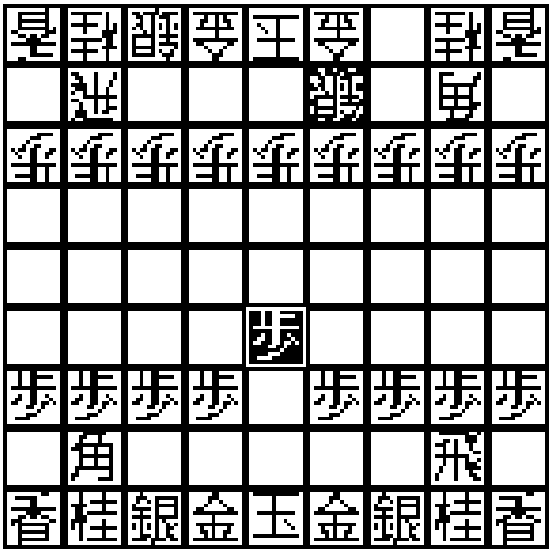
Codexはゲームボーイで
どこまで強い将棋を作れるか

AIコーディングエージェントは、1989年のゲーム機に、どこまで将棋AIを作り込めるのか。
そして、できあがった将棋AIは、実際にはどの程度強いのか。

本稿でいうCodexは、OpenAIのソフトウェア開発用コーディングエージェントである。 リポジトリを読み、コードの編集、コマンド実行、テスト、結果の記録を、与えられた目標と制約のもとで進める。今回使ったモデルは**GPT-5.6 Sol**である。 Codexは開発作業を進めるエージェント、GPT-5.6 Solはその判断とコード生成を担うモデルであり、本稿では両者を区別する。



人間の初手とCPUの応手が終わった対局画面。160×144ピクセルのネイティブ解像度で撮影した。

画面には、9×9の盤と16×16ピクセルの漢字駒が並ぶ。 起動メニューでは人間の先手と後手を選べ、CPUの思考時間も5秒と60秒から選べる。 駒を動かして取り、成り、不成を選び、持駒を打つ、人間対CPUの将棋ソフトである。

このROMは32,768バイト、つまり32KBちょうどに収まっている。 外部ROMやRAMを切り替える**メモリバンクコントローラ**（MBC）は使わない。 通常の着手については、二歩、行き所のない駒、王手放置、打ち歩詰めまで判定する。[1]

完成版が初期局面から探索できるのは、5秒設定で88ノード、60秒設定で1,061ノードだった。 60秒も考えるゲームボーイ将棋は、どの程度の相手と勝負になるのだろうか。

Codexに与えた課題

開発開始時のリポジトリには、README、LICENSE、Gitの除外設定しかなかった。 将棋のソースコードは一行もなく、READMEに次の外枠だけが書かれていた。

- ゲームボーイ上で動き、ROMを32KBに収める。
- 人間とAIが対局でき、人間は先手と後手を選べる。
- AIの思考時間は5秒と60秒から選べる。
- 将棋を知っていれば、独自記号を覚えず操作できるUIにする。
- 動くものを作った後、強さを測る仕組みを導入して定量的に改善する。

実装言語、盤面表現、探索手法、開発ツールは指定しなかった。 CodexはC11のホスト版エンジンから着手し、ゲームボーイ向けのSM83コード、テスト、エミュレータ操作、対局器へ作業範囲を広げた。最初の実装コミットでは17ファイル、約1,700行が追加され、その後も測定と小さな設計変更が細かいコミットとして続いた。

ただし、「ほとんどのコードをCodexが書いた」という説明だけでは分担を捉え損なう。 目標と合否基準を決めたのは人間であり、その枠内で設計、実装、検証を進めたのがCodexだった。 とくに操作感と棋力には、プログラムだけからは決められない問いが残る。

人間は実際にROMを触り、下キーだけで対局が始まる、カーソルの一部が残る、画面が一瞬白くなるといった異常を、画面と操作手順を添えて報告した。 Codexはその報告をエミュレータ上で再現し、原因を絞り、長押しを含む回帰試験に変えた。 棋力評価でも、人間が比較相手を選び、比較の意味が失われた段階で実験を止めた。

この分担は、コードを書く作業量ではなく、誰が問いを設定し、どの結果を合格とみなしたかで見る必要がある。

PC上で将棋を成立させる

ゲームボーイ用コードへ進む前に、Codexは同じCソースをPCで動かせるホスト版エンジンを作った。低速なエミュレータだけで将棋ルールを調べると、一局面の不一致を追うたびに実行時間がかかる。 PC版なら単体テストを高速に回し、外部エンジンとも指し手集合を比較できる。

局面は81升の配列、先後それぞれ7種類の持駒、手番、両玉の位置、手数、千日手用の履歴からなる。盤上の駒は符号付き1バイトで持ち、正を先手、負を後手として同じ駒番号を共有する。 着手前の差分だけを `Undo` に保存し、探索では**make/unmake**、つまり着手と取り消しを繰り返して一つの局面領域を再利用する。

指し手は16ビットに収めた。 移動先と移動元に7ビットずつを割り当て、残る1ビットを成りに使う。 移動元の81から87は7種類の駒打ちを表す。

```
/* bits 0..6: 移動先、7..13: 移動元、14: 成り */
typedef uint16_t Move;
#define MOVE_TO(m)       ((uint8_t)((m) & 127u))
#define MOVE_FROM(m)     ((uint8_t)(((m) >> 7) & 127u))
#define MOVE_PROMOTES(m) (((m) & 0x4000u) != 0)
```

合法手生成は、駒ごとの移動方向をたどって疑似合法手を一手ずつ作る。 成れる手では成りと不成を分け、最下段の歩と香、最下二段の桂には強制成りだけを残す。 駒打ちでは行き所のない駒と二歩を除外する。

その後、候補手を実際に指して、自玉に王手が残る手を捨てる。 歩打ちで相手玉へ王手した場合に限り、相手の合法手がゼロかを再帰的に調べ、詰んでいれば打ち歩詰めとして捨てる。 この順序により、駒の動きだけを扱う生成処理と、局面全体を見なければ決まらない反則を分離した。

71万局面を数える

合法手生成の回帰基準には**perft**を使った。[2] 初期局面から全合法手を再帰的に指し、指定した深さの葉局面数を数える試験である。 この実装では、深さ1から4までが順に30、900、25,470、719,731となる。

なかでも `perft 4 = 719731` を、以後の変更で維持する基準にした。 探索を速くする変更が評価値や手順を変えても、合法手の集合まで変えていないことを約72万局面で検査できる。 make/unmake後に盤面、持駒、玉位置、局面指紋が元へ戻ることもテストした。

