ゲーム制作プロジェクト
解説資料

対NPCのタイプ相性カードバトル
～デモ版から改良版へ向けた設計思想～

名古屋大学 大学院　情報学研究科　複雑系科学専攻

修士1年　鈴木琳久

デモ版・改良版の位置付け

制作動機

デモ版

（2025年5月時点での）自分の力でゲームを作ってみたい！

・大学時代に学んでいたC言語での実装

・企画考案→実装→解説資料作成を全て一人で制作

改良版

C++を学んで得た知識を使ってもっとクオリティの高いゲームを作りたい！

・C++は独学（競技プログラミングの参加等で練習）

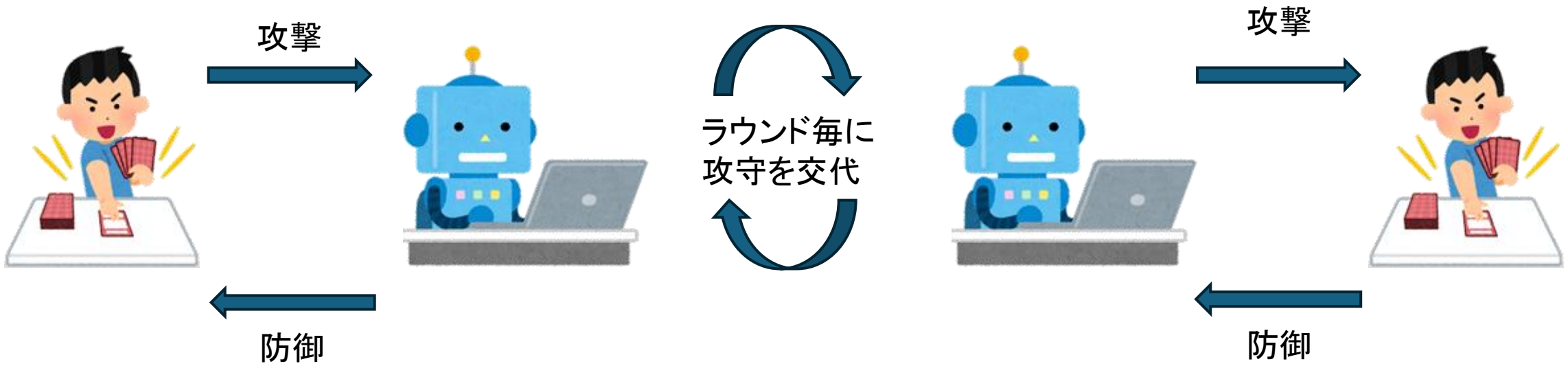
・現時点で実行は不可（現在はUIを製作中）

ゲームの基本構造(共通)

①:「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が手札を使って「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



ゲームの面白さ（共通）

ゲームの醍醐味は、相手との「駆け引き」

相手に勝つためには、、、

→残りの手札から、どうやったら勝てるかを予測する

→強いタイプをどのタイミングで出すか？

実行例(デモ版)

```
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：
```

↑
ラウンドの様子

```
<<<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？()：
```

↑
最終結果の様子

実行できる状態ではあるが、ゲームとして改善すべき点が多い！

問題点と改善方針

＜デモ版の問題点と原因＞

①手札が分かりづらい & ゲームっぽくない

→カードゲームなのにカードっぽくない

②試合が長い & 先攻有利問題

→手札の枚数＝ラウンド数が原因

→ラウンド数＝奇数だと明らかに先攻有利

③単調な試合がずっと続いてしまう

→戦略性が十分に機能していない

④得点計算に一部バグがある

→得点表の手入力が原因？

＜改良後＞

❶UIの実装

→フィールド、デッキ、カードなどの描画を追加してゲームっぽさの演出

❷ラウンド数を10に固定

→ラウンド数は偶数かつ試合が長すぎないような値

❸ゲームルールの変更

→山札を追加(手札の上限は5枚まで)

