

Port Folio

原 拓也

# Cyber Rush!!

制作人数：2人

制作期間：3か月（9月～12月）

- ・9月～9月半ば：企画
- ・9月半ば～10月：設計書制作やチームの工数管理
- ・10月～12月：開発作業

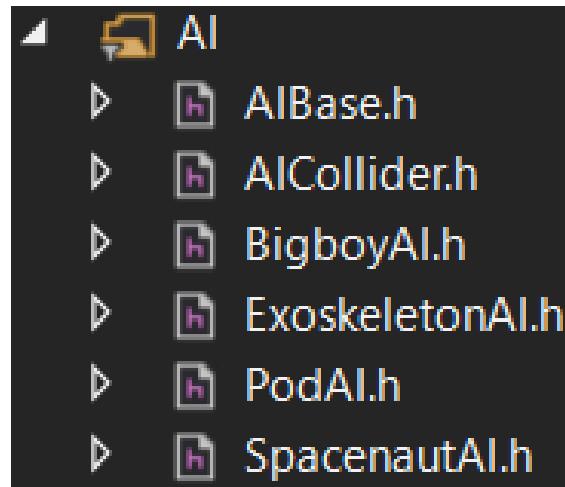
ジャンル：2Dベルトスクロールシューティングアクション

開発言語：C++      使用ライブラリ：DXライブラリ      開発環境：Visual Studio2019

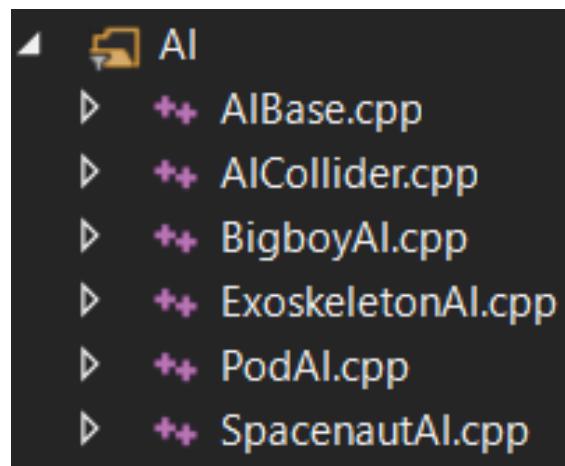
担当箇所：敵AI作成 各UI設置

# ～担当箇所説明～

## ① 敵AI作成



AIBase : AICollider class からの情報や、  
プレイヤーの位置に応じて各敵のAIを  
一括して更新するclass。



AICollider : 敵視点からの味方の位置情報を把握し  
味方が今自分の近くにいるかどうかを  
検知するAIの当たり判定class。

~AI class : 各敵毎のAI行動を記述したclass。

# ～担当箇所の概要～

各敵AIクラス

```
▲ AI
  ▷ AIBase.h
  ▷ AICollider.h
  ▷ BigboyAI.h
  ▷ ExoskeletonAI.h
  ▷ PodAI.h
  ▷ SpaconautAI.h
```

思考した結果で  
各敵の状態を決定



各敵クラス

```
▲ Enemy
  ▷ Bigboy.h
  ▷ Enemy.h
  ▷ Exoskeleton.h
  ▷ Pod.h
  ▷ Spaconaut.h
```

```
▲ Enemy
  ▷ Bigboy.cpp
  ▷ Enemy.cpp
  ▷ Exoskeleton.cpp
  ▷ Pod.cpp
  ▷ Spaconaut.cpp
```

AI管理クラスで各敵AIクラスで  
思考した行動を更新する。



- 1、状態毎にアニメーション遷移
- 2、AICollider(AIの当たり判定Box)  
を持たせ、AIColliderの状態を各敵AIクラス  
に渡す。

行動を思考する



# ～担当箇所説明～

## ②敵AI管理クラス作成

EnemyAIManager.h

```
10  class ControlledPlayer;
11  class AIBase;
12  class Enemy;
13
14  class EnemyAIManager
15  {
16  public:
17      EnemyAIManager(std::list<std::shared_ptr<Enemy>>& friends);
18      ~EnemyAIManager();
19
20      // 敵の行動の決定
21      void UpDate(const std::shared_ptr<Enemy> enemy);
22
23  private:
24      // 自分以外の味方
25      std::list<std::shared_ptr<Enemy>>& friendList_;
26
27      // 各敵のAI関数を敵タイプでの連想配列
28      std::map<ActorType, std::function<void(void)>> aiFunc_;
29
30  };
```

EnemyAIManager.cpp

```
5   EnemyAIManager::EnemyAIManager(std::list<std::shared_ptr<Enemy>>& friends):
6       friendList_(friends)
7   [
8   ]
9
10
11  EnemyAIManager::~EnemyAIManager()
12  [
13  ]
14
15
16  void EnemyAIManager::UpDate(const std::shared_ptr<Enemy> enemy)
17  [
18      // AIBaseを親としているPodAI等の各敵キャラに応じたAIの起動
19      // その先では関数ポッドを回しているので、条件に合致した行動を実行
20      // friendList_ : 味方の敵
21      enemy->GetAISystem()->AIAction(friendList_);
22  ]
```

# ～担当箇所説明～

- AIBase.h

```
1 #pragma once
2 #include <list>
3 #include <memory>
4
5 class Enemy;
6 class AICollider;
7
8 class AIBase
9 {
10 public:
11     AIBase(Enemy& enemy);
12     ~AIBase();
13     // 各敵のAI行動一括実行
14     virtual void AIAction(std::list<std::shared_ptr<Enemy>>& enemies);
15     // 各敵のAI思考ループの更新
16     virtual void Update(std::list<std::shared_ptr<Enemy>>& enemies) = 0;
17     // 初期化
18     virtual void Initialize(void) = 0;
19     // 各コライドの当たり判定(自分自身と自分以外の味方で比較)
20     bool IsHitFriends(std::list<std::shared_ptr<Enemy>>& enemies);
21 protected:
22     // 自分自身
23     Enemy& me_;
24     // 1回銃弾をインスансしたらtrueに
25     bool createBulletFlag_;
26 };
```

# • AIBase.cpp

## ～担当箇所説明～

### AIAction関数

コンストラクタ、デストラクタ

```
10  AIBase::AIBase(Enemy& enemy):
11  {
12      me_(enemy);
13
14      createBulletFlag_ = false;
15
16  ~AIBase()
17  {
18  }
```

```
20  void AIBase::AIAction(std::list<std::shared_ptr<Enemy>>& enemies)
21  {
22      // 自分が現在プレイヤーの左にいるかのフラグ
23      auto playerIsLeft = me_.SearchNearestPlayer()->GetIsTurnFlag();
24      // プレイヤーの位置取得
25      auto playerPos = me_.SearchNearestPlayer()->GetPos();
26      // プレイヤーが後ろにいるかのフラグ(存在すればtrueしなければfalse)
27      auto isBehind = false;
28      // プレイヤーが左に存在し
29      if (playerIsLeft)
30      [
31          // プレイヤーが自分の後ろに存在すれば
32          if (me_.GetPos().x > playerPos.x)
33          [
34              // フラグをtrueに
35              isBehind = true;
36          ]
37      ]
38      // プレイヤーが右に存在し
39      else
40      [
41          // プレイヤーが自分の後ろに存在すれば
42          if (me_.GetPos().x < playerPos.x)
43          [
44              // フラグをtrueに
45              isBehind = true;
46          ]
47      ]
48      // ダメージをくらったら
49      if (me_.GetOnDamaged())
50      [
51          // HPが0以下になると
52          if (me_.GetHP() <= 0)
53          [
54              // 爆発音を鳴らす
55              IpSound.Play("explosion",DX_PLAYTYPE_BACK);
56          ]
57      ]

```

続き

```
65  [
66      // 敵のフィールド移動制御-----
67      // x座標の制御
68      if (me_.GetPos().x <= 50)
69      [
70          me_.GetPos().x = 50;
71      ]
72      if (me_.GetPos().x >= floorX - 40)
73      [
74          me_.GetPos().x = floorX - 40;
75      ]
76      // z座標の制御
77      if (me_.GetZPos() <= -floorZ)
78      [
79          me_.GetZPos() = -floorZ;
80      ]
81      // 敵がボスの場合のz軸移動制御
82      if (me_.GetType() == ActorType::Bigboy)
83      [
84          if (me_.GetZPos() >= -120)
85          [
86              me_.GetZPos() = -120;
87          ]
88      else
89      [
90          if (me_.GetZPos() >= 0)
91          [
92              me_.GetZPos() = 0;
93          ]
94      ]
95      // -----
96      // 敵がプレイヤーの背後にいるかいないかフラグをセット
97      me_.SetIsBehindPlayer(isBehind);
98      // 敵の行動更新
99      Update(enemies);
00  ]

```

## • AIBase.cpp

# ～担当箇所説明～

### IsHitFriends関数

```
103 bool AIBase::IsHitFriends(std::list<std::shared_ptr<Enemy>>& enemies)
104 {
105     // プレイヤーの後ろに敵が存在するかのフラグ
106     auto isBehindPlayer = false;
107     // プレイヤーの後ろに敵が存在するかのチェック
108     for (auto e1 : enemies)
109     {
110         isBehindPlayer = e1->GetIsBehindPlayer();
111     }
112     // 敵の自分自身のAICollider位置(X, Y軸)
113     auto myColPos = me_.GetAICollider()->GetPos();
114     // 敵の自分自身のAICollider位置(Z軸：奥行)
115     auto myColPosZ = me_.GetAICollider()->GetZPos();
```

### 続き

```
116     // プレイヤーの後ろに何もないなれば
117     if (!isBehindPlayer)
118     {
119         for (auto e2 : enemies)
120         {
121             if (me_.GetId() != e2->GetId())
122             {
123                 if (!e2->GetIsHitAICollider() &&
124                     !me_.GetIsHitAICollider())
125                 [
126                     // 自分と当たった味方のAIColliderの位置
127                     auto otherColPos
128                     = Vector2I(e2->GetAICollider()->GetPos().x,
129                             e2->GetAICollider()->GetPos().y +
130                             (e2->GetAICollider()->GetZPos() / 2));
131
132                     // 味方と自分のAIColliderが当たっていたらAI起動
133                     if (abs(otherColPos.x - myColPos.x) <= (size / 2) &&
134                         abs(otherColPos.y - myColPos.y) <= (size / 2))
135                     [
136                         // 自分ではない味方のAIColliderの当たり判定をtrueに
137                         e2->GetIsHitAICollider() = true;
138                         // 自分のAIColliderの当たり判定をtrueに
139                         me_.GetIsHitAICollider() = true;
140                         return true;
141                     ]
142                 ]
143             }
144         }
145     }
146 }
147 }
148 }
149 }
150 }
151 return false;
152 }
```

# ～担当箇所説明～

## ・AICollider.h

```
1  #pragma once
2  #include "Geometry.h"
3  [#include "ActorType.h"
4
5  constexpr int size = 100;
6
7  // 敵のAI実現のためのコライダークラス
8  // 敵同士がぶつかればAI起動
9  class Enemy;
10
11 class AICollider
12 {
13 public:
14     AICollider();
15     ~AICollider();
16     // ポジションアップデート
17     void SetPos(const Vector2I& pos,const int& z);
18     // コライダーとオブジェクト描画
19     void Draw(void);
20
21     // ポジション取得関数
22     const Vector2I& GetPos(void)
23     {
24         return pos_;
25     }
26     // Z軸のポジション取得
27     const int& GetZPos(void)
28     {
29         return z_;
30     }
31
32 private:
33     // ポジション
34     Vector2I pos_;
35     // Z軸のポジション
36     int z_;
37 };
38 ];
```

# ～担当箇所説明～

- AICollider.cpp

コンストラクタ  
デストラクタ

SetPos関数

Draw関数

```
5  AICollider::AICollider()
6  [
7      pos_ = { 0,0 };
8      z_ = 0;
9  ]
10
11 AICollider::~AICollider()
12 [
13 ]
14
15 void AICollider::SetPos(const Vector2I& pos,const int& z)
16 [
17     pos_ = pos;
18     z_ = z;
19 ]
20
21 void AICollider::Draw(void)
22 [
23     // キャラクターを囲むように描画(Debug用)
24     DrawBox(pos_.x - (size / 2),pos_.y + (z / 2) - (size / 2),
25             pos_.x + (size / 2), pos_.y + (z / 2) + (size / 2),
26             0xff0000, false);
27 ]
```

# ～担当箇所説明～

# 各敵毎のAIクラス

## ・ PodAI.h

```
1  #pragma once
2  #include <memory>
3  [#include "AIBase.h"
4
5  class Enemy;
6
7  class PodAI :
8      public AIBase
9  {
10 public:
11     PodAI(Enemy& enemy);
12     ~PodAI();
13     // 更新関数
14     void Update(std::list<std::shared_ptr<Enemy>>& enemies);
15
16     // 初期行動
17     bool Walk(std::list<std::shared_ptr<Enemy>>& enemies);
18     // フレイヤーとのZ軸を合わせる
19     bool Z_Arrangement(std::list<std::shared_ptr<Enemy>>& enemies);
20     // フレイヤーとのZ軸があったら攻撃
21     bool Attack(std::list<std::shared_ptr<Enemy>>& enemies);
22     // タマジを食らったときの行動
23     bool OnDamaged(std::list<std::shared_ptr<Enemy>>& enemies);
24     // フレイヤーの背後に回り込む
25     bool Run(std::list<std::shared_ptr<Enemy>>& enemies);
26     // 死ぬ
27     bool Death(std::list<std::shared_ptr<Enemy>>& enemies);
28
29 private:
30     // 初期化
31     void Initialize(void);
32     // 状態遷移関数体。イタ
33     bool(PodAI::* updater_)(std::list<std::shared_ptr<Enemy>>& );
34
35 };
```

# ～担当箇所説明～

- PodAI.cpp
  - コンストラクタ
  - デストラクタ
  - Update関数

```
6  PodAI::PodAI(Enemy& enemy):
7  {
8  }
9  // 初期化
10 Initialize();
11
12 ~PodAI()
13 {
14 }
15
16 void PodAI::Update(std::list<std::shared_ptr<Enemy>>& enemies)
17 {
18
19  // タメジを食らっていたら
20  if (me_.GetOnDamaged())
21  {
22    // hitアニメーションに変更
23    me_.ChangeAnimation("hit");
24    // 状態をタメジ食らい状態に変更
25    updater_ = &PodAI::OnDamaged;
26  }
27  // 味方の攻撃を食らったら
28  if (me_.GetFriendlyFireFlag())
29  {
30    // 即座に死ぬ
31    me_.ChangeAnimation("death");
32    // 状態をdeathに変更
33    updater_ = &PodAI::Death;
34  }
35  // 状態の更新
36  (this->*updater_)(enemies);
37 }
```

# ～担当箇所説明～

- PodAI.cpp

## Walk関数

```
33 bool PodAI::Walk(std::list<std::shared_ptr<Enemy>>& enemies)
34 {
35     // 自分がプレイヤーよりも右にいたら
36     if (me_.GetPos().x > me_.GetNearestPlayer()->GetPos().x)
37     {
38         // 反転フラグをtrueに
39         // DrawRotaGraphの第7引数のX反転フラグと紐づけている
40         me_.GetIsTurnFlag() = true;
41         // 移動スピードを歩く際のスピードに設定
42         me_.GetSpeed().x = -2;
43     }
44     // 自分がプレイヤーよりも左にいたら
45     if (me_.GetPos().x < me_.GetNearestPlayer()->GetPos().x)
46     {
47         // 反転フラグをtrueに
48         // DrawRotaGraphの第7引数のX反転フラグと紐づけている
49         me_.GetIsTurnFlag() = false;
50         // 移動スピードを歩く際のスピードに設定
51         me_.GetSpeed().x = 2;
52     }
53     // ポジション移動
54     me_.GetPos().x += me_.GetSpeed().x;
55     // ここで距離の計算をやって、変数に入れている
56     auto distance = me_.GetNearestPlayer()->GetPos().x - me_.GetPos().x;
57
58     // Podとプレイヤーとの距離が一定距離になれば
59     // PodはプレイヤーのZ軸に合わせる行動をする
60     if (abs(distance) < 150)
61     {
62         me_.GetAIState() = AIState::ZArrange;
63         updater_ = &PodAI::Z_Arrangement;
64         return true;
65     }
66     // 歩いている際に味方のAIColliderに当たったら
67     if (IsHitFriends(enemies))
68     [
69         // 走るモーションに遷移
70         me_.GetAIState() = AIState::Run;
71         updater_ = &PodAI::Run;
72         return true;
73     ]
74 }
```

## Z\_Arrangement関数

```
82 bool PodAI::Z_Arrangement(std::list<std::shared_ptr<Enemy>>& enemies)
83 {
84     // 左に進む
85     if (me_.GetIsTurnFlag())
86     {
87         me_.GetSpeed().x = -1;
88     }
89     // 右に進む
90     else
91     {
92         me_.GetSpeed().x = 1;
93     }
94     // プレイヤーのZ軸ポジションの取得
95     auto playerZ = me_.GetNearestPlayer()->GetZPos();
96     // プレイヤーよりも手前に存在すれば
97     if (me_.GetZPos() >= playerZ)
98     {
99         // 手前に移動
100        me_.GetZSpeed() = -1;
101    }
102    else
103    {
104        // 奥に移動
105        me_.GetZSpeed() = 1;
106    }
107    // Z軸移動
108    me_.GetZPos() += me_.GetZSpeed();
109    // 味方に当たったら(傍に味方がいれば)
110    if (IsHitFriends(enemies))
111    {
112        // AI状態をRunにする
113        me_.GetAIState() = AIState::Run;
114        // 自身の状態をRunに変更
115        updater_ = &PodAI::Run;
116        return true;
117    }
118    // Z軸がプレイヤーのZ軸と一致したら
119    // 3というのはある程度プレイヤーを視認可能な範囲を広げてるので
120    if (abs(me_.GetZPos() - playerZ) <= 3)
121    {
122        // 攻撃準備アニメーションに変更
123        me_.ChangeAnimation("attack_prepare");
124        // AI状態をAttackに変更
125        me_.GetAIState() = AIState::Attack;
126        // 自身の状態をAttackに変更
127        updater_ = &PodAI::Attack;
128        return true;
129    }
130 }
```

# ～担当箇所説明～

## • PodAI.cpp

### Attack関数

```
116 bool PodAI::Attack(std::list<std::shared_ptr<Enemy>>& enemies)
117 {
118     // アニメーション終了時
119     if (me_.GetIsAnimEnd())
120     {
121         // 砲撃アニメーションに変更
122         me_.ChangeAnimation("attack_release");
123     }
124     // 砲撃アニメーションの場合
125     if (me_.GetCurrentAnimation() == "attack_release")
126     {
127         // 弾が発射する際の火花アニメーションを再生
128         me_.AddMuzzleFlashAnimationCount(0.1f);
129         // 火花アニメーションが最大値になったら
130         if (me_.GetmuzzleFlashAnimationCount() >= 5.0f)
131         {
132             // AIの当たり判定を再度可能にする
133             me_.GetIsHitAICollider() = false;
134             // 火花アニメーションカウンタを0で初期化
135             me_.GetmuzzleFlashAnimationCount() = 0.0f;
136             // walkアニメーションに変更
137             me_.ChangeAnimation("walk");
138             // AI状態をWalkに変更
139             me_.GetAIState() = AIState::Walk;
140             // 自身の状態をwalkに変更
141             updater_ = &PodAI::Walk;
142             return true;
143         }
144     }
145     else
146     {
147         // 左に移動
148         // プレイヤーが自分の左に存在したら
149         if (me_.GetPos().x > me_.GetNearestPlayer()->GetPos().x)
150         [
151             me_.GetIsTurnFlag() = true;
152             me_.GetSpeed().x = -2;
153         ]
154         // 右に移動
155         // プレイヤーが自分の右に存在したら
156         if (me_.GetPos().x < me_.GetNearestPlayer()->GetPos().x)
157         [
158             me_.GetIsTurnFlag() = false;
159             me_.GetSpeed().x = 2;
160         ]
161     }
162 }
```

### OnDamaged関数

```
164 bool PodAI::OnDamaged(std::list<std::shared_ptr<Enemy>>& enemies)
165 {
166     // アニメーション終了時
167     if (me_.GetIsAnimEnd())
168     [
169         // HPが0以下
170         if (me_.GetHP() <= 0)
171         [
172             // 死ぬアニメーションに変更
173             me_.ChangeAnimation("death");
174             // 自身の状態をdeathに変更
175             updater_ = &PodAI::Death;
176             return true;
177         ]
178     else
179     [
180         // まだHPが0以下でなければWalkアニメーションに変更
181         me_.ChangeAnimation("walk");
182         // 自身の状態をWalkに変更
183         updater_ = &PodAI::Walk;
184         return true;
185     ]
186     ]
187     // 1回ダメージを食らったらfalseにしないと以降ダメージが食らわないのでfalseにする
188     me_.GetOnDamaged() = false;
189
190 }
```

# ～担当箇所説明～

- PodAI.cpp

## Run関数

```
209 bool PodAI::Run(std::list<std::shared_ptr<Enemy>>& enemies)
210 {
211     // プレイヤーの向き
212     auto playerIsTurnLeft = me_.GetNearestPlayer()->GetIsTurnFlag();
213     // もしプレイヤーが右を向いていればプレイヤーの左側に行くまで直進
214     if (!playerIsTurnLeft)
215     {
216         me_.GetSpeed().x = -2;
217     }
218     // もしプレイヤーが左を向いていればプレイヤーの右側に行くまで直進
219     else
220     {
221         me_.GetSpeed().x = 2;
222     }
223
224     // 直進させる
225     me_.GetPos().x += me_.GetSpeed().x;
226
227     // プレイヤーのポジション
228     auto playerPos = me_.GetNearestPlayer()->GetPos();
229
230     if (!playerIsTurnLeft)
231     {
232         if (me_.GetPos().x <= playerPos.x - 100)
233         {
234             // AI状態をAttackに変更
235             me_.GetAIState() = AIState::Attack;
236             // 攻撃準備アニメーションに変更
237             me_.ChangeAnimation("attack_prepare");
238             // 自身の行動状態をAttackに変更
239             updater_ = &PodAI::Attack;
240             return true;
241         }
242     }
243     else
244     {
245         if (me_.GetPos().x >= playerPos.x + 100)
246         {
247             // AI状態をAttackに変更
248             me_.GetAIState() = AIState::Attack;
249             // 攻撃準備アニメーションに変更
250             me_.ChangeAnimation("attack_prepare");
251             // 自身の行動状態をAttackに変更
252             updater_ = &PodAI::Attack;
253             return true;
254         }
255     }
256 }
257
258 }
```

## Death関数、Initialize関数

```
260 bool PodAI::Death(std::list<std::shared_ptr<Enemy>>& enemies)
261 {
262     // アニメーションが終了したら
263     if (me_.GetIsAnimEnd())
264     {
265         // 自身を削除
266         me_.Delete();
267         return true;
268     }
269     return false;
270 }
271
272 void PodAI::Initialize(void)
273 {
274     // まず最初の状態Walkにする
275     updater_ = &PodAI::Walk;
276 }
```

# ～担当箇所説明～

# 各敵毎のAIクラス

## ・ ExoskeltonAI.h

```
1 #pragma once
2 #include "AIBase.h"
3 class ExoskeletonAI :
4     public AIBase
5 {
6 public:
7     ExoskeletonAI(Enemy& enemy);
8     ~ExoskeletonAI();
9     // 更新関数
10    void Update(std::list<std::shared_ptr<Enemy>>& enemies);
11
12    // プレイヤーの背後に回り込む
13    bool Run(std::list<std::shared_ptr<Enemy>>& enemies);
14    // 死ぬ
15    bool Death(std::list<std::shared_ptr<Enemy>>& enemies);
16 private:
17     // 初期化
18     void Initialize(void);
19     // 状態遷移関数ポインタ
20     bool (ExoskeletonAI::* updater_)(std::list<std::shared_ptr<Enemy>>& );
21     // 接地した後、プレイヤーの位置をサチするフレームカウント
22     int searchFrame;
23     // ラジアン角度
24     float rad;
25     // プレイヤーのポジション
26     Vector2I player_pos;
27 }
```