さらにFairy-Stockfish 11.1へ同じ局面を渡し、初期局面、駒打ち、王手回避、任意成り、強制成り、左右非対称局面の合法手集合を比較した。 これらは一致したが、打ち歩詰め局面だけは同版の `go perft` が反則手を一手含めた。 外部実装を無条件に正解とせず、本実装側の単体テストと将棋定義を確認して、この一手を既知差分として固定した。

ホスト版にはUSIも実装した。 これにより、局面設定、着手列、探索指示を標準的なプロトコルで送り、自作エンジン同士や外部エンジンとの対局を自動化できた。 ゲームボーイ版を速くする候補も、まずホスト上で合法性と対局結果を調べられるようになった。

8KBのRAMでは普通の探索が動かない

PC上で動くことと、ゲームボーイ上で現実的な時間に動くことは別問題だった。 ゲームボーイのワークRAMは8KBしかない。 初期実装が想定した最大合法手数は600手であり、16ビットの指し手配列だけで1,200バイトを使う。 これを再帰探索の各段へ置く設計は成立しない。

そこで、全合法手を配列へ保存せず、生成器の小さな状態だけを持って一手ずつ返す方式へ変えた。 探索では置換表の手、捕獲と成り、その他の手という順に生成器を走査する。 同じ手を探すために盤面を何度か走査する費用は生じるが、探索の深さごとに1,200バイトを消費せずに済む。

速度の壁も予想以上に低かった。 PC版と同じ4手の静止探索を有効にした最初の計測では、5秒で13ノードしか進まず、深さ1さえ完了しなかった。 固定12,000ノードを読む初期案も、20秒待って終わらなかった。 探索量を先に決める方式を捨て、ハードウェアタイマーで5秒または60秒後に止める方式へ変えた。

何を走査しないか

SM83では、盤面の升番号から筋と段を求める9による除算と剰余が、合法手生成の内側で大きな費用になった。 そこで81要素の筋表と段表をROMに置き、表引きへ置き換えた。 盤上81升を行き先候補として総当たりする方式もやめ、歩、桂、飛車といった駒種ごとの移動ベクトルだけをたどるようにした。 この変更だけで、PC上の初期局面perft depth 4も約0.28秒から0.14秒へ短縮した。

王手判定にも盤面全体の走査が残っていた。 両玉の位置を局面へ1バイトずつキャッシュし、玉から8方向へ最初の駒までたどり、桂馬の二方向だけを別に調べる方式へ変えた。 初期局面のように玉の周囲が埋まった局面では、ほぼ隣接升を見るだけで判定が終わる。 この変更でゲームボーイ上の計測値は、5秒で45から66ノード、60秒で425から729ノードへ増えた。

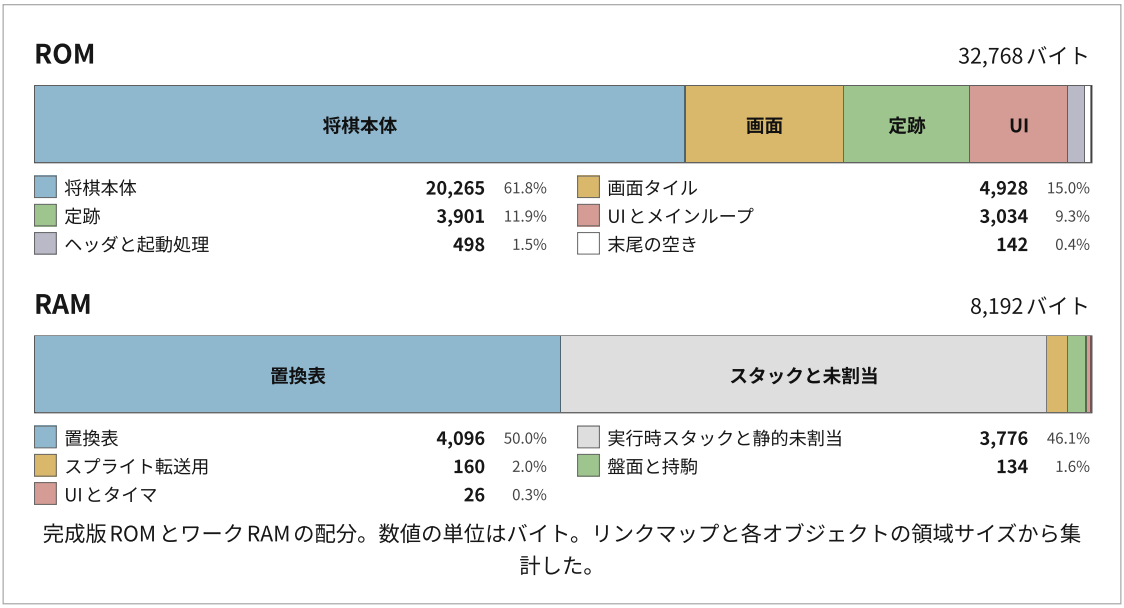
局面の16ビット指紋も、着手のたびに81升と全持駒を数え直すのをやめた。 移動元、移動先、増減した持駒、手番に対応するZobrist値だけを排他的論理和で更新する。 最初に試した単純な線形式は、駒を入れ替えた局面で衝突しやすく、対局でも8局中7敗したため捨てた。 採用品は固定seedから生成した約6KBのZobrist表であり、再計算した指紋との一致をテストしている。

4KBを置換表へ渡す

探索済み局面を再利用する**置換表**には、ワークRAMの半分を割り当てた。 一項目は16ビットの局面鍵、最善手、評価値、深さ、境界種別を合わせた8バイトで、512項目、計4,096バイトである。 残りのRAMには局面、探索状態、描画用の領域などを置く。

ROM と RAM の全体配分

完成版のリンク結果を機能別に分けると、ROM の約 62%を将棋本体が占める。 RAM は置換表だけで半分を使い、静的に確保した領域の合計は 4,416 バイトである。



完成版 ROM とワーク RAM の配分。数値の単位はバイト。リンクマップと各オブジェクトの領域サイズから集計した。

将棋本体 20,265 バイトの内訳には、局面とルール、評価関数、探索に加え、Zobrist 表 6,208 バイトと指し手方向表 648 バイトが含まれる。 定跡 3,901 バイトは、局面と候補手のデータ 3,072 バイトと照合処理 829 バイトに分かれる。 画面タイル 4,928 バイトの大半は、盤と駒に使う 4,096 バイトである。

