人工知能特論最終レポート

2018 年 9 月 14 日

チーム：F

○ M18J4043M 中島 太知

M18J4024Z 阪本 光翼

M18J4026M 佐藤 洸亮

M18J4059B 三宅 裕稀

1　ゲームの説明

1.1　概要

これは，順番に指定範囲内の数字を取り合っていき，終了点に指定された数字を踏んだら負けとなる数取りゲームを木探索を用いた AI を相手に行うものである．今回は探索を複雑にするため 3 人対戦で行い，人間の数と COM の数は任意に決められるものとしている．

1.2　ルール

- AI を含め 3 人対戦で行う．
- 終了点は任意で決められる．
- AI の強さは 3 段階で決められる．
- プレイ順はランダムで決まる．
- 1 度に取れる数値は 1 か 2 のみ．
- 最終的に終了点の数を踏んだら負け．

以上のルールでゲームを行う．

2　ゲームの定義

2.1　状態表現

今回はコンソール上でゲームを行う．よって状態表現はコンソール上に文字として表現する．表示する状態はプレイヤーあるいは AI の選択した数値およびそれによって移動した後の現在地である．また，1 ターン終了度に誰がどの数値をとったかのログを表示する．

2.2　オペレータ

AI は強さに応じた先読みを行い，より勝率が高い結果を返す．返す結果は 1 か 2 である．先読みした勝率が同等の場合，もしくは先読みで結果が読みきれない場合はランダムで 1 か 2 を返すようにする．

2.3　評価関数

前述のように，AI の強さに応じて先読みを行うことで状況判断をおこなう．今回プレイヤーの数を 3 人に設定したため，AI が勝利するにはゲーム終了値から 2 つ手前の数を取ることができれば他プレイヤーが最小値を取得しても勝利することが可能である．ゲーム終了値を 11 とした場合の AI の探索例を図 1 に示す．プレイ順がプレイヤー 1，プレイヤー 2，AI の順であるとすると．プレイヤー 2 が 5 を取得した場合，AI の選択できる数値は図のオレンジ色で示した数値である．終了値が 11 であるため AI は 9 を取得すれば良いことになる．よって図の最右端にある AI の列が 9 になれば良い．このように勝ち数が多くなる選択をすることでゲームを進行して行く．

