INVEN

igc in 성남

VOXEL Navigation

검은사막의 네비게이션 시스템

INVEN GAME CONFERENCE IN SEONGNAM

프로그래머 민경인 in PEARL ABYSS

History

PEARL ABYSS

인턴 입사

2010.12

검은사막

2011.02

컴퓨터 공학과 졸업

HANYANG UNIVERSITY 한양 1939

2016.07

Battle System


Auto Navigation


Remaining Workers : 0 / 0
Weather System
Exploration Nodes : 92

What is…

15만
이상

4개
이상

368.64
㎢

What is…

15만
이상

활동중인 Agent 개체수

368.64
㎢

4개
이상

What is…

15만 이상

활동중인 Agent 개체수

4개 이상

368.64 ㎢

네비게이션 복셀로 관리되는 영역

What is…

15만 이상

활동중인 Agent 개체수

368.64 ㎢

네비게이션 복셀로 관리되는 영역

4개 이상

PC, XBOX, IOS, ANDROID
플랫폼에서 구동중


VOXEL based
NAVIGATION
Pos : [9785,-6377,71818] => [195,-128,1436]
Size : [1,1,1]
Type : ground/NONE
NearNodeBitFlag :
Type : ground
NearNodeBitFlag :
110111111
111111111
111111111

igc in 성남

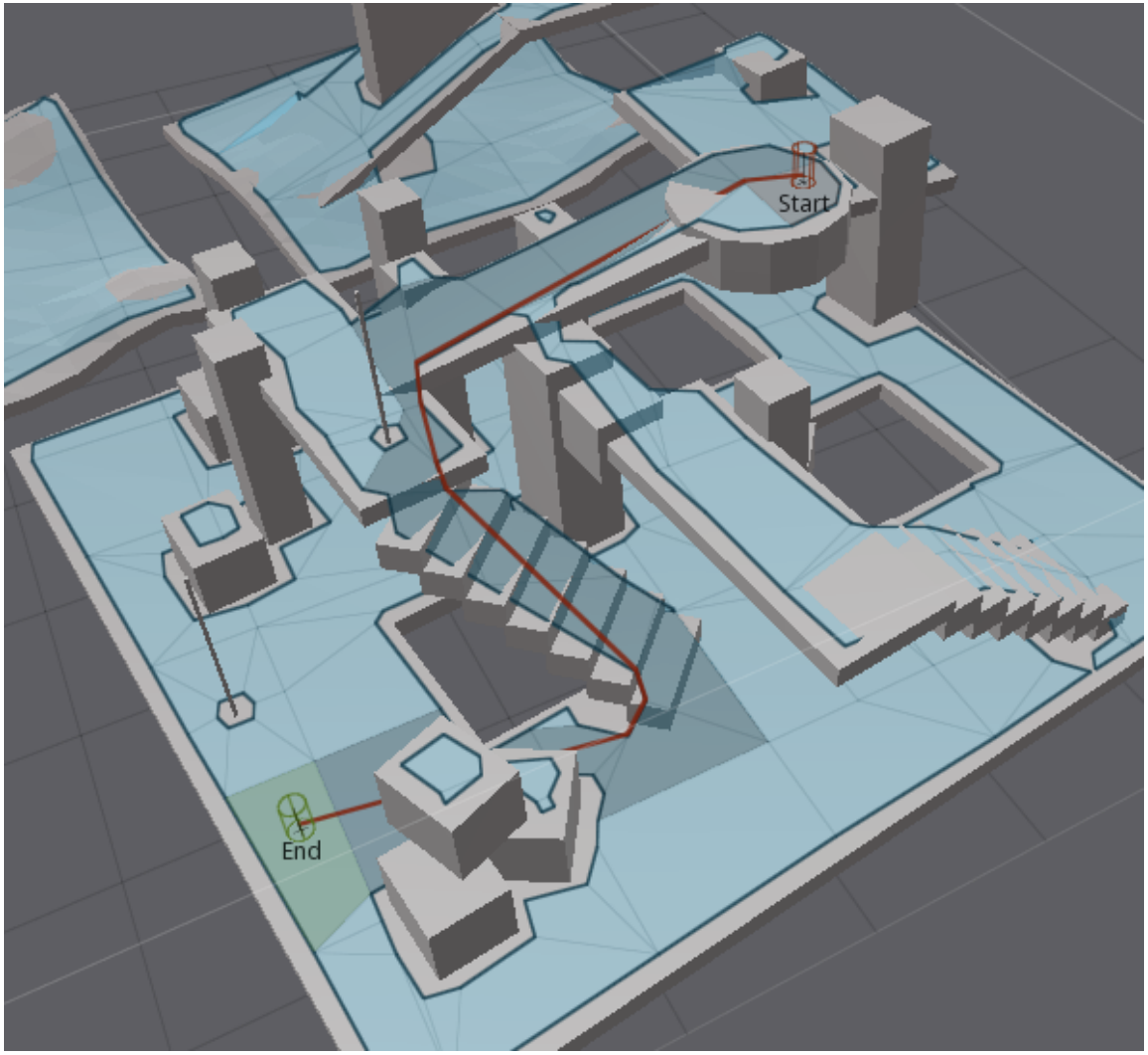
INDEX

1

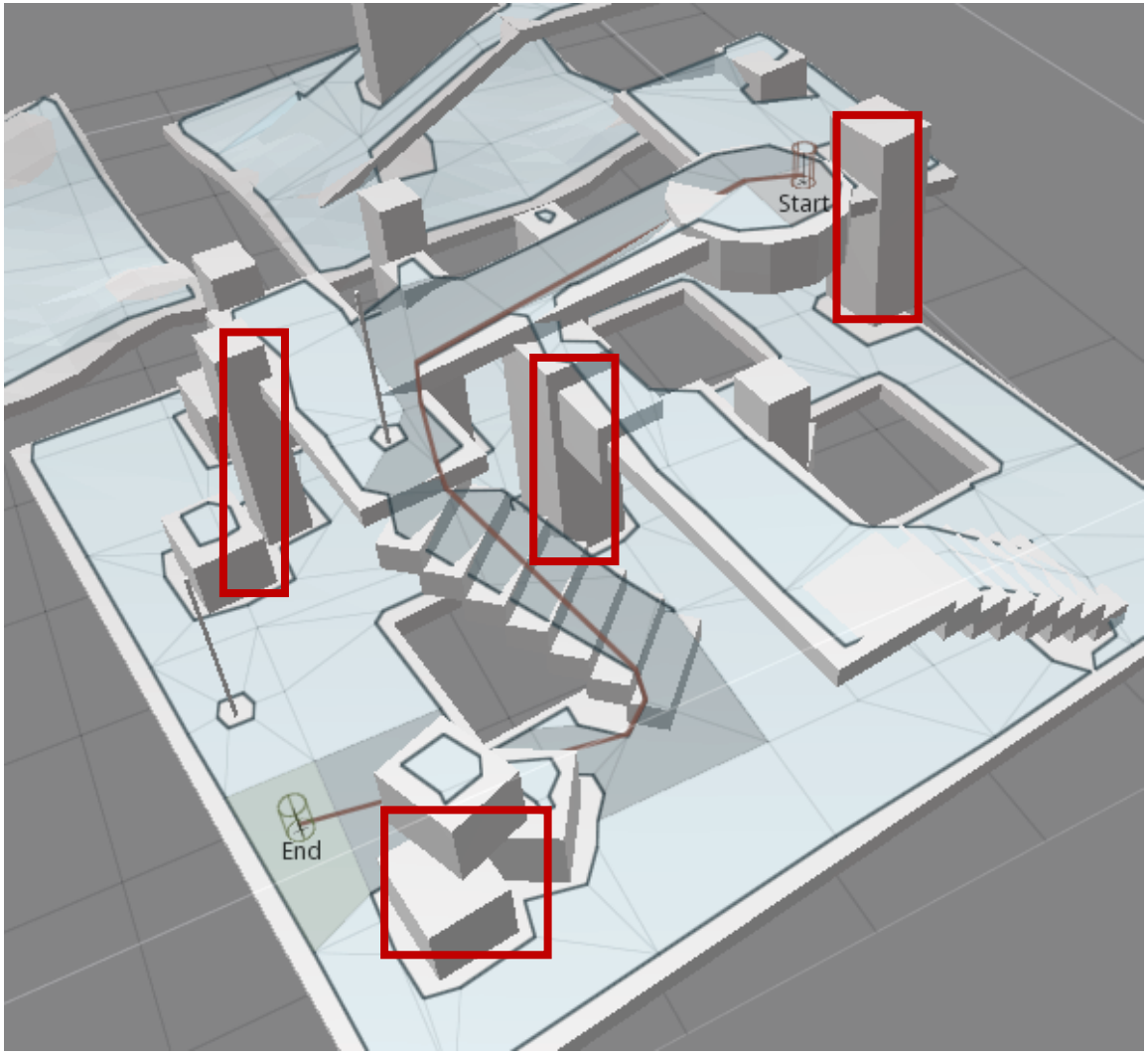
네비게이션 메쉬와 A* 정리

igc in 성남

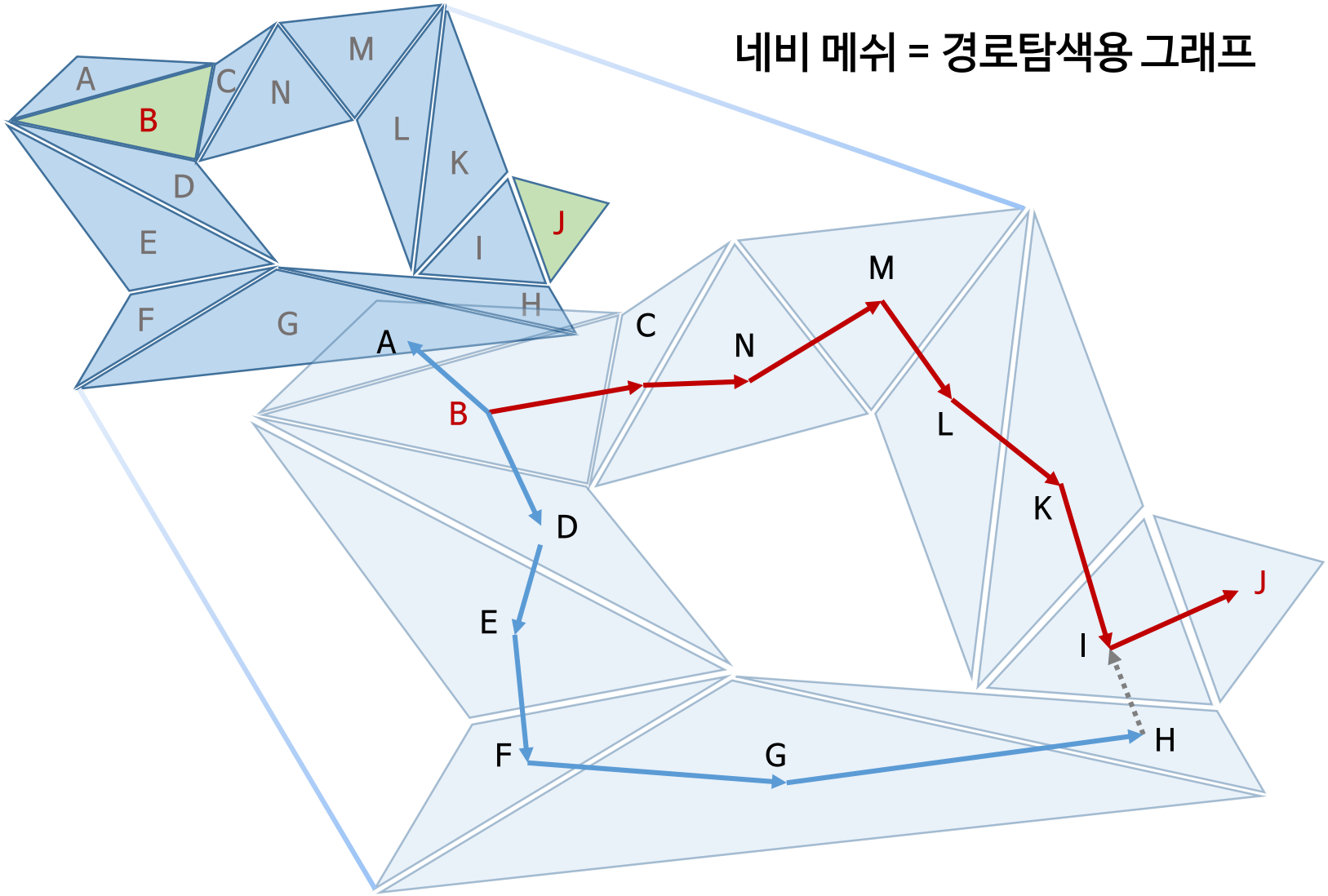
네비게이션 메쉬



네비게이션 메쉬

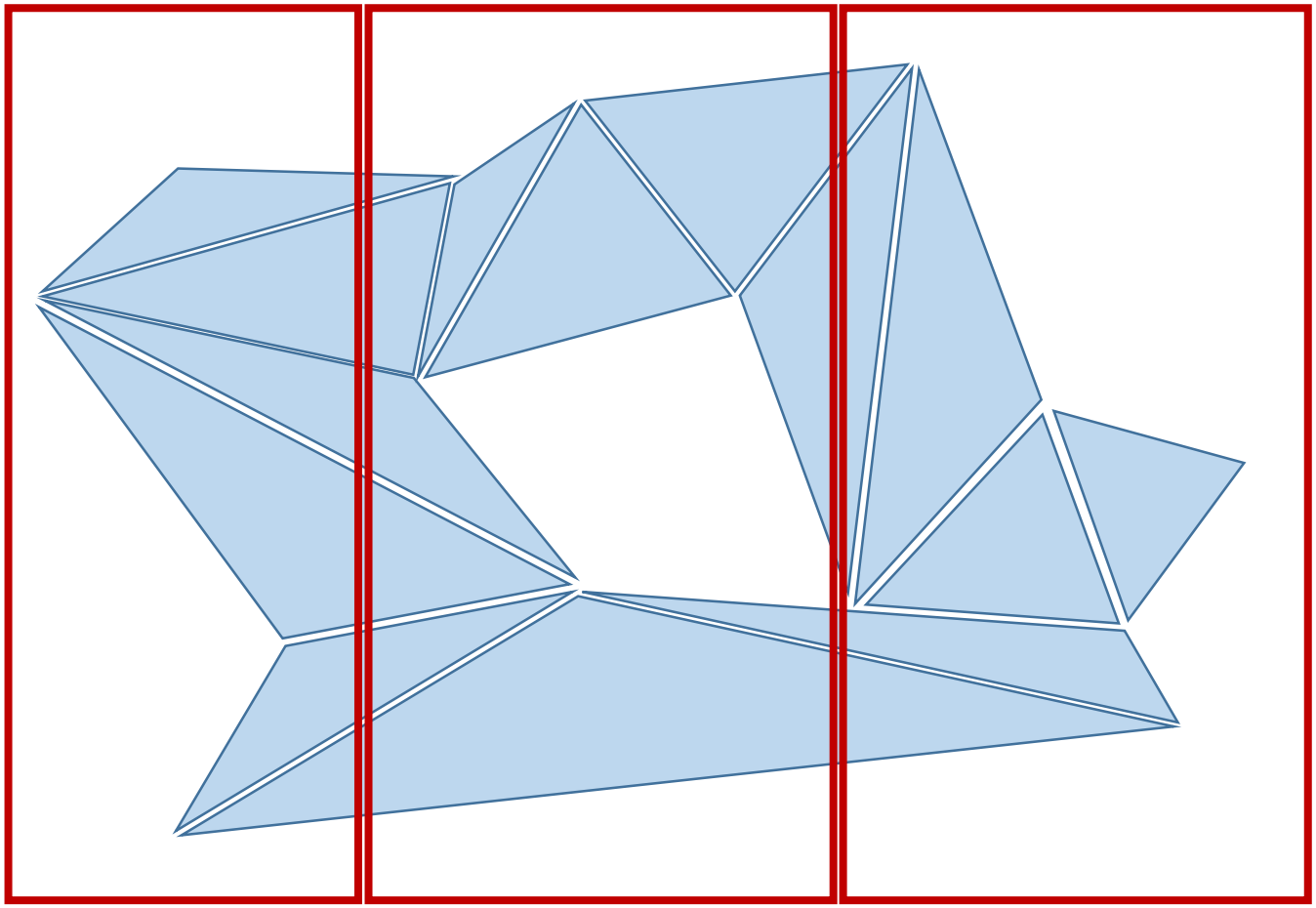


네비게이션 메쉬



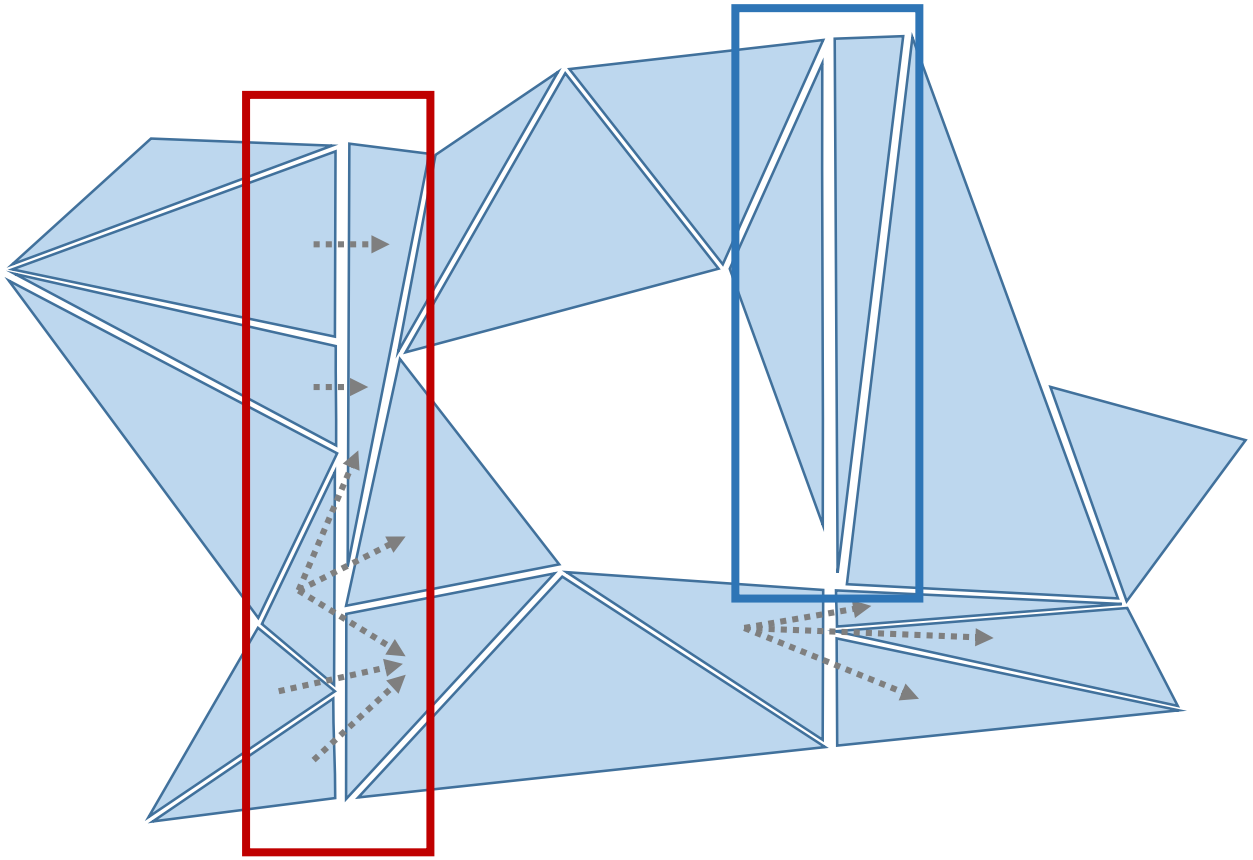
네비게이션 메쉬

네비 메쉬를 하나의 파일로 유지하지 못할 경우

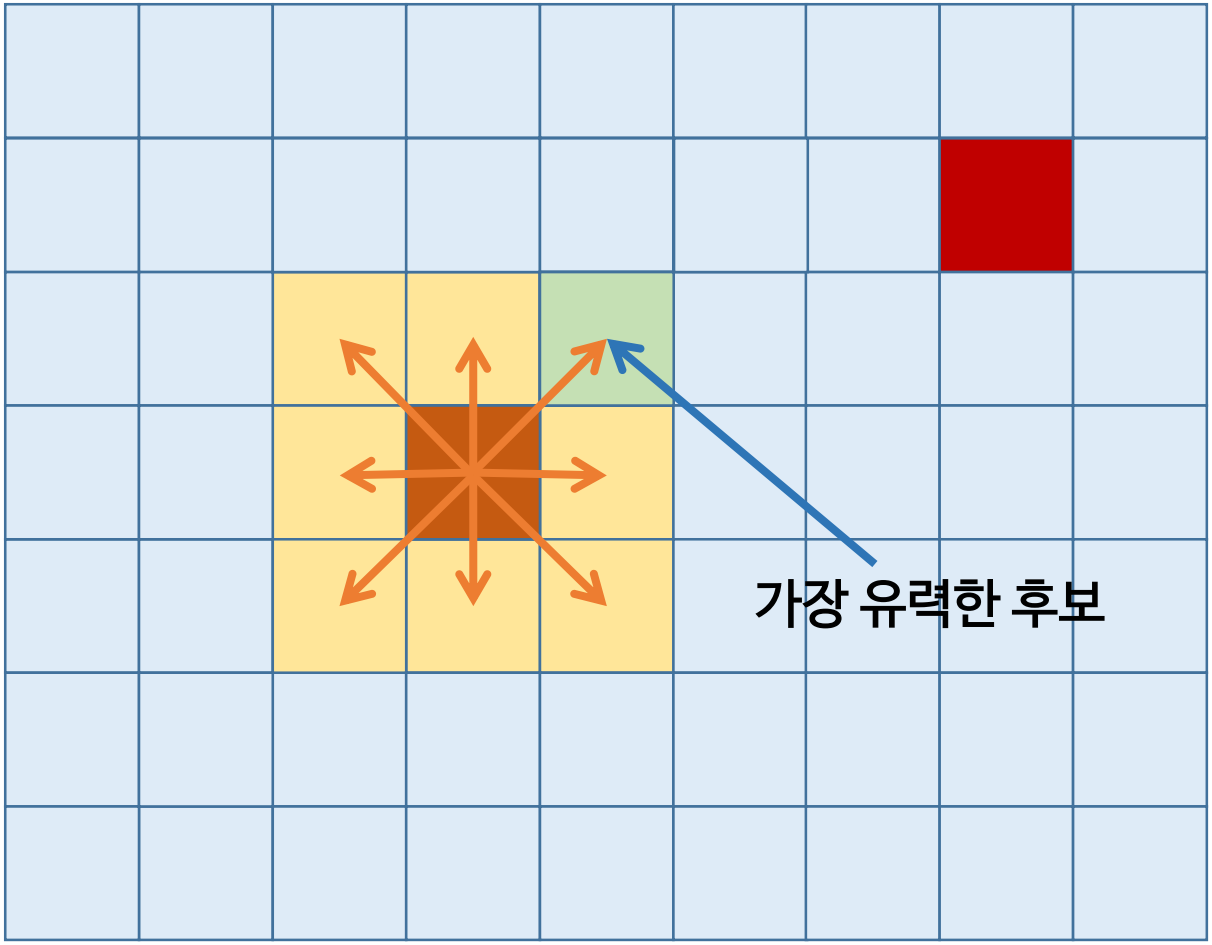


네비게이션 메쉬

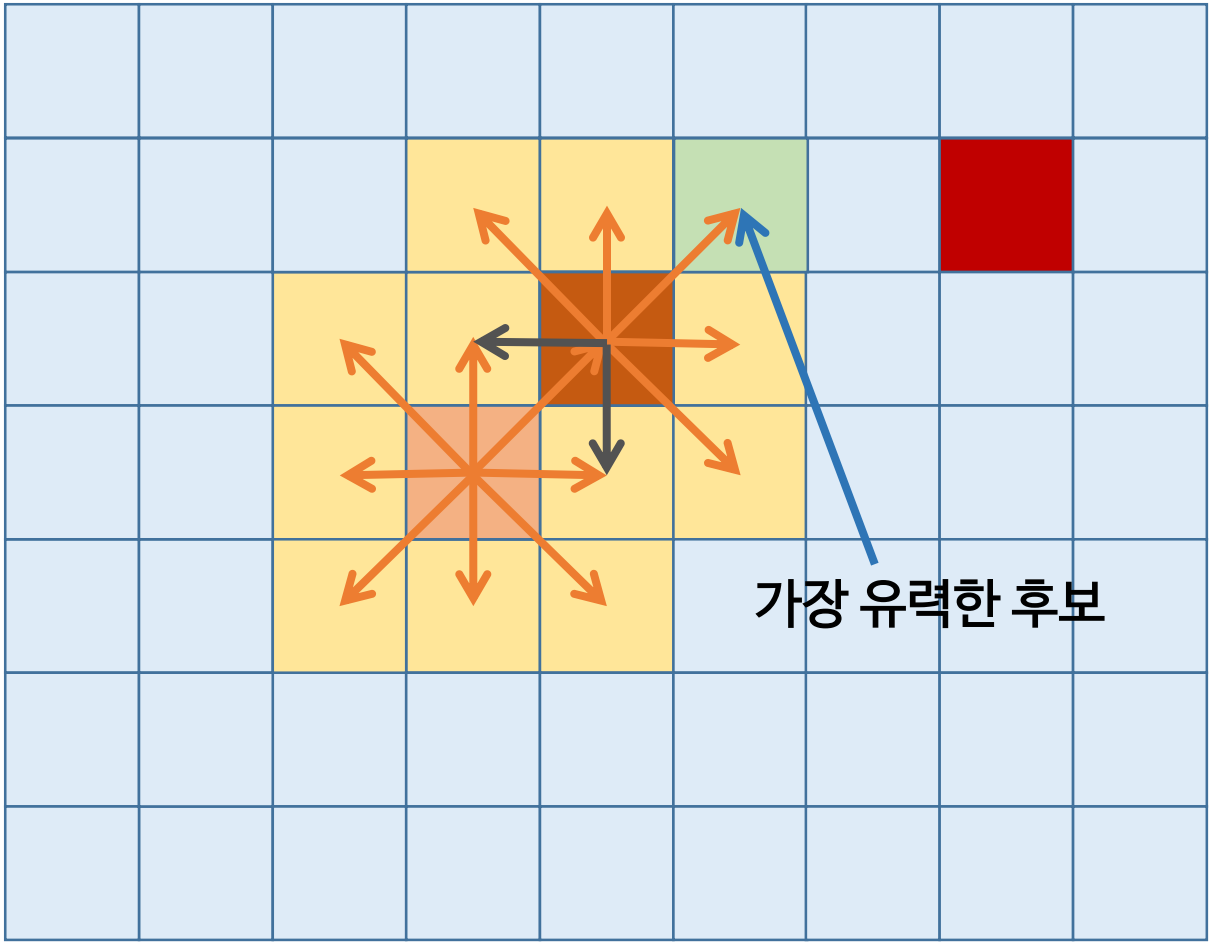
재연결 작업의 복잡성은 **데이터**와 **메쉬 추출방법**에 따라 다릅니다



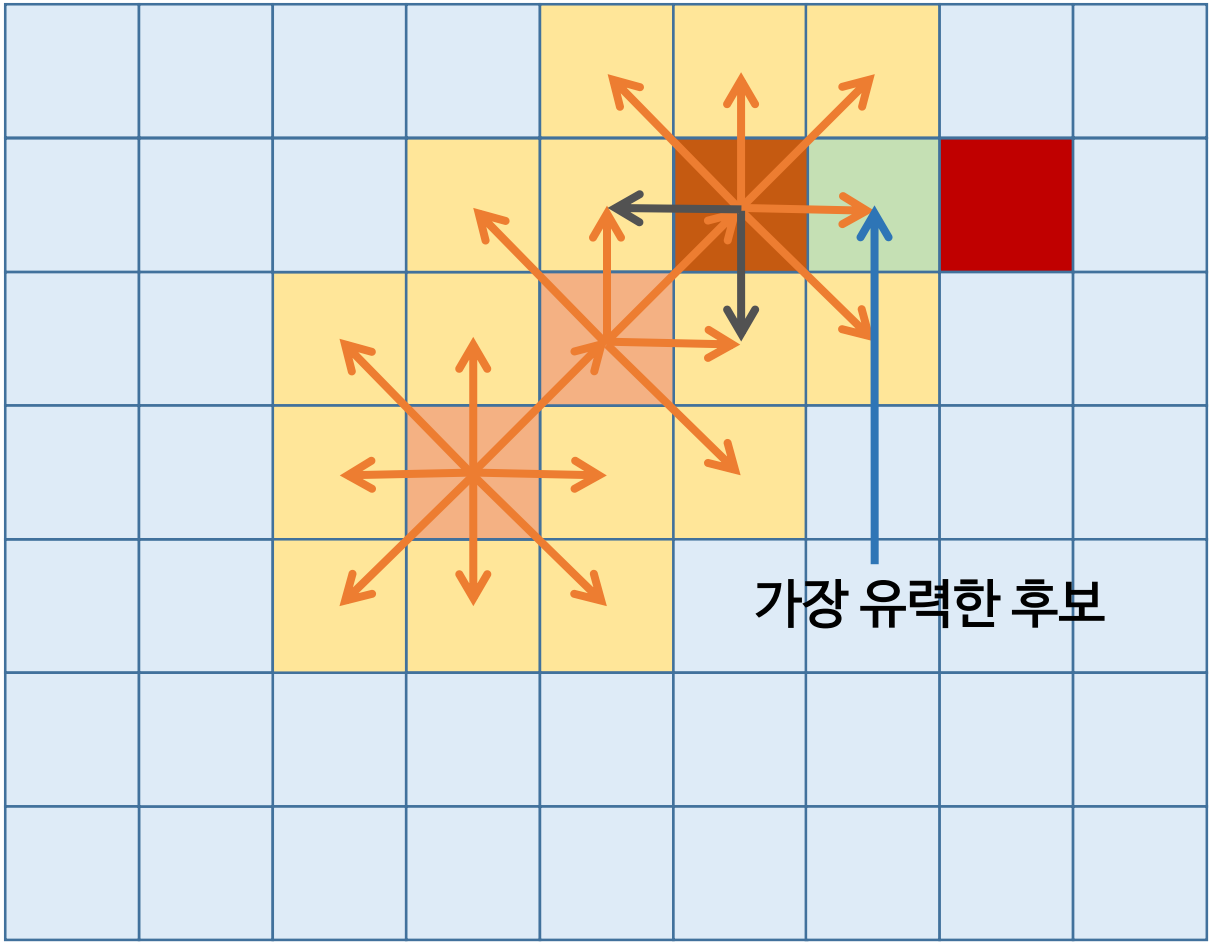
A* 기본 정리



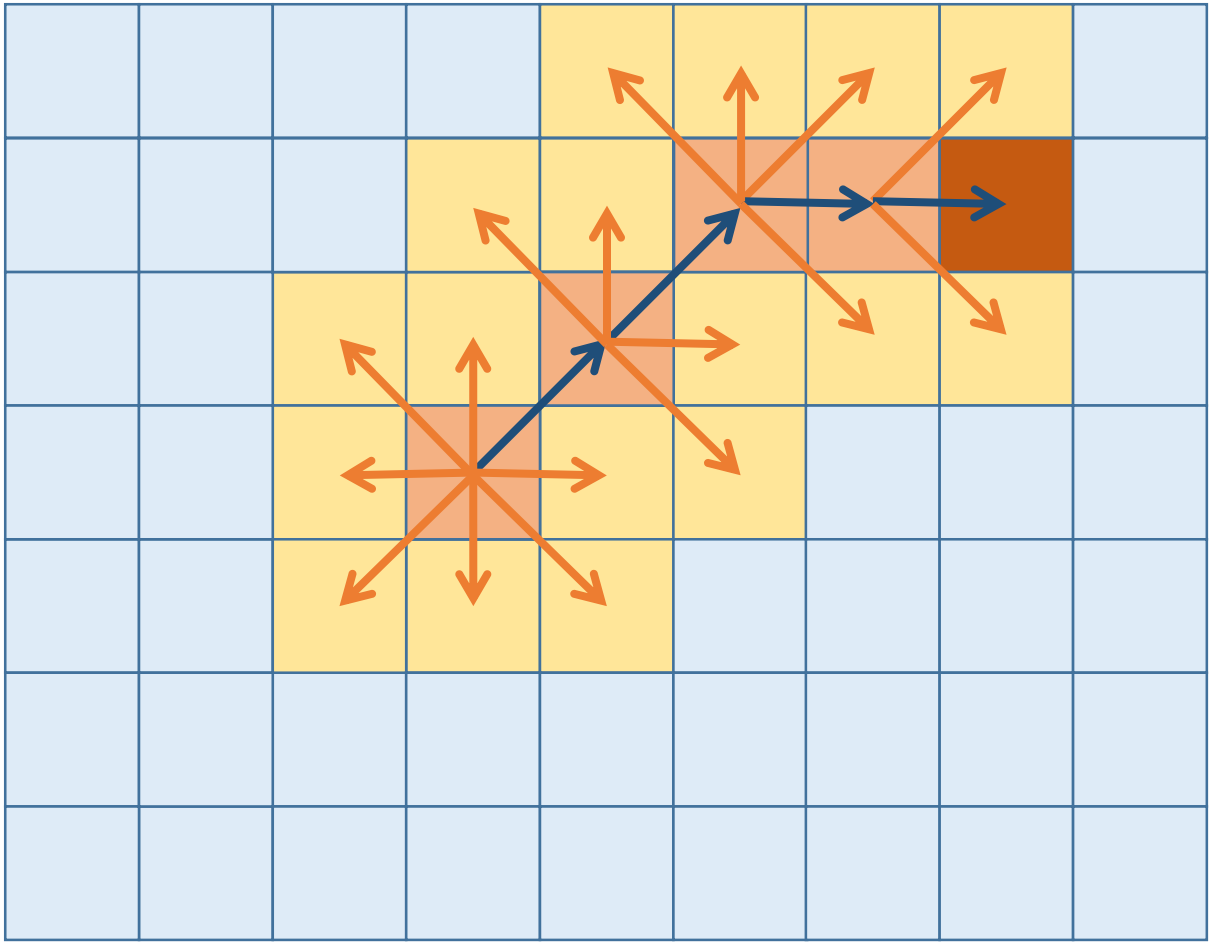
A* 기본 정리



A* 기본 정리



A* 기본 정리



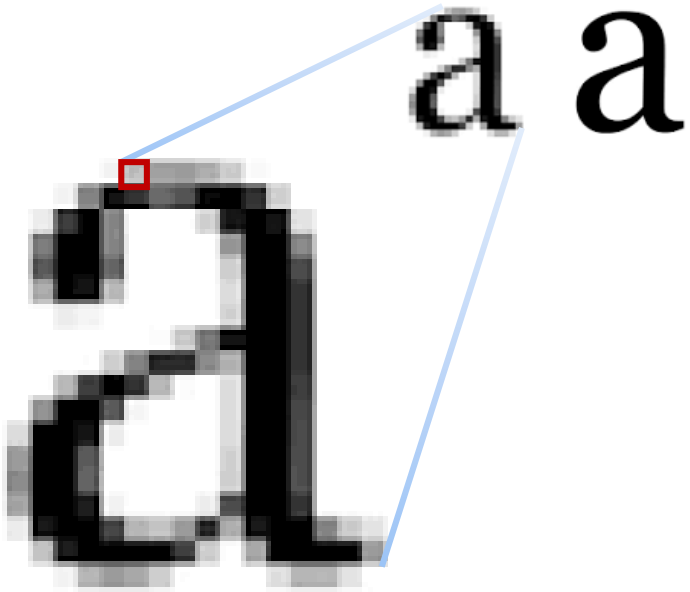
2

VOXEL NAVIGATION 개요

igc in 성남

VOXEL이란?

PIXEL

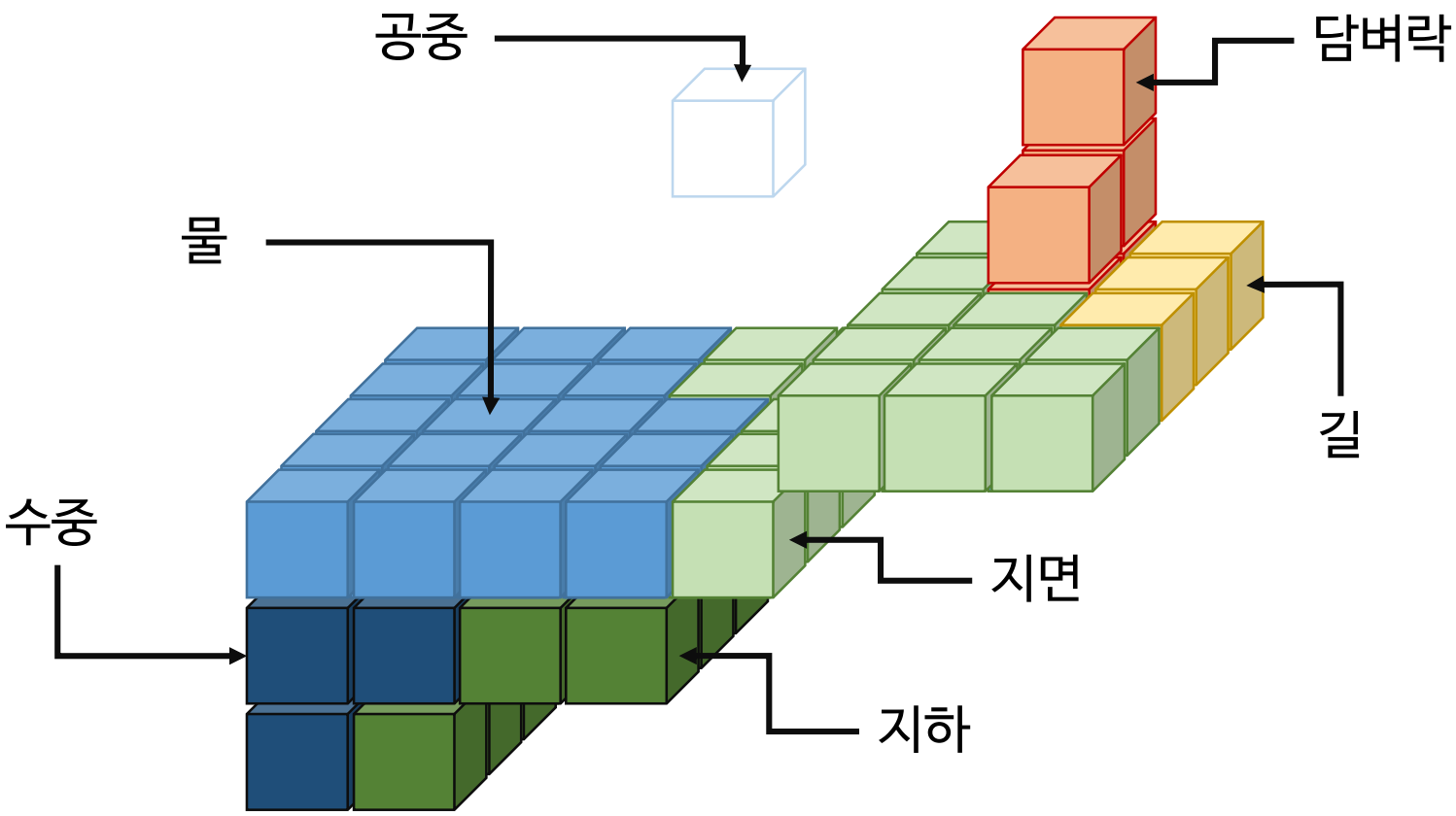


VOXEL이란?

