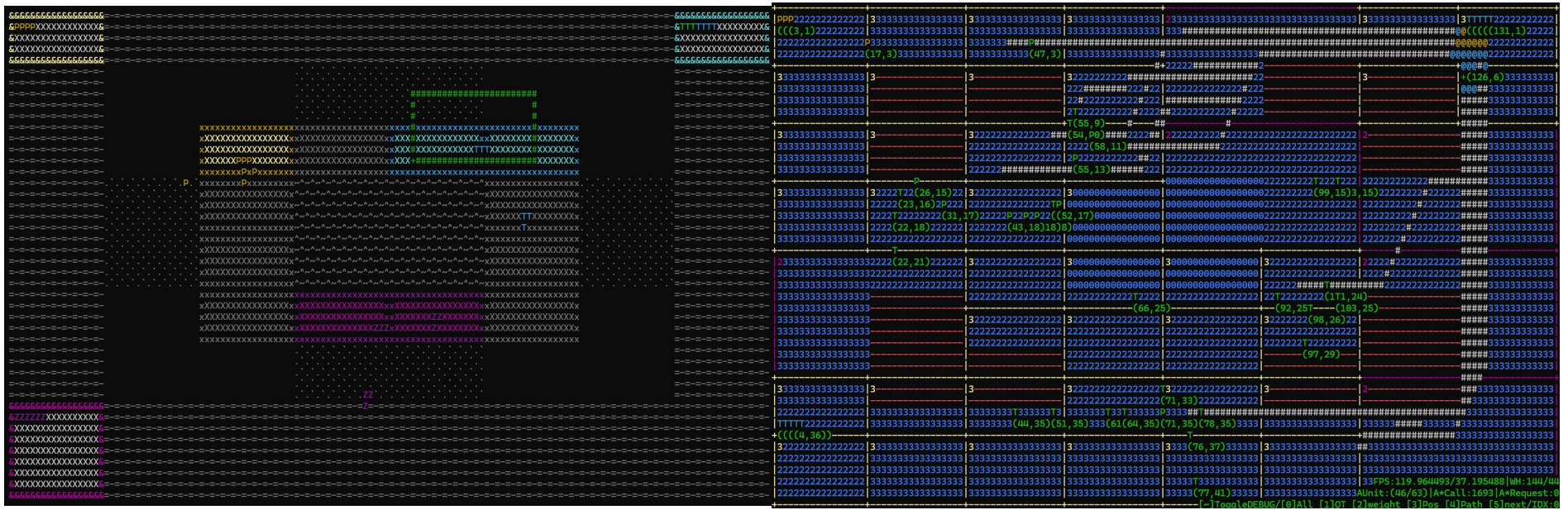
알고리즘 응용 프로젝트
(ASCII WARS)

김주영 / 원티드 포텐업 게임 개발자 양성과정 3기 / 2025-09-01



게임 제목	ASCII WARS
장르	탑뷰, 실시간 전략(RTS, Real-Time Strategy)

