タワーディフェンスゲームにおける タワー配置探索の導入

野村 大輔 [1,a)]　秋岡 明香 [1,b)]

概要：タワーディフェンスゲーム（TD）で開発された AI は実世界上の様々な問題に応用が可能である．そこで，これまで TD でタワーの配置を最適化するプレイヤーAI の開発を行ってきた．しかし，開発したプレイヤーAI が学習したマップしか攻略できないという問題が存在する．そこで，本研究では新しいアプローチとして，TD のプレイヤーAI に評価関数と探索を導入する．評価関数は，ウェーブが攻略できるかを判断する．結果，評価関数はタワーの少ない盤面では有効性が確認されたが，タワーの数が増加した場合の評価関数には課題が残った．一方評価関数が，タワー配置がウェーブを攻略できるかどうかの情報しか持たないために，探索は評価関数に基づいたアルゴリズムで行ったが，人間を超える性能は得られなかった．

Incorporating Optimal Tower Placement Search into Player AI for Tower Defense Games

DAISUKE NOMURA [1,a)]　SAYAKA AKIOKA [1,b)]

Abstract: AI developed for tower defense (TD) games can be applied to a variety of real-world problems. Therefore, we have been developing a player AI that optimizes the placement of towers in TD. However, there is a problem that the developed player AI can only conquer the maps it has learned. Therefore, as a new approach, we introduce an evaluation function and search to TD's player AI. The evaluation function determines whether a wave can be conquered. As a result, we confirmed that the evaluation function is effective on a board with few towers, but the evaluation function remains problematic when the number of towers increases. On the other hand, since the evaluation function only provides information on whether the tower placement can attack the wave or not, the search was performed by an algorithm based on the evaluation function, but the performance did not exceed that of a human.

1. 背景

タワーディフェンスゲーム（TD）とは，決められた地点から自動的に出現する敵が，プレイヤーの拠点に向かって侵攻するゲームで，プレイヤーは敵が拠点に到達する前に，タワーを用いて敵を倒すゲームである．タワーは自動で敵を攻撃するオブジェクトであり，プレイヤーはゲーム中，コストと呼ばれる資金を用いてタワーの設置や強化を行うことができる．タワーはいくつかの種類が存在し，特定の敵に有効な場合や逆にほとんどダメージを与えられない敵も存在する．そのため，プレイヤーはマップの形状や出現する敵の種類に合わせて，持っているコストの範囲でタワーの配置や強化を戦略的に行う必要がある．

TD 攻略のために作成された AI は，TD の敵から拠点を守るというゲーム性から，都市の防衛モデルや，敵を車に，タワーを道路と見立てれば，交通管理モデルへの応用など，

1 明治大学大学院先端数理科学研究科
a) cs243020@meiji.ac.jp
b) akioka@meiji.ac.jp

様々なリアルタイムの実世界問題に応用が可能である．そのため，これまで Unity[2] 上に実際の TD ゲームを模したシミュレータを開発し，Unity のモジュールである ML-Agents[3] を用いてタワーの設置などを行うプレイヤーAI を開発した[1]．開発した AI は，学習に用いたステージでは一定の攻略性能を示したが，学習に用いていないステージでは，学習に用いたステージと難易度があまり変わらないにも関わらず，うまくステージを攻略することはできなかった．

一方で，ゲームの AI の代表例として，チェスや将棋などのボードゲームの AI が挙げられる．これらの AI は，盤面の状態の探索を行わない手法でも，一定の成果を上げ始めている．しかし，依然として探索を用いない AI の性能は探索ありの AI に及ばない．そこで，TD の AI 開発に探索を導入することで，さらなる性能の向上が見込める．しかし，現時点において，TD を対象としたプレイヤーAI の開発は行われていない．そこで，本研究では TD ゲームである Infinitode 2[4] を用い，探索を活用したタワーの配置と強化を最適化するプレイヤーAI の開発を目指す．しかし，TD に適した探索に用いる評価関数や探索手法は確立されていないため，まずは 3 種類のタワーと小規模なマップに制限し，評価関数を設計し，探索を行うプレイヤーAI を開発する．

