ゲーム制作プロジェクト
解説資料①

対NPCのタイプ相性カードバトル

名古屋大学大学院　情報学研究科　複雑系科学専攻
多自由度システム情報論

修士1年　鈴木琳久

はじめに

- 本資料について

ポケモンのタイプ相性表を得点表として、NPCとタイプ相性で対戦するカードゲームの概要とコードの解説をした資料になります。

- 制作物について

本プロジェクトで作成したコードはGitHub上で一般公開しています。
https://github.com/rikuli-35/seminar-materials/tree/main/03_programming

本プロジェクトの流れ

- ゲームの概要
- 設計思想
- コード解説（データセット、関数）
- 改善点（バグ）
- 今後の展望

このゲームを作った理由

ポケモンの過去作品の魅力
→昔のポケモンの「レトロ」な感じが好き！
→3Dのキャラクターだけでなく、2Dのドット絵のキャラクターも好き
→もっさりした戦闘にも味がある
などなど、、、

上記の理由から、昔の作品を今でもプレイしている人達がたくさんいる
→ならば、過去作品の「仕様」にも需要があるのではないか？

どんなゲームが作りたかったか

以上の理由から、このようなゲームを作りたいと考えました。

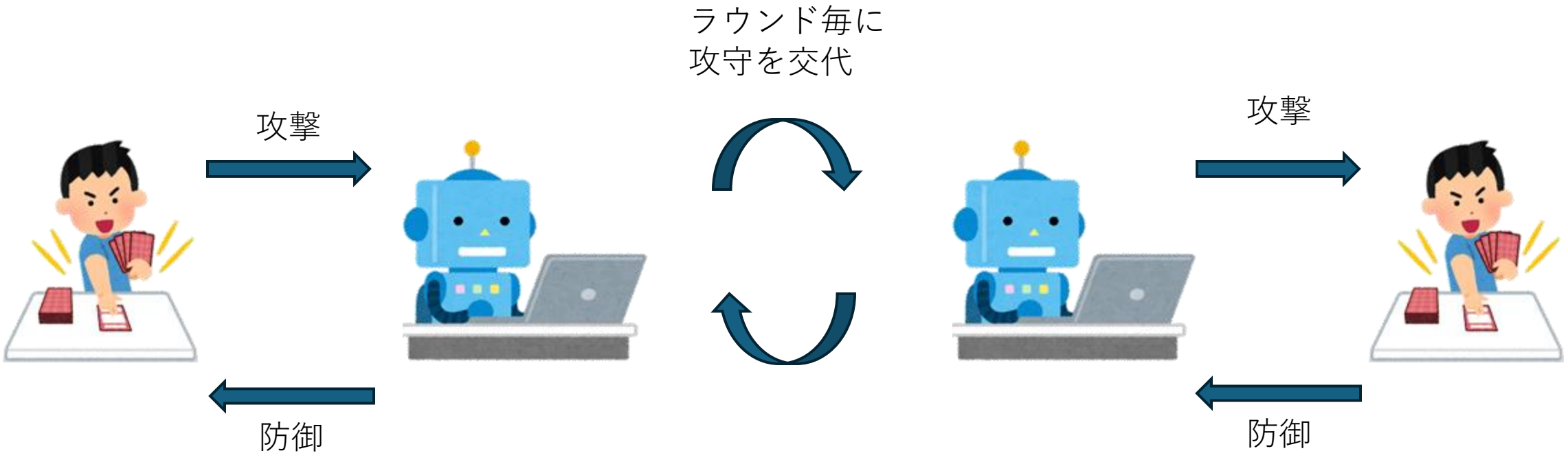
- 今の自分の実力で実装できるようなゲームを作りたい！
- ネットワーク分析で培った知識を応用したい！
- 誰もが楽しめるように「世代を選択」できるような機能を入れたい！
- ゲームが終わった後に、他の世代を続けて遊べるようにしたい！