School Wars

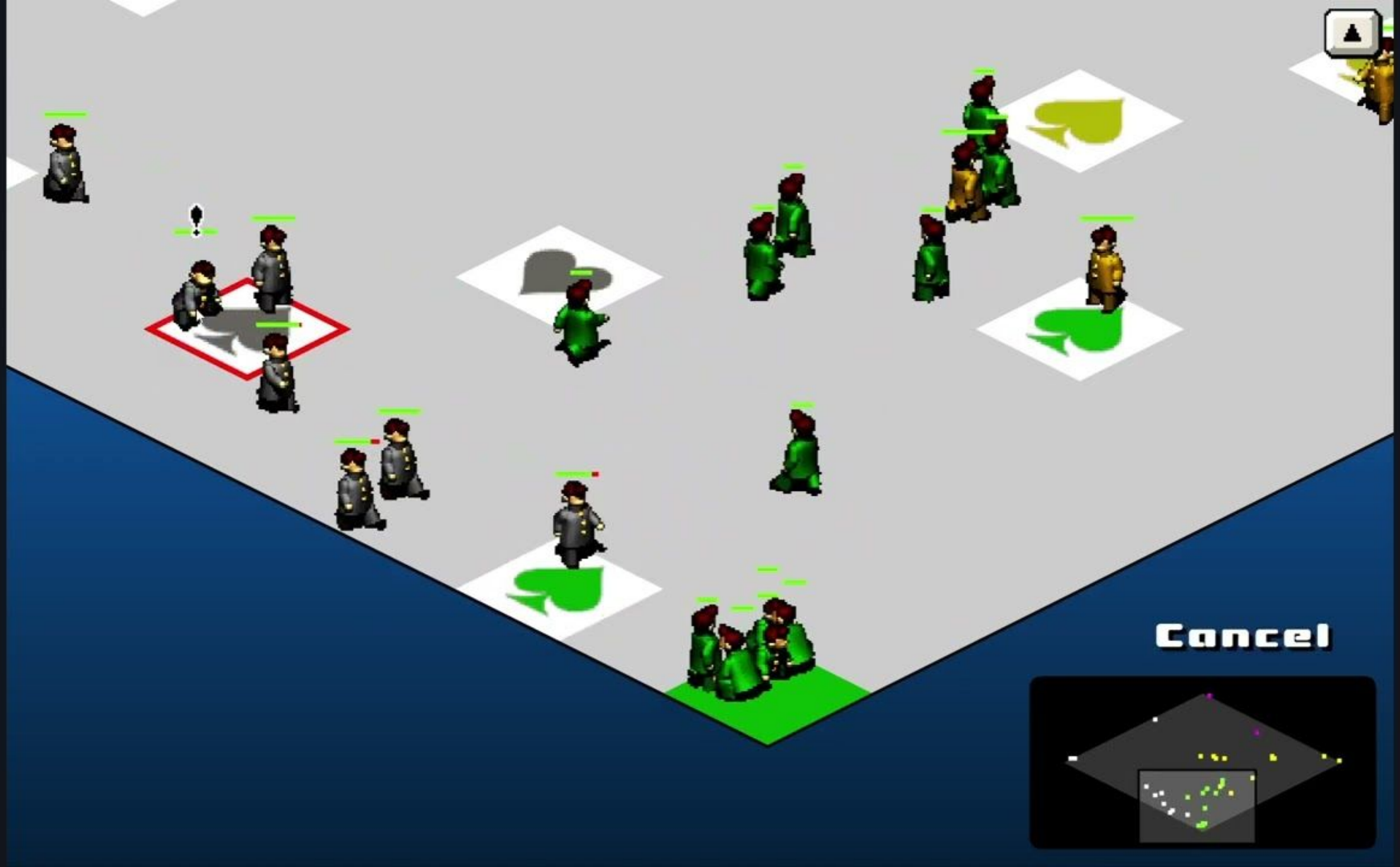
GAMEDESIGN


Black


Green


Yellow


Purple


Cancel

1. 프로젝트 설명

게임 목표

- 최대한 많은 영역 차지하기
- 여러 가지 지형 을 전략적으로 사용해 적의 본진까지 차지하기

프로젝트 목표

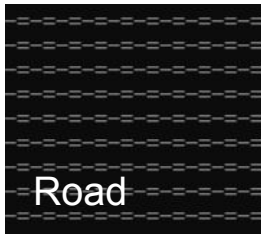
- 각 객체가 위치한 곳을 쿼드트리 화면 분할로 시각화 해보기
- A* 를 통해 RTS 유닛 이동 구현해 보기
- 여럿이 같은 위치를 이동할 때 부자연스럽지 않도록 군집 행동까지 구현해 보기
- (심화) 전투 같은 것도 만들어 보기

2. 구현된 것들

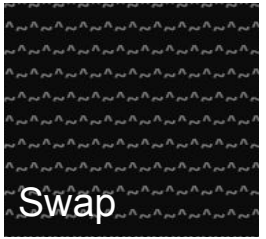
```
enum class TerrainType
{
    Ground = 0, // 잔디밭
    Road = 1, // 도로
    Swamp = 2, // 늪지대
    Pit = 3, // 구멍(이동 못함)
    Territory = 4, // 땅따먹기
    SpawnPool = 5, // 스폰하는 곳
    Size
};

// 속도 계수 테이블
const float speedMultipliers[6] =
{
    1.0f, // Grass
    1.5f, // Road
    0.2f, // Swamp
    -1.f,  // Pit
    1.0f,  // Territory
    1.0f  // Pool
};
```

기본적인 지형



빨리 이동할 수
있는 지형



느리게
이동하는 지형

지나갈 수
없는 지형

점령 가능한
지형

유닛이 생성되고
점령 가능한 지형


// 속도 계수 테이블
const float speedMultipliers[6]
{
1.0f, // Grass
1.5f, // Road
0.2f, // Swamp
-1.f, // Pit
1.0f, // Territory
1.0f // Pool
};
FPS:119.888260/33.602306|WH:144/44
AUnit:(16/63)|A*Call:0|A*Request:0
[~]ToggleDEBUG/[0]All [1]QT [2]weight [3]Pos [4]Path [5]next/IDX:2


