2014年度　卒　業　論　文

エージェント型AIの経路探索における グラフ構築の処理軽減手法に関する研究

指導教員：渡辺 大地 講師
三上 浩司 准教授

メディア学部 ゲームサイエンス ゲームイノベーション プロジェクト
学籍番号　M0111428
門馬　翔

2014 年度　　卒　業　論　文　概　要

論文題目

エージェント型 AI の経路探索における グラフ構築の処理軽減手法に関する研究

メディア学部 **学籍番号：M0111428**	**氏名**	門馬　翔	**指導教員**	渡辺 大地 講師 三上 浩司 准教授

キーワード	経路探索、ゲームエンジン、Unity、 グラフ構造、エージェント型 AI

ビデオゲームにおいて、AI が操作するキャラクタは、ゲーム体験をするプレイヤにとって重要なものである。近年、ゲーム AI で利用され始めてきたエージェント型 AI では、経路探索、意思決定ロジックが重要となっている。特に、経路探索の分野において、オブジェクトがマップ上のある地点からある地点まで障害物に接触しないように移動するための経路を自動的に発見することが重要である。そのため、地形を AI に判別させ、マップをグラフに落とし込む必要がある。グラフを用いることで、AI がマップ上で通るべき経路が明確化され、AI の経路を制御することができる。この時に生成するマップグラフは、より精密で、より速く処理できることが望まれている。現在主流となっているゲームエンジンの Unity では、メッシュグラフの作成をサポートしている。選択したメッシュに応じたグラフを自動生成するといった機能があるが、これは処理負荷的にも軽いものではなく、生成されたメッシュを別のデータとして保存するため、マップグラフを作成する上では、比較的重い処理になる。さらに、細かい場面での手直しをしにくいという欠点がある。そこで、本研究では、エージェント型 AI がマップ内を移動する際に参照するマップグラフに着目した。Unity の NavMesh よりも生成速度が高速であり、簡単に作成でき、データ量が少ないマップグラフの生成を目的とした。本研究ではメッシュで構成されたグラフではなく、ノードという点で構成されたグラフの生成を実現した。

目 次

図 目 次

表 目 次

第 1 章

はじめに

1.1 研究の背景と目的

近年、ゲーム AI の発展に伴い、様々な仕組みの AI がビデオゲームに応用され、独自の進化をしてきた。ゲーム AI は学術的な分野でも近年盛んに取り上げられており、Julian ら [1][2] は、スーパーマリオワールド [3] のシステムを用い、AI によるキャラクターの人工知能プログラムでスコアを競う、AI コンペディションも行われている。AI を主役としたゲームも存在し、アストロノーカ [4] や、頑張れ森川君二号 [5] といったタイトルが挙げられる。さらに、MIT メディアラボ（MIT Media Lab）では、デジタル上で生物の自律的な知性をスケーラブルに表現する手法 [6] や、デジタルで生きるペットの知性を表現した C4 アーキテクチャ [7] の研究が行われている。このように、AI はビデオゲームにおいてなくてはならない存在である。特に、エージェント型 AI と呼ばれる AI は、高度な知性を表現できるシステムとされ、ハイクオリティな AI を必要とするファーストパーソン・シューティングやリアルタイムストラテジーといったジャンルのゲームにおいてよく利用されている。

三宅 [8] はエージェント型 AI を、次の 4 つの要素を満たした AI であると定義している。

1. 体を持つ（エフェクターを持つ）

2. センサーを持つ

3. 役割・目標を持つ

4. 自身で行動を決定できる

1は、ゲームに登場するキャラクターとしての体を持つということである。つまり、ゲーム世界に干渉できる存在であるかということである。2は、AIキャラクター自身が環境に対して情報を集める能力を持っているかということである。例えば、視界を持っているAIキャラクターが、視界に入ったものをAIキャラクター自身だけで判別できるかということである。3は、AIキャラクターが明確な役割や目標を持っているかということで、プレイヤーを倒す等の、AIキャラクターとしての立ち位置が決まっているかどうかということである。最後に、4は意思決定能力があるかどうかである。意思決定ができるAIキャラクターというのは、状況に応じた経路を通り、状況に応じた行動ができるAIキャラクターのことを指す。

AIキャラクターが存在し、複雑な地形を舞台とするコンテンツの作成は非常に手間がかかる作業である。事前にAIキャラクターが移動できる経路をパスや座標で指定したり、AIキャラクターごとに移動できる経路を設定する必要がある。このような作業を軽減するために、AIキャラクターがマップ上である地点からある地点まで障害物に接触しないように移動するための経路を自動的に発見できることが重要である。そして、地形をAIキャラクターに認識させるために、マップをグラフにする必要がある。本研究で扱うグラフというのは、グラフ理論[9]に基づくグラフのことである。マップをグラフ化してAIキャラクターに与えることにより、AIの移動場所を制御することが可能となる。

これまでにも障害物を回避しながら地形上のある位置から別の位置まで移動する経路を探すことができる経路プランニングに関する研究はあった。原川[10]は、ジャンプを考慮した経路探索の手法を提案した。Linら[11]は、ボロノイ図を利用した三次元での経路探索の手法を提案している。Hartら[12]、Lozano-Prez[13]は、パス検索による経路探索の手法を提案した。森江[14]、中部[15]による、集団

で移動する際の経路探索の研究もあった。さらに、三宅 [16] は様々なエージェント型 AI を実現する手法を紹介しており、経路探索においては「世界表現」というキーワードの下、マップグラフの制作工程や、経路プランニング、AI の意思決定の手法を紹介 [17][18][19] している。近年も三宅 [20] は論文を発表しており、現在、ゲーム分野において応用されている人工知能の技術を紹介している。

本研究では、エージェント型 AI がマップ内を移動する際に参照するマップグラフに着目した。既存のマップグラフ制作手法よりも生成速度が高速であり、簡単に作成でき、データ量が少ないマップグラフの生成を、現在主流となっているゲームエンジンの Unity[21] 上で実現することを目的とした。Unity でも、標準機能でメッシュグラフの作成をサポートしている。メッシュグラフとは、マップを表すメッシュによるデータ構造から成るグラフである。しかし、これは処理負荷的にも軽いものではなく、生成したメッシュグラフをマップのメッシュとは別のデータとして保存するため、マップグラフを作成する上では、比較的重い処理になる。さらに、細かい場面での手直しをしにくいという欠点がある。そこで、本研究ではメッシュグラフではなく、ノードという、独立した点で構成したグラフの生成を実現した。生成したグラフは、Unity で生成するメッシュグラフよりもデータ量は軽量に抑えられ、処理負荷も軽いものとなっている。さらに、手直しする際にはノードを好きな位置に動かすことができるため、より直感的に思い描いたグラフを生成できるといった強みを持つことができる。検証では、Unity の機能による NavMesh[22] と提案手法を評価し、パフォーマンス、手直しのしやすさ、グラフの質の三項目にわたって比較した。その結果、パフォーマンスの面では、NavMesh は小さいマップでの精密な経路探索に有効であることがわかったが、大きなマップに対しては、データ容量、グラフ生成速度が著しく悪くなった。対して、提案手法では小さいマップでの経路探索では、差は開かなかったが、大きいマップでは飛び切り効果を発揮した。手直しのしやすさでは、NavMesh より提案手法の方が優位であると分かった。グラフの質では、提案手法の方が直線的ではあるが、ほぼ想定通りのルートを辿ることができた。

