AI 鬼ごっこサンドボックス

ロベルト・アンドレ・モッシ・ミラ
コンピューターサイエンスの大学生, ギャノン大学
ラマクリシュナん・スンダラム教授
電気・サイバー工学を専門とする教授, ギャノン大学

要旨

本論文では、異なる主体間の競争および鬼ごっこの生存要素を通じて、AI 鬼ごっこサンドボックスを構築する。ゲノムを用いることで、生存を目的として自己協調する高度なエンティティを生成する。ゲノムを基盤とした強化学習による複数回の試行を通じて、最も適応度の高い生存遺伝子および自己発展戦略を探索する。さらに、ランダムに生成されたより複雑な環境を導入することで、推論能力の発達を促し、より人間らしい技能の獲得を目指す。本研究と類似する取り組みは、OpenAI による Hide & Seek などのゲームにおいて既に行われているが、本研究ではより人間的な行動を導入し、生物学的観点からその創意性を評価するために ゲノムを活用している。本研究は、コンピュータサイエンス（CS）学科の学生に対し、人工知能の開発および生物学などの他分野が技術発展にどのように寄与するかを示す ことで、教育カリキュラムの発展にも貢献する。タスク、ランダム初期化、制約条件に基づく意思決定と、生物の生体構造との関連性を比較分析し、その行動に基づいて、ゲノムを用いた人間行動の再現に向けた研究 をさらに発展させるためのファインチューニング手法を提案する。

序論

ゲノムを使用することで、被験体の行動はそれぞれの遺伝子によって決定される。遺伝子は最初にランダマイザーによって決定され、これはエンティティに対して、彼らが達成する必要のある生存タスクを学習し、適応するための出発点を与えるために行われる。そこからタスクが与えられ、本研究では二つのグループがランダムに作成され、それらのグループはランナーとシーカーとして構成される。

[1]ASEE 掲載論文（公開先）: https://peer.asee.org/54692
[2]ペンシルベニア州立大学（Sigma Xi）研究発表会にて発表 (https://sites.psu.edu/behrendsigmaxi2025/2025/03/11/tag-ai-sandbox/

これらのグループに与えられるタスクは、シーカーがランナーを捕まえることであり、ハイダーは限られた時間内に彼らから逃げなければならない。ターゲットを捕まえたシーカーは生存することができるが、捕まえられなかった者はその場で死ぬ。対抗チームについても同じ原理が適用されるが、捕まえられないようにするという目的を持って行動する。

本プロジェクト全体を開発するにあたり、Unreal Engine を使用してプロジェクトを開発した。Unreal Engine を使用した理由は、人間のような行動の相互作用には物理演算が必要であると 考えたためである。多くの選択肢が存在したが、Unreal Engine は市場で最も人気のあるものの一つであり、使用言語が C++ であるため、より複雑であることでも知られている。私はこれを挑戦として捉え、可能な限り高いパフォーマンスを得ることを目指した。

これを可能にするために、いくつかのライブラリの支援を受けている。その一つが PyTorch であり、人工知能の開発に使用されている。もう一つのライブラリは CUDA であり、その機能はプロジェクトからグラフィックス処理をグラフィックスカードへ転送し、行動に関する処理についてはコンピュータの残りの部分に任せることである。

関連研究

これまで、生物の研究に焦点を当てた人工知能を作成する試みは数多く行われてきた。本研究の具体的な状況においては、二つの特定のプロジェクトから着想を得ている。一つ目のプロジェクトは、OpenAI によって開発された *Multi-Agent Hide and Seek* [2] である。このプロジェクトは、問題解決における AI の創意工夫を研究するために、ゲームを基盤とした環境を作成するという発想を与えてくれた。二つ目のプロジェクトは、動画 *I programmed some creatures* [3] である。このプロジェクトは、環境の制約を通じてゲノムを用いて生物の行動を管理するという 考え方を主な目的としている。これにより、コンピュータサイエンスと生物学の世界を結び付ける関係性が示されている。

必要要件

種類	バージョン	名称	説明	採用理由
言語	C++20	C++	Unreal Engine で使用される低水準のオブジェクト指向言語	AI プログラミングにおいて広く利用されており、高速な処理性能を持つため。
言語	3.31.6	CMake	各 OS 間での自動化を行うためのビルド記述言語	複数のリポジトリを使用しているセため、ットアップの自動化とクロスプラットフォーム対応を実現するため。
ライブラリ	12.0.0	CUDA	GPU 計算のためのプログラミングモデル	高度なグラフィック処理を実現するため、GPU 計算を用いて処理速度を向上させるため。
グラフィクス	5.4.4	Unreal	エンティティの視覚的表現を生成するためのゲームエンジン	複雑な計算処理を扱うことができ、高いパフォーマンスを発揮できるため。