ゲームの流れ（概略）

①：ポケモンの「タイプ」を手札として、手札がなくなるまで対戦する

②：各ラウンドでプレイヤーとCPUが手札から一枚ずつ選んで「タイプ相性」でバトルする

③：タイプ相性表のダメージ倍率を得点とし、攻撃側に点数が入る



例

- バージョン1（初代ポケモン）の場合：

①：プレイヤーとCPUの初期手札は15枚　（※初代のタイプ数は15）
②：コイントスを行って先攻（攻撃側）と後攻（防御側）を決める　（※コイントスは初回のみ）
③：各ラウンドでお互いに手札を1枚消費する　（※CPUの選択はランダム）
④：タイプ相性表（得点表）に基づいて、攻撃側に得点が入る
⑤：以降、ラウンドを進めるごとに攻守を交代し、手札がなくなるまでラウンドを行う
⑥：最終得点が多い方が勝ち

ゲームの面白さ

- このゲームの一番の醍醐味は、相手との「駆け引き」

相手に勝つためには、、、
→自分、および相手の残りの手札から、どうやったら勝てるかを予測する
→各世代でタイプ相性が一部異なるため、（その世代で）強いタイプをどのタイミングで出すか？

データセット

```
const char* type_names_1[15] =
{
"normal","fire","water","electric","grass","ice","fighting","poison","ground","flying","psychic","bug","rock","ghost","dragon"
};

const char* type_names_2[17] =
{
"normal","fire","water","electric","grass","ice","fighting","poison","ground","flying","psychic","bug","rock","ghost","dragon","dark","steel"
};

const char* type_names_3[18] =
{
"normal","fire","water","electric","grass","ice","fighting","poison","ground","flying","psychic","bug","rock","ghost","dragon","dark","steel","fairy"
};
```