1.2　本論文の構成

本論文は、本章を含め全4章で構成する。第2章では、提案手法、アルゴリズムと、実際にマップグラフを創るための手順を述べ、第3章では実際に本論文で述べた手法でさまざまなマップのグラフを生成し、それぞれの評価を行う。最後に第4章にて研究全体の総括を述べる。

第 2 章

マップグラフの作成

本章では、提案したアルゴリズムから、マップグラフを生成する手順について述べる。まず提案したアルゴリズムについて述べ、その後、実際に制作したマップグラフの利用方法について述べる。本研究におけるマップグラフとは、AI キャラクターがマップ上で通れる経路を示した、グラフ理論に基づくグラフのことである。

2.1 アルゴリズムの概要

現在 Unity で使われている経路探索システムでは、描画マップから NavMesh の機能で、経路探索用のメッシュを生成する。ここで生成されたメッシュはナビゲーションメッシュと呼ぶ。モバイルによるゲーム制作も考慮している Unity にとってこの方法ではデータ容量が膨れ上がってしまう可能性がある。それは描画マップとは別にメッシュのデータを持つことが原因で、そのせいで計算コストが増えてしまうことも懸念点である。さらに、ナビゲーションメッシュは Unity 上のオブジェクト単位でしか設定できないため、扱いにくいものとなっている。例えば、オブジェクトを分割していない状態のマップが与えられ、絶対に AI キャラクターが通れない場所はナビゲーションメッシュ自体を生成しないようにしたい場合、マップ自体を分割する必要があり、手順が煩雑になってしまう。そこで、本研究ではモ

バイルにおけるマップグラフ制作も考慮し、より軽量で手直しやコスト計算が簡単にできるマップグラフによる手法を提案する。

本研究で提案するアルゴリズムでは、ノードを用いる。あらかじめマップ上で移動できると決定し、ノードを格子状に埋める。そして、ノードとノードを結ぶようにしてマップグラフの生成をする。提案するアルゴリズムは、あらかじめレベルデザイン等で決定したマップデザインがあることが前提となる。本研究でマップと定義するものは、AI キャラクターが通れる地形としての役割を持ったメッシュを持つオブジェクトのことである。

2.2 アルゴリズムの内容

初めに、ノードを置きたい領域、高さ、ノード一つ一つの間隔を決め、ユーザー入力により指定する。領域は、マップ全体を覆うような長方形領域を設定する。以下の図 2.1 は、マップを真上から見た時の領域指定のイメージである。

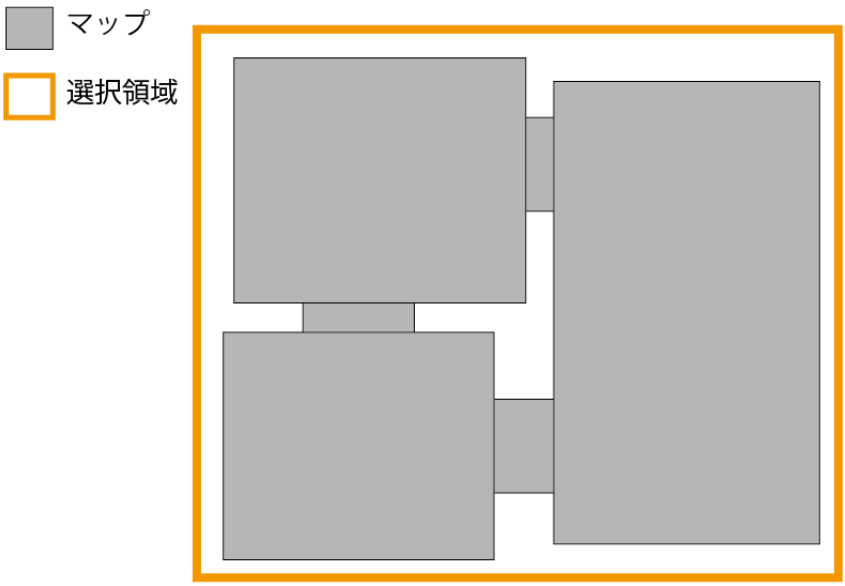

図 2.1: 領域を指定した状態

次に、マップ上にユーザー入力によって決定した高さにノードの仮設置を行う。

この時点でノードがマップに設置できるかを判断する。ノード設置の判断基準は、ノードの下にマップが存在しているかどうかである。下にマップが存在しているかどうかは、それぞれのノードから真下にレイを飛ばし、レイがヒットしたか否かで判断する。レイがヒットしなかった場合、その座標のノードは削除する。以下の図 2.2 は、指定した範囲にノードが設置された様子を真上から見た図である。図 2.3 は、ノードがどのような状況なら設置できるかを示したイメージである。青色の丸がノード、黄色の線がレイキャストである。

