

[illegible]

```

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
{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},
{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,0},
{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,0},
{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},
{0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0},
{0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0},
{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},
{0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0},
{0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0},
{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0},
{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,const char* string)
{
    const char* c;
    glRasterPos2f(x, y);
    for (c = string; *c != '\0'; c++)
    {
        glutBitmapCharacter(font, *c);
    }
}

void Write(const char* string)
{
    while (*string)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *string++);
}

void print(const 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, "GAME OVER");
}
char tmp_str[40];
glColor3f(1, 1, 0);
glRasterPos2f(10, 18);
sprintf_s(tmp_str, "Points: %d", points);
Write(tmp_str);
glColor3f(1, 0, 0);
glRasterPos2f(-5, 18);
sprintf_s(tmp_str, "PAC MAN");
print(tmp_str);
glColor3f(1, 1, 0);
glRasterPos2f(-12, 18);
sprintf_s(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;
}

```