3D Levels

(we need a floor)


PICKED UP A CLIP.
65
AMMO
100%
HEALTH
2 3 4
5 6 7
ARMS
0%
ARMOR
BULL 65 / 200
SHEL 0 / 50
RCKT 0 / 50
CELL 0 / 300


2nd

We can use a cube and scale it to be large along the X and Z axis and small on the Y axis.

Let’s Code!

Add scale to Entity.h and Entity.cpp
(translate, scale rotate)
Add cube.obj and a texture to our project.
Add them to our scene.
Check it out!

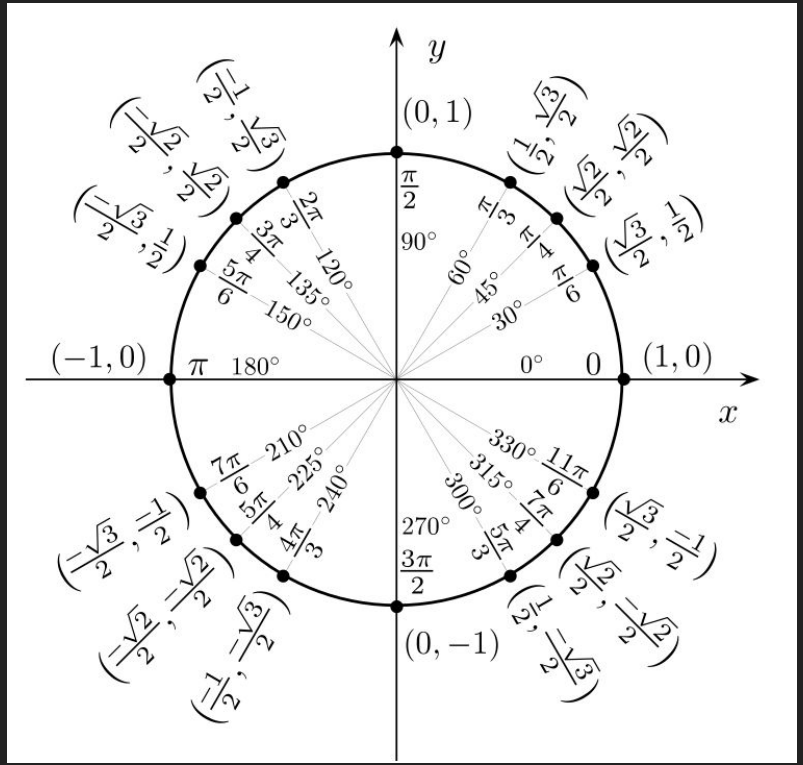
We can grab a texture from this pack:
https://opengameart.org/content/seamless-textures

How can we make the texture repeat/tile?

Moving Around

We want to be able to turn and move in the direction we are facing.

The Unit Circle



We need to make some updates to main.cpp

```
// End of ProcessInput()

const Uint8 *keys = SDL_GetKeyboardState(NULL);

if (keys[SDL_SCANCODE_A]) {
    state.player->rotation.y += 1.0f;
} else if (keys[SDL_SCANCODE_D]) {
    state.player->rotation.y -= 1.0f;
}
```

We need to make some updates to main.cpp

```
state.player->velocity.x = 0;
state.player->velocity.z = 0;

if (keys[SDL_SCANCODE_W]) {
    state.player->velocity.z = cos(glm::radians(state.player->rotation.y)) * -2.0f;
    state.player->velocity.x = sin(glm::radians(state.player->rotation.y)) * -2.0f;
} else if (keys[SDL_SCANCODE_S]) {
    state.player->velocity.z = cos(glm::radians(state.player->rotation.y)) * 2.0f;
    state.player->velocity.x = sin(glm::radians(state.player->rotation.y)) * 2.0f;
}
```

We need to make some updates to main.cpp

```
// Middle of Update()
Make sure the player is updating.
(edit Entity.cpp to not rotate automatically)

// End of Update() - This is the reverse of the player.
viewMatrix = glm::mat4(1.0f);
viewMatrix = glm::rotate(viewMatrix,
    glm::radians(state.player->rotation.y), glm::vec3(0, -1.0f, 0));
viewMatrix = glm::translate(viewMatrix, -state.player->position);

// Top of Render()
program.SetViewMatrix(viewMatrix);
```

Let’s Code!

