PORTFOLIO

KENJI KATO

PROFILE

名前	加藤　謙治
所属	仙台デザイン＆テクノロジー専門学校
希望職種	プログラマー
長所	慎重なところ

自己PR

私は、読みやすいコードを書くことで、プログラム効率が上がると考えています。
後ほどコードを読んだ際に修正がしやすいことや、
他の人が読んでも、変更を加えやすいことから読みやすいコード書くように心がけています。

業界でやりたいこと

プレイヤーの操作感を徹底的によくしたい

使用可能なソフト・言語

- Unity（5本以上のゲームを作成）
- Unreal Engine（基礎的なツールのみ）
- DxLib（3本以上のゲームを作成）
- blender（基礎的なツールのみ）
- SourceTree
- C#（unityで使う程度）
- C++（DxLibで使う程度）
- PHP（学校の講義で触ったときある程度）

プロジェクトデータ＆ビルドデータ

https://87.gigafile.nu/1226-hafa540c650ec85fc6c3c26a845dc5dfa

ダウンロードキー：8978

プレイ動画

https://drive.google.com/file/d/1Q_9FnIUncfGjPo4fWe7bp7Jg70JU1lXi/view?usp=drive_link

概要

ジャンル：3Dアクション
プラットフォーム：PC
開発エンジン：DxLib
言語：C++
人数：1人
制作期間：2月上旬～3月下旬
&9月5日～9月7日

タイトル：MutantAssault

ゲーム内容

プレイヤーキャラを動かして出てくる敵を倒していくゲームの予定です。
回避やパリィなどを使いこなし敵のすきを見て攻撃をし、敵を倒していきます。

コード1

```cpp
class StateBase
{
public:
    StateBase() : isNext(false)
    {
    };

    // アップデートの前に呼ばれる
    virtual void Start() = 0;

    // アップデート
    virtual void Update() = 0;

    // 状態が変わるときの処理
    virtual void OnChangeEvent() = 0;

    // 切り替える条件を取得
    bool IsNext()const { return isNext; }

    //次のStateに移動できるか
    bool CanNextState() {
        if (IsNext()) {
            OnChangeEvent();
            return true;
        }
        return false;
    }

    //次に行く状態を変更
    void chengeNextState(std::string name) {
        nextRegisterName = name;
    }

    //次に行く状態を取得
    std::string getNextName() {
        return nextRegisterName;
    }
protected:
    // 次に行くための条件
    bool isNext;

    //次のStateの名前
    std::string nextRegisterName;

};
```

```cpp
// 状態管理
class StateMachine
{
public:
    StateMachine() :
        nowState()
    {

    }

    void Update()
    {
        nowState->Update();
        if (nowState->CanNextState())
        {
            auto it = states.find(nowState->getNextName());
            if (it == states.end()) return;

            nowState = it->second;
            nowState->Start();
        }
    }

    // 登録する
    void Register(const std::string& name, const std::shared_ptr<StateBase> state)
    {
        states.insert(std::make_pair(name, state));
        std::cout << "StateMachine : " << name << "を登録" << std::endl;
    }

    // 最初から始める状態を設定
    void SetStartState(const std::string& registerName)
    {
        auto it = states.find(registerName);
        if (it == states.end()) return;

        std::cout << "StateMachine : " << it->first << "をスタート状態に設定" << std::endl;

        nowState = it->second;
        nowState->Start();

    }
```

改善した点

プレイヤー操作書くときにboolをたくさん使い条件分けして書いていました。しかし、アクションを追加して行くにつれ条件が多く新しいアクションを追加するのが面倒くさくなりました。そこで、企業の方に質問をしてStateMachineという物があることを知りました。
StateMachineについて調べプログラムに組み込むと、プレイヤーのアクションの追加がとても楽になり、コードも読みやすくなりました。

コード2

BehaviorTree.h

project

```cpp
#pragma once
#include "DxLib.h"
#include <vector>
#include <memory>

//状態
enum class NodeStatus { Success, Failure, Running };

//基底ノード
class BTNode {
public:
    virtual NodeStatus Tick() = 0;
    virtual ~BTNode() {}
};

//セレクター
class Selector : public BTNode {
    std::vector<std::unique_ptr<BTNode>> children;
public:
    void AddChild(std::unique_ptr<BTNode> node) {
        children.push_back(std::move(node));
    }

    NodeStatus Tick() override {
        for (auto& child : children) {
            NodeStatus status = child->Tick();
            if (status != NodeStatus::Failure) {
                return status;
            }
        }
        return NodeStatus::Failure;
    }
};
```

```cpp
//シーケンス
class Sequence : public BTNode {
    std::vector<std::unique_ptr<BTNode>> children;
public:
    void AddChild(std::unique_ptr<BTNode> node) {
        children.push_back(std::move(node));
    }

    NodeStatus Tick() override {
        for (auto& child : children) {
            NodeStatus status = child->Tick();
            if (status != NodeStatus::Success) {
                return status;
            }
        }
        return NodeStatus::Success;
    }
};
```

改善した点

最初は、敵のAIの行動をswitch文を使って実装していましたが、ゲームの進行に伴ってAIの行動が複雑化し、switch文が長く、管理が難しくなってきたため、ビヘイビアツリーに置き換えました。これにより、AIの挙動がより柔軟で拡張性のあるものになりました。結果として、AIの設計がシンプルで視覚的にわかりやすくなり、コードの可読性と保守性が大幅に向上しました。また、ゲームが進行する中で、より複雑なAIの挙動を追加する際もスムーズに対応できるようになりました。

