컴퓨터그래픽스 미니 프로젝트

RAZER AND ME

정보통신공학과 12171756 김동언

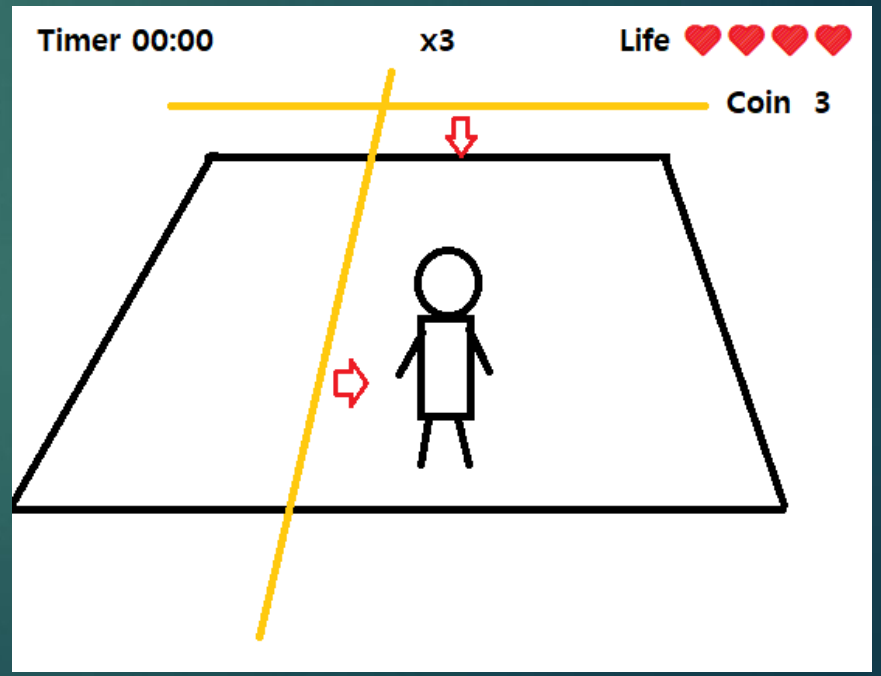
초기 제안서

▪ 게임의 종류

지형 위에서 주인공이 레이저를 피하면서 오랜 시간동안 살아남는 것이 목표인 게임입니다.

▪ 사용자 인터페이스

- 3인칭 카메라 시점으로 맵 위에 놓여진 캐릭터를 바라보게 됩니다.
- 상단에 Timer, 남아 있는 Life의 수,
획득한 Coin의 수, 난이도 단계가 출력됩니다.
난이도 단계가 높을 수록 레이저의 이동 속도,
생성 속도가 빨라집니다.



진행 방법

- 플레이어는 캐릭터를 조작하여 다가오는 레이저를 피하면서 살아남아야 합니다.
- 각 Round에는 Life가 주어집니다. 사방에서 다가오는 레이저에 부딪히면 Life가 감소합니다. 주어진 Life가 모두 사라지면 게임이 종료됩니다.
- 일정 주기로 맵의 구석에 Coin이 등장합니다. Coin을 획득하면 추가 점수가 주어집니다.
- 우클릭(Right Click)으로 Menu를 열 수 있고, 난이도 단계를 선택한 뒤 게임을 시작할 수 있습니다. 또는 Creative Mode에 접근할 수 있습니다.

기능

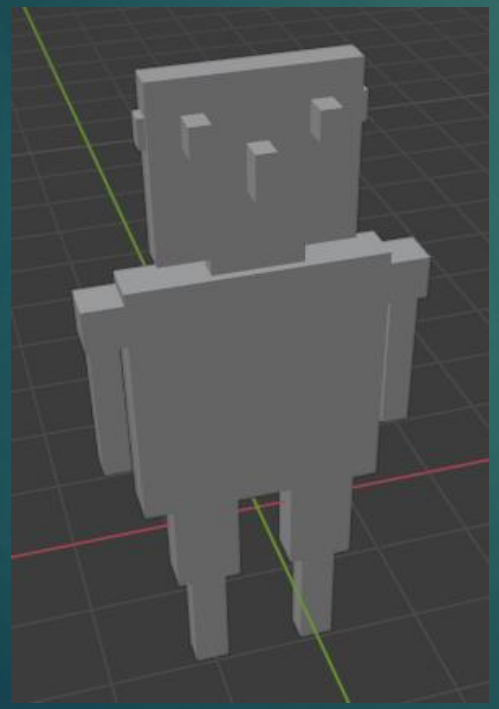
- 캐릭터의 움직임 : W, A, S, D
- 캐릭터의 점프 : Spacebar
- Camera 시점 이동 : Left Click + Drag
- 줌인, 줌아웃 : Mouse Wheel Up, Mouse Wheel Down
- 캐릭터 선택 : 1,2,3,4
- 난이도 선택 : 우클릭 (MENU에서 선택)
- 지시문 출력 : 5

창의적 요소

- 시간이 경과할수록 레이저가 빨라지므로 난이도가 상승합니다.
- 메뉴에서 [Creative] 를 선택하여 Creative Mode에 들어갈 수 있습니다. 게임 내부적 요소들(레이저의 색, 캐릭터 선택 등)을 수정할 수 있습니다.
- 각 캐릭터마다 외형을 다르게 만들고 이동 속도, 점프 높이 등을 다르게 하여 각자 특색을 가지도록 합니다.
- Map의 크기를 선택하는 것도 추가해볼 생각입니다.

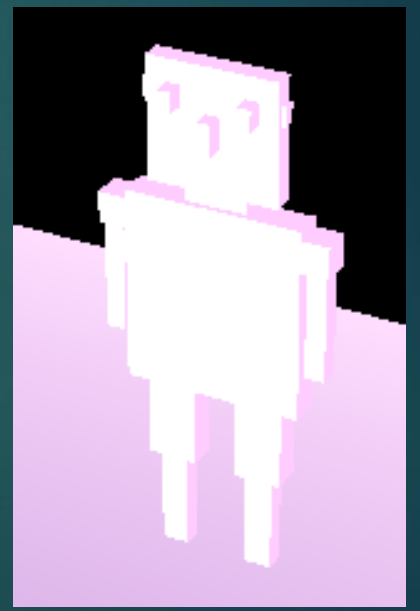
중간 체크 1

- 캐릭터의 움직임 작동을 확인하기 위해 임시로 Blender에서 팔, 다리가 있는 obj를 생성
- 팔, 다리가 움직이는 캐릭터를 구현하기 위해 이후 몸통, 팔, 다리의 obj를 따로 불러와 idle에서 움직이게 할 예정



```
// 블렌더 Character
ObjParser* character;
// 리소스 로드 함수
get_resource("character.obj");
```

```
void draw_obj(ObjParser* objParser)
{
    glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 1.0f, 1.0f);
    for (unsigned int n = 0; n < objParser->getFaceSize(); n += 3) {
        glNormal3f(objParser->normal[objParser->normalIdx[n] - 1].x,
            objParser->normal[objParser->normalIdx[n] - 1].y,
            objParser->normal[objParser->normalIdx[n] - 1].z);
        glVertex3f(objParser->vertices[objParser->vertexIdx[n] - 1].x,
            objParser->vertices[objParser->vertexIdx[n] - 1].y,
            objParser->vertices[objParser->vertexIdx[n] - 1].z);
```

