Tank Escape

Ryley Goodine, Zheyuan Xu

# Introduction

The goal of the game is for the player to move the blue cube tank to the end of the maze without touching the enemy cube tank.

Challenges faced with implementing the game were in the creation of the maze, how the player will be colliding with the maze walls, or the enemy cube tank and in initializing projectiles.

~~Challenges where faced with deciding on the size of the maze and its complexity, as the objects defining each wall had their own texture~~

~~Movement of the player is done through a standard browser game mapped to the WASD key arrangement. Rotation of the first-person perspective camera is given to the Q and E key and rotates left and right respectively. Changing the camera is done through the F key, which cycles through a top-down view and a first-person perspective view.~~

# Methods

## Playing Field / Terrain

图片包含 游戏机, 乐高

描述已自动生成Plane Object

* Summary of functionality
  + The plane object is used as the ground space for the scene. It does not interact with the objects through collisions fields as a physics engine was not included within the game. The plane texture is bump mapped and linked to the object inside the scene.json file.

vec3 normVector = texture(uNormalTexture, oUV).xyz;

normVector = 2.0 \* normVector - 1.0;

float uNormalScale = 5.0;

normVector = uNormalScale \* normVector;

vec3 biTangent = cross(normal, oTangent);

mat3 TBN = mat3(oTangent, biTangent, normal);

normVector = normalize(TBN \* normVector);

* 图片包含 游戏机, 物体, 钟表, 球

  描述已自动生成Link to theory
  + The tangent vector is transformed and passed through in the vertex phase. Within the fragment phase, multiple calculations are performed to result in the TBN matrice.
  + A new normal vector is created from the specified texture and the linked UV values. The normal vector’s range is increased from 0-1 to -1-1.
  + The bitangent vector is calculated from the cross product of the normal vector and the tangent vector.
  + The TBN matrice is then calculated with input from the tangent, bitangent, and normal vectors.
  + The textured normal vector is then normalized with the TBN matrice, giving the new normal vector to be used within light calculations.
* Implementation details
  + Work from lab resources, particularly lab09 example03.js was used as the basis for the fragment shader code for the implementation of bump mapping within the scene.
  + The normal texture was provided as a default in Zach’s engine code.

"normalTexture": "defaultNorm.png",

## Main Player

A picture containing text

Description automatically generatedA picture containing blue

Description automatically generated

* Summary of functionality
  + The player object is a basic cube shape with colour calculated using the Bling-Phong model and diffuse texturing of the alien.jpg image.

vec3 textureColour = texture(uTexture, oUV).rgb;

diffuse = mix(diffuseVal, textureColour.rgb, 0.7);

* + The player is moved using the standard browser WASD key configuration.

this.player.translate(vec3.fromValues(+0.x, +0.0, +0.0)); // move left

this.player.translate(vec3.fromValues(-0.x, +0.0, +0.0)); // move right

this.player.translate(vec3.fromValues(+0.0, +0.0, +0.z)); // move forward

this.player.translate(vec3.fromValues(+0.0, +0.0, -0.z)); // move back

* Link to theory:
  + Movement of the player object on the scene is done through constant translation of the player objects current location.
  + The diffuse texture of the player object is calculated by mixing the uniform diffuseVal with the newly calculated texture value.
* Implementation details
  + Texture resources where used from the defaults provided in Zach’s engine, and the texture colour equation was taken from assignment06.

