FFXIVサーバーサイド
経路探索システム

講演者紹介

- 横山 貴規

- Luminous AIプログラマ
- FFXIV経路探索の環境構築、サーバ組込、ゲームとの接続を担当

- グラヴォ ファビアン

- Luminous AIリサーチャー
- FFXIV経路探索のコアアルゴリズムの考案と実装、落下メッシュ作成を担当

受講対象者

- MMOで経路探索をしたい人

受講対象者

- MMOで経路探索をしたい人
- 軽い経路探索を知りたい人

受講対象者

- MMOで経路探索をしたい人
- 軽い経路探索を知りたい人
- ナビメッシュをツールチェインに組み込む方法を知りたい人

受講対象者

- MMOで経路探索をしたい人
- 軽い経路探索を知りたい人
- ナビメッシュをツールチェインに組み込む方法を知りたい人
- ナビメッシュを作るのが面倒な人

受講対象者

- MMOで経路探索をしたい人
- 軽い経路探索を知りたい人
- ナビメッシュをツールチェインに組み込む方法を知りたい人
- ナビメッシュを作るのが面倒な人
- AIプログラマがやっていることを知りたい人

もくじ

- 第1部 概要

もくじ

- 第1部 概要
- 第2部 経路探索を軽くする

もくじ

- 第1部 概要
- 第2部 経路探索を軽くする
- 第3部 FFXIVでの経路探索

もくじ

- 第１部 概要
- 第２部 経路探索を軽くする
- 第３部 FFXIVでの経路探索
- 第４部 ナビメッシュ自動生成と落下

もくじ

- 第１部 概要
- 第２部 経路探索を軽くする
- 第３部 FFXIVでの経路探索
- 第４部 ナビメッシュ自動生成と落下
- 第５部 開発環境、運用

もくじ

- 第１部 概要
- 第２部 経路探索を軽くする
- 第３部 FFXIVでの経路探索
- 第４部 ナビメッシュ自動生成と落下
- 第５部 開発環境、運用
- 番外編

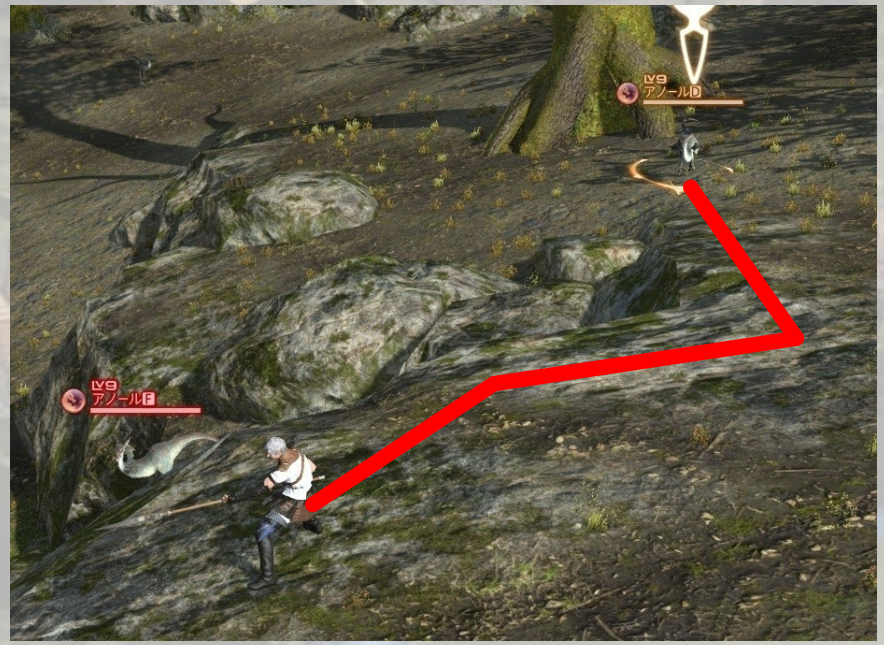
第1部
概要

第1部 概要

- 経路探索の基礎
- FFXIV紹介
- FFXIVで必要な機能、要件

キャラクターの経路探索

- AIキャラクターが歩く道順を知らせる



ナビメッシュとは？

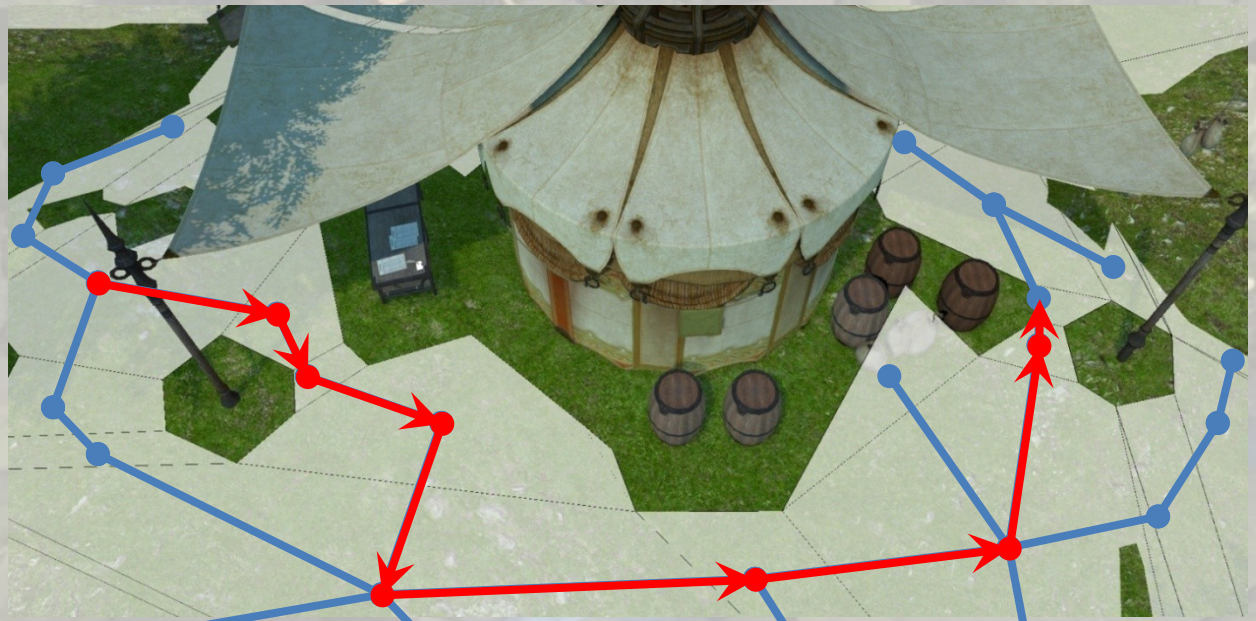
- 移動可能な場所をポリゴンメッシュで表現

グラフ構造

- ナビメッシュをグラフ構造とする

経路探索

- 最短の道を調べられる。



第1部 概要

- 経路探索の基礎
- FFXIV紹介
- FFXIVで必要な機能、要件

Final Fantasy XIV A Realm Reborn

Final Fantasy XIV © 2010-2012 SQUARE ENIX CO., LTD. All rights reserved.

Final Fantasy XIV A Realm Reborn

Final Fantasy XIV © 2010-2012 SQUARE ENIX CO., LTD. All rights reserved.

SQUARE ENIX

FFXIVの経路探索

第1部 概要

- 経路探索の基礎
- FFXIV紹介
- FFXIVで必要な機能、要件

同じマップで敵がたくさん出てくる

- 1マップで非常にたくさんの敵が同時に動く

同じマップで敵がたくさん出てくる

- CPU負荷を軽くする！

同じマップで敵がたくさん出てくる

- CPU負荷を軽くする！ ← 最重要！

マップが広くて、数も多い

- 最大2km四方。マップ数も多い

マップが広くて、数も多い

- 量産しやすい形に！

自然地形のマップ

- ジャンプしたり落下したりできる

Final Fantasy XIV © 2010-2012 SQUARE ENIX CO., LTD. All rights reserved.

今回採用した手法

- CPU負荷を減らす

今回採用した手法

- CPU負荷を減らす
  - 経路テーブルを使った経路探索

今回採用した手法

- CPU負荷を減らす
  - 経路テーブルを使った経路探索
- 自然地形への対応

今回採用した手法

- CPU負荷を減らす
  - 経路テーブルを使った経路探索
- 自然地形への対応
  - ナビゲーションメッシュ

今回採用した手法

- CPU負荷を減らす
  - 経路テーブルを使った経路探索
- 自然地形への対応
  - ナビゲーションメッシュ
- 量産しやすい形

今回採用した手法

- CPU負荷を減らす
  - 経路テーブルを使った経路探索
- 自然地形への対応
  - ナビゲーションメッシュ
- 量産しやすい形
  - ナビゲーションメッシュの自動生成

第2部
経路探索を軽くする

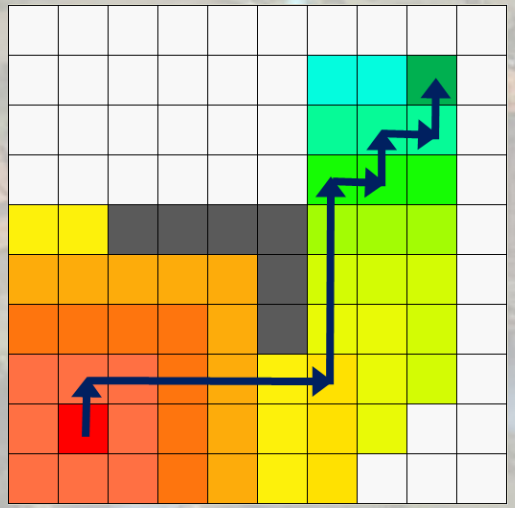
第2部 経路探索を軽くする

- 経路テーブルをたどる
- 分割＆階層化してメモリ削減
- 隣接化して自然な経路探索

SQUARE ENIX

一般的なA*では？

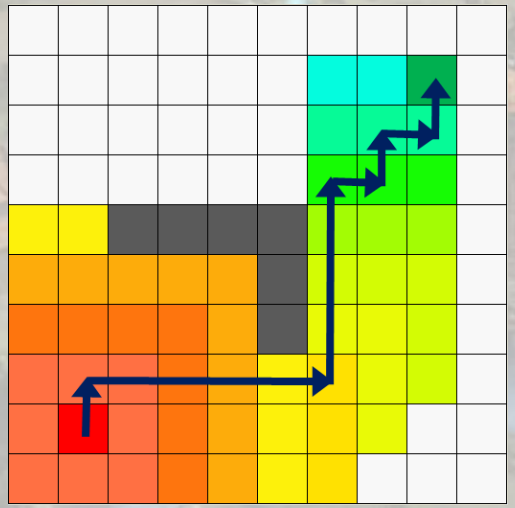
- A*では調べながら最短経路を見つける



SQUARE ENIX

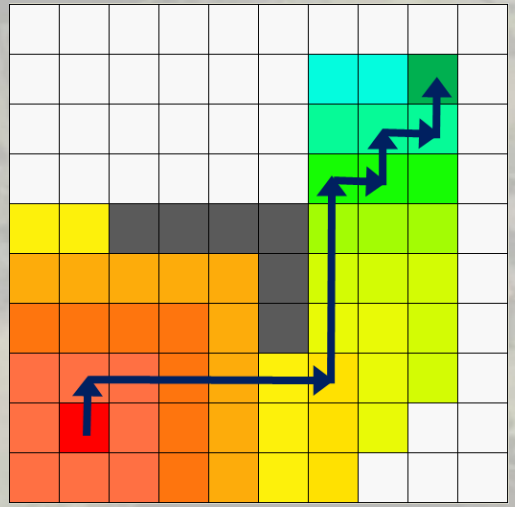
一般的なA*では？

- A*では調べながら最短経路を見つける
  - メモリも必要だし、ひとつひとつ調べる負荷が重い



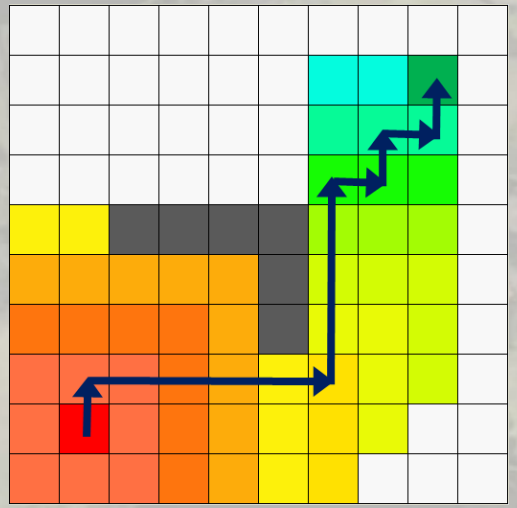
経路テーブルを使った経路探索

- 一歩ずつ調べると負荷が大きいので



経路テーブル

- 結果を経路テーブルにしてしまう



	a	b	c	d	e	f	g	h	i
a		0	2	0	1	0	1	3	3
b	2		2	0	2	0	1	3	3
c	2	1		0	2	2	1	3	3
d	2	1	1		2	2	2	3	3
e	1	1	1	1		2	2	3	3
f	1	0	1	1	1		2	3	3
g	1	0	2	1	1	0		3	3
h	3	3	3	3	3	3	3		1
i	3	3	3	3	3	3	3	2	

経路テーブル

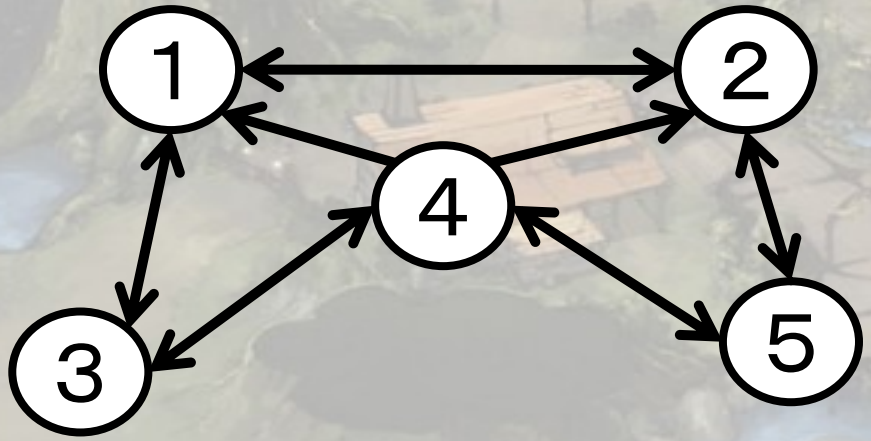
- この手法自体は、10年ほど前からよく使われていた手法です。

AI Game Programing Wisdom

“4.2 Preprocessed Solution for Open Terrain Navigation”

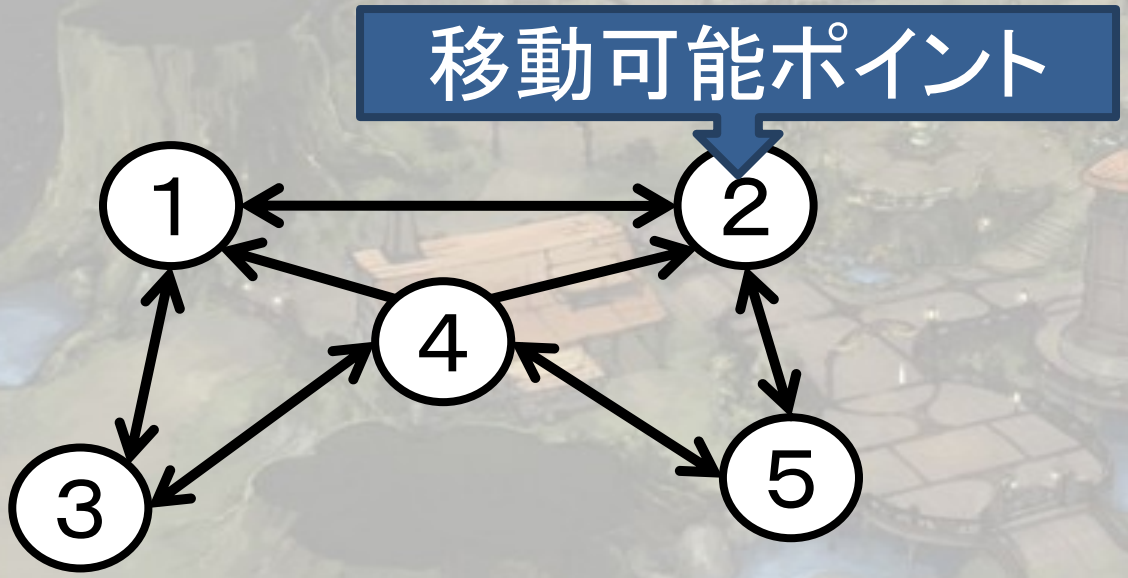
経路テーブルを使った経路探索

- この地図を経路探索してみます



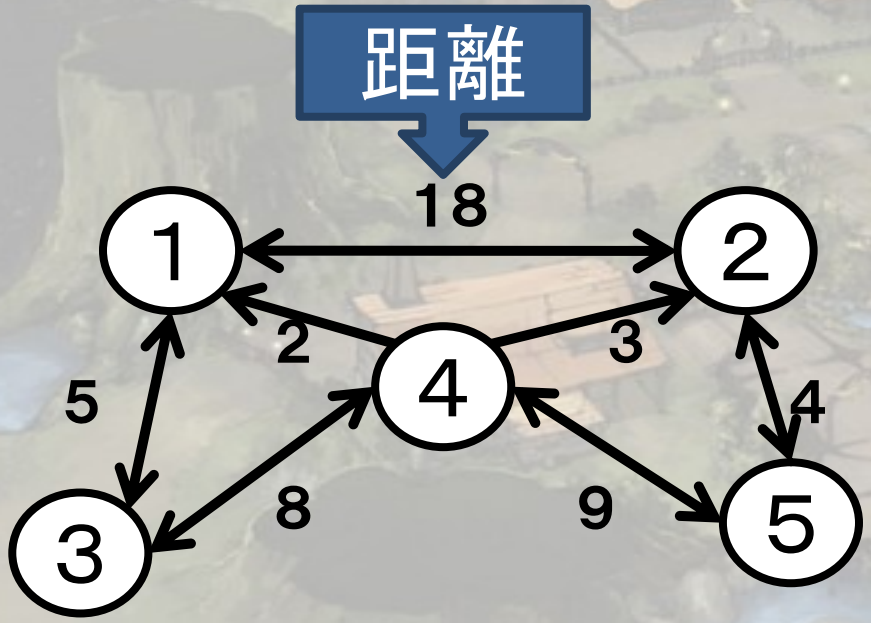
経路テーブルを使った経路探索

- この地図を経路探索してみます



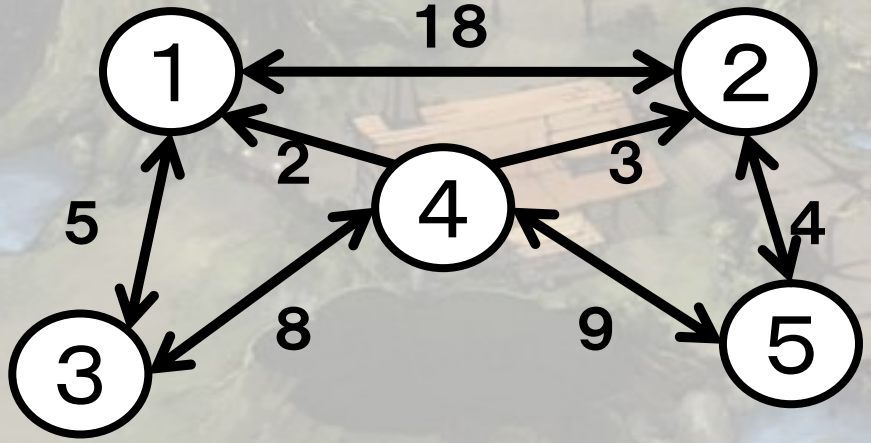
経路テーブルを使った経路探索

- この地図を経路探索してみます



経路テーブルを使った経路探索

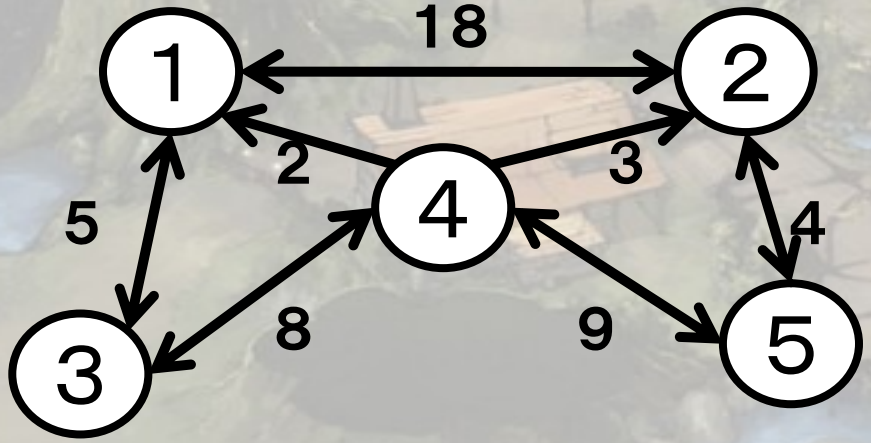
- 経路をテーブルにします



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

- 経路をテーブルにします

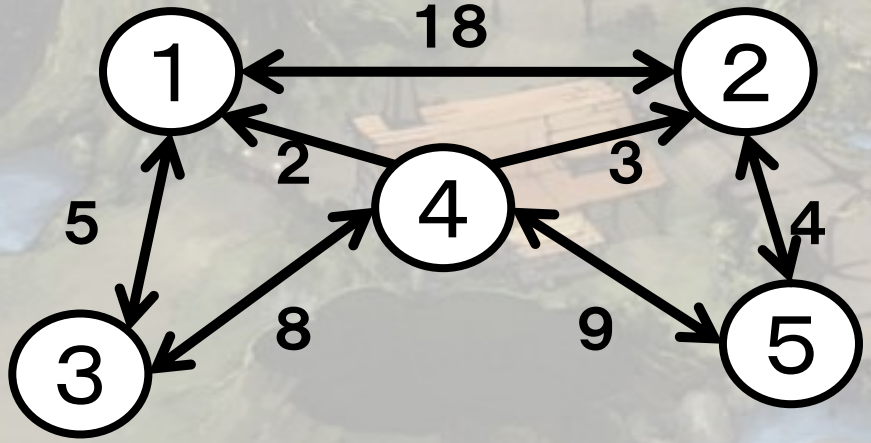


経路テーブル

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

- 表から次の行先がわかる

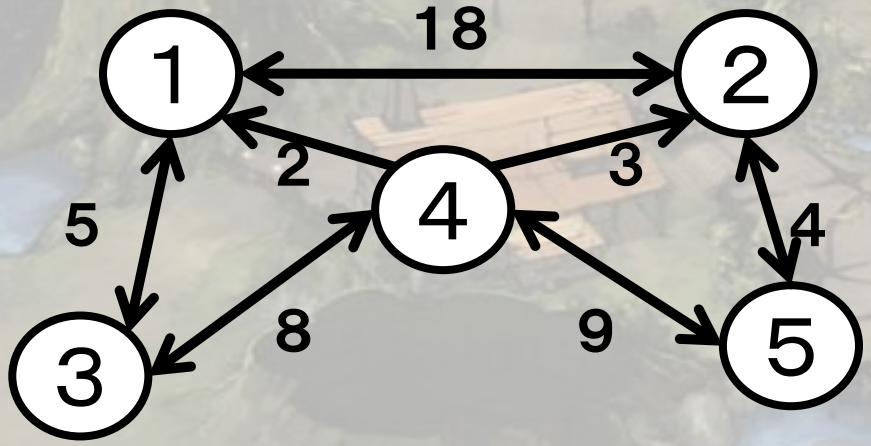


	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

SQUARE ENIX

経路テーブルを使った経路探索

- 表から次の行先がわかる



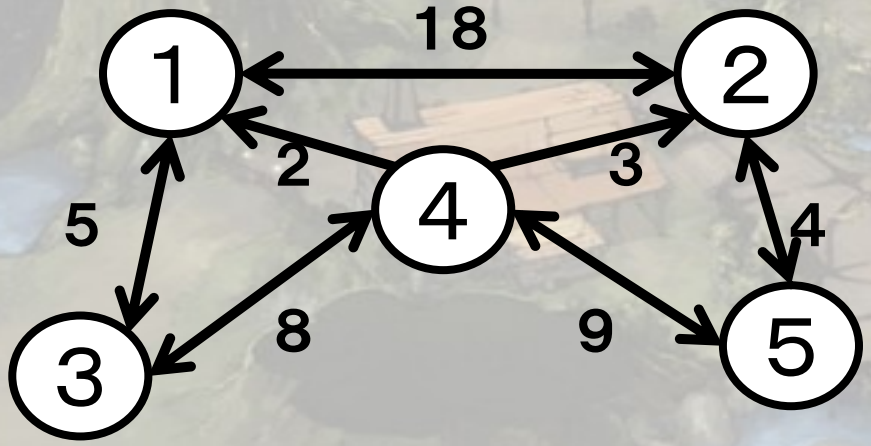
今いる所

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

SQUARE ENIX

経路テーブルを使った経路探索

- 表から次の行先がわかる



今いる所

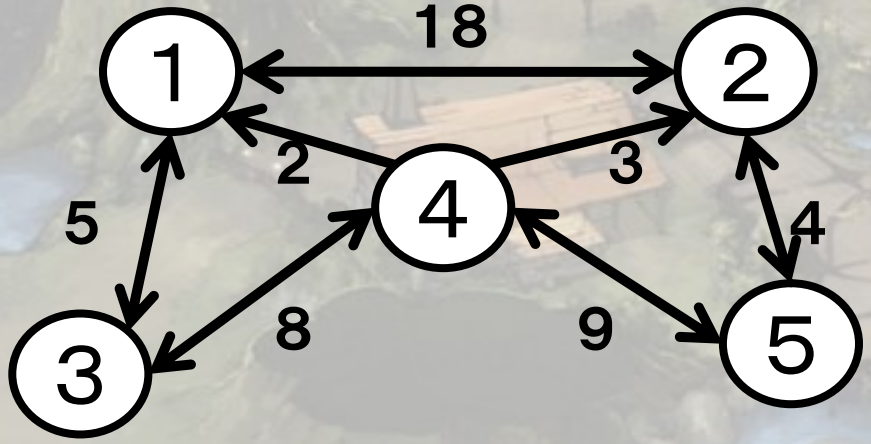
	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

目的地

SQUARE ENIX

経路テーブルを使った経路探索

- 表から次の行先がわかる



今いる所

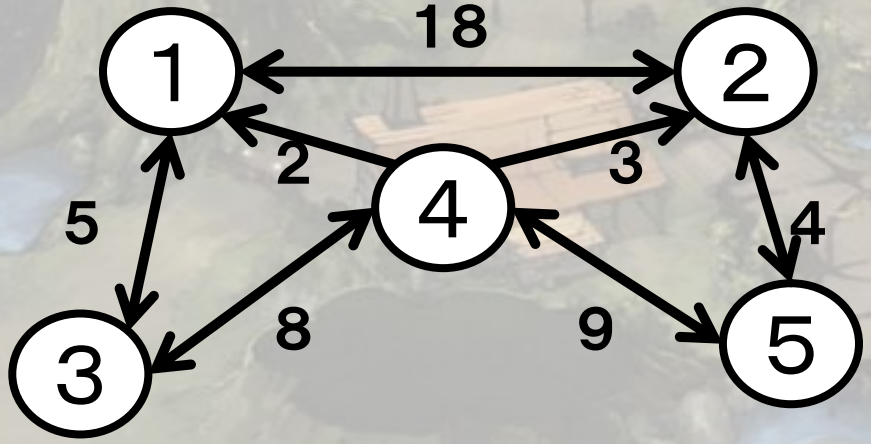
	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

目的地

SQUARE ENIX

経路テーブルを使った経路探索

- 表から次の行先がわかる



今いる所

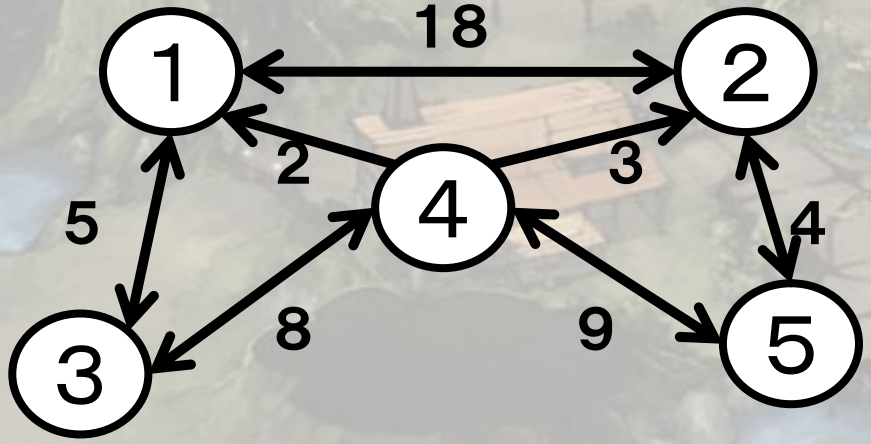
	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

