立体連結ゲーム Bridget のプログラミング
— 古典的手法再び

竹内郁雄[1]，天海良治[2]

概要　英国の立体型連結ゲーム Bridget（ブリジット）は競技してみると予想外に奥深い．両者それぞれテトロミノの駒 14 個を交互に打ち，8 × 8 の盤面の（上から見て）対辺同士を連結すれば勝利するゲームだが，手の分岐数が多いのと，形状の組合せを取り扱うのでプログラミングは比較的面倒である．Bridget は GPCC2019 の問題にもなっており，2020 夏のプロシンで対戦デビューをした．本稿では，深層学習や最新の GUI を使わない「古典的手法」のみで，定跡の半自動生成を含め，ある程度強いプログラムを開発したことについて報告する．

キーワード　Bridget，古典的ゲームプログラミング，定跡半自動生成

1 はじめに

Bridget は英国の立体型連結ゲームである．日本にはまだ販売代理店がなく，ほとんど知られていないようである．起源はスイスのゲーム作家 Stefan Kögel が創作した Caminos で，2011 年のスイスの家庭ゲーム大賞を取った．しかし，スイス国外に出ることがなかった．これを英国の刑務所で生産することにしたのが Et Games Trading Limited である [1].（仕上げ工程だけらしい）生産場所が刑務所であることが理由なのかどうか分からないが，よく品切れになる．

背景はさておき，このゲームはやってみると意外に奥深く，今回対戦プログラムを作成する過程でさらにその奥深さを実感できた．本稿では，深層学習や最新の GUI を使わない「古典的手法」のみで，定跡の半自動生成を含め，ある程度強いプログラムを開発できたことについて報告する．

2 Bridget のルールと記譜の表現法

図 1 に自作した Bridget の盤面と駒を示した．駒は立方体（cubicle，以下小体と呼ぶ）を 4 つ平面上で隣接するように並べてできたテトロミノである．その形から L, O, S, T に分類される（以下，この 4 種類の駒をまとめて LOST と呼ぶ）．各競技者は，L，S, T をそれぞれ 4 個，O を 2 個，計 14 個持つ（明るい色と暗い色で識別されるが，以下では白と黒と呼ぶ）．盤は小体の面と同じ大きさのマス目を 8 × 8 の正方形に配置したものである．便宜上，正方形の盤面に，上から見たときの東西南北 (E, W, S, N) の向きをつける．

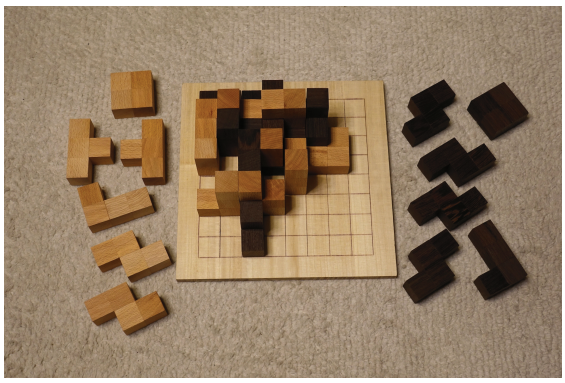


図 1: Bridget の盤と駒（小体は 3cm 立方）

ここでは黒を先手とする．各競技者は交互に自分の駒を盤上に置き，上から見て先に自分の色の小体でいずれかの対辺（東西の辺あるいは南北の辺）をつないだ競技者が勝つ．これをブリッジするという．なお，自分の色は小体の辺同士でつながっていないといけない（斜めにはつながらない）．

駒の置き方には次の制約がある．(1) 駒はその

[1] 東京大学名誉教授　nue@nue.org

[2] プログラマ　amagai@nue.org

小体の少なくとも 1 つが必ず盤面のマス目に合わせて接するように，かつ盤からはみ出さないように置く，(2) 駒の下に空いた空間があってはいけない．

ゲームが進行すると，上記の制約を満たすように手持ちの駒が置けない状況になることがある．そのときはパスしなければならない．もう自分が勝てないと読み切った場合は，敗北を認めるパスをしてもよい．

置ける自分の駒をすべて置いても，両競技者がブリッジできない場合はステイルメイト (stalemate) になり，上から見て自分の色の小体が多いほうが勝者となる．同数の場合は本当の引き分けとなる．

ここまでの記述から明らかなように，Bridget は 2 人完全情報零和ゲームである．

実は上記のルールにはいくつかの変種がある．代表的なものは対辺を自分の色でつなぐときには上から見てではなく，(盤端を除いた) 縦の壁も含めて 3 次元的に同色がつながっていないといけないとする，いわゆる「3 次元ルール」である．しかし，我々がプログラムを書いてみたところ，上から見ただけの「2 次元ルール」でも十分奥深い．2 次元ルールでも両者が最善を尽くすとステイルメイトになりやすい印象があるので，ブリッジに制限のつく 3 次元ルールだとそれがさらに顕著になると考えられる．