コード3

```
56          //タイムライン
57          CsvReader* csv = new CsvReader(folder + filename[i] + ".csv");
58          for (int line = 0; line < csv->GetLines(); line++) {
59              TimeInfo ti;
60              ti.time = csv->GetFloat(line, 0);
61              ti.command = csv->GetString(line, 1);
62              ti.filename = csv->GetString(line, 2);
63              info[i].timeline.push_back(ti);
64          }
65          delete csv;
66      }
```

	A	B	C	D
1	31	AttackStart		
2	45	NextAttack		
3	60	AttackEnd		
4				
5				

```
199     /// <summary>
200     /// アニメーションを更新する
201     /// タイムラインチェック
202     /// </summary>
203     void Player::playAnimation()
204     {
205         float prevFrame = animation->GetCurrentFrame();
206         animation->Update();
207         float curFrame = animation->GetCurrentFrame();
208
209         std::string folder = "data/sound/SE/";
210         for (TimeInfo t : info[animID].timeline) {
211             if (t.time > prevFrame && curFrame >= t.time) {
212                 if (t.command == "SE") {
213                     PlaySound(
214                         "data/sound/SE/Paladin/walk.mp3",
215                         DX_PLAYTYPE_BACK);
216                 }
217                 if (t.command == "AttackStart") {
218                     canAttack = true;
219                 }
220                 if (t.command == "AttackEnd") {
221                     canAttack = false;
222                 }
223                 if (t.command == "NextAttack") {
224                     canNextAttck = true;
225                 }
226                 if (t.command == "AvoidanceStart") {
227                     isAvoidance = true;
228                 }
229                 if (t.command == "AvoidanceEnd") {
230                     isAvoidance = false;
231                 }
232             }
233         }
234     }
```

工夫した点

タイムラインを作ることで、攻撃の判定を始めるタイミングや次のアクションができるタイミング、SEを出すタイミングなどを簡単に変更できるようにしました。
また、アニメーションごとにCSVを用意することでタイムラインの管理を簡単にできるようにしました。

コード4

```
99      if (LockOn) {
100         if (!LockEnd_Y) {
101             //Y軸の回転
102             VECTOR vec1 = direction;
103             VECTOR vec2 = mu->Position();
104
105             VECTOR moveVec = vec2 - position;
106             moveVec.y = 0;
107             float rotateTarget = atan2(moveVec.x, moveVec.z);
108             float diff = rotateTarget - rotation.y;
109             while (diff > DX_PI)
110             {
111                 diff -= DX_PI * 2.0f;
112             }
113             while (diff < -DX_PI)
114             {
115                 diff += DX_PI * 2.0f;
116             }
117             //3.角度差が一定角度（π/4）以内であれば、rotation.y = rotateTarget
118             float limit = DX_PI / 20.0f;
119             if (diff < limit && diff > -limit)
120             {
121                 rotation.y = rotateTarget;
122                 //LockOn = false;
123                 LockEnd_Y = true;
124             }
125             else {
126                 //4. 値が+であれば右回転、値が-であれば左回転
127                 if (diff > 0) {
128                     rotation.y += limit;
129                 }
130                 else {
131                     rotation.y -= limit;
132                 }
133             }
134         }
135
136         if (!LockEnd_X) {
137
138             if (rotation.x > DegToRad(60.0f)) {
139                 rotation.x = DegToRad(59.0f);
140             }
141             if (rotation.x < DegToRad(-60.0f)) {
```

```
140         }
141         if (rotation.x < DegToRad(-60.0f)) {
142             rotation.x = DegToRad(-59.0f);
143         }
144
145         //X軸の移動
146         VECTOR vec3 = direction;
147         VECTOR vec4 = mu->Position();
148         //vec4.y = vec4.y+150;
149
150         VECTOR moveVecX = vec4 - position;
151         moveVecX.y = 0;
152         moveVecX.z = 0;
153         float rotateTargetX = atan2(moveVecX.y, moveVecX.z);
154         float diffX = rotateTargetX-rotation.x;
155         while (diffX > DX_PI)
156         {
157             diffX -= DX_PI * 2.0f;
158         }
159         while (diffX < -DX_PI)
160         {
161             diffX += DX_PI * 2.0f;
162         }
163         //3.角度差が一定角度（π/4）以内であれば、rotation.y = rotateTarget
164         float limitX = DX_PI / 18.0f;
165         if (rotation.x > DegToRad(60.0f)) {
166             LockEnd_X = true;
167             return;
168         }
169         if (rotation.x < DegToRad(-60.0f)) {
170             LockEnd_X = true;
171             return;
172         }
173         if (diffX < limitX && diffX > -limitX)
174         {
175             rotation.x = rotateTargetX;
176             //LockOn = false;
177             LockEnd_X = true;
178         }
179         else {
180             //4. 値が+であれば右回転、値が-であれば左回転
181             if (diffX > 0) {
182                 rotation.x += limitX;
183                 if (rotation.x > DegToRad(60.0f))
184                     LockEnd_X = true;
185                     //rotation.x = 0;
```

反省点

プレイヤーから最も近い敵キャラクターに自動でカメラが向くロックオン機能をで実装しました。
プレイヤーが入力したタイミングで、敵リストから最も近い対象を選び、カメラの回転（X軸・Y軸）を補間して滑らかに移動させます。
回転の範囲制限や、一定角度差以内で停止する処理も導入し、自然なカメラ挙動とプレイヤー操作の快適さを両立しています