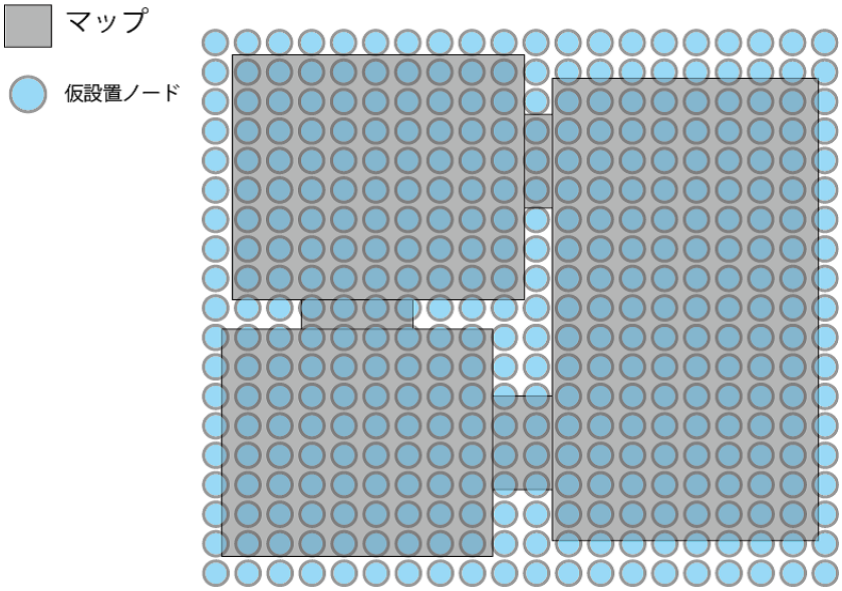

図 2.2: ノードが仮設置された様子

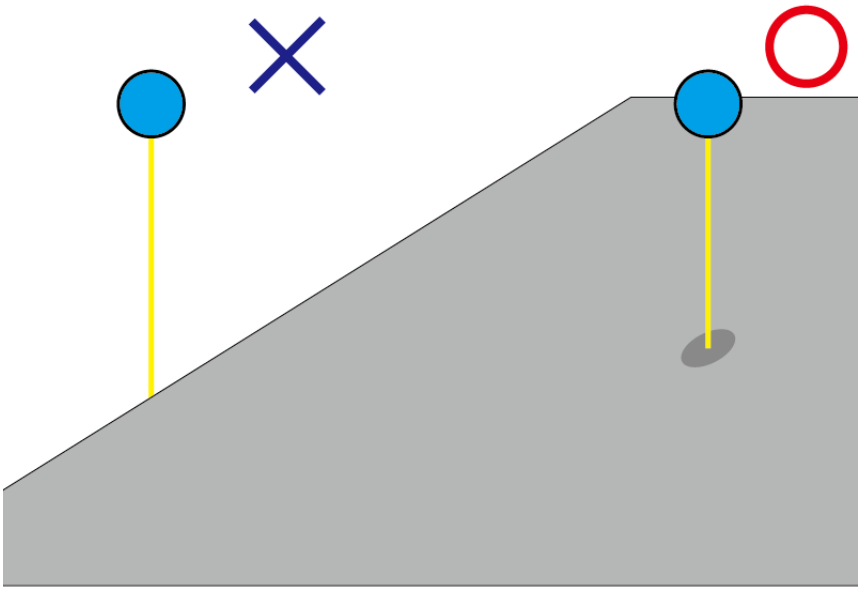

図 2.3: ノードが設置の判断をするイメージ

設置できると判断したノードは、真下に飛ばしたレイとメッシュの交点に設置する。この処理を、指定された領域内の一定間隔に設置されたすべてのノードに対して行う。

2.3 グラフの利用方法

上記アルゴリズムによってできたマップグラフを基に、AI キャラクターは経路探索を行う。ノードによるマップグラフなので、A*アルゴリズムや、ダイクストラ法による探索ができる。さらに、ノードは Unity のシーン上にゲームオブジェクトとして配置されているため手直しが可能であり、スプリクトによって、動的にノードの移動や消去ができる。以下の図 2.4 は、ノードでマップがうめつくされた状態である。これをノードグラフと呼ぶ。

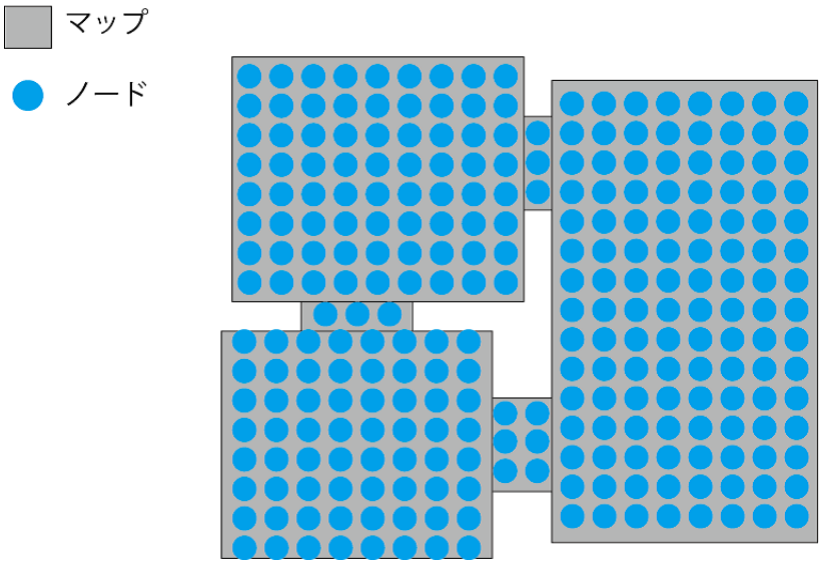

図 2.4: ノードでマップが埋め尽くされた状態

ノード同士をつなぐと、無向グラフ [23] ができる。この無向グラフを A*等で探索することで、AI の経路探索ができるようになる。本研究では、Unity のアセットで配信されている、A* Pathfinding Project[24] を使って、無向グラフ作成、A* による経路探索を実現した。図 2.5 は、ノードグラフを無向グラフにした状態である。

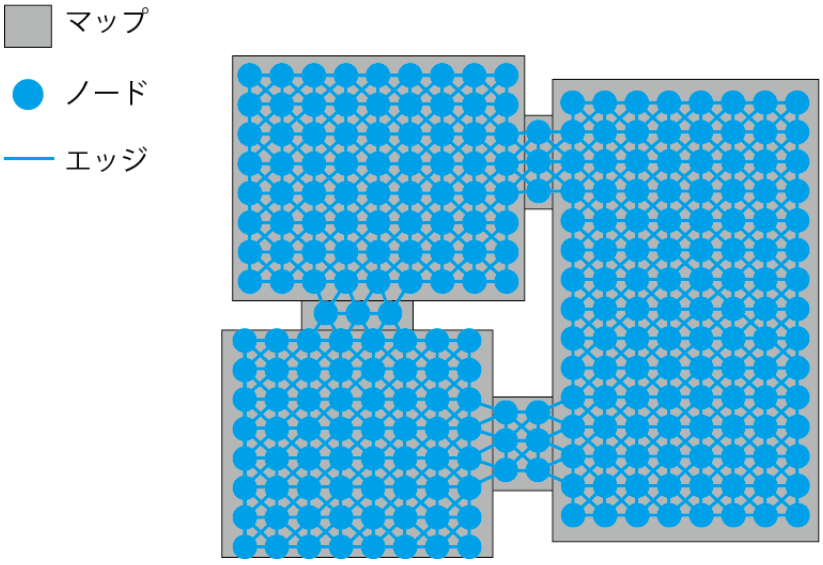

図 2.5: ノードグラフを無向グラフにした状態

第 3 章

検証と考察

3.1 検証の概要

本章では、第 2 章で述べたアルゴリズムによるマップグラフ作成方法が目的を達成しているかを検証し、評価する。検証に使った環境については、表 3.1 に示す。

表 3.1: 検証を行う環境

OS	Windows7 HomePremium
CPU	Intel Core i7-3630QM @2.40GHz
メモリ	16GB
グラフィック	nVIDIA GeForce GT 650M

本実験の検証は、Unity によって制作した 4 つのマップテンプレートに対して行った。このマップテンプレートは、Y 軸を高さとした、XZ 平面を基準に作られている。高低差のあるマップでは、視認性を高めるため高さが違うフロアの色を青色に設定し、障害物は緑色に設定した。Unity 標準の NavMesh、提案手法、それぞれでマップグラフを制作し出来上がった全 8 マップで検証を行う。検証では、Game ビューにはマップグラフを描画しないものとした。完成したマップグラフについて検証する項目は以下の 3 点である。

1. パフォーマンス

- 設定や生成時での処理時間やデータ容量
- 実行時での処理時間と利用メモリ量

2. 設定の容易さ

3. 実用時での質 (エージェントの動きの適切さ)