## Interacting Objects

* Walls

A picture containing text, outdoor, building material, blue

Description automatically generated

* + Summary of functionality
    - Collision detection for the player object, it prevents the player from moving in the direction of the wall.
    - Each wall object has a sphere collider connected to it that specifies the object name, and the affected radius.

createSphereCollider(object, radius, onCollide = null) {

        object.finish=0;

        object.stop=vec3.fromValues(0,0,0);

        object.collider = {

            type: "SPHERE",

            radius: radius,

            onCollide: onCollide ? onCollide : (otherObject) => {

                //console.log(`Collided with ${otherObject.name}`);

                //object.stop=vec3.fromValues(1,1,1);

            }

        };

        this.collidableObjects.push(object);

    }

const wallObject1 = getObject(this.state, "myWall5");

this.createSphereCollider(wallObject5, 0.4);

* + - With the stop vector variable declared during the creation of the collider, the player object will stop movement in each direction of WASD.
    - Such as for the keypress of “a”, as long as the player’s stop vector value in the x-coordinate is less then or equal to 0, the player will be able to move.

if (this.player.stop[0] <= 0) {

this.player.translate(vec3.fromValues(0.25, 0, 0));

* + Link to theory
    - Sphere collision involves testing if the distance between the two centers is larger then the sum of the two radius.

(distance < (object.collider.radius + otherObject.collider.radius))

* + Implementation details
    - Using the sphere collider code was taken from lab10, and player movement was modified
* Finish line

A picture containing text

Description automatically generated

* + Summary of functionality
    - Collision detection for the player object, it indicates that the player has reached the end of the game or has hit the enemy player. Sphere collision is used, and the values are stored within “finish” variable. When the player object collides with the object name “finish”, the player’s finish value is set to 1. From movement of the player, the player is able to traverse through the finish object

if ((otherObject.name === "finish") && (distance < (object.collider.radius + otherObject.collider.radius))) {

this.player.finish = 1;

return;

}

* + - Within the onUpdate function, the code constantly checks if the player’s finish value has changed into 1 or 2.

if (this.player.finish === 1) {

gameState.innerHTML = "YOU WON!";

gameState.style.color = "green";

}

* + - If the player has moved within distance of the “finish” object, then the HTML text will change its display to indicate the game is over and that the player has won.
  + Link to theory
    - Sphere collision is used to check if the player object has moved within the distance of the “finish” object.
  + Implementation details
    - The code from lab10 for the collision detection of the other objects was modified to check if the object has a particular name, and then perform a specific action.

## Non-Player Character

A picture containing underpants

Description automatically generated A picture containing furniture, seat, chair

Description automatically generated

* Summary of functionality
  + The enemy red tank, it moves in a fixed loop around its position.

if ((otherObject.name === "myNPC") && (distance < (object.collider.radius + otherObject.collider.radius))) {

this.player.finish = 2;

vec3.subtract(object.stop, otherMatrix, objectMatrix);

return;

}

* + Linking in the onUpdate function

if (this.player.finish === 2) {

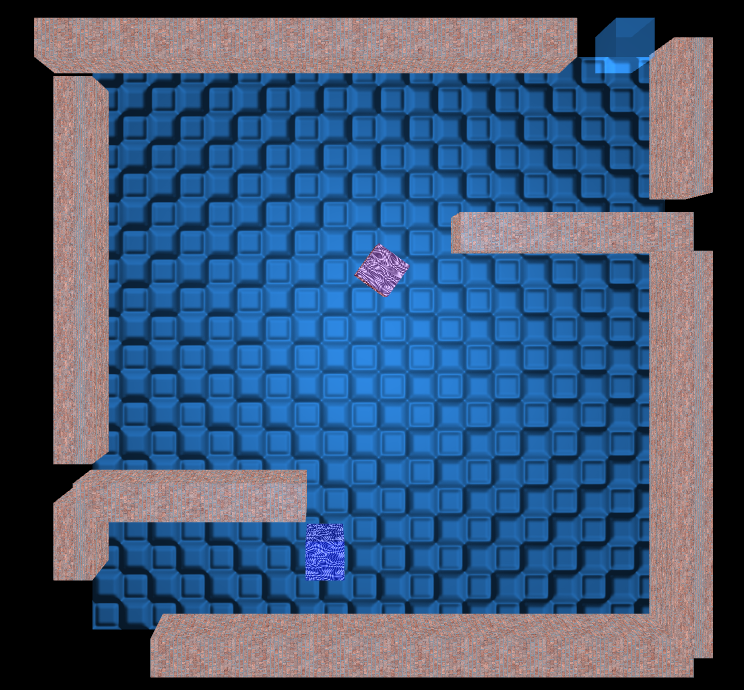
gameState.innerHTML = "YOU LOST!";

gameState.style.color = "red";

