元々レポートだったもの

matsu-ITSP

2019/1/4

1　目的

本ソフトウェアは、艦これにおける支援射撃の火力を調整することを支援するツールである。このゲームにおいて、多くの場合で命中率と攻撃力はトレードオフである。そのため、敵の撃破に必要な攻撃力を維持しつつ、命中率を最大まで高める必要がある。また、出現する敵はある程度決まっている。その敵に対し、支援射撃において一撃で撃破するのに必要な攻撃力を計算するのが本ソフトである。

1.1　ゲームの説明

登場する友軍は、主艦隊、道中支援艦隊、ボス支援艦隊である。一戦ごとの戦闘の進み方は、今回考えることが必要なところは、以下の通りである。
1. 航空戦 2. 支援射撃 3. 主艦隊の砲撃戦 4. 雷撃戦
ダメージの大きさとしては、$3 > 2 > 1 > 4$ である。2. 支援射撃以外は主艦隊のみで行う。これを「道中艦隊」として複数の敵と行った後、「ボス艦隊」とも同じ戦闘スタイルで行う。その「ボス艦隊」の旗艦を倒せばステージクリアとなる。「道中艦隊」では、いかに被害を少なくして「ボス艦隊」までたどり着くかが重要であり、主な被ダメージ源である砲撃戦を極力早く終わらせたい。そのため、砲撃戦の前に行われる支援射撃は非常に重要である。また、「ボス艦隊」においても、主艦隊の砲撃を旗艦に狙わせるため、支援射撃で随伴艦を減らすことは重要である。

また、ユーザーが戦闘の前に選べる「陣形」というものがあり (単縦陣、輪形陣など)、選択によって色々な強化がなされる。通常艦隊だと 6 種類だが、連合艦隊 (通常艦隊より大きい艦隊である。今回考えるところでは大きな変更はない。) では 4 種類になる。似たような補正だがランダムで決まる「交戦隊形」というものがあり (T 字有利、反航戦など)、これによりお互いに攻撃力が 1.2 から 0.6 倍される。

1.2　ダメージ計算式について

計算式および敵データは艦これ wiki(https://wikiwiki.jp/kancolle/) を参考にした。正しいと信じられているサイトである。ダメージ計算は以下の式で表される。[] はガウス記号である。

ダメージ = [[(*CAP* 後攻撃力 − 防御力)] ∗ クリティカル補正]

CAP 後攻撃力 = *CAP* 値 + $\sqrt{(CAP\text{ 前攻撃力} - CAP\text{ 値})}$

CAP 値 = 150

CAP 前攻撃力 = 基本攻撃力× *CAP* 前補正

基本攻撃力 = 火力 + 支援定数 (+4)

CAP 前補正 = 交戦隊形補正 ∗ 陣形補正

防御力 = 装甲× 0.7 + 整数乱数 (0〜装甲 − 1) × 0.6

クリティカル補正 = クリティカルなら 1.5, そうでなければ 1.0 交戦隊形・陣形補正は以下のようになっている。

交戦隊形補正 = (T 有利:1.2 , 同航:1.0 , 反航;0.8 , T 不利:0.6)

陣形補正 (通常) = (単縦:1.0 , 複縦:0.8 , 輪形:0.7 , 梯形:0.6 , 単横:0.6 , 警戒:0.5)

陣形補正 (連合) = (第一:0.8 , 第二:1.0 , 第三:0.7 , 第四:1.0)