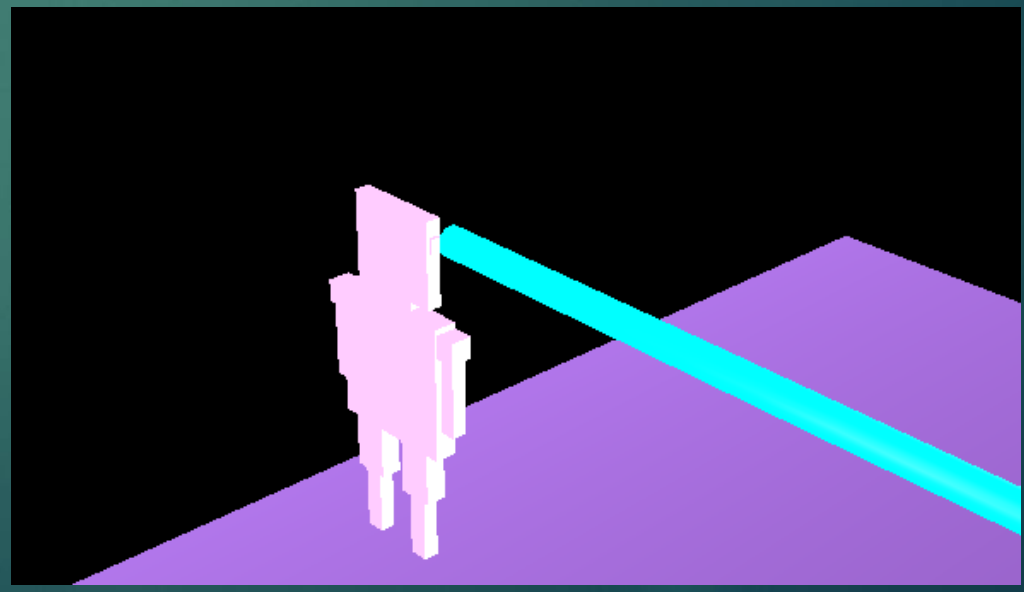


Razer Modeling

- GLU Quadric 객체 포인터 생성 : gluCylinder 함수 호출
- gultIdleFunc에서 4가지 방향, 3단계 높이 중 무작위로 선택하여 레이저를 생성하도록 함
- glutIdleFunc에서 4가지 방향에 따라 x축 혹은 z축을 따라 이동하기 위한 변수 값을 증가시키거나 감소시킴

```
if (path == 1)razer_x += 0.01 * level;
else if (path == 2) razer_x -= 0.01 * level;
```

- Cylinder 내부에 광원 생성 : Razer의 Local 좌표계로 이동 후 광원 설정
- Emission을 부여하여 빛나는 레이저를 구현



Coin Modeling

- GLU Quadric 객체 포인터 생성 -> gluCylinder 함수 호출로 Coin을 그림
- glBindTexture 함수를 이용하여 텍스쳐 read 및 mapping

```
void draw_textcoin() {
    glEnable(GL_TEXTURE_2D);
    glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
    glColor3f(1.0f, 1.0f, 1.0f);
    // 앞면
    glPushMatrix();
    glTranslatef(0, 0.8, 0.1);
    glBindTexture(GL_TEXTURE_2D, *coin);
    gluPartialDisk(qobj, 0, 1.5, 30, 2, 0, 360);
    glPopMatrix();
```

- glutIdleFunc에서 coin_ang를 0~360° 범위를 반복

draw() 에서 Coin의 Local 좌표계를 coin_ang만큼 회전시켜 제자리에서 회전하는 Coin을 구현

```
///////////////   Spin     ///////////////
coin_ang = coin_ang + 0.06;
if (coin_ang > 360) coin_ang -= 360;
```

```
glPushMatrix();
glTranslatef(-3, 0, 3);
glRotatef(coin_ang, 0, 1, 0);
draw_textcoin();
glPopMatrix();
```

출처 :
https://www.flaticon.com/download/icon/landing/2682893?format=png&size=256&type=standard

Keyboard 입력을 통한 Object의 움직임 구현

- glutKeyboardFunc Callback 함수에서 W, A, S, D 입력 시 전역 변수(mov)가 각 case에 해당하는 값으로 변경된다
- Case 32는 Spacebar로 다음 슬라이드의 Jump 구현에 이용

- glutIdleFunc Callback 함수에서 전역 변수 mov 값에 따라 x좌표나 z좌표가 증가하거나 감소함 (W의 경우 z좌표의 값이 감소)
- 전역변수 max=200으로 count가 0부터 200까지 증가하면서 총 2의 거리를 이동함

Key	W	S	A	D
Action	앞으로 이동	뒤로 이동	왼쪽으로 이동	오른쪽으로 이동

```
case 32:
    jump = 1;
    break;
case 'w':
    mov = 1;
    break;
case 'a':
    mov = 2;
    break;
case 's':
    mov = 3;
    break;
case 'd':
    mov = 4;
    break;
```

```
if (mov == 1) { // w
    mov_z -= 0.01;
    count++;
    if (count == max) {
        mov = 0;
        count = 0;
    }
}
```

Keyboard 입력을 통한 Object의 Jump 구현

- 이동과 마찬가지로 glutKeyboardFunc Callback 함수에서 Spacebar 입력 시 각 case에 해당하는 전역 변수(jump) 값이 1로 변경

- glutIdleFunc Callback 함수에서 y좌표 값을 증가시킴
- 최고 높이가 되면 jump값을 변경하고, y좌표 값을 감소시킴 (2021.12.17)
- 원래는 중력 가속도를 고려하려 했지만, 편한 플레이를 위해 일정한 속도로 점프하도록 수정

```
if (jump == 1) {
    jump_y += 0.01;
    if (jump_y > (double)3)
        jump = 2;
}
else if (jump == 2) {
    jump_y -= 0.01;
    if (jump_y < (double)0) {
        jump = 0;
        jump_y = 0.1;
    }
}
```

Key	Spacebar
Action	점프

▪ Mouse 드래그를 통한 Camera View 이동

- Object 중심의 Camera View는 구면 좌표계로 표현되어 있음

```
// Camera의 좌표를 구면좌표계로 변환
cam_mode1[0] = mov_x + radius * sin(theta * PI / 180) * sin(phi * PI / 180);
cam_mode1[1] = jump_y +radius * cos(theta * PI / 180);
cam_mode1[2] = mov_z + radius * sin(theta * PI / 180) * cos(phi * PI / 180);
```