Bridget のルールはほぼ直ちに理解できるが，記譜をどうするかには工夫が必要である．Bridget は GPCC2019 の問題として提案されたが，幹事の藤波順久氏が提案した記法が覚えにくかったので，我々はまず直感的理解ができる記法を考案することから始めた．

盤の座標は，盤を 8×8 行列と見たときの i,j 添字（$1\le i,j\le 8$）である．間にカンマを入れず，2 桁の 10 進数 ij として書く．こうすると，局面を表現するための 3 次元配列を，300 要素の 1 次元配列と表現したときの 10 進数の添字と符合し，盤端のチェックも容易になる．上の階は添字に 100 を足すだけである．要するに実装やデバグが楽になる．

駒が裏返されたり，立体的に置かれたりするので，それぞれの置き方に直感的なイメージを結びつけた．例えば，L を盤上にペタリと置けばもちろん L, 裏返して置けば J, 高さ 2 になる 2 通りの置き方はそれぞれ P(elican), H(andle)，高さ 3 になる 2 通りの置き方は，B(oot)，C(rane) など，これで LOST の 16 種類の置き方にユニークな 1 文字を割り当てることができた．実際にはこれに方角を示す E, S, W, N の文字をつけて本質的に異なる置き方（ここでは駒形どぜうに因んで「駒形」あるいは figure と呼ぶ）を表現する．例えば，鶴 (Crane) の首が東 (E) を向いている場合は CE と書く．具体物がイメージされる場合，向きは，このように自然にそちらを向いていると直感される方角，平面文字の場合は最後のストロークが向かう方角である．

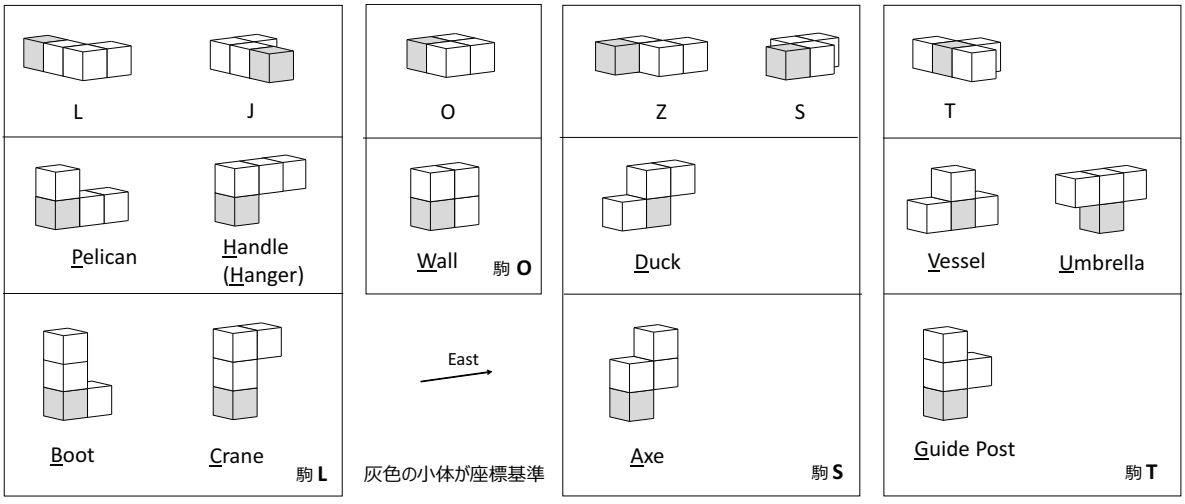


図 2: Bridget の駒の置き方（東向きなので駒形は O 以外はすべて E がつく）

T, S, O の駒は対称性があるので，向きは内部的には東 (E) と南 (S) で正規化する．ただし，着手の入力では西 (W), 北 (N) も可能とする．

詳しくは図 2 を御覧いただきたい．ここで示した置き方はすべて東向きである．だから駒形としてはすべて後ろに E がつく（O だけは例外）．

これでも記譜に使うにはまだ不足で，駒を置いた位置の表現法も決めないといけない．ルールにより，駒は必ず着地するので，その着地の基準となる小体を決める必要がある．Crane であれば着地点はユニークだが，L は 4 つの小体が着地する．考慮の上，以下のように基準を定めた．図 2 で灰色になっている小体である．

(1) 着地点が 1 個しかない場合はそのユニークな小体.
(2) 平面文字は書き始めの位置，ただし，T のようにペンを浮かして 2 ストロークになる場合は 2 番目のストロークの書き始めの位置.
(3) 2 階建て以上の立体で複数着地点がある場合は，折れ曲がっているところの小体.
(4) O と W は置いたときに北西になる（座標 11 に近い）着地点.