FPS:119.980324/32.375988|WH:144/44
AUnit:(25/63)|A*Call:273|A*Request:0
[~]ToggleDEBUG/[0]All [1]QT [2]weight [3]Pos [4]Path [5]next/IDX:4

3. 구현 중 겪었던 문제

(1) 쿼드 트리 정수 나눗셈 오차 문제

(2) A* 최적화 문제

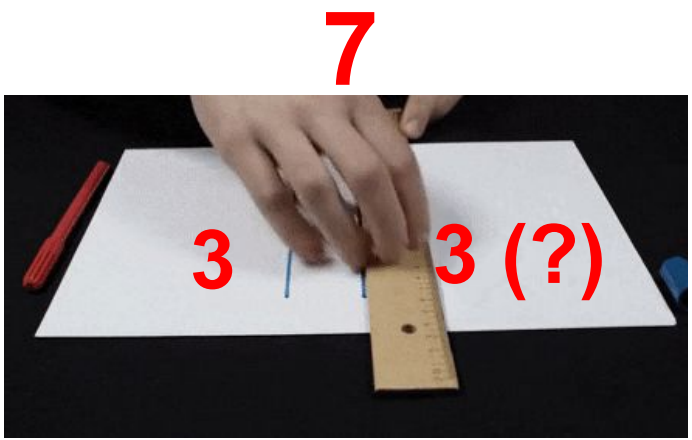
(3) 기타 여러가지 실수들

3 - (1) 쿼드 트리 정수 나눗셈 오차 문제

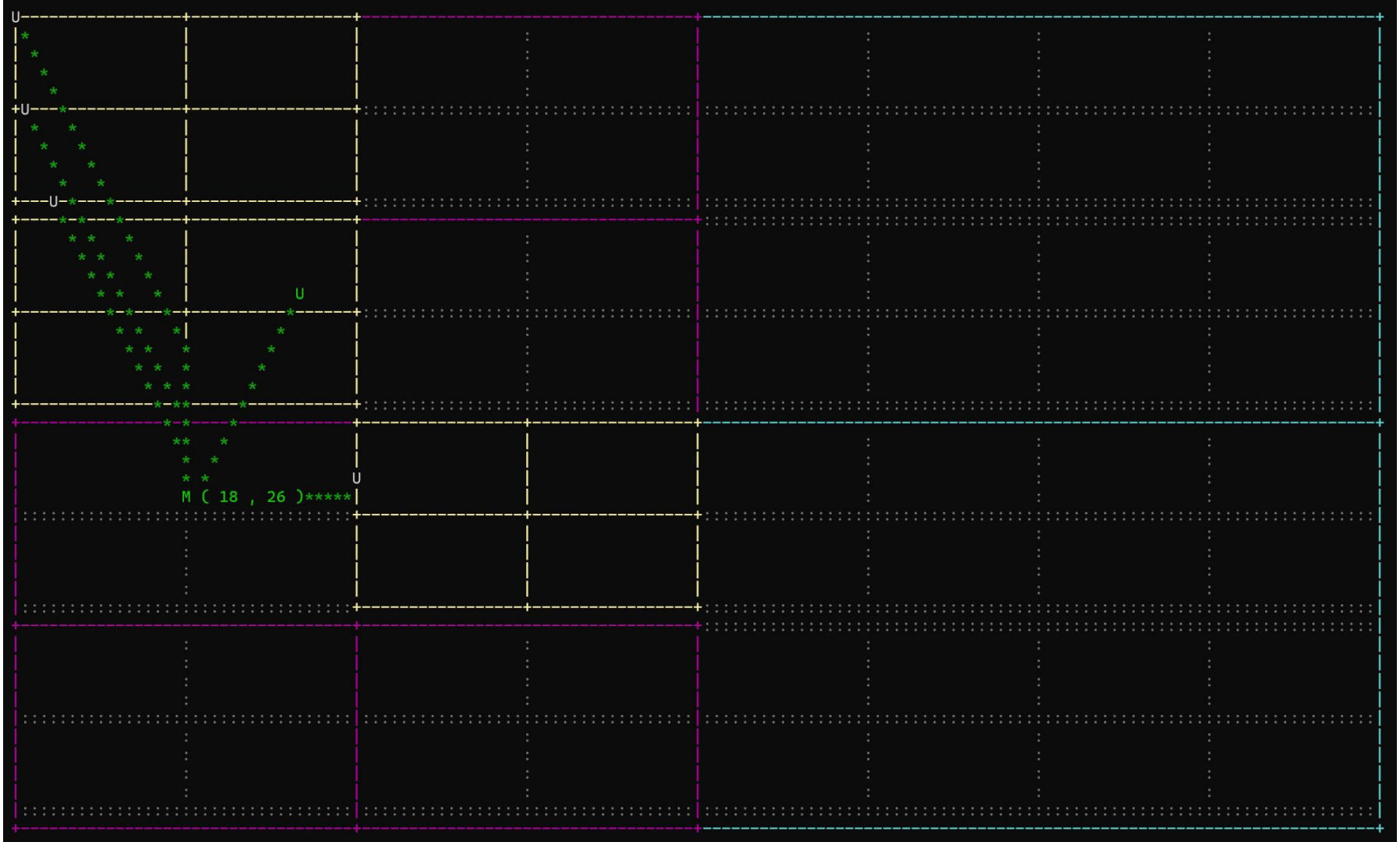
문제

- 수업시간에 배운 쿼드트리의 범위(Bounds)는 소수점 → 오차가 있더라도 근소
- 프로젝트에서 시각화 하려는 터미널 화면은 정수 → **오차 1도 큼**

짝수



홀수


M (18 , 26)*****

3 - (1) 쿼드 트리 정수 나눗셈 오차 문제

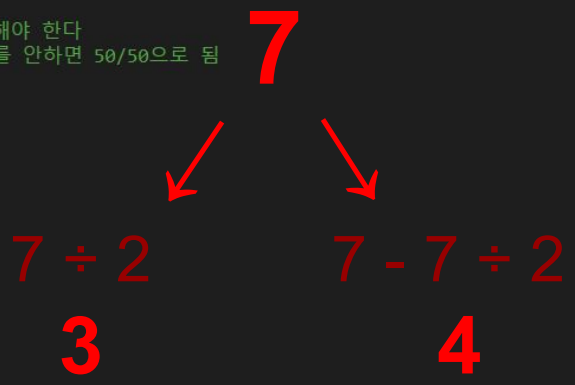
해결

```
// 영역 계산을 위한 변수
int x = bounds.GetX();
int y = bounds.GetY();
int width = bounds.GetWidth();
int height = bounds.GetHeight();
int halfWidth = width / 2;
int halfHeight = height / 2;

// 현재 영역이 정수로만 처리되어서 분할 시 경계선이 어긋날 수 있어서 주의해야 한다
// ex. 너비가 101인 영역을 이등분하면 50/51, 인 영역이 나와야하지만 처리를 안하면 50/50으로 됨

// 분할 4분면 객체 생성
// 4분면 분할 객체 생성
/*                          정수 나누셈에 유의!!
+------------+-------------+
