ポートフォリオ

総合学園ヒューマンアカデミー横浜校
ゲームカレッジ　プログラマー専攻
小川　彩織

目次

プロフィール

氏名　：小川　彩織
性別　：女
生年月日：1999年10月29日
希望職種：プログラマー

スキルシート

言語	
開発環境	Unity / Direct3D Microsoft Visual Studio 2022
ツール	Git(Tortoise git, GitHub)

自己PR

C/C++、C#、Unityでのゲーム開発のスキルを保持しています。その中でも特に、Unityでのチーム制作に力を入れて取り組んでいます。

パズルゲームの制作では、少人数であったことに加えメンバーにCGがいなかったため、企画者と話し合いを重ねたり、CGの先生に相談したりして、より良いゲームに仕上がるように調整しました。

将来はこの経験を活かして、周囲の人と協力しながら操作性の良いゲームを制作していきたいと考えています。

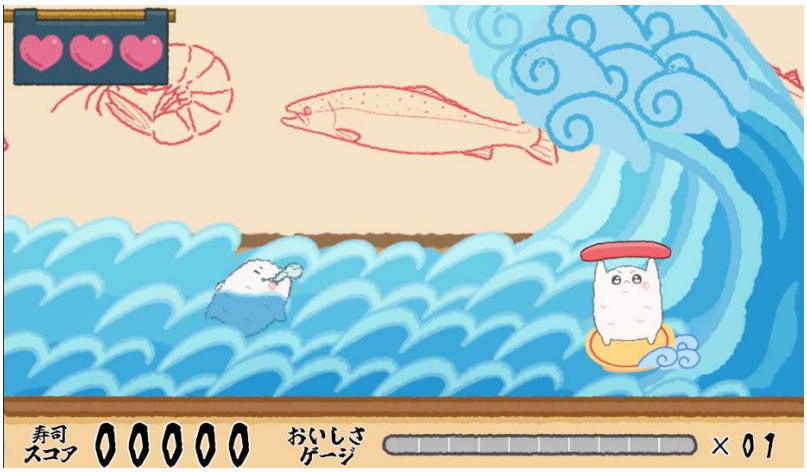
チーム制作

寿司サーファーズ

開発環境	Unity 2022.3.42f1
使用ツール	Visual Studio 2022
言語	C#
ジャンル	サーフィンアクション
動作環境	PC
制作期間	3か月
制作人数	プランナー3人 プログラマー3人 CGデザイナー3人

ゲーム概要

回転寿司を流れる波に乗ってお客さんのところまで到達しましょう。
途中で出てくるシャリや手を避けてゴールを目指します。



シャリを避ける距離によって加算されるスコア、エフェクトが変化するため、判定用のポイントを設定し、距離を取得しました。

GaugeChecker.cs

```
// 交差判定用のポイントを設定
var centerPoint = transform.TransformPoint(center);
// 交差を判定
var hit = Physics2D.OverlapBox(centerPoint, size, transform.eulerAngles.z, targetLayers);
if(hit != null)
{
    // シャリが侵入したらtrue
    IsCasted = true;
    Casted = hit.gameObject;
}
else
{
    // シャリが侵入していないときにfalse
    IsCasted = false;
    Casted = null;
}
```

Player.cs

```
float distance = Vector3.Distance(gauge.Casted.transform.position, transform.position);
// 距離に応じてスコア、おいしさゲージを上昇
if (distance > 2)
{
    // スコア加算
    StageScene.Instance.AddGoodTurn(100);
    // おいしさゲージのポイントを加算
    StageScene.Instance.AddGauge(1);
    // エフェクト再生
    great.Play();
    // SE再生
    audioSource.PlayOneShot(soundOnGreat);
}
else if (distance <= 2)
{
    // スコア加算
    StageScene.Instance.AddExcelentTurn(300);
    // おいしさゲージのポイントを加算
    StageScene.Instance.AddGauge(3);
    // エフェクト再生
    excellent.Play();
    // SE再生
    audioSource.PlayOneShot(soundOnExcellent);
}
```

