自己対戦的手法を用いた格闘ゲームプレイヤの作成

○持田 尚監, 永田 裕一（徳島大学）

Creating a Fighting Game Player Using a Self-Playing Method

* N. Mochida and Y. Nagata (University of Tokushima)

Abstract－ AlphaGo Zero has repeatedly created and learned training data by playing against itself, and has acquired powerful strategies in Go. AlphaZero is a generalized version of AlphaGo Zero. AlphaZero uses Monte Carlo tree search for action selection, but it is difficult to apply it to fighting games due to the short time available for search. Therefore, the maximum depth of the tree to be explored was determined to reduce the search time.
Key Words: Reinforcement learning, Self-play, Fighting game

1 はじめに

近年, 対戦型ゲームの強力な戦略を学習するための様々な研究が行われている. 対戦型ゲームの１つである囲碁分野で強力な戦略の獲得に成功した手法としては, AlphaGo[1] や AlphaGo Zero[2] が特に有名で、これらはモンテカルロ木探索(MCTS)と自己対戦を組み合わせた手法である.

AlphaGo では, はじめに人間のプロプレイヤ同士の対戦棋譜を訓練データとして学習をさせ, その後, 学習済みの AI 同士で対戦した棋譜からさらに学習を行う. 最終的に AlphaGo は, 人間の世界チャンピオンに勝利する程の性能を獲得した. 一方, AlphaGo Zero は自己対戦によって AI 自らが訓練データを作成し, それを学習する操作を繰り返すことで戦略の強化を行う. AlphaGo とは異なり, AlphaGo Zero は人間のプレイヤの対戦棋譜を学習に利用していないにも関わらず最終的に獲得した性能は AlphaGo を超えるものであった.

AlphaGo Zero を囲碁以外でも利用可能なように一般化した手法として AlphaZero[3] が存在する. AlphaZero は囲碁やチェス, 将棋においてそれぞれ世界チャンピオンプログラムに勝利し, 囲碁以外のゲームでも有効な手法であることを示した.

このようにボードゲーム分野において, AlphaZero は人間のプレイヤを超える戦略を持つ AI の作成に成功しているが, 対戦型ゲームの１つである格闘ゲームにおいて効果的な学習手法はまだ確立されていない.

この原因は, 格闘ゲームでの行動選択は相手の多様な行動について考慮しながら, 短時間で行動決定を行う必要があるためだと考えられる. 格闘ゲームでは, 自プレイヤと相手プレイヤが同時進行で行動選択をするため, プレイヤが何も行動をしなくともゲームが進行していく. また, 将棋や囲碁と比較して行動選択に利用可能な時間が短く, 短時間で行動決定を行うアルゴリズムが必要になる. さらに, ボードゲームの場合には, 各プレイヤが順番に行動するため１人のプレイヤが選択した行動によって次の状態が確定するが, 格闘ゲームは２人のプレイヤが同時進行で行動選択しており, 自分の行動が同じであっても相手の行動によって次のゲーム状態が変化する. 加えて, 相手の行動を事前に把握することができず, 次のゲーム状態を正確に予測することが困難である. そのため, 相手が実行可能である多様な行動について考慮しながら, 自分が行動を実行した後のゲーム状態を求める必要がある.

本研究の目的は, AlphaZero のような自己対戦的手法を格闘ゲームに適用し, AI に戦略を学習させることである. AlphaZero の手法をそのまま格闘ゲームに適用した場合, 行動選択で利用する MCTS の探索時間が格闘ゲームで利用するには長くなることが考えられる. また, AlphaZero の手法では自分と相手が交互に行動選択するものとして MCTS を行うが, 実際のゲームで各プレイヤが同時進行で行動しているため, シミュレーションで得られるゲーム状態が実際のゲームに則していない可能性がある.

そのため本稿では, AlphaZero の手法での探索木の構造に OLS を用い, 自分と相手プレイヤが同時に行動決定するものとして探索を行うことを提案する. OLS はシミュレートが毎回ルートノード(木構造の頂点のノード)から末端のノードまで行われ状態を再計算する. これにより自分と相手プレイヤが実行可能な多様な行動の組み合わせについてシミュレートできる. また, OLS において探索する木の最大の深さを事前に決めておくことで探索時間の短縮を図った.

