CREATING A 2D VISIBILITY SHADOW EFFECT ON UNITY PLATFORM WITH ROBUST C# CODE

Prepared by: Chicheng Zheng

Supervisor: Prof. Clark Verbrugge

Preparation for COMP396 Undergraduate Research Project

McGill University

April 2019

# Abstraction

Guards are very common in game, therefore it is important to implement the visibility of the guard in a map with obstacles. This project aims to create a 2D visibility shadow effort and apply it on real scenario.

目录

**[1. Introduction](#_Toc29391_WPSOffice_Level1)** **[2](#_Toc29391_WPSOffice_Level1)**

[1.1 Motivation summary](#_Toc19709_WPSOffice_Level2) [2](#_Toc19709_WPSOffice_Level2)

[1.2 Scope of the report](#_Toc19269_WPSOffice_Level2) [3](#_Toc19269_WPSOffice_Level2)

**[2. Background and Related Work](#_Toc19709_WPSOffice_Level1)** **[4](#_Toc19709_WPSOffice_Level1)**

[2.1 Background](#_Toc16386_WPSOffice_Level2) [4](#_Toc16386_WPSOffice_Level2)

[2.2 Related Work](#_Toc5952_WPSOffice_Level2) [5](#_Toc5952_WPSOffice_Level2)

**[3. Implementation](#_Toc19269_WPSOffice_Level1)** **[7](#_Toc19269_WPSOffice_Level1)**

[3.1 Generate obstacle and boundary](#_Toc9363_WPSOffice_Level2) [7](#_Toc9363_WPSOffice_Level2)

[3.2 Generate Ray Cast](#_Toc26913_WPSOffice_Level2) [9](#_Toc26913_WPSOffice_Level2)

[3.3 Generate Mesh](#_Toc9427_WPSOffice_Level2) [10](#_Toc9427_WPSOffice_Level2)

**[4. Test Case](#_Toc16386_WPSOffice_Level1)** **[12](#_Toc16386_WPSOffice_Level1)**

[4.1 Convex Polygon Obstacle](#_Toc10460_WPSOffice_Level2) [12](#_Toc10460_WPSOffice_Level2)

[4.2 Concave Polygon](#_Toc15830_WPSOffice_Level2) [13](#_Toc15830_WPSOffice_Level2)

[4.3 Partially View](#_Toc2141_WPSOffice_Level2) [14](#_Toc2141_WPSOffice_Level2)

[4.4 Visibility With Range Limitation](#_Toc1621_WPSOffice_Level2) [14](#_Toc1621_WPSOffice_Level2)

**[5. Conclusion(impact)](#_Toc5952_WPSOffice_Level1)** **[14](#_Toc5952_WPSOffice_Level1)**

**[6. Reference](#_Toc9363_WPSOffice_Level1)** **[15](#_Toc9363_WPSOffice_Level1)**

[a) define visiblity star shape polygon citation](#_Toc17905_WPSOffice_Level2) [15](#_Toc17905_WPSOffice_Level2)

[b) Art gallary book Ason’ algorithm](#_Toc9219_WPSOffice_Level2) [15](#_Toc9219_WPSOffice_Level2)

# Introduction

## Motivation summary

The visibility effect of a guard is a core part when implementing a roles play games. It is important to compute the visibility polygon corresponding to different locations of view point.

In addition to get the over all polygon with 360 degrees, in real scenario, instead of 360 degrees of view and infinity range, normally a character only has limited range of sight. Therefore, I further provide the interface for user to customize the parameter based on different purpose. Basically, now we have a relatively useful tool to handle the visibility problem.



The following panel is on the Game object named “InitialViewPoint”.

*“BPartiallyView”* is used for whether the user want to have a 360 degrees range of view or not. If user choose yes, then the next two “start direction” and “end direction” is used for determine the start direction and end direction in clockwise order.

“Range” is the parameter for user to specify the limit range for character, and it will consider infinity large if the value is set less or equal than 0.

“Moveable” is used for whether the user want the view point move with the mouse or not. If choose no, then “Position” is used for the stable view position.

“BMesh” is used for whether the user want to see the shadow effect or not. If the user choose no, it will only generate the debug lines cast.