目的地

SQUARE ENIX

経路テーブルを使った経路探索

- 表から次の行先がわかる



今いる所

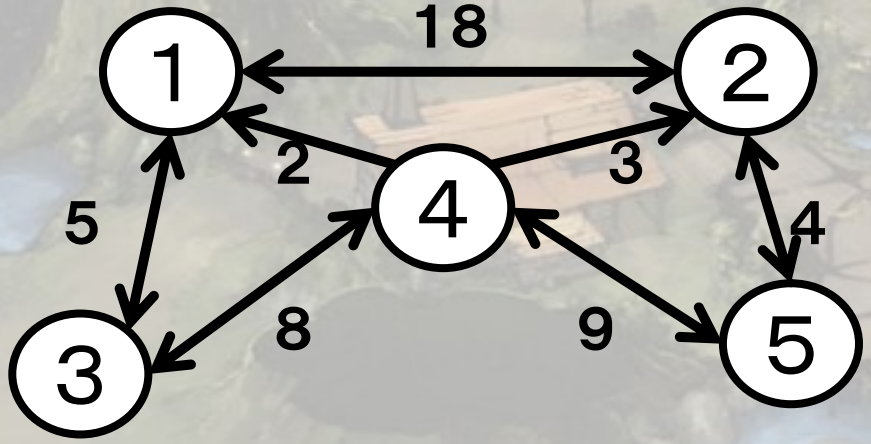
目的地

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

SQUARE ENIX

経路テーブルを使った経路探索

- 表から次の行先がわかる



今いる所

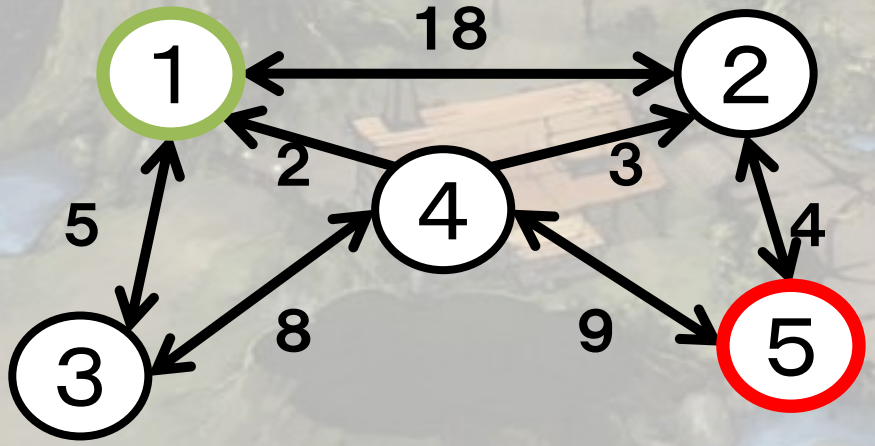
目的地

次の所

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

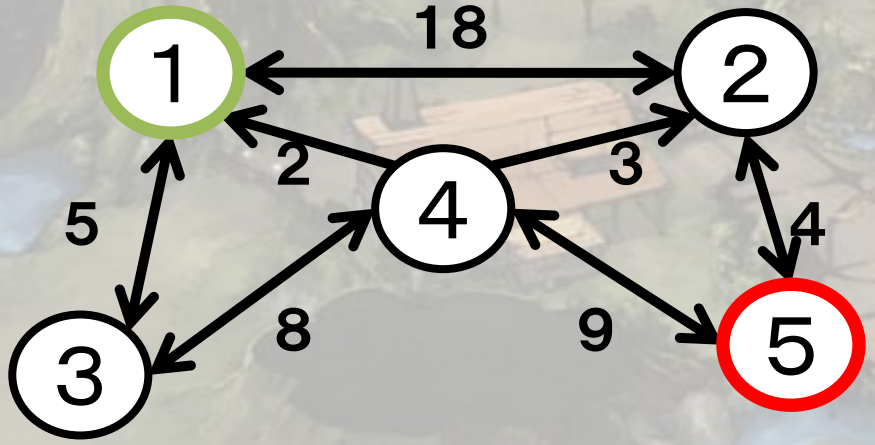
- 1→5にいくとき



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

- 1→5にいくとき
  - 1→5を見ると

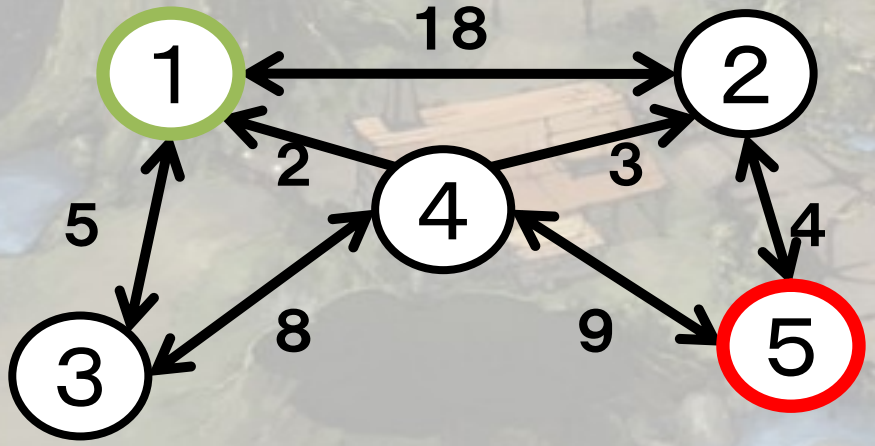


今いる所

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

- 1→5にいくとき
  - 1→5を見ると

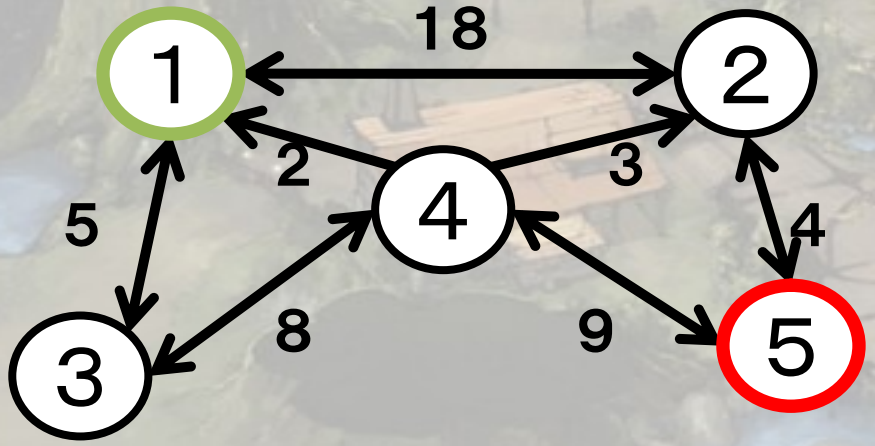


	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

目的地

経路テーブルを使った経路探索

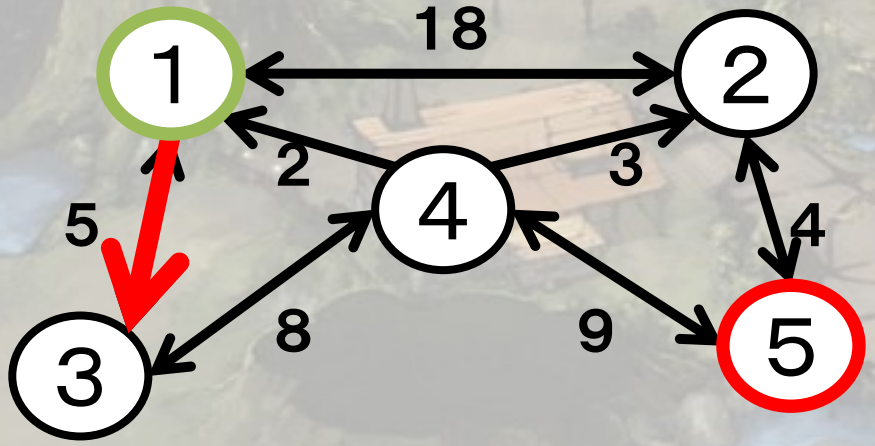
- 1→5にいくとき
  - 1→5を見ると



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

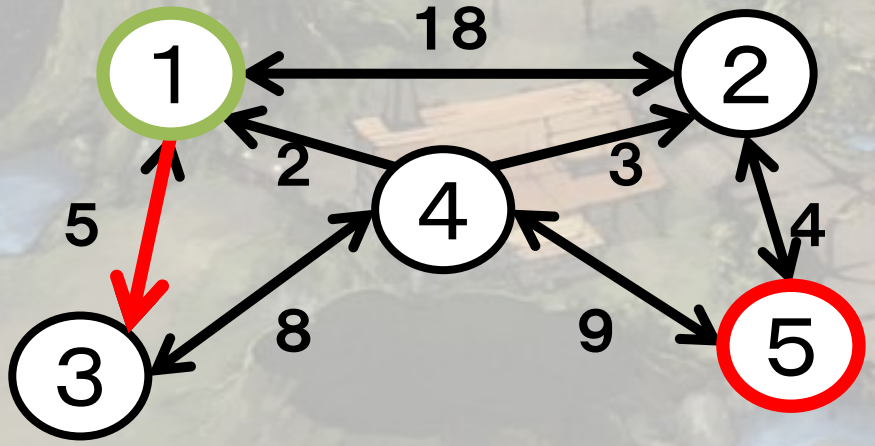
- 1→5にいくとき
  - 1→5を見ると3に行く



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

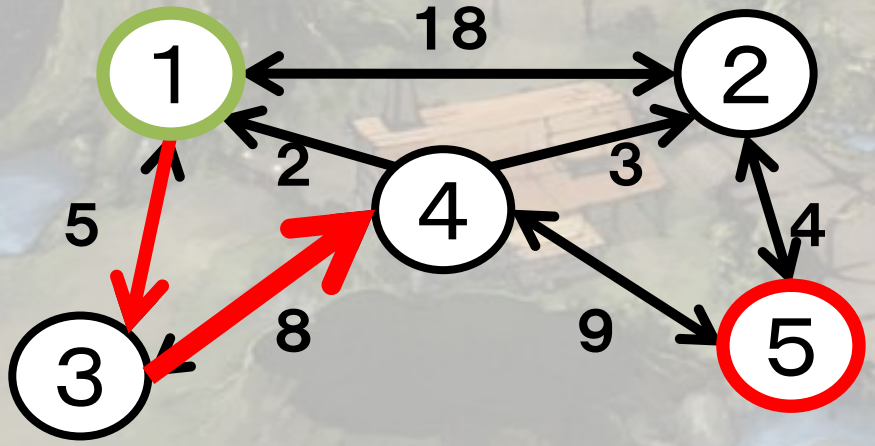
- 1→5にいくとき
  - 3→5を見ると



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

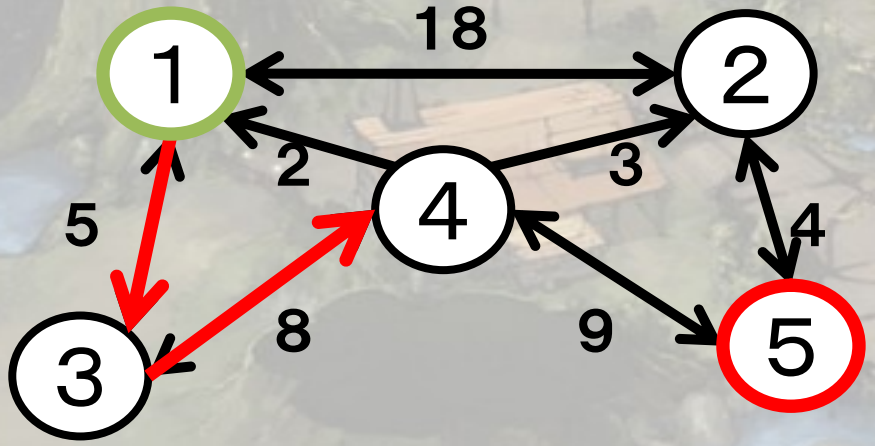
- 1→5にいくとき
  - 3→5を見ると4に行く



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

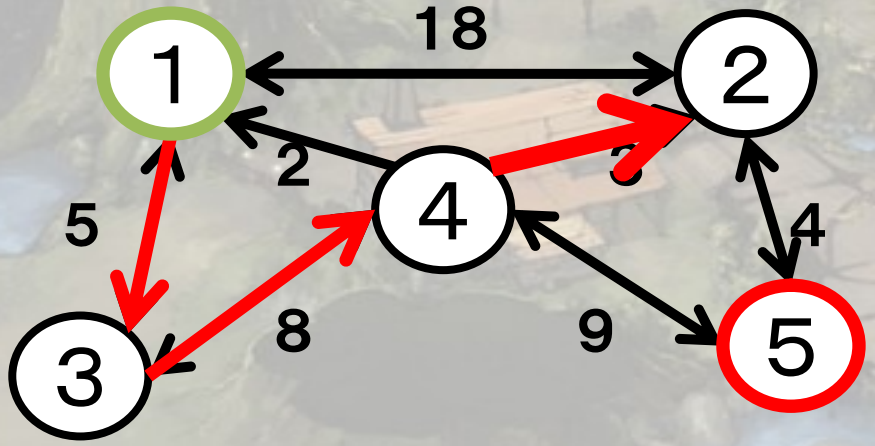
- 1→5にいくとき
  - 4→5を見ると



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

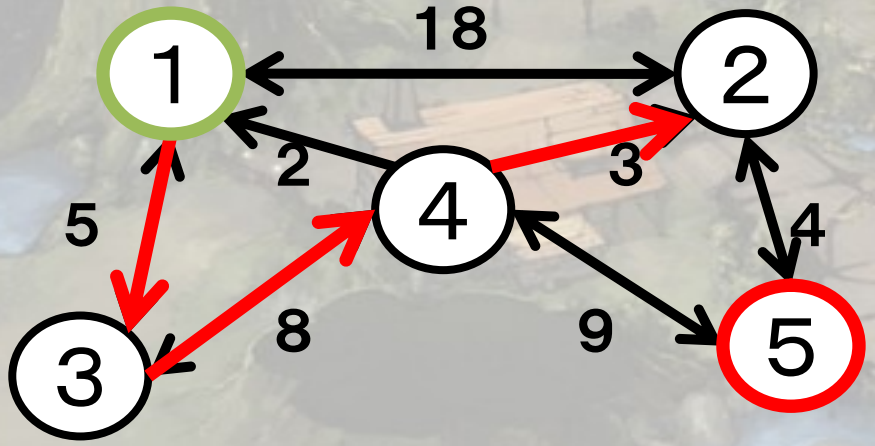
- 1→5にいくとき
  - 4→5を見ると2に行く



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

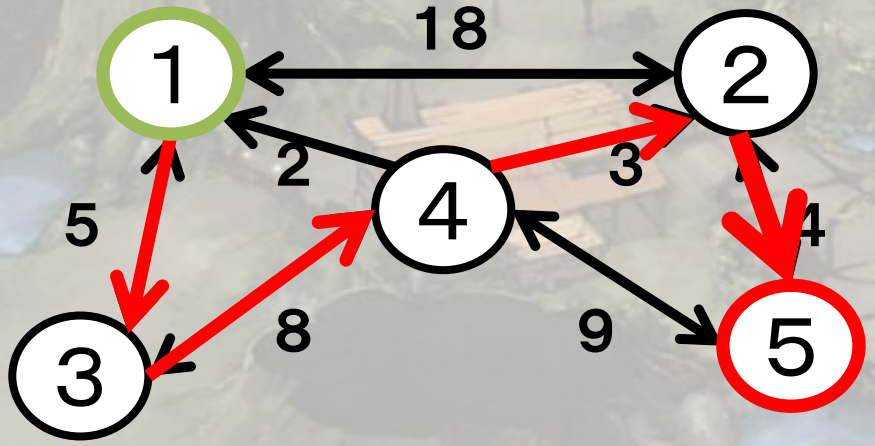
- 1→5にいくとき
  - 2→5を見ると



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

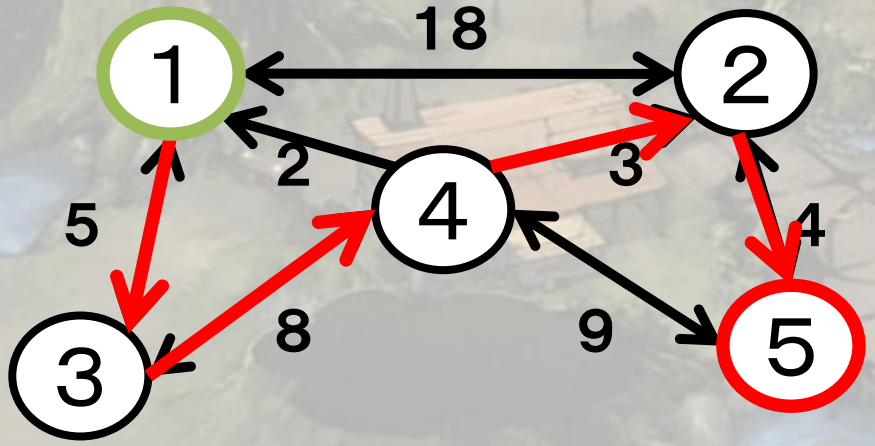
- 1→5にいくとき
  - 2→5を見ると5に行く



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルを使った経路探索

- 1→5にいくとき
  - 5についた。到着！



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルのメリット

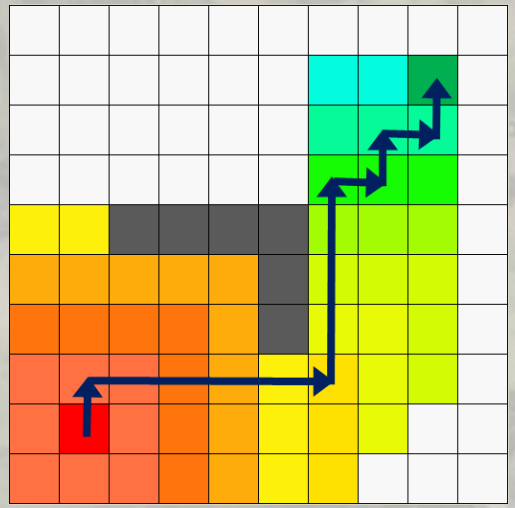
経路テーブルのメリット

- テーブルアクセスだけなので負荷は軽い！

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルのメリット

- メモリサイズが変わらない
  - A*のように検索時にメモリを消費しない



	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルのデメリット

SQUARE ENIX

経路テーブルのデメリット

- メモリ消費量が大きい

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

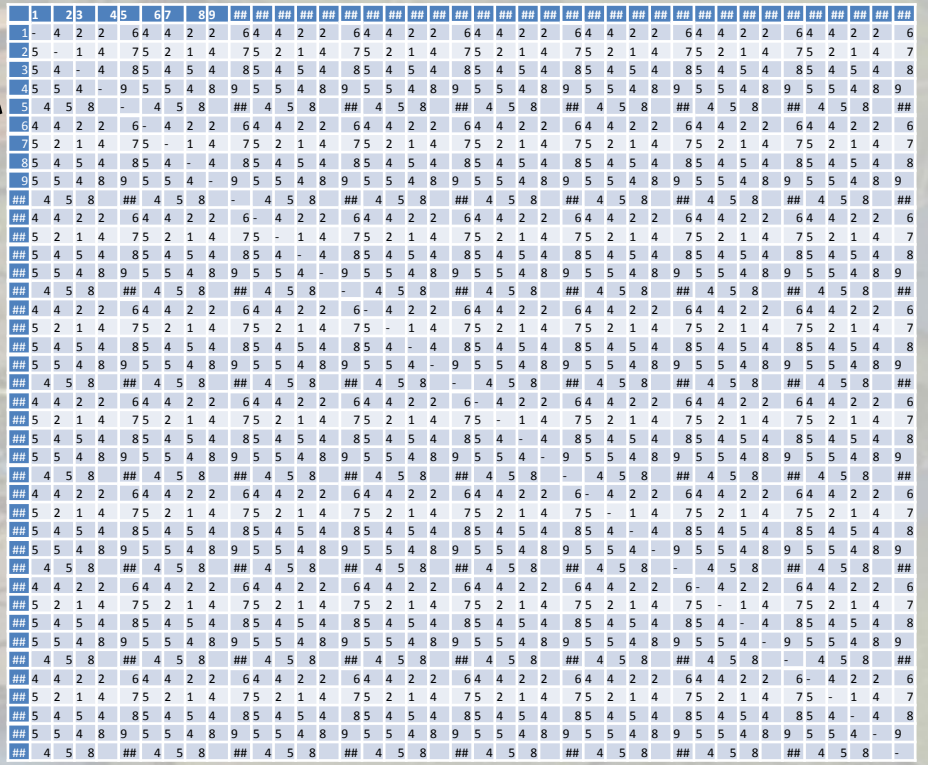
経路テーブルのデメリット

- メモリ消費量が大きい
  - 2万メッシュだと…

	1	2	3	4	5
1	-	5	1	1	4
2	3	-	4	2	2
3	3	5	-	1	4
4	3	5	4	-	4
5	3	5	4	2	-

経路テーブルのデメリット

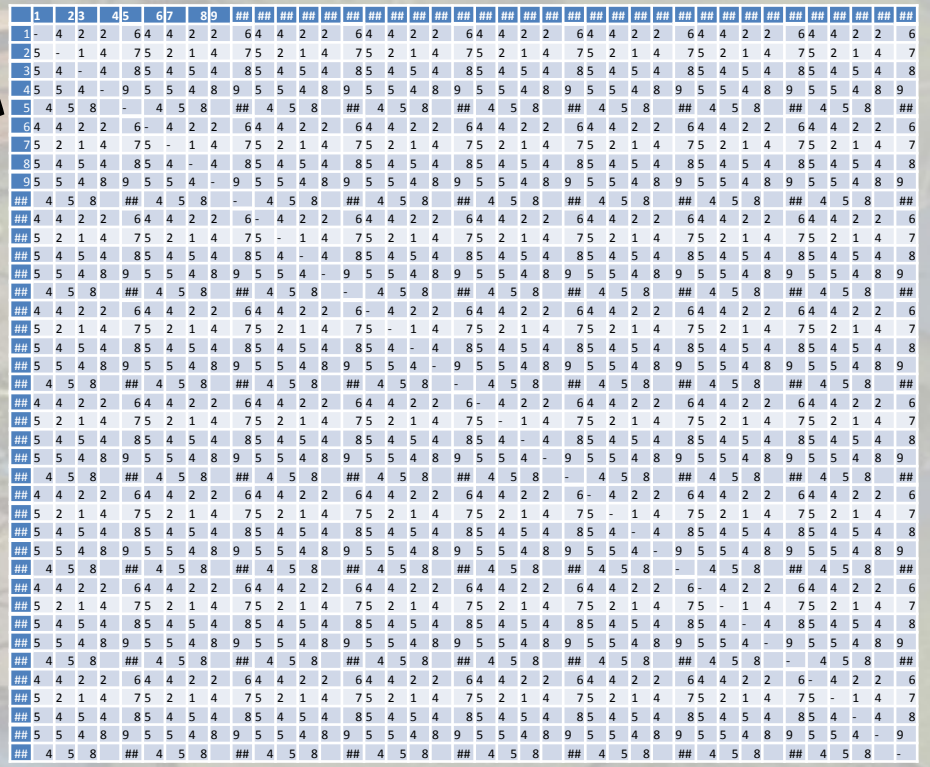
- メモリ消費量が大きい
  - 2万メッシュだと…



SQUARE ENIX

経路テーブルのデメリット

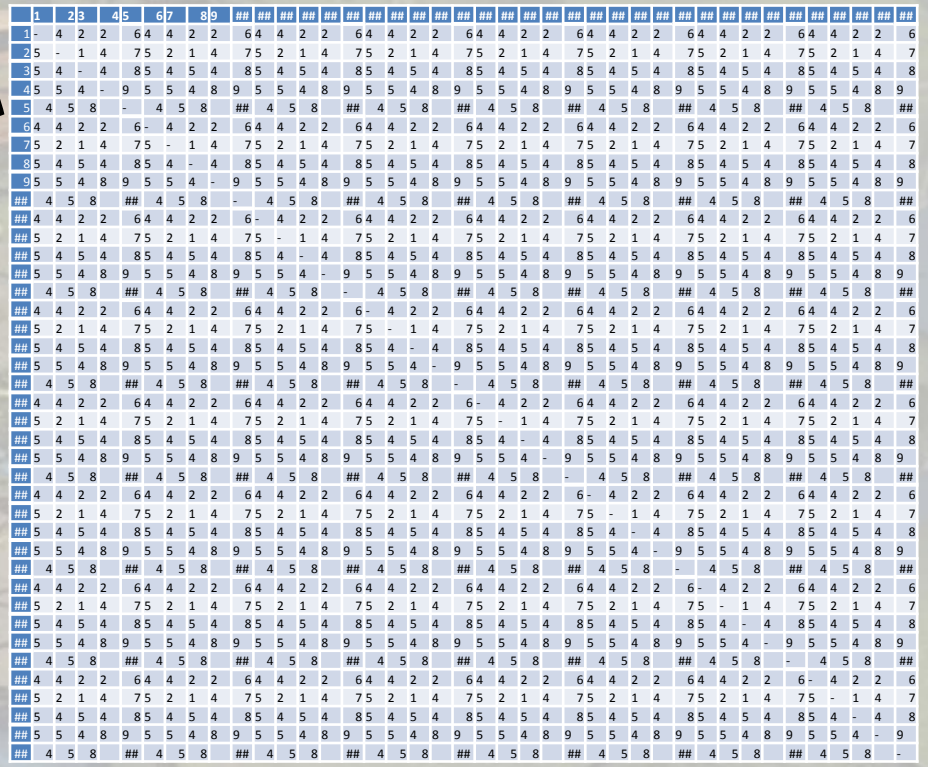
- メモリ消費量が大きい
  - 2万メッシュだと
  - 2万×2万で…



経路テーブルのデメリット

- メモリ消費量が大きい
  - 2万メッシュだと
  - 2万×2万で…

1.1GB！！

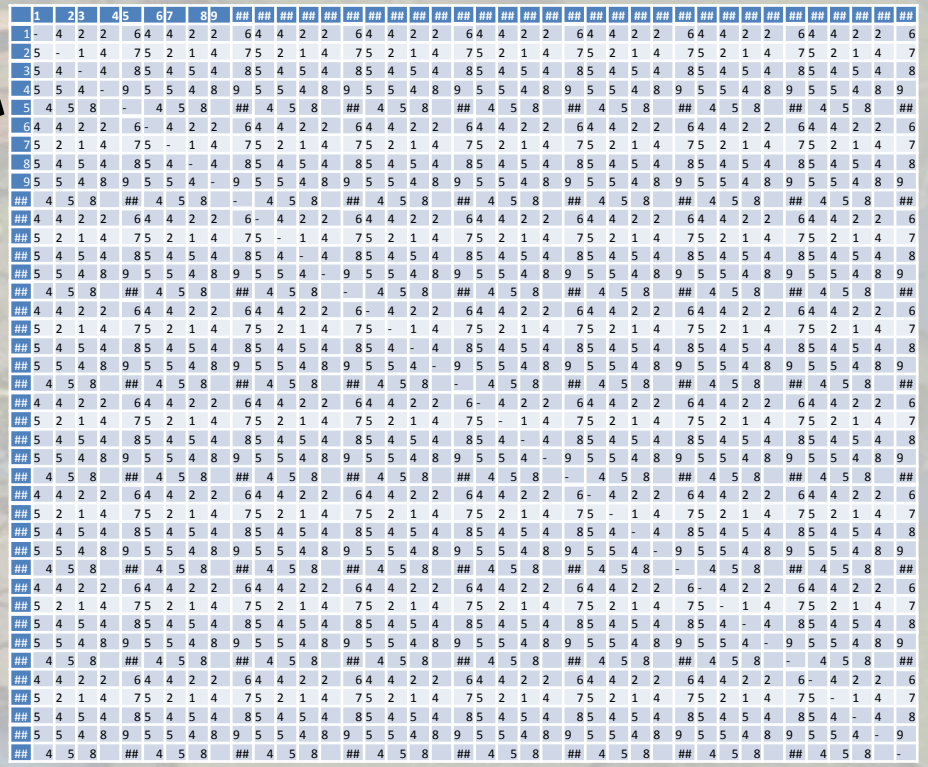


経路テーブルのデメリット

- メモリ消費量が大きい
  - 2万メッシュだと
  - 2万×2万で…

1.1GB！！

さすがにサーバでも厳しい…



経路テーブルのデメリット

- 動くものに対応しづらい

経路テーブルのデメリット

- 動くものに対応しづらい
  - ドアの開閉
  - 壁を破壊

経路テーブル結果

○経路テーブルで、CPU負荷が軽くなった！

×メモリ消費量がとても大きい

第2部 経路探索を軽くする

- 経路テーブルをたどる
- 分割＆階層化してメモリ削減
- 隣接化して自然な経路探索

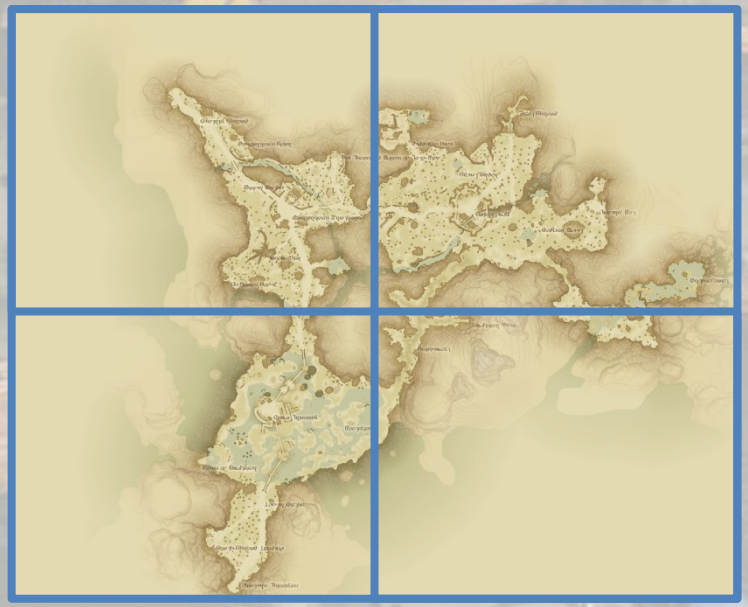
SQUARE ENIX

1つのテーブルにすると大きい

	1	2	3	4	5	6	7	8	9	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##	##
1	-	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6
2	5	-	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7
3	5	4	-	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8
4	5	5	4	-	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9
5	4	5	8		-	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##
6	4	4	2	2	6	-	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6
7	5	2	1	4	7	5	-	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7
8	5	4	5	4	8	5	4	-	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8
9	5	5	4	8	9	5	5	4	-	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9
##	4	5	8		##	4	5	8		-	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##
##	4	4	2	2	6	4	4	2	2	6	-	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6
##	5	2	1	4	7	5	2	1	4	7	5	-	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7
##	5	4	5	4	8	5	4	5	4	8	5	4	-	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8
##	5	5	4	8	9	5	5	4	8	9	5	5	4	-	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9
##	4	5	8		##	4	5	8		##	4	5	8		-	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##
##	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	-	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6
##	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	-	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7
##	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	-	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8
##	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	-	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9
##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		-	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##
##	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	-	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6
##	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	-	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7
##	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	-	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8
##	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	-	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9
##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		-	4	5	8		##	4	5	8		##	4	5	8		##
##	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	-	4	2	2	6	4	4	2	2	6	4	4	2	2	6
##	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	-	1	4	7	5	2	1	4	7	5	2	1	4	7
##	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	-	4	8	5	4	5	4	8	5	4	5	4	8
##	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	-	9	5	5	4	8	9	5	5	4	8	9
##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		-	4	5	8		##	4	5	8		##
##	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	-	4	2	2	6	4	4	2	2	6
##	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	-	1	4	7	5	2	1	4	7
##	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	-	4	8	5	4	5	4	8
##	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	-	9	5	5	4	8	9
##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		-	4	5	8		##
##	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	4	4	2	2	6	-	4	2	2	6
##	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	2	1	4	7	5	-	1	4	7
##	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	5	4	8	5	4	-	4	8
##	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	8	9	5	5	4	-	9
##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		##	4	5	8		-

グリッドで分割

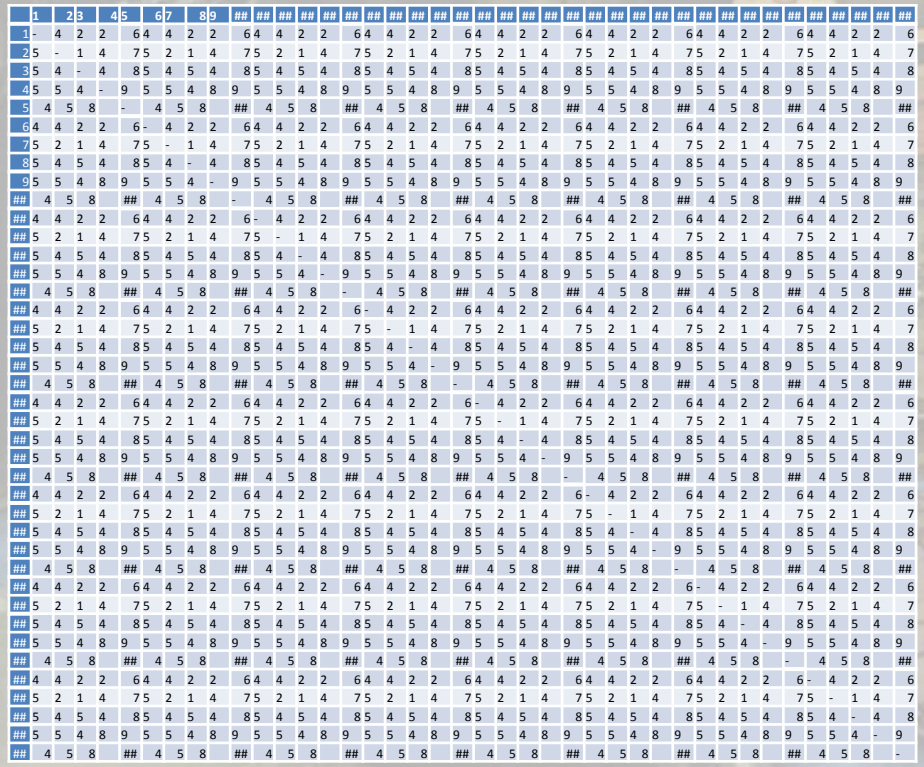
グリッドで分割



グリッドごとにテーブル作成

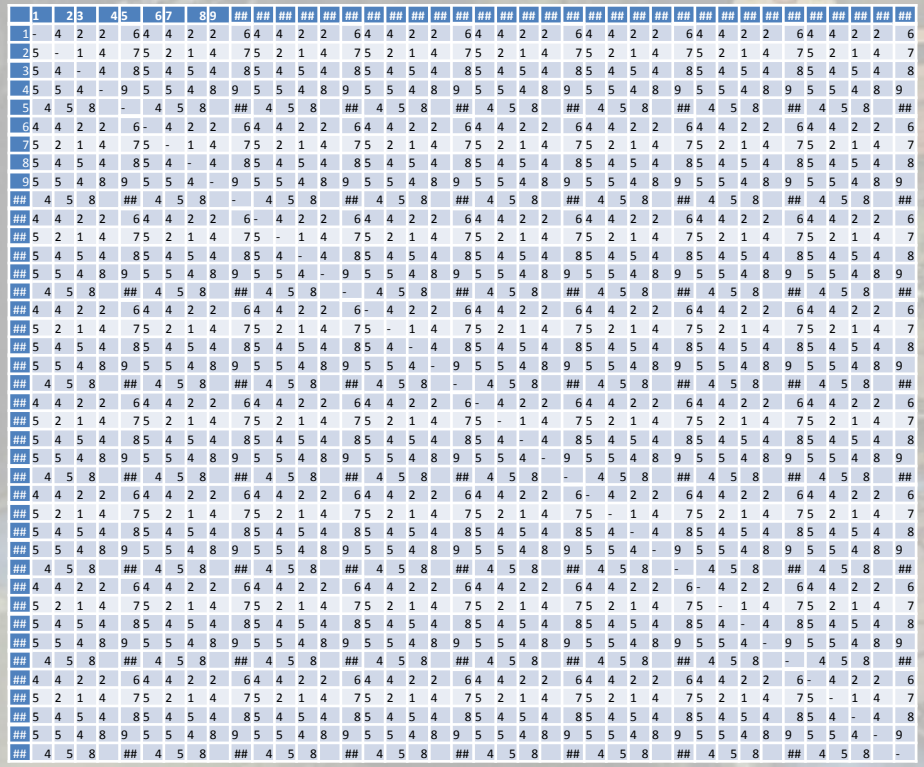
SQUARE ENIX

テーブル分割



SQUARE ENIX

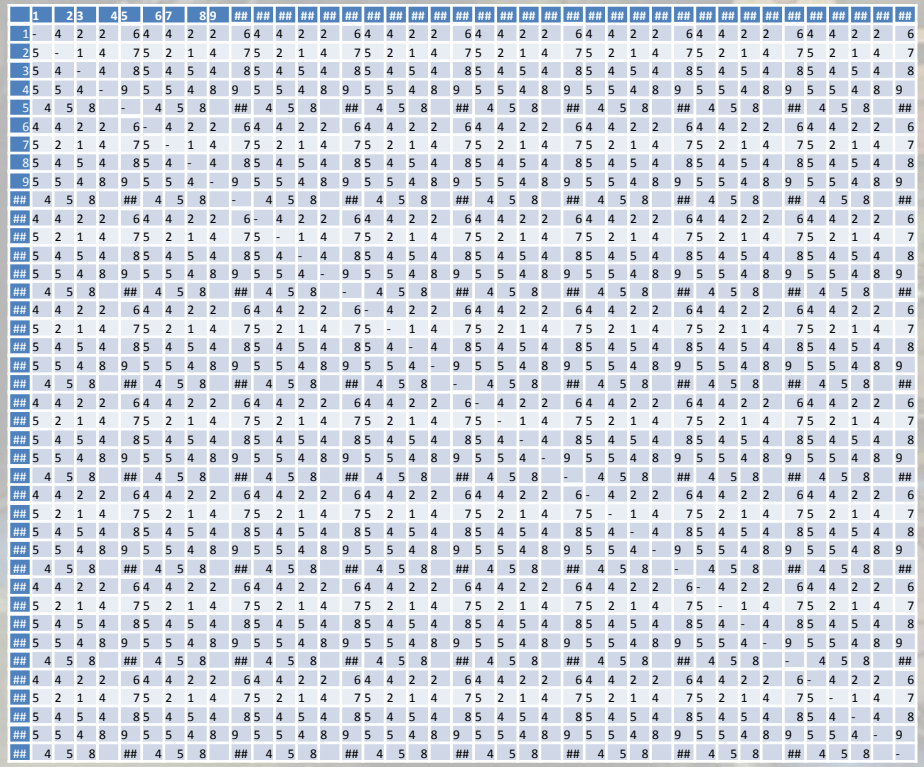
テーブル分割

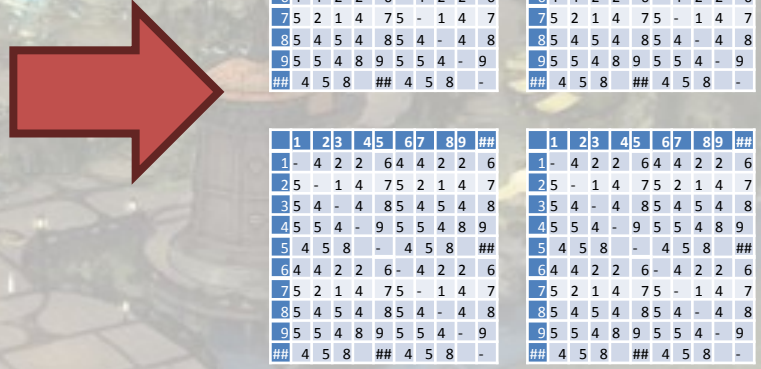


SQUARE ENIX

テーブル分割

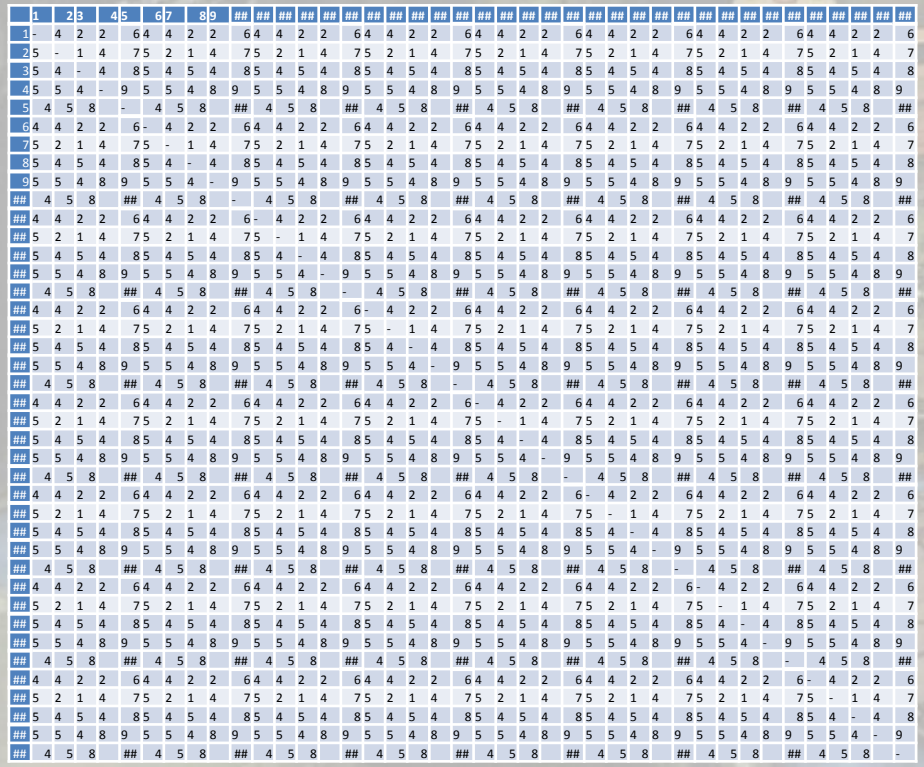
表の面積が小さくなった！





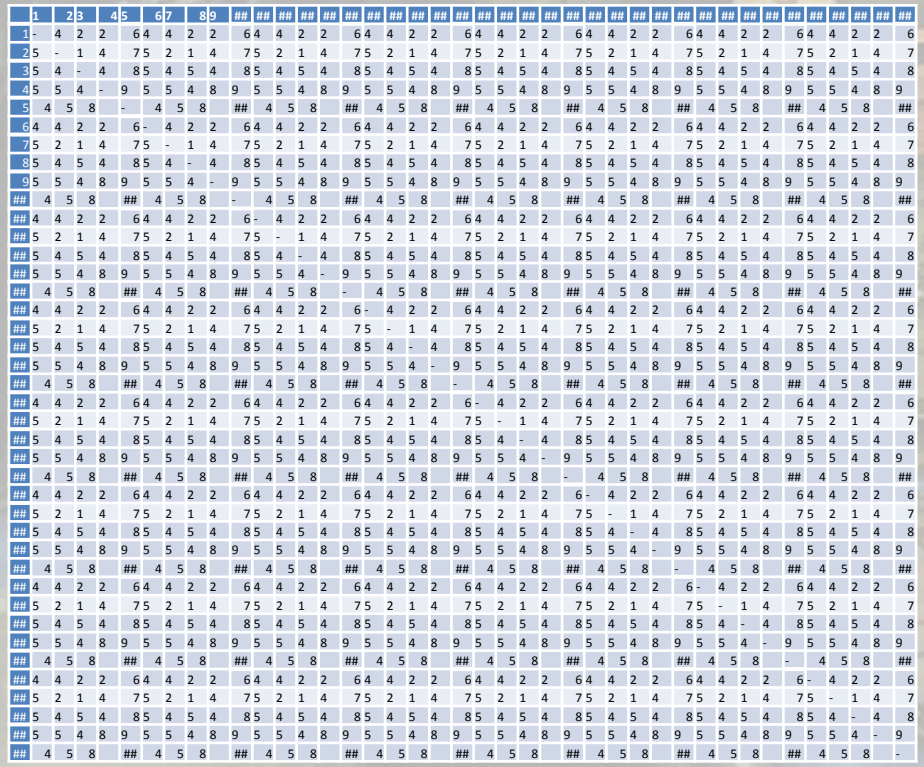
SQUARE ENIX

テーブル階層化



SQUARE ENIX

テーブル階層化

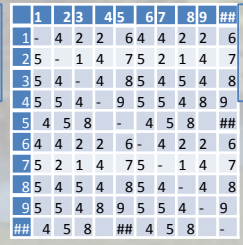


	A	B	C	D
A	-	A	B	C
B	C	-	B	A
C	B	D	-	A
D	D	A	C	-

A

B

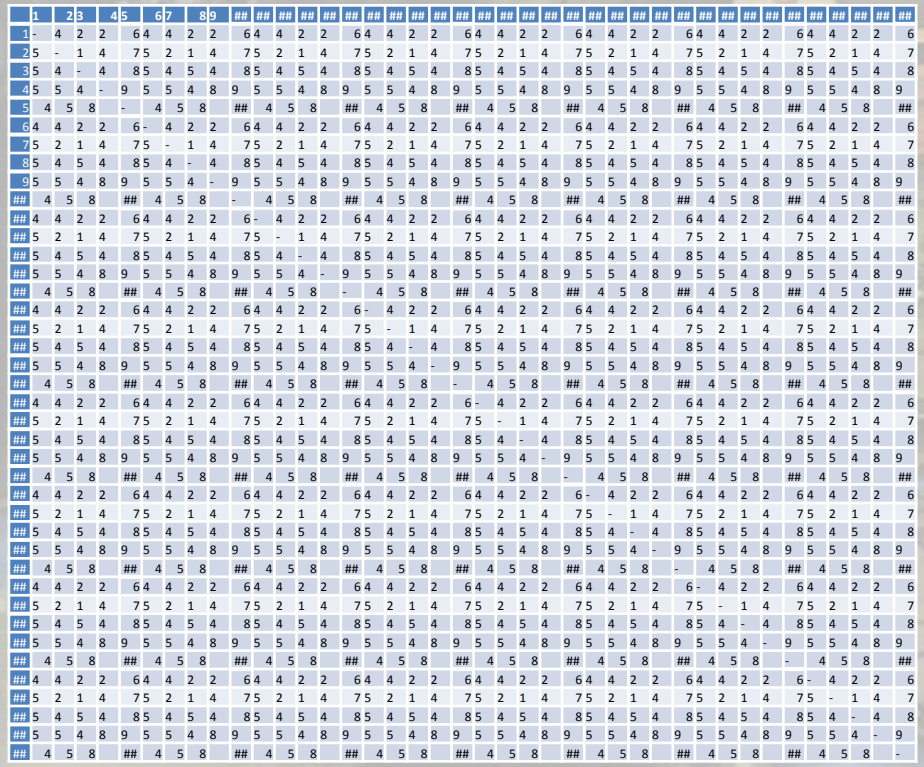
C



D

SQUARE ENIX

テーブル階層化



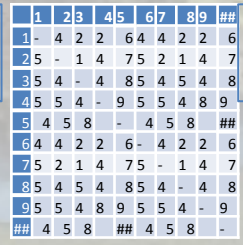
	A	B	C	D
A	-	A	B	C
B	C	-	B	A
C	B	D	-	A
D	D	A	C	-