評価関数のプログラムには再起を用いた探索を利用し，AI の強さによって探索の深さを変更する．プログラムの流れを図 2 に示す．

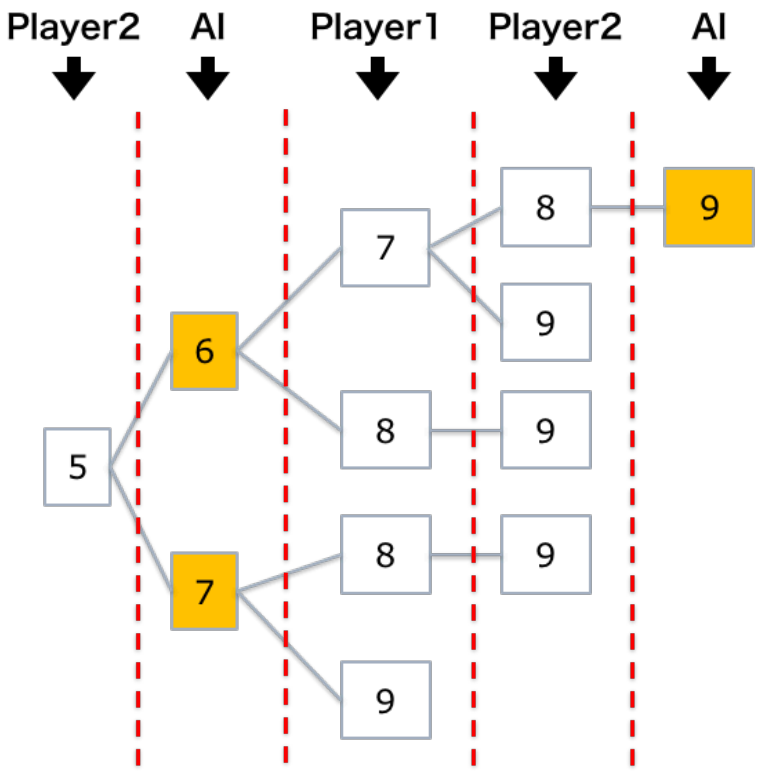

図 1　探索の方法

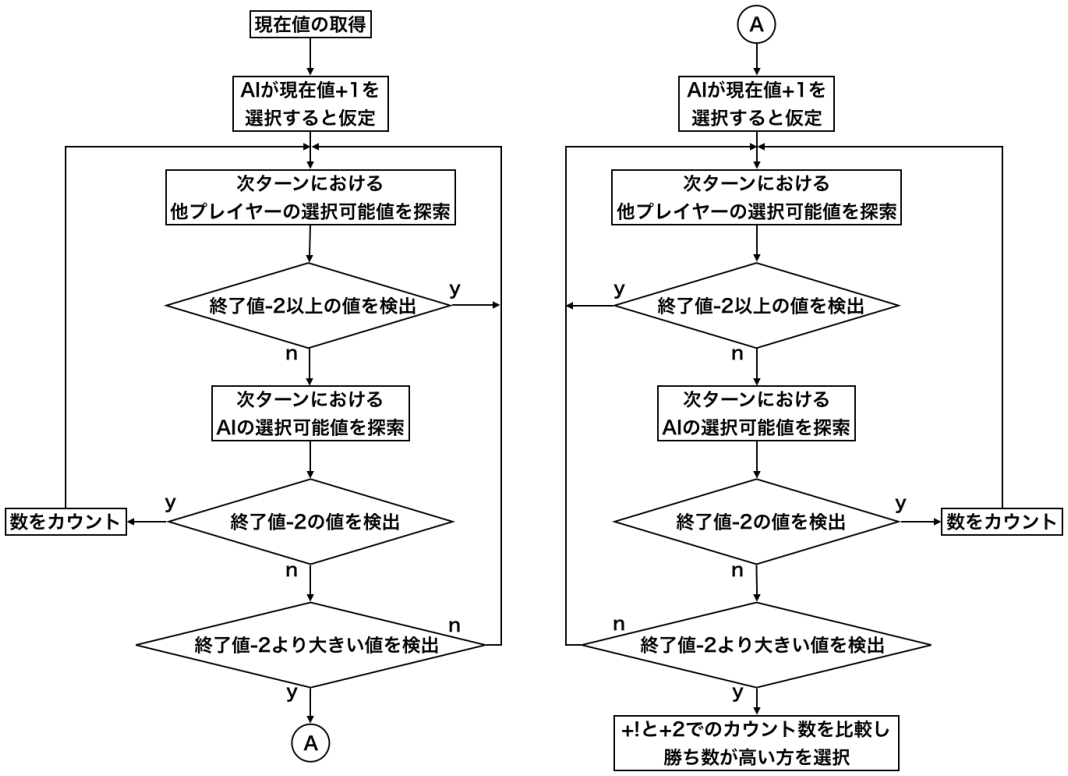

図 2　評価関数の流れ図

3　プログラムの概要

今回, プログラムは C++ を用いて設計した．C++ を用いた理由は, 使い慣れていることと処理速度が早いことの 2 点である．

プログラムの全体の進行は図 3 に示すような流れとなる．

はじめにプレイヤー及び AI の情報入力を行う．プレイヤー及び AI の情報管理は構造体を用いて行っている．当初，配列を用いた管理を行おうと考えていたが，プレイヤーが持つ情報が多く複雑になってしまうため，構造体による管理に仕様を変更した．

プレイヤー情報及び AI の情報入力の次にゲームの設定を行う，ここではゲームの終了点お設定と AI の強さを設定する．最後にプレイ順をランダムで決定し，ゲームを開始する．ゲームの中身は先ほど決定したプレイ順に現在の状態 (数値) に 1 と 2 のどちらを加算するかを決定する動きをループで回すようにする．また，1 ターン終了度にログを表示していき，誰が何番の数字を取得したがわかるようにする．