2Dで奥行き感がでるよう、当たり判定の調整をしました。
別の障害物やアイテムに当たらないように条件付けをしました。

Player.cs

```
// シャリか手に当たった時に呼び出される
Unity メッセージ|0 個の参照
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Hand") ||
        collision.gameObject.CompareTag("Syari") && lane == 0 ||
        collision.gameObject.CompareTag("Syari1") && lane == 1 ||
        collision.gameObject.CompareTag("Syari2") && lane == 2)
```

障害物やアイテムの配置について、それぞれのレーンごとにタグの管理をして、OrderInLayerが変わるようにしました。

objectMove.cs

```
private void Awake()
{

    // OrderInLayerをレーンごとに変更
    spriteRenderer = syari.GetComponent<SpriteRenderer>();
    if(gameObject.tag == ("Syari"))
    {
        spriteRenderer.sortingOrder = -10;
    }
    else if(gameObject.tag == ("Syari1"))
    {
        spriteRenderer.sortingOrder = -110;
    }
    else if(gameObject.tag == ("Syari2"))
    {
        spriteRenderer.sortingOrder = -210;
    }
}
```

hpについて、配列を使用し、ダメージ判定のときに1つずつ非表示するようにしました。
Coroutine、列挙体を使用し、連続してダメージを受けることのないようにしました。

Player.cs

```
// hpを指定します。
[SerializeField] private GameObject[] hp = new GameObject[3];
```

```
// ライフ数の変数を定義
public int hpPoint = 3;
```

```
if (currentState == PlayerState.Idle)
{
    // ダメージを受けたらhpを1つ減らす
    hp[hpPoint - 1].SetActive(false);
    hpPoint--;
    // hpが0のとき
    if (hpPoint == 0)
    {
        currentState = PlayerState.Fall;
        // 落下アニメーション再生
        animator1.SetTrigger("Fall");
    }
    // hpが0でないとき
    else if (hpPoint != 0)
    {
        // StateをDamageに変更
        currentState = PlayerState.Damage;
        // ダメージアニメーション再生
        animator1.SetTrigger("Damage");
        StartCoroutine("OnDamage");
    }
```

```
private IEnumerator OnDamage()
{
    // SE再生
    audioSource.PlayOneShot(soundOnDamage);
    // 1秒後にStateをIdleに戻す
    yield return new WaitForSeconds(1);
    currentState = PlayerState.Idle;
}
```

```
// プレイヤーの状態を管理
22 個の参照
enum PlayerState
{
    // 待機中
    Idle,
    // ターン中
    Turn,
    // ダメージ判定中
    Damage,
    // 無敵判定中
    Muteki,
    // 落下中
    Fall,
    // Clear
    Clear,
}
```

手が近づいたことがわかるよう、一定の位置を越えたら手の影を徐々に大きく、濃くするようにしました。
一定以上の大きさになり手が当たったらオブジェクトを破棄するようにしました。

Shadow.cs

```
if (transform.position.x > 2)
{
    // 右への移動をやめる
    rb.velocity = Vector2.right * 0;
    // 徐々に大きくする
    scale += 0.005f;
    transform.localScale = new Vector3(scale, scale, scale);
    // 徐々に濃くする
    color += 0.02f;
    spriteRenderer.color = new Color(1, 1, 1, color);
}
```

