げえむあきてくちゃ

表題のテキストがないから一応作っておきます。
とはいえ、Unityやら UE4やらの細かい話は別紙になるかなと思います。

はじめに

まず、これは精神面の話で、真面目にゲームプログラマを目指してる人も、そうでない人も聞いておいた方がいいかなと思うのですが、

人間というのは元来怠け者です

で、これに気づかないといとも簡単にダメ人間になってしまいます。既にダメ人間になってる人は、なかなか回復に時間がかかるのですが、誰でもそうなり得ると思っておいた方がいいです。

完全なダメ人間は「やらなきゃ」なんて言う気持ちも沸いてこないと思いますが、そうでもない**大半の人は「やらなきゃ」と思いつつやらずに時間が経過してしまう。**というのを何度も繰り返し「俺はダメな人間だァ…」となります。

で、ダメ人間だからこうなると思いがちですが、「やらなきゃ」って思ってるなら、まだ引き返せるんですよね。
で、なんで、やらなきゃと思いつつ、体が動かない、実際の行動に移せないのかというと、前述のとおり人間は怠け者です。

これは性格の問題と言うより生来そうだと思ったほうがいいです。世の中にはドーパミンでまくりのある意味あたまおかしい人がいるのですが、あれは特別な例で、大半の人は怠け者です。

これには理由があって、野生の動物は「狩りをする」「敵に襲われる」時以外はエネルギーを温存するために怠けています。猫がよく寝ているのもそうですね。野生の世界では「行動」はあっという間にエネルギーを消費してしまうからです。
そもそも狩りや敵から逃げるというのは生死にかかわるので、余計なところでエネルギーを使わないようになってるんです。

ところが人類になって、特に襲われることもなく、狩りなんてしなくても飯が食える。

でも怠け癖だけは残っちゃったんだよね。で、野生時代は怠けた結果の危機はすぐに発生してて、襲われたり飢えたりするんだけど、人間は怠けたツケが１か月後、1年後、数年後になってしまうので、見えないんですよね。

というわけで、明らかに将来にリスクがあるのに、時間を無為に過ごして、クッソ甘いジュース飲んで、バクバク食って、まぁ、ゆっくりゆっくり体を壊したりするんだよね。

で、ゲーム制作もそうですね。ほっときゃ締め切りの３か月後なんてすぐに来るのですが、それすら見誤って怠けちゃう。

そもそも最終目的の就活用の作品作りなんて、なかなか見えやしない、危機感が湧かない。
それが当たり前なのよね。

はい、じゃあその怠け者ぐせをどうすれば軽減できるのか？っていう発想になりましょう。

で、この時にやっちまう間違いの一つとしては「デカすぎる目標」です。
デカすぎる目標というのは「絶対勝てない敵」です。

野生の気持ちになって「絶対に勝てない敵」に出会ったらどうするだろうか？うん、まず逃げますね。

それと同じ行動をとります。「やらなきゃいけない！」と思いつつ忌避行動をとるのはこれです。

いきなりラスボスじゃなくて、これを四天王くらいに分割して、個別撃破するイメージで小さく分けましょう。四天王でも勝てなければもっと雑魚敵になるまで分割しましょう。で、ひとつひとつ潰しましょう。

これがタスク分割と言う奴です。

また、たまに「クソ強い雑魚」がいます。特定のRPGで発生する「デスエンカ」ってやつですが、こいつは相手にしないようにしましょう。レベルが上がってから再挑戦しましょう。

どういう事かというと、「完全な実装」をしようとすると手が止まります。例えば「美しい設計」だの「数学的に難しい」だの「実装が難しい」だのはレベルが足りずに、マヒ行動に陥ります。

なので、とりあえず「今やれる実装」で済ませておいて、後から戻って来ましょう。

例えばの話ですが、「ボタンを押した長さでジャンプの高さが変わる」という機能を実装したいとします。
しかしながら、これは初心者には少々ハードルが高く、ひとによっては現レベルでは実装できない可能性があります(あくまでも例ですよ?)。

