オセロゲーム補足資料

目次

概要（内容）

■内容

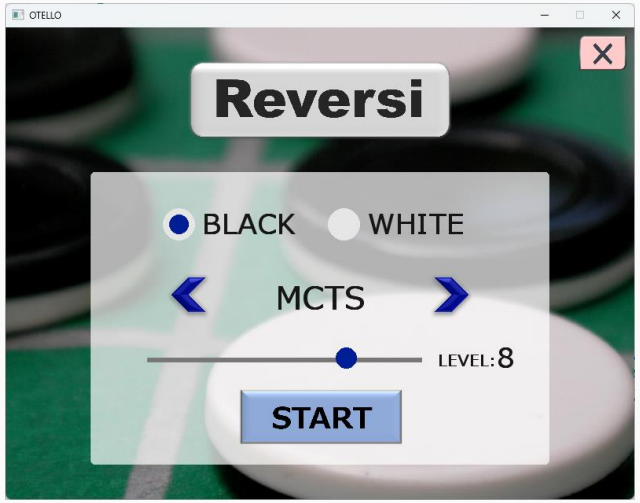
・AIと対戦できるオセロゲーム
・AIの強さを調整することも可能

■開発環境

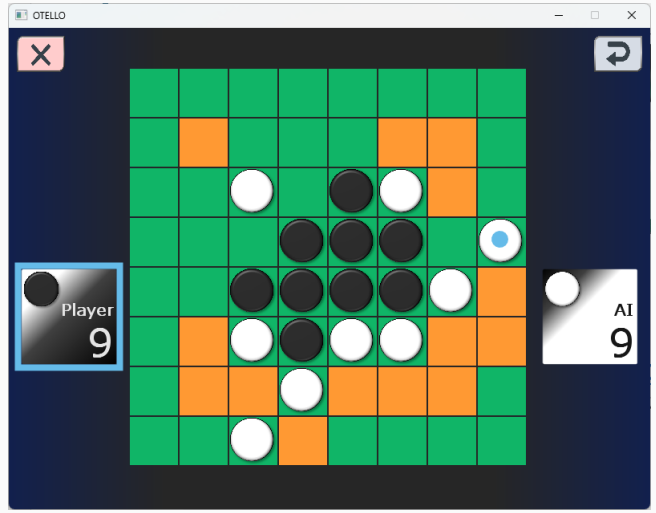
・VisualC++を利用して一人で開発
・DXライブラリを使用(https://dxlib.xsrv.jp/)
・ゲーム内の画像や音声はフリー素材や
PowerPointを使用して作成したもの