- glutMouseFunc Callback 함수에서 Left Click 당시의 x. y 좌표와 state를 저장

```
if (button == 0 && state == GLUT_UP) {
    left = 1; // 좌클릭을 누르고 있는 case
    if(cam_mode==1)
        pos_x1 = x, pos_y1 = y;
    else if (cam_mode==2)
        pos_x2 = x, pos_y2 = y;
}
```

- glutMotionFunc Callback 함수에서 Left Click된 상태로 Motion을 감지했을 때의 x, y좌표와 비교하여 x, y의 증감에 따라 Camera View의 변수인 theta, phi값이 증가하거나 감소함

```
if (x > pos_x1) { // 오른쪽 drag : turn left
    if (phi == 0) phi = 357;
    else phi -= 3;
    pos_x1 = x;
}
else if (x < pos_x1) { // 왼쪽 drag : turn right
    if (phi == 360) phi = 3;
    else phi += 3;
    pos_x1 = x;
```

마우스 커서를 왼쪽에서 오른쪽으로 drag하면, 사물의 왼쪽을 보기 위함이므로 Camera 좌표가 왼쪽으로 이동하도록 함

▪ Mouse Wheel을 통한 Camera Zoom-in, out

- Object 중심의 Camera View는 구면 좌표계로 표현되어 있음

```
// Camera의 좌표를 구면좌표계로 변환
cam_mode1[0] = mov_x + radius * sin(theta * PI / 180) * sin(phi * PI / 180);
cam_mode1[1] = jump_y +radius * cos(theta * PI / 180);
cam_mode1[2] = mov_z + radius * sin(theta * PI / 180) * cos(phi * PI / 180);
```

- glutMouseWheelFunc Callback 함수에서

wheel을 올리면(dir > 0) radius 값을 감소시켜 Camera 좌표가 사물에 가까워짐

wheel을 내리면(dir < 0) radius 값을 증가시켜 Camera 좌표가 사물에서 멀어짐

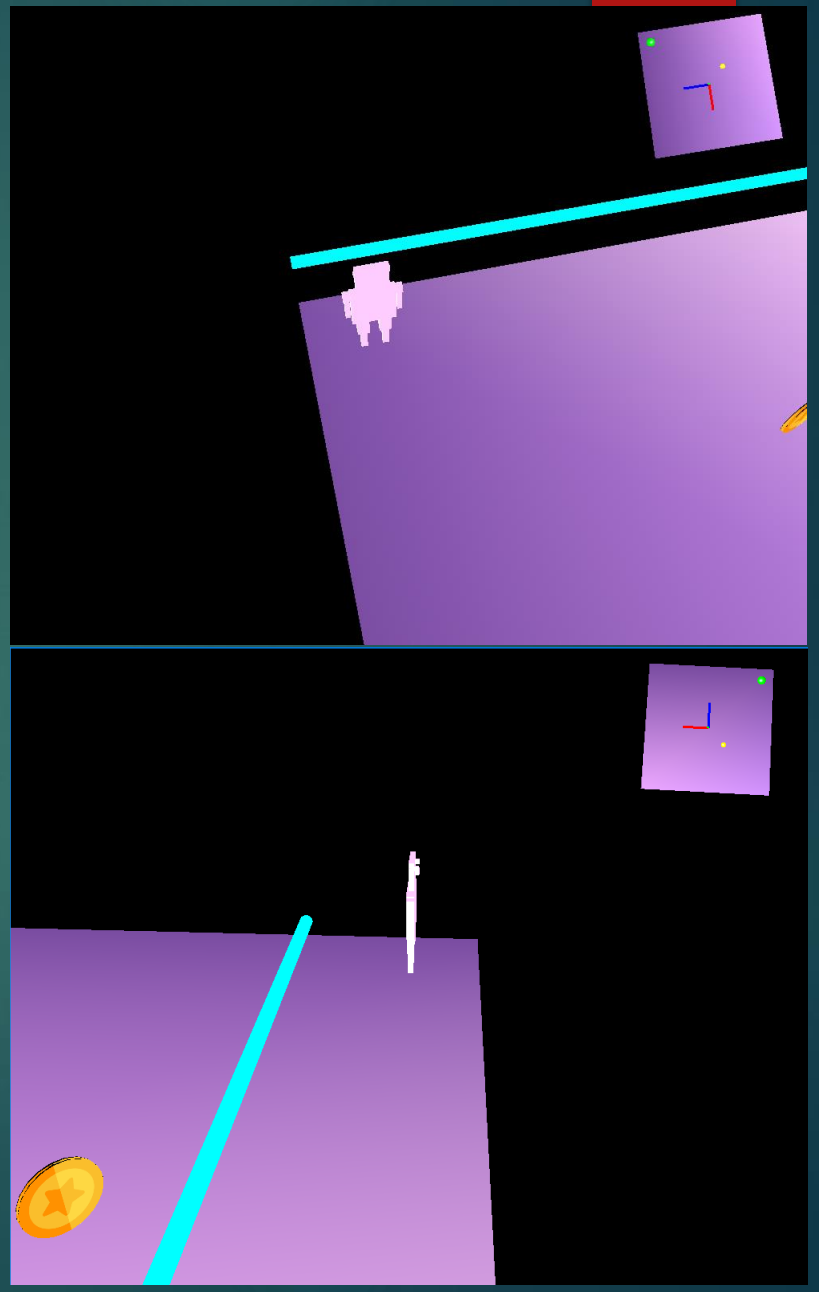
```
void mousewheel(int button, int dir, int x, int y) {

    if (dir > 0 && radius > 5) { // Wheel UP : 사물 확대
        radius -= 1;
    }
    else if (dir < 0) { // Wheel DOWN : 사물 축소
        radius += 1;
    }
    glutPostRedisplay();
}
```

Multi Viewport를 이용한 Mini-map

- draw()에서 Multi Viewport를 이용하여 우측 상단에 Top View를 제공하는 미니맵을 보여주는 minimap 함수를 호출
- PHI값이 기본 Viewport와 일치하여 Coin의 방향을 찾는 데 도움이 됨
- Blue line : 정면 / Red line : 왼쪽
- Character : 초록색 구 / Coin : 노란색 구로 표시