2. Infinitode 2 のルール

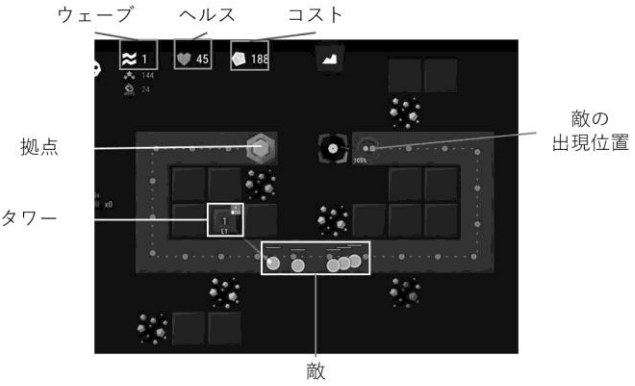

図 1　Infinitode 2 のゲーム画面

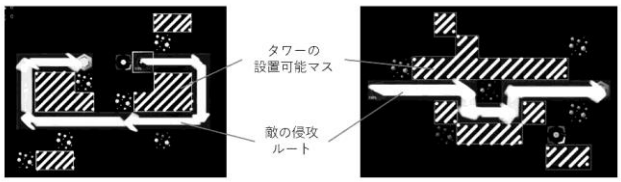

図 2　敵の侵攻ルートとタワー配置可能マス

Infinitode 2 は一般的な TD のルールを踏襲しており，図 1 は実際のゲームのプレイ画面である．また，図 2 はタワーの設置可能位置と敵の侵攻ルートを示した図であり，実際のゲームにはタワーの配置可能マスと，敵の侵攻する道マスから構成される．ゲームはウェーブ単位で進行する．ウェーブが始まると，敵の出現地点から敵が出現する．出現した敵は拠点に向かって，図 2 で示したようなルートで侵攻し，敵が拠点に到達するとプレイヤーはヘルスを失う．この時，ウェーブ中に出現する敵の種類や数，体力は表 1 のようにウェーブごとに全て決まっている．ウェーブが終了すると次のウェーブに進む．プレイヤーは，敵が拠点に到達することを防ぐために，タワーの設置や強化を，持っているコストの範囲で行うことができる．なお，コストは時間経過と敵の撃破で入手することができる．また，タワーの設置は，図 2 のようなタワーの設置可能マスの中から選択する．タワーの設置や強化は，本来ゲーム中任意のタイミングで行うことができる．しかし，本研究では簡単のために，タワーの設置や強化はウェーブ同士の合間に行うとする．加えて，実際のゲームでは，タワーは敵を倒すか，時間経過で経験値を入手し，少しずつ強化される．しかし，本研究では最初から，経験値を十分に与える．そのためタワーの性能は，プレイヤーがコストを使ってタワーを強化した場合のみ変化する．タワーと敵にはいくつか種類が存在し，それぞれ性能や相性が異なる．例えば，Cannon というタワーは広範囲にダメージを与えるので，まとまった敵に有効であり，一方で Strong という敵にはダメージをほとんど与えられない．また，Sniper というタワーは一回の攻撃に少し時間がかかるため，密度が低い敵に有効であり，さらに Strong に対して高いダメージを与えることができる，などである．プレイヤーは，ヘルスが 0 になるとゲームが終了するため，ヘルスが 0 にならないように，より長いウェーブを耐えきることを目的とする．

表 1　ステージ 1-2 の敵のステータス（例）

ウェーブ	敵	密度	体力	数
1	Regular	high	20	13
3	Regular	low	34	9
5	Strong	low	43	4
20	Regular	low	109	13

