提出日　2022/8/5

ソフトウエア実験

最終レポート

学籍番号 20073

氏名　所京太朗

1. 基本仕様

2 人対戦ゲーム「コリドール」の対戦を行うプログラムを作る。このプログラム内では、アルファベータ法を中心として次の手を決める。

まず、盤面の情報を受け取る。その情報から盤面の構造を作る。5 手目までは定石を打ち、5 手目以降は盤面の構造を探索してアルファベータ法により、最善手を求め次の手を返す。最善手を決める際の評価に使う指標は壁の数とゴールまでの距離などを使う。

表 1：機能概要

大まかな流れ		機能の概要	
1	盤面の情報を受け取り、構造に入れる機能	ー1	先手か後手かの情報を受け取り、その手番を出力する
		ー2	盤面の情報を受け取る
		ー3	駒と壁の情報を格納する構造を作る(初期化する)
		ー4	1-2で受け取った情報を1-3の構造に入れる
2	5手までの定石を返す機能	ー0ー1	1手目から3手目までの定石を返す
		ー0ー2	4手目の定石を返す
		ー0ー3	5手目の定石を返す
	アルファベータ法	ー1	アルファベータ法により探索し手を決める
		ー2	与えられた盤面を評価する
		ー3	可能な手を返す
		ー4	移動可能な手を返す
		ー5	与えられたマスから各マスまでの最短距離を返す
		ー6	ゴールまでの最短距離を返す
		ー7	ゴールできるかどうかを返す
		ー8	(相手の駒を無視して)移動可能な手を返す
		ー9	(相手の駒を無視して)与えられたマスから各マスまでの最短距離を返す
		ー10	(相手の駒を無視して)ゴールまでの最短距離を返す
		ー11	壁を置くことが可能な位置を返す
		ー12	与えられた位置に壁がおけるかどうかを返す
		ー13	与えられた駒を与えられた位置に移動させる
		ー14	与えられた位置に壁を置く
		ー15	2-13、2-14で行った操作を戻す
		ー16	is_put_wall内で行う操作を戻す
3	次の手を返す機能		2で決まった最適手を返す

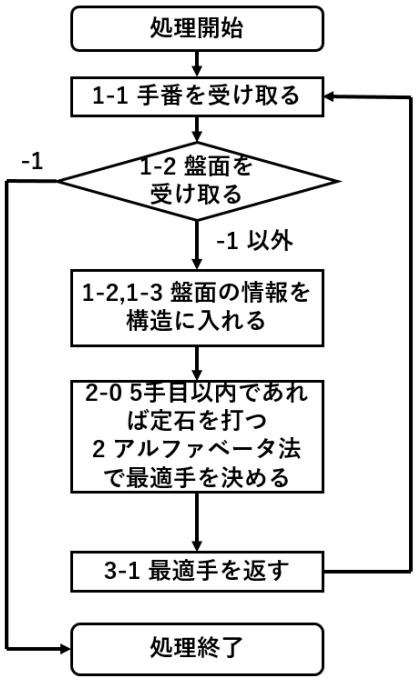


図 1：実行の流れ

表 1 で示した機能を簡単に説明する。2-1 から 2-16 はアルファベータ法を行う中で使用する機能である。詳しくは詳細仕様に記すが、1-1、1-2、3-1 以外は同一のクラス内で実装した。

1 盤面の情報を受け取り、その情報を構造に入れる機能

1-1 先手か後手かの情報を受け取り、その手番を出力する

先手(白)の場合は 1 を受け取り、後手(黒)の場合は 2 を受け取る。勝負がついている場合は-1 を受け取る。

1-2 盤面の情報を受け取る

白の駒の位置、黒の駒の位置、壁の数、壁の位置の情報を受け取る。

1-3 駒と壁の情報を格納する構造を作る(初期化する)

1-2 の情報を格納する盤面を表す構造を作り、初期化しておく。盤面は 17×17 の二重構造で表現する。

1-4 1-2 で受け取った情報を 1-3 の構造に入れる

お互いの駒の情報は白が'2'、黒が'3'で、壁の情報は 1 で格納する。このとき勝負がつい

ていれば、-1 を受け取るのでそこでプログラムが終了するようにする。

2-0　5 手目までは定石を返す機能

2-0-1　1 手目から 3 手目までの定石を返す

1 手目から 3 手目はゴールに向かって前に進む。前に進めない場合は False を返し、アルファベータ法により最善手を見つける。

2-0-2　4 手目の定石を返す

4 手目は後ろに壁を置く。壁を置けない場合は False を返し、アルファベータ法により最善手を見つける。

2-0-3　5 手目の定石を返す

5 手目は後ろに壁を置く。壁を置けない場合は False を返し、アルファベータ法により最善手を見つける。

2-1 最善手をアルファベータ法により見つける機能

2-1 アルファベータ法により探索する

探索する深さを指定し、深さと同じ回数後までの手を考慮し、その手を打った時の盤面を評価する。探索する際には、2-8、2-9、2-10、2-9、2-11 を使用する。評価する際は 2-2 を使用する。探索が終了したら、探索した中で評価が最も高かった手を返す。