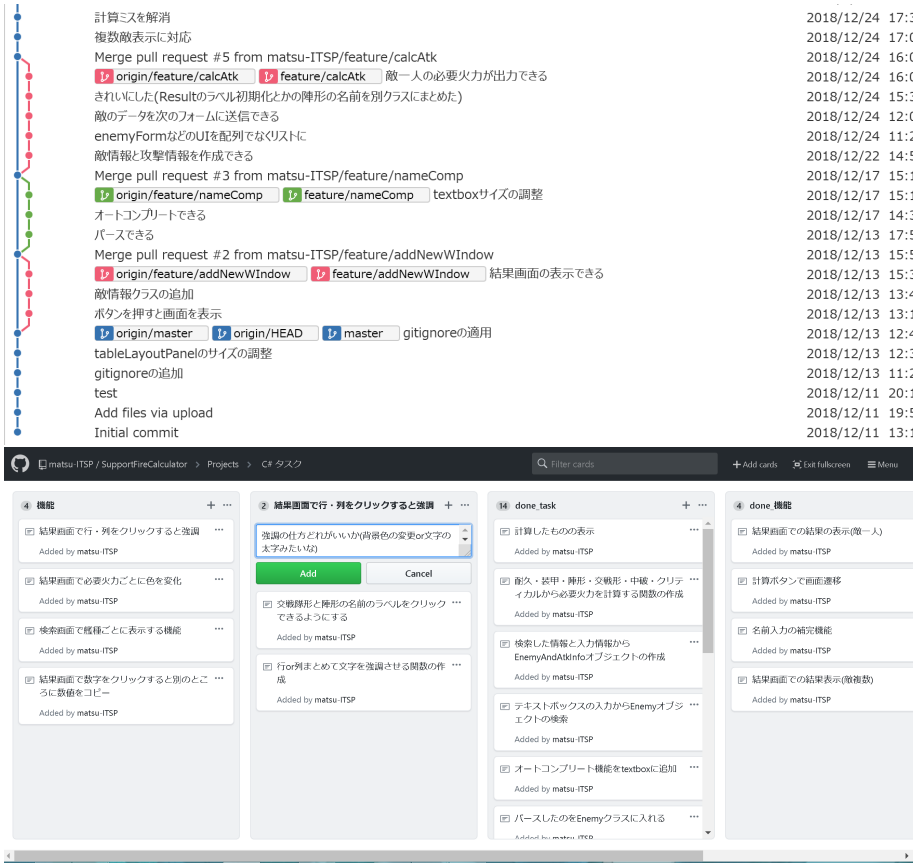
2　実装

2.1　正確な見積もり

登録時は「二週間程度」としたが、これはイベントに間に合わせたかったという理由であり、正しい開始時間が発表された時点で
(https://twitter.com/KanColle_STAFF/status/1077365418229096449)
締め切りは 12/26 の 23:00 となった。

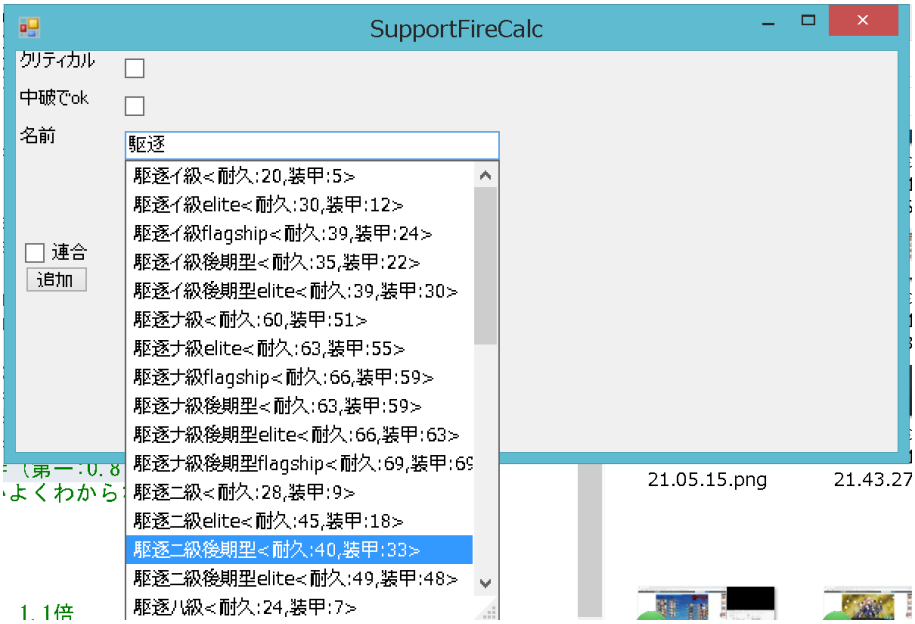
2.2　方針

バージョン管理は github を用いた。ブランチモデルとして gitflow を使おうとしたが、push がうまくいかなかったので途中でやめた。タスク管理は github の Project 機能を用いた。必要なおおまかな機能を書きだした後、その機能を細かくタスクに分割して進めた。以下にブランチと Project の画像を示す。