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.CompareTag("Hand") && transform.localScale.x > 1.0f)
    {
        // 手に当たったらオブジェクトを破棄
        Destroy(gameObject);
    }
}
```

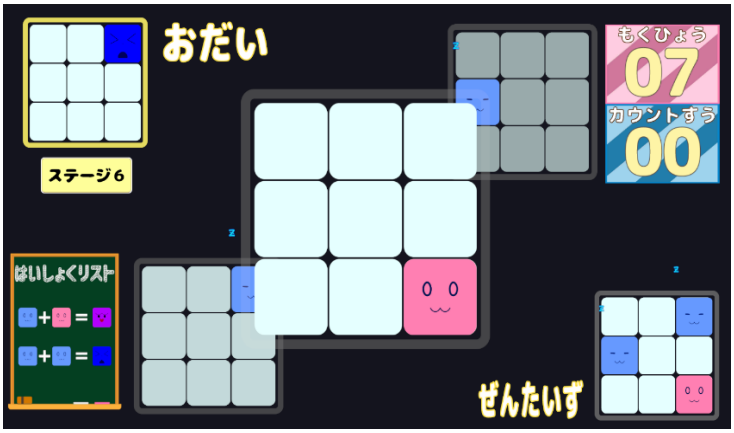
チーム制作

COLOR Layers

開発環境	Unity 6000.0.36f1
使用ツール	Visual Studio 2022
言語	C#
ジャンル	パズルゲーム
動作環境	PC
制作期間	3か月
制作人数	プランナー3人 プログラマー2人

ゲーム概要

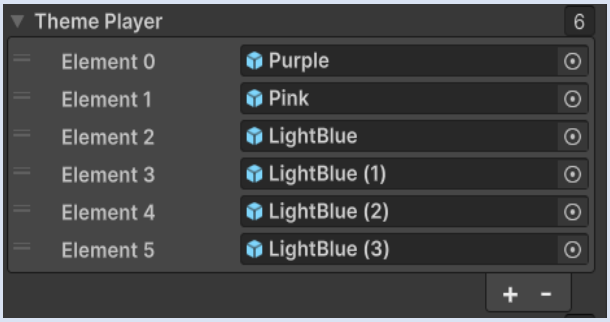
キャラクターを動かし、お題と同じにします。各レイヤーは重なっている状態で、全体図は上から見た状態です。キャラクターの位置が重なることで、色を変化させることができます。



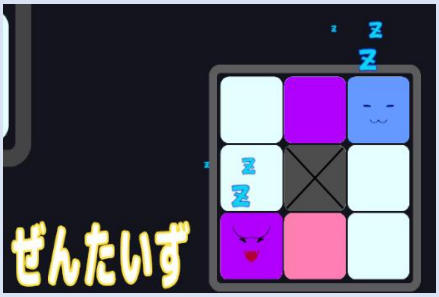
階層構造

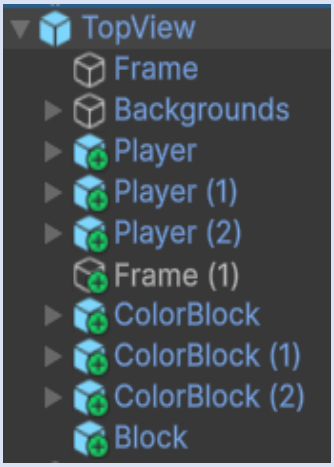
[おだい]

PlayerManager.cs



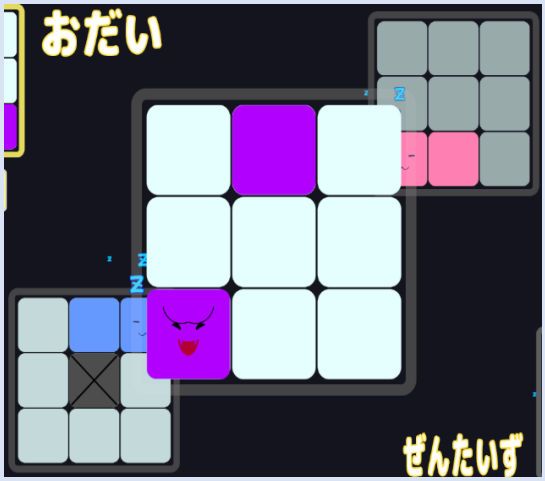
[全体図（レイヤーを重ねて上から見た状態）]

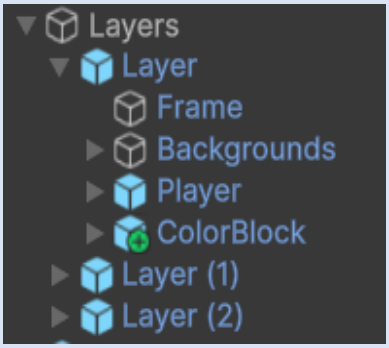




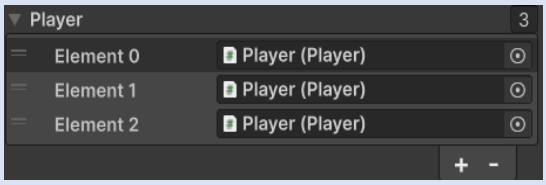
PlayerManager.cs

[レイヤー]





PlayerManager.cs



キャラクターの移動は、InputManagerを使用しました。一番上にあるレイヤーのPlayerのみ移動するため、それぞれのPlayerをPlayerCount変数で管理しました。Player.csでplayerCountを管理し、PlayerManager.csにて移動できるようにしました。StageごとにPlayerの数が異なるため、PlayerManagerにてPlayerを配列で指定しました。操作するPlayerと全体図のPlayerが同じ動きをするように対応するタイルを指定し、それぞれの移動を呼び出しました。

PlayerManager.cs