We should be able to turn Left and Right
and move around!

One that is working, let’s add
some crates to our scene!

3D Collision Detection

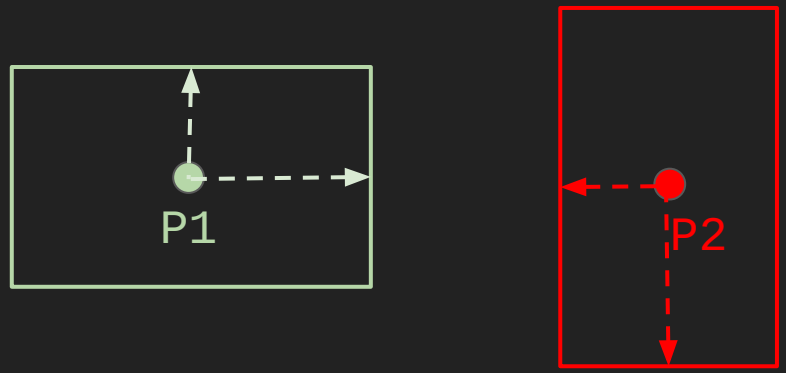
AABB

Axis-Aligned Bounding Box

Just like 2D, we are going to use a box around our objects that is not rotated.

Even if our entity is rotated,
we still keep the same box.
(this is easier than handling rotation)

Review (2D)
Box - Box Collision Detection




```
float xdist = fabs(x2 - x1) - ((w1 + w2) / 2.0f);
float ydist = fabs(y2 - y1) - ((h1 + h2) / 2.0f);

if (xdist < 0 && ydist < 0) // Colliding!
```

3D Box - Box Collision Detection

(just need to add the z)

```
float xdist = fabs(x2 - x1) - ((w1 + w2) / 2.0f);
float ydist = fabs(y2 - y1) - ((h1 + h2) / 2.0f);
float zdist = fabs(z2 - z1) - ((d1 + d2) / 2.0f);

if (xdist < 0 && ydist < 0 && zdist < 0) // Colliding!
```

We need to update Entity.h

```
// Add an enum
enum EntityType { NONE, FLOOR, BOX, ENEMY };



// Add inside of class

EntityType entityType;

bool billboard;
float width;
float height;
float depth;

bool CheckCollision(Entity *other);
void Update(float deltaTime, Entity *player, Entity *objects, int objectCount);
```

We need to update Entity.cpp

```
Entity::Entity()
{
    position = glm::vec3(0);
    scale = glm::vec3(1.0f, 1.0f, 1.0f);
    acceleration = glm::vec3(0, 0, 0);
    rotation = glm::vec3(0, 0, 0);
    width = 1.0f;
    height = 1.0f;
    depth = 1.0f;
}
```

```
bool Entity::CheckCollision(Entity *other)
{
    float xdist = fabs(position.x - other->position.x) - ((width + other->width) / 2.0f);
    float ydist = fabs(position.y - other->position.y) - ((height + other->height) / 2.0f);
    float zdist = fabs(position.z - other->position.z) - ((depth + other->depth) / 2.0f);

    if (xdist < 0 && ydist < 0 && zdist < 0) return true;

    return false;
}
```

We need to update Entity.cpp

```
void Entity::Update(float deltaTime, Entity *player, Entity *objects, int objectCount)
{
    glm::vec3 previousPosition = position;

    velocity += acceleration * deltaTime;
    position += velocity * deltaTime;

    for (int i = 0; i < objectCount; i++)
    {
        // Ignore collisions with the floor
        if (objects[i].entityType == FLOOR) continue;

        if (CheckCollision(&objects[i])) {
            position = previousPosition;
            break;
        }
    }
}
```

Let’s Code!