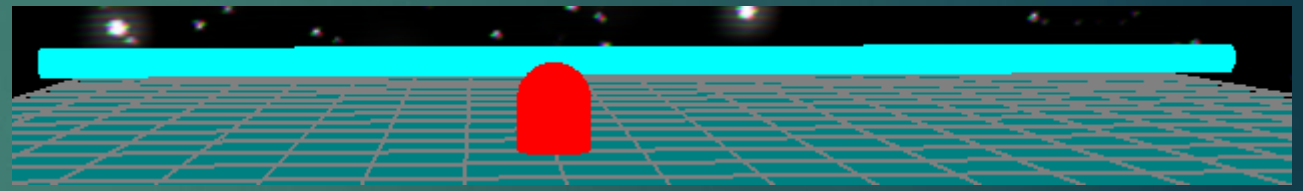
```
void minimap(double x, double y, double z) {
    glViewport(0.75 * current_width, 0.75 * current_height, 0.25 * current_width, 0.25 * current_height);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(x,y,z, 0, 0, 0, 0, up, 0); // Viewing Transform
    glPushMatrix();
        glTranslatef(mov_x, 0, mov_z);
        glColor3f(0.0f, 1.0f, 0.0f); // 캐릭터 -> Green Sphere
        gluSphere(qobj2, 1.3, 30, 30);
    glPopMatrix();
```

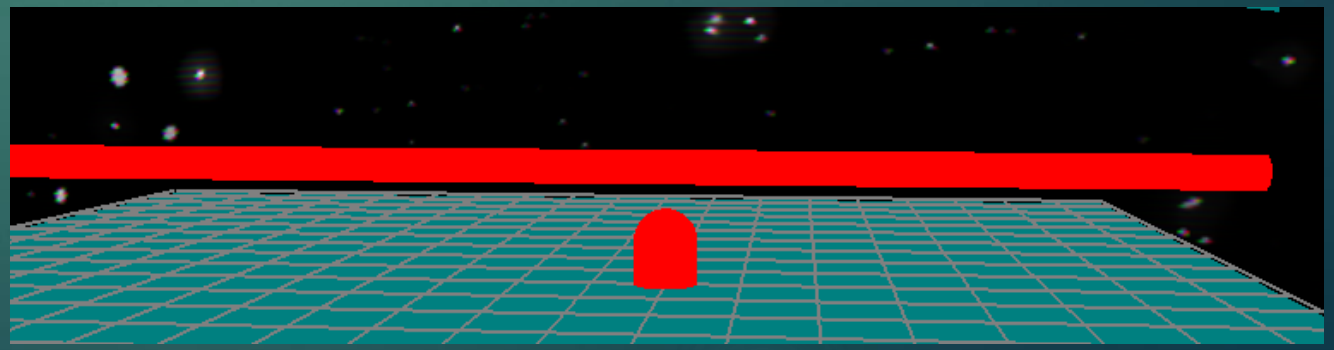


중간 체크 이후

- Razer 시각적 요소 개선

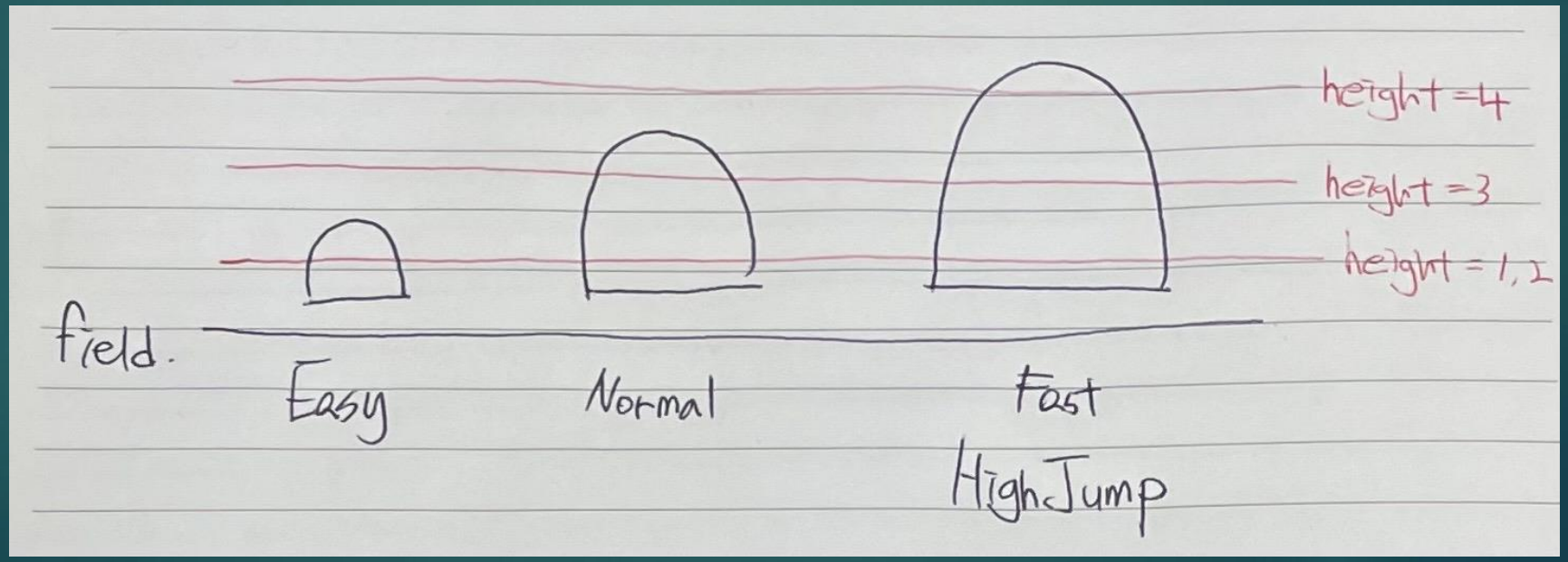
- 4가지 방향에서 무작위로 다가오도록 함
- 3단계 높이에 따라 레이저 색상을 다르게 하여 사용자의 게임 플레이에 도움을 주도록 함
- Cyan : Height 1 & 2
- Yellow : Height 3
- Red : Height 4