```
// MoveActionにて呼び出される
0 個の参照
public void OnMove(InputAction.CallbackContext context)
{
    var axis = context.ReadValue<Vector2>();

    if (context.started && !StageScene.Instance.isPaused && isTileMoved)
    {
        // 右への移動
        if (axis.x > 0)
        {
            for (int i = 0; i < player.Length; i++)
            {
                if (player[i].playerCount == 0 && player[i].positionHorizontal < player[i].maxHorizontal)
                {
```

```
if (isMoved)
{
    // Playerの移動
    player[i].transform.Translate(distance, 0, 0);
    topViewPlayer[i].transform.Translate(distance, 0, 0);
    player[i].positionHorizontal++;
    topViewPlayer[i].positionHorizontal++;
    // 移動数カウント
    StageScene.Instance.AddCount(1);
    // 移動時のSE再生
    audioSource.Play();
}
```

playerCountは、タイルが移動した際に変化するようにしました。

```
public void PlayerCount()
{
    // タイル変更時、何層目かを判断するCount、OrderInLayerを変更
    // 1層目
    if (tile.tileCount == 0)
    {
        playerState = SceneState.Eye;
        playerCount = 0;
```

```
// 2層目
else if (tile.tileCount == 1)
{
    playerState = SceneState.Idle;
    playerCount = 1;
```

```
// 3層目
else if (tile.tileCount == 2)
{
    playerState = SceneState.Idle;
    playerCount = 2;
```

タイルの移動は、列挙型を使用し、キーを一度押したら指定の位置まで移動するようにしました。移動が終わったタイミングで、State、tileCount、playerCountを変更しました。

```
enum TileState
{
    // 静止状態
    Idle,
    // 左へ移動中
    MoveLeft,
    // 右へ移動中
    MoveRight,
}
TileState tileState = TileState.Idle;
```

```
void Update()
{
    switch (tileState)
    {
        case TileState.Idle:
            playerManager.isTileMoved = true;
            break;
        case TileState.MoveLeft:
            playerManager.isTileMoved = false;
            MoveLeft();
            break;
        case TileState.MoveRight:
            playerManager.isTileMoved = false;
            MoveRight();
            break;
    }
}
```

```
//OnMoveActionにて呼び出される(Q/Rキーまたはコントローラーの L/Rキー)
0 個の参照
public void OnMove(InputAction.CallbackContext context)
{
    var axis = context.ReadValue<Vector2>();