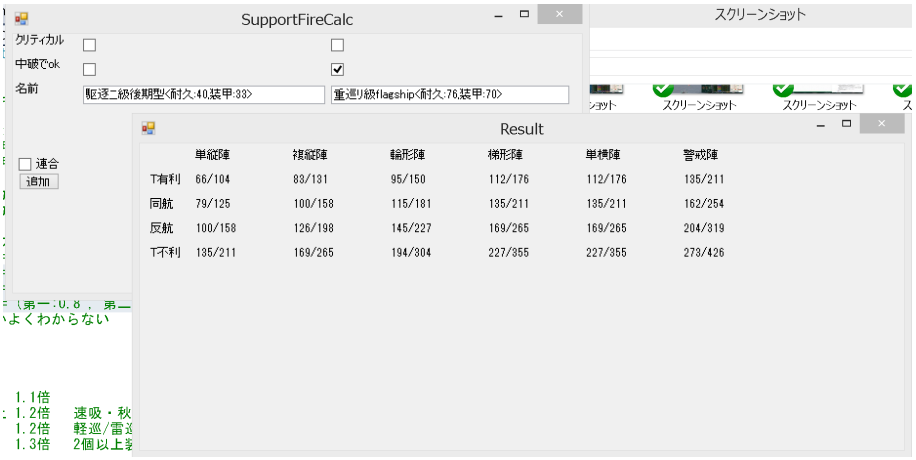
2.3 画面遷移

初期画面は上のものである。文字入力を行うと次のようにオートコンプリート機能が働く。

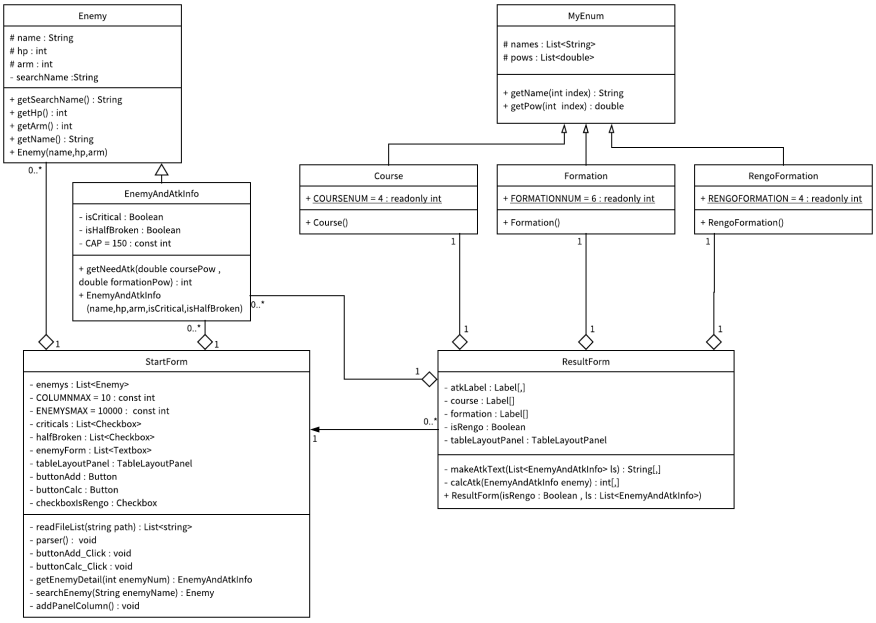


さらに追加を押すと、複数の入力フォームに入力を行える。

計算ボタンを押すと、必要な火力が表示される。/の左が駆逐に対する必要火力、右が重巡に対する必要火力である。



2.4 クラス構成



まずは画面以外のクラスの説明を行う。

- Enemy クラス
  敵の情報が入ったクラスである。名前、耐久、装甲が中心のデータであり、コンストラクタの引数として必要となる。searchName は、コンストラクタ内で作成され、フォームで名前を入力されたときにオートコンプリートを行うために必要である。同じ名前で異なる耐久・装甲をもつものが存在したので必要となった。searchName は