RAM の「実行時スタックと静的未割当」は、常に 3,776 バイトが空いているという意味ではない。 局面を読む再帰呼び出し、指し手生成器、着手を戻すための一時データが、実行中にこの領域を上端から使う。

駒得から軽量位置評価へ

評価関数は、手番側にとって局面がどれだけ良いかを整数で返す。 最初の項目は駒得であり、歩を 100 点として、香 300、桂 320、銀 450、金 520、角 650、飛 750 とした。 盤上の駒と持駒を同じ価値で合計し、成駒には別の値を与える。

ゲームボーイ版は、駒得へ三種類の軽量な位置評価を足す。 小駒の前進度、大駒の簡易機動力、玉に隣接する守備駒である。 盤全体の利きや王手の危険度を数えるのではなく、81 升を一度走査するあいだに求められる特徴へ絞った。

```
static const int piece_value[] = {
    0,100,300,320,450,520,650,750,20000,
    520,520,520,520,850,950
};

if (base == PAWN) v += advancement * SHOGI_PAWN_ADVANCE;
else if (base == LANCE) v += advancement * SHOGI_LANCE_ADVANCE;
else if (base == KNIGHT || base == SILVER)
    v += advancement * SHOGI_MINOR_ADVANCE;
```

前進度は、自陣最下段から相手陣へ何段進んだかである。 5秒設定では一段につき歩3点、香1点、桂と銀2点を加える。 60秒設定では歩3点、香3点、桂と銀1点となり、香の前進をやや強く評価する。 いずれも歩100点に対して小さい値であり、駒損を埋めるほどではない。

飛、角、馬、龍の**簡易機動力**は、各方向の隣接一升だけを見る。 空升または敵駒なら動ける方向として加点し、その先の長い射線は調べない。 角系では中央へ向かう一升を高くして、初期局面で角道を開ける手と端歩を区別する。

```
uint8_t to = ray_next[square][direction];
int8_t target = p->board[to];
if (!target || owner_of(target) != side) {
    if (type == BISHOP || type == HORSE) {
        int f = square_file[to];
        int edge_distance = f > 4 ? f - 4 : 4 - f;
        mobility += 5 - edge_distance;
    } else ++mobility;
}
```

玉の評価も、囲いの名称や玉の位置を知らない。 玉に隣接する金、銀、と金などの自駒を一枚ずつ数え、5秒設定では4点、60秒設定では10点を加える。 このため「玉を安全な場所へ移す」ことと「居玉の周囲へ金銀を集める」ことを区別できない。 後者を選びやすい棋譜は、実装した特徴をそのまま反映している。

```
if (base == KING) for (int d = 0; d < 8; ++d) {
    uint8_t guard_square = ray_next[i][d];
    if (guard_square >= 81) continue;
    int8_t guard = p->board[guard_square];
    if (king_guard_owner[guard + 14] == side)
        v += SHOGI_LOW_KING_GUARD;
}
```

5秒設定と60秒設定の評価処理は別の関数として生成した。 一つのループ内で設定を何度も分岐すると5秒設定が遅くなったため、ROMを使ってコードを二重化した。 評価を呼ぶたびに静止探索の深さで関数を一度選び、駒ごとのループ内では分岐しない。 位置評価を増やした結果を「駒得より賢い」とはみなさず、係数ごとに対局で比較した。

葉で取り合いを止めない

アルファベータ法が所定の深さへ達した瞬間に評価関数を呼ぶと、駒を取った直後だけを見て高得点にする場合がある。 次の一手で取り返される局面まで読むかどうかが探索深さの偶然で決まる現象を、**水平線効果**という。

この偏りを減らすため、通常探索の葉から**静止探索**（quiescence search）へ入る。 王手でなければ現在局面を評価し、その値を基準に捕獲と成りだけを追加で読む。 王手中は静かな合駒や玉の移動も必要なので、全合法手を生成する。

```
if (!checked) {
    stand = search_evaluate(p, s);
    if (stand >= beta) return beta;
    if (stand > alpha) alpha = stand;
    if (!qdepth) return alpha;
}
while (checked ? next_legal_do(p, &g, &m, &u)
               : next_tactical_legal_do(p, &g, &m, &u)) {
    int score = -quiesce(p, -beta, -alpha,
                         qdepth - (qdepth != 0), s);
    /* alphaとbetaの更新 */
}
```

5秒設定は捕獲と成りを原則1手、60秒設定は2手まで読む。 PC版と同じ4手へ戻すと、5秒で13ノードしか進まなかった初期問題が再発するためである。 1手の静止探索を導入したときはノード数が減ったが、32局の予備対局で5秒設定は12勝16分4敗、60秒設定は23勝7分2敗となり、直前版を上回った。

60秒設定の静止探索では**delta pruning**も使う。 ある捕獲や成りで得られる最大の材料点を現在評価へ足してもalphaへ届かないなら、その手を読まない。 将棋では取った駒が持駒になるため、盤上から消す駒と手に入る不成駒の価値を両方数え、成りの増分も加える。 王手には材料点で測れない価値があるので省略しない。

良さそうな手から先に読む

アルファベータ法は、良い手を先に読むほどalphaとbetaの範囲が早く狭まり、後続の枝を多く捨てられる。 そこで本実装は、置換表に保存された最善手、捕獲と成り、その他の静かな手という三段階で生成する。 全手を点数順に並べる配列は作らず、逐次生成器を段階ごとに走査する。

```
for (int phase = tt_move ? 0 : 1; phase < 3; ++phase) {
    MoveGen g; movegen_init(&g);
    while (phase == 1 ? next_tactical_legal_do(p, &g, &m, &u)
                      : next_legal_do(p, &g, &m, &u)) {
        /* phase 0: 置換表手、1: 捕獲と成り、2: 静かな手 */
    }
}
```

置換表には評価値だけでなく、その値が正確か、下限か、上限かを記録する。 同じ局面へ別の手順から到達したとき、保存深さが足りれば探索を終え、足りなくても保存手を最初に読む。 512項目しかないため衝突は多いが、8KBのRAMで手順序と枝刈りの両方に使える。

根では**反復深化**により、深さ1、2、3と一段ずつ探索する。 一見すると最初から深さ3を読むより余計な仕事が増える。 しかし時間切れ時に直前の完了結果が残り、前回の最善手を次の深さで最初に読めるため、手順序も改善する。