## Scope of the report

The effect is implemented on the Unity 3.19 Platform.

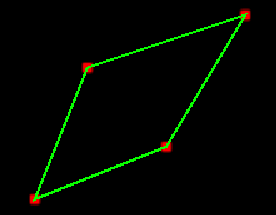
# Background and Related Work

## 2.1 Background

### 2.1.1 Convex Polygon

(“Convex”, 2018)

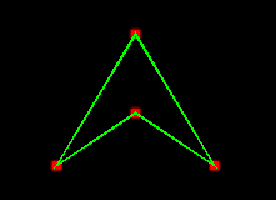
A polygon in which no line segment between two points on the boundary ever goes outside the polygon.



### 2.1.2 Concave Polygon

(“Concave”, 2018)

A polygon which is not convex.



### 2.1.3 Cross Product

(“Cross Product”, 2019)

The cross product of two vectors **a** and **b** is defined as **a**.x\***b**.y-**a**.y\***b**.x.

If the result is smaller then 0, which means the unsigned angle between **a** and **b** isclockwise from **a** to **b**.

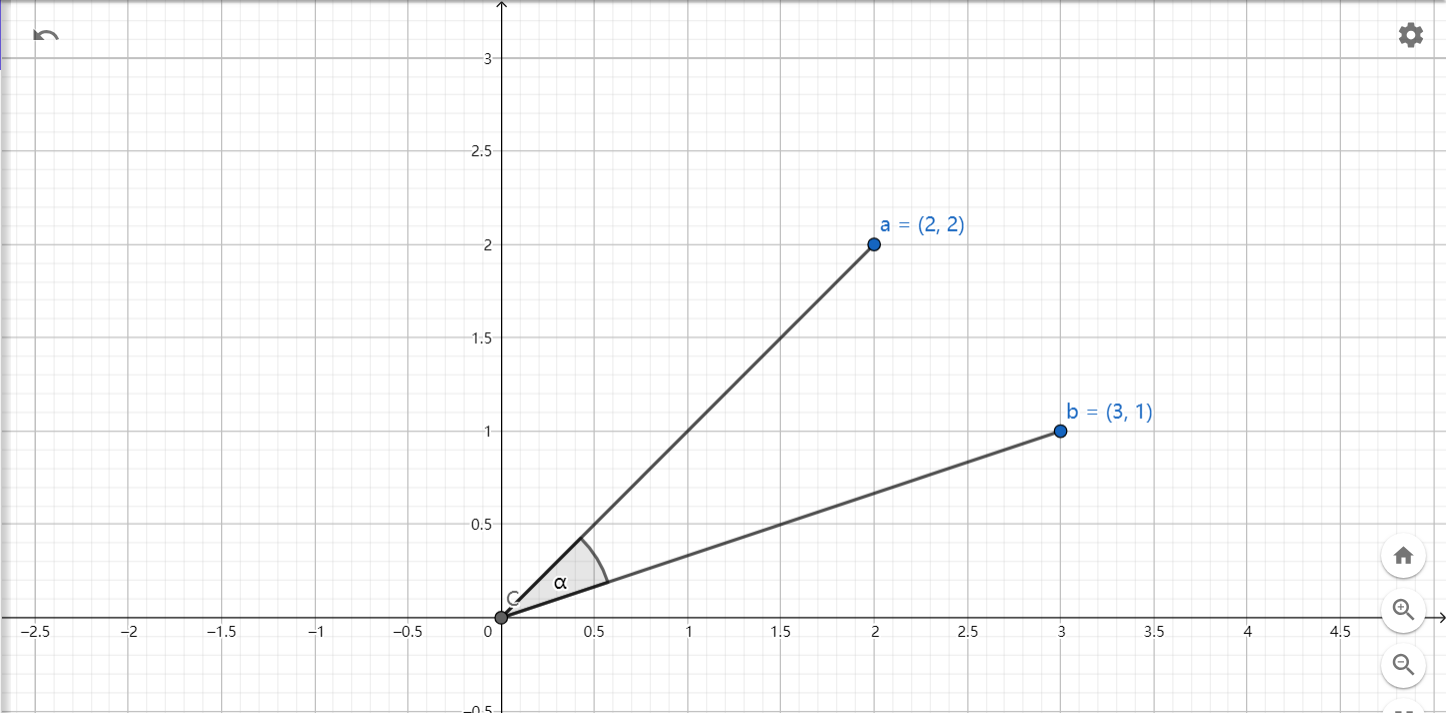
For example,

**a** = (2, 2), **b** = (3, 1)

*α* is the unsigned angle between **a** to **b**.