3. 提案手法

本研究では，ウェーブの攻略可否を評価関数に基づいて判断し，その評価に基づいて最も長くウェーブを攻略できるタワー配置を探索する．

3.1 1 つのタワーの性能の評価

TD においてウェーブの攻略可否を左右する主な要因は，盤面に置かれているタワーが一定時間内に撃破可能な敵の数が，同じ時間内に実際に流れてきている敵の数に満たない場合に失敗することだと考えている．そこで，あるウェーブにおいて，一つのタワーが一定時間内に撃破可能な敵の数の限界N_tを次のように定義する．

2 https://unity.com/
3 https://github.com/Unity-Technologies/ml-agents
4 https://infinitode.prineside.com/

$$N_t = \frac{\text{タワーの秒間攻撃回数}[/s] \times \text{タワーの攻撃範囲}}{\text{タワーが 1 体の敵を倒すのに必要な攻撃回数}} \quad (1)$$

このうち，タワーの秒間攻撃回数はゲーム内で与えられているため，その値を直接使用する．また，タワーが 1 体の敵を倒すのに必要な攻撃回数は，計算したいウェーブの敵の体力とタワーの攻撃力をもとに計算する．例えば，タワーの攻撃力が 350，敵の体力が 1500 であれば，1500 ÷ 350=4.28…の 4.2 を切り上げて 5 回として計算する．タワーの攻撃範囲は，タワーの攻撃範囲に含まれる，敵が進行するルートの長さで定義する．ただし，タワーの攻撃範囲は円形であるため，タワーの攻撃範囲内の敵の侵攻する道マスは，内側と外側で差が生まれる．そこで，本研究では，道の真ん中が含まれている長さを測定する．

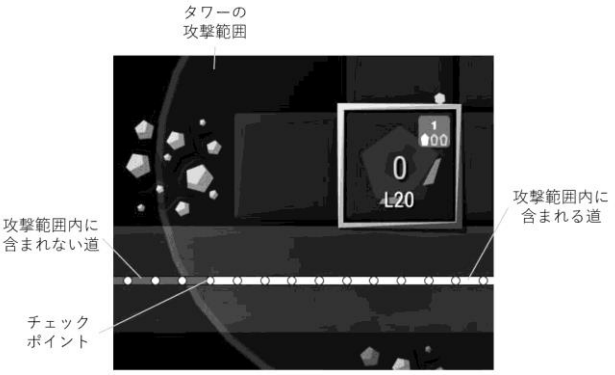

図 3　チェックポイントと攻撃範囲の概形図

図 3 は攻撃範囲内に含まれる敵の侵攻ルートの長さを測定するために利用した，チェックポイントとタワーの攻撃範囲の概形図である．白丸がチェックポイントであり，これを敵の侵攻する道マスの中心に置く．また，そのままでは 1 マス間隔のため，チェックポイント同士の間に 3 つチェックポイントを加える．そのため，チェックポイントは道の中心 0.25 マスごとに置かれる．隣接したチェックポイントが含まれている場合，その部分の敵の侵攻してくる道（区間）がタワーの攻撃範囲内に含まれているとする．この含まれる道の数から，タワーの攻撃範囲内に含まれる敵の侵攻ルートの長さを計算する．

3.2 各タワーの倒すべき数の決定

表 2 本研究で用いたN_dの値

タワー→敵	密度=high	密度=low
Basic→Regular	21.0	9.0
Basic→Strong	9.0	7.5
Sniper→Regular	25.0	11.5
Sniper→Strong	11.0	10.0
Cannon→Regular	5.0	4.0
Cannon→Strong	4.0	4.5