世界の開発

このケースでは、エンティティが移動できる世界を必要とすることは明白であった。それを踏まえ、当初はエンティティが自由に動き回り、自律的にゲームの遊び方を学習できる 汎用的で静的な世界を作成することを考えていた。しかし、ある時点で彼らはその世界内での ゲームプレイに非常に長けるようになる一方で、自ら学習した世界以外では成長の余地が なくなる可能性があることに気づいた。この問題を解決するためには、彼らが新たな状況において自身の学習結果を適用し、意思決定能力をさらに向上させられるよう、新しい世界を生成する必要があると考えた。しかし、そのたびに新しい世界を手作業でモデリングする必要があるということは、長期的に見て多くの時間を要し、理想的な新世界の設計に時間を費やしたくなかった。そのため、本研究ではワールドジェネレーターを作成することに決定した [1]。この判断は、OpenAI の Hide & Seek のような長期的プロジェクトでは非常に自然な選択であるが、小規模なプロジェクトでは依然として多くの場合、手作業で異なる世界をモデリングする傾向がある。しかし、冗長性を排除するという観点からも、ランダムなワールドジェネレーターは有用である。

この世界を作成するにあたり、まず使用するアルゴリズムの検討から始めた。Unreal Engine 内外において、独自のジェネレーターを作成する事例に関する多くの動画や ドキュメントを調査した。その多くで使用されていたのが A* アルゴリズムである。このアルゴリズムは主に自動生成において有用であり、その基本的な考え方は、全ての隣接ノードを考慮するダイクストラ法を基盤とし、終点までの最短経路を探索すると同時に、現在のノードから最終ノードまでの距離も考慮する点にある [4]。これは開始点と終了点が明確に定められている状況において特に有効である [5]。

しかし本研究では、終点を探索するのではなく、配置する部屋の総数のみを目的とした 幅優先探索アルゴリズムに近い手法を採用することにした [6]。この判断の理由として、より複雑なアルゴリズムを用いる利点は、特定の目的地へ到達させる必要がある場合に限られるためである。本研究においてエンティティの目的は、特定の場所へ向かうことではなく、互いに追跡・回避し、捕獲されることを防ぐ行動にある。そのため終点を設定する必要がなく、アルゴリズムの選択肢を絞ることができた。最終的には、よりランダム性を重視した手法を選択した。

このワールド生成手法は以下のように動作する。ワールド生成が開始されると、まず「origin」と呼ばれる一つの開始点を設定する。生成処理が実行されると、origin からランダムな部屋が一つ生成される。各部屋には接続用の出口として矢印を配置し、Exit Arrow フォルダ内に格納する。これらの矢印は、部屋を生成したい角へ移動させ、生成方向を示す向きに回転させる必要がある。これらの出口の中からジェネレーターがランダムに一つを選択し、次の接続された部屋を生成する。この処理を、設定された部屋の総数が生成されるまで繰り返す [7]。