←各世代のタイプ名

各世代の得点表
（タイプ相性表）
↓

```
const double type_char_1[15][15] =
{
    {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.5,0.0,1.0},
    {1.0,0.5,0.5,1.0,2.0,2.0,1.0,1.0,1.0,1.0,1.0,2.0,0.5,1.0,0.5},
    {1.0,2.0,0.5,1.0,0.5,1.0,1.0,1.0,2.0,1.0,1.0,1.0,2.0,1.0,0.5},
    {1.0,1.0,2.0,0.5,0.5,1.0,1.0,1.0,0.0,2.0,1.0,1.0,1.0,1.0,0.5},
    {1.0,0.5,2.0,1.0,0.5,1.0,1.0,0.5,2.0,0.5,1.0,0.5,2.0,1.0,0.5},
    {1.0,1.0,0.5,1.0,2.0,0.5,1.0,1.0,2.0,2.0,1.0,1.0,1.0,1.0,2.0},
    {2.0,1.0,1.0,1.0,1.0,2.0,1.0,0.5,1.0,0.5,0.5,0.5,2.0,0.0,1.0},
    {1.0,1.0,1.0,1.0,2.0,1.0,1.0,0.5,0.5,1.0,1.0,2.0,0.5,0.5,1.0},
    {1.0,2.0,1.0,2.0,0.5,1.0,1.0,2.0,1.0,0.0,1.0,0.5,2.0,1.0,1.0},
    {1.0,1.0,1.0,0.5,2.0,1.0,2.0,1.0,1.0,1.0,1.0,2.0,0.5,1.0,1.0},
    {1.0,1.0,1.0,1.0,1.0,1.0,2.0,2.0,1.0,1.0,0.5,1.0,1.0,1.0,1.0},
    {1.0,0.5,1.0,1.0,2.0,1.0,0.5,2.0,1.0,0.5,2.0,1.0,1.0,0.5,1.0},
    {1.0,2.0,1.0,1.0,1.0,2.0,0.5,1.0,0.5,2.0,1.0,2.0,1.0,1.0,1.0},
    {0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0,2.0,1.0},
    {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0}
};
```

```
const double type_char_2[17][17] =
{
    {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.5,0.0,1.0,1.0,0.5},
    {1.0,0.5,0.5,1.0,2.0,2.0,1.0,1.0,1.0,1.0,1.0,2.0,0.5,1.0,0.5,1.0,2.0},
    {1.0,2.0,0.5,1.0,0.5,1.0,1.0,1.0,2.0,1.0,1.0,1.0,2.0,1.0,0.5,1.0,1.0},
    {1.0,1.0,2.0,0.5,0.5,1.0,1.0,1.0,0.0,2.0,1.0,1.0,1.0,1.0,0.5,1.0,1.0},
    {1.0,0.5,2.0,1.0,0.5,1.0,1.0,0.5,2.0,0.5,1.0,0.5,2.0,1.0,0.5,1.0,0.5},
    {1.0,0.5,0.5,1.0,2.0,0.5,1.0,1.0,2.0,2.0,1.0,1.0,1.0,1.0,2.0,1.0,0.5},
    {2.0,1.0,1.0,1.0,1.0,2.0,1.0,0.5,1.0,0.5,0.5,0.5,2.0,0.0,1.0,2.0,2.0},
    {1.0,1.0,1.0,1.0,2.0,1.0,1.0,0.5,0.5,1.0,1.0,1.0,0.5,0.5,1.0,1.0,0.0},
    {1.0,2.0,1.0,2.0,0.5,1.0,1.0,2.0,1.0,0.0,1.0,0.5,2.0,1.0,1.0,1.0,2.0},
    {1.0,1.0,1.0,0.5,2.0,1.0,2.0,1.0,1.0,1.0,1.0,2.0,0.5,1.0,1.0,1.0,0.5},
    {1.0,1.0,1.0,1.0,1.0,1.0,2.0,2.0,1.0,1.0,0.5,1.0,1.0,1.0,1.0,0.0,0.5},
    {1.0,0.5,1.0,1.0,2.0,1.0,0.5,1.0,1.0,0.5,2.0,1.0,1.0,0.5,1.0,2.0,0.5},
    {1.0,2.0,1.0,1.0,1.0,2.0,0.5,1.0,0.5,2.0,1.0,2.0,1.0,1.0,1.0,1.0,0.5},
    {0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,2.0,1.0,0.5,0.5},
    {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,0.5},
    {1.0,1.0,1.0,1.0,1.0,1.0,0.5,1.0,1.0,1.0,2.0,1.0,1.0,2.0,1.0,0.5,0.5},
    {1.0,0.5,0.5,0.5,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,1.0,0.5}
};
```

```
const double type_char_3[18][18] =
{
    {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,0.5,0.0,1.0,1.0,0.5,1.0},
    {1.0,0.5,0.5,1.0,2.0,2.0,1.0,1.0,1.0,1.0,1.0,2.0,0.5,1.0,0.5,1.0,2.0,1.0},
    {1.0,2.0,0.5,1.0,0.5,1.0,1.0,1.0,2.0,1.0,1.0,1.0,2.0,1.0,0.5,1.0,1.0,1.0},
    {1.0,1.0,2.0,0.5,0.5,1.0,1.0,1.0,0.0,2.0,1.0,1.0,1.0,1.0,0.5,1.0,1.0,1.0},
    {1.0,0.5,2.0,1.0,0.5,1.0,1.0,0.5,2.0,0.5,1.0,0.5,2.0,1.0,0.5,1.0,0.5,1.0},
    {1.0,0.5,0.5,1.0,2.0,0.5,1.0,1.0,2.0,2.0,1.0,1.0,1.0,1.0,2.0,1.0,0.5,1.0},
    {2.0,1.0,1.0,1.0,1.0,2.0,1.0,0.5,1.0,0.5,0.5,0.5,2.0,0.0,1.0,2.0,2.0,0.5},
    {1.0,1.0,1.0,1.0,2.0,1.0,1.0,0.5,0.5,1.0,1.0,1.0,0.5,0.5,1.0,1.0,0.0,2.0},
    {1.0,2.0,1.0,2.0,0.5,1.0,1.0,2.0,1.0,0.0,1.0,0.5,2.0,1.0,1.0,1.0,2.0,1.0},
    {1.0,1.0,1.0,0.5,2.0,1.0,2.0,1.0,1.0,1.0,1.0,2.0,0.5,1.0,1.0,1.0,0.5,1.0},
    {1.0,1.0,1.0,1.0,1.0,1.0,2.0,2.0,1.0,1.0,0.5,1.0,1.0,1.0,1.0,0.0,0.5,1.0},
    {1.0,0.5,1.0,1.0,2.0,1.0,0.5,1.0,1.0,0.5,2.0,1.0,1.0,0.5,1.0,2.0,0.5,0.5},
    {1.0,2.0,1.0,1.0,1.0,2.0,0.5,1.0,0.5,2.0,1.0,2.0,1.0,1.0,1.0,1.0,0.5,1.0},
    {0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,2.0,1.0,0.5,1.0,1.0},
    {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,0.5,0.0},
    {1.0,1.0,1.0,1.0,1.0,1.0,0.5,1.0,1.0,1.0,2.0,1.0,1.0,2.0,1.0,0.5,1.0,1.0},
    {1.0,0.5,0.5,0.5,1.0,2.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0,1.0,1.0,1.0,0.5,2.0},
    {1.0,0.5,1.0,1.0,1.0,1.0,2.0,0.5,1.0,1.0,1.0,1.0,1.0,1.0,2.0,2.0,0.5,1.0}
};
```