}

* Link to theory
  + Creation of the model with its texture, and movement around the scene at fixed locations.
* Implementation details
  + Texture image and colour values retrieved from the scene.js file. Its then linked into the fragment and vertice shader where the Blinn-Phong model equation will be used to calculate colour. Texture colour is calculated

## Change of View

 A picture containing blue, bedclothes

Description automatically generated

* Summary of functionality
  + Two types of views included; top-down and first-person.
* Link to theory
  + Perspective
  + Look-at vector
* Implementation details
  + Two cameras that are initialized within the main javascript file outside of the game.
  + Top down view and first person view are both initially set.
  + First person view follows the movement of the player within the scene.
  + Within drawscene function, camera positioning is indicated with a list index. This affords more flexibility within the game, and allows more camera views without having to include additional code.

## Additional Functionality

* Adding sound from projectiles
  + Summary of functionality
    - Sound file is loaded from the HTML with the audio tag and included with an id.
    - Within the game.js file inside the onPlay function, the specific id is retrieved and linked to a constant variable.
    - The sound is sped up to the maximum amount to allow for multiple firings.
    - Hitting the space key, which creates the projectile, will now also play the sound
  + Link to theory
  + Implementation details
* Adding basic timer
  + Summary of functionality
    - A built-in javascript world-timer without modification
    - It links to an id within the HTML file, and is constantly updated within the onUpdate function in game.js.
  + Link to theory
  + Implementation details
* Adding keymapping to html
  + Summary of functionality
    - The indicated keys are linked to their respective HTML tag within both the keyup and keydown listener.
  + Link to theory
  + Implementation details
* Music
  + Summary of functionality
    - Audio is added inside the HTML file using the audio tag.
    - Soundtrack’s are stored within a list, and the current index is stored within the game variable. The name of the song is linked inside a separate list and linked to an HTML header tag to display output to the user.
  + Link to theory
  + Implementation details
* Changing Lights on a counter
  + Summary of functionality
    - The global light is run through a progressive updater through each RGB colour.
  + Link to theory
  + Implementation details

# Analysis and discussion

* Summarize what we did

We learned through experiment 10 many techniques for using zach's engine including and not limited to apply a diffuse texture, object movement rotation, scene construction, collision volume calculation, etc.

* What we learned from this project

WebGL 3D graphics processing, including position determination, light position processing and light color transformation, object movement rotation and collision processing, object texture, etc.

* What worked and what didn’t (why)

We tried to design a new rectangular collision detection, but for time reasons we did not design a suitable rectangular collision detection, so we chose to follow the spherical collision one, using multiple cubes to form a whole wall, to achieve the same effect of preventing the tank from moving through the wall.

* What would we do to continue/improve

Projectile collision effects and collision sound effects

Better rectangular collision determination

Credits:

Zheyuan XU: Game concept discussion, scene building, protagonist movement as well as camera rotation, Implement bump mapping, so that the plane to achieve the effect of 3d, NPC as well as wall collision adding testing, NPC movement, testing, and short video demo recording and ppt writing.

Ryley Goodine: Game concept discussion, scene building, player movement and initialization of two camera views, fragment shader blinn-phong equation and vertex shader normal mapping code addition. Projectile creation of bullets including its destruction after defined distance from player or number of existing bullets in scene reaches arbitrary amount. Cyclical global light change through iterative addition and subtraction of fixed amount. Implementation of HTML additions including music, sound form bullet activation, dynamic updates in html table on keypresses­