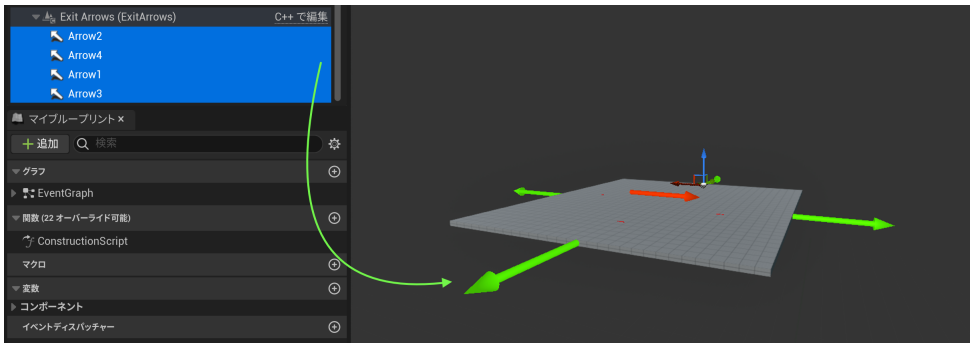

図 1: 部屋設定

[1] コードについては、以下のリポジトリを参照されたい：https://github.com/AndreM222/Procedural-Generator

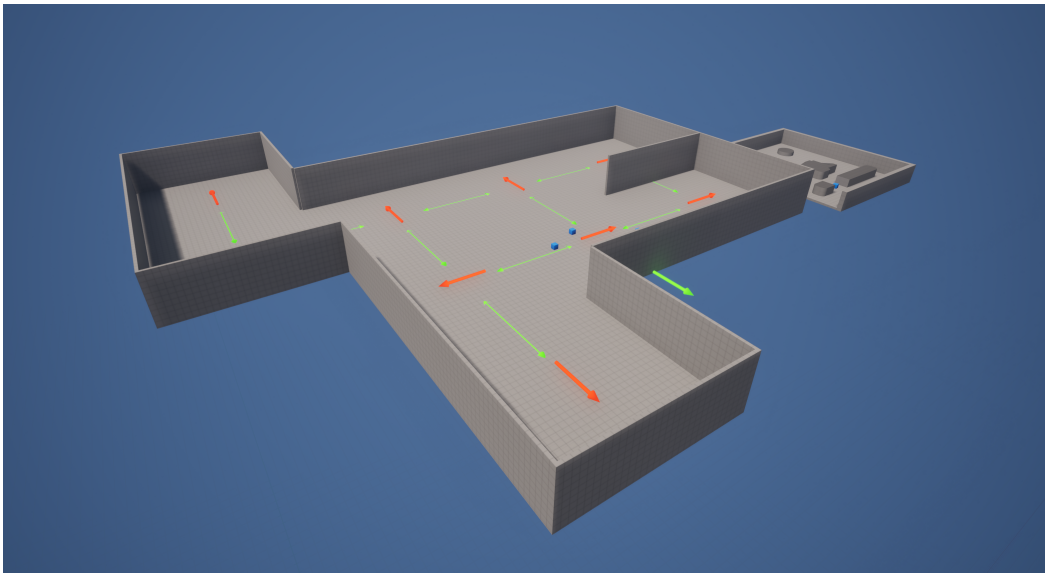

図 2: 世界背性のプレビュー

衝突検出

では、利用可能な出口が一つも残っていない場合はどうなるのだろうか。この疑問を踏まえ、私は二つの追加関数を作成した。それが reboot 関数と delete room 関数である。reboot 関数の役割は、生成されたすべての部屋に対して delete room 関数を実行することであり、すべての部屋が削除された後、現在のジェネレーター自身を削除する delete 関数を実行する。この delete 関数は、残っているすべての命令が実行されるのを待ってから、自身を削除するようになっている。その後、最後の命令として新しいワールドジェネレーターを生成し、与えられた要件に基づいてランダムに新しい世界を生成する処理を最初から再開する。

　では、異なる部屋同士の衝突はどうなるのだろうか。重なってしまうことはないのか。この問題に対応するため、さまざまな状況における衝突を処理する複数の関数を作成した。これらの状況は、大きく二つの異なるカテゴリに分類できる。

- **衝突判定:** 一般的な衝突判定においては、生成しようとしている空間に 何かが存在するかどうかを確認するだけでよい。そのため、この関数では不可視のトレース 用ボックスを生成し、それが何かと衝突した場合には、新しい生成位置を探すよう指示する。

スイープ衝突判定 cpp

```cpp
// Setup Box Dimensions for trace
FVector BoxDimensions = FVector(50, 50, 50);

FVector traceLocation = exitLocation.GetLocation();

// Trace box to check if collision available
FCollisionShape box = FCollisionShape::MakeBox(BoxDimensions);
exitsList[spaceIndex]->GetWorld()->SweepSingleByChannel(
    Hit,
    traceLocation,
    traceLocation,
    FQuat::Identity,
    ECC_Visibility,
    box,
```

```
    QueryParams
);

// Return true if room type has been found connected
if (Cast<AMaster_Room>(Hit.GetActor())) return true;
```

- **寸法スポーナー:** 部屋を生成しようとする際、その部屋の寸法に基づいて、十分な空間が存在するかを確認する必要がある。しかし、現在の Unreal Engine のバージョンでは、部屋の寸法を事前に計算することは不可能である。そのため、本研究ではやや強引な解決策を採用した。具体的には、目的の部屋を一瞬だけ生成してその寸法を取得し、それを変換用の変数に保存した後、直ちにその部屋を削除する方法である。この処理は非常に高速に行われるため、生成された部屋が 画面上に表示されることはない。その後、一般的な衝突判定と組み合わせて、新しく計算された寸法を 用い、トレースボックスのサイズを置き換えることで、より正確な衝突判定を実現している。

寸法スポーナー cpp

```cpp
// Make temporary room
AActor* roomModel = GetWorld()->SpawnActor<AMaster_Room>(
    RoomToSpawn[roomIndex],
    exitLocation.GetLocation(),
    FRotator::ZeroRotator,
    SpawnParams
);

// Setup Box Dimensions for trace
FVector BoxDimensions = FVector(roomModel->GetComponentsBoundingBox().GetExtent() - 4);
BoxDimensions.Z += 3;
roomModel->Destroy(); // Destroy tmp room after setup

FVector traceLocation = exitLocation.GetLocation() + (
    exitLocation.GetLocation() - actorLocation.GetLocation());
traceLocation.Z += BoxDimensions.Z; // Center Box Trace

// Trace box to check if collision available
FCollisionShape box = FCollisionShape::MakeBox(BoxDimensions);
```