crossProduct(**a**,**b**) = 2 - 6 < 0, which means **a** gets angle *α* by go clockwise to **b**.

crossProduct(**b**,**a**) = 6 - 2 > 0, which means **b** gets angle *α* by go anticlockwise to **a**.

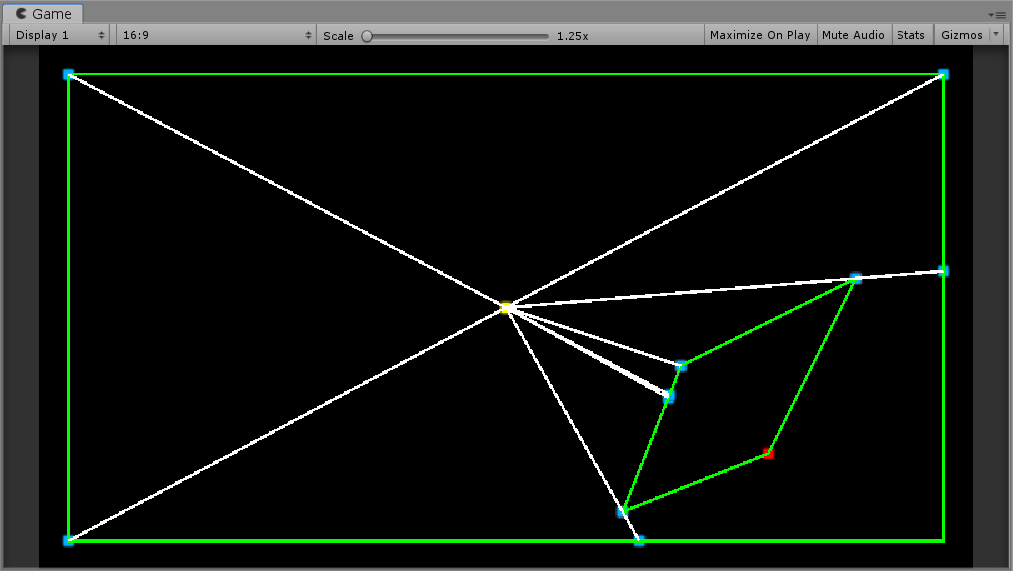


## 2.2 Related Work

In order to understand meaning of some variable, I self defined some terminology.

### 2.2.1 Critical Point

All the hit points generated by the ray cast from the view point, the visibility polygon is surrounded by the critical points. For example, all the blue points are critical points.



### 2.1.4 Sight Range

In the real world, it impossible for a guard to see infinity far away. Therefore, he has a range of visibility. Sight Range is the max value it could see from the view point.

### 2.1.5 Sight Angle

In the real world, it impossible for a guard to see all the things around it. Therefore the sight angle define the max range of perspective.

### 2.1.6 View Point

It is the guard position, and this project is focusing the visibility effect of related to this view point.

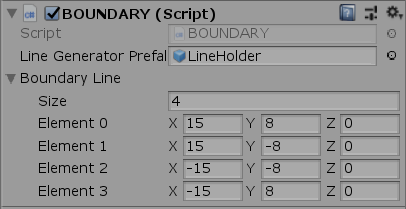
# Implementation

pseducode

## 3.1 Generate obstacle and boundary

### 3.1.1 Boundary

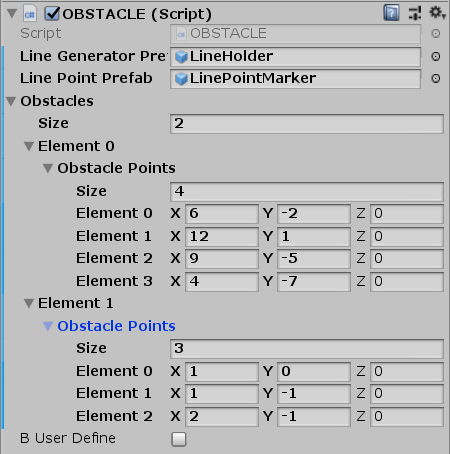
The Boundary Manager receive a list of Vector2D points denoted the vertex of the boundary,



it generate the edge collider 2D by loop through all the vertex and choose the adjacent pairs. We will have a list of edge collider before we generate the visibility area.