パフォーマンスの検証では、第一に、設定や生成時での処理時間、データ量をUnityでの処理時間や、実際のデータ容量から計測した。第二に、実行時の処理速度と利用メモリ量を、Unityでのビルド時間と利用メモリ量から計測した。設定の容易さの検証ではマップグラフの作り直しを想定して、実際にマップグラフを作り直し、その時間を計測して検証した。実用時での質の検証では、あらかじめ決めたルートを、AIがどれだけ正しく辿ることができるかを検証した。

3.2 パフォーマンスの検証

この節では、マップグラフ生成時とプログラム実行時のパフォーマンスを計測結果を述べる。マップグラフ生成時には、Unity上での処理時間とデータ量を計測し、プログラム実行時には、Unityの再生ボタンをおしてから、AIが経路探索を始めるまでの、Unity上での処理が開始するまでの時間と利用メモリ量を計測した。マップの情報として、Unity上で計測したマップの広さを、Unity上の単位を基準としXの長さ$\times$$Z$の長さ として記載した。さらに、描画マップの頂点数、三角ポリゴンの数も記載するものとした。

3.2.1 マップ1

マップ1は、面ポリゴン1つのみのマップである。図3.1,3.2は、マップ1を見下ろした状態と、マップ1の一部を写した図である。マップ1の広さは10×10、描画マップの頂点数は189、三角ポリゴンの数は232である。マップグラフ生成時の検証結果を表3.2に、プログラム実行時の検証結果を表3.3に示す。

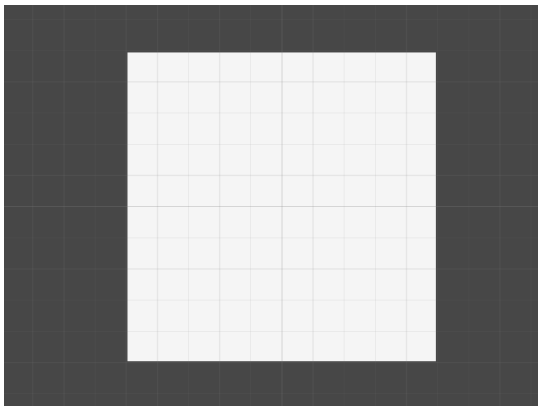

図 3.1: マップ 1 を見下ろした状態

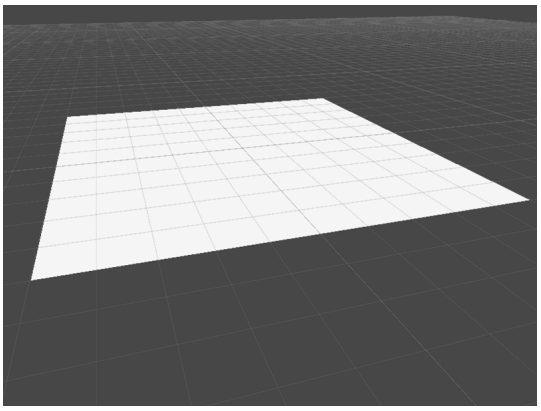

図 3.2: マップ 1 の一部を写した状態

表 3.2: マップ 1 でのグラフ生成時の検証結果

	マップグラフの容量	グラフ制作までの所要時間
NavMesh	4.37KB	0.4 秒
提案手法	64.5KB	1.15 秒

表 3.3: マップ 1 での実行時の検証結果

	処理開始までの時間	利用メモリ量
NavMesh	1.51 秒	456B
提案手法	1.15 秒	9.6KB

3.2.2 マップ 2

マップ 2 は、面ポリゴンを組み合わせて作ったマップである。図 3.3,3.4 は、マップ 2 を見下ろした状態と、マップ 2 の一部を写した図である。マップ 2 の広さは 40 × 40、描画マップの頂点数は 552、三角ポリゴンの数は 832 である。マップグラフ生成時の検証結果を表 3.4 に、プログラム実行時の検証結果を表 3.5 に示す。

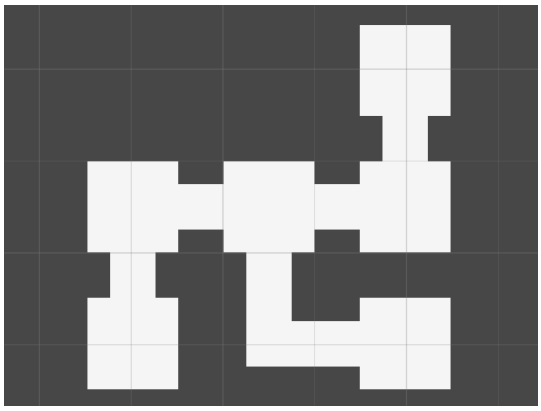

図 3.3: マップ 2 を見下ろした状態

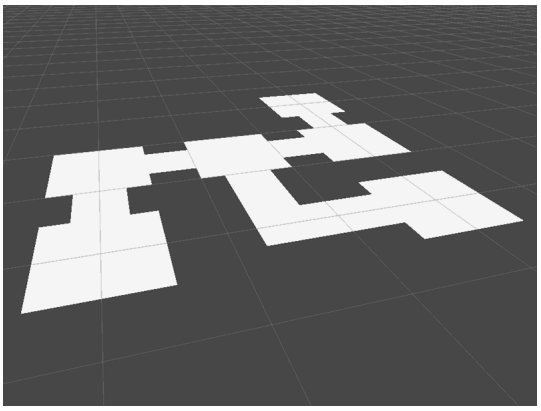

図 3.4: マップ 2 の一部を写した状態

表 3.4: マップ 2 でのグラフ生成時の検証結果

	マップグラフの容量	グラフ制作までの所要時間
NavMesh	10.6KB	0.6 秒
提案手法	85.4KB	1.61 秒

表 3.5: マップ 2 での実行時の検証結果

	処理開始までの時間	利用メモリ量
NavMesh	1.3 秒	6.8KB
提案手法	1.61 秒	9.6KB

3.2.3 マップ 3

マップ 3 は、円状のポリゴンも含めて制作し、さらにトンネル状に起伏があるマップである。図 3.5,3.6 は、マップ 3 を見下ろした状態と、マップ 3 の一部を写した図である。マップ 3 の広さは 48 × 40、描画マップの頂点数は 695、三角ポリゴンの数は 872 である。マップグラフ生成時の検証結果を表 3.6 に、プログラム実行時の検証結果を表 3.7 に示す。