# ～担当箇所説明～

# 各敵毎のAIクラス

- ExoskeltonAI.h

```
1 #pragma once
2 #include "AIBase.h"
3 class ExoskeletonAI :
4     public AIBase
5 {
6 public:
7     ExoskeletonAI(Enemy& enemy);
8     ~ExoskeletonAI();
9     // 更新関数
10    void Update(std::list<std::shared_ptr<Enemy>>& enemies);
11
12    // プレイヤーの背後に回り込む
13    bool Run(std::list<std::shared_ptr<Enemy>>& enemies);
14    // 死ぬ
15    bool Death(std::list<std::shared_ptr<Enemy>>& enemies);
16 private:
17     // 初期化
18     void Initialize(void);
19     // 状態遷移関数ポインタ
20     bool (ExoskeletonAI::* updater_)(std::list<std::shared_ptr<Enemy>>& );
21     // 接地した後、プレイヤーの位置をサチするフレームカウント
22     int searchFrame;
23     // ラジアン角度
24     float rad;
25     // プレイヤーのポジション
26     Vector2I player_pos;
27 }
```

# ～担当箇所説明～

- ExoskeltonAI.cpp

コンストラクタ  
デストラクタ

```
9  void ExoskeletonAI::ExoskeletonAI(Enemy& enemy) :  
10 AIBase(enemy)  
11 {  
12     // 初期化  
13     Initialize();  
14 }  
15  
16 void ExoskeletonAI::~ExoskeletonAI()  
17 {  
18 }
```

## Update関数

```
20 void ExoskeletonAI::Update(std::list<std::shared_ptr<Enemy>>& enemies)  
21 {  
22     // プレイヤーを探し出すか外を加算  
23     // この値が一定になるとプレイヤーに向かって直進  
24     searchFrame++;  
25     // フィールドとの当たり判定  
26     if (BoxOutCollision()(Vector2I(me_.GetPos().x, me_.GetZPos()),  
27         Vector2I(0, -320), Vector2I(floorX, floorZ)))  
28     {  
29         // 地形に当たっていたらdeathアニメーションに変更  
30         me_.ChangeAnimation("death");  
31         // 自身をdeath状態に変更  
32         updater_ = &ExoskeletonAI::Death;  
33     }  
34     // 円の当たり判定  
35     if (CircleCollision()(me_.GetType(),  
36         me_.SearchNearestPlayer()->GetPos() - me_.GetPos(),  
37         me_.GetSize() + me_.GetNearestPlayer()->GetSize(),  
38         me_.GetZPos() - me_.GetNearestPlayer()->GetZPos()))  
39     {  
40         // deathアニメーションの変更  
41         me_.ChangeAnimation("death");  
42         // プレイヤーがダメージを食らったフラグをtrueにする  
43         // プレイヤーにダメージを与える  
44         me_.GetNearestPlayer()->GetOnDamaged() = true;  
45         // 自身の状態をdeathに変更  
46         updater_ = &ExoskeletonAI::Death;  
47     }
```

続き

## 各敵毎のAIクラス

```
47     // フィールドに存在する敵分回す  
48     for (auto enemy : enemies)  
49     {  
50         // 現在回している敵が自分ではないのでこのように記述  
51         if (me_.GetId() != enemy->GetId())  
52         {  
53             // 味方との当たり判定  
54             if (CircleCollision()(enemy->GetType(),  
55                 enemy->GetPos() - me_.GetPos(), enemy->GetSize() + me_.GetSize(),  
56                 enemy->GetZPos() - me_.GetZPos()))  
57             {  
58                 // 味方への攻撃が当たったフラグをtrueに  
59                 enemy->GetFriendlyFireFlag() = true;  
60                 // deathアニメーションに変更  
61                 me_.ChangeAnimation("death");  
62                 // 自身の状態をdeathに変更  
63                 updater_ = &ExoskeletonAI::Death;  
64             }  
65         }  
66     }  
67     // 画像の方向変更  
68     // 左向き  
69     if (rad >= 0)  
70     {  
71         me_.GetIsTurnFlag() = false;  
72     }  
73     // 右向き  
74     else  
75     {  
76         me_.GetIsTurnFlag() = true;  
77     }  
78     (this->*updater_)(enemies);  
79 }
```

# ～担当箇所説明～

- ExoskeltonAI.cpp

## Run関数

```
82 bool ExoskeletonAI::Run(std::list<std::shared_ptr<Enemy>>& enemies)
83 {
84     // 接地していたら
85     if (me_.OnFloor() == true)
86     {
87         // プレイヤーを検知する値が59になる = プレイヤーに直進する
88         if ((searchFrame / 2) % 60 == 59)
89         {
90             player_pos = Vector2I(me_.GetNearestPlayer()->GetPos().x,
91                                   me_.GetNearestPlayer()->GetZPos());
92             rad = atan2(player_pos.y - me_.GetZPos(),
93                         player_pos.x - me_.GetPos().x);
94         }
95         me_.GetPos().x += 5 * cos(rad);
96         me_.GetZPos() += 5 * sin(rad);
97     }
98     // プレイヤーを検知する値が59以下
99     // 直進はしないで常にプレイヤーのポジションを検索
100    // 直進するまでプレイヤーのポジションを更新し続ける
101    else
102    {
103        player_pos = Vector2I(me_.GetNearestPlayer()->GetPos().x,
104                               me_.GetNearestPlayer()->GetZPos());
105        rad = atan2(player_pos.y - me_.GetZPos(),
106                    player_pos.x - me_.GetPos().x);
107    }
108
109    return false;
110 }
```

## Death関数

```
112 bool ExoskeletonAI::Death(std::list<std::shared_ptr<Enemy>>& enemies)
113 {
114     // アニメーション終了時
115     if (me_.GetAnimEnd())
116     {
117         // 自身を削除
118         me_.Delete();
119         return true;
120     }
121 }
122 }
```

## Initialize関数

```
124 void ExoskeletonAI::Initialize(void)
125 {
126     // プレイヤーを検知するカウント
127     searchFrame = 0;
128     // 初期状態としてRunに設定
129     updater_ = &ExoskeletonAI::Run;
130 }
```

# 各敵毎のAIクラス

# ～担当箇所説明～

# 各敵毎のAIクラス

## ・ SpaconautAI.h

```
1 #pragma once
2 #include "AIBase.h"
3
4 class Enemy;
5
6 class SpaconautAI : public AIBase
7 {
8 public:
9     SpaconautAI(Enemy& enemy);
10    SpaconautAI();
11    // 更新
12    void Update(std::list<std::shared_ptr<Enemy>>& enemies);
13    // プレイヤーの位置情報をサーチする
14    bool Search(std::list<std::shared_ptr<Enemy>>& enemies);
15    // 初期行動
16    bool Walk(std::list<std::shared_ptr<Enemy>>& enemies);
17    // プレイヤーのZ軸に自分のZ軸を合わせる
18    bool ArrangementZ(std::list<std::shared_ptr<Enemy>>& enemies);
19    // Z軸がプレイヤーとあつたら攻撃をする(弾を出す)
20    bool Attack(std::list<std::shared_ptr<Enemy>>& enemies);
21    // ダメージ食らい処理
22    bool OnDamaged(std::list<std::shared_ptr<Enemy>>& enemies);
23    // 死ぬ処理
24    bool Death(std::list<std::shared_ptr<Enemy>>& enemies);
25
26 private:
27     // 初期化
28     void Initialize(void);
29     // 状態変数。今は
30     bool (SpaconautAI::* updater_)(std::list<std::shared_ptr<Enemy>>& );
31
32     int frame;
33     // ダメージを食らった時のフレームカウント(自身を赤く点滅させるために使用)
34     int damage_anim_frame;
35     // プレイヤーのZ軸アングル
36     int target_pos_z;
37     // 共に戦う味のポジション
38     Vector2I partnerPos_;
39     // 右に動くフラグ
40     bool moveRight_;
41     // 左に動くフラグ
42     bool moveLeft_;
43     // アニメーション再生可能フラグ
44     bool damage_anim_flag;
45 };
46 
```

# ～担当箇所説明～

- SpacernautAI.cpp

- コンストラクタ
- デストラクタ
- Update関数

## 各敵毎のAIクラス

```
6  SpacernautAI::SpacernautAI(Enemy& enemy) :
7  {
8      // 初期化
9      Initialize();
10 }
11 SpacernautAI::~SpacernautAI()
12 {
13 }
14
15 void SpacernautAI::Update(std::list<std::shared_ptr<Enemy>>& enemies)
16 {
17     // タメジを受けると
18     if (me_.GetOnDamaged())
19     {
20         // アニメーション再生可能フラグをtrueに
21         damage_anim_flag = true;
22         // 自身の状態をダメージを食らった状態にする
23         updater_ = &SpacernautAI::OnDamaged;
24     }
25     // アニメーション再生可能フラグがtrueならば
26     if (damage_anim_flag)
27     {
28         // ダメージアニメーションカウタ-を加算
29         // このことで自身を赤く点滅させる
30         damage_anim_frame++;
31         // 画像を赤く点滅させるためのアルファ値(画像の透過度)計算
32         me_.GetAlpha() = (100 / (((damage_anim_frame / 10) % 2) + 1));
33     }
34     // ダメージアニメーションカウタ-が一定値を超すと
35     if (damage_anim_frame >= 60)
36     {
37         // ダメージアニメーションフラグをfalseに
38         damage_anim_flag = false;
39         // ダメージアニメーションカウタ-を0に
40         damage_anim_frame = 0;
41         // アルファ値(画像の透過度)の初期化
42         me_.GetAlpha() = 100.0f;
43     }
44     // 状態の更新
45     (this->*updater_)(enemies);
46 }
47 }
```

# ～担当箇所説明～

- SpacernautAI.cpp

## Search関数

```
49 bool SpacernautAI::Search(std::list<std::shared_ptr<Enemy>>& enemies)
50 {
51     // プレイヤーのZ軸位置
52     target_pos_z = me_.GetNearestPlayer()->GetZPos();
53     // フィールドに存在する敵全体で回す
54     for (auto enemy : enemies)
55     {
56         // そのボスの中には存在していれば
57         if (enemy->GetType() == ActorType::Bigboy)
58         {
59             // そのボスのポジションを格納
60             partnerPos_ = enemy->GetPosition();
61         }
62     }
63     // 自身の状態をWalkにする
64     updater_ = &SpacernautAI::Walk;
65     return true;
66 }
```

## Walk関数

```
68     bool SpacernautAI::Walk(std::list<std::shared_ptr<Enemy>>& enemies)
69     {
70         // 共に戦っているボスと自身のポジションの距離
71         auto distance = partnerPos_.x - me_.GetPosition().x;
72
73         // -----ここではボスと自身の位置交換させる処理-----
74         // 上記の距離が一定距離になれば
75         if (abs(distance) <= 30)
76         {
77             // 距離が0になったら = ボスと自身の位置が一緒になったら
78             if (distance < 0)
79             {
80                 // 右移動のフラグをtrueに
81                 moveRight_ = true;
82             }
83             else
84             {
85                 // 左移動フラグをtrueに
86                 moveLeft_ = true;
87             }
88         }
89         else
90         {
91             // 自身の移動制御
92             // フィールド外に行かせない処理
93             if (me_.GetPosition().x < 800 - 68 && me_.GetPosition().x > 68)
94             {
95                 // フィールド内に居れば
96                 if (me_.GetPosition().x >= 400)
97                 {
98                     // 移動
99                     me_.GetSpeed().x = 1;
100                }
101                else
102                {
103                    // 移動
104                    me_.GetSpeed().x = -1;
105                }
106            }
107            // フィールド外の場合
108            else
109            {
110                // 移動させない為にフィールドを0に
111                me_.GetSpeed().x = 0;
112                // 自身の状態をZ軸を合わせる行動にする
113                updater_ = &SpacernautAI::Z_Arrangement;
114            }
115            // ポジション移動
116            me_.GetPosition().x += me_.GetSpeed().x;
117        }
118    }
```

## 続き

```
120     // プレイヤーが自身よりも右にいる場合
121     if (me_.GetPosition().x > me_.GetNearestPlayer()->GetPosition().x)
122     {
123         // 右向きにする
124         me_.GetIsTurnFlag() = true;
125     }
126     // プレイヤーが自身よりも左にいる場合
127     if (me_.GetPosition().x < me_.GetNearestPlayer()->GetPosition().x)
128     {
129         // 左向きにする
130         me_.GetIsTurnFlag() = false;
131     }
132     // 右移動フラグがtrueだと
133     if (moveRight_)
134     {
135         // 右移動をさせるために+2の値にする
136         me_.GetSpeed().x = 1;
137     }
138     // 左移動フラグがtrueだと
139     if (moveLeft_)
140     {
141         // 左移動をさせるためにマイナス値にする
142         me_.GetSpeed().x = -1;
143     }
144     // ポジション移動(X)
145     me_.GetPosition().x += me_.GetSpeed().x;
146     // 右移動フラグがtrueの場合
147     if (moveRight_)
148     {
149         // フィールドの右端以上になったら
150         if (me_.GetPosition().x >= floorX - 40)
151         {
152             // フィールドの右端より先に行かせない
153             me_.GetPosition().x = floorX - 40;
154             // 右移動フラグをfalseに
155             moveRight_ = false;
156             // 状態をプレイヤーのZ軸を合わせる行動の状態にする
157             updater_ = &SpacernautAI::Z_Arrangement;
158         }
159     }
160     // 左移動フラグがtrueの場合
161     if (moveLeft_)
162     {
163         // フィールドの左端になったら
164         if (me_.GetPosition().x <= 50)
165         {
166             // フィールドの左端より先に行かせない
167             me_.GetPosition().x = 50;
168             // 左移動フラグをfalseに
169             moveLeft_ = false;
170             // 状態をプレイヤーのZ軸を合わせる行動の状態にする
171             updater_ = &SpacernautAI::Z_Arrangement;
172         }
173     }
174 }
175
176 }
```

# ～担当箇所説明～

- SpacernautAI.cpp

## Z\_Arrangement関数

```
178 bool SpacernautAI::Z_Arrangement(std::list<std::shared_ptr<Enemy>>& enemies)
179 {
180     // プレイヤーのZ軸アニメーション
181     target_pos_z = me_.GetNearestPlayer()->GetZPos();
182     // プレイヤーが自身より手前いる場合
183     if (me_.GetZPos() >= target_pos_z)
184     {
185         // 手前に移動させる
186         me_.GetZSpeed() = -1;
187     }
188     else
189     {
190         // 奥に移動させる
191         me_.GetZSpeed() = 1;
192     }
193     // Z軸移動
194     me_.GetZPos() += me_.GetZSpeed();
195     // 自身のZ軸とプレイヤーのZ軸の距離が3以下で接地していたら
196     // なぜ3以下?? -> ある程度余裕を持たせておくことでプレイヤーに自身の攻撃が当たるようにする
197     if (abs(me_.GetZPos() - target_pos_z) <= 3 && me_.OnFloor() == true)
198     {
199         // 自身の状態をAttack状態にする
200         updater_ = &SpacernautAI::Attack;
201     }
202     return true;
203 }
```

# 各敵毎のAIクラス

## Attack関数

```
205 bool SpacernautAI::Attack(std::list<std::shared_ptr<Enemy>>& enemies)
206 {
207     // 攻撃しているカウントを加算する
208     frame++;
209     // カウントが60フレーム以内だと攻撃をし続ける
210     if (frame <= 60)
211     {
212         // Attackアニメーションに変更
213         me_.ChangeAnimation("attack");
214     }
215     else
216     {
217         // 攻撃しているカウントを0に
218         frame = 0;
219         // Walkアニメーションに変更
220         me_.ChangeAnimation("walk");
221         // 自身の状態をプレイヤーを探している状態にする
222         updater_ = &SpacernautAI::Search;
223     }
224 }
225 }
```

## OnDamaged関数

```
227 bool SpacernautAI::OnDamaged(std::list<std::shared_ptr<Enemy>>& enemies)
228 {
229     // HPが0以下になると
230     if (me_.GetHp() <= 0)
231     {
232         // deathアニメーションに変更
233         me_.ChangeAnimation("death");
234         // 自身の状態をdeathにする
235         updater_ = &SpacernautAI::Death;
236     }
237     else
238     {
239         // Walkアニメーションに変更
240         me_.ChangeAnimation("walk");
241         // 自身の状態をプレイヤーを探している状態にする
242         updater_ = &SpacernautAI::Search;
243     }
244     // 1回ダメージを食らったらfalseにしないといつもダメージが食らわないでfalseにする
245     me_.GetOnDamaged() = false;
246     return false;
247 }
```

# ～担当箇所説明～

- SpacernautAI.cpp

## 各敵毎のAIクラス

### Death関数

```
249     bool SpacernautAI::Death(std::list<std::shared_ptr<Enemy>>& enemies)
250     {
251         // アニメーション終了時
252         if (me_.GetIsAnimEnd())
253         {
254             // 自身を削除する
255             me_.Delete();
256             return true;
257         }
258     }
259 }
```

### Initialize関数

```
261     void SpacernautAI::Initialize(void)
262     {
263         // 攻撃カウントの初期化
264         frame = 0;
265         // タンデムアニメーションカウントの初期化
266         damage_anim_frame = 0;
267         // 共に戦う味方のボディションを初期化
268         partnerPos_ = { 0,0 };
269         // 左移動フラグ の初期化
270         moveLeft_ = false;
271         // 右移動フラグ の初期化
272         moveRight_ = false;
273         // タンデムアニメーションフラグ の初期化
274         damage_anim_flag = false;
275         // 自身の初期状態をSearchに設定
276         updater_ = &SpacernautAI::Search;
277     }
```

# Alone War

## ～ボックス戦争～

制作人数：3人

制作期間：3か月（11月～3月）

- ・12月～12月半ば：企画
- ・12月半ば～1月上旬：設計書制作や工数管理
- ・1月上旬～1月下旬：開発ツール開発作業
- ・2月～3月：ゲーム本編開発作業

ジャンル：疑似3Dアクション

開発言語：C++      使用ライブラリ：DXライブラリ      開発環境：Visual Studio2019

担当箇所：敵AI作成、当たり判定ボックス作成ツール作成、データ読み込みツール作成

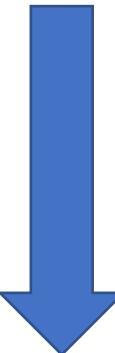
# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

```
▷ Application.cpp  
▷ BaseScene.cpp  
▷ ColliderBox.cpp  
▷ ColliderBoxEdit.cpp  
▷ ColliderBoxManager.cpp  
▷ FileManager.cpp  
▷ Geometry.cpp  
▷ HitCheck.cpp  
▷ ImageManager.cpp  
▷ ImageResourceDataEdit.cpp  
▷ main.cpp  
▷ Mouse.cpp  
▷ Select.cpp  
▷ SpriteDataCache.cpp  
△ ハッダー ファイル  
▷ Application.h  
▷ BaseScene.h  
▷ ColliderBox.h  
▷ ColliderBoxEdit.h  
▷ ColliderBoxManager.h  
▷ FileManager.h  
▷ Geometry.h  
▷ HitCheck.h  
▷ ImageManager.h  
▷ ImageResourceDataEdit.h  
▷ Mouse.h  
▷ Select.h  
▷ SpriteDataCache.h
```

画像アニメーションデータ作成シーン  
でアニメーションデータの作成

ファイル操作クラスにて  
ファイル読み込みや書き込みをする



データ作成

# ～担当箇所説明～

## ImageResourceDataEdit.h

```
9 // [決定] なのか [戻る] なのか
10 enum class OperationProcess
11 {
12     // 決定
13     DECISION,
14     // 戻る
15     RETURN,
16     MAX
17 };
```

入力名でImagesフォルダ 内の画像ファイル名  
を検索する

アニメーションデータに保存する  
各種データの入力処理

# ①アニメーションデータ作成ツール作成

続き

```
21 // ImageDataを編集するシン
22 class ImageResourceDataEdit :
23     public BaseScene
24 {
25     public:
26     ImageResourceDataEdit();
27     ~ImageResourceDataEdit();
28
29     // 更新
30     UniqueScene Update(UniqueScene own, const std::shared_ptr<Mouse>& mouse);
31     // 初期化
32     void Initialize(void);
33     // フェース毎の処理の初期化
34     void InitializeUpdater(void);
35     // フェース毎の描画処理の初期化
36     void InitializeDrawExcecuter(void);
37     // 描画
38     void Draw(void);
39
40     // Phase::INPUTDIRECTORYで使用-----
41     // ディレクトリ名の入力
42     void InputDirectoryName(void);
43     // オブジェクト名の入力
44     void InputObjectName(void);
45     // -----
46
47     // Phase::DATAEDITで使用-----
48     // 最大フレームの入力
49     void InputMaxFrame(void);
50     // ループフラグの入力(0か1でtrueかfalseの判断)
51     void InputLoopFlag(void);
52     // 1コマにかかる時間の入力
53     void InputDuration(void);
```

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

ImageResourceDataEdit.hの続き

```
54 // 【決定】と【戻る】のボタンのポジション
55 ProcessVec processPos_;
56 // 【決定】と【戻る】のボタンサイズ
57 ProcessVec boxSize_;
58 // 今現在の選択事項(【決定】か【戻る】のどちらか)
59 OperationProcess nowOperation_;
60 // クリック時の選択事項(【決定】か【戻る】のどちらか)
61 OperationProcess executeProcess_;
62 // 入力欄のポジション
63 Vector2I inputBoxPos_;
64 // 入力欄のサイズ
65 Vector2I inputBoxSize_;
66 // 【入力】という文字のポジション
67 Vector2I inputStringPos_;
68 // 【入力】という文字のサイズ
69 Vector2I inputStringSize_;
70 // アニメーション名が表示されているポジション
71 std::vector<Vector2I> animNamePos_;
72 // 表示されているアニメーション名全体のサイズ(ファイル名が12文字だったら12*40になる)
73 std::vector<Vector2I> animNameSize_;
74 // 現在のフェーズ
75 Phase nowPhase_;
76 // フェーズ毎の処理を記述
77 std::map<Phase, std::function<void(const std::shared_ptr<Mouse>& mouse)>> updater_;
78 // フェーズ毎の描画処理
79 std::map<Phase, std::function<void()>> drawExecutor_;
80 // 入力ディレクトリ名
81 char inputDirectoryName_[20];
82 // 入力オブジェクト名
83 char inputObjectName_[20];
84 // 入力した最大フレーム数
85 char inputMaxFrame_[20];
```

```
87 // 入力したloopFlag(アニメーションを一回再生するのか(true)しないのか(false))
88 char inputLoopFlag_[20];
89 // 入力したduration(1コマにかかる時間)
90 char inputDuration_[20];
91 // 入力した際に生成されるハンドル
92 int inputHandle_[4];
93 // 入力段階毎の関数数。シタ
94 void (ImageResourceDataEdit::*inputFunc_)(void);
95 // 【入力】というボタンの上にマウスカーソルが来たらtrueになるフラグ
96 bool onInputDirectoryButton_;
97 // ディレクトリ名の入力可能のフラグ
98 bool inputDirectoryFlag_;
99 // マウスボタンが乗ったファイルのナバー
100 int animNoOnMouse_;
101 // 選択したファイル名ナバー(可変長配列で作られているのでこの番号で選択ファイルにアクセス)
102 int selectAnimNo_;
103 // データをセーブしたことを表すフラグ
104 bool saveCmpFlag_;
105 };
```

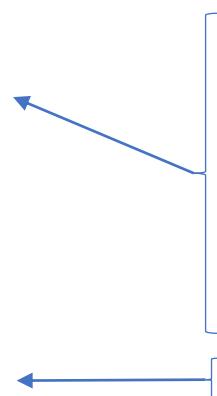
# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