加算されて行った数値が初めに設定した終了点に到達次第ゲームを終了し，終了点の数値を取得してしまったプレイヤーを敗者とし，最終的なゲームログを表示して終了とする．

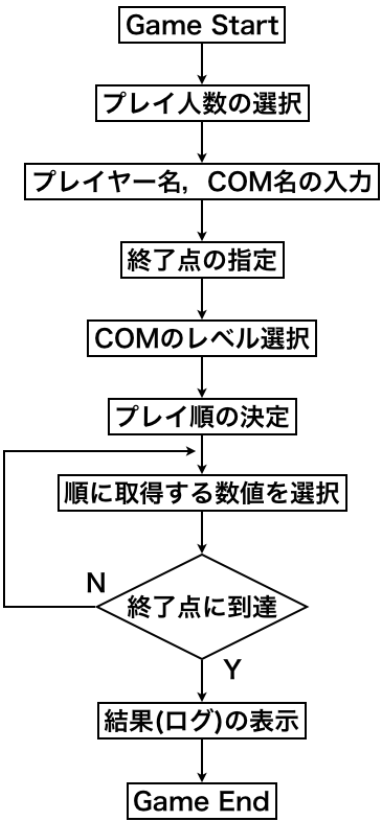

図 3　ゲームの全体の流れ図

4　プログラムの設計

4.1　プレイヤー情報の格納

初めに前述したプレイヤー及び AI の情報管理に用いた構造体について解説する．構造体 ***MEMBER*** は *Name*, *Type*, *Strength* の 3 つの要素で構成される．要素 Name は char 型の変数でプレイヤー及び AI の名前を格納する．要素 Type は int 型の変数である．要素 Type には”0”か”1”のどちらかの数値が格納され，”0”ならば人間，”1”ならば AI であると言う判定に用いる．要素 Strength も int 型の変数であり，要素 Type が”1”，つまり AI 用の構造体の時のみ使用される．これは AI の強さの判定に用いられ，先読みが可能なターン数が格納される．AI の強さが最弱の場合，一手先が読めるよう”2”が格納される．これは後述する評価関数の仕様により [先読みするターン数 × 2] の数が格納される．

4.2　プレイ順の決定

このゲームはプレーヤーと AI を合わせて 3 プレーヤーで行う．この時，プレイ順はランダム関数を用いて決定する．プレイ順は全部で 6 通りであるため，ランダム関数によって 0 から 5 の数値を決定する．プレイ

順の変更にはポインタを用いる．初めは構造体のポインタはプレイヤー情報を入力した順になっているため，ランダム関数によって決定されたプレイ順に前述の構造体のポインタを入れ替えることでメインループにおける実行順を変更する．図 4 は構造体の構成とその入れ替えの様子を示したものである．

