制作演習_課題 02_04

落ち物ゲームのプログラムのリファクタリングを行います。

1. リファクタリングとは？

ソフトウェアの外部仕様を変更せずに内部仕様を変更することです。

→要するに？

→ユーザーが遊んだ時の、ゲームの動作は一切変更せずに、プログラムの設計や実装だけ変更すること。ユーザーからすると何も変わらない。

→なんのためにするの？

→プログラムの設計や実装を整えることによって、続編の製作や、バージョンアップなどのときに、プログラムの変更を行うのが容易になるから。

2. 今回のリファクタリングの内容

前回の製作で、落ち物ゲームにプレイヤーとプレイヤー２のプログラムが追加されましたが、下記のように同じようなコードが複製されており、冗長なコードになっています。

制作演習_課題 02_04/otimono/otimono/Source.cpp

```
//プレイヤー１の移動処理。
PlayerMove(player, 0);

//落ち物との衝突判定。
for (int i = 0; i < otimonoArray.size(); i++) {
        if ( otimonoArray[i].posX == player.posX
                && otimonoArray[i].posY == player.posY
        ) {
                otimonoArray[i].dead = 1;
                player.score++;
        }
}
//プレイヤーを描画する。
kbcDrawMoji(player.posX, player.posY, 'P');

//続いてプレイヤー２の移動処理。
PlayerMove(player2, 1);

//落ち物との衝突判定。
for (int i = 0; i < otimonoArray.size(); i++) {
        if (otimonoArray[i].posX == player2.posX
                && otimonoArray[i].posY == player2.posY
                ) {
                otimonoArray[i].dead = 1;
                player2.score++;
        }
}
//プレイヤーを描画する。
kbcDrawMoji(player2.posX, player2.posY, 'L');
```

同じようなコードが書かれている・・・

このようなコードになっていると、仕様変更が発生したときに、例えば、落ち物を取ったときにサウンドを鳴らすといった仕様変更が起きた時に、下記のように２か所修正する必要が出てきます。

```
//プレイヤー１の移動処理。
PlayerMove(player, 0);

//落ち物との衝突判定。
for (int i = 0; i < otimonoArray.size(); i++) {
        if ( otimonoArray[i].posX == player.posX
                && otimonoArray[i].posY == player.posY
        ) {
                otimonoArray[i].dead = 1;
                player.score++;
                PlayStarGetSound();
        }
}
//プレイヤーを描画する。
kbcDrawMoji(player.posX, player.posY, 'P');

//続いてプレイヤー２の移動処理。
PlayerMove(player2, 1);

//落ち物との衝突判定。
for (int i = 0; i < otimonoArray.size(); i++) {
        if (otimonoArray[i].posX == player2.posX
                && otimonoArray[i].posY == player2.posY
                ) {
                otimonoArray[i].dead = 1;
                player2.score++;
                PlayStarGetSound();
        }
}
//プレイヤーを描画する。
kbcDrawMoji(player2.posX, player2.posY, 'L');
```

２か所修正する必要がある。

これが２か所だけであれば、大した手間ではありませんが、10 か所、20 か所と増えていくとどうでしょうか？

そこで、このような共通処理は関数化できないかを検討してみるべきです。今回の落ち物との衝突判定は関数化することができます。

衝突判定を関数化して、main 関数は下記のように落ち物との衝突判定をまとめた関数を呼び出すだけになるように、リファクタリングを行ってください。

```
//プレイヤー１の移動処理。
PlayerMove(player, 0);
//落ち物との衝突判定。
CheckHitPlayerAndOtimono(player);
//プレイヤー１を描画する。
kbcDrawMoji(player.posX, player.posY, 'P');

//続いてプレイヤー２の移動処理。
PlayerMove(player2, 1);
//落ち物との衝突判定。
CheckHitPlayerAndOtimono(player2);
//プレイヤーを描画する。
kbcDrawMoji(player2.posX, player2.posY, 'L');
```