上位テーブル

A

B

C



D

分割＆階層化してメモリ削減

- 2万メッシュを100グリッドに分けると
- 約1.1GBだったのが…

分割＆階層化してメモリ削減

- 2万メッシュを100グリッドに分けると
- 約1.1GBだったのが…

11MB！！

分割＆階層化してメモリ削減

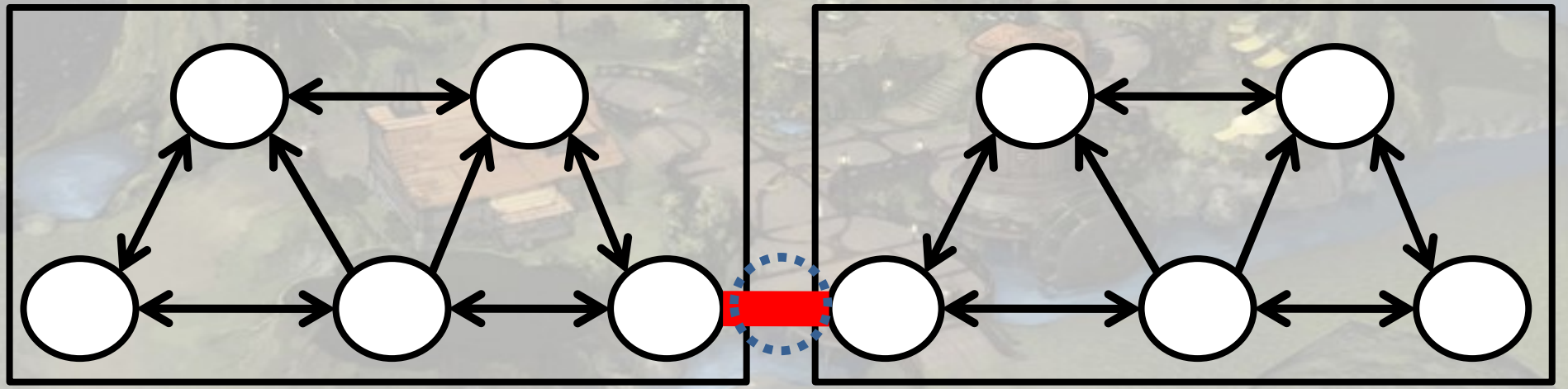
- 2万メッシュを100グリッドに分けると
- 約1.1GBだったのが…

11MB！！

- ただし、またもやデメリットが…

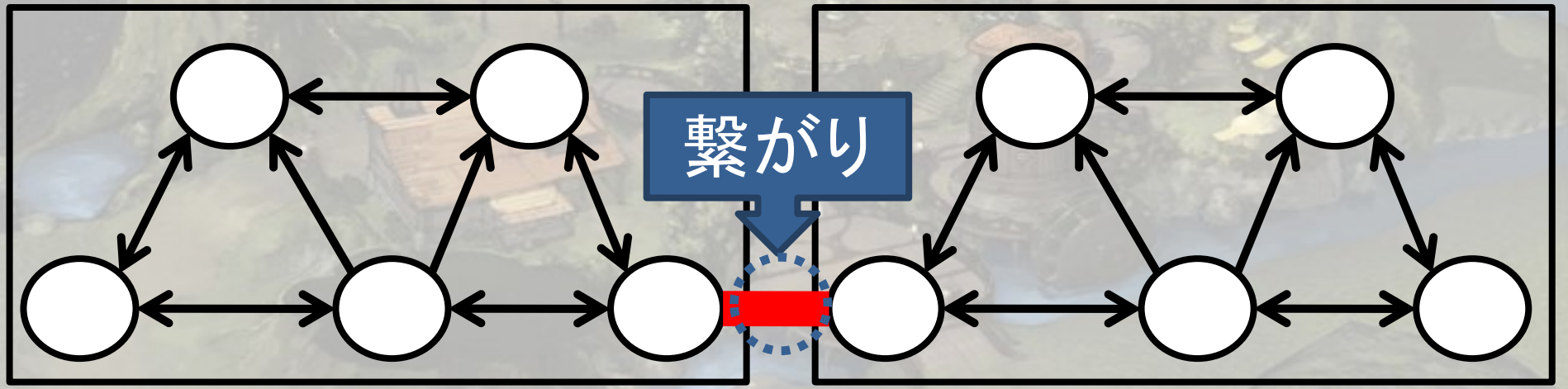
階層化した経路テーブル

- 分割したテーブル同士の「繋がり」をつくらないといけない

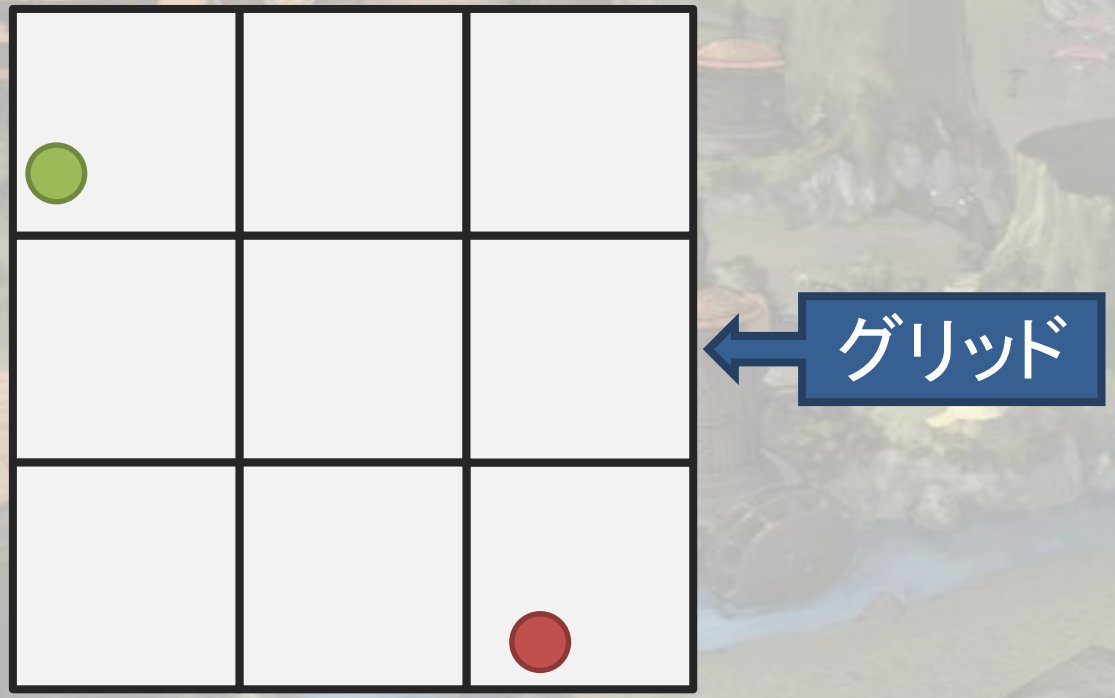


階層化した経路テーブル

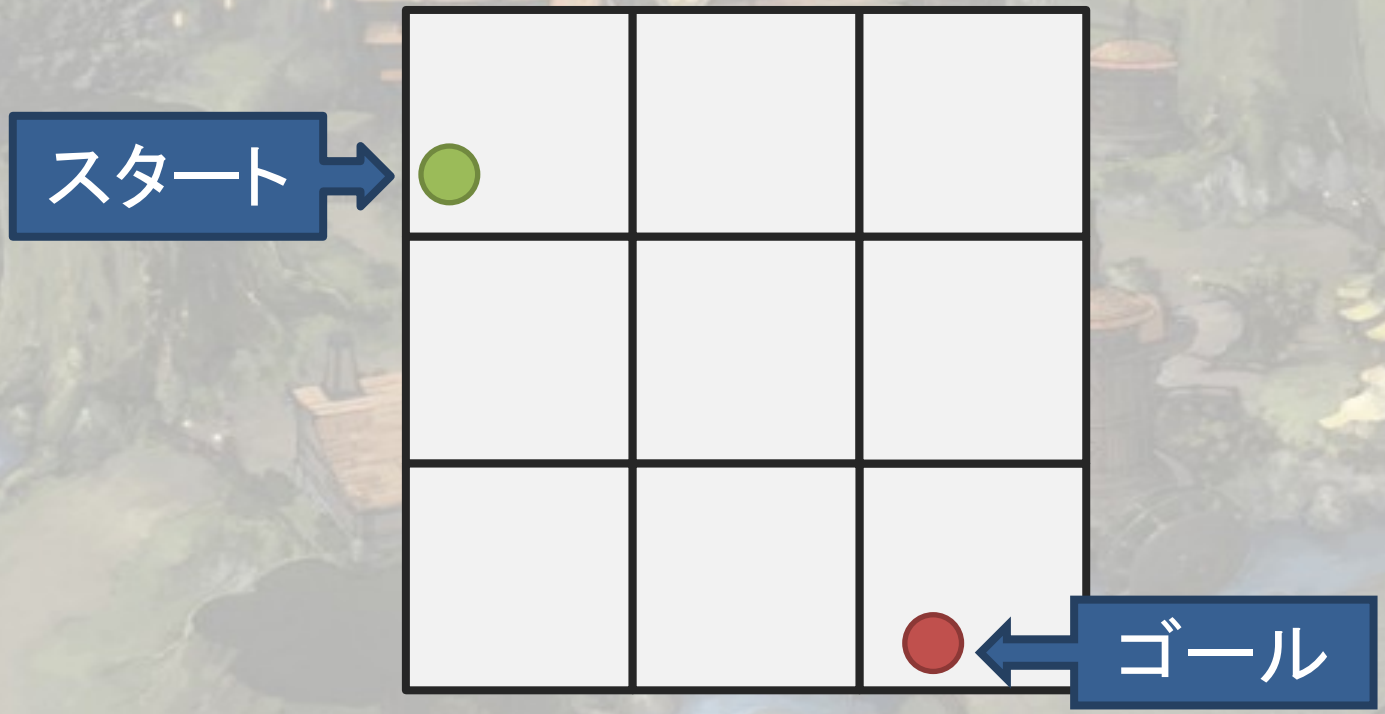
- 分割したテーブル同士の「繋がり」をつくらないといけない



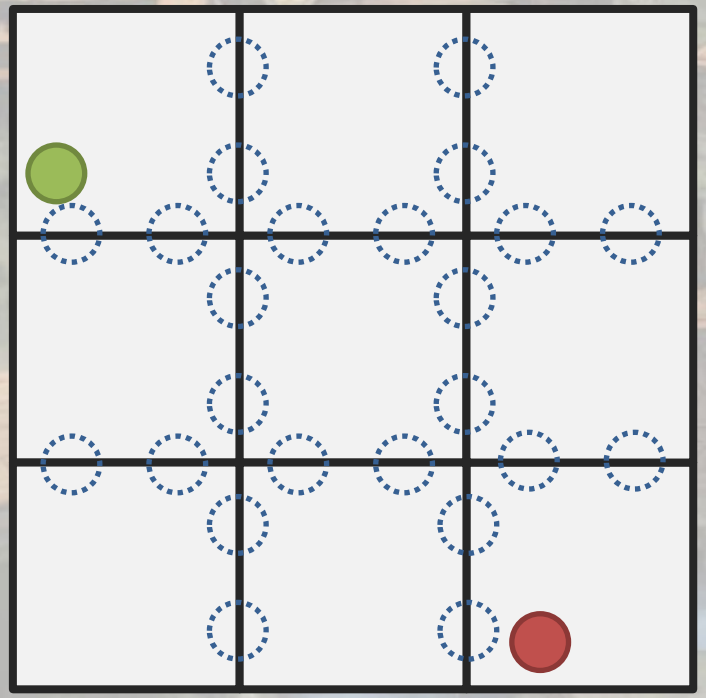
繋がりポイントをたどる



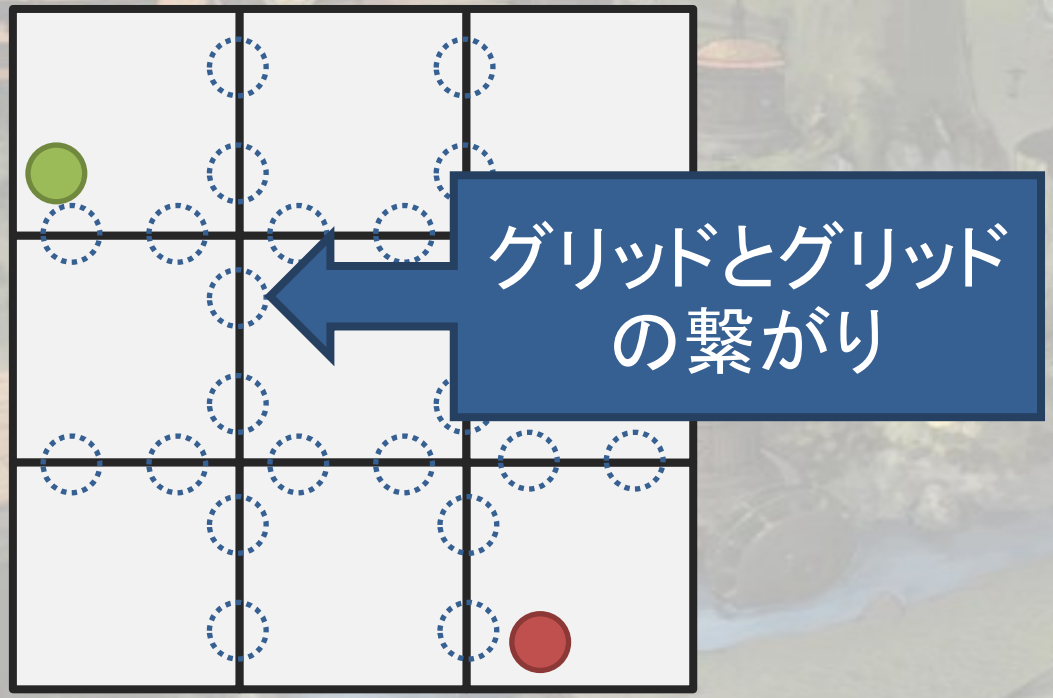
繋がりポイントをたどる



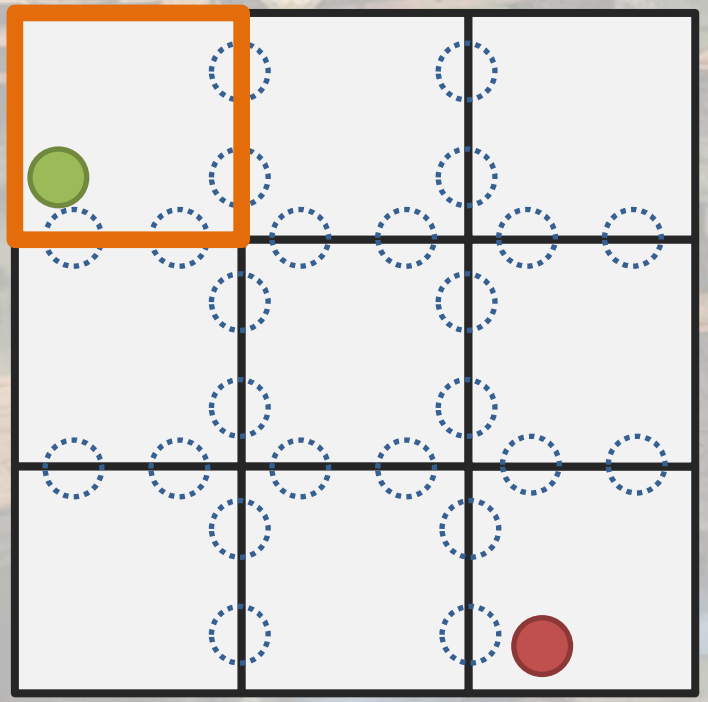
繋がりポイントをたどる



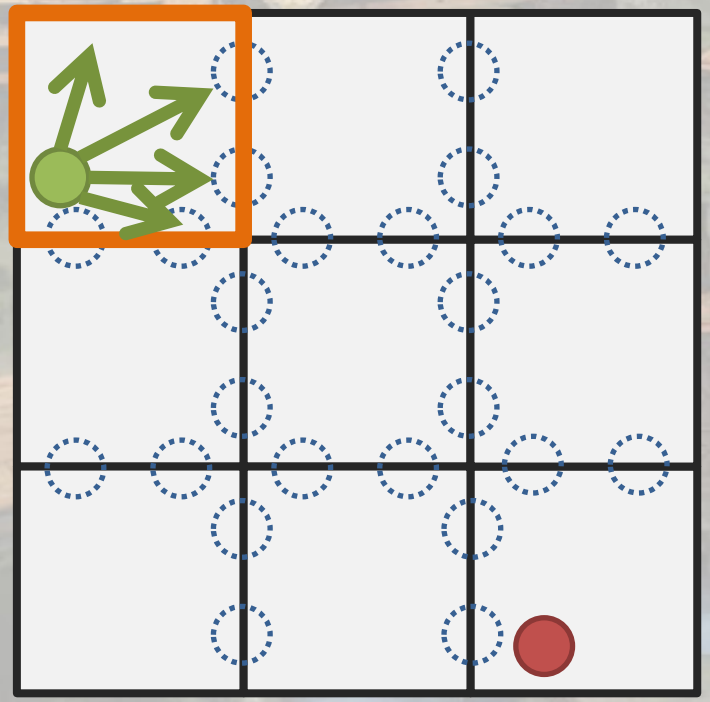
繋がりポイントをたどる



移動できるのはグリッド内だけ

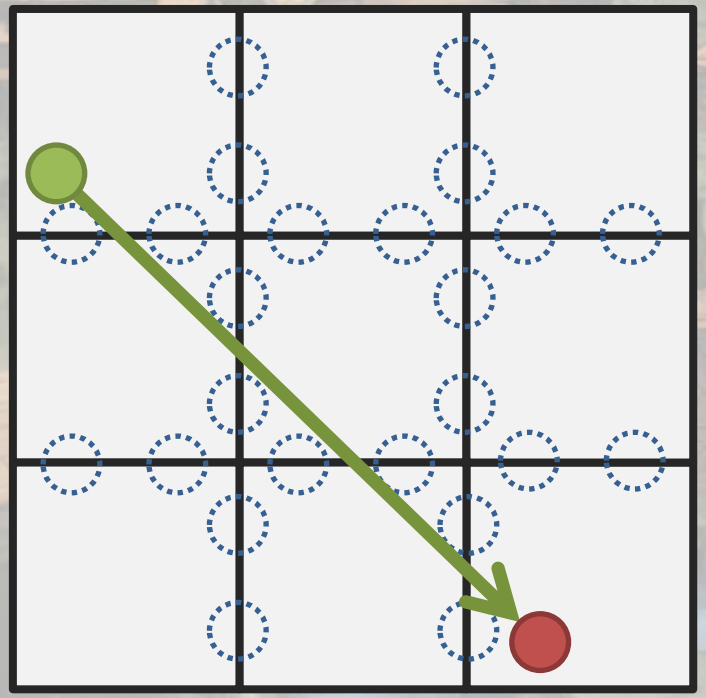


移動できるのはグリッド内だけ

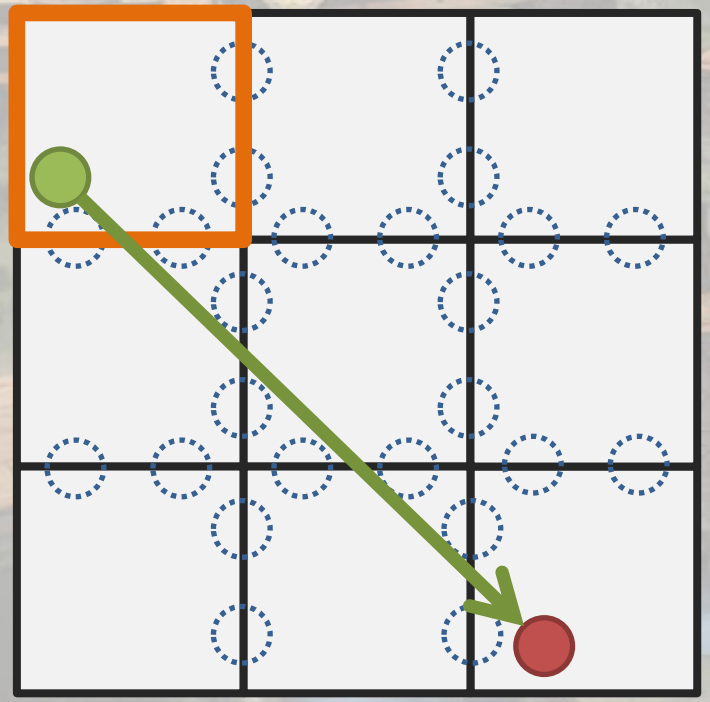


SQUARE ENIX

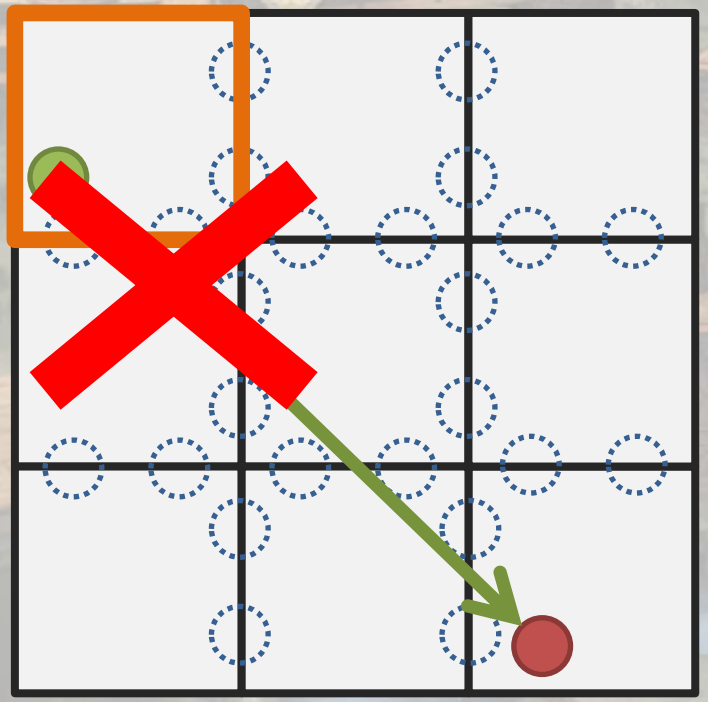
まっすぐ行くと…



グリッド外に出てしまうのでダメ

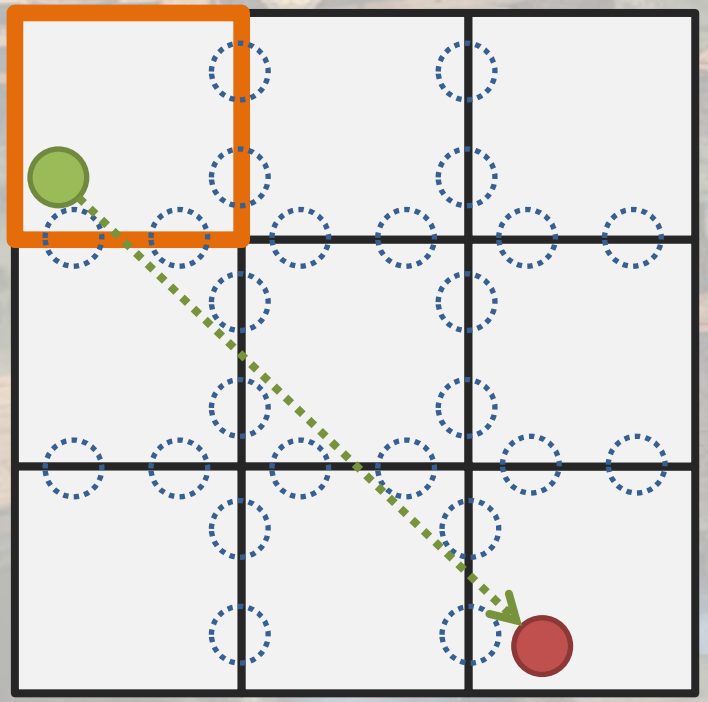


グリッド外に出てしまうのでダメ

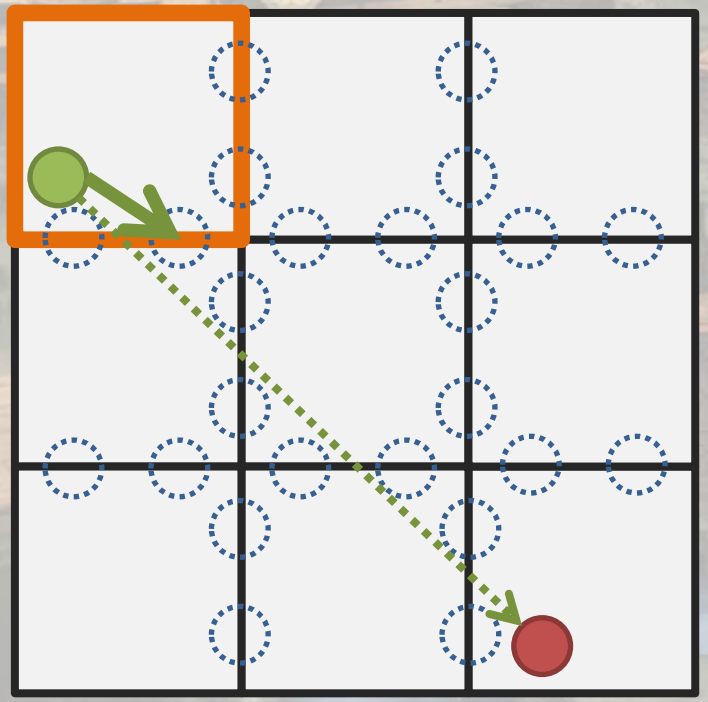


SQUARE ENIX

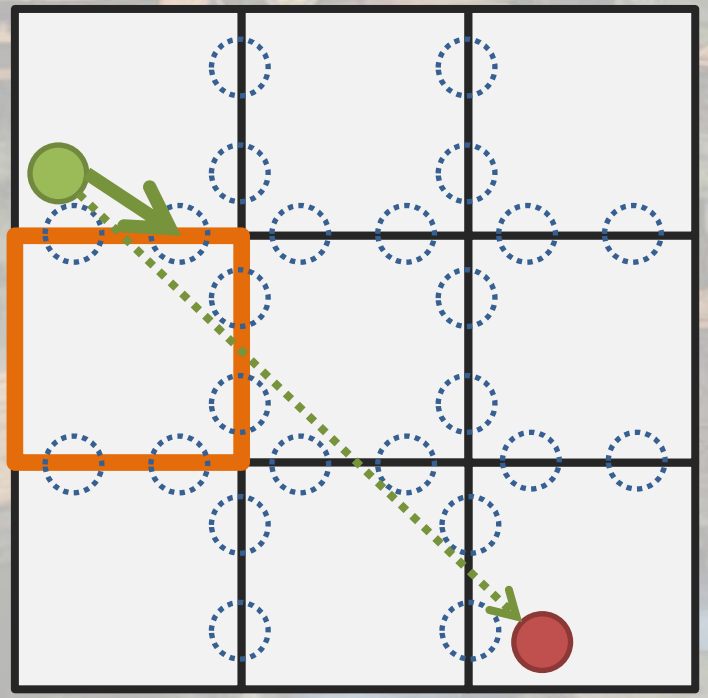
繋がりポイントをたどる



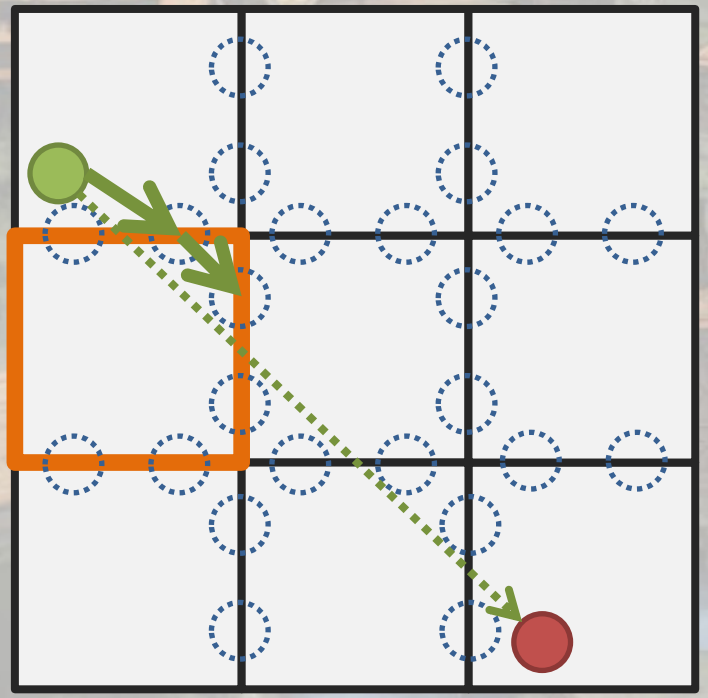
繋がりポイントをたどる



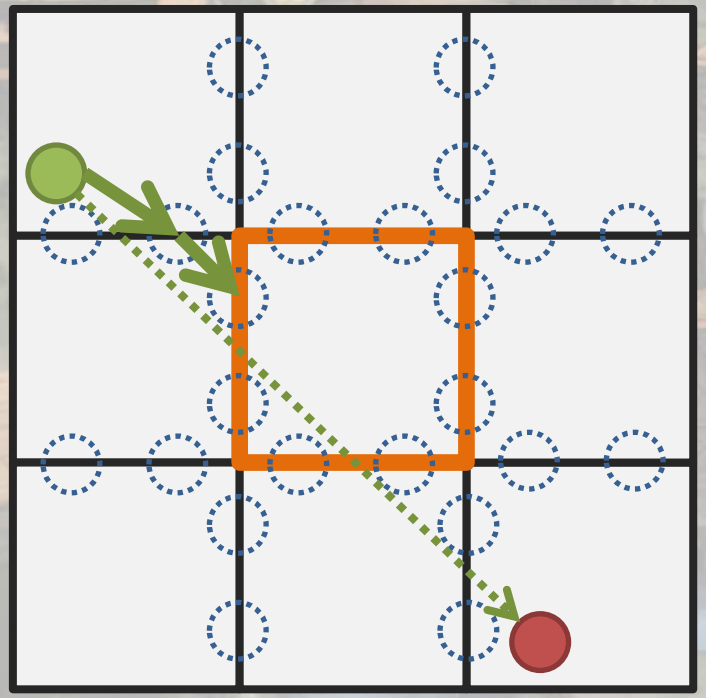
繋がりポイントをたどる



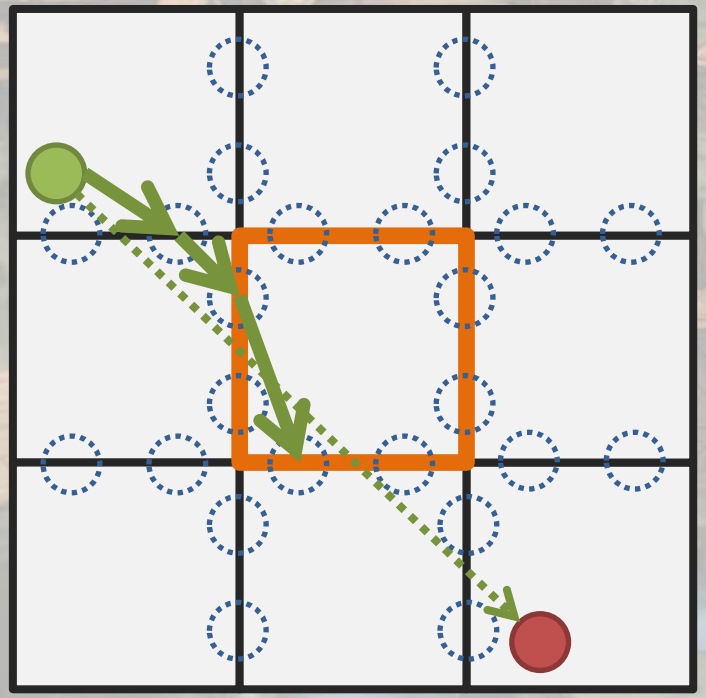
繋がりポイントをたどる



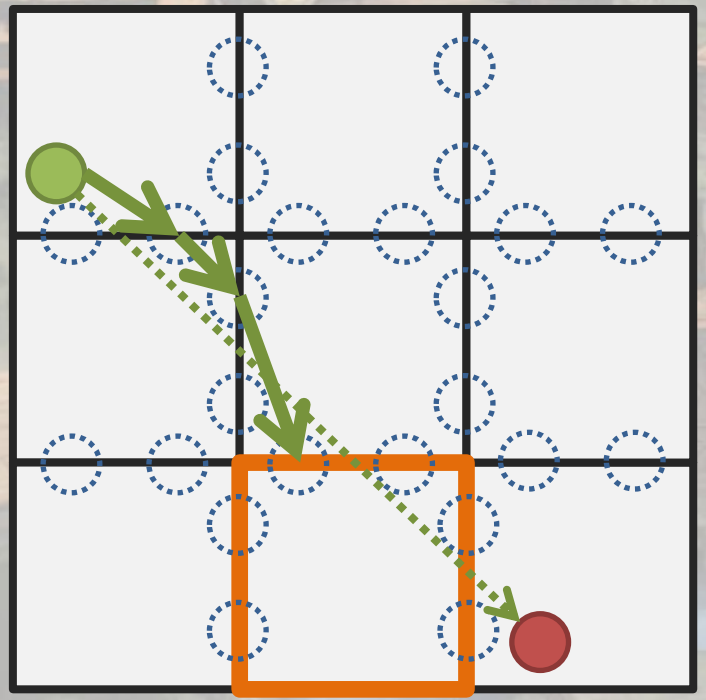
繋がりポイントをたどる



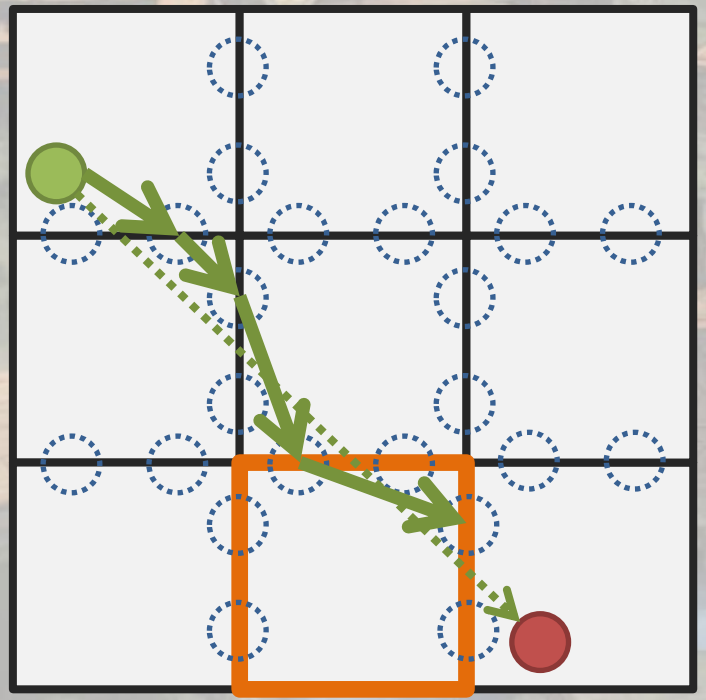
繋がりポイントをたどる



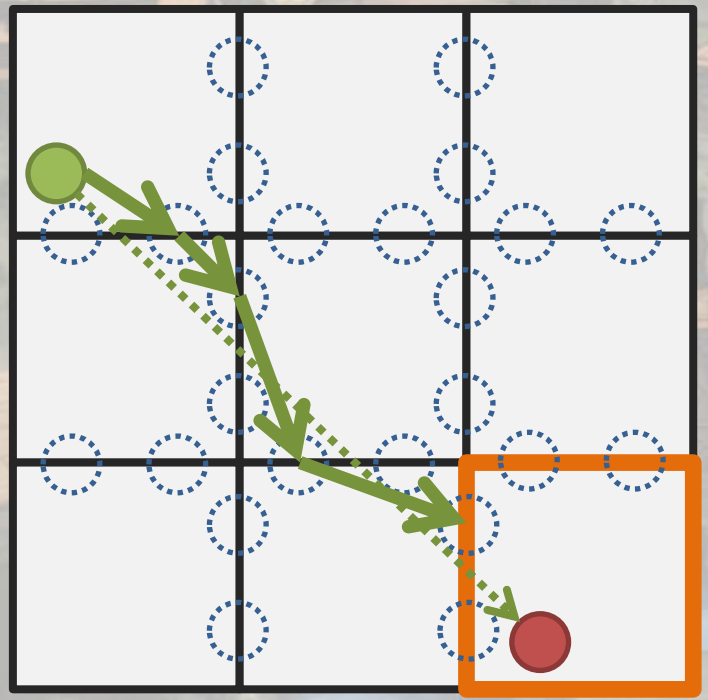
繋がりポイントをたどる



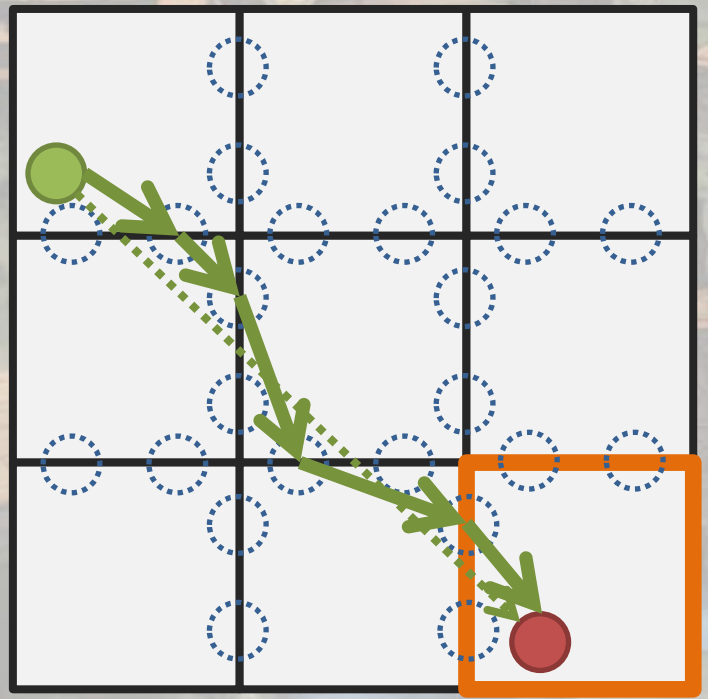
繋がりポイントをたどる



繋がりポイントをたどる

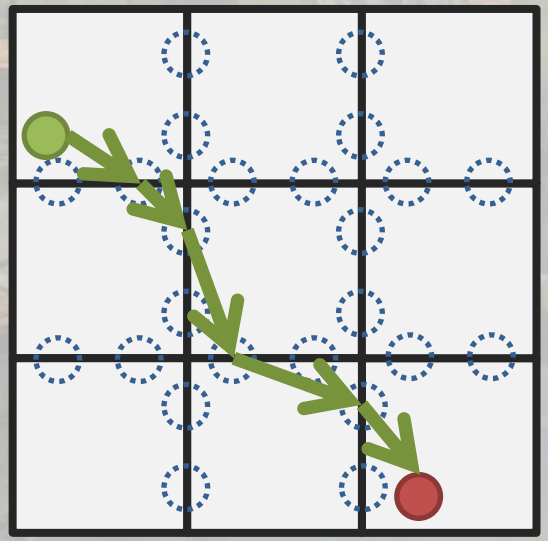


繋がりポイントをたどる



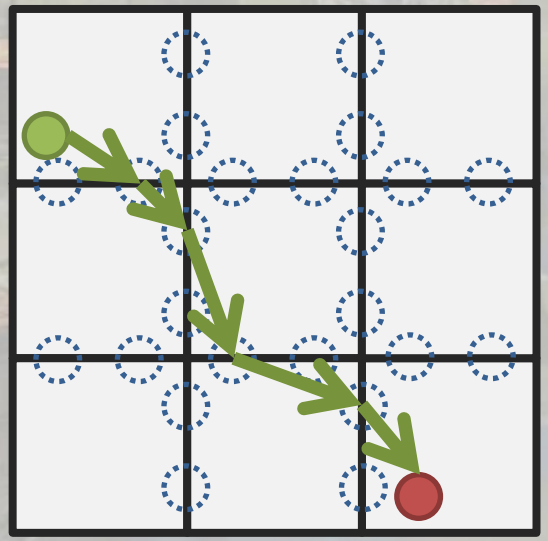
階層化した経路テーブル

- 「繋がり」ポイントを通ると、ぎこちない経路になる



階層化した経路テーブル

- 「繋がり」ポイントを通ると、ぎこちない経路になる
- 開けたマップには向かない



分割＆階層化の結果

○経路テーブルで、CPU負荷が軽くなった！

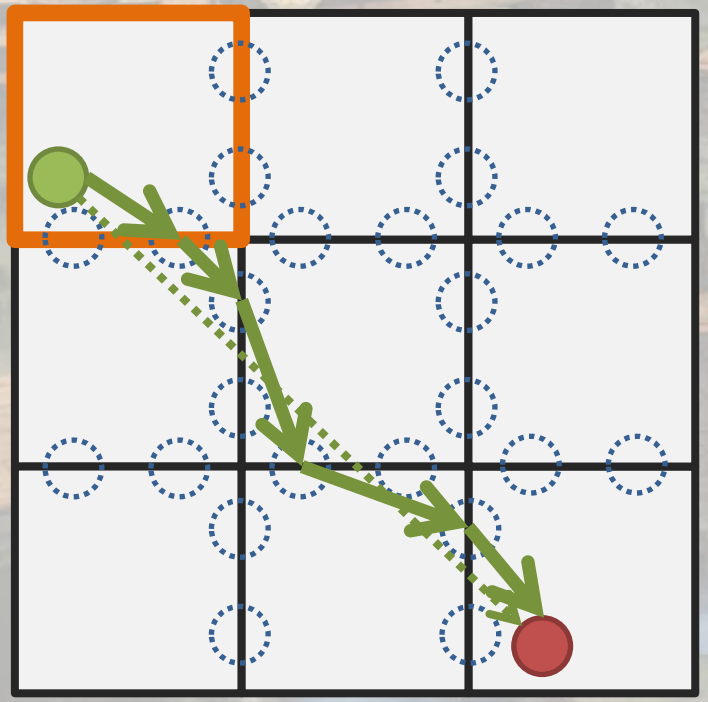
○階層化で、メモリ消費量も小さくなった！

×経路がぎこちなくなった

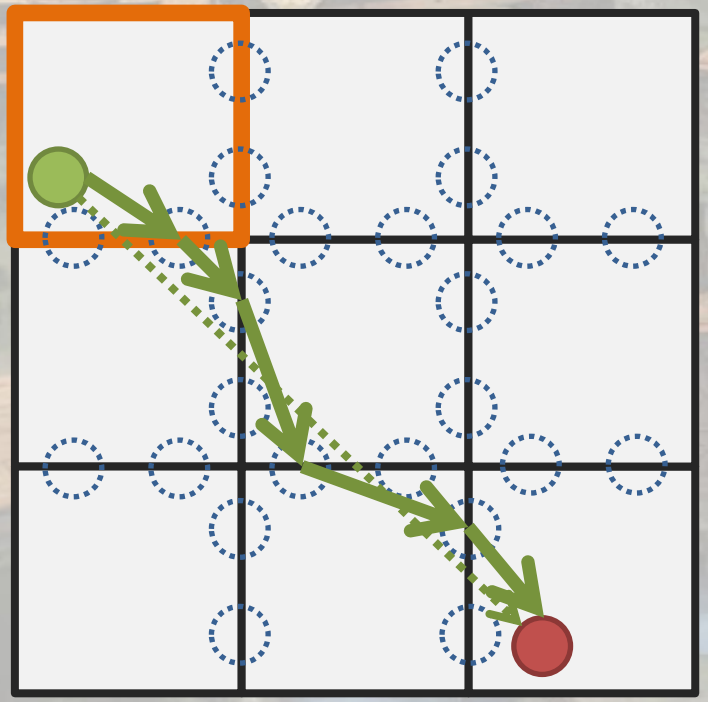
第2部 経路探索を軽くする

- 経路テーブルをたどる
- 分割＆階層化してメモリ削減
- 隣接化して自然な経路探索

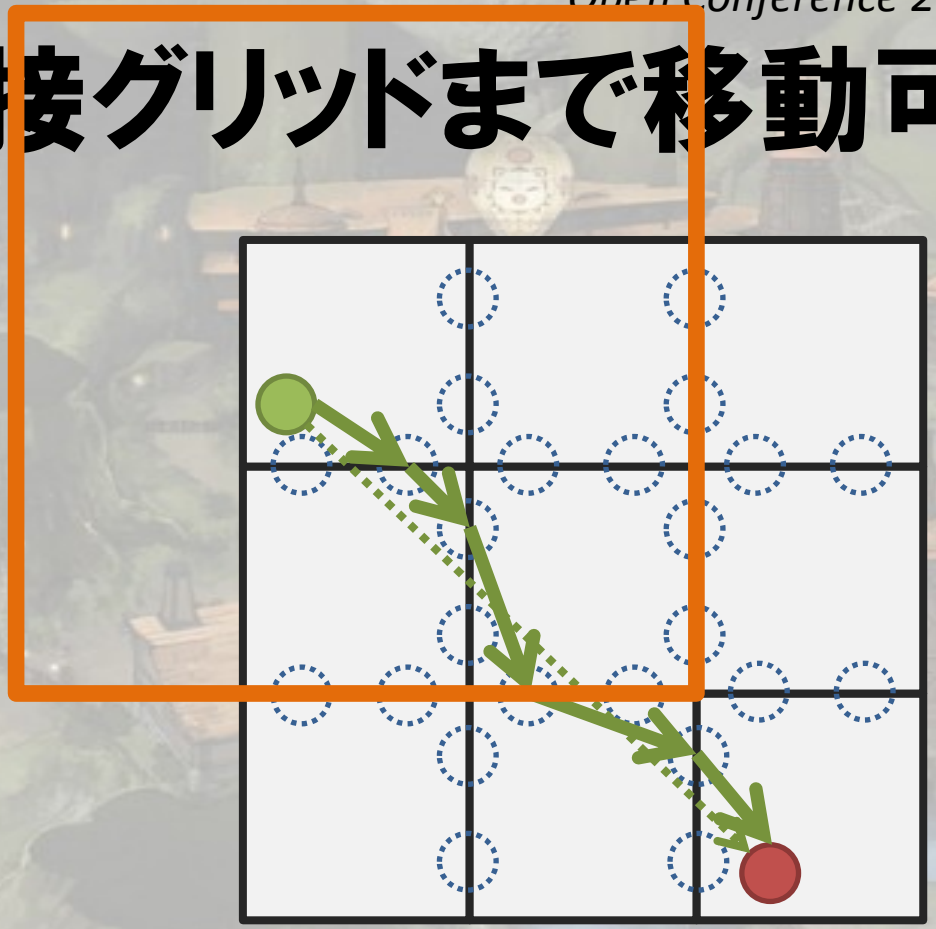
1つのグリッド内で移動していたのを



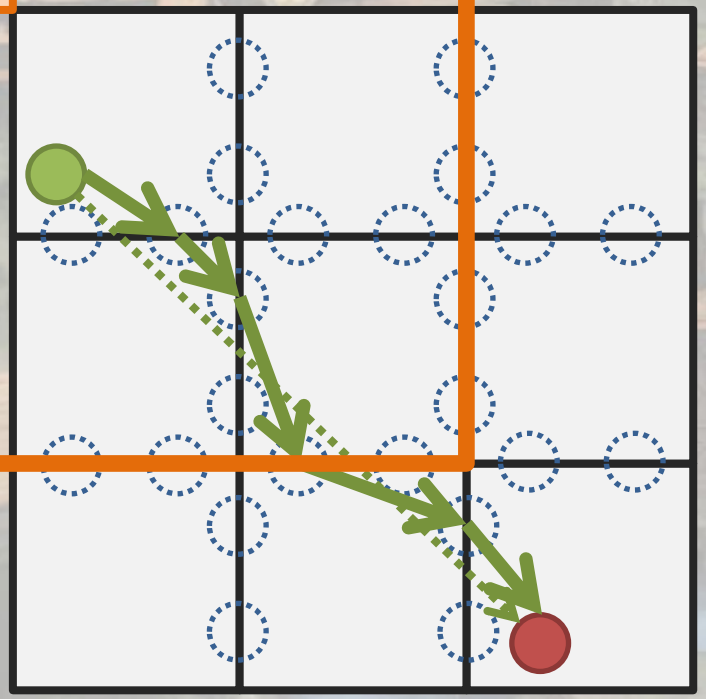
隣接グリッドまで移動可能にする



隣接グリッドまで移動可能にする

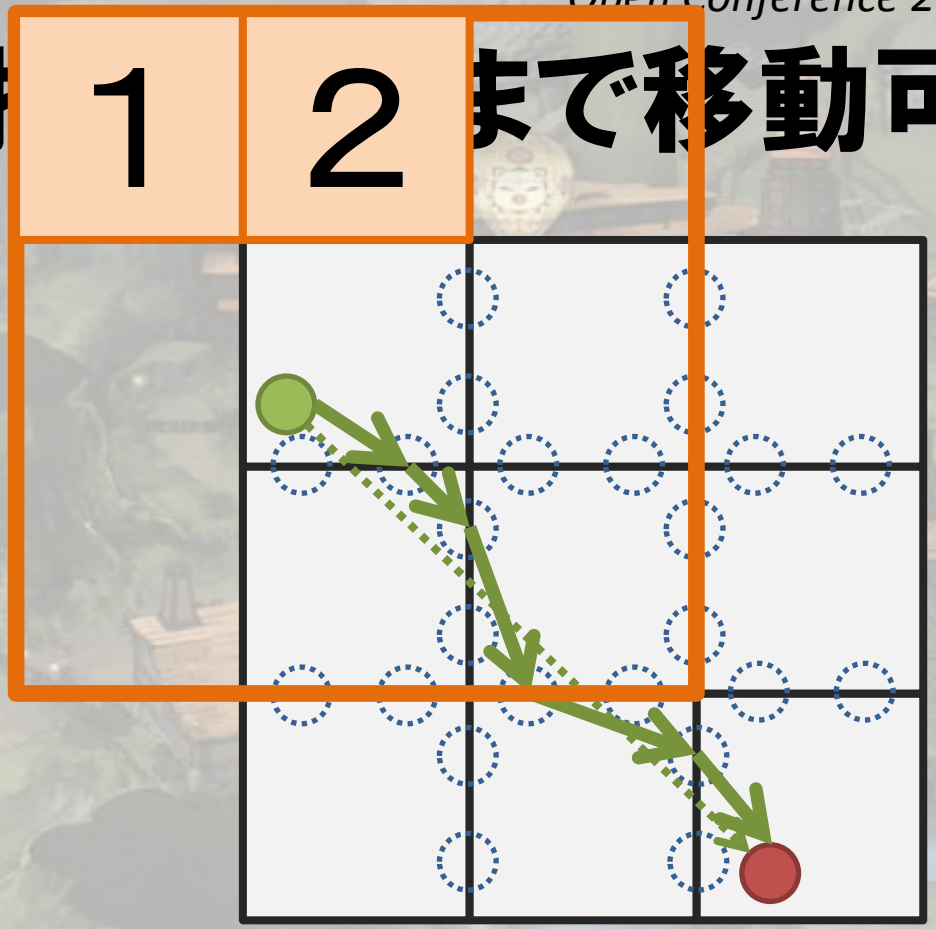


隣接 1 ッドまで移動可能にする



SQUARE ENIX

隣接1 2まで移動可能にする

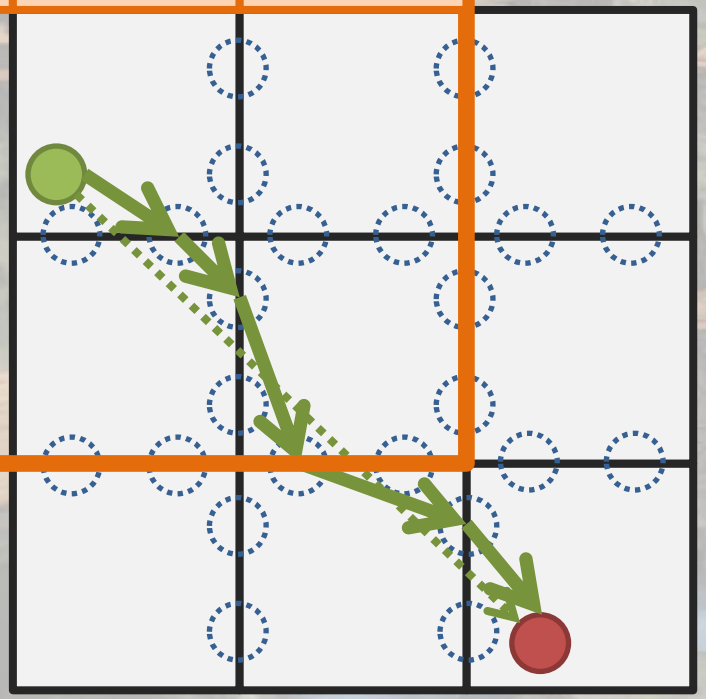


SQUARE ENIX

隣接

1	2	3

移動可能にする

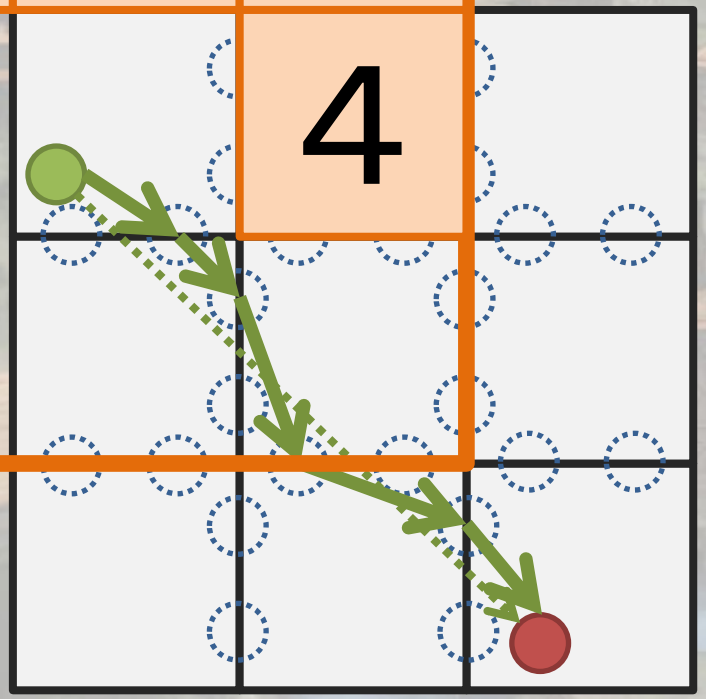


SQUARE ENIX

隣接

1	2	3
		4

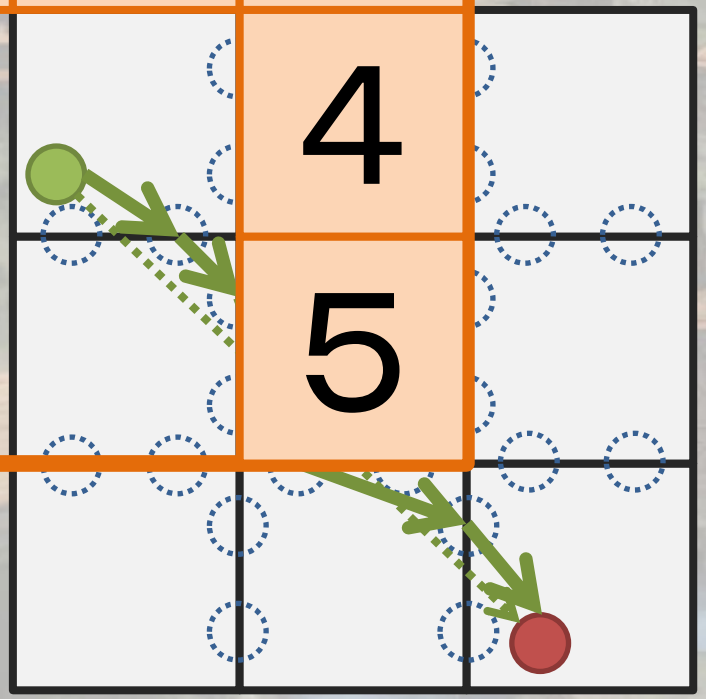
移動可能にする



SQUARE ENIX

隣接

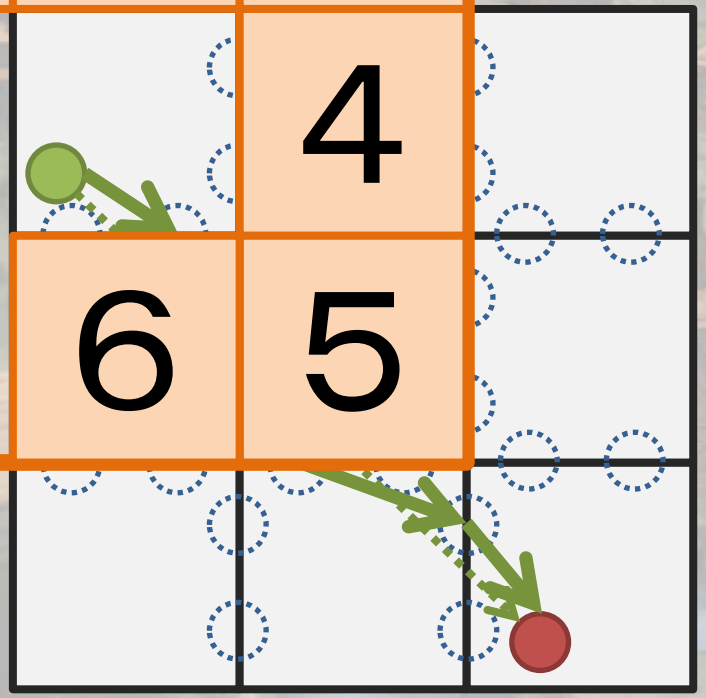
1	2	3
		4
		5



移動可能にする

隣接

1	2	3
		4
	6	5



移動可能にする

SQUARE ENIX

隣接 移動可能にする

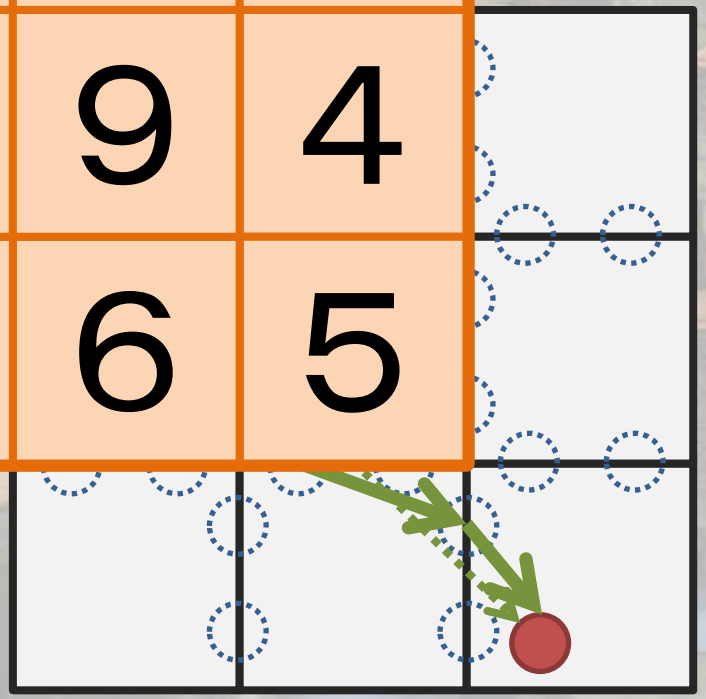
SQUARE ENIX

