沼田研究室 C++/C#コーディング規約

1. はじめに

プロジェクトのすべての参加者が同じレベルのソースコードを共有できるように、このコーディング規約に従ってプログラムを作成してください。

2. インデントについて

インデントは半角スペース 4 文字

インデントの基本は半角スペース４個分とします。波括弧 '{' を書くごとに改行を入れ、その後はインデントを１つ分空けてください。

```
void DoSomething()
{
□□□□for (int i = 0; i < 10; i++) {
□□□□□□□□Console.WriteLine("Hello!!");
□□□□}
}
```

タブ文字は、エディタの設定で使わない設定にしておいてください。タブ文字は環境によって４文字分になったり８文字分になったりするので、環境に応じて見た目が変わるのを防いでおきます。

▶ Xcode の場合、メニューの[Xcode]-[Preferences...]から、[Text Editing]の[Indentation]タブを選択し、「Prefer indent using」を「Spaces」にする。

▶ Visual Studio の場合、メニューの[Visual Studio Community]-[ユーザー設定...]から、[ソースコード]-[コードのフォーマット]-[C#ソースコード]を選択し、「空白 タブ幅」と「空白 インデント幅」をどちらも４に設定し、「テキストのスタイル」の「タブをスペースに変換」にチェックを入れる。

波括弧と丸括弧のインデント

クラスおよびメソッドの実装では、引数宣言の丸括弧を閉じたあとに改行を入れ、次の行の先頭に波括弧を書き、すぐに改行・インデントして実装コードを書きます。

```
class MyClass
{
    void DoSomething()
    {
        // 実装コード
    }
}
```

if, for, while, switch 文などの制御文では、必ず条件記述の丸括弧の前後に半角スペースを１個空け、行を変えずに波括弧を書き、その次の行からブロック内に処理内容を記述してください。条件記述の開き丸括弧'('の直後と、閉じ丸括弧')'の直前には、半角スペースを入れてはいけません。

```
// こう書いてください（良い例）
while (count < 10) {
    count++;
}
```

```
// 制御文の波括弧は行を変えて書いてはいけない。（悪い例）
while (count < 10)
{
```

```
    count++;
}
```

```
// 丸括弧の前後に半角スペースが空いていない。（悪い例）
// 丸括弧の中にも半角スペースが入っている。（悪い例）
while( count < 10 ){
    count++;
}
```

条件記述の丸括弧のあとには必ず波括弧を書き、ブロック内に処理内容を記述するようにしてください。ブロックを書くことで、制御文の対象となる範囲が明確になり、バグが入りにくくなります。

```
// こう書いてください（良い例）
while (count < 10) {
    count++;
}
```

```
// 制御文の対象を波括弧でくくっていない（悪い例）
while (count < 10)
    count++;
```

演算子のインデント

比較演算子や代入演算子などの二項演算子の前後には、半角スペースを 1 個分空けてください。三項演算子は「?」「:」の直後に半角スペースを 1 個分空けます。

```
// こう書いてください（良い例）
if (count == 0) {
    count += 10;
    ret = isHeavy? x: (x + 30);
}
```

```
// 二項演算子の前後に半角スペースがない（悪い例）
if (count==0) {
    count+=10;
    ret=isHeavy?x:(x+30);
}
```

メソッド呼び出しと区切り文字とインデント

メソッド呼び出しは、丸括弧を書いたあとスペースを空けずに書き始め、スペースを空けずに丸括弧を閉じてください。引数が複数ある場合、区切りのためのコンマ (,) の前にはスペースを空けず、後ろに半角スペース 1 個を空けてください。また、関数名やメソッド名と丸括弧の間にはスペースを入れないでください。for 文の文区切りのためのセミコロン (;) についても、引数のコンマと同様です。

```
// こう書いてください（良い例）
Move(3, 4);
```

```
// こうは書かない（悪い例）
Move (3,4);
Move(3 , 4);
Move( 3, 4 );
```

```
// こう書いてください（良い例）
for (int i = 0; i < 10; i++) {
    // ...
}
```

```
// こうは書かない（悪い例）
for ( int i = 0 ; i < 10 ; i++ ) {
    // ...
}
```

