卒業研究報告書

題目

ゴブレットゴブラーズの完全解析

指導教員

石水 隆 講師

報告者

18–1–037–0006

浅野 雄也

近畿大学理工学部情報学科

令和 4 年 1 月 24 日提出

概要

本研究では，後退解析を用いてゴブレットゴブラーズの完全解析を行った．ゴブレットゴブラーズとは○×ゲーム（ Tic-Tac-Toe）をより複雑にした 3 × 3 マスのゲーム盤を使用する 2 人用のボードゲームであり，○×ゲームと同じく，自分のコマが縦横斜めのいずれかに 3 つ並べたプレイヤーが勝ちとなる．双方のプレイヤーは大中小 3 種類のコマを 2 つずつ持ち，各プレイヤーは手番で自コマを盤上に乗せるか盤上にある自コマを他のマスに移動させることができる．このとき，各コマは盤上にあるより小さいコマの上に被せることができる．ゴブレットゴブラーズは二人零和有限確定完全情報ゲームであり，全ての局面において双方最善手を指した場合の勝敗が決定可能である．後退解析を使用すると，勝敗が確定した局面から遡って勝敗をつける方法で，全局面の勝敗を得られる．本研究では，全局面に対して解析を行った結果として，初期局面は 13 手で先手必勝である．また，解析結果を利用することで，常に最善手を指す AI の CPU と対戦可能なゲームアプリケーションを開発した．

目次

1 序論 1

1.1 本研究の背景 1
1.2 ゴブレットゴブラーズ 1
1.3 本研究の目的 2
1.4 本報告書の構成 2

2 研究内容 2

2.1 表記法と総局面の予想 3
2.2 正規化と局面の bit 表現 4
2.3 全局面の列挙 5
2.4 後退解析 5

3 検証と考察 5

3.1 局面の数 5
3.2 初期局面の勝敗 5
3.3 初期局面の勝敗 6
3.4 最長の詰み手 6

4 ゲームアプリケーション 8

4.1 ゲームアプリケーションの概要 8
4.2 ゲームアプリケーションの仕様 9
4.3 ゲームアプリケーションの遊び方 9
4.4 解析結果を利用した AI 11
4.5 ゲームアプリケーションで使用した AI のプログラム 12

5 結論と課題 13

謝辞 13

参考文献 14

付録 A 解析に使用したソースプログラム 15

付録 B ゲームアプリケーションに使用したソースプログラム 47

1　序論

1.1　本研究の背景

将棋，オセロ，三目並べなどのボードゲームは二人零和有限確定完全情報に分類できる．これは対戦人数が二人で（二人），対戦者の得点の合計が 0（零和），可能な局面の数が有限（有限），ランダム性がなく（確定），ゲームの情報が全て公開されている（完全情報）ゲームのことである．この種類のゲームは初期局面を含む全ての局面において，その局面から双方最善手を指した場合の勝敗が定まっている．しかし，将棋・囲碁等では，可能な局面の数が極めて大きいため，現実的な時間で最善手を求めることはできず，その勝敗はまだ分かっていない．例を挙げれば，可能な局面数はリバーシが 10^{28} 通り，チェスが 10^{50} 通り，将棋が 10^{69} 通り，囲碁が 10^{170} 通り程度あるとされており，現在のコンピューターの性能を越えている．一方，どうぶつしょうぎなどの総局面数の少ないゲームであれば，メモリと時間を使って後退解析で全局面の勝敗を求めることができる．

完全解析がされているゲームには，連珠，チェッカー，コネクト 4 等がある．連珠は双方最善手を打った場合，47 手で先手の勝ち，チェッカーは双方最善手を指すと引き分け，コネクト 4 は双方最善手を打つと 41 手で先手の勝ちとなる [4, 5, 6] ．また，将棋・囲碁等については，ゲーム盤の大きさを小さくしたミニゲームに対しては完全解析が行われているものもある．ミニ将棋は、どうぶつしょううぎが双方最善手を指すと 78 手で後手勝ち，アンパンマンはじめてしょうぎが双方最善手を指すと引き分けとなることが示されている [1, 7]．また，ミニ囲碁は，双方最善手を打った場合，4 路盤では引き分け，5 路盤では黒の 24 目勝ちとなる [8, 9]．ミニリバーシは，6x6 盤が 16 対 20 で後手勝ち，4x4 盤が 3 対 11 で後手勝ち，4x6 盤が 20 対 4 で先手勝ち，4x8 盤が 28 対 0 で先手勝ち，4x10 盤が 39 対 0 で先手勝ちとなる [10, 11]．

これらの多くのボードゲームはコンピューター上でもルールを再現したゲームとして遊ばれる．人と人がコンピューター上で対戦する他，コンピューターとの CPU 対戦が可能となってきている．ゴブレットゴブラーズが遊べる既存のアプリケーションはいくつか存在する [12, 13, 14]．解析結果を利用することで常に最善の手を指す AI の CPU を開発できる．

1.2　ゴブレットゴブラーズ

本節ではゴブレットゴブラーズの詳細について述べる [3]．

表 1　製品情報

メーカー	Blue Orange (フランス)
デザイナー	Thierry Denoual
カテゴリー	キッズゲーム
初版発行年	2002 年

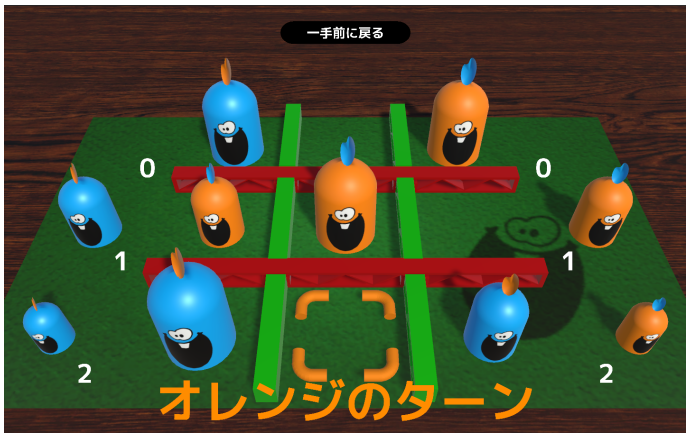

図 1　ゲームの一局面

ゴブレットゴブラーズとは○×ゲーム（ Tic-Tac-Toe）をより複雑にしたボードゲームである．表 1 にゴブレットゴブラーズの製品情報を示す．ゴブレットゴブラーズは二人用の対戦ゲームであり，3 × 3 の 9 マスのゲーム盤と大中小 3 種類の大きさのある青とオレンジの 2 色のコマを用いる．プレーヤーは青かオレンジの一方を選択し，大中小の 3 種類のコマを 2 つずつ互いに手元に持った状態で始める．

手番では手元の自コマをマスに配置する，もしくはボード上の見えている自コマを移動して他のマスに配置することができる．配置する時，コマはよりすでに盤上にあるより小さいコマに被せて配置できるが，より大きいコマ，あるいは同じ大きさのコマがあるマスには配置できない．また，より大きなコマの下に隠されているコマを動かすことはできない．図 1 はゲームの一局面である．

最終的に自コマを縦・横・斜めのいずれか一列に並べたら勝ちである．ただし，被せていたコマを動かす際にその下から出てきたコマにより，相手のコマが一列揃った場合はそのコマを取った時点で負けである．

1.3　本研究の目的

ゴブレットゴブラーズは 13 手で先手必勝であることが佐藤により 2019 年に証明されており，いくつかの勝ち手順が存在することがわかっている [2]．しかし，文献 [2] では全ての局面の解析には至っていない．そこで本研究では全局面を解析し，任意の局面に対して最善手を得られるようにする．

また，解析結果がゴブレットゴブラーズの遊びの幅を広げられることを証明するために，解析結果を利用し，このゲームの本質をより直感的に理解できるようにするゲームアプリケーションを作成する．アプリケーションの作成においては，常に最善手を指す AI の開発を目標とする．

1.4　本報告書の構成

本報告書の構成を以下に示す．2 章で研究内容の説明，3 章で検証と考察を行い，4 章でゲームアプリケーションの解説，5 章で結論と今後の課題について述べる．

2　研究内容

本研究は以下の 2 つの課題を行う．

1. ゴブレットゴブラーズの完全解析を行う．
2. 解析結果を利用したゲームアプリケーションを開発する．

2.1 表記法と総局面の予想

まず，解析を行うために可能な局面数を考える．3 種類 2 つずつのコマの状態はそれぞれボード上（1〜9）もしくは手元に持った状態（0）なので 10 進数で一桁で表せ，先手の大大中中小小（000000）と後手の大大中中小小（000000）の合計 12 桁で表記できる．なお，この論文では先手をオレンジ，後手を青として扱う．

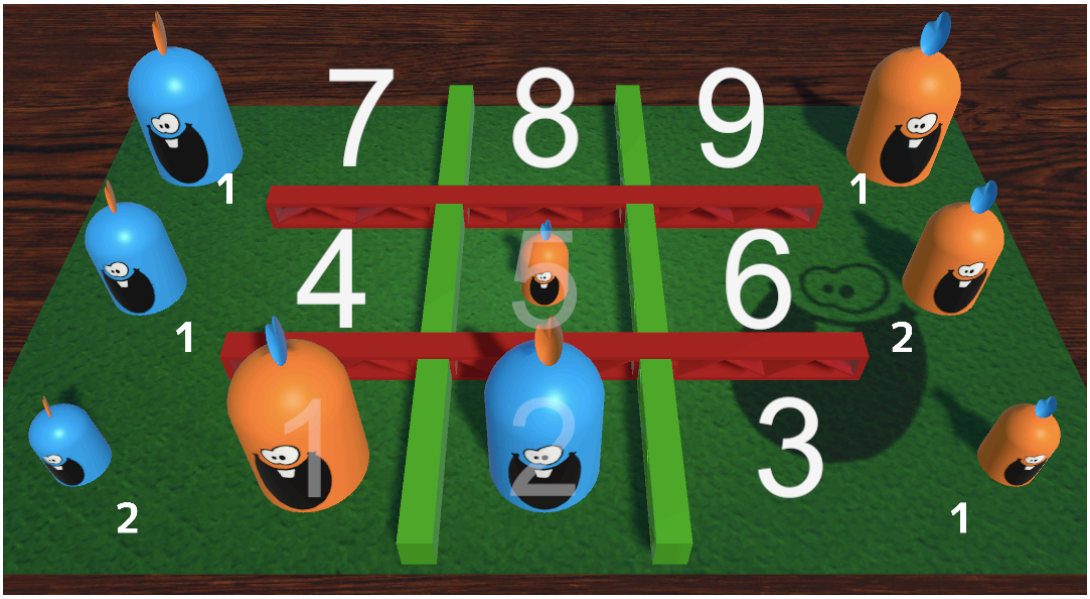

図 2　010005 020100 のコマの配置

例として，図 2 のようにコマを配置した場合を考える．図 2 では先手の大コマが 1 番のマス、小コマが 5 番のマスに配置され，後手の大コマが 2 番のマスに配置されている．また，図 2 では表示されていないが，後手の中コマが 1 番のマスにある先手の大コマの下に隠されている．この時，コマの状態は 010005 020100 と表せる．

総局面は 000000 000000 から 999999 999999 の 1 兆パターンであるが，正規化していくことでかなり減らせる．簡単なプログラムで正規化し，手番のプレイヤーが勝つ局面を除いた結果，可能な局面数は 266219487 局面となった．この局面数は十分に小さいものであり，コンピューターによる後退解析を行うことができる．

2.2　正規化と局面の bit 表現

図 3　対称性のある局面

9 マスのボードは図 3 のように他の 7 つの局面で対称性を認めることができる．そのため，各局面に対して，対称性を持つ局面を求め，bit 表現したときに最も小さい値となる局面で正規化することとする．

手番についてはコマを先手と後手で入れ替えることで表現できるため，局面の手番を固定できる．つまり，手番をすすめたら，コマを入れ替えることで次の手番の局面を表現する．これにより局面の手番の情報を削除できる．

先程の局面の表現と別にコンピューター上で扱う bit 表現について考える．一つのマスは大中小それぞれの状態が先手のコマ，後手のコマ，空の 3 種類の状態で 3 × 3 × 3 の 27 パターンであり，5bit で表現できる．それが 9 つあるので 5 × 9 の 45bit となる．局面はこの 45bit だけでも表現可能であるが，本研究では計算スピードを向上させるため手元のコマの情報も足す．手元のコマの一種類の数は 0 か 1 か 2 の 3 パターンであり，2bit で表現できる．それを 2 人のプレイヤーがそれぞれ 3 種類持つので 2 × 3 × 2 の 12bit を足して表現する．したがって合計で 57bit の 2 進数で局面を表現することができる．

2.3 全局面の列挙

局面の中には，初期局面から到達不能な局面も存在する．そこで，初期局面（000000 000000）を保存局面の配列に入れ，次の全ての局面をキューの中に入れる．キューの先頭の局面を取り出し，保存局面の配列を確認してなければその配列に保存し，その局面の次の局面を全てキューの中に入れる（勝敗が決まっていれば次の局面は出さない）．以降繰り返すことで初期局面から到達可能な局面を全て出した．結果として局面の数は 263096473 で予想の範囲である．

2.4 後退解析

後退解析では勝敗が決している局面から出発して，後ろから遡るようにして解析する．
今回の解析は以下の手順で行う．

1. 局面の次の局面を全て確認する．
2. 次の局面がない場合負け局面とする．
3. 次の局面に一つでも（調べている局面の相手の）負け局面がある場合，勝ち局面とする．
4. 次の局面の全てが（調べている局面の相手の）勝ち局面である場合，負け局面とする．
5. その他の場合引き分けとする．

以上を局面の評価値に変化がなくなるまで繰り返すことで後退解析を行う．なお，2 番目の次の局面がない場合負けとする理由として，そのような局面はコマを移動することで負けてしまう局面であるからである．

本研究では，Intel Core i7 2.9GHz のマシン (メモリは 64GB) で計算を行った．計算に所要した時間は 3 時間ほどであり，メモリも 16GB ほどで良いことが分かった．付録 A に本研究で作成した後退解析プログラムを示す．

3 検証と考察

3.1 局面の数

すべての到達可能局面を出した上で後退解析を行い，その勝敗を求めたところ，先手の勝ち局面 164046788・引き分け局面 113622・負け局面 98936063 であった．出した総局面 263096473 の中で 77915125 局面は局面そのもので勝敗が付いているため、残りの 185181348 局面が勝敗の付いていないゲーム中に存在できる全ての局面である．

3.2 初期局面の勝敗

初期局面（000000 000000）は 13 手で先手勝ちである．また一手目を指し終えた時の勝敗を表 2 に表す．

表 2　一手目を指し終えた時の勝敗と詰み手数

局面	勝敗	最善手を指した場合の詰み手数
000001 000000	勝ち	14
000002 000000	勝ち	12
000005 000000	勝ち	14
000100 000000	負け	13
000200 000000	負け	11
000500 000000	負け	11
010000 000000	勝ち	12
020000 000000	勝ち	14
050000 000000	勝ち	12

3.3　初期局面の勝敗

初期局面の勝敗から，最短で勝つには初手は辺の真ん中に小さいコマを配置するか，四隅か中央に大きいコマを配置する．しかし初手に中くらいのコマを配置する場合，どこに配置しても 2 手目に大きいコマをそのコマに被せることで，先手の負けとなる．大きいコマを中央に配置した場合，基本的に連続リーチによって勝つことができる．

3.4　最長の詰み手

最長で 25 手となる詰み局面を 33 局面確認した．その中で最も初期のコマ数が少なかった局面の詰み手順の一例を表 3 に示す．

表 3　最長の詰み手数となる局面の一例とその詰み手順の一例

局面	手番	最善手を指した場合の詰み手数
004600 000004	後手	25
004600 060004	先手	24
004605 060004	後手	23
004605 060504	先手	22
054605 060504	後手	21
054605 060549	先手	20
594605 060549	後手	19
594605 160549	先手	18
354605 160549	後手	17
354605 165749	先手	16
594605 165749	後手	15
594605 163549	先手	14
594625 163549	後手	13
594625 162549	先手	12
354625 162549	後手	11
354625 672549	先手	10
594625 672549	後手	9
594625 462549	先手	8
354625 462549	後手	7
354625 462548	先手	6
254625 462548	後手	5
254625 462534	先手	4
354625 462534	後手	3
354625 672534	先手	2
154625 672534	後手	1
154625 692534	先手	0（詰み）

