キャラクターAIを作ってみよう

ゲームエンジニア科　オープンキャンパス
体験実習　キャラクターAIを作ってみよう

今日の目標

今日の体験実習では、２Ｄボンバーマン風ゲームのキャラクターAIのプログラムを作りながら、**C++によるゲーム開発の実際の雰囲気**を体験してほしいと思います。

- 開発環境
  - Windows11
  - Visual Studio 2022
- プログラミング言語
  - C++
- 使用ライブラリ
  - DxLib

C++によるプログラム開発の流れ

1. **Build** → ソースをビルドして実行
2. **Edit** → 指定の行番号の定義を『書き換え』
3. **Test** → 動作を確認し、発見・改善

> **POINT**: 小さな変更を加え、すぐにビルド＆実行。これを繰り返すのが開発サイクル！

ゲームAIの種類

1. キャラクターAI
   - キャラクターAIは文字通り、**NPCを動かすために利用**される
   - 格闘ゲームの敵キャラクターや、RPGで主人公を支援する仲間の動作を決める
2. メタAI
   - **ゲーム進行を補助**する
   - **一定の条件を満たしたらイベントを開始**
   - **ステージに応じて敵やアイテムの配置を変更**したりと、ゲーム画面の裏で色々な処理を行う
3. ナビゲーションAI
   - キャラクターAIやメタAIに対し、**各種情報を提供して動作をサポート**する
   - 敵キャラクターに主人公の位置や進行方向、障害物に関する情報を与える
   - **この仕組みがないと、キャラクターAIは正確な動作を行えない**

その１．キャラクターを選ぼう

ゲームに登場するキャラクターは、その種類によっていろいろなパラメータを持っています。
プログラミング内でそれらを表現するときは、すべて数値や文字などの値として表現します。
これらのゲーム中で利用する値に名前をつけたものを「変数」と呼びます。

変数の例：

- プレイヤーキャラ（Player)
  - 位置 (Position)
  - 移動速度 (Speed)
  - ジャンプ力 (JumpPower)
  - 攻撃力 (AttackPower)
  - 所持アイテム (Items)
  - プレイヤーの画像 (PlayerImage)
  - ・・・

体験用プログラムでも同様に、敵キャラクターをEnemyという変数で表しています。

Enemyはその中に更に細かい設定用のパラメータ（変数を持っています）。
その中の、表示画像を表す変数を変更してみましょう。

Enemyの表示画像の変更

> 行番号：74 行目を見て書き換え

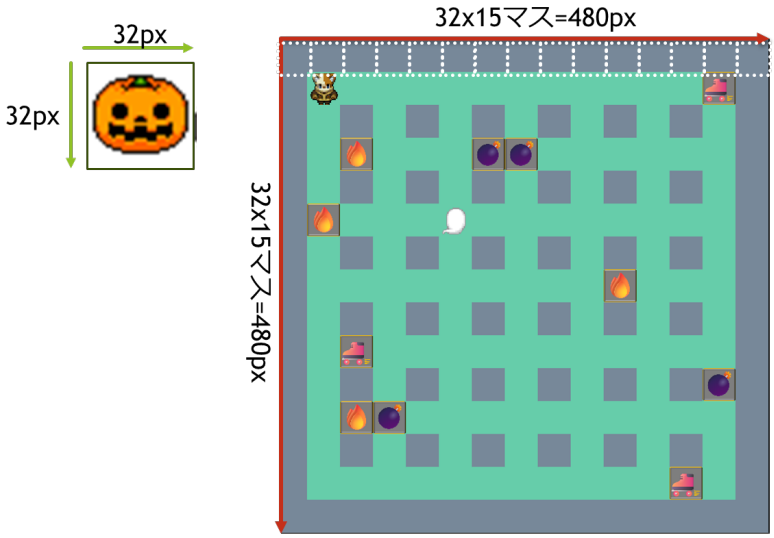
```
42  if (isGraphic) {
43      //#01 敵の画像を選択する！
44      std::string enemyImage = GetEnemyImage(KABOCHA);
45      enemyImage_ = LoadGraph(enemyImage.c_str());
46  }
```

- **やること**：行74 の `KABOCHA` を `OTAKU` / `NEKO` / `OBAKE` に変更
- **学ぶ**：変数と配列アクセス（enum→配列インデックス）

