Texas __ Hold __ em.c 解説

創価大学 4 年生 堀田 祐輝

2025 年 7 月 8 日

1　制作動機

今まで簡単な構成のプログラムしか作成したことがなく，今後のレベルアップを兼ねて C の復習を行いたいと考えていました.
そこで，複数のゲームが存在するプログラムを作成することで復習と C 言語のライブラリについてよく知ろうとしました.
その中の 1 つとしてテキサスホームデムを作成し，思いのほかコードが大型化したため，分離して一つのゲームとして作成しました.

2　使用ツール

使用言語は C で，理由としては，今後のレベルアップを兼ねて大学の講義で学習した C を復習したいと考えたためです.
開発環境は Visual Studio Code です．理由としては，プログラミングを行う際に最も使われている環境であると聞き，その環境に慣れておくことが最善であると考えたためです.
実行環境は Windows10 で，理由としては GhostBreaker と同じで，手元にあるデバイスが Windows10 であったためです.

3　ゲーム実行方法

3.1　Cygwin を用いた実行方法

まず，Cygwin を起動し，ExecuteFolder（実行ファイル）にある Texas __ Hold __ em.c のフォルダまで Change Directory（cd）をします.
その後，”gcc Texsas __ Hold __ em.c” と入力し，”./a” と入力すると実行できます.
画像 1 は実際に cygwin で実行している画像です.

3.2　CommandPrompt を用いた実行方法

コマンドプロンプトを起動し，ExecuteFolder（実行ファイル）にある Texas __ Hold __ em.c のフォルダまで Change Directory（cd）をします.
その後”Texas __ Hold __ em.exe” と入力すると実行できます.
画像 2 は実際に CommandPrompt で実行している画像です.

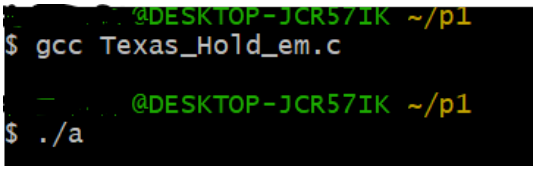

図 1　cygwin での実行方法

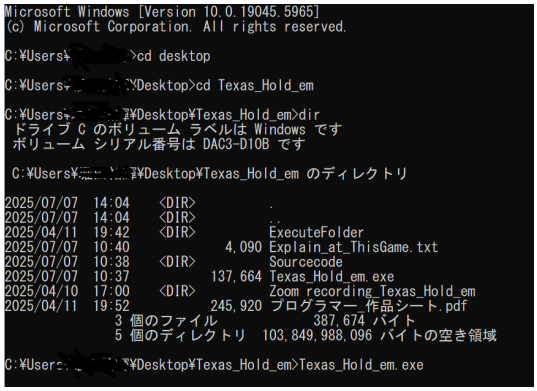

図 2　CommandPrompt での実行方法

4　ゲーム内容説明

4.1　概要

プレイヤー (操作する人) と bot3 人で全 5 ラウンドのポーカーを行い、全てのプレイヤーより多くチップを獲得することで勝利できるゲームです。
アメリカのテキサスホームデムのルールを参考にしたゲームで、親決めのルールやカードの配布方法をテキサスホームデムから踏襲しています。

1 ラウンドにつき 3 フェーズで構成されていて、それぞれ全員が同じチップを賭ける「コール」、賭けるチップ数の倍のチップ数を賭ける「レイズ」、賭けるのをやめてゲームを降りる「フォールド」を行うことでフェーズを進めることが出来ます。
フェーズが進むごとに場のカードが一枚ずつ公開されていき、最後のフェーズが終わると、自分の手札と場のカードを使ってカードの役を作ります。
この時に一番強い役を作れた人がそのラウンドで賭けられたチップを全てもらうことが出来ます。
画像 3 はゲームプレイ画面です．画像 3 の状態では自分のターンでどの行動を行うか数字を入力することで決めます．