一般的なアルファベータ法に、時間制御と候補手の絞り込みを行った。

2-2 与えられた盤面の評価をする

ゴールまでの距離から盤面を評価し、その評価値を返す。評価に使う特徴量は、自分の最短経路、相手の最短経路、自分の残壁数に関するのも 3 つの、計 5 つの特徴量を使用した。特徴量に対する重みは PA の考え方に則った学習を行うことで決定した。

2-3 可能な手を返す

2-4 と 2-11 を使い、次に可能な駒を動かす手と壁を置く手を全て返す。

2-4 移動可能な手を返す

盤面の情報から次に移動可能なマスを見つけ、移動可能なマスを全て返す。

2-5 与えられたマスから各マスまでの最短距離を返す

与えられたマスから移動可能なマスを幅優先探索により探索をしていき、到達可能な各マスまでの最短距離を返す。その際に、2-4 を使用する。

2-6 ゴールまでの最短距離を返す

2-5 を使用し、駒の位置から各マスまでの最短距離を求める。そのあと、ゴールのマスまで到達する距離の中で最も小さいものを返す。ゴールまで到達できないときは False を返す。

2-7 ゴールできるかどうかを返す

2-4 を使い深さ優先探索を行い、ゴールに到達可能かを判定する。到達可能であれば True を不可能であれば False を返す。

2-8 (相手の駒を無視して)移動可能な手を返す

相手の駒を無視(相手の駒がある位置にも移動可能)とした場合の 2-4 と同様の操作を行う。

2-9 (相手の駒を無視して)与えられたマスから各マスまでの最短距離を返す

相手の駒を無視(相手の駒がある位置にも移動可能)とした場合の 2-5 と同様の操作を行う。

2-10 (相手の駒を無視して)ゴールまでの最短距離を返す

相手の駒を無視(相手の駒がある位置にも移動可能)とした場合の 2-6 と同様の操作を行う。

2-11 壁を置くことが可能な位置を返す

盤面構造の中で壁の情報を入れる位置を全探索し、そこに壁が置くことができるかを 2-7 によって判断する。壁を置くことが可能な位置を全て返す。

2-12 与えられた位置に壁が置けるかどうかを返す

与えられた位置にまだ壁がおかれていないか、壁を置いた際にゴールまでたどり着くことができるかを考慮し、壁を置くことができれば True をできなければ False を返す。ゴールができるかの判定は、2-4 を使い深さ優先探索を行い、ゴールに到達可能かを判定する。

2-13 与えられた駒を与えられた位置に移動させる

盤面の構造を与えられた駒を与えられた位置に動かした盤面の構造に更新する。

2-14 与えられた位置に壁を置く

盤面の構造を指定された位置に壁を置いた盤面の構造に更新する。残りの壁の数も更新する。

2-15 2-13,2-14 で行った操作を戻す

2-13 や 2-14 の操作によって変更された盤面の構造を元の盤面の構造に戻す。壁の数も戻す。

2-16 is_put_wall 内で行う操作を返す

2-12 の機能内で行う操作によって変更された盤面の構造を元の盤面の構造に戻す。壁の数も戻す。

3 次の手を返す機能

２の機能で求めた最適手を出力することで、次の手を返す

仕様書に記した<必ず実現する機能>と<余裕があれば実現する機能>は以上の機能で実現できた。<実現はしないが想定される拡張機能>の中の「評価関数における評価に使う指標の重みを機械学習により最適化する」も実現できた。モンテカルロ法や本格的な機械学習を取り入れることはできなかった。

2. 詳細仕様

1-1、1-2、3 以外は同一の make_player クラスで実装する。表 1 にメソッド名を追加した表を表 2 にして以下に示す。実行の流れを図 2 に示す。