name + ”⟨ 耐久:” + hp.ToString() + ”, 装甲:” + arm.ToString() + ”⟩”
という形になっている。例えば、名前が A、耐久 5、装甲 10 の敵が存在したとすると、searchName は、
A⟨ 耐久:5, 装甲:10⟩
となる。フォームに入力したとき、この文字列がオートコンプリートとして表示される。

- EnemyAndAtkInfo クラス
  上の敵の情報に加え、チェックボックスで入力される攻撃情報 (クリティカルで当たったとするか否か、半分減らせば十分とするか) をもつクラスである。getNeedAtk で、オブジェクトが持つ敵情報・攻撃情報と、引数の陣形補正・交戦隊形補正から、必要な攻撃力を計算する。

- MyEnum クラス
  数値データ付き Enum 型を使おうとしたのだが、うまくいかなかったので作った、Enum のような何かである。陣形補正・交戦隊形補正を保管するためのクラスである。名前と補正値を持つ。get であるが、index は何番目の要素かを表す。これが全く Enum らしくないのだが、やりたいことは、要素を順番に取ってくることだけだったので、これでよしとした。

- Course,Formation,RengoFormation クラス
  それぞれ交戦隊形補正、陣形補正、陣形補正 (連合艦隊用) の MyEnum の具体的な実装である。例えば交戦隊形補正は、コンストラクタで

  ```
  names = new List<string>{"T有利","同航戦","反航戦","T不利"};
  pows = new List<double>{1.2 , 1.0 , 0.8 , 0.6};
  ```

  というように具体的なデータを作る。

- StartForm クラスここから画面のクラスであり、これは始めに開く画面である。まずは簡単にクラス変数の説明を行う。

  - enemys:敵の情報、つまり Enemy オブジェクトのリストである。csv で準備した敵データを parser で解析したあとここに保管される。
  - ENEMYMAX:csv に入っている敵の許容できる最大数である。
  - COLUMNMAX:追加ボタンでフォームが追加されるが、それの最大数である。
  - criticals,halfBroken,enemyForm:追加ボタンで追加されるフォームである。クリティカルとして計算するか否かの checkbox、半分減らせば十分とするか否かの checkbox、敵の名前入力を行うための

textbox である。追加ボタンにより数が動的に変わるのでリストで実装した。最大数が COLUMNMAX により決まっているなら配列にすればよい、と思ってはじめは配列で作っていたが、それで実装すると、実際に何個データが入っているか分からなくなったので、リストで実装した。

- tableLayoutPanel:フォームをあてはめるためのレイアウトである。追加ボタンでレイアウトの列が一つ増える。
- buttonAdd,buttonCalc:フォームを追加するためのボタン、計算実行するためのボタンである。
- checkboxIsRengo:連合艦隊か否かを入力するためのチェックボックスである。

次に関数の説明を行う。

- readFileList:引数で得られたパスにあるファイルを読み、行ごとに読んで、String のリストで返す関数である。
- parser：readFileList で csv で準備した敵データを読み、そのデータをパースして enemys に入れる。まず、csv データは以下のような形式になっている。

```
name,hp,arm
駆逐イ級,20,5
駆逐イ級 elite,30,12
...
```

これが readFileList で行ごとに読み取られ、リストで保管されている。そのリストの要素一つ一つに対し、次のようにパースする。

```
String[] temp = lineEnemy[i].Split(‘,’);
//要素数 3 じゃなきゃバグったデータ
//3 でも 2 つ目、3 つ目が数字じゃなきゃバグ
if(temp.Count() == 3 && Int32.TryParse(temp[1],out hp)
                     && Int32.TryParse(temp[2],out arm)){
  name = temp[0];
  enemys.Add(new Enemy(name,hp,arm));
}
```

