第五回講義課題

課題番号15

MIN-MAX法を用いたチェッカーの対戦プログラムの作成

120443
村上晋太郎

0.添付プログラムの使い方

$ make でコンパイルできます。

コンパイル後は

$ ./game
で実行できます。

MIN-MAX法によるプログラムと対戦できます。

数字を32とうつと、(3, 2)にある駒を選択できます。

l or rをうつことで、その駒を左上に動かすか右上に動かすか選択することができます。

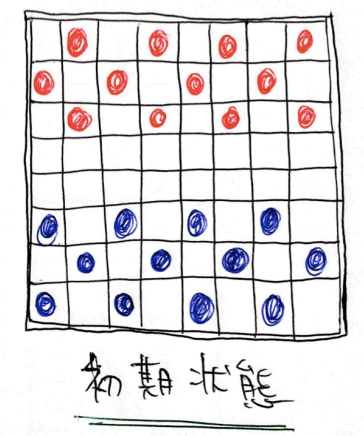
1.この課題について

今回の課題では、MIN-MAX法を用いたチェッカーの対戦プログラムを作成しました。

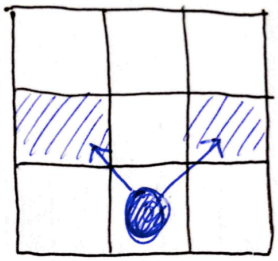
本当はalpha-betaカットも用いたかったのですが、時間がなかったのと、MIN-MAXでも十分強いプログラムが完成したのとで、MIN-MAX法を用いました。

2. チェッカーのルールについて

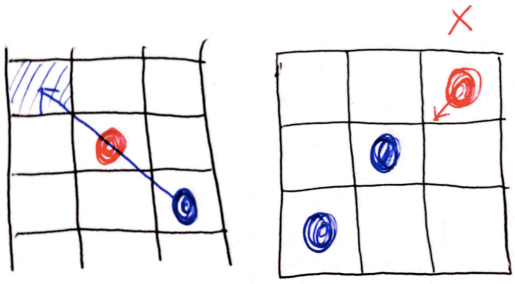
0.最初に、図のように駒を並べるを手前が自分の陣地で、奥が敵の陣地である。



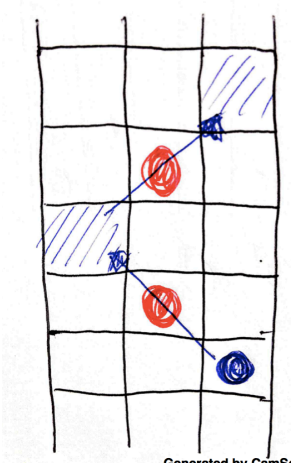
1.駒は、敵の陣地に向かって斜め前にしか動けない



2.敵の駒を飛び越えることで、その駒を取ることができる。(ただし、飛び越えた先に別の駒がある場合は、その駒をとることはできない)

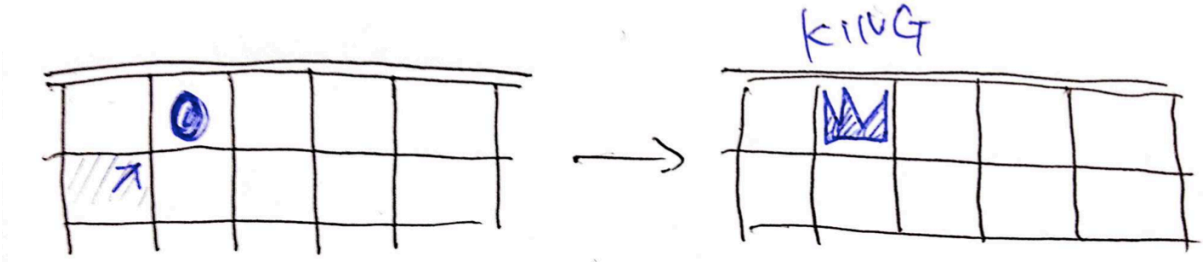


3.敵の駒をとったあとに、さらに取れる駒があれば、同じターンに連続して取ることができる。



4.取れる駒がある時は、必ず取らなければならない。

5.相手の陣地の端に到達した駒は、キングとなる。



6.キングは、斜め前だけでなく、斜め後ろにも動くことができる。

7.自分の手持ちの駒が無くなるか、動かせる駒が無くなると負けである。

チェッカーの試合は、それぞれの駒を相手の陣地に向かわせ、キングにすることを目的とする序盤–中盤と、キングを使ってお互いの駒を取り合い、どちらかが全滅するまで続ける終盤に分けることができる。
序盤–中盤と終盤ではアルゴリズムが異なることと、終盤に入ったときのお互いの手駒の数(特にキングの数)がそのまま試合結果に繋がることを踏まえ、今回は序盤–中盤のみを扱う。

そのために、ルールを以下のように改変した。

5.相手の陣地の端に到達した駒は、キングとはならないが、その代わりに特別点がその駒の持ち主に与えられる。
7.駒が全滅するなどして、どちらかの動かせる駒が無くなると、試合終了となる。
試合終了となった時に、盤面の評価値を算出し、得点の高い方が勝ちとなる。