表 3 に示される手順は一本でなく，数手前に戻ることもある．これらの局面から始めるのも面白いかもしれない．図 4 に表 3 の初期局面と詰み局面の直前の局面を示す．なお，図 4 はどちらも後手（青）の手番であり，その詰み局面の直前の局面は，表 3 の最終行と同様に左上の大きい青コマを右上に配置することで詰む局面である．

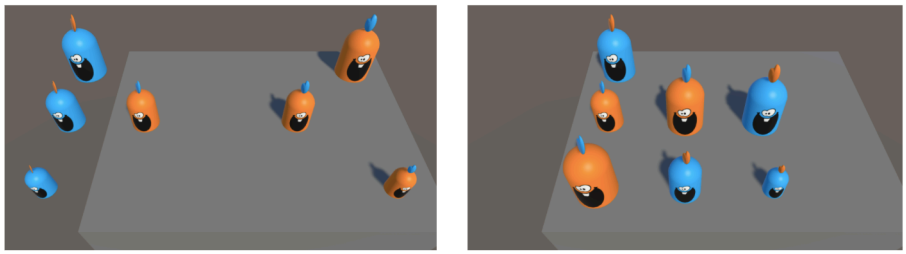

図 4　上記手順の初期局面と詰む直前の局面

4　ゲームアプリケーション

本章では，本研究で作成したゲームアプリケーションについて述べる．

4.1　ゲームアプリケーションの概要

本研究ではゴブレットゴブラーズをプレイすることのできるゲームアプリケーションを作成した．本研究で作成したアプリケーションは UnityC ＃を使用し，基本的なゲーム機能を搭載した上で，解析結果を利用した AI の CPU と対戦して遊べる．図 5 に本研究で作成したアプリケーションの実行の様子を示す．またその情報を表 4 に示す．表 4 のサイズに関して，解析結果のファイルが約 2.5GB 含まれる．

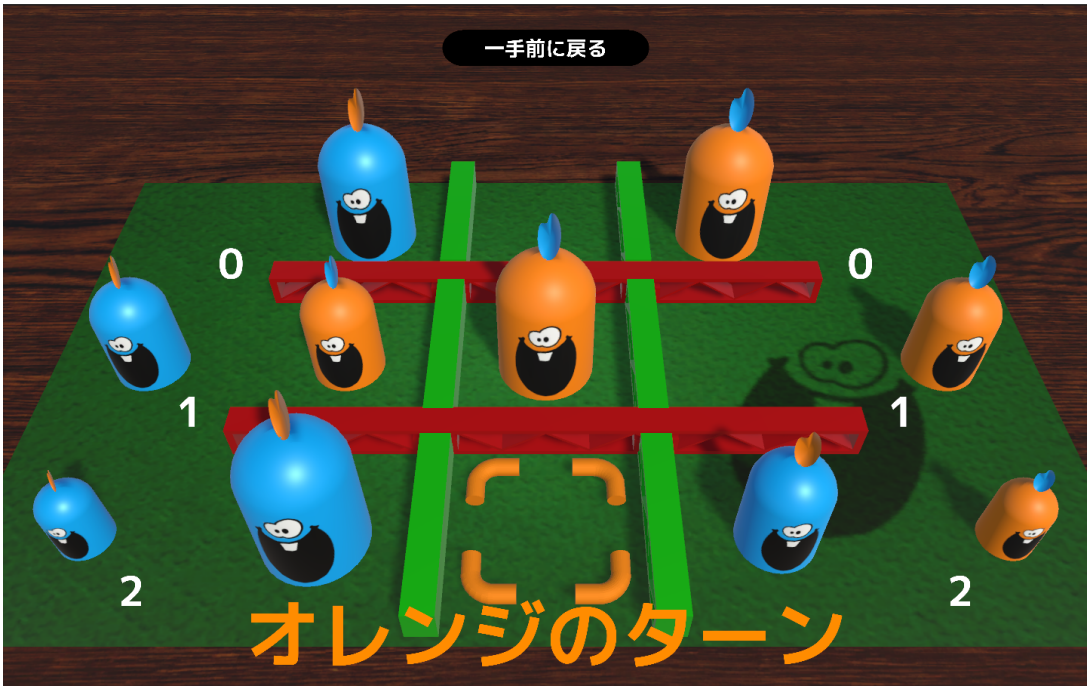

図 5　アプリケーション実行の様子

表 4　ゲームの情報

プラットフォーム	パーソナルコンピューター
動作環境	Mac，Windows10・11（64bit）
形態	スタンドアローンアプリケーション
サイズ	約 2.86GB

4.2　ゲームアプリケーションの仕様

本節ではゲームアプリケーションの仕様を説明する．このアプリケーションは PC 上で動作し，キーボードは使用せず，マウスやタッチパッドで操作する．アプリケーションは単体で起動可能で，終了は起動直後に開かれるタイトル画面の ESC ボタンを押して終了できる．

タイトルやオプション，モード選択など基本的なゲーム機能を搭載した．その機能構成を図 6 に示す．

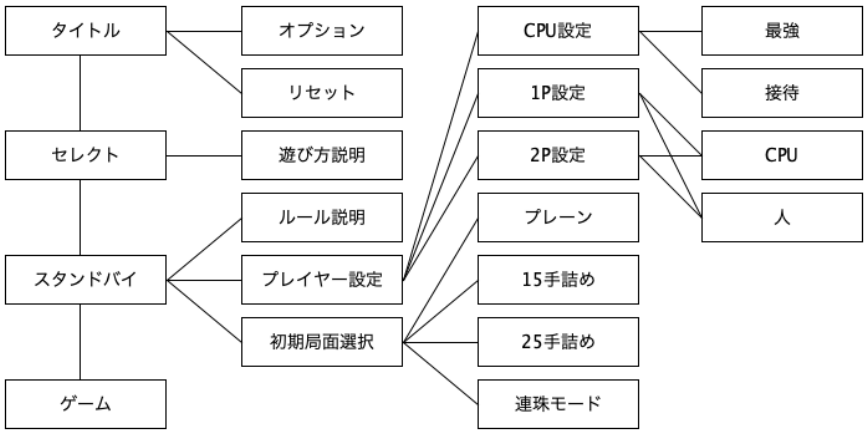

図 6　機能構成

4.3　ゲームアプリケーションの遊び方

本節ではゲームアプリケーションの遊び方を説明する．図 7 にタイトル画面とオプション画面を示す．

図 7　タイトル画面（左）とオプション画面（右）

タイトル画面の右中央のスタートボタンを押すことでセレクト画面に移動できる．左下のリセットボタンでクリアデータを削除でき，右下のオプションボタンでオプション画面を開くことができる．オプション画面では音量の他，ゲーム中の一手戻るボタンのオンオフを変更できる．

図 8 にセレクト画面と遊び方を説明する画面を示す．

図 8　セレクト画面（左）と遊び方説明中の画面（右）

セレクト画面の右側の決定ボタンを押すことでスタンドバイ画面に移動できる．左側の遊び方ボタンで遊び方が図 8 右側のように説明される．遊び方説明はクリックすることで説明が続く．

図 9 にスタンドバイ画面とゲーム中の画面を示す．

図 9　スタンドバイ画面（左）とゲーム中の画面（右）

スタンドバイ画面ではゲームの設定を変更できる．ゲームの対戦は人と CPU の二種類から選べ，CPU は最善の手を指す AI と常に負ける手を指す AI の二種類を選択できる．ゲームは初期局面として，プレーン（000000 000000）・15 手詰め（050400 020006）・25 手詰め（004600 000004）の 3 種類の局面からスタートする 3 種類の通常モードと，4 手目を指す時に手番を入れ替えるか選択できる連珠モード（000000 000000/後手のみ）で遊ぶことができる[*1]．

スタンドバイ画面の右下のプレイボタンを押すことでゲームを始めることができる．ゲーム中，ポインタがプレイヤーの手番の色の時，カーソルを動かすことでポインタを移動でき，自コマに合わせた状態で右クリックすることで，コマを選択状態にできる．選択状態ではコマは画面上で少し浮いた状態になるので，コマを置きたい場所にポインタを移動して再度右クリックすることで，その場所が移動可能ならばコマの配置を行う．その場所が移動不可能な場合，コマの選択状態は解除される．ゲーム中は一手戻るボタンを押すことで局面を戻したり，ホームボタンを押すことでスタンドバイ画面に戻ることができる．

4.4　解析結果を利用した AI

解析結果によって全ての局面の評価値がわかるため，最善の手を指す AI は次のように作成できる．

1. 局面の次の局面を全て確認する．
2. 局面が負け局面の場合，次の局面の中から最も詰むまでの手数が長い局面を抜き出す．
3. 局面が勝ち局面の場合，次の局面の中から最も詰むまでの手数が短い且つ勝ち局面を抜き出す．
4. 抜き出した局面からランダムで一つ選んで次の局面とする．

また常に負ける手を指す AI は次のように作成できる．

1. 局面の次の局面を全て確認する．
2. 次の局面の中から最も詰むまでの手数が短い且つ負け局面を抜き出す．
3. 抜き出した局面からランダムで一つ選んで次の局面とする．

次の手を出せたら，ゲーム情報を更新して画面に局面を表示する．

[*1] 連珠は先手必勝のため，3 手目まで指した後に後手はそのまま後手をしてゲームを進めるか，先手と後手を入れ替わるかを選択することができる

4.5 ゲームアプリケーションで使用した AI のプログラム

本節ではアプリケーションの AI に関わるクラスとプログラムについて説明する．また，本研究で作成したアプリケーションのソースプログラムを付録 B に示す．

・State クラス

State クラスはゲームの局面情報を表すために配列で構成された局面や持ちコマの情報を持つ．

・State.nextStates()

次に遷移可能な局面を List にして返す．

・State.nextMoves()

次に遷移可能な局面へ移行するために必要な，コマの選択座標とコマの移動先座標を合わせて 4 桁の数字にしたものを List にして返す．

・AllStateTable クラス

AllStateTable クラスは全ての局面のデータファイルを読み込んで，いつでも参照できる．

・AllStateTable.find(ulong)

検索したい局面のインデックスを返すことができる．

・WinLoseTable クラス

WinLoseTable クラスは全ての局面の評価値をまとめたファイルと全ての局面の詰みまでの手数をまとめたファイルを読み込んで，参照できる．参照するために AllStateTable クラスから得たインデックスを使う．

・WinLoseTable.nextHand(State)

次の手のために，選択する自コマの座標とその移動先座標を合わせた 4 桁の数字を返す．

・GobbletManager クラス

GobbletManager クラスはゲーム情報の更新や入力処理をしている．CPU のターンの時，GobbletManager.superAI() 関数を使い，次の手を決める．

・GobbletManager.superAI()

CPU の行動を行う．解析結果を利用した AI で次の手を指す．コードの一部を示す．

```
int hand=0;
//WinLoseTable クラスから次の手を 4 桁の数字で入手
hand=winLoseTable.nextHand(state);
int x=(int)hand/1000%10;
int y=(int)hand/100%10;
select(x,y);//座標のコマを選択状態にする
x=(int)hand/10%10;
y=(int)hand%10;
move(x,y);//選択状態のコマを座標に移動させる
```

変数 winLoseTable は WinLoseTable クラスのクラス型変数である．GobbletManager.select(int,int) や GobbletManager.move(int,int) は座標を受け取って，ゲーム情報を更新できる．

5　結論と課題

本研究では，ゴブレットゴブラーズの完全解析を行った．本研究では，後退解析により初期局面から到達できる全ての局面で双方最善手を指した場合の勝敗を求めることができた．また結果を利用したゲームアプリケーションを作成し，このゲームの本質を理解しやすくできた．ゴブレットゴブラーズは総局面が 2 億 5 千万ほどであるが，初期局面からの最善手によるゲーム木はそれほど多くないため，子供でも最善手を把握しやすいゲームである．また，ゴブレットゴブラーズは他の深層学習などの AI の評価にも使用しやすい低難易度のゲームであると言える．

今後の課題として，研究で使いやすくするために $\alpha\,\beta$ 法等を組み合わせて解析結果をより削減したデータからでも最高の手を指す AI の開発することや，解析結果を使用しない他の AI との対戦による評価をすることが挙げられる．

謝辞

ありがとうございました．

参考文献

[1] 田中哲郎:「どうぶつしょうぎ」の完全解析, 情報処理学会研究報告,Vol.2009-GI-22 No.3, pp.1—8 (2009)

[2] 佐藤正隆:ゴブレットゴブラーズの解析, 東邦大学理学部情報科学科卒業研究 (2019).

[3] Blue Orange — Gobblet Gobblers,https://blueorangegames.eu/en/games/gobblet-gobblers/

[4] Janos Wagner and Istvan Virag, Solving renju, ICGA Journal, Vol.24, No.1, pp.30–35 (2001), http://www.sze.hu/~gtakacs/download/wagnervirag_2001.pdf

[5] Jonathan Schaeffer, Neil Burch, Yngvi Bjorsson, Akihiro Kishimoto, Martin Muller, Robert Lake, Paul Lu, and Steve Suphen, Checkers is solved, Science Vol.317, No,5844, pp.1518–1522 (2007), http://www.sciencemag.org/content/317/5844/1518.full.pdf

[6] Victor Allis, A Knowledge-based Approach of Connect-Four, The Game is Solved: White Wins, Master Thesis, Department of Mathematics and Computer Science Vrije Universiteit (1988), http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf

[7] 塩田好, 石水隆, 山本博史, 「アンパンマンはじめてしょうぎ」の完全解析, 2013 年度情報処理学会関西支部支部大会講演論文集, (2013), http://id.nii.ac.jp/1001/00096792/

[8] 清慎一, 川嶋俊，探索プログラムによる四路盤囲碁の解, 情報処理学会研究報告, GI 2000(98), pp.69–76 (2000), https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_uri&item_id=58632

[9] Eric C.D. van der Welf, H.Jaap van den Herik, and Jos W.H.M.Uiterwijk, Solving Go on Small Boards, ICGA Journal, Vol.26, No.2, pp.92-107 (2003).

[10] Joel Feinstein, Amenor Wins World 6x6 Championships!, Forty billion noted under the tree (July 1993), pp.6–8, British Othello Federation's newsletter., (1993), http://www.britishothello.org.uk/fbnall.pdf

[11] 竹下拓輝. 池田諭, 坂本眞人, 伊藤隆夫, 縮小盤オセロにおける完全解析, 情報処理学会九州支部火の国情報シンポジウム, No.1A-2, pp.1-6 (2015), https://www.ipsj-kyushu.jp/page/ronbun/hinokuni/1004/1A/1A-2.pdf

[12] すごろくや：ゴブレットゴブラーズ，Google Play, 2021 年 12 月 19 日, https://play.google.com/store/apps/details?id=com.sugorokuya.gobblet_gobblers_app

[13] Masakazu Sakata：ねこハコ ～進化版まるばつゲーム～　オンライン&オフラインプレイ可能！，Google Play, 2021 年 1 月 27 日, https://play.google.com/store/apps/details?id=com.masanoh.catandbox

[14] Tomoyasu Hidaka, MARUBA / まるばつゲーム進化版 オンライン, Apple Store, 2022 年 1 月 15 日, https://apps.apple.com/jp/app/maruba-まるばつゲーム進化版-オンライン/id1472998169