ファイル末尾の改行

スクロールした時に最後の行が見やすいように、すべてのファイルの最後に、３行以上の改行を入れておいてください。

```
    ...
}
↲
↲
↲
```

3. プログラムの構造について

++と--について

「count++」や「count--」などのインクリメント・デクリメントの式を対象とした直接の比較は、できるだけ行わないようにしてください。

```
// 条件比較と同時に値の変更を記述しています（悪い例）
if (count-- == 10) {
    // ...
}
```

```
// こう書いてください。条件比較した後、値を変更しています（良い例）
if (count == 10) {
    // ...
}
count--;
```

前置型の「++count」や「--count」は、原則として使わないでください。また、「++」や「--」と変数名の間には、スペースを空けないでください。

条件の記述方法

bool 型の変数や bool 型の戻り値の関数・メソッドを条件判定に使用する場合、true や false の定数と「==」演算子や「!=」演算子で比較しないでください。

```
// こう書いてください（良い例）
if (obj.CheckFighting()) {
    // ...
}
```

```
// bool 値を「true」や「false」と比較してはいけません（悪い例）
if (obj.CheckFighting() == true) {
    // ...
}
```

論理否定をとるために不要な丸括弧を挿入するのは、避けてください。

```
// こう書いてください（良い例）
if (!obj.IsFighting) {
    // ...
}
```

```
// 不必要な丸括弧を付けたり、「true」や「false」と比較したものの
// 否定を取ってはいけません（悪い例）
if (!(obj.IsFighting)) {
    // ...
}
if (!(obj.IsFighting == true)) {
    // ...
}
```

4. コメントについて

ドキュメンテーション・コメント

クラス・構造体・列挙型・メソッドなどの詳細な説明を記述するために、ドキュメンテーション・コメントを積極的に書いてください。

▶ Xcode を主に使用する場合は、「Markup Formatting Reference」(https://developer.apple.com/library/archive/documentation/Xcode/Reference/xcode_markup_formatting_ref/)を参考にコメントを書いてください。

▶ Visual Studio を主に使用する場合は、「XML コメントによるコードの文書化」(https://docs.microsoft.com/ja-jp/dotnet/csharp/codedoc)を参考にコメントを書いてください。

TODO コメントと FIXME コメント

実装が必要な箇所には「// TODO: ○○」という形式でコメントを書き、修正が必要な箇所には「// FIXME: ○○」という形式でコメントを書いておきます。あとで「TODO」や「FIXME」を全文検索して利用します。

```
void DrawSpecialEffect()
{
    // TODO: DrawSpecialEffect メソッドを実装する。
}
```

```
void DrawCharacters()
{
    // FIXME: 敵と自機の描画順序が逆になっているのを修正する。
    ...
}
```

実装のないメソッドにもコメントを入れる

実装コードが不要な関数やメソッドがある場合には、実装途中でないことを明示するために、「// Do nothing」というコメントを必ず入れておいてください。

```
void DoSomething()
{
    // Do nothing
}
```

ソースコードを修正したらコメントも修正する

コメントが書かれている箇所の実装コードを修正したら、必ず忘れずにコメントも修正してください。

5. 命名規則について

英語への準拠

クラス名・構造体名・列挙型名・メソッド名・変数名・定数名などは、できるだけ省略しない正しい英単語で構成してください（**辞書を引いて確かめること！**）。ただし、非常に一般的ではないと考えられる単語しかない場合は、ヘボン式ローマ字記述の日本語（し→SHI、しゃ→SHA、つ→TSU、ち→CHI、ん→N (M)、じ→JI、じょ→JO、ふ→FU、ちょ→CHO、長音（ー）は省略）を使用しても構いません。

名詞群（クラス、構造体、変数、定数）

クラス名・変数名・プロパティ名・定数名は名詞として命名し、各単語の頭を大文字にしてつなげます。変数・プロパティは小文字から書き始め、それ以外のクラス名、構造体名、列挙型などは大文字から書き始めてください。

```
（形容詞 / 名詞+）名詞
    Player, Enemy, MapManager, EnemyBullet など
```

ただし、bool 型のフラグ変数やプロパティは、以下のように命名してください。こうすることで、「if (player.canMove) {...}」のように、英語の文章として意味を理解しやすくなります。

```
is + 形容詞（動詞の過去形・現在進行形を含む）（～である）
    isVisible, isJumping など
```

```
can + 動詞（+ 目的語（名詞））（+副詞）（～できる）
    canMove, canBreakWall など
```

```
has + 動詞過去分詞（すでに～した / 直前に～したところである）
    hasMoved, hasRotated など
```

```
動詞の三人称単数の現在形（ずっと～する / ～することになっている）
    hasSpecialItem, showsHintMessage など
```