### 3.1.2 Obstacle

The Obstacle Manager receive a list of obstacles. Each obstacle is a list of Vector2D points.



Based on the list of obstacles, I generated a list of Polygon Collider 2D by loop through all the obstacles. This is basically same as generating the edge Collider 2D.

### 3.1.3 Triangulation

After we generate all the Polygon Collider 2D for each obstacles. We need to detect whether it is a concave obstacle or convex obstacle. If it is a concave obstacle, then we need to do triangulation. This mainly because when the obstacle is a concave, it may have two critical point with one line. However, the mechanism of Collider in Unity can only detect one collision. (“The triangulation in Unity”, 2014)

#### 3.1.3.1 Determine whether it is a concave polygon or not.

#### 3.1.3.2 Find the split-able point

3.1.3.3

## 3.2 Generate Ray Cast

Unity has already implemented a function named Physics2D.RaycastAll(). Basically, this function casts a ray against colliders in the Scene, returning all colliders that contact with it.(“Unity Documentation”, 2018)

### 3.2.1 Stable view point

Loop through all the vertex of obstacles in the map, generates the ray cast from the view point position to the vertex, use the given function RaycastAll() to get the cast result which is an array. Then we loop through the array to check verify each hit result.

#### 3.2.1.1 Generate Lines Cast

Generate Lines Cast()

{

foreach vertex in the obstacle {

Generate Line Cast();

}

}

#### 3.2.1.1 Generate Line Cast

GenerateLineCast(

Vector viewpoint, // view point position

OBSTACLE endPoint, // an obstacle

Vector2 direction, // line direction

int endPointIndex // indicate which point in the obstacle)

{

RaycastHit2D[] rayCastHits2D = Physics2D.RaycastAll(viewpoint, direction);

foreach rayCastHitResult in rayCastHits2D {

// if the hit result is the same position as obstacle position

if (rayCastHitResult == endPoint[endPointIndex ]) {

// If the neighbour endpoints of the hitting result are both in the one side, keep the hitting result

if (isSameSide()) {

critical.add(rayCastHitResult )

continue;

}

else {

critical.add(rayCastHitResult );

break;

}

}

else {

critical.add(rayCastHitResult );

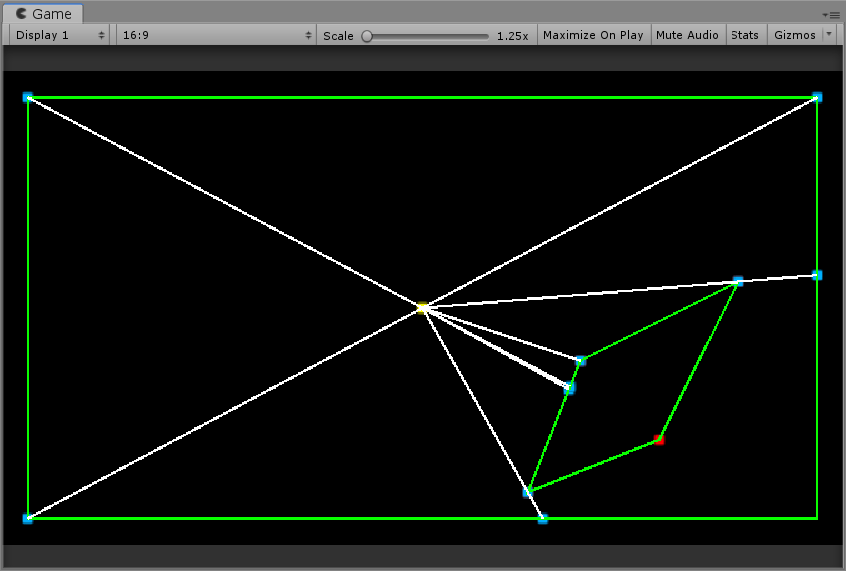
break;

