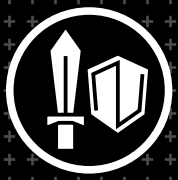
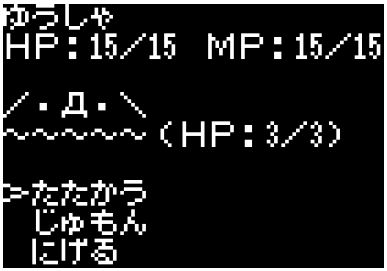
第1章 王道RPGの戦闘シーンを作成する

コマンド選択とメッセージ表示によるターン制バトル



テキストベースで進行するRPGの戦闘シーン



■本章のゲームの画面

本章では、テキストベースで進行するRPGの戦闘シーンを作成します。プレイヤーのコマンドを選択し、プレイヤーと敵が交互に行動するまでを1ターンとして、それを決着が付くまで繰り返します。

この戦闘システムは、ビデオゲーム黎明期（れいめい）の1981年に登場した海外のPCゲーム『ウィザードリィ』から始まり、1986年にファミリーコンピュータ用に発売されたRPG『ドラゴンクエスト』に採用され、それが爆発的にヒットしたことにより、国内で定着しました。

RPGで最も盛り上がる場面と言えば、ラスボスとのラストバトルでしょう。ラスボスとの戦闘は、HPを回復しながらの長期戦が基本です。本章のゲームでは、このラスボス戦が成立する最低限の仕様として、回復呪文を実装します。

プログラムの基本構造を作成する

プログラムのベース部分を作成する

最初に、ソースファイルのどこに何を記述するかを、コメントとして記述しておきます。

```
// [1]ヘッダーをインクルードする場所

// [2]定数を定義する場所

// [3]列挙定数を定義する場所

// [4]構造体を宣言する場所

// [5]変数を宣言する場所

// [6]関数を宣言する場所
```

プログラムの実行開始点である main() 関数を宣言します。

```
// [6]関数を宣言する場所

// [6-6]プログラムの実行開始点を宣言する
int main()
{
}
```

実行すると、ウィンドウが一瞬表示されて終了してしまいます。

次に、戦闘シーンの処理を記述する関数 Battle を宣言します。

```
// [6]関数を宣言する場所

// [6-4]戦闘シーンの関数を宣言する
void Battle()
{
}

...
```

main() 関数から、戦闘シーンの関数 Battle を呼び出します。

```
// [6-6]プログラムの実行開始点を宣言する
int main()
{
    // [6-6-3]戦闘シーンの関数を呼び出す
    Battle();
}
```

これで、ゲームが開始されると戦闘シーンの関数が呼び出されます。

次に、コンソールがすぐに閉じてしまわないように、キーボード入力待ち状態にします。まずその処理に必要な、コンソール入出力ヘッダー <conio.h> をインクルードします。

```
// [1]ヘッダーをインクルードする場所

#include <conio.h>  // [1-5]コンソール入出力ヘッダーをインクルードする
```

戦闘シーンの関数 Battle で、キーボード入力待ち状態にします。

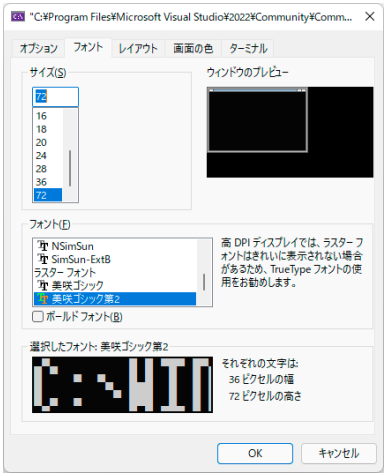
```
// [6-4]戦闘シーンの関数を宣言する
void Battle()
{
    // [6-4-6]キーボード入力を待つ
    _getch();
}
```