■開発期間

・70時間程度



タイトル画面



ゲーム画面

概要（操作方法）

■実行方法

Othello.exeを**pic/**と**sound/**フォルダと同一ディレクトリに
配置して実行

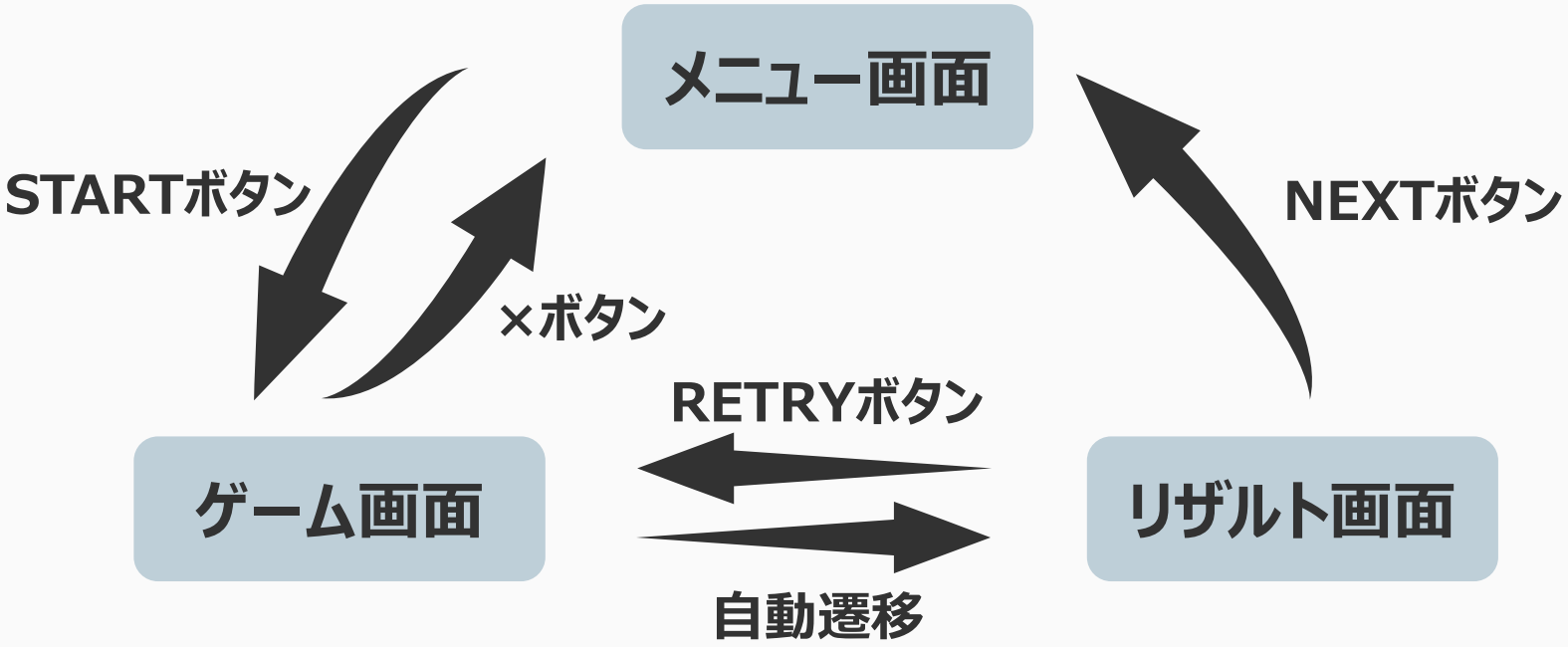
※終了時はWindows標準の×ボタンではなく、メニュー画面右上ゲーム画面内の×ボタンを押す

