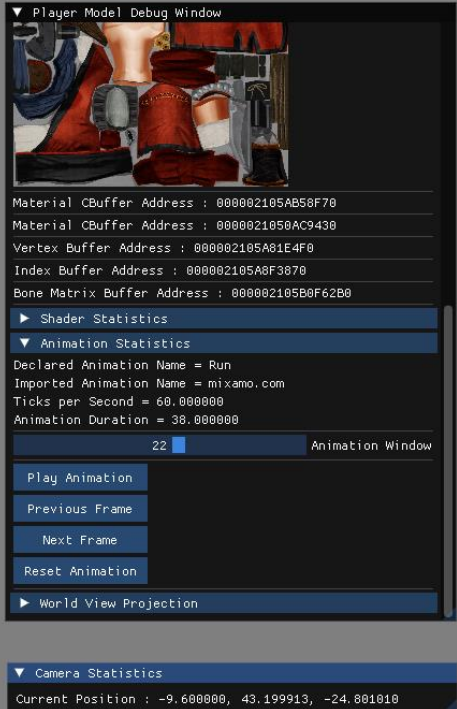


ポートフォリオ

HAL大阪 4年 エルフィン チュアンドラ
ELVIN CUANDRA



自己PR

エンジニアとしての強み

1つ目

最新技術を作品に取り入れる

新しいものにチャレンジすることで、自分自身をより成長できる環境を常に整っています。

2つ目

デバッグしやすい環境を構築

コードの可読性を重視しつつ、積極的にコードレビューとモジュール化を行い、より整合性を維持できます。

なりたいエンジニア像

チームプレイヤーとして

チームのために積極的に**貢献できる人**になりたい

チームの一員として、自分の役割を理解し、全員が納得できる決着点を探ることで、チームの仲間から信頼してもらえるようになりたいです。

プログラマとして

より成長できるために**技術**を使えるようになりたい

新しい技術を継続的に取り組み、自分自身をより成長できることで、ゲームのどの部分にどんなことをするか、どんな技術が最適なのかを判断して、様々な技術に取り込むことで、開発ワークフローをより効率化できるようになりたいと考えています。

AnimFramework

受賞 3年次 HAL EVENT WEEK 構成本賞

環境

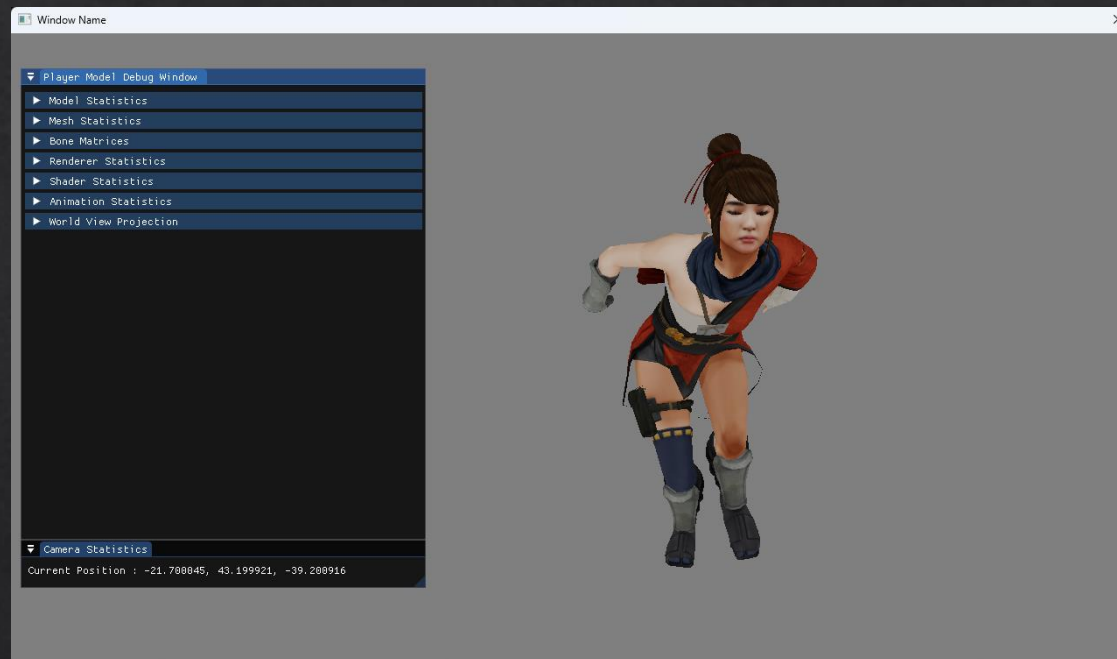
- Visual Studio 2022
- C++17
- DirectX11