PIXEL

VOXEL

VOXEL NAVIGATION



DEGINE OF VOXEL NAVIGATION

Agent Layer

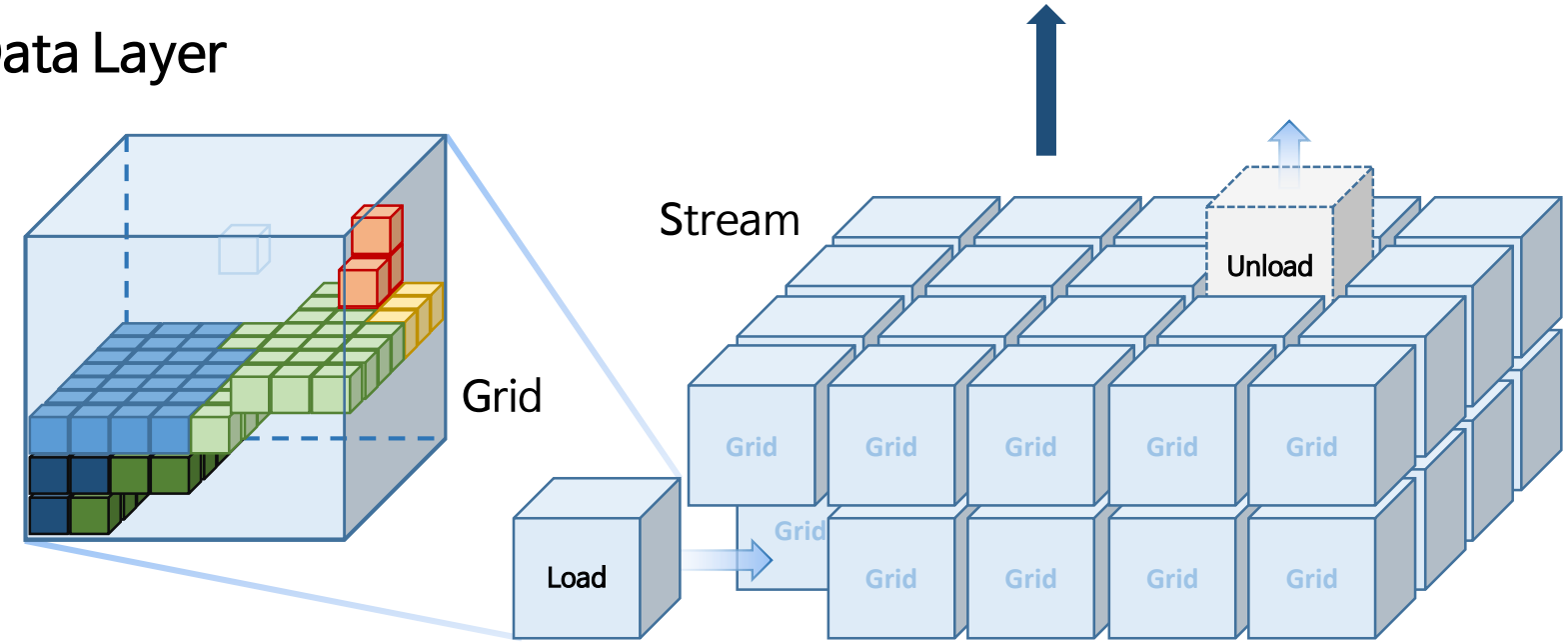
Agent

Obstacle

Logic Layer

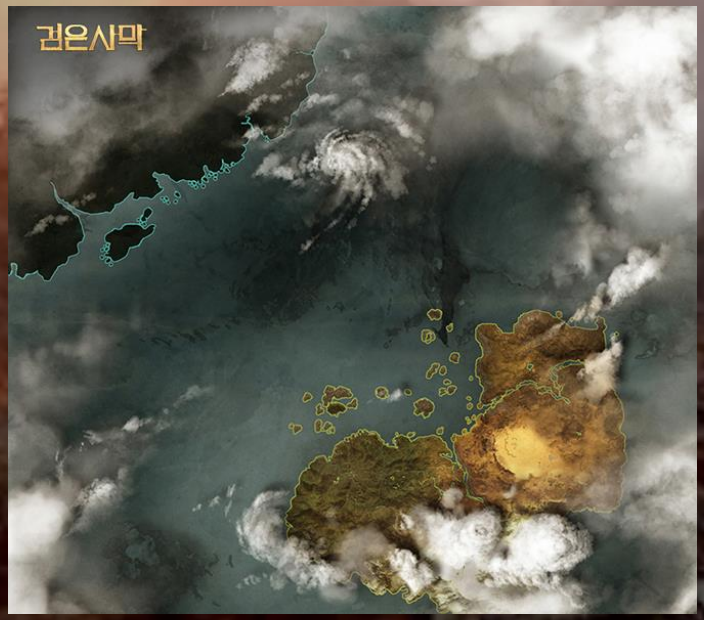
Path Finder
Raycast
Path Post Processor

Data Layer



넓은 월드를 위한

빠른 동적 로딩



어떤 복잡한 지형이라도

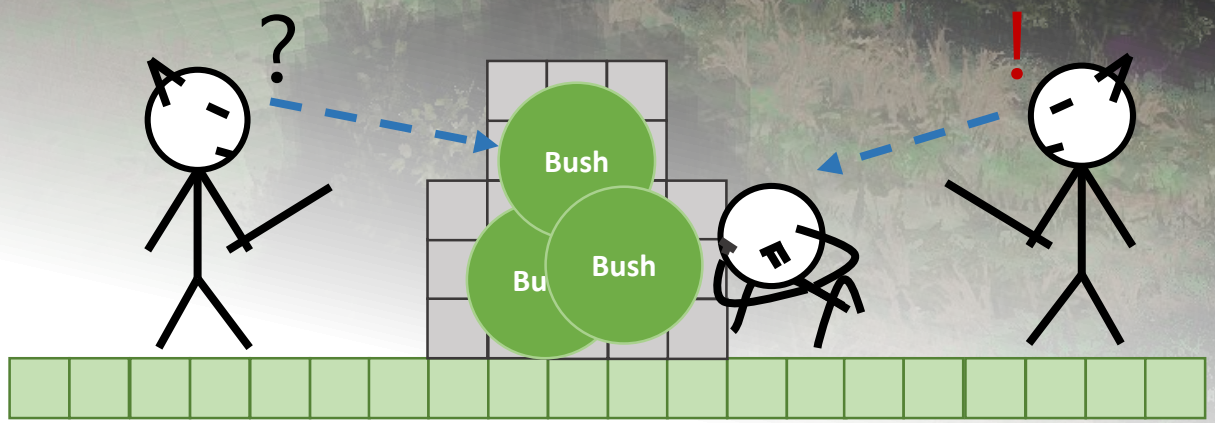
수작업 배제


Voxel based
RAYCAST


Voxel based
RAYCAST

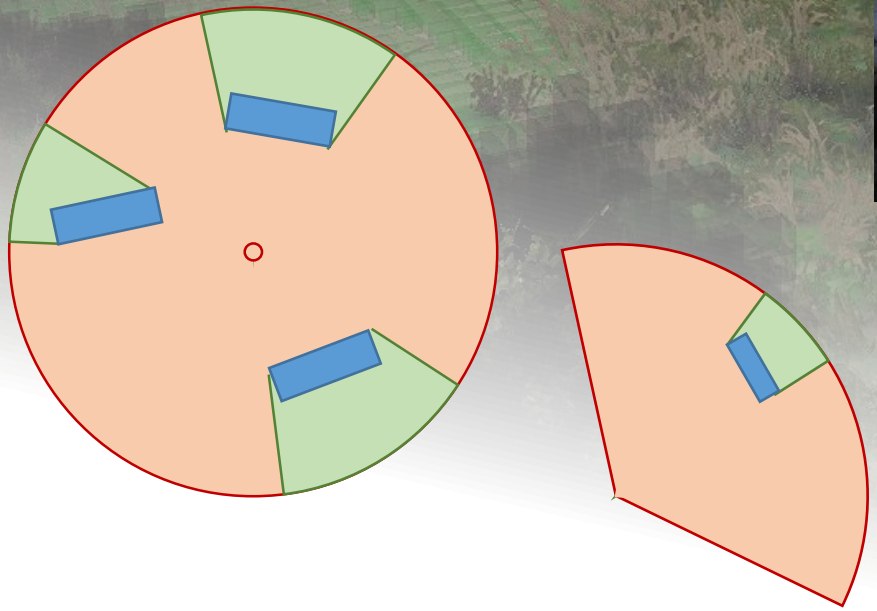
Voxel based RAYCAST

가시성 판단 AI 구현



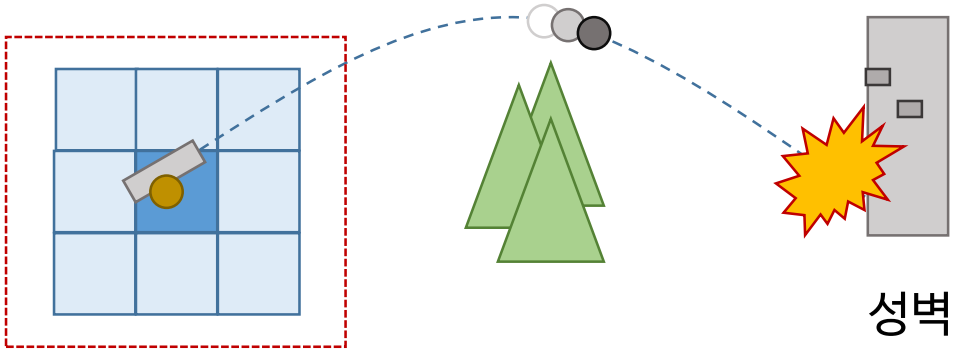
Voxel based RAYCAST

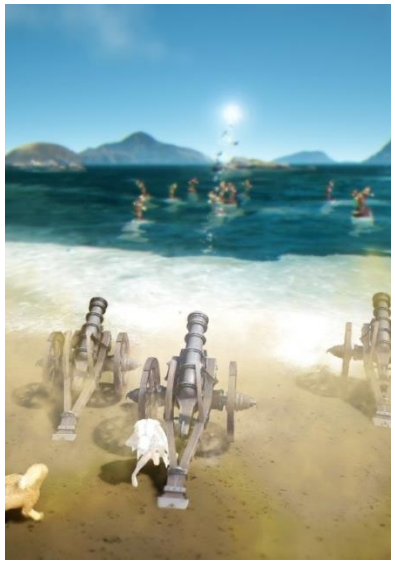
공격범위 차폐 구현



Navigation Raycast

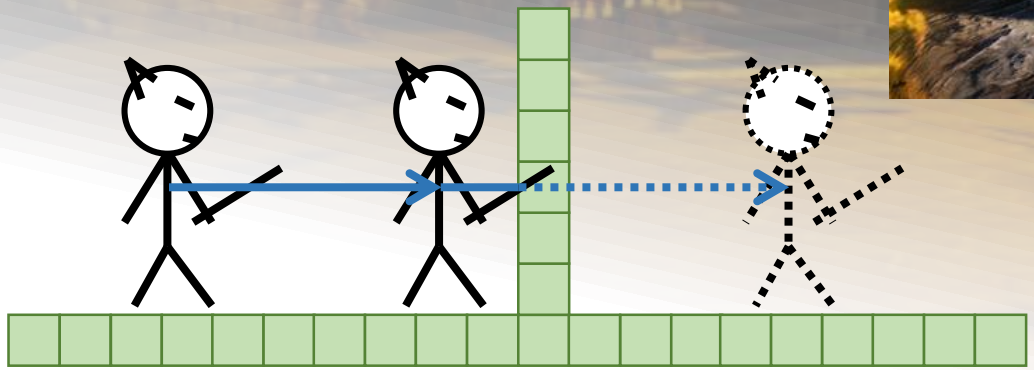
서버 기반 포격 시뮬레이션





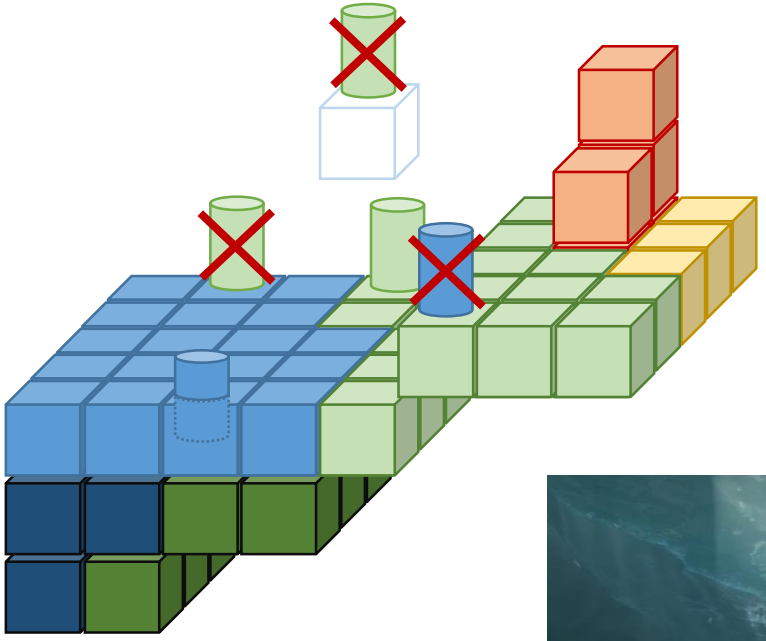
Voxel based RAYCAST

지형지물 관통 검증



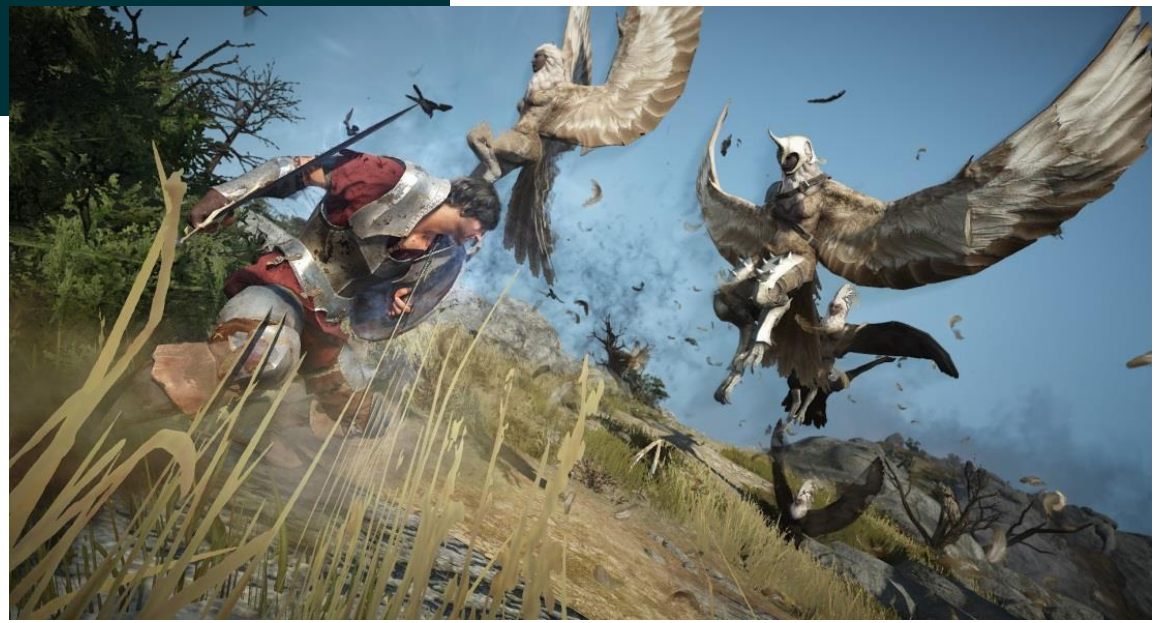
이동 관련 핵툴 체크

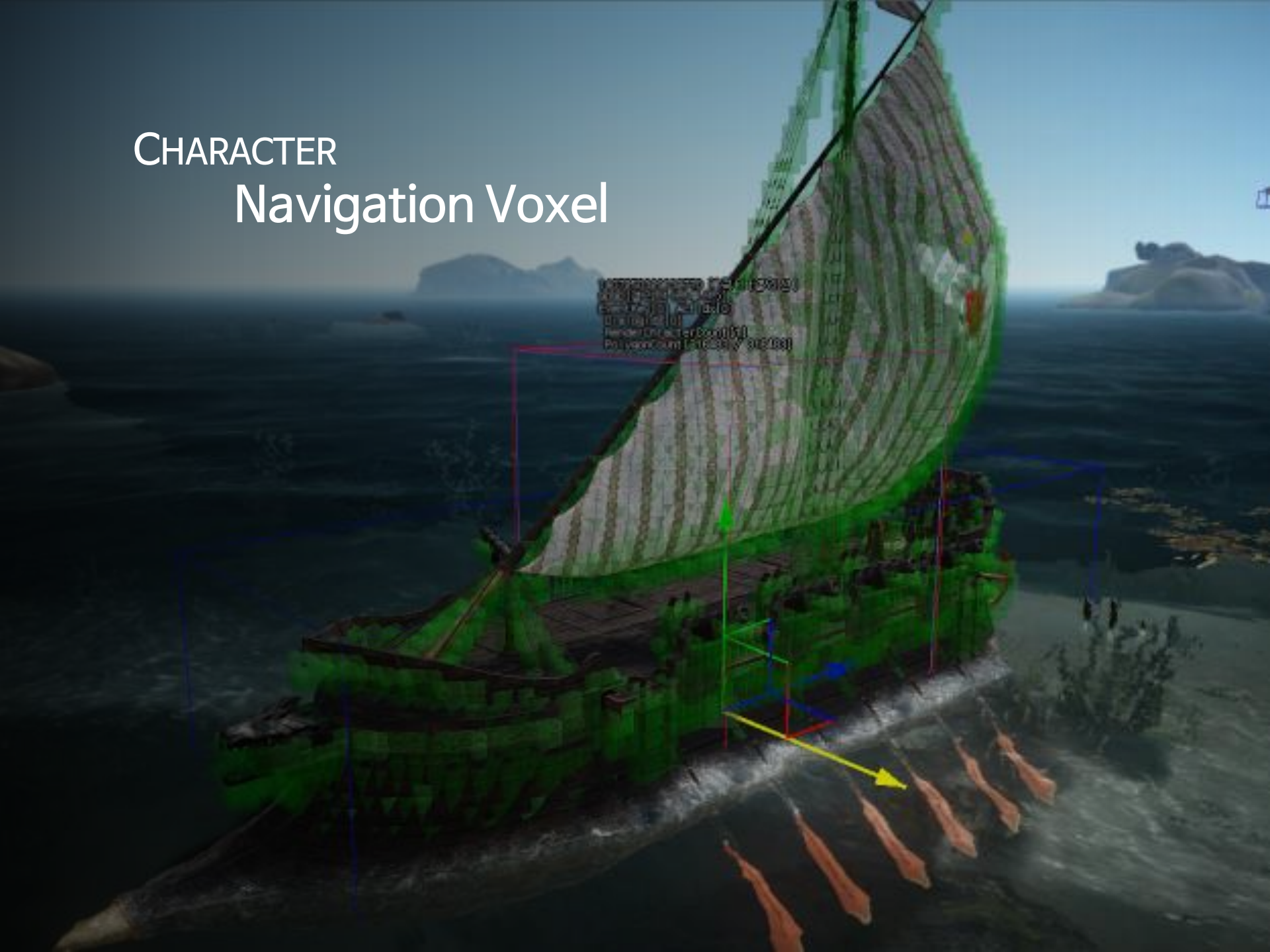
복셀의 속성과 캐릭터 동작의 관계를 이용하면 비정상 플레이 감지 가능



동작 종류	허용 지역
점프	공중 / 지면
달리기	지면
수영	수면 / 수중

다양한 이동 타입의 AI 구현




CHARACTER
Navigation Voxel

CHARACTER
Navigation Voxel

3

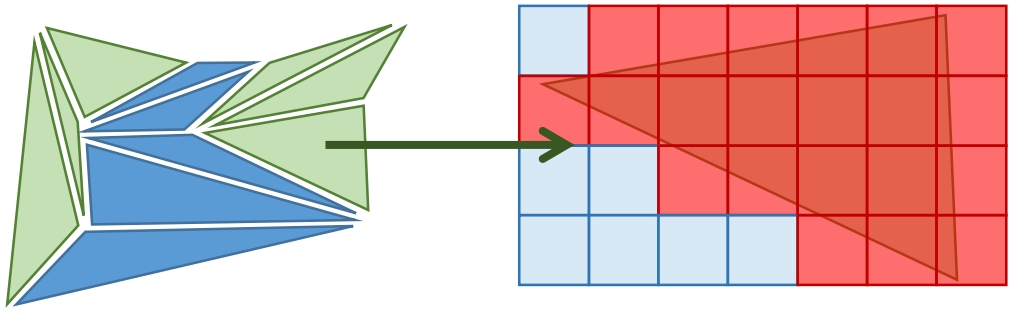
폴리곤 to 복셀 샘플링 기법

igc in 성남

복셀 데이터 추출 (2/4)

