# AI

## AvoidFire()

通过判断CellType是否处于Fire状态，调用DivertPath方法来绕开有火的地块。

## AvoidOtherActors & DivertAroundObjectsOnLayer(Layer)

raycastHit:射线检测，Physics2D.Raycast

如果射线检测到的物体，在主角的视线范围内（IsInLineOfSight），则通过DivertPath方法来绕开物体。

## Awake()

初始化AI的参数。

## CanISeeTarget(targetPlayer)

如果是变态难度(impossible)直接返回true。

如果玩家在伪装状态或在遮挡物中(IsDisguised || IsInTheShadow)：

如果玩家手持金色回旋镖，返回true。(**targetPlayer**.velocityStored.magnitude > 6f - botDifficulty \* 2f

伪装时间小于某个参数（Player.disguiseStartDuration \* (1f + botDifficulty) \* 1.5f）时，返回true。

## CreatePath()

创建路径。

## Dash(dashDirection, avoidObstacles, allowOutOfBounds, allowHazardInFrontOfEndPoint, trackFailedAttempts)

如果满足安全跳跃条件(IsDashSafe)，则执行跳跃操作

IsDashSafe：

## DestroyPath()

到达路径终点=true；path=null。

## DivertPath(avoidPosition)

以回避点为基准随机旋转一个角度（40到70度之间），并行走一个随机距离（3到6米）。

## GetCurrentWaypointPosition()

获得当前路点。

## GetFinalWaypointPosition()

获得最终路点。

## GetRandomReactionTime

随机反应时间：[0, reactionTimes]。

## IsPositionCloseToCurrentWaypoint()

位置是否在当前路点上，如果在湿地上，距离判断x2。

## IsPositionHiddenByPlants(pos, colliderRadius)

判断位置是否隐藏在植物中。

## IsPositionTooFarFromCurrentWaypoint()

判断角色当前位置是否距离路点过远。

this.currentWaypointTooFarDistance

## IsTargetHiddenByPlants(Player targetPlayer)

调用IsPositionHiddenByPlants，判断目标玩家是否隐藏在植物中。

## IsTargetPathable()

目标是否能够寻路到达

this.path != null && (this.GetFinalWaypointPosition() - this.targetPosition).magnitude <= 2f;

## LocateTarget()

返回目标位置。

stateChangeReluctance置为默认值。

扫描兴趣点。

考虑改变状态。

## MoveOnPath(AI.PathEndDelegate OnEndReached = null)

## NearbyPlant(pos)

## OnPathComplete（Path p）

## PushControllerTowardsNextWaypoint()

向当前路点（GetCurrentWaypointPosition）移动。

## ReactToTeleportal()

## ScanAsDesirable(GameObject, AI.DesirableType)

用GetInterestOfDesire方法获得interestLevel。

如果interestLevel大于当前最大欲求值（bestDesireSoFar），则把扫描目标置为欲求目标。

当前最大欲求值 = interestLevel。

this.totalDesireLevel += this.interestLevel;

## ScanAsThreat(GameObject, AI.ThreatType)

与ScanAsDesirable类似。

## ScanAsVictim(Player otherPlayer)

与ScanAsDesirable类似。

## ScanPointsOfInterest()

此处主要用于识别火。

ThreatType: Explosive, Fire

参数枚举：

this.bestDesireSoFar = 0f;  
     this.bestVictimSoFar = 0f;  
     this.bestThreatSoFar = 0f;  
     this.interestLevel = 0f;  
     this.totalDesireLevel = 0f;  
     this.totalThreatLevel = 0f;  
     this.totalVictimLevel = 0f;

## SetRandomTargetPos()

在场景里以场景长度一半加减2米的范围内，随机一个向量，然后随机旋转一个角度，只要不是在后方，赋值给targetPosition。

## SetRandomTargetPosOnMovingFloor(MovingFloor mf)

在移动地板的边界内，随机选择点生成一个向量。

## SetStateChangeReluctance(float changeReluctance, float duration)

设置两个参数：

this.stateChangeReluctance = **changeReluctance**;  
this.stateChangeReluctanceCooldown = **duration**;

## Update()

gameState = Playing, 就执行UpdateBehaviour();