コンストラクタ、デストラクタ

```
17  ImageResourceDataEdit::ImageResourceDataEdit()
18  {
19      exitAppFlag_ = false;
20      Initialize();
21  }
22
23  ImageResourceDataEdit::~ImageResourceDataEdit()
24  {
25  }
```

【戻る】ボタンや【決定】ボタン  
を押したときの処理



```
27  UniqueScene ImageResourceDataEdit::Update(UniqueScene own, const std::shared_ptr<Mouse>& mouse)
28  {
29      // マウスのクリック情報をMouseクラスから取得
30      auto clickData = mouse->GetClick();
31      // マウスホイールの位置
32      auto mPoint = mouse->GetPos();
33
34      nowOperation_ = OperationProcess::MAX;
35
36      if (HitCheck_FromCenter()(processPos_[static_cast<int>(OperationProcess::DECISION)], mPoint,
37          boxSize_[static_cast<int>(OperationProcess::DECISION)]))
38      {
39          nowOperation_ = OperationProcess::DECISION;
40      }
41
42      if (HitCheck_FromCenter()(processPos_[static_cast<int>(OperationProcess::RETURN)], mPoint,
43          boxSize_[static_cast<int>(OperationProcess::RETURN)]))
44      {
45          nowOperation_ = OperationProcess::RETURN;
46      }
47
48      updaters_[nowPhase_](mouse);
49
50
51      ClsDrawScreen();
52      Draw();
53      ScreenFlip();
54  }
55  return move(own);
```

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

### Initialize関数

```
57 void ImageResourceDataEdit::Initialize(void)
58 {
59     // FileManagerの生成(singletonイスタンス)
60     FileManager::Create();
61     // SpriteDataCacheの生成(singletonイスタンス)
62     SpriteDataCache::Create();
63     lpImageManager.Load("Images/button_decide.png");
64     lpImageManager.Load("Images/button_return.png");
65     lpImageManager.Load("Images/inputButton.png");
66     auto port = lpApplication.GetViewport();
67     processPos_ = { Vector2I{port.screen.x / 2 + 200, port.screen.y - 200},
68     |  Vector2I{port.screen.x / 2 + 200, port.screen.y - 100} };
69     inputBoxPos_ = Vector2I{ port.screen.x / 2 - 150, 60 };
70     inputBoxSize_ = Vector2I{ 500, 25 };
71     inputStringSize_ = Vector2I{ 50, 25 };
72     inputStringPos_ =
73         Vector2I{ inputBoxPos_.x + (inputBoxSize_.x / 2) + (inputStringSize_.x / 2), inputBoxPos_.y };
74
75     boxSize_ = { Vector2I{105, 50}, Vector2I{95, 50} };
76
77     nowOperation_ = OperationProcess::MAX;
78     nowPhase_ = Phase::FIRST;
79     excecuteProcess_ = nowOperation_;
80
81     inputDirectoryFlag_ = false;
82     onInputDirectoryButton_ = false;
83     saveCmpFlag_ = false;
84     selectAnimNo_ = -1;
85     animNoOnMouse_ = selectAnimNo_;
86
87     inputFunc_ = &ImageResourceDataEdit::InputDirectoryName;
88
89     InitializeUpdater();
90     InitializeDrawExecuter();
91 }
92 }
```

ボタン画像のロード

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

InitializeUpdater関数 → このシーン内の更新関数の初期化

```
94     void ImageResourceDataEdit::InitializeUpdater(void)
95     {
96         updater_.try_emplace(Phase::FIRST, [&](const std::shared_ptr<Mouse>& mouse) {
97
98             // マウスのクリック情報をMouseクラスから取得
99             auto clickData = mouse->GetClick();
100            // マウスホイントの位置
101            auto mPoint = mouse->GetPos();
102            if (HitCheck_FromCenter()(inputStringPos_, mPoint, inputBoxSize_))
103            {
104                onInputDirectoryButton_ = true;
105            }
106            else
107            {
108                onInputDirectoryButton_ = false;
109            }
110
111            for (auto click : clickData)
112            {
113                // クリックトグル有り
114                if (click.second[static_cast<int>(TRG::NOW)] && !click.second[static_cast<int>(TRG::OLD)])
115                {
116                    if (onInputDirectoryButton_)
117                    {
118                        inputDirectoryFlag_ = true;
119                    }
120                }
121            }
122            if (inputDirectoryFlag_)
123            {
124                (this->*inputFunc_)();
125            }
126        });
127    }
```

処理一段階目の格納

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

```
128     updater_.try_emplace(Phase::SECOND, [&](const std::shared_ptr<Mouse>& mouse) {
129         // マウスのタップ情報をMouseクラスから取得
130         auto clickData = mouse->GetClick();
131         // マウスボタンの位置
132         auto mPoint = mouse->GetPos();
133         // 常にマウスボタンが乗っているアニメ名番号は-1にしておく
134         // こうすることによってマウスボタンがアニメ番号上に来た時にこの番号が更新される仕組み
135         animNoOnMouse_ = -1;
136         // 表示されているアニメ名全てで回す
137         for(int num = 0; num < animNamePos_.size(); num++)
138         {
139             // 表示されているアニメ名とマウスボタンの当たり判定(左端起点として当たり判定をしている)
140             if (HitCheck_FromLeftUp()(animNamePos_[num], mPoint, animNameSize_[num]))
141             {
142                 // マウスボタンが来た番号で更新
143                 animNoOnMouse_ = num;
144             }
145         }
146         for (auto click : clickData)
147         {
148             // クリック外れ有り
149             if (click.second[static_cast<int>(TRG::NOW]) && !click.second[static_cast<int>(TRG::OLD)])
150             {
151                 if (animNoOnMouse_ != -1)
152                 {
153                     // クリックされていたら番号を更新
154                     selectAnimNo_ = animNoOnMouse_;
155                 }
156                 // 【決定】ボタン押下時
157                 if (nowOperation_ == OperationProcess::DECISION)
158                 {
159                     nowPhase_ = Phase::THIRD;
160                 }
161                 // 【戻る】ボタン押下時
162                 if (nowOperation_ == OperationProcess::RETURN)
163                 {
164                     selectAnimNo_ = -1;
165                 }
166             }
167         }
168     });
});
```

処理二段階目の格納

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

```
170     updater_.try_emplace(Phase::THIRD, [&](const std::shared_ptr<Mouse>& mouse) {
171         // FileManagerクラスよりDataFileから読み取ったアニメーション名を取得
172         // ファイル選んだアニメーション名
173         auto animName = IpFileManager::GetAnimationNameList() [selectAnimNo_];
174         // 画像ファイル名の作成
175         auto imageFileName = "Images/Actor/" + (std::string)inputObjectName_ + "/" + animName;
176         // 画像ファイル名より画像ファイル名の検索
177         IpFileManager::SearchFileFromDirectory(imageFileName.c_str());
178     });
179 
```

↑  
処理三段階目の格納

→  
処理四段階目の格納

```
182     updater_.try_emplace(Phase::FOURTH, [&](const std::shared_ptr<Mouse>& mouse) {
183         // マウスのクリック情報をMouseクラスから取得
184         auto clickData = mouse->GetClick();
185         // マウスクリックのオブジェクト
186         auto mPoint = mouse->GetPos();
187         for (auto click : clickData)
188         {
189             // クリックが~有り
190             if (click.second[static_cast<int>(TRG::NOW]) && !click.second[static_cast<int>(TRG::OLD)])
191             {
192                 if (nowOperation_ == OperationProcess::DECISION)
193                 {
194                     // セットする処理
195                     // FileManagerに渡すすべて文字列
196                     std::vector<std::string> strVec;
197                     // 最大フレーム値の格納
198                     strVec.push_back((std::string)inputMaxFrame_);
199                     // リープルフラグの格納
200                     strVec.push_back((std::string)inputLoopFlag_);
201                     // リピートかかる時間の格納
202                     strVec.push_back((std::string)inputDuration_);
203
204                     // 画像ファイル名の格納
205                     for (auto fileName : IpFileManager::GetFileNameList())
206                     {
207                         strVec.push_back(fileName);
208                     }
209                     // 編集しているアニメーション名
210                     auto animName = IpFileManager::GetAnimationNameList() [selectAnimNo_];
211                     // 作成データをデータファイルに書き込む
212                     IpFileManager::WriteImageDataFile((std::string)inputObjectName_ + "/Rect/" + animName, strVec);
213                     saveCmpFlag_ = true;
214
215                 if (nowOperation_ == OperationProcess::RETURN)
216                 {
217                     // やり直しをする処理
218                     nowPhase_ = Phase::FIRST;
219                     // ファイル名群の中を空にする
220                     IpFileManager::ClearFileNameList();
221                     // 入力データを初期状態に変更
222                     inputFunc_ = &ImageResourceDataEdit::InputDirectoryName;
223                     // 各種入力文字列の初期化
224                     for (int i = 0; i < 20; i++)
225                     {
226                         inputDirectoryName_[i] = ' ';
227                         inputObjectName_[i] = ' ';
228                         inputMaxFrame_[i] = ' ';
229                         inputMaxFrame_[i] = ' ';
230                         inputLoopFlag_[i] = ' ';
231                         inputDuration_[i] = ' ';
232                     }
233                 }
234             }
235         });
236 
```

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

InitializeDrawExecuter関数 → このシーンの各段階ごとの描画関数

```
239 void ImageResourceDataEdit::InitializeDrawExecuter(void)
240 {
241     drawExecuter_.try_emplace(Phase::FIRST, [&] () {
242         DrawFormatString(inputBoxPos_.x - (inputBoxSize_.x / 2), inputBoxPos_.y - (inputBoxSize_.y / 2) - 30,
243         0xffffffff, "ファイル名を入力して下さい(※フルパス)");
244
245         DrawBox(inputBoxPos_.x - (inputBoxSize_.x / 2), inputBoxPos_.y - (inputBoxSize_.y / 2),
246         inputBoxPos_.x + (inputBoxSize_.x / 2), inputBoxPos_.y + (inputBoxSize_.y / 2),
247         0xffffffff, false);
248
249         DrawRotaGraph(inputStringPos_.x, inputStringPos_.y, 1.0f, 0.0f,
250         lpImageManager.GetHandle("Images/inputButton.png")[0], true);
251
252         // マウスカーソルが所定の場所に来たら透過された箱を描画
253         if (onInputDirectoryButton_)
254         {
255             // 透過させるためにアルファblendする(そこそこ薄く箱を描画させたいため150という値)
256             SetDrawBlendMode(DX_BLENDMODE_ALPHA, 150);
257             DrawBox(inputStringPos_.x - (inputStringSize_.x/2), inputStringPos_.y - (inputStringSize_.y/2),
258             inputStringPos_.x + (inputStringSize_.x / 2), inputStringPos_.y + (inputStringSize_.y / 2),
259             0x00ff00, true);
260             // アルファblendを箱だけに適応させたいので、箱の描画の最後に0でリセットをかける
261             SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
262         }
263     });
264 }
```

処理一段階目での  
描画処理

```
266     drawExecuter_.try_emplace(Phase::SECOND, [&] () {
267         auto animNames = lpFileManager.GetAnimationNameList();
268         for (int num = 0; num < animNames.size(); num++)
269         {
270             // 無限にpush_backしないようにガードを張る
271             if (animNamePos_.size() < animNames.size())
272             {
273                 // ファイル名サイズを詰めていく
274                 animNameSize_.push_back(Vector2I{ (int)animNames[num].size() * 9, 20 });
275                 // ファイル名の位置情報を詰めていく
276                 animNamePos_.push_back(Vector2I{ 50, 50 + (num * 25) });
277             }
278             // ファイル名表示
279             DrawFormatString(animNamePos_[num].x, animNamePos_[num].y,
280             0xffffffff, "%s", animNames[num].c_str());
281         }
282         if (animNoOnMouse_ != -1)
283         {
284             // 透過させるためにアルファblendする(そこそこ薄く箱を描画させたいため150という値)
285             SetDrawBlendMode(DX_BLENDMODE_ALPHA, 150);
286             DrawBox(animNamePos_[animNoOnMouse_].x, animNamePos_[animNoOnMouse_].y,
287             animNamePos_[animNoOnMouse_].x + animNameSize_[animNoOnMouse_].x,
288             animNamePos_[animNoOnMouse_].y + animNameSize_[animNoOnMouse_].y,
289             0x00ff00, true);
290             // アルファblendを箱だけに適応させたいので、箱の描画の最後に0でリセットをかける
291             SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
292         }
293         DrawFormatString(0, 0, 0xffffffff, "%d", selectAnimNo_);
294     });
295 }
```

処理二段階目での  
描画処理

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

### 処理三段階目の描画処理

```
297     drawExeuctor_.try_emplace(Phase::THIRD, [&] () {
298         DrawFormatString(100, 100, 0xffffffff, "アニメーション最大フレーム数を入力");
299         DrawFormatString(100, 130, 0xffffffff, "%s", inputMaxFrame_);
300
301         DrawFormatString(100, 200, 0xffffffff, "ループフラグ【true or false】)を入力");
302         DrawFormatString(100, 230, 0xffffffff, "%s", inputLoopFlag_);
303
304         DrawFormatString(100, 300, 0xffffffff, "1コマにかかる時間(float型)を入力");
305         DrawFormatString(100, 330, 0xffffffff, "%s", inputDuration_);
306
307         (this->*inputFunc_)();
308     });
309 }
```

```
309     drawExeuctor_.try_emplace(Phase::FOURTH, [&] () {
310         DrawFormatString(50, 50, 0xffffffff, "このデータでよろしいですか？よろしければ【決定】やり直したい場合【戻る】をクリック");
311
312         DrawFormatString(50, 100, 0xffffffff, "最大フレーム数");
313         auto maxFrame = lpSpriteDataCache.GetAnimationData().maxFrame;
314         DrawFormatString(50, 130, 0x00ff00, "%d", maxFrame);
315
316         DrawFormatString(50, 160, 0xffffffff, "loopフラグ");
317         bool loop = lpSpriteDataCache.GetAnimationData().loop;
318         std::string str = "false";
319         if (loop)
320         {
321             str = "true";
322         }
323         DrawFormatString(50, 190, 0x00ff00, "%s", str.c_str());
324
325         DrawFormatString(50, 220, 0xffffffff, "1コマにかかる時間");
326         auto duration = lpSpriteDataCache.GetAnimationData().duration;
327         DrawFormatString(50, 250, 0x00ff00, "%lf", duration);
328
329         DrawFormatString(50, 280, 0xffffffff, "画像パス群");
330         // 画像パス名群
331         auto fileNames = lpFileManager.GetFileNameList();
332         for (int num = 0; num < fileNames.size(); num++)
333         {
334             DrawFormatString(50, 300 + (num * 20), 0x00ff00, "%s", fileNames[num].c_str());
335         }
336         if (saveCmpFlag_)
337         {
338             DrawFormatString(processPos_[static_cast<int>(OperationProcess::DECISION)].x,
339                             processPos_[static_cast<int>(OperationProcess::DECISION)].y - 100, 0xff0000, "セーブしました！");
340         }
341     });
342 }
```

### 処理四段階目の処理

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

Draw関数 → このシーンでの大本の描画

```
344 void ImageResourceDataEdit::Draw(void)
345 {
346     drawExeuctor_[nowPhase_]();
347     DrawRotaGraph(processPos_[static_cast<int>(OperationProcess::DECISION)].x,
348                   processPos_[static_cast<int>(OperationProcess::DECISION)].y, 1.0f, 0.0f,
349                   lpImageManager.GetHandle("Images/button_decide.png")[0], true);
350
351     DrawRotaGraph(processPos_[static_cast<int>(OperationProcess::RETURN)].x,
352                   processPos_[static_cast<int>(OperationProcess::RETURN)].y, 1.0f, 0.0f,
353                   lpImageManager.GetHandle("Images/button_return.png")[0], true);
354     if (nowPhase_ == Phase::FIRST && inputDirectoryName_)
355     {
356         DrawFormatString(inputBoxPos_.x - (inputBoxSize_.x / 2),
357                          inputBoxPos_.y - (inputBoxSize_.y / 2), 0xffffffff, inputDirectoryName_);
358     }
359     // マスキングが所定の場所に来たら透過された箱を描画
360     if (nowOperation_ != OperationProcess::MAX)
361     {
362         // 透過させるためにアルファブレンドする(そこそこ薄く箱を描画させたいため150という値)
363         SetDrawBlendMode(DX_BLENDMODE_ALPHA, 150);
364         DrawBox(processPos_[static_cast<int>(nowOperation_)].x - (boxSize_[static_cast<int>(nowOperation_)].x / 2),
365                 processPos_[static_cast<int>(nowOperation_)].y - (boxSize_[static_cast<int>(nowOperation_)].y / 2),
366                 processPos_[static_cast<int>(nowOperation_)].x + (boxSize_[static_cast<int>(nowOperation_)].x / 2),
367                 processPos_[static_cast<int>(nowOperation_)].y + (boxSize_[static_cast<int>(nowOperation_)].y / 2), 0xff0000, true);
368         // アルファブレンドを箱だけに適応させたいので、箱の描画の最後に0でリセットをかける
369         SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
370     }
371 }
```

InputDirectoryName関数

InputObjectName関数

```
873 void ImageResourceDataEdit::InputDirectoryName(void)
874 {
875     // デルタ外リ名の入力
876     inputHandle_[0] = KeyInputSingleCharString(inputBoxPos_.x - (inputBoxSize_.x / 2),
877                                                inputBoxPos_.y - (inputBoxSize_.y / 2), 100, inputDirectoryName_, true);
878     // オブジェクト名入力に移行
879     inputFunc_ = &ImageResourceDataEdit::InputObjectName;
880
881 }
882 void ImageResourceDataEdit::InputObjectName(void)
883 {
884     // オブジェクト名の入力
885     inputHandle_[1] = KeyInputSingleCharString(inputBoxPos_.x - (inputBoxSize_.x / 2) + 100,
886                                                inputBoxPos_.y - (inputBoxSize_.y / 2), 100, inputObjectName_, true);
887     // デルタ外リ名をstring型にキャスト
888     auto dirName = (std::string)inputDirectoryName_;
889     // ファイル名をデルタ外リ名と入力したオブジェクト名で生成
890     auto filePath = dirName + (std::string)inputObjectName_;
891     // DataFile/よりデータファイルの読み込み
892     lpFileManager.ReadDataFile(filePath);
893     // ファイル名セル外に移行
894     nowPhase_ = Phase::SECOND;
895     inputFunc_ = &ImageResourceDataEdit::InputMaxFrame;
896 }
```

# ～担当箇所説明～

## ①アニメーションデータ作成ツール作成

```
398 void ImageResourceDataEdit::InputMaxFrame(void)
399 {
400     // 最大フレーム値の入力
401     inputHandle_[2] = KeyInputSingleCharString(100, 130, 100, inputMaxFrame_, true);
402     // 入力した文字をstring型にセット
403     auto str = (std::string)inputMaxFrame_;
404     // 入力した値のセット(std::atoi->string型をint型に変更)
405     IpSpriteDataCache.SetMaxFrame(std::atoi(str.c_str()));
406     inputFunc_ = &ImageResourceDataEdit::InputLoopFlag;
407 }
408
409 void ImageResourceDataEdit::InputLoopFlag(void)
410 {
411     // loopFlag(アニメーションをループ再生させるか)の入力(true or false)
412     bool flg = false;
413     inputHandle_[3] = KeyInputSingleCharString(100, 230, 100, inputLoopFlag_, true);
414     // 入力した文字をstring型にセット
415     auto loopStr = (std::string)inputLoopFlag_;
416     // 入力した値が"true"だったら
417     if (loopStr == "true")
418     {
419         // セットするフラグをtrueに
420         flg = true;
421     }
422     // 入力した値のセット
423     IpSpriteDataCache.SetLoopFlag(flg);
424     inputFunc_ = &ImageResourceDataEdit::InputDuration;
425 }
426
427 void ImageResourceDataEdit::InputDuration(void)
428 {
429     // duration(1コマにかかる時間)の入力
430     inputHandle_[4] = KeyInputSingleCharString(100, 330, 100, inputDuration_, true);
431     std::string durationStr = inputDuration_;
432     // 入力した値のセット(std::stof -> string型をfloat型に変更)
433     IpSpriteDataCache.SetDuration(std::stof(durationStr));
434     // 処理段階の移行
435     nowPhase_ = Phase::FOURTH;
436 }
```

InputMaxFrame関数

InputLoopFlag関数

InputDuration関数

# ～担当箇所概要 | ～ ①アニメーションデータ作成ツール

シーン作成



InputDirectoryName関数でデータファイルのディレクトリ名入力

+

InputObjectName関数でオブジェクト名入力

```
92 void FileManager::ReadDataFile(const std::string& pathName)
93 {
94     // 読み込むデータファイル名
95     std::string fileName = pathName + ".dat";
96     // ファイル
97     std::ifstream file(fileName, std::ios::in | std::ios::binary);
98     std::stringstream stringStream;
99     std::string str1;
100
101    do
102    {
103        // カス区切りで一行読む
104        std::getline(file, str1, ',');
105        // アニメーション名のリストに格納していく
106        animationNameList_.push_back(str1);
107    } while (!file.eof());
108    // ファイルを閉じる
109    file.close();
110 }
```

FileManagerクラスの  
ReadDataFile関数で  
InputDirectoryName関数と  
InputObjectName関数で入力した  
文字列でデータファイルを読み込む

オブジェクトに対応する  
アニメーション名達を読み込む



次ページへ続く

D:\RectEngine\DataFiles\Details\Warrior\Sprite\Attack.dat - sakura 2.4.1.2849

ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	<Data>	12,true,0.08														
2	/Data>															
3	<Image=Warrior_Attack_01.png,															
4	Warrior_Attack_02.png,															
5	Warrior_Attack_03.png,															
6	Warrior_Attack_04.png,															
7	Warrior_Attack_05.png,															
8	Warrior_Attack_06.png,															
9	Warrior_Attack_07.png,															
10	Warrior_Attack_08.png,															
11	Warrior_Attack_09.png,															
12	Warrior_Attack_10.png,															
13	Warrior_Attack_11.png,															
14	Warrior_Attack_12.png															
15	/Image>[EOF]															

読み込む際はこのようなファイルを  
読み込む

# ～担当箇所概要 II～ ①アニメーションデータ作成ツール

読み込んだアニメーション名リストから  
特定のアニメーション名を選択

```
31 void FileManager::SearchFileFromDirectory(const std::string& directoryName)
32 {
33     if (fileNameList_.size() <= 0)
34     {
35         HANDLE hFind;
36         WIN32_FIND_DATA win32fd;
37         // 検索するルートに到達するまでのパス
38         std::string path = directoryName + "/";
39         // 対象にするファイル拡張子
40         std::string extension = ".png";
41         // 検索にかける名前
42         // (.pngとは → .pngとつくファイル名全てを検索かけるという事【ワイルドカード】とも呼ぶ)
43         std::string search_name = path + "*." + extension;
44
45         hFind = FindFirstFile(search_name.c_str(), &win32fd);
46         // ファイルがなければ
47         if (hFind == INVALID_HANDLE_VALUE) {
48             throw std::runtime_error("file not found");
49             return;
50         }
51
52         // 指定のディレクトリ以下のファイル名をファイルがなくなるまで取得する
53         do {
54             if (win32fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
55                 // ディレクトリの場合は何もしない
56                 continue;
57             }
58             else {
59                 fileNameList_.push_back(win32fd.cFileName);
60             }
61         } while (FindNextFile(hFind, &win32fd));
62         // ファイルを閉じる
63         FindClose(hFind);
64     }
65 }
```

