Unity シューティングゲーム ポートフォリオ掲載用コード（抜粋）

この資料は、Unity で制作した 2D シューティングゲームにおける代表的なスクリプトを抜粋・説明したものです。

---

1. StageManager.cs（ステージ生成・難易度管理）

概要: 雑魚敵の出現数やボスの出現タイミングを難易度に応じて自動調整し、アイテム生成までを統括するステージ全体の制御スクリプト。

```
private void GenerateStage()
{
  int spawnCount = 1;

  // 難易度に応じて雑魚敵の出現数を調整
  if (difficultyLevel >= 2 && difficultyLevel <= 5)
  {
    spawnCount = 2;
  }
  else if (difficultyLevel >= 6 && difficultyLevel <= 9)
  {
    spawnCount = 3;
  }
```

```
// 雑魚敵をランダムな位置に指定数生成
for (int i = 0; i < spawnCount; i++)
{
    Vector3 pos = GetRandomPosition();
    GameObject enemy = normalEnemies[Random.Range(0, normalEnemies.Count)];
    Instantiate(enemy, pos, Quaternion.identity);
}

// 難易度が 3 の倍数のときにボスを出現させる（順番にローテーション）
if (difficultyLevel % 3 == 0)
{
    Vector3 pos = GetBossPosition();
    int bossIndex = (difficultyLevel / 3 - 1) % bossEnemies.Count;

    GameObject bossPrefab = bossEnemies[bossIndex];
    GameObject bossInstance = Instantiate(bossPrefab, pos, Quaternion.identity);
    bossInstance.tag = "Boss";

    var enemyScript = bossInstance.GetComponent<Enemy_Spine_Anim>();
    if (enemyScript != null)
    {
        enemyScript.isBoss = true; // ボスフラグを設定
        Debug.Log(enemyScript.maxHp);
    }
}
```

```
    // アイテムをランダムな位置に 1 つ出現させる
    Vector3 itemPos = GetRandomPosition();
    Instantiate(itemPrefabs[Random.Range(0, itemPrefabs.Count)], itemPos,
Quaternion.identity);
}
```

————————————————————————————————————

2. ScoreManager.cs（スコア・復活・ゲーム進行管理）

概要: スコアの増減・ハイスコア管理・復活条件やステージの再生成などを担当。

```
public void LoadNextScene()
{
    LevelNum = PlayerPrefs.GetInt("LevelNum", 1) + 1;
    PlayerPrefs.SetInt("LevelNum", LevelNum);

    StageManager stageManager = FindObjectOfType<StageManager>();
    if (stageManager != null)
    {
        stageManager.difficultyLevel++;
        stageManager.RegenerateStage();
    }

    FindObjectOfType<GameClear>()?.ResetClearCheck();
```

```
}
```

---

3. GameClear.cs（ゲームクリア判定）

概要: 敵の全滅やボス撃破によってクリア条件を判定し、UI を表示。次のステージへ進行する処理を含む。

```
void Update()
{
    if (gameClearTrigger) return;

    int bossCount = GameObject.FindGameObjectsWithTag("Boss").Length;
    int enemyCount = GameObject.FindGameObjectsWithTag("Enemy").Length;

    if (bossSpawned && bossCount == 0)
        TriggerGameClear();
    else if (!bossSpawned && enemySpawned && enemyCount == 0)
        TriggerGameClear();
}
```

---

4. Enemy2.cs（弾回避 AI）

概要: プレイヤーの弾を検知し、自動で NavMeshAgent によって避ける回避行動を実装。簡易的な AI として機能し、ゲームの戦略性を高める。

```
bool IsBulletNearby(out Vector3 awayDir)
{
   Collider2D[] hits = Physics2D.OverlapCircleAll(transform.position, 6f,
LayerMask.GetMask("Default", "PlayerBullet"));
   foreach (var hit in hits)
   {
      if (hit.CompareTag("Player-bullet"))
      {
         awayDir = (transform.position - hit.transform.position).normalized;

         // ±45 度の範囲で回転させて、斜めに逃げる
         float angleOffset = Random.Range(-45f, 45f);
         awayDir = Quaternion.Euler(0, 0, angleOffset) * awayDir;

         // 回避距離もランダムに
         float dodgeDistance = Random.Range(5f, 15f);
         Vector3 dodgePos = transform.position + awayDir * dodgeDistance;

         // 回避行動で画面外から出ないように
         dodgePos.x = Mathf.Clamp(dodgePos.x, -16.5f, 16.5f);
         dodgePos.y = Mathf.Clamp(dodgePos.y, -5.9f, 5.9f);

         agent.SetDestination(dodgePos);
         return true;
```

```
        }
    }
    awayDir = Vector3.zero;
    return false;
}
```

---

5. UbhLinearShot.cs（ジョイスティックと弾の発射角制御）

概要: ジョイスティック入力に応じて発射角度を制御し、直線的な弾幕を構築。プレイヤーの操作性に直結する。

```
if (joystick.Horizontal != 0 || joystick.Vertical != 0)
{
    m_angle = Mathf.Atan2(joystick.Vertical, joystick.Horizontal) * Mathf.Rad2Deg - 90f;
    FireBullet(m_angle);
}
```

備考：設計意識と改善への姿勢

- 各機能は責務単位でクラス分離（スコア/ステージ/敵/ゲーム状態）し、管理性と拡張性を高めた設計。

- SerializeField を活用し、エディタ上でパラメータ調整しやすい構造。

- シーン遷移やハイスコア保存には PlayerPrefs を活用。

- 今後は、研究室で機械学習を活用したゲーム難易度調整を研究予定。将来的には自動的にプレイヤーに適応する AI ボスの実装を視野に入れています。

- 改善の余地としては、Manager クラスの役割明確化や UI フローの分離など設計面でのさらなる洗練を進めたいと考えています。

――――――――――――――――――――――――――――――――――――