深さを省く条件

捕獲と成りを先に読んだ後、すべての静かな手を同じ深さで調べる余裕はない。 現行ROMは次の手法を、王手中や戦術手を除く条件で使う。

手法	発動する場所	省く計算	読み直す条件
futility pruning	残り深さ1	見込みのない静かな手を全部省く	なし
late move pruning	60秒設定、残り深さ2以下	12手目以降の静かな手を全部省く	なし
late move reduction	残り深さ4以上	7手目以降の静かな手を1段浅く読む	alphaを上回れば元の深さで再探索
principal variation search	60秒設定の内部節点	2手目以降を狭い窓で読む	alphaを上回れば通常の窓で再探索

futility pruningは、残り深さ1で静的評価に100点を足してもalphaへ届かない場合、残った静かな手を省く。 歩一枚分の余裕を見ても現在の最善値へ届かないなら、捕獲も成りもしない一手で逆転する可能性は低い、という近似である。 王手回避、捕獲、成り、置換表の手には適用しない。

late move pruningは、手順序の後ろへ来た手そのものを省く。 60秒設定の残り深さ2以下で、置換表手と戦術手を含めて11手を調べた後は、残りの静かな手を生成せず終了する。 lateの意味は実際の対局手数ではなく、その局面で何番目に探索したかである。

```
if (qdepth > 1 && phase == 2 && depth <= 2 &&
    !checked && move_count >= 11) {
    position_undo(p, m, &u);
    goto moves_done;
}
```

late move reductionは、後ろの手を完全には捨てず、一段浅く読む。 残り深さ4以上で、7手目以降の静かな非王手だけが対象になる。 浅い探索でalphaを上回った手は元の深さで読み直すため、遅い好手を回収できる余地を残す。

principal variation search（PVS）は、最初の一手で alpha を更新できた後、後続手へ「現在の最善手を本当に上回るか」だけを問う幅1の窓を使う。 principal variation とは、その時点の最善手から続く読み筋である。 上回らなければ安く棄却でき、上回った手だけ通常の窓で読み直す。 60秒設定の内部節点だけで有効にし、根局面では再探索費用が実時間で負けたため使わない。

```
score = -negamax(p, child_depth, -alpha - 1, -alpha, s);
if (!s->stopped && score > alpha && score < beta)
    score = -negamax(p, child_depth, -beta, -alpha, s);
```

これらは正解を保つ証明付きの枝刈りではない。 futility pruning と late move pruning は、条件に合う好手を見落とし得る。 PVS と late move reduction は必要なら再探索するが、その再探索自体が遅い CPU では負担になる。 そのため固定問題の正答だけで採らず、実時間相当の対局まで比較した。

反復の途中で時間が切れたら

深さ2を完了した後、深さ3の途中で60秒になった場合、通常は完了済みの深さ2を返せる。 しかし本実装は、深さ3で評価を終えた根候補も条件付きで使う。 前回の最善手より十分高い候補だけを採り、小さな評価差は探索順によるノイズとして無視する。

```
if (!s.stopped) {
    result->best = best;
    result->completed_depth = depth;
} else if (best != principal &&
           best_score >= principal_score + partial_margin) {
    result->best = best;
}
```

採用幅は5秒設定が2点、60秒設定が175点である。 5秒設定は深さ1の途中結果も使わなければ指し手の比較材料がほとんど残らない。 60秒設定は静止探索と枝刈りによる評価の振れが大きいため、歩一枚を超える差がある候補だけを採る。

深さ2以降では、前回反復の評価に256点を加えた上限を探索窓に使う **aspiration window** も導入した。評価が上限を超えた場合だけ通常の広い窓で読み直す。 下限は最初から広いままにし、時間切れまでに得た部分結果を保ちやすくした。

完成版の初期局面計測は、5秒設定が88ノード、完了深さ1、60秒設定が1,061ノード、完了深さ3となった。 通常 ROM は32,768バイト、静的 WRAM は4,416バイトである。 リンク終端から ROM 末尾までの余裕は142バイトしか残っていない。

速くなっただけでは採用しない

8ビットCPUでは、同じ時間に読めるノードが増える変更は魅力的に見える。 しかしノード数は速度の測定値であり、指し手の質そのものではない。 探索順序や枝刈りが変われば、少ないノードで必要な手を読めることも、速く悪い枝へ進むこともある。

象徴的なのが、置換表を1,024項目へ増やした試みである。 項目を8バイトから7バイトへ圧縮すると、強制成りや一手詰の固定問題は少ないノードで解けた。 それでも現行の深さ3探索で512項目版と対局させると、標準局面、Floodgate局面、16局面スイートの三条件すべてで有意に悪化した。 戦術問題の改善を総合棋力へ結び付けられなかったため、4,096バイトの512項目へ戻した。

機械学習した小型の駒位置表も同じだった。 Floodgate棋譜を学習用と検証用に分けると、生のモデルは棋譜の勝者を当てる精度を52.02%から61.77%へ上げた。 ROM向けに量子化すると53.90%まで下がり、固定戦術を解くノード数とゲームボーイ上の速度も悪化した。 32局の予備対局でも5秒設定、60秒設定とも負け越したため、予測精度の改善を棋力改善とはみなさなかった。

根局面のPVSは、同じ734ノードを使う比較ではわずかに勝ち越した。 しかしゲームボーイでは、狭い探索窓で失敗した候補を再探索する固定費がノード数に現れず、60秒で処理できる量が約7.5%減った。 実時間相当の比較は158勝113分241敗となり、ROMも263バイト増えたため不採用とした。

候補を32局ほどで比べる試験は、明らかに悪い案を落とす予備選別に限った。 採否に小差が残る場合は、Floodgate棋譜から作った512の異なる開始局面を先後交換し、1,024局を対局させた。 勝敗だけでなくWilsonの95%信頼区間と両側二項検定値を出し、同じ開始局面と乱数seedを先後二局で共有した。

それでも、統計値が設計判断を自動的に決めるわけではない。 多数の候補から良かったものだけを選べば、同じ局面集合へ過適合する。 実際、候補選択に使っていない別の512局面で改善が再現せず、採用を見送った評価係数もあった。 速さ、固定問題、対局、未使用局面の結果が何を測っているかを分けることが、32KBへ次の機能を入れる条件になった。

3KBの定跡で序盤を補う

探索を軽くしても、5秒では初期局面の全候補を深さ1まで比較するのが精いっぱいだった。 評価関数も駒組みの形を詳しく知らないため、玉を動かさず、金銀だけを近づける不自然な序盤になりやすい。 そこで、探索を始める前の数手を**定跡**で補うことにした。

