igda Japan chapter

ゲームＡＩ連続セミナー「ゲームＡＩを読み解く」

第１回

Killzone における NPC の動的な制御方法

資料

Based on

Remco Straatman, Arjen Beij, William van der Sterren,

“Killzone's AI : Dynamic Procedural Combat Tactics”

http://www.cgf-ai.com/docs/straatman_remco_killzone_ai.pdf

フロム・ソフトウエア　技術部

三宅　陽一郎

y_miyake@fromsoftware.co.jp

更新履歴

ver1.0　2006 年 12 月 16 日　**第１回セミナーにおいて配布。**

ver1.2　2006 年 12 月 20 日　**公開版として作成。**

(1) **全ての図に引用先とページ数を明記。**

(2) **第３章に「可視距離の計算方法」と「ロバストなシステム」の解説と図を追加。**

(3) **第２，３章冒頭のウエイポイントの配置に関する記述を追加。**

目次

version 1.2　2006/12/20

ゲームＡＩ連続セミナーの目指す方向

次世代機のハードウエアの性能の向上は、ソフトウエアから見た場合、シミュレーション能力の向上として捉えることが出来ます。基本的なゲームのシステムを動かす性能に加えて、さまざまなダイナミクス（運動）のシミュレーションを行うことが出来ます。

- 社会のダイナミクスのシミュレーション（社会シミュレーション）、
- 知能のダイナミクスのシミュレーション（人工知能）、
- 物理のダイナミクスのシミュレーション（物理シミュレーション）、
- 生物のダイナミクスのシミュレーション（人工生命）

などです。これまでアカデミックな分野の研究手法であったシミュレーション技術は、ある程度ゲーム空間へ導入が可能になり、ゲーム開発者は、これまで蓄積されて来たノウハウを調査、研究することで自由にゲームへ応用できるチャンスを持つ立場にあります。ところが、どの分野においても基礎研究から実際の応用には幾つかの困難がつきまといます。

（１）どの分野のどの領域が応用可能であるのか？
（２）どのように応用すればよいのか？
（３）どのように使いこなせばいいのか？
（４）既存の技術と比較してコストを費やしてまで導入するメリットは何か？

ゲームの分野では、上記の一般的な問題に加えて、

（５）ゲームデザインにどう活かせばいいのか？

というコンテンツに依存した問題があります。

このような問題を克服するためには、まず初めに開発者自ら試行錯誤しながら追求する必要がありますが、「他の開発者がどのように、この問題を克服して来たか」を理解し参考にすることもまた非常に有用です。特にゲームＡＩの技術はゲームコンテンツ（キュラクターの振る舞いや思考、ユーザーの行動の解析）を高度に構造化するという特徴も持っており、ゲームの発展に対して多様な可能性を持っています。そして、ゲームコンテンツによって応用する技術が違って来るために、各開発者が使用する技術をそれぞれの方向に発展する、という現象が見られます。つまり、ゲームＡＩは一方向に対する競争よりはむしろ多様性を持った発展を持っています。そういった状況では開発者相互のコミュニケーションとインタラクションが必要であり、欧米ではゲームＡＩにおける技術やノウハウがＧＤＣを始めとして大小さまざまなカンファレンスにおいて公開されています。日本においてはこのような機会は少ないですが、日本のゲーム業界は多様なジャンルのゲームを発展させて来た経緯があり、そこには、世界においても有数の多様なゲームＡＩを発展させるチャンスがあります。

本セミナーでは、主に、知能のダイナミクスのシミュレーション（人工知能）の分野において、人工知能、及びその周辺分野で研究されて来た技術をゲームへ応用することに成功した欧米のタイトルを中心に公開された論文の内容を紹介すると共に、開発者の相互の創造的なインタラクションを目指してゲームＡＩにおけるディスカッションを行い、これからのゲーム開発において独創的なアイデアを発想することを目指します。

ゲームＡＩ連続セミナーは全６回からなり各回ごとに独立した内容を扱いますが、各回ごとの参加も可能であるように構成されています。全６回はゲームＡＩの技術全般をカバーすると共に、それぞれの技術の差異が明瞭に分かるように構成します。予定しているタイトルと技術的テーマは、

- Killzone　地形解析の手法
- F.E.A.R　ゴール指向型ＡＩ
- Halo2　HFSM
- Counter Strike　Navigation Mesh
- Quake　多層型アーキテクチャーＡＩ
- Chrome Hounds　チームＡＩ

などです。

ゲームＡＩの技術の選択はゲームデザインに強く依存します。そこで、新しいゲームＡＩ技術の導入は、ゲーム企画者と技術者の知識の共有とコミュニケーションがあって初めて実現が可能です。本セミナーの目標の一つは、前半において企画者と技術者が同じ講義を受けることを通じて知識を共有し、さらに後半に用意されるディスカッションを通して意見を交換し合うことでアイデアを育む過程を提供することです。また、そのような場として積極的に利用して頂くために、参加者からの意見のフィードバックを取り入れて、毎回、セミナーのあり方を改善して行く予定です。

ゲームＡＩには、多様かつ広大な可能性があります。ただ、これまでは、それが多様さから来るとりつきにくくつかまえにくいという人工知能側の性質と、ハードウエア側のリソースの制限から、導入が為されて来なかったという事情がありました。このセミナーを利用して、これから歩くことになる遠く豊穣に広がっているゲームＡＩの風景を展望して頂くことができれば幸いです。

Killzone における NPC の動的な制御方法

このテキストでは、知識表現と世界表現のゲームＡＩにおける方法を、Killzone(PS2,2004-2005 SCEE. Published by SEGA. Developed by Guerrilla)を題材に解説します。第１章で基本概念について説明し、第２、３章でその応用方法を解説します。第４章では、第２，３章に含まれていた技術をパターンとしてまとめると同時に、他のゲームにおける応用例を紹介します。

第１章　知識表現と世界表現

この章では、Killzone のＡＩの基礎となる概念について解説します。人工知能の賢さは自分の属する世界をどれだけ認識しているかに依存します。人工知能に世界を認識させる方法には二つのアプローチがあり、一つは人工知能に自ら世界を解釈させる方法です。ＡＩの持つ知覚情報から自発的に世界を解釈させます。もう一つの方法は、人間が事前に世界についての知識をＡＩに与えるという方法です。この方法を**知識表現**の方法と言います。前者は、人工知能の中でも難しい方向です。例えば、複数の積み木の山があり、ＡＩに積み木を組み替えさせたいという問題があるとします。前者のアプローチは、積み木の状態をロボットの視覚であるカメラの画像情報からＡＩに認識させ行動させようとします。しかし、どの積み木が一つの塊であるかさえ、そこから判断させることは難しいことです。まして積み木の関係を把握するまではには、たくさんの計算と時間がかかります。一方、後者の知識表現の方法では、積み木の関係をデータ構造として人間が与えてやります。そして、そのグラフ情報を参照して積み木を並べかえさせることができます。

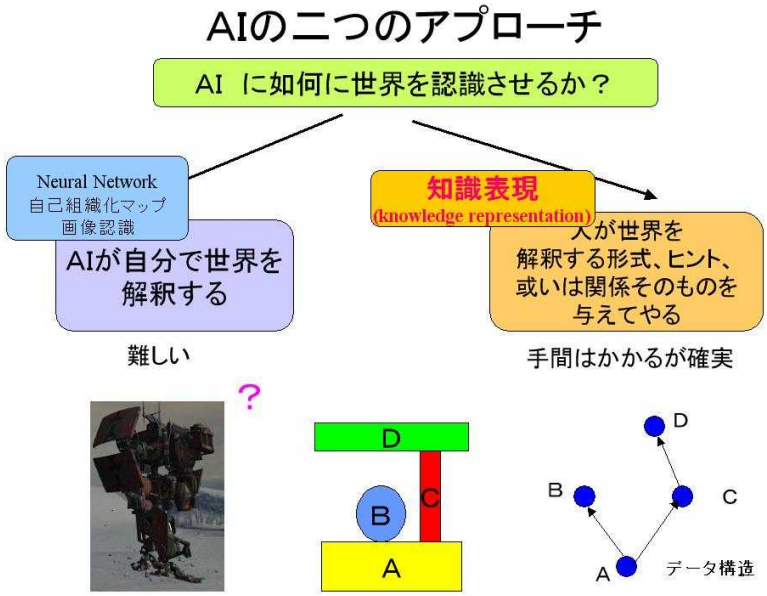


図１　ＡＩの二つのアプローチ。左はＡＩ自身が世界を自ら解釈できることを目指す。右は、人間が世界の情報をいかにうまく表現して（知識表現）ＡＩに与えてやるか、というアプローチ。積み木を例にすれば、前者は目からの情報で積み木A，B，C、Dの関係を理解しようとするが、後者は、グラフ構造をしたデータとして与える。

ゲームＡＩにおいてはゲームシステムからＣＯＭに確実に情報を渡せるという点で後者のアプローチが適しています。このテキストでは知識表現からゲームＡＩを発展させる方法について解説します。するとゲーム

ＡＩにおいては「人工知能の賢さはどのような知識表現を与えるか」という点が非常に重要になると言えます。

ゲームにおける知識表現の最も簡単な例は、ウエイポイントやナビゲーションメッシュと言った、パス検索のためのデータです。ＡＩにレベルデザインの３Ｄポリゴンデータを渡しても何も出来ませんが、レベルデザインからＡＩの通れるポイントをグラフ構造としてデータ化したウエイポイントを与えてやれば、任意のポイントからポイントへスパスを検索することが可能になります。このように、地形などグローバルな世界の情報についての知識表現のことを特に**世界表現(World Representation)**と言います。世界表現には、位置情報の他に、各ウエイポイントの明るさ、視線領域、射線領域、表面状態（水の中なのか、砂の上なのか）や、気温、或いはもっと抽象的にどの国の領土なのかのフラグなど、様々な情報を仕込むことが出来ます。その情報を基にＣＯＭに高度な行動をさせることが可能となります。(例えば、なるべくA国の領土を通らないようにC国へ行くなどの行動を取らせることが出来ます)

Killzone は、この世界表現の方法を利用して「敵の動きを予測して手榴弾を投げる」「敵の進路上に威嚇射撃をする」という高度な動作を実現することに成功しました。このテキストでは Killzone のＡＩを通して世界表現によるゲームデザインついて解説を行います。まず、この章では、その基本となる「位置を基本とした世界表現の方法」について解説し、次の２章、３章で実際の Killzone における実装について解説します。

第１節　位置を基本とした世界表現

位置を基本とした世界表現とは、座標、ウエイポイント、メッシュなど位置を明示する対象に対して情報を蓄積するシステムのことです。以下では、これを単に世界表現と呼びます。例えば、各ウエイポイントに対して、位置だけでなく、その高さ、明るさ、アイテムがそこにあるか、まわりは崖や壁があるか、敵が近くにいるか、などを付属するデータとして持たせるとします。すると、ＣＯＭはその情報を参考にしながら、ステージを移動して行くことが出来ます。策敵をするために見晴らしのよいポイントへ行きたければ、周囲のウエイポイントから最も高い位置にある点を検索し、敵から目立ちたくなければ、明るさの少ないポイントを検索して移動します。あるいは、これらの情報から、「高くてなるべく影になっている場所」も評価関数を導入してみつけることが出来ます。

