 0, -0.5);

glCallList(list[4]);

}

glPopMatrix();

if (tp\_array[ISO][j] > 0)

{

glColor3f(0, 300, 1 / (float)tp\_array[ISO][j]);

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glTranslatef(0.5, 0.5, 0.5);

glutSolidSphere(0.1f \* ((float)tp\_array[ISO][j]), 6, 6);

glPopMatrix();

}

}

}

Pac();

}

bool Open(int a, int b)

{

if (board\_array[b][a] > 0)

{

return false;

}

return true;

}

void RenderScene();

void mykey(unsigned char key, int x, int y)

{

if (start\_timer > 0)

{

start\_timer--;

}

}

void specialDown(int key, int x, int y)

{

if (start\_timer > 0)

start\_timer--;

ckey = key;

if (key == GLUT\_KEY\_UP && (int)a - a > -0.1 && angle1 != 270) //w

{

if (Open(a, b - 1))

{

animate = true;

angle1 = 270;

}

}

else if (key == GLUT\_KEY\_DOWN && (int)a - a > -0.1 && angle1 != 90)// s

{

if (Open(a, b + 1))

{

animate = true;

angle1 = 90;

}

}

else if (key == GLUT\_KEY\_LEFT && (int)b - b > -0.1 && angle1 != 180)//a

{

if (Open(a - 1, b))

{

animate = true;

angle1 = 180;

}

}

else if (key == GLUT\_KEY\_RIGHT && (int)b - b > -0.1 && angle1 != 0)//d

{

if (Open(a + 1, b))

{

animate = true;

angle1 = 0;

}

}

}

void specialUp(int key, int x, int y)

{

}

void P\_Reinit()

{

a = 13.5;

b = 23;

angle1 = 90;

animate = false;

Pac();

}

void G\_Reinit(void)

{

start\_timer = 3;

//ghost initial starting positions

int start\_x[4] = { 11,12,15,16 };

float ghost\_colors[4][3] = { {255,0,0},{120,240,120},{255,200,200},{255,125,0} };

for (int i = 0; i < num\_ghosts; i++)

{

ghost[i]->Reinit();

ghost[i]->x = start\_x[i];

ghost[i]->y = 14;

ghost[i]->eaten = false;

ghost[i]->jail\_timer = i \* 33 + 66;

ghost[i]->max\_speed = 0.1 - 0.01 \* (float)i;

ghost[i]->speed = ghost[i]->max\_speed;

//colorize ghosts

for (int j = 0; j < 3; j++)

ghost[i]->color[j] = ghost\_colors[i][j] / 255.0f;

}

}

void renderBitmapString(float x, float y, void\* font, char\* string)

{

char\* c;

glRasterPos2f(x, y);

for (c = string; \*c != '\0'; c++)

{

glutBitmapCharacter(font, \*c);

}

}

void Write(char\* string)

{

while (\*string)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*string++);

}

void print(char\* string)

{

while (\*string)

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*string++);

}

//Display Function->This Function Is Registered in glutDisplayFunc

void RenderScene()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

//Through Movement->From One End To The Other

if ((int)a == 27 && (int)b == 14 && angle1 == 0)

{

a = 0;

animate = true;

}

else

if ((int)(a + 0.9) == 0 && (int)b == 14 && angle1 == 180)

{

a = 27;

animate = true;

}

//Collision Detection For PacMan

if (animate)

Move();

if (!(Open((int)(a + cos(M\_PI / 180 \* angle1)),

(int)(b + sin(M\_PI / 180 \* angle1)))) &&

a - (int)a < 0.1 && b - (int)b < 0.1)

animate = false;

if (tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] == 1)

{

tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;

pebbles\_left--;

points += 1;

}

//Super Pebble Eating

else if (tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] == 3)

{

tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;

pebbles\_left--;

points += 5;

for (int i = 0; i < 4; i++)

{

if (!ghost[i]->eaten)

ghost[i]->Vulnerable(); //Calls A Function To Make Monster Weak

}

}

//All The Pebbles Have Been Eaten

if (pebbles\_left == 0)

{

G\_Reinit();

P\_Reinit();

tp\_restore();

points = 0;

lives = 3;

}

if (!gameover)

Draw();

for (int d = 0; d < num\_ghosts; d++)

{

if (!gameover && start\_timer == 0)

ghost[d]->Update();

if (!ghost[d]->in\_jail &&

ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

{

bool open\_move[4];

//Finding Moves

for (int ang = 0; ang < 4; ang++)

{

open\_move[ang] = Open((int)(ghost[d]->x + cos(M\_PI / 180 \* ang \* 90)),

(int)(ghost[d]->y + sin(M\_PI / 180 \* ang \* 90)));

}

//Chase Pac Man

if (!ghost[d]->eaten)

{

if (ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

ghost[d]->Chase(a, b, open\_move);

}

else

{

if (ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

ghost[d]->Chase(13, 11, open\_move);

}

}

if (ghost[d]->in\_jail && !(Open((int)(ghost[d]->x + cos(M\_PI / 180 \* ghost[d]->angle)),

(int)(ghost[d]->y + sin(M\_PI / 180 \* ghost[d]->angle)))) && ghost[d]->jail\_timer > 0

&& ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

{

ghost[d]->angle = (double)(((int)ghost[d]->angle + 180) % 360);

}

if (!gameover && start\_timer == 0)

ghost[d]->Move();

ghost[d]->Draw();

if (!(ghost[d]->eaten))