これらのデータ(①②③④)を  
データファイルに書き込む



(オブジェクト名/アニメーション名/)直下にある  
.pngファイルを全て抽出→fileNameList\_①に格納  
InputMaxFrame関数②、InputLoopFlag関数③、  
InputDuration関数④でアニメーションに必要なデータを  
入力させる



```
112 void FileManager::WriteImageDataFile(const std::string& fileName, const std::vector<std::string>& strVec)
113 {
114     std::string dataFileName = "DataFiles/Details/" + fileName + ".dat";
115     // 書き込むデータファイル
116     std::ofstream file(dataFileName);
117     std::stringstream stringStream;
118     std::string str1;
119     int i = 0;
120     for (; i < 3; i++)
121     {
122         // 最初だけ見出しを付ける
123         if (i < 0)
124         {
125             file << "<Data>";
126         }
127         // strVecに格納された文字列を書き込んでいく
128         file << strVec[i];
129
130         // 最後になったら改行
131         if (i >= 2)
132         {
133             file << std::endl;
134             file << "/Data>";
135         }
136         else
137         {
138             // カタログ区切る
139             file << ",";
140         }
141     }
142 }
```

```
142     for (; i < strVec.size(); i++)
143     {
144         // 最初だけ見出しを付ける
145         if (i <= 0)
146         {
147             // 改行
148             file << std::endl;
149             // 見出し
150             file << "<Image>";
151         }
152         // 最後の行じゃなければ
153         if (i < strVec.size() - 1)
154         {
155             // 文字列書き込み
156             file << strVec[i];
157             // カタログ書き込み
158             file << ",";
159             // 改行
160             file << std::endl;
161         }
162         // 最後の行だとカタログは不要なので
163         else
164         {
165             // 文字列書き込み
166             file << strVec[i];
167             // 改行
168             file << std::endl;
169         }
170     }
171     // 最後に付け加える
172     file << "/Image>";
173     // ファイルを閉じる
174     file.close();
175 }
```

# ～担当箇所概要 III～ ①アニメーションデータ作成ツール

データを書き込んだ結果として



このような形で書き込まれる

D:\RectEngin\DataFiles\Details\Warrior\Sprite\Attack.dat - sakura 2.4.1.2849

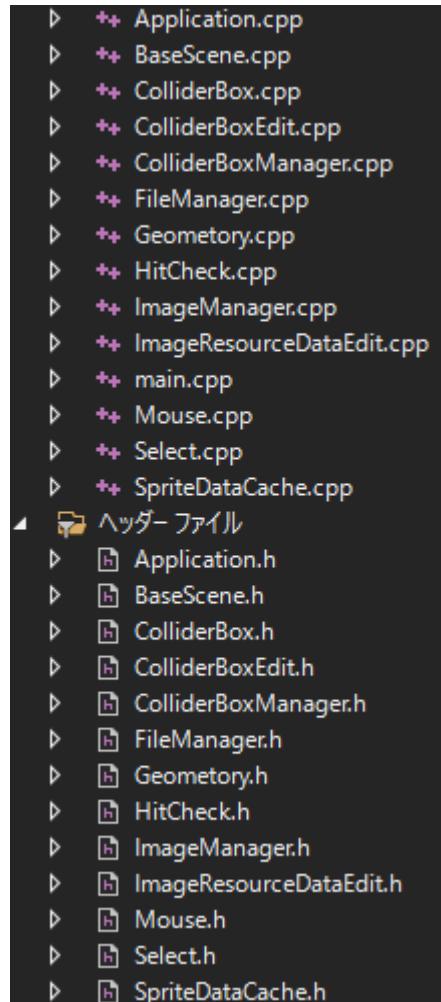
ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)

1 <Data=12,true,0.08↵  
2 /Data↵  
3 <Image=Warrior\_Attack\_01.png,↵  
4 Warrior\_Attack\_02.png,↵  
5 Warrior\_Attack\_03.png,↵  
6 Warrior\_Attack\_04.png,↵  
7 Warrior\_Attack\_05.png,↵  
8 Warrior\_Attack\_06.png,↵  
9 Warrior\_Attack\_07.png,↵  
10 Warrior\_Attack\_08.png,↵  
11 Warrior\_Attack\_09.png,↵  
12 Warrior\_Attack\_10.png,↵  
13 Warrior\_Attack\_11.png,↵  
14 Warrior\_Attack\_12.png↵  
15 /Image>[EOF]

最初の12は最大フレーム(コマ)数  
Trueはループフラグ(アニメーションがループ再生されるか)  
0.08がduration(1コマにかかる時間)  
最後の.pngの文字列が画像ファイル名群

# ～担当箇所説明～

## ②当たり判定Box作成ツール作成



ColliderBoxEditクラス：当たり判定Boxの編集シーン

ColliderBoxクラス：作成した個々の当たり判定Boxクラス  
(ポジション、サイズ、攻撃なのが食らいなのかタイプ  
を持っている)

ColliderBoxManagerクラス：ColliderBoxEditで作成した  
ColliderBox達を一まとめにして一括管理  
するクラス  
(ColliderBoxの生成、追加、描画、破棄を担当)

# ～担当箇所説明～

## ②当たり判定Box作成ツール作成

### ColliderBoxEdit.h

```
10  class ColliderBoxManager;
11
12  enum class CLICK
13  {
14      RIGHT,
15      LEFT,
16      MAX
17  };
18
19  //当たり判定矩形の編集ツール
20  class ColliderBoxEdit :
21  public BaseScene
22  {
23  public:
24      ColliderBoxEdit();
25      ~ColliderBoxEdit();
26      // 更新
27      UniqueScene Update(UniqueScene own, const std::shared_ptr<Mouse>& mouse);
28      // 初期化
29      void Initialize(void);
30      // フェーズ毎の処理の初期化
31      void InitializeUpdater(void);
32      // フェーズ毎の描画処理の初期化
33      void InitializeDrawExecuter(void);
34      // 描画
35      void Draw(void);
36
37      // オブジェクト名を入力する処理
38      void InputObjectName(void);
39      // アニメーション名を入力する処理
40      void InputAnimationName(void);
41      // アニメーションのコマを指定する処理
42      void SelectFrame(const CLICK& click, const Vector2I& mPoint);
43      // 当たり判定矩形の編集
44      void BoxEdit(const CLICK& click, const Vector2I& mPoint);
45
46      private:
47          // 入力オブジェクト名
48          char inputObjectName_[20];
49          // 入力アニメーション名
50          char inputAnimationName_[20];
51          // オブジェクト名を入力するボックスの位置
52          Vector2I objectNameBoxPos_;
53          // オブジェクト名を入力するボックスのサイズ
54          Vector2I objectNameBoxSize_;
55          // アニメーション名を入力するボックスの位置
56          Vector2I animationNameBoxPos_;
57          // アニメーション名を入力するボックスのサイズ
58          Vector2I animationNameBoxSize_;
59          // フレーム数を表示するボタン
60          // フレームだけsizeがあるので可変にしている
61          std::vector<std::pair<Vector2I, int>> frameNumBox_;
62          // フレームを表示するボックスのサイズ
63          Vector2I frameNumBoxSize_;
64          // 矩形タイプ変更を行うボタンの位置
65          std::array<Vector2I, 2> typeChangerPos_;
66          // 矩形タイプ変更を行うボタンのサイズ
67          Vector2I typeChangerBoxSize_;
68          // セーブボタンの位置
69          Vector2I saveButtonPos_;
70          // セーブボタンのサイズ
71          Vector2I saveButtonSize_;
72          // データが無かった時の画像データ編集に移行するボタンの位置
73          Vector2I goImageDataButtonPos_;
74          // 画像データ編集に移行するボタンのサイズ
75          Vector2I goImageDataButtonSize_;
76          // アプリケーションを終了させるボタンの位置
77          Vector2I exitAppButtonPos_;
78          // アプリケーションを終了させるボタンのサイズ
79          Vector2I exitAppButtonSize_;
80          // セット画面に戻るボタンの上にマウスカーソルが来たフラグ
81          bool isOnReturn_;
82          // アプリケーションを終了させるボタンの上にマウスカーソルが来たフラグ
83          bool isOnExitApp_;
84
85          // 編集対象画像の位置
86          Vector2I objImagePos_;
87
88          // マウスカーソルが乗っているフレーム
89          int onPointFrame_;
90          // 決定(左クリック)後のフレーム数
91          int selectedFrame_;
92
93          // 現在の処理段階
94          Phase nowPhase_;
95          // 左右クリック回数
96          CLICK clickDir_;
97          // マウスカーソル
98          Vector2I mPoint_;
99
100         // 入力ターンを表す関数用カウント
101         void(ColliderBoxEdit::* inputFunc_)(void);
102         // 編集処理を表す関数用カウント
103         void(ColliderBoxEdit::* editFunc_)(const CLICK& click, const Vector2I& mPoint);
104         // ツールの中での状態遷移関数をstd::functionでまとめた
105         std::map<Phase, std::function<void(const std::shared_ptr<Mouse>& mouse)>> updater_;
106         // 描画遷移関数をstd::functionでまとめた
107         std::map<Phase, std::function<void(void)>> drawExecutor_;
108         // ファイルが見つかった際にtrue
109         bool isFindFile_;
110
111         // 当たり判定ボックスの生成等を担うManager
112         ColliderBoxManager* boxManager_;
113
114         // ボックスタイプ毎の色
115         unsigned int typeColor_;
116         // ボックスタイプ
117         bool typeFlag_;
118         // セーブしたフラグ
119         std::vector<bool> saveFlag_;
120         // 再度リセットする際にフラグ等をリセット
121         void ResetData(void);
122     };
123 }
```

# ～担当箇所説明～

## ColliderBoxEdit.cpp

### コンストラクタ、デストラクタ

```
14  ColliderBoxEdit::ColliderBoxEdit()
15  {
16      // アプリケーション終了フラグの初期化
17      exitAppFlag_ = false;
18      // 各種変数の初期化
19      Initialize();
20  }
21
22  ColliderBoxEdit::~ColliderBoxEdit()
23  {
24      // このシーンが破棄される際にリセットさせる
25      ResetData();
26  }
```

### Update関数

```
28  UniqueScene ColliderBoxEdit::Update(UniqueScene own, const std::shared_ptr<Mouse>& mouse)
29  {
30      // クリック情報
31      auto clickData = mouse->GetClick();
32      // マウスボタンの位置
33      mPoint_ = mouse->GetPos();
34      // 左右クリック変数を常にMAXにしておく
35      // しておくことで、常に右クリックされている状態や左クリックされている状態を避けるため
36      clickDir_ = CLICK::MAX;
37      // クリック情報を回す
38      for (auto click : clickData)
39      {
40          // クリックトリガー有り
41          if (click.second[static_cast<int>(TRG::NOW)] && !click.second[static_cast<int>(TRG::OLD)])
42          {
43              // 左クリック時
44              if (click.first == INPUT_ID::START)
45              {
46                  clickDir_ = CLICK::LEFT;
47              }
48              // 右クリック時
49              if (click.first == INPUT_ID::RESET)
50              {
51                  clickDir_ = CLICK::RIGHT;
52              }
53          }
54      }
55  }
```

# ②当たり判定Box作成ツール作成

### Update関数の続き

```
55  // 特定のアニメーション名やオブジェクト名でデータファイル検索してデータファイルが見つからなかったとき
56  if (!isFindFile_)
57  {
58      // ImageData編集ボタンにマウスカーソルが来たら
59      if (HitCheck_FromCenter()(goImageDataButtonPos_, mPoint_, goImageDataButtonSize_))
60      {
61          // マウスカーソルが上に来たフラグをtrueに
62         .isOnReturn_ = true;
63          // 左クリック時
64          if (clickDir_ == CLICK::LEFT)
65          {
66              // ImageDataEditに移行
67              return std::make_unique<ImageResourceDataEdit>();
68          }
69      }
70      else
71      {
72          // マウスカーソルが来てない
73          .isOnReturn_ = false;
74      }
75  }
```

```
75  // マウスカーソルがアプリケーション終了ボタンの上に来た時
76  if (HitCheck_FromCenter()(exitAppBarPos_, mPoint_, exitAppBarSize_))
77  {
78      // マウスカーソルが上に来たフラグをtrueに
79      .isOnExitApp_ = true;
80      // 左クリック時
81      if (clickDir_ == CLICK::LEFT)
82      {
83          // アプリケーション終了フラグをtrueにしアプリケーションを終了
84          exitAppFlag_ = true;
85      }
86  }
87  else
88  {
89      // マウスカーソルが来てない
90      .isOnExitApp_ = false;
91  }
92
93  CIsDrawScreen();
94  // 描画
95  Draw();
96  ScreenFlip();
97  // 更新
98  updaters_[nowPhase_](mouse);
99  return move(own);
100 }
```

# ～担当箇所説明～

## ②当たり判定Box作成ツール作成

ColliderBoxEdit.cpp

### Initialize関数

```
103 void ColliderBoxEdit::Initialize(void)
104 {
105     // FileManagerの生成(Singletonインスタンス)
106     FileManager::Create();
107     // 初期入力ターンをオブジェクト名の入力にする
108     inputFunc_ = &ColliderBoxEdit::InputObjectName;
109     // 初期編集ターンをアニメーションのコマを選択させるターンにする
110     editFunc_ = &ColliderBoxEdit::SelectFrame;
111     // 現在の段階をFIRSTにする
112     nowPhase_ = Phase::FIRST;
113     // 各画面サイズの取得
114     auto vp = IApplication::GetViewport();
115
116     // 各ボタンの位置やサイズの初期化-----
117     objectNameBoxPos_ = [ vp.screen.x / 2 - 100, vp.screen.y / 2 - 100 ];
118     objectNameBoxSize_ = [ 100,25 ];
119     animationNameBoxPos_ = [ vp.screen.x / 2 - 100, vp.screen.y / 2 + 50 ];
120     animationNameBoxSize_ = [ 100,25 ];
121     frameNumBoxSize_ = [ 35,35 ];
122     typeChangerPos_ = [ Vector2I[ 200,100 ],Vector2I[200,130] ];
123     typeChangerBoxSize_ = [ 100,25 ];
124     saveButtonPos_ = [ 100,vp.screen.y - 100 ];
125     saveButtonSize_ = [ 65,25 ];
126     goImageDataButtonPos_ = [ vp.screen.x / 2, vp.screen.y / 2 ];
127     goImageDataButtonSize_ = [ 335,30 ];
128     exitAppButtonPos_ = [ vp.screen.x / 2, vp.screen.y / 2 + 100 ];
129     exitAppButtonSize_ = [ 175,26 ];
130     objImagePos_ = [ vp.editScreen.x - 100 + ((vp.screen.x-(vp.editScreen.x - 100))/2),
131     | vp.editScreen.y - 100 + ((vp.screen.y - (vp.editScreen.y - 100)) / 2) ];
132     // -----
133     // アニメーションコマを表示している所にマウスインタが来た時に更新される数字
134     // そのコマ数と同じ数字になる
135     onPointFrame_ = -1;
136     // 最終的に決定したアニメーションコマの初期化
137     selectedFrame_ = onPointFrame_;
138     // データファイル検索フラグの初期化
139     isFindFile_ = false;
140     // 描く矩形のタイプ(Attack(true) or Damage(false))
141     typeFlag_ = false;
142     // ImageDataEditに戻るボタンの上に
143     // マウスインタが来たフラグの初期化
144     isOnReturn_ = false;
145     // アップリケーション終了ボタンの上にマウスインタが来たフラグの初期化
146     isOnExitApp_ = false;
147     // 更新関数達の格納
148     InitializeUpdater();
149     // 描画関数達の格納
150     InitializeDrawExcecute();
151
152     // 各画像のロード -----
153     IImageManager::Load("Images/saveButton.png");
154     IImageManager::Load("Images/goImageDataEditButton.png");
155     IImageManager::Load("Images/exit_application.png");
156     // -----
```

# ～担当箇所説明～

## ②当たり判定Box作成ツール作成

ColliderBoxEdit.cpp

InitializeUpdater関数 → このシーンの各段階における  
更新関数の格納

```
159 void ColliderBoxEdit::InitializeUpdater(void)
160 {
161     // このシーンの1番目に行われる更新処理
162     updater_.try_emplace(Phase::FIRST, [&](const std::shared_ptr<Mouse>& mouse) {
163         // 各種入力処理
164         (this->*inputFunc_)();
165     });
166     // このシーンの2番目に行われる更新処理
167     updater_.try_emplace(Phase::SECOND, [&](const std::shared_ptr<Mouse>& mouse) {
168         // データファイルの検索(入力オブジェクト名とアニメーション名で)
169         isFindfile_ = IpFileManager::IsExistFile("Sprite", inputObjectName_, inputAnimationName_);
170         // データファイルが存在する
171         if (isFindFile_)
172         {
173             // 入力下アニメーション名をstringにセット
174             auto animName = (std::string)inputAnimationName_;
175             // フルパス生成
176             auto fullPath = "Images/Actor/" + (std::string)inputObjectName_ + "/" + animName;
177             // 入力したオブジェクト名とアニメーション名からフルパス内に存在するファイルを検索
178             IpFileManager::SearchFileFromDirectory(fullPath);
179             // データファイルから読み取った画像パスで画像のロード
180             for (auto& fileName : IpFileManager::GetFileNameList())
181             {
182                 IpImageManager::Load("Images/Actor/" + (std::string)inputObjectName_ +
183                                     "/" + animName + "/" + fileName);
184             }
185             // ループ数を表示する箱のresize
186             frameNumBox_.resize(IpFileManager::GetFileNameList().size());
187             // 箱のホリゾンとそこに書いている数字の設定
188             for (int i = 0; i < frameNumBox_.size(); i++)
189             {
190                 frameNumBox_[i].first = { 50 + (i * 50), 30 };
191                 frameNumBox_[i].second = i;
192             }
193             // 次の処理ターゲットに移行
194             nowPhase_ = Phase::THIRD;
195         }
196     });
197     // このシーンの3番目に行われる更新処理
198     updater_.try_emplace(Phase::THIRD, [&](const std::shared_ptr<Mouse>& mouse) {
199         // 編集の際に行われる更新処理(引数: 左右クリック, マウスボイント)
200         (this->*editFunc_)(clickDir_, mPoint_);
201     });
202 }
```

一段階目の処理の格納

二段階目の処理の格納

三段階目の処理の格納

# ～担当箇所説明～

## ②当たり判定Box作成ツール作成

ColliderBoxEdit.cpp

InitializeDrawExeuctor関数

このシーンにおける  
各段階毎の描画処理の格納

二段階目の描画処理の格納

```
204 void ColliderBoxEdit::InitializeDrawExeuctor(void)
205 {
206     // このシーンの1番目に行われる更新処理
207     drawExeuctor_.try_emplace(Phase::FIRST, [&](){
208         // オブジェクト名の入力フォーム
209         DrawFormatString(objectNameBoxPos_.x, objectNameBoxPos_.y - 25, 0xffffffff, "オブジェクト名を入力");
210         DrawBox(objectNameBoxPos_.x, objectNameBoxPos_.y,
211             objectNameBoxPos_.x + objectNameBoxSize_.x, objectNameBoxPos_.y + objectNameBoxSize_.y,
212             0xffffffff, false);
213         // アニメーション名の入力フォーム
214         DrawFormatString(animationNameBoxPos_.x, animationNameBoxPos_.y - 25, 0xffffffff, "アニメーション名を入力");
215         DrawBox(animationNameBoxPos_.x, animationNameBoxPos_.y,
216             animationNameBoxPos_.x + animationNameBoxSize_.x, animationNameBoxPos_.y + animationNameBoxSize_.y,
217             0xffffffff, false);
218         // 入力したオブジェクト名の表示
219         DrawFormatString(objectNameBoxPos_.x, objectNameBoxPos_.y, 0xffffffff, "%s", inputObjectName_);
220         // 入力したアニメーション名の表示
221         DrawFormatString(animationNameBoxPos_.x, animationNameBoxPos_.y, 0xffffffff, "%s", inputAnimationName_);
222     });
223 }
```

一段階目の描画処理の格納

```
224 // このシーンの2番目に行われる更新処理
225 drawExeuctor_.try_emplace(Phase::SECOND, [&](){
226     if (!isFindFile_)
227     {
228         // データファイル検索する際の文字列
229         auto str = (std::string)inputObjectName_ + "_" + (std::string)inputAnimationName_;
230         // データがない事を知らせる
231         DrawFormatString(0, 0, 0xffffffff,
232             "データがありません！%s%のImageDataを先に終了させて下さい。",
233             str.c_str());
234         // ImageDataEditに戻るためのボタンの描画
235         DrawRotaGraph(goImageDataButtonPos_.x, goImageDataButtonPos_.y, 1.0f, 0.0f,
236             lpImageManager.GetHandle("Images/goImageDataEditButton.png") [0],
237             true);
238         // アプリケーション終了するボタンの描画
239         DrawRotaGraph(exitAppButtonPos_.x, exitAppButtonPos_.y, 1.0f, 0.0f,
240             lpImageManager.GetHandle("Images/exit_application.png") [0],
241             true);
242         // 特定の場所にマウスがいたらそこに半透明のボタンを描画-----
243         // ImageDataEditに戻るボタン上
244         if (isOrReturn_)
245         {
246             SetDrawBlendMode(DX_BLENDMODE_ALPHA, 200);
247             DrawBox(goImageDataButtonPos_.x - (goImageDataButtonSize_.x / 2),
248                 goImageDataButtonPos_.y - (goImageDataButtonSize_.y / 2),
249                 goImageDataButtonPos_.x + (goImageDataButtonSize_.x / 2),
250                 goImageDataButtonPos_.y + (goImageDataButtonSize_.y / 2), 0xff0000, true);
251             SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
252         }
253         // アプリケーション終了のボタン上
254         if (isOrExitApp_)
255         {
256             SetDrawBlendMode(DX_BLENDMODE_ALPHA, 200);
257             DrawBox(exitAppButtonPos_.x - (exitAppButtonSize_.x / 2),
258                 exitAppButtonPos_.y - (exitAppButtonSize_.y / 2),
259                 exitAppButtonPos_.x + (exitAppButtonSize_.x / 2),
260                 exitAppButtonPos_.y + (exitAppButtonSize_.y / 2), 0x0000ff, true);
261             SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
262         }
263     }
264 });
265 });
266 });

// -----
```

# ～担当箇所説明～

## ColliderBoxEdit.cpp

### 三段階目の描画処理の格納

```
287 // この一の3番目に行われる更新処理
288 drawExeuctor_.try_emplace(Phase::THIRD, [&] () {
289     // Applicationから各画面物体の取得
290     auto vp = lpApplication.GetViewport();
291     // 背景
292     DrawBox(0, 0, vp.screen.x, vp.screen.y, 0x000fff, true);
293     // やう画像のスクリーン背景
294     DrawBox(vp.editScreen.x - 100, vp.editScreen.y - 100,
295            vp.screen.x, vp.screen.y, 0x000000, true);
296     // ルームを表示させている上に描画
297     DrawFormatString(50, 0, 0xfffff, "各ルーム");
298     // 各ルームタブを1,2,3,4,5,6,7.....と描画していく
299     for (int i = 0; i < frameNumBox_.size(); i++)
300     {
301         DrawBox(frameNumBox_[i].first.x, frameNumBox_[i].first.y,
302                frameNumBox_[i].first.x + frameNumBoxSize_.x,
303                frameNumBox_[i].first.y + frameNumBoxSize_.y,
304                0xfffff, false);
305         DrawFormatString(frameNumBox_[i].first.x + 10,
306                         frameNumBox_[i].first.y + 10, 0xfffff, "x", frameNumBox_[i].second);
307         // saveFlag_が押下されてから描画開始
308         if (saveFlag_.size() > 0)
309         {
310             // そのコマがセーブ完了していたら上に赤色で「済」という文字を表示
311             if (saveFlag_[i])
312             {
313                 DrawFormatString(frameNumBox_[i].first.x + 10, frameNumBox_[i].first.y - 15,
314                                  0xff0000, "済");
315             }
316         }
317         // オブジェクトがそのフレームのボタン上に来たら
318         // オブジェクトが来たフレームを薄い緑で塗りつぶす
319         if (onPointFrame_ > -1)
320         {
321             SetDrawBlendMode(DX_BLENDMODE_ALPHA, 200);
322             DrawBox(frameNumBox_[onPointFrame_].first.x, frameNumBox_[onPointFrame_].first.y,
323                    frameNumBox_[onPointFrame_].first.x + frameNumBoxSize_.x,
324                    frameNumBox_[onPointFrame_].first.y + frameNumBoxSize_.y, 0x00ff00, true);
325             SetDrawBlendMode(DX_BLENDMODE_NOBLEND, 0);
326         }
327     }
328 }
```