表 2：各機能のメソッド名

大まかな流れ		機能の概要		メソッド名
1	盤面の情報を受け取り、構造に入れる機能	ー1	先手か後手かの情報を受け取り、その手番を出力する	(クラス外で実装)
		ー2	盤面の情報を受け取る	(クラス外で実装)
		ー3	駒と壁の情報を格納する構造を作る(初期化する)	__init__(self,player)
		ー4	1-2で受け取った情報を1-3の構造に入れる	input_information(self,list_information)
2	5手までの定石を返す機能	ー0ー1	1手目から3手目までの定石を返す	initial_move_1(self,dir)
		ー0ー2	4手目の定石を返す	initial_move_2(self,dir)
		ー0ー3	5手目の定石を返す	initial_move_3(self,dir)
	アルファベータ法	ー1	アルファベータ法により探索し手を決める	alphabeta(self,depth)
		ー2	与えられた盤面を評価する	eval_0(self),eval_1(self)
		ー3	可能な手を返す	generate_moves(self,po,turn)
		ー4	移動可能な手を返す	return_pos(self,pos)
		ー5	与えられたマスから各マスまでの最短距離を返す	cal_dis(self,pos)
		ー6	ゴールまでの最短距離を返す	min_dis(self,pos,goal)
		ー7	ゴールできるかどうかを返す	is_goal(self,pos,goal)
		ー8	(相手の駒を無視して)移動可能な手を返す	return_pos_ignore_pa(self,pos)
		ー9	(相手の駒を無視して)与えられたマスから各マスまでの最短距	cal_dis_ignore_pa(self,pos)
		ー10	(相手の駒を無視して)ゴールまでの最短距離を返す	min_dis_ignore_pa(self,pos,goal)
		ー11	壁を置くことが可能な位置を返す	return_wall(self)
		ー12	与えられた位置に壁がおけるかどうかを返す	is_put_wall(self,wall_list)
		ー13	与えられた駒を与えられた位置に移動させる	move_pos(self,pos,next_pos,turn)
		ー14	与えられた位置に壁を置く	put_wall(self,wall_list,turn)
		ー15	2-13、2-14で行った操作を戻す	undo(self,undo_list,turn)
		ー16	is_put_wall内で行う操作を戻す	undo_in_is_put_wall(self,undo_list)
3	次の手を返す機能		2で決まった最適手を返す	(クラス外で実装)

表 3 にメンバの一覧を示す。

表 3：メンバ一覧

メンバ名	型	意味	作られるメゾット
self.player	int型	手番	__init__
self.board	list型	盤面	__init__
self.list_information	list型	入力された盤面の情報	input_information
self.player_pos	list型	自分の駒の位置	input_information
self.partner_pos	list型	相手の駒の位置	input_information
self.player_goal	int型	自分のゴール縦列	input_information
self.partner_goal	int型	相手のゴールの縦列	input_information
self.player_wall_nu	int型	自分の残りの壁の数	input_information
self.partner_wall_nu	int型	相手の残りの壁の数	input_information
self.start_time	float型	__init__を行った時点での時間	__init__
self.wall_conf	list型	壁の周りの壁を置く候補	input_information

1 盤面の情報を受け取り、その情報を構造に入れる機能

1-1 先手か後手かの情報を受け取り、その手番を出力する

input 関数で受け取り、print 関数で出力する。受け取った情報は player に代入する。

1-2 盤面の情報を受け取る

input 関数でリストとして受け取る。

1-3 駒と壁の情報を格納する構造を作る(初期化する)メソッド

__init__(self,player)

種類	名前	型	意味
引数	player	int 型	手番を表す(1 なら先手、2 なら後手)

引数 player を self.player メンバに代入する。

self.board メンバに駒と壁の情報を格納する構造を作る。17×17 の 2 重リストで作る。駒の情報は奇数行かつ奇数列のマスに入れる。それ以外のマスには壁の情報を入れる。具体的な例を 1-4 の図 2 に示す。

実行時間で制御できるようにこの時点での時間を self.start_time メンバに代入する。

1-4 1-2 で受け取った情報を 1-3 の構造に入れる

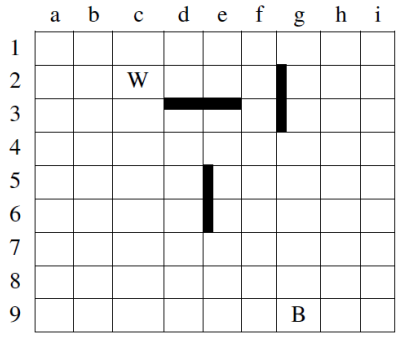
input_information(self,list_information)

種類	名前	型	意味
引数	list_information	list 型	入力された盤面の情報

引数 list_information を self.list_infromation メンバに代入する。

まず、受け取った情報が -1 の場合勝負がついているため、'finish'を返す。'finish'が返ったらすぐにプログラムが終了するようにする。

お互いの駒の情報は白が'2'、黒が'3'で、壁の情報は 1 で self.board に格納する。図 2 に self.board の例を示す。



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	**2**	0	0	0	0	0	0	**1**	0	0	0	0	0
3	0	0	0	0	0	0	**1**	**1**	**1**	0	0	**1**	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	**3**	0	0	0	0

図 2:self.board の例

self.player_pos メンバに自分の駒の位置を、self.partner_pos メンバに相手の駒の位置を入れる。self.player_goal メンバに自分のゴールの行番号を、self.partner_goal メンバに相手のゴールの行番号を入れる。self.player_wall_nu メンバに自分の壁の残りの数を、

self.partner_wall_nu メンバに相手の壁の残りの数を入れる。

壁の情報を入れていくのと同時に、その壁周りの壁を置ける位置に壁を置くことができるならば、探索の際の壁を置く候補とするため、wall_conf リストに壁を置ける位置の情報を入れていく。壁の周りとは図 3 の位置のことである。