素材には、オンライン対局場Floodgateが公開する2026年のCSA棋譜を使った。 72,127局から、両対局者のレートが3,000以上、40手以上指され、投了または入玉宣言で決着した15,279局を選んだ。 各棋譜の先頭24手、合計366,696手を集計し、勝者が指した手を3、敗者が指した手を1として重みを付けた。

完成版へ収録したのは312局面、512候補である。 一候補は16ビットの局面鍵、16ビットの指し手、重み、手数の6バイトなので、全体は3,072バイトに収まる。 一局面の候補は最大2手に絞ったが、同じ手へ固定せず、重みに応じて選ぶ。

乱数には8ビットのxorshiftを使った。 乗算も大きな状態も不要で、実機ではタイマーと画面走査位置から初期値を作れる。 同じseedを与えれば同じ分岐を再現できるため、対局試験では先後交換する二局へ同じseedを割り当てた。

16ビットの局面鍵は、異なる局面同士で衝突する可能性がある。 そのため定跡表から手を選んでも、そのまま指さない。 現在局面の合法手を生成し、選んだ手が実在すると確認してから使い、衝突時には同じ鍵の別候補も調べる。

定跡の目的は、ゲームボーイ将棋を一つの戦型へ固定することではなかった。 初手以降には角道、飛車先、中央歩、端歩、玉や金の整備へ進む分岐がある。 しかし完成版の初手は、7六歩が256通り中255、2六歩が1という強い偏りになった。 浅い探索が相掛かりの定跡離脱後をうまく扱えず、7六歩へ寄せた方が予備対局で崩れにくかったからである。

この偏りには代償もあった。 同じ定跡手を選んだ後は、対戦相手もゲームボーイ側も決定論的に同じ手を指しやすい。 最終対局で同一棋譜が何度も現れた原因を、単に「20局試した」と数えてはいけなくなった。

エミュレータで動けば完成ではない

ルールと探索を自動試験しても、人間が実際に触ったときの不具合は残った。 その代表が、Windows版BGBで起動直後に下キーを押すだけで対局が始まった問題である。

Codexはまず、ヘッドレスのPyBoyで押し始める時刻と保持時間を変えた724条件を走査した。 しかし、対局が始まる条件は一つもなかった。 mGBAのデバッガでも下キーは下として読まれ、STARTは混ざらなかった。

ここで「再現しないから入力コードは正しい」とは判断できなかった。 人間から提供されたBGBの設定ファイルと操作条件を使い、BGB 1.6.6をWine上で動かしたところ、下キーを0.5秒押すだけで修正前ROMが盤面へ進んだ。

原因はゲームボーイの入力レジスタJOYPにあった。 八つのキーは4ビットを二行で共有し、同じbit 3が方向キーの行ではDown、ボタンの行ではSTARTを表す。 修正前コードは行を切り替えた直後に一度だけ値を読み、前の行のDownが残ったままSTARTとして解釈していた。

最初は各行で一回だけ読み捨てたが、右キーがAボタンへ化けて勝手に着手する問題が残った。 最終版では行を選んだ後に四回読み捨て、五回目だけを採用する。 DownとSTARTだけでなく、同じbitを共有する上下左右と四ボタンの全組を長押しして確認した。

自動試験も、人間の押下時間に合わせた。1フレームだけの入力に加え、10、30、120フレーム保持しても一回だけ動くことを調べる。キーを放す途中の一時的なゼロを解放と誤認し、同じ物理押下を二回受け付けた問題には、無入力が2 VBlank続くまで次の操作を受け付けないようにした。

画面が一瞬白くなる問題は、カーソルを動かすたびにLCDを止めていたためだった。座標計算とタイル選択をワークRAM上で先に終え、VBlank中は確定済みの少量のバイトだけをVRAMへ書く構成に変えた。最後の画面だけでなく、録画の途中に一フレームだけ白画面や半更新がないかも検査した。

小さな起動コードの大きな不具合

別の画面崩れは、Cコードではなく専用の起動コードにあった。ゲームボーイ用のC処理へ入る前に、アセンブリで初期値付きデータを転送し、ゼロ初期化領域を消去する。ところがゼロクリアの残量判定に使った論理和の結果がAレジスタへ残り、その非ゼロ値をメモリへ書き続けていた。

この不具合は描画更新件数を19にし、不正なVRAMアドレスを作り、割り込み許可レジスタまで意図しない値に変えた。ループへ入る前に `xor a` を加えてゼロを作り直し、静的領域、割り込み、スプライト転送を明示的に初期化した。

人間が見つけたのは、押したキーと違う動作や、画面に残った数ピクセルだった。Codexはそれを、再現用の設定、長押し列、レジスタ値、録画、回帰試験へ変換した。この往復がなければ、PyBoyとmGBAだけで動くROMを完成品としていた可能性がある。

ゲームボーイROMを対局させる

棋力を測るときも、PC版の軽量探索で代用するわけにはいかない。同じCソースであっても、SM83で一手を生成する時間、タイマー割り込み、画面処理の負荷が、時間切れ時に採用する手を変えるからだ。

そこで、対局専用の評価ROMをmGBA上で動かした。外部から局面と命令を書き込めるワークRAM上の**mailbox**を設け、mGBAデバッガ経由で読み書きする。PC側ではこの操作をUSIエンジンとして見せ、通常の対局runnerへ接続した。

```
対局runner
    ↓ USI
mGBAデバッガ
    ↓ WRAM mailbox
評価用ゲームボーイROM
    ↓ bestmove
cshogiによる合法性と終局判定
```

通常ROMと評価ROMは、探索本体を同じオブジェクトファイルからリンクする。評価ROMが返した手はcshogiの局面へ適用し、合法手、手番、詰み、終局理由を検査した。この構成により、ゲームボーイ用の探索を実際の命令速度で何局も対局させられる。

ただし、評価ROMを作っただけでは通常対局の時間条件を再現できなかった。最初の評価ROMはLCDを止めており、5秒で90ノードを探索したのに、通常ROMは88ノードだった。通常ROMではVBlank割り込みとスプライトのDMA転送がCPU時間を使うためである。

評価ROMでもLCDを動かし、同じVBlankとDMAの負荷を与えると、初期局面の結果が5秒で88ノード、60秒で1,061ノードへ一致した。この修正前に生成した棋譜は評価対象から除外した。「画面を表示しない評価版の方が少し速い」という2ノードの差も、5秒で88ノードしか読めない探索には無視できない。