■操作方法

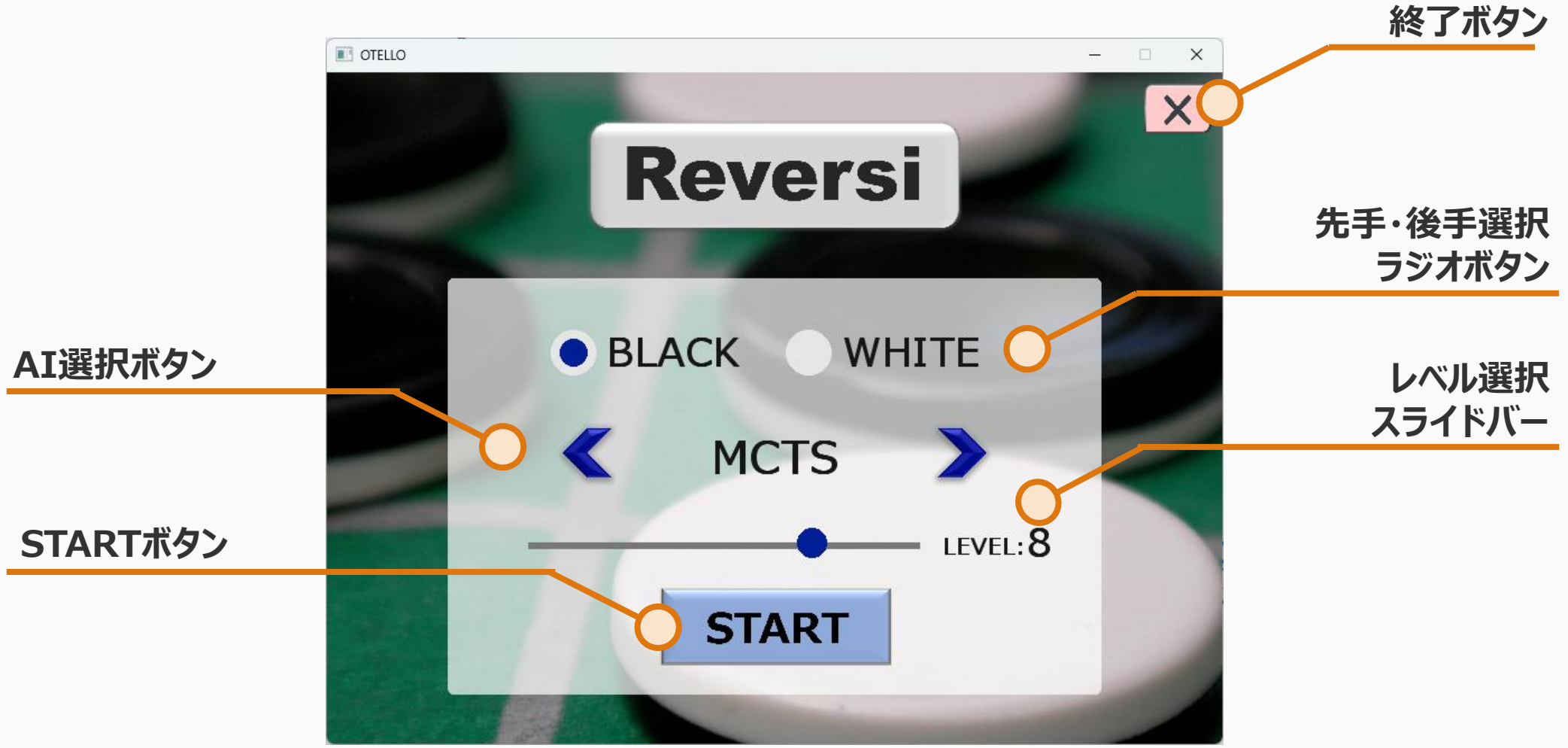
マウスの左クリックですべての操作を行う

概要（画面遷移）

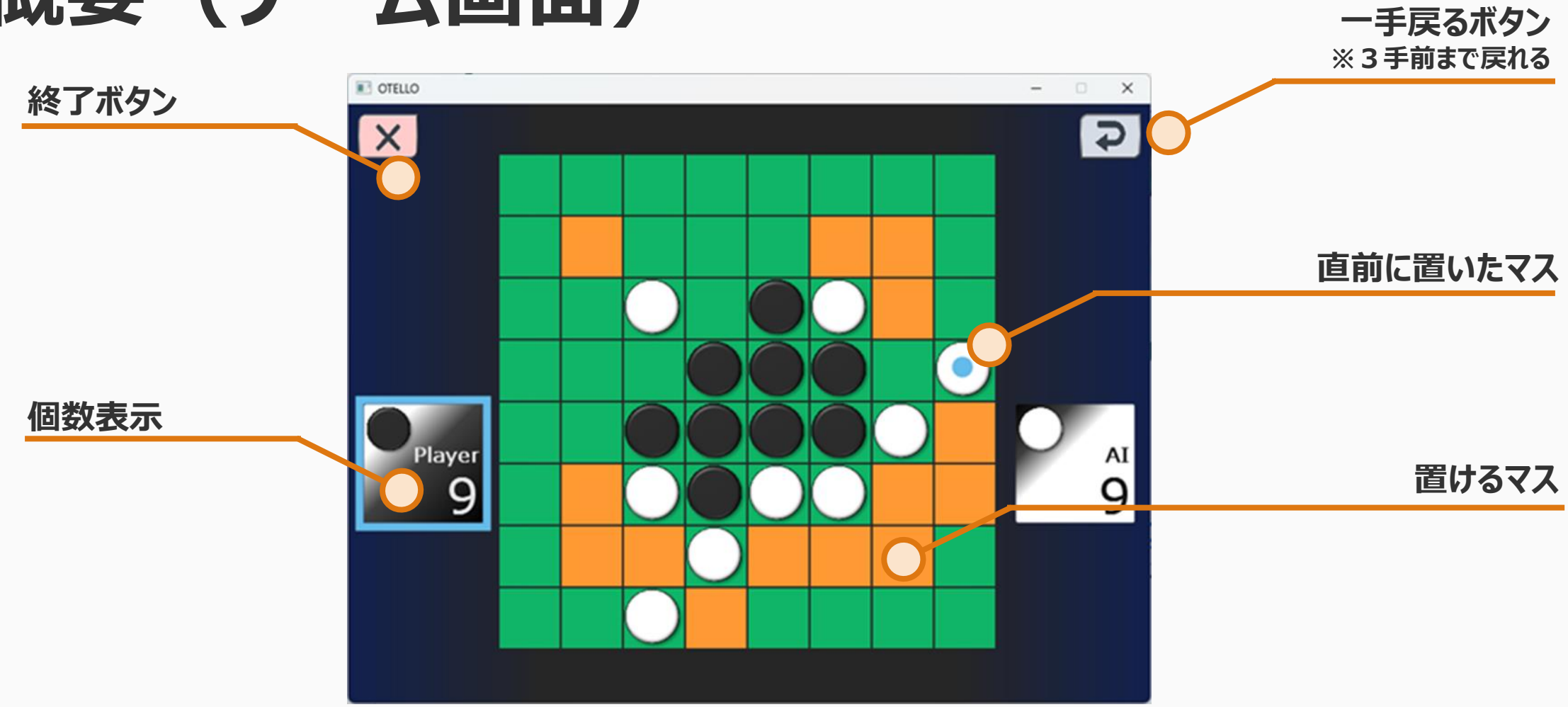