## UpdateBehaviour()

scanTimer：扫描计时器。

scanDelay：扫描延迟，通过GetRandomReactionTime()获取，困难难度的机器人x3

actionTimer：行动计时器，大于行动延迟时，执行TakeAction()方法。

actionDelay：行动延迟。

createPathTimer：创建路径计时器。

stateChangeReluctanceCooldown：状态切换阻力CD。

inputTimer：输入操作计时器。

inputDelay：输入操作延迟。

# PlayerAI

## AimDisc(float holdTime = 1f)

长按投掷按钮操作。

## ApproachTargetFromNewAngle()

在（-5,5）范围内选择随机向量，置为targetPosition。

将状态置为Diverting。

执行ConsiderChangingState();

## AreRetrievingDiscsStuck()

是否回收回旋镖被阻挡。

## Attack()

根据AI难度进行攻击概率判断：简单难度=90%。

如果玩家没有回旋镖，概率变为十分之一。

如果有两个回旋镖且没持有金色回旋镖，70%概率投掷回旋镖。

如果攻击进步范围内不安全，且victimTarget不为空，则对victimTarget投掷回旋镖。

其余情况，点击攻击按钮。

## Attacking\_Enter()

如果处于收回回旋镖的状态中，则停止收回行为。

定位目标点，创建路径，寻路。

修改StateChangeReluctance（1.5， 2.5）

## Attacking\_Exit()

似乎没啥用

## Attacking\_Update()

如果victimTarget不为空，则把victimTarget置为当前目标，并且追击；否则进入休闲状态。

如果victimTarget在草丛中，则选择靠近草丛的点行进。

## BestObjectToAttack()

针对玩家的判定条件集合（各条件之间为且关系）：

* 不在金色回旋镖的模式中 或 目标玩家没有手持金色回旋镖 或 自己没有手持金色回旋镖 或66%概率
* 目标玩家在攻击射程以内
* 目标玩家在前方一定角度内
* 目标玩家处于存活状态
* 目标玩家是敌方玩家
* 能看见目标玩家
* 目标玩家在视线内 或 40%概率

针对开关的判定条件组合（各条件之间为且关系）：

* 开关在攻击射程\*0.75范围内
* 开关在前方一定角度内（要求比玩家判定要高）
* 开关在玩家视线内
* 开关处于激活状态

## BestObjectToBlock()

blockRadiusPlayers = 2f;

blockRadiusDiscs = 10f;

对场景中每个玩家进行判定，条件集合如下：

* 目标玩家在玩家的blockRadiusPlayers以内
* 目标玩家处于存活状态
* 目标玩家是敌对玩家
* 目标玩家在视线内
* 能看见目标玩家

对场景中每个回旋镖进行判定，条件集合如下：

* 回旋镖在玩家的blockRadiusDiscs以内
* 回旋镖归属不为空
* 回旋镖不属于玩家
* 回旋镖不处于掉落Dropped状态
* 回旋镖不处于挤压Squished状态
* 回旋镖处于玩家的敌对状态
* 回旋镖在视线内
* 玩家在回旋镖速度方向的一定角度内 或 回旋镖处于Controlled状态

## BestObjectToThrowAt()

判定条件集合：

* 不在金色回旋镖的模式中 或 目标玩家没有手持金色回旋镖 或 自己没有手持金色回旋镖 或66%概率
* 目标玩家距离在投掷射程内（根据是否蓄力会有不同的投掷射程）
* 目标玩家在前方一定角度内（根据不同难度，角度范围会有所不同）
* 目标玩家处于存活状态
* 目标玩家是敌方玩家
* 能看见目标玩家
* 目标玩家的投掷预判位置在视线内（调用TargetLocationAfterThrow方法）

当玩家拥有技能（爆炸回旋镖、多个 回旋镖、TeleportDisc）或拥有回旋镖数量大于1时，角度要求更小（\*0.75）。

## Block(GameObject)

## Collecting\_Enter()

## Collecting\_Update()

## ConsiderChangingState()

教学阶段，有一个特殊处理，会调用FlickingSwitch状态。

最开始的一段时间（gettingBearingsTimer > 0.1f），处于Wandering状态。

this.gettingBearingsTimer = this.gettingBearingsDuration \* Random.Range(0.75f, 1f);

gettingBearingsDuration在设定AI难度的时候初始化，与AI难度相关。

进入Collecting状态的条件集合（各条件之间为且关系）：

* 当前最佳期待值（bestDesireSoFar） > 当前最佳攻击值（bestVictimSoFar）
* 当前最佳期待值（bestDesireSoFar） > 当前最佳威胁值（bestThreatSoFar）
* 当前最佳期待值（bestDesireSoFar） > 环境兴趣值（ambientInterestLevel）
* 当前最佳期待值（bestDesireSoFar） > 状态改变阻力值（stateChangeReluctance）
* 剩余敌人数量 > 1
* 期待目标（desireTarget）不为空
* 期待目标（desireTarget）和角色的距离大于下个路点的距离（nextWaypointDistance）