}

}

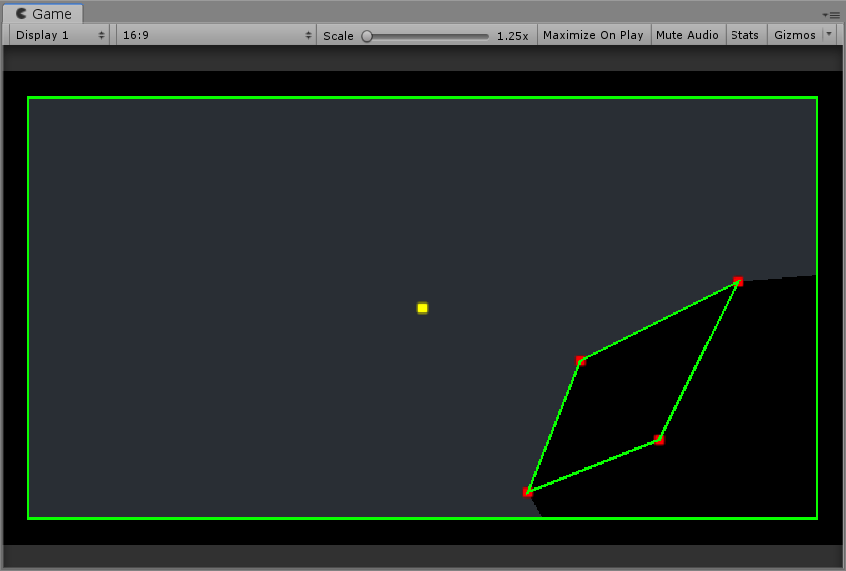
### 3.2.2 Movable view point

We want to have a moveable view point. It can be implemented by connected the viewpoint location and the mouse location. We can generate ray cast in every frame, and then destroy it in a short delay.



## 3.3 Generate Mesh

In order to generate the following visibility effect from the critical point given by the ray cast.



### 3.3.1 Sort the critical point in a clockwise order

First we need to sort all the critical point in a clockwise order(start from base vector (1, 0) by default). I use the cross product to determine the unsigned angle between critical point and base vector is clockwise or not.

### 3.3.2 Adjust the order

int cur = 0;

// the index of previous node

int pre = list.Count - 1;

// the index of next node

int next = 1;

while (cur < list.Count + 1 && round < 3)

{

if (cur == list.Count)

{

cur = 0;

round++;

}

int end = cur;

while (end + 1 < list.Count && compareByAngle(list[end], list[end + 1]) == 0)

{

end++;

next++;

next %= list.Count;

}

// List with index from "cur" to "end" are all in the same line

if (end > cur && flag)

{

Vector2 preNode = list[pre].location;

Vector2 nextNode = list[next].location;

bool test;

if (cur == 0)

{

test = isInSameObstaclesLine(list[end].location, nextNode);

}

else

{

test = isInSameObstaclesLine(list[cur].location, preNode);

}

if (!test)

{

// swap the order

swapOrder(list, cur, end);

}

}

else

{

flag = true;

}

cur = end + 1;

pre = end;

next = cur + 1;

if (next >= list.Count)

{

next -= list.Count;