本研究では，敵の密度は実際のゲーム内で分類されているものと同様の high と low の二種類とし，同一の種類の敵において同種の密度であれば，ウェーブごとに求められる一定時間内に撃破すべき敵の数N_dは変化しないと仮定する．例えば，Basic が密度の高い Regular に攻撃する場合，ウェーブに関係なく，一定時間内に倒すべき敵の数（Basic に求められるタワーの一定時間内に倒せる数）が常に同じとする．よって，式 1 に基づいてN_dを，3 種類のタワーと二種類の敵とそれぞれの密度の組み合わせで全て求める．ただし，実際の一定時間内に倒すべき数N_dを直接計算するのは難しいため，タワーを 1 つ置いた際に，失敗したウェーブでのN_tとして計算された一定時間内に倒せる敵の数の最大値に基づいて求める．また，この際にタワーの置く位置は，タワーの攻撃範囲内に含まれる敵の侵攻ルートが十分長い地点を選択している．具体的には，ステージ 1-2 の，マップ内の（x, y）＝（5, 4）の地点に，Basic，Sniper，Cannon のタワーをそれぞれ置いた時に，レベル 6～10 のそれぞれで，Regular，Strong の high／low の時に，最初に失敗したウェーブで式 1 に従ってN_tを測定し，レベル間で比較し一番高かったN_tを基準にしている．表 2 は，本研究において用いたN_dである．すなわち，Basic が高密度の Regular のウェーブを攻略する際に，$N_t > N_d(= 21.0)$であれば，そのウェーブの敵を倒すのに十分な制圧力があるといえる．

3.3 タワーが複数の場合への拡張

タワーごとに攻撃方法や弾速が異なり，また Cannon は着弾点の広範囲にダメージを与えるタイプのタワーである．そのため前項で求めたN_dはタワーによって異なる．従って，異なる種類のタワーが複数同時に設置されている場合，単純にN_tを足し合わせることができない．そこで，各タワーのN_tを対応するN_dで割ることにより正規化を行う．正規化を行うことで，N_t / N_d が 1 を超えるかどうかでそのウェーブの攻略が可能か否かを判断できる．

以上の議論を踏まえ，本研究では盤面上に存在するすべてのタワーにおいて，N_t / N_dを足し合わせる．和が 1 を超えると，現在のタワー配置でウェーブが攻略可能と判断する．

3.4 探索

本研究での評価関数は，あくまでそのウェーブが攻略できるかどうかの情報しか持たず，敵の種類に依存する．そのため，探索は単純に評価が最も高いものを選択すればよいわけではない．実際に，敵が Regular のウェーブで評価が高いからと言ってそのウェーブが次以降の Strong のウェーブでも評価が高いとは限らない．一方，評価関数に基づいて，ゲームの序盤で最終盤の予測をするには，評価関数に加え，得られるコストの量の予測が必要となり計算量も計り知れない．そこで本研究では，特定の方策に基づいて評価関数のみで探索を行う．具体的には，現在所持しているコストの範囲で，最も長くウェーブを耐えられるタワー配置を，評価関数に基づいて探索し，その配置を採用する．その後，実際にゲームを行い，そのゲーム内で失敗すると評価関数に予想されたウェーブに到達するか，もしくは，ウェーブの攻略を

失敗したタイミングでゲームを一度停止する．その後，停止した時点でのタワー配置と所持しているコストから改めて次の最も長く耐えられるタワーの配置を探索し，その配置でゲームを再開する．これを繰り返すことによって，実行可能な範囲での，長いウェーブを攻略できるタワーの配置を得る．

4. 実験

本研究では，評価関数の最もやってはいけないこととして，成功すると予測されたウェーブが，実際には失敗してしまうこととする．そこで，定義した評価関数が，最初に失敗すると予測したウェーブが，実際にゲームをプレイした際の最初に失敗したウェーブよりも遅いとき，評価関数の予測を失敗したとし，その失敗した回数を評価する．また、提案した探索方法で経由するタワー配置と最終到達ウェーブを調べる．