隣接 動可能にする

SQUARE ENIX

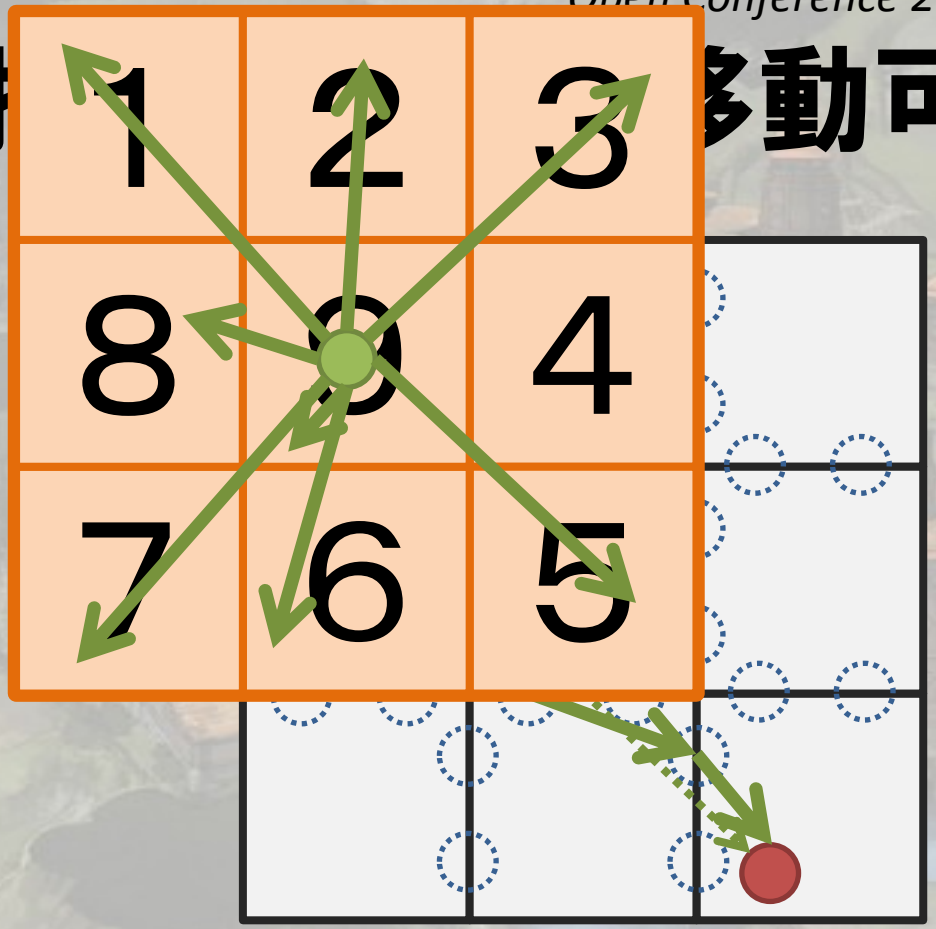
隣接

1	2	3
8	9	4
7	6	5



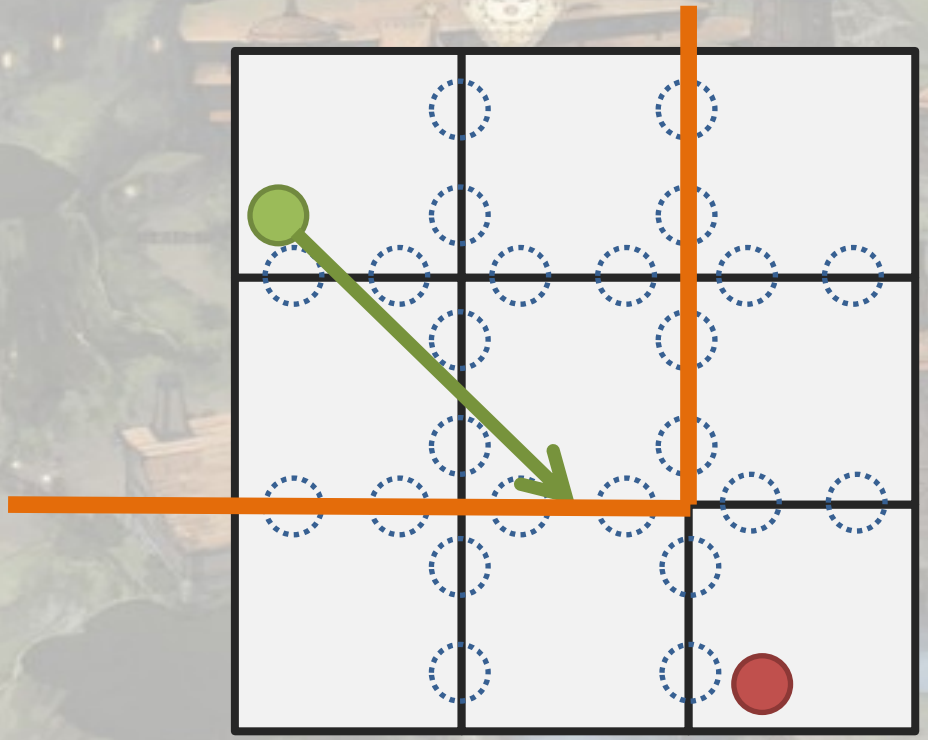
移動可能にする

隣接8方向に移動可能にする

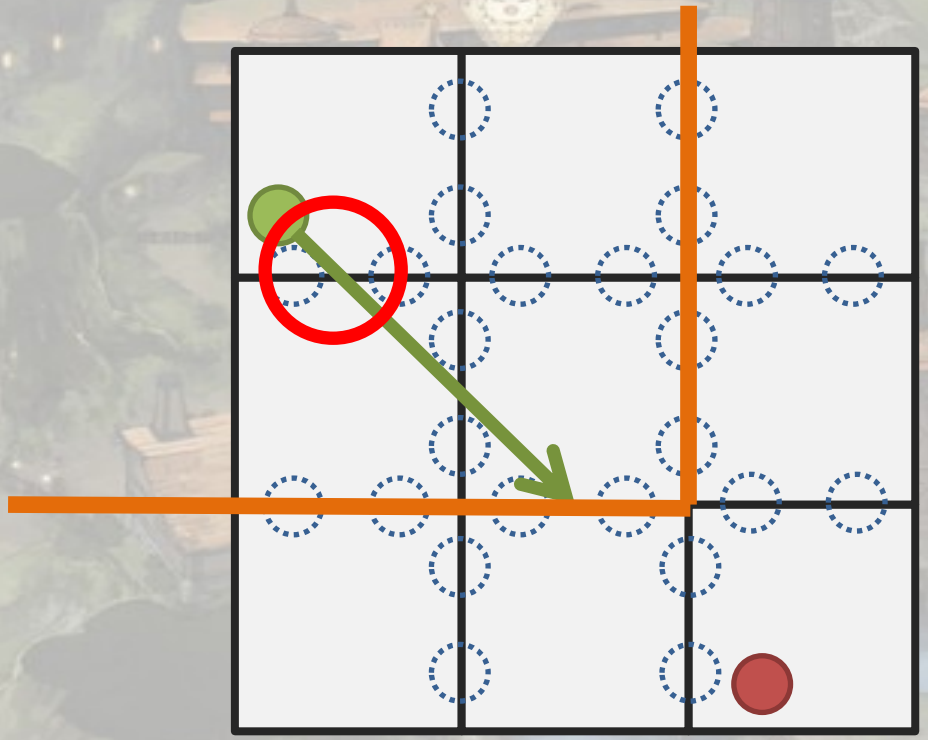


隣接グリッドまで移動可能にする

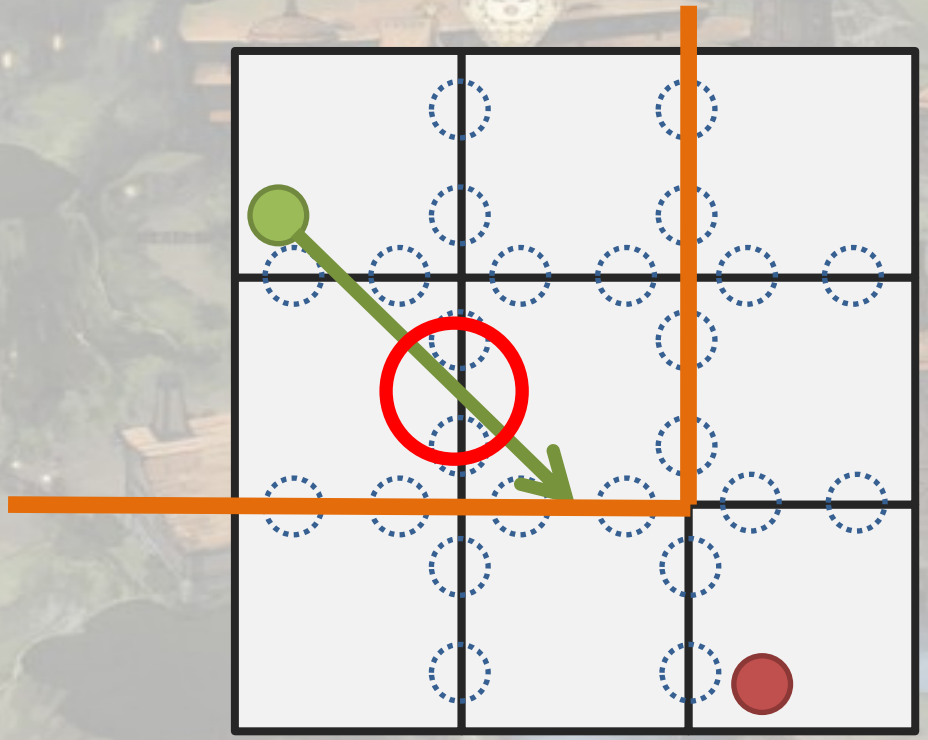
繋がりポイントを通らなくて済む



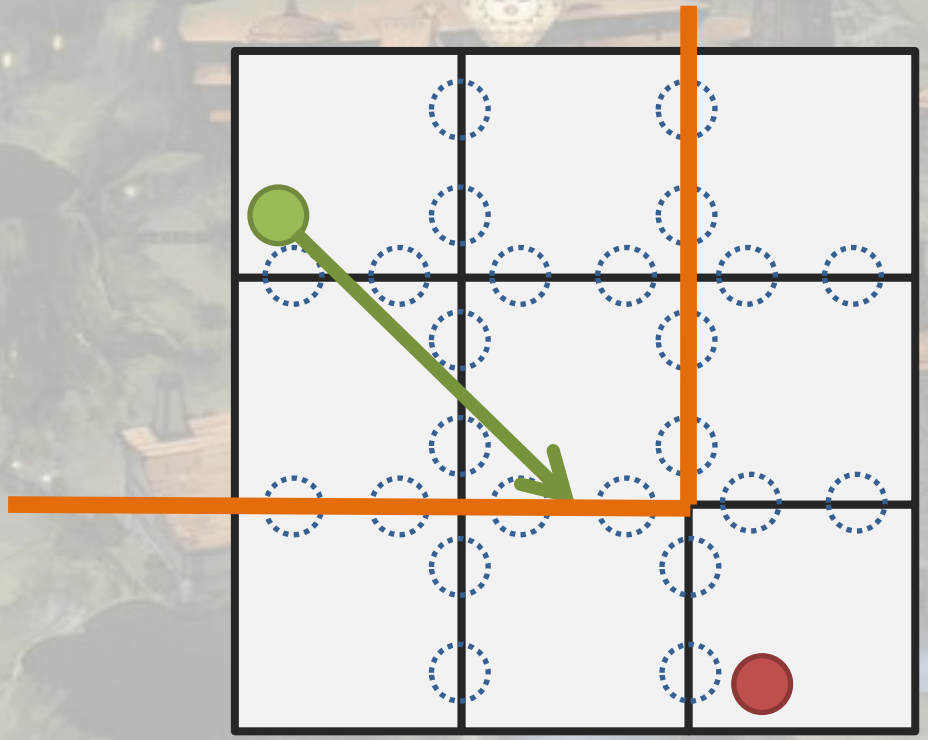
繋がりポイントを通らなくて済む



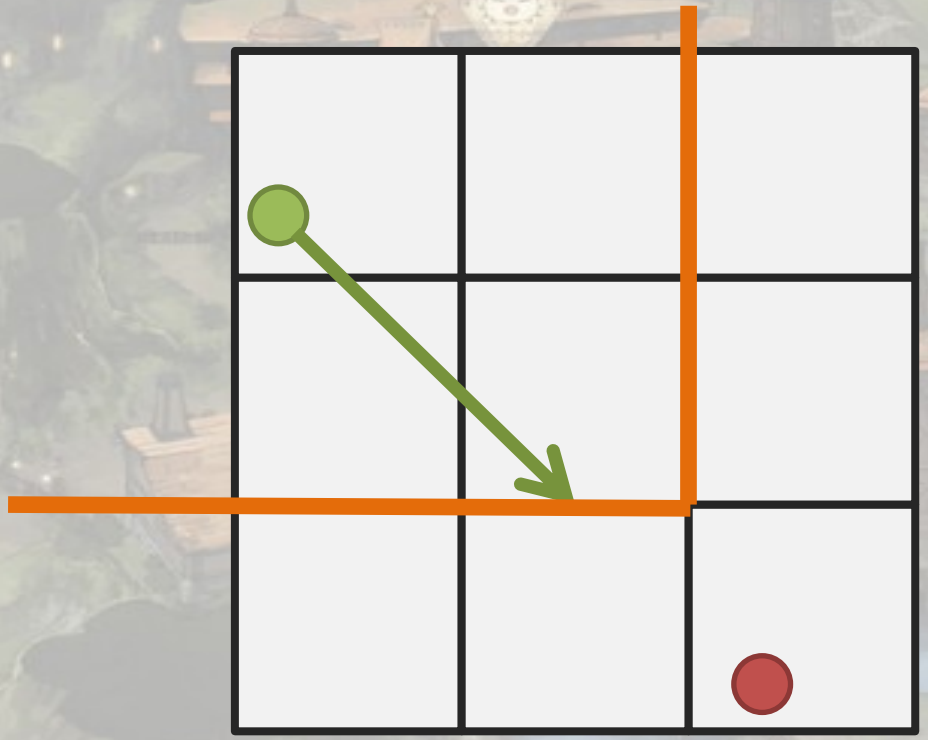
繋がりポイントを通らなくて済む



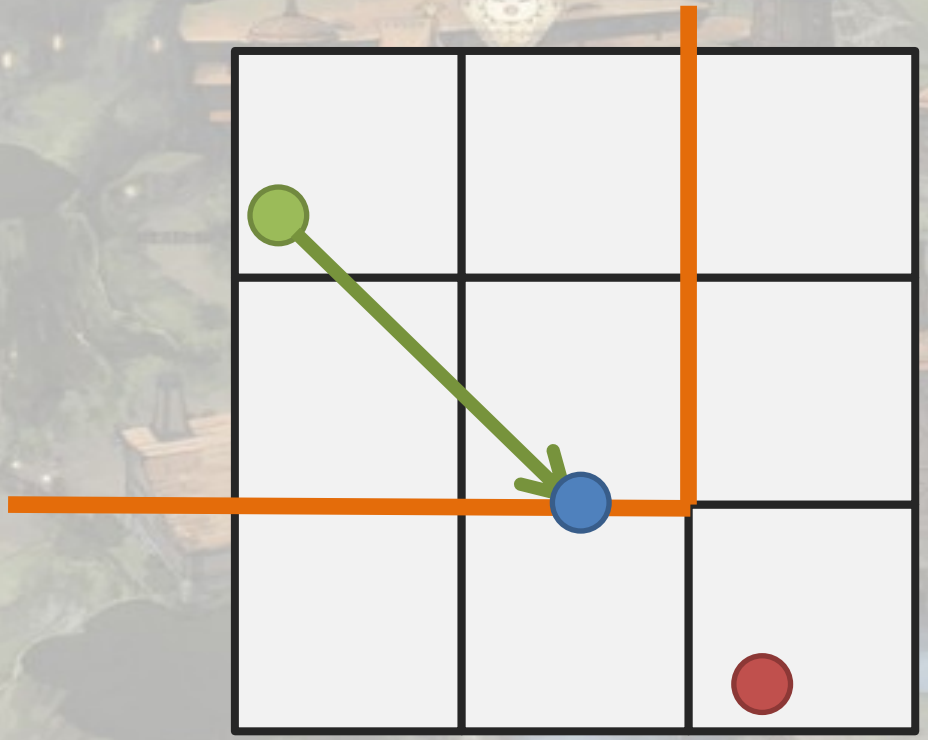
繋がりポイントが不要になる！



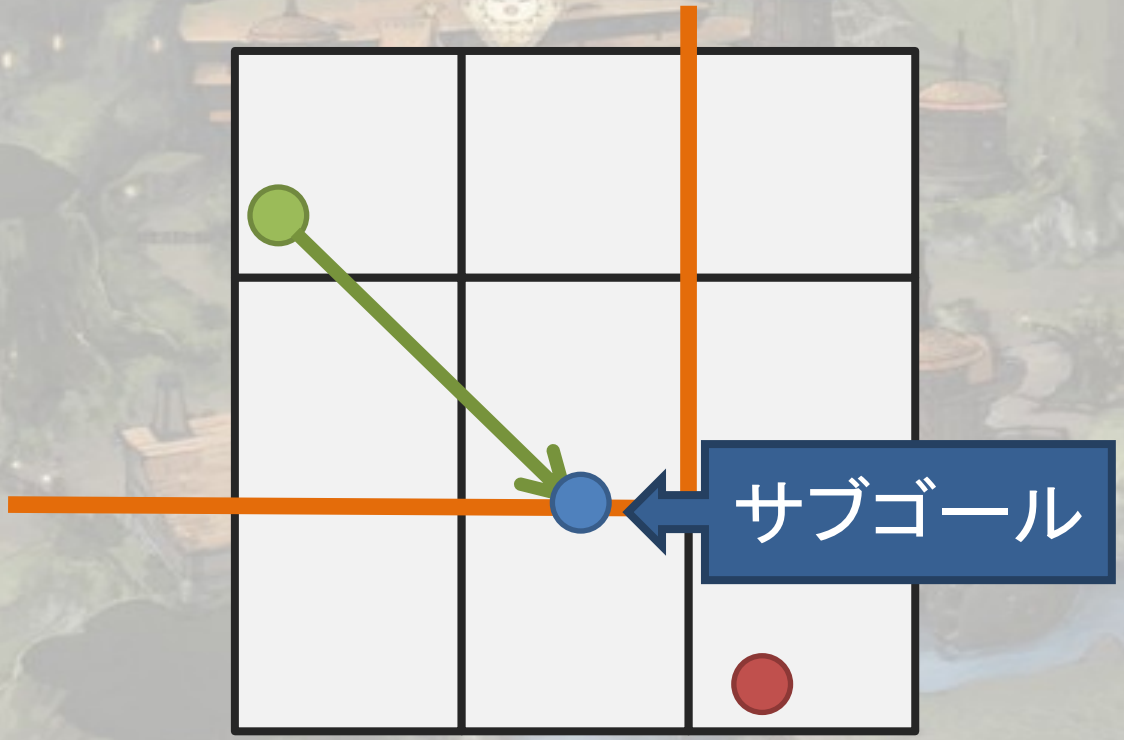
繋がりポイントが不要になる！



サブゴールを一時的な目標にする

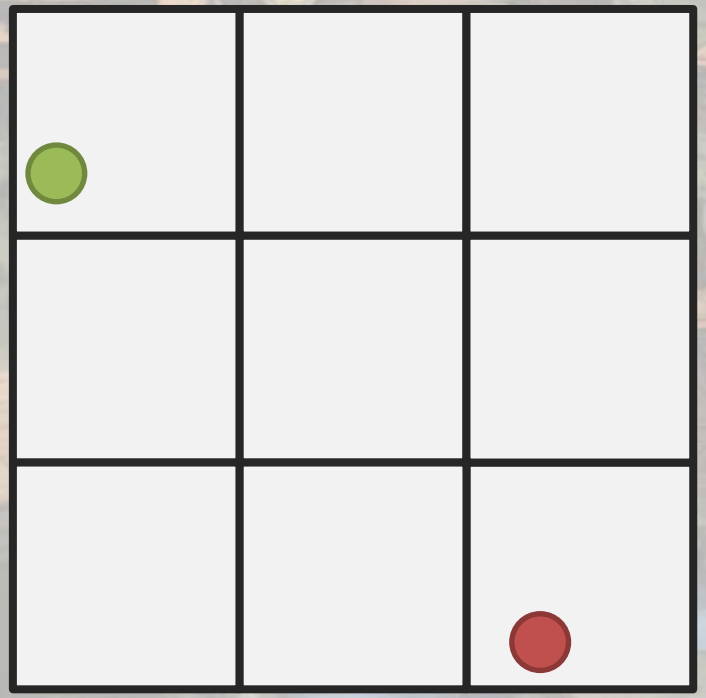


サブゴールを一時的な目標にする

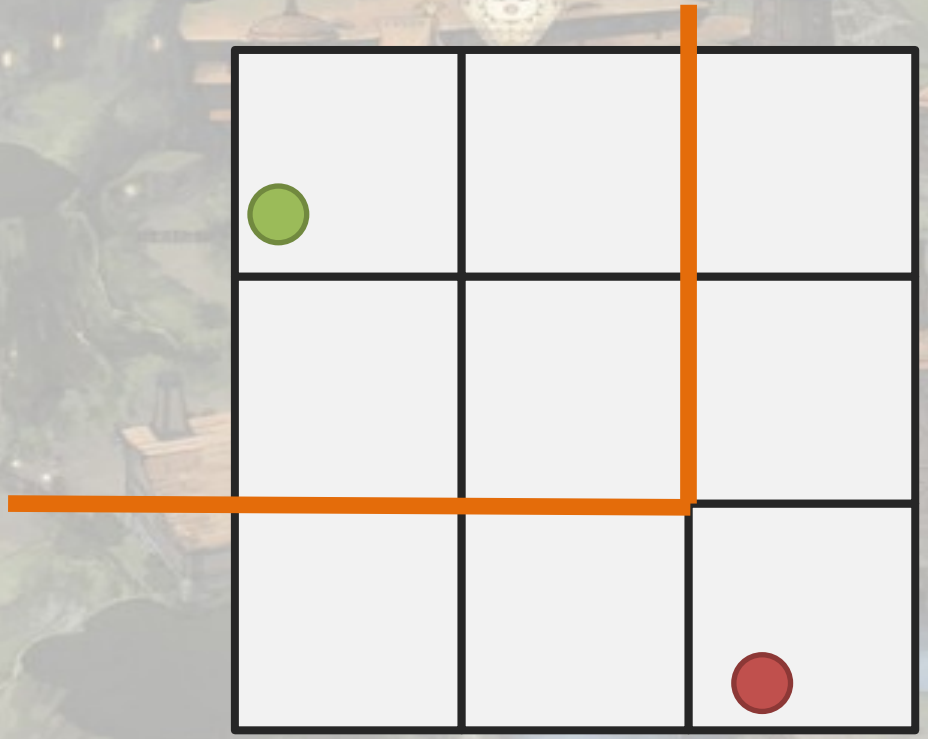


SQUARE ENIX

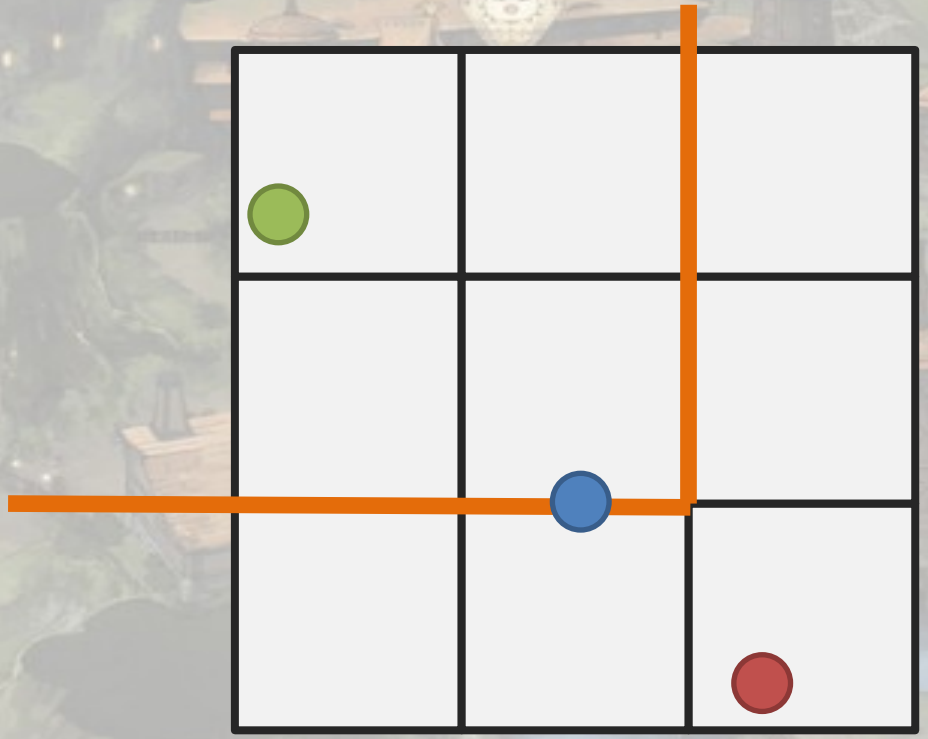
辿る



テーブル用意

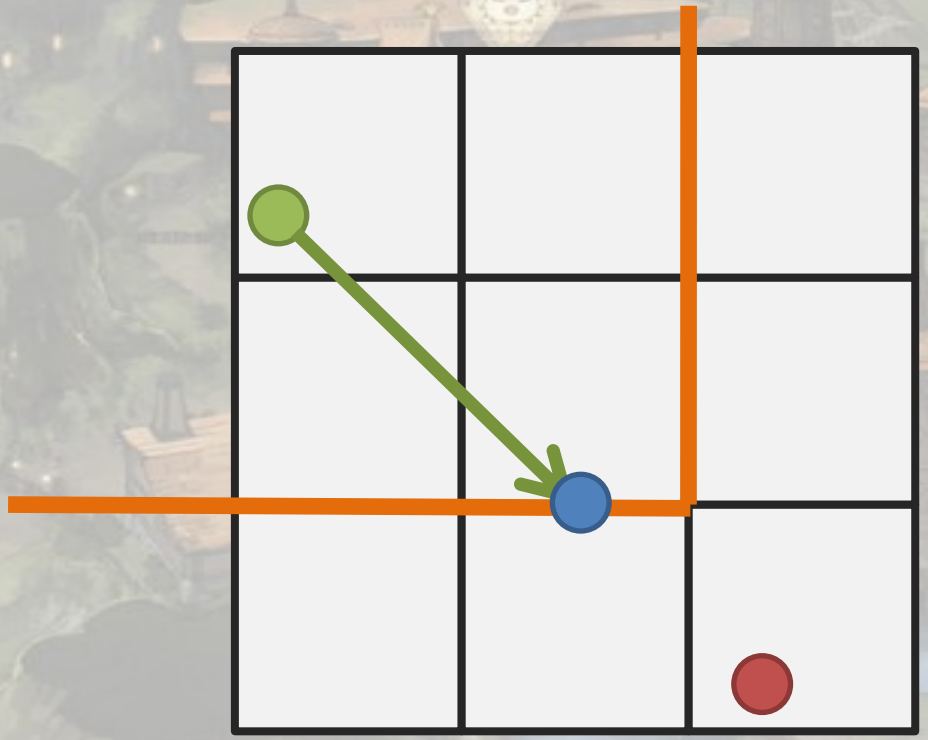


サブゴールを置く



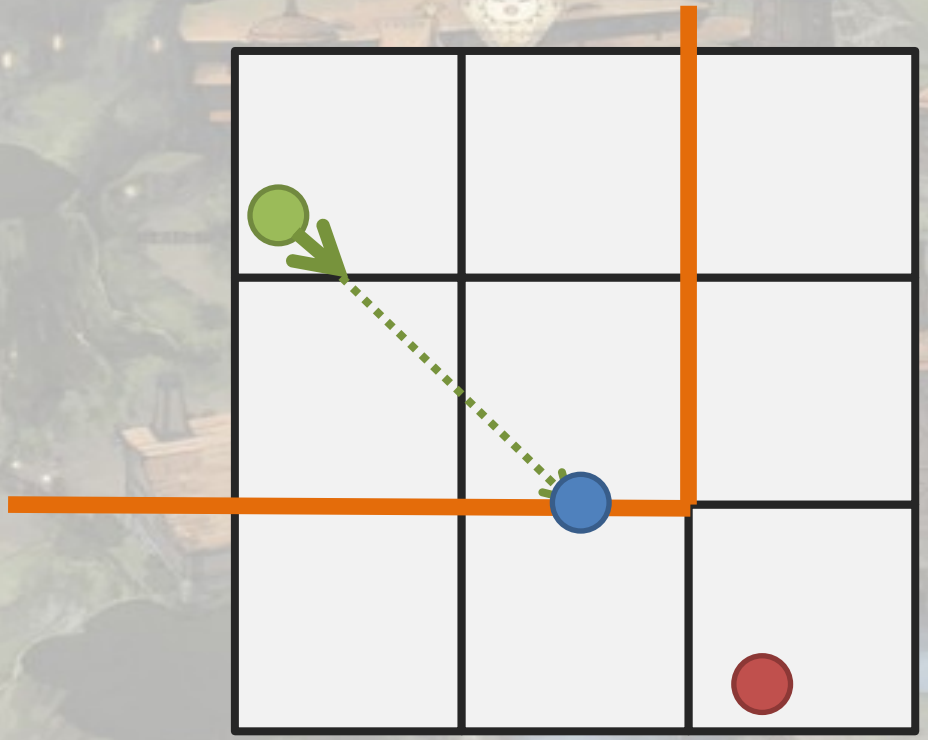
SQUARE ENIX

辿る

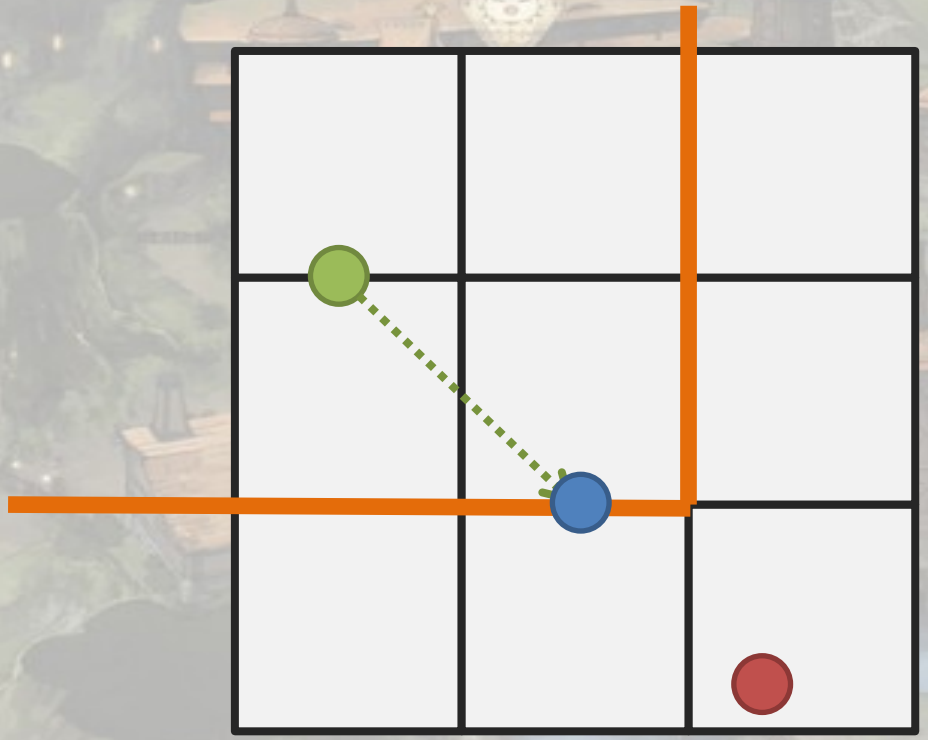


SQUARE ENIX

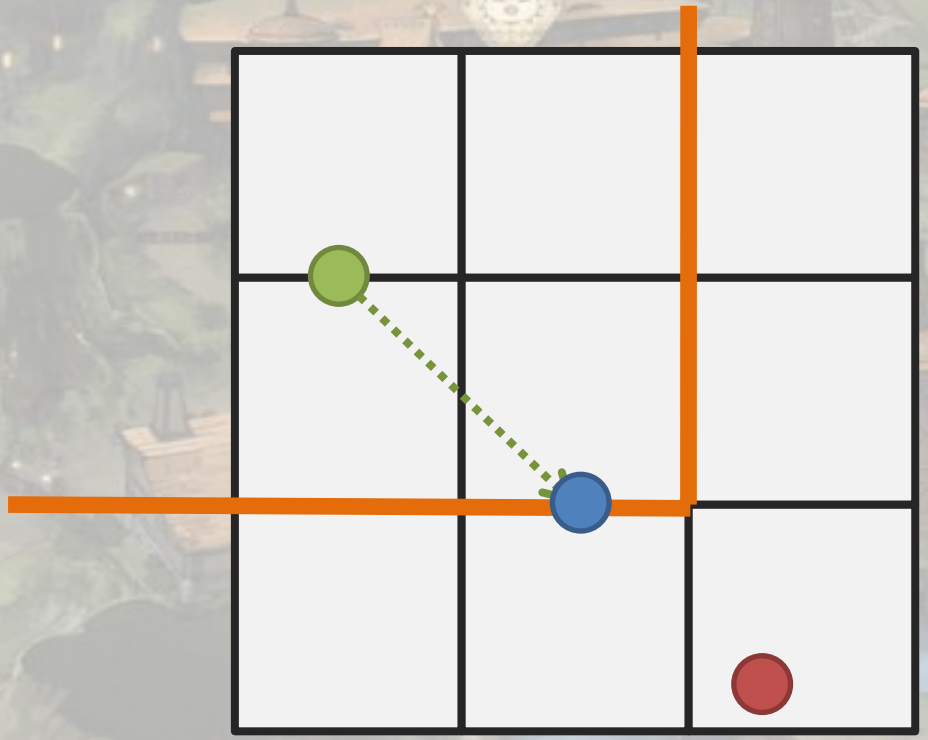
グリッドをまたいだ時に



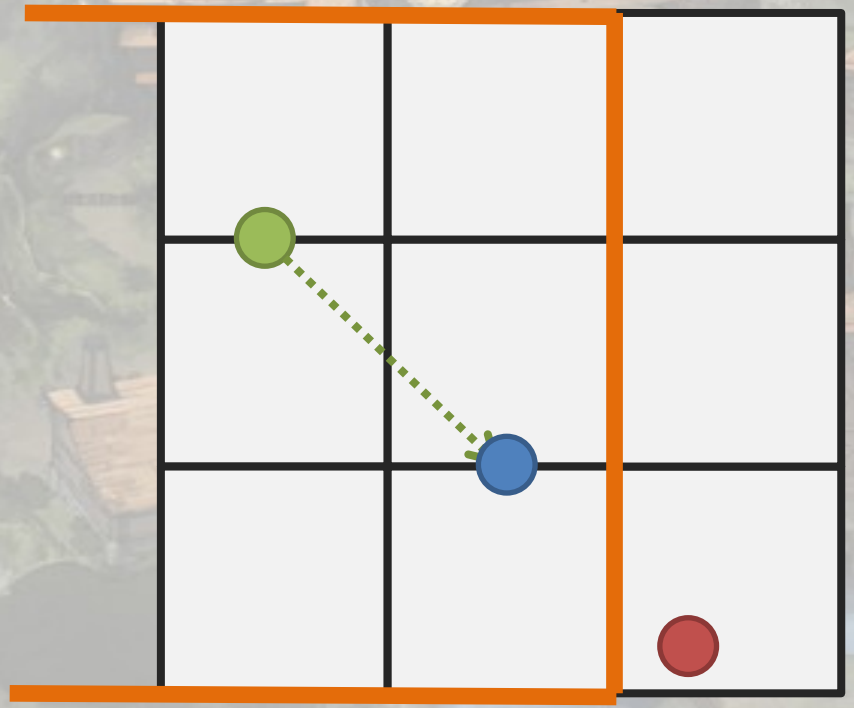
グリッドをまたいだ時に



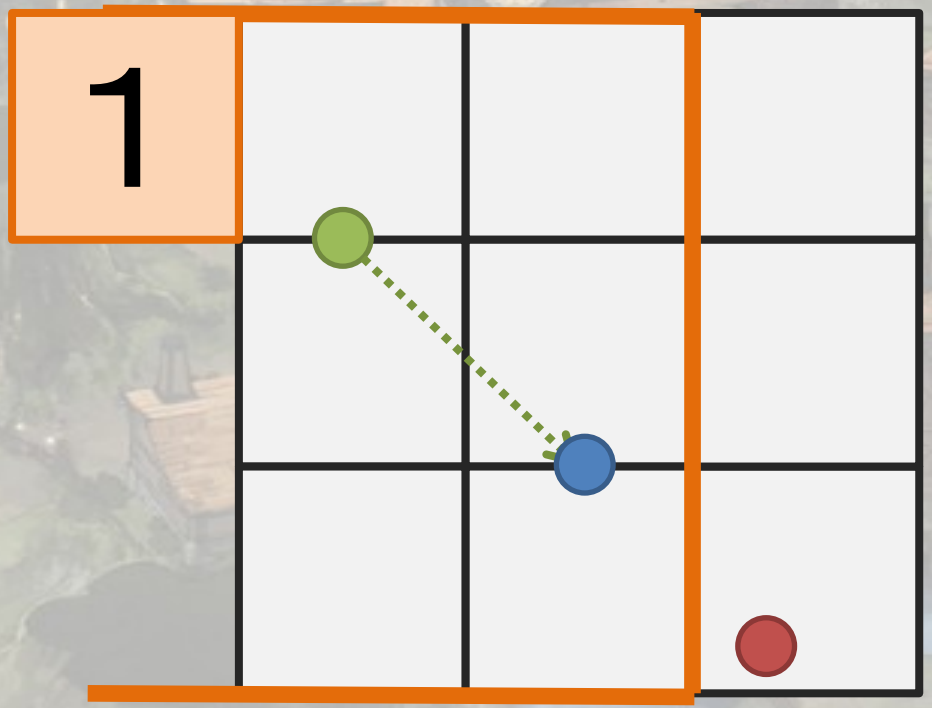
現在のテーブルを交換



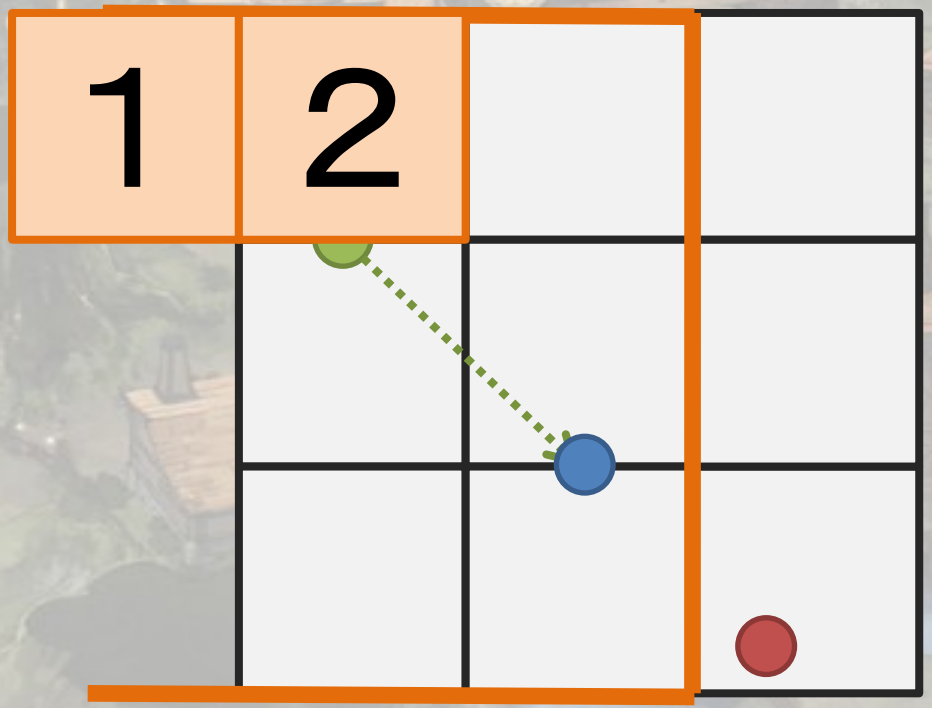
現在のテーブルを交換



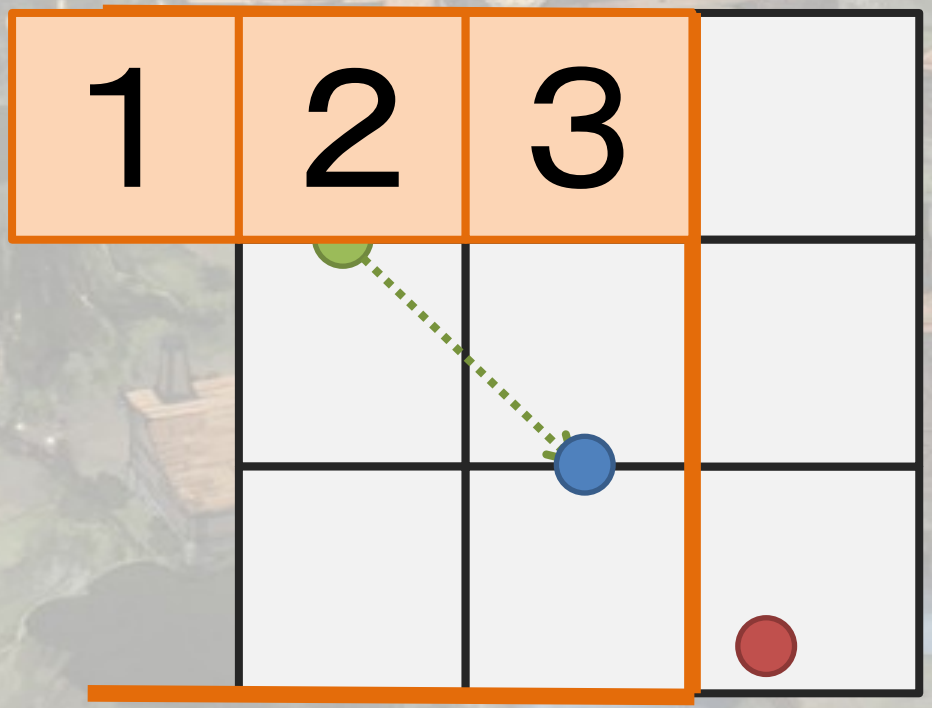
現在のテーブルを交換



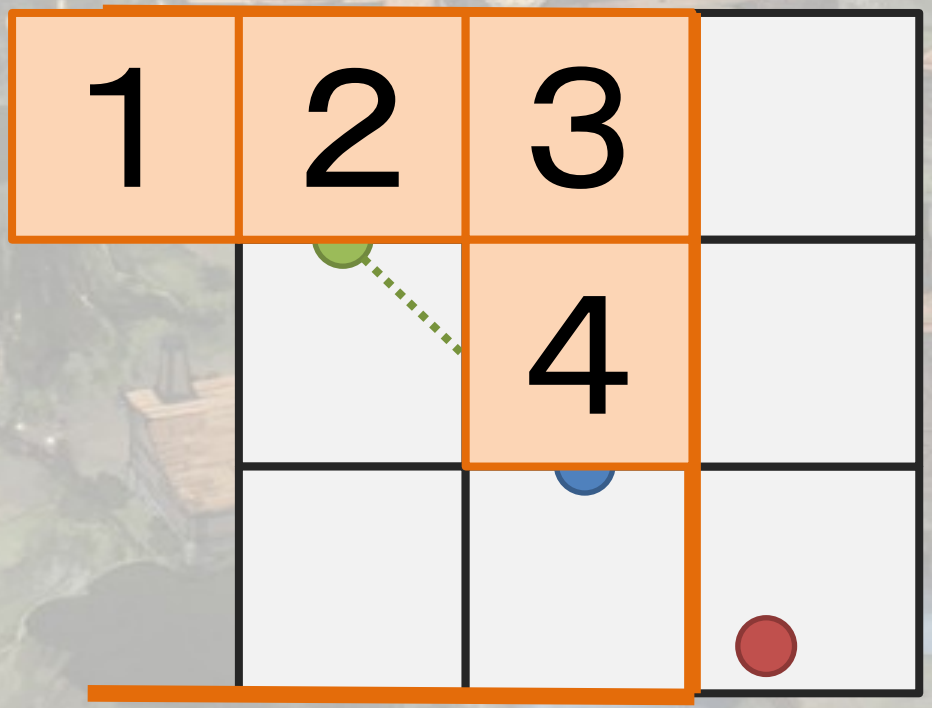
現在のテーブルを交換



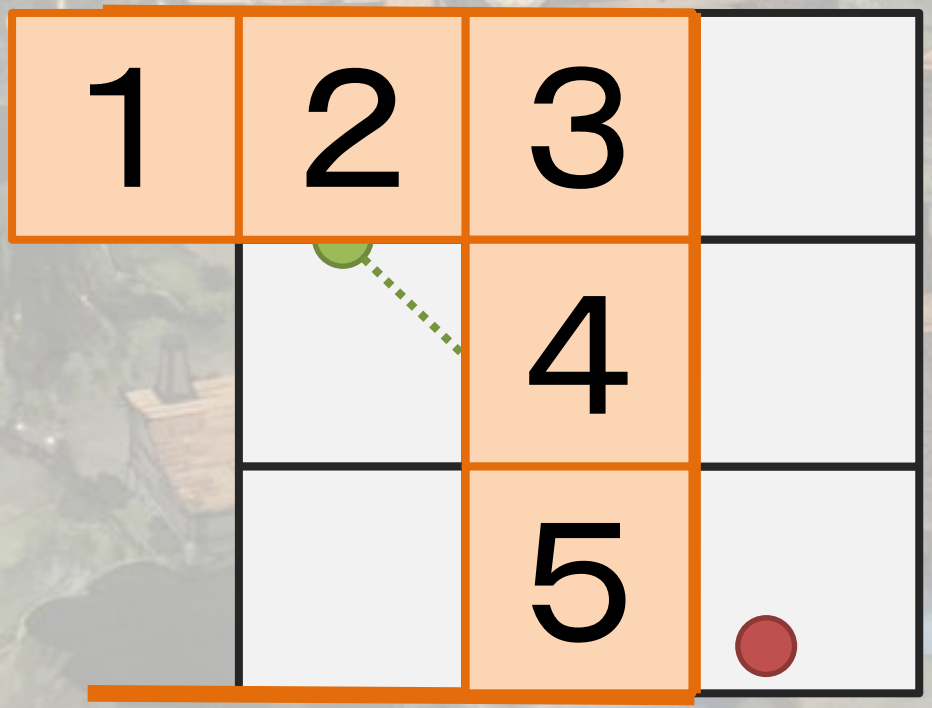
現在のテーブルを交換



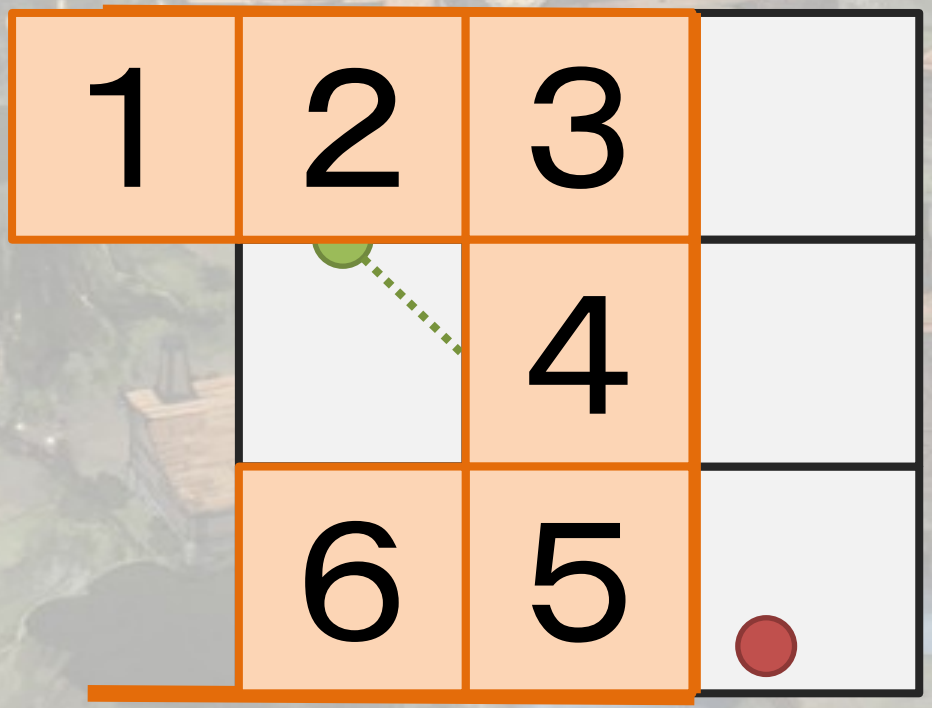
現在のテーブルを交換



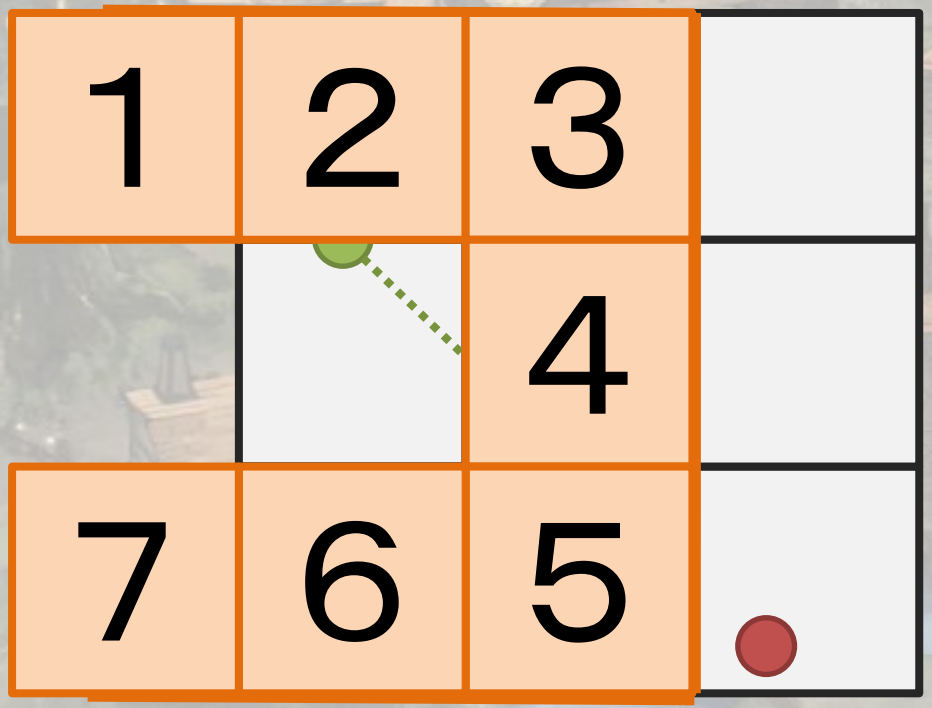
現在のテーブルを交換



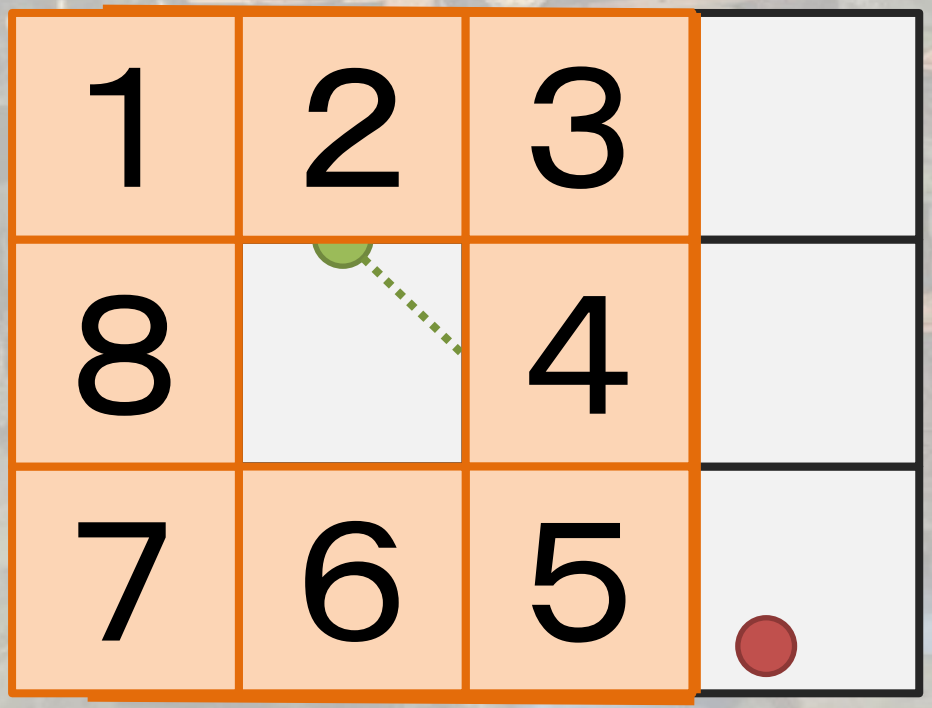
現在のテーブルを交換



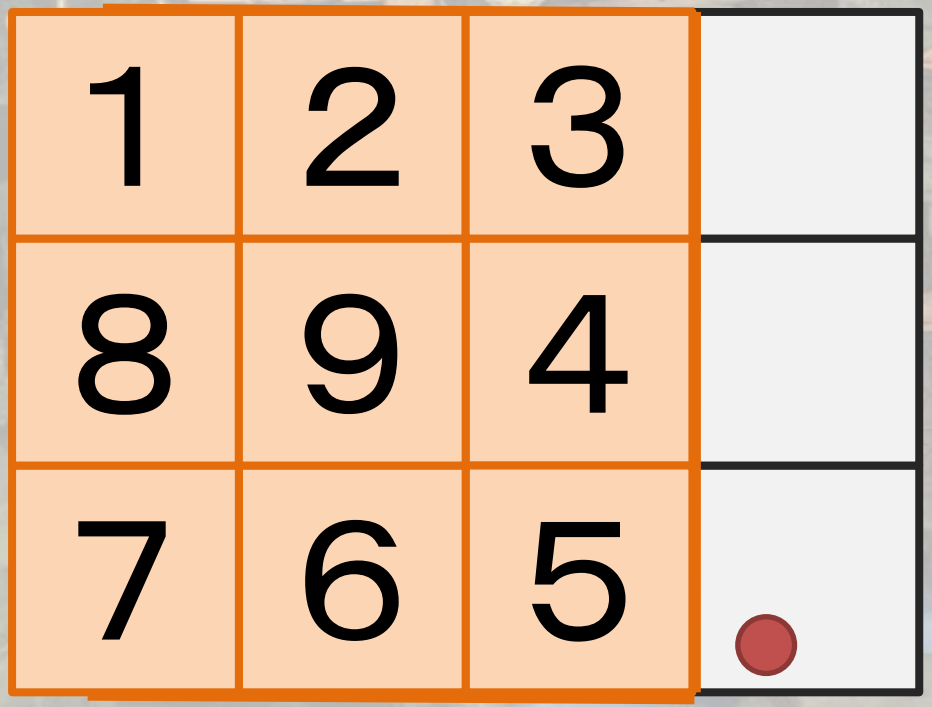
現在のテーブルを交換



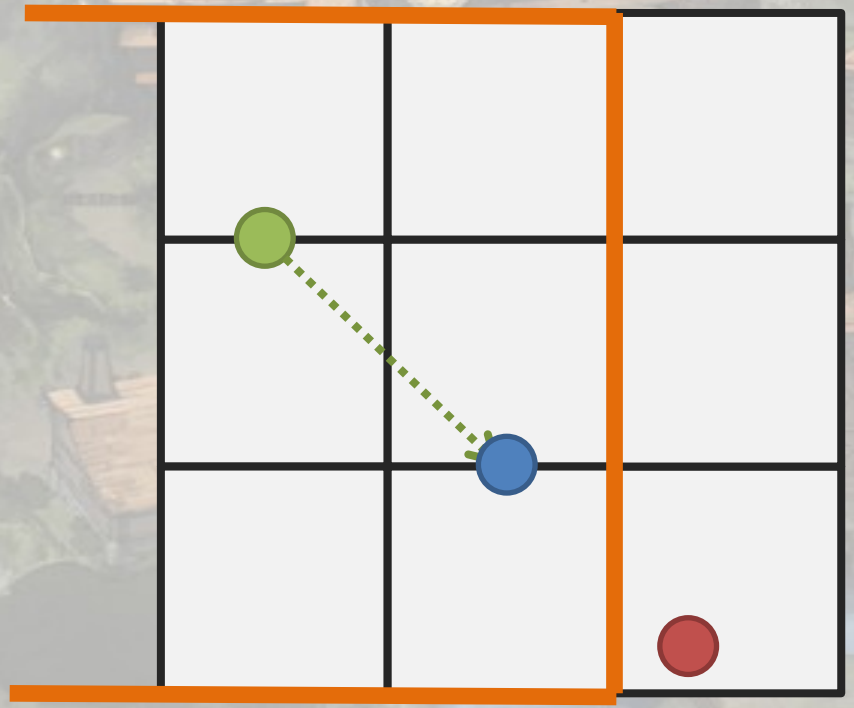
現在のテーブルを交換



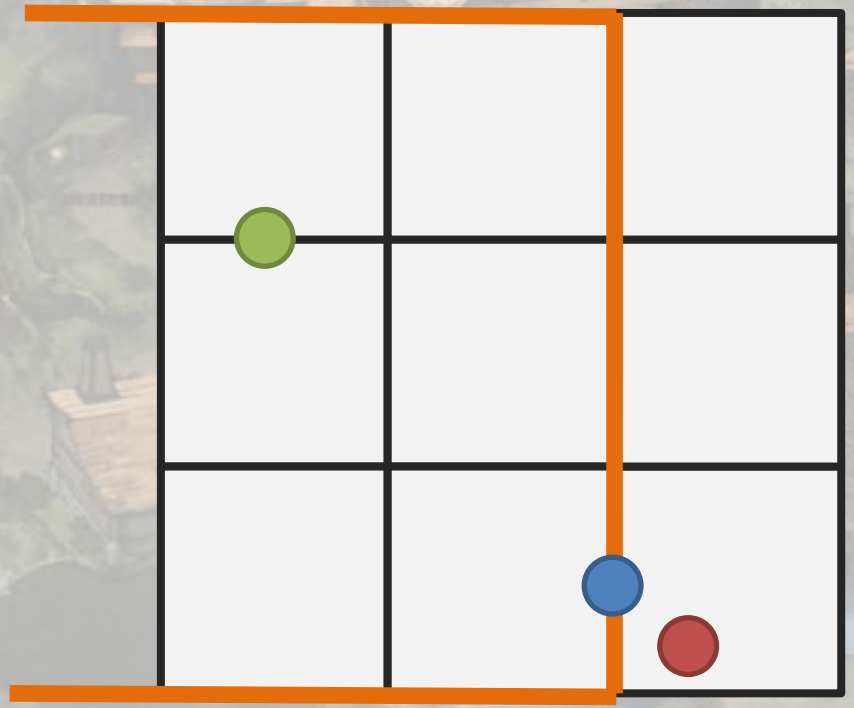
現在のテーブルを交換



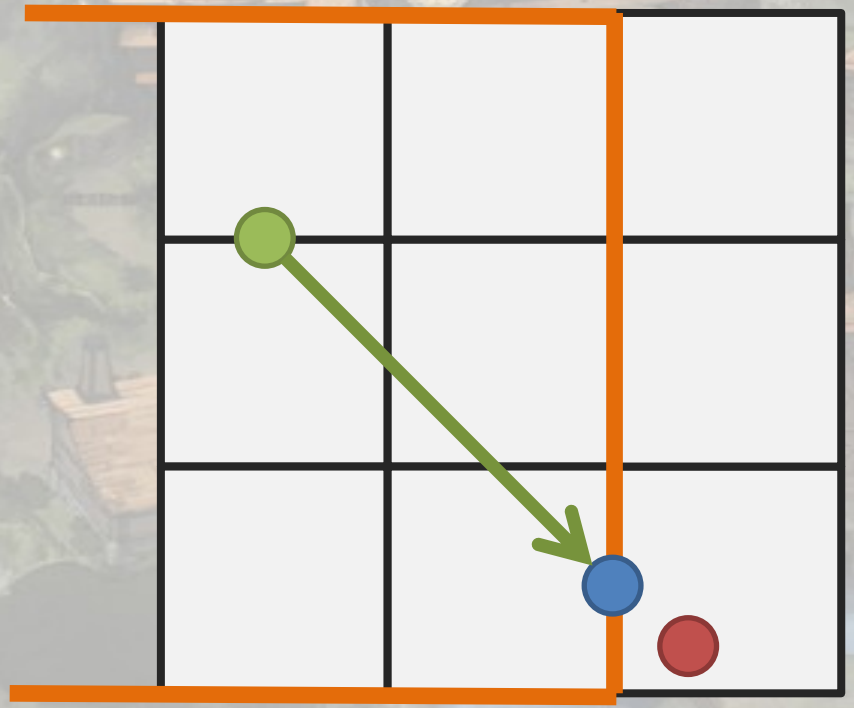
現在のテーブルを交換



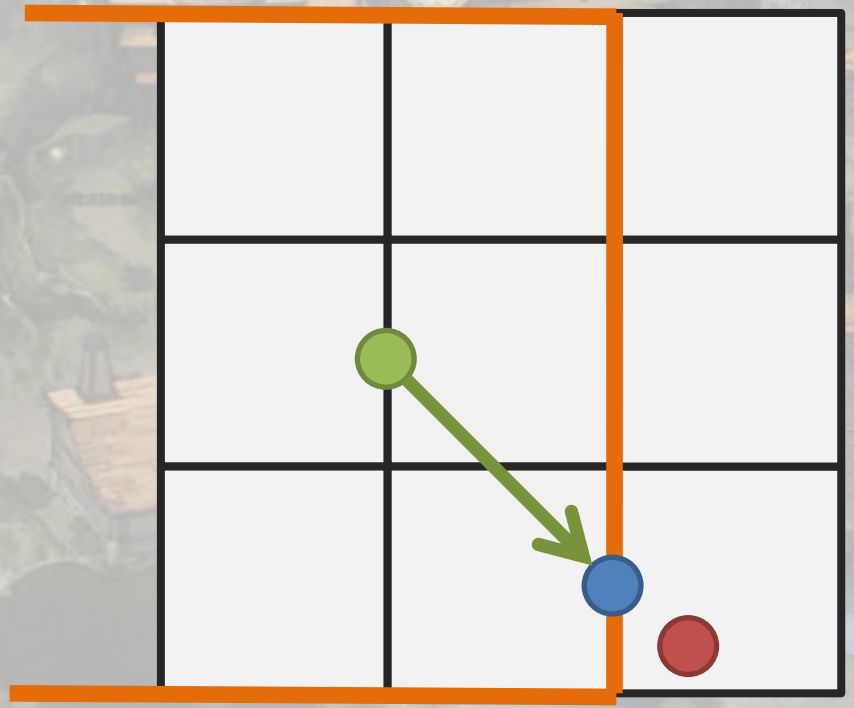
サブゴールも交換



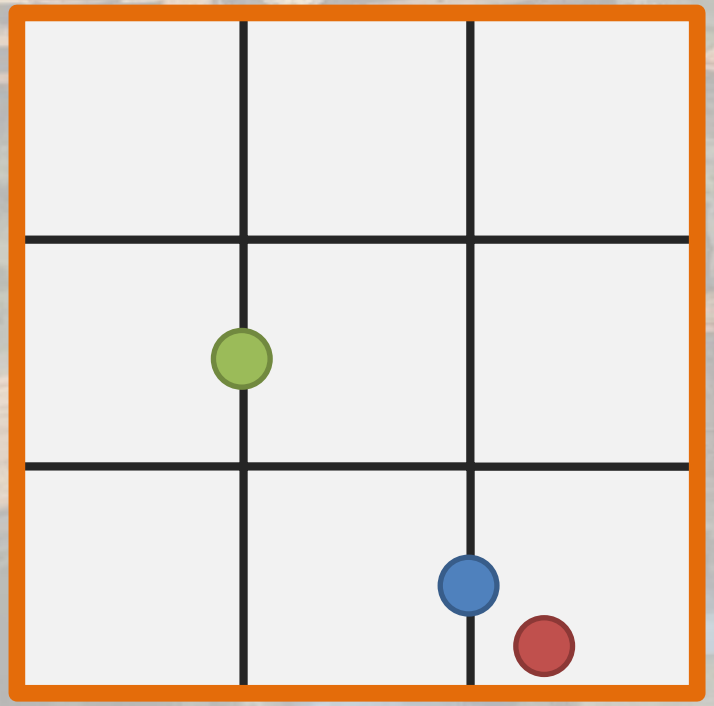
サブゴールまで移動



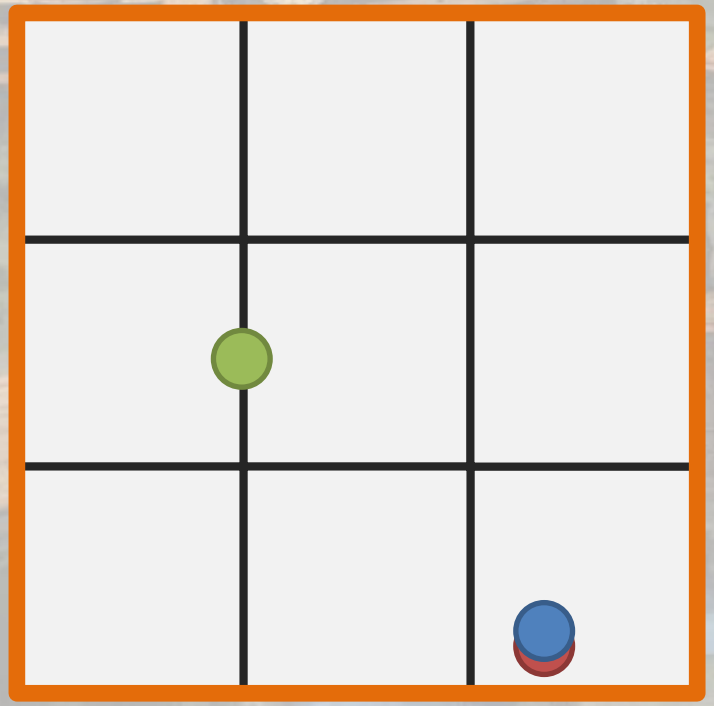
繰り返します



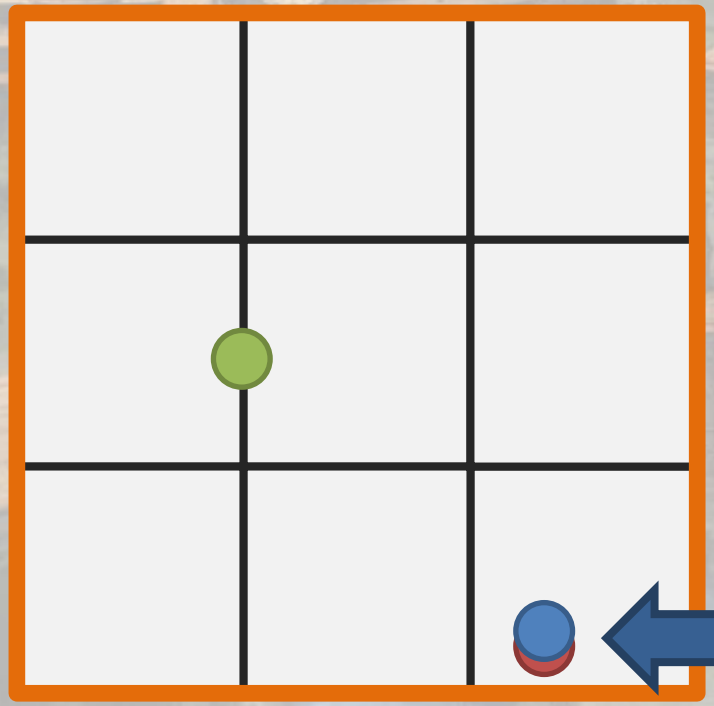
繰り返します



繰り返します

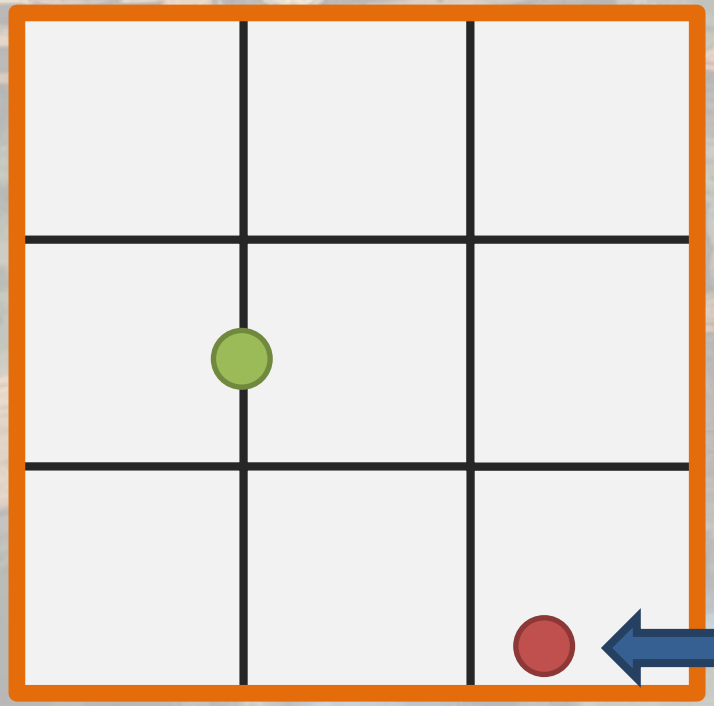


繰り返します

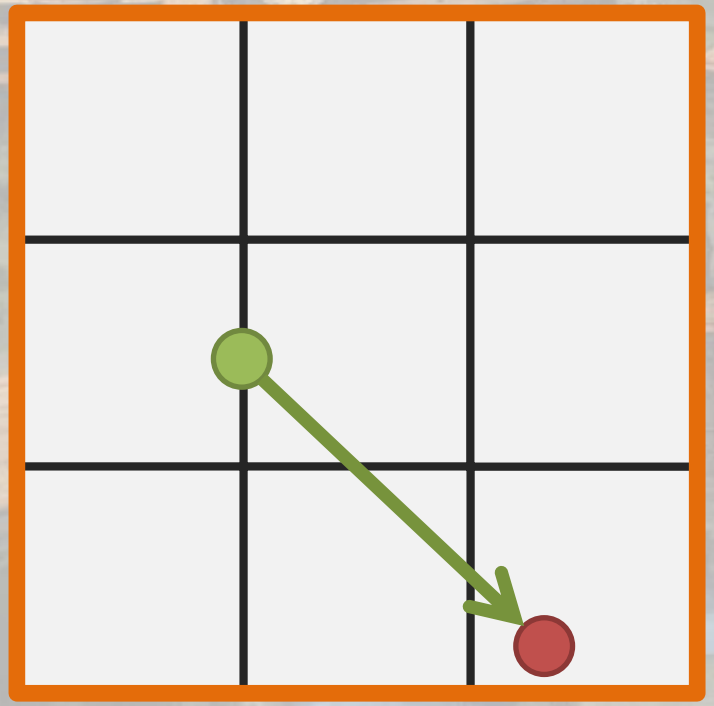


ゴールと
サブゴールが一致

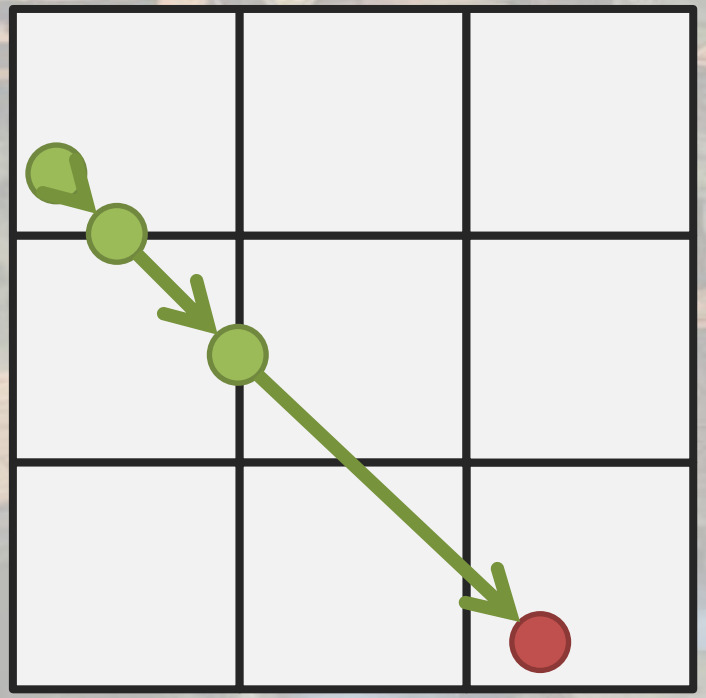
繰り返します



到着！

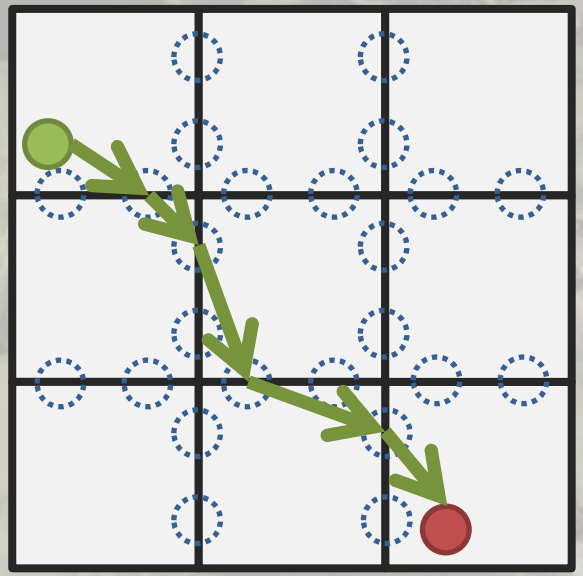


隣接経路テーブル結果



隣接経路テーブル結果

- 「繋がり」ポイントを通る



隣接経路テーブル結果

- 「繋がり」ポイントを通る
- 隣接経路テーブル

隣接経路テーブル結果

○経路テーブルで、CPU負荷が軽くなった！

○階層化で、メモリ消費量も小さくなった！

○隣接化して、経路が自然になった！

隣接経路テーブル結果

○経路テーブルで、CPU負荷が軽くなった！

○階層化で、メモリ消費量も小さくなった！

○隣接化して、経路が自然になった！

MMO-RPGで実用に耐えられる！

隣接経路テーブル結果

○経路テーブルで、CPU負荷が軽くなった！

○階層化で、メモリ消費量も小さくなった！

○隣接化して、経路が自然になった！

隣接経路テーブル結果

○経路テーブルで、CPU負荷が軽くなった！

○階層化で、メモリ消費量も小さくなった！

○隣接化して、経路が自然になった！

階層化隣接経路テーブル

隣接経路テーブル結果

