쿼드트리

1. 왜 쿼드트리를 사용했는가?

포트폴리오에서 언급했는데 이전의 시야처리 방식이 마음에 들지 않아서 사용하게 되었습니다.

이전에는 맵을 정적으로 나누어 관리 했고 최대 4개 영역에 대해서만 시야 검사를 하도록 구현했었는데 최대 4개 영역에 대한 처리만 해놓다 보니 시야범위를 크게 수정하려면 더 검사하는 영역 개수의 최대치를 정해 놓지 않고 상황에 맞게 작동하도록 구현하거나 격자의 크기를 바꾸는 등 코드 수정에 어려움이 있을 것이라고 생각했고 방금 말씀드린 방식으로 코드를 바꾼다고 하더라도 객체가 없는 큰 영역을 여러 개로 나누어 시야처리를 진행해야 한다는 점이 비효율 적이라고 생각해서 시야처리방식을 바꾸려고 생각했고 그래서 동적으로 공간을 분할하면서 객체가 존재하지 않는 영역을 나누지 않아 비효율적인 연산을 줄일 수 있는 쿼드트리를 사용하게 되었습니다.

* 옥트리나 KD트리에 대해서 아는가?

방식을 고민하던 시기에 후보군에 있었던 알고리즘인데 KD트리 같은 경우에는 검색 필드 값이 균등하지 않을 경우 성능이 떨어지는데 몬스터가 있는 구역이 따로 정해져 있었고 해당 지역에 플레이어가 몰릴 것이라고 생각해서 크게 고려하지 않았었고

방식을 고민하던 시기에는 맵에 높낮이가 크지 않은 평탄한 지형을 사용하고 있었고 맵이 바뀌는 일이 없다고 생각해서 쿼드트리를 채택하게 되었습니다만 개발 후기에 높낮이가 큰 지형으로 바뀌게 되어 아쉽게 생각합니다.

1. 가장 어려웠던 점은 무엇 인가?

최적화를 어떻게 해야 하는지가 가장 어려웠었는데 이 부분은 성능 프로파일러를 통해서 시야내의 객체를 검색하는 Search함수에서 가장 부하가 크다는 것을 알아냈고 여러 상황에 대해 테스트 해보며 시야에 완전 포함되는 노드에 대해서 추가적인 시야포함 여부를 판단하지 않게 바꿨고 애초에 Search함수를 최대한 적게 사용하기 위해서 플레이어마다 각자의 시야범위 안에 있는 객체의 ID를 저장해놓고 사용할 수 있는 View List를 따로 만들어서 좌표가 변한 이후에만 Search 함수를 호출하여 사용하도록 만들었고

트리에 객체를 업데이트 해주는 방식도 매프레임 해당 객체를 삽입/삭제 해주는 방식에서 객체가 속해 있는 노드의 범위를 저장해 놓고 이동 시 해당 노드의 범위를 벗어날 때만 업데이트 해주는 방식으로 바꿨습니다.

1. 노드를 나눌 때 부모 노드에 속한 객체를 자식 노드에 뿌려주는 이유

처음에 쿼드트리를 구현할 때는 부모 노드에 속한 객체를 나눠 주지 않는 방식으로 구현을 했었는데 이 방식은 시야에 포함되는 모든 부모 노드들까지 시야에 대한 검색을 해줘야 하는게 비효율적이라고 생각해서 나눠질 때 부모 노드에 속한 객체를 내려주고 Search함수에서 자식이 있는 노드라면 다른 연산 없이 자식 노드들의 Search함수를 호출해라 라는 식으로 구현하였습니다.

* 만약에 이해 못하시면

개발 다 끝나고 알게 된 부분이 보통 사용할 때 런타임에 쪼개는 방식보다 컴파일 타임에 미리 쪼개 놓고 사용한다고 알게 되었는데 구현 당시엔 이런 부분을 잘 몰라서 런타임에 계속 쪼개서 사용하는 방식이라 그런 것 같습니다.

1. 성능 향상은 봤는가?

성능에 대한 부분을 확인 하지 못한게 졸업작품 프로젝트에서는 시야처리 개발 초기부터 쿼드트리를 사용해서 다른 방식과의 차이를 확인하지 못했고 중간중간 더미클라이언트를 사용한 성능 테스트를 해봤는데 최적화가 하나도 안되어 있을 당시에 1초에 한번씩 이동과 정지 패킷을 랜덤으로 보내는 환경에서는 400~450사이 였고 모든 최적화를 끝낸 후에는 2000이상부터 랜덤으로 서버가 다운되어서 최대치를 특정하지 못했고 서버가 다운되는 경우에 대해서 수정을 확실히 하지 못해서 성능향상에 대한 부분은 확실하게 말씀드릴 수 없을 것 같습니다.