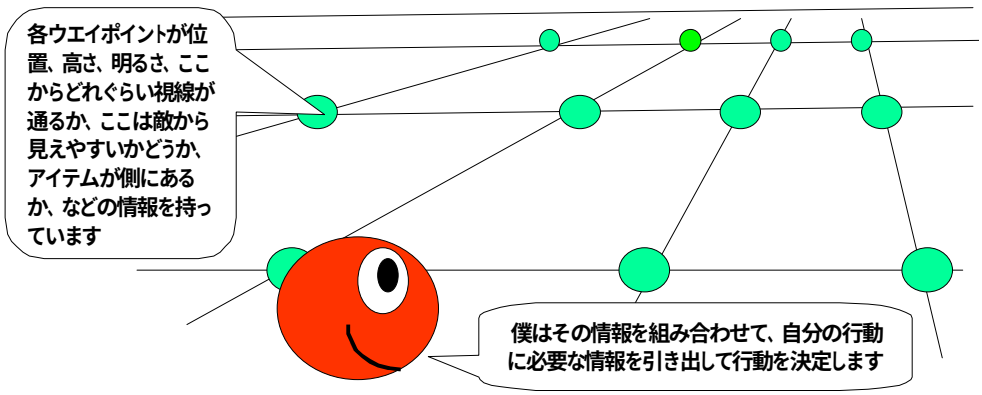


図 2 マップから作成したウエイポイント　各ポイントには位置座標だけでなく、高さ、明るさ、視線領域、

など、ＣＯＭに必要なデータが準備されている。その情報をもとに、ＣＯＭは自分の行動を決定する、というのが今回のセミナーで説明するイメージです。

さらに、より高度な行動を実現させたければ、その行動に必要なデータをポイントごとに埋め込んで行きます。例えば、ＣＯＭに「敵が来たら隠れる」という機能を追加したいとします。「岩の側にある」というフラグをウエイポイントに付属するデータとして用意する必要してやります。するとＣＯＭは、「岩の側にあるポイントの中で敵の進行方向と岩を結んだ方向のポイントへ移動」することで隠れることが可能となります([1])。

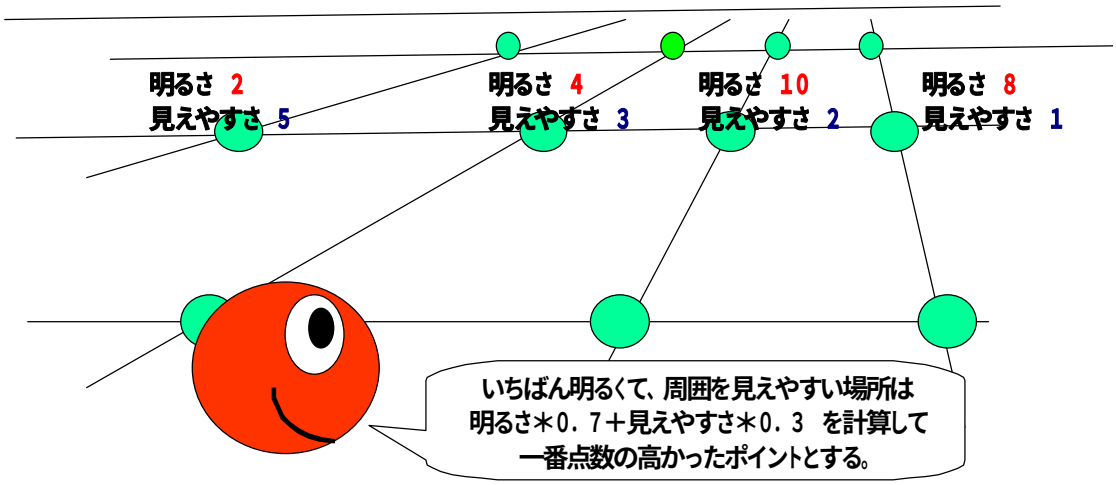


図 3　位置を基本とした世界表現。例えば彼は今、迷子になっていて味方を見つけたいし、また、自分をみつけてもらいたいので、「明るくて見晴らしのよい場所」を探しています。各ポイントには「明るさ」「どれくらい遠くまで見えやすいか」という点数があらかじめ、つけられています。そして「明るくて遠くまで見えやすいか」という総合点をそこから付けます。図の中の式の０．７とか０．３はウエイトで、明るさと見やすさは、７：３の割合で大切だ、ということを現します。この場合は、左から、2.9、3.7、7.6、5.9 となるので、３つ目のポイントへ移動します。

ゲームが発展するに従い、３Ｄアクションゲームは徐々にステージを広く複雑にしました。それに伴い、ＣＯＭに対する地形を認識して上図に動く能力の必要性が上がって来ました。地形解析(terrain reasoning, terrain analysis)の能力です。地形解析は一般に計算負荷がかかると同時に、目的によって抽出する情報が異なります。そこで事前計算（pre-computed、開発の段階においてデータを用意しておくこと）しておいたデータをＣＯＭのために準備しておくことが、解決策の一つとして使用されて来ました。事前計算のよいところは、ゲーム中にリアルタイムに計算する必要をなくし、より精緻なデータを時間をかけて作れるところです。実際、それはＣＯＭにゲーム内のリアルタイム計算では不可能な高度な行動が可能にします。事前計算で用意されるデータは地形の性質やＣＯＭにとって必要なアクションがデータとして各ポイントに用意されます。

このようなシステムを取っているタイトルは少なくありません。

例えば、Soldier of Fortune2 では、ポイントに対してそこで行うべきアクションの行動の ID（ここでは飛ぶ）を持たせることで、ＣＯＭが竪穴をジャンプするように制御します([2])。また、Mat Backland の教科書([3])には、ポイントではなく、ポイント間を結ぶラインに行動にＩＤ（ここでは泳ぐ）を持たせることで、ＣＯＭが河を渡ろうとするときに泳ぐアクションを実現するコーディング方法が解説されています。また、Halo2 では、パス検索のためにナビゲーションメッシュを用意し、戦闘のためには戦術に用いる立ち位置（standing point）がレベルデザイナーによって用意されます。比較的局所的な戦闘におけるユニークな戦術的な行動（敵が来たら後退し、物陰から撃つ、など）を実現しています([4], [5])。地形を解析し、その情報を世界表現として持たせたシステムには、いろいろな可能性と応用があります。Killzone はその中でも、ＡＩシステム全体を世界表現の位置依存情報から組み上げている特徴的な例であると言えます。

第２節 リアルタイムか、事前解析か？

さて、位置依存情報にも、二つの種類があります。

一つは、マップに対して、崖の側である、とか、水の中である、とか、影が落ちている場所である（ライティングが一定なら）、狙撃ポイント（地形が変化しないなら）などゲーム中に変化がない静的(変化しない)な状態から事前に用意することが出来る(pre-computed)情報です。こうった情報は崖の側に近づかないようにとか、水の中は速度が減衰するので、なるべく砂地を通るなど、変化しないマップの性質に対するＣＯＭの制御の用いることが出来ます（[6]）。

もう一つは、状況に対して変化する情報です。例えば「敵から隠れることができるポイント」は敵の位置が変われば変わってしまうものです。そういったデータは事前に用意することが出来ませんので、ゲーム進行中に状況に応じて動的(dynamic)にゲーム進行中に計算によって(procedural)見出す必要があります。

Killzone の AI は、その論文の題名(dynamic procedural)にあるように、後者の動的なシステムの上にＣＯＭを実装しています。しかし、その前に、静的なマップに対する先行研究がありました。第２章ではまず、その Killzone の基本アイデアとなった、事前計算(pre-computed)の方法を２００１年の William van der Sterren の論文（[7]）に沿って説明することにします。そして、第３章で、Killzone の動的な方法を解説します。静的な方法は、計算量が多いので、動的な方法にそのまま発展することはできませんが、アイデアの基本部分は、全て Killzone のシステムに受け継がれています。まず、第２章でアイデアの概要をつかんで、第３章を読まれることをお薦めします。

第２章　静的位置検出

この章では、Killzone のＡＩのシステムとなった、先行研究の論文（[7],[8]）の説明を行います。この論文では、静的なマップ（物が壊れたり移動したりしない、敵の来る方向は決まっている）において、事前計算によって狙撃ポイント(sniping point)を見出す方法が説明されています。

第１節　基本情報をマップから抽出する

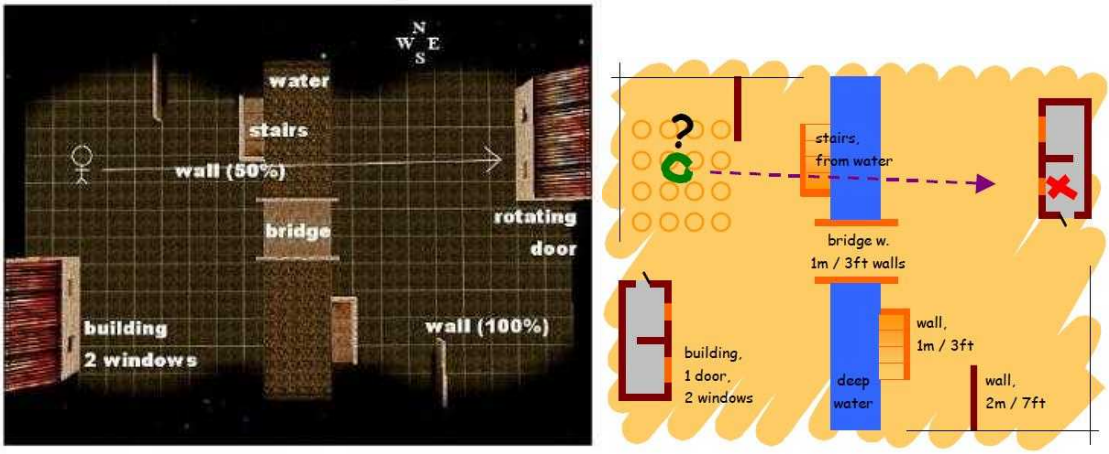


図 4　この章で解析するマップ。対称にオブジェクトが配置されたマップ。地形解析から狙撃ポイントを見つけ出すためのモデルとして用意された。４８０個のウエイポイントからなる。。マップは基本的に平坦で凹凸がありません。上を北、として、東と西に家があり、中央の川がエリアを２分して、橋がそれをつなぎます。川の両岸にか階段があって、川から陸に上がることが出来ます([7] P.9,[8] P.30)。

まず、ウエイポイントを等間隔に敷き詰めます（Sterren の方法は、ウエイポイントをまるでセンサーアレイのように特に工夫することなく並べます。この点は、Halo2 のアプローチとは全く違うところです。Killzone ではオブジェクトの周りのポイントは調整されています。）この各点に対して、この地形情報を埋め込んで行きます。前提としてキャラクターの現在位置は、最も近いウエイポイントとみなします。

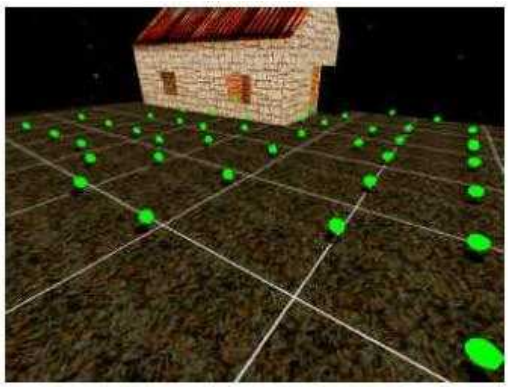


図 5　等間隔にウエイポイントを敷き詰める（[8] P.31)。

地形情報を、以下の４つの関係に分類します。

- **１．局所的な性質**　　どれぐらいの明るさか、そこでＣＯＭが要求されるアクション（かがむのか、泳ぐのか）など。
- **２．グループとしての性質**　一つの部屋の中、或いは、河の中は、ウエイポイントのグループを形成する。
- **３．他のウエイポイントとの関係**　他のウエイポイントに視線が通るか、射線が通るか、どれぐらいの距離か
- **４．フォーカス**　ある性質がどれぐらい偏っているか。例えば、そのウエイポイントが他の方向より東側に、視界が開けているなら、そのウエイポイントは東向きにフォーカスを持つという。