コード5

Enemy.h

```
project                                                    Enemy
#pragma once
#include "../Library/GameObject.h"
#include "animation.h"
#include "../Library/csvReader.h"
#include "Player.h"
#include "Stage.h"
#include "BehaviorTree.h"

/// <summary>
/// 敵の基底クラス
/// </summary>
class Enemy : public GameObject {
public:
    Enemy() {};
    virtual ~Enemy() {};
    virtual void TakeDamage(int damage) = 0;
    virtual void Attack() = 0;

    bool IsDeath()
    {
        return isDeath;
    }
    VECTOR Position()
    {
        return position;
    }
```

```
    bool HitCheck(VECTOR playerPos, VECTOR weaponLine1, VECTOR weaponLine2)
    {
        MV1SetupCollInfo(hModel, -1, 4, 4, 4, -1);
        MV1RefreshCollInfo(hModel);
        bool hit = false;
        for (int i = 1; i <= 4; i++) {
            VECTOR p1 = (weaponLine1 - lastLine1) * (i / 4.0f) + lastLine1;
            VECTOR p2 = (weaponLine2 - lastLine2) * (i / 4.0f) + lastLine2;
            MV1_COLL_RESULT_POLY res = MV1CollCheck_Line(hModel, -1, p1, p2);
            if (res.HitFlag) // 当たっている
            {
                hit = true;
            }
        }

        // 4本線のために、線情報を保存する
        lastLine1 = weaponLine1;
        lastLine2 = weaponLine2;
        return hit;
    }
protected:
    VECTOR position;
    VECTOR rotation;
    int hModel;
    int HP_MAX;
    int HP;
    int attack;
    int defense;
    bool isAttack;
    bool isDeath = false;

    //hitcheckに使う
    VECTOR lastLine1; // 前回の位置
    VECTOR lastLine2;
};
```

工夫した点

このEnemyクラスでは、ゲーム内での敵キャラクターの挙動を柔軟に管理できるように、共通機能の一元管理を徹底しました。まず、ダメージ処理や死亡判定を基底クラスに組み込むことで、敵キャラクターの基本的な動作を統一しました。これにより、個別の敵キャラクターごとの重複したコードを減らし、コードの再利用性を高めています。
また、ヒットチェックにおいては、攻撃が敵キャラクターのモデルに当たるかどうかを判定するために、線分での衝突判定を導入しました。さらに、前回の攻撃線情報を保持し、攻撃が滑らかに判定されるように工夫しています。これにより、プレイヤーの攻撃に対する反応がより自然で正確になり、ゲームプレイのクオリティが向上しました。
さらに、今後新しい敵キャラクターを追加する際の拡張性を意識して、クラス設計を行いました。例えば、TakeDamageやAttackメソッドを純粋仮想関数として定義することで、派生クラスで個別の挙動を簡単に実装できるようにしています。これにより、今後新たな敵キャラクターや特殊な挙動を持つ敵を追加する際に、最小限の変更で済む設計にしています。

コード6

TargetManager.cpp*

project

```cpp
#include "TargetManager.h"

namespace
{
    vector<Enemy*> targets;
    vector<Enemy*> enemys;
    bool isEnemyIntroPlaying;
}

/// <summary>
/// 初期化
/// </summary>
void TargetManager::TargetmanagerInit()
{
    SetIsEnemyIntroPlaying(true);/*
    ClearTarget();
    ClearEnemy();*/
}

/// <summary>
/// ターゲットモンスターを追加
/// </summary>
/// <param name="enemy"></param>
void TargetManager::AddTarget(Enemy* enemy)
{
    targets.push_back(enemy);
    AddEnemy(enemy);
}
```

```cpp
/// <summary>
/// Enemyを追加
/// </summary>
/// <param name="enemy"></param>
void TargetManager::AddEnemy(Enemy* enemy)
{
    enemys.push_back(enemy);
}

/// <summary>
/// リソース開放
/// </summary>
void TargetManager::ClearTarget()
{
    for (Enemy* e : targets) {
        delete e;
    }
    targets.clear();
}

void TargetManager::ClearEnemy()
{
    for (Enemy* e : enemys) {
        delete e;
    }
    enemys.clear();
}
```

```cpp
/// <summary>
/// 全ターゲットモンスターを取得
/// </summary>
/// <returns></returns>
vector<Enemy*> TargetManager::getTargets()
{
    return targets;
}

/// <summary>
/// 全モンスターを取得
/// </summary>
/// <returns></returns>
vector<Enemy*> TargetManager::getEnemy()
{
    return enemys;
}

/// <summary>
/// 登場演出のboolを返す
/// </summary>
/// <returns></returns>
bool TargetManager::IsEnemyIntroPlaying()
{
    return isEnemyIntroPlaying;
}

/// <summary>
/// 登場演出のbool変更
/// </summary>
/// <param name="a"></param>
void TargetManager::SetIsEnemyIntroPlaying(bool a)
{
    isEnemyIntroPlaying = a;
}
```

工夫した点

敵キャラクターの管理と演出フラグ制御を行うクラスを実装しました。
ターゲットとして扱う敵と全体の敵リストを Enemy* ポインタで分けて管理し、登場演出の状態はフラグ (isEnemyIntroPlaying) で制御しています。リスト内のオブジェクトは明示的に delete でリソースを解放し、メモリリーク防止も意識した実装にしています。
AddTarget() 内で AddEnemy() を呼び出すことで二重登録の管理も行っており、効率的なオブジェクト管理を実現しています。