提案手法の性能評価は, FightingICE[5] の公式ホームページで公開されている既存手法 AI と対戦させることにより行なった.

2 関連研究

2.1 AlphaZero

AlphaZeroは, ニューラルネットワーク(NN)とMCTSを組み合わせた手法で, MCTSではNNを利用して次の行動を選択しながら探索を行い, プレイヤはMCTSの探索結果によって次の行動を決定する. MCTSを利用することで自己対戦結果から訓練データが作成され, 訓練データからNNの学習を行うことができる. そのため, ルールを教えるだけで事前に訓練データを用意することなく学習が行え, NNのパラメータを学習することで性能を向上させる. NNの出力は２つに分岐しており, 現在のゲーム状態を入力することで方策と価値を出力する. 方策は行動の確率分布で各行動の有望さを表し, 価値は勝敗予測で状態の有望さを表す.

自己対戦の大まかな流れをFig. 1に示す. 囲碁や将棋などのボードゲームの場合, 自分と相手の行動選択が交互に行われる. まず, 環境から現在の状態s_tを受け取り, それを基にMCTSを行う. MCTSの結果として行動の確率分布π_tが得られ, プレイヤはπ_tにしたがって次の自分の行動a_tを選択する. 次に, 行動a_t実行後

のゲーム状態s_{t+1}を受け取り，自分の行動決定時と同様に相手の行動a_{t+1}を決定する．これらの処理をゲーム終了まで繰り返し，対戦結果から自プレイヤの評価zが得られる．自己対戦によって訓練データ[s_t, π_t, z], [s_{t+1}, π_{t+1}, -z]が作成され，NNの方策とπ_tの類似度が最大でかつ，価値とzの差が最小になるように，NNのパラメータが更新される．

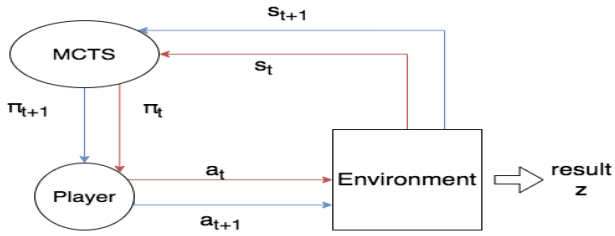

Fig. 1:Self_play process

AlphaZeroはNNとMCTSを組み合わせた探索により行動決定を行う．初期の探索木は，ルートノードとルートノードから見て子にあたるノードのみを持つ．各ノードはゲーム状態を表し，ノードとノードをつなぐ線(エッジ)は実行した行動を表している．探索はルートノードから始まり，(1)式に基づいて次の行動を決定しながらノードを移動していき，子ノードを持たないノード(リーフノード)まで移動する．

$$\frac{w}{n} + C * p * \frac{\sqrt{N}}{n+1} \quad (1)$$

ここで，wは累計価値，nはそのノードの試行回数，Cは定数，Nは親ノードの試行回数，pは行動の選択確率で，現在のノードのゲーム状態を入力した際にNNから出力された方策のうち，現在評価しているノードに対応する行動の値である．(1)式で行動選択を行うことにより，価値の期待値，手の確率が高く，試行回数が少ないノードを選択できる．

リーフノードまで到達した際には，現在いるノードが持つゲーム状態をNNに入力してノードの有望さを推論し，ノードの展開(子ノードの作成)を行う．NNを探索に利用しない場合には複数回のプレイアウトによりノードの有望さを求める必要があるが，AlphaZeroではNNから得られた価値でノードの有望さが推論できるため１度目の試行でノードを展開する(Fig. 2参照)．その後，今回のシミュレーションでたどってきたノードをルートノードまで戻りながら，各ノードの累計価値にNNから推論した価値を加算し，試行回数に1を加算する．ルートノードからリーフノードまでシミュレーションし，ノードの情報を更新しながらルートノードまで戻る処理を繰り返すことで探索を行う．

探索終了後，ルートノードの各子ノードの試行回数を確率分布に変換しこれを探索結果とする．(1)式に基づいてノードの選択をしているため，最終的には有望なノードほど試行回数が多くなっており，試行回数を確率分布に変換したものを方策とすることで，NNの方策より改善された方策を利用することができる．