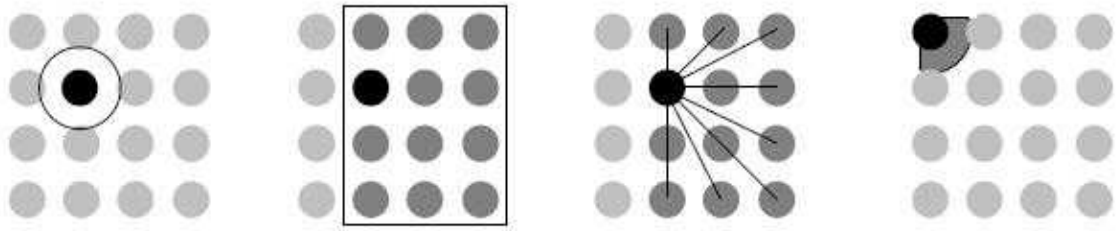


図 6　地形情報の４つの分類。左から、局所的な性質、グループとしての性質、他のウエイポイントとの関係、フォーカス。([7] P.5)

そして、ここがこのこの論文の最も特徴的なアイデアなのですが、「ウエイポイント間の関係によって地形情報を抽出する」ということします。視線情報に注目してみます(下図３つ目)。「北向き４５度方向の視線の距離」と言った場合、実際には、レイキャストして３Ｄオブジェクトと交差するポイントまでの距離ですが、ここでは、その距離はその方向に視線の通る最も遠いウエイポイントまでの距離であると定義します。それを、４５度づつの各方向に対してチェックします。

第２節　狙撃ポイントを見出す

地形情報をもとに、より抽象的な位置評価関数(position evaluation function)によって情報を抽出します。位置評価関数とは、地形情報から抽象的な情報に点数をつける関数のことを言います。例えば、第１章の図３で、「高くて明るい場所」を見つける、ということをしました。しかし、高くて明るい場所という量は存在しないので、高さと明るさを変数とする簡単な関数を作って点数をつけました。このような関数を位置評価関数と言います。ここでは、一章よりずっと抽象的な量を扱います。

狙撃ポイントに必要な性質は、

- 気づかれにくい
- 遠くまで見渡すことができる
- 敵から近づきにくい
- 自分を遮蔽してくれるものがある
- ターゲットが裸である（遮蔽がない）

...
などです。この性質を、評価する（点数をつける、数値化する）ためには、

- light level　－　どれぐらい明るいか
- obstacles　－　障害物があるか
- special terrain　－　特殊な地形か
- line distance　－　直線距離
- travel time　－　移動時間
- line of sight　－　視線が通るかどうか
- line of fire　－　射線が通るかどうか
- ...

などの情報が必要です。この対応を一覧表にしたのが、以下の図となります。

表 1　各特性（a,b,c...行）に対して、地形情報（列）を用いて評価するか、の対応表（[7] P.8）

	description \ terrain property	light level	obstacles	special terrain	line distance	travel time	line of sight	line of fire	portal member	game activity		property	directional	higher level	focus
a	inconspicuous	✔										✔			
b	overlook distant spots				✔		✔	✔					✔		✔
c	hard to reach					✔							✔		✔
d	cover from targets					✔	✔	✔					✔		✔
e	target without cover						✔	✔					✔		✔
f	overlook key spots			✔			✔	✔	✔				✔	✔	✔
g	observed approaches					✔	✔						✔		
h	focus														✔
i	local movement		✔	✔								✔			
j	*overlook traffic*			✔					✔				✔		✔

各ウエイポイントに対して、a,b,c... の抽象的な特性(description)をウエイポイントに付属させた地形情報から評価関数を用いて評価します。例えば、ここで、「敵からどれだけ隠れているか」(cover from target)という量の評価の仕方を見てみましょう。地形情報として、travel time、line of sight(視線)、line of fire(射線)を使って、以下の図の様に、敵の周囲のポイントから自分のポイントに射線が通るかどうかで評価します。

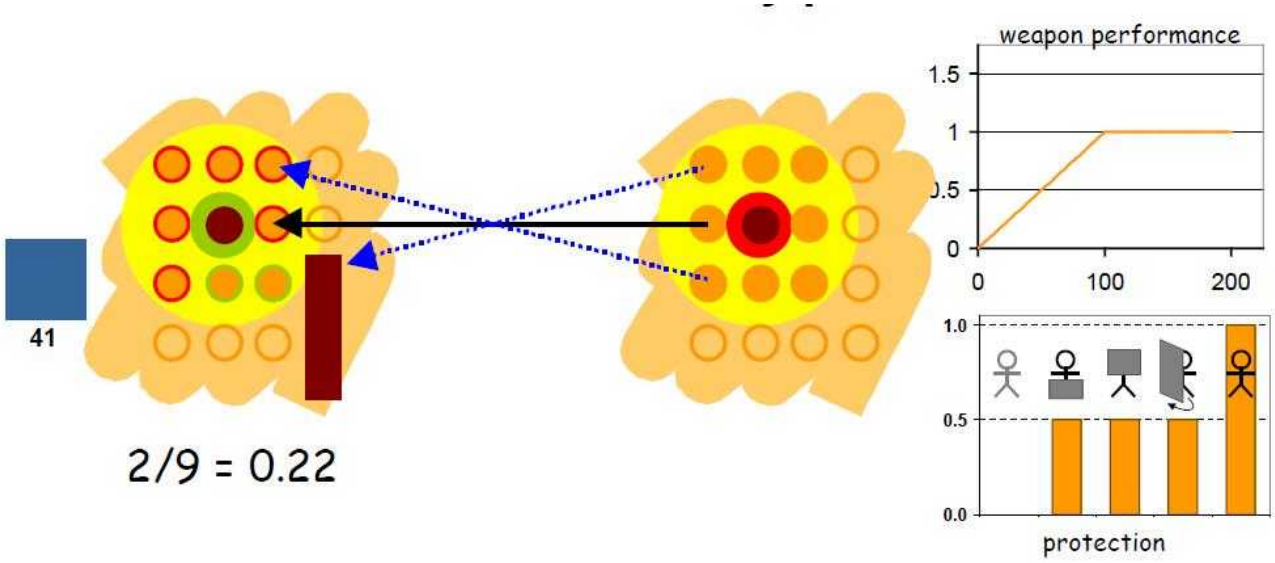


図 7　遮蔽「敵からどれだけ隠れているか」(cover from target)の評価の仕方。自分が敵から見て、どれだけ体の部分を隠していられているか、で計算する。敵（右）から自分（左）の方向へ向かった敵の３つのウエイポイントから自分の周囲のポイントに対する射線の通り方から評価する。上では、９つのうち、２しか隠れていないので、2/9 が評価値とする([8] P.41)。

このように、抽象的な量(「敵からどれだけ隠れているか」)を地形情報(視線や距離)から計算して行きます(他の評価関数の詳細については、論文をご覧ください)。そして、最終的に各ポイント(w)の方向 d（今は東向きを考えているのでｄは東向き）狙撃のための総合評価値は、

位置wのｄ方向に対する狙撃ポイントの評価値
＝　（局所的な性質を評価した点数）
＋　（グループとしての性質を評価した点数）
＋ (フォーカス)×　(他のウエイポイントとの関係を評価した点数)

となります。ここでは東向きに狙撃するとして、ステージ全体の各ポイントの評価値を計算します。評価値の全体の分布を見るためにビジュアル的な表現を用いて以下の下図（右）のように点数の高いところ（黄色）と低いところ（青）と気温図の等高線のように可視化します。(ビジュアル表現は、デバッグ機能として非常に有用です。数値で表現してもいいですが、一つ一つ数字を読む必要があります)。狙撃ポイントとして評価値の高いところは、『背面が壁になっており（後ろから狙われる心配がない)、かつ東側に視界が開けている場所』のはずです。図を見ればわかるように、そのように川の西壁と南西にある家の側が高い評価値を獲得していることがわかります。きちんと狙撃に適したポイントが導き出されていることがわかります。

おそらくここまで読まれた方は、あたりまえの結果がコンピューターに導かれただけだと感じられるかもしれません。人間なら、瞬時にわかることをこんな面倒くさい計算で求めて何の意味があるかと思われるかもしれません。確かに、この例だけではそうです。しかし、コンピューターのよいところは、同じ計算を疎まずにしてくれるところです。例えば、ここでは東向きの狙撃ポイントを求めましたが、入力として方向 d を

変えるだけで、すぐに候補となる狙撃ポイントを導き出してくれます。このように、全体のマップの中から目的に合うポイントを見つける技術を位置検出(point picking)と言います。

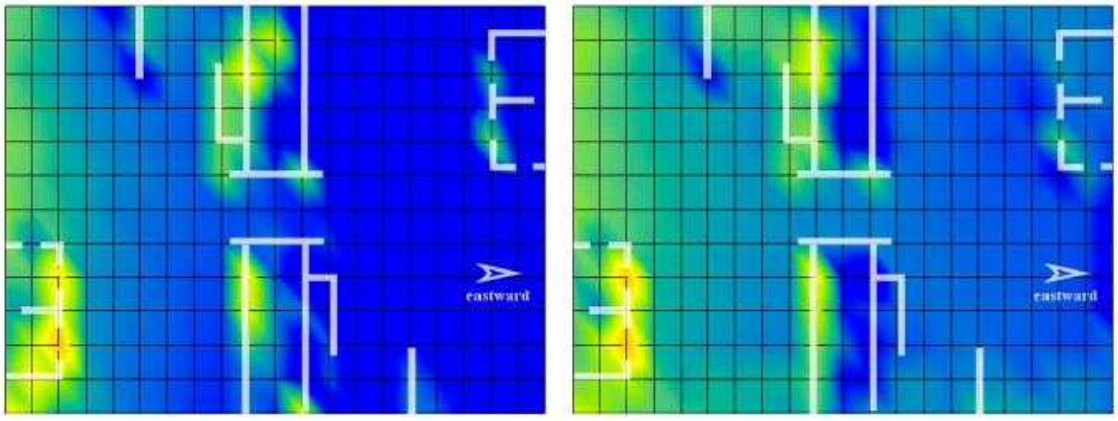


図 8　左は、東方向（右）に対するフォーカス（そのポイントがどちら向きに視線が開けているか）という評価値を示した図であり、黄色が評価値の高いポイント、青が評価値の低いポイントとなっている。右は、最終的な狙撃目的のための評価値を示したもの（[7] P.9）。

第３節 まとめ

世界表現と位置評価関数の方法によって「狙撃ポイント」をコンピューターに導かせることができました。それは位置検出と呼ばれます。もちろん「狙撃ポイント」は一つの例であり、さまざまな戦術的な性質を持つポイントを見出すことができます。確かにこれはゲームＡＩの一つの技術ですが、この技術が実際にゲームに使えるかどうかは、ゲームの質や種類、ハードウエアの性能に依存します。また、このままでは応用できないかもしれませんが、もう少し工夫すればゲームに使えることができるかもしれません。その道筋をみつけることが出来た開発者が、このシステムを自分のゲームへ導入することが出来ます。それを見てとったのが、Killzone の技術者でした。彼らは、このシステムをアレンジして自分のゲームへ応用することに挑戦し成功しました。次章で、この技術を基本として Killzone のアイデアの広げ方、技術的応用の仕方を見てみましょう。

第３章　動的位置検出

前章では、事前計算によって、あらかじめ狙撃ポイントの位置を検出する手法を紹介しました。しかし、例えば「ＣＯＭがプレイヤーに対して障害物の影に隠れる」という動作を実現しても、プレイヤーの位置によって隠れられる場所は違って来ます。そこで、ゲーム内でリアルタイム(rea;-time)に、その場の状況に応じて計算して(dynamic procedural)、隠れることが出来るポイントを見出す(point picking)必要があります。「時々刻々と変化するゲーム状況の中で、最も戦術的に有利な点を見出してそこへ移動して敵と戦う」必要があり、Killzone では、これを戦術的位置検出(tactical point picking)と呼んでいます（[5],[9]）。
この章では戦術的位置検出を解説します。