外部ライブラリ&ツール

- SimpleMath
- WICTextureLoader
- Assimp
- GitHub
- ImGui

制作目的

アニメーションの原理を学ぶ



挑戦したこと

描画APIを利用したスケレタルアニメーションの再生

制作期間: 約1.5月間

参考書

- Gregory, J. (2019). *Game Engine Architecture*. CRC Press, Taylor & Francis Group. [Game Engine Architecture \(gameenginebook.com\)](http://gameenginebook.com)
- Mukai T., Kawachi K., Miyake Y. (2020). キャラクターアニメーションの数理とシステム. コロナ社.
[キャラクターアニメーションの数理とシステム - 3次元ゲームにおける身体運動生成と人工知能 - | コロナ社 \(coronasha.co.jp\)](http://coronasha.co.jp)
- Nystrom, R., & Nystrom, R. (2015). *Design patterns Für Die Spieleprogrammierung*. mitp. [Game Programming Patterns](http://GameProgrammingPatterns)

アピールポイント

頂点シェーダを用いたスケレタル
アニメーションの描画の高速化

CPU側ボーン行列計算

```
outMatrix.a1 = matrix[0].a1 * deformVertex->BoneWeight[0]
+ + + + matrix[1].a1 * deformVertex->BoneWeight[1]
+ + + + matrix[2].a1 * deformVertex->BoneWeight[2]
+ + + + matrix[3].a1 * deformVertex->BoneWeight[3];

outMatrix.a2 = matrix[0].a2 * deformVertex->BoneWeight[0]
+ + + + matrix[1].a2 * deformVertex->BoneWeight[1]
+ + + + matrix[2].a2 * deformVertex->BoneWeight[2]
+ + + + matrix[3].a2 * deformVertex->BoneWeight[3];

outMatrix.a3 = matrix[0].a3 * deformVertex->BoneWeight[0]
+ + + + matrix[1].a3 * deformVertex->BoneWeight[1]
+ + + + matrix[2].a3 * deformVertex->BoneWeight[2]
+ + + + matrix[3].a3 * deformVertex->BoneWeight[3];

outMatrix.a4 = matrix[0].a4 * deformVertex->BoneWeight[0]
+ + + + matrix[1].a4 * deformVertex->BoneWeight[1]
+ + + + matrix[2].a4 * deformVertex->BoneWeight[2]
+ + + + matrix[3].a4 * deformVertex->BoneWeight[3];
```



頂点シェーダボーン行列計算

```
float4x4 combMtx = (float4x4) 0;
for (int i = 0; i < 4; i++){
    combMtx += BoneMatrix[vin.BoneIndex[i]] * vin.BoneWeight[i];
}

float4 pos;
pos = mul(combMtx, float4(vin.Position, 1.f));
vin.Position = pos;

float4 normal;
normal = float4(vin.Normal.xyz, 0.f);

combMtx._41 = 0.f;
combMtx._42 = 0.f;
combMtx._43 = 0.f;
combMtx._44 = 1.f;
```

GPUの並列処理を利用し、ボーン
行列の計算を行うことでCPU側
におけるボーン情報の更新経過時間を
103ミリ秒→18ミリ秒に抑えられます。



CPUの負荷を軽減できる

CPU 経過時間 (PIXによるデバッグデータ)

Thread	Thread 26912
Address	0x00007FF7BAEB2BE0
Estimated Duration	103.875 ms
Estimated Start	2,966.614 ms
Estimated End	3,070.489 ms
Sample Count	309

変更前




Thread	Thread 10480
Address	0x00007FF709F82330
Source	C:\Users\Elvin\Desktop\Project
Estimated Duration	18.803 ms
Estimated Start	1,910.007 ms
Estimated End	1,928.810 ms
Sample Count	19

変更後

アピールポイント

依存性注入によるコードのモジュール化

依存性注入を行うことにより、
テスト容易性が高まり、単価テスト
で依存関係を明確にします。その
結果、品質が向上できます。

```
// Forward Declaration
class ModelLoader;

class ModelRenderer
{
public:
void Init(std::unique_ptr<ModelLoader>& model)
{
    if (model == nullptr) {
        throw std::invalid_argument("model should not be null");
    }
    m_pModel = std::move(model);
}

private:
std::unique_ptr<ModelLoader> m_pModel;
}
```