}

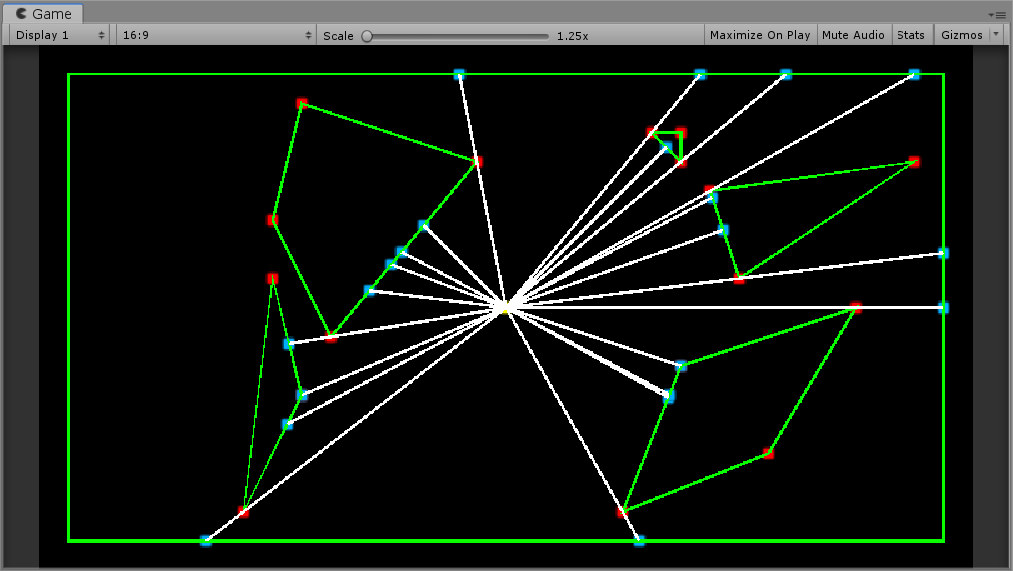
## 3.4 Implement

## 3.5 Implement range limit

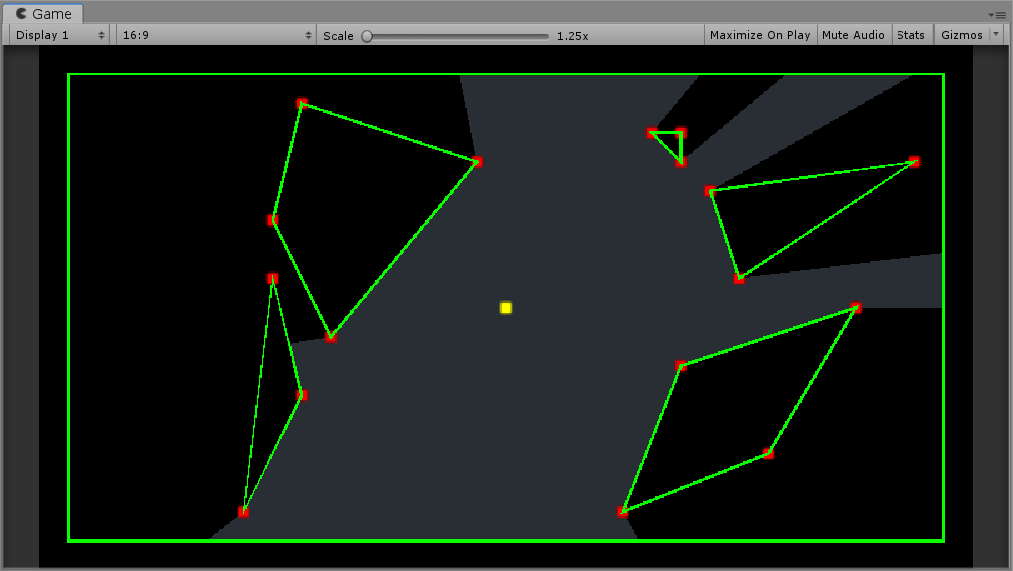
# Test Case

## 4.1 Convex Polygon Obstacle

a)