    if (tileState == TileState.Idle && !StageScene.Instance.isPaused && context.started)
    {
        if(tile.Length > 1)
        {
            // タイル移動時にSE再生
            audioSource.Play();
        }
        if (axis.x < 0)
        {
            tileState = TileState.MoveLeft;
        }
        else if (axis.x > 0)
        {
            tileState = TileState.MoveRight;
        }
    }
}
```

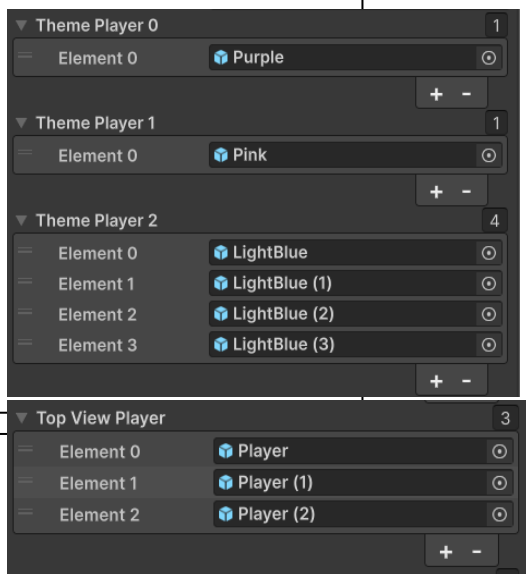
```
public void MoveLeft()
{
    // タイル数が1つの時
    if(tiles == 0)
    {
        tileState = TileState.Idle;
    }
    // タイル数が2つの時
    else if(tiles == 1)
    {
        // tile[0]が上の場合
        if (tile[0].tileCount == 0)
        {
            // 指定したtargetの位置まで移動
            tile[0].transform.position = Vector3.MoveTowards(tile[0].transform.position, target[1].transform.position,
                speed * Time.deltaTime);
            tile[1].transform.position = Vector3.MoveTowards(tile[1].transform.position, target[0].transform.position,
                speed * Time.deltaTime);

            if (tile[0].transform.position == target[1].transform.position)
            {
                // Stateを待機中に変更
                tileState = TileState.Idle;
                //tileCountを変更
                tile[0].tileCount = 1;
                tile[1].tileCount = 0;
                // TileのOrderInLayer、透明度変更
                background[0].Color();
                background[1].Color();
```

```
                for(int i = 0; i < playerManager.player.Length; i++)
                {
                    // PlayerCount、ColorBlockCount変更
                    player[i].PlayerCount();
                    topViewPlayer[i].PlayerCount();
                    // 全体図のPlayerのsleepEffectは手前に見えるように設定
                    topViewPlayer[i].sleepRenderer.sortingOrder = 1;
                }
```

クリアー判定では、お題、全体図のPlayerをそれぞれ指定し、位置と色が一致しているかどうかを判定しました。クリアーパターンの数がステージごとに異なるため、for文を使用して対応しました。

```
// お題のPlayerをそれぞれ指定
[SerializeField] private GameObject[] themePlayer0 = new GameObject[1];
[SerializeField] private GameObject[] themePlayer1 = new GameObject[1];
[SerializeField] private GameObject[] themePlayer2 = new GameObject[1];
// 全体図のPlayerをそれぞれ指定
[SerializeField] private GameObject[] topViewPlayer = new GameObject[1];
```

```
for (int i = 0; i < topViewPlayer.Length; i++)
{
    // 位置、色が一致しているとき(1つ目)
    if (themePlayer0[0].transform.localPosition == topViewPlayer[i].transform.localPosition &&
        themeSprite0[0].color == topViewSprite[i].color)
    {
        if(topViewPlayer.Length == 1)
        {
            if(topViewColorBlock.Length == 0)
            {
                StageScene.Instance.GameClear();
            }
            else
            {
                ColorBlock();
            }
        }
        else
        {
            Players2();
        }
```




```
private void Players2()
{
    // クリアーパターンの数だけ繰り返す
    for(int i = 0; i < themePlayer1.Length; i++)
    {
        for(int j = 0; j < topViewPlayer.Length; j++)
        {
            // 2つ目の位置、色を判定
            if (themePlayer1[i].transform.localPosition == topViewPlayer[j].transform.localPosition &&
                themeSprite1[i].color == topViewSprite[j].color)
            {
                if(topViewPlayer.Length == 2)
                {
                    // カラーブロックがないとき
                    if (topViewColorBlock.Length == 0)
                    {
                        StageScene.Instance.GameClear();
                    }
```

個人制作

ButtleForTheTreasure

開発環境	Unity 2022.3.42f1
使用ツール	Visual Studio 2022
言語	C#
ジャンル	3Dアクション
動作環境	PC
制作期間	1か月
制作人数	1人

ゲーム概要

島に散らばった宝石を集めましょう。
敵はジャンプで倒すことができます。
5つの宝石を集めるとクリアです。

移動の際、位置を更新し、Quaternionを使用して向きが変わるようにしました。

Player.cs