Update Entity.h and Entity.cpp

Update main.cpp
There should be 2 boxes to try to walk into.

Billboards

(sprites that always face the camera)

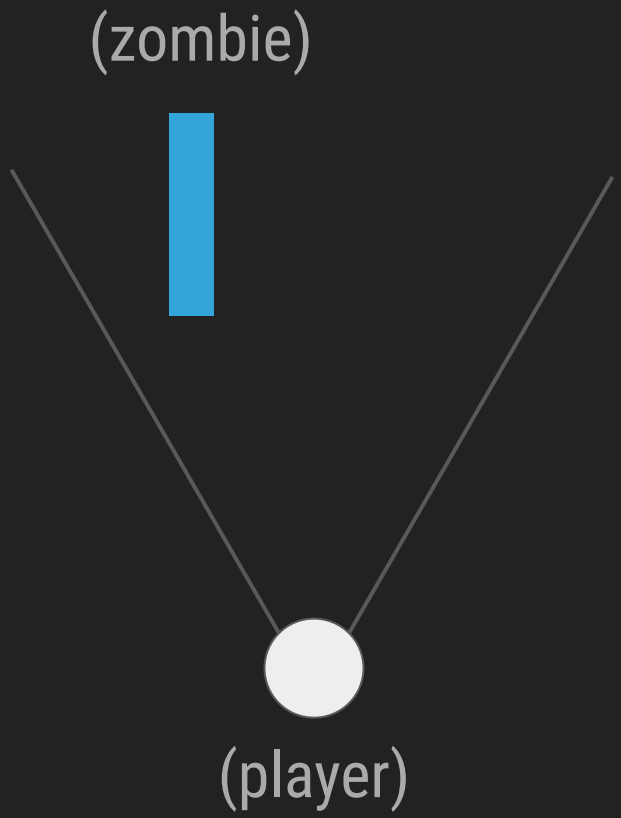

107
AMMO
77%
HEALTH
2 3 4
5 6 7
ARMS
176%
ARMOR
BULL 107 / 200
SHEL 24 / 50
ROKT 0 / 50
CELL 0 / 300


00:00:26
0 RCKT
111 LBUL
12 SHEL
99 BULL
+30 LARGE BULLETS
83
26

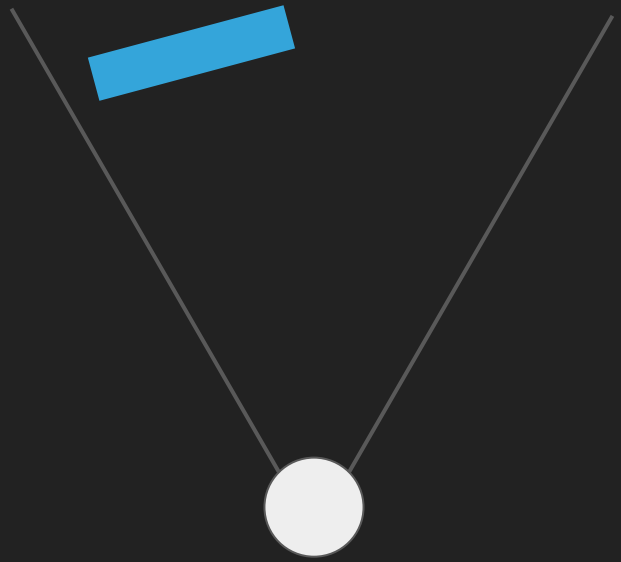
Using Billboards

2 Triangles
Texture