付録 A　解析に使用したソースプログラム

Makefile

```
ALL : makeAllState makeWinLose checkState
CXX = g++
gobblet.o : gobblet.h
#コンパイル用 makefaile コマンド make で全てのファイルをコンパイルできる
allStateTable.o : allStateTable.h gobblet.h
winLoseTable.o : winLoseTable.h gobblet.h
makeAllState.o : gobblet.h
makeAllState : makeAllState.o gobblet.o
$(CXX) -o makeAllState makeAllState.o gobblet.o
makeWinLose.o : gobblet.h
allStateTable.o : allStateTable.h gobblet.h
makeWinLose : makeWinLose.o gobblet.o allStateTable.o
$(CXX) -o makeWinLose makeWinLose.o gobblet.o allStateTable.o
checkState.o : gobblet.h
checkState : checkState.o gobblet.o allStateTable.o winLoseTable.o
$(CXX) -o checkState checkState.o gobblet.o allStateTable.o winLoseTable.o
clean:
-rm *.o
```

gobblet.h

```
#ifndef _DOBUTSU_H
#define _DOBUTSU_H
#include <cassert>
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <vector>
#include <complex>
#include <deque>
#include <string>
using namespace std;
typedef complex<int> Point;

typedef unsigned long long uint64;//64bit 整数
```

```
typedef vector<uint64> vUint64;//64bit 整数のベクター
typedef vector<char> vChar;
typedef unsigned char uchar;
typedef vector<uchar> vuChar;
typedef vector<short> vShort;
typedef vector<int> vInt;
enum Player{
  BLACK=1, WHITE=-1
};
 /**
  * 局面の一座標の状態を表す列挙型
  * SML はサイズ、E は相手コマ、数が負の時は全てに E をつける、つまり ES は-1 で表す
  * 数が正ならは見えてるコマ（一座標上の最も大きいコマ）が自コマである
 **/
struct Ptype{
  enum {
    EMPTY=0,
    S=1,
    M=2,
    L=3,
    SM=4,
    SL=5,
    ML=6,
    ESM=7,
    ESL=8,
    EML=9,
    SML=10,
    ESML=11,
    SEML=12,
    ESEML=13,
  };
};
 /**
  * ゲームの一局面の情報を表すクラス
 **/
struct State{
    char board[3*3];//局面情報
    int stands[2][3];//持ちコマ情報
    int turn;//基本先手（黒=1）
    State(){//初期局面手持ちコマ全種類 2 つでコンストラクタ
```

```
  for(int x=0;x<3;x++)
    for(int y=0;y<3;y++)
    board[x*3+y]=Ptype::EMPTY;
  for(int i=0;i<3;i++){
    stands[0][i]=2;
    stands[1][i]=2;
  }
  turn=BLACK;
}
/**
* データ破損チェック用、コマ数に欠陥があると false
**/
bool check(){
  int all=0;
  for(int x=2;x>=0;x--)
    for(int y=2;y>=0;y--){
      if(board[x*3+y]==Ptype::EMPTY)continue;
      if(board[x*3+y]>=-3&&board[x*3+y]<=3){all++;
      }else if(board[x*3+y]>=-9&&board[x*3+y]<=9){all+=2;
      }else{all+=3;}
    }
  all+=stands[0][0];
  all+=stands[0][1];
  all+=stands[0][2];
  all+=stands[1][0];
  all+=stands[1][1];
  all+=stands[1][2];
  State news=rotateChangeTurn();
  if(news.isLoseBlack())return false;
  if(all==12)return true;

  return false;
}
/**
* show() で使用する関数群
**/
void ps(uint64& a,uint64 b)const{
  if(a%100==0){a+=b+1;
  }else if(a/10%10==0)a+=(b+1)*10;
}
```

```
void pm(uint64& a,uint64 b)const{
  if(a/100%100==0){a+=(b+1)*100;
  }else if(a/1000%10==0)a+=(b+1)*1000;
}
void pl(uint64& a,uint64 b)const{
  if(a/10000%100==0){a+=(b+1)*10000;
  }else if(a/100000%10==0)a+=(b+1)*100000;
}
void pes(uint64& a,uint64 b)const{
  if(a/1000000%100==0){a+=(b+1)*1000000;
  }else if(a/10000000%10==0)a+=(b+1)*10000000;
}
void pem(uint64& a,uint64 b)const{
  if(a/100000000%100==0){a+=(b+1)*100000000;
  }else if(a/1000000000%10==0)a+=(b+1)*1000000000;
}
void pel(uint64& a,uint64 b)const{
  if(a/10000000000%100==0){a+=(b+1)*10000000000;
  }else if(a/100000000000%10==0)a+=(b+1)*100000000000;
}
/**
* 論文で使用した表記法に基づいて局面を表示ただし後手が左で先手が右側
* 後手の大大中中小小（000000）と先手の大大中中小小（000000）の合計 12 桁
**/
uint64 show() const{
  uint64 a=0;
  for(int x=2;x>=0;x--)
    for(int y=2;y>=0;y--){
      if(board[x*3+y]==Ptype::EMPTY)continue;
      if(board[x*3+y]==Ptype::S){ps(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::S){pes(a,x*3+y);
      }else if(board[x*3+y]==Ptype::M){pm(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::M){pem(a,x*3+y);
      }else if(board[x*3+y]==Ptype::L){pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::L){pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::SM){ps(a,x*3+y);pm(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::SM){pes(a,x*3+y);pem(a,x*3+y);
      }else if(board[x*3+y]==Ptype::SL){ps(a,x*3+y);pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::SL){pes(a,x*3+y);pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::ML){pm(a,x*3+y);pl(a,x*3+y);
```

```
      }else if(board[x*3+y]==-Ptype::ML){pem(a,x*3+y);pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::ESM){pes(a,x*3+y);pm(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::ESM){ps(a,x*3+y);pem(a,x*3+y);
      }else if(board[x*3+y]==Ptype::ESL){pes(a,x*3+y);pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::ESL){ps(a,x*3+y);pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::EML){pem(a,x*3+y);pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::EML){pm(a,x*3+y);pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::SML){ps(a,x*3+y);pm(a,x*3+y);pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::SML){pes(a,x*3+y);pem(a,x*3+y);pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::ESML){pes(a,x*3+y);pm(a,x*3+y);pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::ESML){ps(a,x*3+y);pem(a,x*3+y);pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::SEML){ps(a,x*3+y);pem(a,x*3+y);pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::SEML){pes(a,x*3+y);pm(a,x*3+y);pel(a,x*3+y);
      }else if(board[x*3+y]==Ptype::ESEML){pes(a,x*3+y);pem(a,x*3+y);pl(a,x*3+y);
      }else if(board[x*3+y]==-Ptype::ESEML){ps(a,x*3+y);pm(a,x*3+y);pel(a,x*3+y);
      }
    }
    return a;
}
/**
* 表記法から生成するコンストラクタに使用する関数群
**/
void putS(uint64 a){
  board[a]=Ptype::S;
}
void putES(uint64 a){
  board[a]=-Ptype::S;
}
void putM(uint64 a){
  if(board[a]==Ptype::EMPTY){board[a]=Ptype::M;
  }else if(board[a]==Ptype::S){board[a]=Ptype::SM;
  }else if(board[a]==-Ptype::S){board[a]=Ptype::ESM;
  }
}
void putEM(uint64 a){
  if(board[a]==Ptype::EMPTY){board[a]=-Ptype::M;
  }else if(board[a]==Ptype::S){board[a]=-Ptype::ESM;
  }else if(board[a]==-Ptype::S){board[a]=-Ptype::SM;
  }
}
```

```
void putL(uint64 a){
  if(board[a]==Ptype::EMPTY){board[a]=Ptype::L;
  }else if(board[a]==Ptype::S){board[a]=Ptype::SL;
  }else if(board[a]==-Ptype::S){board[a]=Ptype::ESL;
  }else if(board[a]==Ptype::M){board[a]=Ptype::ML;
  }else if(board[a]==-Ptype::M){board[a]=Ptype::EML;
  }else if(board[a]==Ptype::SM){board[a]=Ptype::SML;
  }else if(board[a]==-Ptype::SM){board[a]=Ptype::ESEML;
  }else if(board[a]==Ptype::ESM){board[a]=Ptype::ESML;
  }else if(board[a]==-Ptype::ESM){board[a]=Ptype::SEML;
  }
}
void putEL(uint64 a){
  if(board[a]==Ptype::EMPTY){board[a]=-Ptype::L;
  }else if(board[a]==Ptype::S){board[a]=-Ptype::ESL;
  }else if(board[a]==-Ptype::S){board[a]=-Ptype::SL;
  }else if(board[a]==Ptype::M){board[a]=-Ptype::EML;
  }else if(board[a]==-Ptype::M){board[a]=-Ptype::ML;
  }else if(board[a]==Ptype::SM){board[a]=-Ptype::ESEML;
  }else if(board[a]==-Ptype::SM){board[a]=-Ptype::SML;
  }else if(board[a]==Ptype::ESM){board[a]=-Ptype::SEML;
  }else if(board[a]==-Ptype::ESM){board[a]=-Ptype::ESML;
  }
}
/**
* 表記法から生成するコンストラクタ->State.show() で得られる
* 後手の大大中中小小（000000）と先手の大大中中小小（000000）の合計 12 桁
* 引数に再現したい局面の数字と bool 型を入れて生成できる
**/
State(uint64 a,bool test)//数字に対して一桁ずつ確認し、局面を生成
{
  State s;
  //S
  if(a%100==0){
    s.stands[0][0]=2;
  }else if(a/10%10==0){
    s.stands[0][0]=1;
    s.putS(a%10-1);
  }else{
    s.stands[0][0]=0;
```

```
  s.putS(a%10-1);
  s.putS(a/10%10-1);
}
//ES
if(a/1000000%100==0){
  s.stands[1][0]=2;
}else if(a/10000000%10==0){
  s.stands[1][0]=1;
  s.putES(a/1000000%10-1);
}else{
  s.stands[1][0]=0;
  s.putES(a/1000000%10-1);
  s.putES(a/10000000%10-1);
}
//M
if(a/100%100==0){
  s.stands[0][1]=2;
}else if(a/1000%10==0){
  s.stands[0][1]=1;
  s.putM(a/100%10-1);
}else{
  s.stands[0][1]=0;
  s.putM(a/100%10-1);
  s.putM(a/1000%10-1);
}
//EM
if(a/100000000%100==0){
  s.stands[1][1]=2;
}else if(a/1000000000%10==0){
  s.stands[1][1]=1;
  s.putEM(a/100000000%10-1);
}else{
  s.stands[1][1]=0;
  s.putEM(a/100000000%10-1);
  s.putEM(a/1000000000%10-1);
}
//L
if(a/10000%100==0){
  s.stands[0][2]=2;
}else if(a/100000%10==0){
```

```
      s.stands[0][2]=1;
      s.putL(a/10000%10-1);
    }else{
      s.stands[0][2]=0;
      s.putL(a/10000%10-1);
      s.putL(a/100000%10-1);
    }
    //EL
    if(a/10000000000%100==0){
      s.stands[1][2]=2;
    }else if(a/100000000000%10==0){
      s.stands[1][2]=1;
      s.putEL(a/10000000000%10-1);
    }else{
      s.stands[1][2]=0;
      s.putEL(a/10000000000%10-1);
      s.putEL(a/100000000000%10-1);
    }
    *this=s;
  }
  /**
  * コンピューター上の局面データから生成するコンストラクタ->State.pack() で得られる
  * 57bit の 2 進数、解析ファイルは全てこの情報で保存されているのでこのコンストラクタを使う
  **/
  State(uint64 p,Player pl=BLACK)
  {
    if(pl==BLACK) *this=makeBlackFromUint64(p);
    else *this=makeBlackFromUint64(p).rotateChangeTurn();
  }
  static State makeBlackFromUint64(uint64 p){//バイナリデータから局面を生成する
    State s;
    int i=0;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
char c=(p>>(i*5))&31;
if((c&16)!=0) c-=32;
s.board[x*3+y]=c;
i++;
      }
    s.stands[0][0]=(p>>(45))&3;
```

```
    s.stands[0][1]=(p>>(45+2))&3;
    s.stands[0][2]=(p>>(45+4))&3;
    s.stands[1][0]=(p>>(45+6))&3;
    s.stands[1][1]=(p>>(45+8))&3;
    s.stands[1][2]=(p>>(45+10))&3;
    s.turn=BLACK;
    return s;
  }
  /**
  * ターンを変える、手を進めた時に使用
  * 解析では常に先手盤とするために毎回使用する
  **/
  State rotateChangeTurn() const
  {
    State ret;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
ret.board[x*3+y]= -board[x*3+y];

    ret.stands[0][0]=stands[1][0];
    ret.stands[0][1]=stands[1][1];
    ret.stands[0][2]=stands[1][2];
    ret.stands[1][0]=stands[0][0];
    ret.stands[1][1]=stands[0][1];
    ret.stands[1][2]=stands[0][2];
    ret.turn= -turn;
    return ret;
  }
  void changeTurn()
  {
    turn = -turn;
  }
  /**
  * 局面の左右を反転
  **/
  State sayuuHannten() const
  {
    State ret;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
```

```
ret.board[x*3+y]= board[x*3+(2-y)];

    ret.stands[0][0]=stands[0][0];
    ret.stands[0][1]=stands[0][1];
    ret.stands[0][2]=stands[0][2];
    ret.stands[1][0]=stands[1][0];
    ret.stands[1][1]=stands[1][1];
    ret.stands[1][2]=stands[1][2];
    return ret;
  }
  /**
  * 局面の上下を反転
  **/
  State zyougeHannten() const
  {
    State ret;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
ret.board[x*3+y]= board[(2-x)*3+y];

    ret.stands[0][0]=stands[0][0];
    ret.stands[0][1]=stands[0][1];
    ret.stands[0][2]=stands[0][2];
    ret.stands[1][0]=stands[1][0];
    ret.stands[1][1]=stands[1][1];
    ret.stands[1][2]=stands[1][2];
    return ret;
  }
  /**
  * 局面を右に一回転
  **/
  State kaiten() const
  {
    State ret;

  ret.board[0]= board[6];
    ret.board[1]= board[3];
    ret.board[2]= board[0];
    ret.board[3]= board[7];
    ret.board[4]= board[4];
```

```
    ret.board[5]= board[1];
    ret.board[6]= board[8];
    ret.board[7]= board[5];
    ret.board[8]= board[2];

    ret.stands[0][0]=stands[0][0];
    ret.stands[0][1]=stands[0][1];
    ret.stands[0][2]=stands[0][2];
    ret.stands[1][0]=stands[1][0];
    ret.stands[1][1]=stands[1][1];
    ret.stands[1][2]=stands[1][2];
    return ret;
  }
  /**
  * 局面のデータを 57bit2 進数で表す
 *一つのマスは大中小それぞれの状態が先手のコマ，後手のコマ，空の 3 種類の状態で 3 × 3 × 3 の 27 パターンであり，
 *5bit で表現できる．それが 9 つあるので 5 × 9 の 45bit となる．
 *局面はこの 45bit だけでも表現可能であるが，計算スピードを向上させるため手元のコマの情報も足す．
 *手元のコマの一種類の数は 0 か 1 か 2 の 3 パターンであり，2bit で表現できる．
 *それを 2 人のプレイヤーがそれぞれ 3 種類持つので 2 × 3 × 2 の 12bit を足して表現する．
  **/
  uint64 pack() const
  {
    assert(turn==BLACK);
    uint64 ret=0ull;
    int i=0;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
ret|=static_cast<uint64>(board[x*3+y]&31)<<(i*5);
i++;
      }
    for(int i=0;i<2;i++)
    for(int j=0;j<3;j++){
      ret|=static_cast<uint64>(stands[i][j])<<(45+(i*3+j)*2);
    }
    return ret;
  }
  /**
  * 1 つの局面は他 7 種類の局面と同じとみなせるので
```

```
* 対称性のある局面を全て出して bit 表現で最も低い数で正規化
**/
uint64 normalize() const{
  if(turn==BLACK){
    uint64 u1=pack();
    State a = sayuuHannten();
    uint64 u2=a.pack();
    u1=std::min(u1,u2);
    u2=a.kaiten().pack();
    u1=std::min(u1,u2);

    a = zyougeHannten();
    u2=a.pack();
    u1=std::min(u1,u2);
    u2=a.kaiten().pack();
    u1=std::min(u1,u2);

    a=kaiten();
    u2=a.pack();
    u1=std::min(u1,u2);
    a=a.kaiten();
    u2=a.pack();
    u1=std::min(u1,u2);
    a=a.kaiten();
    u2=a.pack();
    u1=std::min(u1,u2);
    return u1;
  }
  else{
    State news=rotateChangeTurn();
    uint64 u1=news.pack();
    State a = news.sayuuHannten();
    uint64 u2=a.pack();
    u1=std::min(u1,u2);
    u2=a.kaiten().pack();
    u1=std::min(u1,u2);

    a = news.zyougeHannten();
    u2=a.pack();
    u1=std::min(u1,u2);
```

```
    u2=a.kaiten().pack();
    u1=std::min(u1,u2);

    a=news.kaiten();
    u2=a.pack();
    u1=std::min(u1,u2);
    a=a.kaiten();
    u2=a.pack();
    u1=std::min(u1,u2);
    a=a.kaiten();
    u2=a.pack();
    u1=std::min(u1,u2);
    return u1;
  }
}
/**
* 局面の詰みを判定
**/
bool isLoseBlack() const{
  char boardJ[3*3];
  for(int x=0;x<3;x++)
    for(int y=0;y<3;y++){
      if(board[x*3+y]>=0){boardJ[x*3+y]=0;
      }else{boardJ[x*3+y]=-1;}
    }
  if(boardJ[0]+boardJ[1]+boardJ[2]==-3||
    boardJ[3]+boardJ[4]+boardJ[5]==-3||
    boardJ[6]+boardJ[7]+boardJ[8]==-3||
    boardJ[0]+boardJ[3]+boardJ[6]==-3||
    boardJ[1]+boardJ[4]+boardJ[7]==-3||
    boardJ[2]+boardJ[5]+boardJ[8]==-3||
    boardJ[0]+boardJ[4]+boardJ[8]==-3||
    boardJ[2]+boardJ[4]+boardJ[6]==-3
    )return true;
    return false;
}
/**
* nextStates() で使用する関数群
* 引数は次の局面を入れる配列とボードに配置するときにスキップさせる座標
**/
```

```
  void setS(vUint64& ret,int a=-1)const{
//9 マスのボードそれぞれに小コマを置いた場合の局面を返す
    if(stands[0][0]>0)
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
        if(x*3+y==a)continue;
        if(board[x*3+y]==Ptype::EMPTY){
            State s(*this);
              s.board[x*3+y]=Ptype::S;
              s.stands[0][0]--;
              s.changeTurn();
              ret.push_back(s.normalize());
        }
      }
  }
  void setM(vUint64& ret,int a=-1)const{
//9 マスのボードそれぞれに中コマを置いた場合の局面を返す
    if(stands[0][1]>0)
      for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
        if(x*3+y==a)continue;
          if(board[x*3+y]==Ptype::EMPTY){
            State s(*this);
              s.board[x*3+y]=Ptype::M;
              s.stands[0][1]--;
              s.changeTurn();
              ret.push_back(s.normalize());
          }else if(board[x*3+y]==Ptype::S){
             State s(*this);
              s.board[x*3+y]=Ptype::SM;
              s.stands[0][1]--;
              s.changeTurn();
              ret.push_back(s.normalize());
          }else if(board[x*3+y]==-Ptype::S){
             State s(*this);
              s.board[x*3+y]=Ptype::ESM;
              s.stands[0][1]--;
              s.changeTurn();
              ret.push_back(s.normalize());
          }
```

```
    }
  }
  void setL(vUint64& ret,int a=-1)const{
//9 マスのボードそれぞれに大コマを置いた場合の局面を返す
    if(stands[0][2]>0)
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
        if(x*3+y==a)continue;
        if(board[x*3+y]==Ptype::EMPTY){
            State s(*this);
              s.board[x*3+y]=Ptype::L;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.normalize());
        }else if(board[x*3+y]==Ptype::S){
            State s(*this);
              s.board[x*3+y]=Ptype::SL;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.normalize());
        }else if(board[x*3+y]==-Ptype::S){
            State s(*this);
              s.board[x*3+y]=Ptype::ESL;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.normalize());
        }else if(board[x*3+y]==Ptype::M){
            State s(*this);
              s.board[x*3+y]=Ptype::ML;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.normalize());
        }else if(board[x*3+y]==-Ptype::M){
            State s(*this);
              s.board[x*3+y]=Ptype::EML;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.normalize());
        }else if(board[x*3+y]==Ptype::SM){
            State s(*this);
```

```
            s.board[x*3+y]=Ptype::SML;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.normalize());
      }else if(board[x*3+y]==-Ptype::SM){
          State s(*this);
            s.board[x*3+y]=Ptype::ESEML;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.normalize());
      }else if(board[x*3+y]==Ptype::ESM){
          State s(*this);
            s.board[x*3+y]=Ptype::ESML;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.normalize());
      }else if(board[x*3+y]==-Ptype::ESM){
          State s(*this);
            s.board[x*3+y]=Ptype::SEML;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.normalize());
      }
    }
}
/**
* 次に遷移可能な局面を全て出す（57bit2 進数）
**/
vUint64 nextStates() const{
  vUint64 ret;
  setS(ret);//持ちコマを配置する場合
  setM(ret);
  setL(ret);
  //以降局面上のコマの移動の場合
  for(int x=0;x<3;x++)
    for(int y=0;y<3;y++){
      if(board[x*3+y]<=0)continue;
      if(board[x*3+y]==Ptype::S){
        State s(*this);
            s.board[x*3+y]=Ptype::EMPTY;
```

```
        //if(s.isLoseBlack())continue;
        s.stands[0][0]++;
        s.setS(ret,x*3+y);
}else if(board[x*3+y]==Ptype::M){
    State s(*this);
        s.board[x*3+y]=Ptype::EMPTY;
        //if(s.isLoseBlack())continue;
        s.stands[0][1]++;
        s.setM(ret,x*3+y);
}else if(board[x*3+y]==Ptype::L){
    State s(*this);
        s.board[x*3+y]=Ptype::EMPTY;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::SM){
    State s(*this);
        s.board[x*3+y]=Ptype::S;
        //if(s.isLoseBlack())continue;
        s.stands[0][1]++;
        s.setM(ret,x*3+y);
}else if(board[x*3+y]==Ptype::SL){
    State s(*this);
        s.board[x*3+y]=Ptype::S;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ML){
    State s(*this);
        s.board[x*3+y]=Ptype::M;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ESM){
    State s(*this);
        s.board[x*3+y]=-Ptype::S;
        if(s.isLoseBlack())continue;
        s.stands[0][1]++;
        s.setM(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ESL){
```

```
            State s(*this);
                s.board[x*3+y]=-Ptype::S;
                if(s.isLoseBlack())continue;
                s.stands[0][2]++;
                s.setL(ret,x*3+y);
        }else if(board[x*3+y]==Ptype::EML){
            State s(*this);
                s.board[x*3+y]=-Ptype::M;
                if(s.isLoseBlack())continue;
                s.stands[0][2]++;
                s.setL(ret,x*3+y);
        }else if(board[x*3+y]==Ptype::SML){
            State s(*this);
                s.board[x*3+y]=Ptype::SM;
                //if(s.isLoseBlack())continue;
                s.stands[0][2]++;
                s.setL(ret,x*3+y);
        }else if(board[x*3+y]==Ptype::ESML){
            State s(*this);
                s.board[x*3+y]=Ptype::ESM;
                //if(s.isLoseBlack())continue;
                s.stands[0][2]++;
                s.setL(ret,x*3+y);
        }else if(board[x*3+y]==Ptype::SEML){
            State s(*this);
                s.board[x*3+y]=-Ptype::ESM;
                if(s.isLoseBlack())continue;
                s.stands[0][2]++;
                s.setL(ret,x*3+y);
        }else if(board[x*3+y]==Ptype::ESEML){
            State s(*this);
                s.board[x*3+y]=-Ptype::SM;
                if(s.isLoseBlack())continue;
                s.stands[0][2]++;
                s.setL(ret,x*3+y);
                }
        }
        return ret;

}
```

```
/**
* nextDirections() で使用する関数群
* 引数は次の局面を入れる配列とボードに配置するときにスキップさせる座標
**/
  void setdS(vUint64& ret,int a=-1)const{
  if(stands[0][0]>0)
  for(int x=0;x<3;x++)
    for(int y=0;y<3;y++){
      if(x*3+y==a)continue;
      if(board[x*3+y]==Ptype::EMPTY){
          State s(*this);
            s.board[x*3+y]=Ptype::S;
            s.stands[0][0]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
      }
    }
}
void setdM(vUint64& ret,int a=-1)const{
  if(stands[0][1]>0)
    for(int x=0;x<3;x++)
    for(int y=0;y<3;y++){
      if(x*3+y==a)continue;
        if(board[x*3+y]==Ptype::EMPTY){
          State s(*this);
            s.board[x*3+y]=Ptype::M;
            s.stands[0][1]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
        }else if(board[x*3+y]==Ptype::S){
           State s(*this);
            s.board[x*3+y]=Ptype::SM;
            s.stands[0][1]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
        }else if(board[x*3+y]==-Ptype::S){
           State s(*this);
            s.board[x*3+y]=Ptype::ESM;
            s.stands[0][1]--;
            s.changeTurn();
```

```
            ret.push_back(s.rotateChangeTurn().show());
        }
  }
}
void setdL(vUint64& ret,int a=-1)const{
  if(stands[0][2]>0)
  for(int x=0;x<3;x++)
    for(int y=0;y<3;y++){
      if(x*3+y==a)continue;
      if(board[x*3+y]==Ptype::EMPTY){
          State s(*this);
            s.board[x*3+y]=Ptype::L;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
      }else if(board[x*3+y]==Ptype::S){
          State s(*this);
            s.board[x*3+y]=Ptype::SL;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
      }else if(board[x*3+y]==-Ptype::S){
          State s(*this);
            s.board[x*3+y]=Ptype::ESL;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
      }else if(board[x*3+y]==Ptype::M){
          State s(*this);
            s.board[x*3+y]=Ptype::ML;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
      }else if(board[x*3+y]==-Ptype::M){
          State s(*this);
            s.board[x*3+y]=Ptype::EML;
            s.stands[0][2]--;
            s.changeTurn();
            ret.push_back(s.rotateChangeTurn().show());
      }else if(board[x*3+y]==Ptype::SM){
```

```
            State s(*this);
              s.board[x*3+y]=Ptype::SML;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.rotateChangeTurn().show());
        }else if(board[x*3+y]==-Ptype::SM){
            State s(*this);
              s.board[x*3+y]=Ptype::ESEML;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.rotateChangeTurn().show());
        }else if(board[x*3+y]==Ptype::ESM){
            State s(*this);
              s.board[x*3+y]=Ptype::ESML;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.rotateChangeTurn().show());
        }else if(board[x*3+y]==-Ptype::ESM){
            State s(*this);
              s.board[x*3+y]=Ptype::SEML;
              s.stands[0][2]--;
              s.changeTurn();
              ret.push_back(s.rotateChangeTurn().show());
        }
      }
}
/**
* 次に遷移可能な局面を正規化せずに全て出す（表記法 12 桁）
* 次の遷移可能局面を人間が見てもわかるように出せる
**/
vUint64 nextDirections() const{
  vUint64 ret;
  setdS(ret);//持ちコマを配置する場合
  setdM(ret);
  setdL(ret);
  //以降局面上のコマを配置する場合
  for(int x=0;x<3;x++)
    for(int y=0;y<3;y++){
      if(board[x*3+y]<=0)continue;
      if(board[x*3+y]==Ptype::S){
```

```
    State s(*this);
        s.board[x*3+y]=Ptype::EMPTY;
        //if(s.isLoseBlack())continue;
        s.stands[0][0]++;
        s.setdS(ret,x*3+y);
}else if(board[x*3+y]==Ptype::M){
    State s(*this);
        s.board[x*3+y]=Ptype::EMPTY;
        //if(s.isLoseBlack())continue;
        s.stands[0][1]++;
        s.setdM(ret,x*3+y);
}else if(board[x*3+y]==Ptype::L){
    State s(*this);
        s.board[x*3+y]=Ptype::EMPTY;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::SM){
    State s(*this);
        s.board[x*3+y]=Ptype::S;
        //if(s.isLoseBlack())continue;
        s.stands[0][1]++;
        s.setdM(ret,x*3+y);
}else if(board[x*3+y]==Ptype::SL){
    State s(*this);
        s.board[x*3+y]=Ptype::S;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ML){
    State s(*this);
        s.board[x*3+y]=Ptype::M;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ESM){
    State s(*this);
        s.board[x*3+y]=-Ptype::S;
        if(s.isLoseBlack())continue;
        s.stands[0][1]++;
```

```
        s.setdM(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ESL){
    State s(*this);
        s.board[x*3+y]=-Ptype::S;
        if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::EML){
    State s(*this);
        s.board[x*3+y]=-Ptype::M;
        if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::SML){
    State s(*this);
        s.board[x*3+y]=Ptype::SM;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ESML){
    State s(*this);
        s.board[x*3+y]=Ptype::ESM;
        //if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::SEML){
    State s(*this);
        s.board[x*3+y]=-Ptype::ESM;
        if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
}else if(board[x*3+y]==Ptype::ESEML){
    State s(*this);
        s.board[x*3+y]=-Ptype::SM;
        if(s.isLoseBlack())continue;
        s.stands[0][2]++;
        s.setdL(ret,x*3+y);
        }
}
return ret;
```

```
  }
};
#endif
```