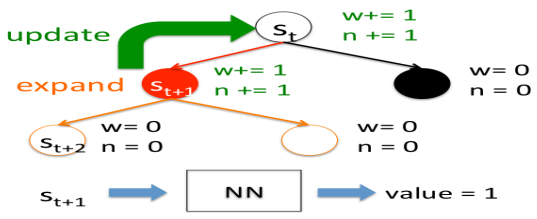

Fig. 2:Mcts process

2.2 MCTSを用いた格闘ゲームプレイヤ

吉田ら [6] は，ルールベース AI が予め決められた行動しか実行できず，対戦相手に対処されやすい問題を解決するため，MCTS を用いて行動選択を行う格闘ゲーム AI を提案した．MCTS はランダムシミュレーションにより行動決定するため，MCTS を適用することで現在の状況に応じた行動選択が可能になる．MCTS を格闘ゲームに適用する際の問題として，囲碁をはじめとしたターン制のゲームと比較して探索に利用できる時間が少なく，ゲーム終了状態(終端状態)まで探索することが困難であることが挙げられる．そのため，吉田らは探索する木の最大の深さを事前に定義し，現在の状態から移動できる深さを制限することで探索時間を削減した．

探索する深さを制限した場合，勝敗でノードの評価をできないという問題がある．勝敗でノードの評価をする場合には終端状態までシミュレートする必要があり，終端状態までシミュレートしなかった場合には評価することができない．そこで，吉田らは非終端状態を評価する(2)式を定義し，終端状態までシミュレートすることなく状態を評価可能にした．

$$(afterHP^{my} - beforeHP^{my}) - (afterHP^{opp} - beforeHP^{opp}) \quad (2)$$

ここで，$afterHP^{my}$はシミュレーション後の自キャラクタの体力，$beforeHP^{my}$はシミュレーション前の時キャラクタの体力，$afterHP^{opp}$はシミュレーション後の相手キャラクタの体力，$beforeHP^{opp}$はシミュレーション前の相手キャラクタの体力を表す．したがって，第１, 2 項はそれぞれシミュレーション前後の自キャラクタの体力差(被ダメージ量)，相手キャラクタの体力差(与ダメージ量)を表し，自分の被ダメージより与ダメージが多い場合，正の報酬が得られる．

2.3 OLSを用いた格闘ゲームプレイヤ

吉田ら [7] は，格闘ゲームにおいて探索木を再利用できない問題を解決するため，行動決定の際の木探索にOLS(Open Loop Search)を適用することを提案した．

格闘ゲームでは，自分の実行した行動が同じでも相手の行動によって次の状態が変化して一意に定まらず，探索木のノードが持つ状態と実際のゲーム状態が一致しない可能性があるため，ノードがゲーム状態を持つ場合，探索木は行動選択後に破棄される．OLS ではノードが状態を持たず，累計価値や試行回数といった情報のみを持たせることで探索木の再利用を可能にする．

また，一般的な MCTS では各ノードがゲーム状態を持つためルートノード以外からもシミュレートできるが，OLS はルートノード以外からシミュレートできず，毎回ルートノードから末端のノードまでシミュレートし状態が再計算される．これによって，各ノードが試行ごとに異なる状態を持つことができ，相手が取り得

る様々な行動についてシミュレーションすることが可能になる．そのため，２プレイヤが同時進行で行動決定しており，次の状態が自分の行動だけでは決まらない環境に適している．

探索木を再利用する際には，プレイヤが実行した行動に対応するノードを次のルートノードとして木を再利用する．その際，新たなルートノードと同じ深さにあるノードは到達できなくなったため破棄される(Fig. 3 参照)．探索木を再利用する際の問題点として，実際のゲームとは異なる相手の行動を選択したシミュレーションによる価値が，累計価値に含まれたまま引き継がれることが挙げられる．吉田らは探索木を引き継ぐ際に，試行回数がルートノードの試行回数と比較して一定割合以下であるノードは，シミュレーション回数が十分でなく信頼性が低いとしてノードの情報をリセットする処理を行った．