最低設定のやねうら王との対局

比較相手には、全合法手から一様に手を選ぶ**random**、やねうら王の純粋駒得評価版である**MaterialLv1**、水匠5を用意した。当初は対戦相手のノード数を調整し、ゲームボーイ版と互角になる点を求める計画だった。

しかし、やねうら王と水匠5へ指定できる最小値の`go nodes 1`でも差が大きかった。16開始局面を先後交換した32局の予備対局では、5秒設定が水匠5へ1勝1分30敗、MaterialLv1へ1勝31敗だった。60秒設定も、水匠5へ1勝31敗、MaterialLv1へ12勝1分19敗で、すべての条件が五分を下回った。

互角になるノード数は正の整数で指定できる範囲より下にある。 この時点で、水匠5との追加対局、未使用局面での検証、互角点の精密測定を中止した。

代わりに、平手初期局面からの対局を各条件20局ずつ保存した。 これは棋譜例であり、一般的な勝率の推定ではない。

思考時間	相手	GB勝	引分	GB敗
5秒	MaterialLv1	0	0	20
5秒	random	17	0	3
60秒	MaterialLv1	3	0	17
60秒	random	20	0	0

random戦は、5秒設定と60秒設定の各20局がすべて異なる着手列になった。 一方、MaterialLv1戦は各設定とも、着手列が実質3種類しかない。 重複数は10局、7局、3局であり、定跡分岐後に両者が同じ決定論的な手順を再生した結果だった。

したがって、60秒設定の3勝を「独立した20回の試行で3回勝った」と解釈してはならない。 三局はすべて同じ棋譜であり、一つの勝ち筋を三回再生した例である。 表は保存棋譜の内訳を示すが、信頼区間を付けて母集団の勝率へ広げられる標本ではない。

「駒得だけに負けた」という誤解

MaterialLv1という名前は、単純な一手先の相手を連想させる。 しかし `Lv1` が指すのは**静的評価関数**だけであり、探索器全体ではない。 局面の位置、玉形、利きを評価へ加えず、先後の駒価値の差だけを返す点が「Material」である。

その評価値を使う探索器は通常のやねうら王である。 ルートの全合法手を列挙し、反復深化したアルファベータ探索を行い、葉では捕獲、成り、王手、王手回避を再帰的な静止探索で調べる。 一手詰めの専用判定、置換表、SEEによる不利な交換の除外、捕獲手の手順序も残っている。 **SEE**（static exchange evaluation）は、ある升で駒を取り合った場合の材料収支を、通常探索より安く見積もる処理である。

さらに、`go nodes 1` は一手を指して直ちに止まる指定ではなかった。 やねうら王v7.6.3は、深さがゼロならノード上限を検査する前に静止探索へ入り、静止探索中には上限を検査しない。 そのため最初の深さ1反復では、全ルート手と各手の静止探索を終えてから停止する。

平手の保存棋譜では、MaterialLv1が一手に報告したノード数は5秒設定との対局で平均86.0、60秒設定との対局で平均104.2だった。 指定値は1でも、実際の処理量は最大319ノードに達し、すべての着手で深さ1を完了した。

対するゲームボーイ版は、通常探索と静止探索の各ノードで時間切れを検査する。 5秒設定の非定跡715手では359手が深さ0で、全合法手を同じ深さで比較し終えていない。 約90ノードという近い数字でも、どこまで完了してから止まるかが違う。

ゲームボーイ版の評価関数は、純粋な駒得より項目数が多い。 駒の前進度、飛角の簡易機動力、玉周辺の守備駒も数える。 しかし評価項目が多いことは、評価が正確であることを保証しない。 駒価値の比率、特徴の粗さ、探索未完了、短い静止探索が組み合わされば、純粋駒得評価の相手より悪い手を選ぶ。

今回の結果は、「位置評価を持つゲームボーイ版が、一手先の駒得だけに負けた」という比較ではない。浅くても全候補と取り合いを調べ終える現代の通常探索器と、8ビットCPUで時間切れになるよう切り詰めた探索との差である。 評価関数だけの優劣を知るには、同じ探索へ二つの評価関数を差し替える別の実験が必要になる。

棋譜に現れた強さと弱さ

保存棋譜は勝率推定には使えないが、どのような手を選んだかは観察できる。 random戦とMaterialLv1戦から一局ずつ取り上げる。

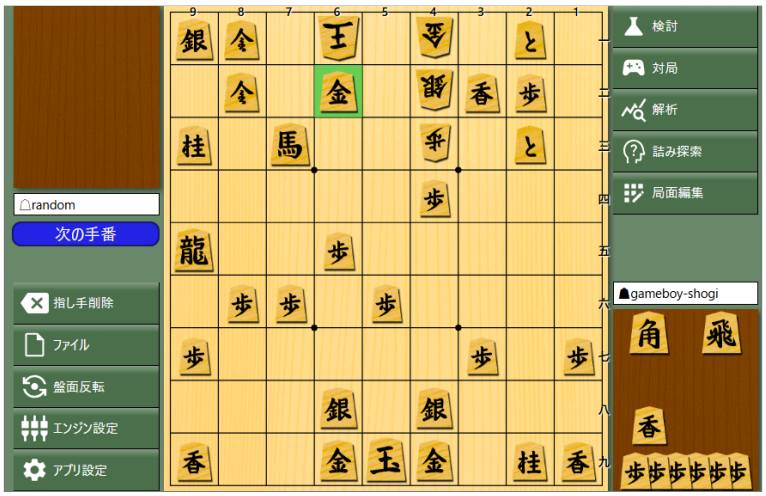
局面図は、保存した棋譜をShogiHomeで再生した。

5秒設定対random（ゲームボーイ先手）

5秒設定は、玉を居玉のままにして金銀を近づける。 定跡を外れた後の駒組みとして自然とは言いにくく、玉の安全を周辺の守備駒数だけで測る評価の粗さが表れている。

ゲームボーイ先手の5秒設定対randomの序盤。玉を動かさず、金銀を近くへ寄せている。

それでも終盤には、相手玉の周囲へ飛車、龍、馬、金を運び、詰みまで到達した。 randomは局面の目的を持たず、王手や駒得も他の合法手と同じ確率で選ぶ。 ゲームボーイ版の囲いは奇妙でも、駒得、前進、王手を探索する動きには方向がある。