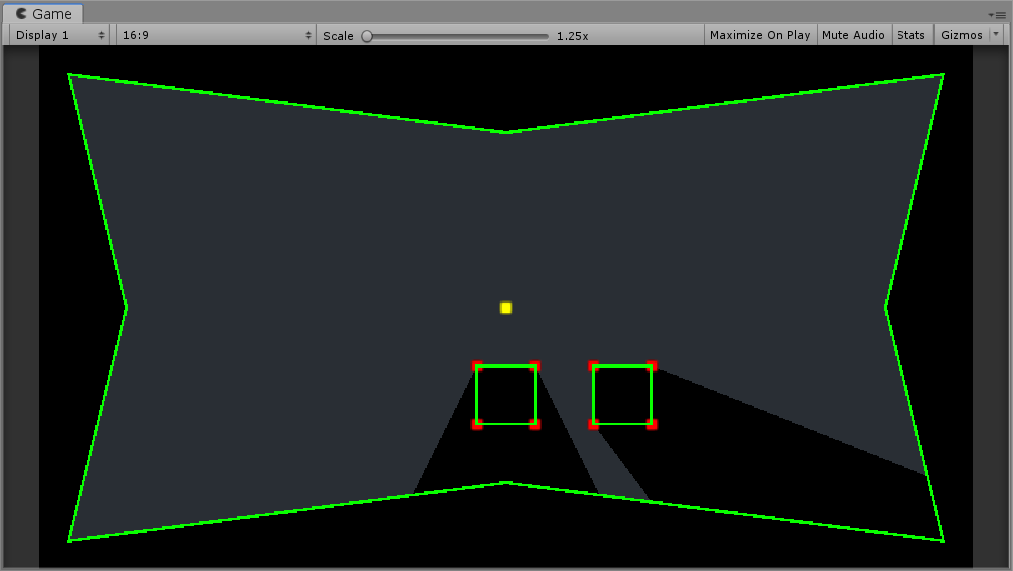
The polygon surrounded by the red points are obstacle and the view point is denoted by yellow point. The blue points are the critical point.



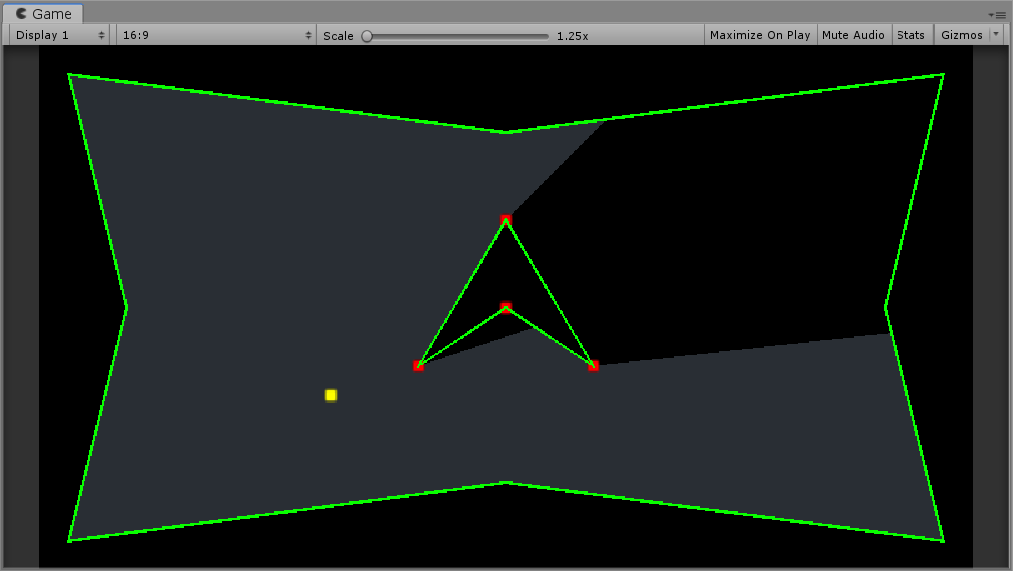
The gray area is the visibility effect of that view point without sight range or sight angle restrictions.