2 次元ルールを採用したので，開発には盤を真上から見たテキスト記法を工夫する必要があった．図 3 に示した例で説明すると，1 階の小体は■と□，2 階の小体は◆と◇，3 階の小体は★と☆とした．直感に反するという意見もあったが，あとあとこのデザインは思考の節約に役立った．

```
#16 << WHITE 41 HN >>
▽ 1  2  3  4  5  6  7  8
1  ■  ■  ■  ◆  ・  ・  ・  ・
2  ◇  ☆  ◇  ☆  ☆  ★  ・  ・
3  ◇  ★  ◆  ◆  ■  ◆  □  ・
4  ◇  ・  ★  ☆  ◇  □  □  ・
5  ・  ・  ★  ・  ☆  ・  ・  ・
6  □  ☆  ☆  ◇  ・  ・  ・  ・
7  ・  ・  ◆  ・  ・  ・  ・  ・
8  ・  ・  ■  ・  ・  ・  ・  ・
Black: 1140 (14), White: 1122 (17)
```

図 3: 局面の文字による表現

なお，図 3 の見方は，# の右が 1 を初手とする手番の数，その右が着手の表記（座標と駒形），下の 4 桁の数はそれぞれの競技者の LOST の残り数，括弧の中は上から見たときのその色の小体の数である（ステイルメイトのときに必要になる）．ちなみに図 3 は図 1 の局面を文字情報にしたもので，黒の次の 1 手問題にもなっている．図に書かれた情報だけから問題を解くことができる．

最初のうちは多少の慣れが必要だったが，この記法により，実際の盤や駒をまったく使わずにプログラムの開発を行うことができた．

3 Bridget ゲームの性質

Bridget は 28 手以内に必ずゲームが終了するので，ゲームの木は非常に浅い．しかし，分岐数（選択可能な着手の数）が非常に多いのでゲームの木は極端に幅広である．対称性を全部排除しても，初手は 160 種類で，囲碁の 55 種類の 3 倍に近い．しかも囲碁のように盤端に初手を打つなどの明らかな悪手がない．2 手目，3 手目は，囲碁もそうであるが，対称性がなくなることがほとんどなので，さらに分岐数は多くなる．終盤は分岐数がかなり減るが，少なくとも序盤では分岐が多いままである．中盤では比較的簡単にリーチ（あと 1 手で対辺がつながる状態）になり，それを防御する手の種類はそれほど多くはない．

序盤で間違えると，ずるずると負けに至る展開になりやすいので，序盤で間違った手を打たないことが重要である．幅が非常に広く，浅い先読みでも非常に時間がかかる，しかも評価関数を正しく作るのが（少なくとも現時点では）容易ではない．できれば，事前に序盤定跡を整備しておくのがよい．

中盤では，リーチがかかりやすく，それを防御する手もまたリーチになるといった目まぐるしい展開になり，気が抜けない．ここでも 1 手緩着すると取返しがつかないことになりやすい．リーチをかけることによって先読みはある程度しやすくなるが，リーチが良い手とは限らない．自分の残り駒の状況に常に注意しておかないといけない．特に L の駒は攻撃にも防御にも効果的なので，L の駒が不足すると苦戦しやすい．ただし，絶対というわけではない．

終盤では両者がブリッジできなかったステイルメイトの対策，つまり上から見て自分の小体の色が多くなることを心がける必要がある．もちろんブリッジが最終盤に実現することもある．残り駒の状況がさらにシビアに影響してくる．相手の 1 階の小体が 1 つ離れて位置しているときに，T の駒の U(mbrella) で蓋をして，プラスマイナス 5 で上から見た自分の色を増やせると非常に気持がよい．H(andle) でも同じようなことが可能である．

このゲームを進めるにあたって重要な戦略の 1 つは，相手に分断されない不可侵な自分の色の連鎖をうまく作っていくことである．3 階が連続すれば間違いはないが，2 階や 1 階でも周囲をうまく利用して不可侵状態にしてやれば低コストで長い不可侵連鎖を作ることができる．

このゲームは，相手の小体の上に自分の駒の小体を乗せることによって相手の連鎖を分断することが可能である．だから，黒の初手は白に次に乗られる，つまり相手に利用されるだけという不利感がある．ステイルメイトになると，最後に駒を置くのが，パスでないかぎり白なので，白がどうしても有利となる．これらの印象から，このゲームは後手の白が必勝なのではないかと直感できる．我々の開発したプログラムでも自己対戦させると白の勝率がいつも少し高かった．置いた駒で対辺をつなぐというゲームの目的から考えると，逆説的である．

4 Bridget 競技プログラム

今日的な常識からすれば，深層学習を利用して，勝手に強くなるプログラムを書けばいいのだろうが，Bridget は，我々自身がゲームに精通していない，つまり強いプレイヤになっていないので，自分たちを鍛えるためにも，古典的な手法でプログラムを書くことにした．その過程でいろいろな発見をして，ゲームのノウハウをつかみたいということである．