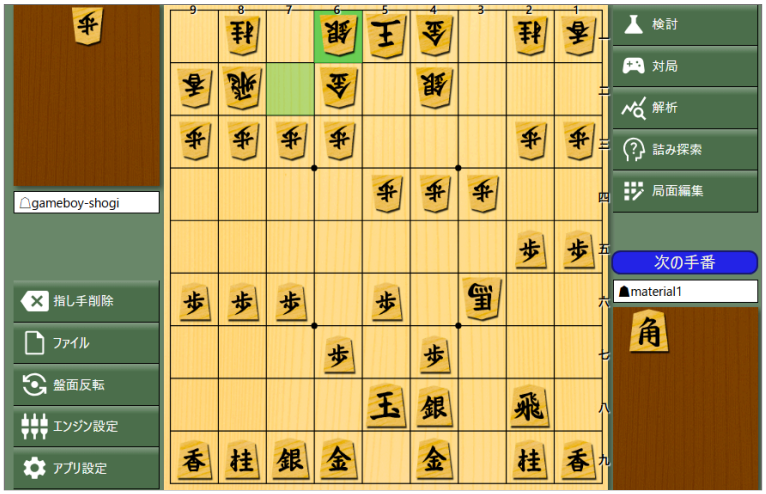


ゲームボーイ先手の5秒設定対randomの終局図。相手玉の周囲へ攻め駒を集めて詰ませた。

この一局だけから「randomへ何割勝てる」とは言えない。 ただし異なる着手列となった保存40局では37勝しており、全合法手の一様選択より目的のある手を指す段階には達している。

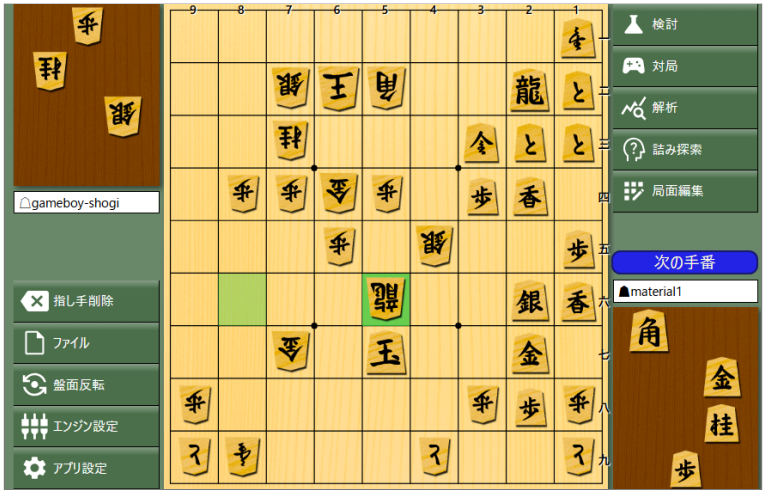
60秒設定対MaterialLv1（ゲームボーイ後手）

60秒設定でも、玉を動かさず金銀を近づける傾向は残った。 探索が深くなっても、評価関数が好む玉周辺の形そのものは変わらない。



ゲームボーイ後手の60秒設定対MaterialLv1の序中盤。居玉の周辺へ金銀を集めている。

紹介する勝ち棋譜では、多くの駒を相手へ渡しながら、最後は龍で詰ませた。 MaterialLv1の静的評価は玉の安全を数えないが、通常探索器の一手詰め判定は働く。 この局面は、ゲームボーイ版が偶然の非合法手で勝ったのではなく、合法な詰みへ到達した例である。



ゲームボーイ後手の60秒設定対MaterialLv1の終局図。駒を渡しながらも、相手玉を詰ませた。

ただし60秒設定の3勝は、すべてこの同じ着手列だった。 三つの異なる勝ち方を見つけたのではなく、同じ定跡分岐から同じ勝ち筋を再生した。

1989年の本体に、何年の技術を載せたか

ゲームボーイが日本で発売されたのは1989年4月21日である。[3] Codexが古いゲーム機へ現代の将棋AIを移植した、と説明すると、探索手法まで発売後に生まれたように読める。 しかし論文の発表年をたどると、ROMへ入れた探索の骨格はゲームボーイより古かった。

探索の骨格は本体より古い

静止探索の考え方は、ゲームボーイ発売の39年前まで遡る。 Claude Shannonは1950年の論文で、駒交換の途中に評価関数を適用しても意味が薄く、強制的な変化を比較的静かな局面まで進めてから評価する方法を記した。[4] 本実装が捕獲と成りを追加で読む理由と同じである。

公開年	技術または資料	本実装との関係
1950	静止局面まで読む考え方	捕獲と成りを読む静止探索
1970	Zobristの盤面ハッシュ	16ビット局面指紋と置換表
1975	アルファベータ法の解析	通常探索の骨格
1980	Scout	後続候補を狭い窓で確かめる系譜
1983	NegaScout	上回った候補だけを再探索する系譜
1986	PVSとaspirationの比較	60秒設定の内部探索と探索窓
1989年4月21日	ゲームボーイ発売	比較の基準日

Zobristは1970年の技術報告で、チェス、チェッカー、囲碁を対象に盤面をハッシュ化する方法を示した。[5] アルファベータ法も、KnuthとMooreが1975年に正しさと計算量を論じている。[6]

PVSの狭い探索窓にも発売前の系譜がある。 Judea Pearlは1980年にScoutを発表し、Alexander Reinefeldは1983年にその改良であるNegaScoutを発表した。[7] 1986年のレビューは、PVSとaspirationを既存のゲーム木探索手法として比較している。[8]

late move reductionの年代は、一本の発明年へ整理できなかった。 手ごとに探索量を変えるSEX algorithmの論文は1989年3月に発表されており、ゲームボーイ発売より約7週間早い。 一方、アルファベータ法へfutility pruning、null move pruning、late move reductionを組み合わせた将棋での公開評価は2012年である。[9] 同論文は、生の分岐数が平均約80の将棋終盤で、実効分岐数を2.8まで減らせると報告した。

現ROMのlate move pruningとdelta pruningについては、初出を確認できる一次資料を特定できなかった。 由来が不明な手法を「ゲームボーイ発売後の発明」とは分類しない。

発売後にそろった開発基盤

発売後に登場したのは、ROM内の基本探索よりも、実装、検証、対局を反復するための基盤だった。次の年は、公式仕様、プロジェクトの履歴、原案、運用記録で公開時期を確認できる。

公開年	技術または基盤	この開発で担った役割
1999	SDCCの公開開発基盤	CからSM83用ROMを生成
2005	Git	小さな変更と不採用案の履歴を保存
2007	USI原案	エンジンと対局器をテキストで接続
2008	Floodgate連続対局	定跡用棋譜と検証局面を提供
2011	C11	ホスト版とゲームボーイ版の言語仕様
2016	mGBAのゲームボーイ対応	デバッガ、自動操作、実時間測定
2025	Codex	実装、試験、測定、記録を反復