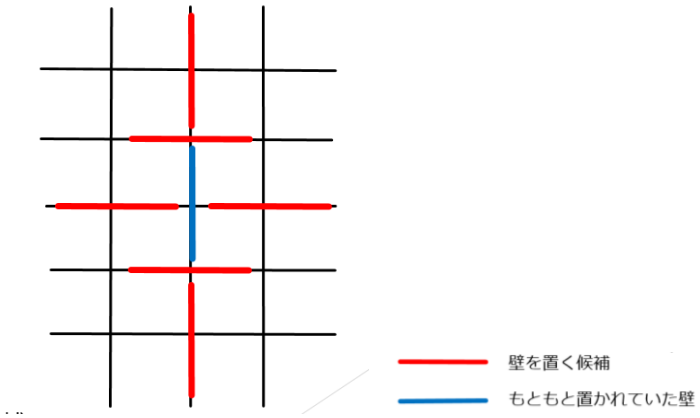


図 3：壁を置く候補の位置

2-0　5 手目までは定石を返す機能

5 手目までは図 4 のような定石を打つようにする。

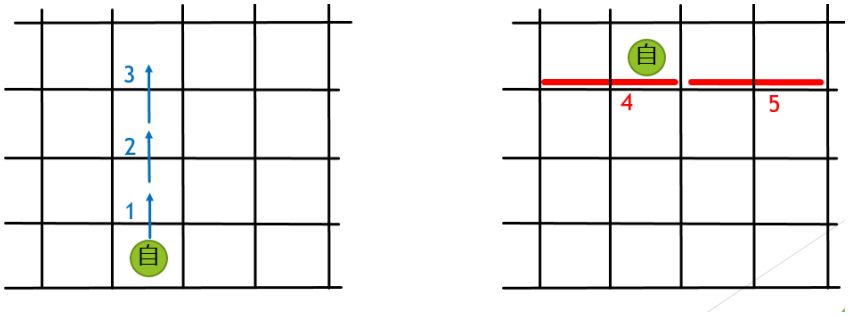


図 4：5 手目までの定石

2-0-1　1 手目から 3 手目までの定石を返す

initial_move_1(self,dir)

種類	名前	型	意味
引数	dir	list 型	ゴールの方向

1 手目から 3 手目はゴールに向かって前に進む。前に進めない場合は False を返し、アルファベータ法により最善手を見つける。前に進める場合はその手を返す。

2-0-2　4 手目の定石を返す

initial_move_2(self,dir)

種類	名前	型	意味
引数	dir	list 型	ゴールの方向

4 手目は後ろに壁を置く。壁を置けない場合は False を返し、アルファベータ法により最善手を見つける。壁を置ける場合はその手を返す。

2-0-3　5 手目の定石を返す

initial_move_3(self,dir)

種類		名前	型	意味
引数		dir	list 型	ゴールの方向

5 手目は後ろに壁を置く。壁を置けない場合は False を返し、アルファベータ法により最善手を見つける。壁を置ける場合はその手を返す。

2 最善手をアルファベータ法により見つける機能

2-1 アルファベータ法により探索する

alphabeta(self,depth) , alphabeta_r(self,alpha,beta,depth)

種類		名前	型	意味
引数		depth	int 型	探索深さ
引数		alpha	int 型	アルファ値
引数		beta	int 型	ベータ値

まず、alphabeta により、alpha と beta を-∞と∞に設定して、alphabeta_r メソッドを返す。

alphabeta_r メソッドでは、depth が 0 の場合、それ以上は探索を行わずに、その時点での盤面の評価値を返す。評価には 2-2 の eval を利用する。

深さが 0 出ない場合、まず best_score を-∞に、best_move を None で初期化しておく。depth から現在の手番に対応する pos と turn をつくる。turn は自分が手番のときは'player'を相手が手番のときは'partner'を入れる。

次に可能な手を 2-8 の generate_moves により全て上げ moves に入れる。moves の要素一つ一つについて盤面を更新し、深さを一つ下げ探索を再帰により繰り返す。場面の更新には、2-9 の move_pos と 2-10 の put_wall を使う。pos や turn は 2-8 から 2-9 の引数に使う。

今行った操作により self.board などが変化するので 2-11 の undo により戻す。返ってきた評価値と best_score を比較し、評価値の方が高ければ、best_score と best_move を更新する。すべての探索が終了したら、最終的な best_score と best_move を返す。

上記の一般的なアルファベータ法にいくつかの機能を追加した。

候補手の探索を行う前に候補手の入ったリストを候補手が評価順になるように並び替えるようにした。評価順に並び替えることで、その盤面での評価順に探索を行うことができる。評価の際に用いる評価関数は、通常の残壁数を考慮した eval_1 ではなく、最短距離のみを考慮した eval_0 を使用した。評価順に探索を行えれば、途中で探索を止めたとしてもある程度良い手を見つけることができるため、実行時間が一定値に達したら探索を終了させることができる。今回は__init__メソッドを実行した時間から 4.98s を超えていたらその時点で探索を終了させ、その時点でのもっともよい手を返すようにした。しかし、4.98s はタイムエラーが起こる 5s に対してぎりぎりのため、授業内で対戦を行った際、一部のパソコン(パソコンに依存するかはわからないが)ではタイムエラーが出る。自分のパソコンでは 200 回以上行って一度もタイムエラーにはならなかったため、盤面ではなく実行環境によって変わってくると考えられる。また、最終レポート作成時に気づいたことだが、実行時間が実行ごとに多少違うことから先読みをする手の数が変化し、その結果実行ごとに違う手を打つことがある。