このように分岐数の大きいゲームでは，候補手の良質な絞り込みが重要であり，それを行うことは我々自身のゲーム感覚の向上に直結する．

プログラムは大まかに以下の 5 つのモジュールからなる．すべて Lisp（筆者らが開発した TAO のエミュレータを C で書いたもの）で書かれている．

(1) **base**: ゲームの進行や局面をデータ構造で表現し，AI が参照する諸々の情報を整備する．例えば，上から見た小体の個数や連結成分のデータ化，リーチの判定，リーチを防ぐ候補手の列挙などは base の仕事である．

(2) **AI**: 局面の数値評価，手筋のパターン検索，候補手の絞り込み，先読み（α-β 法），対戦の制御などを行う．

(3) **projection**: 局面を付帯情報とともにグラフィカルに表示する．上から見た局面，斜め上を 4 つの方向から見た立体図を Postscript で描いて，Ghostscript で表示する（図 4 参照）．

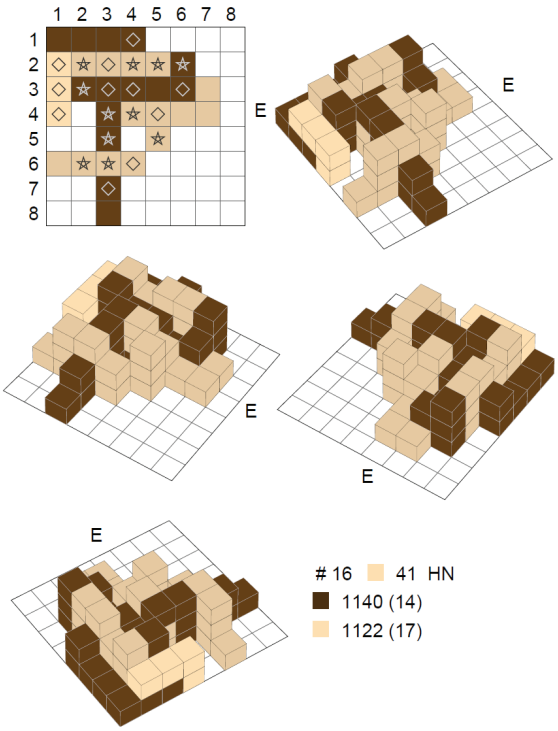


図 4: Postscript で作成された局面の図解

(4) **book**: 序盤の 7 手目またはそれ以前までは，短時間で打てるように定跡情報を提供する．序盤はあまりにも分岐数が多い．まともに先読みすると時間がかかるため，人間との対局をスムーズにする意味もある．

(5) **aiai-match**: プログラム同士で自己対戦させ，定跡の登録に役立つ，あるいは AI の改良に役立つ棋譜を生成する．

以下，それぞれのモジュールの技術的なポイントについて少し詳しく述べる．

4.1 base モジュール

内部データとしての駒形は全部で 51 種類あるが，図 2 に示した東向きの 16 種類の記述から初期化のときに自動生成している．駒形テーブルの各エントリーが持つ情報は，駒形名，基準座標の小体からの他の小体へのオフセットの列，空間が小体直下にある場合はその空間のオフセット，基準座標となり得る盤面の範囲 (bounding box)，駒の種類（LOST のうちの 1 つ），それが置けたときに上から見た小体が何個増えるかの情報（例えば，Handle は 2 個の小体を覆い隠し，自分の小体 3 個を見えるようにする），上から見たときに見える座標のオフセットと，ボードゲームの駒の情報としては多い．逆にこれだけの情報を使わないと局面の正しい理解ができないことを意味する．

着手の探索の順を固定しないために，盤面のインデックス情報をゲーム毎にランダムに組み替えている．大きな効果があったとは言えないが，まったく同じゲーム進行にしない効果は若干あった．

上から見た小体の連結成分 (connected-group) の情報は局面理解や評価の肝であるが，これは着手があるたびにインクリメンタルに作成している．つまり，直前の局面の情報を利用して差分のあったところだけを調べている．ゲームの進行につれて，これらの情報を全部保存し，待ったやゲームの木の先読みで手を戻す (undo) ときには，スタックのようにポップアップして元の状態に戻す．Bridget は分岐数が大きいため，深さ優先探索を行うので，このスタックは枝分かれしない．ゲームの局面のスタックは 1 本で高々深さ 28 である．

局面の情報には，同色の小体の連結グループが，対辺の連結に寄与できるかどうかも含まれている．相手に阻まれていて寄与する可能性のないものは dead と呼び，可能性のあるものは live と呼ぶ．AI の局面評価関数では dead な連結グループは，ステイルメイトのときに上から見える小体の数のみでしか（実質的に）評価しない．

リーチ状態かどうかの判定もこのモジュールで行う．AI 以前の問題だからである．その流れで，リーチを防ぎそうな手の列挙も簡易に行う．簡易というのは小体の配置パターンだけを見て行うからである．それが本当に防ぐ手になっているかどうかは，AI の先読みにゆだねる．