## 4.2 Concave Polygon

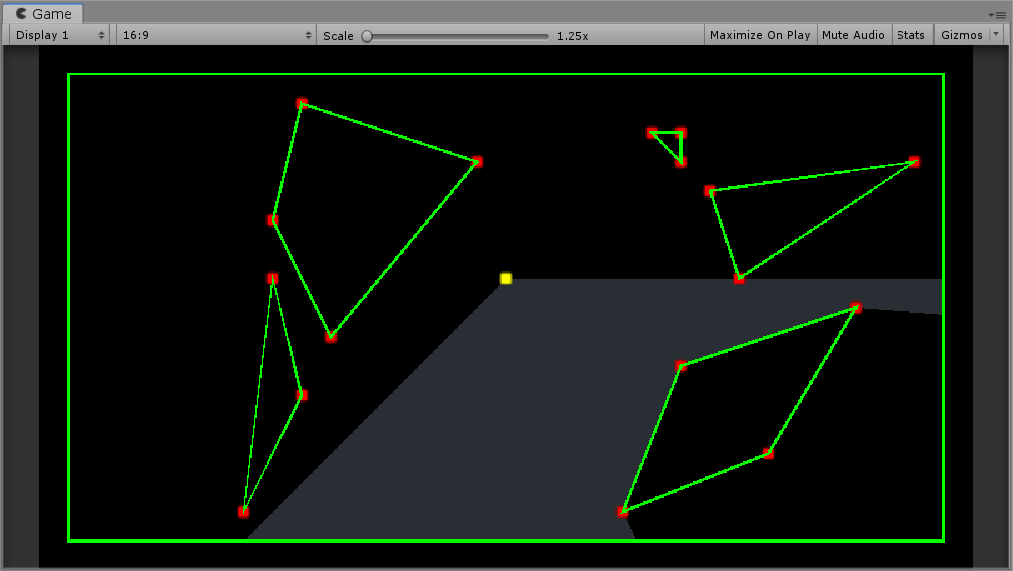
### 4.2.1 Concave Boundary



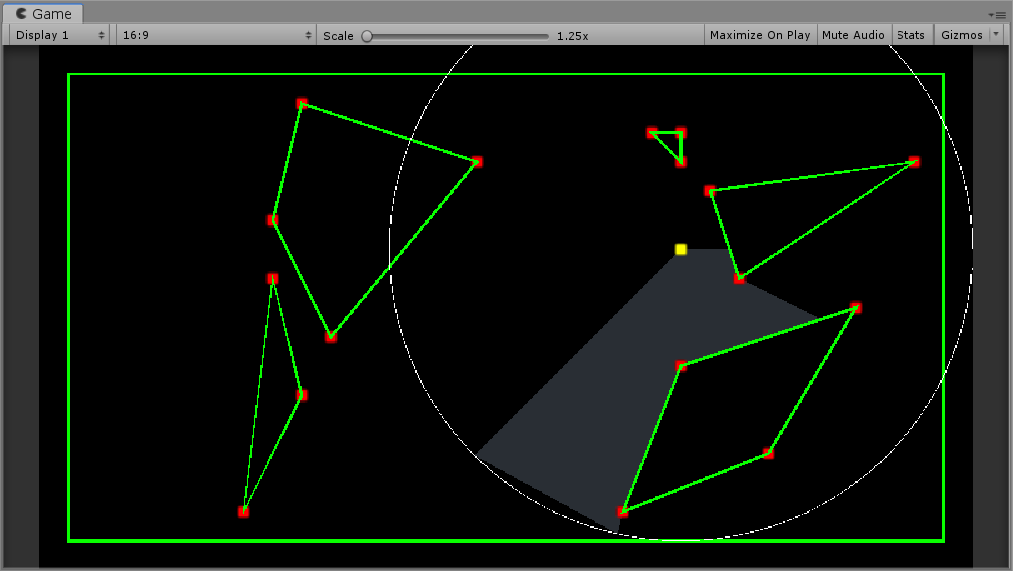
### 4.2.2 Concave Obstacle



## 4.3 Partially View



## 4.4 Visibility With Range Limitation



# Conclusion(impact)

Astano’s

Art gallary joe & simpson

# Reference

In *Wikipedia, the free encyclopedia.* Retrieved May 16, 2018, from <https://en.wikipedia.org/wiki/Concave> (Accessed:18 March 2019)

In *Wikipedia, the free encyclopedia.* Retrieved May 16, 2018, from <https://en.wikipedia.org/wiki/Convex> (Accessed:18 March 2019)

In *Wikipedia, the free encyclopedia.* Retrieved March 17, 2019, from <https://en.wikipedia.org/wiki/Cross_product> (Accessed:18 March 2019)

LAN\_YT, (2014)*The triangulation in Unity.* [https://www.cnblogs.com/lan-yt/p/9200621.html](Casts a ray against colliders in the Scene, returning all colliders that contact with it.) (Accessed:18 March 2019)

Unity Documentation, (2018) <https://docs.unity3d.com/ScriptReference/Physics2D.RaycastAll.html> (Accessed: 18 March 2019)

* 1. define visiblity star shape polygon citation
  2. Art gallary book Ason’ algorithm