4.1 評価関数の評価

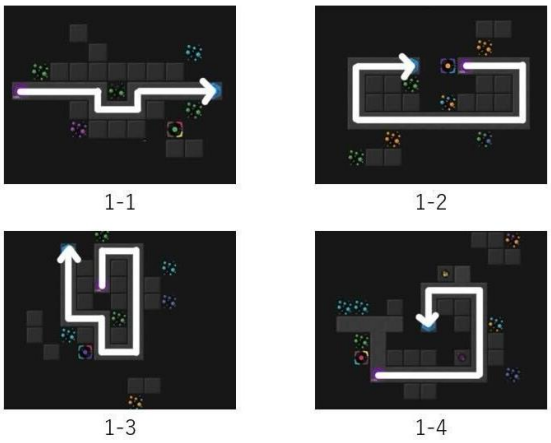

図 4　敵の侵攻ルートとタワー配置可能マス

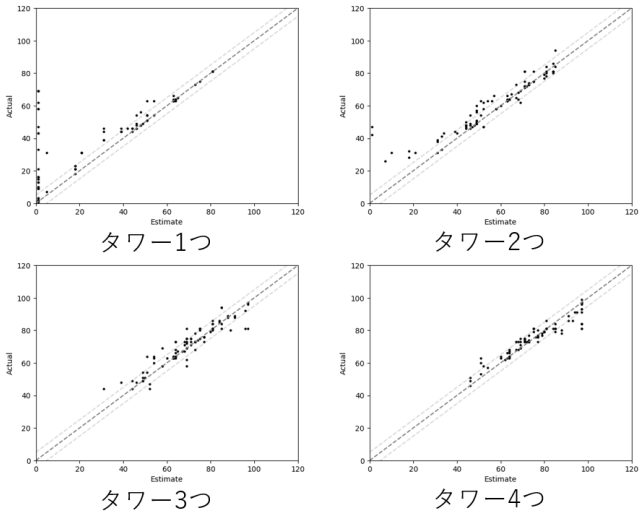

図 5　1-2 における評価関数と実際の結果の比較

表 2　ステージと失敗回数の関係

ステージ	設置したタワーの数					
	1	2	3	4	>5	平均
1-1	2	13	37	40	40	26.4
1-2	1	14	20	32	48	23.0
1-3	5	18	20	38	42	24.6
1-4	0	14	30	24	38	21.2

1-1 から 1-4 までの 4 つのステージにおいて，タワーを 1，2，3，4，>5 個のタワーを設置し，それぞれランダムにタワーを強化した状態で 100 ゲームずつ実際にゲームを行った際に，評価関数の予測を失敗した回数を表 2 に示した．また図 4 は，評価に用いたステージの形状と，敵の侵攻ルートを示した図である．さらに，図 5 はステージ 1-2 において，評価関数が予測した最初に失敗するウェーブ（Estimate:横軸）と，実際のゲームで最初に失敗したウェーブ（Actual:縦軸）について，設置したタワーの数が 1 つから 4 つ置いた場合で比較した図である．また，破線は Estimate = Actual±5 および Estimate＝Actual を示している．すなわち，濃い破線上を含むその破線より上の領域上の点が，評価関数の予測を成功したタワーの配置を示す．

表 2 の結果から，ステージ間で大きな性能差が見られなかったため，本研究で定義した評価関数は，特定のステージに限定されず，様々なステージで利用できることができる可能性が高いことが示された．一方，問題点としてタワーの数が増加するにつれて評価関数の精度が低下している点が挙げられる．

図 5 において，タワーが増えるに従って，破線より下の点が増えている．原因としては二つ考えられる．一つ目の原因は，タワー同士の攻撃範囲が重なっている場合，敵を攻撃する際にタワーの発射した玉が無駄になってしまうことである．これは，タワーが敵に玉を打った際に，敵に着弾するまでに時間がかかり，その間に他のタワーの攻撃によって狙った敵が倒されると，その玉が無駄になってしまうために，敵を倒すために必要な玉が増え，結果的に評価関数の精度が下がる．そのため，タワーの数が増えれば増えるほど，攻撃範囲の重なっている範囲も増加し，評価関数の精度も下がる．この問題の解決方法として，タワーの攻撃範囲が被っている際に，ペナルティを加える必要がある．ただし，単純にタワーの数だけを考えればよいわけではなく，攻撃範囲のうち，タワー間で被っている範囲の形状やタワーの種類も考慮しなければならない．そのため，この点は今後の大きな研究課題の一つである．