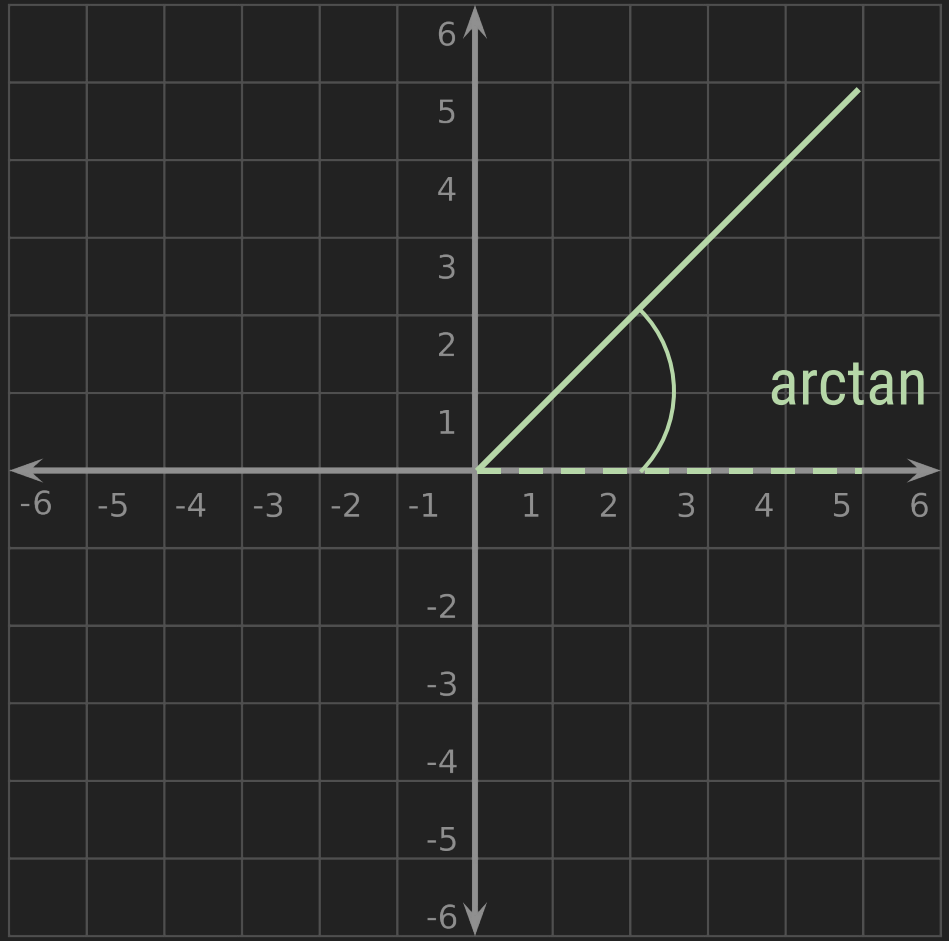
Rotate to always face the Camera (Player)


(zombie)
(player)

(zombie)



(player)


6
5
4
3
2
1
-6
-5
-4
-3
-2
-1
1
2
3
4
5
6
-2
-3
-4
-5
-6
arctan

Calculating how we need to turn to face the player.

```
// Inside of Entity::Update

if (billboard) {
    float directionX = position.x - player->position.x;
    float directionZ = position.z - player->position.z;
    rotation.y = glm::degrees(atan2f(directionX, directionZ));
}
```