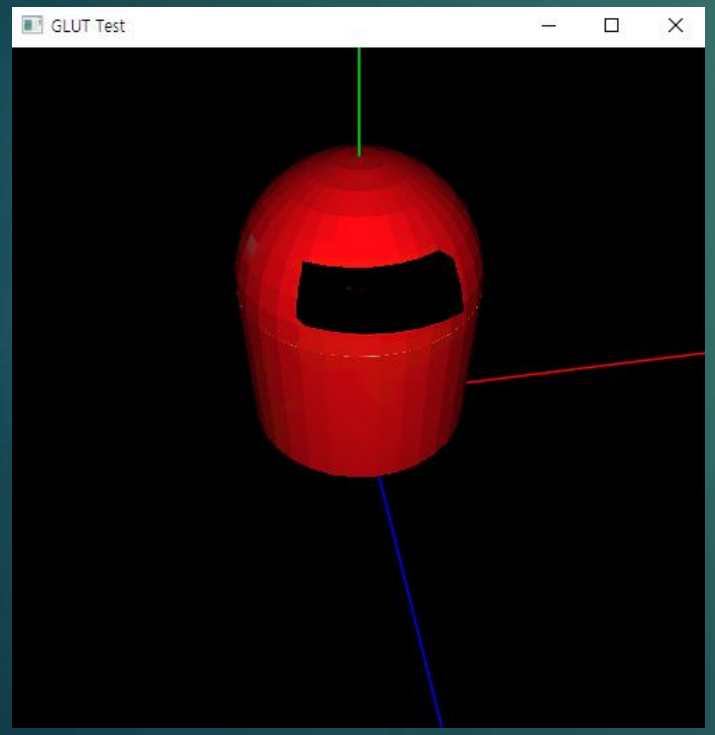
개성 있는 Character Design

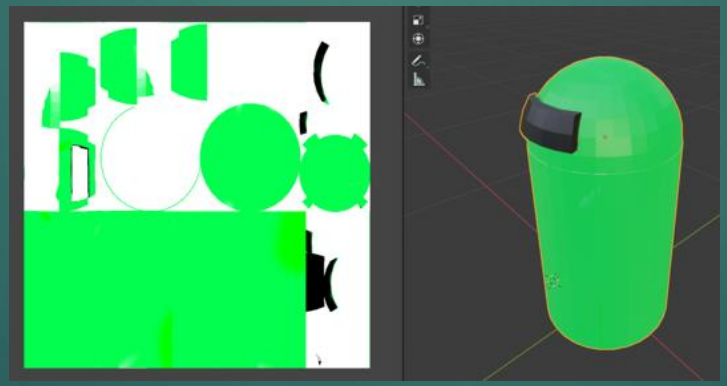
- 각 캐릭터들의 외형을 다르게 하고, 특성을 부여하여 플레이 방식이 다르게 함
- 총 3가지의 캐릭터를 구상
- Blue : 쉬운 난이도, 가장 낮은 Razer에만 닿음
- Yellow, Red : 중간 난이도(Default), 가장 높은 Razer에는 닿지 않음
- Green : 모든 Razer에 닿지만, 점프력이 높고 이동 속도가 빠르게 함

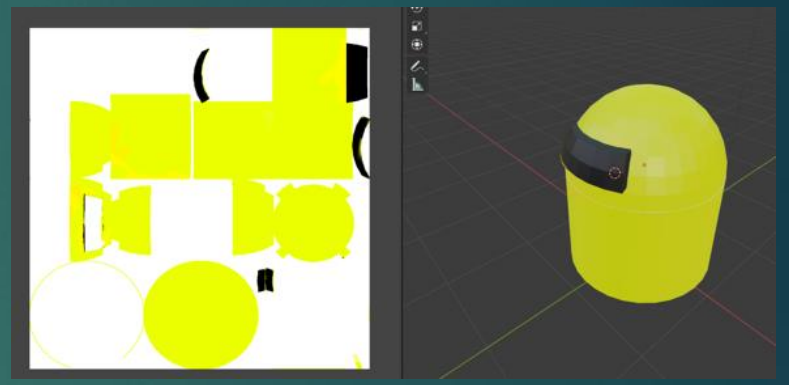


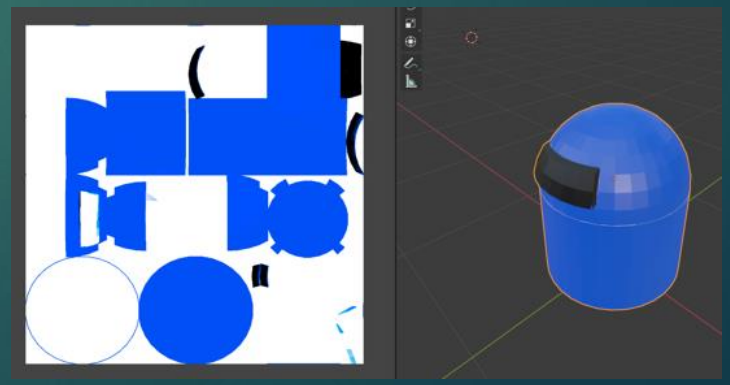
Blender에서 Character의 obj, texture bmp 생성

- Obj를 만들고, 텍스쳐를 직접 그려 bmp파일 생성
- Blue : 쉬운 난이도, 가장 낮은 Razer에만 닿음
- Yellow, Red : 중간 난이도(Default), 가장 높은 Razer에는 닿지 않음
- Green : 모든 Razer에 닿지만, 점프력이 높고 이동 속도가 빠르게 함