もう一方の問題は，今回の評価関数では二種類で常に固定とした敵の密度が，実際にはゲームが進行しウェーブが増加するにつれて，敵の密度も少しずつ増加している可能性がある．この場合，評価関数で与えた，タワーの撃破すべき数が前半では少なく，後半では多くなる．そのため，評価関数が前半では実際に比べて厳しく，逆に後半では緩くな

ってしまう．図 5 のタワーを 1 つしか置いていない場合を見ると，図の左側に点がいくつか固まっている．これは，評価関数では最初のウェーブすら攻略できないと予測したのにも関わらず，実際にはもっと先のウェーブまで攻略できていることを示す．これは，序盤のウェーブの敵の密度が低く，実際の倒すべき敵の数N_dが小さいことの根拠となる．一方で，図 5 でタワーの数が増えるたびに，全体的にグラフの各点が右上に寄る．これは，予測した最初に失敗するウェーブも，実際に失敗した最初のウェーブも，タワーが増えるたびに増加することを示す．タワーの数が増加すれば，次第にそのタワーの数の範囲で実行可能なタワー配置の，最初に失敗するウェーブも全体的に増加する．そのため，タワーの数が増えた際，ゲーム終盤の敵の密度が上がった状態で，実際には倒すべき敵の数N_dが増加しているために，評価関数の値が実際よりも小さく出てしまうことで，失敗しているとも考えられる．これを改善するためには，今後倒すべきN_dをウェーブ数に合わせて動的に変化させる，もしくは敵が出現したタイミングで密度を測定し，N_dを動的に調整する必要がある．しかし，実際に動的に調整する場合，ステージ間で大きく変わってしまう可能性がある．そのため，今後はそれぞれのステージで動的なN_dを開発し，それをステージ間で共有できるようにすることも，今後の研究課題である．

また，タワー攻撃範囲に含まれる敵の侵攻ルートが非常に短く，かつタワーのレベルが高いときにも，失敗するウェーブの予測を失敗することがある．これは，タワーが敵を認識して，攻撃を行い，着弾するまでの一連の時間に，敵が攻撃範囲外に出てしまったり，拠点に到達してしまうことが原因である．この場合，評価関数ではタワーのレベルが高いために，攻撃範囲と攻撃速度によってウェーブを攻略できると計算するが，実際には攻撃できない場合であるために発生する．このため，評価関数ではタワーの攻撃範囲と攻撃できる回数を比例すると仮定したが，実際にはタワーの攻撃範囲から一定の値を引いた量が，攻撃できる回数と比例すると考えられる．そのため，今後はこの量を測定することも必要である．

4.2 探索の評価

評価関数の評価で用いた 4 ステージを探索する．ただし，本研究での評価関数はタワーの数が増えるたびに性能が下がってしまっている．また，実際に計算量もタワーの配置数が増えるたびに爆発的に増加する．そこで，本稿では，探索する際に，タワーの追加配置を 1 つまでと制限する．すなわち，ゲームが始まって何もない状態で探索した場合は，タワーが一つの，初期に持っているコスト内で実現可能なすべての盤面のうち，最も遠くのウェーブまで到達できるタワーの配置を採用する．一方すでに，タワーが 3 つ置かれている場合は，タワーが 4 つ以下の範囲で持っているコストの範囲内で，現在のタワーの配置から実現可能なタワー配置のうち，最も長いウェーブを攻略できるタワー配置を採用することにする．また，最も長いウェーブが攻略できると予想したタワー配置が複数存在する場合，失敗すると予想したウェーブの評価関数の値の最大値で決定する．それでも複数ある場合は，探索の際に先に発見されたタワー配置を採用する．

