IsInCrossfireSituation()

이 함수는 게임 AI에서 봇이 "교차 사격 상황"에 놓였는지를 판별합니다. 교차 사격 상황은 봇이 두 다른 봇 사이의 직선 경로 근처에 있어 공격받을 위험이 높은 상태를 의미합니다.

1. **목표 확인**: 봇이 목표(Target)가 없으면 교차 사격 여부를 판단하지 않고 바로 false를 반환합니다.
2. **가장 가까운 두 봇 찾기**:
    - 자신의 월드 내 모든 봇을 순회하면서, 자신과 죽은 봇을 제외한 나머지 봇 중에서 가장 가까운 두 봇(bot1, bot2)을 찾습니다.
3. **조건 확인**: 가까운 봇이 두 명 이상 존재하지 않으면 교차 사격 상황이 아니므로 false를 반환합니다.
4. **위치 계산**:
    - bot1과 bot2의 위치를 이용해 두 봇을 잇는 직선(선분)을 정의합니다.
    - 현재 봇의 위치가 이 선분의 연장선 밖에 있는지 확인합니다. 연장선 밖이라면 교차 사격 상황이 아니라고 판단합니다.
5. **선분과 봇 사이 거리 계산**:
    - 현재 봇이 두 봇을 잇는 선분으로부터 얼마나 떨어져 있는지(수직 거리)를 계산합니다.
6. **결론**:
    - 이 거리가 설정된 임계값(threshold) 이하라면 교차 사격 상황이라고 판단하고 true를 반환합니다. 그렇지 않으면 false를 반환합니다.

GetCrossfireDangerLevel()

이 함수는 현재 봇이 교차 사격 상황에서 얼마나 위험한지를 "위험 수준"으로 반환합니다. 위험 수준은 0.0에서 1.0 사이의 값으로 표현되며, 1.0에 가까울수록 위험도가 높음을 의미합니다. 이를 단계적으로 설명하겠습니다:

1. 가장 가까운 두 봇 찾기

- 모든 봇을 순회하면서 자신과 죽은 봇을 제외한 나머지 봇 중 가장 가까운 두 봇(bot1과 bot2)을 찾습니다.
- 두 봇까지의 거리(제곱 값)를 계산하여 가장 작은 두 값을 갱신하며, 해당 봇들을 저장합니다.

2. 조건 확인

- 만약 가까운 두 봇(bot1, bot2)이 존재하지 않으면 위험 상황이 아니라고 간주하고 0.0을 반환합니다.

3. 두 봇 간 거리 계산

- bot1과 bot2 사이의 실제 거리(distanceBetweenBots)를 계산합니다.
- 이 거리는 두 봇 사이의 공간적인 관계를 측정하는 데 사용됩니다.

4. 현재 봇과 두 봇 사이의 중점 거리 계산

- midpoint: bot1과 bot2의 중점 좌표를 계산합니다.
- distanceToMidpoint: 현재 봇이 이 중점까지 얼마나 떨어져 있는지를 계산합니다.

5. 위험 수준 계산

- **근접 점수(proximityScore)**:
    - 1.0 - (distanceToMidpoint / (distanceBetweenBots / 2.0))로 계산됩니다.
    - 현재 봇이 두 봇의 중점에 가까울수록 점수가 높아집니다.
    - distanceToMidpoint가 중점에서 두 봇 사이의 반지름(distanceBetweenBots / 2)보다 크면, 점수는 음수 또는 0에 가까워지게 됩니다.

6. 점수 범위 조정

- Clamp 함수를 사용하여 점수를 0.0에서 1.0 사이로 제한합니다.

- proximityScore가 0.0보다 작으면 0.0으로, 1.0보다 크면 1.0으로 설정됩니다.

7. 결과 반환

- 최종적으로 위험 수준을 나타내는 점수(proximityScore)를 반환합니다.
- 이 값이 높을수록 봇이 교차 사격 상황에서 위험한 위치에 있음을 나타냅니다.

결론

이 함수는 교차 사격 위험도를 정량적으로 평가하기 위해:

1. 두 봇의 중점과 자신의 위치를 비교하여 얼마나 가까운지를 계산하고,
2. 두 봇 사이의 거리로 정규화하여 근접 점수를 산출합니다.
3. 이 값은 AI의 전략적 판단(위험 회피, 위치 변경 등)에 활용될 수 있습니다.

CalculateSafePosition()

이 함수는 현재 봇이 교차 사격 상황에서 벗어나기 위한 안전한 위치를 계산합니다. 안전한 위치는 봇이 두 다른 봇 사이의 직선 경로에서 벗어나도록 설정됩니다. 아래는 함수의 주요 흐름과 로직을 설명합니다:

1. 가장 가까운 두 봇 찾기

- 월드에 존재하는 모든 봇을 순회하면서 자신과 죽은 봇을 제외한 나머지 봇 중 가장 가까운 두 봇(bot1, bot2)을 찾습니다.
- minDistance1과 minDistance2를 사용해 가장 가까운 봇과 두 번째로 가까운 봇을 선택합니다.

2. 예외 처리

- 만약 가까운 두 봇이 없다면 안전한 위치를 계산할 필요가 없으므로, 현재 위치를 그대로 반환합니다.

3. 두 봇을 연결하는 직선 계산

- A와 B: bot1과 bot2의 위치입니다.
- AB: 두 봇을 연결하는 단위 벡터입니다(Vec2DNormalize를 사용하여 길이가 1인 방향 벡터로 정규화).

4. 현재 위치의 투영점 계산

- AP: 현재 봇에서 첫 번째 봇(A)으로 향하는 벡터입니다.
- projectionLength: AP를 AB 방향으로 투영한 길이입니다.
- closestPoint: 현재 봇에서 두 봇을 연결하는 직선(선분) 상의 가장 가까운 점입니다.

5. 직선에 수직한 방향 계산

- perpendicular: AB에 수직한 벡터입니다. (-AB.y, AB.x 형태로 계산)
- escapeDistance: 교차 사격 상황에서 벗어나기 위한 거리(10.0)입니다.
- potentialPosition1과 potentialPosition2: 수직 방향으로 escapeDistance만큼 떨어진 두 위치를 계산합니다.

6. 안전한 위치 선택

- **맵 유효성 검사**:
    - map->IsValidPosition: 해당 위치가 맵에서 유효한지 확인합니다.
- **가장 멀리 있는 위치 선택**:
    - 두 위치 중에서 현재 위치에서 더 멀리 떨어진 위치를 우선적으로 선택합니다.

7. 반환 값

- 조건에 따라 유효한 위치가 있다면 해당 위치를 반환합니다.
- 유효한 위치가 없거나 계산이 불가능한 경우, 현재 위치를 그대로 반환합니다.

결론

이 함수는:

1. 두 가까운 봇의 위치를 기준으로 교차 사격 위험에서 벗어날 수 있는 두 가지 가능한 위치를 계산합니다.
2. 맵 내에서 유효하며 현재 위치에서 가장 안전한 위치를 선택합니다.
3. 이 로직은 봇의 생존 가능성을 높이는 AI 행동 전략에 활용될 수 있습니다.