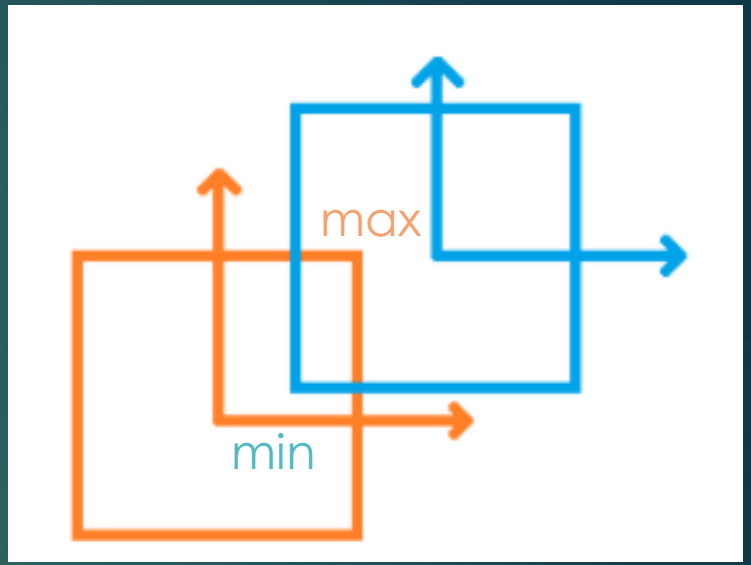






Character와 Razer의 충돌감지 구현

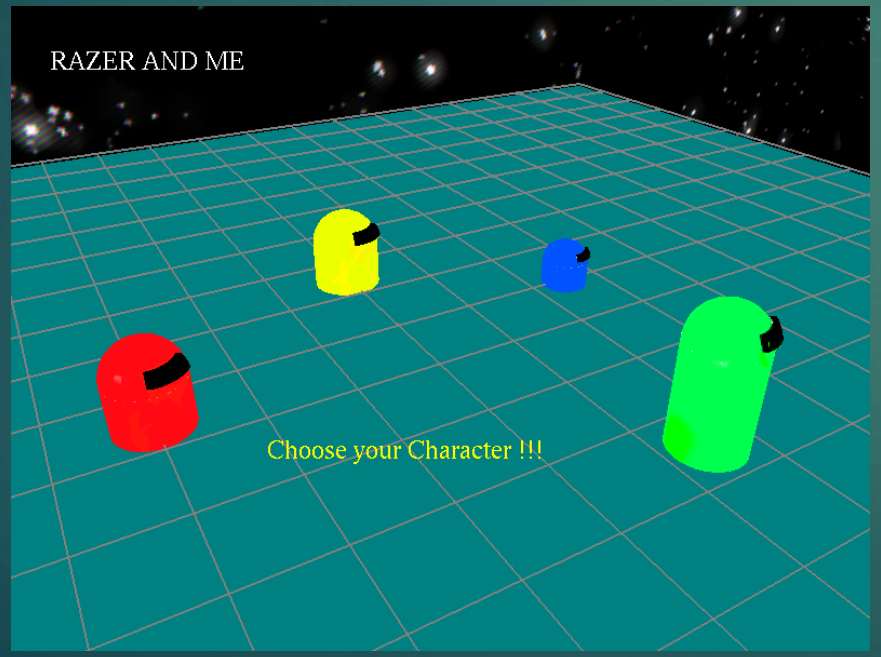
- 충돌: aabb(Axis-Aligned Bounding Box)를 이용
- Character와 Razer 설계를 바탕으로 각 Character와 움직이는 Razer의 실시간 xyz 최소, 최대 좌표를 계산하여 배열에 저장
- 어느 aabb의 x,y,z의 최대 좌표가 상대 aabb의 x,y,z의 최소 좌표보다 작으면 충돌이 일어난 것
- 충돌이 일어나면 Life가 감소

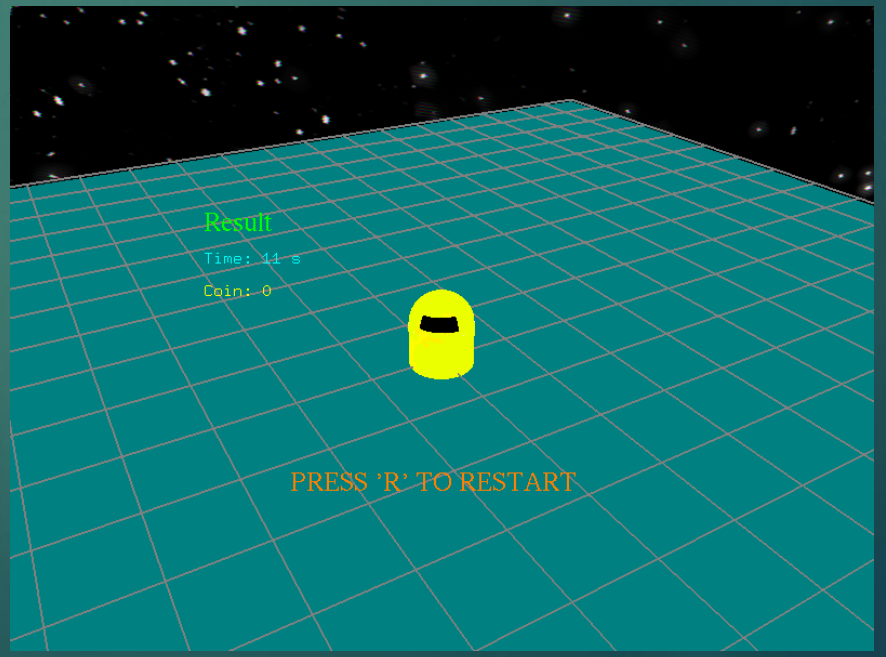



```
int collision() {
    if (razer_hit == 1) return 0;
    else {
        if (aabb_obj[3] < aabb_razer[0] || aabb_razer[3] < aabb_obj[0]) return 0;
        else if (aabb_obj[4] < aabb_razer[1] || aabb_razer[4] < aabb_obj[1]) return 0;
        else if (aabb_obj[5] < aabb_razer[2] || aabb_razer[5] < aabb_obj[2]) return 0;
        else return 1;
    }
}
```

Character Select, Result 화면

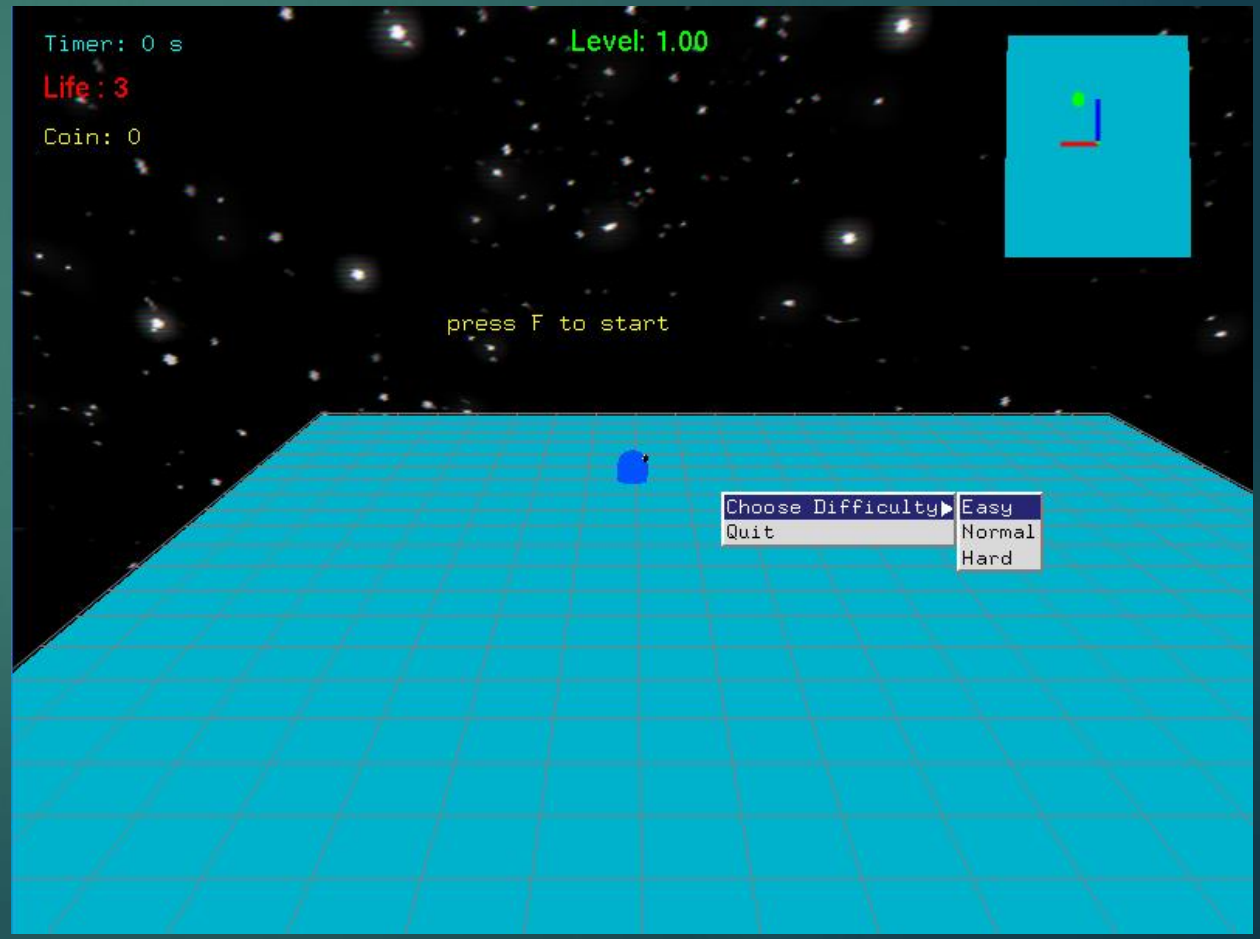
- 시작 화면으로 캐릭터 선택 화면이 등장함
- 총 4가지 Character가 등장하여 사용자가 게임에 들어가기 전 원하는 캐릭터를 선택할 수 있게 함 (Keyboard로 선택)
- 모든 Life를 잃으면 Result 화면으로 이동하여 생존 시간, Coin 수를 보여줌
- 'R'을 누르면 초기 캐릭터 선택 화면으로 이동함