Rendering

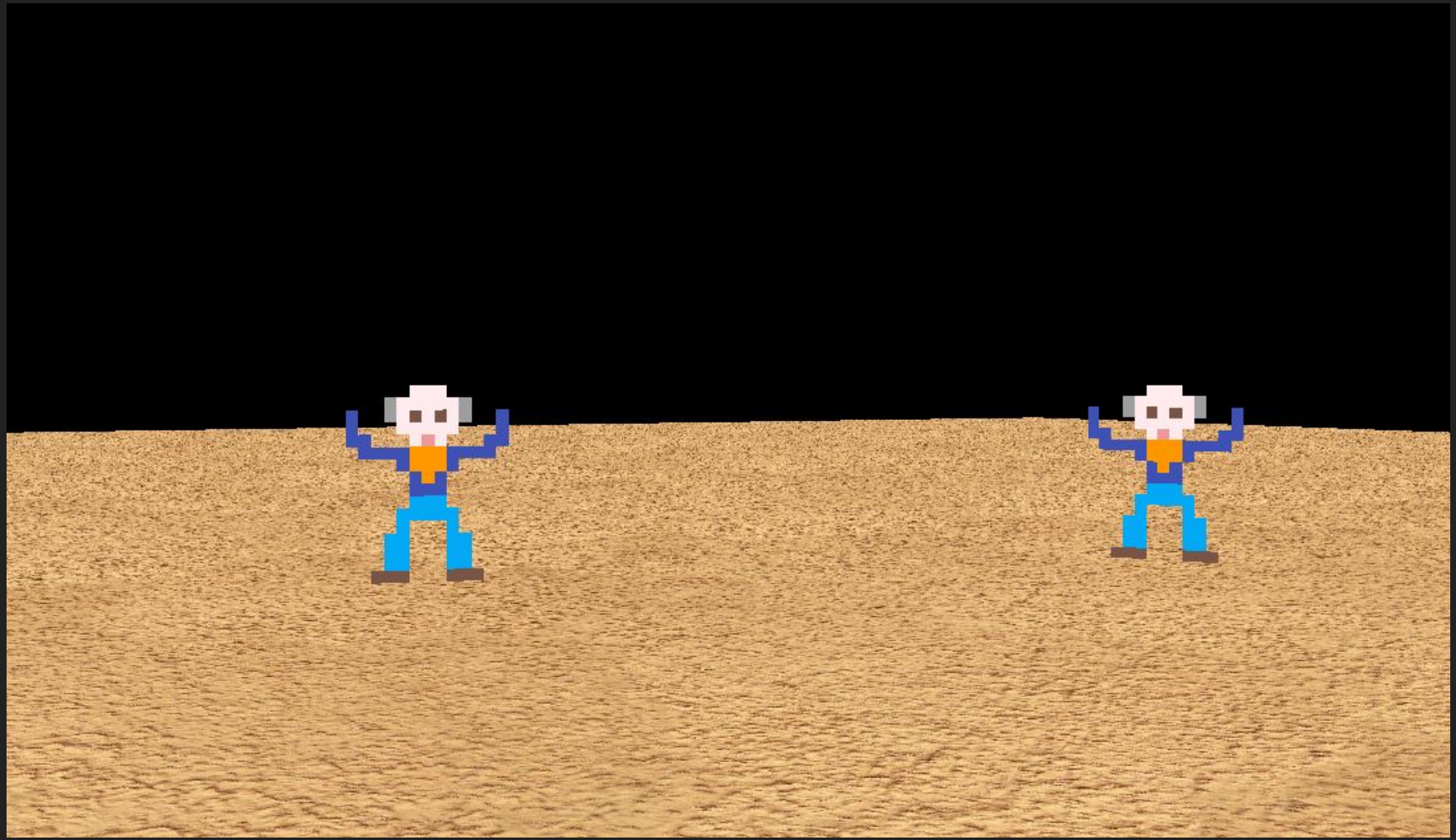
```
// Inside of Entity::Render

glBindTexture(GL_TEXTURE_2D, textureID);

if (billboard) {
    DrawBillboard(program);
} else {
    mesh->Render(program);
}
```