gobblet.cc

```
#include "gobblet.h"
```

allStateTable.h

```
#ifndef _ALL_STATE_TABLE_H
#define _ALL_STATE_TABLE_H
#include "gobblet.h"
#include <string>
using namespace std;
/**
 * 全ての局面のデータファイルを参照してインデックス等を返す
 */

class AllStateTable {
  vUint64 contents;
  size_t c_size;
  mutable ifstream ifs;
public:
 AllStateTable(size_t size) : contents(size),c_size(size) {}
  AllStateTable(string const& fileName,bool lazy=true);
  ~AllStateTable();
  size_t size() const{ return c_size; }
  /**
   * 局面のインデックスを返す、なければ-1
   */
  int find(uint64 v) const;
  /**
   * インデックスの局面データを返す
   */
  const uint64 operator[](size_t i) const {
    if(contents.size()>0){
      return contents[i];
    }
    else{
      ifs.seekg(i*sizeof(uint64),ios_base::beg);
      uint64 ret;
```

```
      ifs.read((char *)&ret,sizeof(uint64));
      return ret;
    }
  }
};
#endif
```

allStateTable.cc

```
#include "allStateTable.h"
#include <sys/types.h>
#include <sys/stat.h>
/**
 * 全ての局面のデータファイルを開く
 */
AllStateTable::AllStateTable(string const& fileName,bool lazy)
{
  struct stat st;
  if(stat(fileName.c_str(),&st)<0){
    abort();
  }
  c_size=st.st_size/sizeof(uint64);
  ifs.open(fileName.c_str(),std::ios::binary);
  if(lazy==false){
    contents.resize(c_size);
    ifs.read((char *)&contents[0],c_size*sizeof(uint64));
    ifs.close();
  }
}
//デコンストラクタ
AllStateTable::~AllStateTable()
{
  if(contents.size()==0){
    ifs.close();
  }
}
/**
 * 検索したい局面のインデックスを返す
 */
int AllStateTable::find(uint64 v) const
```

```
{
  int low=0, high=(int)c_size-1;
  while(low<high){
    int mid=(low+high)/2;
    uint64 v1=(*this)[mid];
    if(v1==v) return mid;
    else if(v1>v){ high=mid-1; }
    else {
      low=mid+1;
    }
  }
  if((*this)[low]!=v) return -1;
  return low;
}
```

winLoseTable.h

```
#ifndef _WIN_LOSE_TABLE_H
#define _WIN_LOSE_TABLE_H
#include "gobblet.h"
#include "allStateTable.h"
/**
* 局面の評価値をまとめたファイルと詰みまでの手数をまとめたファイルを開く
*/
class WinLoseTable {
  AllStateTable const& allS;
  vChar winLose;
  vuChar winLoseCount;
  mutable ifstream ifs1,ifs2;
public:
  WinLoseTable(AllStateTable const& allS_,
  string const& wlName, string const& wlcName,bool lazy=true);
  ~WinLoseTable();
  AllStateTable const& getAllS() const { return allS; }
 /**
 * インデックスの局面の評価値を返す
 */
  int getWinLose(size_t index) const {
    if(winLose.size()>0){
      return winLose[index];
```

```
    }
    else{
      ifs1.seekg(index,ios_base::beg);
      char ret;
      ifs1.read((char *)&ret,1);
      return ret;
    }
  }
 /**
 * インデックスの局面の詰みまでの手数を返
 */
  int getWinLoseCount(size_t index) const {
    if(winLoseCount.size()>0){
      return winLoseCount[index];
    }
    else{
      ifs2.seekg(index,ios_base::beg);
      unsigned char ret;
      ifs2.read((char *)&ret,1);
      return ret;
    }
  }
  int getWL(uint64 v, int& wlc) const;
  void showSequence(State const& s) const;
};
#endif
```