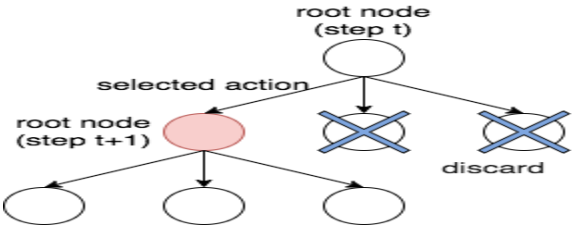

Fig. 3:Reuse of search trees

2.4 kNNを用いた対戦相手の行動予測

山本ら[8)]はルールベースのAIが同じ行動パターンを繰り返す問題を解決するため，対戦中に対戦相手の行動を記録し，記録したデータに対して kNN(k-Nearest Neighbor)を適用することで対戦相手の攻撃行動を予測し，それに応じて自分の次の行動を決定する AI を提案した．本節では，山本らの手法における対戦相手の行動予測部分について説明する．

対戦相手はそれぞれが異なった方策を持ち行動のパターンも異なるため，対戦相手の行動予測に利用するデータは対戦中にリアルタイムで収集される．対戦開始時から対戦相手が攻撃行動を実行する度，[相手が実行した行動，2 キャラクタ間の距離]を記録する．記録の際，データは各キャラクタがそれぞれ地上と空中のどちらにいるかによって，４つのデータセットに分けて保存される．

対戦相手の次の行動は相手の過去の行動履歴に基づいて予測され，まず相手が攻撃しようとしているかを予測する．現在の各キャラクタ位置が地上，空中のいずれかによって，４つのデータセットからあてはまるものを選択する．現在のキャラクタ間の距離(点 now)から距離 distThreshold の範囲内に，事前に決定した閾値 numAct 個以上のデータが存在する場合，現在と近い状況で相手が攻撃してきた記録が numAct 個以上存在するため相手が攻撃してくると予測し，numAct より少ない場合には攻撃してこないと予測する．攻撃してくると予測した場合には，対象のデータセットに対してkNN を適用し最も多い攻撃行動を抽出する(Fig. 4 参照)．Fig. 4 の例では，点 now から最も近い点を 5 個取り出すと，actionA が 1 個，actionB が 4 個含まれており，最も多い行動が actionB であるため，次の相手の行動が actionB であると予測する．

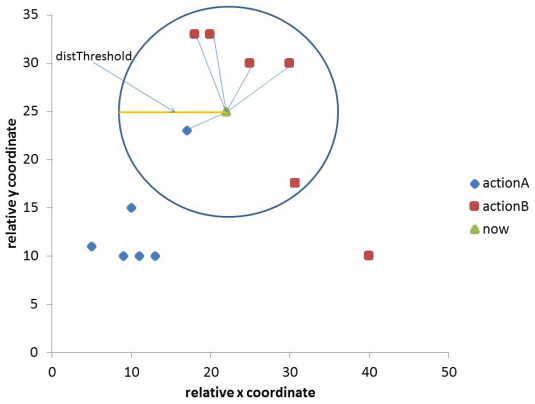

Fig. 4:Prediction of the opponent's next action with the k-nearest neighbor algorithm(k = 5).(from Yamamoto, Kaito, et al.[8)])

3 提案手法

提案手法では，AlphaZeroの手法を格闘ゲームに適用する際の，行動選択時間の長さとシミュレーションで得られるゲーム状態が実際のゲームに則していない可能性があるという問題に対応するためにNNとOLSを組み合わせた行動選択を行う．

OLSはシミュレートが毎回ルートノードから末端のノードまで行われ状態が再計算される．これにより自分と相手プレイヤが実行可能な多様な行動の組み合わせについてシミュレートできる．また，OLSにおいて探索する木の最大の深さを事前に決めておくことで探索時間を短縮し，自分と相手プレイヤが同時に行動決定するものとして探索することで実際のゲームに近いシミュレーションを行うようにした．

学習方法はAlphaZeroと同様に，自己対戦結果によって訓練データを作成し，作成した訓練データからNNのパラメータを学習することで性能を向上させる．NNの出力は２つに分岐しており，現在のゲーム状態を入力することで方策と価値を出力する．方策は行動の確率分布で各行動の有望さを表し，価値は勝敗予測で状態の有望さを表す．

