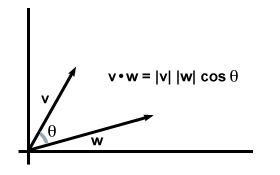
[게임수학 | 유니티] 벡터의 내적을 이용한 시야각 구현하기  
  
**1. 벡터의 내적이란?**



두 벡터의 내적은 아래 공식으로 표현된다.

a벡터와 b벡터의 크기를 각각 곱한 다음 사이각의 cosθ 값을 곱한다.

벡터와 벡터의 내적의 결과는 벡터가 아닌 스칼라 값임에 주의한다.

**자기 자신과 내적하면 제곱이다.**  
cosθ 값이 자기 자신이기 때문에 1이 된다. 결과적으로 같은 벡터 2개를 내적하면 제곱이 된다.

**두 단위벡터가 평행하면 절대값 1이다.**  
벡터 두개가 평행하는 경우는 같은 방향으로 향하거나, 반대 방향으로 향하는 것이다. 따라서 cosθ 값이 1 혹은 -1 이다. 절대값을 취하면 1이된다.

**두 벡터가 직교하는 경우 값이 0이 된다.**

cos90˚는 0이 되기 때문에 내적값 또한 0이된다.

**2. 유니티에서 벡터의 내적을 이용한 시야각 구현하기**

플레이어의 시야각을 θ라고 하면,  Forward 단위벡터와 타겟과 플레이어의 거리 차이로 나오는 단위벡터A 간의 내적이 cos(θ/2)보다 커야 시야 내에 존재한다.



이미지 출처: http://rapapa.net/?p=2974

**※ 프로젝트 파일 필요하신 분은 댓글로 메일주소 남겨주세요.**

|  |
| --- |
| using *System*.*Collections*.*Generic*;  using *UnityEngine*;    public class SightCtrl : *MonoBehaviour*  {  public float ViewAngle; //시야각  public float ViewDistance; //시야거리  public *LayerMask* TargetMask; //Enemy 레이어마스크 지정을 위한 변수  public *LayerMask* ObstacleMask; //Obstacle 레이어마스크 지정 위한 변수  private *Transform* \_transform;  void Awake()  {  \_transform = *GetComponent*<*Transform*>();  }  void Update()  {  DrawView(); //Scene뷰에 시야범위 그리기  FindVisibleTargets(); //Enemy인지 Obstacle인지 판별  }  public *Vector3* DirFromAngle(float angleInDegrees)  {  //탱크의 좌우 회전값 갱신  angleInDegrees += *transform*.*eulerAngles*.*y*;  //경계 벡터값 반환  return new *Vector3*(*Mathf*.*Sin*(angleInDegrees \* *Mathf*.*Deg2Rad*), 0, *Mathf*.*Cos*(angleInDegrees \* *Mathf*.*Deg2Rad*));  }  public void DrawView()  {  *Vector3* leftBoundary = DirFromAngle(-ViewAngle / 2);  *Vector3* rightBoundary = DirFromAngle(ViewAngle / 2);  *Debug*.*DrawLine*(\_transform.*position*, \_transform.*position* + leftBoundary \* ViewDistance, *Color*.*blue*);  *Debug*.*DrawLine*(\_transform.*position*, \_transform.*position* + rightBoundary \* ViewDistance, *Color*.*blue*);  }  public void FindVisibleTargets()  {  //시야거리 내에 존재하는 모든 컬라이더 받아오기  *Collider*[] targets = *Physics*.*OverlapSphere*(\_transform.*position*, ViewDistance, TargetMask);  for (int i = 0; i < targets.*Length*; i++)  {  *Transform* target = targets[i].*transform*;  //탱크로부터 타겟까지의 단위벡터  *Vector3* dirToTarget = (target.*position* - \_transform.*position*).*normalized*;  //\_transform.forward와 dirToTarget은 모두 단위벡터이므로 내적값은 두 벡터가 이루는 각의 Cos값과 같다.  //내적값이 시야각/2의 Cos값보다 크면 시야에 들어온 것이다.  if (*Vector3*.*Dot*(\_transform.*forward*, dirToTarget) > *Mathf*.*Cos*((ViewAngle / 2) \* *Mathf*.*Deg2Rad*))  //if (Vector3.Angle(\_transform.forward, dirToTarget) < ViewAngle/2)  {  float distToTarget = *Vector3*.*Distance*(\_transform.*position*, target.*position*);  if (!*Physics*.*Raycast*(\_transform.*position*, dirToTarget, distToTarget, ObstacleMask))  {  *Debug*.*DrawLine*(\_transform.*position*, target.*position*, *Color*.*red*);  }  }  }  }  } |