第１節 基本システム

まず、マップに対して、だいたい２ｍおきにウエイポイントをマップに敷き詰めます。地形に凹凸があり障害物が多いので、今度は、正方ではなく、ある程度ランダムな配置になっています。オブジェクトの周囲のポイントは調整されています。各キャラクターの位置は、最寄のウエイポイントと同一視します。

前章と違って、プレイヤーやＣＯＭたちが動的に動きまわる情報を取り込む必要がありますので、地形解析の静的な性質として用意するデータ（事前解析による、前章と殆ど同じ）と、リアルタイムに変化する動的なデータ（ゲーム進行中に計算）の両者を組み合わせて戦術的位置検出を行います。

第１項　射線判定、遮蔽判定

前章でも射線判定、遮蔽判定（敵から隠れているかどうか）は重要な役割を果たしました。ここでは、Killzone におけるそれを説明します。まず、複雑な地形に対応するために、４５度づつの８方向に分けた方向について、その位置にいる兵士が見える限界の距離を事前計算し、データとして用意します。この距離をここでは可視距離(visibility)と呼ぶことにします。

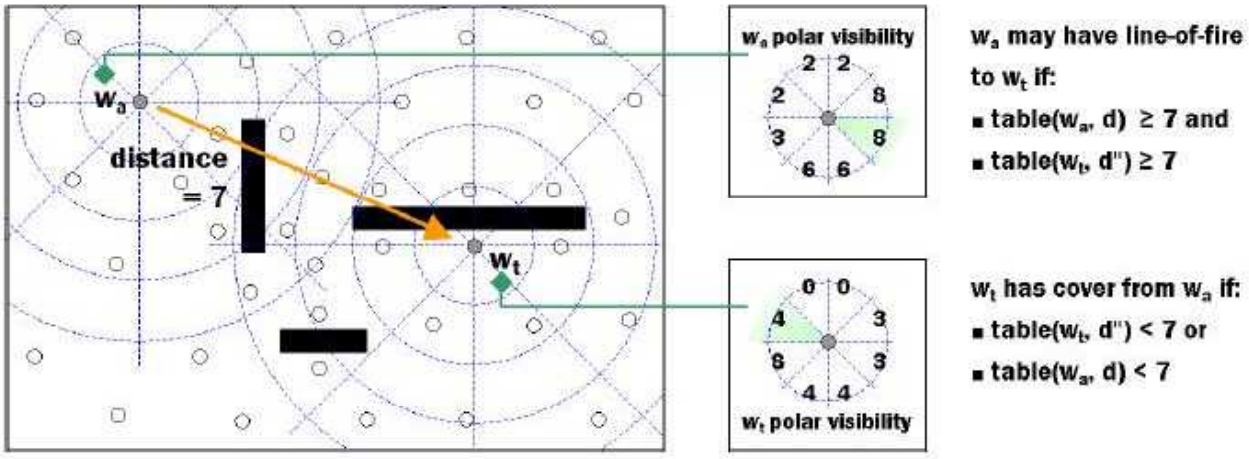


図 9　Killzone の世界表現の一つ。それぞれのポイントにおいて８方向に対して、兵士が立っている場合と、かがんでいる場合について、可視距離（そのポーズから見ることが出来る最大距離、立っているポーズと座っているポーズ

について計算する）を事前計算したデータを用意します。射線が通る＝遮蔽されていない、という関係となっています。（[5] P.9）

一つの方向に対する可視距離は、まず対象とするポイントWを中心に８個に分けた扇形の一つ（セクター）に含まれる全てのポイントから、対象とするポイントWに向けての ray cast 計算によって行います。まずWに兵士を座らせた場合について解説します。セクター内の全てポイントに兵士が立った場合と座った場合の ray cast 計算を行い、その中の最大値を、その方向の兵士が座った場合の可視距離とします。Wに兵士を立たせた場合も同様に計算し、このデータをテーブルにしておきます。（つまり、データの形式は、（ポイント、姿勢、可視距離）となります。）

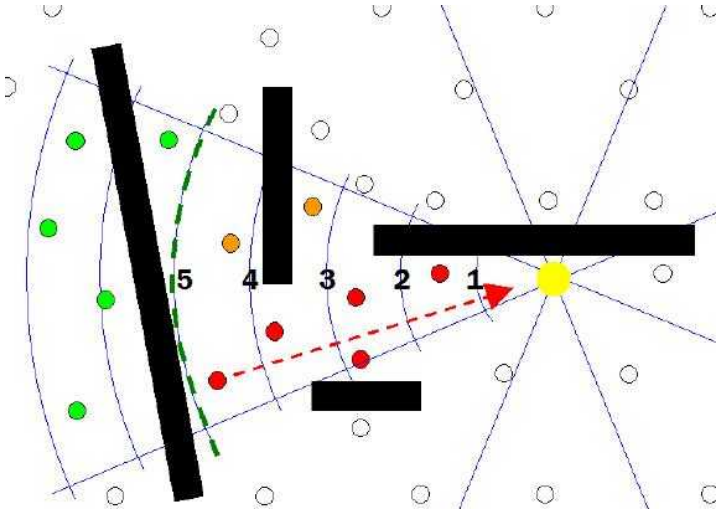


図 10　８方向に分けた扇形（セクター）内の可視距離の判定方法。対象となるポイント（黄色）に対して、セクター内の全てのポイントからの可視性(可視距離)を計算する。この例では、壁際のポイントから ray cast が通るので、この距離(5)がこの方向の可視距離となります([9] P.38)。

このデータをゲーム中に利用して射線判定と遮蔽判定を簡単なアルゴリズムで行います。上図では、例として７m離れた二つの地点 Wa 、Wt にＣＯＭと敵がいる場合の例を示しています。

射線判定は右上に書かれているように、敵の位置から見た自分の方向の可視距離が、実際の距離（７m）より長く、かつ、自分に位置から見た敵の位置の可視距離が、実際の距離（７m）より長ければ、射線が通っていると判定します。引用する距離はキャラクターの姿勢に従います。

遮蔽判定は、敵の位置から見た自分の方向の可視距離が、実際の距離（７m）より短い、或いは、自分に位置から見た敵の位置の可視距離が、実際の距離（７m）より短ければ遮蔽されていると判定します（射線判定は、遮蔽判定の論理的否定になっています)。遮蔽判定は、両方のうちどちらかの遮蔽があれば、という意味になっています。引用する距離は、キャラクターの姿勢に関わらず、二つのデータのうち大きな方を採用します。

ここまでがアルゴリズムの解説ですが、このシステムを少し深く考えてみましょう。

事前計算によって作ったテーブルは、その方向に対して、セクター内の最大値を取っているので少々悲観的過ぎるような気がします。実際には敵はもっと近くにいるにしても、遮蔽されてこちらが見えないという可能性だってあるのです。つまり、Killzone では、事前計算によって最悪の場合を想定した(worst case assumption)データが用意されています。また、遮蔽に関して言えば、姿勢にかかわらず、大きな方の可視距離を採用します。これも悪い方のケースを想定しているわけです。

これには２つ理由があります。一つは位置の精度に対する対処です。方向を離散的に（８つに）取っている精度の粗さを持っているので、それに対する保証をする必要があるのです。もう一つは、時間に対するものです。計算する場合には、敵のキャラクターの瞬時的な位置を取っています。しかし、キャラクターは移動しますので､ある敵の場所からは見えていなくても､少し動いたり姿勢を変えるとこちらが見えてしまう、ということがあるかもしれません。そういった場合に対処するために、あらかじめ遮蔽に対してはもっとも厳しい仮定のもとで可視距離を計算しておき、実際の判定でも悪い方（二つの姿勢のうち可視距離が長い方）を仮定する必要があるのです。このようにシステムの微細な特徴（ここでは厳密な精度）に依存せず、少々不備があっても安定なシステムとして機能する（ここではＣＯＭとして機能するということ）システムをロバスト(robust)なシステムと言います。この概念は、ＣＯＭを作る過程で重要な概念の一つでありますので、覚えておくと有用です。

このよう事前に用意したデータをゲーム中に動的に使用することで、射線チェックと遮蔽判定を簡単なアルゴリズムで代用し、負荷の軽いシステムを実現すると同時に、詳細な地形情報を利用した行動を取らせることが出来ます。

[プログラマー向け情報]　Killzone では、２０００ウエイポイントに対して、この事前計算のルックアップテーブルは全体で 32Kbyte のデータとなります。

第２項　　抽象的な性質の評価

前章と違って、移動する敵、味方の情報が必要ですので、戦術的位置検出のために

- 現在の位置からどれだけ近いか
- 敵から遮蔽されているか
- 敵の射程からどれぐらい離れているか
- 味方の射程からどれだけ離れているか
- 味方からどれだけ離れているか

…

のような情報を獲得する必要があります。そのためには、

- 第一脅威への距離
- 第一脅威への射線があるか（これは上記の静的なデータから計算）

- ◆ 第一脅威から遮蔽されているか
- ◆ 第二脅威から遮蔽されているか
- ◆ 自分の射線が敵へ通っているか

…

などの地形情報が必要です。それをまとめたものが、以下の表です。どんな情報(position evaluation input)とどのような特徴(description)の評価が必要なのかを列挙したものです。

表 2　左の列が位置評価入力のパラメーター。右が求めたい抽象敵特性。敵、友、自分、地形、４者の関係なので、いろいろと必要な性質が多い（[5] P.5）。

Position evaluation input	Description: (The evaluation function awards a higher score for ...)
Proximity to current position	being closer to or more quickly reached from the current position
Proximity from specific location	being closer to or more quickly reached from a specific location
Cover from primary threat	offering cover from the primary threat (given a stance)
Line-of-fire to primary threat	offering cover from the primary threat (given a stance)
Distance to primary threat	being preferred fighting distance from the primary threat
Outside danger zone	being outside the blast range of (expected) projectile
Cover from secondary threats	offering cover from one or more specified threats other than the primary threat
Outside friendly line-of-fire	being outside the line-of-fire of specified friendly units
Distance from friendly positions	being some distance away from friendly positions
Wall hugging	being close to a wall or obstacle in a specified direction
Nearby cover	being near positions offering cover from the primary threat
Player line-of-fire	not being a position across the player's line-of-fire from the current position
Preferred fighting range	being inside the preferred fighting range