검은사막 배치데이터를 이용하는 전용 샘플러를 구현

샘플링할 대상(배치 데이터)의 모든 폴리곤을 순회하며 샘플링



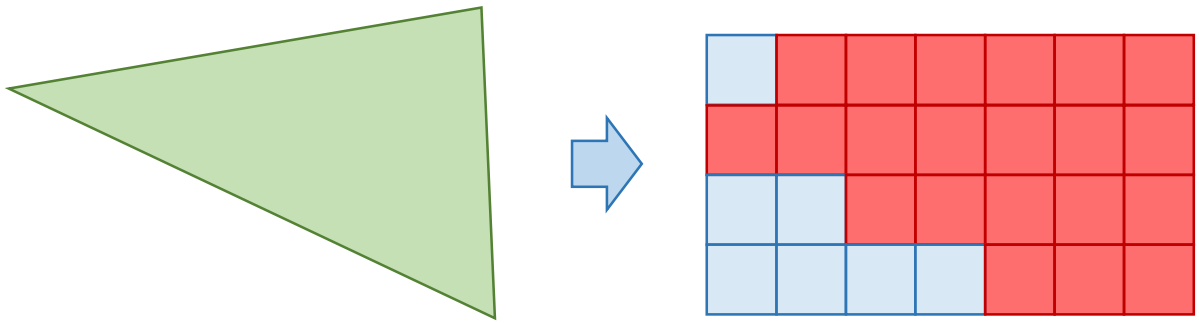
샘플러가 인식하는 데이터 종류가 다양해짐에따라 자동 속성 판단가능
(파일명, 툴상에서 지정하는 옵션들, 충돌로 뽑히지 않는 메쉬들…)

복셀 데이터 추출 (3/4)

Triangle Sampling to Voxel

나이퀴스트-섀넌 표본화 정리

"만약 신호가 대역제한(bandlimited)신호이고,
표본화 주파수가 신호의 대역의 **두 배 이상**이라면
표본으로부터 연속 시간 기저 대역 신호를 **완전히 재구성**할 수 있다."

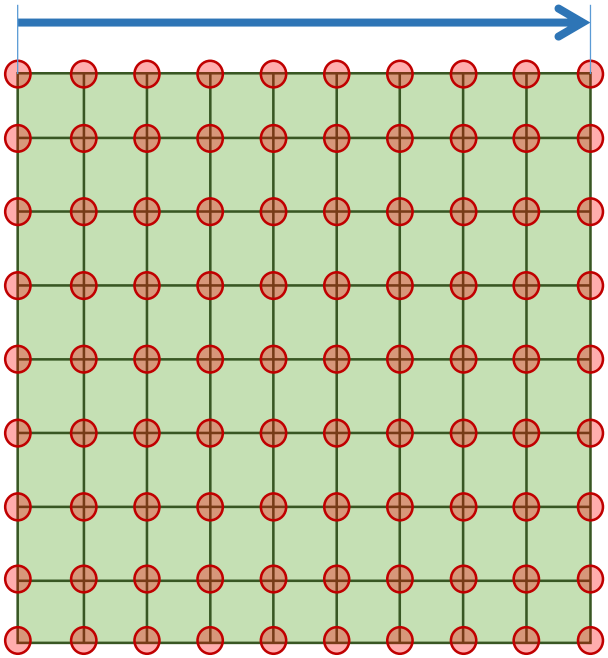


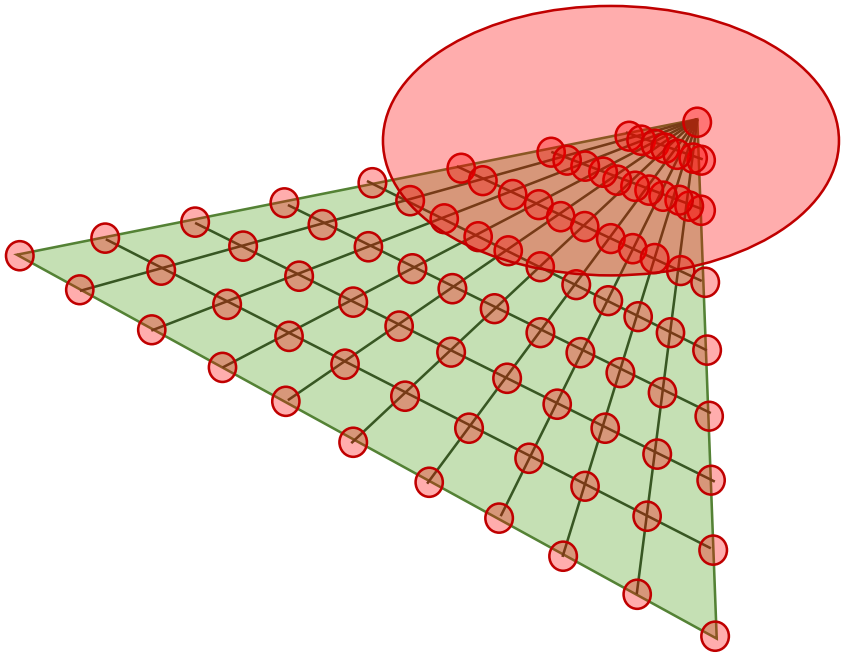
Analog Data
(Polygon)

Digital Data
(Voxel)

복셀 데이터 추출 (4/4)

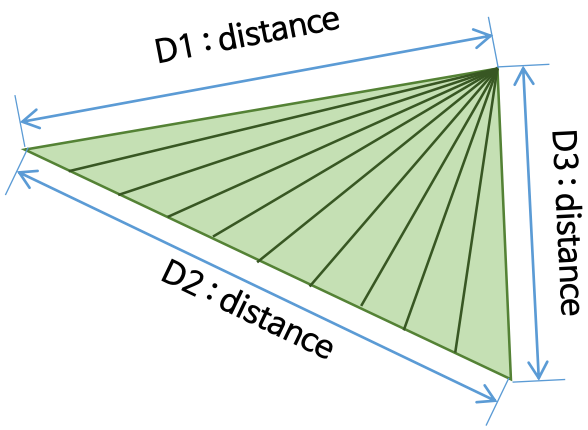
Triangle Sampling to Voxel

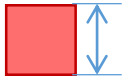




복셀 데이터 추출 (4/4)

Triangle Sampling to Voxel





V : voxel size

Sampling Count : Distance / (V * 2.0)

Sampling Step (D2 〉D1 〉D3)
- D1, D2 에 대해 D2 / (V * 2.0) 갯수로 분할

- 분할한 P0`, P0`` 에 대해 다시 샘플링결과 누적
- PN`, PN`` 까지 반복

샘플링 정확도를 보장하려면
꼭지점, 각 변의 점들은 반드시 포함

4

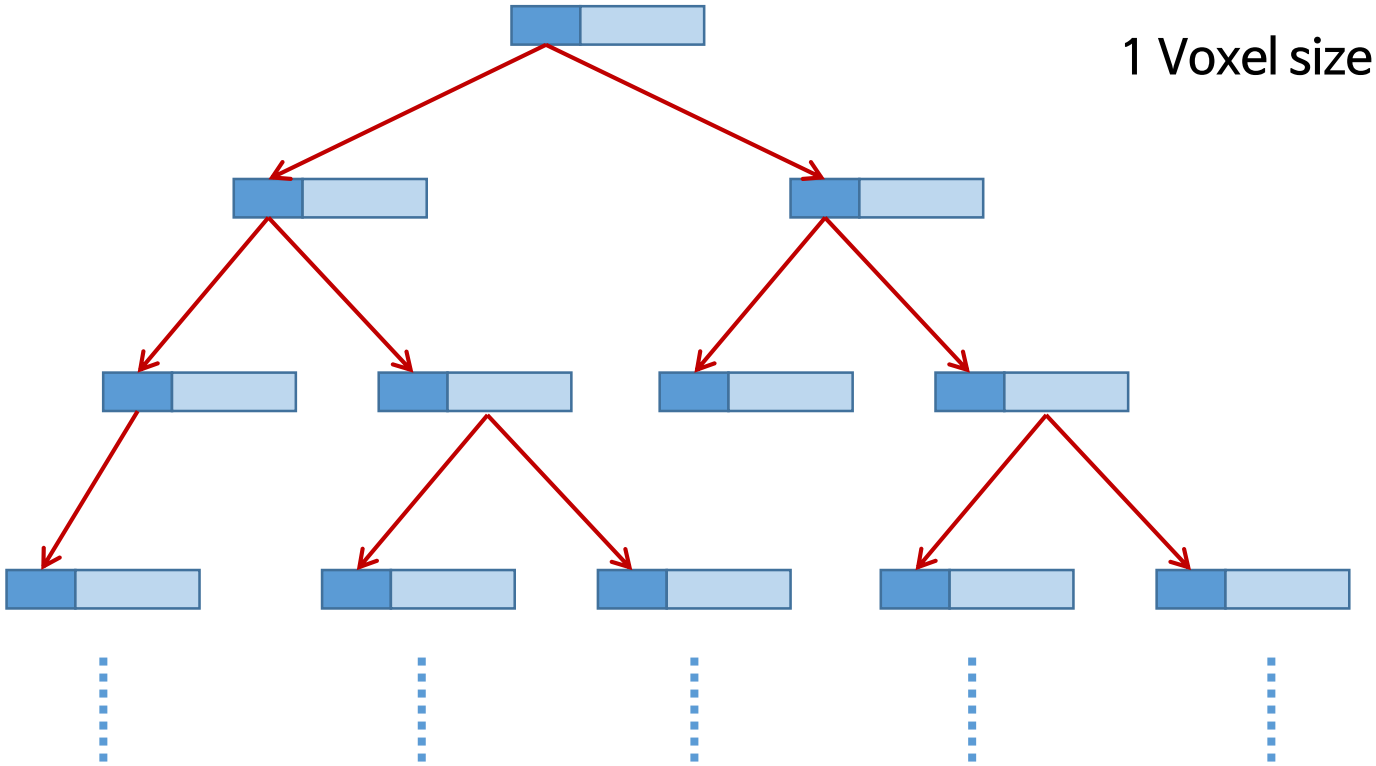
고정데이터 Map 데이터 용량 절약

igc in 성남

Grid 메모리 개선

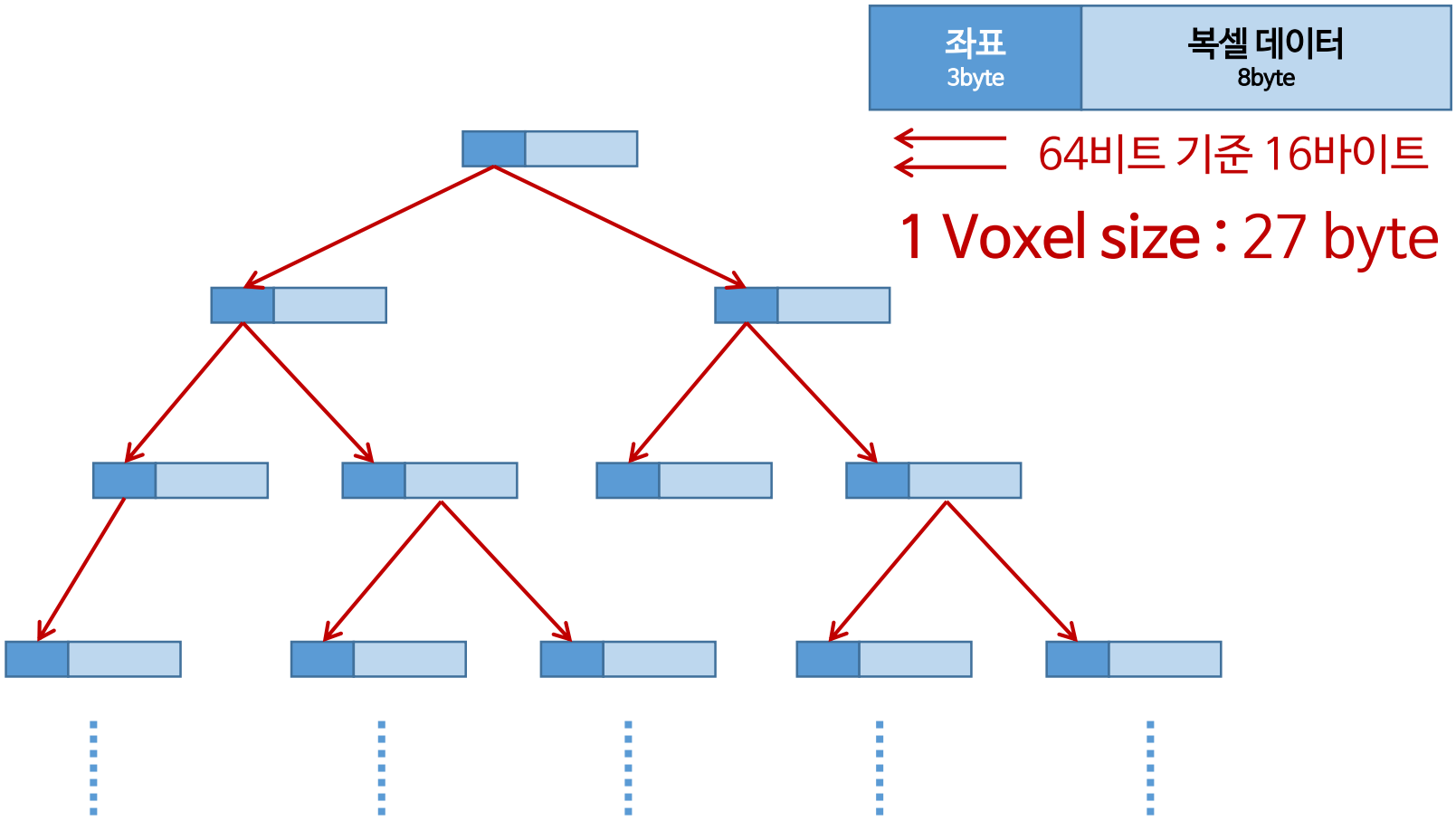
최초 구성 : STL Map

1 Voxel size :11 byte



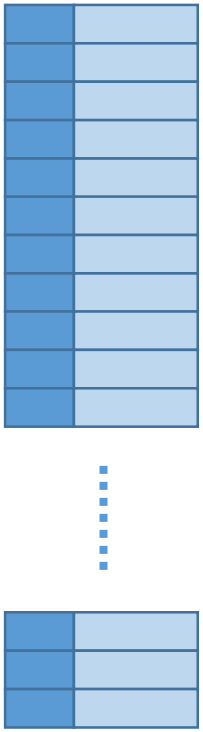
Grid 메모리 개선

최초 구성 : STL Map



Grid 메모리 개선

개선 후 구성 : 정렬된 벡터 + Map 인터페이스



좌표 3byte	복셀 데이터 8byte

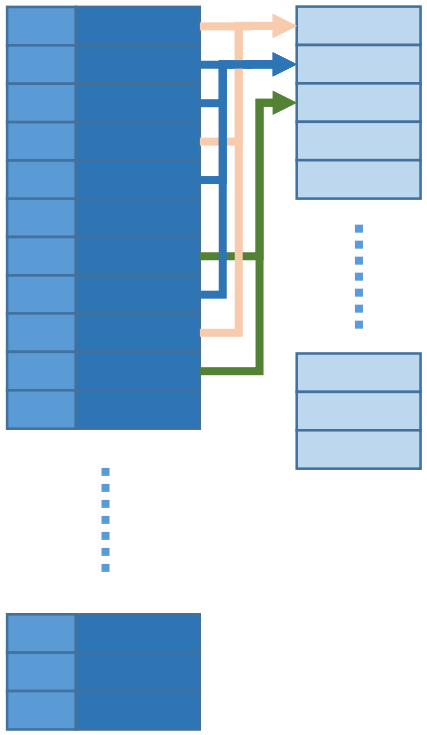
1 Voxel size : 11 byte

```
template <typename Key,  typename Value>
class BinaryMap
{
                iterator        Insert( std::pair<Key, Value> );
                iterator        Erase(Key);
                iterator        Find(Key);
};
```

내부 구현이 벡터로 변경되어, File I/O 과정이 축약됨

Grid 메모리 개선

개선 후 구성 : 정렬된 벡터 + Map 인터페이스



좌표	인덱스	복셀 데이터
3 byte	1~4 byte	8byte

1 Voxel size : 4 ~ 15 byte

인덱스를 통해 중복데이터 공유

가변크기 인덱스 적용하여 메모리 부담 최소화

복셀 분포가 넓지 않으면 좌표도 압축

최종적으로 메모리 사용량 약 50% 절약

5

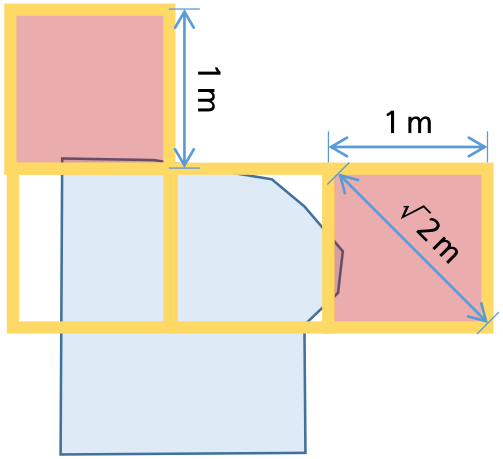
SubVoxel Resolution

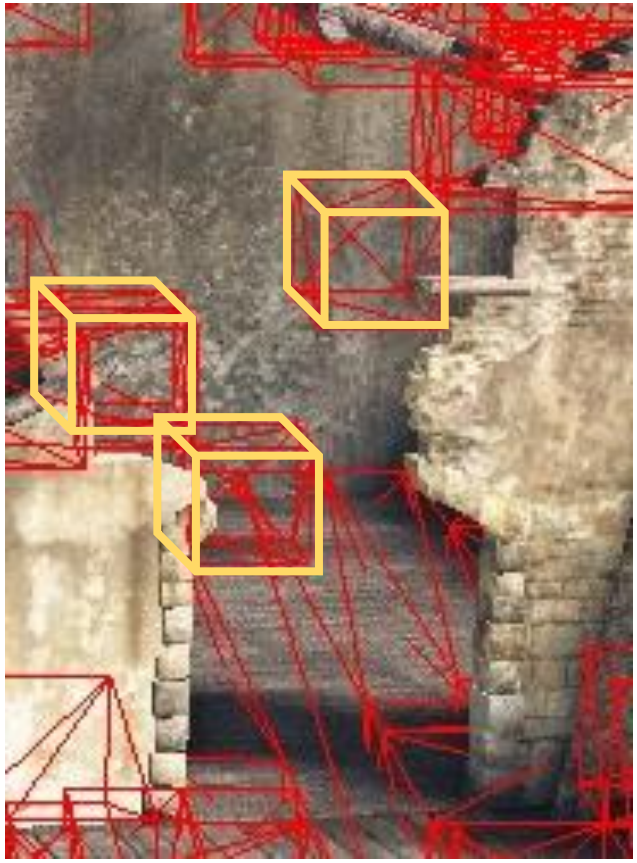
igc in 성남

코드도 실전이다

적용 후 좁은 통로에 대한 에러가 많은 곳에서 발견됨

(AI들이 몇몇 지역에서는 벽을 넘어가기도…)





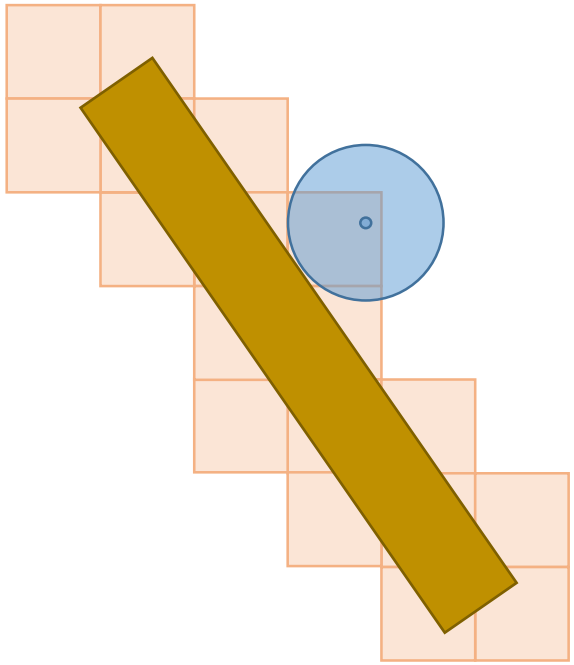
해상도 문제이기 때문에,
메모리 사용량과 길찾기 성능 개선이 절실

중간 개선 (해상도 증가, 복셀 크기 1m -> 50cm)

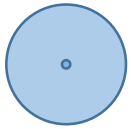
Sub Voxel

조금 더 해상도를 높여야 한다!