|    0 (TL)  |    1 (TR)   | 1 TR, 3 BR
|            |             | 너비: width - halfWidth
+------------+-------------+
|    2 (BL)  |    3 (BR)   |
|            |             | 2 BL, 3 BR
|            |             | 높이: height - halfHeight
+------------+-------------+
*/

// 0
topLeft = new QNode(Bounds(x, y, halfWidth, halfHeight), depth + 1);
// 1
topRight = new QNode(Bounds(x + halfWidth, y, width - halfWidth, halfHeight), depth + 1);
// 2
bottomLeft = new QNode(Bounds(x, y + halfHeight, halfWidth, height - halfHeight), depth + 1);
// 3
bottomRight = new QNode(Bounds(x + halfWidth, y + halfHeight, width - halfWidth, height - halfHeight), depth + 1);
```



3 - (2) A* 최적화 문제

문제

- A* 로 경로를 찾는 유닛이 많아지면 프레임이 급격히 떨어짐

3 - (2) A* 최적화 문제

해결

- 로직 최적화(자료구조로 변경, 부모포인터 대신 인덱스로, 동적 할당 없이 처리 등)
- 유닛이 길찾기를 시도 하는 횟수를 3번으로 제한
- 유닛이 길찾기를 직접 요청하는 대신 중간 매니저 클래스에게 요청한 뒤 여러 프레임 나눠서 처리 (요청 도중 같은 유닛이 새로운 경로를 요청하거나, 유닛이 삭제될때 요청 대기열도 같이 삭제 처리하도록 유의)