winLoseTable.cc

```
#include "winLoseTable.h"
#include <cassert>
#include <sys/types.h>
#include <sys/stat.h>
/**
 * 局面の評価値をまとめたファイルと詰みまでの手数をまとめたファイルを開く
 */
WinLoseTable::WinLoseTable(AllStateTable const& allS_,string const& wlName,
string const& wlcName,bool lazy)
  :allS(allS_)
{
  struct stat st1,st2;
```

```
  if(stat(wlName.c_str(),&st1)<0 || stat(wlcName.c_str(),&st2)<0){
    assert(0);
  }
  if(allS.size() != st1.st_size || st1.st_size != st2.st_size){
    std::cerr << "File size miss match " << wlName << " <=> " << wlcName << std::endl;
    assert(0);
  }
  ifs1.open(wlName.c_str(),std::ios::binary);
  ifs2.open(wlcName.c_str(),std::ios::binary);
  if(!lazy){
    winLose.resize(st1.st_size);
    winLoseCount.resize(st2.st_size);
    ifs1.read((char *)&winLose[0],st1.st_size);
    ifs2.read((char *)&winLoseCount[0],st2.st_size);
    ifs1.close();
    ifs2.close();
  }
}
//デコンストラクタ
WinLoseTable::~WinLoseTable(){
  if(winLose.size()==0){
    ifs1.close();
  }
  if(winLoseCount.size()==0){
    ifs2.close();
  }
}
/**
 * 評価値や詰み手数の更新
 */
int WinLoseTable::getWL(uint64 v,int& wlc) const
{
  int index=allS.find(v);
  wlc = (int)getWinLoseCount(index);
  return (int)getWinLose(index);
}
uint64 uinttoState(uint64 a){
    State s(a,BLACK);
    return s.show();
}
```

```
/**
 * 解析結果を表示するための関数
 * 局面を入力して遷移可能局面全てとその評価値及び詰み手数を表示
 */
void WinLoseTable::showSequence(State const& s) const
{
  cin.exceptions(ios::failbit);
  while(true){
  uint64 v=s.normalize();
  int index=allS.find(v);
  std::cerr << "------------------" << std::endl;
  std::cerr << s.show() <<"!!!"<<s.normalize()<<"???"<<index <<std::endl;
  std::cerr << (int)(s.turn==BLACK ? getWinLose(index) : -getWinLose(index)) <<
    "(" << (int)getWinLoseCount(index) << ")" << std::endl;
            vUint64 moves=s.nextStates();
            vUint64 directionInfos=s.nextDirections();
            if(moves.size()==0)break;
            int i=1;
            for(auto &a : moves){
              int wl,wlc;
              wl=getWL(a,wlc);
              std::cerr << i << " : " << directionInfos[i-1] << " "
            << -wl << "(" << wlc << ")" << std::endl;
            i++;
            }
            uint64 ia;
            try{
              cin>>ia;
            }catch(...){
              cin.clear();
              cin.seekg(0);
              continue;
            }
            if(ia==0)break;
            State news(ia,true);
            showSequence(news);
            break;
        }
}
```

makeAllState.cc

```
#include "gobblet.h"

#include <unordered_set>
typedef deque<uint64> dUint64;

struct hashUint64{
  int operator()(uint64 const& v) const{
    return (v>>32)^(v&0xffffffffull);
  }
};
typedef unordered_set<uint64,hashUint64> hUint64;
/**
 * 全ての局面を出すのに使用
 */
int main()
{
  vUint64 allIS;
  dUint64 q;
  hUint64 v;
  q.push_back(State().normalize());//初期局面をキューに入れる
  while(!q.empty()){
    uint64 is=q.front();
    q.pop_front();//キューから先頭を取り出す
    hUint64::const_iterator it=v.find(is);
    if(it==v.end()){//v に保存されてない場合（初めての局面の場合）
      State s(is,BLACK);
      if(!s.check())cout << s.show() << endl;
      allIS.push_back(is);
      v.insert(is);//v に保存
      if(!s.isLoseBlack()){//詰み局面でない場合
    vUint64 ns=s.nextStates();//次の局面を全て出す
      std::sort(ns.begin(), ns.end());
      ns.erase(std::unique(ns.begin(), ns.end()), ns.end());
    for(size_t i=0;i<ns.size();i++){
        q.push_back(ns[i]);//次の局面を全てキューに入れる
      }
      }
    }
```

```
    if(v.size()%1000000==0)cout << v.size() << endl;
  }
  cout << v.size() << endl;//以降データ表示
  sort(allIS.begin(),allIS.end());
  cout << allIS.size() << endl;
  ofstream ofs("allstates.dat",std::ios::out | std::ios::binary);
  ofs.write((char *)&allIS[0],allIS.size()*sizeof(uint64));//データ保存
}
```

makeWinLose.cc

```
#include "gobblet.h"
#include "allStateTable.h"
/**
 * 局面 (v) を評価する
 */
int newWinLoss(AllStateTable const& allIS,vChar const& winLoss,uint64 v){
  State s(v);
  vUint64 ns=s.nextStates();//次の局面を全て出す
  if(ns.size()==0)return -1;//次の局面がない場合負け局面とする
  bool alllose=true;
  for(size_t j=0;j<ns.size();j++){
    int i1=allIS.find(ns[j]);
    assert(0<=i1 && i1<allIS.size());
    if(winLoss[i1]== -1) return 1;//次の局面に一つでも勝ち局面がある場合，勝ち局面とする.
    if(winLoss[i1]==0) alllose=false;
  }
  if(alllose) return -1;//次の局面の全てが負け局面である場合，負け局面とする.
  else return 0;//その他の場合引き分け局面とする
}
/**
 * ゴブレットゴブラーズの後退解析に使用
 */
int main()
{
  AllStateTable allIS("allstates.dat",false);//全ての局面データ読み込み
  int dSize=allIS.size();//局面データサイズ
  cout << dSize << endl;
  vChar winLoss(dSize,0);
  vChar winLossCount(dSize,0);
  vInt count(3);
```

```
  for(size_t i=0;i<dSize;i++){//全ての局面で繰り返し
    State s(allIS[i]);
    if(s.isLoseBlack()){//局面が詰み局面の場合
      winLoss[i]=-1;
      count[0]++;//負け局面としてカウント
    }
    else count[1]++;//引き分け局面としてカウント
  }
  for(int c=1;;c++){//カウントに変化がなくなるまで繰り返し
    vChar winLossNew(winLoss);
    std::cout << "iteration " << c << std::endl;
    for(int j=0;j<3;j++){//負け引き分け勝ち局面のカウントを表示
      std::cout << (j-1) << " : " << count[j] << std::endl;
    }
    bool changed=false;
    for(size_t i=0;i<dSize;i++){//全ての局面で繰り返し
      if(winLoss[i]==0){//局面が引き分け局面の時
        int nv=newWinLoss(allIS,winLoss,allIS[i]);
        if(nv!=0){//局面の評価が引き分けでなかったとき
          winLossNew[i]=nv;//評価の更新
          winLossCount[i]=c;//カウントの更新
          count[nv+1]++; count[1]--;
          changed=true;//変化あり
        }
      }
    }
    winLoss.swap(winLossNew);//評価を更新した配列と交換して更新
    if(changed==false) break;
  }
  {
    ofstream ofs("winLoss.dat",std::ios::out | std::ios::binary);
    ofs.write((char *)&winLoss[0],winLoss.size());//評価値保存
  }
  {
    ofstream ofs("winLossCount.dat",std::ios::out | std::ios::binary);
    ofs.write((char *)&winLossCount[0],winLossCount.size());//詰み手数保存
  }
}
```

checkState.cc

```
#include "gobblet.h"
#include "allStateTable.h"
#include "winLoseTable.h"
#include <fstream>
#include <unordered_set>
typedef deque<uint64> dUint64;
typedef deque<char> dChar;

struct hashUint64{
  int operator()(uint64 const& v) const{
    return (v>>32)^(v&0xffffffffull);
  }
};
typedef unordered_set<uint64,hashUint64> hUint64;
void usage()
{
  std::cerr << "Usage: checkcsa csafile" << std::endl;
}
/**
 * winLose.showSequence(State) を使用して局面の評価を確認
 */
int main(int ac,char **ag)
{
  State s = State(000000000000,true);
  AllStateTable allS("allstates.dat");
  WinLoseTable winLose(allS,"winLoss.dat","winLossCount.dat");
  winLose.showSequence(s);
}
```

付録 B　ゲームアプリケーションに使用したソースプログラム

State.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using UnityEngine;
using UnityEngine.SceneManagement;
/**
```

```
  * 局面の一座標の状態を表す列挙型
  * SML はサイズ、E は相手コマ、数が負の時は全てに E をつける、つまり ES は-1 で表す
  * 数が正ならは見えてるコマ（一座標上の最も大きいコマ）が自コマである
 **/
public enum Ptype{
    EMPTY=0,
    S=1,
    M=2,
    L=3,
    SM=4,
    SL=5,
    ML=6,
    ESM=7,
    ESL=8,
    EML=9,
    SML=10,
    ESML=11,
    SEML=12,
    ESEML=13,
};
/**
  * 局面の状態を表すクラス
 **/
public class State{
    public int[] board = new int [3*3];//局面情報
    public int[,] stands = new int[2,3];//持ちコマ情報
    public bool firstTurn;//先手番かどうか
    public bool selecting;//コマが選択状態かどうか
    public int selectSize;//選択状態のコマの大きさ 1 小 2 中 3 大
    int banSet;//コマの元の位置であるために設置できない座標 1~9
    public State(){//コンストラクタ
    firstTurn=true;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
      board[x*3+y]=(int)Ptype.EMPTY;
    for(int i=0;i<3;i++){
      stands[0,i]=2;
      stands[1,i]=2;
      }
    }
```

```
    public State(State s){//複製用コンストラクタ
      firstTurn=s.firstTurn;
      for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
      board[x*3+y]=s.board[x*3+y];
      for(int i=0;i<3;i++){
      stands[0,i]=s.stands[0,i];
      stands[1,i]=s.stands[1,i];
      }
    }
    public State(ulong p){//58bit 用コンストラクタ
        int i=0;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
int c=(int)(p>>(i*5))&31;
if((c&16)!=0) c-=32;
board[x*3+y]=c;
i++;
      }
    stands[0,0]=(int)(p>>(45))&3;
    stands[0,1]=(int)(p>>(45+2))&3;
    stands[0,2]=(int)(p>>(45+4))&3;
    stands[1,0]=(int)(p>>(45+6))&3;
    stands[1,1]=(int)(p>>(45+8))&3;
    stands[1,2]=(int)(p>>(45+10))&3;
    firstTurn=true;
    }
    public State(ulong a,bool firstTurn){//表記法用コンストラクタ
    this.firstTurn=firstTurn;
    if(a%100==0){
      stands[0,0]=2;
    }else if(a/10%10==0){
      stands[0,0]=1;
      setS((int)(a%10-1));
    }else{
      stands[0,0]=0;
      setS((int)(a%10-1));
      setS((int)(a/10%10-1));
    }
    //ES
```

```
if(a/1000000%100==0){
  stands[1,0]=2;
}else if(a/10000000%10==0){
  stands[1,0]=1;
  setES((int)(a/1000000%10-1));
}else{
  stands[1,0]=0;
  setES((int)(a/1000000%10-1));
  setES((int)(a/10000000%10-1));
}
//M
if(a/100%100==0){
  stands[0,1]=2;
}else if(a/1000%10==0){
  stands[0,1]=1;
  setM((int)(a/100%10-1));
}else{
  stands[0,1]=0;
  setM((int)(a/100%10-1));
  setM((int)(a/1000%10-1));
}
//EM
if(a/100000000%100==0){
  stands[1,1]=2;
}else if(a/1000000000%10==0){
  stands[1,1]=1;
  setEM((int)(a/100000000%10-1));
}else{
  stands[1,1]=0;
  setEM((int)(a/100000000%10-1));
  setEM((int)(a/1000000000%10-1));
}
//L
if(a/10000%100==0){
  stands[0,2]=2;
}else if(a/100000%10==0){
  stands[0,2]=1;
  setL((int)(a/10000%10-1));
}else{
  stands[0,2]=0;
```

```
      setL((int)(a/10000%10-1));
      setL((int)(a/100000%10-1));
    }
    //EL
    if(a/10000000000%100==0){
      stands[1,2]=2;
    }else if(a/100000000000%10==0){
      stands[1,2]=1;
      setEL((int)(a/10000000000%10-1));
    }else{
      stands[1,2]=0;
      setEL((int)(a/10000000000%10-1));
      setEL((int)(a/100000000000%10-1));
    }
    }
    public void changeTurn(){//ターン変更
        firstTurn=!firstTurn;
    }
  public void rotateChangeTurn(){//ターン変更（先手と後手も入れ替わるのでゲーム中使用しない）
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
board[x*3+y]= -board[x*3+y];
    int stand;
    stand=stands[0,0];
    stands[0,0]=stands[1,0];
    stands[1,0]=stand;
    stand=stands[0,1];
    stands[0,1]=stands[1,1];
    stands[1,1]=stand;
    stand=stands[0,2];
    stands[0,2]=stands[1,2];
    stands[1,2]=stand;
    firstTurn= !firstTurn;
  }
/**
  * 表記法用関数郡 Debug でしか使わない
 **/
  void ps(ref ulong a,ulong b){
    if(a%100==0){a+=b+1;
    }else if(a/10%10==0)a+=(b+1)*10;
```

```
  }
  void pm(ref ulong a,ulong b){
    if(a/100%100==0){a+=(b+1)*100;
    }else if(a/1000%10==0)a+=(b+1)*1000;
  }
  void pl(ref ulong a,ulong b){
    if(a/10000%100==0){a+=(b+1)*10000;
    }else if(a/100000%10==0)a+=(b+1)*100000;
  }
  void pes(ref ulong a,ulong b){
    if(a/1000000%100==0){a+=(b+1)*1000000;
    }else if(a/10000000%10==0)a+=(b+1)*10000000;
  }
  void pem(ref ulong a,ulong b){
    if(a/100000000%100==0){a+=(b+1)*100000000;
    }else if(a/1000000000%10==0)a+=(b+1)*1000000000;
  }
  void pel(ref ulong a,ulong b){
    if(a/10000000000%100==0){a+=(b+1)*10000000000;
    }else if(a/100000000000%10==0)a+=(b+1)*100000000000;
  }
/**
  * 上記の表記法用関数郡を使用した局面情報可視化関数 Debug でしか使わない
 **/
  public ulong show(){
    ulong a=0;
    for(int x=2;x>=0;x--)
      for(int y=2;y>=0;y--){
        if(board[(ulong)((ulong)(x*3+y))]==(int)(int)Ptype.EMPTY)continue;
        if(board[(ulong)(x*3+y)]==(int)Ptype.S){ps(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.S){pes(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.M){pm(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.M){pem(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.L){pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.L){pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.SM)
        {ps(ref a,(ulong)(x*3+y));pm(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.SM)
        {pes(ref a,(ulong)(x*3+y));pem(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.SL)
```

```
        {ps(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.SL)
        {pes(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.ML)
        {pm(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.ML)
        {pem(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.ESM)
        {pes(ref a,(ulong)(x*3+y));pm(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.ESM)
        {ps(ref a,(ulong)(x*3+y));pem(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.ESL)
        {pes(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.ESL)
        {ps(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.EML)
        {pem(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.EML)
        {pm(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.SML)
        {ps(ref a,(ulong)(x*3+y));pm(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.SML)
        {pes(ref a,(ulong)(x*3+y));pem(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.ESML)
        {pes(ref a,(ulong)(x*3+y));pm(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.ESML)
        {ps(ref a,(ulong)(x*3+y));pem(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.SEML)
        {ps(ref a,(ulong)(x*3+y));pem(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.SEML)
        {pes(ref a,(ulong)(x*3+y));pm(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==(int)Ptype.ESEML)
        {pes(ref a,(ulong)(x*3+y));pem(ref a,(ulong)(x*3+y));pl(ref a,(ulong)(x*3+y));
        }else if(board[(ulong)(x*3+y)]==-(int)Ptype.ESEML)
        {ps(ref a,(ulong)(x*3+y));pm(ref a,(ulong)(x*3+y));pel(ref a,(ulong)(x*3+y));
        }
      }
      return a;
  }
/**
```

```
  * 局面情報及び持ちコマ情報可視化関数 Debug でしか使わない
 **/
  public void showDB(){

      for(int y=0;y<3;y++)
      for(int x=0;x<3;x++){
          Debug.Log(board[x+3*y]);
      }
      for(int i=0;i<3;i++){
      Debug.Log("stands"+stands[0,i]);
      Debug.Log("stands2"+stands[1,i]);
    }
  }
/**
  * 勝敗判定をする
 **/
  public bool isLoseBlack(){
      int[] boardJ=new int[3*3];
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
        if(board[x*3+y]>=0){boardJ[x*3+y]=0;
        }else{boardJ[x*3+y]=-1;}
      }
    if(boardJ[0]+boardJ[1]+boardJ[2]==-3||
      boardJ[3]+boardJ[4]+boardJ[5]==-3||
      boardJ[6]+boardJ[7]+boardJ[8]==-3||
      boardJ[0]+boardJ[3]+boardJ[6]==-3||
      boardJ[1]+boardJ[4]+boardJ[7]==-3||
      boardJ[2]+boardJ[5]+boardJ[8]==-3||
      boardJ[0]+boardJ[4]+boardJ[8]==-3||
      boardJ[2]+boardJ[4]+boardJ[6]==-3
      )return true;
      return false;
  }
/**
  * コマを選択状態にできるか判定
 **/
  public bool canSelect(int x,int y){
      if(x==3||x==-1){if(stands[0,y]>=1)return true;
      }else if(board[x+3*y]>=1&&board[x+3*y]<=13) return true;
```