②当たり判定Box作成ツール作成

```
308 // 編集対象キャラ画像の描画と編集対象画像のファイル名の描画
309 if (selectedFrame_ != -1)
310 {
311     auto fileName = "Images/Actor/" + (std::string)inputObjectName_ +
312     "/" + (std::string)inputAnimationName_ + "/" + lpFileManager.GetFileNameList()[selectedFrame_];
313     DrawRotatGraph(objImagePos_.x, objImagePos_.y, 1.0f, 0.0f, lpImageManager.GetHandle(fileName)[0], true);
314     DrawFormatString(vp.editScreen.x - 100, vp.editScreen.y - 125, 0xfffff, "%s", fileName.c_str());
315 }
316 // 矩形編集の処理
317 if (editFunc_ == &ColliderBoxEdit::BoxEdit)
318 {
319     // オブジェクトの先に矩形描画毎の色で小さな丸を塗りつぶして描画
320     DrawCircle(mPoint_.x, mPoint_.y, 5, typeColor_, true);
321 }
322 // 矩形タイプの変更がめの描画
323 DrawFormatString(typeChangerPos_[0].x - 80, typeChangerPos_[0].y, 0xfffff, "矩形タイプ:");
324 for (auto typePos : typeChangerPos_)
325 {
326     DrawBox(typePos.x, typePos.y,
327             typePos.x + typeChangerBoxSize_.x, typePos.y + typeChangerBoxSize_.y, 0xfffff, false);
328 }
329 DrawFormatString(typeChangerPos_[0].x + 5, typeChangerPos_[0].y + 5, 0xff0000, "攻撃矩形");
330 DrawFormatString(typeChangerPos_[1].x + 5, typeChangerPos_[1].y + 5, 0x00f00, "タグ矩形");
331 //
332 // 矩形タイプ番号
333 auto typeNo = -1;
334 // 攻撃矩形だと
335 if (typeFlag_)
336 {
337     // 0
338     typeNo = 0;
339 }
340 // タグ矩形だと
341 else
342 {
343     // 1
344     typeNo = 1;
345 }
346 // 攻撃矩形かタグ矩形ならば
347 if (typeNo != -1)
348 {
349     // 現在選択されている横に←描画
350     // 攻撃矩形 ← や タグ矩形 ← みたいに描画させる
351     DrawFormatString(typeChangerPos_[typeNo].x + 120, typeChangerPos_[typeNo].y + 5,
352                      0xfffff, "<→", true, 10);
353 }
354 // boxManager_が生成されていて、ループ数が設定されていたら
355 if (boxManager_ && selectedFrame_ != -1)
356 {
357     // 矩形の始点
358     auto begPos = boxManager_->GetBegPos();
359     // 矩形の始点がエーペンション上にない場合
360     if (begPos.x > -1 && begPos.y > -1)
361     {
362         // 画像編集画面(黒色の画面)にマウスが有れば
363         if (HitCheck_FromLeftUp()
364             (Vector2I(vp.editScreen.x - 100, vp.editScreen.y - 100), mPoint_),
365             Vector2I((vp.screen.x - (vp.editScreen.x - 100)),
366                      ((vp.screen.y - (vp.editScreen.y - 100))))))
367         {
368             // 現在編集している矩形の描画
369             DrawBox(begPos.x, begPos.y, mPoint_.x, mPoint_.y, typeColor_, false);
370         }
371     }
372     // boxManager_内に追加されたcolliderBox達の描画(指定したフレームの矩形で)
373     boxManager_->Draw(selectedFrame_);
374 }
375 // セーブボタンの描画
376 DrawGraph(saveButtonPos_.x, saveButtonPos_.y,
377           lpImageManager.GetHandle("Images/saveButton.png")[0], true);
378 // セーブボタンの周りに箱を描画
379 DrawBox(saveButtonPos_.x, saveButtonPos_.y,
380         saveButtonPos_.x + saveButtonSize_.x, saveButtonPos_.y + saveButtonSize_.y,
381         0xfffff, false);
382 });
383 
```

# ～担当箇所説明～

## ColliderBoxEdit.cpp

このシーンの  
Draw関数 → 大本の描画処理

```
386 void ColliderBoxEdit::Draw(void)
387 {
388     // 各段階毎の描画
389     drawExeuctor_[nowPhase_]();
390 }
```

## InputAnimationName関数

```
402 void ColliderBoxEdit::InputAnimationName(void)
403 {
404     // アニメーション名の入力
405     if (KeyInputSingleCharString(animationNameBoxPos_.x, animationNameBoxPos_.y, 20, inputAnimationName_, true) == 1)
406     {
407         // 次の処理プロセスに移行
408         nowPhase_ = Phase::SECOND;
409     }
410 }
```

## InputObjectName関数

```
392 void ColliderBoxEdit::InputObjectName(void)
393 {
394     // オブジェクト名の入力
395     if (KeyInputSingleCharString(objectNameBoxPos_.x, objectNameBoxPos_.y, 20, inputObjectName_, true) == 1)
396     {
397         // 終了時にアニメーション名入力に移行
398         inputFunc_ = &ColliderBoxEdit::InputAnimationName;
399     }
400 }
```

## ②当たり判定Box作成ツール作成

### SelectFrame関数

```
412 void ColliderBoxEdit::SelectFrame(const CLICK& click, const Vector2I& mPoint)
413 {
414     // クリックがどのフレームのボックスに来ているかの変数に
415     // 収集値を格納させる(常に)
416     onPointFrame_ = -1;
417     // フレームの数だけで回す
418     for (auto frameBox : frameNumBox_)
419     {
420         // クリックが特定のフレーム番号の上に来た時
421         if (HitCheck_FrmLeftUp()(frameBox.first, mPoint, frameNumBoxSize_))
422         {
423             // クリックが上に来たフレームを格納
424             onPointFrame_ = frameBox.second;
425             // 左クリック時
426             if (clickDir_ == CLICK::LEFT)
427             {
428                 // 選択フレームをクリックが来ているフレームにする
429                 selectedFrame_ = onPointFrame_;
430                 if (!boxManager_)
431                 {
432                     // ボックスマネージャーの生成
433                     boxManager_ = new ColliderBoxManager(frameNumBox_.size());
434                 }
435             }
436             saveFlag_.resize(frameNumBox_.size());
437             // 箱の編集段階に移行
438             editFunc_ = &ColliderBoxEdit::BoxEdit;
439         }
440     }
441 }
442 }
```

# ～担当箇所説明～

## ColliderBoxEdit.cpp

### BoxEdit関数

実際に当たり判定Boxを  
編集する処理

```
444 void ColliderBoxEdit::BoxEdit(const CLICK& click, const Vector2I& mPoint)
445 {
446     // 矩形タイプの変更
447     // 0:攻撃矩形タイプ 1:ダメージ矩形タイプ
448     if (HitCheck_FromLeftUp()(typeChangerPos_[0], mPoint, typeChangerBoxSize_))
449     {
450         if (click == CLICK::LEFT)
451         {
452             typeFlag_ = true;
453         }
454     }
455     if (HitCheck_FromLeftUp()(typeChangerPos_[1], mPoint, typeChangerBoxSize_))
456     {
457         if (click == CLICK::LEFT)
458         {
459             typeFlag_ = false;
460         }
461     }
462     // -----
463     // 攻撃矩形編集だと
464     if (typeFlag_)
465     {
466         // 赤に
467         typeColor_ = 0xff0000;
468     }
469     // ダメージ矩形編集だと
470     else
471     {
472         // 青に
473         typeColor_ = 0x00ff00;
474     }
}
```

## ②当たり判定Box作成ツール作成

```
475     // 各画面サイズの取得
476     auto vp = IApplication::GetViewport();
477     // 編集画面上にマウスがインタが存在すれば
478     if (HitCheck_FromLeftUp()
479         (Vector2I(vp.editScreen.x - 100, vp.editScreen.y - 100), mPoint_,
480          Vector2I((vp.screen.x - (vp.editScreen.x - 100)),
481                  ((vp.screen.y - (vp.editScreen.y - 100))))))
482     {
483         // 左クリック時
484         if (click == CLICK::LEFT)
485         {
486             // 矩形の始点の設定
487             boxManager_->SetBegPos(mPoint_);
488         }
489         // 右クリック時
490         if (click == CLICK::RIGHT)
491         {
492             // 終点が決まった事を意味するので
493             // この時点でcolliderBoxのAdd
494             boxManager_->AddColliderBox(selectedFrame_, mPoint_, typeFlag_);
495         }
496     }
497     // ベーピー上にマウスがインタが存在すれば
498     if (HitCheck_FromLeftUp()(saveButtonPos_, mPoint_, saveButtonSize_))
499     {
500         // 左クリック時
501         if (click == CLICK::LEFT)
502         {
503             // 現在の矩形データをセーブした方がtrueに
504             saveFlag_[selectedFrame_] = true;
505             // アメーバソマ数選択に変更
506             editFunc_ = &ColliderBoxEdit::SelectFrame;
507             // カホンインが存在するフレームの初期化
508             onPointFrame_ = -1;
509             // 決定時のフレームの初期化
510             selectedFrame_ = onPointFrame_;
511             // 全てのループ画像の編集の終了方が全てで戻す
512             auto flag = true;
513             for (auto frameFlag : saveFlag_)
514             {
515                 // 1つでもfalseだとfalseになるように論理積を取る
516                 flag &= frameFlag;
517             }
518         }
519     }
520     // セーブするデータ名
521     std::string fileName = (std::string)inputObjectName_ + "/Rect/" + (std::string)inputAnimationName_;
522     // データファイルを作成し、その中にデータを書き込んでいく
523     IFileManager::WriteRectDataFile(fileName, *boxManager_);
524     // 各変数等のリセット
525     ResetData();
526 }
527 }
528 }
529 }
530 }
```

# ～担当箇所説明～

ColliderBoxEdit.cpp

## ②当たり判定Box作成ツール作成

アプリを再開可能にするように  
ResetData関数 → 一回使用した変数等のリセットをする処理

```
532     void ColliderBoxEdit::ResetData(void)
533     {
534         // 初期入力カウンタをオブジェクト名の入力にする
535         inputFunc_ = &ColliderBoxEdit::InputObjectName;
536         // 初期フレーム選択カウンタをフレーム選択のターゲットにする
537         editFunc_ = &ColliderBoxEdit::SelectFrame;
538         // 各入力名の初期化
539         for (int i = 0; i < 20; i++)
540         {
541             inputObjectName_[i] = ' ';
542             inputAnimationName_[i] = ' ';
543         }
544         // boxManagerのdelete(開放)
545         delete boxManager_;
546         // boxManager_を空にしておく
547         boxManager_ = nullptr;
548         onPointFrame_ = -1;
549         selectedFrame_ = onPointFrame_;
550         // フレームボックスのプロパティコレクションを空にする
551         frameNumBox_.clear();
552         // セーブ完了フラグの中身を空にする
553         saveFlag_.clear();
554         // データファイル検索フラグの初期化
555         isFindFile_ = false;
556         // 矩形タイプ変数の初期化
557         typeFlag_ = false;
558         // 処理段階を初期に戻す
559         nowPhase_ = Phase::FIRST;
560     }
```

# ～担当箇所説明～

## ColliderBox.h

```
4 // 矩形クラス
5 class ColliderBox
6 {
7 public:
8     ColliderBox(const Vector2I& pos, const Vector2I& size, const bool& typeFlag);
9     ~ColliderBox();
10    // 矩形のタイプ取得
11    const bool& GetTypeFlag(void) const
12    {
13        return typeFlag_;
14    }
15    // 矩形の中心ポジションの取得
16    const Vector2I& GetPos(void) const
17    {
18        return pos_;
19    }
20    // 矩形サイズの取得
21    const Vector2I& GetSize(void) const
22    {
23        return size_;
24    }
25    // 自身の描画
26    void Draw(void);
27
28 private:
29     // 矩形の中心をさす
30     const Vector2I pos_;
31     // 矩形のサイズ
32     const Vector2I size_;
33     // Damage矩形かAttack矩形かの識別子
34     // Damage -> false Attack -> true
35     const bool typeFlag_;
36 };
```

## ②当たり判定Box作成ツール作成

## ColliderBox.cpp

```
4 ColliderBox::ColliderBox(const Vector2I& pos, const Vector2I& size, const bool& typeFlag):
5     pos_(pos), size_(size), typeFlag_(typeFlag)
6 {
7 }
8
9 ColliderBox::~ColliderBox()
10 {
11 }
12
13 void ColliderBox::Draw(void)
14 {
15     // 矩形の色
16     unsigned int color = 0xffffffff;
17     // 矩形タイプで色の変更
18     if (typeFlag_)
19     {
20         // 赤に(攻撃矩形の場合)
21         color = 0xffff00;
22     }
23     else
24     {
25         // 緑に(ダメージ矩形の場合)
26         color = 0x00ff00;
27     }
28     // 矩形の描画
29     DrawBox(pos_.x - (size_.x / 2), pos_.y - (size_.y / 2),
30             pos_.x + (size_.x / 2), pos_.y + (size_.y / 2), color, false);
31 }
32 }
```

# ～担当箇所説明～

## ColliderBoxManager.h

```
6   class ColliderBox;
7
8   // shared_ptrで生み出さないを防ぐ
9   using SharedCollider = std::shared_ptr<ColliderBox>;
10  // ループ数だけ可変で存在するのでresize可能なようにvector長にしておく
11  using ColliderVec = std::vector<SharedCollider>;
12
13 //当たり判定矩形をまとめて管理するクラス
14 // こちらの方がまとめて処理しやすい
15 class ColliderBoxManager
16 {
17 public:
18     ColliderBoxManager(int maxFrame);
19     ~ColliderBoxManager();
20
21     // 始点座標のセット
22     // クリックされたらセットする
23     void SetBegPos(const Vector2I& beg);
24
25     // 矩形の始点の取得
26     const Vector2I& GetBegPos(void) const
27     {
28         return begPos_;
29     }
30
31     // colliderBoxList_に追加
32     // param@ frame : 現在のフレーム番号
33     // param@ boxSize : 箱のサイズ
34     void AddColliderBox(const int& frame, const Vector2I& endPos, const bool& typeFlag);
35
36     void Draw(const int& frame);
37     // 矩形リストの取得
38     const std::vector<ColliderVec>& GetColliderBoxList(void) const
39     {
40         return colliderBoxList_;
41     }
42
43 private:
44     // 箱を描画し始める始点座標
45     Vector2I begPos_;
46     // 1フレームにも複数箱が存在するので可変にしておく
47     std::vector<ColliderVec> colliderBoxList_;
```

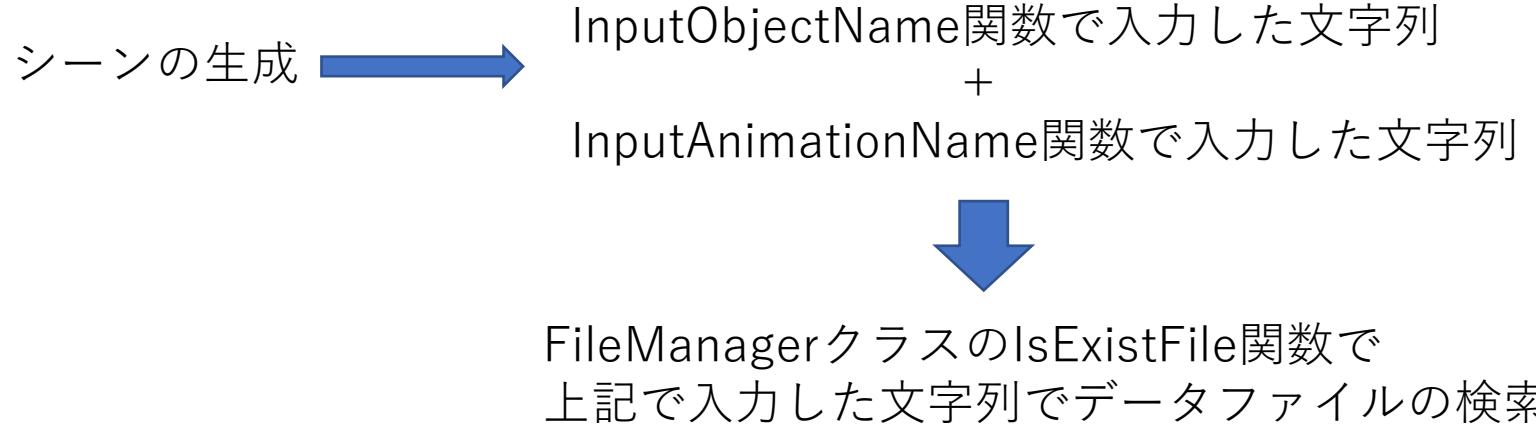
## ②当たり判定Box作成ツール作成

## ColliderBoxManager.cpp

```
4     ColliderBoxManager::ColliderBoxManager(int maxFrame)
5     {
6         // 生成時にフレームだけresizeしておく
7         colliderBoxList_.resize(maxFrame);
8         // 始点を0以外であればなんでもいいので
9         // とりあえず-1だったら引っかかる事もないで
10        // この数字で初期化
11        begPos_ = { -1,-1 };
12    }
13
14    ColliderBoxManager::~ColliderBoxManager()
15    {
16    }
17
18    void ColliderBoxManager::SetBegPos(const Vector2I& beg)
19    {
20        // 始点の設定
21        begPos_ = beg;
22    }
23
24    void ColliderBoxManager::AddColliderBox(const int& frame, const Vector2I& endPos, const bool& typeFlag)
25    {
26        // サイズの設定
27        auto size = Vector2I(endPos.x - begPos_.x, endPos.y - begPos_.y);
28        // 中心座標の設定
29        auto center = Vector2I(begPos_.x + (size.x / 2), begPos_.y + (size.y / 2));
30        // 矩形の生成
31        auto tmpCol = std::make_shared<ColliderBox>(center, size, typeFlag);
32        // 例外に追加
33        colliderBoxList_[frame].push_back(tmpCol);
34        // 始点のリセット
35        begPos_ = { -1,-1 };
36    }
37
38    void ColliderBoxManager::Draw(const int& frame)
39    {
40        // Addされた矩形の一括描画
41        for (auto& box : colliderBoxList_[frame])
42        {
43            box->Draw();
44        }
45    }
```

# ～担当箇所概要説明 | ~

## ②当たり判定Box作成ツール概要



```
67     bool FileManager::IsExistFile(const std::string& dataName,const std::string& objName,const std::string& animName)
68     {
69         // 検索をかけるファイル名
70         auto fileName = "DataFiles/Details/" + objName + "/" + dataName + "/" + animName;
71         HANDLE hFind;
72         WIN32_FIND_DATA win32fd;
73         // 検索にかける名前
74         std::string search_name = fileName + ".dat";
75
76         hFind = FindFirstFile(search_name.c_str(), &win32fd);
77
78         // ファイルが無ければfalseを返す
79         if (hFind == INVALID_HANDLE_VALUE) {
80             FindClose(hFind);
81             return false;
82         }
83         else
84         {
85             // ファイルが有ればtrueを返す
86             FindClose(hFind);
87             return true;
88         }
89     }
90 }
```

ここで検索するデータファイルは前述したアニメーションデータファイルを検索するので  
前もってアニメーションデータを作成していないといけない。

## ～担当箇所概要説明 II～

### ②当たり判定Box作成ツール概要

もし事前にデータファイルが作成されていなければ  
このように表示させるようにしている。



データファイルが作成されていると  
このような画面が表示される



# ～担当箇所概要説明 II～

## ②当たり判定Box作成ツール概要

現在編集している画像ファイル名

SelectFrame関数で

特定のアニメーションの

編集対象画像1枚を選択する

例) Attackアニメーションの5コマ目の画像

現在編集中のBoxタイプの選択

マウスポインタ

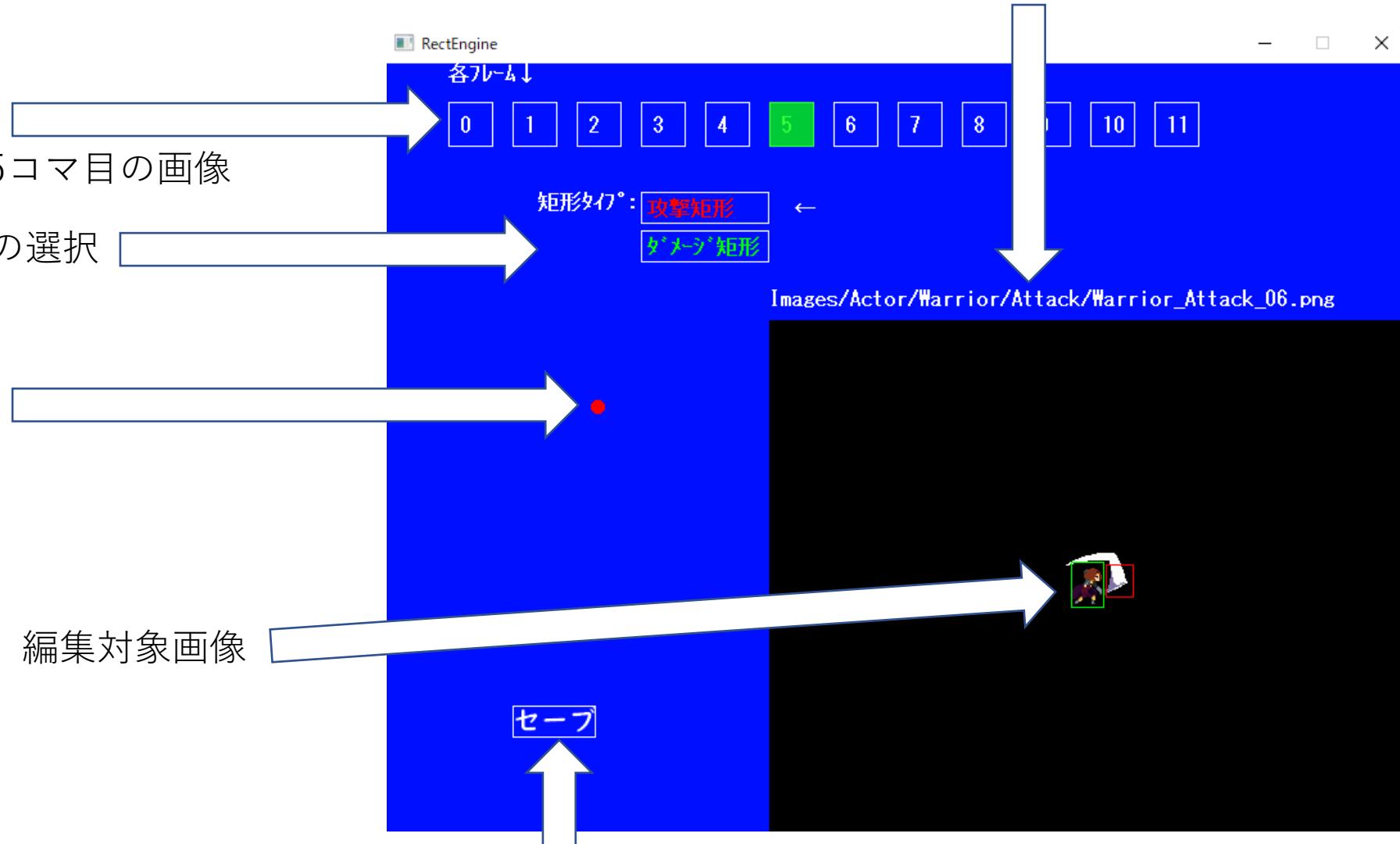
左クリックでBoxの始点

右クリックでBoxの終点

Boxタイプで色が変化

攻撃：赤(0xff0000)

ダメージ：緑(0x00ff00)