コード7

```
/// <summary>
/// プレイヤーのほうを見る
/// </summary>
void Mutant::LookPlayer()
{
    VECTOR pPos = pl->Position();
    //内積
    //TODO:敵がブルブルするから修正する
    VECTOR forPlayer = pPos - position;
    VECTOR right = VGet(1, 0, 0) * MGetRotY(rotation.y);
    float dot = VDot(right, forPlayer);
    if (dot > 0) {
        rotation.y -= DegToRad(3.0f);
    }
    if (dot < 0) {
        rotation.y += DegToRad(3.0f);
    }
}
```

工夫した点

敵がプレイヤーの方向を見るコードを書くときに内積を使うことで短く読みやすいコードになった。

BOXRAWL

概要

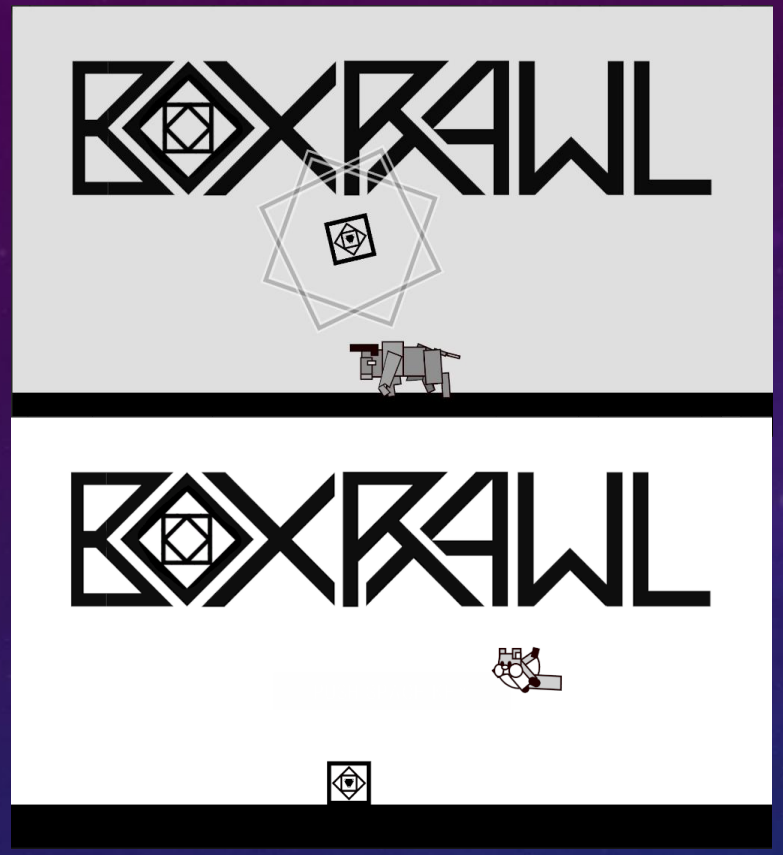
ジャンル：2Dアクション
プラットフォーム：PC
開発エンジン：unity
言語：C#
人数：６人
制作期間：４ヶ月
使用アセット：DoTween

タイトル：BOXRAWL

ゲーム内容

次々に出てくる敵を踏み潰す
アクションゲームです。
複数の敵を同時に踏み潰すことで、
爽快感を味わうことができる。

コード1




```
58      /// <summary>
59      /// 敵の動き
60      /// </summary>
61      void eMove()
62      {
63          if (TitleManager.isStart)
64          {
65              //moveSpeed = 2;
66              if(EnemyNum == 0)
67              {
68                  this.gameObject.transform.position += new Vector3(-4.0f * moveSpeed, 0, 0) * Time.deltaTime;
69              }
70          }
71          else
72          {
73              this.gameObject.transform.position += new Vector3(-4.0f * moveSpeed, 0, 0) * Time.deltaTime;
74          }
75
76          if (this.gameObject.transform.position.x < -11
77              && !TitleManager.isStart)
78          {
79              TitlePlayer.isJump = false;
80              EnemyNum = Random.Range(0, 2);
81              if (EnemyNum == 0)  //牛
82              {
83                  EnemyRg.velocity = Vector2.zero;
84                  EnemyRg.angularVelocity = 0.0f;
85                  EnemyRg.gravityScale = 0.0f;
86                  this.gameObject.transform.position = enemyStartPos;
87
88                  //enemySpr.sprite = enemySprite[EnemyNum];
89                  enemySkin[0].SetActive(true);
90                  enemySkin[1].SetActive(false);
91
92                  moveSpeed = cowMoveSpeed;
93              }
94              else if (EnemyNum == 1) //モモンガ
95              {
96                  isUp = true;
97                  EnemyRg.velocity = Vector2.zero;
98                  EnemyRg.angularVelocity = 0.0f;
99                  EnemyRg.gravityScale = 0.12f;
100                 this.gameObject.transform.position = new Vector3(enemyStartPos.x, 4.3f, enemyStartPos.z);
101
102                 //enemySpr.sprite = enemySprite[EnemyNum];
103                 enemySkin[0].SetActive(false);
104                 enemySkin[1].SetActive(true);
105                 moveSpeed = 1;
```