表 3 から表 6 は 1-1 から 1-4 までの 4 ステージにおいて，探索した結果である．表中の操作は，タワーを新しく設置した場合+，既存のタワーを強化した場合↑の記号を加えている．また，記号の右の 4 つの数字はそれぞれタワーの位置（x, y），タワーの種類（0 : Basic, 1 : Sniper, 2 : Cannon），タワーのレベルを表す．例えば，表 3 の行動 3 は，すでに（3,6）に置かれている Sniper のタワーをレベル 8 まで強化し，さらに新しく（4,4）に Basic を置き，レベル 10 まで強化することを示す．また，到達予測は，操作を行った後のタワー配置に対して，評価関数が予測した，クリアできる限界のウェーブである．表中の結果は，予想されたウェーブまでに，ダメージを受けたかどうかを示す．例として，表 3 の行動 3 は，予想到達が 89 に対して，実際は 88 である．これは，88 ウェーブ目で拠点に敵が到達することでダメージを受け，ウェーブの攻略を失敗したことを示す．逆に，表 4 の行動 2 では，ダメージを受けずに予想した 78 ウェーブまで到達できたことを示している．例外として，表 4 の行動 8 以降では，探索内で攻略できると予想できたタワー配置が存在しなかったが，実際のゲーム中では先のウェーブまで攻略できている，そのため，到達予測よりも高い値が表中の結果に記入されている．実際の表中次の残体力は，実際に示されているウェーブが終了した際に残っていたプレイヤーのヘルスである，したがって，表 5 の行動 2 後の配置は，81 ウェーブ目に 1 体拠点に到達し，ウェーブの攻略を失敗するという意味である．また，所持コストは結果のウェーブが終了した時点での所持コストである．したがって，次の行動はこの値を用いて探索される．

まず，表 3 から表 6 の 4 ステージにおいて，共通する点は，最初は Basic のタワーをコスト目いっぱいで強化して置く．さらに，次に置かれるタワーは Sniper である．これは，Basic が序盤のウェーブで有効とされており，Sniper は Basic に比べ，Strong に対してとても有効であるためだと考えられる．

表 3　1-1 での探索結果

行動	操作	到達予測	結果	残体力	所持コスト
1	+ [3,4,0,6]	42	42	45	3689
2	↑ [3,4,0,10] + [3,6,1,5]	77	77	33	6459
3	↑ [3,6,1,8] + [4,4,0,10]	89	88	31	3742
4	+ [5,4,0,10]	95	93	21	1768
5	+ [4,5,1,6]	97	94	8	771
6	+ [6,4,1,5]	100	100	8	1873
7	↑ [3,6,1,9] + [4,1,1,2]	105	101	0	

表 4　1-2 での探索結果

行動	操作	到達予測	結果	残体力	所持コスト
1	+ [5,4,0,6]	42	42	45	3614
2	↑ [5,4,0,10] + [3,1,1,5]	78	78	45	6797
3	↑ [3,1,1,8] + [4,1,1,9]	92	92	45	3545
4	+ [8,4,2,8]	94	94	45	1592
5	+ [9,4,0,8]	100	100	45	2382
6	↑ [3,1,1,9] + [4,4,1,5]	106	104	38	1275
7	+ [4,5,0,8]	106	106	35	764
8	+ [9,5,0,7]	106	112	34	1754
9	↑ [9,5,0,8] + [10,4,0,8]	106	113	33	639
10	+ [10,5,0,7]	106	114	31	739
11	+ [9,8,0,7]	106	116	26	814
12	+ [10,8,0,7]	110	117	12	412
13	+ [3,5,1,4]	110	118	5	322
14	+ [3,4,0,5]	110	119	1	317
15	↑ [3,4,0,6]	110	120	0	