再利用性と可読性を検討しつつ、モジュールを書くことで
バグを減らすだけでなく、生産性もより向上できる



indie-us games賞

環境

- ・Visual Studio 2022
- ・UE5.0.3
- ・C++ & BP

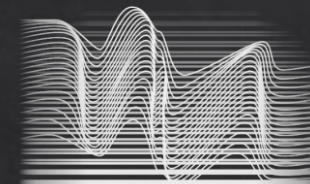
チーム構成

15名
プログラマー5、プランナ3、
デザイナー4、コンポザー3

担当箇所

リードプログラマ

- ・プレイヤー全般
- ・UX全般(UI & VFXを除く)
- ・Git管理



浪速とMoire

制作期間: 約4月間

作品概要

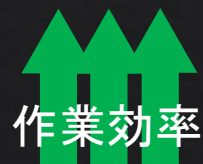
「ギリギリに迫るスリル」をコンセプトにした3Dアクションゲームです。
目標位置まで高速で移動でき、チェーンソーで敵を倒していく爽快感を感じる。

アピールポイント



プレイヤーのパラメータが
調整しやすいように作りました



プログラマー以外でも調整できるため、
メインプログラマーとして、調整より、
実装に時間をかけることができました



作業効率

▼ Recovery Rate		
Engine RPMRecovery Per...	33.333332	
▼ Drain Rate		
Engine RPMDrain Per Sec...	6.666667	
▼ Curve		
Movement Curve	 Movement_Curve	↶ ↷
Engine Curve	 Engine_Curve	↶ ↷
▼ Engine		
Min Engine RPM	0.0	
Max Engine RPM	100.0	
Current Engine RPM	0.0	
▼ Engine		
▼ Recovery Rate		
Engine RPMRecovery Per...	33.333332	
▼ Drain Rate		
Engine RPMDrain Per Sec...	6.666667	
▼ Heal		
Healing Pod	8	
Total Healing Amount	60.0	
Instant Heal Rate	0.3333	
Heal Predict Value	0.0	
Is Healing	<input type="checkbox"/>	
Can Continuous Heal	<input type="checkbox"/>	

レベルデザイナーを干渉せずに、楽々に
バランス調整ができる環境を用意しています。

* ついでに、調整禁止変数をC++側から編集できないように設定しました。

アピールポイント

インターフェスを常に利用する



インターフェスを使うことで依存関係を減らせる

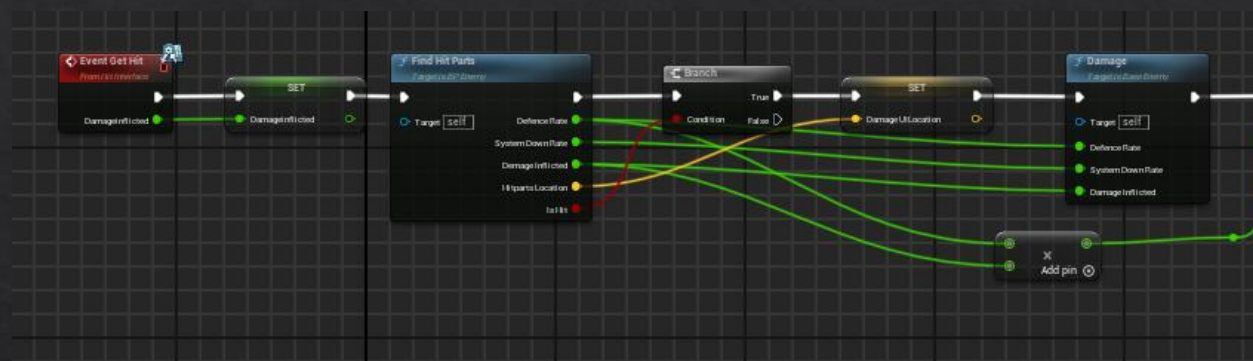
```
class DRIVE_API IHitInterface
{
    GENERATED_BODY()

    // Add interface functions to this class. This is the class that will be inherited.
public:
    // 攻撃がHitした時のダメージを渡すための関数
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "HitInterface")
    void GetHit(const float DamageInflicted);

    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "HitInterface")
    void GetHit_Player(const float DamageInflicted, EEnemyAttackHitReactionType HitRea);

    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category = "HitInterface")
    void GetHit_JustDodge(const bool WorldDynamic);
};
```

インターフェス



敵ベースクラス

知らない相手でも処理を呼び出すことができるため、
キャストを利用せずに、ゲームの負荷を軽減できる！