世界のカプセル化

このプロジェクトで作成する世界では、エンティティに対していくつかの制限を加える必要がある。これらの制限は、エンティティがマップの範囲外へ歩き出したり、虚無空間へ落下したりすることを 防ぐためのものである。本プロジェクトの主目的は、落下位置を学習させることではなく、複数の エンティティ間の相互作用にどのように反応するかを学習させることにある。そのため、生成終了時に 空いたままとなった出口は、壁によって封鎖することにした。

壁を閉じる際には、他の部屋と接続している場所に壁が生成されないようにする必要がある。もしこれを行わなければ、単なる迷路を作成するのと変わらなくなってしまう。ここで、本研究における「世界」と「迷路」の違いを明確にする。一般的に迷路とは制約が多く、壁に囲まれた通路、すなわちコリドー によって移動が制限される構造である。一方、本研究における世界は、コリドーと開けた空間の両方から 構成されており、より自由度の高い構造となっている。

壁が無差別に生成されることを防ぐため、小型のトレースボックスを使用している。このトレースボックスは、壁を配置しようとする位置に接続された部屋が存在するかどうかを判定するために用いられる。もし接続された 部屋が存在する場合、その出口は開放出口リストから削除され、次の出口の処理へと移行する。すべての出

口に 対して部屋を生成できない場合、その部屋は生成候補から削除され、次の部屋へと処理が進む。この処理は、利用可能な出口および部屋がなくなるまで繰り返される。

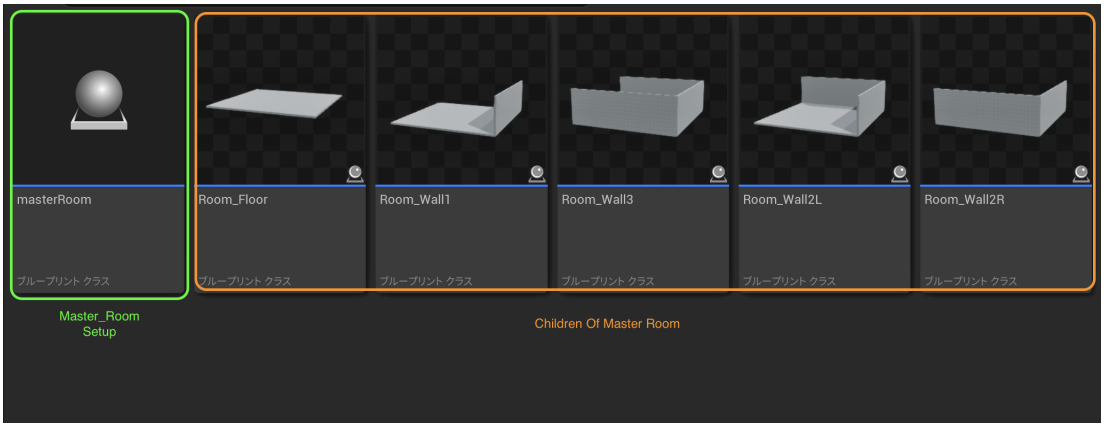

図 3: 部屋の生成

シード値

ある時点で、エンティティを特定の部屋に一定期間留めておきたいという要件が生じた。そのため、シード値を指定できる機能を追加した[2]。シードを使用すると、その数値に応じて生成される部屋構成が決定され、何度再読み込みを行っても常に同一の 世界が生成される。

しかし、シード値を使用することには問題も存在する。指定された総部屋数をすべて生成できない可能性がある点である。これは、部屋が特定の経路に沿って生成され、最終的に自分自身の経路を塞いでしまい、利用可能な出口がなくなることで発生する。この問題に対処するため、ジェネレーターには、要件を満たす 生成が行われるまで新しいジェネレーターを生成し続ける処理を実装した。これにより、無限ループの発生 を防いでいる。