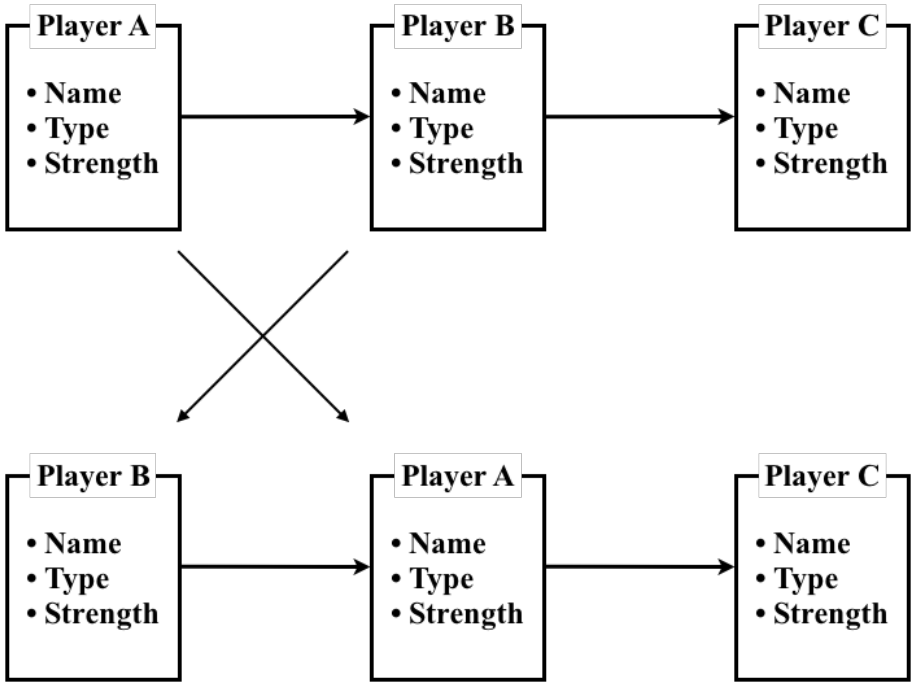

図 4　構造体の構成と入れ替え

4.3　メインループ

ゲーム進行を行うメインループの設計について解説する．メインループ内は主に *GetPlayer*，*GetAI*，*GetOperate*，*GetMemory* の 4 つの関数で構成されている．

4.3.1　GetPlayer 関数と GetAI 関数

GetPlayer 関数と GetAI 関数は主にプレイヤーへの指示及び状態の表示を行う関数であり，bool 型の関数である．メインループも冒頭で呼び出され，指定のポインターが示す構造体の要素 Strength が”0”か”1”を条件分岐により判定し，関数 GetPlayer か関数 GetAI のいづれかに振り分ける．

関数 GetPlayer では現在の位置を表示後，プレイヤーに +1 か +2 のどちらかを選択してもらう．指定された数を現状態に加え，$True$ を返す．この時，変更後の現状態がゲーム終了点に達していれば $False$ を返し，ゲームを終了させる．

一方，GetAI 関数は構造体の要素 *Strength* が”1”であり，AI のターンの場合に呼び出される．GetAI 関数内では GetOperate 関数を”1”を加える場合と”2”を加える場合で 2 回呼び出しており，GetOperate 関数によって得られた数値を比較し，評価の高い結果を現状態に加える．ただし，GetOperate 関数によって得られた 2 つの数値が同じであった場合はランダムで加える数値を決定する．この時，GetPlayer 関数と同様に通常時 $True$ を返すが，更新した現状態がゲーム終了値であれば $False$ を返す．

4.3.2 GetOperate 関数

GetOperate 関数は GetAi 関数内で呼び出される void 型の関数であり，AI の評価関数に相当する．この関数は再帰によって先の手を探索して行く．探索の深さは構造体の要素 $Strength$ によって決定し，1 手先読み，4 手先読み，全探索の 3 パターンが存在する．ここでは 4 手先読みを例にアルゴリズムの説明をする．図 5 は AI が +1 を選択し次のターンを探索する様子を表す図である．

思考する AI から見ると，次の AI のターンに回ってくる可能性がある数は

[思考中の AI が選択した数字 + 思考中の AI を除いた 2 プレイヤーが選択した数値の合計]

である．すなわち，探索は AI が加算する範囲 (1~2) と他のプレイヤーが選択する範囲 (2~4) を交互に探索する形となる．探索は AI が 1 を選んだ場合と 2 を選んだ場合の 2 パターンで行う．それぞれの探索結果を比較しより良い結果が出る値を採択する．4 手先読みを行う時，探索の深さを決定する変数 **dep** には 8 が格納されている．AI の選ぶ数別に探索を行うため初めの探索は他プレイヤーの選択する数値になる．この時，変数 $dep = 8$ であり 1 階層分の探索を終える度にディクリメントを行う．次は 1 手先の AI ターンを探索する，この時変数 dep はディクリメントされ $dep = 7$ となる．すなわち変数 dep が偶数の時は他プレイヤーの探索，奇数の時は AI の探索を行っている状態となる．これを利用し，dep が奇数の時に勝ち数を踏めた数の合計をグローバル変数 CNT に格納する．これを現在のターンで +1 を選んだ場合と +2 を選んだ場合とで比較することでより勝率の高い選択を行えるようになる．この時，両パターンの勝ち数が同数だった場合はランダムで選択する数値を決定することで単調なゲーム進行になることを防いでいる．