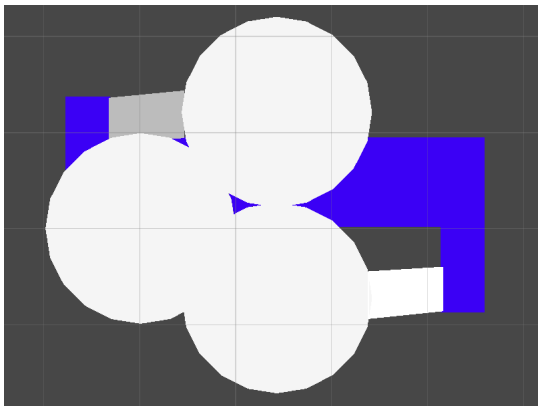

図 3.5: マップ 3 を見下ろした状態

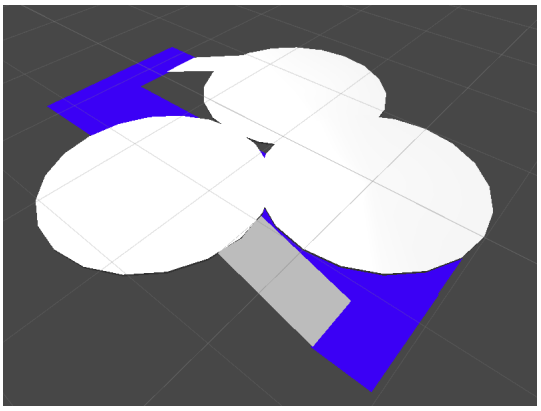

図 3.6: マップ 3 の一部を写した状態

表 3.6: マップ 3 でのグラフ生成時の検証結果

	マップグラフの容量	グラフ制作までの所要時間
NavMesh	13.8KB	0.6 秒
提案手法	101KB	1.81 秒

表 3.7: マップ 3 での実行時の検証結果

	処理開始までの時間	利用メモリ量
NavMesh	1.25 秒	9.9KB
提案手法	1.81 秒	9.6KB

3.2.4 マップ 4

マップ 4 は、Unity の Terrain(地形生成) 機能を利用して制作したマップである。図 3.7,3.8 は、マップ 4 を見下ろした状態と、マップ 4 の一部を写した図である。マップの広さは 2000 × 2000、描画マップの頂点数は 357、三角ポリゴンの数は 605 である。マップグラフ生成時の検証結果を表 3.8 に、プログラム実行時の検証結果を表 3.9 に示す。

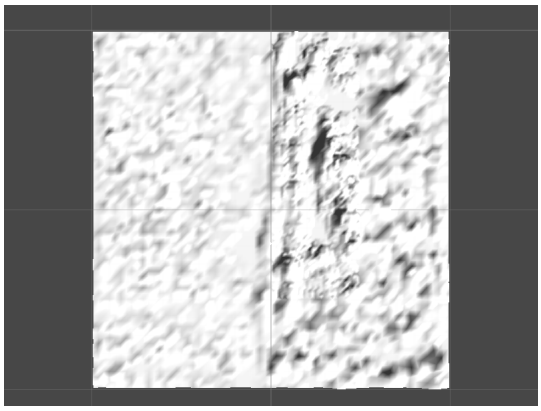

図 3.7: マップ 4 を見下ろした状態

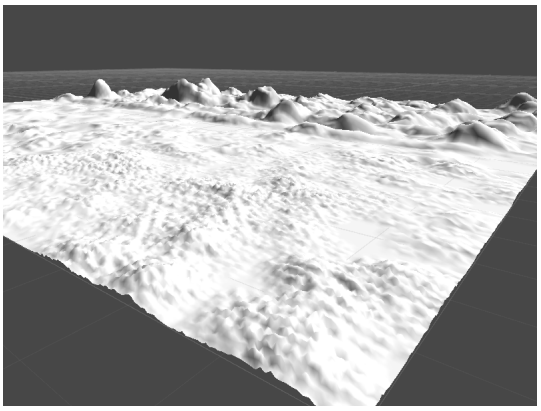

図 3.8: マップ 4 の一部を写した状態

表 3.8: マップ 4 でのグラフ生成時の検証結果

	マップグラフの容量	グラフ制作までの所要時間
NavMesh	21.4MB	554 秒
提案手法	1.09MB	7.1 秒

表 3.9: マップ 4 での実行時の検証結果

	処理開始までの時間	利用メモリ量
NavMesh	2.5 秒	21.5MB
提案手法	2.1 秒	9.6KB

3.3 設定の容易さの検証

設定の容易さの検証では、まず、用意されたマップテンプレートからマップグラフを生成し、その後、マップの中で通路を生成しない場所を決め、マップグラフを生成しないよう設定をするという作業をし、もう一度マップグラフの生成をする。その設定作業にかかった時間を検証した。ここで使用したマップを、マップ 5 とし、図 3.9,3.10 に示す。

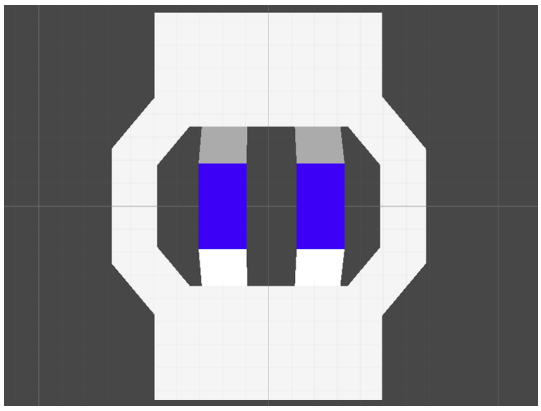

図 3.9: マップ 5 を見下ろした状態

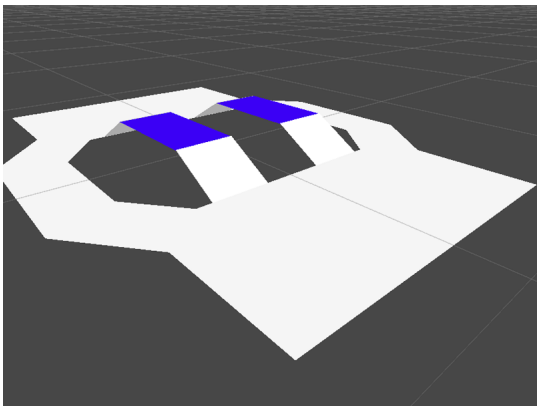

図 3.10: マップ 5 の一部を写した状態

マップ 5 の中で、マップグラフを作り直す場所を以下の図 3.11 に示す。図の赤く塗られている場所は、絶対に AI が通れないようにする場所とする。このため、赤い場所には AI 用のグラフデータは必要ないということになるので削除する。

