AI　モンスターの知能

モンスターに視覚を与える

isFindPlayer は誰が決めているのか？

1. 前回までの確認

前回、モンスターの行動を `if文` で切り替える考え方を学びました。

例えば、次のような処理です。

```
if (isAttackRange)
{
    Attack();
}
else if (isFindPlayer)
{
    ChasePlayer();
}
else
{
    Idle();
}
```

この処理では、モンスターは次のように行動します。

条件	行動
攻撃範囲にいる	攻撃する
プレイヤーを見つけている	追いかける
どちらでもない	待機する

このように、条件によって行動を変えることで、モンスターが考えて動いているように見えます。

2. ここで問い

次の変数に注目してください。

```
isFindPlayer
```

これは、

```
プレイヤーを見つけているか？
```

を表すフラグです。

しかし、ここで考えるべきことがあります。

`isFindPlayer` は誰が決めているのか？

プログラムの中で、

```
if (isFindPlayer)
```

と書くことはできます。

しかし、`isFindPlayer` が `true` になる理由を作らなければ、モンスターはプレイヤーを見つけることができません。

つまり、

```
isFindPlayer = true;
```

になる条件を、こちらで作る必要があります。

3. ゲームAIには「知覚」が必要

モンスターが行動を決めるには、まずゲームの状況を知る必要があります。

例えば、

- プレイヤーが近くにいる
- プレイヤーが前方にいる
- プレイヤーとの間に壁がない
- プレイヤーが音を立てた
- プレイヤーが攻撃してきた

などです。

このように、AIがゲームの状況を知る仕組みを **知覚** と呼びます。

4. 人間の「見る」とAIの「見る」は違う

人間は目で映像を見ています。

しかし、ゲームAIは本当に画面を見ているわけではありません。

ゲームAIは、次のような数値や条件を使って「見えているか」を判定します。

```
距離は近いか？
前方にいるか？
壁に隠れていないか？
```

つまり、ゲームAIの視覚とは、

```
画像を見ること
```

ではなく、

```
条件を満たしたら「見えている」とする処理
```

です。

5. まず一番簡単な視覚を作る

最初は、距離だけで考えます。

例えば、

```
プレイヤーが10m以内にいたら発見
```

というルールにします。

この場合、モンスターはプレイヤーとの距離を調べます。

```
VECTOR toPlayer = VSub(playerPos, enemyPos);
float distance = VSize(toPlayer);

if (distance < VIEW_DISTANCE)
{
    isFindPlayer = true;
}
else
{
    isFindPlayer = false;
}
```

ここで、`VIEW_DISTANCE` は視界距離です。

```
const float VIEW_DISTANCE = 10.0f;
```

---

6. 距離だけの視覚の問題点

距離だけで判定すると、かなり単純なAIになります。

例えば、次のような問題があります。

問題1：後ろにいても見つかる

距離だけで判定すると、プレイヤーがモンスターの後ろにいても見つかってしまいます。

```
プレイヤー
  ↓
モンスター → 前を向いている
```

本当なら、モンスターの後ろにいるプレイヤーは見えないはずです。

でも距離だけで判定していると、近ければ見つかります。

問題2：壁越しでも見つかる

距離だけで判定すると、壁の向こうにいるプレイヤーも見つかってしまいます。

```
モンスター　｜壁｜　プレイヤー
```

本当なら、壁があるので見えないはずです。

でも距離だけで判定していると、近ければ見つかります。

---

7. 視覚に必要な条件

より自然な視覚を作るには、次の3つを考えます。

```
1. 距離
2. 向き
3. 障害物
```

---

8. 条件1：距離

まずは距離です。

```
プレイヤーが遠すぎるなら見えない
```

これは分かりやすい条件です。

```
if (distance > VIEW_DISTANCE)
{
    isFindPlayer = false;
}
```

---

9. 条件2：向き

次に、プレイヤーがモンスターの前方にいるかを調べます。

モンスターには向きがあります。

```
モンスター → 前方
```

プレイヤーが前方にいれば見える。

```
モンスター → プレイヤー
```

しかし、プレイヤーが後ろにいるなら見えません。

```
プレイヤー ← モンスター
```

---

10. 前方にいるかをどう調べるか

ここで使うのが **内積** です。

必要なベクトルは2つです。

```
モンスターの前方向
モンスターからプレイヤーへの方向
```

---

11. モンスターからプレイヤーへの方向

まず、モンスターからプレイヤーへのベクトルを求めます。

```
VECTOR toPlayer = VSub(playerPos, enemyPos);
```

次に、長さを1にします。

```
float distance = VSize(toPlayer);
VECTOR dirToPlayer = VScale(toPlayer, 1.0f / distance);
```

この `dirToPlayer` が、モンスターから見たプレイヤーの方向です。

---

12. モンスターの前方向

次に、モンスターの前方向を用意します。

例として、Y軸回転から前方向を作るなら、次のようになります。

```
VECTOR forward =
{
    sinf(enemyRotY),
    0.0f,
```

```
    cosf(enemyRotY)
};
```

ただし、モデルの正面方向や座標系によって、`sinf` と `cosf` の使い方は変わることがあります。

---

13. 内積で角度を調べる

2つのベクトルの内積を取ります。

```
float dot = VDot(forward, dirToPlayer);
```

`dot` の値は、だいたい次のように考えられます。

dotの値	意味
1.0 に近い	正面にいる
0.0 に近い	横にいる
-1.0 に近い	後ろにいる

---

14. 視野角の判定

例えば、次のようにします。

```
const float VIEW_DOT = 0.5f;

if (dot > VIEW_DOT)
{
    isFindPlayer = true;
}
else
{
    isFindPlayer = false;
}
```