以上条件满足后，把当前目标置为期待目标（desireTarget），并且角色 进入Collecting状态。

进入Retrieving状态的条件集合（各条件之间为且关系）：

* 所有Collecting状态的条件都需要满足
* 上一个状态（fsm.LastState）不是Retrieving
* 角色不拥有爆炸回旋镖技能
* 角色在安全区内
* 角色手中没有回旋镖
* 角色扔出去的回旋镖数量 > 0
* 角色当前目标为回旋镖
* 角色当前目标在角色的直线视线内（IsInLineOfSight）
* 以下两个条件满足一个 ：
  + 目标和角色之间的距离 > 角色跳跃距离的2倍 且 以下两个条件满足一个：
    - 简单难度，随机10%概率，否则随机33%概率
    - 目标不在安全区内
  + 目标所在位置跳跃不安全
* 当前最佳威胁值 < 当前最佳期待值 \* 0.5

进入Attacking状态的条件集合（各条件之间为且关系）：

* 当前最佳攻击值（bestVictimSoFar）> 当前最佳期待值（bestDesireSoFar）
* 当前最佳攻击值（bestVictimSoFar）> 当前最佳威胁值（bestThreatSoFar）
* 当前最佳攻击值（bestVictimSoFar）> 环境兴趣值（ambientInterestLevel）
* 当前最佳攻击值（bestVictimSoFar）> 状态改变阻力值（stateChangeReluctance）

如果满足上述条件集合，则把当前目标置为攻击目标（victimTarget），并且进入Attacking状态。

如果攻击目标处于草丛隐藏状态，则先执行靠近靠近草丛行为，再进入Attacking状态。

进入Fleeing状态的条件集合（各条件之间为且关系）：

* 当前最佳威胁值（bestThreatSoFar）> 当前最佳期待值（bestDesireSoFar）
* 当前最佳威胁值（bestThreatSoFar）> 当前最佳攻击值（bestVictimSoFar）
* 当前最佳威胁值（bestThreatSoFar）> 环境兴趣值（ambientInterestLevel）
* 当前最佳威胁值（bestThreatSoFar）> 状态改变阻力值（stateChangeReluctance）

如果满足上述条件集合，并且角色当前状态不处于Fleeing状态，或者随机33%的概率成功，则把当前目标置为威胁目标（threatTarget），并且进入Fleeing状态。

以上情况都不满足时，如果环境兴趣值（ambientInterestLevel）> 状态改变阻力值（stateChangeReluctance），则角色进入Wandering状态。

## Dash(Vector2 dashDirection, bool avoidObstacles = true, bool allowOutOfBounds = true, bool allowHazardInFrontOfEndPoint = true, bool trackFailedAttempts = true)

## DashAcrossHazards()

## DashTowardsDroppedDisc()

## Diverting\_Enter()

this.CreatePath();  
base.SetStateChangeReluctance(2f, 1f);

## Diverting\_Update()

## DivertPathAfterThrow()

## FindSafePlaceToDash()

## Fleeing\_Enter()

停止飞镖蓄力

定位目标，创建路径。

把状态改变阻力值（stateChangeReluctance）设置为1.5，持续1秒。

## Fleeing\_Update()

        if (this.reachedEndOfPath)  
        {  
            this.ScanPointsOfInterest();  
            this.ConsiderChangingState();  
            this.LocateTarget();  
            this.CreatePath();  
        }  
        this.MoveOnPath(new AI.PathEndDelegate(this.DashAcrossHazards));

## FlickingSwitch\_Enter()

## FlickingSwitch\_Update()

## GetInterestOfDesire(GameObject desirable, AI.DesirableType desirableType)

基础系数 = Min(1, Max(1.25 – 期待目标和角色距离 /场景大小， 0))