■コンソールが表示され続ける

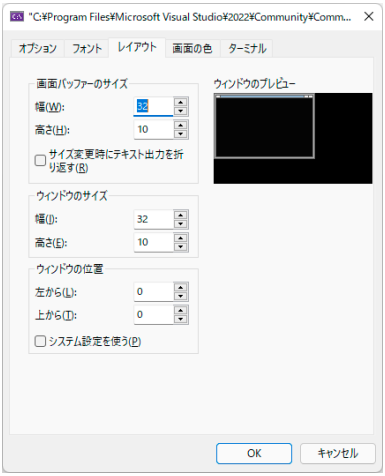
実行するとキーボード入力待ち状態になり、コンソールが表示され続けます。キーボードを押すと、キーボード入力待ち状態が終了し、プログラムも終了します。

コンソールの設定

コンソールのプロパティは、フォントのサイズを72、画面バッファーとウィンドウの幅を32、高さを10に設定します。



■フォントの設定



■レイアウトの設定

キャラクターのステータスを表示する

最初に、画面の上のほうにプレイヤーのステータスを表示します。

プレイヤーのステータスを作成する

キャラクターのデータを保持する構造体 `CHARACTER` を宣言します。メンバー変数の `hp` はHP、`maxHp` はHPの最大値、`mp` はMP、`maxMp` はMPの最大

値、name はキャラクターの名前です。文字列の配列 name の「 4 * 2 + 1 」というサイズは、「(4文字)×(全角文字の2バイト)+(文字列終了コード1バイト)」ということです。

```
// [4]構造体を宣言する場所

// [4-1]キャラクターの構造体を宣言する
typedef struct {
    int hp;                 // [4-1-1]HP
    int maxHp;              // [4-1-2]最大HP
    int mp;                 // [4-1-3]MP
    int maxMp;              // [4-1-4]最大MP
    char name[4 * 2 + 1];   // [4-1-6]名前
}CHARACTER;
```

モンスターの種類を定義します。プレイヤーにもモンスターと同様のステータスがあるので、モンスターの一種として管理します。

```
// [3]列挙定数を定義する場所

// [3-1]モンスターの種類を定義する
enum
{
    MONSTER_PLAYER, // [3-1-1]プレイヤー
    MONSTER_MAX     // [3-1-4]モンスターの種類の数
};
```

モンスターの初期データを保持する配列 monsters を宣言し、プレイヤーの初期ステータスを設定します。

```
// [5]変数を宣言する場所

// [5-1]モンスターのステータスの配列を宣言する
CHARACTER monsters[MONSTER_MAX] =
{
    // [5-1-1]MONSTER_PLAYER    プレイヤー
    {
        15,         // [5-1-2]int hp                HP
        15,         // [5-1-3]int maxHp             最大HP
        15,         // [5-1-4]int mp                MP
        15,         // [5-1-5]int maxMp             最大MP
        "ゆうしゃ", // [5-1-7]char name[4 * 2 + 1]  名前
    },
};
```

戦闘に登場するキャラクターの種類を定義します。

```
// [3]列挙定数を定義する場所
...

// [3-2]キャラクターの種類を定義する
enum
{
    CHARACTER_PLAYER,    // [3-2-1]プレイヤー
    CHARACTER_MONSTER,   // [3-2-2]モンスター
    CHARACTER_MAX        // [3-2-3]キャラクターの種類の数
};
```

キャラクターのデータを保持する配列 `characters` を宣言します。

```
// [5]変数を宣言する場所
...

// [5-2]キャラクターの配列を宣言する
CHARACTER characters[CHARACTER_MAX];
```

プレイヤーのステータスを初期化する

プレイヤーのデータを初期化するために、ゲームを初期化する処理を記述する関数 `Init` を宣言します。

```
// [6]関数を宣言する場所

// [6-1]ゲームを初期化する関数を宣言する
void Init()
{
}

...
```

ゲームを初期化する関数 `Init` を、`main()` 関数から呼び出します。

```
// [6-6]プログラムの実行開始点を宣言する
int main()
{
    // [6-6-2]ゲームを初期化する関数を呼び出す
    Init();

    ...
}
```

これで、ゲームが起動したらゲームを初期化する関数 `Init` が呼び出されるようになります。

次に、ゲームを初期化する関数 `Init` で、プレイヤーのステータスを初

期化します。

```
// [6-1]ゲームを初期化する関数を宣言する
void Init()
{
    // [6-1-1]プレイヤーのステータスを初期化する
    characters[CHARACTER_PLAYER] = monsters[MONSTER_PLAYER];
}
```