また，表 7 は提案手法に基づいて，ゲーム内での最終到達ウェーブ（ヘルスが 0 になった時点のウェーブ）と，実際に執筆者が同じ条件でゲームを行った結果の比較である．この表 7 によれば，1-2 のみが，提案手法が執筆者よりも高い攻略性能を示している．一方で 1-1，1-3 は悪く，1-4 も最終的な倒せた敵の数はわずかに執筆者が上回った．これは 1-1 や 1-3，1-4 で Cannon を使っておらず，Cannon の性能が過小評価されていることが原因と考えている．一方 1-2 が成功した理由もその Cannon が使えていることだと考えている．また，提案手法の後半は，タワーを低いレベルでもできる限り置くようになってしまっている．そのため，後半にタワーの数が増えて評価関数の精度も悪くなってしまっていることも問題である．今後は，探索方法に改善を加えることを研究課題としたい．

表 5　1-3 での探索結果

行動	操作	到達予測	結果	残体力	所持コスト
1	+ [5,6,0,6]	52	52	45	5231
2	↑ [5,6,0,10] + [3,8,1,7]	92	81	44	5997
3	↑ [3,8,1,9] + [5,7,0,10]	104	99	43	5180
4	↑ [3,8,1,10] + [3,6,0,9]	111	101	38	1122
5	+ [5,8,1,5]	114	104	37	1231
6	+ [3,7,1,6]	115	109	32	1831
7	↑ [3,6,0,10] + [7,7,1,4]	117	112	25	1144
8	+ [3,4,0,8]	117	114	22	795
9	+ [5,4,0,7]	117	115	18	528
10	+ [7,6,1,4]	121	116	12	453
11	+ [0,5,1,4]	122	118	0	

表 6　1-4 での探索結果

行動	操作	到達予測	結果	残体力	所持コスト
1	+ [7,4,0,5]	48	48	45	4711
2	↑ [7,4,0,10] + [4,0,1,6]	81	78	43	6159
3	↑ [4,0,1,9] + [5,2,0,9]	94	90	38	4151
4	↑ [5,2,0,10] + [7,2,0,9]	101	95	35	2435
5	↑ [7,2,0,10] + [7,5,0,8]	104	99	34	1362
6	+ [5,0,1,6]	106	104	26	1587
7	↑ [7,5,0,9] + [6,5,0,7]	106	105	20	468
8	+ [3,2,1,4]	108	106	17	417
9	+ [4,2,1,4]	108	107	10	275
10	+ [3,5,1,3]	108	108	0	

表 7　結果の比較

ステージ	提案手法	執筆者
1-1	101	106
1-2	120	107
1-3	118	124
1-4	108	108

5. まとめ

本研究では，TD におけるタワーの配置と強化を行うプレイヤーAI を評価関数と探索を用いて実装した．実装した評価関数はタワーの少ないときはうまく機能したが，タワーが増えると精度が下がってしまったため，今後はタワーが増えたときの評価関数の改善を行う．また，探索に関しては，まだ，人間の能力に追いついているとは言えず，探索方法を改善する必要がある．

6. おわりに

本研究では，評価関数はタワーがウェーブを攻略できるかどうかを判断することを目的にしたが，完全に予測できるようになれば，TD の AI の性能は飛躍的に向上すると考えている．これは，ウェーブの攻略可否が分かれば，シミュレータを経由する必要が無く，たくさんのタワー配置での結果データを集めることができる．今後も，評価関数の改善を中心に多角的に TD のプレイヤーAI 開発を進めていきたいと思う．

参考文献

[1] 野村大輔，秋岡明香：実際のタワーディフェンスゲームを模したシミュレータでのステージ AI 開発，第 16 回データ工学と情報マネジメントに関するフォーラム予稿集，2024