そう言った場合は「ひとまず固定高さジャンプを実装する」や、「複数ボタンで 3 段階のジャンプを用意する」などして、まずは実装の手を止めないようにしましょう。

で、自分のレベルが上がったかなと思ってそこに戻って見直してみましょう。もしかしたらスッと実装できるかもしれません。

それか他人に頼ったりしましょう。心の中に「保留してる」という認識にしておいて、一旦そこを離れるというのは意外と重要です(諦めたじゃなくて保留ってしとくと挫折感もなく、精神的にもいい)

と、最初に精神的なところを語ってきましたが、これを話した理由は開発という作業は自分との闘いだからです。

この辺をおろそかにして根性任せでやっても息切れして失敗するし、そもそも「最初にとりかかる事」すらろくにできないだろう。

こういう人間の習性に抵抗する手段が、ガントチャートだったり、ToDo リストだったり、スケジュール帳だったりするわけだ。

毎日の作業が「惰性でも行えるレベル」になるのが理想で、一般的な会社員のお仕事はこれに近い。別に批判的な意味じゃなくて、それくらいモチベーションと言うのは長続きしないのだ。

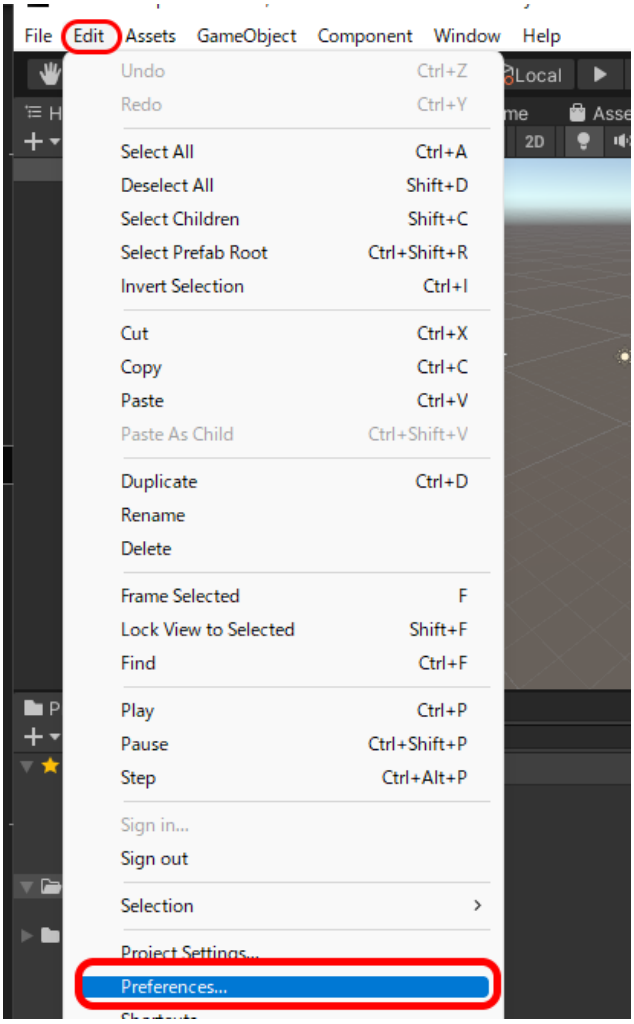
会社というのはやる気が特になくても仕事をそこそこなせる環境ができてるというわけだ。