これで、ゲームが起動するとプレイヤーの初期ステータスが設定されます。

プレイヤーのステータスを表示する

プレイヤーのステータスを表示します。ステータスが変化するごとに再描画が必要になるので、基本部分を描画する処理を関数にしておきます。戦闘シーンの画面の基本的な部分を描画する関数 `DrawBattleScreen` を宣言します。

```
// [6]関数を宣言する場所
...

// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
}

...
```

戦闘シーンの関数 `Battle` から、画面の基本部分を描画する関数 `DrawBattleScreen` を呼び出します。

```
// [6-4]戦闘シーンの関数を宣言する
void Battle()
{
    // [6-4-4]戦闘シーンの画面を描画する関数を呼び出す
    DrawBattleScreen();

    ...
}
```

文字列を表示するために、標準入出力ヘッダー<stdio.h>をインクルードします。

```
// [1]ヘッダーをインクルードする場所

#include <stdio.h>  // [1-1]標準入出力ヘッダーをインクルードする
#include <conio.h>  // [1-5]コンソール入出力ヘッダーをインクルードする
```

基本部分を描画する関数DrawBattleScreenで、プレイヤーの名前を表示します。

```
// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
    // [6-2-2]プレイヤーの名前を表示する
    printf("%s¥n",characters[CHARACTER_PLAYER].name);
}
```

■プレイヤーの名前が表示される

実行すると、プレイヤーの名前が表示されます。

次に、プレイヤーのステータスも表示します。

```
// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
    ...

    // [6-2-3]プレイヤーのステータスを表示する
    printf("ＨＰ：%d／%d　ＭＰ：%d／%d¥n",
        characters[CHARACTER_PLAYER].hp,
        characters[CHARACTER_PLAYER].maxHp,
        characters[CHARACTER_PLAYER].mp,
        characters[CHARACTER_PLAYER].maxMp);
}
```

■プレイヤーのステータスも表示される

実行すると、プレイヤーのステータスも表示されます。

最後に、次の表示に備えて1行空けておきます。

```
// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
    ...

    // [6-2-4]1行空ける
    printf("¥n");
}
```

モンスターのステータスを作成する

モンスターの種類として、雑魚モンスターのスライム `MONSTER_SLIME` を追加します。

```
// [3-1]モンスターの種類を定義する
enum
{
    MONSTER_PLAYER, // [3-1-1]プレイヤー
    MONSTER_SLIME,  // [3-1-2]スライム
    MONSTER_MAX     // [3-1-4]モンスターの種類の数
};
```

スライムのステータスを設定します。

```
// [5-1]モンスターのステータスの配列を宣言する
CHARACTER monsters[MONSTER_MAX] =
{
    ...

    // [5-1-8]MONSTER_SLIME スライム
    {
        3,          // [5-1-9]int hp                HP
        3,          // [5-1-10]int maxHp            最大HP
        0,          // [5-1-11]int mp               MP
        0,          // [5-1-12]int maxMp            最大MP
        "スライム", // [5-1-14]char name[4 * 2 + 1] 名前
    },
};
```

キャラクターの構造体 `CHARACTER` に、アスキーアートを保持するメンバー変数 `aa` を追加します。

```
// [4-1]キャラクターの構造体を宣言する
typedef struct {
    ...
    char aa[256];   // [4-1-7]アスキーアート
}CHARACTER;
```

スライムのデータに、アスキーアートを設定します。「Д」はキリル文字で「デー」と読みます。

```
// [5-1-8]MONSTER_SLIME スライム
{
    ...

    // [5-1-15]char aa[256] アスキーアート
    "／・Д・＼¥n"
    "〜〜〜〜〜"
```

```
},
```

これでスライムのデータができました。

モンスターのステータスを初期化する

戦闘シーンの関数 `Battle` を呼び出すときに、モンスターの種類を指定できるようにします。そこで、関数 `Battle` の引数に、モンスターを指定する引数 `_monster` を追加します。

```
// [6-4]戦闘シーンの関数を宣言する
void Battle(int _monster)
{
    ...
}
```