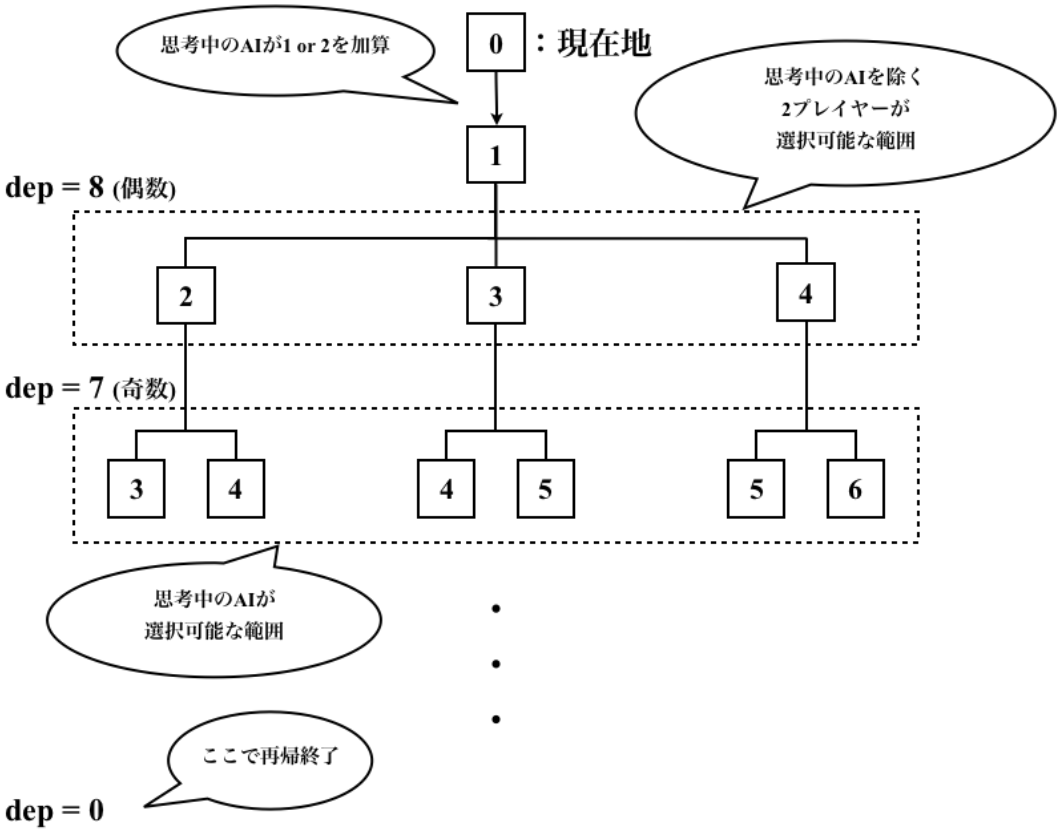

図 5　GetOperate 関数のアルゴリズム

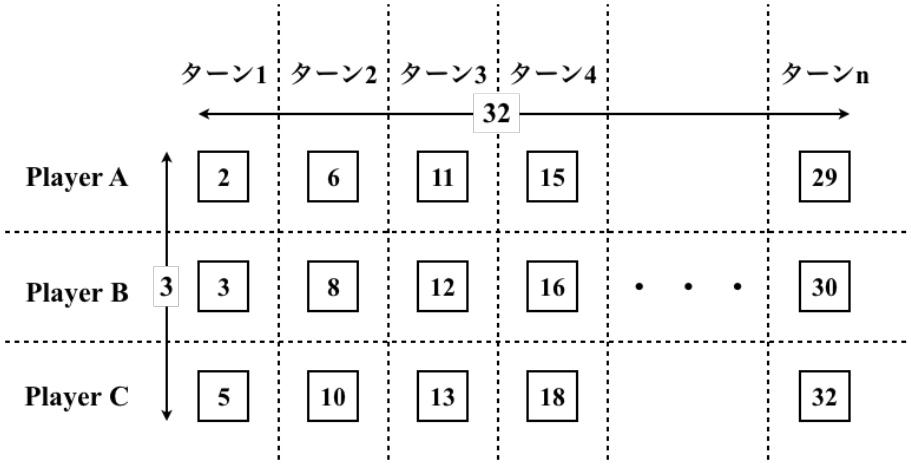

図 6　配列 memory の中身

4.4　GetMemory 関数

GetMemory 関数はゲームログの表示を担う関数である．実際のゲームログは，二次元配列 $memory$ に格納される．配列 $memory$ は 3×32 の二次元配列であり，プレイヤーごとの取得した数値を保管して行く．ログはプレイ順に格納されて行くため GetMemory 関数では配列の中身を順に表示するだけで良い．図 6 は配列 memory に格納されたログの例を示している．GetMemory 関数は毎ターン終了時およびゲーム終了時に呼び出される．