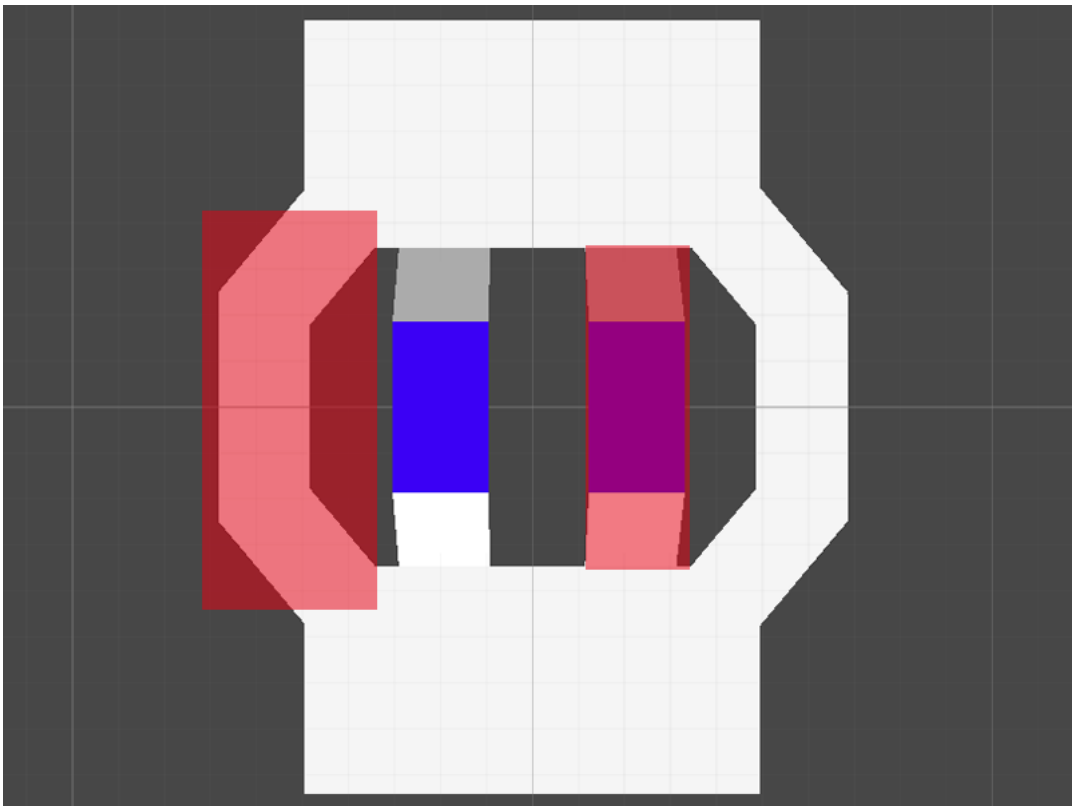

図 3.11: マップ 5 の作り直す場所

以上の条件で検証を行った結果は以下の通りになった。図 3.12,3.13 は、Unity の

NavMesh で生成されたマップグラフ及び、作り直したマップグラフ、図 3.14,3.15 は、提案手法で生成されたマップグラフである。検証の結果は表 3.10 に示す。この結果から、設定の容易さでは、提案手法が優位性を持っていると言える。

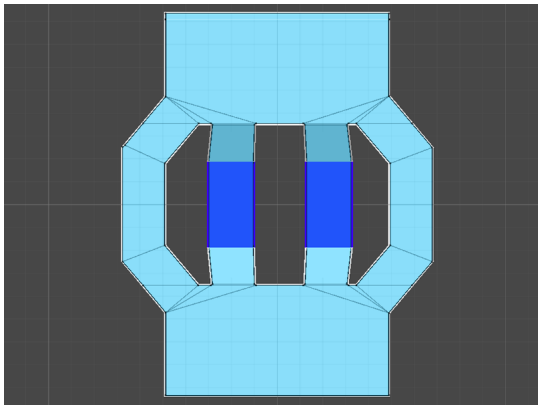

図 3.12: NavMesh でマップグラフを生成した状態

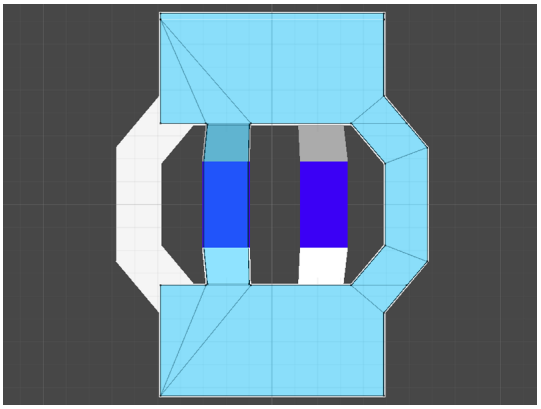

図 3.13: NavMesh のマップグラフを作り直した状態

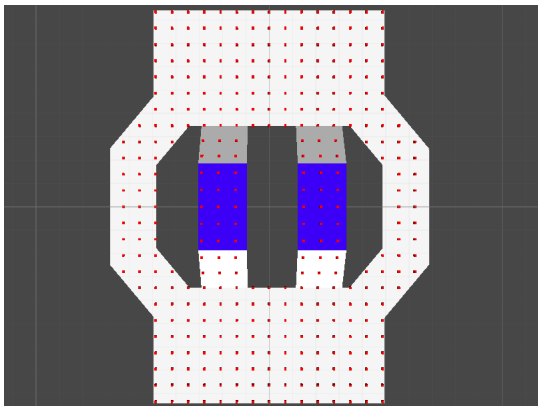

図 3.14: 提案手法でマップグラフを生成した状態

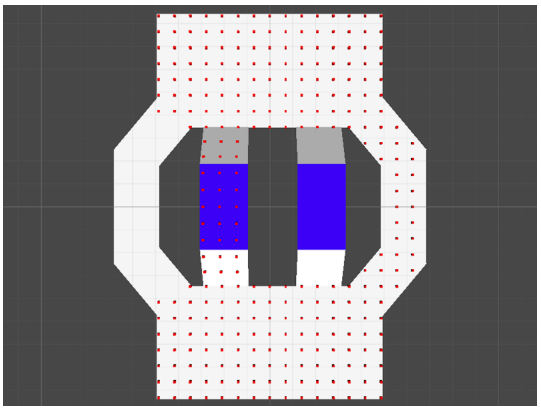

図 3.15: 提案手法のマップグラフを作り直した状態

表 3.10: マップ 5 での検証結果

	作り直しにかかった時間
NavMesh	23.5 秒
提案手法	10.3 秒

この検証をした結果わかったこととして、NavMesh によるマップグラフでは、

同一ポリゴン内の中では行ける場所、行けない場所を細かく設定できず、マップとして利用しているモデルを分割する必要がある。制作された描画用マップで、どうしてもマップのポリゴン分割ができない状況になった場合は、NavMesh で選択されたオブジェクトに対してマップグラフを作成してしまうため対応が難しい。しかし提案手法であれば、オブジェクトの状態にかかわらずノードを調整することが可能である。そのため、マップオブジェクトの状態がどうであれ、マップグラフを自由に変更、調整できるといった強みがある。

3.4 実用時での質の検証

実用時での質の検証では、生成されたマップグラフを利用し、あらかじめ想定していた経路をどれだけちゃんと辿れるか、AI が何秒でチェックポイントを通過できるかの二点を検証した。ここで使用したマップをマップ 6 とし、図 3.16,3.17 に示す。このマップにのみ登場する緑色のオブジェクトは障害物を表している。