일부 벽 근처에서 아직도 캐릭터가 지형 검증에 걸리는 현상 발생



크기 50 cm



반지름 40 cm

안전한 복셀 크기
$40 / \sqrt{2} = 28.2885\cdots$

아이디어를 만나다


Menu (F10)
1265
5690
500
Turalyon
Level 1
109/180
Armor: 10
Damage: 3-19
Range: 1
Sight: 6
Speed: 13
Magic:
34
400
100
MENU

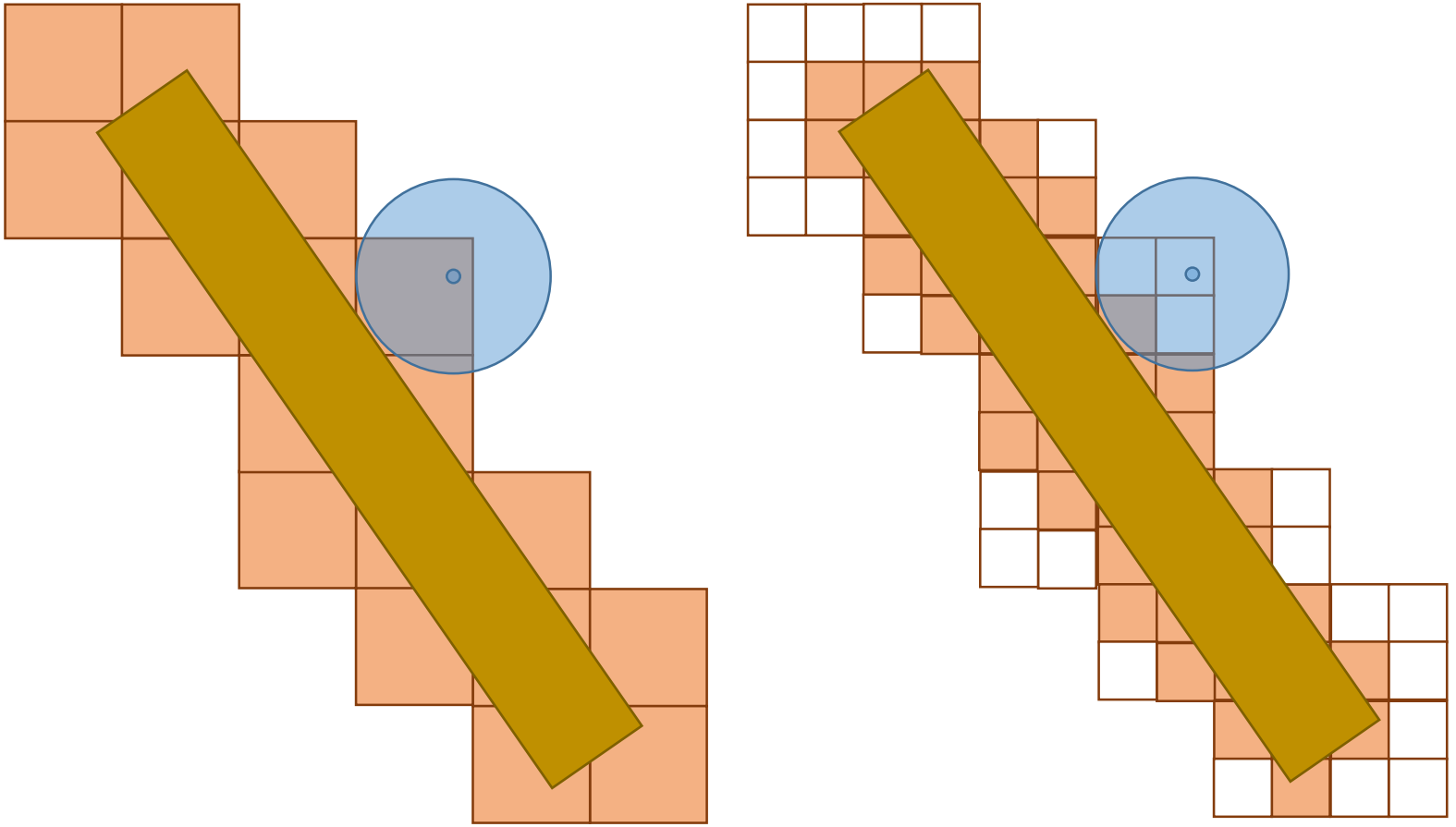
SubVoxel

Sub Pixel Resolution : 4bit (Sub Voxel 8bit)

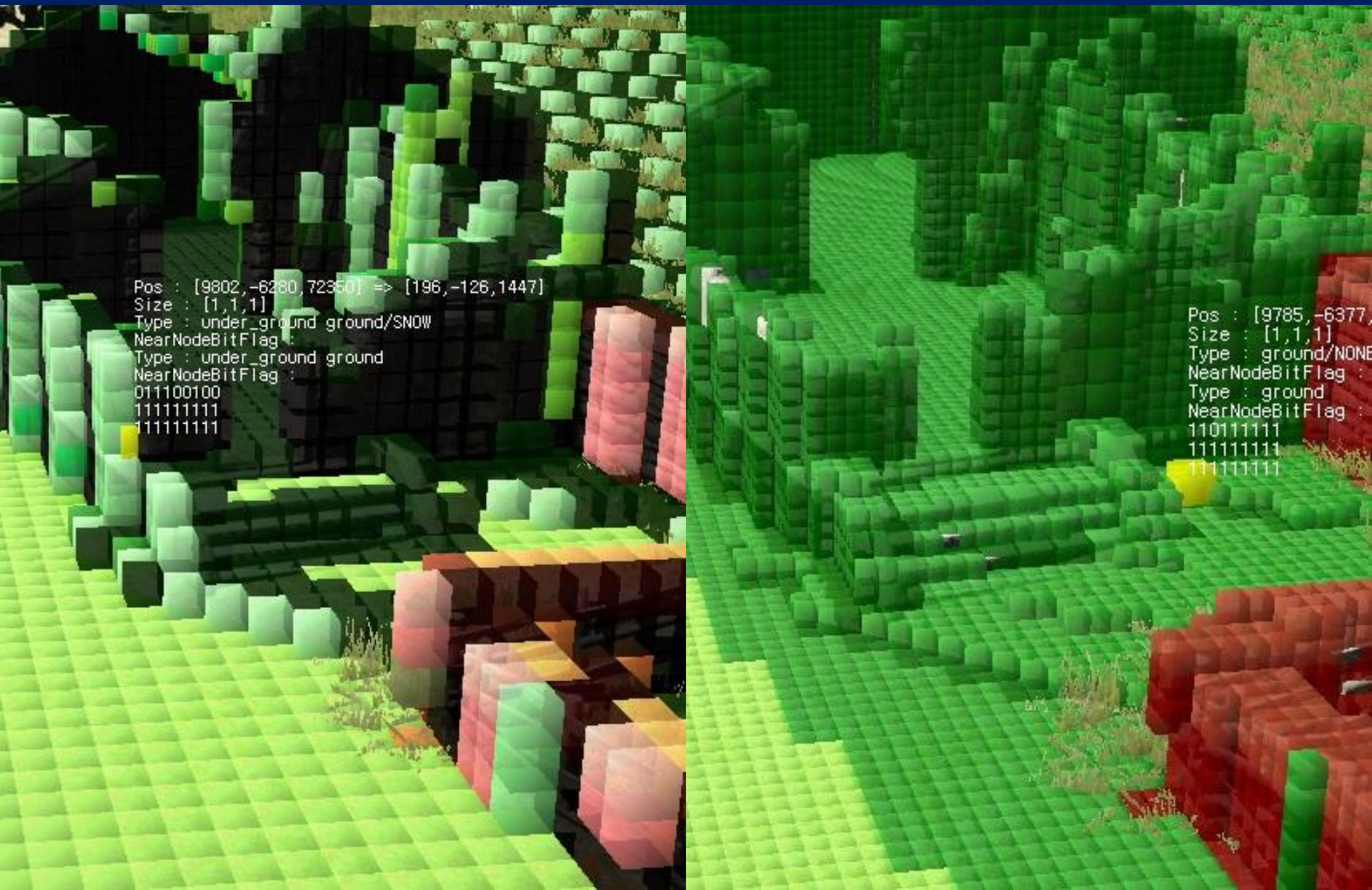
0 0 0 0	0000	0 0 0 1	0001	0 0 1 0	0010	0 0 1 1	0011
0 1 0 0	0100	0 1 0 1	0101	0 1 1 0	0110	0 1 1 1	0111
1 0 0 0	1000	1 0 0 1	1001	1 0 1 0	1010	1 0 1 1	1011
1 1 0 0	1100	1 1 0 1	1101	1 1 1 0	1110	1 1 1 1	1111

SubVoxel

Sub Voxel 적용한 가상 해상도 사용가능



SubVoxel Resolution



6

A* 길찾기 성능 최적화

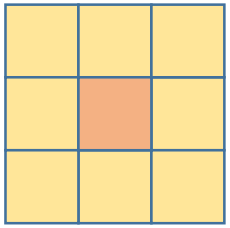
igc in 성남

Path Finding 성능 개선

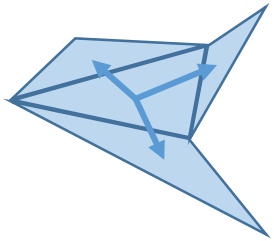
복셀 기반 A* 의 성능 문제

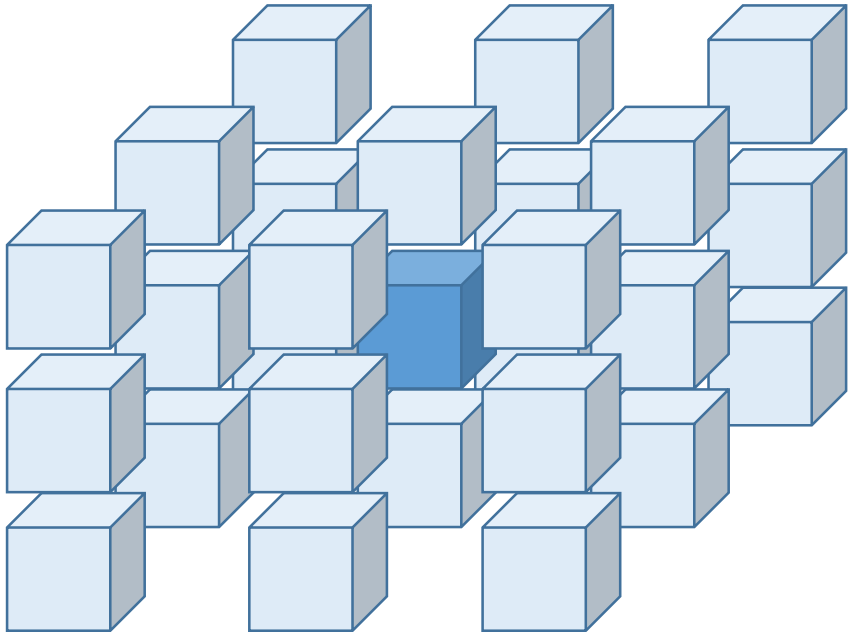
3차원 복셀의 경우 인접한 노드가 너무 많다! 무려 26개!

2차원 타일은 8개



네비 매쉬는 3~5개



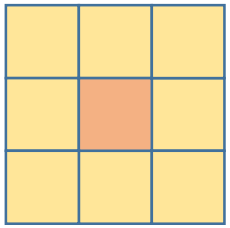


Path Finding 성능 개선

복셀 기반 A* 의 성능 문제

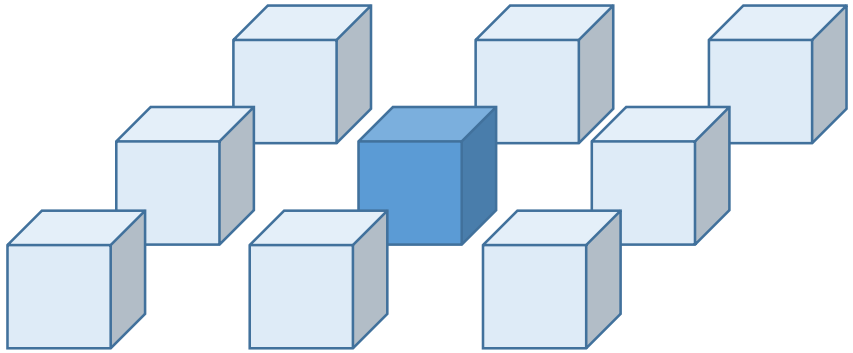
모든 인접노드가 활성화된 복셀은 거의 없다.
최초 데이터를 뽑을 때 인접노드 정보를 포함해 불필요한 조회를 줄인다

2차원 타일은 8개



≒

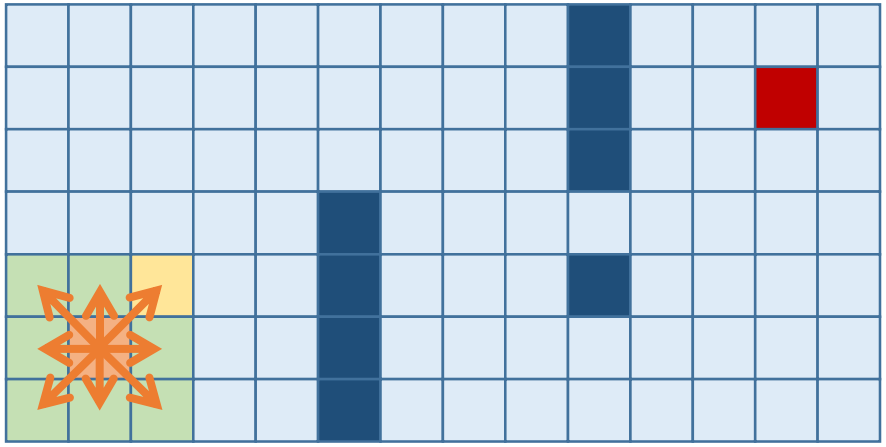
평지 데이터의 인접노드



Optimal vs Greedy A*

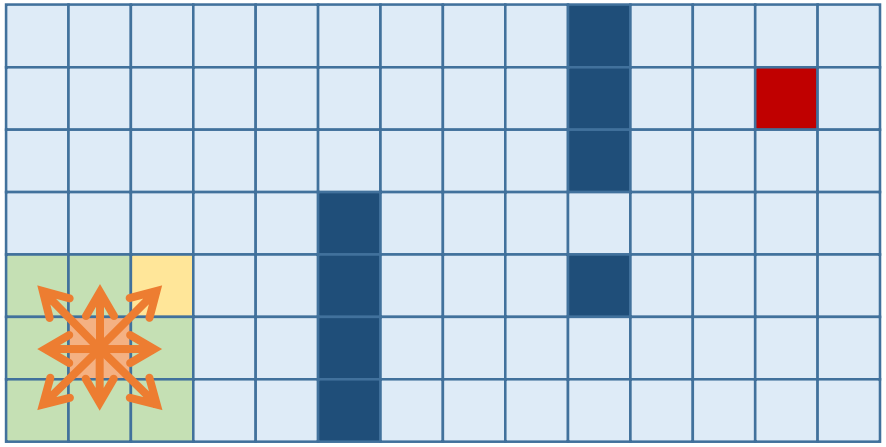
Optimal A*

Candidate : 9
Calc Weight : 8
Campare : 0



Greedy-First Search A*

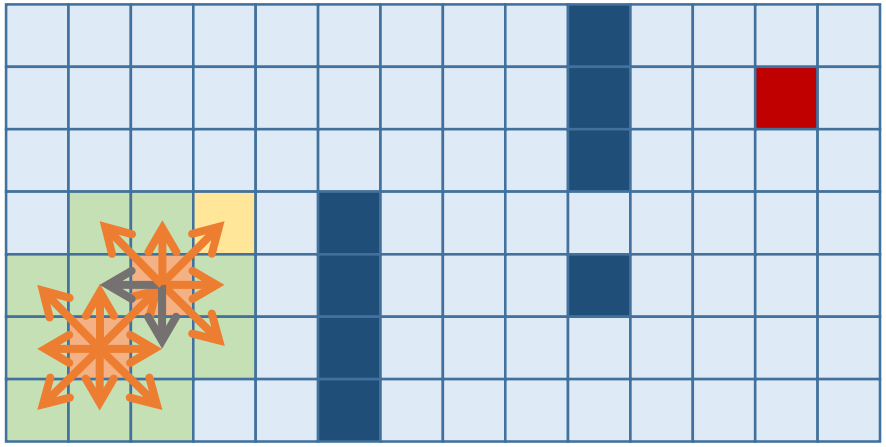
Candidate : 9
Calc Weight : 8
Campare : 0