リストや配列など、複数の数値やオブジェクトを格納するためのコンテナ変数の場合は、「enemies」「items」のように名詞の部分を複数形にするか、「enemyList」「itemList」のようにコンテナ名を後ろに付けます。

定数名の頭には小文字の「k」を付けてください（ハンガリアン記法の一部導入）。

```
const int kDefaultFontSize = 20;
```

特殊な変数名として、for 文に使用する変数には、i, j, k, l, m, n あるいは x, y, xx, yy, x2, y2 などの名

前を使います。また戻り値を格納しておくための変数に「ret」という名前を使うこともできます。１文字の文字を格納するための変数には「c」という名前が使えます。

テクスチャ・BGM・効果音など、同じクラスのインスタンスが多数出てくる場合には、「tex」「bgm」「se」などのプリフィクスを先頭に付けた変数名を使用してください。

```
texPlayer, texEnemy, bgmNormal, bgmDanger,
seShot, sePowerUp など
```

動詞群（関数、メソッド）

関数やメソッドの名前は動詞として命名し、先頭および各単語の頭を大文字から書き始めます。bool 値をリターンする関数については、名詞の bool 型のフラグの命名規則に倣ってください。

```
動詞（＋目的語（名詞））（＋副詞）
   Move(), Step(), MoveForward(), MoveSlowly()
   CheckHit(), PickUpFirstEnemy(), DrawCharacter() など
```

英語の省略形について

よく使われる省略形は以下のとおりですが、多くの場合、他の単語と組み合わせる場合に使われます。とくにクラス名などの型名には、ほとんどの場合、省略形は使われません。その省略形が正しいという自信がもてない場合は、省略せずにフルスペルで記述してください。

- absolute → abs
- acceleration → a / acc / accel
- alpha → a
- application → app
- argument → arg
- attribute → attr
- blue → b
- buffer → buf
- calculate → calc
- character → c / chara
- command → cmd
- context → ctx
- current → cur
- decrement → dec
- delete → del
- description → desc
- destination → dst / dest
- dictionary → dict
- directory → dir
- document → doc
- element → elem
- error → err
- extension → ext
- frequency → freq
- function → func
- green → g
- history → hist
- image → img
- implimentation → impl
- increment → inc
- information → info
- initialize → init
- manager → man
- maximum, minimum → max, min
- minute → min
- navigation → nav
- number → num
- object → obj
- operand, operation → op
- position → pos
- preference → pref
- previous → prev
- progress → prog
- random → rand
- rectangle → rect
- red → r
- request → req
- second → sec
- source → src
- string → str
- temporary → temp
- value → val
- vector → vec
- vertex → vert
- velocity → v
- version → ver
- window → win

列挙型の名前について

C++の場合、列挙型の各要素の名前には、先頭に列挙型の型名とアンダースコア(_)を付けてください。こうすることで、予測変換で補完しやすくなります。

```
例. enum EnemyType {
        EnemyType_Normal,
        EnemyType_Fire,
        EnemyType_Stone,
    };
```

C#の場合、列挙型の各要素の名前はそのまま記述します。

```
例. enum EnemyType {
        Normal,
        Fire,
        Stone,
    }
```

インタフェースの名前

インタフェースとして使用するクラスの名前は、「I」から始めます。

```
○ class ILocatable
× class Locatable
```

6. オブジェクト指向的な指針

メンバ変数は原則 private にする

原則として、メンバ変数はまず private で作成し、必要に応じて protected にしてください。構造体として利用するクラスでない限り、public なメンバ変数は作らず、getter メソッドや、それに類似した他のメソッドを用意することで代替してください。

実装コードは短く

プログラムの構成を工夫して、メソッドの実装は、できるだけ 20 行程度に収まるようにしてください。どれだけ長くなっても 100 行を超えるのは避けるように努力してください。また、変数の数もなるべく増やさないように努力してください。

付録. Visual Studio Community の設定

Visual Studio のメニューの[Visual Studio Community]-[ユーザー設定...]から、[ソースコード]-[コードのフォーマット]-[C#ソースコード]を選択し、「C#フォーマット」タブから「編集」ボタンを押すと、以下の項目について編集できる。このコーディング規約に則った設定は、以下の通りである。