上から見える小体が不可侵（上に乗られることがない）かどうかもこのモジュールで調べる．具体的には不可侵と可侵で色分け（数値分け）した配列（実際には 100 個の要素の 1 次元配列）を作成する．これを利用して AI は局面の評価や，有望そうな着手の選択を行う．一見やさしそうだが，周りの小体の高さや相手の残り駒によって不可侵か可侵かが変わるので比較的面倒なプログラムを書かないといけない．図 2 の中心部にある 8 個の小体の連結グループは途中 1 階もあるが，完全な不可侵グループである．また，相手の駒 L がなくなっていれば，2 階建てはもちろん，1 階でもほかの小体の後ろにあれば，Handle に乗られることがなくなる．

4.2 AI モジュール

対戦を可能にするモジュールであり，対局において最も時間を消費する．この中で重要なのは，局面の評価と，候補手の生成である．

局面の評価については，どこにも明解な基準がないので，まだ浅い競技経験しかない我々の直観のみで作成するしかない．これは試行錯誤の連続であったが，aiai-match モジュールで自動対戦させたり，我々がプログラムと対戦したりして，悪手を発見するたびにパラメータを増やす，あるいパラメータ値の調整を行った．

現在使っているパラメータは以下の通りで，味方・相手で対称的である．

- リーチ 1: リーチがかかっている
- リーチ 2: 別の経路のリーチもかかっている
- 中央のマスを占めている
- 長い live の不可侵連鎖がある
- 1 階，2 階，3 階それぞれの不可侵小体の数，広い空き地につながっているか
- 不可侵ではなくても長い live の連結成分になっているか
- 駒 L の数が相手より何個多いか
- 小体の数

これらの評価値は現在何手目 (Turn-no) かの局面に応じて，線型，あるいは非線型に重みが変化するようになっている．例えば，ステイルメイトに近くなる 24 手以降は最後の小体の数の重みが評価値の大きな要素となるよう Turn-no に応じて線型に伸びていく．中央のマス重視は序盤のみである．リーチ 2 は防ぎようのないダブルリーチを高く評価するためのものである（実際には連結成分分岐の根元で防げる場合がある）．リーチはまだ序盤に毛の生えた程度の段階ではまったく評価しない．早すぎるリーチは相手に乗られて悪い結果になりやすいからである．不可侵連鎖の長さは弱い指数関数で高く評価するようにしている．などなど．

棋譜を見て試行錯誤で調整した結果，評価値のインフレが起こったので，一度強力なデフレ改訂を行ったりもした．現在はある程度安定しているが，それでも，お互いにリーチを掛け合う激しい中盤では，評価値が揺れがすさまじく，このゲームの難しさをまだ制御しきれていない印象がある．

最近の自動対戦の棋譜を見ていると，S と T の駒をバランスよく使うことが必要だった試合がある．このバランスをパラメータとして導入するかどうか考慮中である．

Bridget のような分岐数の大きいゲームでは，候補手の良い絞り込みが特に重要になる．あれこれ実験した結果，8 手目以降は候補手を最大 10 手ぐらいに制限して 3 手先読みにしないと妥当な時間内（最長 3 分）に手を打ってこないことが判明した．予想外に遅いが，これまで述べてきた処理内容がそれほど軽いものではないことから了解されよう．この時間は 2 年以上前の Intel Core i7 CPU のシングルコアで走らせたものである．

つまり，3 手読みの制限の中で，古典 AI 的にいかに頑張るかという問題になった．

まず，相手の長い連結の妨害をする，味方の連結成分を橋渡してより長い連結成分にする，相手に挟まれたところからオープンスペースに顔を出す，4 つの隅のいずれかに重要な連結成分の不可侵拠点を確立する，などといった急所の手筋をパターン検索などで見つけて優先候補手とする．この中には棋譜から見出したかなりアドホックな手（通常の候補手列挙では時間の制約によって無視される手）もある．次に，相手に乗る，あるいは接触する手を列挙して，1 手だけ先読みして評価値の高い 7〜8 手を優先候補手の次の候補手とする．このことから分かるように，原則，離れた場所に駒を打つ手は候補手には入らない．

リーチがかかったときはその防御が最優先になるので上と異なる候補手の選び方になる．base モジュールからもらった防御候補が本当に防御になっているかを 1 手先読みで調べ，残った手を評価値の高い順にソートして候補とする．

あとは通常の α-β 法を使うのだが，上で挙げられた候補手が 15 以上になった場合には改めて 1 手先読みをして候補を絞る．

10 手目から 16 手目あたりの中盤の非常に難しい局面では 3 分ほどの長考をすることがあるが，10 秒程度で手を打ってくることもある．上記の急所の手筋が候補手の頭にある場合は，それらの中に好手があることが多いので，時間制限（通常 2 分あるいは 2 分 30 秒）をつけた上で 4 手先読みを行う．そこでは制限時間内で求まった最善の手を打つ．かなり難しい中盤でも 2 分 30 秒の制限時間で好手を打ってくれることが多い．4 手先読みで初めてブリッジが見えることもあるからである．