Unity について

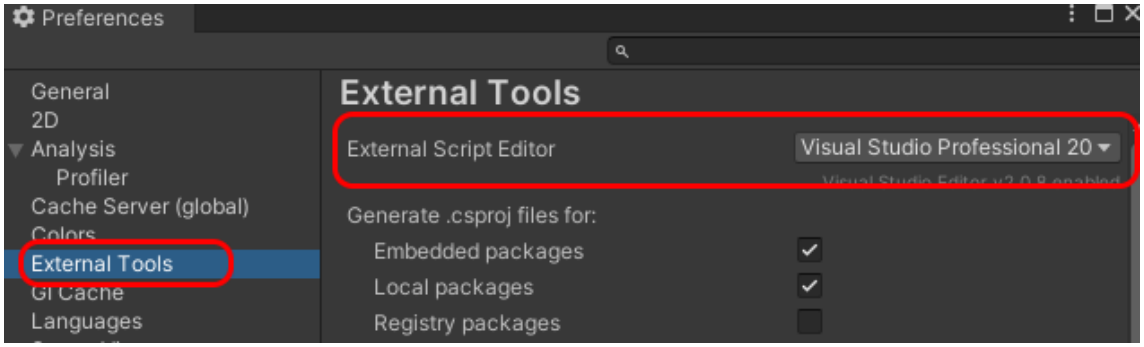
ハイなんかもう Unity 既にやってるしーという事なので、ここでは Unity のエディタ作成について書いていきましょう。

ところで、最初に Unity のエディタの連携が取れてなくて、VisualStudio のインテリセンスが動作しないという事がありました。

なので、同現象に悩まされてる人は、まずメニューの Edit→Preferences を選択



で、中に ExternalTools というタブがあるので、その中の ExternalScriptEditor を現在使用中の VisualStudio にしましょう。VS が嫌いなら、別のエディタにしてもかまいませんが、その場合はインテリセンスがどう動くのかの保証は致しません。

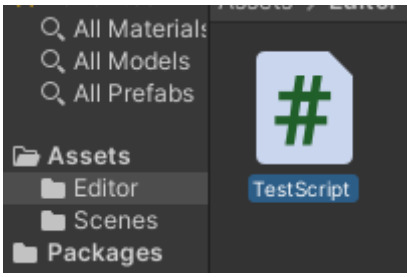


はい。とりあえず下準備はこんなもんで、さっそく Editor の話です。

UnityEditor の作り方(メニュー編)

まず、作るだけなら簡単です。Asset の中のどこでもいい(トップフォルダ直下である必要はないです)ので Editor というフォルダを作ってください。

で、その中に適当な C#スクリプトを作ります。ゲーム内オブジェクトのコンポーネントではないため MonoBehavior から継承する必要はないようです。



はい、でひとまず何かを出力するものを作ってみましょう。

テストコード(外部に何かを出力する)

なお、C#において、先頭にある using 句は、C++における include みたいなものだと思ってもらえればいい。
ネイティブの C++とはビルドの仕組みがかなり違うので「同一」とは言えないが、つまるところ、デフォルトで入っていない機能を追加するものだと思っておこう。

で、大抵の場合必要になるのがこれなので追加しておこう。

```
using UnityEditor;
```

また、今回は外部にファイルを出力するので

```
using System.IO;
```

も追加しておこう。

ではまず、メニューに自分が作った項目を表示してみよう。
クラスにこういう関数を追加してください。

```
[MenuItem(“メニューの名前”)]
private static void 適当な関数名(){
        適当な処理
}
```

ここで重要なのは角カッコ？の内部です。関数じたいは名前が何でも構いません。

上の角カッコは「属性」と言って、
なお、注意点として、メニューの名前は

- 日本語不可
- 必ず１つ以上の階層にしておくこと(階層は/で分割される)

という制約がありますので

```
[MenuItem("Test/TestExport")]
private static void TestExport() {
```

こんな感じで関数作っておきましょう。

では次に「何かを出力する」ですが、BinaryWriter を使いましょう。これは前述の System.IO を using しておかないと使えないので注意。

使い方は Stream を File.OpenWriter で開いておいて、それを BinaryWriter に食わせることで使います。

```
private static void TestExport() {
    var stream = File.OpenWrite("./test.dat");
    BinaryWriter binaryWriter = new BinaryWriter(stream);
    binaryWriter.Write(65535);
    binaryWriter.Close();
    stream.Close();
}
```