候補手を評価順に並べ替えることで、候補手のうち評価の高いいくつかの手に絞って探索を行うこともできる。今回は上位 18 手のみを探索するようにした。

また、図 5 のように、先読みした盤面での評価値で次の手を決めるため、ゴールにすぐに行ける場合でも、ゴールしようとしない場合があったので、すぐにゴールできる場合は探索を行わずその手を返すようにした。

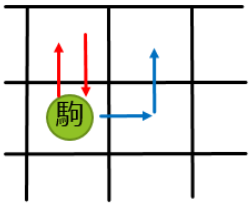


図 5:最短でゴールする手(赤)ではなく最終的にゴール
する手(青)の方が良い手と判断される盤面例

2-2 与えられた盤面の評価をする

eval_0(self),eval_1(self)

eval_0 は残壁数を考慮しない関数、eval_1 は残壁数を考慮する関数で、alphabeta_r メソッド内で使う。

eval_0 は自分のゴールまでの最短距離を mi_pl にいれ、相手のゴールまでの最短距離を mi_pa に入れ、mi_pl*1+mi_pa*(-1)を評価値とした。このメソッドを使用するのは、候補手を評価順に並び替える際に使うものなので、厳密な評価をしなくてもよいため、重さは自分で決めたものにした。

eval_1()は、mi_pl と mi_pa に加え、自分の残壁数が 6、 4 、 3 以上のときに作用す特徴

量 3 つを加えた。また、ゴールした盤面の時は他の盤面と区別するために評価値を大きく変えるようにした。各特徴量の重さはリスト W の各要素で指定した。

W は PA(Passive Aggressive)という線形分類器で二値分類の学習を行うことで決定した。更新式は図 6 で表され、これを実装すると図 7 のようになった。PA の実装は参考文献[4]を参考にした。

<更新式>

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + min\left(C, \frac{l_{hinge}(\mathbf{x}^{(t)}, y^{(t)}, \mathbf{W}^{(t)})}{\|\mathbf{x}^{(t)}\|^2}\right) y^{(t)}\mathbf{x}^{(t)}$$

$$l_{hinge}(\mathbf{x}, y, \mathbf{W}) = max(0, 1 - y\mathbf{W}^{\mathrm{T}}\mathbf{x})$$

図 6:PA の更新式

```
class PassiveAggressive:
    def __init__(self,w):
        self.t = 0            #tは更新回数を示すが使用しない
        self.w = w            #wの初期化

    def L_hinge(self, vec_x, y):    #1-yWxの計算
        return max([0, 1-y*self.w.dot(vec_x)])

    def calc_eta(self, loss, vec_x):    #l/x^2の計算
        l2_norm = vec_x.dot(vec_x)
        return loss/l2_norm

    def update(self, vec_x, y):     #Wの更新
        loss = self.L_hinge(vec_x, y)
        eta = self.calc_eta(loss, vec_x)
        self.w = self.w + eta*y*vec_x*0.01   #学習率を0.01に
        self.t += 1

    def fit(self, vec_feature, y):
        if self.w is None:
            weight_dim = len(vec_feature)
            self.w = np.ones(weight_dim)
        self.update(vec_feature, y)
        return self.w

class PassiveAggressiveOne(PassiveAggressive):
    def __init__(self, w, c=0.1):
        self.c = c                      #cの設定
        PassiveAggressive.__init__(self,w)

    def calc_eta(self, loss, vec_x):    #min(c,l/x^2)の計算
        l2_norm = vec_x.dot(vec_x)
        return min(self.c, loss/l2_norm)
```

図 7:PA の実装

学習方法は、自己対戦で対戦をして得られた一つ一つの盤面における特徴量を訓練データとして、勝った側なら 1 のラベルを、負けた側なら-1 のラベルを与えたものを使って W

を更新していく。

学習を行う上で工夫した点がいくつかある。一つは、図 6 の更新式をそのまま使うと更新幅が大きくなってしまうため、更新式に学習率 0.01 をかけて更新幅を小さくなるようにした。また、評価するうえでよくない盤面の影響を少なくするために、4 回対戦を行ってから 4 回分の更新を行うようにした。様々な盤面でのデータを取るために、初期は位置にランダム性を持たせる工夫も行った。

学習は 1 回で 4 対戦を行い更新するのを 15 回繰り返して最終的な W とした。図 9 に更新結果を示す。

特徴量	自分のゴールまでの最短距離	相手のゴールまでの最短距離	残壁が6個以上あるときの残壁数	残壁が4個以上あるときの残壁数	残壁が2個以上あるときの残壁数
重さ	0.8857026	-0.87554909	0.46791599	0.26234566	-0.23649033