`dot > 0.5f` なら、前方のある範囲内にいると判断できます。

---

15. VIEW_DOT の目安

条件	見える範囲の目安
`dot > 0.5f`	前方約120度
`dot > 0.707f`	前方約90度
`dot > 0.866f`	前方約60度

数値を大きくすると、視野が狭くなります。

数値を小さくすると、視野が広くなります。

---

16. 距離と向きを組み合わせる

距離と向きを組み合わせると、次のような判定になります。

```
bool CheckCanSeePlayer()
{
    VECTOR toPlayer = VSub(playerPos, enemyPos);
    float distance = VSize(toPlayer);

    if (distance > VIEW_DISTANCE)
    {
        return false;
    }
```

```
    if (distance < 0.001f)
    {
        return true;
    }

    VECTOR dirToPlayer = VScale(toPlayer, 1.0f / distance);

    VECTOR forward =
    {
        sinf(enemyRotY),
        0.0f,
        cosf(enemyRotY)
    };

    float dot = VDot(forward, dirToPlayer);

    if (dot < VIEW_DOT)
    {
        return false;
    }

    return true;
}
```

この関数は、

```
プレイヤーが見えているなら true
見えていないなら false
```

を返します。

---

17. 条件3：障害物

まだ問題があります。

距離が近くて、前方にいても、壁の向こうにいるなら本当は見えません。

```
モンスター ---- 壁 ---- プレイヤー
```

この場合は、見えないようにしたいです。

---

18. 壁で見えない処理

考え方は単純です。

```
モンスターからプレイヤーまで線を引く
↓
途中に壁があるか調べる
↓
壁があれば見えない
↓
壁がなければ見える
```

このような判定を、レイ判定やライン判定と呼びます。

---

19. 疑似コード

```
bool isHitWall = CheckLineHitWall(enemyPos, playerPos);

if (isHitWall)
{
    return false;
}
```

`CheckLineHitWall` は、敵とプレイヤーの間に壁があるか調べる処理だと考えてください。

20. 最終的な視覚判定

距離、向き、壁を組み合わせると、次のような流れになります。

```
bool CheckCanSeePlayer()
{
    VECTOR toPlayer = VSub(playerPos, enemyPos);
    float distance = VSize(toPlayer);

    // 遠すぎるなら見えない
    if (distance > VIEW_DISTANCE)
    {
        return false;
    }

    // ほぼ同じ位置なら見えている扱い
    if (distance < 0.001f)
    {
        return true;
    }

    // 方向を求める
    VECTOR dirToPlayer = VScale(toPlayer, 1.0f / distance);

    // モンスターの前方向
    VECTOR forward =
    {
        sinf(enemyRotY),
        0.0f,
        cosf(enemyRotY)
    };

    // 前方にいるか調べる
    float dot = VDot(forward, dirToPlayer);

    if (dot < VIEW_DOT)
    {
        return false;
    }

    // 壁があれば見えない
    if (CheckLineHitWall(enemyPos, playerPos))
    {
        return false;
    }

    return true;
}
```

21. 前回のAIに接続する

視覚判定ができたら、前回のAIに接続できます。

```
isFindPlayer = CheckCanSeePlayer();

if (isAttackRange)
{
    Attack();
}
else if (isFindPlayer)
{
    ChasePlayer();
}
```

```
else
{
    Idle();
}
```

これで、

```
見えたら追いかける
見えなければ待機する
```

というモンスターになります。

---

22. 何がAIらしくなるのか

距離だけで判定していたときは、

```
近ければ必ず見つかる
```

という動きでした。

しかし、視野角や壁判定を入れると、次のようになります。

```
後ろに回ると気づかれない
壁に隠れると見つからない
正面に出ると追いかけてくる
```

このようなルールがあると、プレイヤーは、

```
敵に見つかった
敵から隠れた
敵の背後を取った
```

と感じやすくなります。

つまり、AIが少し賢く見えます。

---

23. 重要な考え方

ゲームAIは、本当に人間のように世界を理解しているわけではありません。

しかし、次のような情報を使うことで、考えているように見せることができます。

```
距離
方向
角度
障害物
時間
状態
```

AIは、世界をそのまま理解しているのではなく、ゲームに必要な情報だけを取り出して判断しています。

まとめ

今回のポイントは次の通りです。

```
isFindPlayer は勝手に決まらない
```

```
AIには知覚が必要
```

ゲームAIの視覚は画像を見ることではない

距離・向き・壁で「見えている」を作る

見えているかどうかをフラグにして、行動分岐に使う

最後の確認問題

問1

次のフラグは何を表しているか説明しなさい。

```
bool isFindPlayer;
```

問2

距離だけでプレイヤーを発見するAIには、どんな問題がありますか。
2つ書きなさい。

問3

モンスターの前方にプレイヤーがいるかを調べるために使う計算は何ですか。

問4

壁の向こうにいるプレイヤーを見えないようにするには、どのような判定が必要ですか。

問5

次の処理の意味を説明しなさい。

```
isFindPlayer = CheckCanSeePlayer();
```

発展課題

次のようなモンスターを考えなさい。

```
普段は巡回している。
プレイヤーが視界に入ったら追いかける。
壁の向こうに逃げられたら見失う。
見失ったら、その場で3秒間探す。
それでも見つからなければ巡回に戻る。
```

このモンスターに必要なフラグを考えなさい。

例：

```
bool isFindPlayer;
bool isLostPlayer;
bool isSearching;
```

```
bool isPatrol;
```

さらに、どのような順番で `if文` を書けばよいか考えなさい。

Revision #10
★Created 19 May 2026 00:17:39 by youe2
Updated 2 June 2026 17:28:02 by youe2