そのフレーム画像内のBoxデータをセーブするためのボタン

# ～担当箇所概要説明Ⅲ～

データセーブボタンを押した



FileManagerクラスのWriteRectDataFile関数で  
引数で設けた文字列でデータファイルを作成し  
そのファイル内にデータ書き込み

## ②当たり判定Box作成ツール概要

```
177 void FileManager::WriteRectDataFile(const std::string& fileName, const ColliderBoxManager& boxManager)
178 {
179     std::string dataFileName = "DataFiles/Details/" + fileName + ".dat";
180     std::ofstream file(dataFileName);
181
182     // int型からstring型に変換する方法
183     auto GetStr = [&](const int& val) {
184         return std::to_string(val);
185     };
186
187     // ColliderBoxEditで格納していった変数を利用して
188     // 矩形データを取得
189     auto colliderBoxes = boxManager.GetColliderBoxList();
190     // Manager内に入っている矩形全部で回す
191     for (int num = 0; num < colliderBoxes.size(); num++)
192     {
193         // 最初に見出し
194         file << "<RectNum=";
195         // その間に矩形が何個存在するかを書き込む
196         file << GetStr(colliderBoxes[num].size());
197         // 記号書き込み
198         file << ">";
199         // フラグの中で存在する矩形の数で回す
200         for (int i = 0; i < colliderBoxes[num].size(); i++)
201         {
202             // 1矩形データ
203             auto box = colliderBoxes[num][i];
204             // {O,O,O}みたいな感じで矩形の位置情報を書き込む
205             file << "(" + GetStr(box->GetPos().x) + "," + GetStr(box->GetPos().y) + ")";
206             // カッコ
207             file << ",";
208             // {O,O,O}みたいな感じで矩形の射程を書き込む
209             file << "(" + GetStr(box->GetSize().x) + "," + GetStr(box->GetSize().y) + ")";
210             // カッコ
211             file << ",";
212             // 矩形射程の書き込み-----
213             auto boolStr = "";
214             if (box->GetTypeFlag())
215             {
216                 boolStr = "true";
217             }
218             else
219             {
220                 boolStr = "false";
221             }
222             file << boolStr;
223             // -----
224             // 最後の行で無ければカッコを付ける
225             if (num < colliderBoxes.size() - 1)
226             {
227                 file << ",";
228             }
229             // 改行
230             file << std::endl;
231         }
232         // 最後に付け加え
233         file << "/RectData>";
234         // ファイルを閉じる
235         file.close();
236     }
}
```

D:\RectEngin\DataFiles\Details\Warrior\Rect\Attack.dat(更新) - sakura 2.4.1.2849  
ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)  

```
1 <RectNum=1>,{556,408},{49,47},true,  
2 <RectNum=1>,{548,408},{55,47},false,  
3 <RectNum=1>,{552,408},{37,54},false,  
4 <RectNum=1>,{555,395},{96,77},false,  
5 <RectNum=2>,{547,405},{30,42},false,{569,387},{43,41},true,  
6 <RectNum=1>,{558,417},{54,39},false,  
7 <RectNum=1>,{564,412},{52,51},false,  
8 <RectNum=1>,{537,393},{131,79},false,  
9 <RectNum=1>,{538,376},{63,82},false,  
10 <RectNum=1>,{530,381},{59,85},false,  
11 <RectNum=1>,{550,418},{101,58},false,  
12 <RectNum=1>,{540,405},{70,58},false  
13 </RectData>[EOF]
```

結果として



# ～担当箇所説明(コード)～

## FileManager.h

```
5  #define IpFileManager FileManager::getInstance()
6
7  class ColliderBoxManager;
8  // ファイル書き込みや読み込み、オブジェクト操作関係をまとめたクラス
9  // シングルトンクラス
10 class FileManager
11 {
12 public:
13     static FileManager& getInstance(void)
14     {
15         return *instance_;
16     }
17
18     static void Create(void);
19     void Destroy(void);
20     // 指定ディレクトリよりそのディレクトリ内に存在するファイルを検索して、検索結果を格納する
21     void SearchFileNameFromDirectory(const std::string& directoryName);
22     // 指定名でファイルを検索して指定名のファイルが有ればtrueを返す
23     bool IsExistFile(const std::string& dataName,const std::string& objName,const std::string& animName);
24     // テクスチャの読み込み
25     void ReadDataFile(const std::string& pathName);
26
27     // データのファイル書き込み
28     void WriteImageDataFile(const std::string& fileName,const std::vector<std::string>& strVec);
29     // 矩形データのファイル書き込み
30     void WriteRectDataFile(const std::string& fileName, const ColliderBoxManager& boxManager);
31     // ファイル名のリストの中身のクリア
32     void ClearFileNameList(void);
33
34     // 検索した結果のファイル名のリストの取得
35     const std::vector<std::string>& GetFileNameList(void) const
36     {
37         return fileNameList_;
38     }
39     // アニメーション名のリストの取得
40     const std::vector<std::string>& GetAnimationNameList(void) const
41     {
42         return animationNameList_;
43     }
44 private:
45     FileManager() = default;
46     ~FileManager();
47     // コピーコンストラクタ禁止
48     void operator=(const FileManager&) = delete;
49     static FileManager* instance_;
50     // ディレクトリ内部に存在するファイル名達
51     std::vector<std::string> fileNameList_;
52     // オブジェクト名に対するアニメーション名のリスト
53     std::vector<std::string> animationNameList_;
54 };
55 }
```

# ～担当箇所説明(コード)～

## FileManager.cpp

### Create関数(生成)、Destroy関数(破棄)

```
10     FileManager* FileManager::instance_ = nullptr;
11
12     void FileManager::Create(void)
13     {
14         if (!instance_)
15         {
16             // 生成
17             instance_ = new FileManager();
18         }
19     }
20
21     void FileManager::Destroy(void)
22     {
23         if (instance_)
24         {
25             // 破棄
26             delete instance_;
27         }
28         instance_ = nullptr;
29     }
```

### SearchFileFromDirectory関数

```
31     void FileManager::SearchFileFromDirectory(const std::string& directoryName)
32     {
33         if (fileNameList_.size() <= 0)
34         {
35             HANDLE hFind;
36             WIN32_FIND_DATA win32fd;
37             // 検索するルートに到達するまでのパス
38             std::string path = directoryName + "/";
39             // 対象にするファイル名拡張子
40             std::string extension = ".png";
41             // 検索にかける名前
42             std::string search_name = path + "*." + extension;
43
44             hFind = FindFirstFile(search_name.c_str(), &win32fd);
45             // ファイルがなければ
46             if (hFind == INVALID_HANDLE_VALUE) {
47                 throw std::runtime_error("file not found");
48                 return;
49             }
50
51             // 指定のディレクトリ以下のファイル名をファイルがなくなるまで取得する
52             do {
53                 if (win32fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
54                     // ディレクトリの場合は何もしない
55                     continue;
56                 }
57                 else {
58                     fileNameList_.push_back(win32fd.cFileName);
59                 }
60             } while (FindNextFile(hFind, &win32fd));
61             // ファイルを閉じる
62             FindClose(hFind);
63         }
64     }
```

# ～担当箇所説明(コード)～

## FileManager.cpp

### IsExistFile関数

```
67     bool FileManager::IsExistFile(const std::string& dataName,const std::string& objName,const std::string& animName)
68     {
69         // 検索をかけるファイル名
70         auto fileName = "DataFiles/Details/" + objName + "/" + dataName + "/" + animName;
71         HANDLE hFind;
72         WIN32_FIND_DATA win32fd;
73         // 検索にかける名前
74         std::string search_name = fileName + ".dat";
75
76         hFind = FindFirstFile(search_name.c_str(), &win32fd);
77
78         // ファイルが無ければfalseを返す
79         if (hFind == INVALID_HANDLE_VALUE) {
80             FindClose(hFind);
81             return false;
82         }
83         else
84         {
85             // ファイルが有ればtrueを返す
86             FindClose(hFind);
87             return true;
88         }
89     }
90 }
```

### ReadDataFile関数

```
92     void FileManager::ReadDataFile(const std::string& pathName)
93     {
94         // 読み込むデータファイル名
95         std::string fileName = pathName + ".dat";
96         // テキストファイル
97         std::ifstream file(fileName, std::ios::in | std::ios::binary);
98         std::stringstream stringStream;
99         std::string str1;
100
101        do
102        {
103            // カタ区切りで一行読む
104            std::getline(file, str1, ',');
105            // アニメーション名のリストに格納していく
106            animationNameList_.push_back(str1);
107        } while (!file.eof());
108
109    }
110 }
```

# ～担当箇所説明(コード)～

## FileManager.cpp

### WriteImageDataFile関数

```
112 void FileManager::WriteImageDataFile(const std::string& fileName, const std::vector<std::string>& strVec)
113 {
114     std::string dataFileName = "DataFiles/Details/" + fileName + ".dat";
115     // 書き込むデータファイル
116     std::ofstream file(dataFileName);
117     std::stringstream stringStream;
118     std::string str1;
119     int i = 0;
120     for (; i < 3; i++)
121     {
122         // 最初だけ見出しを付ける
123         if (i < 0)
124         {
125             file << "<Data=";
126         }
127         // strVecに格納された文字列を書き込んでいく
128         file << strVec[i];
129
130         // 最後になつたら改行
131         if (i >= 2)
132         {
133             file << std::endl;
134             file << "/Data>";
135         }
136         else
137         {
138             // コマで区切る
139             file << ",";
140         }
141     }
```

続き

```
142     for (; i < strVec.size(); i++)
143     {
144         // 最初だけ見出しを付ける
145         if (i <= 3)
146         {
147             // 改行
148             file << std::endl;
149             // 見出し
150             file << "<Image=";
151         }
152         // 最後の行じゃなければ
153         if (i < strVec.size() - 1)
154         {
155             // 文字列書き込み
156             file << strVec[i];
157             // カマ書き込み
158             file << ",";
159             // 改行
160             file << std::endl;
161         }
162         // 最後の行だとカマは不要なので
163         else
164         {
165             // 文字列書き込み
166             file << strVec[i];
167             // 改行
168             file << std::endl;
169         }
170     }
171     // 最後に付け加える
172     file << "/Image>";
173     // ファイルを閉じる
174     file.close();
175 }
```

# ～担当箇所説明(コード)～

FileManager.cpp

WriteRectDataFile関数

```
177 void FileManager::WriteRectDataFile(const std::string& fileName, const ColliderBoxManager& boxManager)
178 {
179     std::string dataFileName = "DataFiles/Details/" + fileName + ".dat";
180     std::ofstream file(dataFileName);
181
182     // int型からstring型に変換するうげ
183     auto GetStr = [&](const int& val) {
184         return std::to_string(val);
185     };

```

データ編集終了後  
もう一回(違うファイルの)  
ClearFileNameList関数 → 編集をしたい時に  
バッファに溜まったものを  
削除

```
238 void FileManager::ClearFileNameList(void)
239 {
240     // ファイル名リストの中身を空にする
241     fileNameList_.clear();
242     // アニメーション名リストの中身を空にする
243     animationNameList_.clear();
244 }
245
246 FileManager::FileManager()
247 {
248 }
```

続き

```
186 // ColliderBoxEditで格納していった変数を利用して
187 // 矩形データを取得
188 auto colliderBoxes = boxManager.GetColliderBoxList();
189 // Manager内に入っている矩形全部で回す
190 for (int num = 0; num < colliderBoxes.size(); num++)
191 {
192     // 最初に見出し
193     file << "<RectNum=";
194     // そのコマに矩形が何個存在するかを書き込む
195     file << GetStr(colliderBoxes[num].size());
196     // 記号書き込み
197     file << ">";
198     // コマの中で存在する矩形の数で回す
199     for (int i = 0; i < colliderBoxes[num].size(); i++)
200     {
201         // 1矩形データ
202         auto box = colliderBoxes[num][i];
203         // {〇〇,〇〇}みたいな感じで矩形の座標情報を書き込む
204         file << "[" + GetStr(box->GetPos().x) + "," + GetStr(box->GetPos().y) + "]";
205         // コマ
206         file << ",";
207         // {〇〇,〇〇}みたいな感じで矩形のサイズを書き込む
208         file << "[" + GetStr(box->GetSize().x) + "," + GetStr(box->GetSize().y) + "]";
209         // コマ
210         file << ",";
211         // 矩形属性の書き込み-----
212         auto boolStr = "";
213         if (box->GetTypeFlag())
214         {
215             boolStr = "true";
216         }
217         else
218         {
219             boolStr = "false";
220         }
221         file << boolStr;
222         // -----
223     }
224     // 最後の行で無ければコマを付ける
225     if (num < colliderBoxes.size() - 1)
226     {
227         file << ",";
228     }
229     // 改行
230     file << std::endl;
231 }
232 // 最後に付け加え
233 file << "/RectData";
234 // ファイルを閉じる
235 file.close();
236 }
```

### ③ 敵AI作成

## ～担当箇所概要①～ 敵の生成

Actor  
  Actor.h  
  Enemy.h  
  Fireball.h  
  Player.h

Actor  
  Actor.cpp  
  Enemy.cpp  
  Fireball.cpp  
  Player.cpp

ObjectPooling法を用いた  
EnemyPoolクラスの生成

```
30 void EnemyPool::Create(int size, std::shared_ptr<Player>& player)
31 {
32     // sizeに応じてpoolの大きさが決まる
33     for (int s = 0; s < size; s++)
34     {
35         // まずとにかく敵の箱を用意
36         poolMembers_.push_back(std::make_shared<Enemy>(10,player));
37     }
38     // poolのサイズ
39     poolSize_ = size;
40 }
```

バッファのサイズを指定し  
バッファ領域を確保し、バッファ内に  
空の敵を領域分生成

バッファの中から敵を使い回す

```
42 std::shared_ptr<Enemy>& EnemyPool::Pickup(int spawnCnt)
43 [
44     for (int s = 0; s < spawnCnt; s++)
45     {
46         // poolのサイズで回す
47         for (int i = 0; i < poolSize_; i++)
48         {
49             // 死んでいるやつがいればpoolから拾い上げる
50             if (!poolMembers_[i]->GetAlive())
51             {
52                 // 生き返らせる
53                 poolMembers_[i]->SetAlive(true);
54                 // HPの初期化
55                 poolMembers_[i]->SetHP(10);
56                 // ビハビアーデータのセット
57                 poolMembers_[i]->SetBehavior(&behaviors_[static_cast<int>(ENEMYTYPE::Cultist)]);
58                 // 敵のセット
59                 poolMembers_[i]->SetEnemy(ENEMYTYPE::Cultist, Vector3F(300.0f, 100.0f, 100.0f));
60                 // 未使用なので再利用可能
61                 return poolMembers_[i];
62             }
63         }
64     }
65 ]
```

敵が死んだらバッファに戻す

バッファの中の敵が死んでいたらその敵に  
敵タイプを付与し、新しい敵にして左記の  
PickUp関数で拾い上げる

新たにインスタンスをすることなく敵が生成可能

### ③ 敵AI作成

## ～担当箇所概要①～ 敵の生成

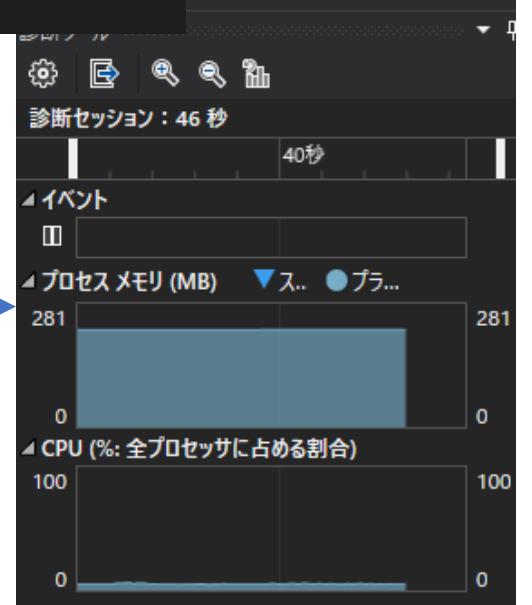
実際に敵バッファを1000体格納可能にして  
敵をリスポーンし続けると……

```
113 // 敵バッファの生成
114 enemyPool_ = std::make_unique<EnemyPool>();
115 // 敵タイプに対応するビヘイビアリテータの生成
116 enemyPool_->BehaviorRegistrar(ENEMYTYPE::Cultist);
117 // 敵1体(引数の"1"を変更すれば何体でも生成可能)の生成
118 enemyPool_->Create(1000, player_);
119 // 生成した敵がバッファの中で泳いでいるので、バッファから拾い上げる
120 // (これも引数の数字を変更することで拾いあげる敵数変更可能)
121 // ただし、拾いあげる数だけ生成されていないとアサートを出す
122 enemyPool_->Pickup(1);
```

281MBからは増えないようになった！！

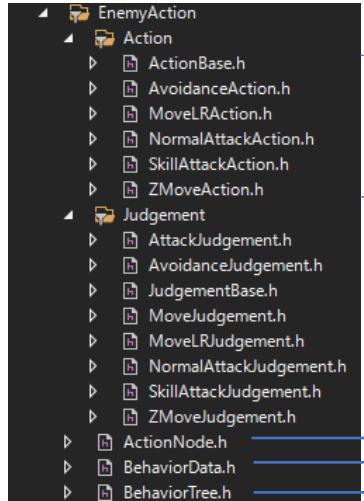


結果として処理速度の向上につながった！！



# ③ 敵AI作成

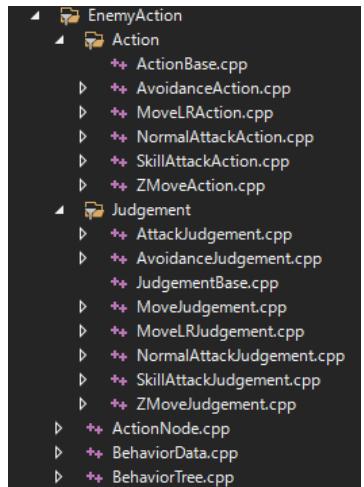
## ～担当箇所概要②～ BehaviorBaseAI



行動実行クラス

行動判定クラス

ビヘイビアツリーに追加していく思考ノード  
ビヘイビアツリーの内部データの操作関係  
ビヘイビアツリー(思考ノードの木構造クラス)



ビヘイビアツリー内の  
Inference関数でノード推論  
(行動判定でtrueになった  
行動実行を返す)

ビヘイビアツリーに思考ノード追加

```

15 void BehaviorTree::AddNode(std::string searchName, std::string entryName, int priority, SelectRule selectRule, JudgementBase* judgement, ActionBase* action)
16 {
17     // searchNameの文字列が空だと
18     if (searchName.size() != 0)
19     {
19         // まず親ノードを探す
20         ActionNode* searchNode = root_>>SearchActionNode(searchName);
21         // 親がいるれば
22         if (searchNode != nullptr)
23         {
24             // 末尾の子供を探す
25             ActionNode* sibling = searchNode->GetLastChild();
26             // 追加するノードの作成
27             ActionNode* addNode = new ActionNode(entryName, searchNode, sibling, priority, selectRule, judgement, action);
28             // ノードの追加
29             searchNode->AddChild(addNode);
30         }
31     }
32     else
33     {
34         // そもそもルートがなければ
35         if (root_ == nullptr)
36         {
37             // ルートの作成
38             // childとsiblingは共にnullptr(自信が親なので)
39             root_ = new ActionNode(entryName, nullptr, nullptr, priority, selectRule, judgement, action);
40         }
41     }
42     else
43     {
44         // アサートをかける
45         ASSERT("root is already registered!!");
46     }
47 }
48 
```



```

16 ActionNode* ActionNode::Inference(Entity* enemy, BehaviorData* data)
17 {
18     // 実行ノード格納用
19     ActionNode* result = nullptr;
20
21     for (int i = 0; i < child_.size(); i++)
22     {
23         if (child_[i]->judgement_ != nullptr)
24         {
25             // 判定がtrueになったら実行ノード格納用変数に格納
26             if (child_[i]->judgement_(enemy))
27             {
28                 result = child_[i];
29             }
30         }
31     }
32
33     if (result != nullptr)
34     {
35         // 実行データをもっていれば終了
36         if (result->HasAction())
37         {
38             return result;
39         }
40     }
41
42     // もっていなければ推論続行
43     result = result->Inference(enemy, data);
44
45
46
47
48 } 
```

## ～担当箇所概要②～ BehaviorBaseAI

## ③ 敵AI作成

Inference関数で推論した結果

```
96     bool ActionNode::HasAction(void)
97     {
98         // 実行ノードをもっていればtrue
99         return action_ != nullptr?true:false;
100    }
```

True(行動実行クラスが存在する)

False(行動実行クラスが存在しない)

行動実行

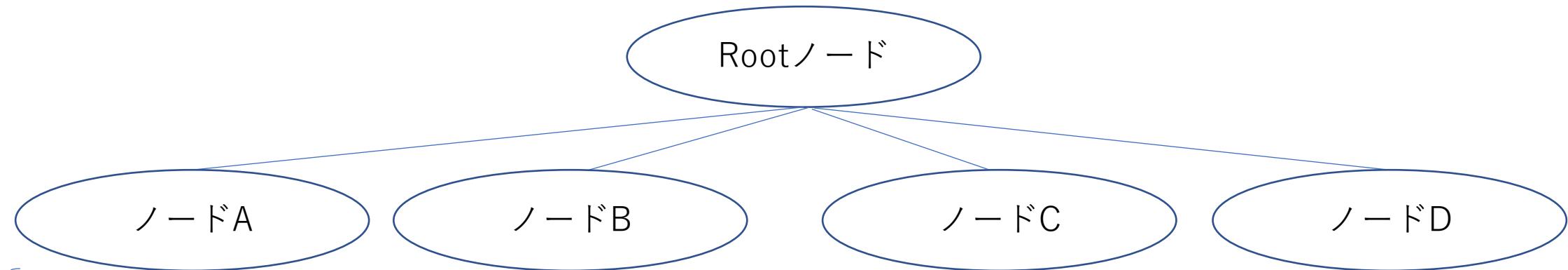
推論で返ったノードで  
推論を再試行

```
102 void ActionNode::Run(Enemy* enemy)
103 {
104     // 行動の実行
105     if (action_ != nullptr)
106     {
107         action_->operator()(enemy);
108     }
109 }
```

## ～担当箇所概要②～

## BehaviorBaseAI

## ③ 敵AI作成



行動実行クラス(ActionBase型)

実行判定クラス(JudgementBase型)

思考選択ルール(SelectRule型)

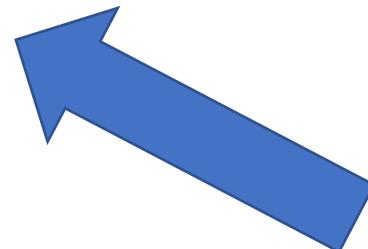
優先度(int型)

親ノード(ActionNode型)

兄弟ノード(ActionNode型)

ノード名(string型)