○経路テーブルで、CPU負荷が軽くなった！

○階層化で、メモリ消費量も小さくなった！

○隣接化して、経路が自然になった！

Hierarchical Neighborhood lookup-Table (HiNT)

第3部
FFXIVでの経路探索

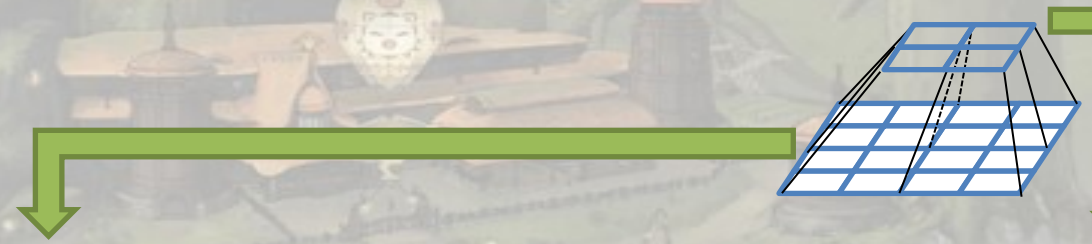
第3部 FFXIVでの経路探索

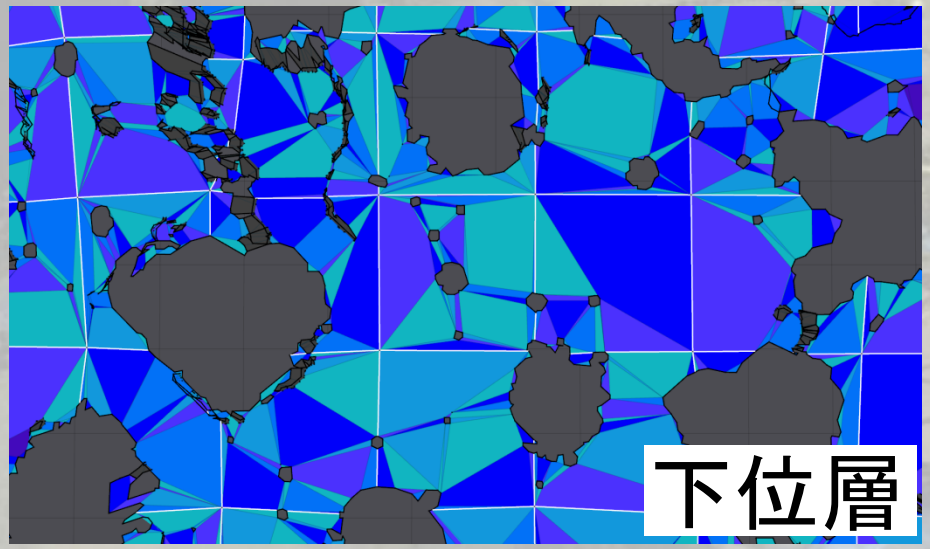
- テーブル階層化
- 経路テーブルについて
  - コンポーネントの作り方
  - 隣接経路テーブル
- テーブルをつくるアルゴリズム
- 階層化隣接経路テーブルの例

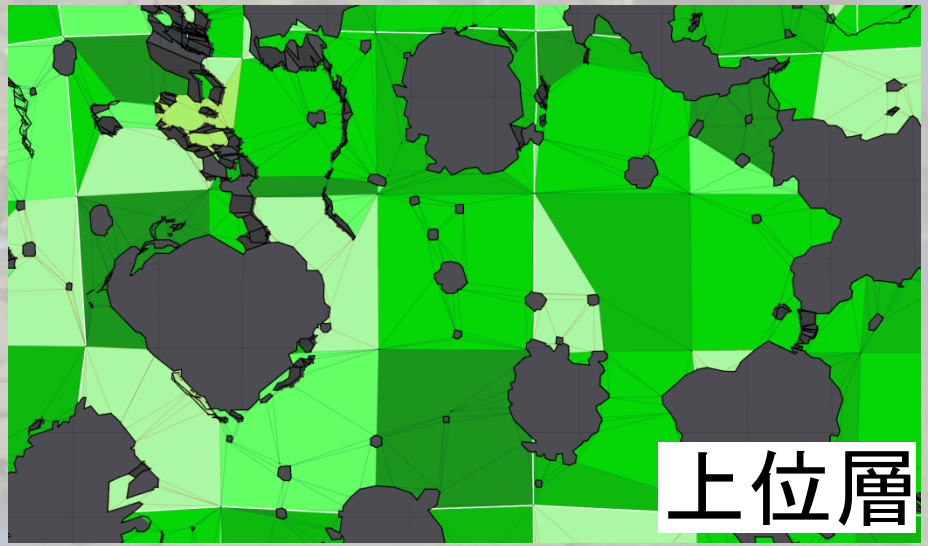
第3部 FFXIVでの経路探索

- テーブル階層化
- 経路テーブルについて
  - コンポーネントの作り方
  - 隣接経路テーブル
- テーブルをつくるアルゴリズム
- 階層化隣接経路テーブルの例

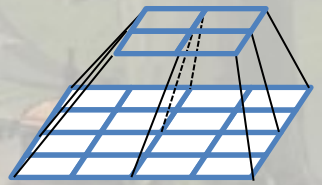
テーブル階層化





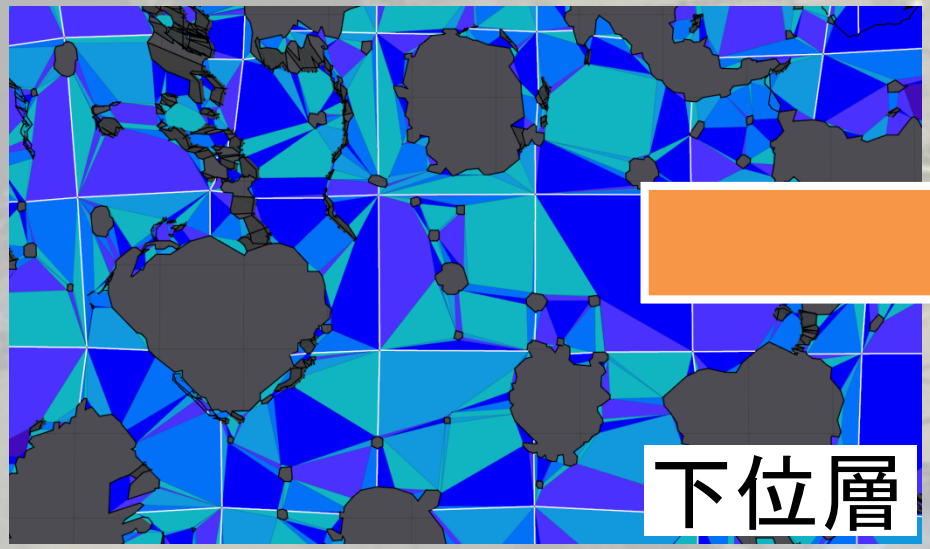


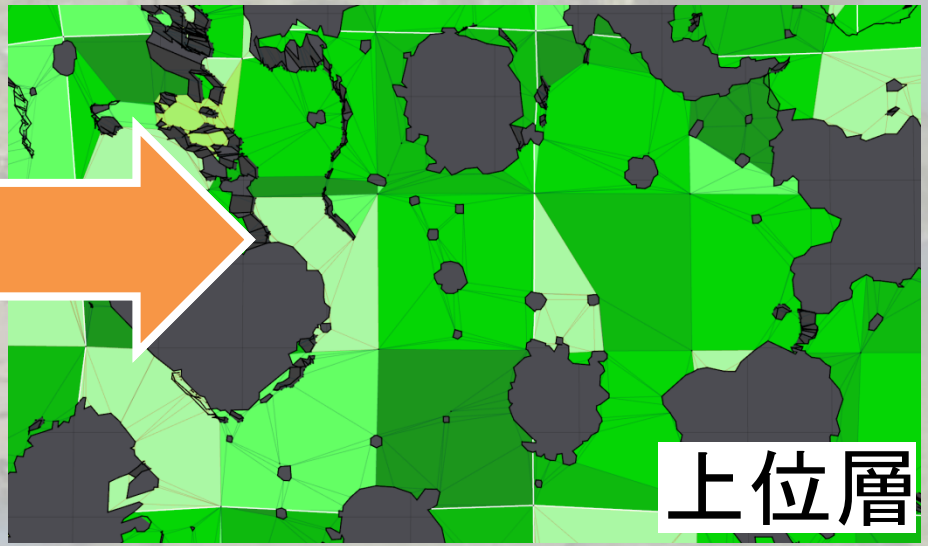
テーブル階層化



上位層

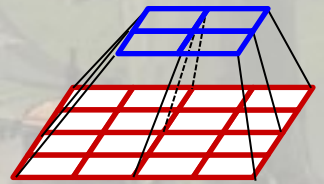
下位層





テーブル階層化

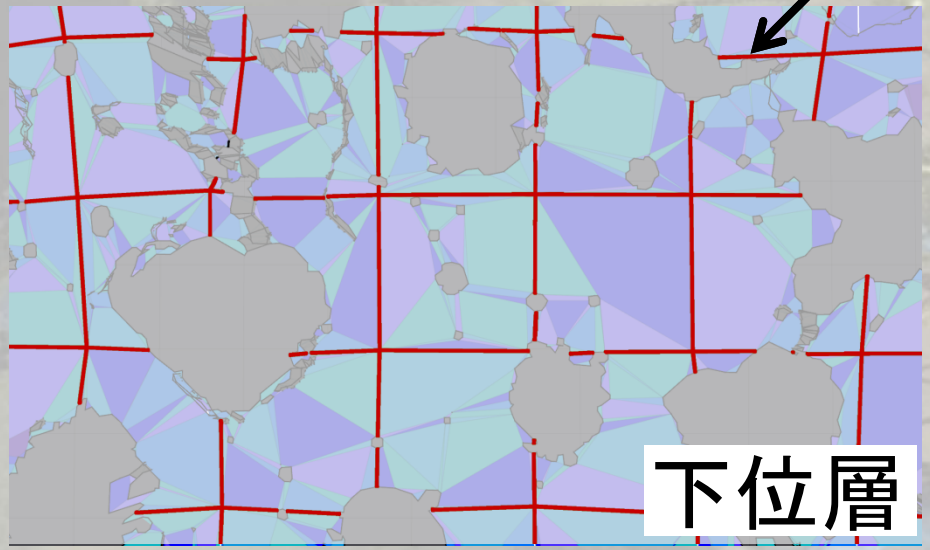
- グリッドで分ける

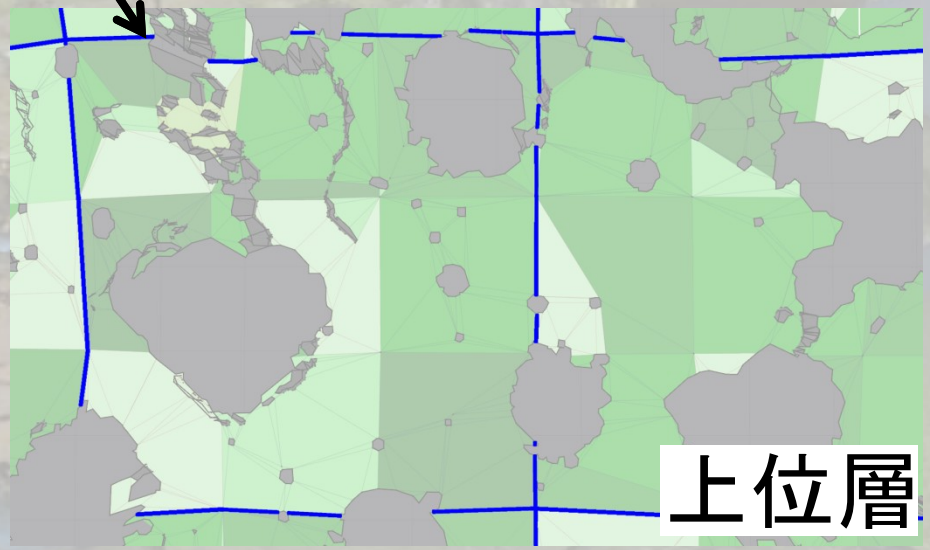


上位層

下位層

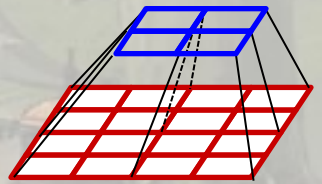
グリッド





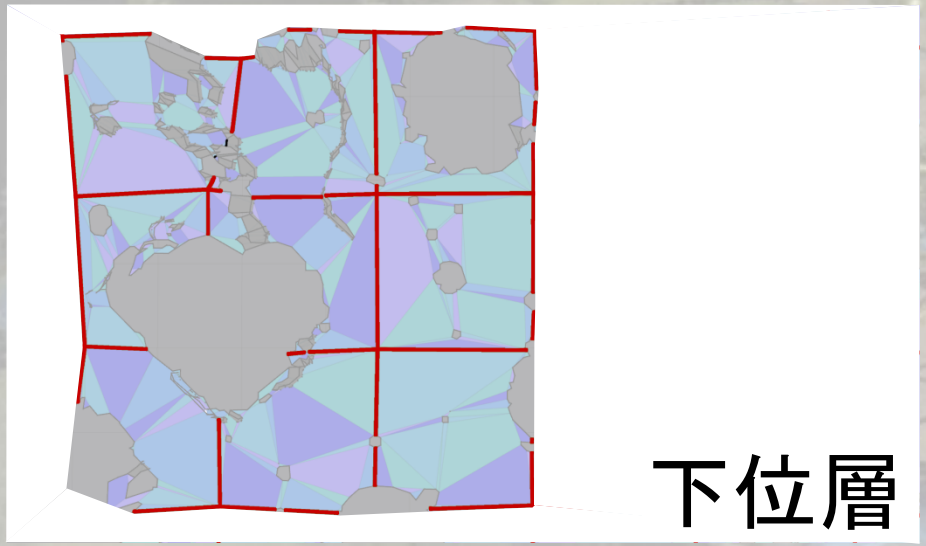
テーブル階層化

- グリッドで分ける
- グリッドを集めて上の層にする

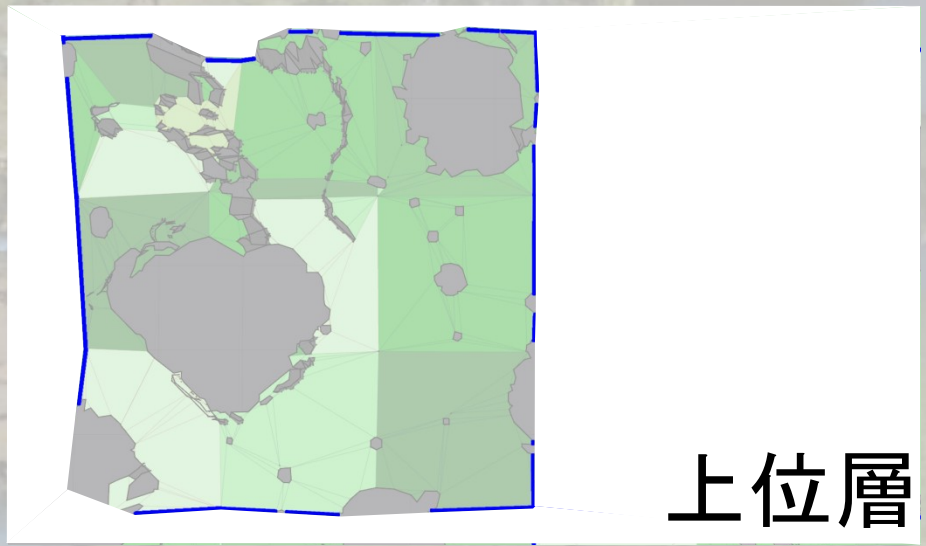


上位層

下位層

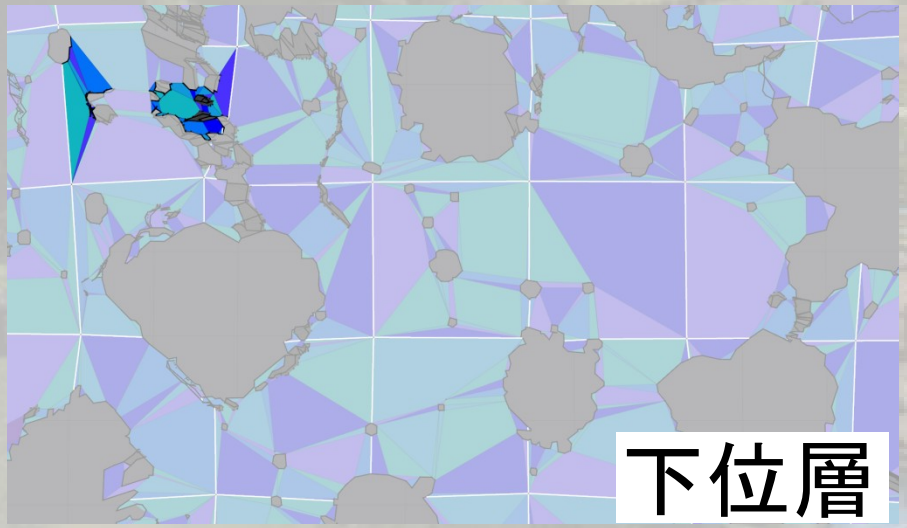


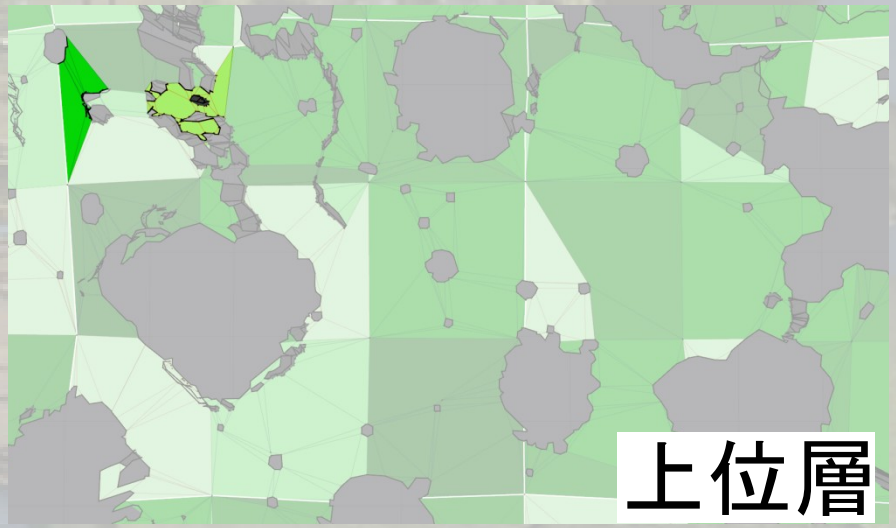
下位層



上位層

テーブル階層化

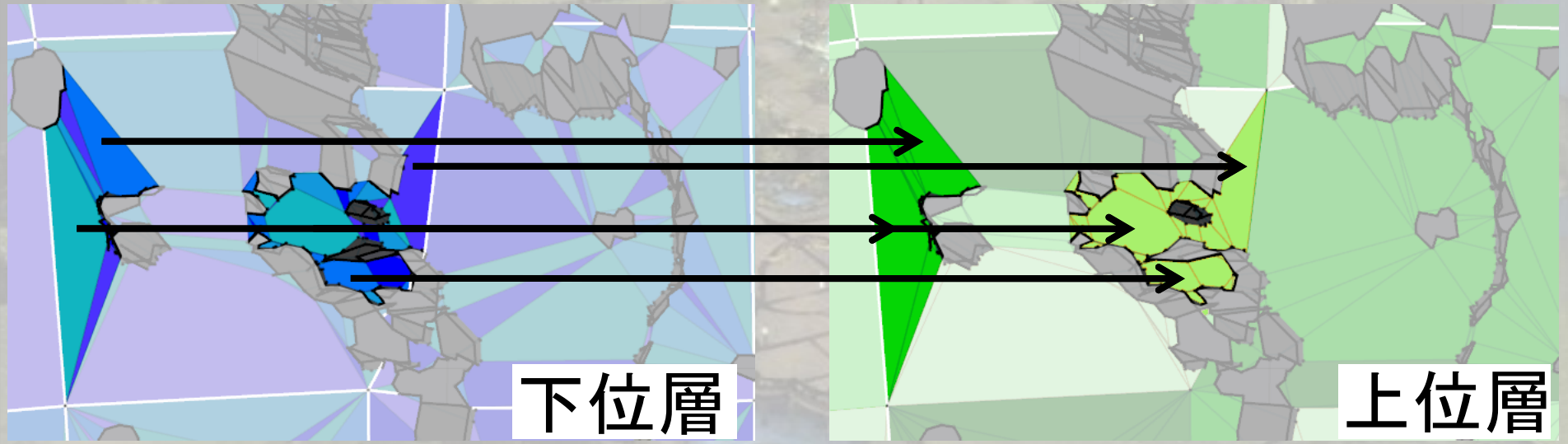




テーブル階層化

- 上位層でポリゴンをグループ化。

ポリゴン → コンポーネント

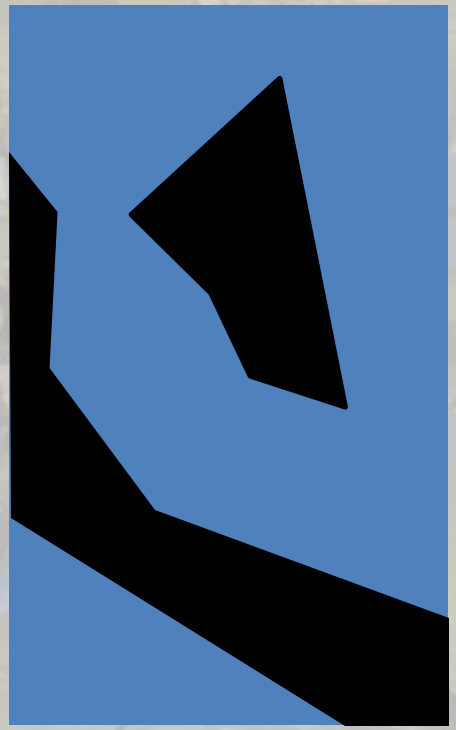


第3部 FFXIVでの経路探索

- テーブル階層化
- 経路テーブルについて
  - コンポーネントの作り方
  - 隣接経路テーブル
- テーブルをつくるアルゴリズム
- 階層化隣接経路テーブルの例

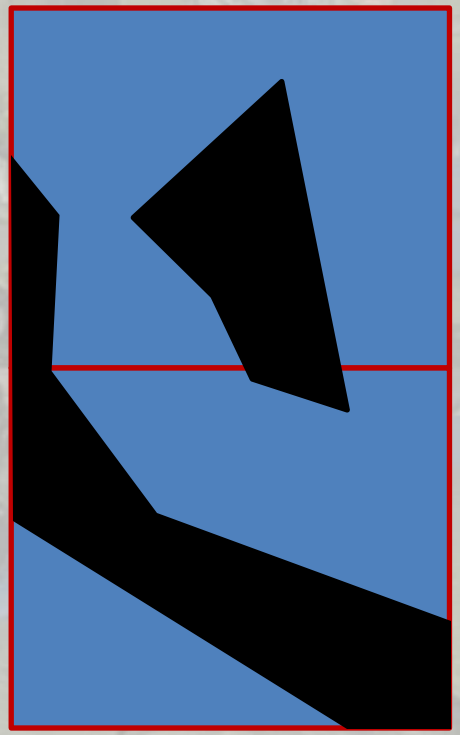
コンポーネントの作り方

- 例：
  - 歩ける所
  - 障害物



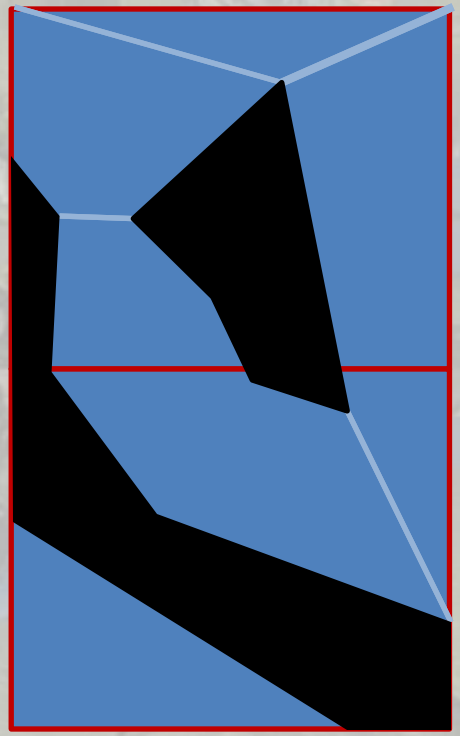
コンポーネントの作り方

- 例：
  - 歩ける所
  - 障害物
    - グリッド2個



コンポーネントの作り方

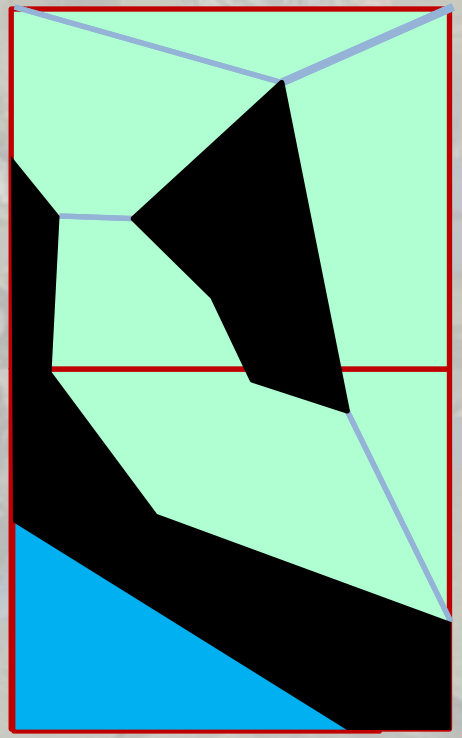
- 例：
  - 歩ける所
  - 障害物
    - グリッド2個
    - 7ポリゴン



コンポーネントの作り方

① 隣接ポリゴンをグループ化

2コンポーネント

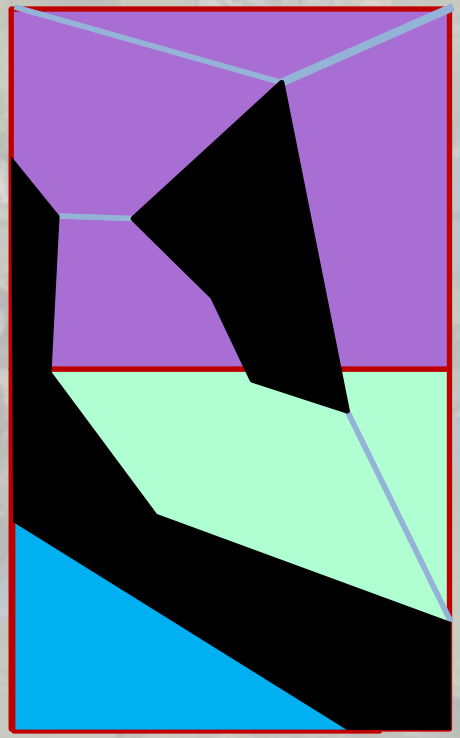


コンポーネントの作り方

① 隣接ポリゴンをグループ化

② グリッドでわける

3コンポーネント

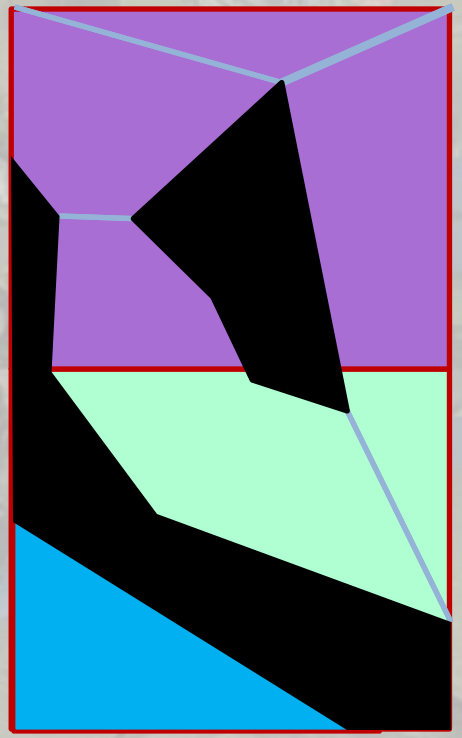


コンポーネントの作り方

① 隣接ポリゴンをグループ化

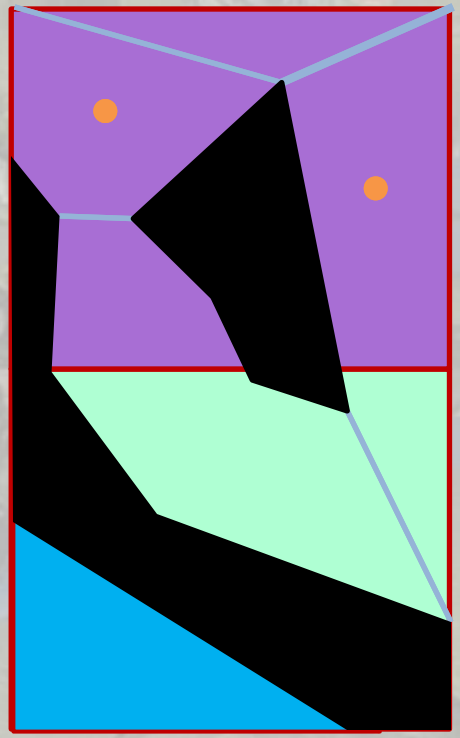
② グリッドでわける

③ 経路をチェックして分割する



コンポーネントの作り方

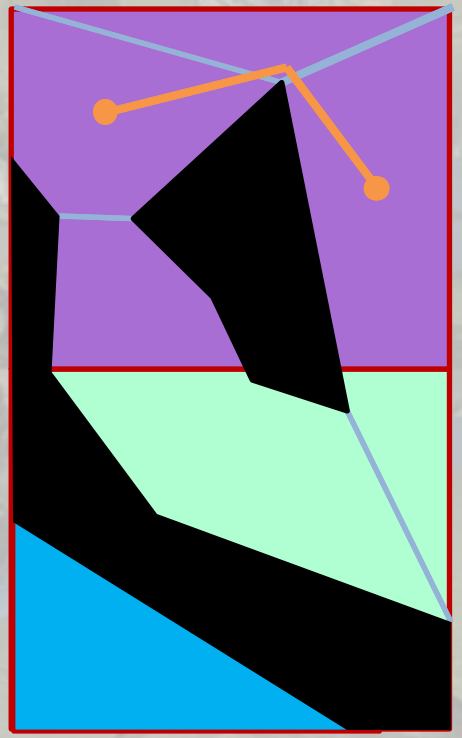
① 隣接ポリゴンをグループ化
② グリッドでわける
③ 経路をチェックして分割する
  – 同じコンポーネントから2点を選ぶ



SQUARE ENIX

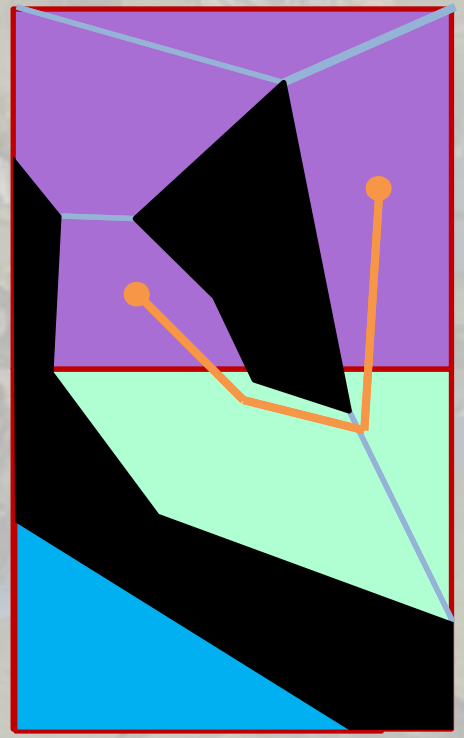
コンポーネントの作り方

① 隣接ポリゴンをグループ化
② グリッドでわける
③ 経路をチェックして分割する
  – 同じコンポーネントから2点を選ぶ
  – 経路探索する
  – 別なコンポーネントを通れば分割



コンポーネントの作り方

① 隣接ポリゴンをグループ化
② グリッドでわける
③ 経路をチェックして分割する
  – 同じコンポーネントから2点を選ぶ
  – 経路探索する
  – 別なコンポーネントを通れば分割



コンポーネントの作り方

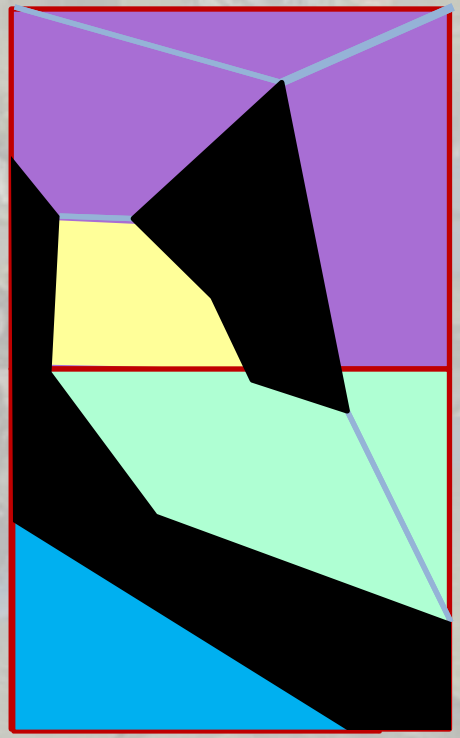
① 隣接ポリゴンをグループ化
② グリッドでわける
③ 経路をチェックして分割する
  – 同じコンポーネントから2点を選ぶ
  – 経路探索する
  – 別なコンポーネントを通れば分割

SQUARE ENIX

コンポーネントの作り方

① 隣接ポリゴンをグループ化
② グリッドでわける
③ 経路をチェックして分割する
  – 同じコンポーネントから2点を選ぶ
  – 経路探索する
  – 別なコンポーネントを通れば分割