select_version関数

```c
int select_version(){
    int n;
    while(1){
        printf("\nどのバージョンで遊ぶ？\n");
        printf("1：初代ポケモン\n");
        printf("2：第2世代～第5世代ポケモン\n");
        printf("3：第6世代～第9世代ポケモン\n");
        printf("\n1~3を入力して選んでね！：");
        if(scanf("%d",&n) != 1) {
            printf("[エラー]：1~3の数字のみ入力してください！\n");
            while(getchar() != '\n');
            continue;
        }
        if(n == 1 || n == 2 || n == 3) return n;
        printf("[エラー]：1~3の数字を入力してください！\n");
    }
}
```

プレイヤーにバージョン選択を要求する関数

- 機能①：１～３の数字の入力を求める

→例えば、1を入力すると「初代ポケモン」のタイプ相性でプレイすることができる

- 機能②：１～３以外の入力に対してはエラー文を出力

→正しい入力がされるまで、繰り返し入力を求める

player_choice関数

```c
int player_choice(const char** type_names, int type_count, int* flag){

    while(1) {
        printf("\n[いまあなたが持っている手札]:\n");
            for(int i = 0; i < type_count; i++){
                if(flag[i] != 1) {
                    printf("<");
                    printf("%s",type_names[i]);
                    printf(">");
                    printf(",");
                }
            }

            printf("\n\nあなたのタイプ：");
            char type_input[100];
            scanf("%s",type_input);

            int player_index = -1;
            for(int j = 0; j < type_count; j++){
                if(strcmp(type_input, type_names[j]) == 0){
                    player_index = j;
                    break;
                }
            }
            if(player_index == -1){
                printf("[エラー]：タイプ名が正しくありません！\n");
                continue;
            }
            if(flag[player_index] == 1){
                printf("[エラー]：そのタイプをあなたは既に使用しています！\n");
                continue;
            }
            flag[player_index] = 1;
            return player_index;
        }
    }
```