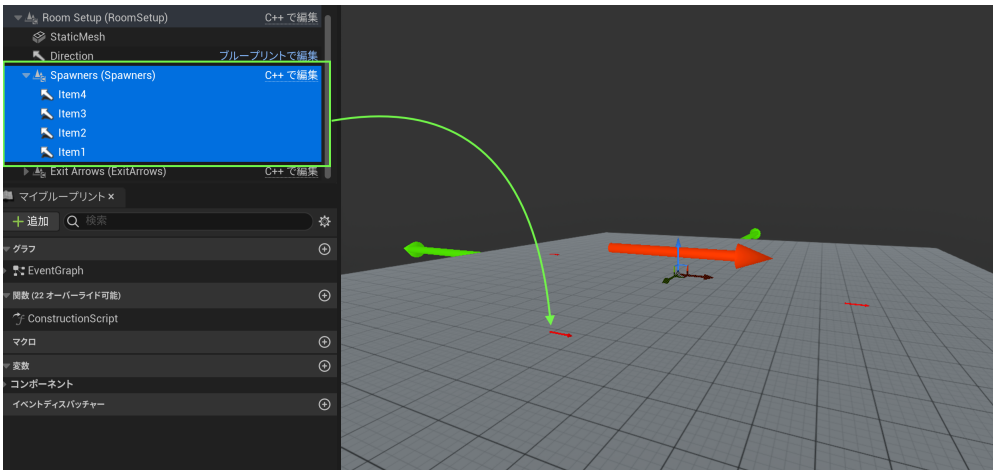

図 4: プロップのスパウナー

[2]もっと詳しい情報については、以下のリポジトリを参照されたい：https://github.com/AndreM222/Procedural-Generator/blob/main/docs/setup.md

図 5: プロップの設定

プロップおよびスポーナー

エンティティ用のスポーナーや障害物を追加するため、部屋全体に対してランダムにアクターを 生成できるスポーン関数を実装した。この関数にはフィルタリング機能があり、生成したい アクターのみを対象とすることができる。スポーン位置は、スポーナーフォルダ内に配置された スポーン用の矢印を参照して決定される。

キャラクター設定

3D モデルおよびアニメーションには、*Advance Locomotion System V4* [8] というアセットパックを使用している。このパックには、多数の有用なアニメーションと簡易的なモデルが含まれており、キャラクターの行動や思考の開発に集中することが可能となった。

このパックには移動用のセットアップも含まれているが、それらはすべて Blueprint で構築されており、C++ から 直接扱うことが難しい。そのため、本研究ではエンティティの空間内での相互作用に関する独自のロジックを実装した [3]。プロジェクト内の行動ロジック自体は Blueprint によって構築されているが、現時点では C++ から Blueprint への 直接的な干渉は不可能であり、その逆のみが可能である。この制約を踏まえ、アニメーションは既存のパッケージを 使用しつつ、より現実的な移動を実現するための独自ロコモーションを実装した。

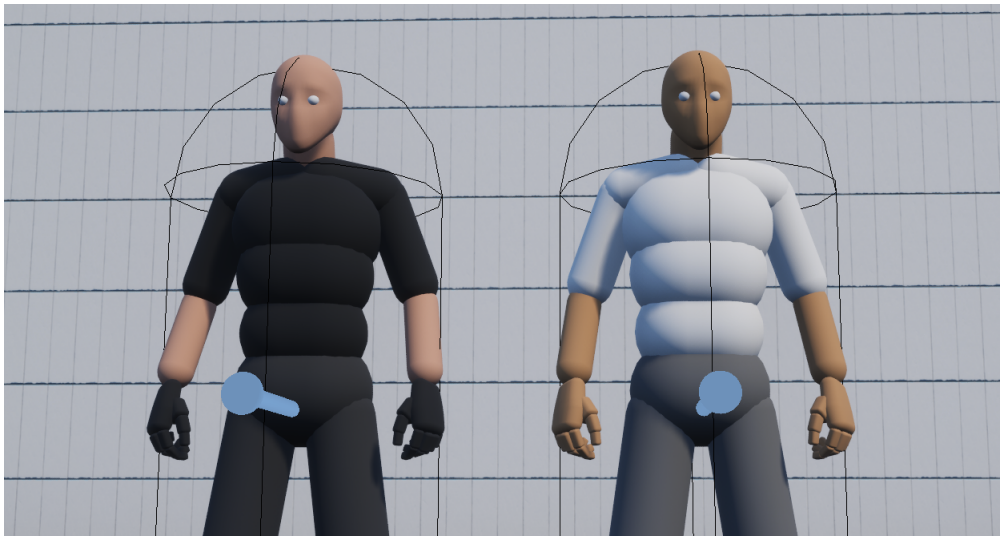

図 6: キャラクター設定

[3] コードについては、以下のリポジトリを参照されたい：https://github.com/AndreM222/AI-Entities

状態

状態とは、エンティティが相互作用に応じて変化する異なる姿勢（スタンス）を指す。より人間らしい振る舞いを再現するため、アニメーションを用いた状態遷移を導入した。アニメーションを使用しなくても相互作用は可能であるが、反応の遅延や視覚的な分かりやすさ を考慮し、本研究ではアセットパックのアニメーションを活用している。以下に、現在実装されている 状態を示す。

- **使用中:** 何らかの相互作用のために現在使用されている。
- **予定:** 使用可能な状態ではあるが、有効化するためのロジックとはまだ結合されていない。