5　実装結果

実際にシステムを実装した結果を示す．今回は人間 1:COM2 と COM のみでゲームを行なった．

5.1　動作テスト

図 7 はゲームの初期設定を行なっている場面である．ユーザー情報入力時では構造体に Player_A, COM_A, COM_B の順に情報が格納されているが，順番の変更に伴って COM_A，COM_B，Player_A の順に変更されていることがわかる．今回ゲーム終了点は 24，AI の強さは COM_A を弱い，COM_B を強いに設定した．図 9 は全体のゲーム進行の様子および最終的な結果である．結果は COM_A の敗北となり，AI の強さに従った結果となった．毎ターン表示されるゲームログもプレイ順に正しく表示されている．ゲーム終了時のログ図 9 の最下部にあるように未プレイのプレイヤーのログには何も表示されないようになっている．また，図 8 はゲーム中のエラー処理の様子であり，プレイヤーが範囲外の数値を指定してしまった場合に，正しい数値を指定するように促すようになっている．

次に AI のみで対戦を行い，強さを変更しながら結果を観察した．終了点は全て 24 で行っている．設定ごとの結果を以下に示す．

弱い vs 弱い vs 強い

この設定では強い AI が弱い AI に敗北することはなかった．弱い AI のどちらが負けるかはランダムであった．

弱い vs 普通 vs 強い

勝敗は AI の強さとは関係なくランダムであった．

普通 vs 普通 vs 強い

基本的に普通の AI のどちらかが敗北した，しかし数回に 1 回の頻度で強い AI も敗北していた．強い AI が敗北する頻度は [弱い vs 普通 vs 強い] の時よりも低く感じた．

弱い vs 弱い vs 弱い

勝敗は完全にランダムであった．

普通 vs 普通 vs 普通

上述の [弱い vs 弱い vs 弱い] と同様に勝敗は完全にランダムであった．

強い vs 強い vs 強い

必ずプレイ順が 2 番手となった COM が敗北した．

以上が今回の実装で得られた結果である．

5.2 考察

今回得られた結果から，1 手先しか読むことのできない AI は実際に最弱ではあるが，4 手先読みが可能な AI と全探索が可能な AI では大きく強さに差が出ないことがわかった．これはゲームの特性上，前半の数値の選択による勝敗への影響が少ないため，ある程度の探索ができれば強さへの影響も小さくなってしまうのではないかと考えた．

また，全て同じ強さに統一した時に**強い**の時のみ結果が変わらなかったのは，全ての AI が全探索を行ってしまうため，ゲーム毎の数値選択に変化が現れずプレイ順によって結果が固定しまったと思われる．

探索に関しては，+1 を選んだ方が分岐の回数も置くなり，必然的に +2 を選ぶより多くの勝ち数を踏む可能性が高くなる．そのため，最終局面以外での AI の選択は +1 が多くなる．ゲームに抑揚を持たせるためにも，+1 と +2 のパターンで探索に重み付けを行えばゲームの進行に抑揚が生まれるように思われた．

6 結論

このゲームには必勝法に近いゲーム展開が存在するため，考えてゲームを進行すれば人間が AI に負けることはなさそうであった．今回はゲーム終了点を 24 に設定したが，終了点が 100 などの大きな数字になると全探索を行うのにかなりの時間を要してしまった．前述したようにゲームの特性上，選択がゲームに及ぼすのは後半のみであるため，このゲームには全探索を行うメリットはあまりなかったと考えられる．そのため AI の探索深度は 1 手先読み，3 手先読み，5 手先読み程度の小さい範囲でのバランスが良さそうである．