まったくの初心者が対戦するときは，上記の制限時間を 1 分 40 秒程度にすると，ときどき間違えてくれるので試合としては楽しくなるだろう．

最終盤は候補手がかなり少なくなるので，ほぼ全探索を行う．だから，ステイルメイトになった場合はほとんど間違えなくなる．

4.3 projection モジュール

立体をグリグリ動かせるように表示するライブラリはすでに多数あるが，敢えて Postscript で描くという「古典的手法」にこだわった．

立方体を透視投影変換し, 8 頂点の xyz 空間座標を XY 平面座標に変換しておく．隣接する小体の座標は XY を保持したコンス（Lisp の cons）ごと共有する．ここまでは事前計算で求めておく．また，上からと四方斜め上からの 5 つの立方体についてすべて事前に計算しておく．base で管理している盤情報から, 視線の奥の小体から最大 3 面を描画し, Postscript ファイルを生成する．Postscript はピクセルを上書きしていくので, 奥から書くこと

で隠面処理が不要になるのがミソである．外部ライブラリなどをまったく使わない生の Postscript を出力するので, 記譜を PDF にして配布する, といったことがとても簡単になった．

なお，対戦に慣れてくると，真上からの表示以外は，悩んだときに眺める程度である．

4.4 定跡モジュール

定跡は序盤のみで，初手から 7 手目までとする．先手の手番で始まり，先手の手番で終わるのは，Bridget は先手が少し不利と思えるからである．

初手はすべての可能な初手を許容する．絶対に不利という初手がないように見えるからである．ただし，初手は勝負がもつやすいものの確率を高くするようにした．定跡の途中でも複数の手を許容していて，そこからランダムに選択して「長く遊べる」ようにした．

定跡は Trie 木で保持する．以下のように，car に着手, cdr に次着手候補リストをもったリストの再帰構造になっている．

（着手 次着手候補 $_0$ 次着手候補 $_1$ …）
（次着手候補 $_0$ 次々着手候補 $_0$ 次々着手候補 $_1$ …）

Trie 木は回転・反転対称を重複させない正規形のみで着手を保存していて，実際の着手を正規形に変換して探索している．

定跡は当初人手で登録していたが，現在は aiai-match の結果から半自動生成している．これについては次節で説明する．

4.5 aiai-match モジュール

当初，局面の評価関数の性能を評価するために，バージョンの異なる評価関数を持つプログラムをランダムな初手で自己対戦させ，評価関数の改良が本当に改良になっているかどうかを調べた．4.1 節で述べたようにゲームの初期化のときに，手の選択の順序がランダムに変わるようにしたので，同じ初手でも異なる展開になることがあり，異なる展開の 10 試合のログを取った．評価関数以外（例えば急所の候補手の列挙）は共通なので，この自動対戦ではほぼ評価関数に使われているパラメータの種類や値の調整のみの検証が行えた．

評価関数がある程度安定したあと，10 回以上のバージョン改訂のたび，自己対戦を行った．ログを調べると，勝ちの見逃しや,（まだ負けていないのに敗戦を認める）リーチ防御の見逃しなどのバグも見つかったが，評価関数が悪くて候補手が正しく絞りこめていないケースも多々発見された．しかし，評価関数のパラメータ調整は，あちら立てればこちら立たずといったもぐら叩き的なところがある．パラメータ調整でうまくいかないと思われたときは，急所の候補手を列挙するよう，ロジックの変更を行った．

現状でも好手の見逃しはあるが，深い先読みで得られる情報を，高々 3 手の浅い先読みで得られるようにすることは，評価関数の改良や（パターンから判定される）急所の候補手列挙だけで達成するのは難しい．しかし，4.2 節で述べたような時間制限つきの α-β 法により，中盤でも部分的な 4 手先読みを行うようになって，この問題はある程度解決した．

自己対戦で発見されるバグやポカがかなり減少してきたので，現在，自己対戦は主に定跡の半自動生成のために使われている．aiai-match モジュールは 160 種類の初手について自己対戦を行い，20 手数以上かかって勝負がつく，つまり勝負がもつれた試合の最初の 7 手を自動的に登録するようにしている．それより短い手数で勝負がついた試合は序盤で敗者が間違った手を打ったと見なして，我々が序盤の手を調査するというプロセスを繰り返す．こうせざるを得ないのは序盤での評価関数の作り方に確たる基盤がないからである．