これにより、正しいデータのみを読みだして enemys に入れることができるようになっている。

- buttonAdd_Click： 追加ボタンが押されたときに呼び出される。addPanelColumn() を呼び出すだけである。

- buttonCalc_Click： 計算ボタンが押されたときに呼び出される。実行されたとき、getEnemyDetail を複数回呼び出して、リストに EnemyAndAtkInfo オブジェクトを enemyForm の数と同じだけ入れる。次にそのリストと chechboxIsRengo のチェックの中身を、ResultForm に与え、結果画面を表示する。
- getEnemyDetail : 何番目のフォームに入力された敵かを受け取り、その入力内容を EnemyAndAtkInfo オブジェクトにまとめて返す。実行されると、まず enemyForm,isCritical,isHalfBroken の指定されたインデックスの要素を String,Boolean,Boolean としてローカル変数に保管する。次に得られた名前を正しい形にする。

  名前がオートコンプリート使わずに入力されていたときは、searchEnemy を用いてその名前を持つ Enemy オブジェクトを得て、そのオブジェクトと攻撃情報から EnemyAndAtkInfo オブジェクトを作成する。

  オートコンプリート機能を用いたときは、Enemy クラスで記述した通り、ユーザーは (名前)⟨ 耐久:n, 装甲:m⟩ の形で入力していると考えられる。この形で記述されていたときは、この文字列を分割して Enemy オブジェクトを作る。具体的には、はじめから ⟨ までを name、:から, までを hp、:から ⟩ までを arm とできる。これらの情報と攻撃情報から、EnemyAndAtkInfo オブジェクトを作成できる。
- searchEnemy： 敵の名前を受け取り、その名前を持つ Enemy オブジェクトを返す。enemys の要素一つ一つについて、その名前 (enemys[i].getName()) と受け取った名前を比較し、同じになったらその enemys[i] を返す。
- addPanelColumn： tableLayoutPanel に一列加える関数である。実際にやっていることは、tableLayoutPanel を一列増やし、criticals,halfBroken,enemyForm の要素を一つ増やし、増やした要素を tableLayoutPanel に当てはめ、増やした enemyForm 要素にオートコンプリート設定を行う。

関数の説明は以上である。最後に StartForm の起動時の流れを説明する。

このソフトを起動させると、StartForm が作成され、まず各変数が初期化される。初期状態の UI は、tableLayoutPanel にフォームは入っていない状態である。次に parser() が呼び出され、敵情報が入った csv ファイルを開き、パースして enemys にデータを入れる。最後にフォームを一つ分追加し、入力待ち状態になる。

- ResultForm クラス StartForm で計算ボタンを押したときに開く画面である。まずは変数の説明を簡単に行う。

- atkLabel：陣形・交戦隊形補正ごとに計算された必要火力を、ラベルとして表示するための変数である。
- course： 交戦隊形の名前をラベルとして表示するための変数である。
- formation：陣形の名前をラベルとして表示するための変数である。
- isRengo：連合艦隊か否かを保管するための変数である。
- tableLayoutPanel：ラベル群を表として表示するためのレイアウトである。

次に関数の説明を行う。

- makeAtkText：EnemyAndAtkInfo のリストを受け取り、必要火力を String の二次元配列として出力する。ローカル変数 atk として int の二次元配列のリストをもつ。まずは calcAtk に受け取ったリストを渡し、その atk に必要火力の集合を入れる。次に返す変数である String 二次元配列 ans に、次のようにデータを入れる。

```
for(int i = 0 ; i < COURSENUM ; i++){
  for(int j = 0 ; j < formationNum() ; j++){
    ans[i,j] = null;
    for(int k = 0 ; k < ls.Count ; k++){
      ans[i,j] = ans[i,j] + atk[k][i,j].ToString() + "/";
    }
    ans[i,j] = ans[i,j].Remove(ans[i,j].Length-1);
    if(ans == null){
      ans[i,j] = "0";
    }
  }
}
```