Rendering

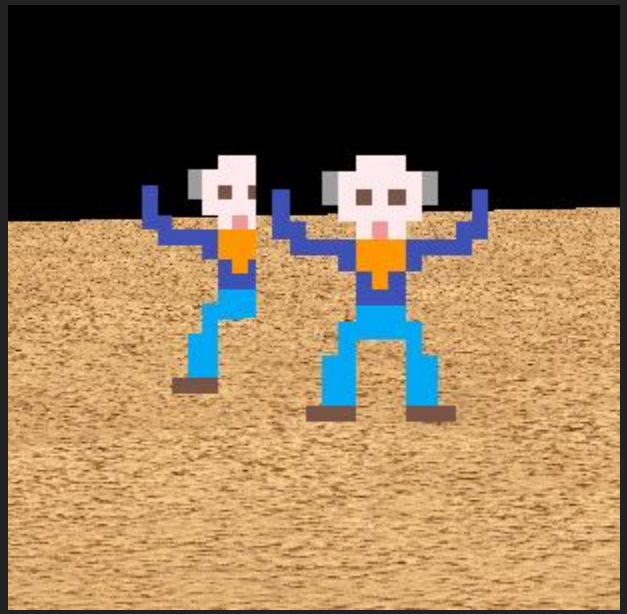
```
void Entity::DrawBillboard(ShaderProgram *program) {
    float vertices[] = { -0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5 };
    float texCoords[] = { 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0 };

    glVertexAttribPointer(program->positionAttribute, 2, GL_FLOAT, false, 0, vertices);
    glEnableVertexAttribArray(program->positionAttribute);

    glVertexAttribPointer(program->texCoordAttribute, 2, GL_FLOAT, false, 0, texCoords);
    glEnableVertexAttribArray(program->texCoordAttribute);

    glDrawArrays(GL_TRIANGLES, 0, 6);

    glDisableVertexAttribArray(program->positionAttribute);
    glDisableVertexAttribArray(program->texCoordAttribute);
}
```

Initializing

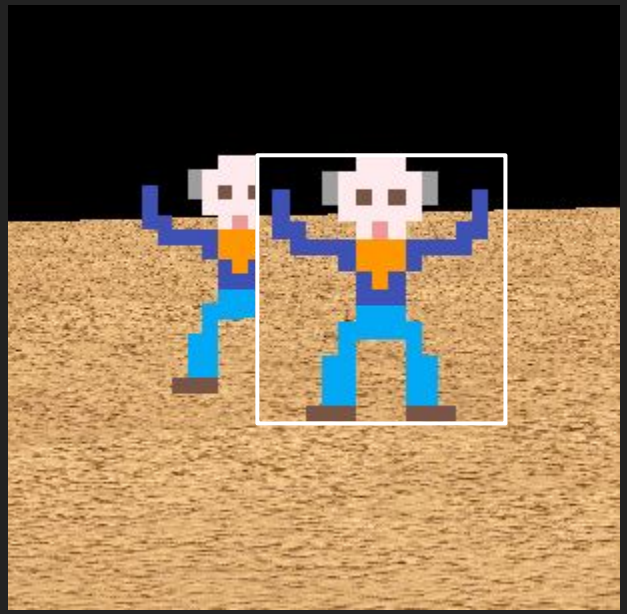
```
// Inside of main.cpp Initialize

GLuint enemyTextureID = Util::LoadTexture("ctg.png");

for (int i = 0; i < ENEMY_COUNT; i++) {
     state.enemies[i].billboard = true;
     state.enemies[i].textureID = enemyTextureID;
     state.enemies[i].position = glm::vec3(rand() % 20 - 10, 1, rand() % 20 - 10);
     state.enemies[i].rotation = glm::vec3(0, 0, 0);
     state.enemies[i].acceleration = glm::vec3(0, 0, 0);
}
```



You might see an error caused by the depth buffer and alpha channel.