→手札の2枚同時使用の追加（ハイリスク・ハイリターンな戦略）

→ラウンド効果による運要素の追加（得点2倍やタイプ相性逆転）

→ラウンド開始時にドローorハンデスの選択権を追加

❹CSVファイル読み込みによる計算バグの防止

→ファイルはデータ分析プロジェクトのものを使用

改良版での狙い

選択肢を増やし、プレイヤーの判断が結果に反映される設計

相手に勝つためには、、、

→手札を2枚使用して一気に攻める

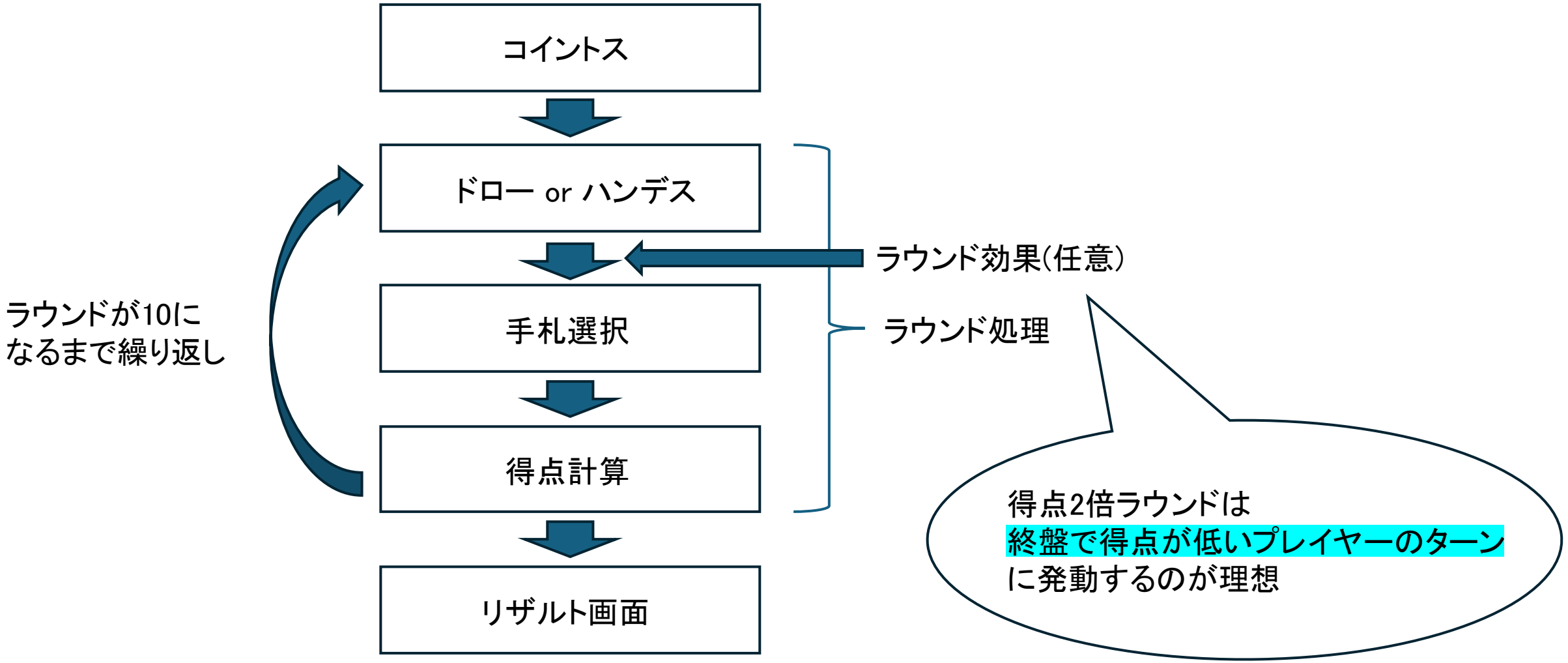
→ハンデスを選択して相手を妨害する

→最後の運要素で逆転を狙う

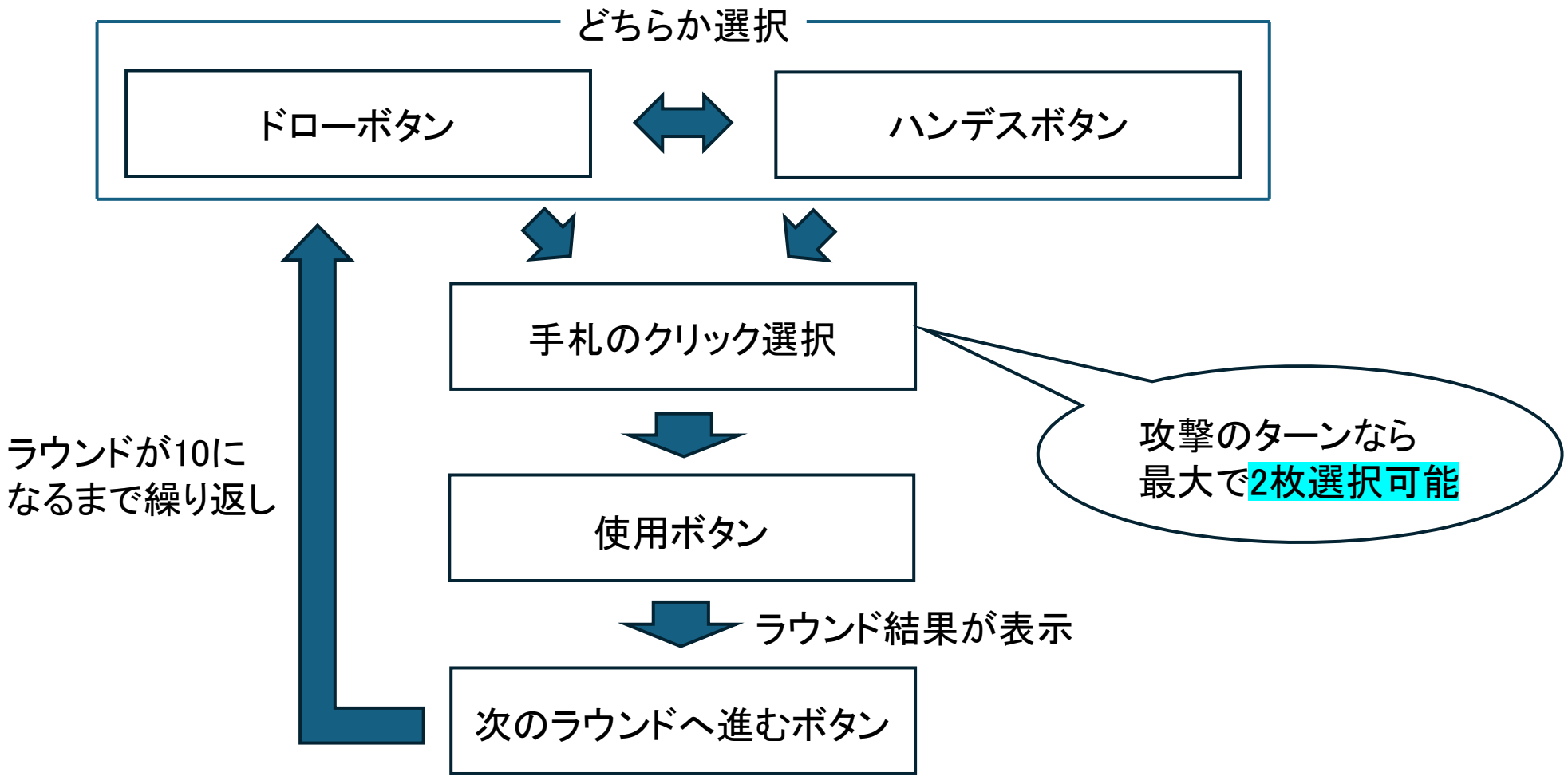
etc.