最後の Close は忘れないようにしましょう。

で、これどこに出力されてるんですかね？そう。Unity プロジェクト直下です。これでは外部で使うのにはちょっと不便ですね。

ということで、ダイアログを出します。セーブダイアログを出すには EditorUtility 内の SaveDialogPanel を使用します。

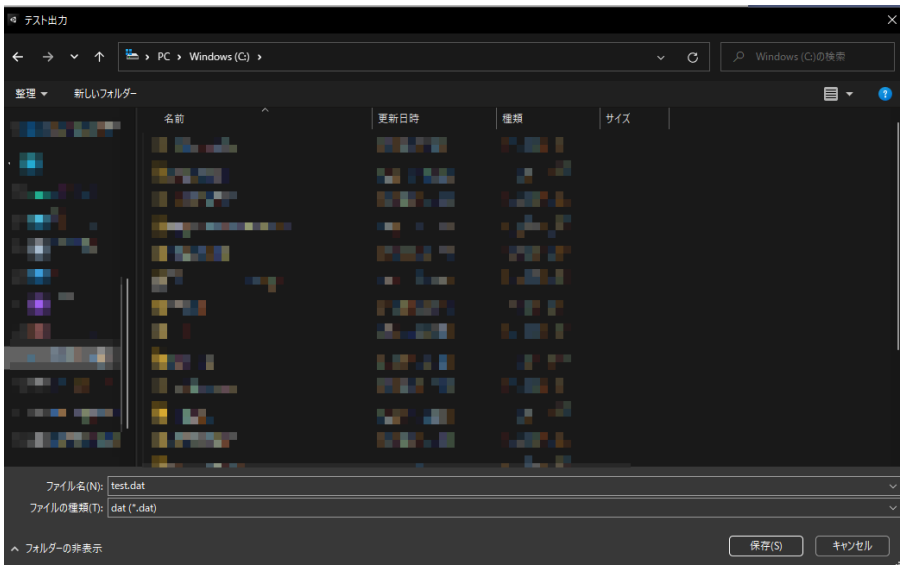
```
var filePath = EditorUtility.SaveFilePanel("タイトル", "デフォルトディレクトリ", "デフォルト名", "拡張子");
```

はい、こんな感じで指定します。

```
var filePath = EditorUtility.SaveFilePanel("テスト出力", ".", "test", "dat");
```

で、メニューから該当のボタンを押すと

こういうのが出て来ます。なお、SaveFilePanel は別に出力してくれるわけじゃなく、パスを取得してくれるだけなので、得られた文字列を先ほどのファイルオープンに突っ込みます




```
        var stream = File.OpenWrite(filePath);
```

はい、これで出力ができました。

なお、ダイアログをキャンセルされたとき用の処理も追加しときましょう。

```
var filePath = EditorUtility.SaveFilePanel("テスト出力", ".", "test", "dat");
if (filePath == "") return;
```

オブジェクトの座標を出力する

座標を出力すること自体は簡単です。gameObject の transform をとってこれればそこに情報が入っています。

でも、その gameObject はどうしましょうか。なんでもとってきては困りますので、選択部分のを出力することにしましょうか。

```
var gameObject = Selection.activeObject as GameObject;
```

なお、C#においては、(型名)のキャストよりも上記の as のほうが使用されます。

さて、更に面倒なんですが、この gameObject 内に階層構造に gameObject が配置されてる事がままあります(ていうか基本そうです)

なので、再帰構造を使って、階層の下までサーチできるようにします。なお、自分の子ノードはなぜか transform 内にあって、その情報は GetChild で取ってきます。

ちなみに子ノードも gameObject なのでこんな感じの関数になります。

```
void TraverseNode(GameObject gameObject) {
    var cnt = gameObject.transform.childCount;
    for(int i = 0; i < cnt; ++i) {
        TraverseNode(gameObject.transform.GetChild(i).gameObject);
    }
}
```

Traverse ってのは木構造を巡回していくって意味です。

はい、もちろんこれだけでは芸がないので、Debug.Log か何かで情報を出力しましょう。

で、ここで問題になるのが、情報を出力したくないゲームオブジェクトもあり得るという事です。

多分、カメラとか、パーティクルはいらないし、そもそもノードだけ一要は中身にメッシュがないようなノードまで出力する必要があるか？一ってことです。

そういうのはどう判断したらいいのでしょうか？

ここで登場するのがGetComponent<T>です。