定跡を登録するための自己対戦は第 7 手までを時間のかかる 5 手先読み（通常は 4 手先読み），急所のある局面では完全な 4 手先読みを行う．人との対戦ではないため時間を気にする必要がないからである．ただし，相当の計算機パワーが必要になる．前述したように，言語は我々が 1980 年代に開発した ELIS という Lisp マシン上で走る TAO である．ELIS のマイクロプログラムを天海が C 言語でエミュレートしたのが celis であり，現時点では Intel や AMD の CPU で当時の ELIS の 200 倍近い速度で動いている [2]．ELIS をエミュレートしているので使用するメモリは高々256MB である．だから，現在のマルチコア，16GB のメイン

メモリといったマシンでは同時に何個も celis を走らせることができる．セルなどに必要なメモリ空間は当然別々である．

ただし，celis 上の TAO は独自のファイルシステムを持っており，並列に走らせて書込みが起こるとファイルシステムの一貫性が壊れてしまう．そこで，celis にファイル書込み禁止のオプションを用意し，試合のログは projection モジュールと同様，外部プログラムを起動し，Linux ファイルシステムに直接書き出すようにした．

自己対戦が走るマシンは Intel Core i9-7900X (3.30GHz–4.30GHz), 10 コア/20 スレッド, DDR4-2400 DIMM 16G × 4 という強力なマシンであり，9 個の celis を同時に走らせている（1 コアは日常業務のために使用）．図 5 は 160 試合の 1 試合あたりの所要時間のヒストグラムである．黒 64 勝，白 91 勝（ステイルメイト 34 試合の内訳は黒 6 勝，白 23 勝，引分け 5）で，平均所要時間 51.8 分，最長 100 分であった．オレンジは定跡自動登録に該当する 20 手数以上かかった試合（84 試合）だが，まだそうでないもの（緑）が多い．黄色は緑色から手作業で定跡を作成したものである（途上）．この自己対戦には序盤のノウハウが必要である．

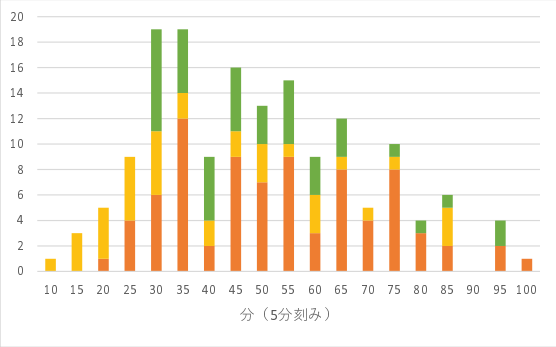


図 5: 定跡探索のための自己対戦時間の分布

5 むすび

2020 年 9 月 26 日にオンラインで開かれた夏のプログラミングシンポジウムでは対戦のみを行った．参加者の合議チームと 3 回対戦し，1 勝 2 敗だった．当時のプログラムは自己対戦を始めたばかりで，規模も半分以下だった．その後改良を続けたプログラムを本稿で紹介した．コメント行を含めて執筆時点で，Lisp としてはかなり大きい 5000 行を超えている．竹内が昨年発表した「3 人の賢者」のプログラム [3] がコメントを含めても 110 行程度だったのに比べると，2 人掛りとはいえ，大規模である．

本稿の最初でも述べたが，我々の研究では Bridget の記譜法を決めることが重要なマイルストーンとなった．直感的な記法を決めることで，開発がかなり容易になった．実際，開発を始めてから今日まで実際の駒や盤面を使用することはまったくなく，本稿の図 1 の写真を撮るために棋譜を並べたのが唯一の利用だった．むしろ，そのときにまったく違うゲームをやっている感覚に襲われたものである．ちなみに付録にこの試合の図解棋譜を掲載しておく．プログラムが自己対戦したものだが，十分に鑑賞に値する．

タイトルにあるように，我々の手法は 40 年以上前の古典的なものである．しかし，そのことによって我々自身が Bridget という奥深いゲームの実力を高めることができた．これはまさにプログラミングによって得られたゲーム自体への洞察と言えるだろう．

しかし，ある意味現代的な「プレイアウトを重ねることによって定跡テーブルを作成する」という手法も導入した．現在はまだ半自動的な定跡生成だが，さらに計算機パワーをかければ全自動生成が可能になると考えられる．これは，Alpha Zero[4] のような手法で，自動的に Bridget の強いプログラムが作れることを示唆している．そのような研究が現れることを期待したい．

[参考文献]

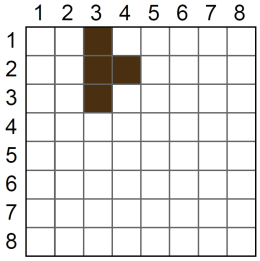
[1] https://etgames.co.uk/product/bridget/

[2] 天海良治: マイクロプログラムの静的変換による記号処理システムの移植とその性能評価, 電子情報通信学会論文誌, Vol.J84-D-I, No.1 (Jan. 2001) pp.100-107.