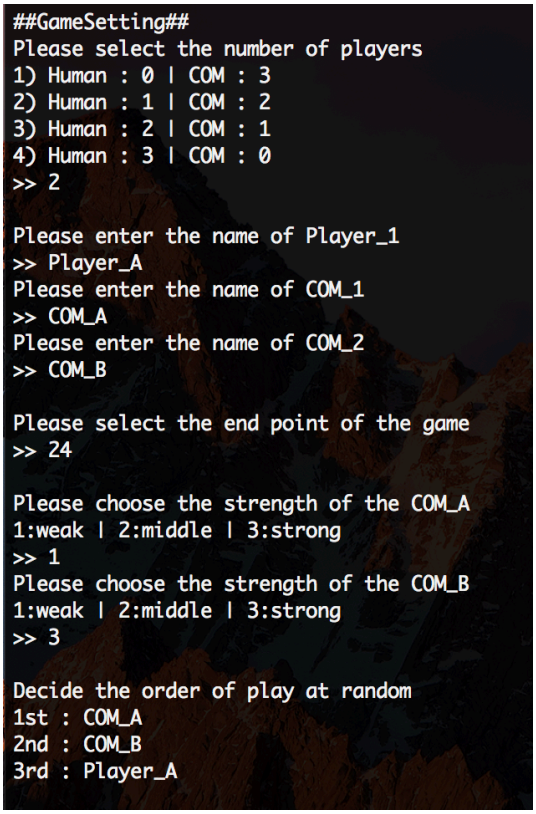

図 7　ゲームの初期設定部分の例

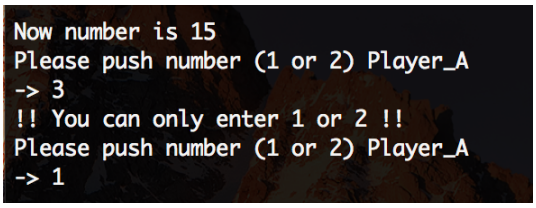

図 8　エラー処理

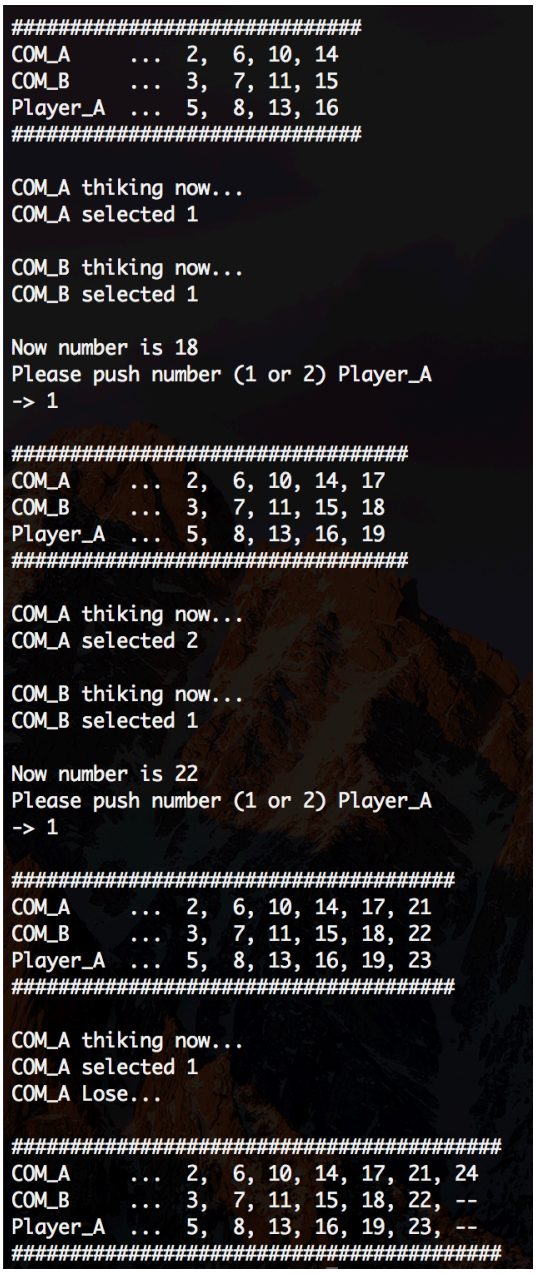

図 9　ゲームの流れと結果