情報システム工学 PBL オセロレポート

私はこのオセロプログラムを作るにあたり、たくさんの試行・改善を繰り返してきた。それらによって生じた問題や工夫、結果を、時系列順で簡潔に示していきたいと思う。

1. 1 つ目のプログラム*1では、角や辺を取りやすく、など、マス 1 つずつにそれぞれ優先度をつけて判定した。計算量は O(n)…(n は大体 64*len(moves))…。

2. 2 つ目のプログラム*2では、インターネットからオセロの型についての情報を調べ、一部を実装して判定した。縦取りと牛取りを実装したものの、思うように動かず、原因も分からずでこの方針は頓挫した。のちに、「board の二次元リストは board[縦][横]で管理されているが、moves や、返却すべき一次元リストは move[横][縦]で管理されていること」が判明した。この仕様を見落としていたことで、この後も開発が何度か滞った。計算量はほぼ O(1)。

3. 3 つ目のプログラム*3では、オセロの強い友人から聞いた、「次のターンの相手の合法手の数を少なくする」方法を実装した。このプログラムでは、配布されていた OthelloLogic.py を import して使用することで、非常に短いコードで実装することができた。しかし、こちらも仕様上の不具合があることが分かった。それは、OthelloLogic.execute を実行すると、引数に入れた board 自体が変更されてしまうのである。これは、OthelloAction から引数に入れた board と、OthelloLogic.execute で扱う board のオブジェクト ID がプログラムの中で一致してしまっているため、同じものであるとみなされ、変更されてしまうのだと分かった。この不具合を解消するために、後日作ったプログラムでは copy モジュールの deepcopy を使用することで、引数の board は変更されずに、execute 後の board を取得することが可能になった。計算量は O(n)。

---

*1 https://github.com/Alnya/PBL_Othello/blob/master/20201215.py

*2 https://github.com/Alnya/PBL_Othello/blob/master/20201220_mold.py

*3 https://github.com/Alnya/PBL_Othello/blob/master/20201220_strong.py

4. 4 つ目のプログラム*4では、弱いオセロプログラムの作成を行った。3 つ目のプログラム同様、オセロの強い友人から、「とにかく一番多く石を取れる場所に置く」という方法を実装した。並びに、一般的に強いとされる角には置かないようにした。計算量は大体 O(n)。

5. 5 つ目のプログラム*5では、インターネットで調べた「序盤は少なく、終盤は多くとる」という方法を実装した。このプログラムからは、board の中の 0 の数を数えることで、現在が何手目かを確認できる機能を実装した。この手数の確認機能により、序盤や終盤で違うアルゴリズムを組むことが可能になった。また、1 つ目のプログラムで用いた優先度を改良して転用した。ここで、「角周り 3 つのマスには普段は置かないようにして、もしその角に自分の石を置くことができたら優先して周り 3 つのマスに置くようにする」と強いのではないかと考え、実装した。(しかし、この機能に関しては 2 つ目のプログラムと同じく moves の[横][縦]問題に気づいていなかったため、このままでは意図したとおりに動かなかった)。肝心の、序盤は少なく、終盤は多くとるアルゴリズムは、4 つ目のプログラムを改良して実装した。

6. 6 つ目のプログラムから 12 つ目のプログラムまではほとんど同じ方針で作成しているので、まとめて説明することとする。ここでは、最終稿となり、最後の授業でも実際に動かした 12 つ目のプログラム*6から説明する。こちらのプログラムでは、上述してきたような様々な仕様上のミスを把握したうえで、しっかりと対応をとって実装している。まず、2 つ目のプログラムで実装した型の判定を改良し、2 手目か 3 手目の時にのみ、一般的に良いとされている手を打つような機能を実装した。次に、4 手目から 15 手目にかけて計算するプログラムを作成した。このプログラムでは、所謂「開放度」が少なくなるようにとるようになっている。詳細は長くなるため省くが、この中にも 5 つ目のプログラム同様優先度のシステムは組み込んである。ここで、どうしても角周り 3 つのマスに置かざるを得なくなった場合、次の手で相手が角を取れるようなマスにはできるだけ置かないようにしている。そして、16 手目から 47 手目にかけて、概要を説明すると、出来るだけ角と辺を取り、それが出来ないのであれば上述した開放度で判定するようなプログラムになっている。ここで特に工夫した点は、辺で取れるマスがあった時、その次の手で相手にひっくり返されては意味がないため、次の相手の手を確かめ、ひっくり返される心配がない時のみ置くようにしている。そして最後の 48 手目から 60 手目の 13 手に関しては、最後まで完全に読み切ること