对于技能道具：

如果是Debuff类（目前只有操作反向），则期待值 = 0

其他技能道具，期待值 = 基础系数 \* 1.5 \* greedyMod

**以下都针对回旋镖**

如果期待目标是金色回旋镖：

期待值 = 100

如果期待目标是有爆炸技能的回旋镖 或 正在燃烧的回旋镖 或 处于危险位置没有拥有者的回旋镖：

期待值 = 0

对场景内每个回旋镖进行检测，如果玩家拥有回旋镖数量 + 场景中飞行的回旋镖数量大于1：

期待值 = 0

如果期待目标是回旋镖且回旋镖的拥有者是该角色：

则增益系数 = 4

期待值 = 基础系数 \* 增益系数 \* greedyMod

如果回旋镖没有拥有者 或 游戏模式允许抢夺回旋镖，并且玩家处于解除武装状态：

则增益系数 = 4

期待值 = 基础系数 \* 增益系数 \* greedyMod

如果回旋镖没有拥有者 或 游戏模式允许抢夺回旋镖，并且期待目标不在警戒线范围内：

则增益系数 = 0.5

期待值 = 基础系数 \* 增益系数 \* greedyMod

如果期待目标x坐标大于40 或 y坐标大于30：

期待值 = 0

如果回旋镖没有拥有者 或 游戏模式允许抢夺回旋镖，并且拥有金色回旋镖（持有或扔出）：

则增益系数 = 0.5

期待值 = 基础系数 \* 增益系数 \* greedyMod

对于回旋镖没有拥有者 或 游戏模式允许抢夺回旋镖的其他情况：

增益系数 = 3

期待值 = 基础系数 \* 增益系数 \* greedyMod

如果回旋镖有拥有者且拥有者死亡：

期待值 = 0

如果回旋镖有拥有者且回旋镖与角色是敌对状态：

期待值 = 0

## GetInterestOfThreat(GameObject threat, AI.ThreatType threatType)

**以下是威胁目标为玩家的情况**

如果威胁目标处于解除武装状态或被冰冻状态：

系数1 = 0

如果角色处于解除武装状态：

系数1 = 5.25

如果角色拥有回旋镖数量 = 0：

系数1 = 2

如果威胁目标拥有的回旋镖数量 小于 角色的回旋镖数量：

系数1 = 0.8

如果威胁目标拥有的回旋镖数量 等于 角色的回旋镖数量：

系数1 = 1

如果威胁目标处于着火状态并且角色不处于着火状态：

系数2 = 5

其余情况：

系数2 = 1

系数3 = 0.45 – 0.05 \* cos(威胁目标和角色正前方夹角)

如果角色能够看见威胁目标：

系数4 = 1

如果角色不能看见威胁目标：

系数4 = 0.5

如果威胁目标在角色的直线视线内：

系数5 = 1

如果威胁目标不在角色的直线视线内：

系数5 = 0.6

如果威胁目标正在瞄准 并且 cos(威胁目标正前方和角色夹角) > 0.25：

系数6 = 2

否则：

系数6 = 1

系数7 = Max(1 – 威胁目标和角色距离 /场景大小 \* 2, 0)

威胁值 = 系数1 \* 系数2 \* 系数3 \* 系数4 \* 系数5 \* 系数6 \* 系数7 \* nervousMod

**以下是威胁目标为回旋镖(Explosive)的情况**

如果威胁目标在角色的直线视线内并且威胁目标与角色距离 < 17：

威胁值 = 6 \* nervousMod

其余情况：

威胁值 = 1 \* nervousMod

**以下是威胁目标为火焰的情况**

系数8 = Max(1 – 威胁目标和角色距离 /场景大小 \* 2, 0)

如果系数8 < 0.5：

威胁值 = 0

其余情况：

威胁值 = 系数8 \* (0.5 – 0.5 \* cos(威胁目标和角色正前方夹角)) \* nervousMod

## GetInterestOfVictim(Player otherPlayer)

如果角色处于解除武装状态：

攻击值 = 0

对于金色回旋镖模式，如果攻击目标拥有金色回旋镖：

攻击值 = 10

对于金色回旋镖模式，如果角色拥有金色回旋镖并且胜利条件为击杀：

系数1 = 2

对于金色回旋镖模式，如果角色不拥有金色回旋镖：

系数1 = 0.5

对于其他游戏模式：

系数1 = 1

如果攻击目标处于解除武装状态：

系数2 = 1.2

如果攻击目标拥有的回旋镖数量小于或等于角色拥有的回旋镖数量：

系数2 = 1

其余情况：

系数2 = 0.8

系数3 = Min(cos(威胁目标和角色正前方夹角) \* 0.2 + 1, 1)