スタンス	説明	状態
Default	中立的かつ平常な状態を示す基本姿勢	予定
Masculine	戦闘準備状態、または力を誇示する姿勢	使用中
Feminine	より優雅で落ち着いた姿勢	予定
Injured	エンティティの体力が低下している状態を示す姿勢	使用中
HandsTied	行動制限が付与されている状態を示す姿勢	予定
Rifle	ライフルを保持している姿勢	予定
Pistol 1H	片手でピストルを保持している姿勢	予定
Pistol 2H	両手でピストルを保持している姿勢	予定
Bow	弓を保持している姿勢	予定
Torch	照明用の松明を保持している姿勢	予定
Binoculars	遠距離視認のため双眼鏡を使用している姿勢	予定
Box	箱を運搬している姿勢	予定
Barrel	樽を運搬している姿勢	予定
Reach	対象に触れるため手を伸ばす姿勢	予定

行動

行動とは、エンティティが世界と相互作用するために持つ能力を指す。これにより、エンティティの思考 にはより高い複雑性と多様性が加えられる。

ゲノムによる意思決定

キャラクターの知能開発は、ゲノムを用いた手法から着手したが、親から提供される世代データ を用いて合理的な判断を行うための機能は、現時点ではまだ不足している。

この手法は、エンティティがマップ上の特定地点へ移動するような単純なタスクでは成功を収めた。しかし、鬼ごっこ（Tag）はより複雑な課題であり、エンティティにはさらに多くの情報と能力が求められる。

行動	説明	状態
Run	高速で移動する動作	使用中
Sprint	中程度の速度移動する動作	使用中
Walk	低速で移動する動作	使用中
Crouch	狭い空間でしゃがんで移動する動作	使用中
Climb	壁や障害物を登る動作	使用中
Grab	プロップを掴む動作	予定
Drop	保持しているアイテムを落とす動作	予定
Jump	ジャンプに夜移動動作	使用中
Touch	対象に接触する動作	使用中

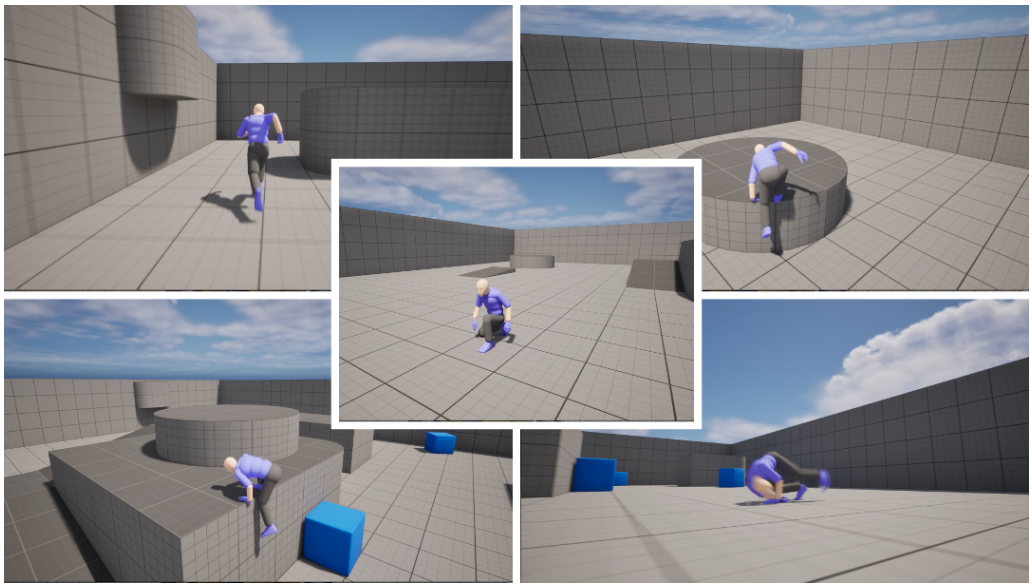

図 7: 行動のプレビュー

遺伝的アルゴリズム

この仕組みは、遺伝的アルゴリズム [9] に基づいている。これは進化 の概念を意思決定に応用するものである。単一のエンティティが突然変異によって自身の思考を構築すること も可能だが、その成長には限界があり、より良い問題解決方法を見つけられない可能性がある。

自己成長には内部要因と外部要因が存在する。内部要因とは自身の経験に基づく学習であり、外部要因とは他者の経験から学ぶことである。外部要因を取り入れることで、単なる成長だけでなく、より効率的な成長が 可能となる。