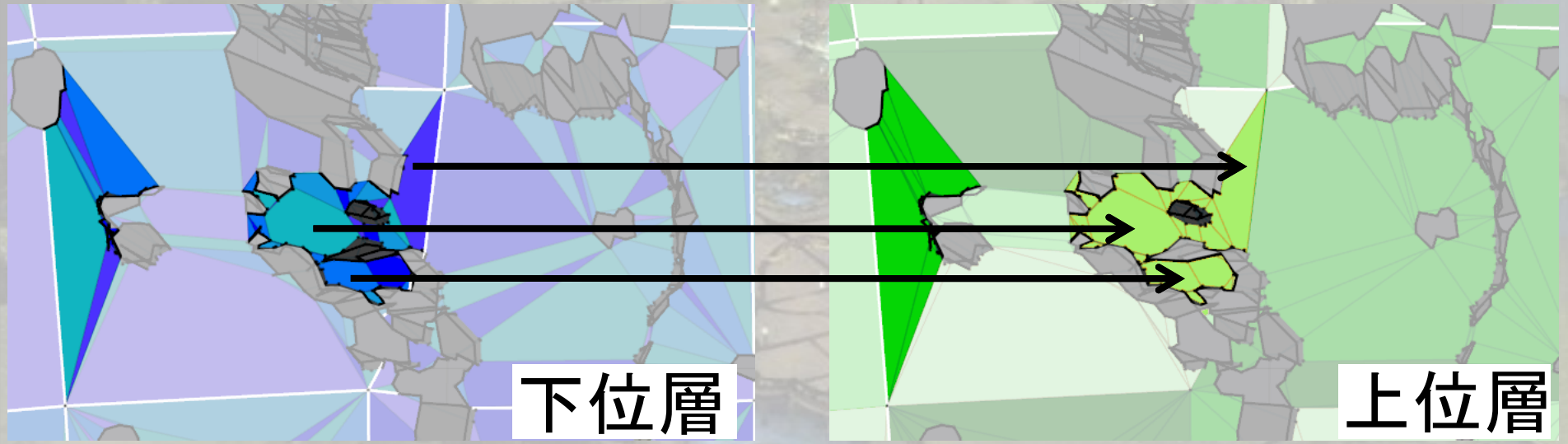
4コンポーネント



SQUARE ENIX

コンポーネントの作り方

ポリゴン → コンポーネント

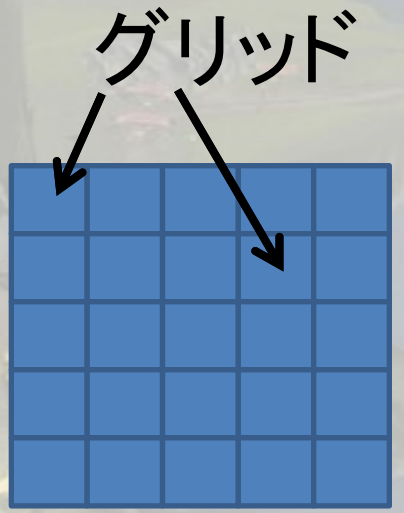


第3部 FFXIVでの経路探索

- テーブル階層化
- 経路テーブルについて
  - コンポーネントの作り方
  - 隣接経路テーブル
- テーブルをつくるアルゴリズム
- 階層化隣接経路テーブルの例

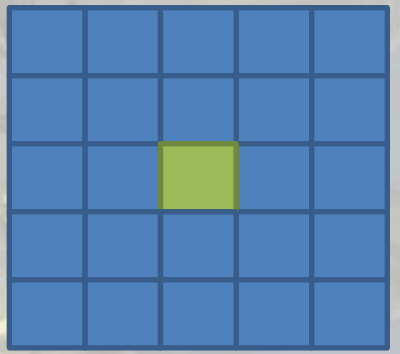
SQUARE ENIX

隣接経路テーブル



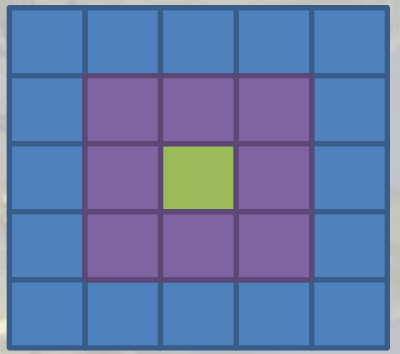
隣接経路テーブル

- 現在のグリッドから



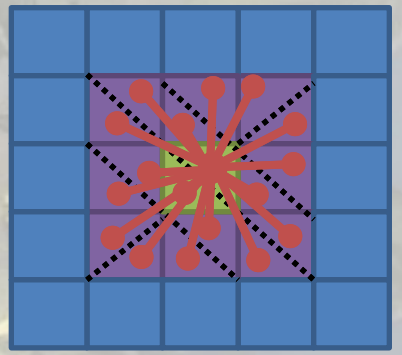
隣接経路テーブル

- 現在のグリッドから
- 現在 と隣の8グリッド へは、



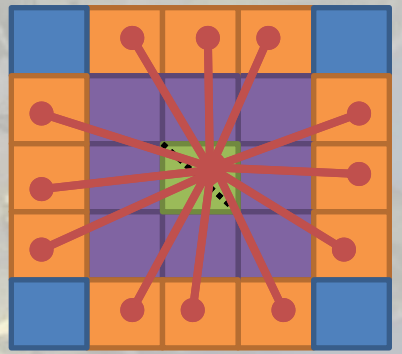
隣接経路テーブル

- 現在のグリッドから
- 現在 と隣の8グリッド へは、
  どの**ポリゴン**にも移動できる



隣接経路テーブル

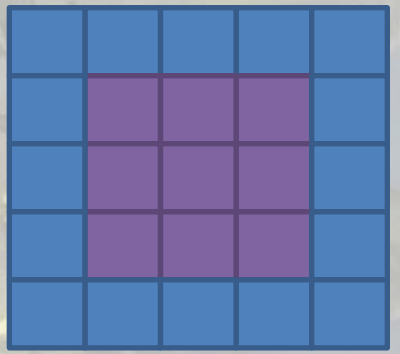
- さらに隣の１２グリッド ■ へは、
  どの**コンポーネント**へ移動できる。



隣接経路テーブル

テーブルの目標：

- グリッド：ポリゴンの集まり

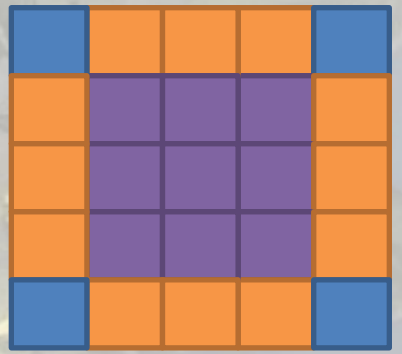


SQUARE ENIX

隣接経路テーブル

テーブルの目標：

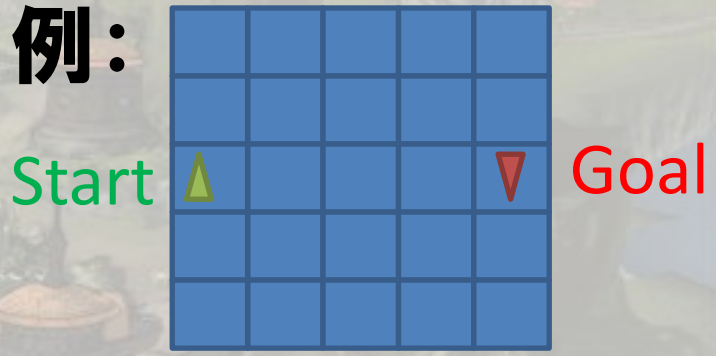
- グリッド：ポリゴンの集まり
- グリッド：コンポーネットの集まり



隣接経路テーブル

- スタートからゴールまで経路探索

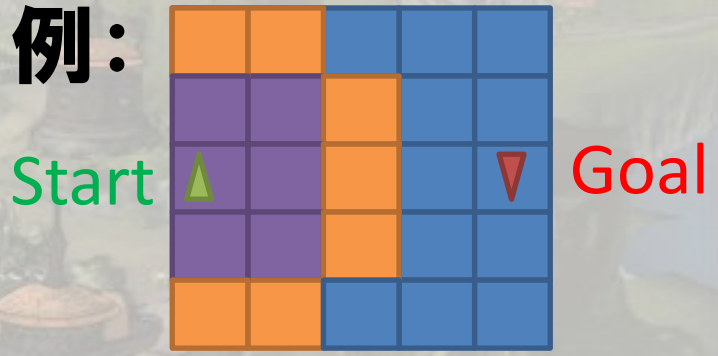
例：



隣接経路テーブル

- 現在のテーブルを使う

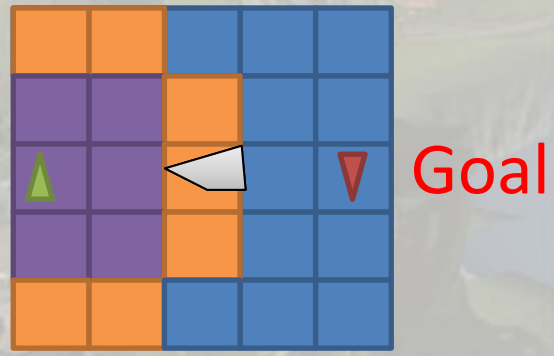
例：



隣接経路テーブル

- 現在のテーブルを使う
- 一番遠いサブゴールを選ぶ

例：



隣接経路テーブル

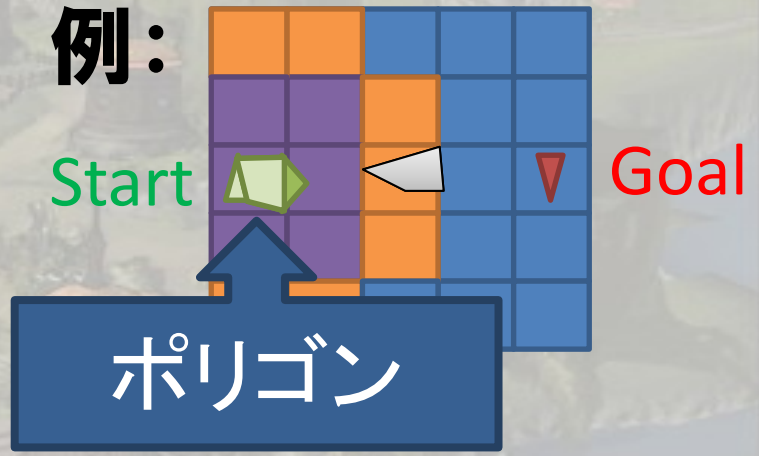
- 現在のテーブルを使う
- 一番遠いサブゴールを選ぶ

例：

隣接経路テーブル

- 現在のテーブルを使う
- 一番遠いサブゴールを選ぶ
- 次のポリゴンに移動

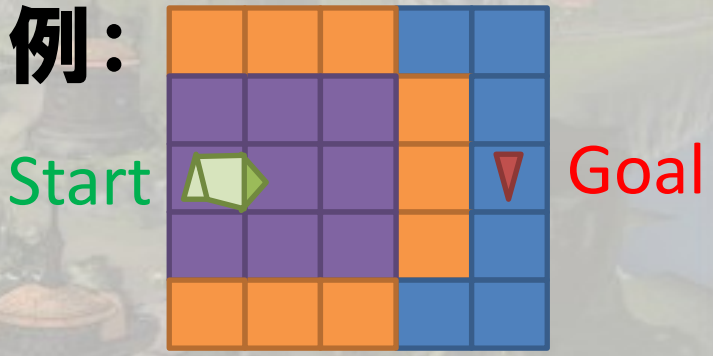
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える

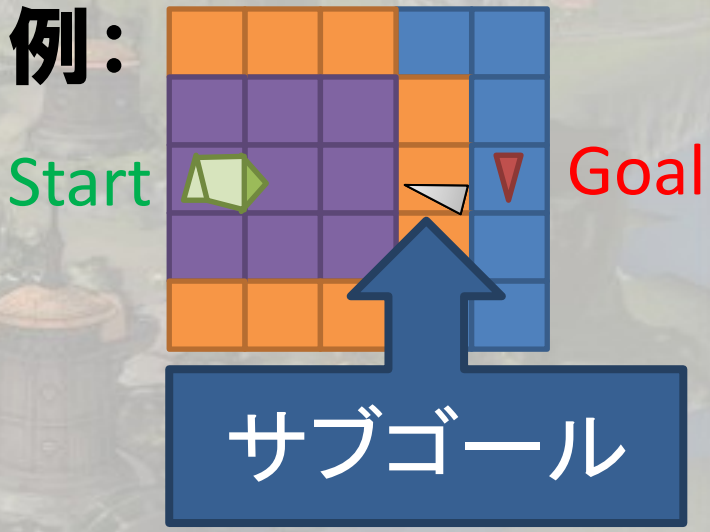
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える
- サブゴールも変える

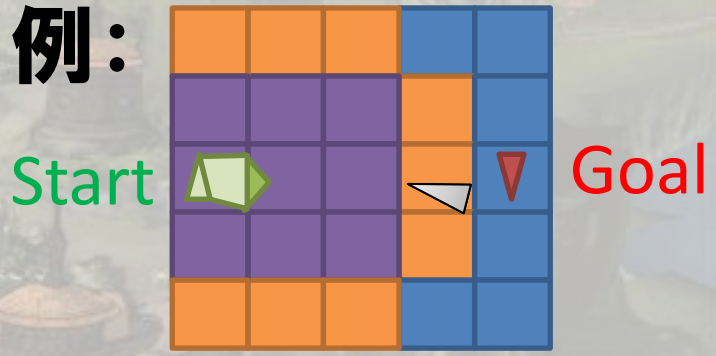
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える
- サブゴールも変える
- 次のポリゴンに移動

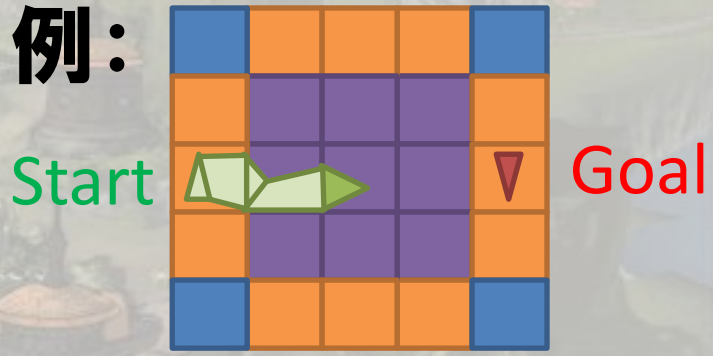
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える

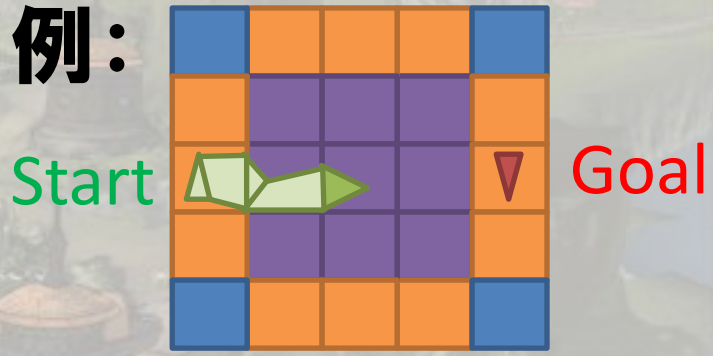
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える
- ゴールがテーブルに入った

例：

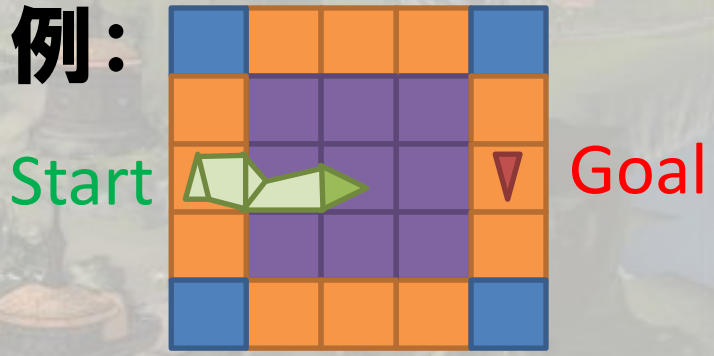


SQUARE ENIX

隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える
- ゴールがテーブルに入った
- 次のポリゴンに移動

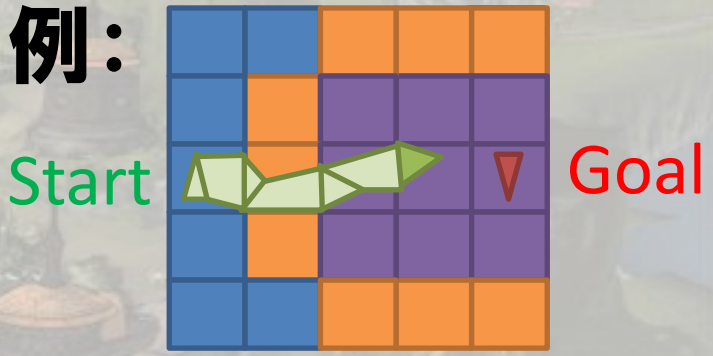
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える

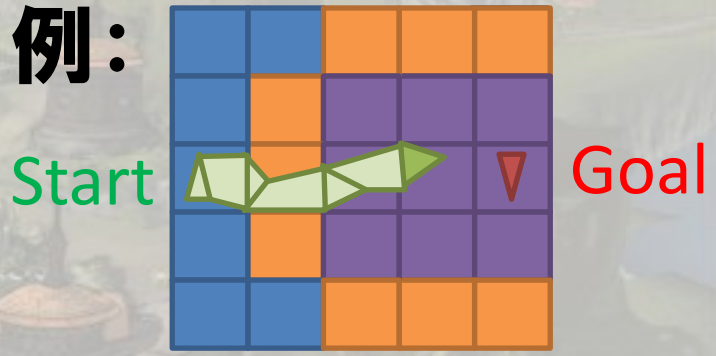
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える
- ゴールがテーブルに入った

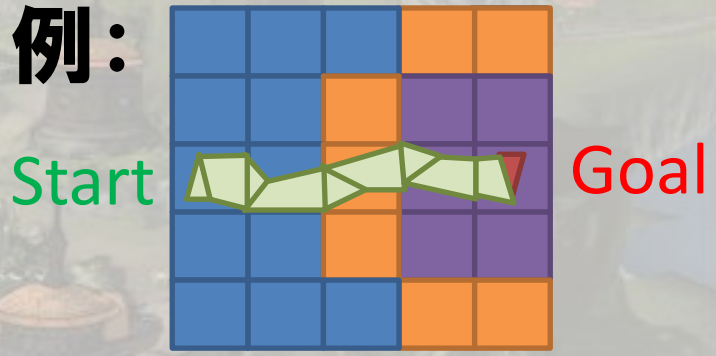
例：



隣接経路テーブル

- 次のグリッドに入った時に、テーブルを切り替える
- ゴールがテーブルに入った
- あとはまっすぐたどるだけ

例：



第3部 FFXIVでの経路探索

- テーブル階層化
- 経路テーブルについて
  - コンポーネントの作り方
  - 隣接経路テーブル
- テーブルをつくるアルゴリズム
- 階層化隣接経路テーブルの例

テーブル作成のアルゴリズム

1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る

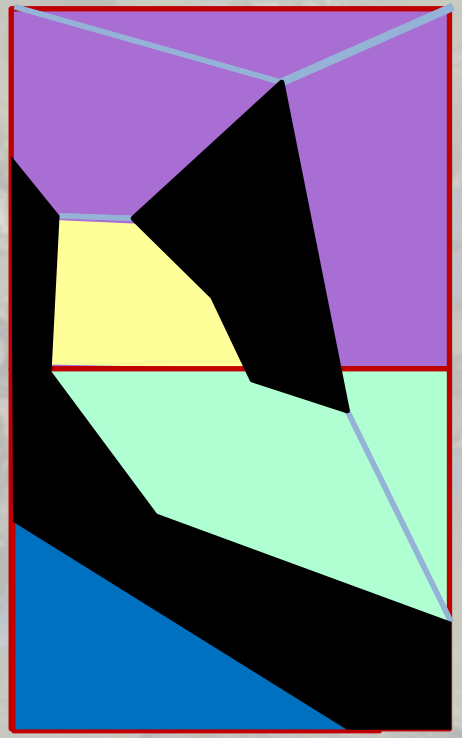
テーブル作成のアルゴリズム

1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る

	1	2	3	4
1	-	3	3	3
2	5	-	5	5
3	1	4	-	4
4	1	2	1	-
5	4	2	4	4

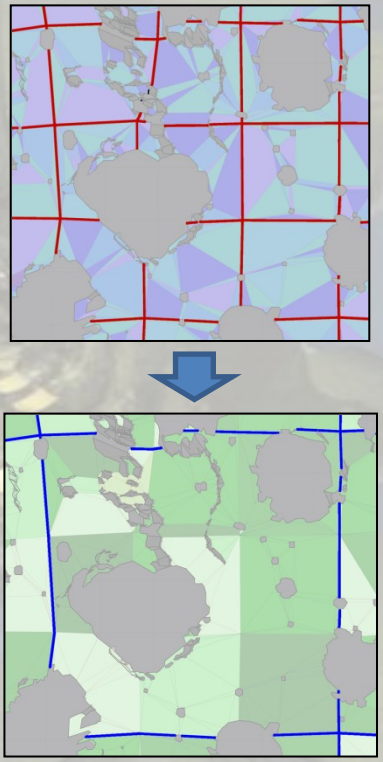
テーブル作成のアルゴリズム

1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る



テーブル作成のアルゴリズム

1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る



Final Fantasy XIV © 2010-2012 SQUARE ENIX CO., LTD. All rights reserved.

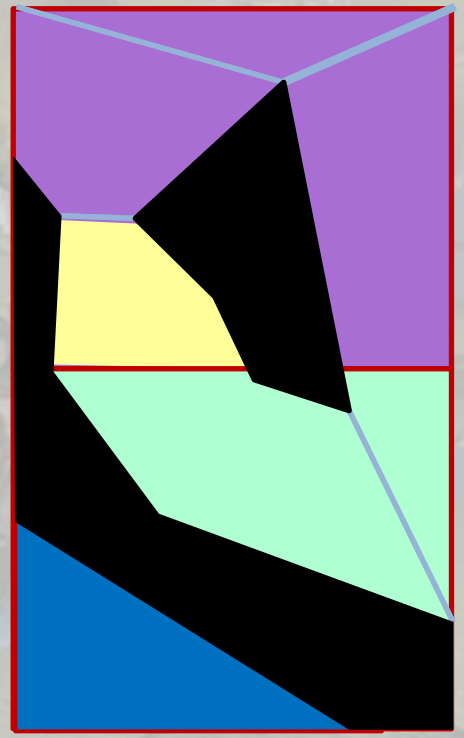
テーブル作成のアルゴリズム

1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る

	1	2	3	4
1	-	3	3	3
2	5	-	5	5
3	1	4	-	4
4	1	2	1	-
5	4	2	4	4

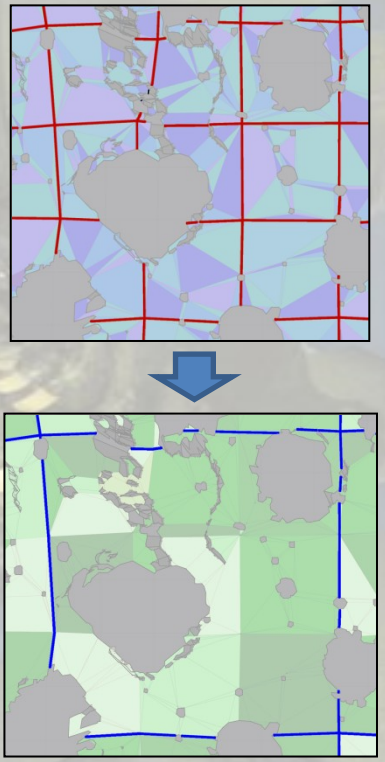
テーブル作成のアルゴリズム

1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る



テーブル作成のアルゴリズム

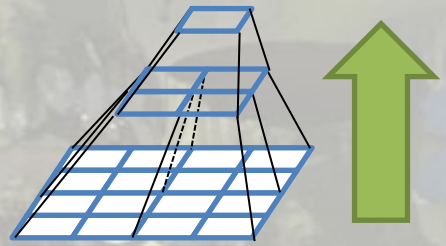
1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る



Final Fantasy XIV © 2010-2012 SQUARE ENIX CO., LTD. All rights reserved.

テーブル作成のアルゴリズム

1. メッシュ初期化
2. グリッドごとのテーブル作成
3. コンポーネント作成
4. コンポーネントから階層を作る

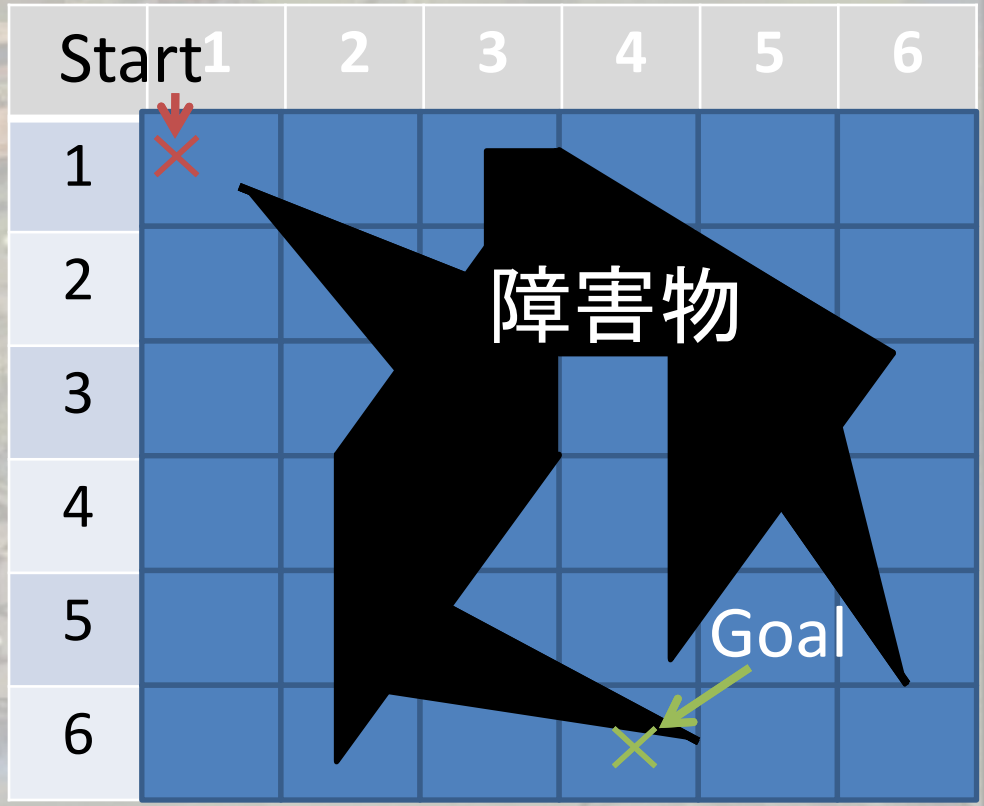


第3部 FFXIVでの経路探索

- テーブル階層化
- 経路テーブルについて
  - コンポーネントの作り方
  - 隣接経路テーブル
- テーブルをつくるアルゴリズム
- 階層化隣接経路テーブルの例

階層化隣接経路テーブルの例

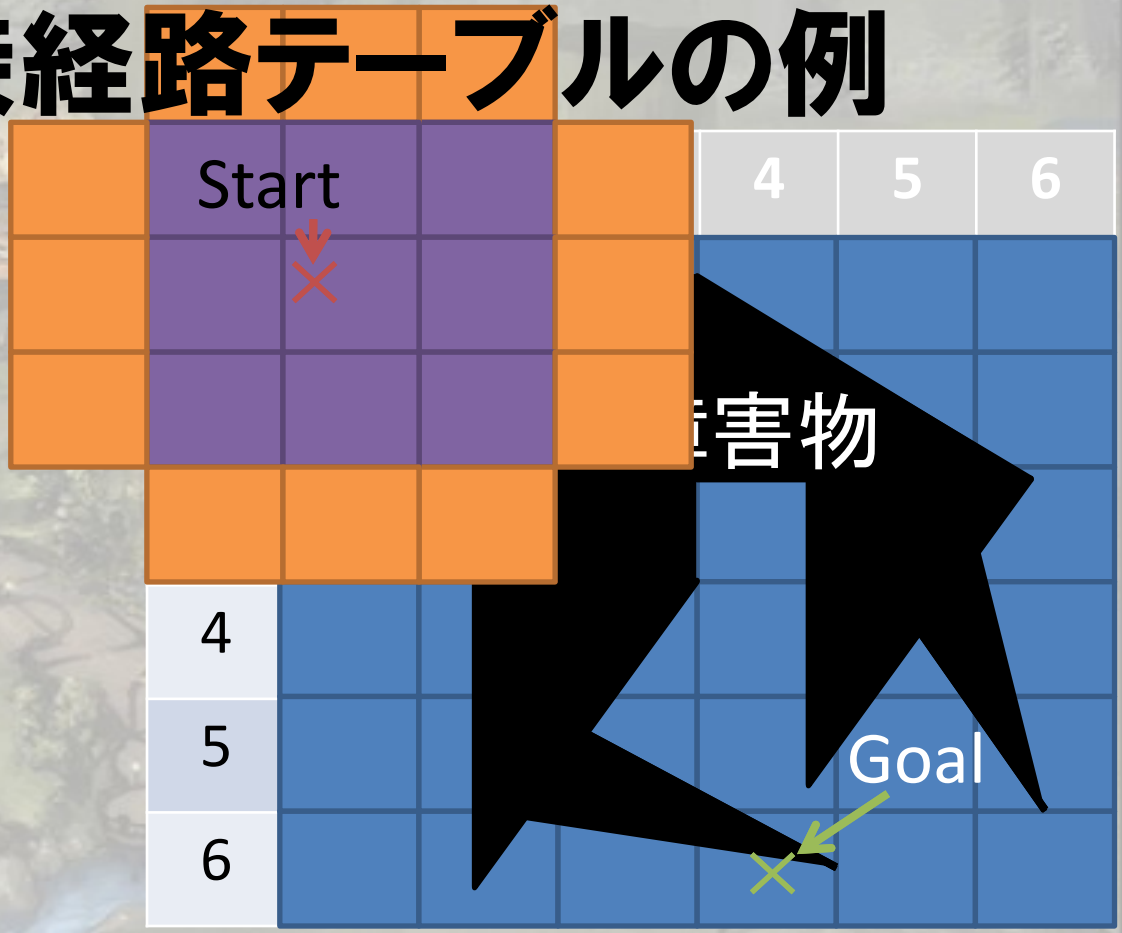
- 経路を見つける



SQUARE ENIX

階層化隣接経路テーブルの例

- 経路を見つける
  - 現在のテーブル



階層化隣接経路テーブルの例

- 経路を見つける
  - 現在のテーブルにゴールがない

⇒上位テーブルを調査

階層化隣接経路テーブルの例

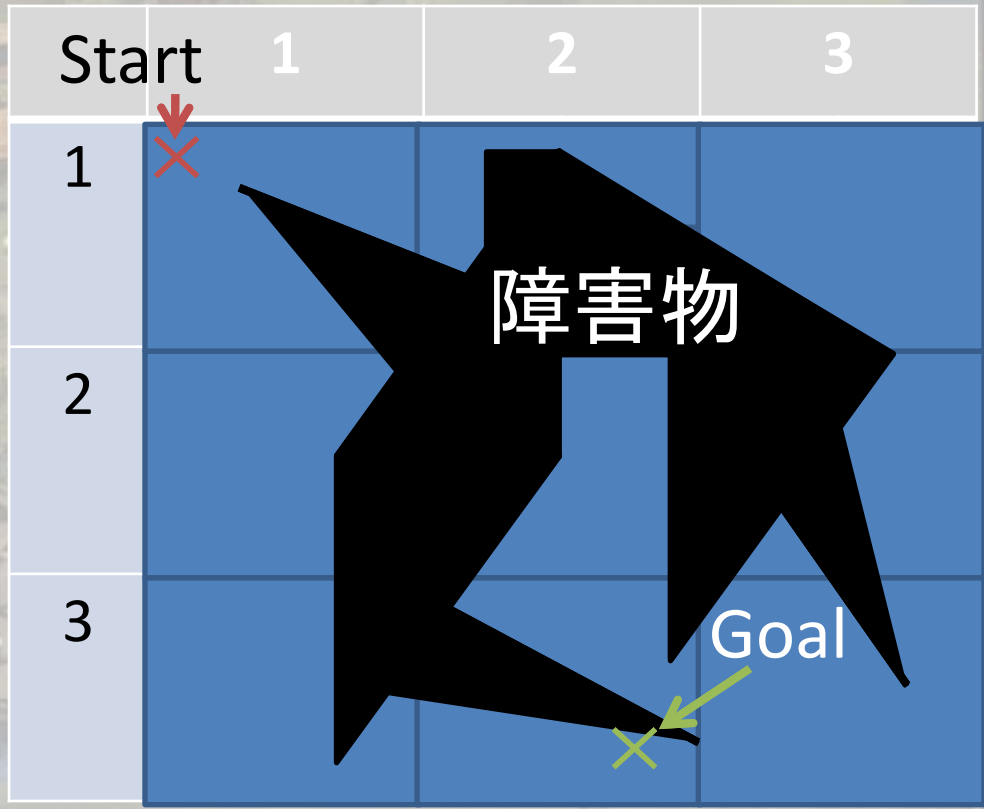
- 経路を見つける
  - 現在のテーブルに
    ゴールがない

    ⇒上位テーブルを調査

    ⇒ゴールがあった

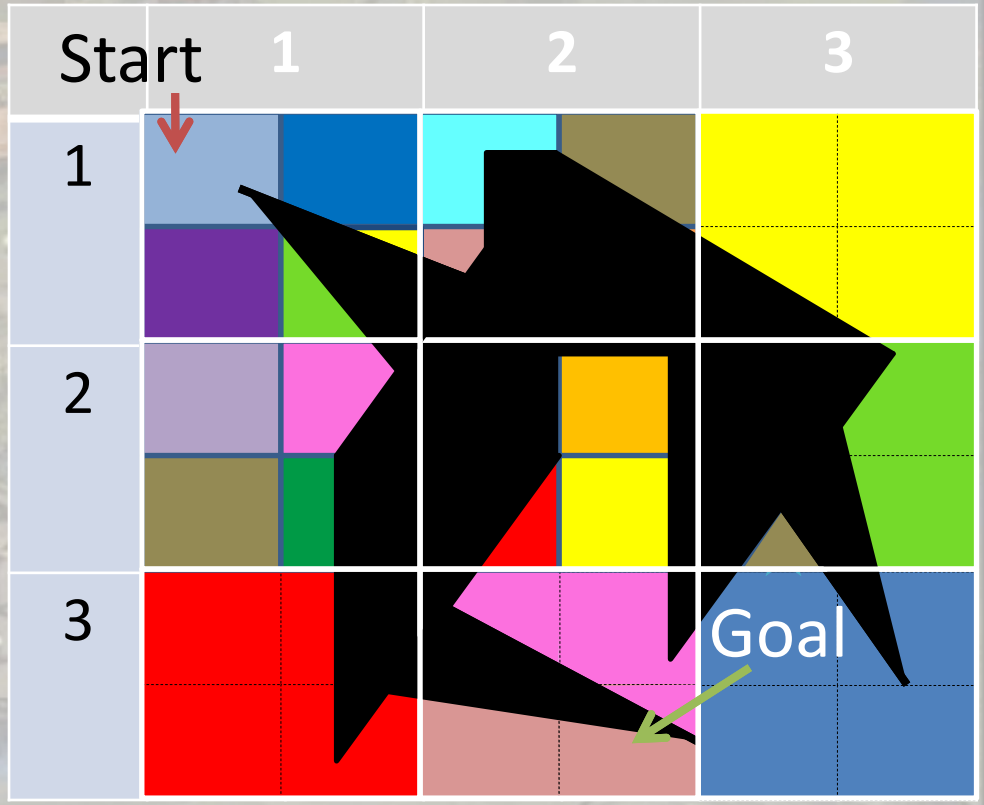
階層化隣接経路テーブルの例

- 上位層のコンポーネントを調べる



階層化隣接経路テーブルの例

- 上位層のコンポーネントを調べる

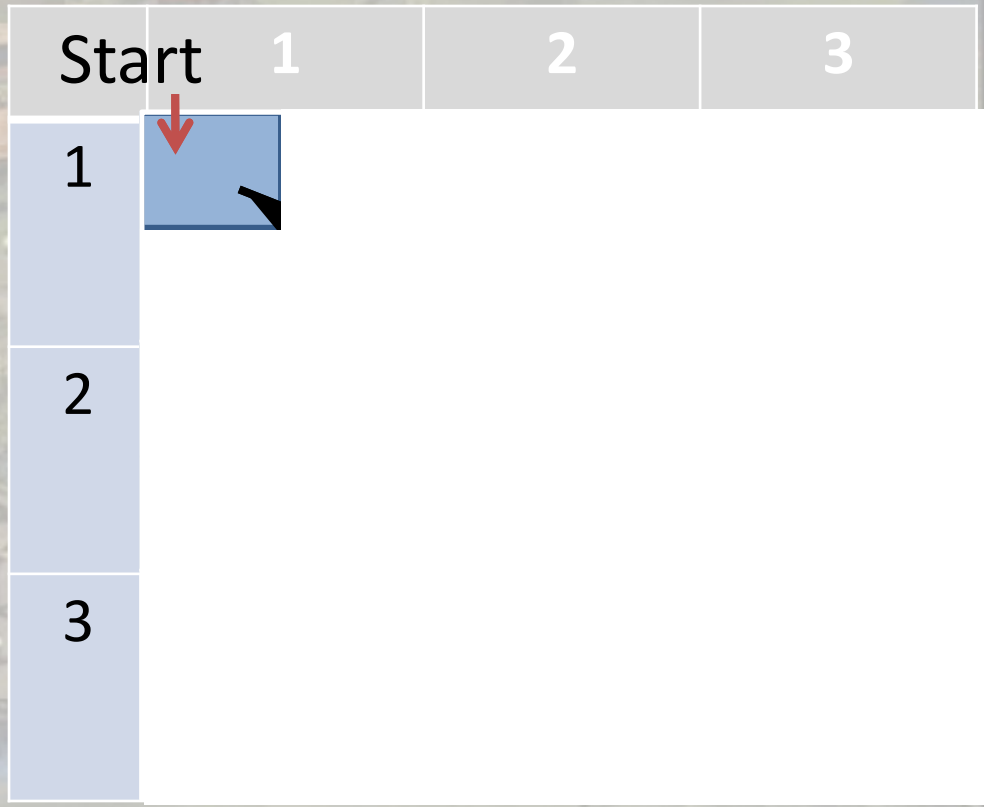


階層化隣接経路テーブルの例

- 上位層のコンポーネントを調べる

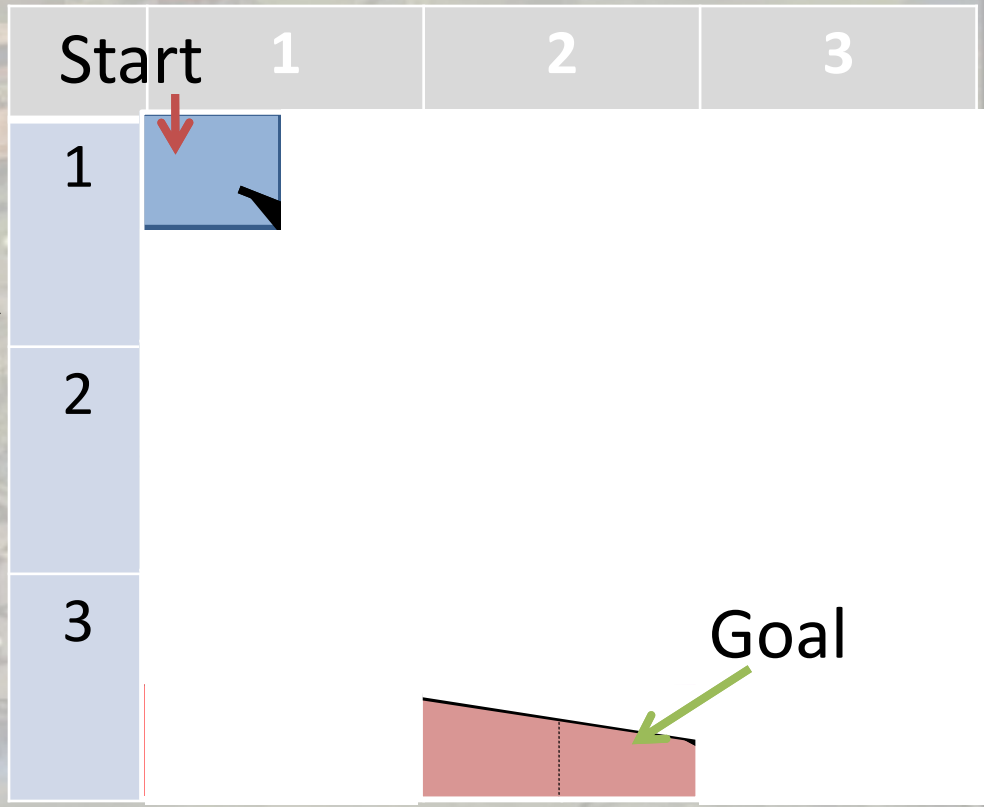
階層化隣接経路テーブルの例

- 上位層のコンポーネントを調べる
- スタートから、



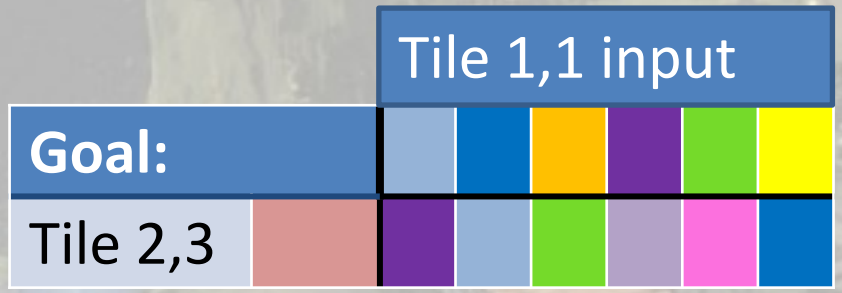
階層化隣接経路テーブルの例

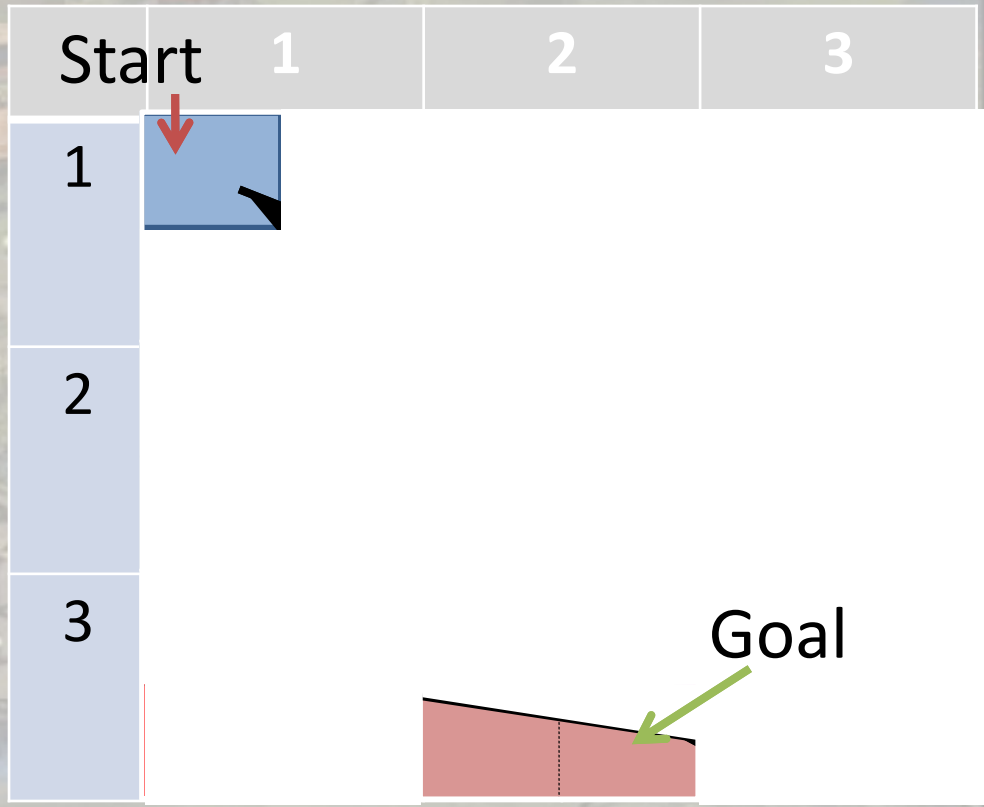
- 上位層のコンポーネントを調べる
- スタートから、ゴールまで



階層化隣接経路テーブルの例

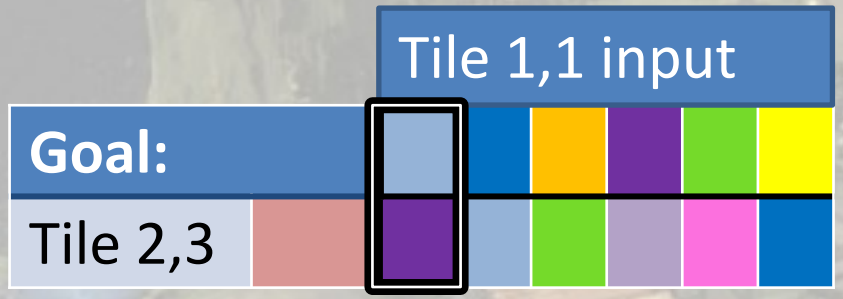
- 経路を見つける
  - をゴールにする



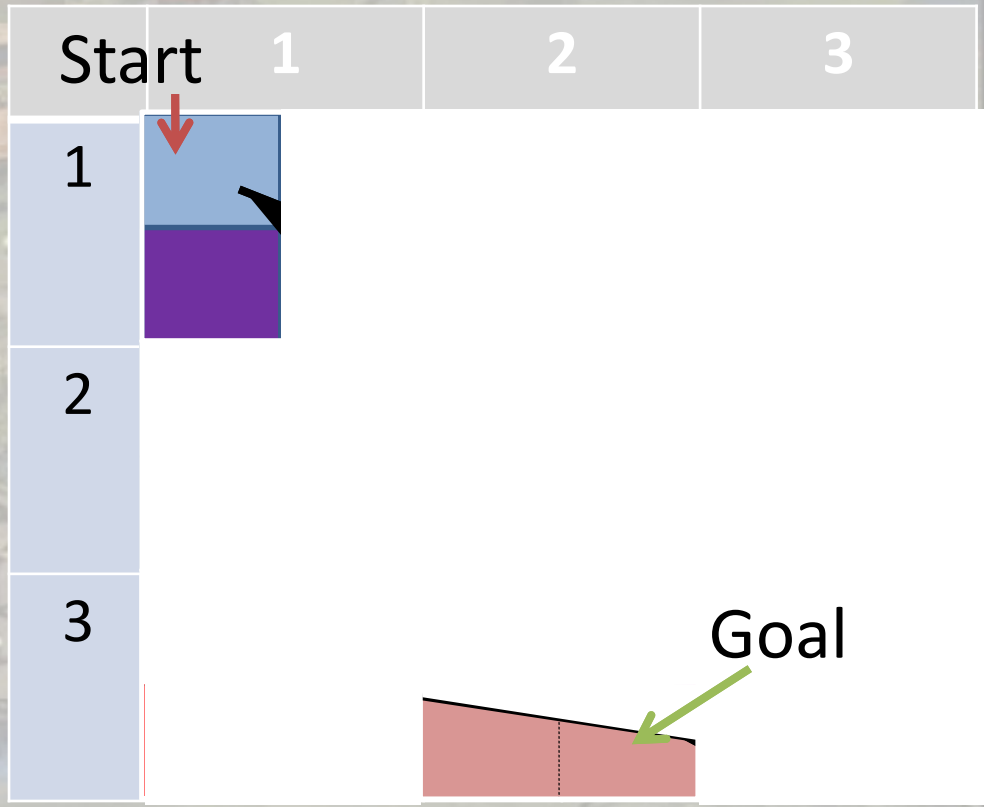


階層化隣接経路テーブルの例

- 経路を見つける
  - をゴールにする

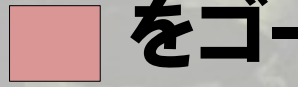


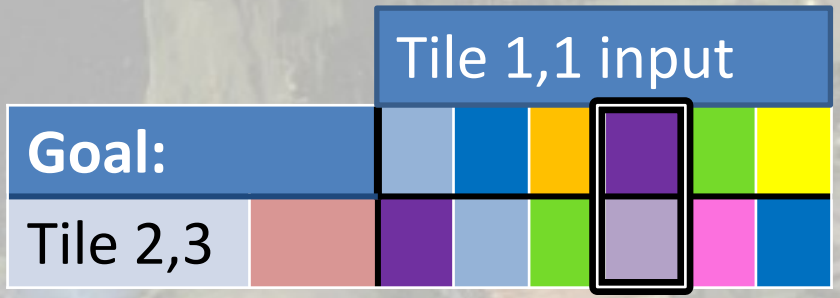
パス:



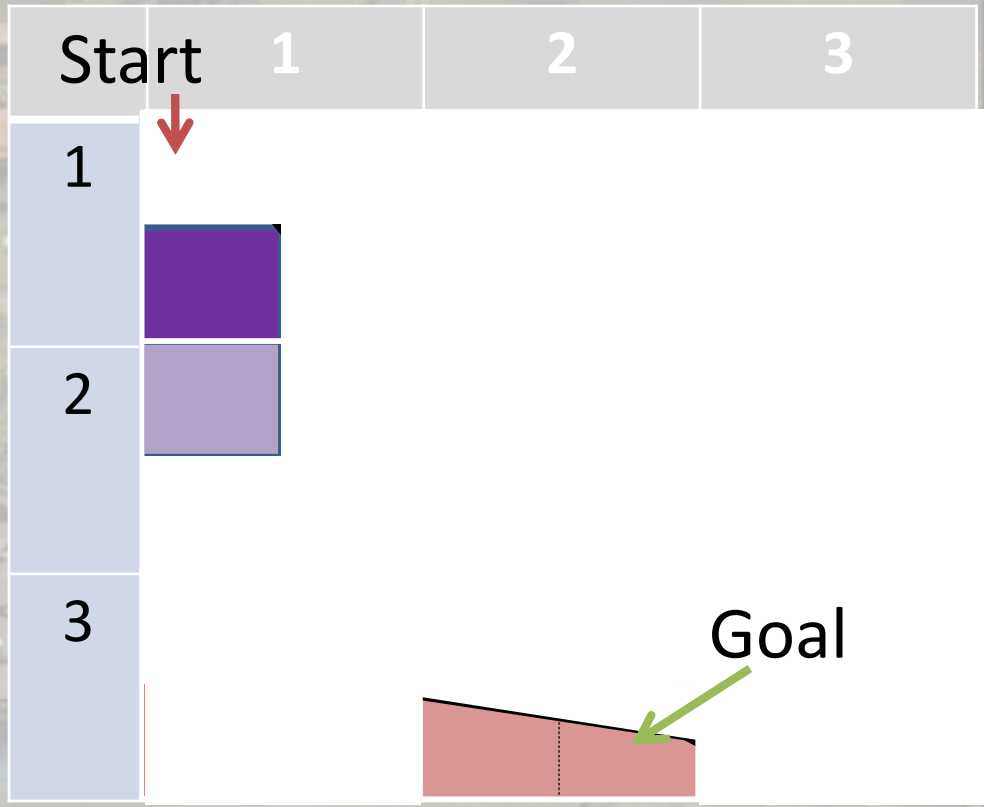
階層化隣接経路テーブルの例

- 経路を見つける


をゴールにする

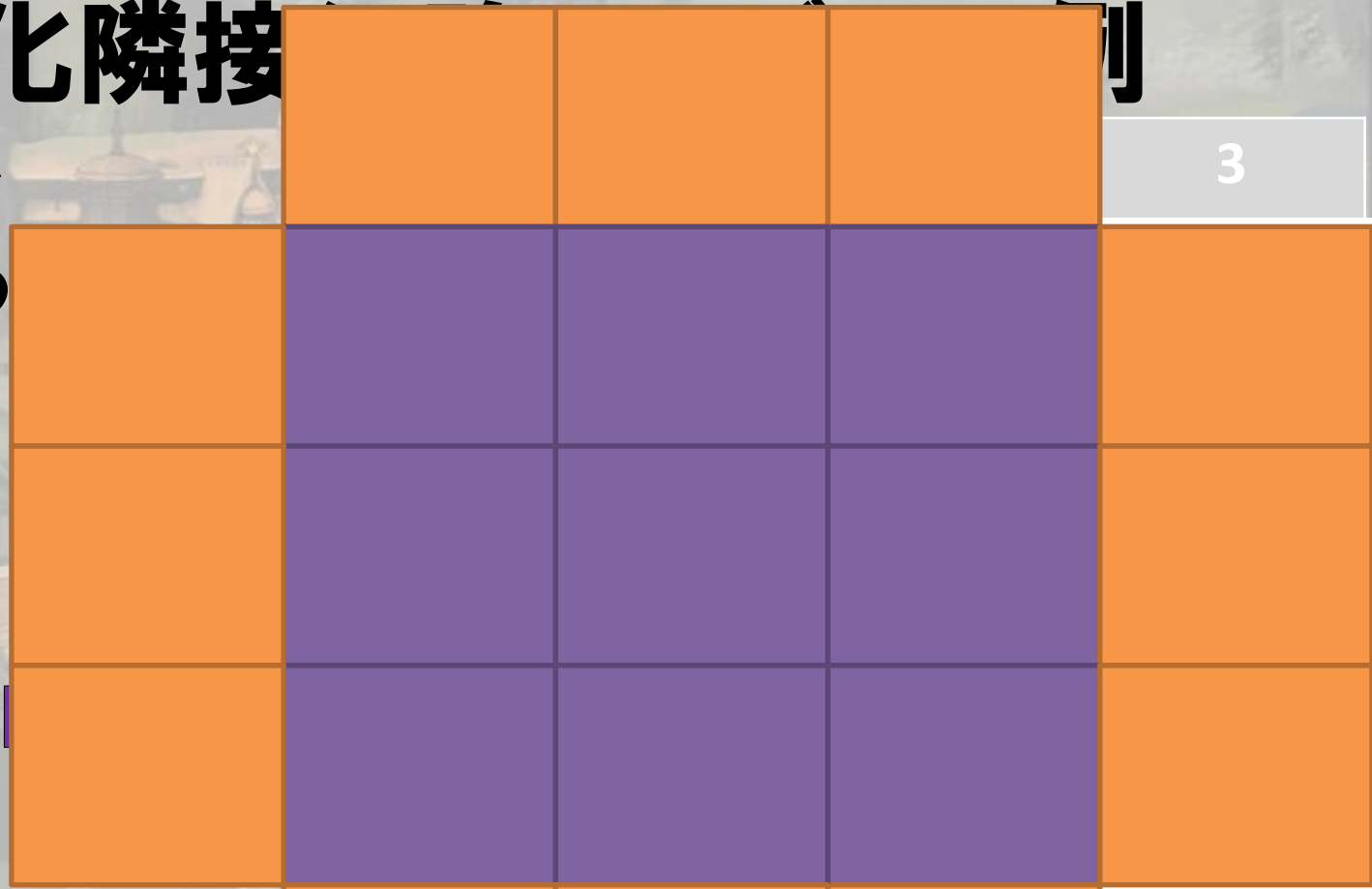


パス:



階層化隣接

- テーブルを
  切り替える

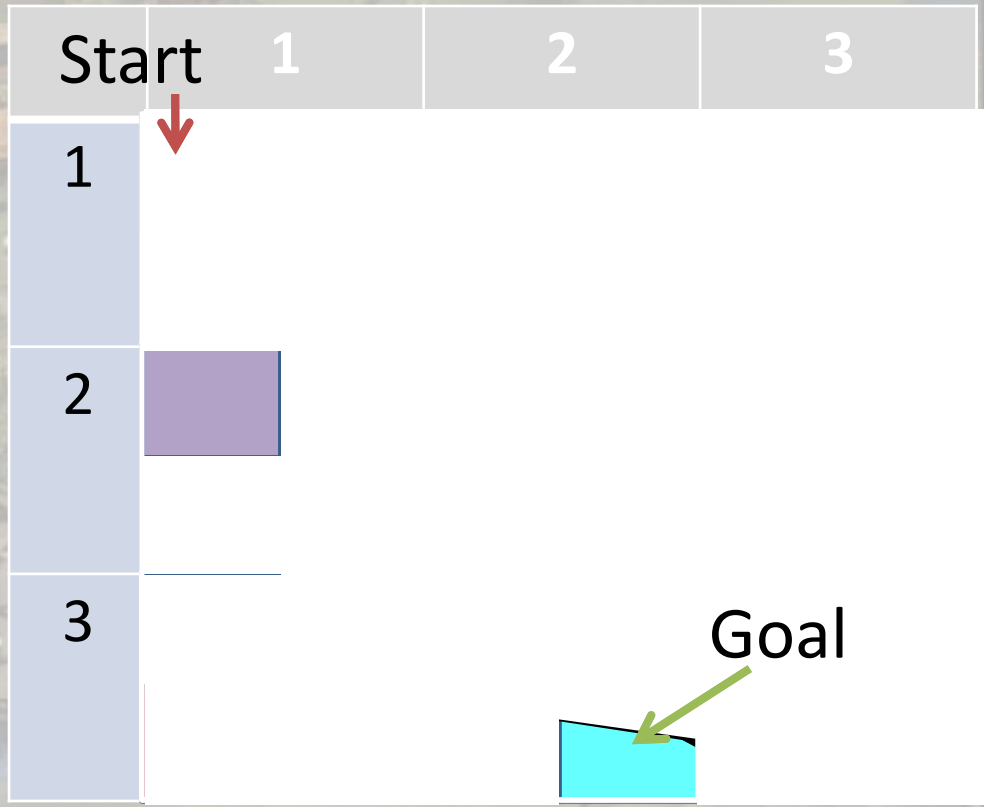


パス:

階層化隣接経路テーブルの例

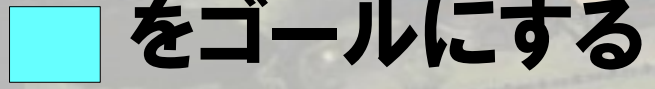
- 経路を見つける
  - をゴールにする

パス:

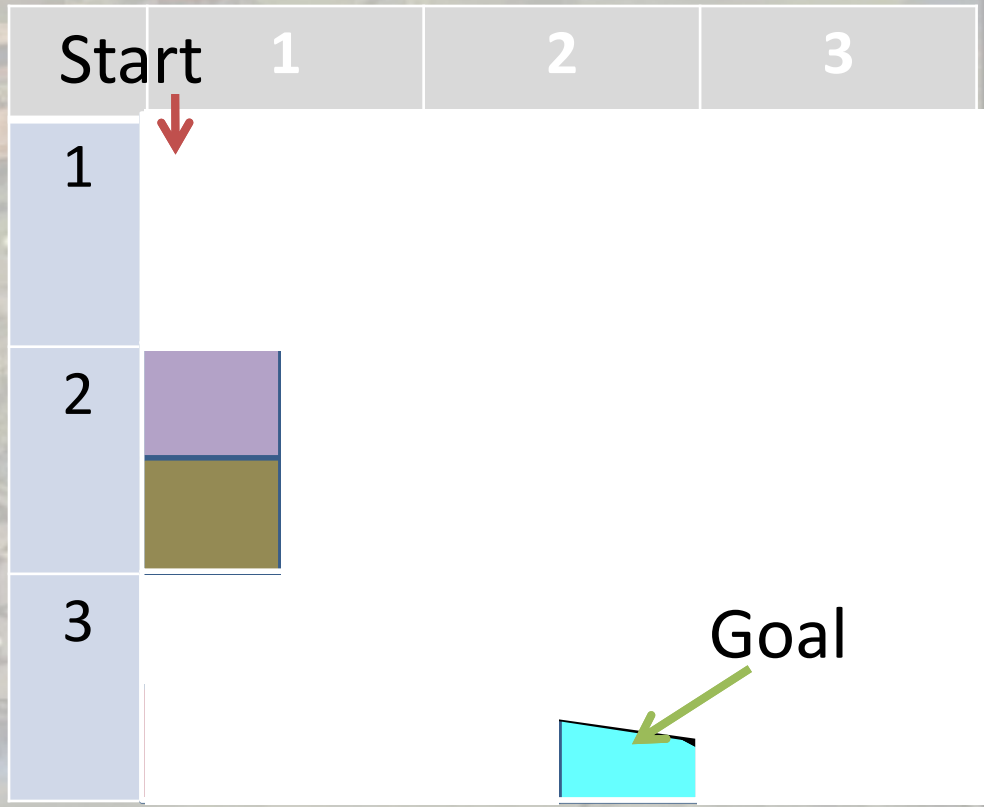


階層化隣接経路テーブルの例

- 経路を見つける

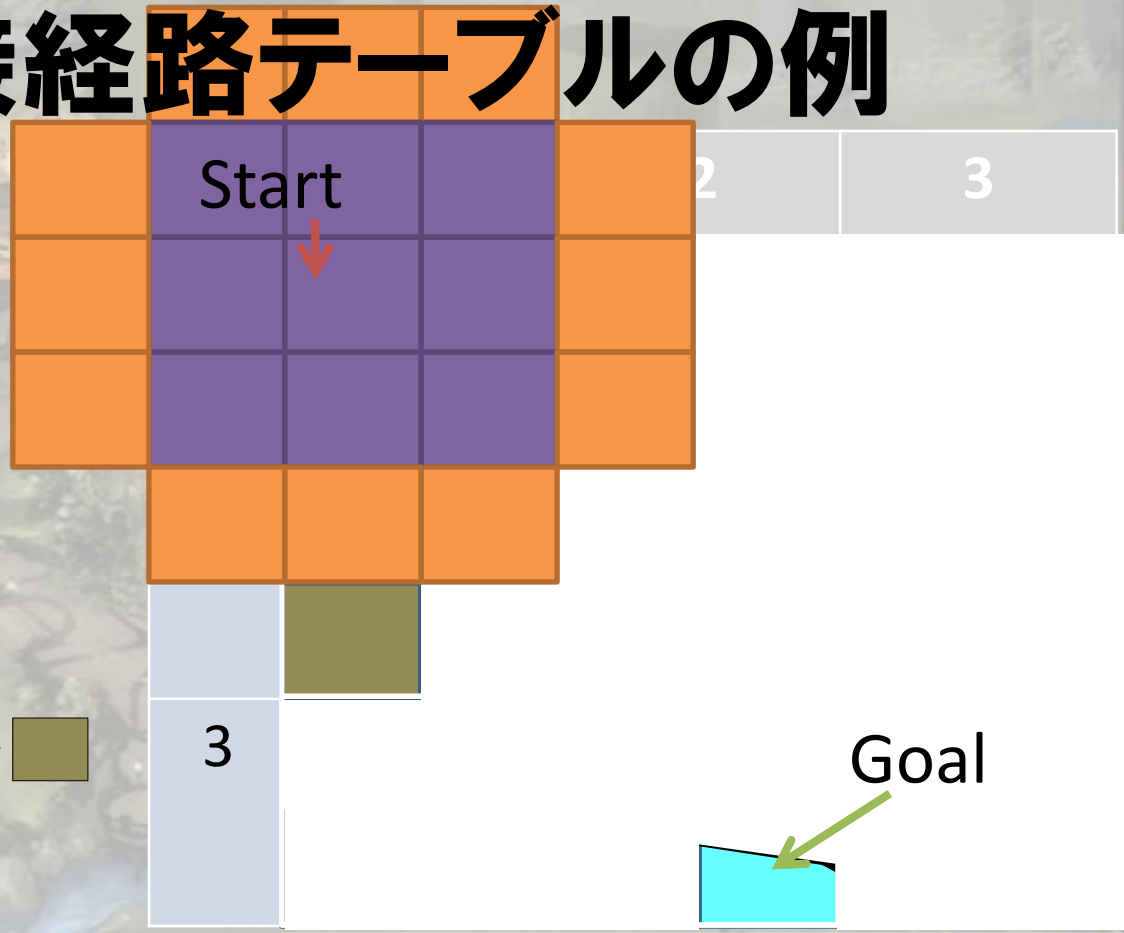

をゴールにする

パス:



階層化隣接経路テーブルの例

- 経路を見つける
  - をゴールにする

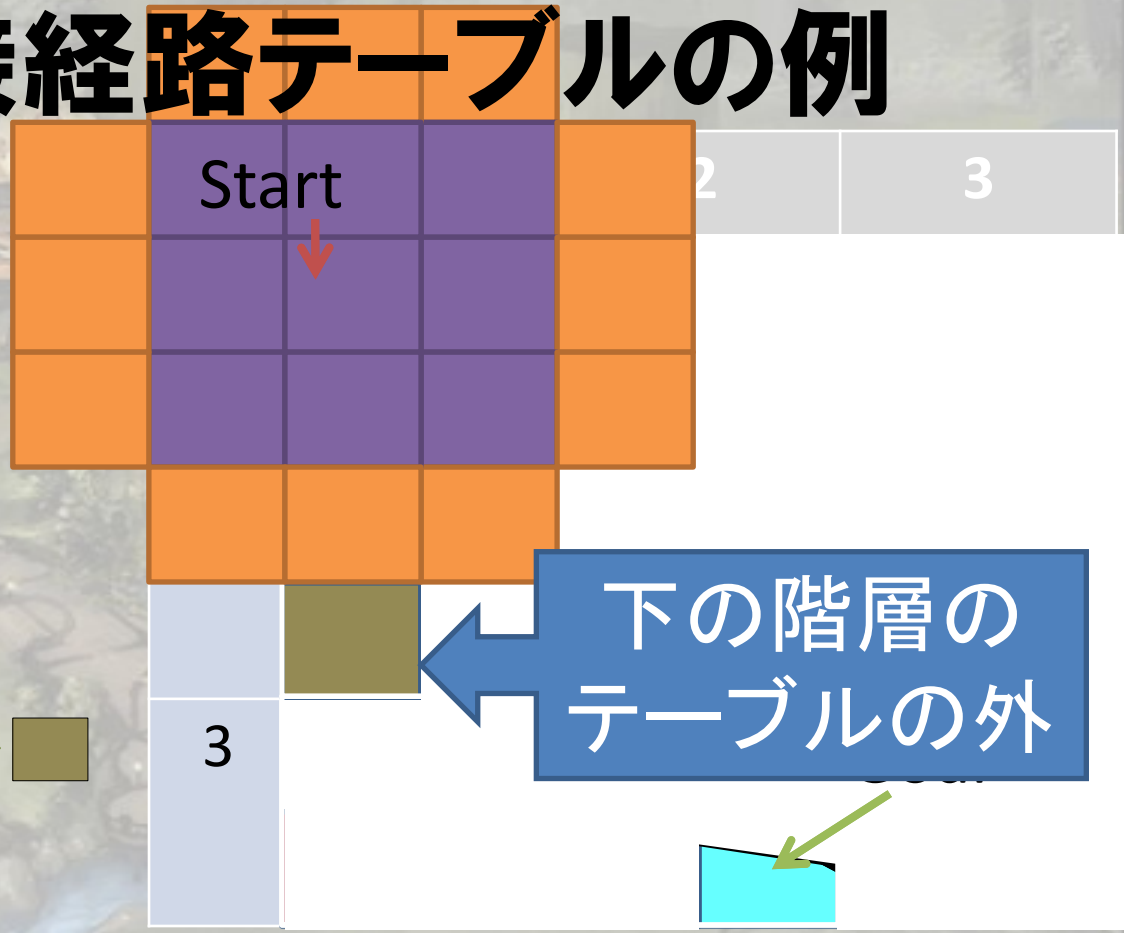


パス:

SQUARE ENIX

階層化隣接経路テーブルの例

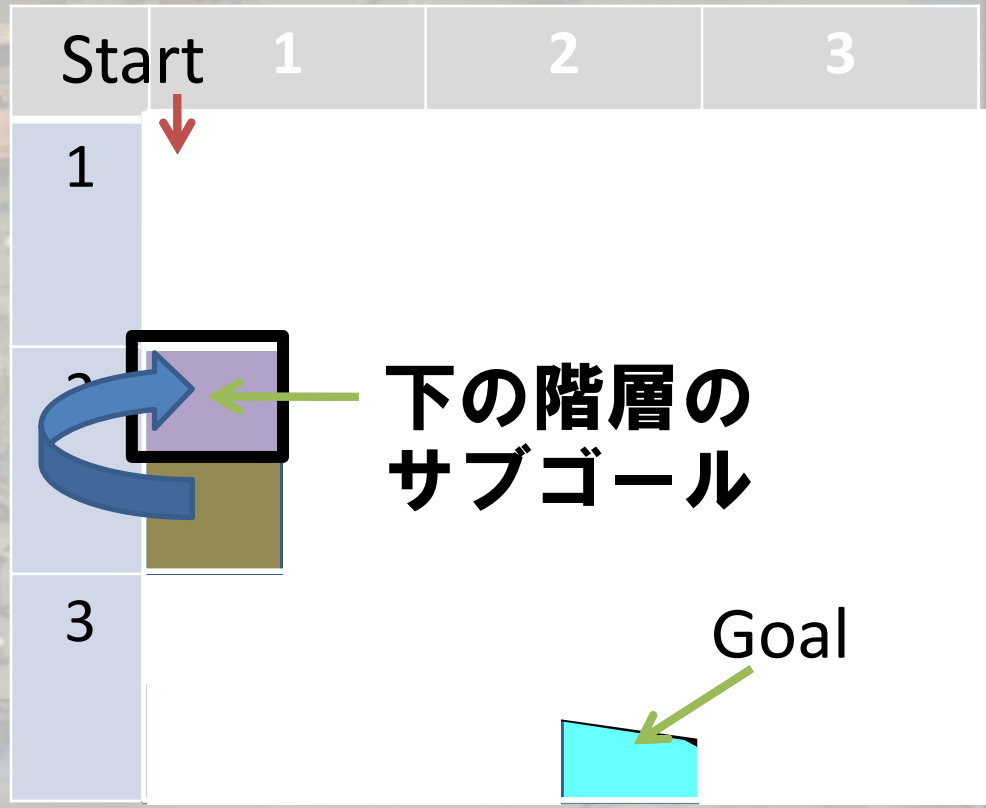
- 経路を見つける
  - をゴールにする



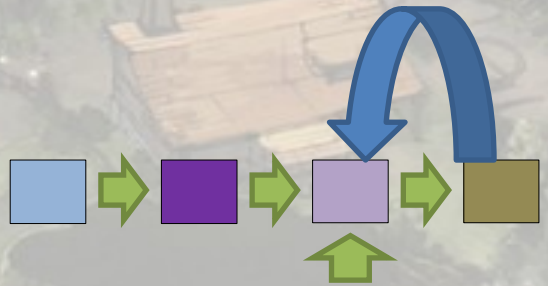
パス:

階層化隣接経路テーブルの例

- 経路を見つける
  - をゴールにする



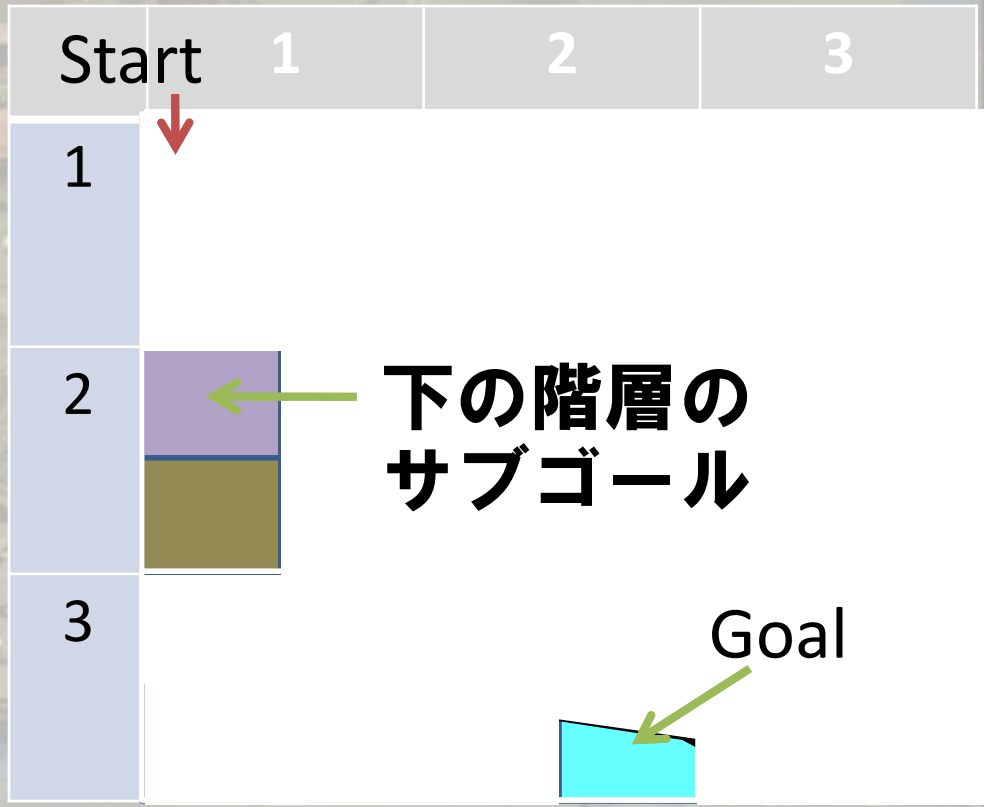
パス:



下の階層のサブゴール

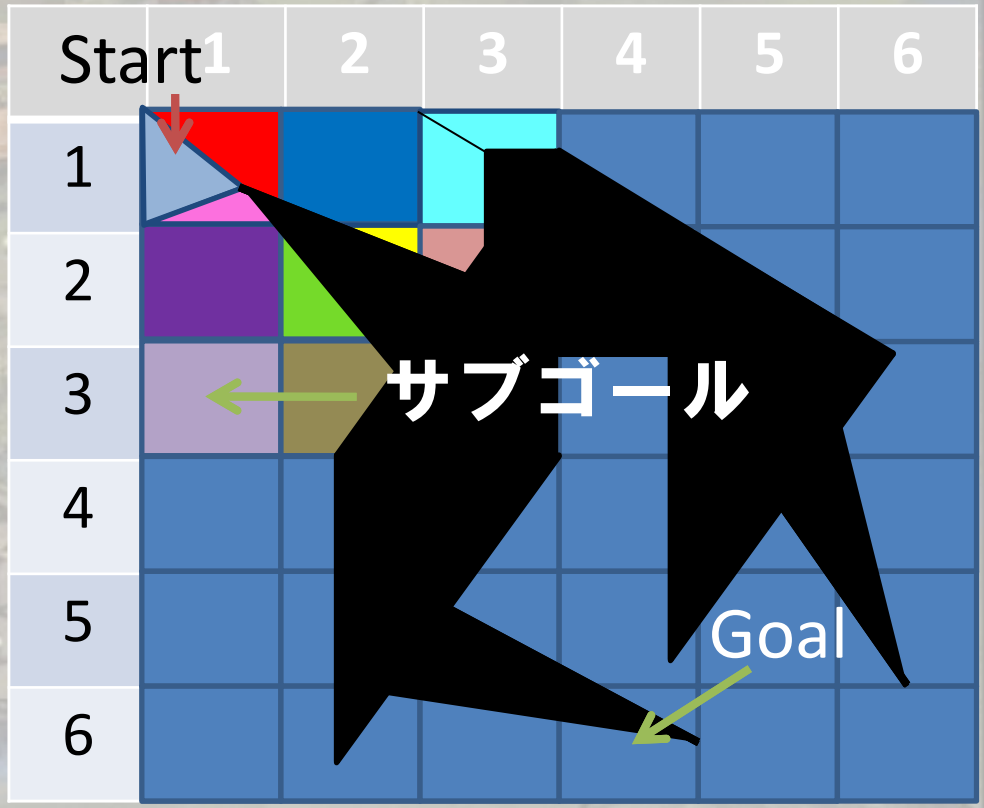
階層化隣接経路テーブルの例

上の階層から…



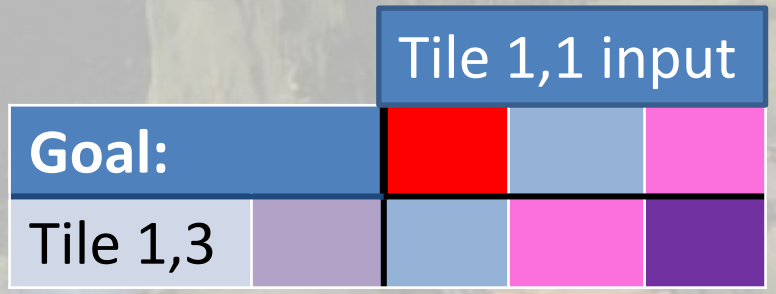
階層化隣接経路テーブルの例

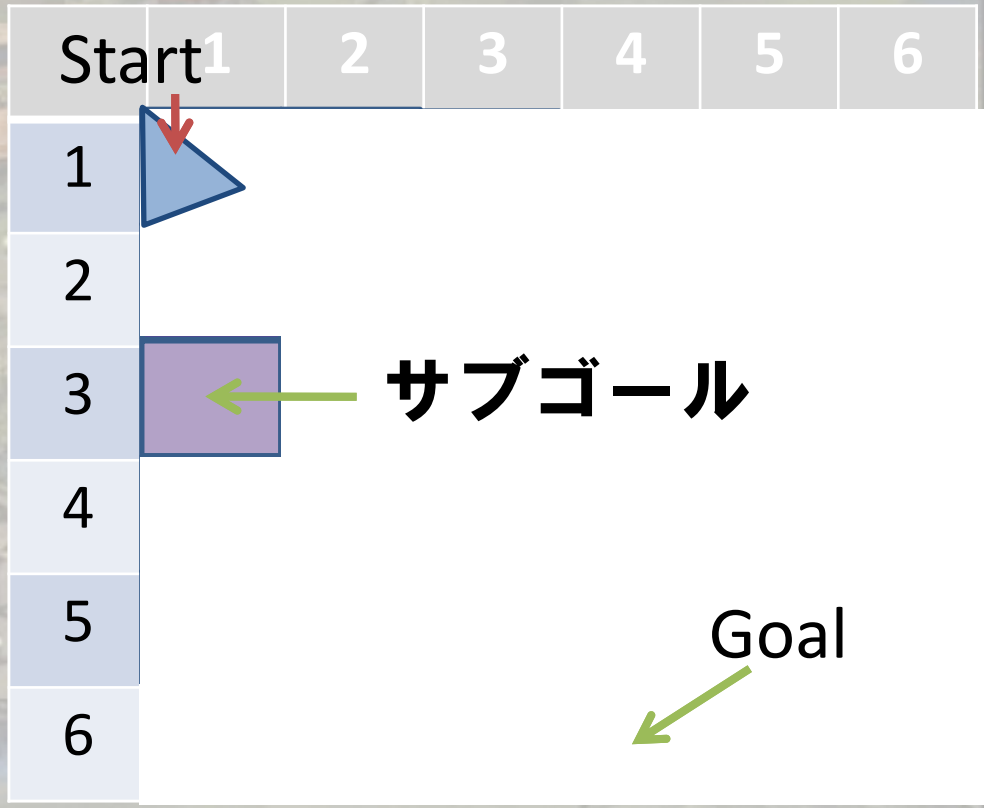
上の階層から、
下の階層に切り替える



階層化隣接経路テーブルの例

- 経路を見つける
  - をサブゴールにする

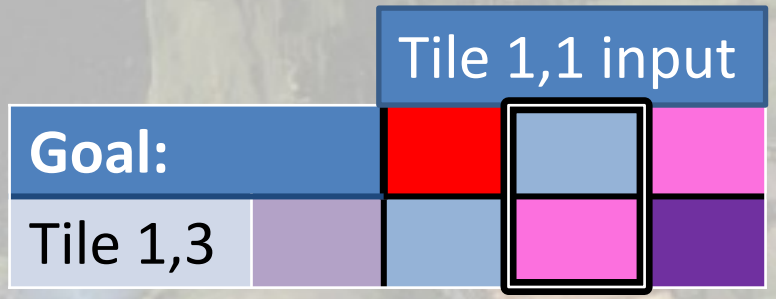




階層化隣接経路テーブルの例

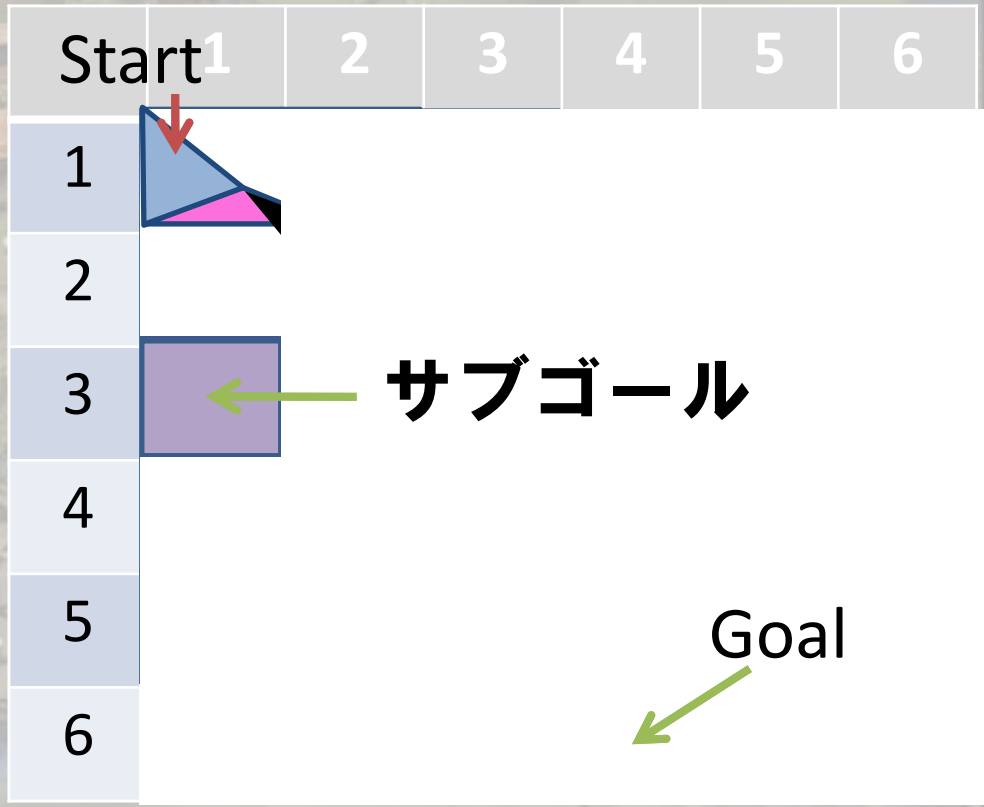
- 経路を見つける

■ をサブゴールにする



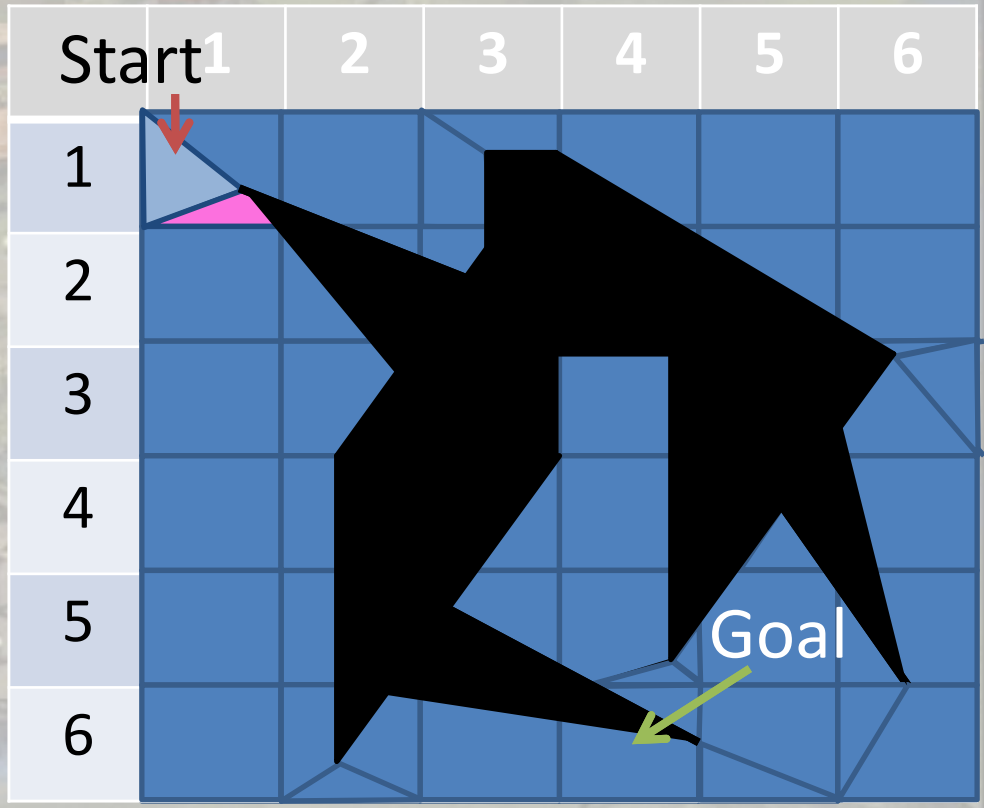
パス:

次のメッシュ



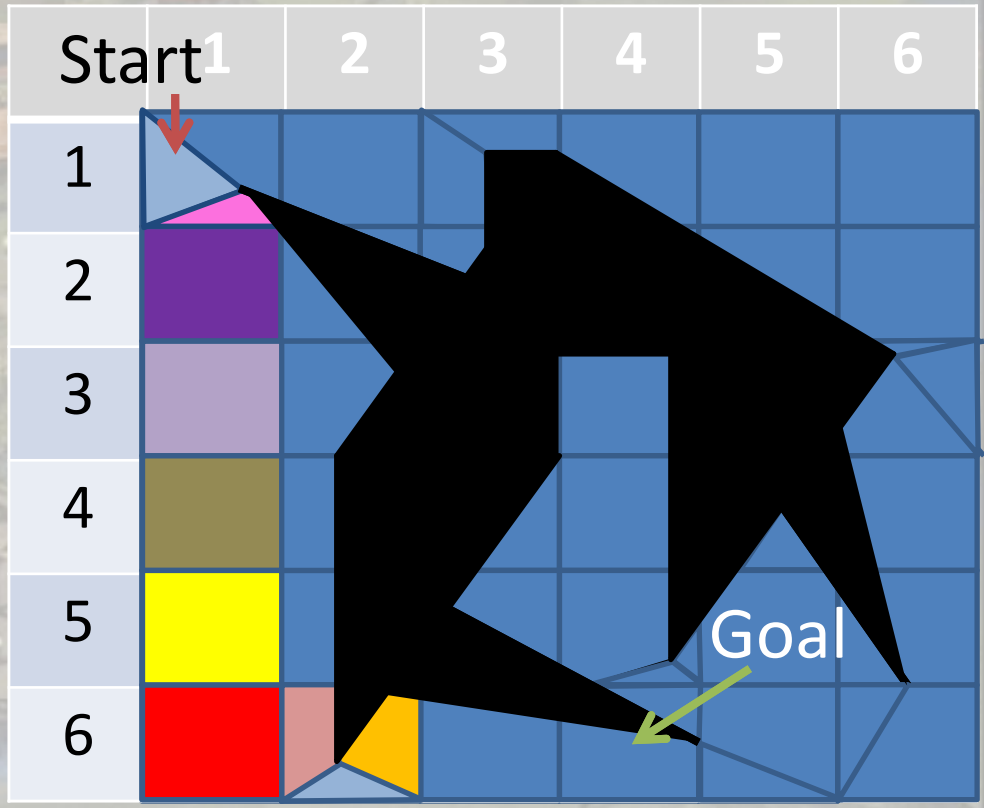
階層化隣接経路テーブルの例

- 次のポリゴン



階層化隣接経路テーブルの例

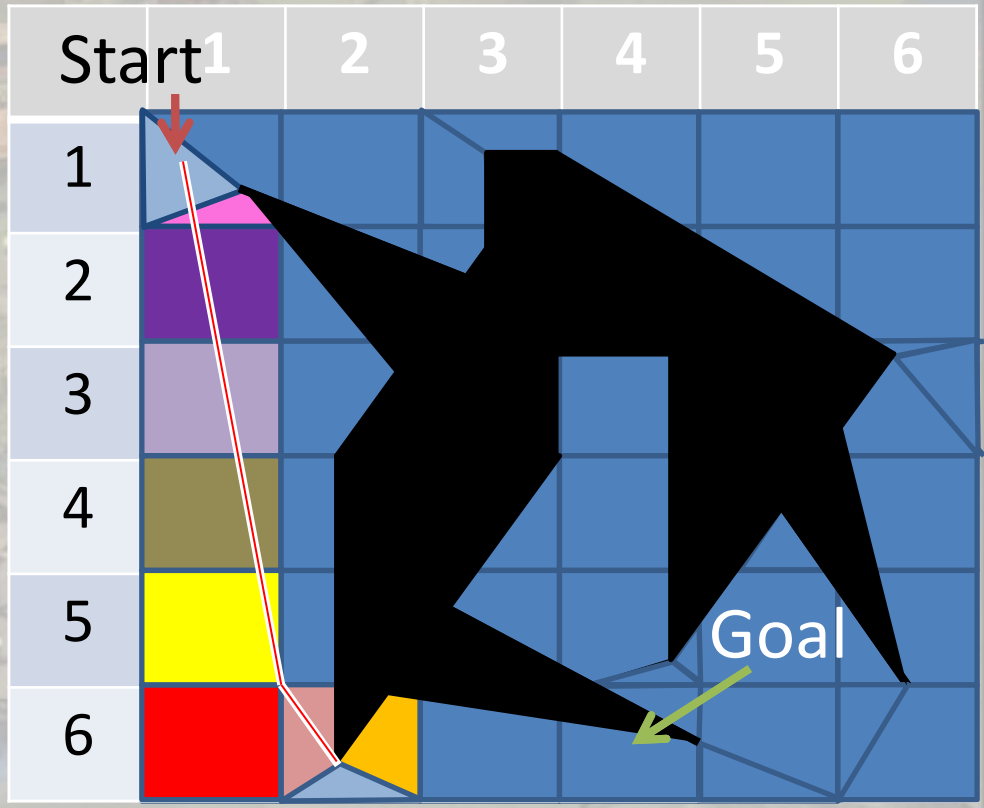
- くりかえし



階層化隣接経路テーブルの例

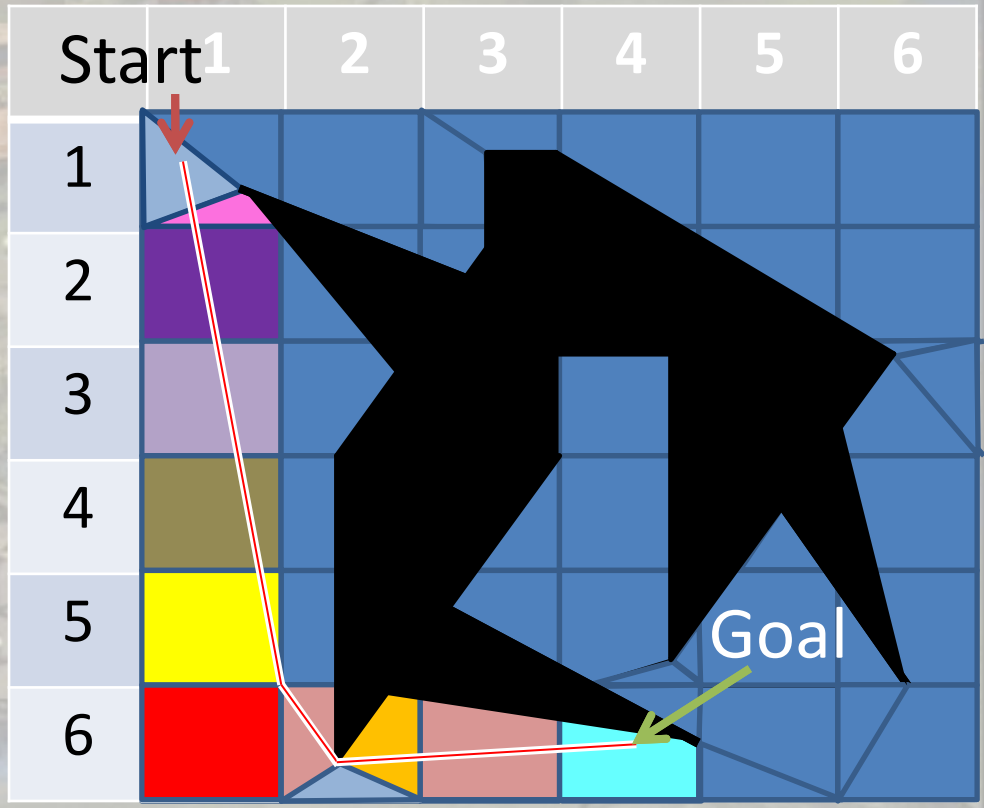
- スムージングする

※ファンネルアルゴリズム



階層化隣接経路テーブルの例

- くりかえし
- ゴールに着きました



第4部
ナビメッシュ自動生成と落下

実際の経路探索の例

- 動画

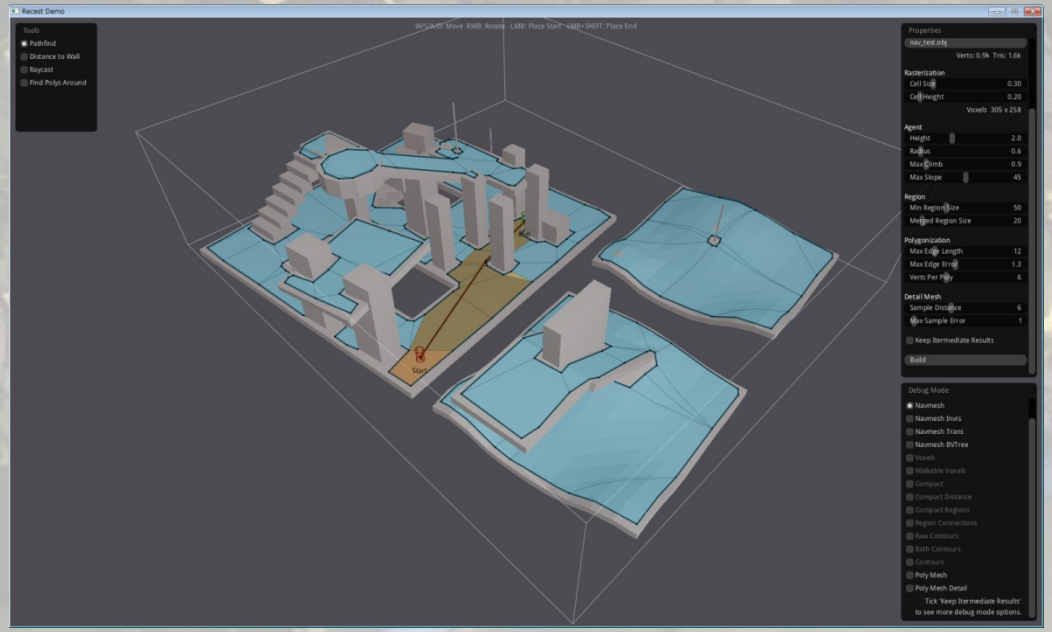
第4部 ナビメッシュ自動生成と落下

- ナビメッシュ自動生成
- 歩けるメッシュ生成（Recast）
- 落下メッシュ生成

SQUARE ENIX

ナビメッシュ自動生成

- Recast Navigationを使用



Recast Navigation

- オープンソース(MITライセンス)のナビゲーションメッシュ作成ツール
- Mikko Mononen 氏が作成
- C++ソースコードで提供
- http://code.google.com/p/recastnavigation/

第4部 ナビメッシュ自動生成と落下

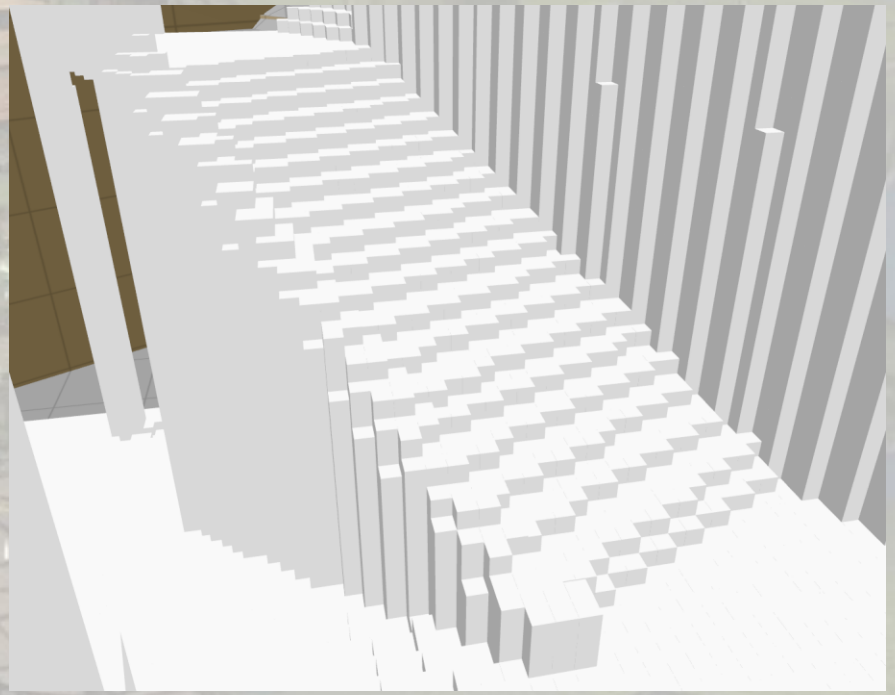
- ナビメッシュ自動生成
- 歩けるメッシュ生成（Recast）
- 落下メッシュ生成