第２節　戦術的位置決定(tactical point picking)とＣＯＭの動作の実現

さて、以上の準備のもとに、

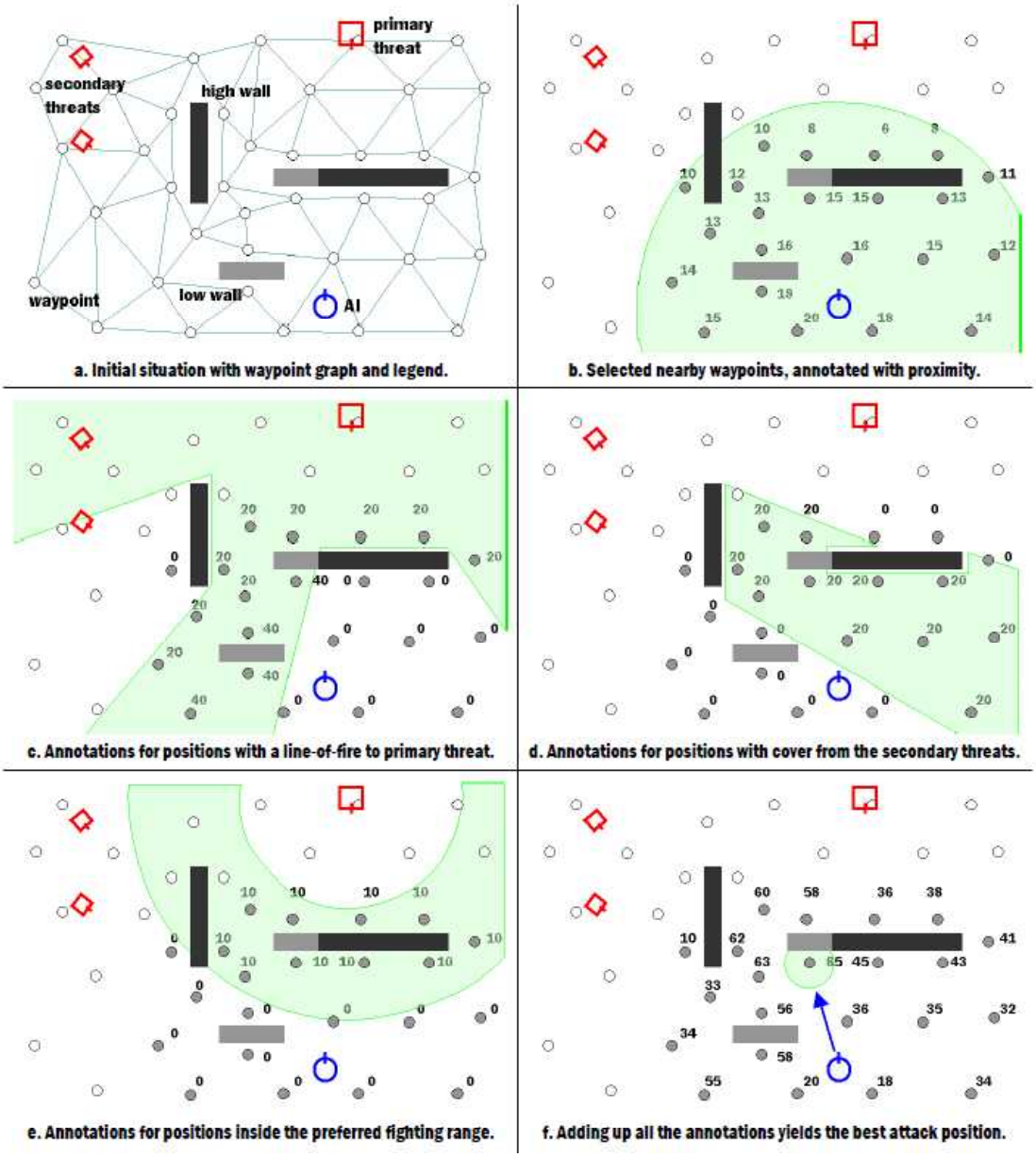
① 敵に対して最適な位置を取って攻撃する。
② 敵の動きを予測して手榴弾を投げる。
③ 敵の移動を予測して威嚇射撃を行う。
④ 敵の射線にさらされないように移動する。

の４つのＣＯＭの動作を戦術的位置検出によって実現する方法を説明します。

ここで大切なことは、先に説明したように、ray cast など計算負荷の高い複雑な計算を行う必要がない、ということです。事前計算したデータと簡単な計算から「敵に対して最適なポイント」「敵が移動しそうなポイント」「敵が出現そそうなポイント」を求めます。

①　敵に対して最適な位置を取って攻撃する

図 11 敵を撃つための戦術的位置取りの例。ＣＯＭ（AI）は第一の敵（primary threat）に対して射線が通り、第２の敵(secondary threat)からは障壁に隠れていられて、第一の敵から部分的に障壁があり（しゃがめば隠れられる）攻撃最適領域に捉えられる場所である場所を探して移動する。黒いバーは高い障害物、グレイは低い障害物でかがめば隠れることが出来る（[5] P.6）。



a.　初期配置とウエイポイントの図。

b.　ＡＩがこれからの決められた時間に移動の候補とする領域。ＡＩを中心として一定半径内で、近いほど評価値が高い。以下はこの領域のみを評価の対象とする。

c.　第一の敵（四角）に対して射線が通る領域。４０は立った状態で射線が通り座れば射線が通らないポイントの評価値。

d.　第二の敵（２体の菱形）から隠れることができる領域。隠れることができれば全て同じ評価値。

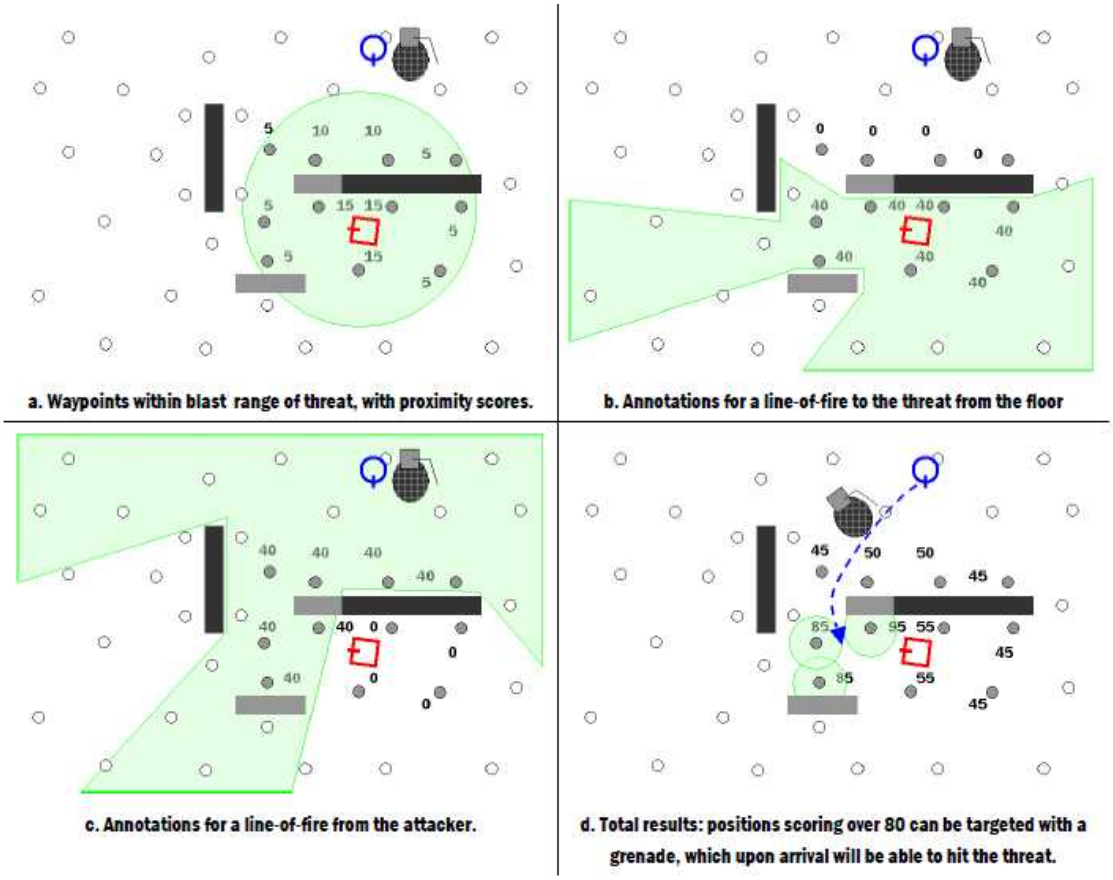
e.　第一の敵に対する最適な攻撃領域。二つの半径から指定される。

f.　a ~e から計算した総合評価値。低い障壁があって攻撃もできる場所が選ばれている。

②　敵の動きを予測して手榴弾を投げる

これまでは戦術的位置取りをＣＯＭに対して適用して来ました。しかし、ここで一つの発想として、ＣＯＭに適用するシステムを、敵（プレイヤー、敵のＣＯＭ）の側に適用することで、敵に対して最適なダメージを与えられる攻撃目標地点を見出す、ということをしてみます。

図 12　COM(AI、マップ上部)が敵（四角）に対して、敵が高確率でいるであろう場所を予測して手榴弾を投げる（[5] P.11）。



a.　敵の位置を中心に敵の移動候補領域を挙げる。移動時間が短いほど評価値が高い。

b.　敵の射線が通る場所の評価値。これは、手榴弾のダメージが敵に届く位置を評価するために使っている。

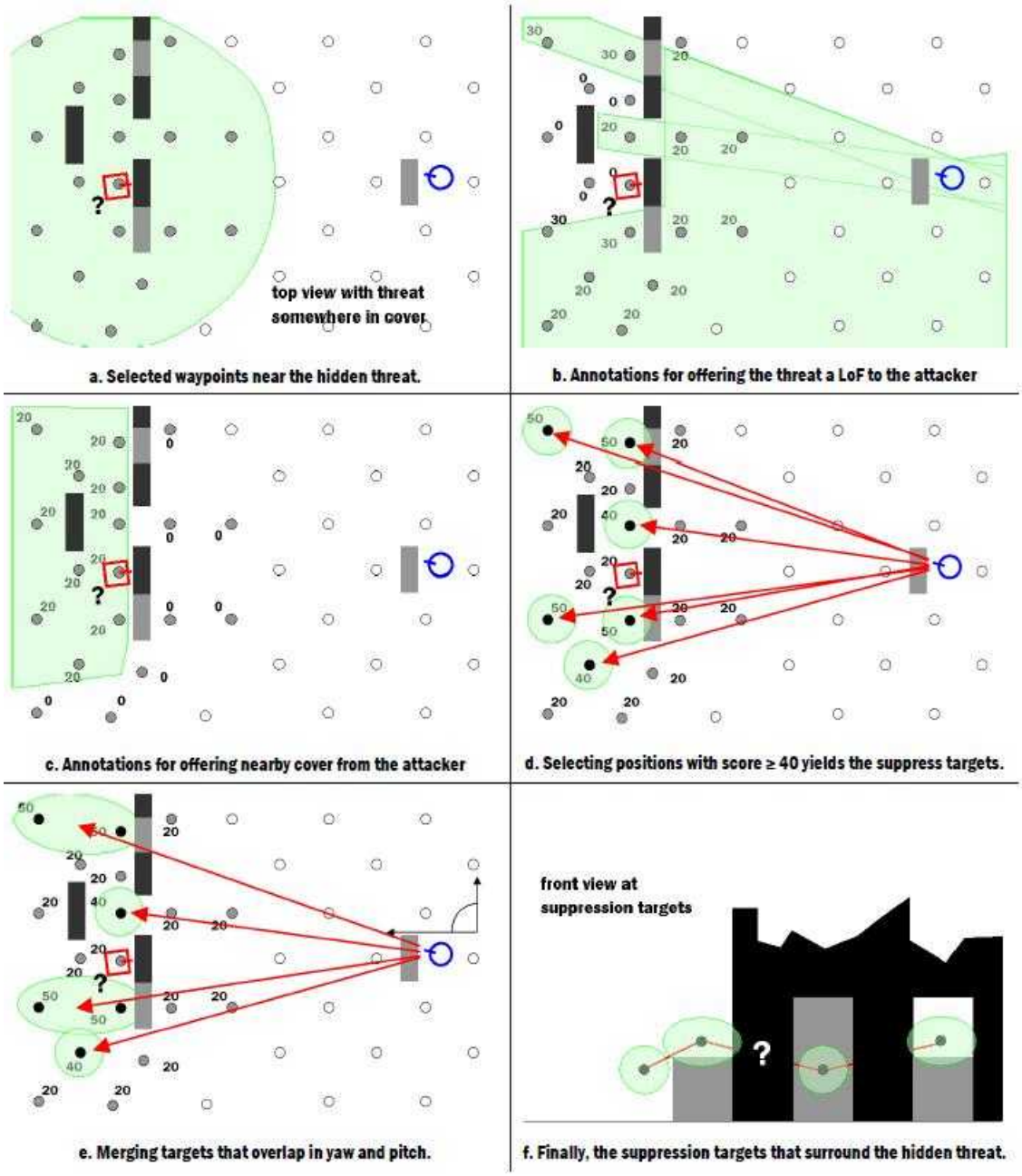
c.　ＣＯＭからの射線が通る領域。手榴弾が届くかどうかを評価する。

d.　a~ｃから敵に最もダメージを与えられる地点に手榴弾を投げ込む。
(手榴弾の軌道が通る計算は　ＣＯＭからこの位置に対して一度だけ ray cast 計算をで行う)