{

bool collide = ghost[d]->Catch(a, b);

//Monster Eats PacMan

if (collide && !(ghost[d]->edible))

{

lives--;

if (lives == 0)

{

gameover = true;

lives = 0;

ghost[d]->game\_over();

}

P\_Reinit();

d = 4;

}

//PacMan Eats Monster And Sends It To Jail

else if (collide && ((ghost[d]->edible)))

{

ghost[d]->edible = false;

ghost[d]->eaten = true;

ghost[d]->speed = 1;

}

}

}

if (gameover == true)

{

glColor3f(1, 0, 0);

renderBitmapString(-5, 0.5,GLUT\_BITMAP\_HELVETICA\_18,(char\*)"gameover");

}

char tmp\_str[40];

glColor3f(1, 1, 0);

glRasterPos2f(10, 18);

sprintf(tmp\_str, "Points: %d", points);

Write(tmp\_str);

glColor3f(1, 0, 0);

glRasterPos2f(-5, 18);

sprintf(tmp\_str, "PAC MAN");

print(tmp\_str);

glColor3f(1, 1, 0);

glRasterPos2f(-12, 18);

sprintf(tmp\_str, "Lives: %d", lives);

Write(tmp\_str);

glutPostRedisplay();

glutSwapBuffers();

}

void create\_list\_lib()

{

//Set Up Maze Using Lists

list[1] = glGenLists(1);

glNewList(list[1], GL\_COMPILE);

//North Wall

glBegin(GL\_QUADS);

glColor3f(0, 0, 1);

glNormal3f(0.0, 1.0, 0.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 1.0);

glEnd();

glEndList();

list[2] = glGenLists(1);

glNewList(list[2], GL\_COMPILE);

glBegin(GL\_QUADS);

//North Wall

glColor3f(0, 0, 1);

glNormal3f(0.0, 1.0, 0.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 1.0);

//South Wall

glColor3f(0, 0, 1);

glNormal3f(0.0, -1.0, 0.0);

glVertex3f(1.0, 0.0, 0.0);

glVertex3f(1.0, 0.0, 1.0);

glVertex3f(0.0, 0.0, 1.0);

glVertex3f(0.0, 0.0, 0.0);

glEnd();

glEndList();

list[3] = glGenLists(1);

glNewList(list[3], GL\_COMPILE);

glBegin(GL\_QUADS);

//North Wall

glColor3f(0, 0, 1);

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 1.0);

//East Wall

glColor3f(0, 0, 1);

glNormal3f(1.0, 0.0, 0.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 0.0, 1.0);

glVertex3f(1.0, 0.0, 0.0);

glEnd();

glEndList();

list[4] = glGenLists(1);

glNewList(list[4], GL\_COMPILE);

glBegin(GL\_QUADS);

//Top Wall

glColor3f(-1, 0.3, 0);

glNormal3f(1.0, 0.0, 1.0);

glVertex3f(1, 1, 1.0);

glVertex3f(0, 1, 1.0);

glVertex3f(0, 0, 1.0);

glVertex3f(1, 0, 1.0);

glEnd();

glEndList();

}

void init()

{

/\* float color[4];

Enable Lighting.

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING);

Ambient And Diffuse Lighting

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

glEnable(GL\_COLOR\_MATERIAL);

color[0] = 1.0f; color[1] = 1.0f; color[2] = 0.0f; color[3] = 0.0f;

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, color);

color[0] = 1.0f; color[1] = 0.0f; color[2] = 1.0f; color[3] = 1.0f;

glLightfv(GL\_LIGHT0, GL\_AMBIENT, color);\*/

glEnable(GL\_NORMALIZE);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60, 1.33, 0.005, 100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(-1.5, 0, 40, -1.5, 0, 0, 0.0f, 1.0f, 0.0f);

}

void erase()

{

glColor3f(0.1, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(0, 0);

glVertex2f(0.5, 0);

glVertex2f(0.25, 0.5);

glEnd();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH);

glutInitWindowSize(1200, 780);

glutInitWindowPosition(0, 0);

glutCreateWindow("Pac GL 3D");

init();

glutDisplayFunc(RenderScene);

create\_list\_lib();

glutKeyboardFunc(mykey);

glutSpecialFunc(specialDown);

glutSpecialUpFunc(specialUp);

glEnable(GL\_DEPTH\_TEST);

int start\_x[4] = { 11,12,15,16 };

for (int ISO = 0; ISO < num\_ghosts; ISO++)

{

ghost[ISO] = new Ghost(start\_x[ISO], 14);

}

float ghost\_colors[4][3] = { {255,0,0},{120,240,120},{255,200,200},{255,125,0} };

int ISO;

for (ISO = 0; ISO < num\_ghosts; ISO++)

{

ghost[ISO]->x = start\_x[ISO];

ghost[ISO]->y = 14;

ghost[ISO]->eaten = false;

ghost[ISO]->max\_speed = 0.1 - 0.01 \* (float)ISO;

ghost[ISO]->speed = ghost[ISO]->max\_speed;

//colorize ghosts

for (int j = 0; j < 3; j++)

ghost[ISO]->color[j] = ghost\_colors[ISO][j] / 255.0f;

}

for (ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = 0; j < BOARD\_Y; j++)

{

tp\_array[ISO][j] = pebble\_array[ISO][j];

}

}

pebbles\_left = 244;

glShadeModel(GL\_SMOOTH);

glutMainLoop();

return 0;

}