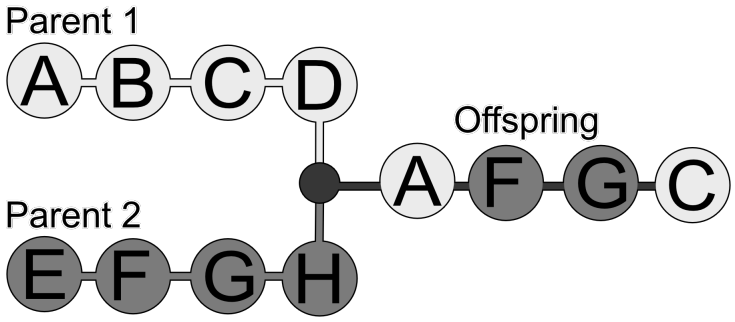

図 8: 遺伝的アリゴリズム

NEAT アルゴリズム

NEAT アルゴリズム [10] は遺伝的アルゴリズムを基盤とし、ゲノムの概念 を実際に脳の発達へと応用する。感覚、行動、内部ニューロンを持ち、それらが拡張トポロジー として結合されることで、脳構造そのものが進化する。ただし、制御のためニューロン数には上限を設けている。

このアルゴリズムでは、タスクの達成度に基づいて評価された二つの親を選択し、類似度（フェロモン）に基づいて結合を行う。ニューロン同士は相互に、あるいは自己接続も可能であるが、不要な接続 を削除するクリーンアップ処理を実装している。

感覚が多すぎるとエンティティは情報過多となり、行動が適切に行えなくなる。一方で、感覚 が少なすぎると、最適な問題解決方法を見つけることができない。

感覚

本研究では、人間の感覚に近い仕組みをエンティティに実装し始めた。ここで言う感覚とは、視覚や聴覚などを指すが、本プロジェクトでは以下の感覚に限定している。

すべての入力には細分化された要素が存在し、それがさらなる複雑性を生み出している。例えば視覚 を実現するためには、アクターとの距離、壁との距離、個体数などを取得する必要がある。そのため、距離や数量、正面に存在する物体などを計算するために、複数のトレース処理を使用している。

初期段階では、行動は走行、ジャンプ、登攀のみであり、感覚としては個体密度や距離、最小限の境界検知の

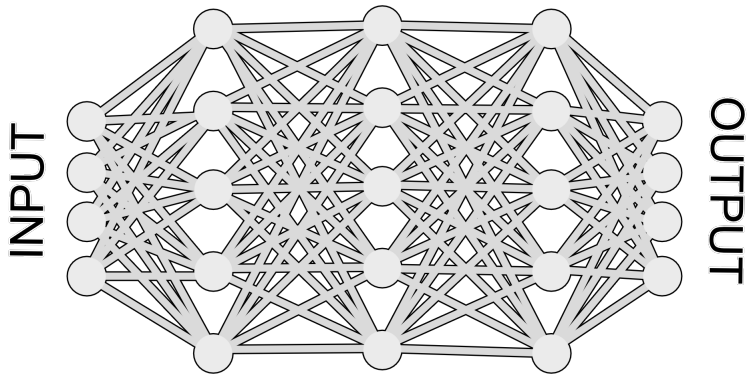

図 9: NEAT アルゴリズム

感覚の種類	説明	状態
空間	エンティティが自身の位置を把握する能力を与える	使用中
触覚	エンティティとの相互作用の情報を取得し、それを好むか嫌うかを判断するために使用される	開発中
資格	エンティティが先方にある物体を認識し、次の行動を判断するために使用される	使用中

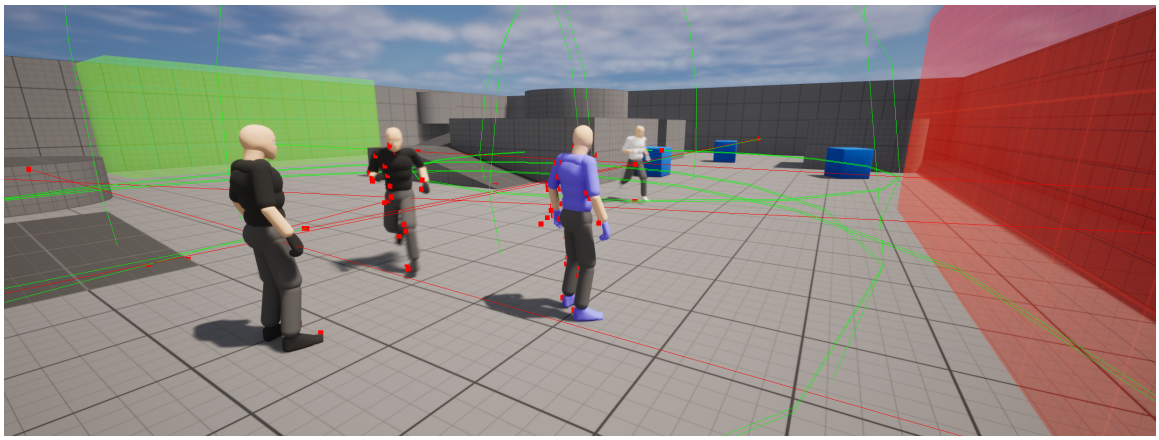

図 10: 感覚のプレビュー

みを実装していた。その結果、キャラクターは同じ場所を周回するだけで、特筆すべき行動を示さなかった。

そこで、崖検知、境界に基づく現在位置、正規化された値などの感覚と行動を追加したところ、思考過程が拡張され、より活発に移動するようになった。中には何もしない個体や、壁を検知した 際にジャンプする個体も現れた。この変化は、感覚と行動が増えるにつれて発達が促進される様子 を観察でき、非常に興味深い結果となった。