Optimal vs Greedy A*

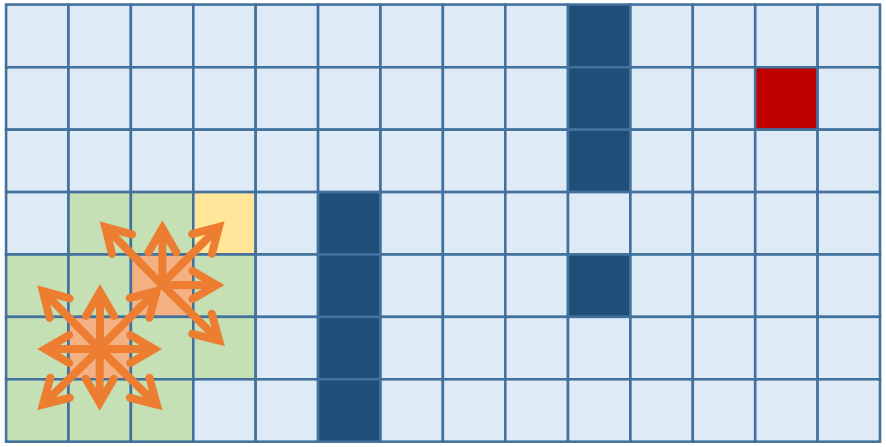
Optimal A*

Candidate : 14
Calc Weight : 16
Campare : 2



Greedy-First Search A*

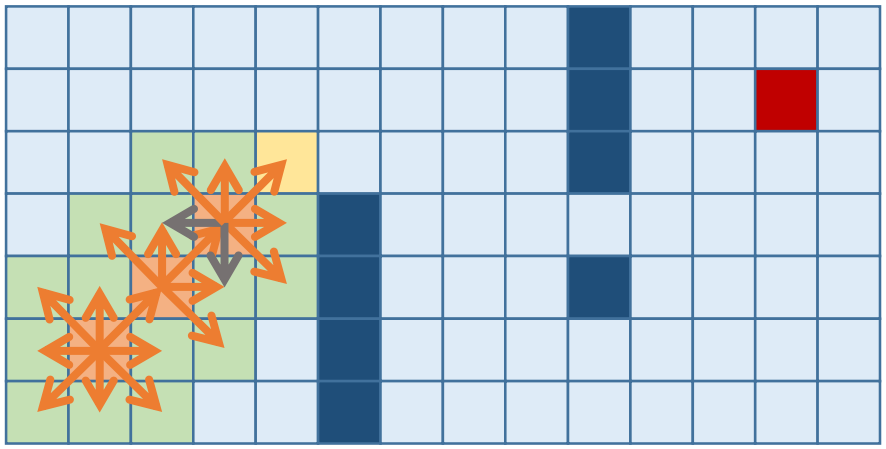
Candidate : 14
Calc Weight : 14
Campare : 0



Optimal vs Greedy A*

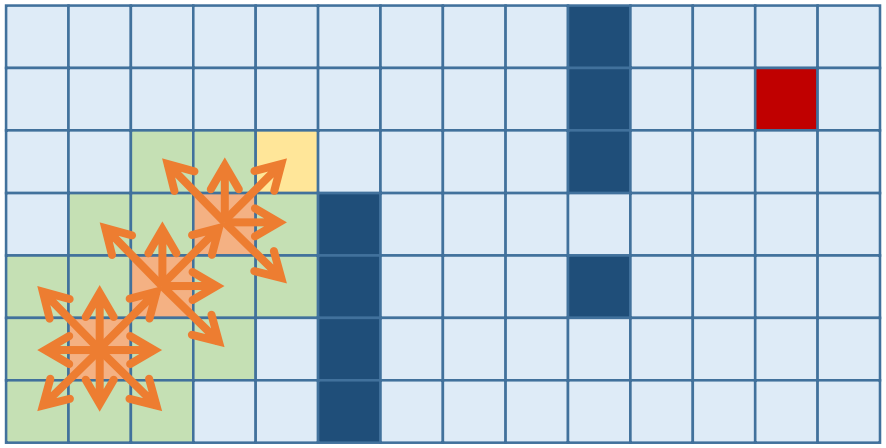
Optimal A*

Candidate : 19
Calc Weight : 23
Campare : 4



Greedy-First Search A*

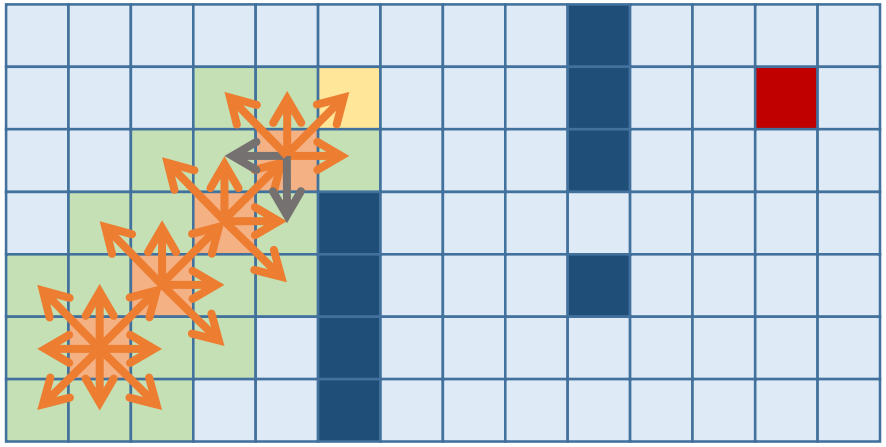
Candidate : 19
Calc Weight : 19
Campare : 0



Optimal vs Greedy A*

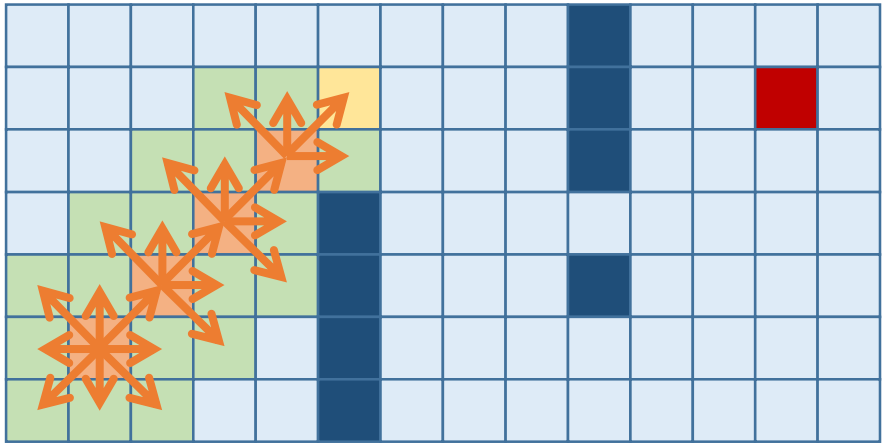
Optimal A*

Candidate : 23
Calc Weight : 29
Campare : 6



Greedy-First Search A*

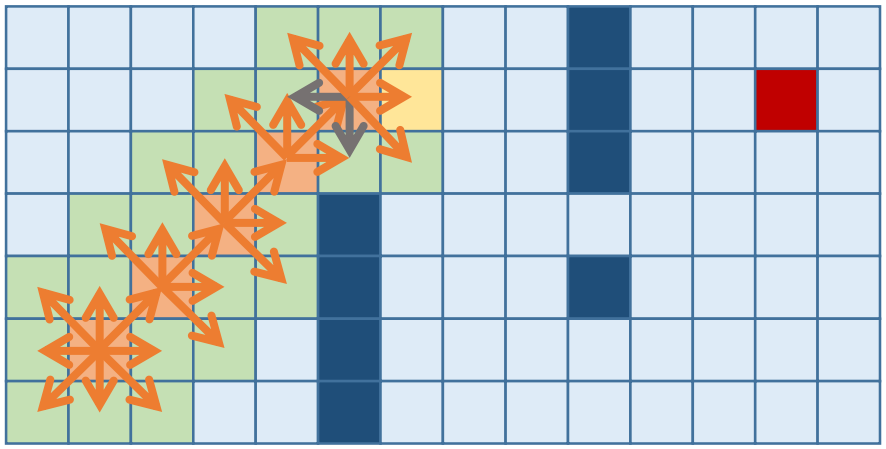
Candidate : 23
Calc Weight : 23
Campare : 0



Optimal vs Greedy A*

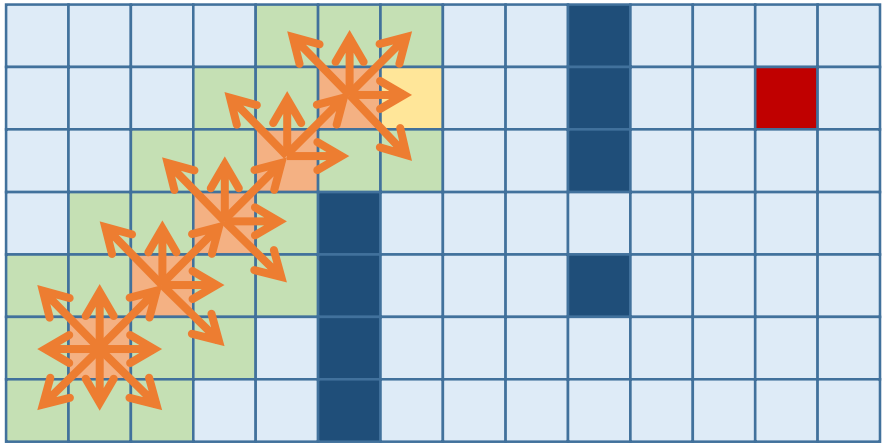
Optimal A*

Candidate : 28
Calc Weight : 36
Campare : 8



Greedy-First Search A*

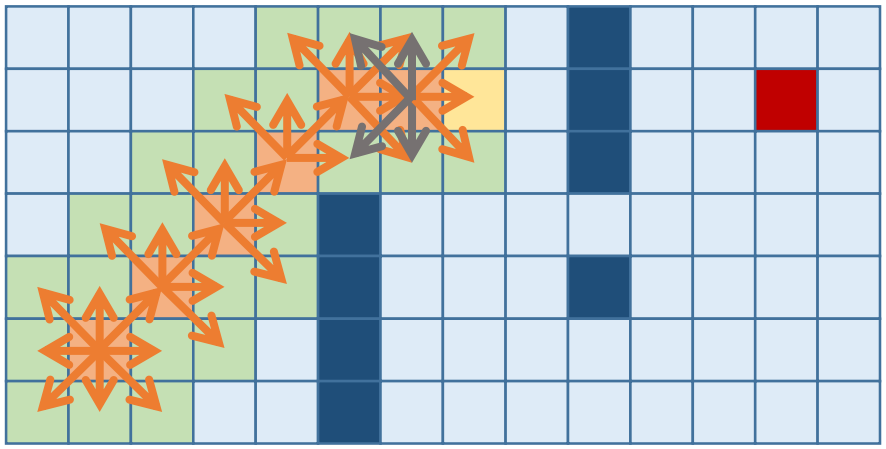
Candidate : 28
Calc Weight : 28
Campare : 0



Optimal vs Greedy A*

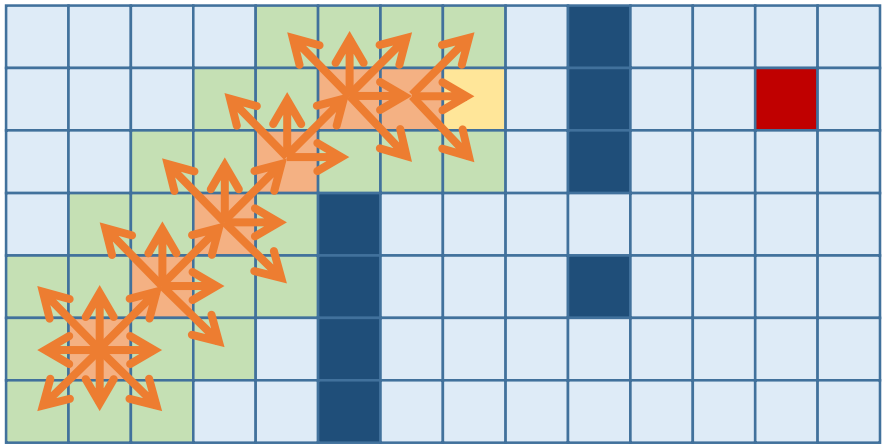
Optimal A*

Candidate : 31
Calc Weight : 43
Campare : 12



Greedy-First Search A*

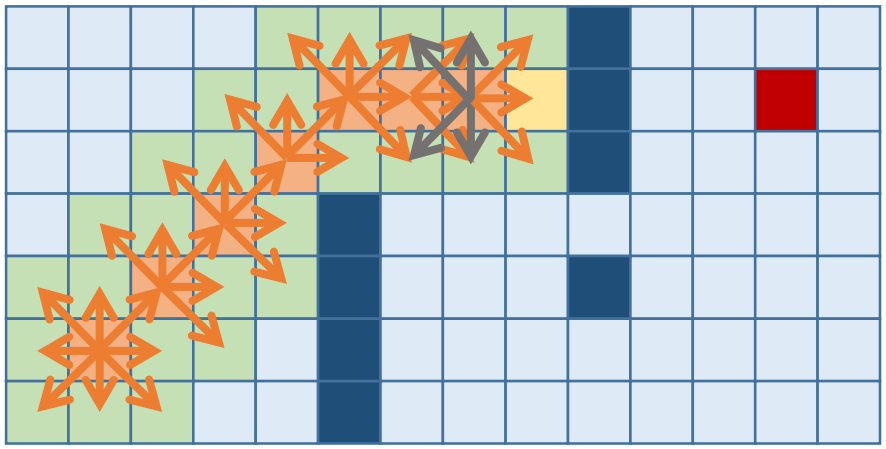
Candidate : 31
Calc Weight : 31
Campare : 0



Optimal vs Greedy A*

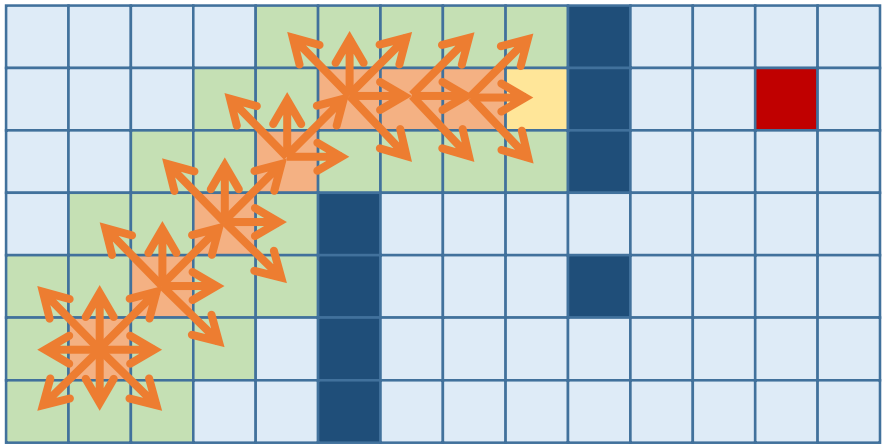
Optimal A*

Candidate : 34
Calc Weight : 50
Campare : 16



Greedy-First Search A*

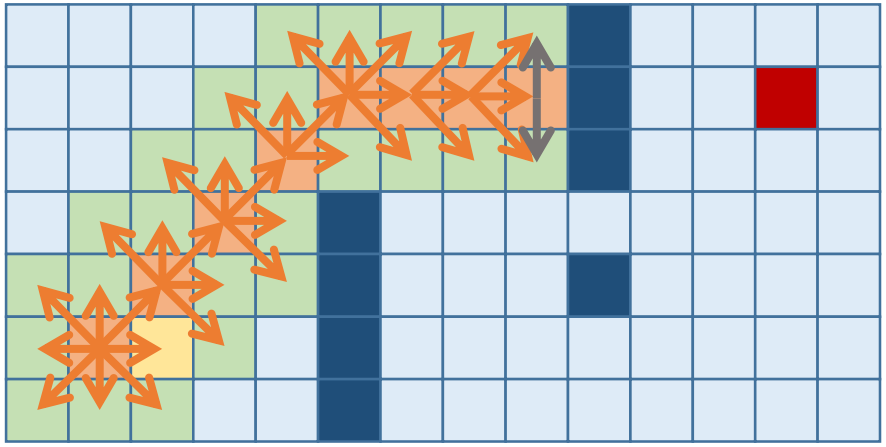
Candidate : 34
Calc Weight : 34
Campare : 0



Optimal vs Greedy A*

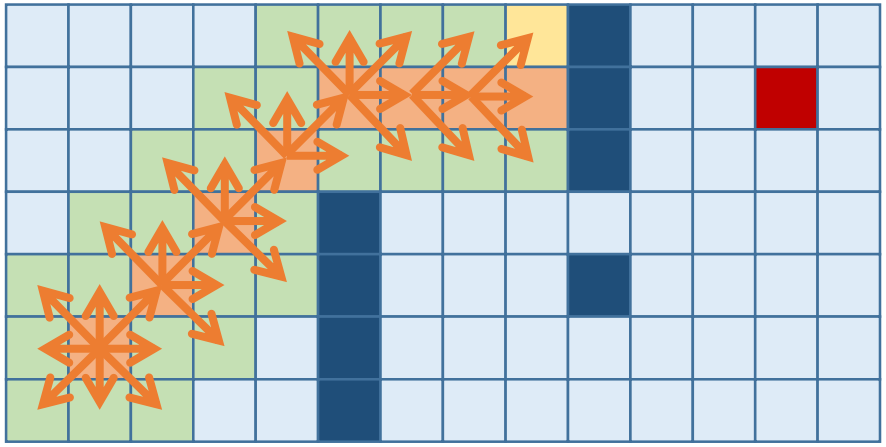
Optimal A*

Candidate : 34
Calc Weight : 52
Campare : 18



Greedy-First Search A*

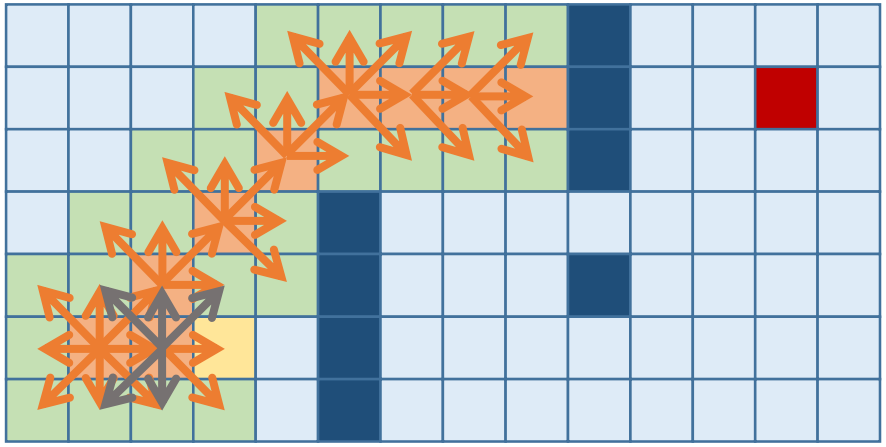
Candidate : 34
Calc Weight : 34
Campare : 0



Optimal vs Greedy A*

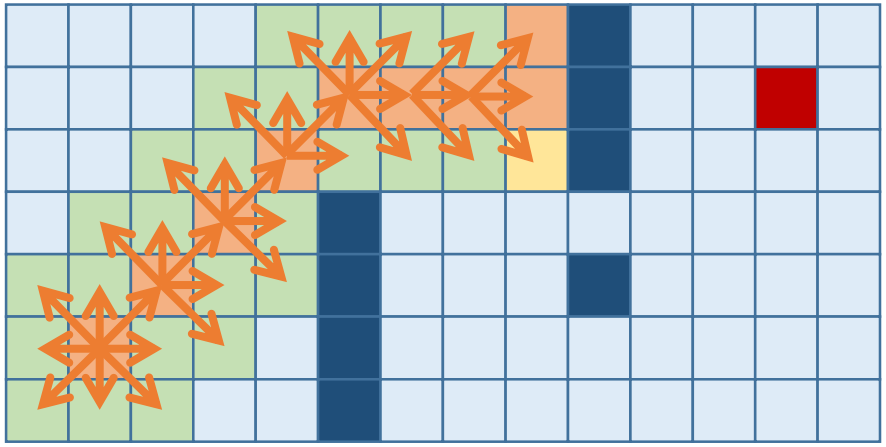
Optimal A*

Candidate : 35
Calc Weight : 59
Campare : 23



Greedy-First Search A*

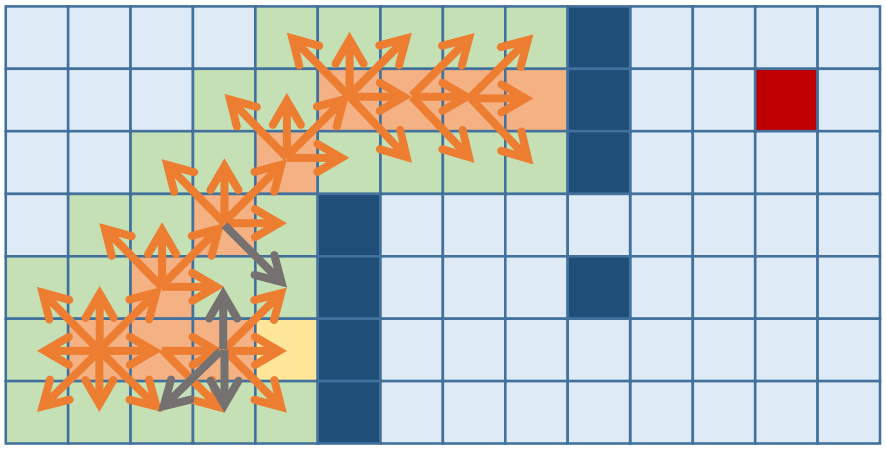
Candidate : 34
Calc Weight : 34
Campare : 0