プレイヤーの選択の幅が広い
↓
さらなる駆け引きを生み出す
↓
より面白いゲームに！

改良版でのゲームフロー



改良版でのUIフロー


どちらか選択
ドローボタン
ハンデスボタン
手札のクリック選択
攻撃のターンなら
最大で2枚選択可能
使用ボタン
ラウンド結果が表示
次のラウンドへ進むボタン
ラウンドが10に
なるまで繰り返し

ゲームロジックとUIの住み分け

game.cpp

```
#include "game.h"

using namespace std;

// タイプ名一覧
vector<string> type_list =
{

// タイプ数変数
int type_count = 19;

// 得点表の変数
vector<vector<double>> PointGraph;

// ファイル読み込みを行う関数
vector<vector<double>> ReadCSV(const string& filename) {

// バージョン読み込みを行う関数
void LoadVersion(GameVersion version, GameData& game) {

// ゲームの初期状態
void InitGame(GameData& game) {

// ドローを行う関数
void HandDraw(GameData& game, PlayerID id) {

// ハンデスを行う関数（
void HandDeath(GameData& game, PlayerID id) {

// プレイヤーが手札を選択する関数
void PlayerSelect(GameData& game, int index, int count) {

// CPUが手札を選択する関数
void CpuSelect(GameData& game, int count) {

// 得点計算を行う関数
double RoundPoint(const vector<int> &attack, int defense) {

// ラウンド終了後の手札処理を行う関数
void RoundManagement(GameData& game) {

// ラウンド開始
void RoundStart(GameData& game) {
```

game.h

```
#pragma once
#include <vector>

constexpr int TYPE_COUNT = 19;
constexpr int MAX_ROUNDS = 10;

// プレイヤータグ
enum class PlayerID { …

// バージョンタグ
enum class GameVersion {…

// ゲームの状態管理
enum class GameState {…

// プレイヤーデータ
struct Player {…

// ゲーム全体データ
struct GameData {…

// 関数の宣言（準備・初期化）
void LoadVersion(GameVersion version, GameData& game);
void InitGame(GameData& game);

// 関数の宣言（ラウンド処理）
void RoundStart(GameData& game);
void RoundEnd(GameData& game);
bool IsRoundFinished(const GameData& game);

// 関数の宣言（プレイヤー操作）
void HandDraw(GameData& game, PlayerID id);
void HandDeath(GameData& game, PlayerID id);
void PlayerSelect(GameData& game, int index, int count);

// 関数の宣言（CPU操作）
void CpuSelect(GameData& game, int count);
```

ゲームロジック部分

ui.cpp

```
#include "ui.h"

// マウス座標の用意
static Vector2 mousePoint;

// システムボタン一覧
HitBox start_Button;
HitBox game_explanation_Button;
HitBox type_explanation_Button;

// チャート一覧
UIButton title_chart;
UIButton back_ground;

/*
// 手札
Hand hand;
*/

// システムボタン・チャートを作成する関数
UIButton CreateSystemButton(const char* filename, float x, float y, float scale) {

// 透明ボタンを作成する関数
HitBox CreateHitBox(float x, float y, float w, float h) {

/*
// タイプボタンを作成する関数
UIButton CreateCardButton(const char* filename) {

    UIButton card;
    card.tex = LoadTexture(filename);
    card.bounds = (Rectangle){0, 0, (float)card.tex.width * 0.5f, (float)card.tex.height * 0.5f};

    return card;

}
*/

// ボタンを描画する関数
void DrawButton(const UIButton& btn) {

// チャートを描画する関数
void DrawChart(UIButton btn) {
```

ui.h

```
#pragma once
#include "raylib.h"
#include <vector>
#include "game.h"


enum class HandView {…

// 手札座標の構造体
struct Hand{…

// 画像挿入の構造体
struct UIButton{…

// 透明ボタンの当たり判定
struct HitBox { Rectangle bounds; };

// 手札描画用配列・変数
static UIButton typeButtons[TYPE_COUNT];
static UIButton cardback;

// 画面状態の管理
enum class UIState {…

// UIの発生イベント
enum class UIEvent {…

// 関数の宣言
void InitResources();
void UnloadResources();
UIEvent UpdateUI(UIState state);
void DrawUI(UIState ui, GameData& game);
```

UI部分

仕様変更に対応できるような設計を意識

現在行っている取り組み・課題

- デザインの知識がなくてもゲームっぽい画面を作りたい！

→生成系AIにゲーム内画像を生成させる試み

→画像生成時に文字化けしてしまうため、生成された画像に文字のフォントを後付けする作業が必要

- UIの遷移が上手く動作しないバグが発生している、、、

→ゲームタイトル画面にゲーム内で使用するはずの画像が表示されてしまう

→戻るボタンを押しても画面が戻らない