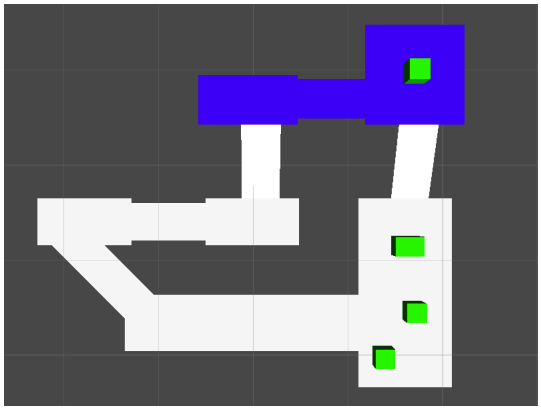

図 3.16: マップ 6 を見下ろした状態

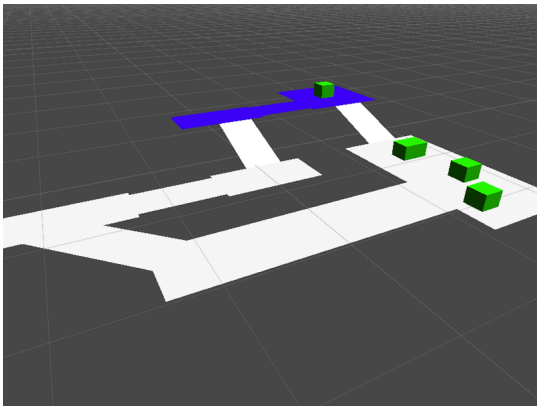

図 3.17: マップ 6 の一部を写した状態

あらかじめ AI に辿ってほしいと想定したルートを以下の図 3.18 に示す。このルートをどれだけ正しく辿れるかを評価した。さらに、AI がチェックポイントを通過するまでの時間も計測した。

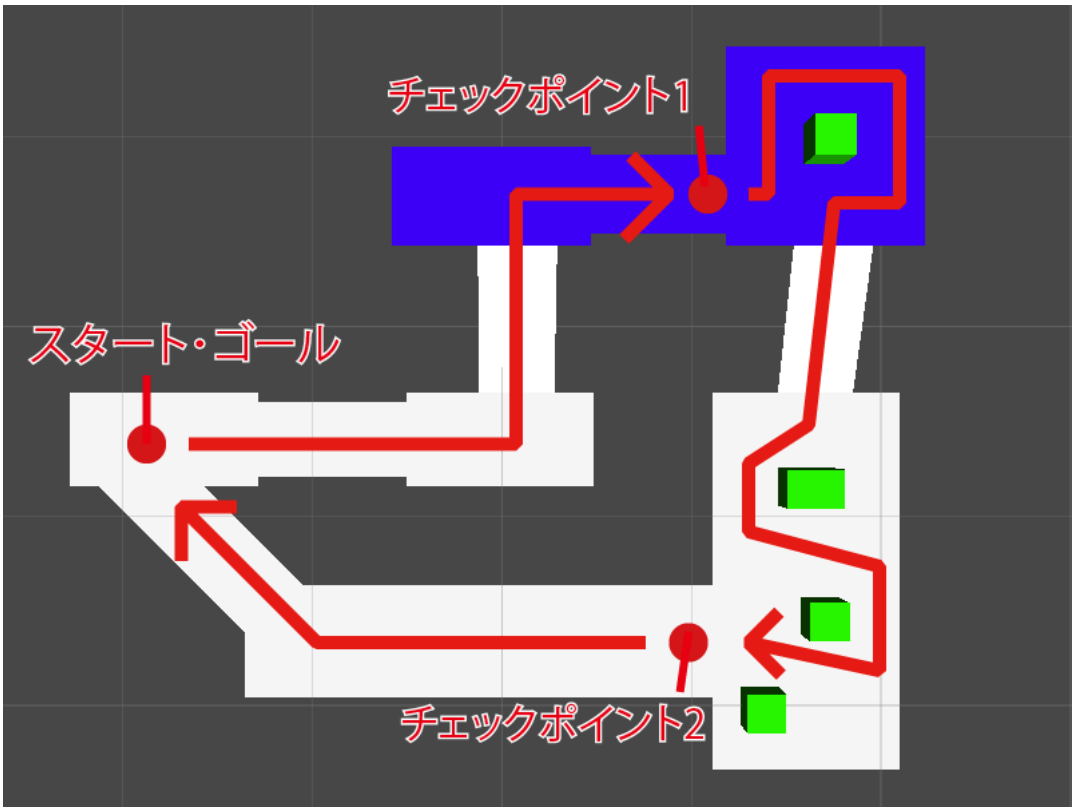

図 3.18: マップ 6 の想定ルート

以上の条件で検証を行った結果は以下の通りとなった。図 3.19 は、NavMesh、提案手法のそれぞれで制作されたマップグラフを利用して AI に A*による経路探索を行った場合の経路である。NavMesh では、道の真ん中ではなく、端ばかりを通ってしまった。この問題はノードのように、基準となるオブジェクトを増やせば対応することができるが、追加の作業が必要になってしまう。しかし、提案手法では、ノードの位置を調整するだけで対応することができる。

表 3.11 はそれぞれのチェックポイントを AI キャラクターが通過したタイムである。提案手法では、余計な経路を通っていないため、NavMesh よりも断然早くチェックポイントを通過することができた。しかし、直線的な経路を通っており、曲がり角などでは不自然に見えてしまうことがある。NavMesh は曲がり角では大きく回りこんでしまったが、自然な経路を通っていた。