Optimal vs Greedy A*

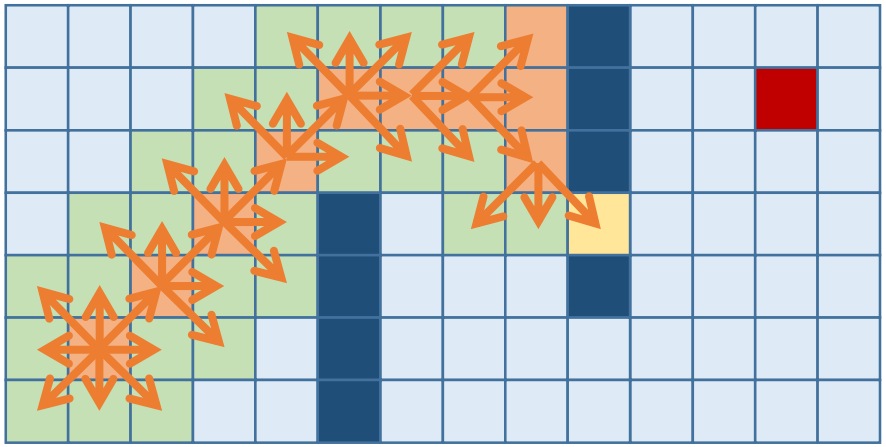
Optimal A*

Candidate : 37
Calc Weight : 65
Campare : 27



Greedy-First Search A*

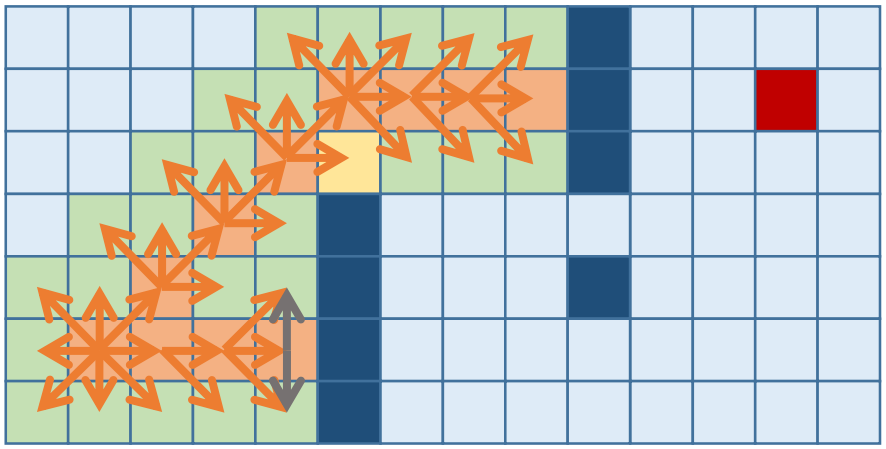
Candidate : 37
Calc Weight : 37
Campare : 0



Optimal vs Greedy A*

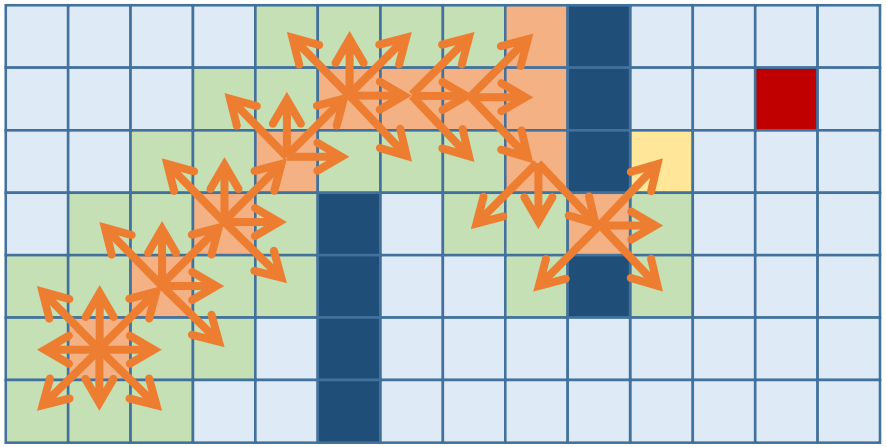
Optimal A*

Candidate : 37
Calc Weight : 67
Campare : 29



Greedy-First Search A*

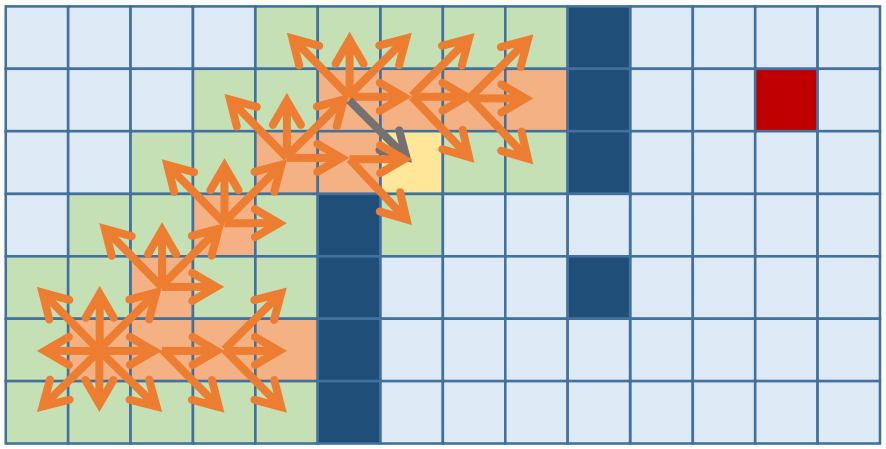
Candidate : 41
Calc Weight : 41
Campare : 0



Optimal vs Greedy A*

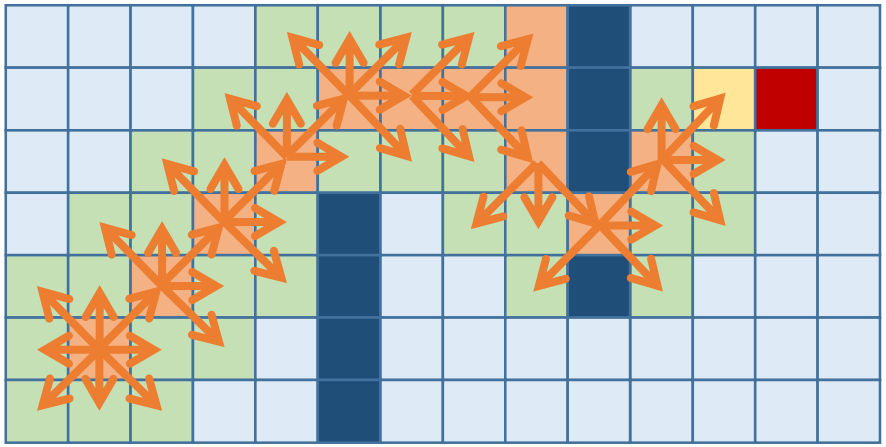
Optimal A*

Candidate : 38
Calc Weight : 69
Campare : 30



Greedy-First Search A*

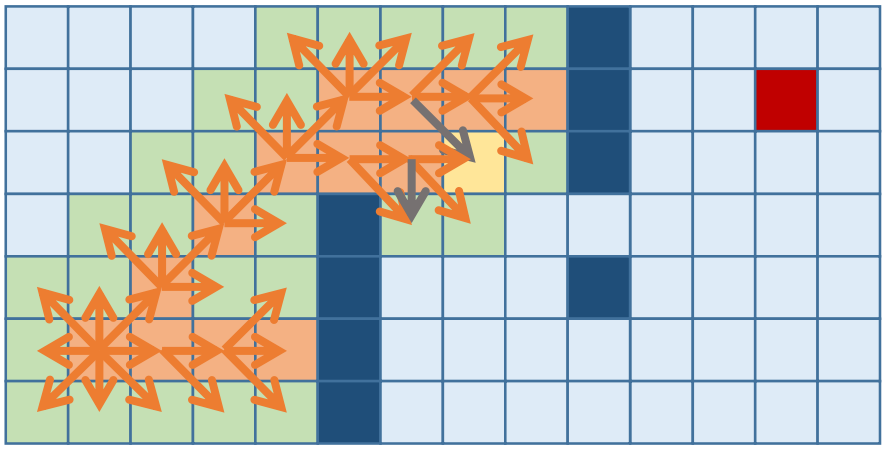
Candidate : 45
Calc Weight : 45
Campare : 0



Optimal vs Greedy A*

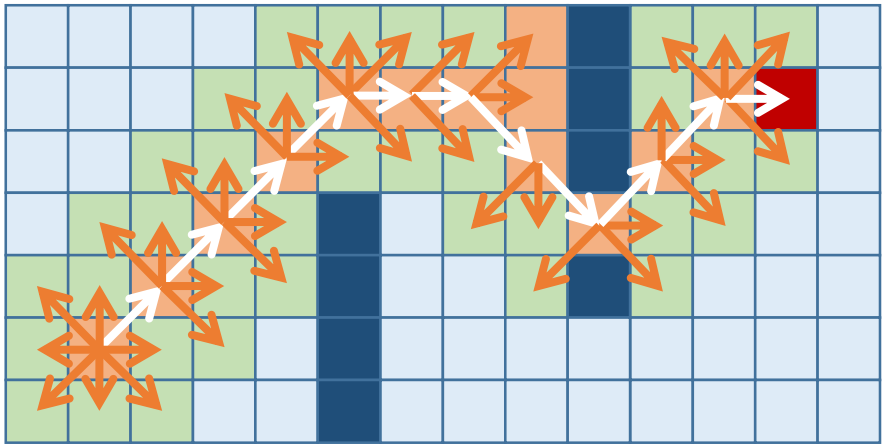
Optimal A*

Candidate : 39
Calc Weight : 72
Campare : 32



Greedy-First Search A*

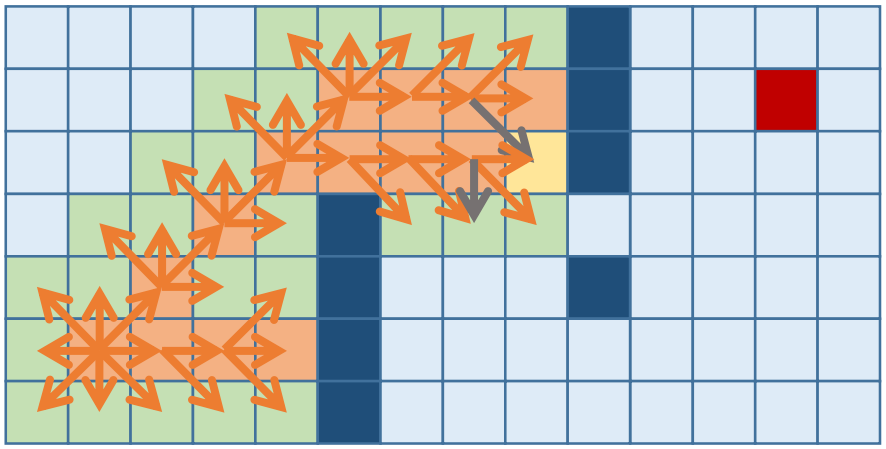
Candidate : 50
Calc Weight : 50
Campare : 0



Optimal vs Greedy A*

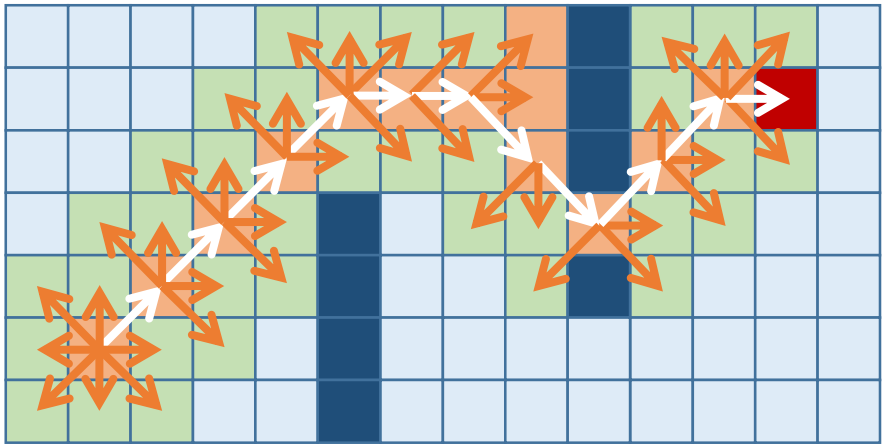
Optimal A*

Candidate : 40
Calc Weight : 75
Campare : 34



Greedy-First Search A*

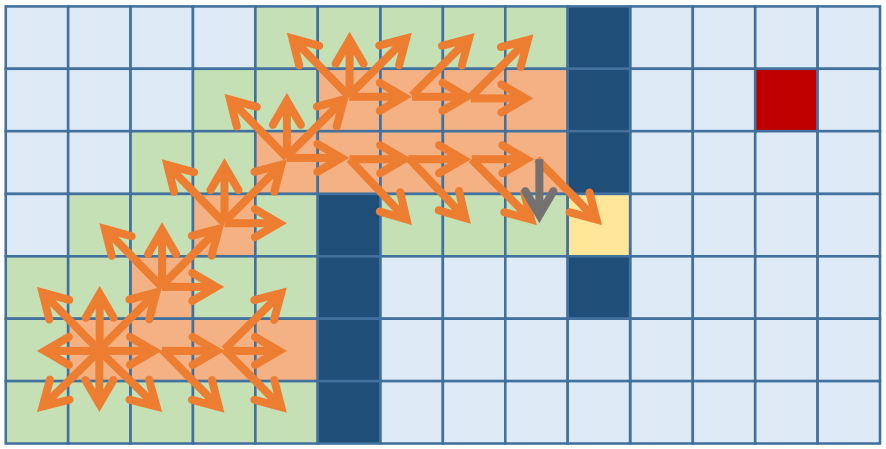
Candidate : 50
Calc Weight : 50
Campare : 0



Optimal vs Greedy A*

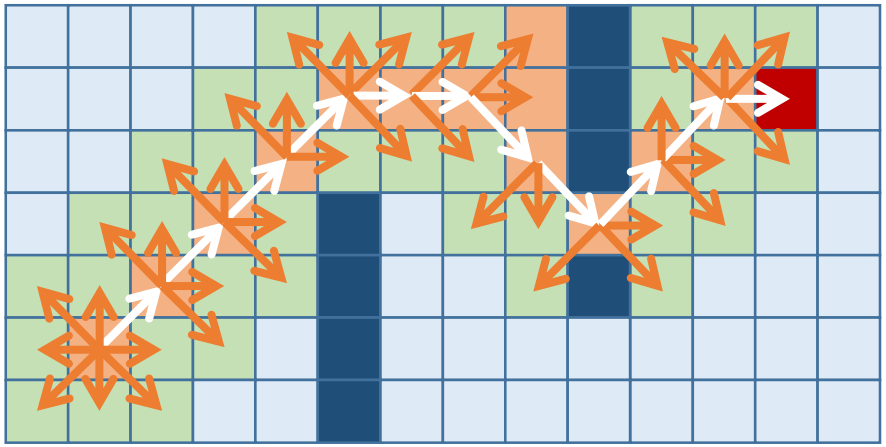
Optimal A*

Candidate : 41
Calc Weight : 77
Campare : 35



Greedy-First Search A*

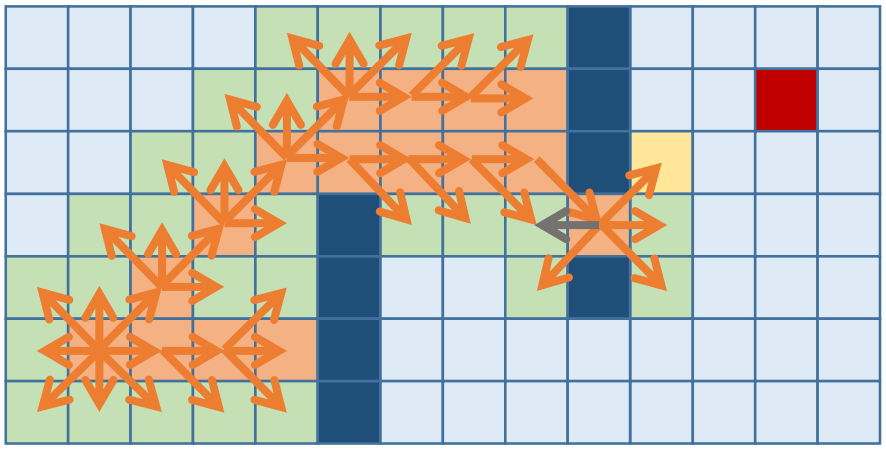
Candidate : 50
Calc Weight : 50
Campare : 0



Optimal vs Greedy A*

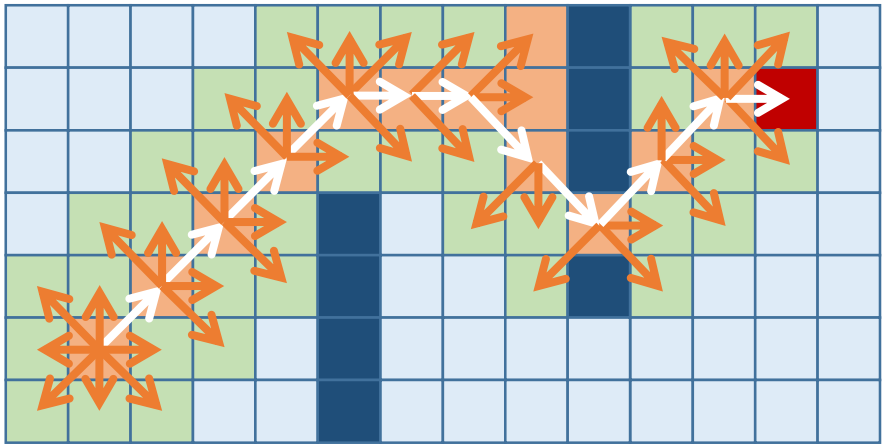
Optimal A*

Candidate : 45
Calc Weight : 82
Campare : 36



Greedy-First Search A*

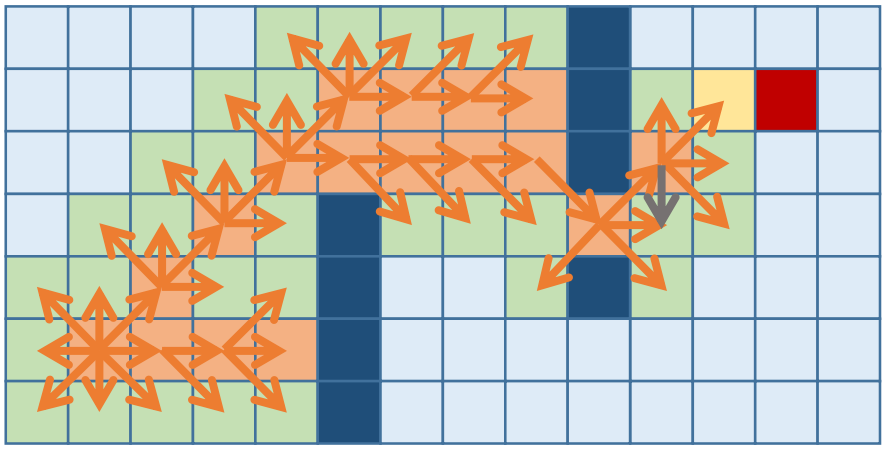
Candidate : 50
Calc Weight : 50
Campare : 0