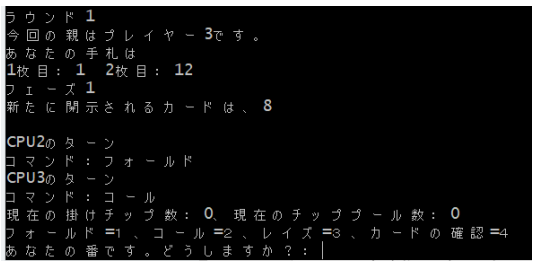

図 3　ゲームプレイ画面

4.2　細かい仕様

本ゲームはプレイヤーは常に数字を入力することで行動を決定する仕様の他に，フォールドしたプレイヤーのターンを飛ばすようにしています．画像 4 はフォールドしたプレイヤーが飛ばされている状況の画像です．

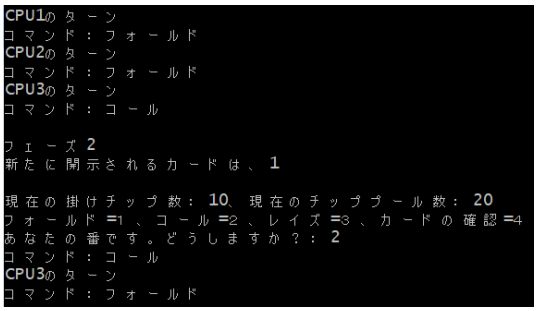

図 4　フォールドしたプレイヤーを飛ばす

5　アピールポイント (工夫した点)

この作品の一番のアピールポイントは状況を見て判断するような思考能力を bot に導入したことです．
導入した理由としては，完全ランダムによる判断ではゲームとして成り立ちにくいと考えたためです．
そのために工夫した点はまずバグや不正がないことは前提として、「相手の下した判断」「自分の手札の組み合わせ」「自分の手札と開示されてないカードの組み合わせ」
この 3 つの要素をそれぞれ点数付けして，合計点数ごとに閾値を設定して，フォールドするか，コールするか，レイズするか CPU に判断させました．
また，数字しか入力を受け付けないため，提示された数字以外が入力された場合はエラーを返すようにしました．

6　苦戦した点

最も苦戦した箇所は，デバッグ作業でのエラー箇所の発見です．
アルゴリズムの構成としては，一ラウンドの流れを記述して for 文で繰り返すという構成であるため，segmentation error(core dunped) と表示され，どこがダメなのかわからないという問題が頻発しました．
そこで，大学の友人や先輩と共に一つ一つ printf などを用いてエラーチェックを行ってエラー箇所を探し，エラーをなくしていくことで解決しました．この経験から，タスクが多い作業ややりなおすのが大変な作業ほど，ごまかしたり横着したりせず，丁寧にやることと，心の底から反省しました．具体的には，デバッグ作業は一気にやるのではなく，個別でファイルを作成して動かしてみる，header を用いて視認性を良くすることで，人力でのエラーチェックをしやすくすること，関数がより高い独立性を持つような構成を意識すること等を行うことにします．

7　課題点と今後の展望

本ゲームは手札や相手の下した判断等に適当な重み付けして判断させていますが，やはり甘い計算式であり，対戦相手が簡単にフォールドをしてしまう

ことが頻発しています.
よりよい判断ができるように，さらにブラフやブラフを見抜けるような思考能力を持つように改善します.
そのために，判断できる要素を摸索し，追加したり，少し閾値を増減させるような方法を取ることを考えています.

8　本作品にて学習できた点

本ゲームの作成を通して，header の重要性やオブジェクト指向の重要性を身をもって知ることが出来ました．何故なら，本ゲームは時間をかけて作成したのですが，自分だけでコードを記述しているのにも関わらず，途中で全体の流れや構成が分からなくなってしまうことが多々ありました.
なによりテストとしてプログラムを実行した際にエラーが長文で流れてどこが問題であるか，どの箇所が原因で連鎖的にエラーが出ているのか等が非常に分かりにくいことがありました．個人開発であったため，何とかなった点ことであるため，今後の開発に向けて複数のファイルに分けたり，独立性のある関数にすることを心掛けることにしました.
また，変数名もほとんど同じような名前だと勘違いが発生したため，input __ player や for __ changeNumber など目的に沿った名前にすることを意識したいと考えました.