③　敵の移動を予測して威嚇射撃を行う

ここでは、ＣＯＭを制御するための「戦術的位置取りのシステム」を敵（プレイヤー、敵）に対して適用することで、敵が出現しそうな場所を予測する、ということをします。

図 13　ＣＯＭ（右）が隠れた敵（左）に対して威嚇射撃を行うことで、足止めを行う（[5] P.13）。



a.　敵の位置を中心に敵の移動候補領域を挙げる。特に評価はしない。

b.　COM から射線が通るポイントを評価する。

c.　敵にとって近くに障壁があるポイントを評価する。

d.　a~c を総合して評価する。

e.　COM から見て同じ角度方向にある候補領域をまとめる。

f.　ＣＯＭから見た威嚇射撃領域。

④ 敵の射線をかわしつつ移動する

ここでは「なるべく敵の射線に身をさらさないように移動する」パス検索を行う、という方法を説明します。Killzone では、パス検索に A*アルゴリズムを用いています。A* アルゴリズムを簡単に説明しますと、スタート地点から目標地点までが最小コストになるパスを検索する計算方法で、予想される計算量が最も少ない方法でゲームＡＩで最もよく使われるアルゴリズムです(文献[3]の解説が優れている)。普通、コストは、各ポイント間の距離を設定しますので、最短経路を導き出しますが、応用として、そのコストに付加的な情報を加えることで、知的なパス検索をしてくれます。例えば、水がある領域は速度が遅くなるので、水中にあるウエイポイントにはコストをプラスアルファしておくと、川を通るパスよりは橋を通るパスを導き出してくれる、ということが出来ます([6])。Killzone では下図のように敵の射線にさらされているウエイポイントにコストを課すことで、敵の射線にさらされないパスを優先的に検索します。

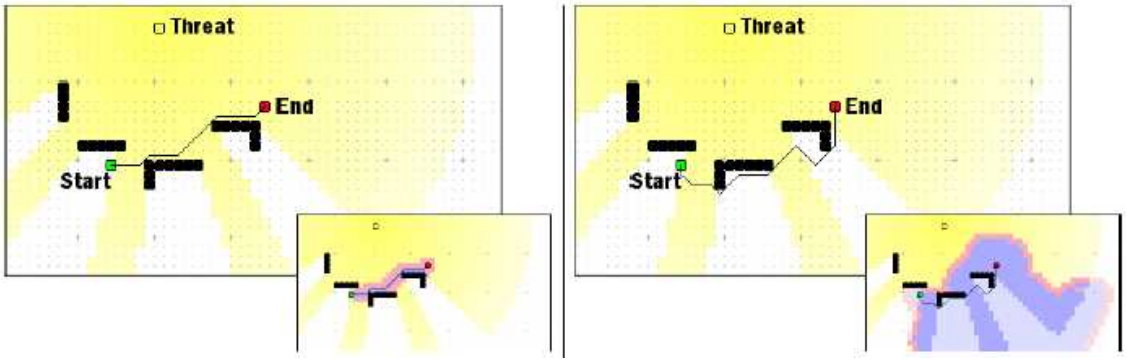


図 14　右が最短距離を導く普通の A*アルゴリズムによるパス検索。右が敵の射線が通るポイントにコストを足した場合の A*パス検索（[5] P.7）。

A* は、もともと計算量の少ないアルゴリズムですが Killzone ではさらに、先に見たように、ＡＩが次に動く領域を限定しているので、この方法がさらに有効に働きます。

第３節　まとめ

さて、以上が戦術的位置検出による動的な計算によるＡＩシステムによるＣＯＭの動作の説明です。
第１章では、位置に情報を埋め込むこと技術について説明し、第２章では、静的なマップにおいて狙撃ポイントのために位置検出について説明し、第３章では、実際のゲームに動的な状況に対する戦術的位置検出を解説しました。Killzone のＡＩシステムは、世界表現として、各ウエイポイントに８方向に対する可視距離のデータを事前計算して用意しました。このデータと簡単な不等式のある囲碁リズムによって、射線判定とカバー判定の計算負荷を圧倒的に軽くすることに成功し、この二つの判定を多用する戦術的位置検出のシステムをゲーム内で動作可能なシステムに作り上げました。また、射線判定をＣＯＭの思考のみならず、敵（プレイヤー、ＣＯＭ）に対して動作させることで、攻撃ポイントや敵の行動予測を行うという新しい機能を実現しました。また、工程という面から見ると、このシステムは、ロジックを埋め込むシステムと違って、同じシステムをそのまま他のステージやゲームでも動かすことができるという利点があります。

さて、ここで二つだけ注意点があります。

一つ目はこのシステムは、遮蔽物が破壊されるということに対して対応していません。Killzone では、殆ど全ての遮蔽となる物体は壊れません。これがこの手法の制限条件となります。この制限を外すためには、ゲーム中にリアルタイムにデータを更新できるなら、オブジェクトの破壊に対応出来ます。Killzone２では、そこにどう対応するのかが、注目点となります。

二つ目は精度の問題です。キャラクターの位置は最寄のウエイポイントと見なしているために、位置の精度がそれほどありません。しかし、位置の精度があったところで、キャラクターは移動するものなので、どんなにキャラクターの位置を正確にとって戦術を決めても、正しいとは限りません。むしろ、ある程度の位置のずれがあっても敵に対応できるＡＩシステムを組むことが大切です。このような考え方を ロバストなシステムと言います。例えば、飛行機は、安定であるように設計することが大切ですが、例えば機体の一部が壊れても、ある程度安定性を確保できるように設計する必要があります。同じように、ＡＩもある程度の情報がアバウトであれ、また、情報が欠損しても、行動を組み立てることができるシステムであるべきだ、という考え方がロバストな考え方です。Killzone もロバストなシステムを目指しています。

　具体的には、射線と遮蔽判定を取るアルゴリズムでは、射線が実際に通っていなくても、射線が通っていると判定する可能性があります。しかし、それは、キャラクターの移動に対してロバストなシステムになっています。キャラクターが少々移動して射線が通る場合に備えて、あらかじめ多くの点を射線が通るとしています。遮蔽という点から見ると、実際の遮蔽領域より小さくとることになり、結果として敵の移動に対してロバストなシステムとなります。

第４章　自分の開発するゲームのために

以上で、Killzone のＡＩの説明は終わりです。より詳しく知りたい方は原論文やＧＤＣのパワーポイントを読まれることをお薦めします([7],[8],[5],[9])。この章では、これまでに解説した事項を整理し、さらにそこから広がっている応用の方向を説明します。それは、学んだ人工知能技術を自分のゲーム開発に活かす過程を提示することが目的です。

第１節　ゲームＡＩのパターンを抽出する

この節では、上記で説明して来た事項をパターンにまとめると同時に、そのパターンの例として、他のゲームの応用例を示します。具体的に学んだことを抽象的なパターンとして整理しておくことは学習にとってたいへん有用ですし、また、そういったパターンを軸に逆にさまざまなタイトルのゲームＡＩを調査・研究することはもっと有用です。ソフトウエアのアーキテクチャーにとっては常識となりつつあるデザインパターン([10])の考え、或いはパターンランゲージの方法([11])は、ゲームＡＩにも応用することが出来ます。ゲームＡＩのパターンを整理するという試みは、私の知る限りまだ行われていませんが、おそらく数年後には研究が蓄積され立派な書物が書かれていることだと思います。日本の開発者もこのパターンの集積に貢献することを期待します。ここでは先取りして、今回説明した内容のまとめを兼ねて、**ゲームＡＩのパターン**を抽出してみたいと思います。パターンの説明には通常、適用範囲とか動機などの詳細を説明する必要がありますが、ここでは簡単な説明にとどめます。

パターン１

ＡＩに能力を持たせるには、たとえそれが同じ対象であっても、その問題の種類に応じた表現のデータを作成しなければならない。

(multiple representations for the same thing, because different representations are useful for different kinds of problems)（[4],P.20）

Killzone では、地形解析のデータを作成しました。地形データはもちろん、まずそれが描画されるためにあり、地形の性質の全てはそこ（或いは当りモデル）にありますが、ポリゴンやテクスチャーのデータそのものがＡＩにとって必要なデータではなく、そこから、様々な性質を目的に応じて抜き出すことが必要でした。特に、８方向に対する可視距離の世界表現は、Killzone のＡＩにとって中心的な役割を果たしました。のこのように、ＡＩにある行動をさせるためには、それに応じた世界表現が必要です。一般にＡＩの開発においても、知識表現、世界表現の問題は重要な問題です（[12]）。Halo2 のＡＩのチームは知識表現の問題に真剣に取り組み、プレイヤーへのアピールの強いユニークなＡＩのシステムを構築しました。例えば、Halo2 のＡＩはプレイヤーの攻撃に対して隠れながら撤退して行きます。([4],[13],[14])。

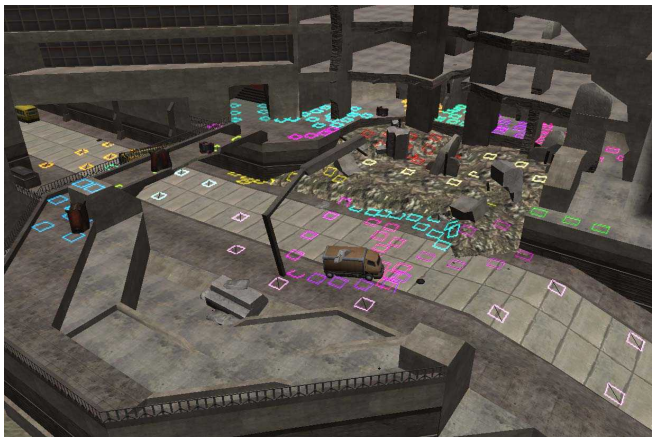



図 15　Halo2 のデバッグ画面。（左図）中央に一台の車があります。ＡＩにとって、この車のモデルの詳細が必要ではありません。「プレイヤーから身を守りつつ攻撃する」目的のためには、車の周囲のどの場所に立てばいいか、という表現情報が必要です。Halo2 では、レベルデザイナーが候補となる適切な戦術位置をグループ化して用意します（[4] P.15,22, [13] ,[14]）。

車のドアを開けて盾として使わせるためには、「ドアは動かすことができるものである」というフラグを車に対するＣＯＭのためにデータに持たせておく必要があり、また、「車を移動させ障壁として使いながらプレイヤーに近づく」ためには、「車を何処から押せばどの方向に進むか」という情報を与えておく必要があります。こういった「～ができる」という情報（アフォーダンスの情報と言います）は、物理エンジンの導入と共にＡＩにとってますます重要になって来ると思われます。実際、Halo2 のＡＩには、こういった情報が各オブジェクトにアフォーダンス情報が埋め込まれています（[4],[13],[14]）。例えば、（右図）の車には、どちらへ動かすことができるか、という情報がＣＯＭのために埋め込まれています。