```
// 移動量
Vector3 diff = transform.position - latestPosition;
// 静止状態では回転しない
if (diff == Vector3.zero)
    return;
// 移動前の位置を更新
latestPosition = transform.position;
// 移動量が0.01以上の時に向きを変える
if (diff.magnitude >= 0.01f)
{
    transform.rotation = Quaternion.LookRotation(diff, Vector3.up);
}
```

敵をジャンプにて踏んだ後、少し上へ上がるようにしました。

Player.cs

```
else if(other.gameObject.tag == "Enemy")
{
    // 敵を踏んだら倒す
    if (transform.position.y > other.gameObject.transform.position.y + 0.5f)
    {
        // 踏んだ瞬間に少し上へ上がる
        rigidbody.velocity = Vector3.up * 3.5f;
        // SE再生
        effectAudio.PlayOneShot(soundOnAttack);
        // 点数追加
        StageScene.Instance.AddScore(100);
        // オブジェクト破棄
        Destroy(other.gameObject);
    }
```

敵とプレイヤーの距離を取得し、近づくと敵がプレイヤーを追ってくるようにしました。敵が追いかける際、移動方向を向くようにしました。

Enemy.cs

```
// プレイヤーとの距離を取得
distance = Vector3.Distance(transform.position, player.transform.position);

if(distance < 7)
{
    // プレイヤーへの方向ベクトルを計算
    Vector3 direction = player.position - transform.position;
    // 移動方向を決定
    transform.rotation = Quaternion.LookRotation(-direction, Vector3.up);
    // オブジェクトを移動
    transform.position += direction * speed * Time.deltaTime;
}
```

プレイヤーの動きに合わせてカメラが移動するようにしました。

ChaseCamera.cs

```
void Update()
{
    // 追尾
    var position = transform.position;
    position.x = target.position.x + offset.x;
    position.y = target.position.y + offset.y;
    position.z = target.position.z + offset.z;
    transform.position = position;
}
```

アイテムを取得した際に得点が追加されるようにしました。それをPlayerPrefsにて保存し、クリア画面で表示しました。

StageScene.cs

```
// アイテム取得時の点数を取得
3 個の参照
public int ScoreCount { get => scoreCount; private set => scoreCount = value; }
int scoreCount = 0;

2 個の参照
public void AddScore(int value)
{
    ScoreCount += value;
}
```

Item.cs

```
private void OnTriggerEnter(Collider other)
{
    // プレイヤーに当たったら破棄
    if (other.gameObject.tag == "Player")
    {
        // 100点追加
        StageScene.Instance.AddScore(100);
        Destroy(gameObject);
    }
}
```

StageScene.cs

```
public void GameClear()
{
    gameState = SceneState.GameClear;
    // 最終スコアを保存
    PlayerPrefs.SetInt("TotalScoreCount", ScoreCount);
    // Clearシーンを読み込む
    StartCoroutine(OnLoadScene("Clear"));
}
```

TotalScoreUI.cs

```
void Start()
{
    scoreText = GetComponent<TextMeshProUGUI>();
    // スコアを取得して表示
    var totalScoreCount = PlayerPrefs.GetInt("TotalScoreCount", 0);
    scoreText.text = totalScoreCount.ToString();
}
```

最終スコアに応じて、クリア画面で表示する星の数が変わるようにしました。

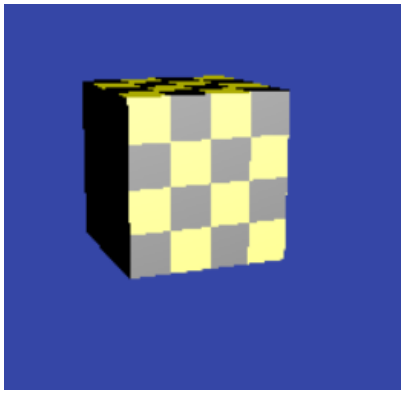
RankUI.cs