■メニュー画面、ゲーム画面、リザルト画面を遷移



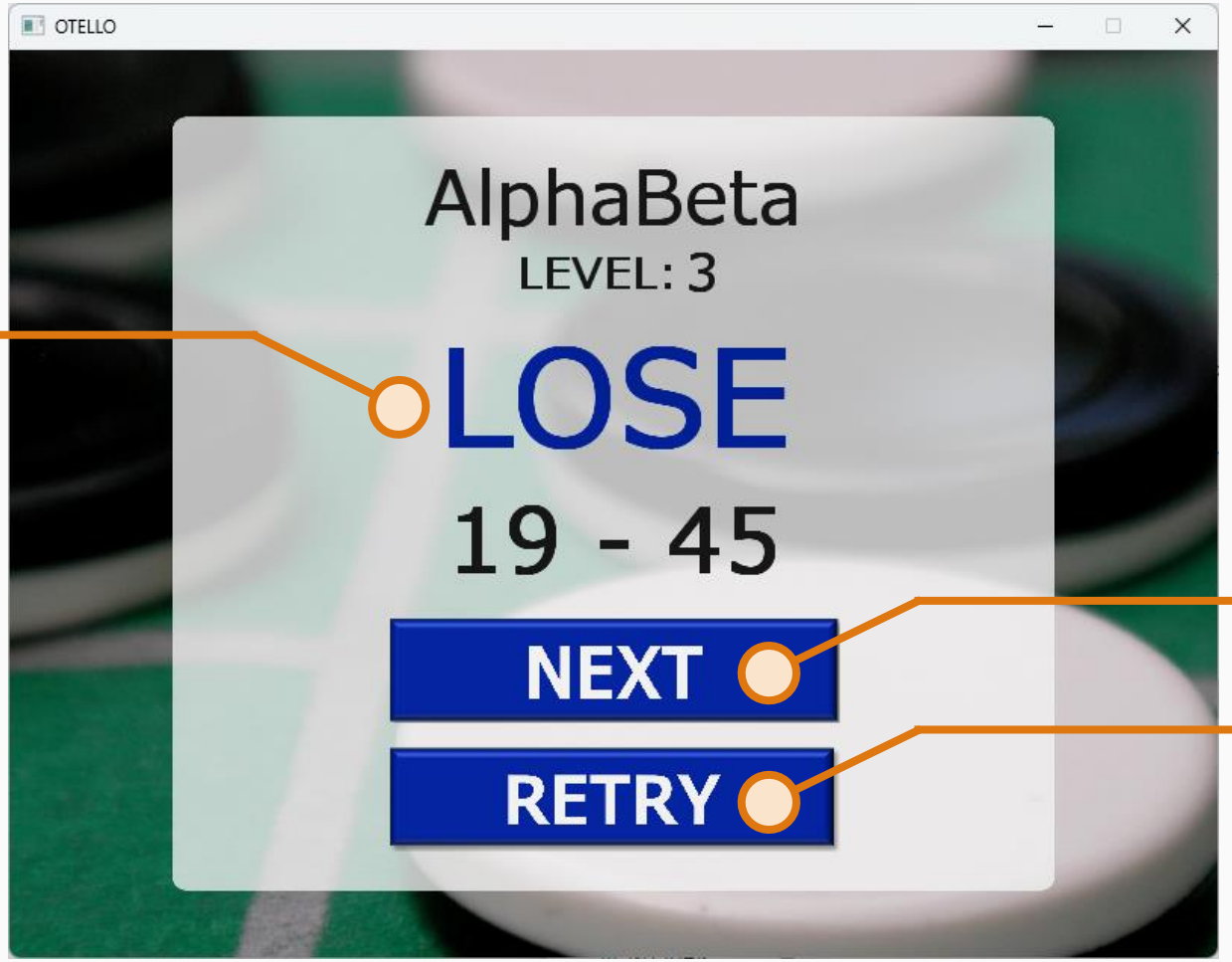
概要（メニュー画面）


終了ボタン
OTELLO
Reversi
先手・後手選択
ラジオボタン
BLACK
WHITE
AI選択ボタン
MCTS
レベル選択
スライドバー
LEVEL:8
STARTボタン
START