```
      return false;
  }
/**
  * コマの選択状態をキャンセル、banSet に戻し、状態を更新
 **/
  public void canselSelect(){
    if(selecting==false)return;
    selecting=false;
    if(banSet==-1){stands[0,selectSize-1]++;}
  else if(board[banSet]==(int)Ptype.EMPTY&&selectSize==(int)Ptype.S){board[banSet]=(int)Ptype.S;}
  else if(board[banSet]==(int)Ptype.EMPTY&&selectSize==(int)Ptype.M){board[banSet]=(int)Ptype.M;}
  else if(board[banSet]==(int)Ptype.EMPTY&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.L;}
  else if(board[banSet]==(int)Ptype.S&&selectSize==(int)Ptype.M){board[banSet]=(int)Ptype.SM;}
  else if(board[banSet]==(int)Ptype.S&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.SL;}
  else if(board[banSet]==(int)Ptype.M&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.ML;}
  else if(board[banSet]==-(int)Ptype.S&&selectSize==(int)Ptype.M){board[banSet]=(int)Ptype.ESM;}
  else if(board[banSet]==-(int)Ptype.S&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.ESL;}
  else if(board[banSet]==-(int)Ptype.M&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.EML;}
  else if(board[banSet]==(int)Ptype.SM&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.SML;}
  else if(board[banSet]==(int)Ptype.ESM&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.ESML;}
  else if(board[banSet]==-(int)Ptype.ESM&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.SEML;}
  else if(board[banSet]==-(int)Ptype.SM&&selectSize==(int)Ptype.L){board[banSet]=(int)Ptype.ESEML;}
    else{Debug.Log("cancelfalse");}
    selectSize=0;
  }
/**
  * 引数の座標のコマを選択状態にする局面上のコマを手札に戻し、selectSize と banSet を更新
 **/
  public void selectPiece(int x,int y){
      banSet=x+3*y;
      if(x==3||x==-1){stands[0,y]--;banSet=-1;selecting=true;selectSize=y+1;}
      else if(board[x+3*y]==(int)Ptype.S)
{board[x+3*y]=(int)Ptype.EMPTY;selecting=true;selectSize=(int)Ptype.S;}
      else if(board[x+3*y]==(int)Ptype.M)
{board[x+3*y]=(int)Ptype.EMPTY;selecting=true;selectSize=(int)Ptype.M;}
      else if(board[x+3*y]==(int)Ptype.L)
{board[x+3*y]=(int)Ptype.EMPTY;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.SM)
{board[x+3*y]=(int)Ptype.S;selecting=true;selectSize=(int)Ptype.M;}
      else if(board[x+3*y]==(int)Ptype.SL)
```

```
{board[x+3*y]=(int)Ptype.S;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.ML)
{board[x+3*y]=(int)Ptype.M;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.ESM)
{board[x+3*y]=-(int)Ptype.S;selecting=true;selectSize=(int)Ptype.M;}
      else if(board[x+3*y]==(int)Ptype.ESL)
{board[x+3*y]=-(int)Ptype.S;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.EML)
{board[x+3*y]=-(int)Ptype.M;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.SML)
{board[x+3*y]=(int)Ptype.SM;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.ESML)
{board[x+3*y]=(int)Ptype.ESM;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.SEML)
{board[x+3*y]=-(int)Ptype.ESM;selecting=true;selectSize=(int)Ptype.L;}
      else if(board[x+3*y]==(int)Ptype.ESEML)
{board[x+3*y]=-(int)Ptype.SM;selecting=true;selectSize=(int)Ptype.L;}
      else{selecting=false;selectSize=0;}
  }
/**
  * 選択状態のコマを引数の座標にコマが配置可能か判断する
 **/
  public bool canSet(int x,int y){
      if(banSet==x+3*y)return false;
      int xy=Math.Abs(board[x+3*y]);
      if(xy==0)return true;
      if(selectSize==1){
          return false;
      }else if(selectSize==2){
          if(xy==1)return true;
      }else{
          if(xy==1||xy==2||xy==4||xy==7)return true;
      }
      return false;
  }
/**
  * 選択状態のコマを引数の座標に配置する
 **/
  public void setPiece(int x,int y){
      if(selectSize==1)setS(x+3*y);
```

```
        if(selectSize==2)setM(x+3*y);
        if(selectSize==3)setL(x+3*y);
        selecting=false;
    }
/**
    * コマの配置に必要な関数郡
 **/
    public void setS(int p){
        board[p]=(int)Ptype.S;
    }
    public void setES(int p){
        board[p]=-(int)Ptype.S;
    }
    public void setM(int p){
        if(board[p]==(int)Ptype.EMPTY)board[p]=(int)Ptype.M;
        if(board[p]==(int)Ptype.S)board[p]=(int)Ptype.SM;
        if(board[p]==-(int)Ptype.S)board[p]=(int)Ptype.ESM;
    }
    public void setEM(int p){
        if(board[p]==(int)Ptype.EMPTY)board[p]=-(int)Ptype.M;
        if(board[p]==(int)Ptype.S)board[p]=-(int)Ptype.ESM;
        if(board[p]==-(int)Ptype.S)board[p]=-(int)Ptype.SM;
    }
    public void setL(int p){
        if(board[p]==(int)Ptype.EMPTY)board[p]=(int)Ptype.L;
        if(board[p]==(int)Ptype.S)board[p]=(int)Ptype.SL;
        if(board[p]==-(int)Ptype.S)board[p]=(int)Ptype.ESL;
        if(board[p]==(int)Ptype.M)board[p]=(int)Ptype.ML;
        if(board[p]==-(int)Ptype.M)board[p]=(int)Ptype.EML;
        if(board[p]==(int)Ptype.SM)board[p]=(int)Ptype.SML;
        if(board[p]==-(int)Ptype.SM)board[p]=(int)Ptype.ESEML;
        if(board[p]==(int)Ptype.ESM)board[p]=(int)Ptype.ESML;
        if(board[p]==-(int)Ptype.ESM)board[p]=(int)Ptype.SEML;
    }
    public void setEL(int p){
        if(board[p]==(int)Ptype.EMPTY)board[p]=-(int)Ptype.L;
        if(board[p]==(int)Ptype.S)board[p]=-(int)Ptype.ESL;
        if(board[p]==-(int)Ptype.S)board[p]=-(int)Ptype.SL;
        if(board[p]==(int)Ptype.M)board[p]=-(int)Ptype.EML;
        if(board[p]==-(int)Ptype.M)board[p]=-(int)Ptype.ML;
```

```
      if(board[p]==(int)Ptype.SM)board[p]=-(int)Ptype.ESEML;
      if(board[p]==-(int)Ptype.SM)board[p]=-(int)Ptype.SML;
      if(board[p]==(int)Ptype.ESM)board[p]=-(int)Ptype.SEML;
      if(board[p]==-(int)Ptype.ESM)board[p]=-(int)Ptype.ESML;
  }
/**
  * 局面を左右反転する
 **/
  public State sayuuHanten()
  {
    State ret = new State();
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
ret.board[x*3+y]= board[x*3+(2-y)];

    ret.stands[0,0]=stands[0,0];
    ret.stands[0,1]=stands[0,1];
    ret.stands[0,2]=stands[0,2];
    ret.stands[1,0]=stands[1,0];
    ret.stands[1,1]=stands[1,1];
    ret.stands[1,2]=stands[1,2];
    return ret;
  }
/**
  * 局面を上下反転する
 **/
  public State zyougeHannten()
  {
    State ret = new State();
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++)
ret.board[x*3+y]= board[(2-x)*3+y];

    ret.stands[0,0]=stands[0,0];
    ret.stands[0,1]=stands[0,1];
    ret.stands[0,2]=stands[0,2];
    ret.stands[1,0]=stands[1,0];
    ret.stands[1,1]=stands[1,1];
    ret.stands[1,2]=stands[1,2];
    return ret;
```

```
  }
/**
  * 局面を右回転する
 **/
  public State kaiten()
  {
    State ret = new State();

ret.board[0]= board[6];
    ret.board[1]= board[3];
    ret.board[2]= board[0];
    ret.board[3]= board[7];
    ret.board[4]= board[4];
    ret.board[5]= board[1];
    ret.board[6]= board[8];
    ret.board[7]= board[5];
    ret.board[8]= board[2];

    ret.stands[0,0]=stands[0,0];
    ret.stands[0,1]=stands[0,1];
    ret.stands[0,2]=stands[0,2];
    ret.stands[1,0]=stands[1,0];
    ret.stands[1,1]=stands[1,1];
    ret.stands[1,2]=stands[1,2];
    return ret;
  }
/**
  * 局面を 58bit 整数にして返す
 **/
  public ulong pack()
  {
    ulong ret=0;
    int i=0;
    for(int x=0;x<3;x++)
      for(int y=0;y<3;y++){
ret|=(ulong)(board[x*3+y]&31)<<(i*5);
i++;
      }
    for(int x=0;x<2;x++)
    for(int y=0;y<3;y++){
```

```
      ret|=(ulong)(stands[x,y])<<(45+(x*3+y)*2);
    }
    return ret;
  }
/**
  * 局面を 58bit 整数かつ対称性のある局面を最も少ない数で正規化して返す
 **/
  public ulong normalize(){
      ulong u1=this.pack();
      State a = this.sayuuHanten();
      ulong u2=a.pack();
      u1=Math.Min(u1,u2);
      u2=a.kaiten().pack();
      u1=Math.Min(u1,u2);

      a = zyougeHannten();
      u2=a.pack();
      u1=Math.Min(u1,u2);
      u2=a.kaiten().pack();
      u1=Math.Min(u1,u2);

      a=kaiten();
      u2=a.pack();
      u1=Math.Min(u1,u2);
      a=a.kaiten();
      u2=a.pack();
      u1=Math.Min(u1,u2);
      a=a.kaiten();
      u2=a.pack();
      u1=Math.Min(u1,u2);
      return u1;
  }
/**
  * nextStates() で使用する関数郡
  * 引数は次の局面を入れる配列とボードに配置するときにスキップさせる座標
 **/
  void setkS(List<ulong> ret, int a = -1){
        if (stands[0,0] > 0)
            for (int x = 0; x < 3; x++)
                for (int y = 0; y < 3; y++) {
```

```
                if (x * 3 + y == a)continue;
                if (board[x * 3 + y] == (int)Ptype.EMPTY) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.S;
                    s.stands[0,0]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
            }
}
void setkM(List<ulong> ret, int a = -1){
    if (stands[0,1] > 0)
        for (int x = 0; x < 3; x++)
            for (int y = 0; y < 3; y++) {
                if (x * 3 + y == a)continue;
                if (board[x * 3 + y] == (int)Ptype.EMPTY) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.M;
                    s.stands[0,1]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
                else if (board[x * 3 + y] == (int)Ptype.S) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.SM;
                    s.stands[0,1]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
                else if (board[x * 3 + y] == -(int)Ptype.S) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.ESM;
                    s.stands[0,1]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
            }
}
void setkL(List<ulong> ret, int a = -1){
    if (stands[0,2] > 0)
```

```
for (int x = 0; x < 3; x++)
    for (int y = 0; y < 3; y++) {
        if (x * 3 + y == a)continue;
        if (board[x * 3 + y] == (int)Ptype.EMPTY) {
            State s=new State(this);
            s.board[x * 3 + y] = (int)Ptype.L;
            s.stands[0,2]--;
            s.rotateChangeTurn();
            ret.Add(s.normalize());
        }
        else if (board[x * 3 + y] == (int)Ptype.S) {
            State s=new State(this);
            s.board[x * 3 + y] = (int)Ptype.SL;
            s.stands[0,2]--;
            s.rotateChangeTurn();
            ret.Add(s.normalize());
        }
        else if (board[x * 3 + y] == -(int)Ptype.S) {
            State s=new State(this);
            s.board[x * 3 + y] = (int)Ptype.ESL;
            s.stands[0,2]--;
            s.rotateChangeTurn();
            ret.Add(s.normalize());
        }
        else if (board[x * 3 + y] == (int)Ptype.M) {
            State s=new State(this);
            s.board[x * 3 + y] = (int)Ptype.ML;
            s.stands[0,2]--;
            s.rotateChangeTurn();
            ret.Add(s.normalize());
        }
        else if (board[x * 3 + y] == -(int)Ptype.M) {
            State s=new State(this);
            s.board[x * 3 + y] = (int)Ptype.EML;
            s.stands[0,2]--;
            s.rotateChangeTurn();
            ret.Add(s.normalize());
        }
        else if (board[x * 3 + y] == (int)Ptype.SM) {
            State s=new State(this);
```

```
                    s.board[x * 3 + y] = (int)Ptype.SML;
                    s.stands[0,2]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
                else if (board[x * 3 + y] == -(int)Ptype.SM) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.ESEML;
                    s.stands[0,2]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
                else if (board[x * 3 + y] == (int)Ptype.ESM) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.ESML;
                    s.stands[0,2]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
                else if (board[x * 3 + y] == -(int)Ptype.ESM) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.SEML;
                    s.stands[0,2]--;
                    s.rotateChangeTurn();
                    ret.Add(s.normalize());
                }
            }
    }
/**
  * 次に遷移可能な局面を全て出す（57bit2 進数）
  * 引数は次の局面を入れる配列とボードに配置するときにスキップさせる座標
 **/
    public List<ulong> nextStates(){
        List<ulong> ret= new List<ulong>();
        setkS(ret);
        setkM(ret);
        setkL(ret);
        for (int x = 0; x < 3; x++)
            for (int y = 0; y < 3; y++) {
                if (board[x * 3 + y] <= 0)continue;
```

```
if (board[x * 3 + y] == (int)Ptype.S) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.EMPTY;
    //if(s.isLoseBlack())continue;
    s.stands[0,0]++;
    s.setkS(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.M) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.EMPTY;
    //if(s.isLoseBlack())continue;
    s.stands[0,1]++;
    s.setkM(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.L) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.EMPTY;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setkL(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.SM) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.S;
    //if(s.isLoseBlack())continue;
    s.stands[0,1]++;
    s.setkM(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.SL) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.S;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setkL(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.ML) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.M;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
```

```
    s.setkL(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.ESM) {
    State s=new State(this);
    s.board[x * 3 + y] = -(int)Ptype.S;
    if (s.isLoseBlack())continue;
    s.stands[0,1]++;
    s.setkM(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.ESL) {
    State s=new State(this);
    s.board[x * 3 + y] = -(int)Ptype.S;
    if (s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setkL(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.EML) {
    State s=new State(this);
    s.board[x * 3 + y] = -(int)Ptype.M;
    if (s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setkL(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.SML) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.SM;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setkL(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.ESML) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.ESM;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setkL(ret, x * 3 + y);
}
else if (board[x * 3 + y] == (int)Ptype.SEML) {
    State s=new State(this);
    s.board[x * 3 + y] = -(int)Ptype.ESM;
```

```
                    if (s.isLoseBlack())continue;
                    s.stands[0,2]++;
                    s.setkL(ret, x * 3 + y);
                }
                else if (board[x * 3 + y] == (int)Ptype.ESEML) {
                    State s=new State(this);
                    s.board[x * 3 + y] = -(int)Ptype.SM;
                    if (s.isLoseBlack())continue;
                    s.stands[0,2]++;
                    s.setkL(ret, x * 3 + y);
                }
            }
        return ret;

    }
/**
  * nextMoves() に必要な関数郡
  * 引数は次の局面を入れる配列とボードに配置するときにスキップさせる座標
 **/
  void setdS(List<int> ret, int a=0,int b=3){
      if (stands[0,0] > 0)
            for (int x = 0; x < 3; x++)
                for (int y = 0; y < 3; y++) {
                    if (x * 3 + y == a * 3 + b && b != 3)continue;
                    if (board[x * 3 + y] == (int)Ptype.EMPTY) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                }
  }
  void setdM(List<int> ret, int a = 1, int b = 3){
        if (stands[0,1] > 0)
            for (int x = 0; x < 3; x++)
                for (int y = 0; y < 3; y++) {
                    if (x * 3 + y == a * 3 + b && b != 3)continue;
                    if (board[x * 3 + y] == (int)Ptype.EMPTY) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == (int)Ptype.S) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
```