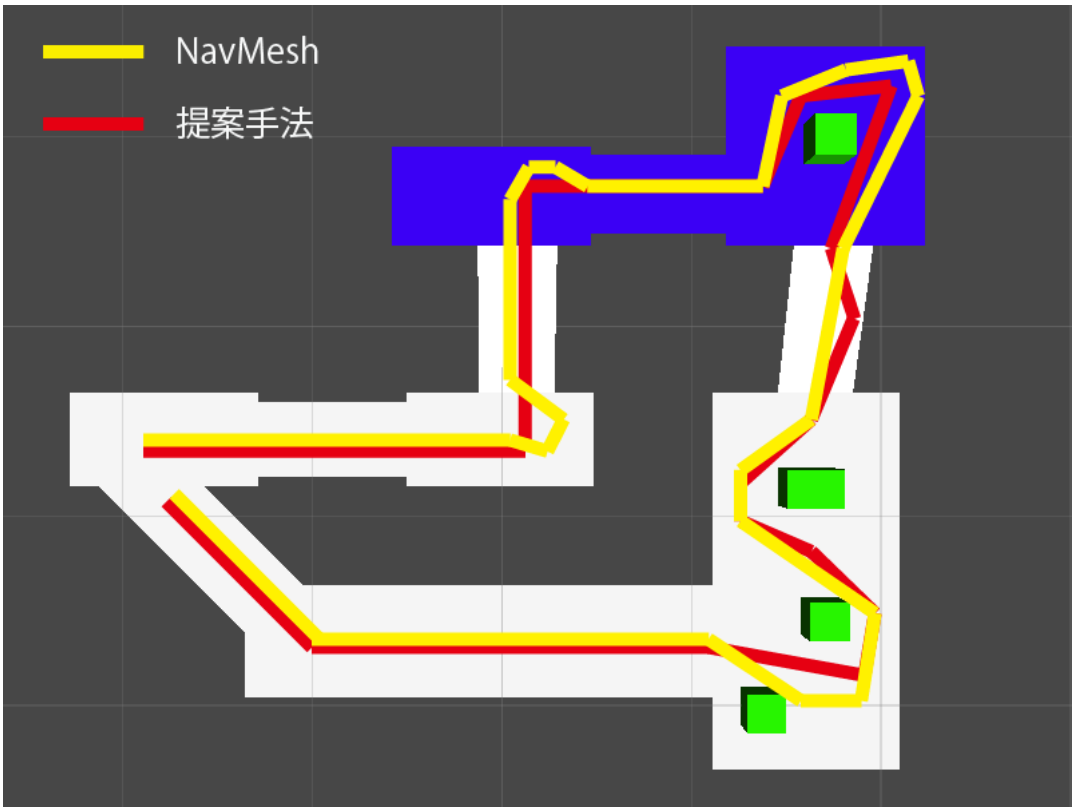

図 3.19: マップ 6 での検証結果

表 3.11: チェックポイント通過タイム

	チェックポイント 1	チェックポイント 2	ゴール
NavMesh	22 秒	35 秒	1 分 21 秒
提案手法	20 秒	31 秒	1 分 12 秒

3.5 考察

検証の結果、パフォーマンスの評価では、小さいマップであれば NavMesh の方が優れていることがわかった。しかし、Terrain を使って制作したマップのように大きなものになると、ビルド時間とデータ容量、共に大きな差が現れた。ポリゴンが多いマップや、大きいマップであればあるほどデータ容量、ビルド時間共に、Unity 標準の NavMesh よりも提案手法が優れていることが分かった。手直しのし

やすさでは、NavMeshが23秒、提案手法が10秒となり、提案手法の方が優位であるという結果であった。さらに、NavMeshではしにくかった修正も簡単にできることがわかった。実用時での質の検証では、NavMeshと提案手法に大きな差はなかった。つまり、質を変えずに手直しのしやすいマップグラフの生成が可能になったと言える。しかし問題点として、重なりあったマップが与えられたとき、一番上のオブジェクトにのみノードが置かれてしまうという欠点があることがわかった。

第 4 章

まとめ

本論文では、エージェント型 AI におけるマップグラフの生成に着目し、Unity 標準の NavMesh よりも生成速度が早く、よりデータ容量が軽く、より設定しやすいグラフの構築を目的に研究を行った。提案した手法によって生成されたマップグラフは、Unity のコンポーネントで作られているため、A*アルゴリズムや、パス検索、ダイクストラ法などに利用することが可能である。データ容量の観点から見ても削減されているため、モバイル向けにも利用しやすくなったと言える。また、NavMesh によるマップグラフでは、同一ポリゴン内の中では行ける場所、行けない場所を細かく設定できず、マップとして利用しているモデルを分割する必要がある。もし、制作された描画用マップで、どうしてもマップのポリゴン分割ができない状況になった場合、NavMesh では、選択されたオブジェクトに対してマップグラフを作成してしまうため対応が難しい。しかし、提案手法ではノードの位置を少し調整するだけで対応できるという利点がある。

今後の課題として、重なり合った地形が与えられたとき、一番上のオブジェクトにしかノードが設置されなくなってしまう問題が残った。これは、Unity のエディター拡張によって、ドラッグで指定した範囲にのみノードを設置する等の工夫ができると考えている。さらに、アルゴリズムもより最適化の余地があるため、今後も改良していきたい。

謝辞

本論文制作に当たり、終始丁寧にご指導下さった渡辺先生をはじめとする指導教員の方々、様々な助言を下さった大学院生、提案手法を改良してくださった安部先生、小島先輩、実装を手伝っていただいた奥山先輩に感謝の意を表します。本当にありがとうございました。

参考文献

[1] Togelius. Julian, Karakovskiy. Sergey, and Baumgarten. Robin. The 2009 mario ai competition. *Evolutionary Computation (CEC), 2010 IEEE Congress on*, 2010.

[2] S. KARAKOVSKIY. The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAG)*, 2012.

[3] スーパーマリオワールド, 1990. http://www.nintendo.co.jp/wii/vc/vc_smw/ 参照: 2015-2-12.

[4] MuuMuu CO.LTD. アストロノーカ, 1998. http://www.jp.playstation.com/software/title/jp0082npjj00168_000000000000000001.html 参照: 2015-2-12.

[5] MuuMuu CO.LTD. 頑張れ森川君 2 号, 1997. http://www.jp.playstation.com/software/title/jp9000npji00013_000000000000000001.html 参照: 2015-2-12.

[6] B. M. Blumberg and T. A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, 1995.

[7] D. Isla, R. Burke, M. Downie, and B. Blumberg. A layered brain architecture for synthetic creatures. *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, 2001.

[8] 三宅陽一郎. ゲームの中の人工知能. 玉川大学講演会資料, 2014.

[9] Shinshu University. グラフ理論. `http://markun.cs.shinshu-u.ac.jp/learn/graph/GraphTheory-j.html` 参照: 2015-2-12.

[10] 原川和泰. ジャンプする物体の経路プラニング. 東京工科大学学士卒業論文, 2003.

[11] M. Foskey, M. Garber, M. C. Lin, and D. Manocha. A voronoi-based hybrid motion planner. `http://cs.unc.edu/?geom/voronoi/vplan/`.

[12] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics4(2):100*, 1968.

[13] T Lozano-Prez. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM22(10):560*, 1979.

[14] 森江修也. 集団移動シミュレーションにおける移動障害物の回避に関する研究. 東京工科大学学士卒業論文, 2002.

[15] 中部智也. 群集の経路プランニング. 東京工科大学学士卒業論文, 2004.

[16] 三宅陽一郎. y-miyake のゲーム ai 千夜一夜. `http://blogai.igda.jp/article/66585525.html` 参照: 2015-2-12.

[17] 三宅陽一郎. デジタルゲームにおける最新の人工知能システム. 東京工芸大学講演会資料, 2014.

[18] 三宅陽一郎. エージェント・アーキテクチャから作るキャラクター ai. CEDEC2007 講演資料, 2007.

[19] 三宅陽一郎. エージェント・アーキテクチャに基づくキャラクター ai の実装 ―クロムハウンズのキャラクター ai を例として―. 第 4 回デジタルコンテンツシンポジウム, 2008. 第 4 回デジタルコンテンツシンポジウム公演予稿集.

[20] 三宅陽一郎. ディジタルゲームにおける人工知能技術の応用の現在. 人工知能学会誌 30 巻 1 号, 2015.

[21] ユニティ・テクノロジーズ. Unity, 2005. http://japan.unity3d.com/ 参照: 2015-2-12.

[22] ユニティ・テクノロジーズ. ナビゲーション レイヤーおよびコスト. http://docs-jp.unity3d.com/Documentation/Manual/NavLayersAndCosts.html 参照: 2015-2-12.

[23] 無向グラフ, 有向グラフ, 混合グラフ. http://www.di.kagawa-nct.ac.jp/~matusita/Arena/graph-and-algorithm/page01-02.html 参照: 2015-2-12.

[24] A* pathfinding project. http://arongranberg.com/astar/ 参照: 2015-2-12.