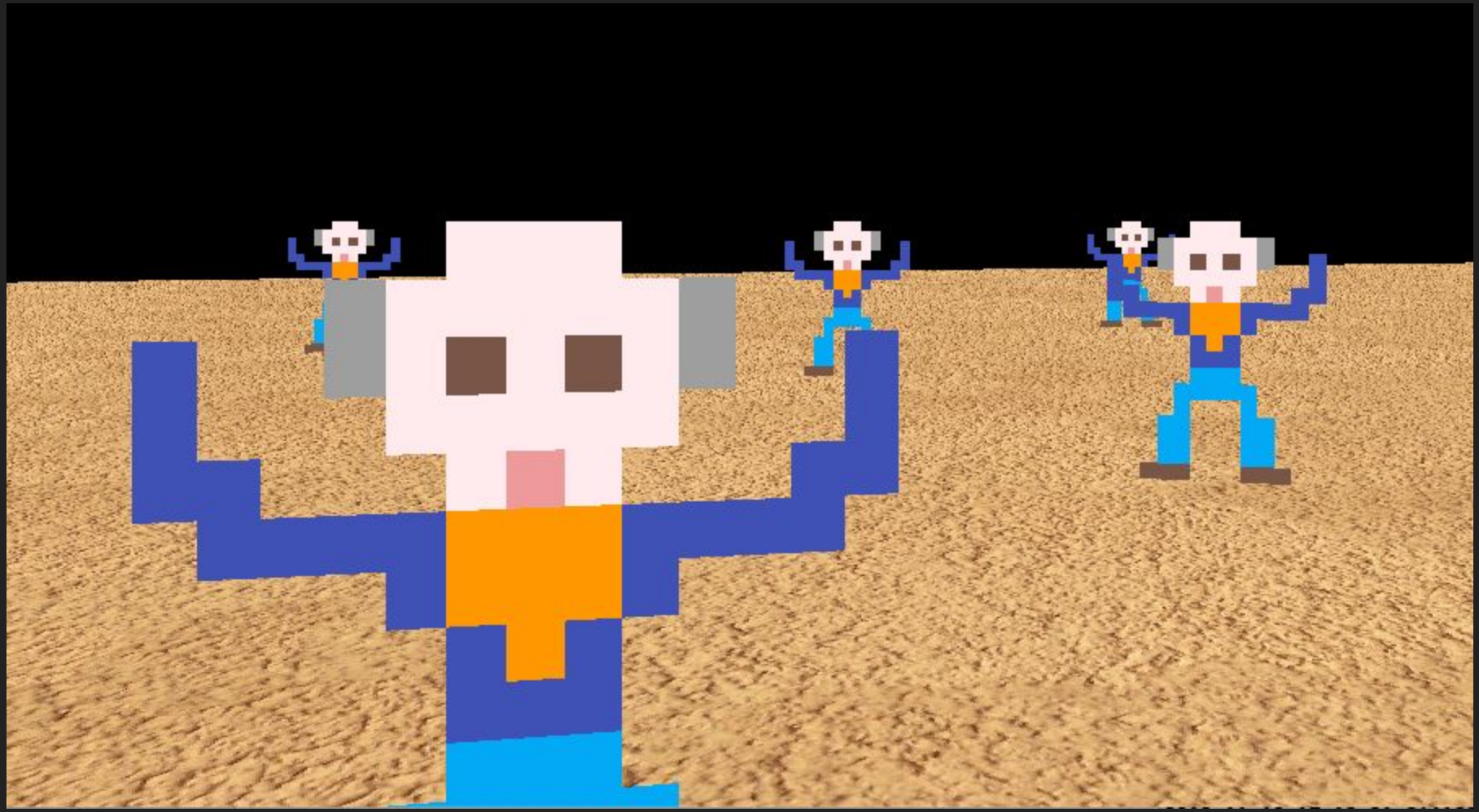


You might see an error caused by the depth buffer and alpha channel.



We need to update our fragment shader to ignore some pixels entirely.

```
uniform sampler2D diffuse;
varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D(diffuse, texCoordVar);
    if (gl_FragColor.a == 0.0) {
        discard;
    }
}
```



Let’s Code!

Update Entity.h and Entity.cpp

Update main.cpp
Add Enemies
Update
Render