Optimal vs Greedy A*

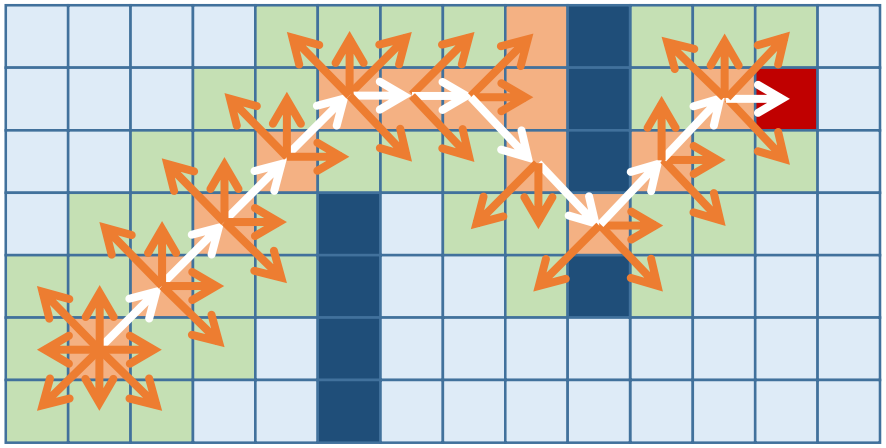
Optimal A*

Candidate : 49
Calc Weight : 87
Campare : 37



Greedy-First Search A*

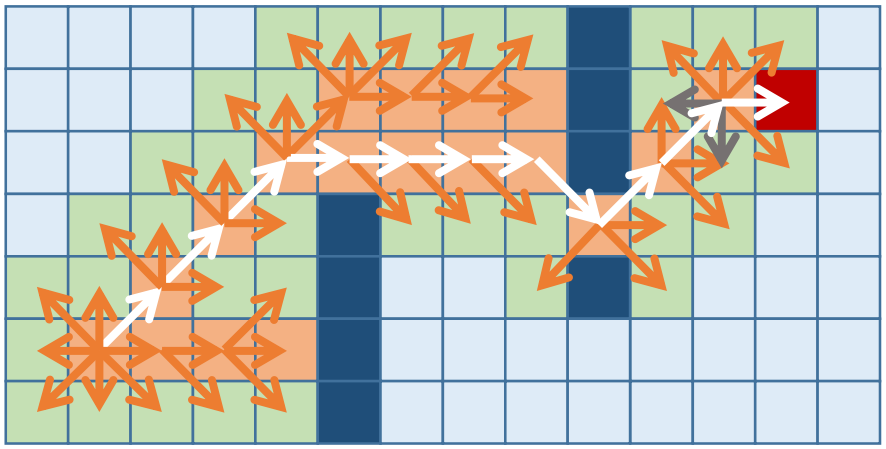
Candidate : 50
Calc Weight : 50
Campare : 0



Optimal vs Greedy A*

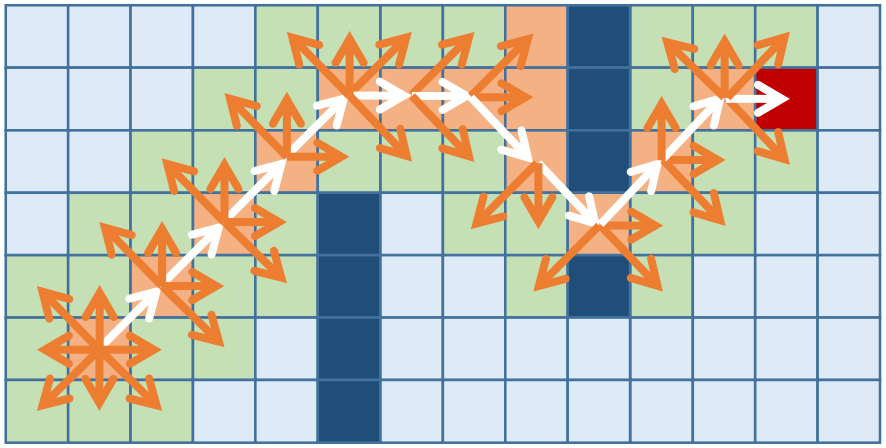
Optimal A*

Candidate : 54
Calc Weight : 94
Campare : 39



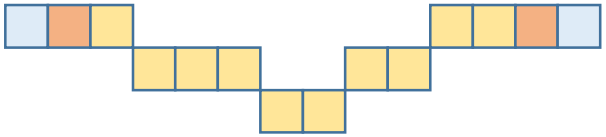
Greedy-First Search A*

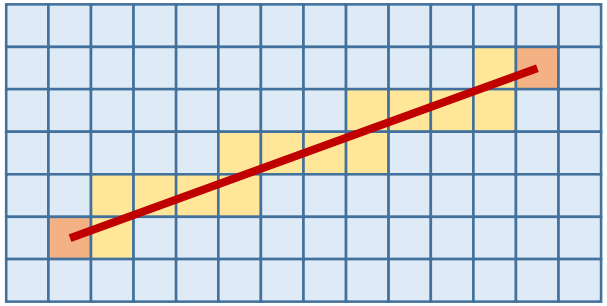
Candidate : 50
Calc Weight : 50
Campare : 0

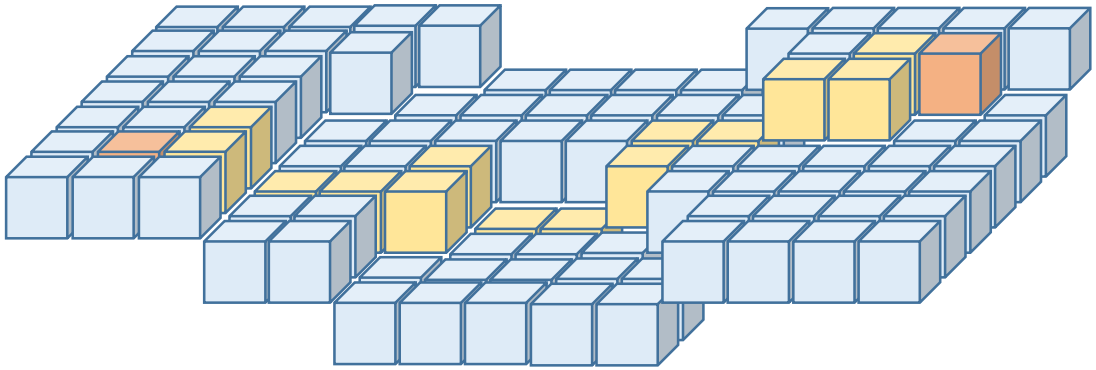


A*까지 가는 경우를 최대한 줄여야 한다!

가장 좋은 방법은 A*를 돌리지 않고 길을 찾는 것





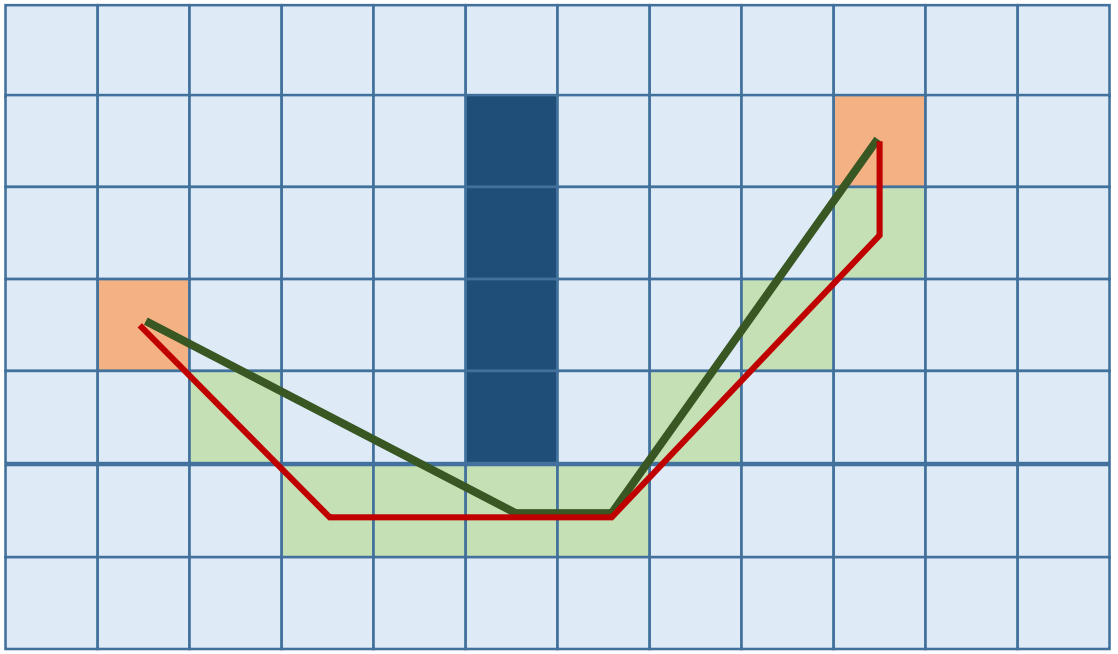


Path Finding 성능 개선

패스 최적화

A* 결과

최적화 결과

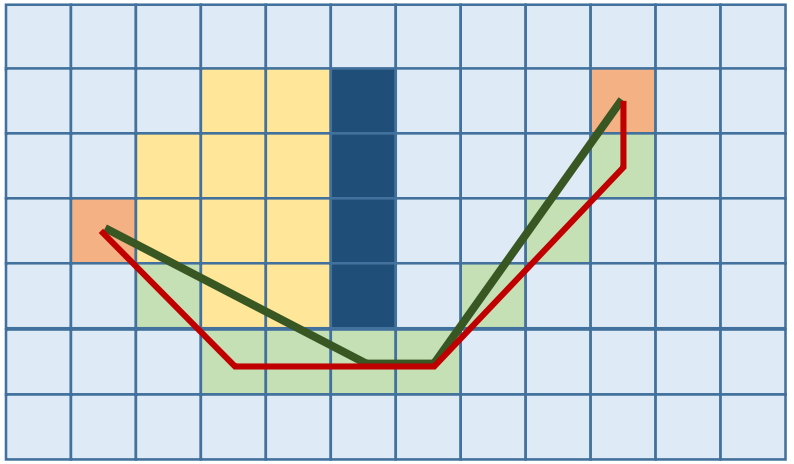


Path Finding 성능 개선

패스 최적화

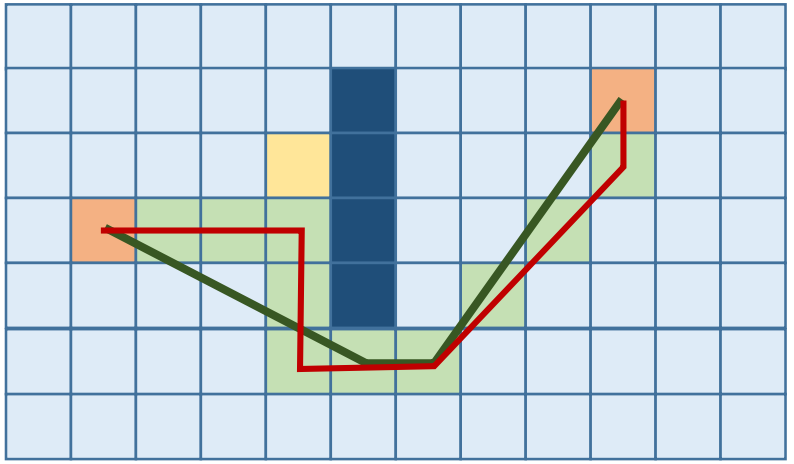
Optimal A*

최적화 결과



Greedy A*

최적화 결과

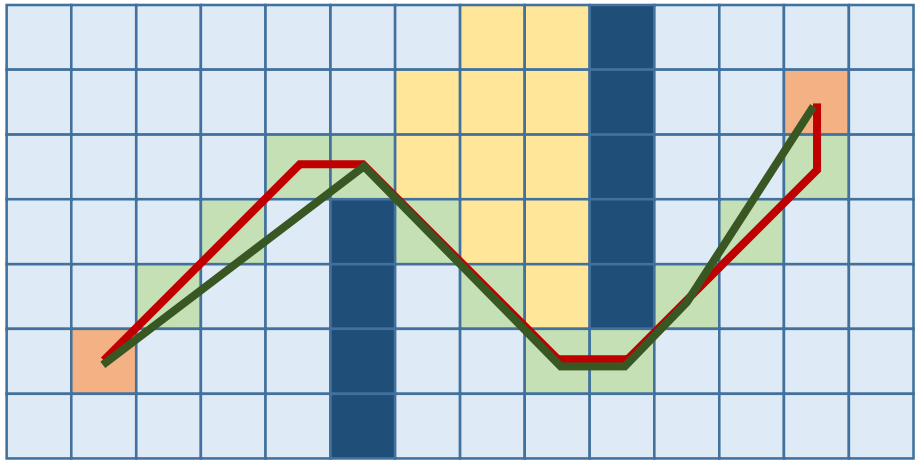


Path Finding 성능 개선

패스 최적화

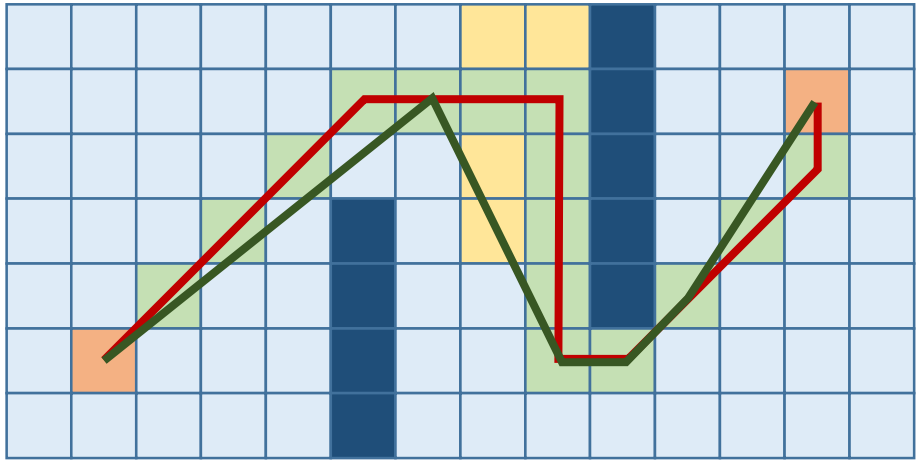
Optimal A*

최적화 결과

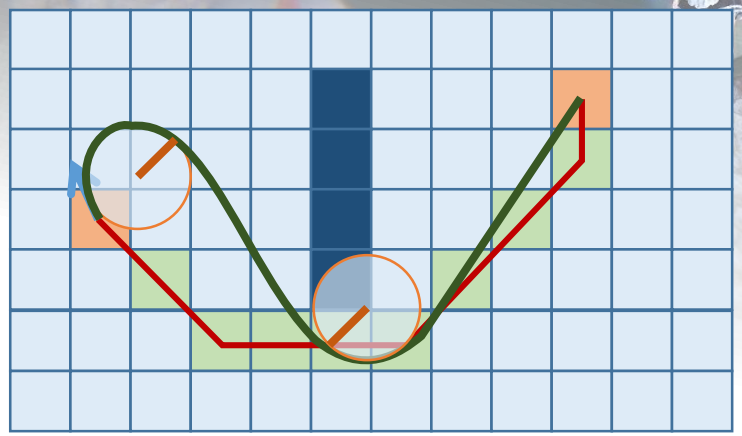


Greedy A*

최적화 결과



곡선 경로 탐색 (X) - 패스 곡선화

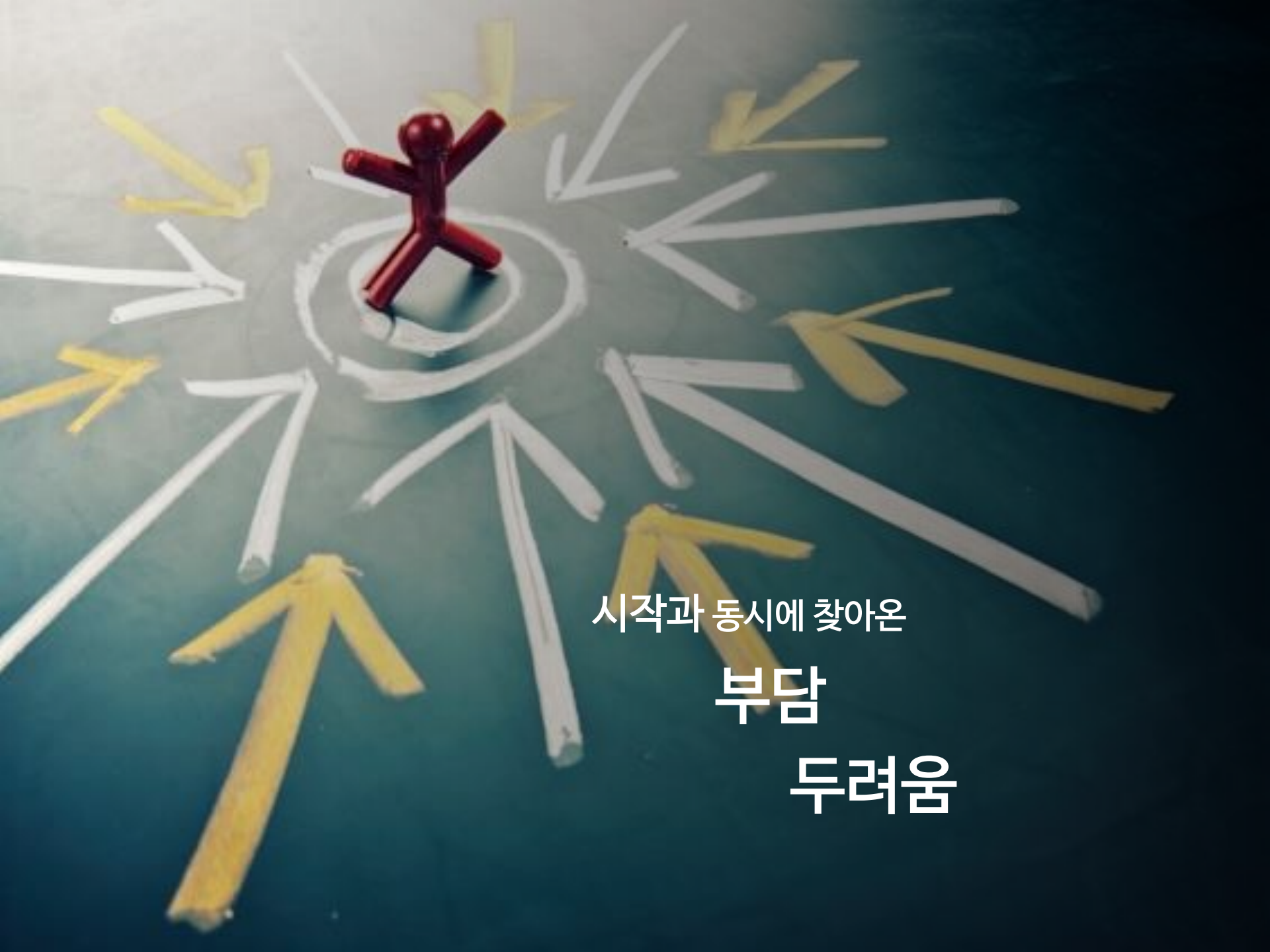


Agent 초기방향
회전 반경
처리 전
처리 후 결과

7

맨 땅에서 프로토 타입까지…

igc in 성남


시작과 동시에 찾아온
부담
두려움

Recast / Detour


대상에 대한
부족한 이해

Prototype

마치며…

늦는 것을 두려워 말고
멈추는 것을
두려워하라

不怕慢只怕站

igc in 성남

감사합니다.

프로그래머 민경인

in PEARL ABYSS