---

*4 https://github.com/Alnya/PBL_Othello/blob/master/20201220_weak.py

*5 https://github.com/Alnya/PBL_Othello/blob/master/20201222.py

*6 https://github.com/Alnya/PBL_Othello/blob/master/OthelloAction-strong.py

で、一気に勝率を上げられるようなプログラムになっている。ここでは、再帰関数を作ることで、効率的なコードを書くことができた。当初、終盤から最後の手にかけて、全ての手に関して解析し、合法手一覧から一番高い勝率の手を選んで返すという方式を取っていたのだが、この方式では計算量 $O(n^{(61\text{-turn})})$もかかってしまい、指数関数上に爆発的な計算時間がかかってしまうことが問題になってしまった。この状態では多くてもラスト 7 手目くらいまでしかまともに計算できなかった。この問題を解決するにあたり、大きなインスパイアを受けたのが、ジョン・フォン・ノイマンとオスカー・モルゲンシュテルンの「ゲーム理論」である。概要を説明すると、自分と相手が、それぞれにとって最善な行動を取ると考えた時、導かれる答えは唯一つに集約されるというものである。私はこの「ゲーム理論」の考えに則り、再帰関数の中で、「自分の番ならば、相手に勝てる手が見つかったら即 return する。相手の番ならば、相手が勝てる手が見つかったら即 return する。」という処理を加え、計算量を大幅に削減することに成功した。仕様上、早ければ計算量 $O(n^{(61\text{-turn-1})/2})$で終わり、どんなに遅くとも計算量 $O(n^{(61\text{-turn})}/2)$で終わる。この「ゲーム理論」の考えに則る方式に変えたところ、ラスト 7 手目からラスト 13 手目まで計算範囲を拡大することができた。ここで、「ゲーム理論」に則った確率を theory_rate とし、この確率は仕様上 100 %か 0 %しか存在しない状態になった。ここで、theory_rate が 100 %の場合は、このオセロにおいて完全に勝つことが確定したのを意味する。相手がその先どんな手を打とうとも、必ず勝つことができるのである。逆に、theory_rate が 0 %の場合は、相手が相手にとって最善手を打ち続ければ負けてしまうが、相手が最善手を打たなかった場合、勝てる可能性をはらんでいることを意味する。theory_rate が 0 %になった時は、16 手目から 47 手目までで使用した機能を使って処理を続行するようになっている。このプログラムでは、基本的に最後の処理が始まる手番の時(大抵 48 手目か 49 手目)で一番計算時間が長く、長い時は 10 秒から 20 秒もかかることがある。逆に短い時は 1 秒程度で計算が終わってしまうこともある。もし theory_rate が 100 %ならば、現在の手から 2 手目の時点で return するので、計算が早く終わることが多い。逆に、theory_rate が 0 %の時は、最後まできっちり計算した挙句、さらに処理を追加しているため、長い時間がかかる。したがって、処理にかかる計算時間で、負けを察することもある。以上が、最終的なプログラムの概要である。他にも、ターミナル上に計算時間を出力したり、良い手が打てた時や、やむを得ず悪い手を打ってしまった時などの計算結果に応じてメッセージを出力したりするようにした。これにより、オセロ実行中でも自分が良い手を打てているのかどうかを確かめることができる。

7. 13 つ目のプログラム[*7]では、単純に 12 つ目のプログラムの中の処理を反転させて、弱いオセロプログラムとした。計算の手順はほとんど 12 つ目のプログラムと同様なため、割愛する。

まとめ

以上が、今回私が作成したプログラムの概要、工夫、結果である。紆余曲折があったものの、最終的には自分の納得のいくプログラムを作ることができた。

感想

今回の PBL を通して、実際に Python のコーディングをしながら Python の仕様を理解するきっかけになったり、今まで使うことの少なかった再帰関数を積極的に使うことになったりして、自分の成長に繋がったと実感しています。このプログラムで最終回の授業先手後手共に全勝することができたのは本当に嬉しく思います。ありがとうございました。このプログラムを改良するかどうかは分かりませんが、是非融資によるリーグ戦にも参加したいと思います。よろしくお願いします。

*7 https://github.com/Alnya/PBL_Othello/blob/master/OthelloAction-weak.py