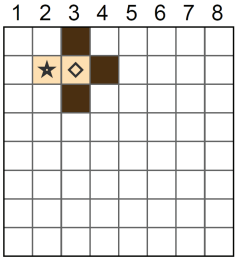
[3] 竹内郁雄: 3 人の賢者の復讐，第 61 回プログラミングシンポジウム報告集，2020 年 1 月．

[4] David Silver et al: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, Science 362 (6419), 2018.

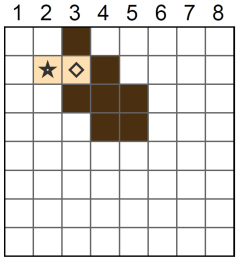
[付録] 図 4（図 1, 図 3）の局面に至る棋譜．この次に黒は何をどう置くべきでしょうか？



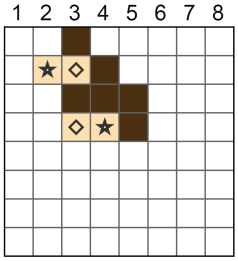
1　■ 23 TE
■ 4243 (4)
□ 4244 (0)



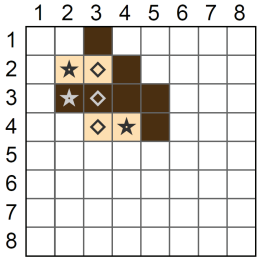
2　□ 22 GE
■ 4243 (3)
□ 4243 (2)



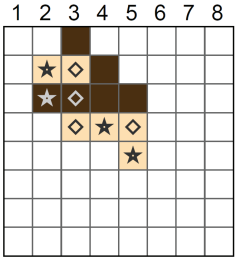
3　■ 34 O
■ 4143 (7)
□ 4243 (2)



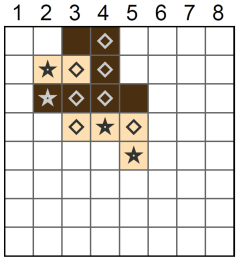
4　□ 43 AE
■ 4143 (6)
□ 4233 (4)



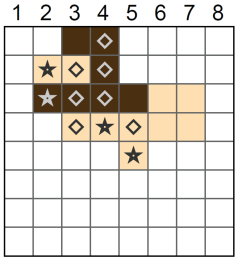
5　■ 32 GE
■ 4142 (7)
□ 4233 (4)



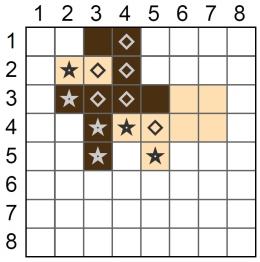
6　□ 55 GN
■ 4142 (6)
□ 4232 (6)



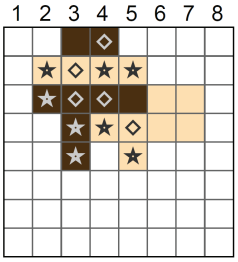
7　■ 14 HS
■ 3142 (7)
□ 4232 (6)



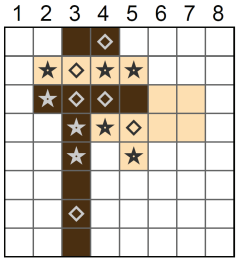
8　□ 36 O
■ 3142 (7)
□ 4132 (10)



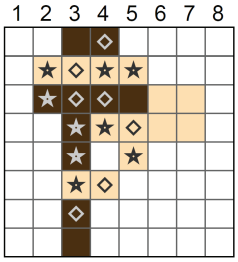
9　■ 53 CN
■ 2142 (9)
□ 4132 (9)



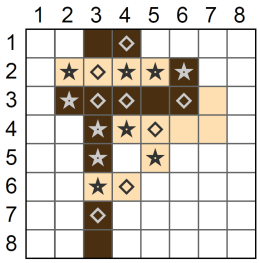
10　□ 25 CW
■ 2142 (8)
□ 3132 (11)



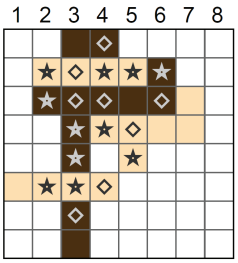
11　■ 73 VS
■ 2141 (11)
□ 3132 (11)



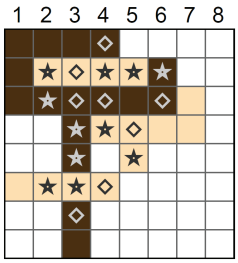
12　□ 64 AW
■ 2141 (10)
□ 3122 (13)



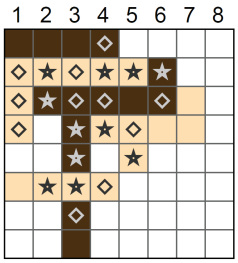
13　■ 26 GS
■ 2140 (12)
□ 3122 (12)



14　□ 62 BW
■ 2140 (12)
□ 2122 (14)



15　■ 31 JE
■ 1140 (16)
□ 2122 (14)



16　□ 41 HN
■ 1140 (14)
□ 1122 (17)