概要（ゲーム画面）


終了ボタン
一手戻るボタン
※３手前まで戻れる
直前に置いたマス
個数表示
置けるマス
OTELLO
Player
9
AI
9

概要（リザルト画面）

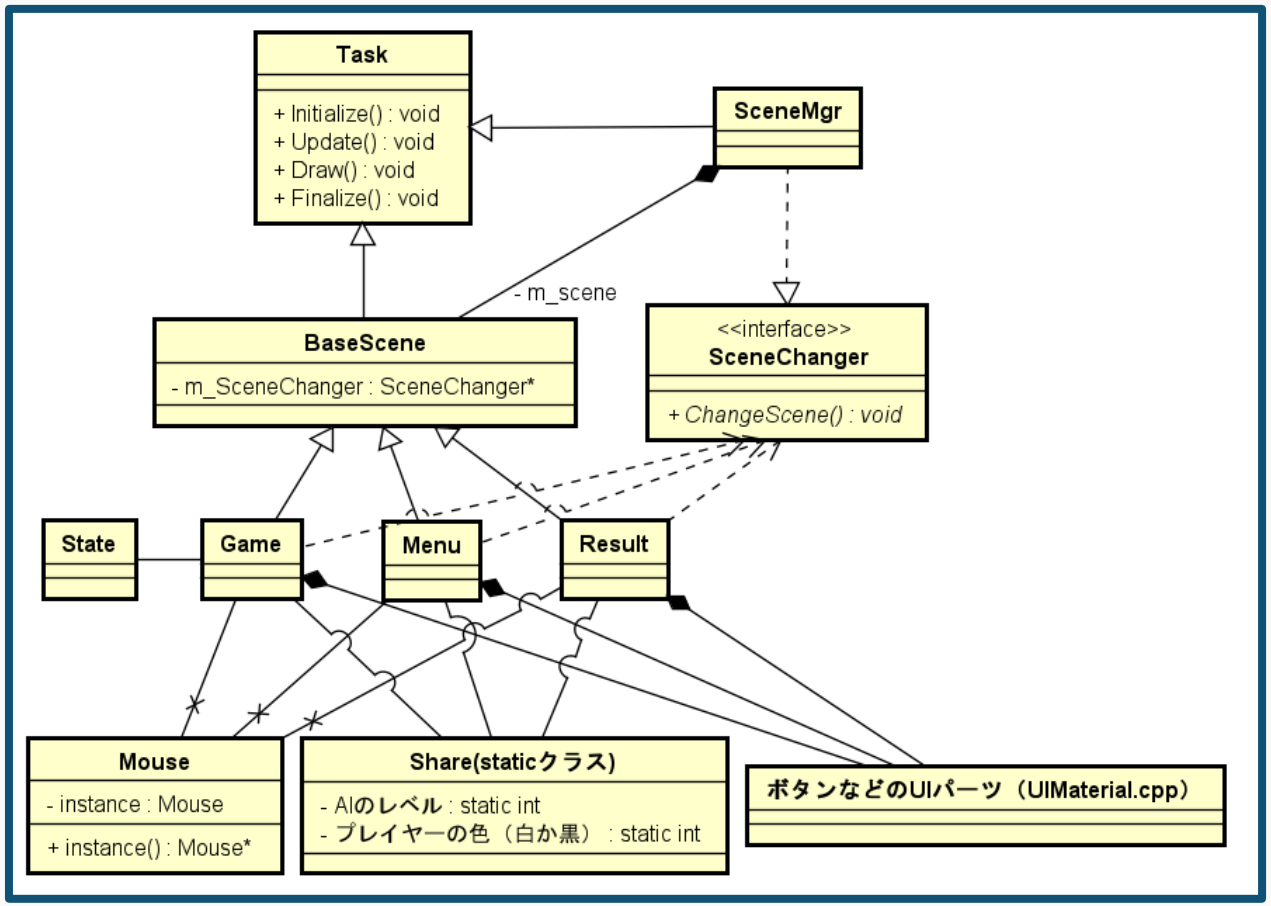


結果表示

NEXTボタン

RETRYボタン

クラス構成



■ 画面遷移

・SceneMgrクラスでSceneChanger
インターフェースを実装
・SceneChangerのポインタを
もつBaseSceneクラスを各画面で継承

各画面クラスからSceneMgr内の
ChangeSceneメソッドのみを呼び出せる構造

■ シングルトン

・Mouseクラスや画面間で共有すべき情報を
保持するShareクラスはシングルトンで実装

オセロAI

2種類のオセロAIを搭載

■**MCTS(モンテカルロ木探索)**

ランダムなシミュレーションを行い有効な着手を探すAI

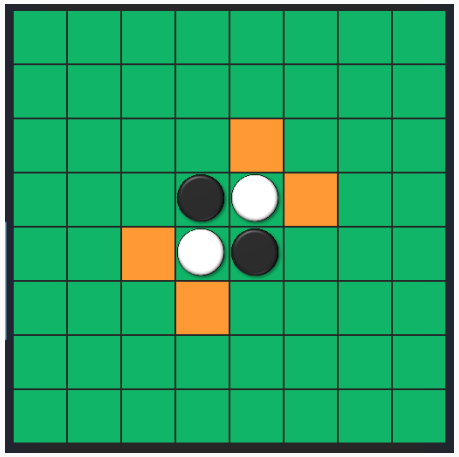
■**αβ法**

相手が最善手を打つと仮定して先読みを行うAI

価値の高いマス(隅)を取れるように選択する（**MCTSよりも強い**）

ビットボードによる管理

高速化のため、8×8の盤面を配列で管理せずに**ビットボード**で管理している。
→**手番プレイヤーの駒**と、**両プレイヤーの駒**のボードをそれぞれ記録



黒の手番

```
uint64_t my_board  = 0b00000000'
                     00000000'
                     00000000'
                     00010000'
                     00001000'
                     00000000'
                     00000000'
                     00000000;
```

手番プレイヤーのビットボード

```
uint64_t all_board = 0b00000000'
                     00000000'
                     00000000'
                     00011000'
                     00011000'
                     00000000'
                     00000000'
                     00000000;
```