Game Play 화면

- UI : 타이머, Life, 획득한 Coin , 미니맵, 현재 Level
- Level은 시간이 지날 수록 높아지며, Razer의 이동 속도와 비례함
- 게임 시작 전, Menu를 통해 난이도를 선택할 수 있음
- Easy : 1.0 / Normal : 2.0 / Hard : 3.0
- ‘F’를 누르면 선택한 난이도로 게임 시작



▪ Effect 추가 : 화면에 알림 메시지 출력

- Coin은 7초간 등장했다가 사라짐 : 등장 3초 전부터 경고 메시지가 출력

- Coin을 획득하면 Coin을 획득했다는 상태 메시지 출력

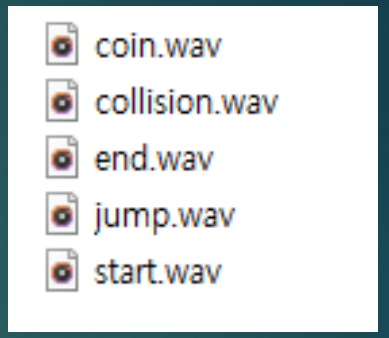
  획득한 Coin은 사라짐

- Razer와 충돌하면 피격당하여 Life가 감소했다는 상태 메시지 출력

  충돌한 Razer는 사라짐

Effect 추가 : Action에 해당하는 효과음 재생

- PlaySound 함수를 이용하여 다음 Action에 효과음을 추가
- Game Start, Game Over, Jump, Collision, Coin Get
- 메시지를 포함한 여러 방법으로 자신의 Status를 파악할 수 있게 함

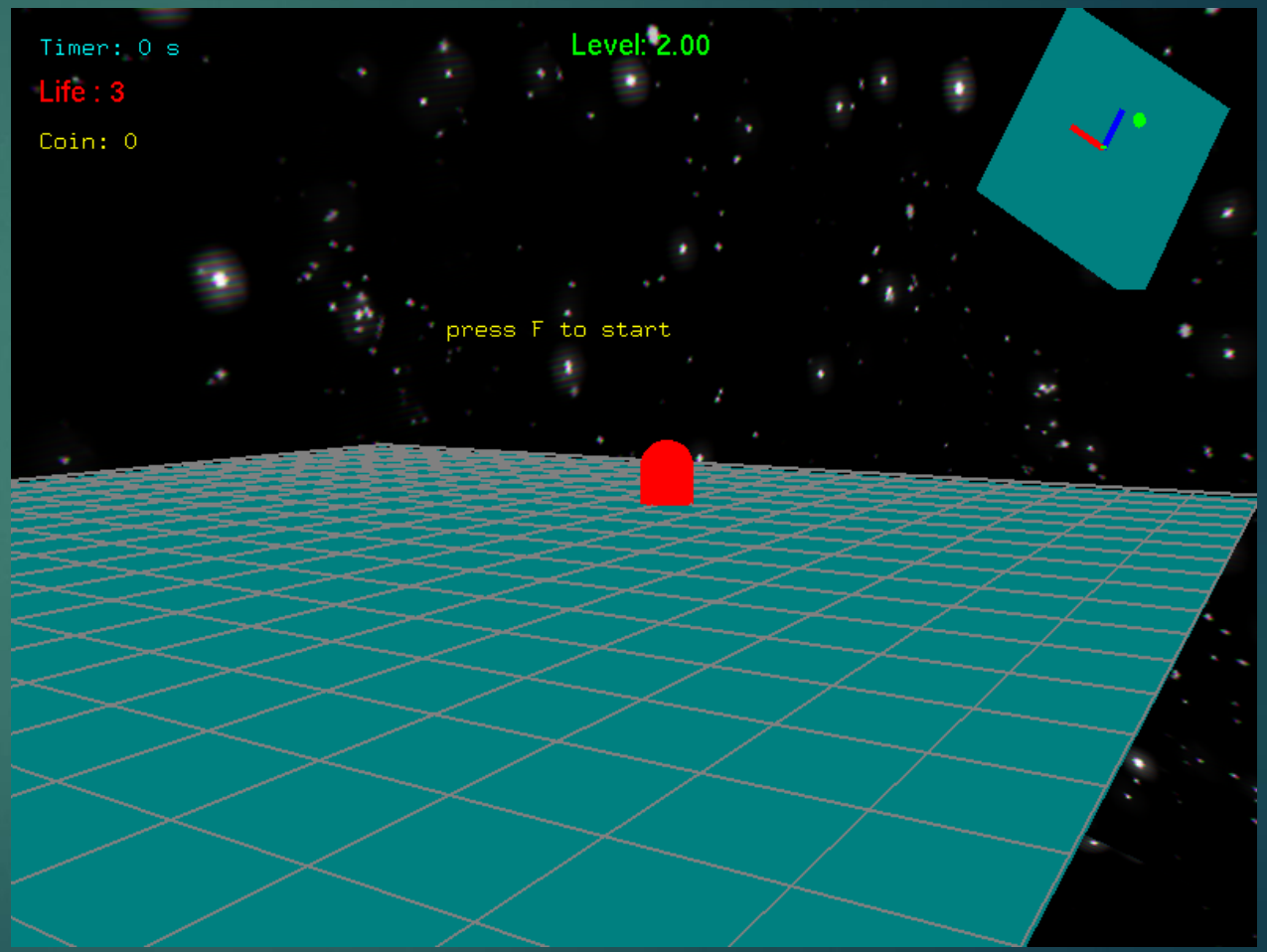



```
if (life == 0 && mode == 2) {
    g_start = false;
    mode = 3;
    PlaySound(TEXT("sound/end.wav"), NULL, SND_ASYNC | SND_ALIAS); // GAME OVER 효과음
}
if (g_start == true && collision() == 1 && razer_hit == 0) { // 충돌 시 Life 감소
    if (life > 1)PlaySound(TEXT("sound/collision.wav"), NULL, SND_ASYNC | SND_ALIAS); // 충돌 효과음
    life--;
    razer_hit = 1;
}
```