今現在勉強中なので今回はSequenceのみでやっている。  
Sequenceというのは、行動が失敗や順番に実行出来なかった場合にRootノードに戻る仕組み。



各ノードこれらを持っていてRootノードから実行判定を行いつつビヘイビアツリー内の思考を続ける。

# ～担当箇所説明～

## Enemy.h

```
6   // Z軸方向のプラスマイナスどちらかに移動するかのタイプ
7   enum class ZPosFromPlayer
8   {
9     // プレイヤーのZ軸よりもマイナスにある場合
10    Minus,
11    // プレイヤーのZ軸よりもプラスにある場合
12    Plus,
13    // プレイヤーのZ軸との差が一定以下の場合
14    Non,
15    Max
16  };
17
18  // 動きのタイプ
19  enum class MoveType
20  {
21    // 左右横に動く
22    MoveLR,
23    // 奥行に動く
24    MoveZ,
25    // 動かない
26    Non,
27    Max
28  };
29
30  // 自分の視界
31  struct VisionRange
32  {
33    // runアニメーションを行う視界
34    float runRange_;
35    // attackアニメーションを行う視界
36    float attackRange_;
37
38    VisionRange() :runRange_(0.0f), attackRange_(0.0f) {};
39    VisionRange(float runR, float atR) :runRange_(runR), attackRange_(atR) {};
40  };
41
42  // 各アニメーションの実行関数ポインタ
43  typedef void(Enemy::*AnimExecutor)();
```

```
49  class Enemy :
50  |  public Actor
51  {
52  |  public:
53  |    Enemy(int hp, std::shared_ptr<Player>& player);
54  |    ~Enemy();
55  |    // 更新
56  |    void Action(void);
57  |    // 左右の動き
58  |    void Move(void);
59  |    // Z軸を合わせるアクション
60  |    void ZMove(void);
61  |    // 攻撃(Actorからの継承)
62  |    void Attack(void);
63  |    // Attack関数に含めるためのスキル攻撃関数(引数の敵のタイプ毎に分岐)
64  |    void SkillAttack(ENEMYTYPE type);
65  |    // 回避行動
66  |    void Avoidance(void);
67  |    // 初期化
68  |    virtual bool Initialize(void) { return false; };
69  |    // poolの中から死んでいるやつをpickUpするのでpickUpしたやつにタイプを与えてやったり
70  |    // ポジションをセットしてやる関数
71  |    void SetEnemy(ENEMYTYPE type, Vector3F pos);
72  |    // ヒュエイビアリーのセット
73  |    void SetBehavior(BehaviorTree* behaviorTree);
74  |    // 敵タイプのセット関数
75  |    void Set(ENEMYTYPE type);
76  |    // 視界の取得
77  |    const VisionRange& GetVisionRange(void)
78  |    [
79  |      return visionRange_;
80  |    ]
81  |    // あるポジションとの距離を計算して取得
82  |    const Vector3F& GetDistance(void);
83  |    // テレルック表示
84  |    void DebugDraw(void);
```

# ～担当箇所説明～

## Enemy.hの続き

```
85     // Z軸方向のマスク方向に動かすのかY軸方向に動かすのかのセット関数
86     void SetZPosStatus(ZPosFromPlayer zStatus);
87     // アニメーション実行関数に実行関数のセット
88     void SetAnimExcecuter(void(Enemy::* func)());
89
90     // ビューハイドアリーラの取得関数
91     BehaviorTree* GetBehaviorTree(void)
92     {
93         return behaviorTree_;
94     }
95
96     // 動きのタイプの取得
97     const MoveType& GetMoveType(void)
98     {
99         return moveType_;
100    }
101
102    // 動きのタイプのセット
103    void SetMoveType(MoveType moveType);
104
105    // 自分が見ているプレイヤーの取得
106    const std::shared_ptr<Player>& GetPlayer(void)
107    {
108        return player_;
109    }
110
111    // 行動実行関数のインスタンス取得関数
112    const AnimExcecuter& GetAnimExcecuter(void)
113    {
114        return animExcecuter_;
115    }
116
117    // スキルを持っているればtrueを返す
118    const bool& HasSkill(void)
119    {
120        return hasSkill_;
121    }
```

```
122     protected:
123         // 行動の実行関数。インタフェース
124         AnimExcecuter animExcecuter_;
125         // 自分のタイプ変数
126         ENEMYTYPE myType_;
127         // 自信のビューハイドアリーラ(思考)
128         BehaviorTree* behaviorTree_;
129         // ビューハイドアリーラ(sequenc等のデータ)
130         BehaviorData* behaviorData_;
131         // 実行ノード
132         ActionNode* runningNode_;
133         // 視界
134         VisionRange visionRange_;
135         // プレイヤーの参照
136         std::shared_ptr<Player>& player_;
137         // プレイヤーに対する自分のZ軸の状態
138         ZPosFromPlayer zPosStatus_;
139         // 動きのタイプ変数
140         MoveType moveType_;
141         // 当たり判定を受け付けないフラグ
142         bool noAcceptHitFlag_;
143         // スキルを持っているかのフラグ
144         bool hasSkill_;
145     };
```

# ～担当箇所説明～

## Enemy.cpp

### コンストラクタ、デストラクタ

```
18  Enemy::Enemy(int hp, std::shared_ptr<Player>& player):
19  {
20      Actor(hp),player_(player)
21      // 生きているフラグの初期化
22      isAlive_ = false;
23      // 当たり判定フラグの初期化
24      isHit_ = false;
25      // 当たり判定を受け付けないフラグの初期化
26      noAcceptHitFlag_ = false;
27      // 現在思考しているビヘイビアツリーのノード
28      runningNode_ = nullptr;
29      // ビヘイビオデータの作成
30      behaviorData_ = new BehaviorData;
31  }
32
33  Enemy::~Enemy()
34  {
35  }
```

### Action関数

```
37  void Enemy::Action(void)
38  {
39      // 床の上にいる時
40      if (onFloor_)
41      {
42          // 当たり判定は何もない時
43          if (!isHit_)
44          {
45              // 現在の行動にビヘイビアツリーのrootから推論した結果のノードを格納
46              runningNode_ = behaviorTree_->Inference(this,behaviorData_);
47              // 格納する行動が有れば
48              if (runningNode_ != nullptr)
49              {
50                  // ビヘイビアツリーから、runningNode_に対応したノードを走らせる
51                  behaviorTree_->Run(this, runningNode_, behaviorData_);
52              }
53          }
54          else
55          {
56              // 何も行動をしないときはIdleを設定
57              SetAnimExcecuter(&Enemy::Idle);
58          }
59          // 行動に応じたアニメーションと動きを関数ポインタで遷移させる
60          (this->*animExcecuter_)();
61      }
62      else
63      {
64          // 何かに当たった時はこっちでHitやDeathが回る
65          (this->*stateExcecuter_)();
66      }
67  }
```

# ～担当箇所説明～

Enemy.cpp

Action関数の続き

```
68     // 攻撃の当たり判定-----
69     // player -> enemy
70     if (!isHit_ && animExeceuter_ != &Enemy::Avoidance)
71     {
72         if (CollisionCheck()(colData_[1].first, player_->GetColliderBox(0).first))
73         {
74             isHit_ = true;
75         }
76     }
77
78     // enemy -> player
79     if (!player_->GetIsHit())
80     {
81         if (CollisionCheck()(colData_[0].first, player_->GetColliderBox(1).first))
82         {
83             player_->SetIsHit(true);
84         }
85     }
86
87     //-----
88     // アニメーションの更新
89     UpdateAnimation(currentAction_);
90 }
91 }
```

Move関数

```
93 void Enemy::Move(void)
94 {
95     // 走るアニメーションに変更
96     ChangeAction("Run");
97     // 左向きであれば
98     if (direction_ == Direction::Left)
99     {
100        speed_ = { -1.0f, 0.0f, 0.0f };
101    }
102    // 右向きであれば
103    else
104    {
105        speed_ = { 1.0f, 0.0f, 0.0f };
106    }
107    // 移動処理
108    pos_ += speed_;
109 }
```

# ～担当箇所説明～

Enemy.cpp

## ZMove関数

```
111 void Enemy::ZMove(void)
112 {
113     // 走るアニメーションに変更
114     ChangeAction("Run");
115     // 自分のZ軸ポジションの状態(位置状態)
116     switch (zPosStatus_)
117     {
118         // プレイヤーよりも手前にいる場合
119         case ZPosFromPlayer::Minus:
120             speed_ = { 0.0f, 0.0f, -0.5f };
121             break;
122         // プレイヤーよりも奥にいる場合
123         case ZPosFromPlayer::Plus:
124             speed_ = { 0.0f, 0.0f, 0.5f };
125
126             break;
127         // 一致している場合
128         case ZPosFromPlayer::Non:
129             speed_ = { 0.0f, 0.0f, 0.0f };
130
131             break;
132         default:
133             break;
134     }
135     // Z軸ポジションの移動処理
136     pos_.z += speed_.z;
137 }
```

## Attack関数

```
139 void Enemy::Attack(void)
140 {
141     // 通常の攻撃の場合
142     if (attackFlag_)
143     {
144         // 通常攻撃のアニメーションに切り替え
145         ChangeAction("Attack");
146         // アニメーションの終了
147         if (isAnimEnd_)
148         {
149             // 攻撃フラグをfalse
150             attackFlag_ = false;
151             // 回避フラグをfalse
152             avoidFromAttackFlag_ = false;
153             // アニメーションをidleに設定(default)
154             currentAction_ = "Idle";
155         }
156     }
157     // スキル攻撃の場合
158     if (skillAttackFlag_)
159     {
160         // スキル攻撃のアニメーションに設定
161         ChangeAction("SkillAttack");
162
163         // タイプ毎のスキル攻撃
164         SkillAttack(myType_);
165         // アニメーション終了時
166         if (isAnimEnd_)
167         {
168             // スキル攻撃フラグをfalse
169             skillAttackFlag_ = false;
170             // 回避フラグをfalse
171             avoidFromAttackFlag_ = false;
172             // アニメーションをidleに(default)
173             currentAction_ = "Idle";
174         }
175     }
176 }
```

## SkillAttack関数

```
178 void Enemy::SkillAttack(ENEMYTYPE type)
179 {
180     // 敵タイプで分岐
181     switch (type)
182     {
183         case ENEMYTYPE::CultistAssassin:
184             // スキル攻撃をする時のアニメーション間隔
185             if (animCnt_ >= 8.0f && animCnt_ <= 8.5f)
186             {
187                 // 攻撃をくらわないフラグをtrueに
188                 noAcceptHitFlag_ = true;
189                 // プレイヤーの後ろのポジションの取得
190                 // プレイヤーの後ろに回って攻撃させてるので
191                 pos_ = player_->GetBehindPos();
192             }
193             break;
194         case ENEMYTYPE::Cultist:
195             break;
196         case ENEMYTYPE::TwistedCultist:
197             break;
198         case ENEMYTYPE::Max:
199             break;
200         default:
201             break;
202     }
203 }
```

# ～担当箇所説明～

Enemy.cpp

## Avoidance関数

```
205 void Enemy::Avoidance(void)
206 {
207     // 攻撃をかわすアニメーションに変更
208     ChangeAction("Avoidance");
209     // 回避行動を始めた際、skill、normalの攻撃flagは折る
210     attackFlag_ = false;
211     skillAttackFlag_ = false;
212     // 回避行動で後ろに下がるタイミング >> アニメーションと同期させたいのでマッカンパーにしている
213     if (animOnt_ >= 1.0f && animOnt_ <= 1.5f)
214     {
215         // 後ろに下がる予定のポジション
216         auto backPos = pos_;
217         // 自分が現在向いている方向
218         switch (direction_)
219         {
220             // 右向きの場合
221             case Direction::Right:
222                 backPos.x = backPos.x - 10.0f;
223                 break;
224             // 左向きの場合
225             case Direction::Left:
226                 backPos.x = backPos.x + 10.0f;
227                 break;
228             case Direction::Max:
229                 break;
230             default:
231                 break;
232         }
233         // ポジションの変更
234         SetPosition(backPos);
235     }
236     // アニメーション終了時
237     if (isAnimEnd_)
238     {
239         // 回避行動の終了
240         avoidFromAttackFlag_ = false;
241         // 回避行動が終わればidleに戻す
242         animExecutor_ = &Enemy::Idle;
243     }
244 }
```

## SetEnemy関数

```
246 void Enemy::SetEnemy(ENEMYTYPE type, Vector3F pos)
247 {
248     // 敵タイプセット
249     myType_ = type;
250     // ポジションセット
251     pos_ = pos;
252     // 敵の生成
253     Set(type);
254 }
255
256 void Enemy::SetBehavior(BehaviorTree* behaviorTree)
257 {
258     // 敵タイプに沿ったビヘイビアツリーのセット
259     behaviorTree_ = behaviorTree;
260 }
```

## SetBehavior関数

# ～担当箇所説明～

Enemy.cpp

## GetDistance関数

```
296 const Vector3F& Enemy::GetDistance(void)
297 {
298     // プレイヤーのポジション
299     auto plPos = player_>GetPosition();
300     // プレイヤーとの距離を測っている
301     auto distance = plPos - pos_;
302
303     // 同時に方向も変更
304     if (distance.x < 0.0f)
305     {
306         direction_ = Direction::Left;
307     }
308     else
309     {
310         direction_ = Direction::Right;
311     }
312
313     return distance;
314 }
```

## DebugDraw関数

## SetAnimExecuter関数

## SetPosZStatus関数

## SetMoveType関数

```
316     // デバッグを行う際に使用する
317     void Enemy::DebugDraw(void)
318     {
319         DrawFormatString(0, 50, 0xffffffff, "enPos:(%f,%f,%f)", pos_.x, pos_.y, pos_.z);
320         DrawFormatString(0, 100, 0xff0000, "enHP:%d", hp_);
321     }
```

```
323     // Zポジションの状態のセット
324     void Enemy::SetZPosStatus(ZPosFromPlayer zStatus)
325     {
326         zPosStatus_ = zStatus;
327     }
```

```
329     void Enemy::SetAnimExecuter(void(Enemy::* func)())
330     {
331         // アニメーション状態管理関数。イタのセット
332         animExecuter_ = func;
333     }
```

```
335     void Enemy::SetMoveType(MoveType moveType)
336     {
337         // 移動タイプのセット
338         // 奥行移動なのか横移動なのか
339         moveType_ = moveType;
340     }
```

# ～担当箇所説明～

JudgementBase.h

```
3  class Enemy;
4  // 判定クラスの親クラス
5  struct JudgementBase
6  {
7      public:
8
9      bool operator () (Enemy* enemy)
10     {
11         return Judgement(enemy);
12     }
13     private:
14     virtual bool Judgement(Enemy* enemy) = 0;
15
16 };
```

# ～担当箇所説明～

## AttackJudgement.h

```
4  class Enemy:
5
6      // 攻撃行動の判定クラス
7      struct AttackJudgement : public JudgementBase
8      {
9          public:
10
11          static AttackJudgement& getInstance(void)
12          {
13              static AttackJudgement sInstance_;
14              return sInstance_;
15          }
16
17          bool Judgement(Enemy* enemy);
18      };
19  };
```

## AttackJudgement.cpp

```
4  bool AttackJudgement::Judgement(Enemy* enemy)
5  {
6      // 攻撃していれば移動はさせない
7      if (enemy->GetAttackFlag() || enemy->GetSkillAttackFlag())
8      {
9          return true;
10     }
11
12     // プレイヤーとの距離を測る
13     auto distance = enemy->GetDistance();
14
15     // 通常攻撃
16     if ((abs(distance.x) <= enemy->GetVisionRange().attackRange_) && (abs(distance.z) <= 5.0f))
17     {
18         // 通常攻撃をtrueにする
19         enemy->SetAttackFlag(true);
20         // skillAttackをfalseにする
21         enemy->SetSkillAttackFlag(false);
22         return true;
23     }
24
25     // skillAttack攻撃
26     // スキルを持っていれば
27     if (enemy->HasSkill())
28     {
29         if ((abs(distance.x) <= enemy->GetVisionRange().attackRange_ + 50.0f) && (abs(distance.z) <= 5.0f))
30         {
31             // skillAttackをtrueに
32             enemy->SetSkillAttackFlag(true);
33             // 通常の攻撃をfalseに
34             enemy->SetAttackFlag(false);
35         }
36     }
37
38     // 何にも当てはまらなければどちらもfalseにする
39     enemy->SetAttackFlag(false);
40     enemy->SetSkillAttackFlag(false);
41
42 }
43 }
```

# ～担当箇所説明～

NormalAttackJudgement.h

```
4  class Enemy;
5
6  // 通常攻撃の判定クラス
7  struct NormalAttackJudgement :
8      public JudgementBase
9  {
10
11     public:
12
13     static NormalAttackJudgement& getInstance(void)
14     {
15         static NormalAttackJudgement sInstance_;
16         return sInstance_;
17     }
18
19 };
```

NormalAttackJudgement.cpp

```
4     bool NormalAttackJudgement::Judgement(Enemy* enemy)
5     {
6         // 通常攻撃のフラグがtrueの場合trueを返し、実行ノードに渡す
7         if (enemy->GetAttackFlag())
8         {
9             return true;
10        }
11    }
12 }
```

# ～担当箇所説明～

## AvoidanceJudgement.h

```
4 class Enemy;
5
6 // 攻撃回避行動の判定クラス
7 struct AvoidanceJudgement : public JudgementBase
8 {
9     static AvoidanceJudgement& getInstance(void)
10    {
11        static AvoidanceJudgement sInstance_;
12        return sInstance_;
13    }
14
15    bool Judgement(Enemy* enemy);
16 };
17 
```

## AvoidanceJudgement.cpp

```
5     bool AvoidanceJudgement::Judgement(Enemy* enemy)
6     {
7         // 既にavoidance行動実行中ならば
8         if (enemy->GetAnimExececutor() == &Enemy::Avoidance)
9         {
10             return true;
11         }
12         // プレイヤーとの距離を測る
13         auto distance = enemy->GetDistance();
14         auto pl = enemy->GetPlayer();
15         // プレイヤーが真正面にいるかどうかのフラグ
16         bool plIsNear = false;
17
18         if ((abs(distance.x) <= 50.0f) &&
19             (abs(distance.y) <= 0.0f) &&
20             (abs(distance.z) <= 5.0f))
21         {
22             plIsNear = true;
23         }
24
25         // プレイヤーが真正面にいて
26         if (plIsNear)
27         {
28             // プレイヤーが攻撃していたら
29             if (pl->GetAttackFlag() && !enemy->GetAvoidFlag())
30             {
31                 int random = 0;
32                 // HPが80%以上の時
33                 if (enemy->IsHPPcentOver(80))
34                 {
35                     random = 20;
36                 }
37                 // HPが50%以上の時
38                 else if (enemy->IsHPPcentOver(50))
39                 {
40                     random = 50;
41                 }
42                 // HPが30%以上の時
43                 else if (enemy->IsHPPcentOver(30))
44                 {
45                     random = 65;
46                 }
47                 // HPが30%以下の時
48                 else
49                 {
50                     random = 70;
51                 }
52                 // 回避行動フラグをtrueにする
53                 // 1回しか回ってほしくないため
54                 // 何回も回っていたら100%回避行動をしてしまう
55                 // ランダム値を生成した後に回避フラグをtrueにする
56                 enemy->SetAvoidFlag(true);
57                 // ランダムに回避行動をする
58                 if (rand() % 100 <= random)
59                 {
60                     return true;
61                 }
62             }
63             else
64             {
65                 return false;
66             }
67         }
68     }
69 
```

# ～担当箇所説明～

## MoveJudgement.h

```
5  class Enemy;
6
7  // 移動の判定クラス
8  // これの子供がMoveLRとMoveZとなる
9  struct MoveJudgement : public JudgementBase
10 {
11     static MoveJudgement& getInstance(void)
12     {
13         static MoveJudgement sInstance_;
14         return sInstance_;
15     }
16
17     bool Judgement(Enemy* enemy);
18 }
19 
```

## MoveJudgement.cpp

```
4     bool MoveJudgement::Judgement(Enemy* enemy)
5     {
6         // 攻撃している時は動かさない
7         if (enemy->GetAttackFlag() || enemy->GetSkillAttackFlag())
8         {
9             return false;
10        }
11
12        // プレイヤーとの距離を測る
13        auto dis = enemy->GetDistance();
14
15        // 奥行移動
16        if (abs(dis.x) > enemy->GetVisionRange().attackRange_)
17        {
18            enemy->SetMoveType(MoveType::MoveLR);
19            return true;
20        }
21
22        // 奥行移動
23        if (abs(dis.x) > enemy->GetVisionRange().attackRange_)
24        {
25            enemy->SetMoveType(MoveType::MoveLR);
26            return true;
27        }
28
29        // Z軸の状態を見ている
30        // プレイヤーよりも手前にいる場合
31        if (dis.z < 0.0f)
32        {
33            enemy->SetZPosStatus(ZPosFromPlayer::Minus);
34            enemy->SetMoveType(MoveType::MoveZ);
35            return true;
36        }
37
38        // Z軸の状態
39        // プレイヤーよりも奥にいる場合
40        if (dis.z > 0.0f)
41        {
42            enemy->SetZPosStatus(ZPosFromPlayer::Plus);
43            enemy->SetMoveType(MoveType::MoveZ);
44            return true;
45        }
46
47        // 何にも当てはまらない場合MoveTypeをNon(動いていない状態)に
48        enemy->SetMoveType(MoveType::Non);
49
50        return false;
51    }
52 }
```

# ～担当箇所説明～

MoveLRJudgement.h

```
4     class Enemy;
5
6     // 横移動の判定クラス
7     struct MoveLRJudgement :
8         public JudgementBase
9     {
10     public:
11
12     static MoveLRJudgement& getInstance(void)
13     {
14         static MoveLRJudgement sInstance_;
15         return sInstance_;
16     }
17
18     bool Judgement(Enemy* enemy);
19 };
```

MoveLRJudgement.cpp

```
4     bool MoveLRJudgement::Judgement(Enemy* enemy)
5     {
6         // 横移動可能であり、攻撃していなければtrueを返して、実行ノードに受け渡す
7         if (enemy->GetMoveType() == MoveType::MoveLR)
8         {
9             return true;
10        }
11    }
12 }
```

# ～担当箇所説明～

ZMoveJudgement.h

```
4  class Enemy;
5
6  // 奥行移動の判定クラス
7  struct ZMoveJudgement : public JudgementBase
8  {
9      public:
10
11     static ZMoveJudgement& getInstance(void)
12     {
13         static ZMoveJudgement sInstance_;
14         return sInstance_;
15     }
16
17     bool Judgement(Enemy* enemy);
18 };
19 }
```

ZMoveJudgement.cpp

```
4     bool ZMoveJudgement::Judgement(Enemy* enemy)
5     {
6         // 奥移動可能であり、攻撃していなければtrueを返して、実行ノードに受け渡す
7         if (enemy->GetMoveType() == MoveType::MoveZ)
8         {
9             return true;
10        }
11    }
12 }
```

# ～担当箇所説明～

## AvoidanceAction.h

```
4  class Enemy;
5
6  // 攻撃を避ける行動クラス
7  struct AvoidanceAction :
8      public ActionBase
9  {
10     static AvoidanceAction& getInstance(void)
11     {
12         static AvoidanceAction sInstance_;
13         return sInstance_;
14     }
15
16     void Run(Enemy* enemy);
17 }
```

## AvoidanceAction.cpp

```
4  void AvoidanceAction::Run(Enemy* enemy)
5  {
6      // アニメーション状態を攻撃回避行動状態に変更
7      enemy->SetAnimExecuter(&Enemy::Avoidance);
8 }
```

# ～担当箇所説明～

ActionBase.h

```
3  class Enemy;
4
5  // 行動クラスの親クラス
6  struct ActionBase
7  {
8      public:
9
10     // 実行
11     void operator () (Enemy* enemy)
12     {
13         Run(enemy);
14     }
15     // 純粹仮想オーバーライド
16     virtual void Run(Enemy* enemy) = 0;
17
18     protected:
19
20 };
```

# ～担当箇所説明～

MoveLRAction.h

```
5  class Enemy;
6  // 横移動のアクション実行クラス
7  struct MoveLRAction :
8      public ActionBase
9  {
10     public:
11         static MoveLRAction& getInstance(void)
12         {
13             static MoveLRAction sInstance_;
14             return sInstance_;
15         }
16
17         void Run(Enemy* enemy);
18     };
```

MoveLRAction.cpp

```
4     void MoveLRAction::Run(Enemy* enemy)
5     {
6         // 既にMoveLR状態だと何もしない
7         if (enemy->GetMoveType() != MoveType::MoveLR)
8         {
9             return;
10        }
11        // アニメーション状態をMove状態に変更
12        enemy->SetAnimExecuter (&Enemy::Move);
13    }
```

# ～担当箇所説明～

ZMoveAction.h

```
4  class Enemy;
5
6  // 奥行歩行のアクション実行クラス
7  struct ZMoveAction :
8  {
9      public:
10
11     static ZMoveAction& getInstance(void)
12     {
13         static ZMoveAction sInstance_;
14         return sInstance_;
15     }
16
17     void Run(Enemy* enemy);
18
19 };
20
```