(注意)スコアは小さい方がよい

図 9：学習結果

2-3 可能な手を返す

generate_moves(self,pos,turn)

種類	名前	型	意味
引数	pos	list 型	移動させたい駒の位置
引数	turn	str 型	手番

まず、可能な手を入れていくリスト moves を空のリストとしてつくる。引数の pos は return_block に渡す。

2-4 の return_block により移動可能なマスを探し、moves に入れていく。

turn で手番をみて、それに対応する壁の数(self.player_wall_nu か self.partner_wall_nu)を確認し、0 であればここまでの moves を返す。0 でなければ壁を置く手を 2-11 の return_wall によって探し、moves に入れてから、moves を返す。

2-4 移動可能な手を返す

return_pos(self,pos)

種類	名前	型	意味
引数	pos	list 型	移動させたい駒の位置

リスト Dir の中に上下左右の四方向のベクトルを入れておく(Dir=[[1,0],[-1,0],[0,1],[0,-1]])。また、移動可能なマスを入れていく空のリスト block をつくっておく。

pos から Dir の方向に動かした時に移動可能なら、移動先のマスを block に入れる。移動先に相手も駒があり、その先に壁がなければその先のマスを block に入れる。移動先に相手

の駒がありさらにその先に壁がある場合、相手の駒の左右(上下)に壁がなければ、その先のマスを block に入れる。すべての方向の探索が終了したら、list 型の block を返す。

2-5 与えられたマスから各マスまでの最短距離を返す

cal_dis(self,pos)

種類	名前	型	意味
引数	pos	list 型	探索を開始するマスの位置

visited という 9×9 の二重リストを作り、一つ一つのリストの中に空のリストを用意しておく。初期値として、visited の pos に対応するリストに 0 を入れる。また、next_q にデックを作り最初は pos を入れておく。

幅優先探索により各マスまでの最短距離を見つけていく。ここからの操作は next_q の大きさが 0 になるまで繰り返す。next_q から next_q に入れたのが最も早かったマスを表すリストを取り出し、now_block に代入する。そのマスから移動可能なマスを 2-4 の return_pos を使い見つける。それらを next_block に代入する。visited の next_block に対応するリストが空のリストであればまだ探索をしていないということなので、now_block までの距離+1 を入れる。すべての探索が終わったら visited を返す。この visited には到達可能なすべてのマスまでの最短距離が入っている。

2-6 ゴールまでの最短距離を返す

min_dis(self,pos,goal)

種類	名前	型	意味
引数	pos	list 型	探索を開始するマスの位置
引数	goal	int 型	ゴールの縦列

引数の pos は 2-5 の cal_dis を用いる際に渡す。goal はゴールとなる縦列の番号の int 型である。goal には 1 か 9 が入る。

min_ route の初期値は 9×9+1 としておく。2-5 により返ってきたリストを visited に入れ、ゴールに対応する visited の要素を見ていき、最短距離が一番小さいものを更新していく。

min_ route が 9×9+1 のままであれば、ゴールまで到達できないということなので False を返す。そうでなければ最短距離 min_route を返す。

2-7 ゴールできるかどうかを返す

is_goal(self,pos,goal)

種類	名前	型	意味
引数	pos	list 型	探索を開始するマスの位置
引数	goal	int 型	ゴールの縦列

最初は 2-6 でゴールに到達可能かを判定していたが、実行時間短縮のため、ゴール到達可能かどうかのみを判定するメソッドをつくった。2-6 と違う点は、距離を visirted に入れていくのではなく探索済みを表す 1 を入れていく点、ゴールに到達したら即終了する点、幅優先探索ではなく深さ優先探索する点である。

2-8 (相手の駒を無視して)移動可能な手を返す

return_pos_ignore_pa(self,pos)

種類	名前	型	意味
引数	pos	list 型	移動させたい駒の位置

相手の駒を無視(相手の駒がある位置にも移動可能)とした場合の 2-4 と同様の操作を行う。実装方法は 2-4 と同様である。

2-9 (相手の駒を無視して)与えられたマスから各マスまでの最短距離を返す

cal_dis_ignore_pa(self,pos)

種類	名前	型	意味
引数	pos	list 型	移動させたい駒の位置

相手の駒を無視(相手の駒がある位置にも移動可能)とした場合の 2-5 と同様の操作を行う。実装方法は 2-5 と同様である。

2-10 (相手の駒を無視して)ゴールまでの最短距離を返す

min_dis_ignore_pa(self,pos,goal)

種類	名前	型	意味
引数	pos	list 型	移動させたい駒の位置
引数	goal	int 型	ゴールの縦列

相手の駒を無視(相手の駒がある位置にも移動可能)とした場合の 2-6 と同様の操作を行う。実装方法は 2-6 と同様である。

2-8 から 2-10 までにより相手の駒を無視した場合の最短距離を求められるようにした理由は、図 10 のように一時的にゴールが不可能な盤面が出てきて、この場合の最短距離が求められないことから、盤面の評価が得られないというのを避けるためである。