3 - (3) 기타 여러가지 실수

```c
for (NodeIndex& index : quads)
{
  if (index == NodeIndex::TopLeft)
  {
    topLeft->Query(queryBounds, possibleNodes);
  }
  else if (index == NodeIndex::TopRight)
  {
    topRight->Query(queryBounds, possibleNodes);
  }
  else if (index == NodeIndex::BottomLeft)
  {
    bottomLeft->Query(queryBounds, possibleNodes);
  }
  else if (index == NodeIndex::BottomLeft) // ❌ 여기 문제
  {
    // ❌ 여기 문제
    bottomLeft->Query(queryBounds, possibleNodes);
  }
}
```

```c
for (NodeIndex& index : quads)
{
  if (index == NodeIndex::TopLeft)
  {
    topLeft->Query(queryBounds, possibleNodes);
  }
  else if (index == NodeIndex::TopRight)
  {
    topRight->Query(queryBounds, possibleNodes);
  }
  else if (index == NodeIndex::BottomLeft)
  {
    bottomLeft->Query(queryBounds, possibleNodes);
  }
  else if (index == NodeIndex::BottomRight) // ✅ 수정
  {
    // ✅ 수정
    bottomRight->Query(queryBounds, possibleNodes);
  }
}
```

3 - (3) 기타 여러가지 실수

```
// 속도 계수 테이블
const float speedMultipliers[5] =
{
  1.0f, // Grass
  1.5f, // Road
  0.5f, // Swamp
  0.7f  // Forest <---------------- 쉼표 없는데도 빌드되었음
  - 1.f  // Mountain
};
```

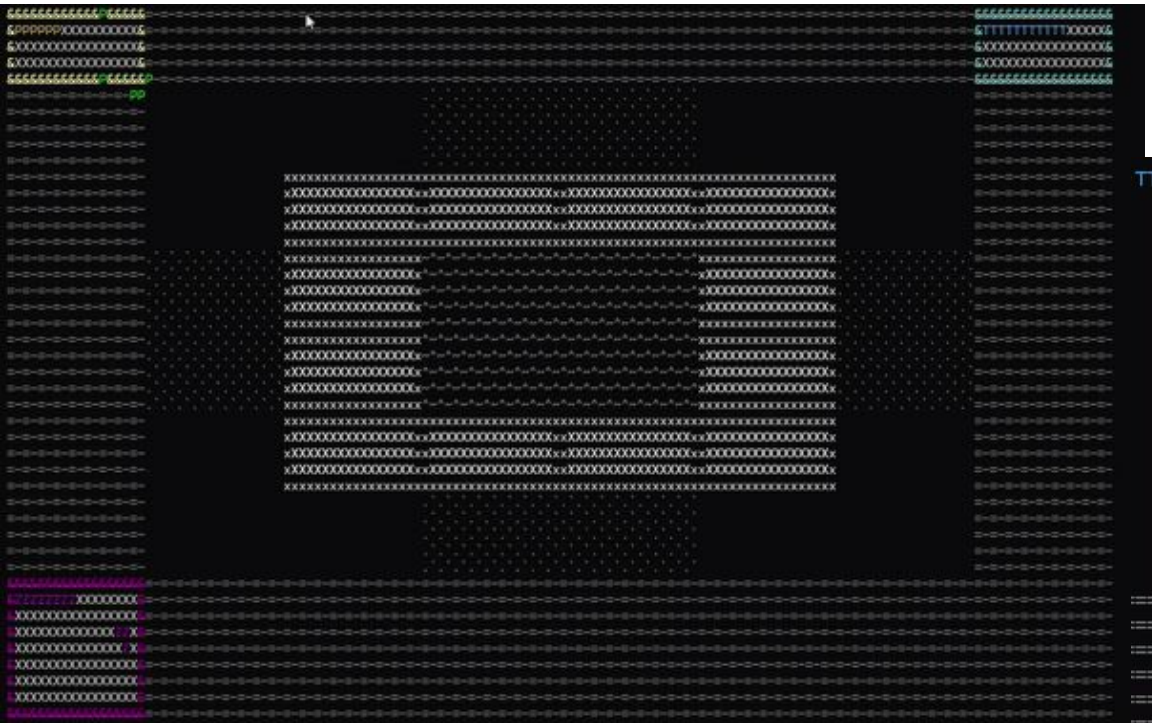
```
  }

ublic: // MESSAGE

  float FCost() { return hCost + hCost; }

ublic: // DATA
```

3 - (3) 기타 여러가지 실수



4. 느낀 점

- 막연했던 쿼드 트리와 A* 에 대한 응용력과 최적화에 대한 자신감
- 스마트 포인터의 필요성

  (날포인터를 쓰면서 댕글링 참조 문제를 많이 겪음)
- 스타크래프트1 유닛이 왜 그렇게 안 모이던지 이해 할 수 있었다

  (스타 잘 못하고 봇전에서 심시티만 했음)

감사합니다.

궁금한 점이 있으면 언제든지 질문주세요!