工夫点

タイトル画面で放置していると、ランダムで敵キャラが通過して行くようにしました。
また、ランダムでプレイヤーキャラが敵キャラを倒したりよけたりするようにしました。

コード2

```
249     /// <summary>
250     /// 古いスコア表示
251     /// 後で消す
252     /// </summary>
253     void scoreDisplay()
254     {
255         switch (result)
256         {
257             case Result.Exp:
258                 scoreText[1].DOFade(1, 1.0f);
259                 if (counter >= 0.5f)
260                 {
261                     result = Result.Time;
262                 }
263                 break;
264             case Result.Time:
265                 scoreText[2].DOFade(1, 1.0f);
266                 if (counter >= 1.0f)
267                 {
268                     result = Result.Enemy;
269                 }
270                 break;
271             case Result.Enemy:
272                 scoreText[3].DOFade(1, 1.0f);
273                 if (counter >= 1.5f)
274                 {
275                     result = Result.SmallEnemy;
276                 }
277                 break;
278             case Result.SmallEnemy:
279                 scoreText[4].DOFade(1, 1.0f);
280                 if (counter >= 2.0f)
281                 {
282                     result = Result.Total;
283                 }
284                 break;
285             case Result.Total:
286                 if (countScore < totalScore)
287                 {
288                     countScore += (totalScore * Time.deltaTime) / 3;
289                     scoreText[0].text = countScore.ToString("f0");
290                 }
291                 else if (countScore > totalScore)
292                 {
293                     countScore = totalScore;
294                     scoreText[0].text = totalScore.ToString("f0");
295                     result = Result.ClearCheck;
296                 }
297 
298                 if (0 >= totalScore)
299                 {
300                     result = Result.ClearCheck;
301                     Debug.Log("クリアチェック");
302                 }
303 
304                 if (Input.GetKey(KeyCode.Return))
305                 {
306                     countScore = totalScore;
307                     scoreText[0].text = totalScore.ToString("f0");
308                     result = Result.ClearCheck;
309                 }
310 
311                 break;
312             case Result.ClearCheck:
313                 if (clearScore <= totalScore)
314                 {
315                     mask.padding -= new Vector4(0, 0, ((75 * Time.deltaTime) * 5), 0);
316                     if (mask.padding.z < 0)
317                     {
318                         result = Result.None;
319                     }
320                 }
321                 else
322                 {
323                     result = Result.None;
324                 }
325                 break;
326             case Result.None:
327                 pressText.SetActive(true);
328                 scoreDisplayEnd = false;
329                 break;
330             default:
331                 break;
332         }
```

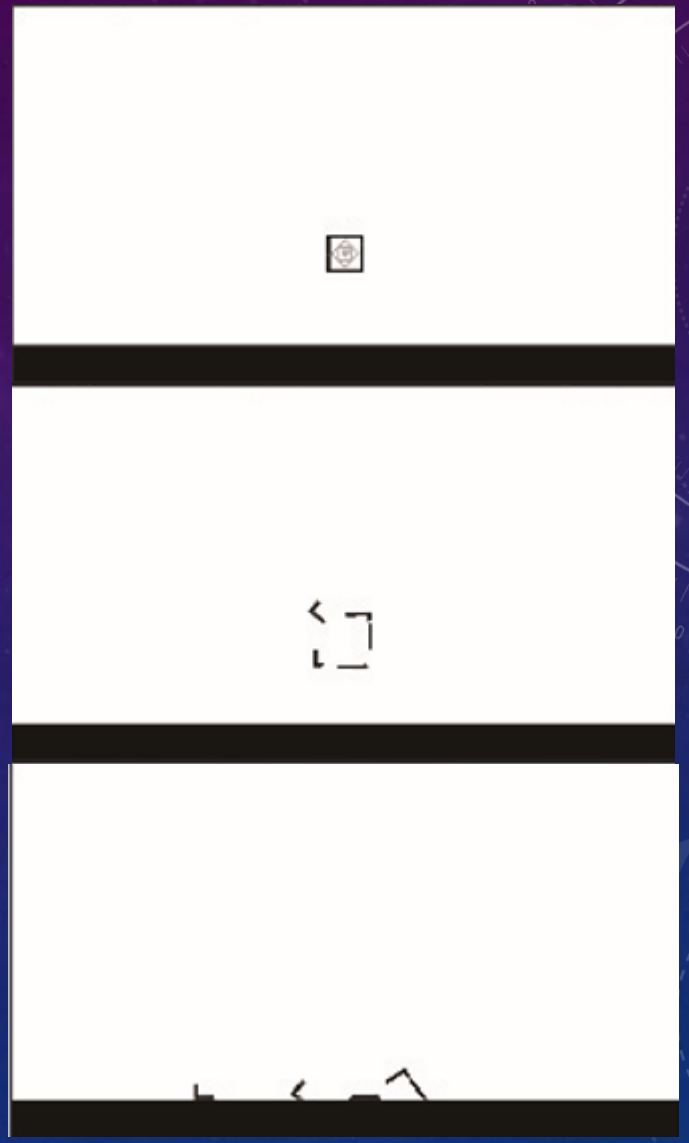
```
207     /// <summary>
208     /// スコアを表示
209     /// </summary>
210     /// <returns></returns>
211     public IEnumerator scoreDisplay2()
212     {
213         for(int i = 1; i < scoreText.Length; i++)
214         {
215             scoreText[i].DOFade(1, 1.0f);
216             yield return new WaitForSeconds(0.5f);
217         }
218 
219         if (countScore < totalScore)
220         {
221             countScore += (totalScore * Time.deltaTime) / totalScoreCountupTime;
222             scoreText[0].text = countScore.ToString("f0");
223         }
224         else if (countScore > totalScore)
225         {
226             countScore = totalScore;
227             scoreText[0].text = totalScore.ToString("f0");
228             result = Result.ClearCheck;
229         }
230         yield return new WaitForSeconds(totalScoreCountupTime);
231         if (clearScore <= totalScore) //クリアチェック
232         {
233             mask.padding -= new Vector4(0, 0, ((75 * Time.deltaTime) * 5), 0);
234             if (mask.padding.z < 0)
235             {
236                 result = Result.None;
237                 if (se == false)
238                 {
239                     se = true;
240                     SEController.check = true;
241                 }
242             }
243         }
244         scoreDisplayEnd = true;
245         pressText.SetActive(true);
246         yield return null;
247     }
```