```
                else if (board[x * 3 + y] == -(int)Ptype.S) {
                    ret.Add(1000 * b + 100 * a + 10 * y + x);
                }
            }
    }
    void setdL(List<int> ret, int a = 2, int b = 3){
        if (stands[0,2] > 0)
            for (int x = 0; x < 3; x++)
                for (int y = 0; y < 3; y++) {
                    if (x * 3 + y == a * 3 + b && b != 3)continue;
                    if (board[x * 3 + y] == (int)Ptype.EMPTY) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == (int)Ptype.S) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == -(int)Ptype.S) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == (int)Ptype.M) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == -(int)Ptype.M) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == (int)Ptype.SM) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == -(int)Ptype.SM) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == (int)Ptype.ESM) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                    else if (board[x * 3 + y] == -(int)Ptype.ESM) {
                        ret.Add(1000 * b + 100 * a + 10 * y + x);
                    }
                }
    }
/**
```

```
 * 次に遷移可能な局面に移動する際に必要な選択コマ座標２桁＋設置コマ座標２桁をリストにして返す
**/
   public List<int> nextMoves(){
     var ret = new List<int>();
     setdS(ret);
       setdM(ret);
       setdL(ret);
       for (int x = 0; x < 3; x++)
           for (int y = 0; y < 3; y++) {
               if (board[x * 3 + y] <= 0)continue;
               if (board[x * 3 + y] == (int)Ptype.S) {
                   State s=new State(this);
                   s.board[x * 3 + y] = (int)Ptype.EMPTY;
                   //if(s.isLoseBlack())continue;
                   s.stands[0,0]++;
                   s.setdS(ret, x, y);
               }
               else if (board[x * 3 + y] == (int)Ptype.M) {
                   State s=new State(this);
                   s.board[x * 3 + y] = (int)Ptype.EMPTY;
                   //if(s.isLoseBlack())continue;
                   s.stands[0,1]++;
                   s.setdM(ret, x, y);
               }
               else if (board[x * 3 + y] == (int)Ptype.L) {
                   State s=new State(this);
                   s.board[x * 3 + y] = (int)Ptype.EMPTY;
                   //if(s.isLoseBlack())continue;
                   s.stands[0,2]++;
                   s.setdL(ret, x, y);
               }
               else if (board[x * 3 + y] == (int)Ptype.SM) {
                   State s=new State(this);
                   s.board[x * 3 + y] = (int)Ptype.S;
                   //if(s.isLoseBlack())continue;
                   s.stands[0,1]++;
                   s.setdM(ret, x, y);
               }
               else if (board[x * 3 + y] == (int)Ptype.SL) {
                   State s=new State(this);
```

```
    s.board[x * 3 + y] = (int)Ptype.S;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setdL(ret, x, y);
}
else if (board[x * 3 + y] == (int)Ptype.ML) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.M;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setdL(ret, x, y);
}
else if (board[x * 3 + y] == (int)Ptype.ESM) {
    State s=new State(this);
    s.board[x * 3 + y] = -(int)Ptype.S;
    if (s.isLoseBlack())continue;
    s.stands[0,1]++;
    s.setdM(ret, x, y);
}
else if (board[x * 3 + y] == (int)Ptype.ESL) {
    State s=new State(this);
    s.board[x * 3 + y] = -(int)Ptype.S;
    if (s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setdL(ret, x, y);
}
else if (board[x * 3 + y] == (int)Ptype.EML) {
    State s=new State(this);
    s.board[x * 3 + y] = -(int)Ptype.M;
    if (s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setdL(ret, x, y);
}
else if (board[x * 3 + y] == (int)Ptype.SML) {
    State s=new State(this);
    s.board[x * 3 + y] = (int)Ptype.SM;
    //if(s.isLoseBlack())continue;
    s.stands[0,2]++;
    s.setdL(ret, x, y);
}
```

```
                else if (board[x * 3 + y] == (int)Ptype.ESML) {
                    State s=new State(this);
                    s.board[x * 3 + y] = (int)Ptype.ESM;
                    //if(s.isLoseBlack())continue;
                    s.stands[0,2]++;
                    s.setdL(ret, x, y);
                }
                else if (board[x * 3 + y] == (int)Ptype.SEML) {
                    State s=new State(this);
                    s.board[x * 3 + y] = -(int)Ptype.ESM;
                    if (s.isLoseBlack())continue;
                    s.stands[0,2]++;
                    s.setdL(ret, x, y);
                }
                else if (board[x * 3 + y] == (int)Ptype.ESEML) {
                    State s=new State(this);
                    s.board[x * 3 + y] = -(int)Ptype.SM;
                    if (s.isLoseBlack())continue;
                    s.stands[0,2]++;
                    s.setdL(ret, x, y);
                }
            }
        return ret;

  }

}
```

AllStateTable.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System;
/**
* 全ての局面のデータファイルを参照してインデックス等を返す
*/
public class AllStateTable
{
    public FileStream ifs;
```

```
    private int c_size;
    public ulong this[int index]{
        get{
            ifs.Seek((long)index*sizeof(ulong),SeekOrigin.Begin);
            var buffer = new byte[sizeof(ulong)];
            ifs.Read(buffer,0,sizeof(ulong));
            return BitConverter.ToUInt64(buffer,0);
        }
    }
    public AllStateTable(string fileName){
        string path=Application.streamingAssetsPath + "/"+fileName;
        ifs = new FileStream(path, FileMode.Open);
        c_size=(int)(ifs.Length/sizeof(ulong));
        Debug.Log(c_size);
    }
    ~AllStateTable(){
        ifs.Close();
    }
/**
  * 局面のインデックスを返す、なければ-1
 **/
    public int find(ulong v){
        int low =0;
        int high=c_size-1;
        while(low<high){
            int mid=(low+high)/2;
            ulong v1=(ulong)(this[mid]);
            if(v1==v)return mid;
            else if(v1>v){
                high=mid-1;
            }else{
                low=mid+1;
            }
        }
        if(this[low]!=v)return -1;
        return low;
    }
}
```

WinLoseTable.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System;
/**
* 局面の評価値をまとめたファイルと詰みまでの手数をまとめたファイルを開く
*/
public class WinLoseTable
{
    AllStateTable allS;
    public FileStream ifs1;
    public FileStream ifs2;
    public int c_size;

    public WinLoseTable(AllStateTable allS,string wlName,string wlcName){
        this.allS=allS;
        string path=Application.streamingAssetsPath + "/"+wlName;
        ifs1 = new FileStream(path, FileMode.Open);
        c_size=(int)(ifs1.Length);
        path=Application.streamingAssetsPath + "/"+wlcName;
        ifs2 = new FileStream(path, FileMode.Open);
        c_size=(int)(ifs2.Length);
    }
    ~WinLoseTable(){
        ifs1.Close();
        ifs2.Close();
    }
/**
* インデックスの局面の評価値を返す
*/
    public int getWinLose(int index){
        ifs1.Seek(index,SeekOrigin.Begin);
        int ret = ifs1.ReadByte();
        return ret==255?-1:ret;
    }
/**
* インデックスの局面の詰みまでの手数を返す
*/
    public int getWinLoseCount(int index){
```

```
        ifs2.Seek(index,SeekOrigin.Begin);
        return ifs2.ReadByte();
    }
/**
* 局面のインデックスを調べて引数の評価値と詰みまでの手数を更新する
*/
    public void updateWL(ulong s,ref int wl,ref int wlc){
        int index = allS.find(s);
        wl=(int)getWinLose(index);
        wlc=(int)getWinLoseCount(index);
    }
/**
* 次の最善の手を探す
*/
    public int nextHand(State s){
        ulong v = s.normalize();
        int index = allS.find(v);
    if(getWinLose(index)>0) {//勝ち局面の場合
        List<ulong> nextStates = s.nextStates();
        List<int> nextMoves = s.nextMoves();
        if (nextStates.Count == 0)return 0;
        int i = 0;
        int min = 26;//最大手数 25 のため
        List<int> hand=new List<int>();
        foreach(ulong a in nextStates) {
            int wl=0;
            int wlc=0;
            updateWL(a,ref wl,ref wlc);
            if (min > wlc && -wl > 0) {//勝ち局面かつ詰み手数が最も短い
                min = wlc;
                hand.Clear();
                hand.Add(i);
            }
            else if (min == wlc && -wl > 0) {//勝ち局面かつ詰み手数が同じ
                hand.Add(i);
            }
            i++;
        }
        return nextMoves[hand[UnityEngine.Random.Range(0,hand.Count)]];
    }
```

```
    else {//負け局面の場合
        List<ulong> nextStates = s.nextStates();
        List<int> nextMoves = s.nextMoves();
        if (nextStates.Count == 0)return 0;
        int i = 0;
        int max = 0;
        List<int> hand=new List<int>();
        foreach(ulong a in nextStates) {
            int wl=0;
            int wlc=0;
            updateWL(a,ref wl,ref wlc);
            if (max < wlc) {//詰み手数が最も長い
                max = wlc;
                hand.Clear();
                hand.Add(i);
            }
            else if (max == wlc) {//詰み手数が同じ
                hand.Add(i);
            }
            i++;
        }
        return nextMoves[hand[UnityEngine.Random.Range(0,hand.Count)]];
    }
    }
/**
* 次の接待用の手を探す（負ける）
*/
    public int nextZakoHand(State s){
        Debug.Log("zako");
        ulong v = s.normalize();
        int index = allS.find(v);
        List<ulong> nextStates = s.nextStates();
        List<int> nextMoves = s.nextMoves();
        if (nextStates.Count == 0)return 0;
        int i = 0;
        int min = 26;
        List<int> hand=new List<int>();
        foreach(ulong a in nextStates) {
            int wl=0;
            int wlc=0;
```

```
            updateWL(a,ref wl,ref wlc);
            if (min > wlc && -wl < 0) {//詰み手数が最も短いかつ負ける
                min = wlc;
                hand.Clear();
                hand.Add(i);
            }
            else if (min == wlc && -wl < 0) {//詰み手数が同じかつ負ける
                hand.Add(i);
            }
            i++;
        }
        if(hand.Count==0)return nextMoves[UnityEngine.Random.Range(0,nextMoves.Count)];
        return nextMoves[hand[UnityEngine.Random.Range(0,hand.Count)]];

