ARROW

STATE TO MAINTAIN:

Start Position (Current Rooms Position)
Targe Position (Next Room Position)
List of Rooms it will travel.
Arrow Count
WasFired (Boolean)

PRE-CONDITION

Will be provided the current room.
Will be provided list of rooms that arrow will travel.
There can be no more than 5 rooms traveled.
Start Position is current room's position.
Target Position will be next room's position.
Wumpus, Bat are not in current Room

FUNCTIONS

Bool Shoot()

If Arrow Count > 0
WasFired True;
Iterate through each room it will travel
If next room contains Wumpus
YOU WIN return true.
If next room contains Bat
Remove Bat
Decrease arrow count by 1
If next room is the current room
YOUR DEAD return false
If angle is less than 98 degrees
If there are still rooms to travel
Set Target to next room's position
Move to next room
Decrease arrow count by 1
Else
Decrease arrow count by 1
Else
It ricochets off the wall
Decrease arrow count by 1

Bool IsLessThan98()

X and Y are Rooms Coordinates. 1 = Current Room, 2 = Next Room

Delta X = $X_2 - X_1$
Delta Y = $Y_2 - Y_2$

Angle = Arctangent (DeltaX, DeltaY)

If (Angle < 98)
Return true
Else
Return false.

```
class Arrow
{
    Position Start { get; }
    Position Target { get; set}
    Room[] Rooms {get; set;)       //List of Rooms Player has selected.
    int Count;

    // Assumption is that Room objects
    // contain Position for each object in room
    public Arrow(Room start, Room[] rooms)
    {
        Start = start.Position;
        if (rooms < 6)  //assumed that there can be no more than 5 rooms
        {
            Target = rooms[0].Position;  // Assume that rooms are in order that arrow travels.
        }
    }

    bool shoot()
    {
        if (Arrow.Count > 0)
            HasBeenFired = true;
            for (int i = 0; i < Rooms.Length; i++)
                if (Target == Rooms[i].Wampus.Postion)
                    return true;                                //Display message to console
                else if (Target == Rooms[i].Bat.Postion)
                    Rooms[i].Bat.Count--;                       //Display message to console
                    Arrow.Count--;
                else if (Target == Start)return false;
                if (IsLessThan98Deg)
                    if(i + 1 != Rooms.Length)                  //Not going pass the last room
                        Target = Rooms[i + 1].Position;
                        i++;                                   // Arrow will move to next room
                        Arrow.Count--;
                    else
                        Arrow.Count--;
                else
                    Arrow.Count--;        //Arrow has ricochet off the wall and no longer travels to other rooms
        else
             //Display no arrows to console
    }

    bool IsLessThan98Deg()
    {
        int delta_x = source.x - target.x;
        int delta_y = source.y - target.y;
        double angle = Math.Atan2(source, target);

        if (angle < 98) return true;
        else return false;
    }
```

WUMPUS

STATE TO MAINTAIN:

- Current position of the Wumpus
- IsAwake to check it status
- Players current position
- List of Rooms in the cave

PRE-CONDITION

- Wumpus Position is randomly placed where Player isn't
- Will need to know all the rooms positions

FUNCTIONS

```
void Move()
    while isAwake
        pick a random room
        If new Room doesn't contain player
            Wumpus position = new postion
            isAwake = false
    }

Void Awake()

    Move()
    IsAwake = true
```

```
class Wumpus
{
    Postion Postion { get; set; }
    bool IsAwake { get; set; }
    Postion Player { get; set; }
    Rooms[] Rooms { get; set; }

    public Wumpus(Postion player, Rooms[] rooms)
    {
        Rooms = rooms;
        Player = player;
        Move();
    }

    public Move()
    {
        Random rand = new Random();
        while (IsAwake)
        {
            int roomIndex = rand(0, rooms.Length);
            Position newpostion = rooms[roomIndex].Position;
            if (Player != newposition)
                Position = newpostion;
                IsAwake = false;
        }
    }

    public void Awake()
    {
        IsAwake = true;
        Move();
    }

}
```