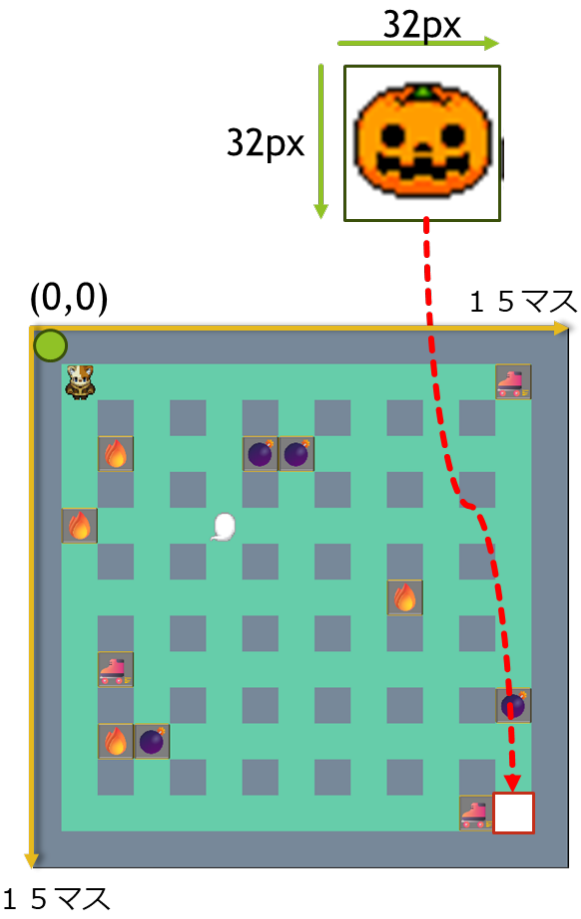
その２．初期出現位置を変更してみよう

初期出現位置の設定

初期位置をステージの右下に設定してみよう！
キャラクターは、座標系に設置されます。（座標も**変数**で表すよ！）
2Dならxy平面、3Dならxyz空間に(10,20)とか(40,50,1)のように設定されます。



座標は、x座標とy座標の値を持っています。
初期位置は、ちょっとした計算によって以下のように設定されています。



- 実際のx座標　＝　x方向に何マス目？　×　キャラ幅
- 実際のy座標　＝　y方向に何マス目？　×　キャラ高さ
  - （一番左は０マス目って数えることに注意！）

x, y方向に何マス目は２６行目辺りで設定されているよ！

書き換えてみよう

> **行番号：26行目を見て書き換え**

```
24  //#02 敵の初期出現位置を変更してみよう
25  //ステージの右下に設定するには？
26  const Pointf INIT_POS{ 5, 5 };  // #02

//２６行目を書き換えると、６６行目の計算に反映される
66  pos_ = { INIT_POS.x * CHA_WIDTH, INIT_POS.y * CHA_HEIGHT };
```

- **やること**：行26の `5, 5` を `書き換える`
  - ステージの大きさは、どんな変数で表されているかな？（`STAGE_WIDTH, STAGE_HEIGHT`）
  - 一番外側は壁だから、計算すると。。。
- **学ぶこと**：定数、算術演算（掛け算）、座標→ピクセル変換

その３．初期進行方向を変更してみよう

次の目標

次の目標として、ステージの右下から、ステージの左端まで移動させたい
Enemy飲む気はどのように決められているのか見てみよう。（どうせ方向も変数なんでしょ！）

方向を表す変数

初期方向はINIT_DIRという変数に設定されています。

```
//#03 敵の初期進行方向を変更してみよう
// 色々試して、最後は左に設定しよう
const DIR INIT_DIR = UP;
```

- **やること**：行30の `UP` を変更して、左向きにする
- **学ぶこと**：if文による条件分岐

その４．速度を与えて動かそう

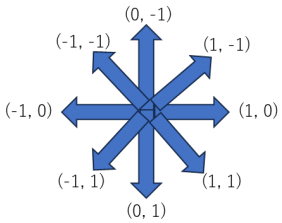
ゲームの移動処理の基本

ゲームのキャラクタは、方向を示す**ベクトル**と、**スピード**、**移動時間（フレーム間時間）**で、次のフレームの位置を計算します。
これは、小学生の時に習った、

$\text{移動距離} = \text{速度} \times \text{時間}$

の式に方向がついたものを使います。って言っても、

移動に、方向が加わるだけであとは小学校で習ったのと一緒です。
上の図でいうと、
AからBに１フレームで距離１６移動している
スピードは？
下の図のようにx軸方向、ｙ軸方向にはどのぐらい移動してるかな？
みたいな話です。



ゲームのフレーム（画面更新）の仕組み

ゲームは1秒間に何回画面を更新している？
一般的には **60FPS (Frame Per Second)** が標準です。FPSとは「1秒あたりのフレーム数」を表し、フレームとは「画面を1回描き直すこと」です。