両プレイヤーのビットボード

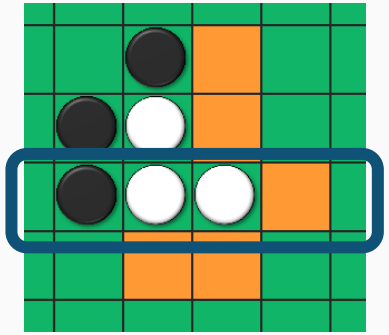
ビットボードによる管理（合法手の探索）

■目的

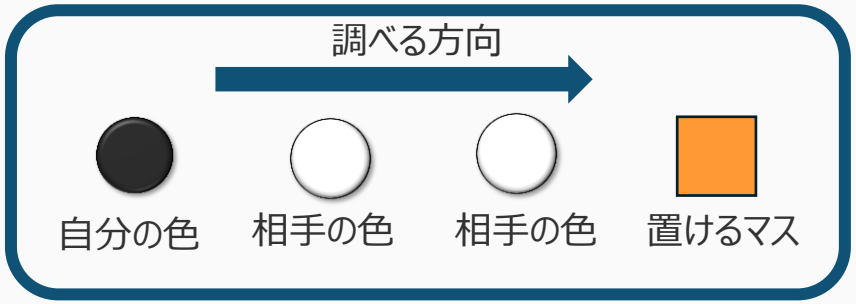
合法手（置けるマス）が1のビットボードを取得する

■オセロの合法手

自分の色からある方向に移動したとき、**相手の色→置いていないマス**となっているマス



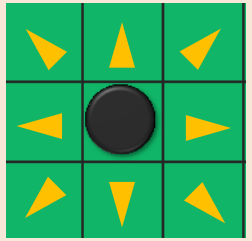
黒の手番



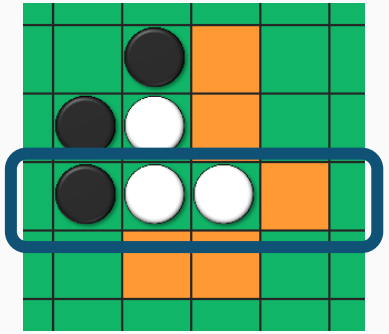
■アルゴリズム

1.自分のボードをある方向にシフトする
2.敵のボードとのAND演算で相手の色が隣にあるか確認
3.さらに自分のボードをシフトし、相手の色の隣が空いているか確認
4.シフト先が
　空いている→置けるマス
　自分の色→置けないマス
　相手の色→不明（さらに奥を調べる）
とし、「不明のマス」があれば3に戻る

1~4を８方向全てに対して行う



ビットボードによる管理（合法手の探索例）



0100 0100
1000 1100
1000 1110
0000 0000
my all

0000 0000
0000 0100
0000 0110
0000 0000
result enemy
(my XOR all)

0010 0000 0000
0100 0100 0100
0100 & 0110 = 0100
0000 0000 0000
my>>1 enemy my
自分の駒と相手の駒が
隣接しているマス

0000 1011 0000
0010 0011 0010
0010 & 0001 = 0000
0000 1111 0000
my>>1 unput 置けるマス
さらに奥を調べる (~all)