戦闘シーンの関数 `Battle` を呼び出すときに、スライム `MONSTER_SLIME` を設定します。

```
// [6-6]プログラムの実行開始点を宣言する
int main()
{
    ...

    // [6-6-3]戦闘シーンの関数を呼び出す
    Battle(MONSTER_SLIME);
}
```

戦闘シーンの関数 `Battle` に入ったら、モンスターのデータに指定されたモンスターのデータをコピーします。

```
// [6-4]戦闘シーンの関数を宣言する
void Battle(int _monster)
{
    // [6-4-1]モンスターのステータスを初期化する
    characters[CHARACTER_MONSTER] = monsters[_monster];

    ...
}
```

これで、戦闘が開始するときに指定したモンスターのデータが設定されます。

モンスターを表示する

基本部分を描画する関数で、モンスターのアスキーアートを描画します。

```
// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
    …

    // [6-2-5]モンスターのアスキーアートを描画する
    printf("%s", characters[CHARACTER_MONSTER].aa);
}
```

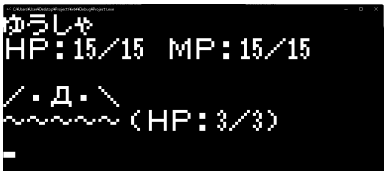
■モンスターのアスキーアートが表示される

実行すると、モンスターのアスキーアートが表示されます。

デバッグのために、モンスターの横にHPを表示します。

```
// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
    …

    // [6-2-6]モンスターのＨＰを表示する
    printf("（ＨＰ：%d／%d）¥n",
        characters[CHARACTER_MONSTER].hp,
        characters[CHARACTER_MONSTER].maxHp);
}
```



■モンスターのHPも表示される

実行すると、モンスターの右にHPが表示されます。

これでモンスターの表示ができました。最後に、メッセージの表示に備えて、1行空けておきます。

```
// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
    …

    // [6-2-7]1行空ける
    printf("¥n");
}
```

戦闘の流れを作成する

プレイヤーとモンスターが交互に攻撃し合う、戦闘の基本的な流れを作成します。

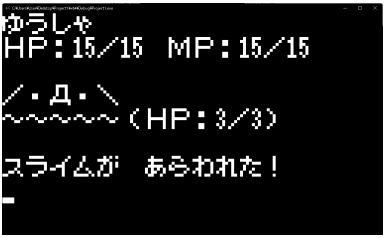
戦闘開始のメッセージを表示する

戦闘シーンの関数の最初に、モンスターと遭遇したメッセージを表示します。

```
// [6-4]戦闘シーンの関数を宣言する
void Battle(int _monster)
{
    ...

    // [6-4-5]戦闘シーンの最初のメッセージを表示する
    printf("%sが　あらわれた！¥n", characters[CHARACTER_MONSTER].name);

    ...
}
```



■戦闘の最初のメッセージが表示される

実行すると、モンスターと遭遇したメッセージが表示されます。

コマンドのデータを作成する

戦闘でプレイヤーとモンスターが選択し得るコマンドの種類を定義します。

```
// [3]列挙定数を定義する場所
...

// [3-3]コマンドの種類を定義する
enum
{
    COMMAND_FIGHT,  // [3-3-1]戦う
    COMMAND_SPELL,  // [3-3-2]呪文
    COMMAND_RUN,    // [3-3-3]逃げる
    COMMAND_MAX     // [3-3-4]コマンドの種類の数
};
```

キャラクターの構造体 CHARACTER に、現在選択中のコマンドを保持するメンバー変数 command を追加します。

```
// [4-1]キャラクターの構造体を宣言する
typedef struct {
    ...
    int command;    // [4-1-8]コマンド
}CHARACTER;
```

各キャラクターに攻撃をさせる

戦闘シーンの関数 Battle で、戦闘が終了するまで無限ループに入ります。

```
// [6-4]戦闘シーンの関数を宣言する
void Battle(int _monster)
{
    ...

    // [6-4-7]戦闘が終了するまでループする
    while (1)
    {
    }
}
```