SDCCは1999年12月に公開開発の場をSourceForgeへ移し、現在はSM83を対象CPUに含む。[10] Gitは2005年に生まれ、このリポジトリでは動く節目と実験の差分を残した。[11]

USIの初稿は2007年1月24日付で、GUIと将棋エンジンを標準入出力のテキストで分離する目的を明記している。[12] 本プロジェクトはこの分離を使い、ホスト版を対局器へ接続した。 Floodgateの連続対局モードは2008年2月9日に運用を始め、その公開棋譜を定跡と異なる開始局面の両方へ利用した。[13]

C言語自体はゲームボーイより古いが、採用したC11規格の発行は2011年12月である。[14] mGBAは2013年に始まり、ゲームボーイ対応を2016年に追加した。[15] 実機相当のノード数とRAM上の探索結果を自動で読む環境は、このエミュレータのデバッガを前提としている。

Codexのクラウド型ソフトウェア工学エージェントが研究プレビューとして公開されたのは2025年5月16日だった。[16] アルファベータ法や静止探索を発明したのではなく、既存の手法を32KBと8KBへ合わせ、試験と対局を回し、不採用理由までリポジトリへ残す役割を担った。

したがって、このROMを構成する機械語の中心は、1989年の知識でも設計できた可能性がある。 同じ開発過程は再現できない。 標準プロトコル、公開棋譜、デバッガ付きエミュレータ、分散型の履歴管理、コーディングエージェントは、いずれも発売後にそろったためである。

Codexに任せられたこと、人間が決めたこと

Codexは、将棋エンジンとROMだけでなく、その正しさと強さを調べる仕組みまで作った。 合法手生成をperftと外部エンジンで検査し、エミュレータをデバッガから操作し、Floodgate棋譜を定跡と開始局面へ加工した。 評価ROM、USIブリッジ、並列対局器、統計集計、再現手順、失敗した候補の記録も残した。

大量の候補比較は、とくにコーディングエージェントへ任せやすかった。 定数や実装を一つ変え、ルールテスト、perft、固定戦術、実機ノード、対局を順に回し、悪ければ戻す作業を繰り返せる。 不採用案まで文書へ残したため、同じ案を別の条件で再評価するときも、過去の前提と比較できた。

一方、人間は、実際に触ったときの違和感を最初に見つけた。 下キーの誤認も、カーソルの残像も、最終画面の自動比較だけでは見つからなかった。 人間が画面と操作手順を示した後、Codexが再現可能な試験へ変換した。

棋力評価の意味も人間が決めた。 どの相手なら比較として理解しやすいかを選び、最小条件ですでに大差となったとき、精密測定を続けないと判断した。 途中まで終わった108局を結果へ足せば数字は増えるが、選択途中標本を正式結果へ混ぜない決定も人間の役割だった。

「どの程度強いのか」は、対局数を増やせば自動的に答えが出る問いではない。 独立していない棋譜を数えず、比較相手の `nodes 1` が何を実行するかをソースまで戻って調べ、測定が支える範囲まで結論を狭める必要がある。

ゲームボーイでは頑張ったけれど弱すぎる

完成したROMは32KBに収まり、ゲームボーイ上で成り、駒打ち、二歩、王手放置、打ち歩詰めを扱う将棋を指す。 5秒設定では88ノード、60秒設定では1,061ノードを探索し、保存したrandom戦40局では37勝した。

しかし60秒考えても、現代の通常探索器へ純粋な駒得評価を載せたMaterialLv1には大きく及ばなかった。 保存40局では3勝しただけで、その3勝も同じ棋譜の再生だった。 水匠5とMaterialLv1の最小ノード設定より弱かったため、互角点を測る当初の計画も成立しなかった。

弱さの原因を「ゲームボーイだから」で終わらせず、探索範囲として測れたことは成果だった。 5秒設定は約90ノードを使っても、非定跡着手の半数で深さ1を完了しない。 一方、MaterialLv1は指定1ノードでも、全ルート手と再帰的静止探索を終えてから止まる。 この停止方式と探索の完成度の差が、評価項目を少し増やす工夫では埋まらなかった。

Codexは設計、実装、テスト、エミュレータ自動化、大量対局を自走できた。 人間には、画面と操作の異常を感じ取り、比較の意味を問い、実験を止める判断が残った。

ゲームボーイでは頑張ったけれど、将棋ソフトとしては弱すぎる。 それでも、どこまで作れたかだけでなく、どこから届かなかったかを同じROMで測り、説明できるところまで開発を進められた。

---

1. 千日手だけは省メモリ化している。直近16局面の16ビット指紋を保存し、同じ指紋が4回現れた時点で引き分けとする。連続王手の反則負けは扱わず、16局面より長い周期や指紋衝突には完全対応しない。↩
2. performance testの略称として使われる。局面を評価せず、合法手生成と着手、取り消しだけを再帰的に実行するため、将棋エンジンではルール実装の検査に使える。↩
3. 任天堂のニュースリリースは、ゲームボーイの発売日を1989年4月21日としている。↩
4. Claude E. Shannon, “Programming a Computer for Playing Chess”, 1950.↩

5. Albert L. Zobrist, “A New Hashing Method With Application for Game Playing”, University of Wisconsin Technical Report 88, 1970.↩
6. Donald E. Knuth and Ronald W. Moore, “An Analysis of Alpha-Beta Pruning”, 1975.↩
7. Judea Pearl, “SCOUT: A Simple Game-Searching Algorithm with Proven Optimal Properties”, 1980. Alexander Reinefeld, “An Improvement to the Scout Tree Search Algorithm”, 1983.↩
8. T. A. Marsland, “A Review of Game-Tree Pruning”, 1986.↩
9. David Levy, David Broughton and Mark Taylor, “The SEX Algorithm in Computer Chess”, 1989. Kunihito Hoki and Masakazu Muramatsu, “Efficiency of Three Forward-Pruning Techniques in Shogi”, 2012.↩
10. SDCC 公式サイトと SourceForge のプロジェクト履歴による。↩
11. Git 公式文書 A Short History of Git による。↩
12. Tord Romstad, “The Universal Shogi Interface, draft 1”, 2007年1月24日。↩
13. コンピュータ将棋対局場の公式歴史ページによる。↩
14. ISO の ISO/IEC 9899:2011 は、発行日を2011年12月としている。↩
15. mGBA の公式年表による。↩
16. OpenAI, “Introducing Codex”, 2025年5月16日。↩