  これは、まず一つの交戦隊形・陣形に注目し、そこの ans を null で初期化する。次にを atk 一つずつ取ってきて、ans および'/' と結合する。最後に一番後ろの'/' を削除する。このようにすることで、必要火力をまとめることができる。

- calcAtk：EnemyAndAtkInfo オブジェクトを受け取り、陣形・交戦隊形ごとに必要火力を計算したものを int の二次元配列にして返す。

```
for(int i = 0 ; i < COURSENUM ; i++){
  for(int j = 0 ; j < formationNum() ; j++){
    if(enemy == null){
```

```
        ans[i,j] = 0;
      }else if(isRengo){
        ans[i,j] = enemy.getNeedAtk(c.getPow(i),rf.getPow(j));
      }else{
        ans[i,j] = enemy.getNeedAtk(c.getPow(i),f.getPow(j));
      }
    }
  }
```

一つの交戦隊形・陣形に注目し、それぞれについて enemyAndAtkInfo で述べた getNeedAtk を用いて必要火力を計算して、その位置の ans に代入する。

最後に ResultForm 起動時の動作を説明する。まずは tableLayoutPanel を作成する。引数の isRengo が true なら 5*4, そうでなければ 5*6 のテーブルが作成される。次にラベルの配列を初期化する。ここでは特に値は代入しない。次に makeAtkText で必要火力の計算を行い、完成したラベルを tableLayoutPanel に当てはめる。この画面では特に入出力は行わないので、この画面を表示し続けるだけである。

3 考察

3.1 ブランチモデルについて

gitflow について書いたとき、push が上手くいかないと述べた。後半で develop に push するときは、git push でうまくいっていた。gitflow のように、ブランチを切って進めていくときは、git push origin (branch 名) を行い、二回目以降の push は git push でよい、と調べたら出てきた。はじめのほうは上手くいっていたのだが、途中で何故か push(git push origin (branch 名) の方) ができなくなった。調べても原因は見つからず、push や fetch を色々(git ⟨push or fetch⟩ origin ⟨ master or develop or 現ブランチ名) やってみたら何故か push できた。何が原因で詰まったのか、どれのおかげで解決できたのか、全く分からなかったため、これ以降ブランチを切らなくなった。

ここはただの要望であるが、git が全く分からないまま使っている感じがあり、自分で調べてもよく分からない (そもそもどういうキーワードで調べればよいのかもあまり分かっていない) ので、git について教える授業が欲しい。

3.2 Enemy クラス、EnemyAndAtkInfo クラスについて

Enemy クラスと EnemyAndAtkInfo クラスを比較すると、後者に追加で入っている情報が少ない、前者に必要火力を計算する関数を入れれば十分、

などの理由で、後者のクラスはいらないのではないか、と考えるかもしれない。後者を実装した理由を述べる。

StartForm 画面上で追加ボタンを押すと、名前入力フォームと攻撃情報フォームが追加される。これらはデータとしては一つのオブジェクトとしてまとめたほうが綺麗だと考えたので、子クラスを実装した。

理由はもう一つあり、Enemy クラスは、ゲーム中に存在する全ての敵のデータをオブジェクトとして持つ。一方で、結果画面に渡す必要があるのは、その敵のデータとユーザーが入力した攻撃情報である。敵データが一つだけならそれぞれのデータのまま結果画面に渡せばよいが、複数渡すことになるので、結果画面の引数が複雑になると考えた。そこでそれらの情報をまとめたクラスを作っておけばよいと考えた。

3.3 入力処理とオートコンプリートの文字列について

オートコンプリート用文字列として、
A⟨ 耐久:5, 装甲:10⟩
を用意すると述べた。また、ここから実際に計算するときは、これを分割して EnemyAndAtkInfo オブジェクトを作ると述べた。オートコンプリート機能を使っていなければ、name=A を持つ Enemy オブジェクトから EnemyAndAtkInfo オブジェクトを作ると述べた。