출처 :
https://www.pacdv.com/sounds/mechanical_sound_effects/gun-reload-1.wav
https://www.soundeffectsplus.com/product/wrong-answer-fall-03/
https://www.pacdv.com/sounds/mechanical_sound_effects/coin_2.wav
https://www.pacdv.com/sounds/mechanical_sound_effects/cling_2.wav
https://youtubelab.tistory.com/28

- Environment Mapping

– 우주를 배경으로 한 Cube Map을 생성

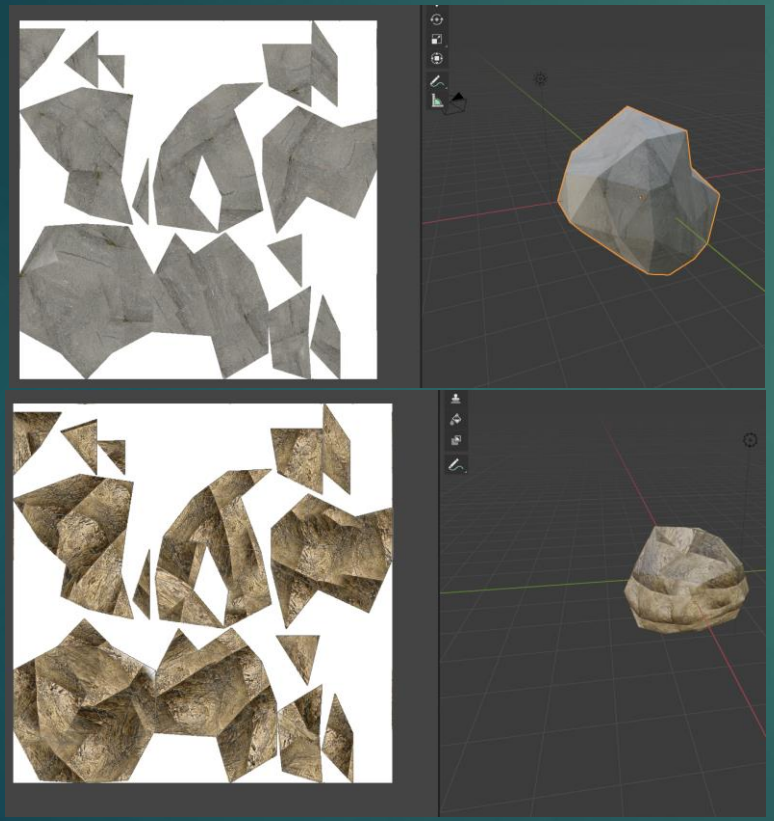


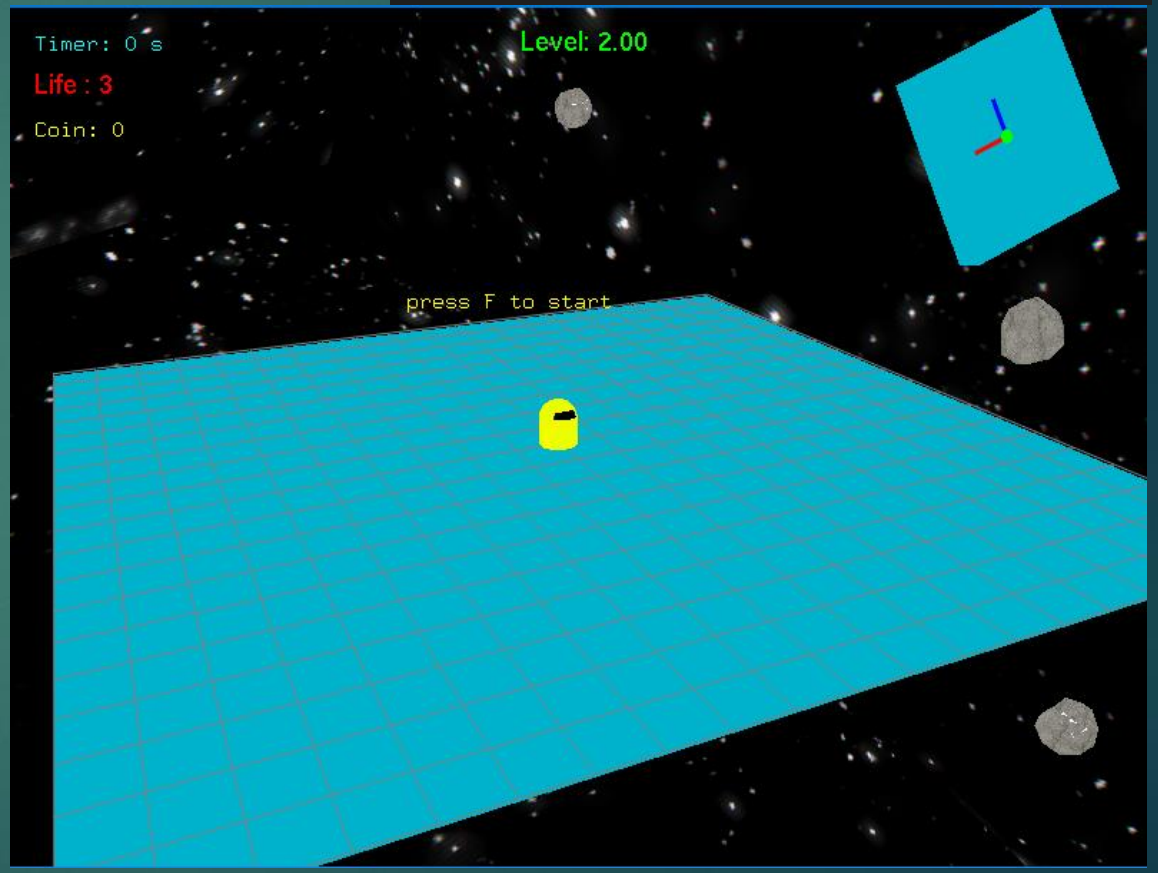
출처 :
https://mblogthumb-phinf.pstatic.net/20140809_41/libet92_1407513107338sB1eB_JPEG/21.jpg?type=w800

Blender – Trash Modeling

- 우주에 떠다니는 소행성들을 모델링
- Parent 좌표계를 회전 => 맵 중앙을 기준으로 주변을 회전함

```
glPushMatrix();
glRotatef(trash1_ang, 0, 0.5, 0.5);
glTranslatef(45, 0, 0);
draw_trash1(trash1);
glPopMatrix();
```





출처 : http://texturelib.com/texture/?path=/Textures/metal/bare/metal_bare_0010
http://texturelib.com/texture/?path=/Textures/plastic/plastic_0032

감사합니다