```
void Start()
{
    // ステージシーンのスコアを取得
    var totalScoreCount = PlayerPrefs.GetInt("TotalScoreCount", 0);

    // スコアによってランク変更
    if (totalScoreCount >= 500)
    {
        star[0].SetActive(true);
        if(totalScoreCount >= 600)
        {
            star[1].SetActive(true);
            if(totalScoreCount >= 800)
            {
                star[2].SetActive(true);
            }
        }
    }
}
```

Direct3Dゲームエンジン開発

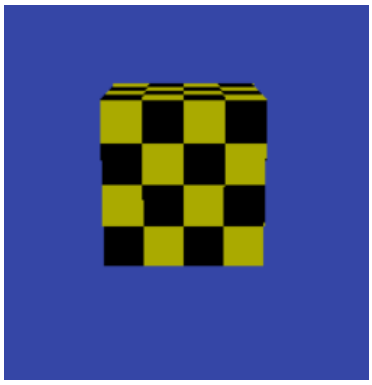
使用ツール	Visual Studio 2022
言語	C++、HLSL
動作環境	PC

HLSL言語によるプログラミング習得として、手始めに基本的なPhongシェーディングを実装しています。

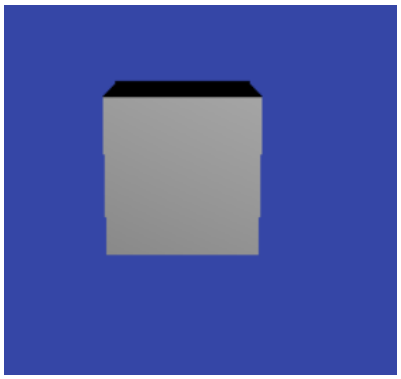


Phongシェーディングでキューブの描写をしました。
DiffuseとSpecularをそれぞれ実装しました。

拡散反射(Diffuse)



鏡面反射(Specular)



Diffuse + Specular

BasicPixelShader.hlslでピクセルシェーダーでのPhong反射モデルのプログラムを実装しました。

```
// ライトへのベクトル
float3 light = normalize(LightPosition.xyz - LightPosition.w * input.worldPosition.xyz);

// 拡散反射、lightとworldNormalを使って計算する
float diffuse = max(dot(light, input.worldNormal), 0.0f);
float3 diffuseColor = diffuse * MaterialDiffuse.rgb;

// 鏡面反射
float3 reflect = 2 * input.worldNormal * dot(input.worldNormal, light) - light;
// 視線方向(ViewPosition:視線の位置、input.worldPosition:反射光の位置
float3 viewDir = normalize(ViewPosition - input.worldPosition).xyz;
float specular = pow(saturate(dot(reflect, viewDir)), MaterialSpecularPower);
float3 specularColor = specular * MaterialSpecularColor.rgb;

float4 texel = diffuseTexture.Sample(diffuseTextureSampler, input.texCoord);

return float4(texel.rgb * diffuseColor + specularColor, MaterialDiffuse.a * texel.a);
```

Game.cppで、DiffuseとSpecularを定数バッファーを通してシェーダーに入力しました。

```
// マテリアル
if (GetAsyncKeyState('D')) {
    constantBufferPerFrame.materialDiffuse = DirectX::XMFLOAT4(1, 1, 1, 1);
}
else {
    constantBufferPerFrame.materialDiffuse = DirectX::XMFLOAT4(0, 0, 0, 1);
}
if (GetAsyncKeyState('S')) {
    constantBufferPerFrame.materialSpecularColor = DirectX::XMFLOAT3(1, 1, 1);
    constantBufferPerFrame.materialSpecularPower = 1;
}
else {
    constantBufferPerFrame.materialSpecularColor = DirectX::XMFLOAT3(0, 0, 0);
}
```