歩けるメッシュ生成

- コリジョン入力
- ボクセル化
- 歩けるボクセル検出
- 歩ける領域作成
- 輪郭作成
- メッシュ作成

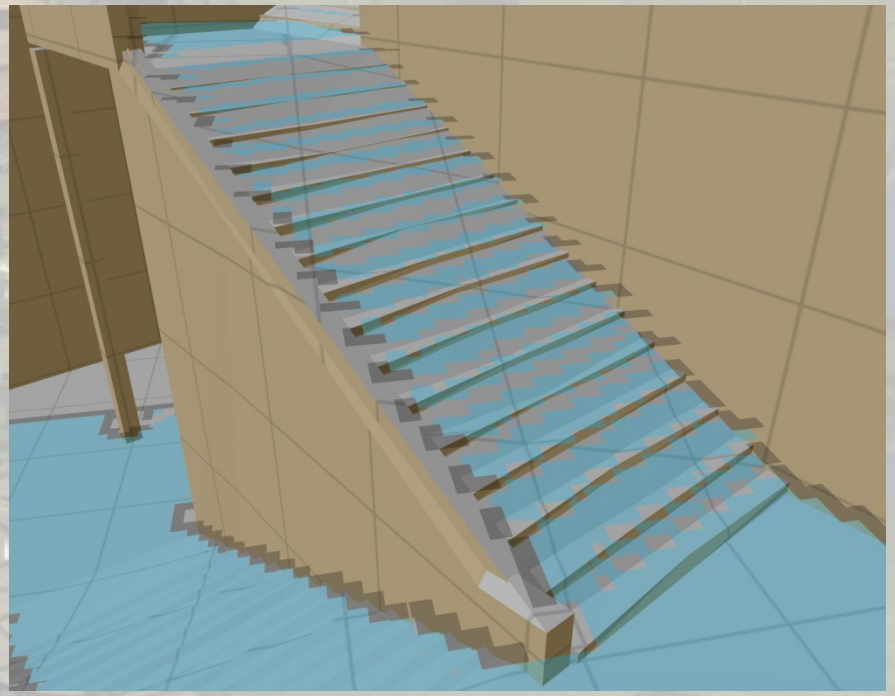
歩けるメッシュ生成

- コリジョン入力
- ボクセル化
- 歩けるボクセル検出
- 歩ける領域作成
- 輪郭作成
- メッシュ作成



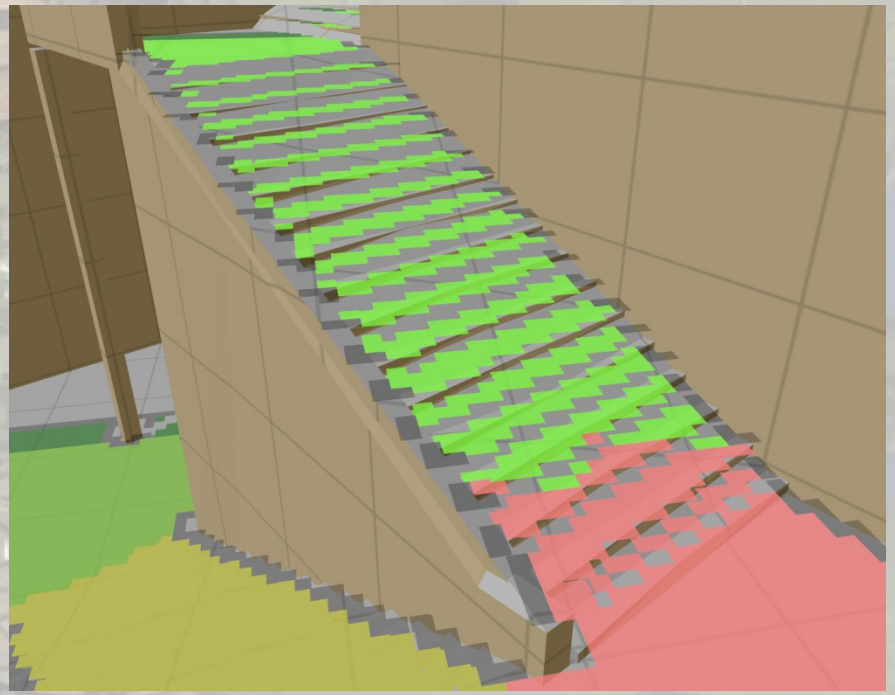
歩けるメッシュ生成

- コリジョン入力
- ボクセル化
- **歩けるボクセル検出**
  - **キャラクター半径、高さ、登れる高さ、登れる角度**
- 歩ける領域作成
- 輪郭作成
- メッシュ作成



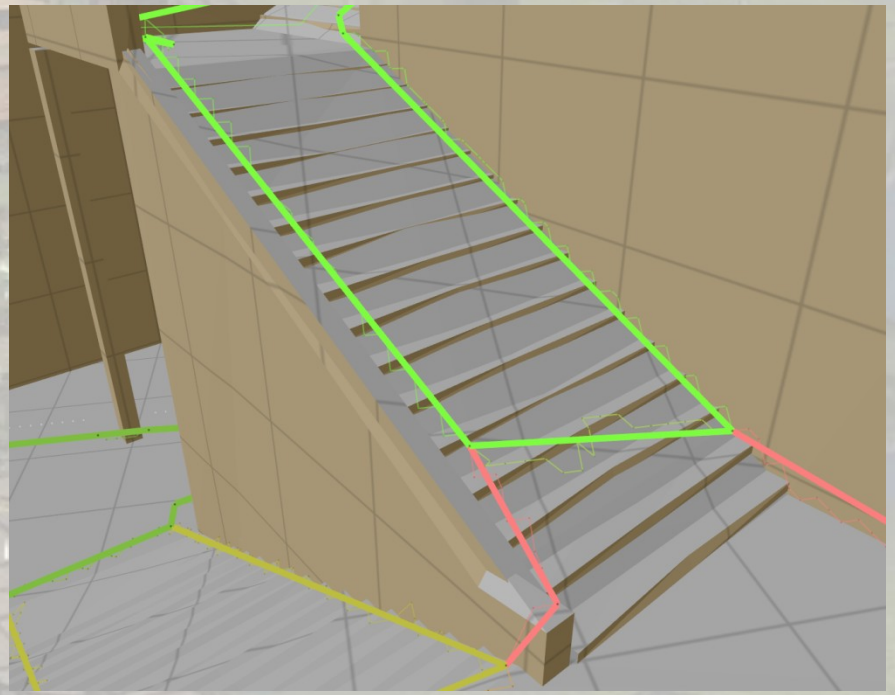
歩けるメッシュ生成

- コリジョン入力
- ボクセル化
- 歩けるボクセル検出
- **歩ける領域作成**
- 輪郭作成
- メッシュ作成



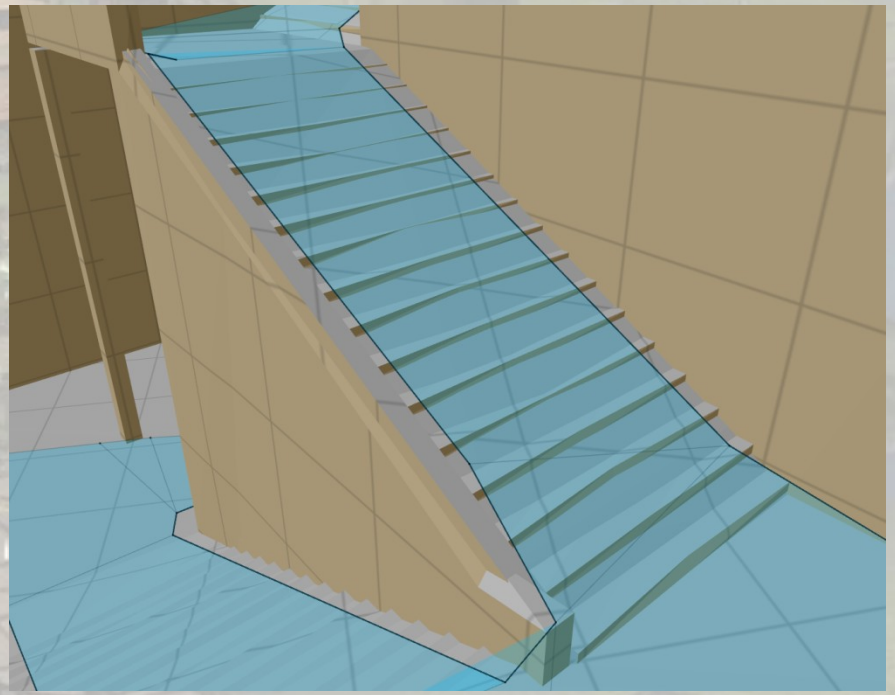
歩けるメッシュ生成

- コリジョン入力
- ボクセル化
- 歩けるボクセル検出
- 歩ける領域作成
- **輪郭作成**
- メッシュ作成



歩けるメッシュ生成

- コリジョン入力
- ボクセル化
- 歩けるボクセル検出
- 歩ける領域作成
- 輪郭作成
- メッシュ作成



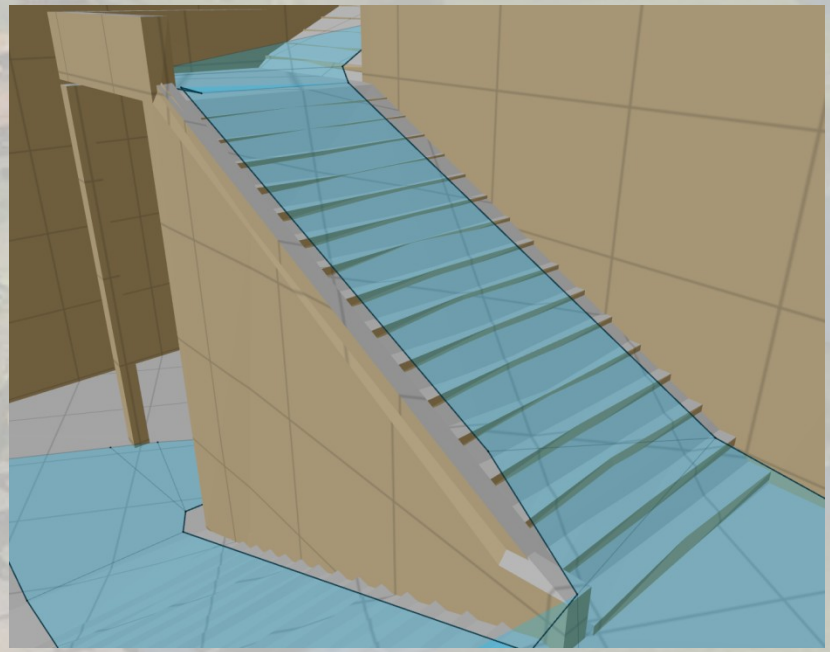
FFXIVでのナビメッシュ

第4部 ナビメッシュ自動生成と落下

- ナビメッシュ自動生成
- 歩けるメッシュ生成（Recast）
- 落下メッシュ生成

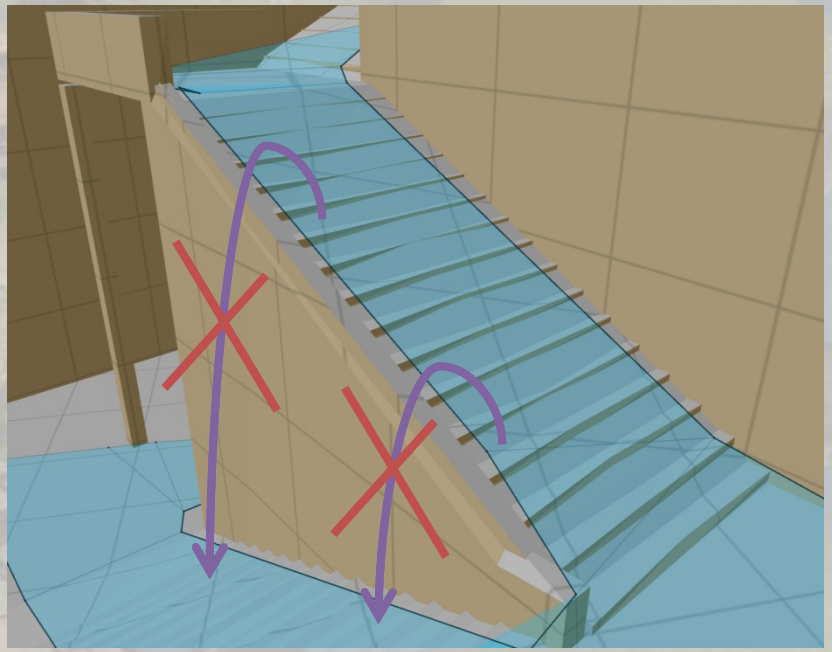
落下メッシュ生成

- メッシュ
  - どこからでも落下できる



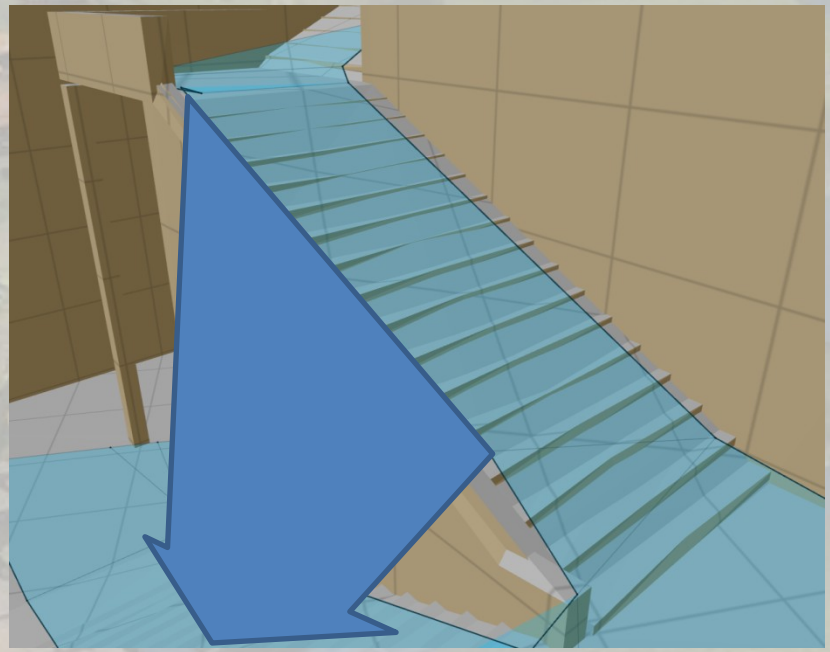
落下メッシュ生成

- メッシュ
  - どこからでも落下できる



落下メッシュ生成

- メッシュ
  - どこからでも落下できる
- アルゴリズム
  - どこから落下するか
  - どこに落下するか



落下メッシュ生成

- 歩けるメッシュ作成の場合
  - コリジョン入力
  - ボクセル化
  - 歩けるボクセル検出
  - 歩ける領域作成
  - 輪郭作成
  - メッシュ作成

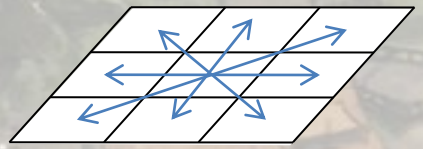
SQUARE ENIX

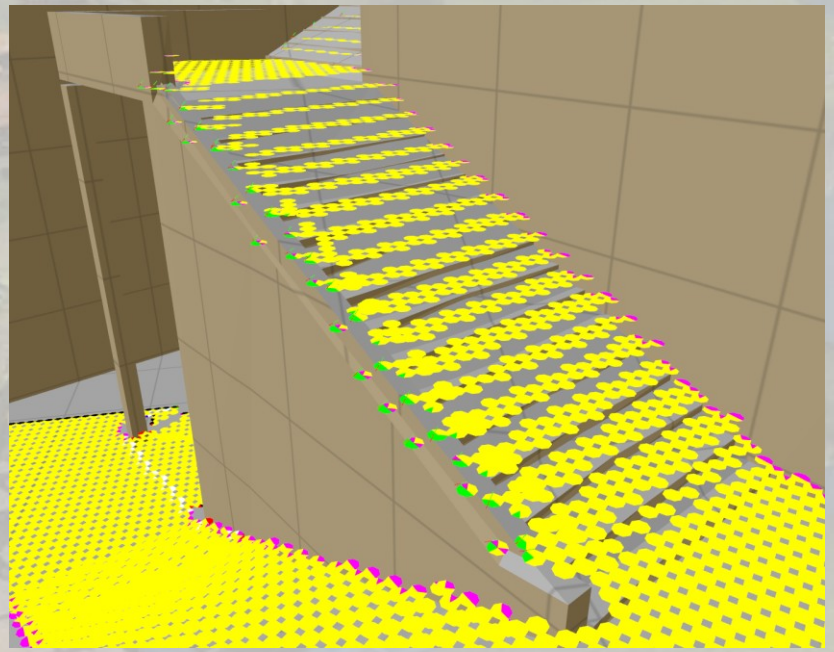
落下メッシュ生成

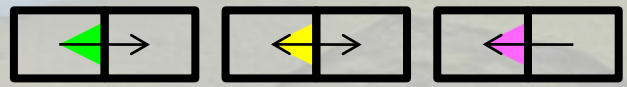
- ボクセル化
- 歩けるボクセル検出
- **落下するボクセル検出**
- 歩ける領域作成
- 輪郭作成
- **落下開始する境界を検出**
- **落下境界への処理**
- メッシュ作成

落下メッシュ生成

- 落下するボクセル検出
  - 各ボクセルから8方向に落下できる

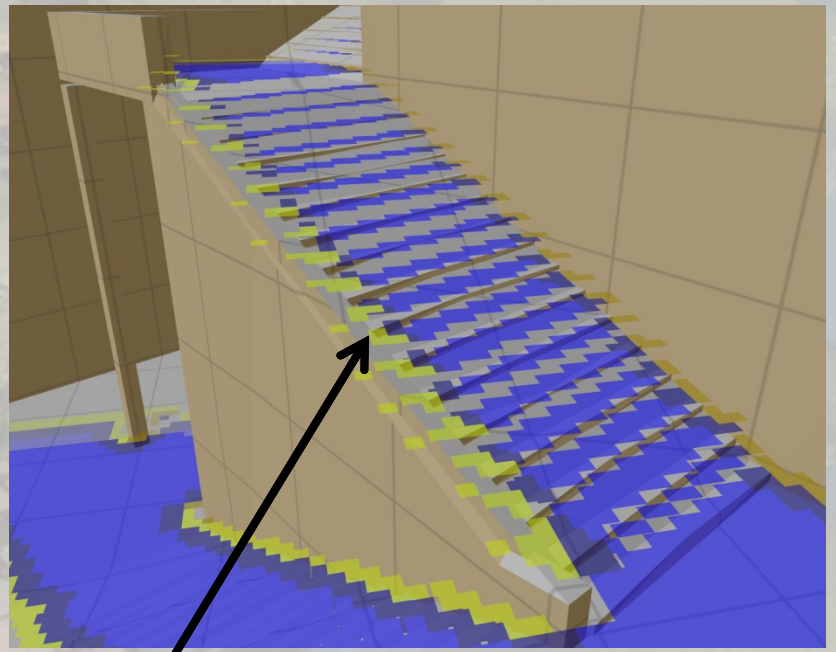






落下メッシュ生成

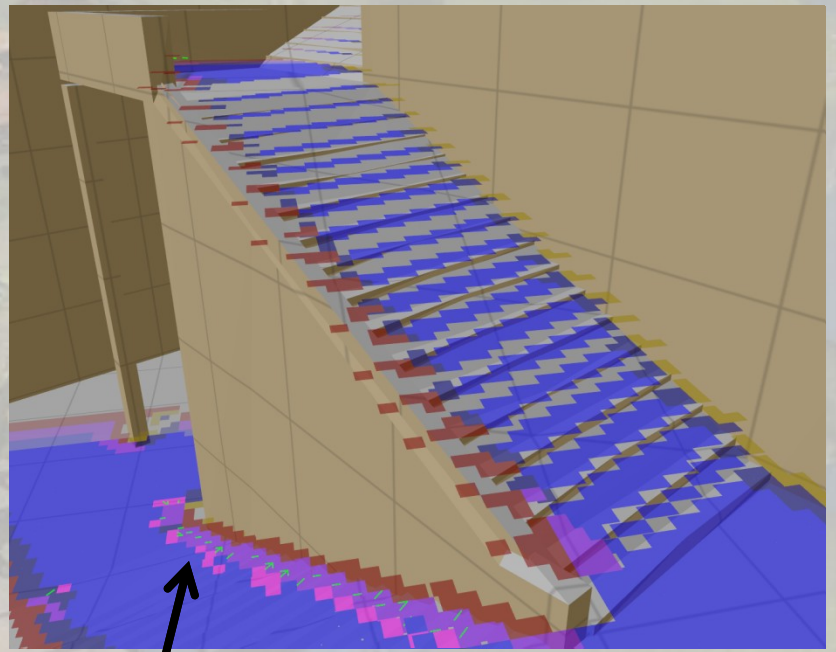
- 落下するボクセル検出
  - キャラクター半径も考慮



落下できる所

落下メッシュ生成

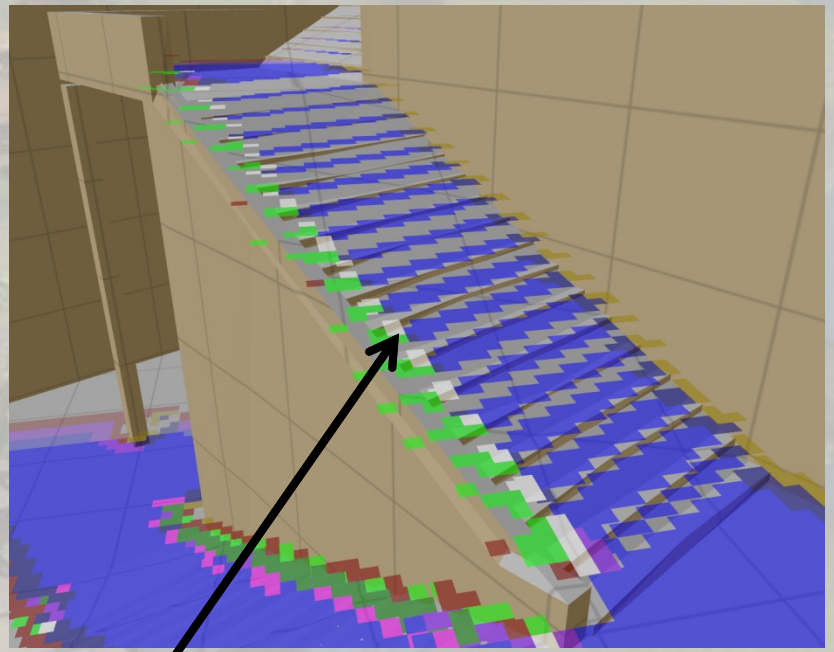
- 落下開始する境界を検出
  - 着地できるところを探す



着地できる所

落下メッシュ生成

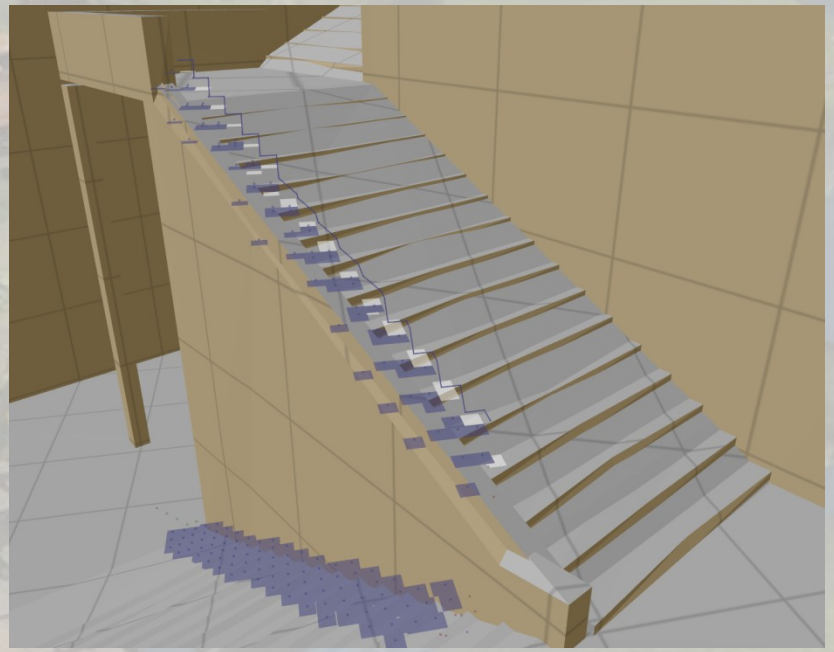
- 落下開始する境界を検出
  - 落下できるところを探す
  - 登れるところを探す



□ 落下スタート地点

落下メッシュ生成

- 落下境界に対して
  - 落下領域を作る

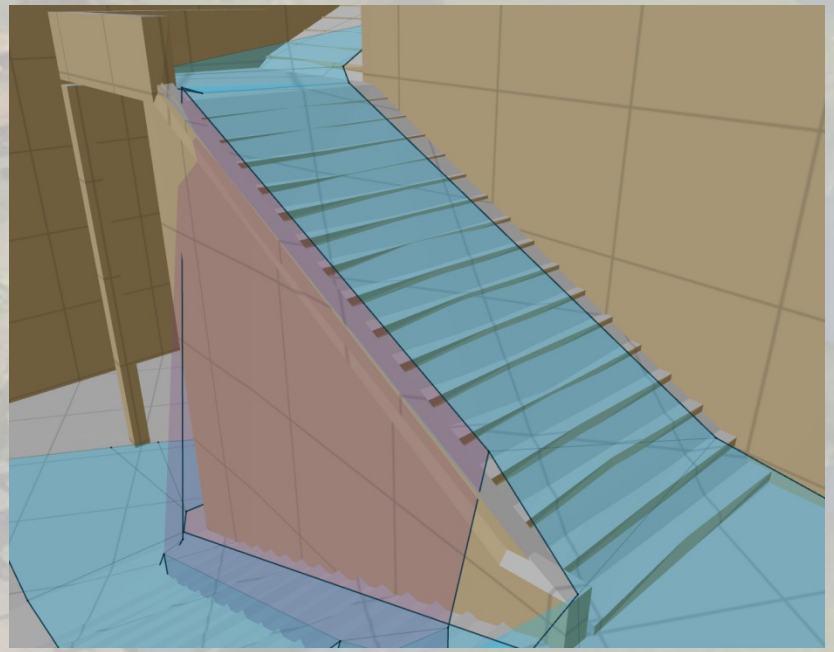


落下メッシュ生成

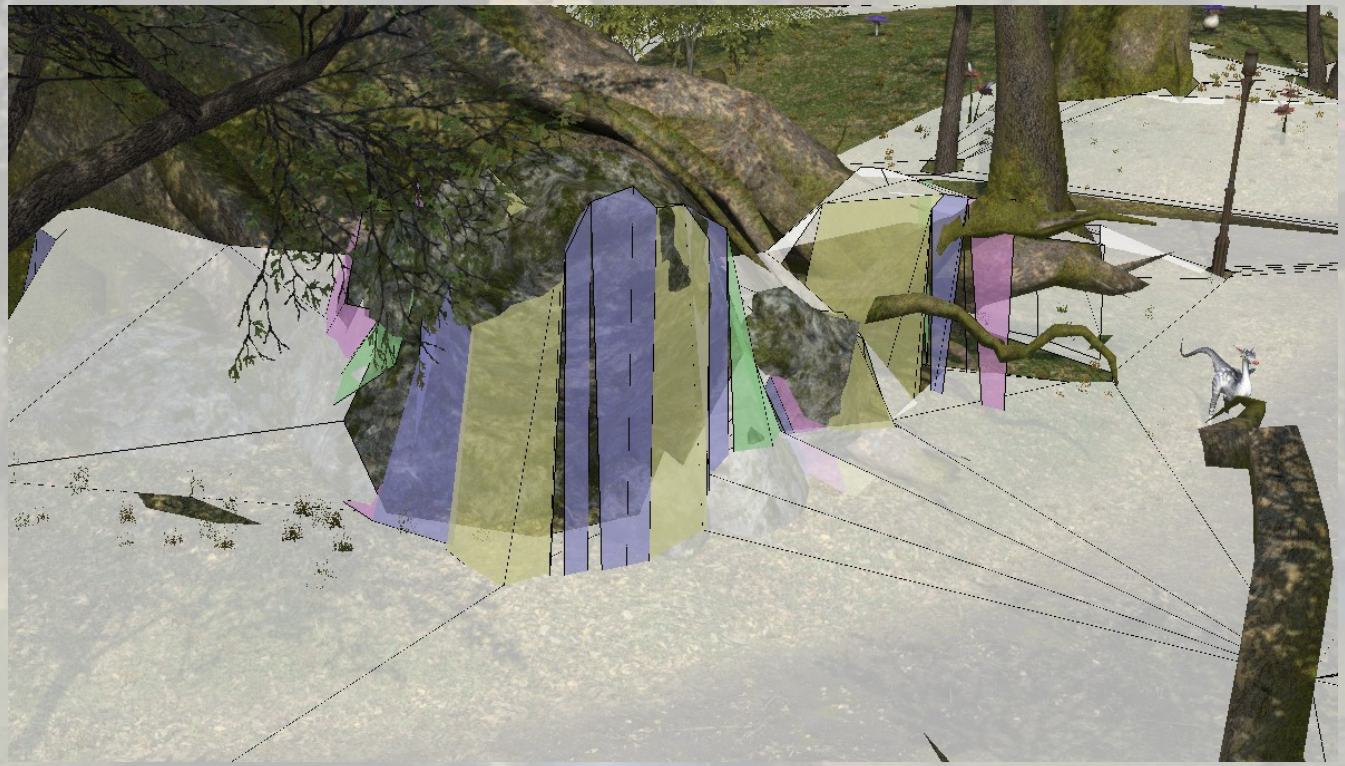
- 落下境界に対して
  - 落下用の輪郭を作る

落下メッシュ生成

- 落下メッシュを作る



FFXIVでのナビメッシュ



第5部
開発環境、運用

第5部 開発環境、運用

- データ制作環境・運用
- 自動生成の難しさ
- まとめ

データ制作環境＆運用

- 制作環境の要件を集めて、環境構築

SQUARE ENIX

データ制作環境＆運用

- 制作環境の要件を集めて、環境構築

グラフィックデザイナーの要望

SQUARE ENIX

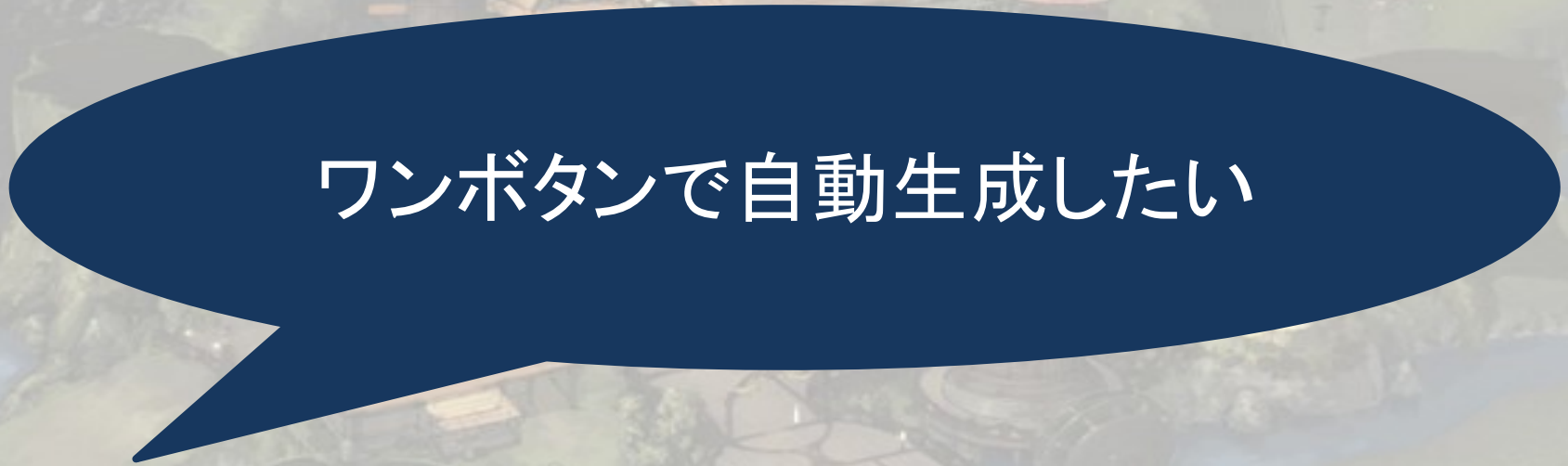
データ制作環境＆運用

- 制作環境の要件を集めて、環境構築

AIプログラマの要望

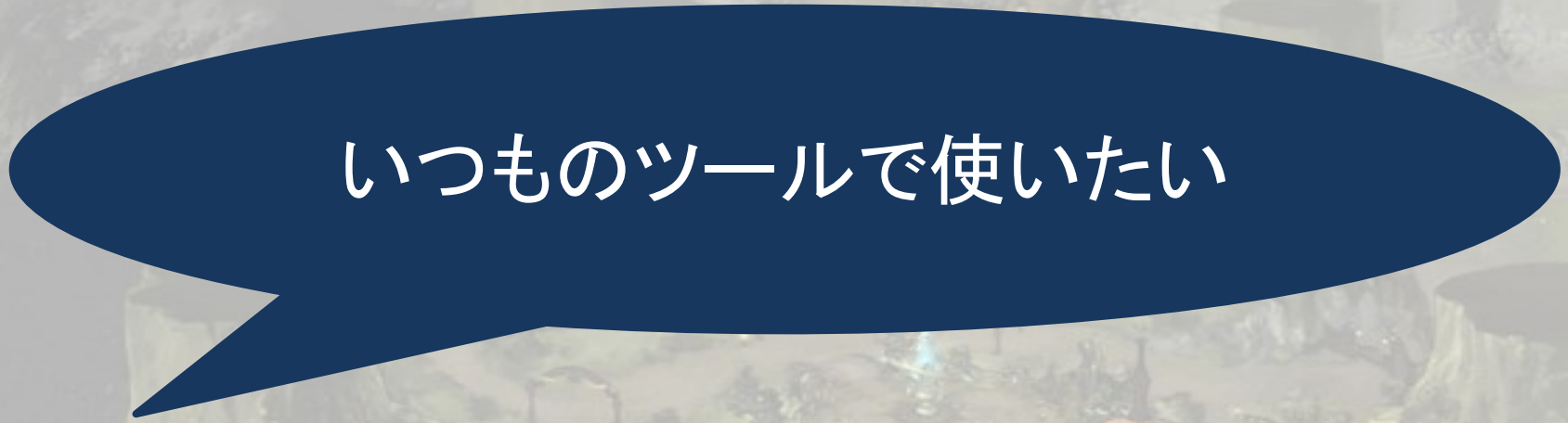
データ制作環境＆運用

- Q&A方式でまとめました



ボタンを押すだけ！

エディタでボタンを押す



いつものツールで使いたい

コンバータの
メンテナンスをしたくない

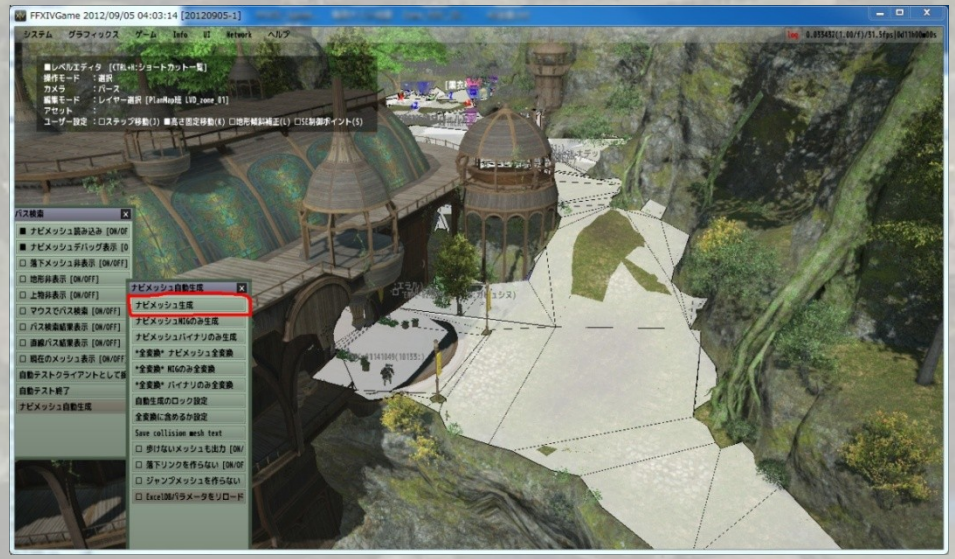
レベルエディタに組み込み

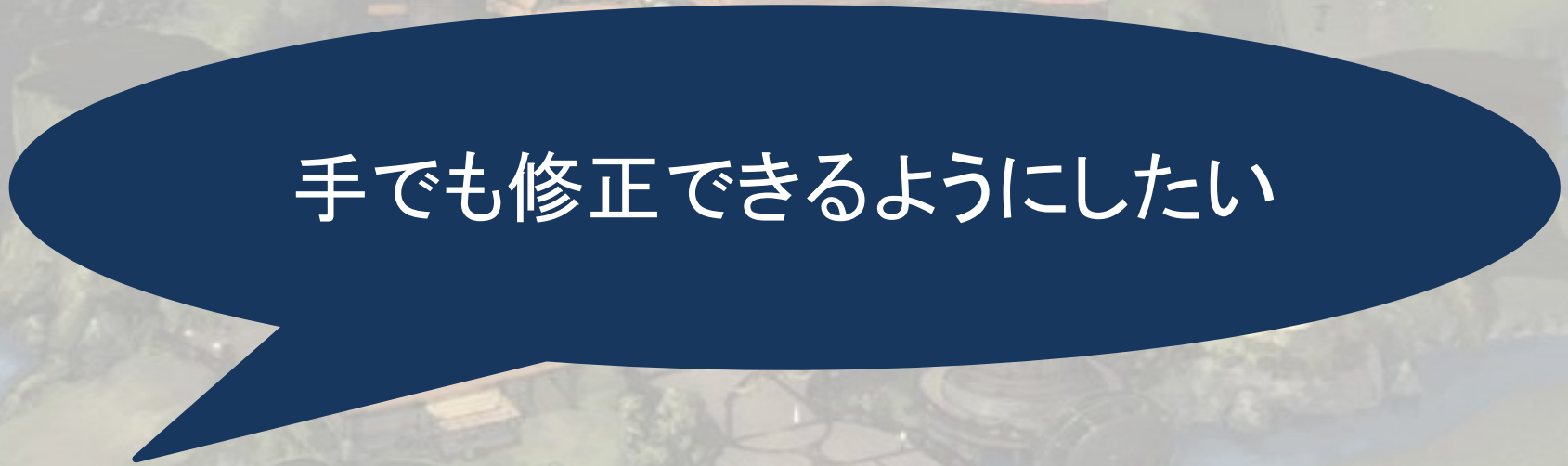
レベルエディタに組み込み

- いつもよく使っているなじみのあるツール

レベルエディタに組み込み

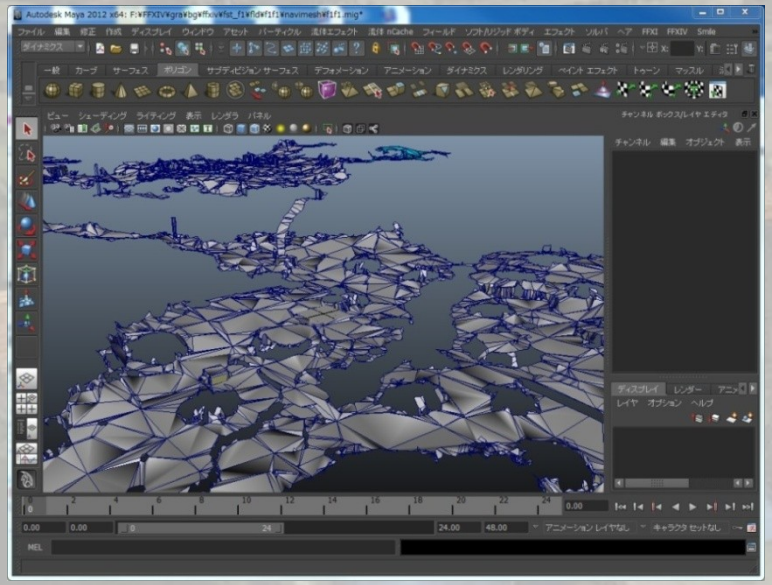
- ツール内のデータがすべて利用可能に



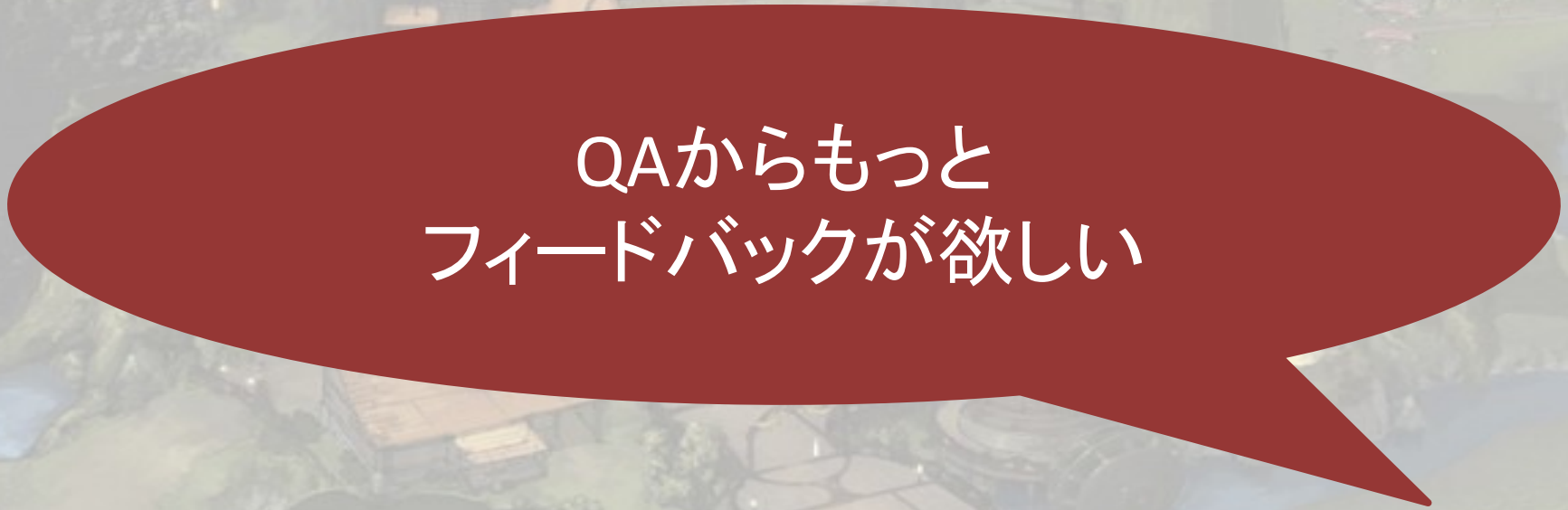


中間データはDCCツールで読める形

- DCCツールで手修正も可能



Autodesk® Maya®.



事前QAチェック

- AIの不具合は開発中には表面化しにくい
- QAチームと連携し、事前にチェックしてもらう
- 全マップチェックの前にプログラムを仕上げる
- チーム内にQA班があるとやりやすい

第5部 開発環境、運用

- データ制作環境・運用
- 自動生成の難しさ
- まとめ

ナビメッシュ自動生成の難しさ

- 自動生成は手間と時間が大幅に省ける！

ナビメッシュ自動生成の難しさ

- 自動生成は手間と時間が大幅に省ける！
- ただし、難しいところもあります。

部分的な修正が難しい

部分的な修正が難しい

- 問題になりやすかった箇所

部分的な修正が難しい

- 問題になりやすかった箇所
  - 狭い道

部分的な修正が難しい

- 問題になりやすかった箇所
  - 狭い道
  - 微妙に通れる/通れない道

部分的な修正が難しい

- 問題になりやすかった箇所
  - 狭い道
  - 微妙に通れる/通れない道
- ナビメッシュが適切に作られない

部分的な修正が難しい

- どうやって解決できるか？

部分的な修正が難しい

- どうやって解決できるか？
  - アルゴリズムを修正するか？

部分的な修正が難しい

- どうやって解決できるか？
  - アルゴリズムを修正するか？
  - 生成後のデータを直接編集するか？

部分的な修正が難しい

- どうやって解決できるか？
  - アルゴリズムを修正するか？
  - 生成後のデータを直接編集するか？
  - コリジョンデータを修正するか？

部分的な修正が難しい

- どうやって解決できるか？
  - アルゴリズムを修正するか？
  - 生成後のデータを直接編集するか？
  - コリジョンデータを修正するか？

FFXIVではコリジョン修正が多かった

第5部 開発環境、運用

- データ制作環境・運用
- 自動生成の難しさ
- まとめ

結果

- 比較的大きな森のマップ
  - 28000ポリゴン

結果

- 比較的大きな森のマップ
  - 28000ポリゴン
  - 自動生成時間 約1分30秒
  - データサイズ 約7MB

結果

- 比較的大きな森のマップ
  - 28000ポリゴン
  - 自動生成時間 約1分30秒
  - データサイズ 約7MB
- ダンジョンマップ
  - 1600ポリゴン

結果

- 比較的大きな森のマップ
  - 28000ポリゴン
  - 自動生成時間 約1分30秒
  - データサイズ 約7MB
- ダンジョンマップ
  - 1600ポリゴン
  - 自動生成時間 約10秒
  - データサイズ 約400KB

SQUARE ENIX

まとめ

- ナビメッシュ自動生成
  - Recast Navimeshを使ったナビメッシュ自動生成

まとめ

- ナビメッシュ自動生成
  - Recast Navimeshを使ったナビメッシュ自動生成
- 階層化隣接経路テーブル(HiNT)
  - サーバで動かしても問題のないぐらいの負荷
  - メモリもあまりかからない
  - 経路も自然になった

番外編

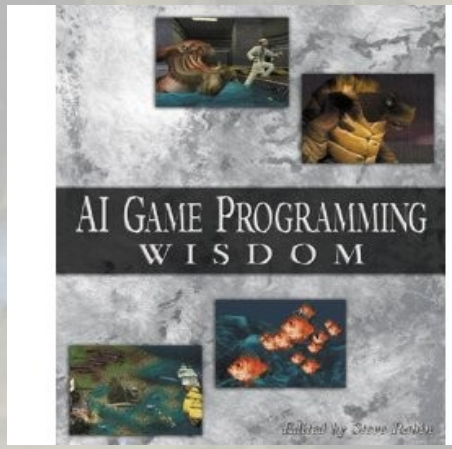
2Dマップ

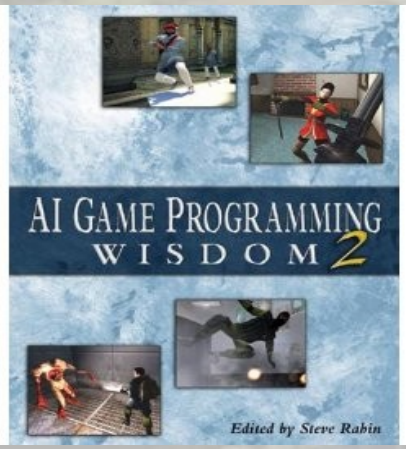
- 2Dワールドマップをナビメッシュを元に生成

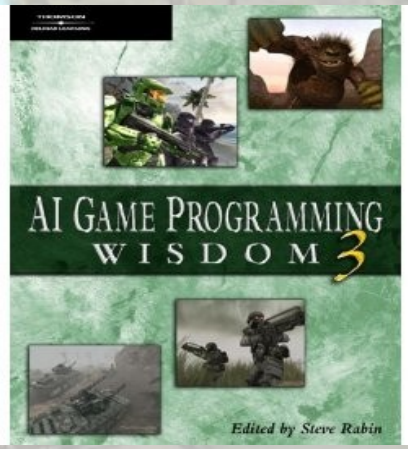
SQUARE ENIX

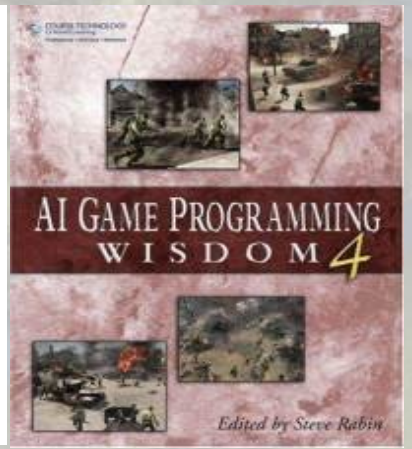
番外編 AI-Wisdom

- AI Game Programing Wisdom 5に掲載予定！
- 来年3月出版予定









ご静聴ありがとうございました