    }
}
```

GobbletManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
/**
* 手番のプレイヤーの種類用列挙型
*/
public enum Player{
  HUMAN=0, CPU=1
};
/**
* ゴブレットゴブラーズを行う
*/
public class GobbletManager : MonoBehaviour
{
    static List<List<GameObject>> boardSpace;
    static List<GameObject> Pieces;
    static State state;//局面情報を扱う
    static int firstTurnPlayer;
```

```
static int secondTurnPlayer;
static GameObject selectedPiece;
public GameObject pieceOrange;
public GameObject pieceBlue;
public GameObject cursor;
public Text osText;
public Text omText;
public Text olText;
public Text bsText;
public Text bmText;
public Text blText;

private bool gameOver;//ゲーム終了か
private bool moving;//移動中か
private GameManager gameManager;//ゲーム運営用マネージャー
private SaveManager saveManager;//セーブデータ用マネージャー
private SoundManager soundManager;
string gameOverCanvasName = "GameOverCanvas";

Canvas gameOverCanvas;
Canvas rennjuCanvas;
Button undoButton;
Text turnText;
MeshRenderer cursorRenderer;

AllStateTable allS;
WinLoseTable winLose;
private List<State> stateList;//局面保存用リスト

[SerializeField]
AudioClip bgm = null;
[SerializeField]
AudioClip setSe = null;
[SerializeField]
AudioClip gameStartSe = null;
[SerializeField]
AudioClip buttonSe = null;
[SerializeField]
AudioClip pickSe = null;
[SerializeField]
```

```
    AudioClip cancelSelctSe = null;
    [SerializeField]
    AudioClip noSe = null;
    [SerializeField]
    AudioClip cursorSe = null;

    bool AfterRennju;//連珠モード判定

    private int preCursorPosition;//カーソル位置 0~15

    void Start()//初期局面で初期化
    {
        gameManager = GameObject.Find("GameManager").GetComponent<GameManager>();
        saveManager =  GameObject.Find("GameManager").GetComponent<SaveManager>();
        soundManager = gameManager.GetComponent<SoundManager>();
        state=new State(gameManager.initialState);//初期局面の設定
        firstTurnPlayer=gameManager.player1;
        secondTurnPlayer=gameManager.player2;
        if(gameManager.rennju){//モード判定
            firstTurnPlayer=(int)Player.CPU;
            secondTurnPlayer=(int)Player.HUMAN;
            gameManager.settai=false;
        }
        AfterRennju=false;
//ゲームオブジェクト検索とリンク
        gameOverCanvas = GameObject.Find(gameOverCanvasName).GetComponent<Canvas>();
        rennjuCanvas = GameObject.Find("RennjuCanvas").GetComponent<Canvas>();
        undoButton = GameObject.Find("UndoButton").GetComponent<Button>();
        turnText = GameObject.Find("TurnText").GetComponent<Text>();
        osText = GameObject.Find("OSText").GetComponent<Text>();
        omText = GameObject.Find("OMText").GetComponent<Text>();
        olText = GameObject.Find("OLText").GetComponent<Text>();
        bsText = GameObject.Find("BSText").GetComponent<Text>();
        bmText = GameObject.Find("BMText").GetComponent<Text>();
        blText = GameObject.Find("BLText").GetComponent<Text>();
        cursorRenderer = GameObject.Find("Cursor").GetComponent<MeshRenderer>();
//ゲームオブジェクトの配置とオンオフ
        boardSpace = new List<List<GameObject>>();
        Pieces = new List<GameObject>();
        GameObject orange11 =
```

```
Instantiate(pieceOrange,new Vector3((float)(3.5),0,(float)(0.5)),pieceOrange.transform.rotation);
        GameObject orange12 =
Instantiate(pieceOrange,new Vector3((float)(3.5),0,(float)(0.5)),pieceOrange.transform.rotation);
        GameObject orange21 =
Instantiate(pieceOrange,new Vector3((float)(3.5),0,(float)(1.5)),pieceOrange.transform.rotation);
        orange21.transform.localScale = new Vector3(9,9,9);
        GameObject orange22 =
Instantiate(pieceOrange,new Vector3((float)(3.5),0,(float)(1.5)),pieceOrange.transform.rotation);
        orange22.transform.localScale = new Vector3(9,9,9);
        GameObject orange31 =
Instantiate(pieceOrange,new Vector3((float)(3.5),0,(float)(2.5)),pieceOrange.transform.rotation);
        orange31.transform.localScale = new Vector3(12,12,12);
        GameObject orange32 =
Instantiate(pieceOrange,new Vector3((float)(3.5),0,(float)(2.5)),pieceOrange.transform.rotation);
        orange32.transform.localScale = new Vector3(12,12,12);
        GameObject blue11 =
Instantiate(pieceBlue,new Vector3((float)(-.5),0,(float)(0.5)),pieceBlue.transform.rotation);
        GameObject blue12 =
Instantiate(pieceBlue,new Vector3((float)(-.5),0,(float)(0.5)),pieceBlue.transform.rotation);
        GameObject blue21 =
Instantiate(pieceBlue,new Vector3((float)(-.5),0,(float)(1.5)),pieceBlue.transform.rotation);
        blue21.transform.localScale = new Vector3(9,9,9);
        GameObject blue22 =
Instantiate(pieceBlue,new Vector3((float)(-.5),0,(float)(1.5)),pieceBlue.transform.rotation);
        blue22.transform.localScale = new Vector3(9,9,9);
        GameObject blue31 =
Instantiate(pieceBlue,new Vector3((float)(-.5),0,(float)(2.5)),pieceBlue.transform.rotation);
        blue31.transform.localScale = new Vector3(12,12,12);
        GameObject blue32 =
Instantiate(pieceBlue,new Vector3((float)(-.5),0,(float)(2.5)),pieceBlue.transform.rotation);
        blue32.transform.localScale = new Vector3(12,12,12);
        for(int i=0;i<15;i++){
            boardSpace.Add(new List<GameObject>());
        }
        stateList = new List<State>();
        stateList.Add(new State(gameManager.initialState));
        gameOver=false;
        moving=false;
        gameOverCanvas.enabled=false;
        rennjuCanvas.enabled=false;
```

```
        Pieces.Add(orange11);
        Pieces.Add(orange12);
        Pieces.Add(orange21);
        Pieces.Add(orange22);
        Pieces.Add(orange31);
        Pieces.Add(orange32);
        Pieces.Add(blue11);
        Pieces.Add(blue12);
        Pieces.Add(blue21);
        Pieces.Add(blue22);
        Pieces.Add(blue31);
        Pieces.Add(blue32);
        reflectBoard();
        allS = new AllStateTable("allstates.dat");
        winLose = new WinLoseTable(allS,"winLoss.dat","winLossCount.dat");

        if(bgm != null)//サウンド関連
        {
        soundManager.PlayBgmByName(bgm.name);
        }
        soundManager.PlaySeByName(gameStartSe.name);

        if(!gameManager.allowBackState)GameObject.Find("UndoButton").SetActive(false);
        if(gameManager.formatCount==2)start3Animation();
    }
/**
* デバッグ用局面保存をみる
*/
    public void debugList(){
        foreach(var a in stateList){
            Debug.Log(a.show());
        }
    }
/**
* 一手戻るボタンを押した処理
*/
    public void onBackState(){
        if(stateList.Count<=2){
            soundManager.PlaySeByName(noSe.name);
```

```
            debugList();
            onReset();
            reflectBoard();
        }else{
        soundManager.PlaySeByName(buttonSe.name);
        stateList.RemoveAt(stateList.Count-1);
        stateList.RemoveAt(stateList.Count-1);
        state=new State(stateList[stateList.Count-1]);
        reflectBoard();
        debugList();
        AfterRennju=false;
        gameOver=false;
        }
        if(gameManager.rennju&&stateList.Count<=4){
            firstTurnPlayer=(int)Player.CPU;
            secondTurnPlayer=(int)Player.HUMAN;
        }
    }
/**
* ポーズをする処理
*/
    public void onPose(){
        moving=true;
    }
/**
* ポーズを解除する処理
*/
    public void onUnpose(){
        moving=false;
        undoButton.interactable=true;
    }
/**
* リセットボタンを押した処理
*/
    public void onReset(){
        soundManager.PlaySeByName(buttonSe.name);
        gameOverCanvas.enabled=false;
        rennjuCanvas.enabled=false;
        undoButton.interactable=true;
        AfterRennju=false;
```

```
        state=new State(gameManager.initialState);
        reflectBoard();
        stateList.Clear();
        stateList.Add(new State(gameManager.initialState));
        gameOver=false;
        if(gameManager.rennju){
            firstTurnPlayer=(int)Player.CPU;
            secondTurnPlayer=(int)Player.HUMAN;
        }
    }
/**
* スタンドバイ画面に戻るボタンを押した処理
*/
    public void onMoveStandby(){
        soundManager.PlaySeByName(noSe.name);
        gameManager.standby=true;
        soundManager.StopBgm();
        gameManager.MoveToScene("Title");
    }
/**
* 連珠モードで手番交代ボタンを押した処理
*/
    public void onSwitchRennju(){
        firstTurnPlayer=(int)Player.HUMAN;
        secondTurnPlayer=(int)Player.CPU;
        undoButton.interactable=true;
    }

    // Update is called once per frame
    void Update()
    {
//カーソル用情報
        Ray mouse = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit info;
        if(moving)return;
        if(gameManager.rennju&&stateList.Count==4&&AfterRennju==false){
            moving=true;
            AfterRennju=true;
            rennjuCanvas.enabled=true;
            undoButton.interactable=false;
```

```
        }
        if(gameOver){//ゲームが終了したあと
            if(Input.GetMouseButtonDown(0)){
                if(Physics.Raycast(mouse,out info,500.0f)){
                    int x = (int)Mathf.Floor(info.point.x);
                    int y = (int)Mathf.Floor(info.point.z);
                    if(!(x<-1||x>=4||y<0||y>=3)){
                        gameOverCanvas.enabled=true;
                        undoButton.interactable=false;
                    };
                }
            }
            return;
        }else{//ゲーム中
            if(state.isLoseBlack()){//ゲームが終了の時
                gameOver=true;
                reflectBoard();
                if(!state.firstTurn){turnText.text=("オレンジの勝ち");
                turnText.color=new Color(1f,0.55f,0f,1.0f);}
                else{turnText.text=("ブルーの勝ち");turnText.color=new Color(0f,0.7f,0.99f,1.0f);}
                saveClear();
                return;
            }
        }//カーソル情報更新
        if(Physics.Raycast(mouse,out info,500.0f)){
            int x = (int)Mathf.Floor(info.point.x);
            int y = (int)Mathf.Floor(info.point.z);
            if((!(x<-1||x>=4||y<0||y>=3))){
                cursor.transform.position=new Vector3((float)(x+.5),0,(float)(y+.5));
                if(preCursorPosition!=getPlace(x,y)){
                    preCursorPosition=getPlace(x,y);
                    soundManager.PlaySeByName(cursorSe.name);
                }
            }else{
                cursor.transform.position=new Vector3((float)(x+.5),-90,(float)(y+.5));
            }
        }
//人間のターン
        if(((firstTurnPlayer==(int)Player.HUMAN&&state.firstTurn)//コマが選択状態でない
||(secondTurnPlayer==(int)Player.HUMAN&&!state.firstTurn))&&!state.selecting){
```

```
            if(Input.GetMouseButtonDown(0)){
                if(Physics.Raycast(mouse,out info,500.0f)){
                    int x = (int)Mathf.Floor(info.point.x);
                    int y = (int)Mathf.Floor(info.point.z);
                    if(!select(x,y))Debug.Log("this was not a valid select");
                }
            }

        }else if(((firstTurnPlayer==(int)Player.HUMAN&&state.firstTurn)//コマが選択状態
||(secondTurnPlayer==(int)Player.HUMAN&&!state.firstTurn))&&state.selecting){
            if(Input.GetMouseButtonDown(0)){
                if(Physics.Raycast(mouse,out info,500.0f)){
                    int x = (int)Mathf.Floor(info.point.x);
                    int y = (int)Mathf.Floor(info.point.z);
                    if(!move(x,y))Debug.Log("this was not a valid move");
                }
            }
        }else{//コンピューターのターン
            Debug.Log("AI Turn");
            superAI();
        }
    }
/**
* ゲームの局面情報の更新
*/
    void reflectBoard(){
        ulong a = state.show();
        Debug.Log("!!!!!!!!!"+state.show());
        Debug.Log(state.firstTurn);
        foreach(List<GameObject> obs in boardSpace){
                obs.Clear();
        }
        if(state.firstTurn){
            if(a/100000000000%10==0)
{boardSpace[14].Add(Pieces[10]);}else{;boardSpace[(int)(a/100000000000%10-1)].Add(Pieces[10]);}
            if(a/10000000000%10==0)
{boardSpace[14].Add(Pieces[11]);}else{boardSpace[(int)(a/10000000000%10-1)].Add(Pieces[11]);}
            if(a/100000%10==0)
{boardSpace[11].Add(Pieces[4]);}else{boardSpace[(int)(a/100000%10-1)].Add(Pieces[4]);}
            if(a/10000%10==0)
```

```
{boardSpace[11].Add(Pieces[5]);}else{boardSpace[(int)(a/10000%10-1)].Add(Pieces[5]);}

            if(a/1000000000%10==0)
{boardSpace[13].Add(Pieces[8]);}else{boardSpace[(int)(a/1000000000%10-1)].Add(Pieces[8]);}
            if(a/100000000%10==0)
{boardSpace[13].Add(Pieces[9]);}else{boardSpace[(int)(a/100000000%10-1)].Add(Pieces[9]);}
            if(a/1000%10==0)
{boardSpace[10].Add(Pieces[2]);}else{boardSpace[(int)(a/1000%10-1)].Add(Pieces[2]);}
            if(a/100%10==0)
{boardSpace[10].Add(Pieces[3]);}else{boardSpace[(int)(a/100%10-1)].Add(Pieces[3]);}

            if(a/10000000%10==0)
{boardSpace[12].Add(Pieces[6]);}else{boardSpace[(int)(a/10000000%10-1)].Add(Pieces[6]);}
            if(a/1000000%10==0)
{boardSpace[12].Add(Pieces[7]);}else{boardSpace[(int)(a/1000000%10-1)].Add(Pieces[7]);}
            if(a/10%10==0)
{boardSpace[9].Add(Pieces[0]);}else{boardSpace[(int)(a/10%10-1)].Add(Pieces[0]);}
            if(a/1%10==0)
{boardSpace[9].Add(Pieces[1]);}else{boardSpace[(int)(a/1%10-1)].Add(Pieces[1]);}
        }else{
            if(a/100000000000%10==0)
{boardSpace[11].Add(Pieces[4]);}else{boardSpace[(int)(a/100000000000%10-1)].Add(Pieces[4]);}
            if(a/10000000000%10==0)
{boardSpace[11].Add(Pieces[5]);}else{boardSpace[(int)(a/10000000000%10-1)].Add(Pieces[5]);}
            if(a/100000%10==0)
{boardSpace[14].Add(Pieces[10]);}else{boardSpace[(int)(a/100000%10-1)].Add(Pieces[10]);}
            if(a/10000%10==0)
{boardSpace[14].Add(Pieces[11]);}else{boardSpace[(int)(a/10000%10-1)].Add(Pieces[11]);}

            if(a/1000000000%10==0)
{boardSpace[10].Add(Pieces[2]);}else{boardSpace[(int)(a/1000000000%10-1)].Add(Pieces[2]);}
            if(a/100000000%10==0)
{boardSpace[10].Add(Pieces[3]);}else{boardSpace[(int)(a/100000000%10-1)].Add(Pieces[3]);}
            if(a/1000%10==0)
{boardSpace[13].Add(Pieces[8]);}else{boardSpace[(int)(a/1000%10-1)].Add(Pieces[8]);}
            if(a/100%10==0)
{boardSpace[13].Add(Pieces[9]);}else{boardSpace[(int)(a/100%10-1)].Add(Pieces[9]);}

            if(a/10000000%10==0)
{boardSpace[9].Add(Pieces[0]);}else{boardSpace[(int)(a/10000000%10-1)].Add(Pieces[0]);}
```

```
            if(a/1000000%10==0)
{boardSpace[9].Add(Pieces[1]);}else{boardSpace[(int)(a/1000000%10-1)].Add(Pieces[1]);}
            if(a/10%10==0)
{boardSpace[12].Add(Pieces[6]);}else{boardSpace[(int)(a/10%10-1)].Add(Pieces[6]);}
            if(a/1%10==0)
{boardSpace[12].Add(Pieces[7]);}else{boardSpace[(int)(a/1%10-1)].Add(Pieces[7]);}
        }

        for(int y=0;y<3;y++){
            for(int x=0;x<3;x++){
                foreach(GameObject ob in boardSpace[x+3*y]){
                ob.transform.position=new Vector3((float)(x+.5),0,(float)(y+.5));
                }
            }
        }
        for(int p=9;p<=11;p++){
            foreach(GameObject ob in boardSpace[p]){
                ob.transform.position=new Vector3((float)(3+.5),0,(float)(p-9+.5));
            }
        }
        for(int p=12;p<=14;p++){
            foreach(GameObject ob in boardSpace[p]){
                ob.transform.position=new Vector3((float)(-1+.5),0,(float)(p-12+.5));
            }
        }
        osText.text=boardSpace[9].Count.ToString();
        omText.text=boardSpace[10].Count.ToString();
        olText.text=boardSpace[11].Count.ToString();
        bsText.text=boardSpace[12].Count.ToString();
        bmText.text=boardSpace[13].Count.ToString();
        blText.text=boardSpace[14].Count.ToString();
        if(state.firstTurn){
            cursorRenderer.material.color = new Color(1f,0.55f,0f,1.0f);
        }else{
            cursorRenderer.material.color = new Color(0f,0.7f,0.99f,1.0f);
        }
        if(state.firstTurn)
        {turnText.text=("オレンジのターン");turnText.color=new Color(1f,0.55f,0f,1.0f);}
        else{turnText.text=("ブルーのターン");turnText.color=new Color(0f,0.7f,0.99f,1.0f);}
    }
```

```
/**
* コマを選択状態にする->更新
*/
    bool select(int x,int y){
        if(state.firstTurn&&(x<0||x>=4||y<0||y>=3))return false;
        if(!state.firstTurn&&(x<-1||x>=3||y<0||y>=3))return false;
        if(x==-1&&!state.firstTurn)x=3;

        if(!state.canSelect(x,y))return false;

        state.selectPiece(x,y);
        selectedPiece=boardSpace[getPlace(x,y)][0];
        boardSpace[getPlace(x,y)].RemoveAt(0);
        selectedPiece.transform.position+=new Vector3(0,1,0);
        soundManager.PlaySeByName(pickSe.name);
        return true;
    }
/**
* 選択状態のコマを配置する->更新
*/
    bool move(int x,int y){
        if(x<0||x>=3||y<0||y>=3)
{state.canselSelect();reflectBoard();soundManager.PlaySeByName(cancelSelctSe.name);return false;}

        if(!state.canSet(x,y))
{state.canselSelect();reflectBoard();soundManager.PlaySeByName(cancelSelctSe.name);return false;}
        moving=true;
        StartCoroutine(dealMoving(x,y));
        return true;
    }
/**
* 配置アニメーション処理
*/
    public IEnumerator dealMoving(int x,int y){
        state.setPiece(x,y);
        float currentTime=0;
        Vector3 target =new Vector3((float)(x+.5),0,(float)(y+.5));
        Vector3 startPosition = selectedPiece.transform.position;
        while(currentTime<0.2f){
            currentTime+=Time.deltaTime;
```

```
            selectedPiece.transform.position = Vector3.Lerp(startPosition,target,currentTime/0.2f);
            yield return null;
        }
        boardSpace[getPlace(x,y)].Insert(0,selectedPiece);
        selectedPiece=null;
        state.rotateChangeTurn();
        stateList.Add(new State(state));
        if(stateList.Count==100)gameOver=true;
        reflectBoard();
        soundManager.PlaySeByName(setSe.name);
        moving=false;

    }
/**
* AI 処理、通常と接待
*/
    void superAI(){
        int hand=0;
        if(gameManager.settai){hand=winLose.nextZakoHand(state);}
        else if(gameManager.rennju&&stateList.Count<=3&&UnityEngine.Random.value<0.5)
{hand=winLose.nextZakoHand(state);}
        else{hand=winLose.nextHand(state);}
        int x=(int)hand/1000%10;
        if(!state.firstTurn&&x==3)x=-1;
        int y=(int)hand/100%10;
        select(x,y);
        x=(int)hand/10%10;
        y=(int)hand%10;
        move(x,y);
    }
/**
* xy 座標を 0~14 に直す
*/
    int getPlace(int x,int y){

        if(x>=0&&x<3&&y>=0&&y<3){
            return x+3*y;
        }else if(x==3&&state.firstTurn){
            return 9+y;
        }else return 12+y;
```

```
    }
/**
* データセーブ用関数
*/
    private void saveClear(){
        if(gameManager.formatCount==0){
            if(!state.firstTurn&&!gameManager.settai&&gameManager.player1==0
&&gameManager.player2==1){
                saveManager.clear1 = true;
                saveManager.Save();
            }
        }else if(gameManager.formatCount==1){
            if(state.firstTurn&&!gameManager.settai&&gameManager.player1==1
&&gameManager.player2==0){
                saveManager.clear2 = true;
                saveManager.Save();
            }
        }else if(gameManager.formatCount==2){
            if(state.firstTurn&&!gameManager.settai&&gameManager.player1==1
&&gameManager.player2==0){
                saveManager.clear3 = true;
                saveManager.Save();
            }
        }
    }
/**
* 初期局面 3 のアニメーション用関数
*/
    private void start3Animation(){
        selectedPiece=boardSpace[3][0];
        StartCoroutine("start3");

    }
    public IEnumerator start3(){
        float currentTime=0;
        Vector3 target =new Vector3(0.5f,0f,1.5f);
        selectedPiece.transform.position += new Vector3(0f,2f,0f);
        yield return new WaitForSeconds(1.0f);
        Vector3 startPosition = selectedPiece.transform.position;
        while(currentTime<0.5f){
```

```
            currentTime+=Time.deltaTime;
            selectedPiece.transform.position = Vector3.Lerp(startPosition,target,currentTime/0.5f);
            yield return null;
        }
        selectedPiece=null;
    }
}
```