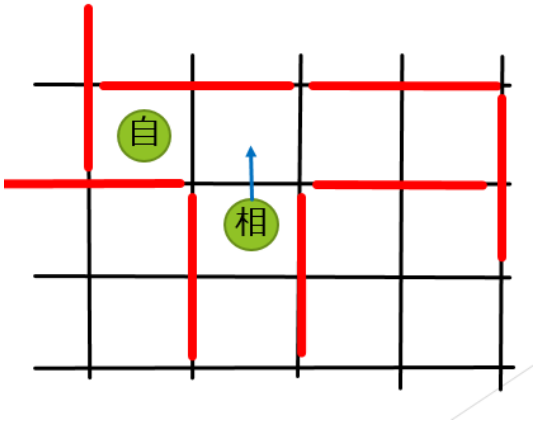


図 10：最短距離が求められない例

2-11 壁を置くことが可能な位置を返す

return_wall(self)

wall という空のリストをつくり、ここに壁を置くことができる位置を表すリストを入れていく。壁を置く候補は、お互いの駒のある周り(図 11)と self.wall_conf メンバに入っている位置だけである。壁が置けるかどうかは 2-12 の is_put_wall を使う。すべての探索が終わったら、wall リストを返す。

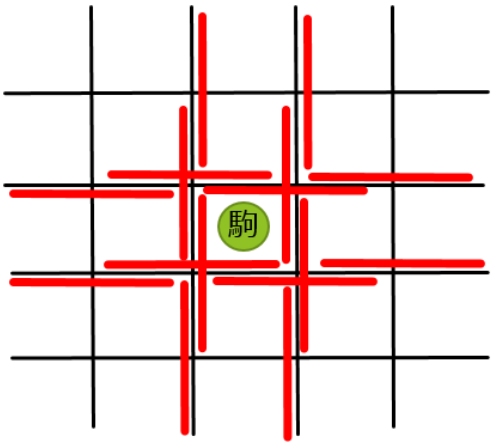


図 11:駒の周りの壁候補

2-12 与えられた位置に壁が置けるかどうかを返す

is_put_wall(self,wall_list)

種類	名前	型	意味
引数	wall_list	list 型	壁が置けるかどうかを判定する位置

壁が置けるかどうかの判定は、すでに壁が置かれていないかと、壁を置いたときにゴールに到達できなくならないかを考える。

壁が置けるかどうかは壁を置こうとしている位置に対応する self.board の要素がすべて 0 かどうかで判断する。壁がおけなければ False を返し、終了する。壁がおければ、壁を置いた場合の盤面になるように self.board を更新し、次の判定に移る。

壁を置いたときにゴールに到達できるかどうかは、自分の駒と相手の駒について考える。fl_1 と fl_2 を用意する。fl_1 は自分の駒がゴールに到達できるかどうかを表し、fl_2 は相手の駒がゴールに到達できるかどうかを表す。fl_1 や fl_2 を判定する際には 2-7 の is_goal を使う。

fl_1 と fl_2 がともに True になれば、壁がおけるということなので True を返し、それ以外のときは False を返す。壁を置いて判定したため、2-16 の undo_in_is_put_wall により、置いた壁を戻す。

2-13 与えられた駒を与えられた位置に移動させる

move_pos(self,pos,next_pos,turn)

種類	名前	型	意味
引数	pos	list 型	移動させたい駒の位置
引数	next_pos	list 型	移動先の位置
引数	turn	str 型	手番

self.board を移動した後の盤面に更新する。つまり、駒を表現する pos の位置の数字(2 か 3)pre_nu を next_pos の位置にいれ、pos の位置に何もないことを表す 0 を入れる。また、turn で手番をみて、それに対応する駒の位置(self.player_pos か self.partner_pos)を更新する。

2-14 与えられた位置に壁を置く

put_wall(self,wall_list,turn)

種類	名前	型	意味
引数	wall_list	list 型	置きたい壁の位置
引数	turn	str 型	手番

self.board を壁を置いた後の盤面に更新する。つまり、wall_list に対応する self.board の要素を壁が置かれていることを表す 1 にする。また、turn で手番をみてそれに対応する壁の数(self.player_wall_nu か self.partner_wall_nu)を減らす。

2-15 2-13,2-14 で行った操作を戻す

undo(self,undo_list,turn)

種類	名前	型	意味
引数	undo_list	list 型	戻したい操作
引数	turn	str 型	手番

2-13 や 2-14 の操作によって変更された盤面の構造を元の盤面の構造に戻す。引数の undo_list には戻したい操作が入っている。例えば、(1,5)の駒を(2,5)の位置に戻したい場合は[1,1,5,2,5]であり、[4,2,2]で表現される壁を取り除きたいといは[2,4,2,2]である。つまり、リストの先頭が 1 の場合は駒を動かした操作、2 のときは壁を置いた操作を表している。