resultとOR演算をして保存

0000
0000
0010
0000
my
置けるマスとして判定されたか、
自分の色があった箇所を 0 にする

if(my != 0)
まだ置ける可能性の
あるマスがある

if(my == 0)
探索終了

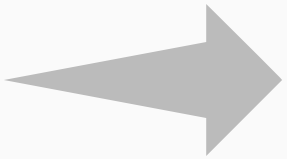
※シフトした後にはビットマスクを使用して不要な情報を削除

ビットボードによる管理（処理速度）

■ビットボードを用いることで**大幅な高速化に成功**

シミュレーションの質は変えずに、100回AI同士で対戦するのにかかった時間

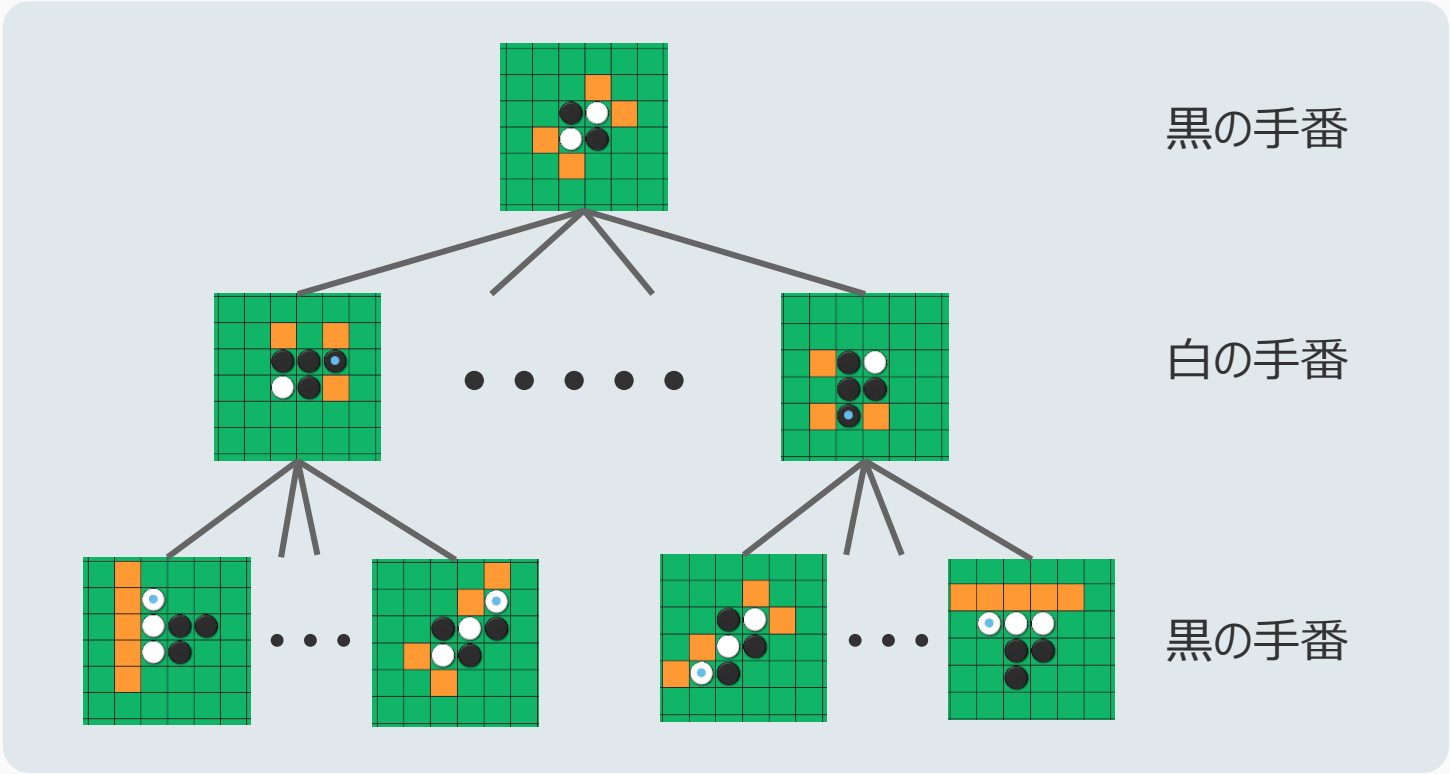
配列を使った場合
230,667ms



ビットボードを使った場合
78,431ms

約**66**%の**高速化**

AIの詳細（ゲーム木）



■ **ゲーム木**

手番ごとのゲームの状態を
木構造にして管理したもの
→すべての状態を列挙する
ことはできない

重要な手を効率よく探索することで
強いAIを作成できる

AIの詳細（MCTS）

■MCTS

ゲーム終了まで探索することは不可能であるため、ある程度のところからは
ランダムAI同士で対戦させ、その勝率で有効な手を選ぶ手法

どの手をランダムシミュレーションをするかは、以下のUCB値によって決定する

$$UCB(i) = \bar{x_i} + \sqrt{\frac{2logN}{n_i}}$$

$\bar{x_i}$・・・手iを選んだ時の勝率　n_i・・・手iを選んだ回数　N・・・シミュレーションした回数の合計

UCB値は探索していない、勝率が高い手ほど高くなる
→バランスの良い探索

AIの詳細（αβ法）

■αβ法

それぞれのマスに価値を設定し、価値の合計が最も高くなるように手を選択する

→**相手も価値を最大化する手を選ぶことを前提として先読みする**