戦闘シーンのループの中で、プレイヤーとモンスターを反復します。

```
// [6-4-7]戦闘が終了するまでループする
while (1)
{
    // [6-4-9]各キャラクターを反復する
    for (int i = 0; i < CHARACTER_MAX; i++)
    {
    }
}
```

各キャラクターがどのコマンドを選択しているかで、処理を分岐させます。

```
// [6-4-9]各キャラクターを反復する
for (int i = 0; i < CHARACTER_MAX; i++)
{
    // [6-4-11]選択されたコマンドで分岐する
    switch (characters[i].command)
    {
    case COMMAND_FIGHT: // [6-4-12]戦う
        break;

    case COMMAND_SPELL: // [6-4-22]呪文
        break;
```

```
    case COMMAND_RUN:   // [6-4-35]逃げる
        break;
    }
}
```

現状ではキャラクターデータがクリアされた状態なので、コマンドは0番目の「戦う」コマンドが選ばれています。「戦う」コマンドが選ばれていれば攻撃をするメッセージを表示して、キーボード入力待ち状態にします。

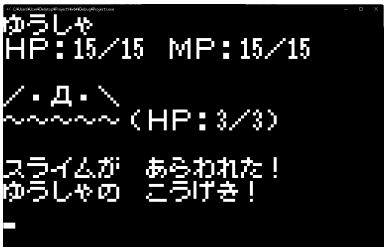
```
// [6-4-11]選択されたコマンドで分岐する
switch (characters[i].command)
{
case COMMAND_FIGHT: // [6-4-12]戦う

    // [6-4-13]攻撃をするメッセージを表示する
    printf("%sの　こうげき！¥n", characters[i].name);

    // [6-4-14]キーボード入力を待つ
    _getch();

    break;

...
}
```



■攻撃のメッセージが表示される

実行すると攻撃のメッセージが表示されますが、最初のエンカウントメッセージが残ったままです。

このメッセージを消すために、行動するキャラクターが切り替わるごとに画面を再描画します。

```
// [6-4-9]各キャラクターを反復する
for (int i = 0; i < CHARACTER_MAX; i++)
{
    // [6-4-10]戦闘シーンの画面を描画する関数を呼び出す
    DrawBattleScreen();

    ...
}
```

実行すると、画面が連続で表示されて乱れてしまいます。そこで、画面をクリアするために、標準ライブラリヘッダー<stdlib.h>をインクルードします。

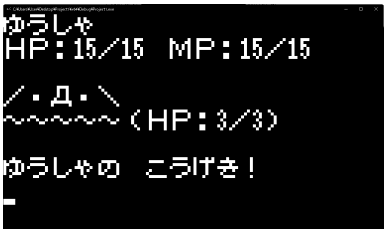
```
// [1]ヘッダーをインクルードする場所

#include <stdio.h>  // [1-1]標準入出力ヘッダーをインクルードする
#include <stdlib.h> // [1-2]標準ライブラリヘッダーをインクルードする
#include <conio.h>  // [1-5]コンソール入出力ヘッダーをインクルードする
```

戦闘シーンの画面を描画する関数 `DrawBattleScreen` の最初で、画面をクリアします。

```
// [6-2]戦闘シーンの画面を描画する関数を宣言する
void DrawBattleScreen()
{
    // [6-2-1]画面をクリアする
    system("cls");

    ...
}
```



■エンカウントメッセージが表示されなくなる

実行すると、今度は攻撃のメッセージのみが表示されます。キーボード入力で進めると、プレイヤーとモンスターが交互に攻撃コマンドを実行します。これで、プレイヤーとモンスターが交互に行動するという戦闘シーンの基本的なループができました。

コマンド選択インターフェイスを作成する

プレイヤーのコマンドを選択するインターフェイスを作成します。

コマンドを選択する処理を呼び出す

プレイヤーのコマンドを選択する関数 `SelectCommand` を宣言します。

```
// [6]関数を宣言する場所
...

// [6-3]コマンドを選択する関数を宣言する
```