プレイヤーにタイプ選択を要求する関数

- 機能①：現在の残りの手札を表示する
→初代でも15タイプあり覚えるのが大変なため、ラウンド毎に残りの手札を表示させる

- 機能②：タイプの入力を要求
→入力したタイプはそのラウンドで表示する

- 機能②：タイプ名以外の入力に対してエラー文を出力
→タイプミス等をしてしまっても、再度入力が可能

- 機能③：使用済みタイプの入力に対しても同様

- 機能④：入力したタイプを使用済み判定に更新する
→配列flagを更新する

cpu_choice関数

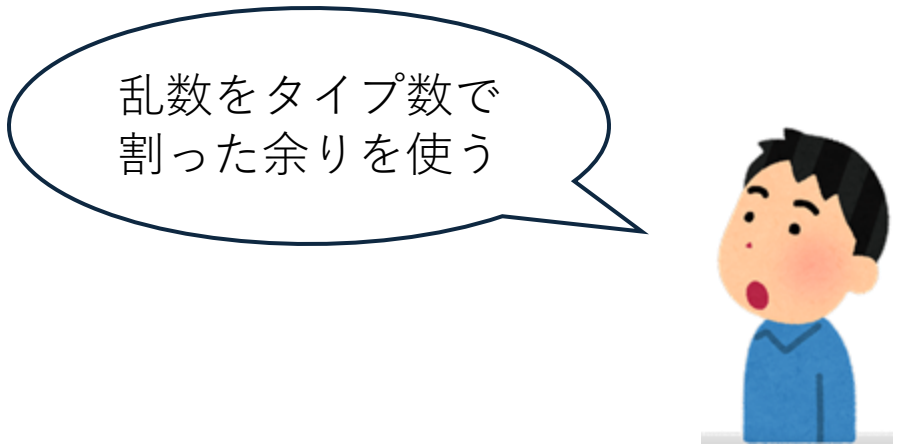
```c
int cpu_choice(const char** type_names, int type_count, int* flag){

    while(1) {
        int cpu_index = rand() % type_count;
        if(flag[cpu_index] == 0) {
            flag[cpu_index] = 1;
            printf("相手が使ったタイプ：%s\n", type_names[cpu_index]);
            return cpu_index;
        }
    }
}
```

CPUがタイプを選択するための関数

- 機能①：タイプの選択を行う

→CPU側は乱数を生成し、未使用のタイプからランダムに1つ選択する

→選択したタイプはそのラウンドで表示させる



point_calculation関数

```
double point_calculation(int attack, int defense, const double* chart, int type_count){
    return chart[attack * type_count + defense];
}
```

得点計算を行う関数

- 機能①：得点の計算を行う

→2次元配列のタイプ相性表（得点表）から、得点計算に使う要素を取り出す

→配列の引数は、「行の番号×列の数＋列の番号」で算出

→得点表の行は「攻撃側のタイプ」、列は「防御側のタイプ」を表す

round_result関数

```c
void round_result(int round, double player_score, double cpu_score) {

    printf("\n---------------現在のスコア---------------\n");
    printf("あなた：%.1f    |    相手：%.1f\n", player_score, cpu_score);
    printf("-----------------------------------------\n");

    if(player_score > cpu_score)
    printf("--- あなたが%.1f点リードしています！ ---\n", player_score - cpu_score);
    else if(player_score < cpu_score)
    printf("--- 相手が%.1f点リードしています、、、 ---\n", cpu_score - player_score);
    else
    printf("--- いまは引き分けです ---\n");

}
```

各ラウンドの結果を表示する関数

- 機能①：現在のスコアを表示する

→各ラウンド終了時に現時点でのスコアを表示する

- 機能②：合計得点が優勢な方を表示する

play_game関数（前半部分）

```
void play_game(int version) {

    const double* chart;
    const char** type_names;
    int type_count;
    double point;
    double player_score = 0.0;
    double cpu_score = 0.0;

    switch(version) {
        case 1:
        type_names = type_names_1;
        chart = &type_char_1[0][0];
        type_count = type_count_1;
        printf("\n<<<初代ポケモンのカードバトル!>>>\n");
        break;

        case 2:
        type_names = type_names_2;
        chart = &type_char_2[0][0];
        type_count = type_count_2;
        printf("\n<<<第2世代～第5世代のポケモンカードバトル!>>>\n");
        break;

        case 3:
        type_names = type_names_3;
        chart = &type_char_3[0][0];
        type_count = type_count_3;
        printf("\n<<<第6世代～第9世代のポケモンカードバトル!>>>\n");
        break;

        default:
        printf("\nエラー：バージョン不明\n");
    }
```