パターン２

地形解析など計算量の多い解析は、開発中に事前にデータとして用意しておくことで、ゲーム中の負荷を軽減することが出来る。

これは Killzone のＡＩの方針そのものだと言ってよいと思います。さらに加えるなら、

パターン２－１

事前計算したデータとアルゴリズムを工夫することで、リアルタイム計算だけでは実現できない行動をＣＯＭに実装することが出来る。

この方法は、Killzone では事前計算に用いたオブジェクトが破壊されないことを前提にしています。もし、破壊される場合には、再計算を行う必要があります。こういった事前計算の方法はＡＩに限ったことではなく、グラフィックにおいても PRT など事前計算によってＧＰＵに負荷をかけないシェーディングの手法（[15]）が提案されています。Killzone の方法を参考にして、事前計算を応用したゲームとして Chrome Hounds(Xbox360, SEGA Corporation / FromNetworks, Inc. / FromSoftware, Inc., [6])があります。以下の図にあるように、クロムハウンズは、ナビゲーションメッシュを採用しており、各メッシュに対して、地形

情報をレベルマップから抽出して埋め込んでいます。

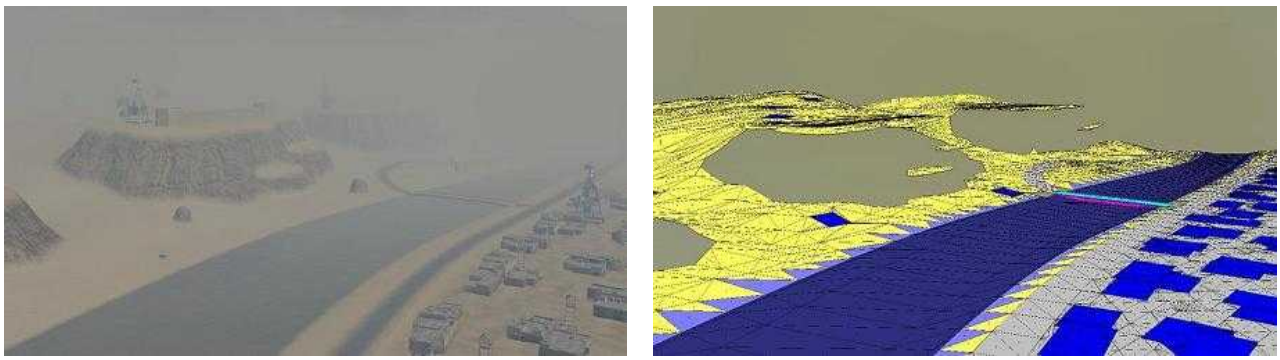


図 16　Chrome Hounds における地形モデル（左）とそこから自動的に作成したナビゲーションメッシュ。各、メッシュには、表面の情報（水、砂、道路）や、オブジェクトとしての性質（橋、橋の下）や、破壊状況などのフラグが埋め込まれている([6] P.90)。

パターン３

ＣＯＭのＡＩシステムを、プレイヤーに適用することでプレイヤーの動きを予測する。

Killzone において「威嚇射撃を行う」の思考を思い出してください。威嚇射撃では敵と自分の位置の両方から、敵が出現しそうで狙いをつけられるポイントを見出しました。これは、一般に、プレイヤーとＣＯＭが対等な立場にあるゲームにおいては応用することが出来ます。

当然のことですが、ＣＯＭに対して作ったＡＩのシステムには、ＣＯＭの状態を入力して行動を決定します。この入力を敵（プレイヤー、ＣＯＭ）の状態として出てくる行動の結果を、敵が取るだろう行動として測に使うことが出来ます。このパターンは、プレイヤーとＣＯＭが同等の条件で対戦するゲームにおいて有用です。またＡＩの優秀さを、プレイヤーを予測できる程度によって測ることができます。プレイヤーを予測できればできるほど、そのＣＯＭは人間らしいＡＩである言うこともでます。もちろん、これは完全なチート（本来ＣＯＭが得られない敵の情報を得ること）を前提にしているという欠点があります。

パターン４

キャラクター同士のインタラクションを場を介して行うことで、地形情報とキャラクターたちの関係から行動を決定する。

このパターンは見えにくいですが、一般に、キャラクター同士の認識には二つのモデル(Appendix 参照)があります。Killzone では、まず、キャラクターがマップ上の位置データとして捉えられ、そこで、事前に用意されたデータを参照して、戦術を決定して行きます。この方法は、インフルエンス・マップ（[16],[17]）の方法に近いですが、Killzone の論文の著者は、その違いを、「インフルエンスマップが一つの影響変数しか準備しないのに対して、多数のパラメーターを準備している」点、そして、「インフルエンス・マップがマップ全体に渡ってを常に影響を計算しているのに対し、局所的な戦闘に限定して計算している点」と説明しています。

パターン５

地形解析を行う場合に、マップモデルから行うのではなく、ウエイポイントを置いて、ウエイポイント間の間だけで解析を行い、必要な分解能で地形の性質を抽出する。

グラフィックのリアリティーはこの数年で飛躍的にアップしました。しかし、残念ながらＣＯＭのＡＩのリアリティーという意味では、全体的に見れば、それほど発展しているとは言えないのが状況です。もちろん、それはハードウエアの制限があり、ＡＩにリソースを当てられなかったことが原因の一つです。次世代機においては、ある程度のリソースを確保することが出来ます。

ＡＩのリアリティーの一つに、そのＡＩが、どれだけ周囲の状況を認識して動いているか、という問題があります。特に、次世代機になりマップが広くかつ複雑になった場合に、広範囲を自律的に動きまわることができるＮＰＣを作成するには、地形の性質をＡＩが使用できるデータとして用意する必要があります。データのベースとなるものは、Killzone ではウエイポイント、Jack and Daxter ([18]) , Counter Strike ([19]) , Chrome Hounds（[6]） では Navigation Mesh を用いています。そのベースとなる情報体の上に、このセミナーで解説したように情報を載せて行きます。Counter Strike では、ユーザーが作ったレベルデザインに自動的に Navigation Mesh を作成してくれるツールが存在します([20])。Killzone の特徴的な点は、ウエイポイント間同士の関係（視線が通るかどうか）を地形情報として引き出した点にあります。これは、通常の ray cast 計算より精度が落ちます。しかし、ＡＩにおいては、必ずしも厳密な精度の上にシステムを築くことが良いとは限りません。なぜなら、現実には、キャラクターは突然、いろいろな方向へ動くために、ある程度、位置がずれることを前提としてシステムを築いておく必要があります。こういった状況の揺らぎに対して安定なシステムをロバストなシステムと言います。

第２節　作業工程、そしてデバッグ

技術論文では、どうしてもその内容に目が行ってしまうのですが、開発者としては、その技術を実現するための作業工程（ワークフロー）とデバッグの工程を考えてみることが有用です。ここでは、この二つについて説明します。

第１項　　作業工程

作業工程を考えることで、その技術をより現実的なものに近づけることができます。そういった視点は開発者は常に持っているものです。しかし、論文によっては、そういった内容について説明していないものもあります。Killzone の論文もそういうことあまり親切に書いていませんので、少ない情報から推論するしかありません。

Killzone の工程でポイントになる点は、

- （１）　ウエイポイントは誰がどのように配置するのか？
- （２）　評価関数の中のパラメーターはどのように調整するのか？
- （３）　難易度はどのように調整しているのか？

です。まず、（1）は、レベルデザイナーが置いています。Killzone の開発においては、「レベルデザイナーが自分の設計したマップにＡＩを掘り込んだらすぐ賢く動く」ことを理想としました。Killzone のシステムはスクリプトの埋め込みのように特定のマップに依存しませんので、ある程度、その目標を達成することが出来た、と言えます。

（2）のＡＩの調整は、以下のデバッグの説明記述するように、テストマップで行っています。パラメーターとして調整できる要素は、

- 評価関数の中のウエイト
- 移動半径
- 攻撃半径

などです。どれもＣＯＭの判断に影響します。ウエイトは直接評価値に影響します。移動半径は、評価候補点を絞るのに使いますので、これが小さいと遠くまでのエリアを考慮に入れないＣＯＭになります。逆に大きいと遠くのポイントまで評価しますが計算量が多くなります。また、Killzone の行動の単位は、だいたい５秒から１５秒です。それ以上長いと状況が変わってしまいます。そこで、それに応じた移動半径を取ります。攻撃半径は、持っている武器によって変化しています。実際、３Ｄシューティングでは、持っている武器によって移動位置を決定する思考が要求されます。この問題は二つあって、

（1） 今持っている武器に最適な攻撃距離に近付く
（2） 持っている武器の中で、それを使えば最も効果的にダメージを与えられる武器を選択して、それに適した位置に移動する。

Killzone はおそらく（1）であると思われます。（2）はは例えば、射程の長いスペシャルウエポンを持っているなら、敵から遠ざかって撃つ方が効果的であり、頭上から投げるタイプなら敵の頭を取れる位置へ移動する必要があります。Chrome Hounds　では、４つの武器を同時に持つことが出来、敵との間合いから最適な武器を評価関数によって選択します。また、逆に、ある特殊武器を使うためにそれに最適な場所に移動するようにＡＩが作られています。

第２項　　デバッグ

新しい技術には、それに対応したデバッグ方法を対として用意する必要があります。新しい技術は、新しい力をゲーム空間にもたらすと同時に、それまでになかった現象を引き起こす可能性があります。実際、ＡＩの技術はＣＯＭに新しい自由度を与えてくれます。そこでデバッグによって、新しい技術を実際のゲームに導入可能であるレベルにあることを判定し保証する必要があります。

最も単純な例としてパス検索を考えてみます。固定パスであった場合には、用意したパスにキャラクターを移動させてみることでデバッグを行うことが出来ます。ところが、パス検索機能は、キャラクターがマップの全領域を自在に移動することを可能にします。ここには二つの問題があります。一つはパス検索でパス

がみつかるということと、移動できることは異なるということです。実際は、オブジェクトの端に引っ掛かったり、わずかな傾きで崖の下に落ちて行く可能性があるかもしれない。二つ目は、パスの数はマップの居スケールによっては事実上無限であることです。例えば、Killzone のあるステージでは、２０００のウエイポイントがあります。そこで全てのパスをチェックすることは出来ません。多くパス検索のデバッグには工夫が必要です。

Killzone では、キャラクターが崖から落ちることができないように設定されています。見えない壁が設定されています。しかし、オブジェクトに引っ掛った場合は、最初からデバッグによって完全に取り除かれているのか、或いは、引っ掛かったという状態を検出して対応しているのか、何れかわかりません。

Chrome Hounds では、完全なデバッグは諦めて、

（１） 主要なルートのパスを通行可能かどうか検証する（３５００回のチェック）。

（２） 崖や壁に隣接するメッシュにフラグをつけて、パス検索ではなるべく崖や壁の側に近寄らないパスを検索する。

（３） 一定時間内に目的地に到達しない場合、或いは、突然加速度がかかり落下したと判定された場合は、パスの再検索を行う

とアルゴリズムを重ねることで対応しています。

また、ＡＩの調整、デバッグを考えてみます。ＡＩは特にゲームを開発しながら調整し、或いは発展する対象でありますので、人工知能を作る足場としてデバッグ画面を工夫しておくことは効率的です([21])。第２章でパラメーターをサーモグラフィーのようにビジュアル化した表現は、異常な評価値を検出することが出来ます。Killzone では、評価値を下図のようなにラメーターをウエイポイントに明示した図を利用しています。また、以下のようにＡＩのデバッグ、調整専用のテストステージが用意されています。