盤面の評価値は、以下のようにして算出する。
1.自分の駒一つにつき、基本点20点
2.その駒と自分の陣地の端との距離に応じて、マス目×5点の加算
3.その駒が後ろから味方に守られていれば、味方一つにつき10点加算
4.その駒が相手に取られる位置にあれば、10点減算
5.その駒が相手の陣地の端に到達していれば50点加算
6.全ての駒の評価値の総和が盤面の評価値となる。

これらの値はグローバル変数定義されており、main関数の一番最初で定義するようになっている。

3. プログラムの実装について
具体的な実装を示す。
ゲームの本体は、struct stateという構造体で管理している。
いかにその内容を示す。

```
typedef struct state{
  pKind_t bd[SIZE][SIZE];//盤面の状態を保存する
  piece_t piece[(SIZE*3)/2];//次に動かせる駒を保存する。
  piece_t attackingPiece;//攻撃中の駒。二度目の攻撃が起こったときに使用
  //ターンごとのパラメータ
  pKind_t player;//今誰が操作しているか
  int activePiece;//動かせる駒の数
  int attack;//取れる駒があるか  あれば1, 二度目以降の攻撃で2
```

```
  piece_t pieceToMove;//動かそうとする駒
  //minmax関連
  int value;//その盤の状態の評価値
  int valueIndex;//その評価値を与える駒のインデックス
  direction_t valueDirection;
  //ゲーム全体のパラメータ
  runMode_t mode;//人が戦っているか、コンピュータがたたかっているか
  pKind_t winner;//最終的な勝者
}* state_t;
```

piece_tは、駒のデータを保持する構造体である。以下にその内容を示す。

```
typedef struct piece{
  int enable;//有効か無効か 0:無効 1:動ける 2:取れる駒がある
  struct point p;//その駒の座標を示す
  int left;//左にうごけるか 0:動けない 1:動ける 2:取れる駒がある
  int right;//右に動けるか 0:動けない 1:動ける 2:取れる駒がある
  struct state * leftSt;//その駒を左に動かした場合のSt
  struct state * rightSt;//その駒を右に動かした場合のSt
}* piece_t;
```

今回は、ルールに従った盤面の操作のみしか行わせないための仕組み作りに苦労した。
そのためのインターフェース関数群がcheck.c, move.cに定義されている。
まず、init.cに定義されている関数群で、ゲームの初期化を行う。
各ターンの初めに、check.cで定義されているcheckBoard関数で盤面をチェックし、現在動かせる駒をstruct stateの中のpiece配列に保存する。

ルール4により、もしも取れる駒が存在した時は、checkBoardForAttack関数を使用し、攻撃できる駒だけを抽出する。

動かせる駒を確定させたら、move.cの中にある関数群を用いて、動かせる駒を操作する。move.cの中の関数群では、まずstruct stateに保存されている「動かせる駒」の中から駒を選択し、次に右前に動かすか左前に動かすかを選択するようにしている。

MIN-MAX法の実装
MIN-MAX法の実装は、minmax.c内のminmax関数によって行った。
以下にMIN-MAX法の概要を示す。

minmax関数は

```
int minmax(struct state * st, int depth);
```

と宣言してある。関数の動作は以下の通りである。

A. depth==0の時
estimate.cに定義されているestimate関数によって盤面の評価値を算出し、それをst->valueに保管する。
盤面の評価値は、勝敗を決める時と同じで、以下の通りである。

1.自分の駒一つにつき、基本点20点
2.その駒と自分の陣地の端との距離に応じて、マス目×5点の加算
3.その駒が後ろから味方に守られていれば、味方一つにつき10点加算
4.その駒が相手に取られる位置にあれば、10点減算
5.その駒が相手の陣地の端に到達していれば50点加算
6.全ての駒の評価値の総和が盤面の評価値となる。

B. depth>0の時
盤面の中で動かせる駒を調べ、(check.cのcheck関数を使う)動かせる駒がある場合は、その駒を右前に動かした場合と左前に動かした場合の盤面をnextStとし、
それぞれに対してminmax(nextSt, depth - 1)を行う。
それぞれのnextSt->valueをチェックし、全ての動かせる駒について、nextSt->valueの最大値、最小値を調べる。
そのターンが先手側の攻撃であった場合、nextSt->valueの最大値がその盤面のst->valueとなる。
そのターンが後手側の攻撃であった場合、nextSt->valueの最小値がその盤面のst->valueとなる。

動かせる駒が無い場合は、estimate関数を使いその評価値をstate->valueとする。

※今回は、先手側の得点を+、後手側の得点を-とし、st->valueがプラスなら先手側の勝ち、st->valueがマイナスなら後手側の勝ちとした。

この関数minmaxを使い、先手側なら、現在の盤面の最大の評価値を与える次の手を、後手側なら、現在の盤面の最小の評価値を与える次の手を選ぶようにして、チェッカーで戦うプログラムを実装した。

以上です。