ゲームの設定を担う関数

- 機能⓪：変数の用意

→タイプ名、得点表、タイプ数の変数を用意
→初期得点は0点

- 機能①：バージョン毎に使用するデータを選択する

→使用するタイプ名、得点表、タイプ数を決定する

（例）バージョン1が選択された場合：
初代ポケモンのタイプ名セット（normal～dragonまで）
初代ポケモンのタイプ相性表（得点表）
初代ポケモンのタイプ数（15）

が変数に格納される

play_game関数（後半部分）

```
    srand(time(NULL));
    int coin_toss = rand() % 2;
    if(coin_toss == 1) printf("\n---あなたは先攻（攻撃側）です！---\n");
    else printf("\n---あなたは後攻（防御側）です！---\n");

    for(int round = 0; round < type_count; round++) {

        printf("\n===第 %d ラウンド===\n",round + 1);

        int player_index = player_choice(type_names, type_count, player_flag);
        int cpu_index = cpu_choice(type_names, type_count, cpu_flag);
        int turn_decide = (round + coin_toss) % 2;

        if(turn_decide == 1) {
            printf("\n---あなたの攻撃！---\n");
            point = point_calculation(player_index, cpu_index, chart, type_count);
            player_score += point;
            printf("\nあなたは%.lf点獲得！\n",point);
            round_result(round, player_score, cpu_score);
            if(round < type_count - 1) {
            printf("\n\n<<<次のターンは防御側です！>>>\n\n");
            }
        } else {
            printf("\n---相手の攻撃！---\n");
            point =  point_calculation(cpu_index, player_index, chart, type_count);
            cpu_score += point;
            printf("\n相手は%.lf点獲得！\n",point);
            round_result(round, player_score, cpu_score);
            if(round < type_count - 1) {
            printf("\n\n<<<次のターンは攻撃側です！>>>\n\n");
        }
    }
}

    if(player_score > cpu_score)
    printf("\n---あなたの勝ち！！！---\n");
    else if(player_score < cpu_score)
    printf("\n---あなたの負け、、、---\n");
    else
    printf("\n---引き分け---\n");
```

ゲームの設定を担う関数

- 機能②：先攻と後攻を決定する（coin_toss）

→初回のラウンドは乱数を生成して決定するため、結果はランダム
→2回目以降は自動で決定する（先攻や後攻は連続しない）

- 機能③：タイプ数に応じてラウンドを実行する

→手札が切れるまでラウンドを続ける
→point_calculationを呼び出して得点を計算
→計算結果を各スコアに加算

- 機能④：最終結果の表示

→プレイヤーとCPUの得点を比べて高い方が勝ち

メイン関数

```c
int main() {

    while(1) {
    int version = select_version();

    memset(player_flag, 0, sizeof(player_flag));
    memset(cpu_flag, 0, sizeof(cpu_flag));

    printf("\n--------------------------------------------------------------------------\n");
    printf("[ゲームの説明1]：各ラウンドで手札の中からタイプを1つ選択してください。\n");
    printf("[ゲームの説明2]：一度使用したタイプは使えません！よく考えて使いましょう！\n");
    printf("[ゲームの説明3]：各世代のタイプ相性表をもとに、CPUと対戦します\n");
    printf("[ゲームの説明4]：最終ラウンドで合計得点が多い方が勝ちです！\n");
    printf("--------------------------------------------------------------------------\n");

    play_game(version);

    printf("もう一度遊びますか？():");
    char again[10];
    scanf("%s",again);
    if(again[0] != 'y' && again[0] != 'Y') {
        printf("ゲームを終了します\n");
        break;
    }
    }
    return 0;

}
```