まず、こうなった原因は同名の異なる敵が存在したからである。解決策として検索用の別の文字列を用意し、それを再度パースして EnemyAndAtkInfo オブジェクトを作る、という手段をとった。

これには問題点がある。ユーザーが適当に文字列を編集したとき (A⟨ 耐久:5, 装甲:10⟩ を、A⟨ 耐久:5, 装甲:100⟩ にしてしまう、のような)、正しい結果が得られない。直感的な解決策としては、テキストボックスを用いないということがある。ドロップダウンリストに敵をすべて入れるというのが一つの例だろう (この例は入れたい敵を探すのが簡単ではなくなるため、適切ではない。よりよい UI が浮かばなかったのも、これを採用しなかった理由である)。UI を変更する必要があるので、何をどのような配置にするかなど、考えることが多い。今回ははじめに UI があって、それを実装するという方向性があったため、これは見送った。

他の解決策としては、名前そのものを少し変更するということがあげられる。例えば、

```
name=PT 子鬼群,hp=18,arm=39
name=PT 子鬼群,hp=15,arm=29
...
```

というデータがある。これを、csv から読み取った時点で、

```
name=PT 子鬼群_A,hp=18,arm=39
name=PT 子鬼群_B,hp=15,arm=29
...
```

のように置き換える。すると ⟨⟩ 内の文字を削除して、searchEnemy に投げることができる。これのメリットは、入力を受け取った時点で、⟨⟩ があってもなくても、⟨⟩ 内の文字を削除し、検索を行うので、処理を簡潔に書けるようになることだ。しかし欠点もあり、オートコンプリートを使わないときにうまく動かなくなると考えられる。現状はオートコンプリートを無視して検索すると、同じ名前の敵のうち、一番上の敵がヒットする。これは (個人的には) 使い勝手がよい。なぜなら、csv データには上部に同名の強い敵が入っていて、敵として出てくるのはたいてい一番強い敵だからだ。しかし、名前を置き換えると、オートコンプリートを使わないとヒットしなくなる。これで実装するならば、名前の付け方に工夫が必要だろう。

3.4 MyEnum について

Enum を使おうとしたところ、うまくいかなかったので自分で抽象クラスを準備したと述べた。Enum でやりたかったことは、たとえば Course の場合、

```
public static enum Course { Tyuri , Douko , Hanko , Tfuri };
```

と準備しておき、静的関数として toString と toPow を用意する、ということだ。ResultForm と必要火力の二か所で使い、値は変わらないので、静的で宣言している。まずこの時点で問題があり、この enum と関数を ResultForm か EnemyAtkAndInfo のどちらで宣言するかだ。両方で宣言すると、もし交戦隊形が増えたとき、二か所を修正する必要が発生し、適切ではない。決まったとしても、交戦隊形補正の倍率が画面クラス上に保管されている、もしくは敵情報クラスに交戦隊形の名前が保管されているという、違和感のある状態になる。そこで、定数のみをもつクラスを準備すればよいという考えになり、Course クラスを作った。その後、Formation と RengoFormation も構造が同じであるということに気づき、MyEnum という親クラスで構造をまとめた。

3.5 保守について

交戦隊形や陣形はともかく、敵は高頻度で追加される。保守性の向上のため、敵情報を含む csv ファイルだけプログラム上に書かない、という形をとった。しかし手動で更新する必要があるため、wiki の情報を参照して自動更新する、という形にすれば、さらに保守性は向上するだろう。

また、csv にはすべての敵のデータが入っているが、すべては必要ないと考えられる。支援のみで倒せないような強力な敵やボスキャラは、達成不能な

桁違いの計算結果が現れる。例えばよく出てくるボスである戦艦棲姫は、最も有利な条件を設定しても必要火力は 1 万を超える。達成可能な火力は最大で 299 である。従って、このような強力な敵はデータとして入れておく必要はないと思われる。