図 17　評価値を表示したデバッグ画面（左）とＡＩの調整とデバッグのための画面（右）([5] P.1,14)。

第３節 まとめ

この章では、第１節で Killzone で紹介した技術を、パターンとしてまとめると同時に、類似した他のタイトル応用例を紹介しました。第２節では工程とデバッグ環境について紹介しました。ＡＩの作成のためのパターンは、今後も多くの開発、研究の成果をまとめる形式として優れており、一つのタイトルの技術的成果を

抽出し、次のタイトルへ橋渡しをする役割をします。また、異なるタイトル間で共通に使われている技術を、技術からまとめることを可能にし、技術を集積する方法として優れています。

第５章　展望

このテキストでは、まず第１章で知識表現について解説し、その中でも世界のグローバルな知識表現として世界表現、特に位置に依存した世界表現の方法を解説しました。そして、第２、３章ではその方法が、Killzone 及び、その先行研究において利用され、実際のゲームＡＩを高度に構造化していることを見ました。第４章では、そういったゲームタイトルの実装方法の知識を自分の開発に活かすために、より一般的にパターンによって技術を抽出し、他のタイトルの応用事例を見ました。

人工知能技術は時間と空間に渡る抽象的な技術であり、他人の方法をそのまま自分の開発へ活かせるケースは多くありません。むしろ、そこから抽象した知識や思考を渡って変形や応用を施しながら自分のゲームへ導入するように再構成するということが必要です。世界表現の方法も、ゲーム設計から来る要求に応じて様々な形を取り、時には他人の表現の仕方を参考にし、時には自分で全く新しい方法を見出す必要があります。それは一つの課題であると同時に大きなチャンスです。是非、様々な表現を学習し、発想し、できれば自分のアイデアを他の開発者に伝えることで、ゲームＡＩを発展させて頂きたいと思います。

また、今日のゲーム開発は１，２年から４、５年という期間を要しますので、ゲームＡＩを設計する機会をそれほど多く持つことはできませんが、ディスカッションではゲームＡＩについて他の開発者と議論することで模擬的にそういったゲームＡＩの設計を体験、演習できます。筆者は会社においてそのようなセミナーを週に１回開いて、これまでに６０回ほど行って来ました。セミナーにとって大切なのは、ゲームＡＩの技術の理解も大切ですが、企画者と技術者が同じテーブルを囲んでアイデアを話合うことです。技術者は企画者が想像もつかなかったことが実現できることを示すこともあり、また、企画者が技術者にゲームデザインのアイデアから面白いテーマを与えることも出来ます。そして、企画者はその情報をもとにゲームアイデアを発展させ、技術者はそこで得たテーマをもとにまた、技術的な可能な領野を探索します。私にはその相互のコミュニケーションの中から新しいゲームの発想が生まれるような気がするのです。ゲームＡＩがゲームを発展させ、ゲームがゲームＡＩを発展させ、共進化して行く、企画者が技術者を成長させ、技術者が企画者を成長させる、少なくとも自分が関わった開発で私はそれを肌で感じました。このセミナー後に、社内でそういったコミュニケーションをして頂ける土壌を作ることが、このセミナーの目的の一つでもあります。

第１回目のテキストはここで終わりです。広大なゲームＡＩのフィールドの中の一つのの領野を渡りました。第２回は、時間をテーマにプランニングの技術について解説します。第１回に対する感想や批判などは、メイルやアンケートを通して三宅までお知らせ頂いて、第２回のセミナーに活かしたいと思います。

２００６年１２月

yoichi-m@pk9.so-net.ne

.jp

Appendix　プレイヤーの認識について

「ＣＯＭが敵のプレイヤーを認識する」というテーマについて、まず一般的に解説します。ゲームにおけるＡＩの賢さというものは、「如何にキャラクター（プレイヤー、他のＣＯＭ）の情報とゲーム世界（マップ）の情報を表現し取り込み、その情報を反映した意思決定を行いどうＣＯＭを動かすか」に依存します。一般に、ＣＯＭがプレイヤーを認識する方法は二つのあります。

- ＣＯＭが持つ感覚によってプレイヤーを認識する。
- 場を介して認識する。

前者は、エージェントモデル([12])であり、視覚であれば、視線が通っているか、聴覚ならば、プレイヤーの立てた音がＣＯＭまで聞こえる距離か、後ろ向きにぶつかったとか、香水の匂いがしたなど、五感の情報によって敵を認識します。

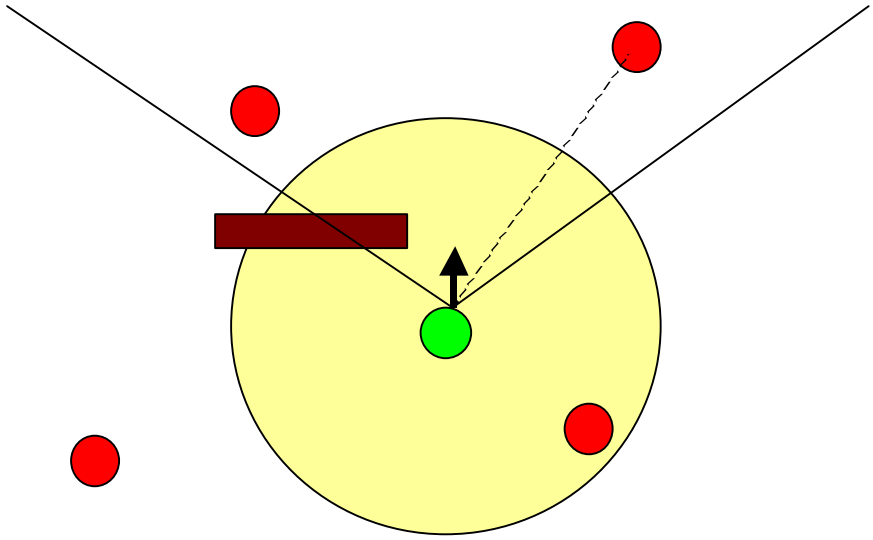


図 18　感覚によってプレイヤーを認識する。視覚による認識なら、ＣＯＭの視野内で視線が通っていると判定できた場合、プレイヤーを「認識」する。緑がＣＯＭを現し、赤が敵を現す。２本の線に挟まれた領域が視野角であり、その中で視線が通った敵を「認識」することが出来る。黄色い円は、聴覚で位置を特定できる範囲を表し、この中で敵が音を立てると「認識」することが出来る。ＣＯＭとプレイヤーが直接インタラクションしている。

後者は、キャラクターの情報をウエイポイントやメッシュに反映させ、その情報から他のキャラクターが動作を決定します。例えば、ＡＩのために、以下の図のようなセルの連なったマップを用意します。各セルに対して、「危険度」を設定します。例えば、プレイヤーがいるセルは危険度が高く、そこから遠ざかるに従って危険度を現象させるとします。すると、ＣＯＭはこの情報を使って危険度の低いパスを検索することが出来ます。このような方法を、**インフルエンス・マップ（Influence map）の方法**と言います。ゲームＡＩでは、ＦＳＭについでよく使われる方法です。（教科書[16],[17]にはインフルエンス・マップの様々な応用方法が解説されています）。

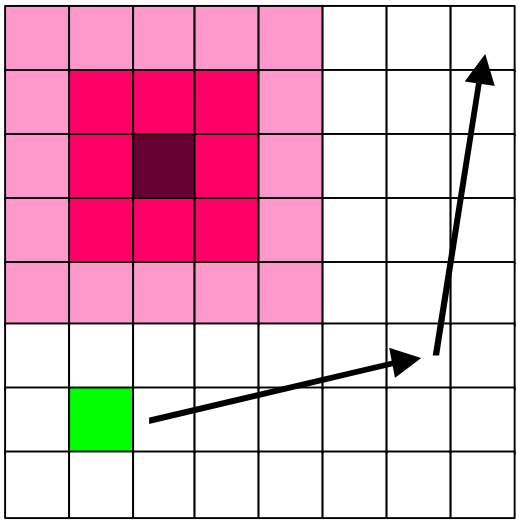


図 19　インフルエンス・マップ。左上の最も赤いセルが敵がいるポイントで、危険度が最も高い。そこから、遠ざかるに従って、危険度が減少して行く。緑がＣＯＭにのいる地点で、右上まで行きたいので、敵の脅威を避けてパス検索をして移動する。インフルエンス・マップを通して相互にインタラクションしている。

参考文献

[1] **Borut Pfeifer** , "Automatic Cover Finding with Navigation Meshes", *Game Programming Gems*, **5**, 299.

[2] **Christopher Reed, Benjamin Geisler** , "Jumping, Climbing, and Tactical Reasoning: How to Get More Out of a Navigation System", *AI Game Programming Wisdom*, **2**, 141. (2003)

[3] **Mat Buckland** , "Programming Game AI By Example", *WORDWARE Publishing*, (2005)

[4] **Damian Isla**,"Dude, where's my Warthog? From Pathfinding to General Spatial Competence", *http://www.aiide.org/aiide2005/talks/isla.ppt* (2005)

[5] **Remco Straatman, Arjen Beij, William van der Sterren**,"Killzone's AI : Dynamic Procedural Combat Tactics", *http://www.cgf-ai.com/docs/straatman_remco_killzone_ai.pdf* (2005)

[6] **三宅 陽一郎** , "クロムハウンズにおける人工知能開発から見るゲームAIの展望（ＣＥＤＥＣ２００６開催後３ヶ月はＣＥＤＥＣ２００６ＷＥＢページで公開。それ以降は y_miyake@fromsoftware.co.jp までお知らせください）", *CEDEC2006*, (2006)

[7] **William van der Sterren**,"Terrain Reasoning for 3D Action Games", *http://www.cgf-ai.com/docs/gdc2001_paper.pdf*

[8] **William van der Sterren**,"Terrain Reasoning for 3D Action Games(GDC2001 PPT)", *http://www.cgf-ai.com/docs/gdc2001_slides.pdf*

[9] **Arjen Beij, William van der Sterren**,"Killzone's AI : Dynamic Procedural Combat Tactics (GDC2005)", *http://www.cgf-ai.com/docs/killzone_ai_gdc2005_slides.pdf*

[10] **エリック ガンマ ,ラルフ ジョンソン , リチャード ヘルム , ジョン ブリシディース** , "オブジェクト指向における再利用のためのデザインパターン", *ソフトバンクパブリッシング*, (1999)

[11] **C.アレグザンダー** , "オレゴン大学の実験", *鹿島出版会*,

[12] **スチュワート ラッセル ,ピーター ノーヴィグ** , "エージェントアプローチ 人工知能", *共立出版*, (1995)

[13] **Chris Butcher,Jaime Griesemer**,"The Illusion of Intelligence The Integration of AI and Level Design in Halo", *http://www.gamasutra.com/features/gdcarchive/2002/jaime_griesemer.ppt*

[14] **GDC 2005 Proceedings**,"Handling Complexity in the Halo 2 AI", *http://www.gamasutra.com/gdc2005/features/20050311/isla_01.shtml*

[15] **Yoshiharu Gotanda**,"Game programmer no tame no syokyu PRT (Introduction of PRT for game programmers)", *http://research.tri-ace.com/*

[16] **Brian Schwab** , "AI Game Engine Programming", *River Media. CHARLES RIVER MEDIA*, (2004)

[17] **Ian Millington** , "Artificial Intelligence For Games", *Morgan Kaufmann Series in Interactive 3d Technology*, (2006)

[18] **Stephen White & Christopher Christensen** , "高速なナビゲーションメッシュ", *Game Programming Gems（日本語版）*, **3**, 306.

[19] **Michael Booth** , "The Making of the Official Counter-Strike Bot", *GDC2004*,

[20] "Bot Navigation for Counter-Strike:Source:jp", *http://developer.valvesoftware.com/wiki/Bot_Navigation_for_Counter-Strike:Source:jp*

[21] **Paul Tozour** , "Building an AI Diagnostic Toolset", *AI Game Programming Wisdom*, **1**, 39.