機能①：select_versionの呼び出し

機能②：2つ配列（player_flag、cpu_flag）の初期化
→連続でゲームを続けることが可能

機能③：ゲーム開始時のルール説明
→プレイヤーに向けたルール説明をゲーム開始時に表示

機能④：play_gameの呼び出し

機能⑤：連続でゲームをプレイするかを確認
→連続でプレイする場合、世代選択画面に戻る

実行例①

```
(base) rikuli_35@s-nice99 ゲーム制作 % gcc pokemon_card_unf.c -o test
(base) rikuli_35@s-nice99 ゲーム制作 % ./test

どのバージョンで遊ぶ？
1：初代ポケモン
2：第2世代～第5世代ポケモン
3：第6世代～第9世代ポケモン

1~3を入力して選んでね！：
```

←testというファイルを作成して実行

←最初にバージョン選択画面が表示されます
（今回はバージョン１を選択します）

```
どのバージョンで遊ぶ？
1：初代ポケモン
2：第2世代～第5世代ポケモン
3：第6世代～第9世代ポケモン

1~3を入力して選んでね！：0
[エラー]：1～3の数字を入力してください！

どのバージョンで遊ぶ？
1：初代ポケモン
2：第2世代～第5世代ポケモン
3：第6世代～第9世代ポケモン

1~3を入力して選んでね！：
```

←正しくない入力には再度要求

実行例②

```
1~3を入力して選んでね！： 1

----------------------------------------------------------------
[ゲームの説明1]：各ラウンドで手札の中からタイプを1つ選択してください。
[ゲームの説明2]：一度使用したタイプは使えません！よく考えて使いましょう！
[ゲームの説明3]：各世代のタイプ相性表をもとに、CPUと対戦します
[ゲームの説明4]：最終ラウンドで合計得点が多い方が勝ちです！
----------------------------------------------------------------

<<<初代ポケモンのカードバトル!>>>

---あなたは後攻（防御側）です！---

===第 1 ラウンド===

[いまあなたが持っている手札]:
<normal>,<fire>,<water>,<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：
```

←ゲームの説明を表示

←最初のラウンドが開始
（今回は後攻からスタート）

←現在の手札を表示

↑防御で使用するタイプ名を入力(今回はnormalを使用)

```
あなたのタイプ：normall
[エラー]：タイプ名が正しくありません！

[いまあなたが持っている手札]:
<normal>,<fire>,<water>,<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：
```

←正しくない入力には再度要求

実行例③

```
あなたのタイプ：normal
相手が使ったタイプ：fire

---相手の攻撃！---

相手は1点獲得！

---------------現在のスコア---------------
あなた：0.0   |   相手：1.0
----------------------------------------
--- 相手が1.0点リードしています、、、 ---


<<<次のターンは攻撃側です！>>>


===第 2 ラウンド===

[いまあなたが持っている手札]:
<fire>,<water>,<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：
```

←CPUがタイプを選択
（今回はfireを選択）

←得点表をもとにCPUが点数獲得
←現在の総合スコアを表示
←優勢な方を表示

←次は攻撃側

↑攻撃で使用するタイプを入力(先ほど使用したnormalは手札に無い)

```
[いまあなたが持っている手札]:
<fire>,<water>,<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：normal
[エラー]：そのタイプをあなたは既に使用しています！
```

←使用済みの手札の再使用は不可

実行例④

```
あなたのタイプ：fire
相手が使ったタイプ：ground

---あなたの攻撃！---

あなたは1点獲得！

----------------現在のスコア----------------
あなた：1.0   |   相手：1.0
----------------------------------------
--- いまは引き分けです ---


<<<次のターンは防御側です！>>>


===第 3 ラウンド===

[いまあなたが持っている手札]:
<water>,<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：
```