駒を戻す際は、self.board の更新と、turn で手番をみてそれに対応する駒の位置(self.player_pos か self.partner_pos)を更新する。

壁を取り除く際は、self.board の更新と、turn で手番をみてそれに対応する壁の数(self.player_wall_nu か self.partner_wall_nu)を増やす。

2-16 is_put_wall 内で行う操作を戻す

種類	名前	型	意味
引数	undo_list	list 型	戻したい操作

2-12 の機能内で行う操作によって変更された盤面の構造を元の盤面の構造に戻す。壁の数も戻す。undo と処理方法は同じであるが、残壁数の更新をしなくてよい点が違う。

3 次の手を返す機能

alphabeta によって見つけた最適手を best_move にいれ、これを出力する。

3. 計画の達成度と残された課題

仕様書作成時に立てた計画を図 12 に示す。

①ネガマックス法の実装、仕様書作成

②評価値の工夫

③アルファベータ法の実装

④interative Deppning の実装

⑤ビームサーチの実装

⑥発表資料作成

⑦レポート作成

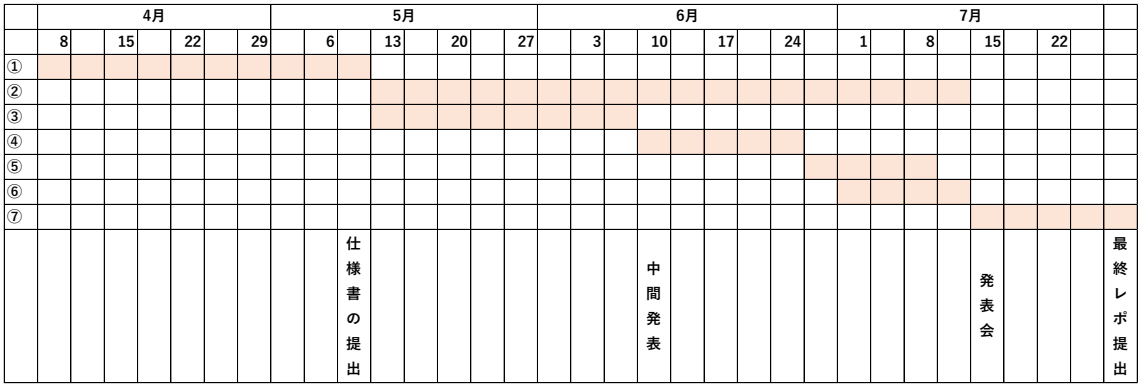


図 12：計画表

④は時間制御を表し、⑤は候補手を評価順にして絞ることを表している。④と⑤は 7 月 8 日以降に実装したが、ほとんど計画通りに進み、実現する予定だった機能はすべて実現することができた。

残りの課題として、評価関数の設計が挙げられる。今回学習により特徴量の重さを決めたが、プログラムにはランダム性がないため、自己対戦では偏ったデータしか取れなかったと考えられる。それでは、良い評価関数が作れたといえない。また、今の評価関数には特徴量が 5 つしかなく、これでは盤面を適切に評価できているとは言えない。壁がおかれている位置などの情報も評価関数に入れるべきであると思う。

授業内で対戦した上で、勝率をあげるには、壁を置く候補を増やす必要がありそうだと感じた。

4. 実行手順

提出した quoridor_report_sd20073.zip の中は以下のとおりである。

```
quoridor_report_sd20073
 |— make_player.py
 |— README.txt
 |— 最終レポート.pdf
 └ 発表資料.pptx
```

make_player.py が今回作成したプログラムであり、最終レポートと発表資料同じディレクトリに入れた。

ゲームの進行は配布された quoridor.zip の中にある対戦サーバープログラム server.py によって行う。作成した make_player 同士を対戦させるには、以下のようにサーバーコマンドを実行する。

```
python3 server.py -1 make_player.py -2 make_player.py
```

5. まとめ感想

1 からこのような大きなプログラムを作成するのは初めてであり、クラスの使い方をはじめとして多くのことを学ぶことができた。

また、強化学習を知るきっかけにもなったので良かった

参考文献

[1] ‘ソフトウェア実験の手引き’

[2] ‘世界四連覇 AI エンジニアがゼロから教えるゲーム機探索入門’,Qiita,2022/5/8 アクセス
https://qiita.com/thun-c/items/058743a25c37c87b8aa4#iterative-deepning%E5%8F%8D%E5%BE%A9%E6%B7%B1%E5%8C%96

[3] ‘貪欲法、山登り法、焼きなまし、ビームサーチ、これらの間の関係について’, 2022/5/8 アクセス
https://kmyk.github.io/blog/blog/2019/03/07/local-search-and-greedy/

[4] ‘オンライン機械学習の弱点って？？：Passive Aggressive のプロセスを実装＆可視化’
https://qiita.com/Wotipati/items/a8eda3f246eb07c516ca