提案手法での自己対戦の大まかな流れをFig. 5に示す．ゲームから現在のゲーム状態s_tを受け取り，OLSにより行動の確率分布π_tを求める．プレイヤ１はπ_tに従ってランダムに次の行動a_tを決定する．このとき並行してプレイヤ２も行動選択を行なっており，プレイヤ１と同様にして次の行動a_{t+1}を決定する．ゲーム終了までプレイさせた後，対戦結果と(3)式から評価値を計算する．

$$z = (HP^{p1} - HP^{p2}) / HP^{max} \quad (3)$$

ここで，HP^{p1}は対戦終了後のプレイヤ1の体力，HP^{p2}は対戦終了後のプレイヤ2の体力，HP^{max}は体力の最大値である．

自己対戦によって訓練データ[s_t，π_t，z]が作成され，NNの方策とπ_tの類似度が最大でかつ，価値とzの差が最小になるように，NNのパラメータが更新される．

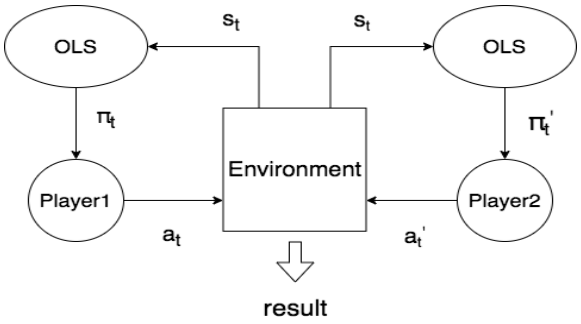

Fig. 5: Self-play of proposed method

提案手法の行動決定時の木探索アルゴリズムを以下に示す．初期の探索木は，ルートノードとルートノードから見て子にあたるノードのみを持つ．

1. (1)式に従って自分の行動，kNNで対戦相手の行動を決定する．
2. 処理1で決定した行動から，次のゲーム状態を計算し，選択した自分の行動に対応する子ノードに移動する．
3. リーフノードに到達するまで処理1, 2を繰り返す．
4. 現在いるリーフノードの深さD_{cur}が，探索する最大の木の深さD_{max}よりも小さい場合のみノードを展開する(Fig. 6 参照)．
5. 現在いるノードの価値をNNから求め，今回のシミュレーションで通過したノード全てに価値を加算し，試行回数に 1 を加算しながらルートノードまで戻る．
6. 探索の制限時間が来るまで処理1~5を繰り返す．

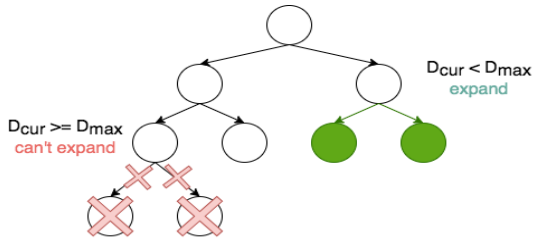

Fig. 6:Select action step4

4 実験

実験環境にはFightingICEを利用し，提案手法で学習を行ったAIと既存手法のAIを対戦させ，対戦結果から提案手法の性能を評価した．既存手法としてMctsAi, Machete, RandomAIを使用した．

対戦結果をFig. 7に示す．Scoreは以下の(4)式で計算される．

$$Score = HP^{my} - HP^{opp} \quad (4)$$

ここで，HP^{my}は対戦終了時の提案手法AIの体力，HP^{opp}は対戦終了時の対戦相手AIの体力を表す．RandomAI，Macheteに対しては僅かにスコアの向上が見られたが，MctsAiに対してはスコアの変化がほとんど見られなかった．

次に，対戦の際にOLSを行わずNNから出力された方策のうち値が最大の行動を選択するようにして対戦を行った．結果をFig. 8に示す．現在のゲーム状態での方策をNNから求め，それのみに従って行動決定を行っており，Fig. 8の対戦結果からNNの方策が学習できているかが判断できると考えられる．RandomAIに対しては僅かにスコアの向上が見られたが，Machete, MctsAiに対してはスコアの悪化が見られた．したがって，RandomAIに対して有効な方策の学習はできていたが，任意の対戦相手に有効な方策の学習はできていなかった．

Machete，MctsAiに対する対戦ではFig. 8でスコアが下がっているのに対して，Fig. 7ではスコアが上昇または横ばいになっている．提案手法では，NNで推論した価値と方策を用いたOLSによって行動選択しているため，価値は学習ができているのではないかと考えられる．自己対戦時の行動選択はOLSの結果である確率分布にしたがってランダムに選択しており，ある程度異なった行動が選択される．それによって，様々な対戦結果が得られた結果，価値は学習できたことが推測される．また，Macheteはルールベースの AIで，行動選択条件の１つが２キャラクタ間の距離であるため，OLSで利用しているkNNによる対戦相手の行動予測と相性が良かったことも考えられる．