←今回はfireを選択

←1ラウンド目と同様の処理

←スコアの更新

←次は防御側

以降のラウンドはCPUと攻守を交代しながら手札を消費していく

実行例⑤

```
<<<次のターンは防御側です！>>>


===第 15 ラウンド===

[いまあなたが持っている手札]:
<dragon>,

あなたのタイプ：dragon
相手が使ったタイプ：psychic

---相手の攻撃！---

相手は1点獲得！

----------------現在のスコア----------------
あなた：11.0   |   相手：8.5
-----------------------------------------
--- あなたが2.5点リードしています！ ---

---あなたの勝ち！！！---
もう一度遊びますか？()：
```

←最後のラウンド

色んなバージョンを
連続でプレイできる！

←最終スコアが高い方が勝ち

↑”Yes”を入力するとバージョン選択画面に戻る→

```
---あなたの勝ち！！！---
もう一度遊びますか？()：Yes

どのバージョンで遊ぶ？
1：初代ポケモン
2：第2世代～第5世代ポケモン
3：第6世代～第9世代ポケモン

1~3を入力して選んでね！：
```

改善点①（バグ）

```
===第 2 ラウンド===

[いまあなたが持っている手札]:
<fire>,<water>,<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：fire
相手が使ったタイプ：flying
```

Fire	Flying	1

```
---あなたの攻撃！---

あなたは1点獲得！

---------------現在のスコア---------------
あなた：1.0   |   相手：1.0
-----------------------------------------
--- いまは引き分けです ---


<<<次のターンは防御側です！>>>


===第 3 ラウンド===

[いまあなたが持っている手札]:
<water>,<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：water
相手が使ったタイプ：fire
```

Fire	Water	0.5

```
---相手の攻撃！---

相手は0点獲得！

---------------現在のスコア---------------
あなた：1.0   |   相手：1.5
-----------------------------------------
--- 相手が0.5点リードしています、、、 ---
```

←2ラウンド目では
1.0 vs 1.0

←3ラウンド目では相手が0.5点獲得
1.0 vs 1.5　になるはず

←0点獲得と表示されてしまう！

←でもスコアは問題ない？？？

改善点②（バグ）

```
===第 4 ラウンド===

[いまあなたが持っている手札]:
<electric>,<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：electric
相手が使ったタイプ：fighting
```

Electric	Fighting	1

```
---あなたの攻撃！---

あなたは1点獲得！

----------現在のスコア----------
あなた：2.0   |   相手：1.5
----------------------------------
--- あなたが0.5点リードしています！ ---


<<<次のターンは防御側です！>>>


===第 5 ラウンド===

[いまあなたが持っている手札]:
<grass>,<ice>,<fighting>,<poison>,<ground>,<flying>,<psychic>,<bug>,<rock>,<ghost>,<dragon>,

あなたのタイプ：grass
相手が使ったタイプ：dragon
```

Dragon	Grass	1

```
---相手の攻撃！---

相手は0点獲得！

----------現在のスコア----------
あなた：2.0   |   相手：1.5
----------------------------------
--- あなたが0.5点リードしています！ ---
```

←4ラウンド目では
　正しくスコアが更新されている

←5ラウンド目では相手が0.5点獲得
　2.0 vs 2.5　になるはず

←表記もスコアも正しく更新できていない！

今後の展望

まずは、現状のバグを解消しなければならない、、、

さらに面白くするために、データ分析プロジェクトの結果をもとに以下を実装したいと考えています。

- ラウンド数を（偶数に）絞る

→タイプ数が奇数だと先攻が有利になってしまう
→実行すると１ゲームが長くなってしまい、飽きてしまう
→バージョン毎の各タイプの強さを鑑みて調整する必要がある

- プレイヤーがゲーム開始前に「自分で考えたタイプを追加」できるようにする

→新しいタイプにはある程度の制限をつける（強すぎると面白みが欠けてしまうため）
→世代選択後に得点表を更新する処理が必要