改善した点

スコア画面でswitch文で表示したら次のcaseに進むというコードを書いた結果コードが長くなってしまった、
どうにかして短くできないか悩んだ結果、コルーチンを使えば短くなるんじゃないかと思いコードを書き直した。
その結果、コーダが短くなり読みやすくなった。

コード3

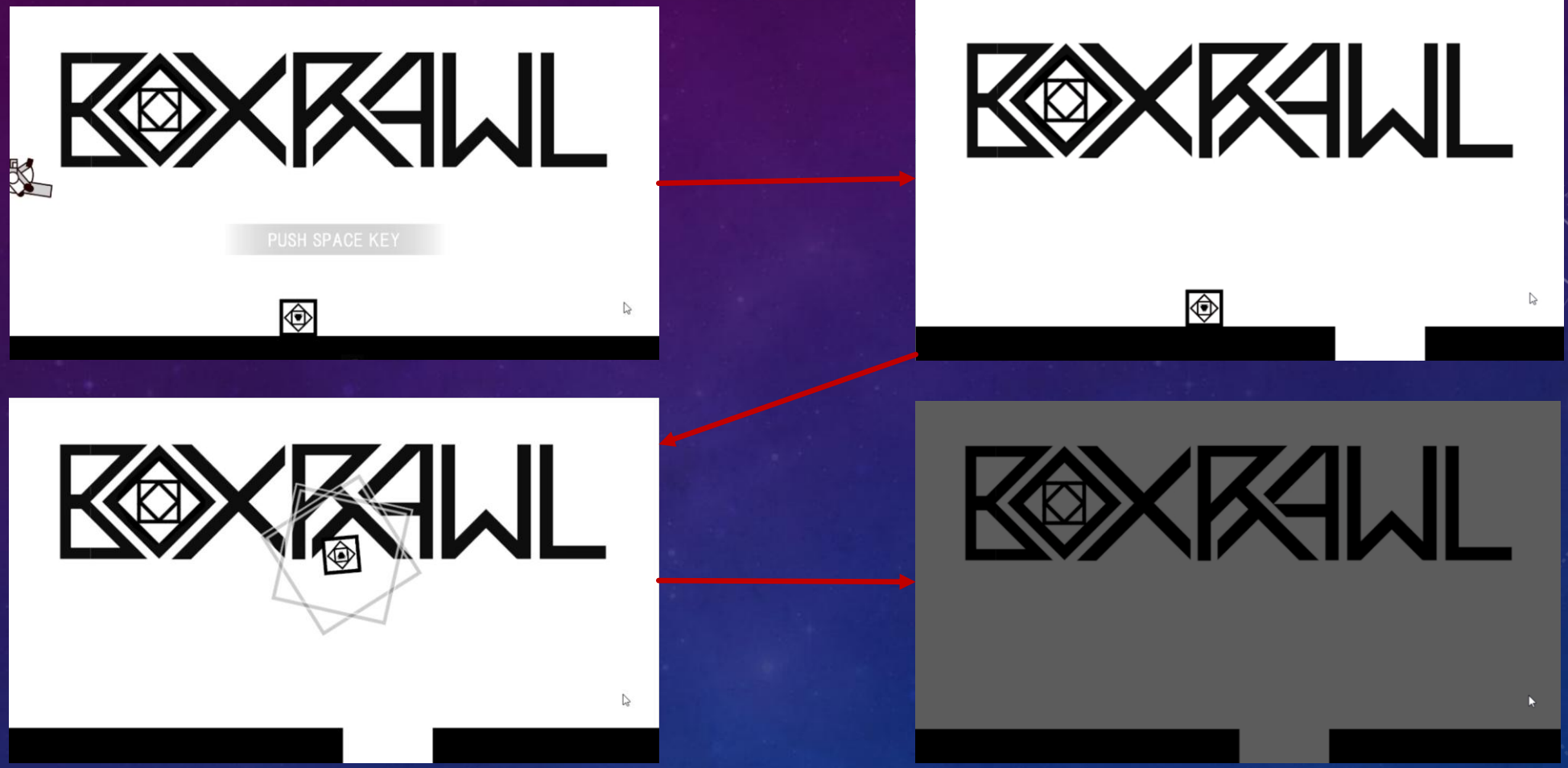
```
public class Death : MonoBehaviour
{
    [SerializeField] GameObject[] fragmentObj = new GameObject[5];
    Rigidbody2D[] Rg = new Rigidbody2D[5];

    [SerializeField] SpriteRenderer[] skinSpr = new SpriteRenderer[2];

    public int breakPower = 600;

    void Start()
    {
        for (int i = 0; i < fragmentObj.Length; i++)
        {
            Rg[i] = fragmentObj[i].GetComponent<Rigidbody2D>();
        }
        breakPower = 600;

        skinSpr[0].DOFade(0, 1.0f);
        skinSpr[1].DOFade(0, 1.0f);
        Invoke("breack", 1);
    }

    /// <summary>
    /// 壊れる演出
    /// </summary>
    void breack()
    {
        for (int i = 0; i < fragmentObj.Length; i++)
        {
            Rg[i].constraints = RigidbodyConstraints2D.None;

            Vector2 force = fragmentObj[i].transform.position - this.gameObject.transform.position;
            Vector2 force_test = new Vector2(10, 10);
            Rg[i].AddForce(force * breakPower);
        }
    }
}
```