ZMoveAction.cpp

```
4  void ZMoveAction::Run(Enemy* enemy)
5  {
6      // 既にMoveZ状態だと何もしない
7      if (enemy->GetMoveType() != MoveType::MoveZ)
8      {
9          return;
10     }
11
12     // アニメーション状態をZMoveに変更
13     enemy->SetAnimExecuter (&Enemy::ZMove);
14 }
```

# ～担当箇所説明～

NormalAttackAction.h

```
4 // 通常攻撃の実行クラス
5 struct NormalAttackAction :
6     public ActionBase
7 {
8     public:
9     static NormalAttackAction& getInstance(void)
10    {
11        static NormalAttackAction sInstance_;
12        return sInstance_;
13    }
14
15    void Run(Enemy* enemy);
16 }
```

NormalAttackAction.cpp

```
4 void NormalAttackAction::Run(Enemy* enemy)
5 {
6     // アニメーション状態をAttack状態に変更
7     enemy->SetAnimExecuter (&Enemy::Attack);
8 }
```

# ～担当箇所説明～

SkillAttackAction.h

```
4  class Enemy;
5
6
7  // skillAttackの実行クラス
8  struct SkillAttackAction :
9      public ActionBase
10 {
11     public:
12
13     static SkillAttackAction& getInstance(void)
14     {
15         static SkillAttackAction sInstance_;
16         return sInstance_;
17     }
18
19     void Run(Enemy* enemy);
20 }
```

SkillAttackAction.cpp

```
4  void SkillAttackAction::Run(Enemy* enemy)
5  {
6      // アニメーション状態をAttack状態に変更
7      enemy->SetAnimExecuter (&Enemy::Attack);
8 }
```

# ～担当箇所説明～

## ActionNode.h

```
11 // 優先度をとりあえず一番小さい数にしておく
12 #define DefPriority -100000
13
14 class Enemy;
15 class ActionNode;
16 class BehaviorData;
17
18 // ビューハイアード(思考ノード)クラス
19 class ActionNode
20 {
21 public:
22     ActionNode(std::string nodeName,ActionNode* parent,ActionNode* sibling,int priority,BehaviorTree::SelectRule rule,JudgementBase* judgement,ActionBase* action);
23     // ノード推論
24     ActionNode* Inference(Enemy* enemy,BehaviorData* data);
25     // ノードの検索
26     ActionNode* SearchActionNode(std::string nodeName);
27     // 自分の子供に追加
28     void AddChild(ActionNode* childNode);
29     // 末尾を取得
30     ActionNode* GetLastChild(void);
31     // 判定
32     bool Judgement(Enemy* enemy);
33     // 優先度の取得
34     const int& GetPriority(void)
35     {
36         return priority_;
37     }
38     // 実行データをもっているか
39     bool HasAction(void);
40     // 実行
41     void Run(Enemy* enemy);
42
43     // ノード名の取得
44     std::string GetNodeName(void)
45     {
46         return nodeName_;
47     }

```

## ActionNode.hの続き

```
51     protected:
52         // ノードの名前
53         std::string nodeName_;
54         // 自分の子ノード
55         std::vector<ActionNode*> child_;
56         // 自分の親ノード
57         ActionNode* parent_;
58         // 自分の兄弟ノード
59         ActionNode* sibling_;
60         // 実行ノード
61         ActionBase* action_;
62         // 判定ノード
63         JudgementBase* judgement_;
64         // 選択法
65         BehaviorTree::SelectRule selectRule_;
66         // 優先度
67         int priority_;
68     };

```

# ～担当箇所説明～

## ActionNode.cpp

### コンストラクタ

```
6  ActionNode::ActionNode(std::string nodeName, ActionNode* parent,
7  ActionNode* sibling,int priority, BehaviorTree::SelectRule rule,
8  JudgementBase* judgement, ActionBase* action):
9  nodeName_(nodeName),parent_(parent),
10 sibling_(sibling),priority_(priority),
11 selectRule_(rule),judgement_(judgement),
12 action_(action),child_(NULL)
13 [
14 ]
```

### Inference関数

```
16  ActionNode* ActionNode::Inference(Enemy* enemy,BehaviorData* data)
17  {
18      // 実行ノード格納用
19      ActionNode* result = nullptr;
20
21      for (int i = 0; i < child_.size(); i++)
22      {
23          if (child_[i]->judgement_ != nullptr)
24          {
25              // 判定がtrueになったら実行ノード格納用変数に格納
26              if (child_[i]->Judgement(enemy))
27              [
28                  result = child_[i];
29              ]
30          }
31
32          if (result != nullptr)
33          [
34              // 実行データをもっていれば終了
35              if (result->HasAction())
36              [
37                  return result;
38              ]
39          else
40          [
41              // もっていないければ推論続行
42              result = result->Inference(enemy,data);
43          ]
44
45
46
47      }
48 }
```

# ～担当箇所説明～

ActionNode.cpp

SearchActionNode関数

```
45
46     ActionNode* ActionNode::SearchActionNode(std::string nodeName)
47     {
48         if (nodeName_ == nodeName)
49         {
50             return this;
51         }
52         else
53         {
54             // 子ノードで検索
55             for (auto itr = child_.begin(); itr != child_.end(); itr++)
56             {
57                 ActionNode* node = (*itr)->SearchActionNode(nodeName);
58                 if (node != nullptr)
59                 {
60                     return node;
61                 }
62             }
63         }
64     }
65
66     return nullptr;
67 }
68
69 void ActionNode::AddChild(ActionNode* childNode)
70 {
71     child_.push_back(childNode);
72 }
73
74 }
```

Judgement関数  
HasAction関数  
Run関数

```
75
76     bool ActionNode::Judgement(Enemy* enemy)
77     {
78         // 行動の判定
79         if (judgement_ != nullptr)
80         {
81             return judgement_->operator()(enemy);
82         }
83     }
84
85     bool ActionNode::HasAction(void)
86     {
87         // 実行ノードをもっていればtrue
88         return action_ != nullptr?true:false;
89     }
90
91     void ActionNode::Run(Enemy* enemy)
92     {
93         // 行動の実行
94         if (action_ != nullptr)
95         {
96             action_->operator()(enemy);
97         }
98     }
99
100 }
```

# ～担当箇所説明～

## BehaviorData.h

```
8     class ActionNode;
9     class Enemy;
10
11    // ビヘイビューデータ
12    class BehaviorData
13    {
14        public:
15            BehaviorData();
16            ~BehaviorData();
17            // ビヘイビューデータの初期化
18            void Init(void);
19            // シーケンスノードのpush
20            void PushSequenceNode(ActionNode* node);
21            // シーケンスノードのポップ
22            // 中身の取り出し
23            ActionNode* PopSequenceNode(void);
24            // ノードの使用判定
25            bool IsUsedNode(std::string nodeName);
26            // 使用済みノードに登録
27            void EntryUsedNode(std::string nodeName);
28            // シーケンステップの取得
29            int GetSequenceStep(std::string nodeName);
30            // シーケンステップのセット
31            void SetSequenceStep(std::string nodeName, int step);
32            // 使用済みノードのリセット
33            void ResetNodeUsed(std::vector<ActionNode*>* resetHierarchy);
34        private:
35            // シーケンスノードスタック
36            std::stack<ActionNode*> sequenceStack_;
37            // 実行シーケンスのステップマップ
38            std::map<std::string, int> runSequenceStepMap_;
39            // ノードの使用判定マップ
40            std::map<std::string, bool> usedNodeMap_;
41    };
```

# ～担当箇所説明～

BehaviorData.cpp

コンストラクタ  
デストラクタ  
Init関数

```
5  BehaviorData::BehaviorData()
6  {
7      // ビヘイビアデータの初期化
8      Init();
9  }
10
11 BehaviorData::~BehaviorData()
12 {
13 }
14
15 void BehaviorData::Init(void)
16 {
17     runSequenceStepMap_.clear();
18     while (sequenceStack_.size() > 0)
19     {
20         sequenceStack_.pop();
21     }
22 }
```

# ～担当箇所説明～

BehaviorData.cpp

PushSequenceNode関数  
PopSequenceNode関数  
IsUsedNode関数

```
24 void BehaviorData::PushSequenceNode(ActionNode* node)
25 {
26     sequenceStack_.push(node);
27 }
28
29 ActionNode* BehaviorData::PopSequenceNode(void)
30 {
31     // 中身がないならnullptrを返す
32     if (sequenceStack_.empty())
33     {
34         return nullptr;
35     }
36     ActionNode* node = sequenceStack_.top();
37
38     // 取り出したデータがnullptrでなければ
39     if (node != nullptr)
40     {
41         // 取り出したデータを削除
42         sequenceStack_.pop();
43     }
44
45     return node;
46 }
47
48 bool BehaviorData::IsUsedNode(std::string nodeName)
49 {
50     // 使用していないかったらfalse
51     if (usedNodeMap_.count(nodeName) == 0)
52     {
53         return false;
54     }
55     // 使用していたらそのノードを返す
56     return usedNodeMap_[nodeName];
57 }
```

EntryUsedNode関数

GteSequenceStep関数

SetSequenceStep関数

ResetNodeUsed関数

```
59 void BehaviorData::EntryUsedNode(std::string nodeName)
60 {
61     // 使用しているというフラグを立てる
62     usedNodeMap_[nodeName] = true;
63 }
64
65 int BehaviorData::GetSequenceStep(std::string nodeName)
66 {
67     // 今現在の行動ステップの取得
68     if (runSequenceStepMap_.count(nodeName) == 0)
69     {
70         // ステップがツリーの始点の場合0にセットする
71         runSequenceStepMap_[nodeName] = 0;
72     }
73     return runSequenceStepMap_[nodeName];
74 }
75
76 void BehaviorData::SetSequenceStep(std::string nodeName, int step)
77 {
78     // 思考ステップのセット
79     runSequenceStepMap_[nodeName] = step;
80 }
81
82 void BehaviorData::ResetNodeUsed(std::vector<ActionNode*>* resetHierarchy)
83 {
84     // ノードを使用しているフラグのリセット
85     for (auto itr = resetHierarchy->begin(); itr != resetHierarchy->end(); itr++)
86     {
87         usedNodeMap_[(*itr)->GetnodeName()] = false;
88     }
89 }
```

# ～担当箇所説明～

## BehaviorTree.h

```
5     struct JudgementBase;
6     class BehaviorData;
7     struct ActionBase;
8     class Enemy;
9     class ActionNode;
10    // 思考ルートの木データクラス
11    class BehaviorTree
12    {
13    public:
14        // 選択ルール
15        enum SelectRule
16        [
17            Non,
18            // 優先順位
19            Priority,
20            // シーケンス
21            Sequence,
22            // シーケンシャルルーティング
23            SequentialLooping,
24            // ランダム
25            Random,
26            // オン・オフ
27            On_Off,
28        ];
29
30        BehaviorTree();
31        ~BehaviorTree();
32        // behaviorTreeへのノードの追加
33        void AddNode(std::string searchName, std::string entryName, int priority, SelectRule selectRule, JudgementBase* judgement, ActionBase* action);
34        // ノードの推論
35        ActionNode* Inference(Enemy* enemy, BehaviorData* data);
36        // シーケンスノードから推論
37        ActionNode* SequenceInference(ActionNode* sequenceNode, Enemy* enemy, BehaviorData* data);
38        // 実行
39        void Run(Enemy* enemy, ActionNode* actionNode, BehaviorData* data);
40    private:
41        // ルート(ノードの一番最初)
42        ActionNode* root_;
43    };
```

# ～担当箇所説明～

BehavioTree.cpp

コンストラクタ、デストラクタ

```
7  void BehaviorTree::BehaviorTree()
8  {
9  }
10
11 void BehaviorTree::~BehaviorTree()
12 {
13 }
```

## AddNode関数

```
15 void BehaviorTree::AddNode(std::string searchName, std::string entryName, int priority, SelectRule selectRule, JudgementBase* judgement, ActionBase* action)
16 [
17     // searchNameの文字列が空だと
18     if (searchName.size() != 0)
19     [
20         // まず親ノードを探す
21         ActionNode* searchNode = root_->SearchActionNode(searchName);
22         // 親がいれば
23         if (searchNode != nullptr)
24         [
25             // 末尾の子供を探す
26             ActionNode* sibling = searchNode->GetLastChild();
27             // 追加するノードの作成
28             ActionNode* addNode = new ActionNode(entryName, searchNode, sibling, priority, selectRule, judgement, action);
29             // ノードの追加
30             searchNode->AddChild(addNode);
31         ]
32     ]
33     else
34     [
35         // そもそもルートがなければ
36         if (root_ == nullptr)
37         [
38             // ルートの作成
39             // childとsiblingは共にnullptr(自信が親なので)
40             root_ = new ActionNode(entryName, nullptr, nullptr, priority, selectRule, judgement, action);
41         ]
42     ]
43     else
44     [
45         // アサートをかける
46         _ASSERT("root is already registered!!");
47     ]
48 ]
```

## Inference関数

```
50     ActionNode* BehaviorTree::Inference(Enemy* enemy, BehaviorData* data)
51     [
52         // ノード推論
53         return root_->Inference(enemy, data);
54     ]
55
56     ActionNode* BehaviorTree::SequenceInference(ActionNode* sequenceNode, Enemy* enemy, BehaviorData* data)
57     [
58         // ノード推論をして推論した結果、行動可能なノードを返す
59         return sequenceNode->Inference(enemy, data);
60     ]
61
62     void BehaviorTree::Run(Enemy* enemy, ActionNode* actionNode, BehaviorData* data)
63     [
64         // ノードの実行
65         actionNode->Run(enemy);
66     ]
```

# ～担当箇所説明～

## ObjectPool.h

```
5 // ポул親クラス
6 class ObjectPool
7 {
8 public:
9     ObjectPool() {};
10    virtual ObjectPool() {};
11    // poolの破棄
12    virtual void Destroy(void) = 0;
13    // poolの更新
14    virtual void Update(void) = 0;
15    // poolの中身の人たちの描画
16    virtual void Draw(void) = 0;
17 private:
18 protected:
19     // poolのサイズ
20     int poolSize_;
21 };
22 }
```

# ～担当箇所説明～

## EnemyPool.h

```
10 // 敵のタイプ数の思考ループ配列
11 using Behavior = std::array<BehaviorTree, static_cast<int>(ENEMYTYPE::Max)>;
12 // 敵の箱を保管しておくバッファ(ObjectPooling法)
13
14 class EnemyPool :
15     public ObjectPool
16 {
17 public:
18     EnemyPool();
19     ~EnemyPool();
20     // poolの生成
21     void Create(int size, std::shared_ptr<Player>& player);
22     // poolの中から死んでいるやつの拾い上げ
23     std::shared_ptr<Enemy>& Pickup(int spawnCnt);
24     // poolの破棄(Gameの終了時に使用)
25     void Destroy(void);
26     // poolの中身のメンバーの更新(生きているやつのみ)
27     void Update(void);
28     // poolの中身のメンバーの描画(生きているやつのみ)
29     void Draw(void);
30     // ColliderBoxの描画(debug表示)
31     void DrawCollider(void);
32
33     // タイプに合わせたヒューリックの作成
34     void BehaviorRegistrator(ENEMYTYPE type);
35
36 private:
37     // poolに溜まっている敵たち
38     std::vector<std::shared_ptr<Enemy>> poolMembers_;
39     // poolから拾い上げる敵に与えるヒューリック
40     // タイプ毎に異なる
41     Behavior behaviors_;
42 }
```

# ～担当箇所説明～

## EnemyPool.cpp

コンストラクタ  
デストラクタ  
Create関数

```
21  EnemyPool::EnemyPool()
22  [
23  ]
24
25
26  EnemyPool::~EnemyPool()
27  [
28  ]
29
30  void EnemyPool::Create(int size, std::shared_ptr<Player>& player)
31  [
32      // sizeに応じてpoolの大きさが決まる
33      for (int s = 0; s < size; s++)
34      [
35          // まずとにかく敵の箱を用意
36          poolMembers_.push_back(std::make_shared<Enemy>(10,player));
37      ]
38      // poolのサイズ
39      poolSize_ = size;
40 ]
```

## PickUp関数

```
42  std::shared_ptr<Enemy>& EnemyPool::Pickup(int spawnCnt)
43  [
44      for (int s = 0; s < spawnCnt; s++)
45      [
46          // poolのサイズで回す
47          for (int i = 0; i < poolSize_; i++)
48          [
49              // 死んでいるやつがいればpoolから拾い上げる
50              if (!poolMembers_[i]->GetAlive())
51              [
52                  // 生き返らせる
53                  poolMembers_[i]->SetAlive(true);
54                  // HPの初期化
55                  poolMembers_[i]->SetHP(10);
56                  // ピペピアのアタッ
57                  poolMembers_[i]->SetBehavior(&behaviors_[static_cast<int>(ENEMYTYPE::Cultist)]);
58                  // 敵のセット
59                  poolMembers_[i]->SetEnemy(ENEMYTYPE::Cultist, Vector3F(300.0f, 100.0f, 100.0f));
60                  // 未使用なので再利用可能
61              ]
62          ]
63      ]
64  ]
65 ]
```

# ～担当箇所説明～

EnemyPool.cpp

Destroy関数  
Update関数

```
67 void EnemyPool::Destroy(void)
68 [
69     for (auto itr = poolMembers_.begin(); itr != poolMembers_.end(); itr++)
70     {
71         poolMembers_.erase(itr);
72     }
73 ]
74
75 void EnemyPool::Update(void)
76 [
77     for (int i = 0; i < poolSize_; i++)
78     {
79         // 生きていれば
80         if (poolMembers_[i]->GetAlive())
81         {
82             // 更新
83             poolMembers_[i]->Update();
84         }
85     }
86 ]
```

Draw関数 → バッファに存在し、且つ  
生きているオブジェクトを描画  
DrawCollider → 当たり判定Boxの描画  
主にDebugで使用

```
88 void EnemyPool::Draw(void)
89 [
90     for (auto member : poolMembers_)
91     {
92         // 生きていれば
93         if (member->GetAlive())
94         {
95             // 描画
96             member->Draw();
97         }
98     }
99 ]
100
101 void EnemyPool::DrawCollider(void)
102 [
103     for (auto member : poolMembers_)
104     {
105         // 生きていれば
106         if (member->GetAlive())
107         {
108             // コライダーボックスの描画
109             member->DebugColliderDraw();
110         }
111     }
112 ]
```

# ～担当箇所説明～ EnemyPool.cpp

## BehaviorRegistrar関数

→ 追加する敵タイプのビヘイビアツリーに  
思考ノードの格納

```
114 void EnemyPool::BehaviorRegistrar(ENEMYTYPE type)
115 {
116     // まずルートの生成
117     behaviors_[static_cast<int>(type)].AddNode("", "root", 0, BehaviorTree::SelectRule::Priority, nullptr, nullptr);
118     switch (type)
119     {
120         case ENEMYTYPE::CultistAssassin:
121             // 移動の判定クラスと実行クラスの格納(root -> moveノードに繋げる)
122             behaviors_[static_cast<int>(type)].AddNode("root", "move", 3,
123                 BehaviorTree::SelectRule::Random,
124                 &MoveJudgement::getInstance(),
125                 nullptr);
126             // 攻撃の判定クラスと実行クラスの格納(root -> attackノードに繋げる)
127             behaviors_[static_cast<int>(type)].AddNode("root", "attack", 2,
128                 BehaviorTree::SelectRule::Sequence,
129                 &AttackJudgement::getInstance(),
130                 nullptr);
131             // 回避行動の判定クラスと実行クラスの格納(root -> avoidanceノードに繋げる)
132             behaviors_[static_cast<int>(type)].AddNode("root", "avoidance", 1,
133                 BehaviorTree::SelectRule::Non,
134                 &AvoidanceJudgement::getInstance(),
135                 &AvoidanceAction::getInstance());
136             // 横移動行動の判定クラスと実行クラスの格納(root -> move -> moveLRと繋げる)
137             behaviors_[static_cast<int>(type)].AddNode("move", "moveLR", 1,
138                 BehaviorTree::SelectRule::Non,
139                 &MoveLRJudgement::getInstance(),
140                 &MoveRAction::getInstance());
141             // 奥移動行動の判定クラスと実行クラスの格納(root -> move -> moveZと繋げる)
142             behaviors_[static_cast<int>(type)].AddNode("move", "moveZ", 2,
143                 BehaviorTree::SelectRule::Non,
144                 &ZMoveJudgement::getInstance(),
145                 &ZMoveAction::getInstance());
146             // 普通の攻撃行動の判定クラスと実行クラスの格納(root -> attack -> normalAttackと繋げる)
147             behaviors_[static_cast<int>(type)].AddNode("attack", "normalAttack", 1,
148                 BehaviorTree::SelectRule::Non,
149                 &NormalAttackJudgement::getInstance(),
150                 &NormalAttackAction::getInstance());
151             // skillAttack行動の判定クラスと実行クラスの格納(root -> attack -> skillAttackと繋げる)
152             behaviors_[static_cast<int>(type)].AddNode("attack", "skillAttack", 2,
153                 BehaviorTree::SelectRule::Non,
154                 &SkillAttackJudgement::getInstance(),
155                 &SkillAttackAction::getInstance());
156
157         break;
158     }
159 }
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197 }
```

## BehaviorRegistrar関数の続き

```
case ENEMYTYPE::Cultist:
    // 移動の判定クラスと実行クラスの格納(root -> moveノードに繋げる)
    behaviors_[static_cast<int>(type)].AddNode("root", "move", 3,
        BehaviorTree::SelectRule::Random,
        &MoveJudgement::getInstance(),
        nullptr);
    // 攻撃の判定クラスと実行クラスの格納(root -> attackノードに繋げる)
    behaviors_[static_cast<int>(type)].AddNode("root", "attack", 2,
        BehaviorTree::SelectRule::Sequence,
        &AttackJudgement::getInstance(),
        nullptr);
    // 回避行動の判定クラスと実行クラスの格納(root -> avoidanceノードに繋げる)
    behaviors_[static_cast<int>(type)].AddNode("root", "avoidance", 1,
        BehaviorTree::SelectRule::Non,
        &AvoidanceJudgement::getInstance(),
        &AvoidanceAction::getInstance());
    // 横移動行動の判定クラスと実行クラスの格納(root -> move -> moveLRと繋げる)
    behaviors_[static_cast<int>(type)].AddNode("move", "moveLR", 1,
        BehaviorTree::SelectRule::Non,
        &MoveLRJudgement::getInstance(),
        &MoveRAction::getInstance());
    // 奥移動行動の判定クラスと実行クラスの格納(root -> move -> moveZと繋げる)
    behaviors_[static_cast<int>(type)].AddNode("move", "moveZ", 2,
        BehaviorTree::SelectRule::Non,
        &ZMoveJudgement::getInstance(),
        &ZMoveAction::getInstance());
    // 普通の攻撃行動の判定クラスと実行クラスの格納(root -> attack -> normalAttackと繋げる)
    behaviors_[static_cast<int>(type)].AddNode("attack", "normalAttack", 1,
        BehaviorTree::SelectRule::Non,
        &NormalAttackJudgement::getInstance(),
        &NormalAttackAction::getInstance());
    break;
case ENEMYTYPE::TwistedCultist:
    break;
case ENEMYTYPE::Max:
    break;
default:
    break;
}
```

# ～担当箇所説明～

使用例(GameScene.cppの中身の一部)

```
113 // 敵パッファの生成  
114 enemyPool_ = std::make_unique<EnemyPool>();  
115 // 敵タイプに応じたビヘイビアツリーテータの生成  
116 enemyPool_->BehaviorRegistrator(ENEMYTYPE::Cultist);  
117 // 敵1体(引数の"1"を変更すれば何体でも生成可能)の生成  
118 enemyPool_->Create(1, player_);  
119 // 生成した敵がパッファの中で泳いでいるので、パッファから拾い上げる  
// (これも引数の数字を変更することで拾いあげる敵数変更可能)  
// ただし、拾いあげる数だけ生成されていないとアサートを出す  
120  
121 enemyPool_->Pickup(1);
```

→ パッファの生成

↓  
敵タイプに応じた  
ビヘイビアツリーのセット

↓  
パッファに指定数だけ  
空の敵を生成

↓  
パッファから指定数拾います