如果攻击目标在角色的直线视线内：

系数4 = 1.1

如果攻击目标不在角色的直线视线内：

系数4 = 1

如果能够看见攻击目标：

系数5 = 1

如果不能看见攻击目标：

系数5 = 0.25

系数6 = Min(Max(1.5 – 攻击目标与角色距离/场景大小, 0), 2)

如果攻击目标被冰冻：

系数7 = 1.3

如果攻击目标不被冰冻：

系数7 = 1

如果攻击目标是领先玩家：

系数8 = 2.5 - Min(剩余胜利分数 \* 0.2, 1.25)

如果攻击目标不是领先玩家：

系数8 = 1

攻击值 = 系数1 \* 系数2 \* 系数3 \* 系数4 \* 系数5 \* 系数6 \* 系数7 \* 系数8 \* aggressionMod

## GetRandomReactionTime()

## HoldButtonDown(PlayerAI.ButtonType buttonType, float duration)

## Idle\_Enter()

## Idle\_Update()

## Init()

AI初始化。

执行RandomiseInterestMods()

aggressionMod = 0

状态机初始化为Wandering状态。

unsafeDashAttempts = 0（如果这个值大于7，就执行FindSafePlaceToDash()）

## IsPositionOnTheHill(Vector3 pos)

## OnPathComplete(Path p)

## PushButtonDown(PlayerAI.ButtonType buttonType)

## PushControllerTowardsNextWaypoint()

## RandomiseInterestMods()

调整兴趣值相关的各个参数：

ambientInterestMod：环境兴趣值

greedyMod：对应期待值

nervousMod：对应威胁值

aggressionMod：对应攻击值

针对不同难度，设定不同的ambientInterestMod：

简单：random(1, 2)

中等：random(0.9, 1.5)

困难：random(0.8, 1.2)

变态难度：random(0.6, 1)

初始aggressionMod = random(0.6, 1)

如果难度大于中等难度：aggressionMod 增加0.3。

如果胜利条件时击杀并且角色拥有金色回旋镖：

aggressionMod = aggressionMod \* random(1, 2)

如果胜利条件是存活并且角色拥有金色回旋镖：

nervousMod = random(1, 2)

其余情况：

nervousMod = random(0.8, 1.2)

如果是简单难度：

nervousMod 增加0.3。

如果玩家拥有控制反向技能：

nervousMod = nervousMod \* 1.5

ambientInterestMod = ambientInterestMod \* 5

如果玩家处于Retrieving状态：

nervousMod = nervousMod \* 1.25

greedyMod初始值 = random(0.8, 1.2)

## ReleaseButtonUp(PlayerAI.ButtonType buttonType)

## RemoveComponent()

## Retrieving\_Enter()

## Retrieving\_Update()

## ScanPointsOfInterest()

对于场景中的每个玩家，如果处于存活状态并且是关系敌对：

则设定为攻击目标

且设定为威胁目标

对于场景中每个激活状态的回旋镖，

如果处于掉落状态：

则设定为期望目标。

如果处于有伤害状态或多回旋镖状态：

则设定为威胁目标

对于场景中每个技能道具，

设定为期望目标。

## SetDifficulty()

设定难度参数

reactionTimes：数组，有三个参数，在三个参数中随机选择一个，乘以与出生时间相关的系数，生成反应时间GetRandomReactionTime()。在AI类中，scanDelay和actionDelay都通过这个方式生成。

inputDelay：AI点击按钮的延迟时间。

gettingBearingsDuration：出生时间设定值，难度越高，数值越小。

gettingBearingsTimer：出生时间计时器，在UpdateBehaviour()中递减。

gettingBearingsSpeed：出生时间内的移动速度系数，50%概率=1, 50%概率=0.05。

## SetRandomTargetPos()

执行父类中的同名函数。

## StopHoldingThrow()

## TakeAction()

在父类的UpdateBehaviour()函数中每隔actionDelay时间调用一次TakeAction()函数。

## TargetLocationAfterThrow(GameObject go)

## Throw(GameObject go)

## TryToAttack()

## TryToHitSwitch()

## UpdateBehaviour()

## Wandering\_Enter()

## Wandering\_Update()