工夫した点

プレイヤーのHPが0になったときにキャラがバラバラに弾け飛ぶようにしました。
弾け飛ぶ威力を調整できるようにしました。

Start演出



タイトル画面でSPACEを押すと穴が現れてその穴に入っていく演出です。
穴に落ちたらフェードアウトしていきます。
DoTweenを使い制作しました。

コード4

```
/// <summary>
/// スタート演出1
/// </summary>
void startGame()
{
    GroundObj.transform.DOMove(tergetObj.transform.position, 5.0f);
    if(TitleEnemy.getTitleEnemy() == 1)
    {
        if (playerObj.transform.position.x > EnemyObj.transform.position.x)
        {
            EnemyObj.transform.DOMove(EnemyObj.transform.position + (tergetObj.transform.position + GroundObj.transform.position), 5.0f);
        }
        else
        {
            EnemyObj.transform.Rotate(0, -180, 0);
            EnemyObj.transform.DOMoveX(EnemyObj.transform.position.x + (GroundObj.transform.position.x - tergetObj.transform.position.x), 5.0f);
        }
    }

    StartCoroutine(playerJump());
}
```

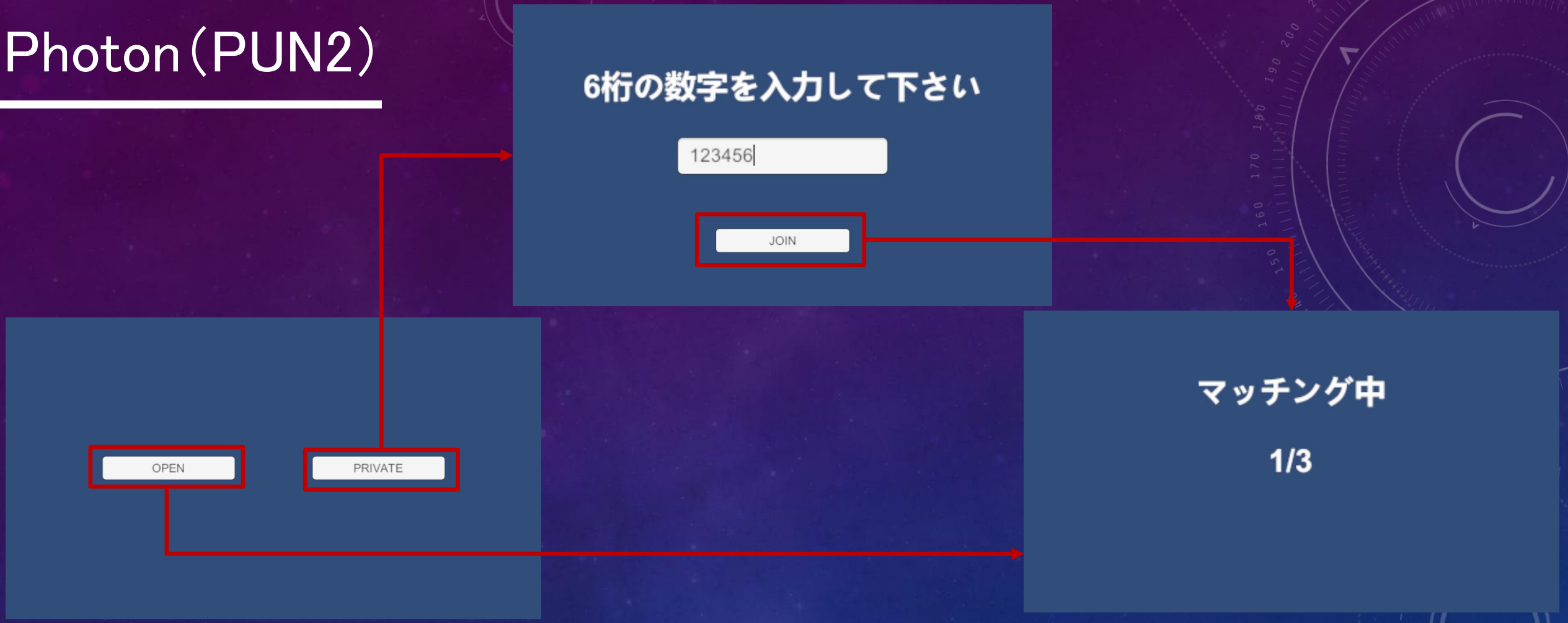
```
/// <summary>
/// スタート演出2
/// </summary>
/// <returns></returns>
public IEnumerator playerJump()
{
    yield return new WaitForSeconds(3.0f);
    Vector2 force = new Vector3(1.0f, 9.5f);
    rg.AddForce(force *50);
    paryObj.SetActive(true);
    PlayerSkin.Rota = true;
    PlayerSkin.rota = 1;

    yield return new WaitForSeconds(2.0f);
    startFlag = true;
    yield return new WaitForSeconds(2.0f);
    if(isTutorial <= 0)
    {
        SceneManager.LoadScene("Tutorial");
        isTutorial = 1;
    }
    else if (isTutorial >= 1)
    {
        SceneManager.LoadScene("Menu");
    }
    PlayerPrefs.SetInt("Tutorial", isTutorial);
    PlayerPrefs.Save();
    yield return null;
}
```