- **フレーム更新の流れ**
  1. キーボードやゲームパッドからの入力を取得
  2. ゲーム内のキャラクターや敵などの状態を計算（移動や当たり判定）
  3. 画面に描画する（スプライトや3Dモデルを GPU へ指示）
  4. GPUがフレームバッファに描いた結果をディスプレイに表示
- **ΔTime（デルタタイム）**
  フレーム間の時間差（秒）を取得し、移動距離やアニメーションを滑らかにします。
  例：1秒間に60回更新するなら、ΔTime = 1/60 ≒ 0.0167秒。
- **なぜ60FPS？**
  - 人間の目がスムーズに映像と認識するには、30FPS以上あればOKですが、60FPSにするとより滑らかに感じられます。
  - ハードウェアやゲームの快適さを考慮したバランスです。

方向とフレーム間時間を考慮した移動処理

```
Pointf npos = {
    pos_.x + move.x * speed_ * Time::DeltaTime() ,
    pos_.y + move.y * speed_ * Time::DeltaTime()
};
```

- move.x →　右か左か？(右：1、左：-1 画面の左がｘ座標の＋方向だから)
- move.y →　上か下か？(上：-1、下：1　画面の下がy座標の＋方向だから)
- speed_　→　移動スピード、１秒間の移動量、単位はピクセル（px）
- Time::DeltaTime()　→　フレーム間の経過時間（60フレームなら１/60秒）

１フレームごとにこんな感じに、計算していきます。

速度の変数を変更する

向きが決まったら、その方向にEnemyを進めてみよう。
現在は、（わざとらしく）スピード設定が０になっている。

> **行番号：16 行目を見て書き換え**

```
14  //#04 敵の移動速度を変更してみよう
15  //ちょうどよい速さはどのぐらいかな？
16  const float SPEED = 0.0f;  // #04
```

- **やること**：行16 の `0.0f` を変更する
- **学ぶこと**：フレームの原理、時間差分（ΔTime）による移動量計算

その５．キャラクターAI（移動処理の追加）

移動パターンを考える

ゲームのキャラクタは、ゲーム内で得た情報を使っていろいろな情報を使って、その場面場面に適した動きを切り替えます（**状態遷移**）これらを自動的に行うのがAIの仕事になります。

- **状態管理**：`enemyState_` による生存／死亡フェーズ切替
- **判定ロジック**：`isHitWa``ll()` や `isHitPlayer()` で環境を認識
- **行動関数**：`TurnRight()`, `ChasePlayer()` など“何をするか”を切り出し
- **更新サイクル**：`UpdateEnemyAlive()` → 移動 or ターン → アニメーション更新

簡単な移動のパターンを考えてみよう

- 往復運動
- 外周を回る
- ランダムに動く

現在のEnemyは、プレイヤーの情報などを知る手段がないので、自分ができる範囲（自分の位置、ステージの情報）を使ってできる移動はこのような感じになると思います。

往復運動

Enemyは、移動していて外周にあたった、という情報のみを知ることができます。
isHitWall()が
true(真）のとき ：壁にぶつかった
false(偽）のとき：ぶつかってない

前に出てきた**条件分岐の処理**を応用することで、往復運動を実現できます。

1. **移動方向** `forward_` を持ち、その方向に毎フレーム進む
2. **壁判定**：`isHitWall()` が true になると「壁に当たった」と認識
3. **反転（180度ターン）**：現在の方向を逆向きに変更
4. **継続移動**：逆向きのまま動き続け、再び壁に当たるまで直進

つまり、右に進んで壁にぶつかったら左に進み、左の壁にぶつかったらまた右に進む、をくり返すことで**往復運動**を実現します。

反転の処理

> **行番号：206行目を見て書き換え**

```
// #05 方向転換ロジック
void Enemy::Trurn180()
{
 if (forward_ == UP)
  forward_ = NONE;
 else if (forward_ == LEFT)
  forward_ = NONE;
 else if (forward_ == DOWN)
  forward_ = NONE;
 else if (forward_ == RIGHT)
  forward_ = NONE;

}
```

- **やること**：206–217 行目の `NONE` に反転方向を書く！
- **学ぶ**：関数定義、if-else による分岐ロジック

> **行番号：53 行目を見て有効化**

```
51  //#06 敵の進行方向を変える関数を実装しよう
52  // 外周を回ってみよう！壁に当たったら方向転換。
53  if (isHitWall_) { TurnRight(); }  // #06
```

- **やること**：コメントアウトされている場合は外し、壁判定後に `Turn180()` が呼ばれるようにしよう
- **学ぶ**：ループ内判定、AI の行動サイクル（判定→行動→移動）

応用

同様に、TurnLeftかTurnRightを使って、外周を回り続けるには？

Revision #10
★Created 26 July 2025 07:16:33 by youe2
Updated 12 June 2026 09:23:36 by youe2