統合アーキテクチャ

本プロジェクトは規模が大きく、またテスト用のアセットが最終成果物には不要となるため、各コンポーネントを 別々のリポジトリとして管理した。ただし、最終的にはそれらを統合する必要がある。Unreal Engine では、他のツールと比べて複数プロジェクトの統合が容易ではないため、CMake を用いた自動マージ機構を構築した [4]。

このプロジェクトは、すべてのコンポーネント間の通信および UI を担当し、元のスクリプトに影響を与えない 形で統合を行う。これにより、コードを分離したまま安全に管理することが可能となる

CMake 選定理由

当初は Bash スクリプトを用いてプロジェクト統合を行うことも検討したが、この方法では特定の OS に 依存してしまう。CMake はインストールが必要ではあるものの、クロスプラットフォームで動作するため、理想的な選択肢となった。

導入方法

セットアップを簡略化するため、統合対象のプロジェクトを列挙する JSON ファイルを使用し、フィルタリング機能も実装した。このマージ処理では、ビルドファイルやメインスクリプトなど不要なファイルを削除し、すべての API を現在のプロジェクト用に置き換える。

評価

経路行動

エンティティは、特定の方向へ移動し続け、対象に接触するか、あるいは回避するまで同様の行動パターン を示す傾向がある。しかし、時間の経過とともに、異なる環境における適応能力と行動には継続的な改善が 見られる。

I programmed some creatures [3] と同様の条件でテストを 行い、特定地点へ移動するという単純なタスクを与えたところ、同様の反応が確認された。これは、鬼ごっこというタスクが、視覚、触覚、空間認識といった感覚を必要とし、より合理的な意思決定 を求めるためであると考えられる。

[4]統合アーキテクチャについては、以下のリポジトリを参照されたい：https://github.com/AndreM222/Tag-AI-Sandbox

障害物

現時点ではテスト回数が少ないため、エンティティは特定の角度から壁に向かって歩行し、必要に応じて反対側へ移動する様子が確認されている。

今後はさらなるテストを行い、行動が時間とともにどのように改善されるかを検証する必要がある。

今後の課題

今後は、意思決定構造の開発を継続しつつ、スーパーコンピュータ上へプロジェクトを移行し、機能実装と 並行して大量のデータ取得を行う予定である。また、各世代の挙動データを取得し、スクリーンショットを 自動保存できる UI の開発も必要となる。

エンティティの意思決定を改善するため、さらなる感覚を追加する予定である。これにより、過去の位置、接触、視認情報に基づいて行動を選択できるようになり、対象を興味深いもの、危険なもの、重要でないものとして判断できるようになる。

現時点では、エンティティの行動はまだ原始的であり、さらなる改良が必要である。今後の反復実験を通じて、環境との相互作用がどのように発展し、対象を捕まえる、あるいは逃げ切ることを学習できるかを検証していく。

参考文献

[1] R. A. Mossi and R. Sundaram, “Tag ai-sandbox,” in *2025 ASEE North Central Section (NCS) Annual Conference*, Marshall University, Huntington, West Virginia: ASEE Conferences, Mar. 2025. [Online]. Available: https://peer.asee.org/54692

[2] OpenAI. “Multi-agent hide and seek,” Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=kopoLzvh5jY

[3] D. R. Miller. “I programmed some creatures,” Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=N3tRFayqVtk&t=2368s

[4] GeeksForGeeks. “A* search algorithm,” Accessed: Feb. 14, 2025. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/

[5] Jacob. “I rewrote my dungeon generator!” Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=NpS5v_Tg4Bw&t=200s

[6] GeeksForGeeks. “Breadth first search or bfs for a graph,” Accessed: Feb. 5, 2025. [Online]. Available: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[7] R. Ree. “How to create epic procedural dungeons,” Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=4ddbnQIuwAM

[8] Longmire. “Advance locomotion system v4,” Accessed: Feb. 14, 2025. [Online]. Available: https://www.fab.com/listings/ef9651a4-fb55-4866-a2d9-1b38b028f9c7

[9] GeeksforGeeks. “Genetic algorithms,” Accessed: Mar. 30, 2025. [Online]. Available: https://www.geeksforgeeks.org/genetic-algorithms/

[10] R. MacWha. “Evolving ais using a neat algorithm,” Accessed: Mar. 30, 2025. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/