工夫した点

DoTweenを使用し、スタート演出を実装しています。
地面オブジェクトをターゲット位置まで移動させ、条件に応じて敵キャラクターの移動・回転演出を追加。プレイヤーにはジャンプによる登場演出を加え、ゲーム開始への期待感を演出しています。
DOMove や DOMoveX を活用した自然なアニメーション表現により、開始時の没入感を高めています。

その他

Photon（PUN2）



上記の図のようにphotonを使い簡単なマッチング機能を作成しました。
OPENのほうはランダムにマッチングするようになっています。
PRIVATEのほうは同じ6桁の数字を入力した人とのみマッチングします。
※マッチング中画面は同じなので同じ画像を使用しています

photonコード1

```
using Photon.Pun;
using Photon.Realtime;

public class OpenMatch : MonoBehaviourPunCallbacks
{
    /// <summary>
    /// ランダムで部屋にマッチングする
    /// </summary>
    public void RandomMuch()
    {
        PhotonNetwork.JoinRandomRoom();
    }

    /// <summary>
    /// マッチングできなかった時に部屋を作る
    /// </summary>
    /// <param name="returnCode"></param>
    /// <param name="message"></param>
    public override void OnJoinRandomFailed(short returnCode, string message)
    {
        var roomOptions = new RoomOptions();
        roomOptions.MaxPlayers = 3;
        PhotonNetwork.CreateRoom(null, roomOptions);
    }

    /// <summary>
    /// 部屋に入れた時
    /// </summary>
    public override void OnJoinedRoom()
    {
        if (PhotonNetwork.CurrentRoom.PlayerCount == PhotonNetwork.CurrentRoom.MaxPlayers)
        {
            PhotonNetwork.CurrentRoom.IsOpen = false;
        }
    }
}
```

工夫した点

Photon PUN2を使用した、ランダムマッチング機能の実装コードです。
既存の部屋に入れなかった場合には、自動で新しい部屋を作成し、プレイヤー数が最大に達した場合は IsOpen を false に設定してマッチングを締め切るように制御しています。
マルチプレイゲームの基本となる、シンプルかつ実用的なマッチングロジックを意識しました。

photonコード2

```
    /// <summary>
    /// 部屋に入れた時の処理
    /// </summary>
    public override void OnJoinedRoom()
    {
        Debug.Log("接続しました");
        passwordObj.SetActive(false);
        waitObj.SetActive(true) ;

        int a = PhotonNetwork.CurrentRoom.PlayerCount;
        waitText.text = a + "/3";
    }

    /// <summary>
    /// 部屋に入れなかった時の処理
    /// </summary>
    /// <param name="returnCode"></param>
    /// <param name="message"></param>
    public override void OnJoinRoomFailed(short returnCode, string message)
    {

    }

    /// <summary>
    /// 他のプレイヤーが部屋に入ってきたら呼ばれる
    /// </summary>
    /// <param name="newPlayer"></param>
    public override void OnPlayerEnteredRoom(Player newPlayer)
    {
        int a = PhotonNetwork.CurrentRoom.PlayerCount;
        waitText.text = a + "/3";
    }
}
```

```
using UnityEngine;
using Photon.Pun;
using Photon.Realtime;
using UnityEngine.UI;

public class PrivateMatch : MonoBehaviourPunCallbacks
{
    public GameObject selectButtonObj;
    public GameObject passwordObj;
    public GameObject waitObj;
    public InputField passwordInputField;
    public Button joinButton;

    public Text waitText;

    void Start()
    {
        selectButtonObj.SetActive(true);
        passwordObj.SetActive(false);
        waitObj.SetActive(false);

#if true //デバッグ用
        PhotonNetwork.ConnectUsingSettings();
#endif
    }

    void Update()
    {
        joinButton.interactable = (passwordInputField.text.Length == 6);
    }

    /// <summary>
    /// privateボタンを押したときの処理
    /// </summary>
    public void privateMatch()
    {
        selectButtonObj.SetActive(false);
        passwordObj.SetActive(true);
    }

    /// <summary>
    /// passwordを入力した後に部屋に接続ボタンを押したときの処理
    /// </summary>
    public void joinButtonClick()
    {
        var roomOptions = new RoomOptions();
        roomOptions.MaxPlayers = 3;
        roomOptions.IsVisible = false;

        PhotonNetwork.JoinOrCreateRoom(passwordInputField.text, roomOptions,TypedLobby.Default);
    }
```

工夫した点

Photon PUN2 を使用し、パスワードを用いたプライベートマッチ機能を実装しています。
入力された6桁のパスワードをルーム名として使用し、JoinOrCreateRoom() により部屋を作成または参加。部屋の可視性は非公開 (IsVisible = false) にし、他プレイヤーが参加すると UI に人数を反映させる処理も行っています。
UIとの連携を意識し、ゲーム内で手軽にフレンド同士が集まれる機能を構築しました。