任意の対戦相手に有効な方策が学習できなかった理由として，提案手法では自己対戦を行っており自分自身としか対戦していないため学習する方策が偏ったものになったことが考えられる．そのため，異なった方策を持つ対戦相手が用意可能なように手法を変更する必要がある.過去世代の個体を対戦相手に利用することや，複数の初期個体を用意し共進化的にそれぞれを学習させることで改善できる可能性がある．

また，Fig. 7, Fig. 8の各対戦相手に対するスコアの平均値がFig. 8の方が大きくなっていた(Table 1参照)．そのため，今以上に行動選択処理を高速化することで，スコアを向上させることができると考えられる．提案手法では，0.1秒間OLSを行っているため行動入力が0.1秒間隔になっており，対戦相手AIやFig. 8で使用したプレイヤは60分の1(約0.0167)秒間隔で行動入力を行える．行動の実行中などは行動入力できないため，毎回60分の1秒間隔で行動入力できる訳ではないが，提案手法AIよりは細かく行動入力できるので，行動選択の高速化が必要になる．

	RandomAI	MctsAi	Machete
Fig. 7	−33.8	−331.8	−338.6
Fig. 8	−7.1	−305.2	−241.1

Table 1:Average score

5 結論

本稿では，AlphaZeroの手法においてMCTSで探索する木の最大の深さを事前に決めておき，OLSを用いて行動決定することで，AlphaZeroのような自己対戦的手法を格闘ゲームに適用することを提案した．

ランダムに行動する個体に対しては性能の改善が確

認されたが，任意の対戦相手に対して有効であることは示せなかった．

学習時の対戦相手として異なった戦略を持つ対戦相手を用意できる手法に変更し，また，行動選択の処理を高速化する必要があると考えられる．

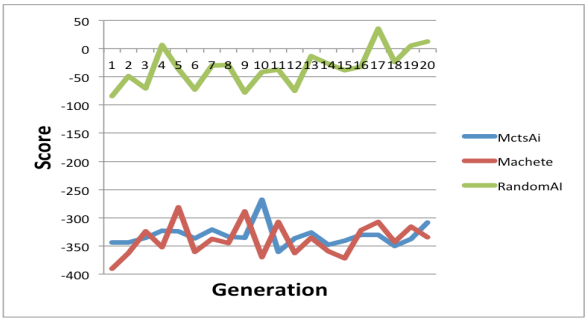

Fig. 7:Evaluation results

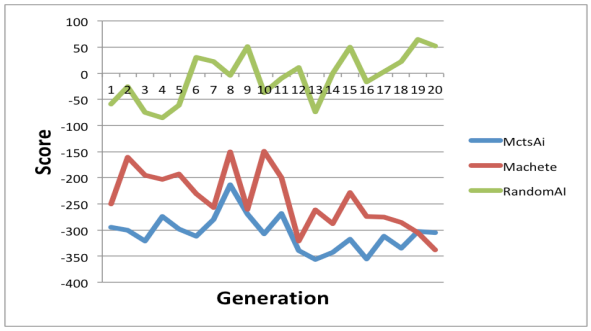

Fig. 8:Evaluation results without OLS

参考文献

1) Silver, D. et al.: Mastering the game of Go with deep neural networks and tree search, Nature 529, 484-489 (2016)
2) Silver, D. et al.: Mastering the game of Go without human knowledge, Nature 550, 354-374 (2017)
3) Silver, D. et al.: A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, Science 362, 1140-1144 (2018)
4) Perez Liebana, Diego, et al.: Open loop search for general video game playing, Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 337-344 (2015)
5) http://www.ice.ci.ritsumei.ac.jp/~ftgaic/
6) Yoshida, Shubu, et al.: Application of Monte-Carlo tree search in a fighting game AI, 2016 IEEE 5th Global Conference on Consumer Electronics. IEEE, 1-2 (2016)
7) 吉田修武, et al.: オープンループサーチを用いた対戦格闘ゲーム AI の提案, 2016 年度 情報処理学会関西支部支部大会 講演論文集 (2016)
8) Yamamoto, Kaito, et al.: Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator, 2014 IEEE conference on computational intelligence and games. IEEE, 1-5 (2014)