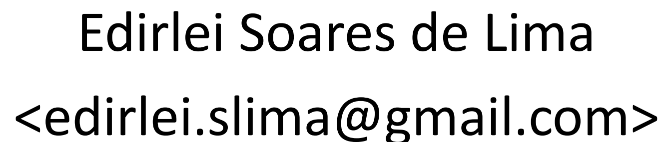
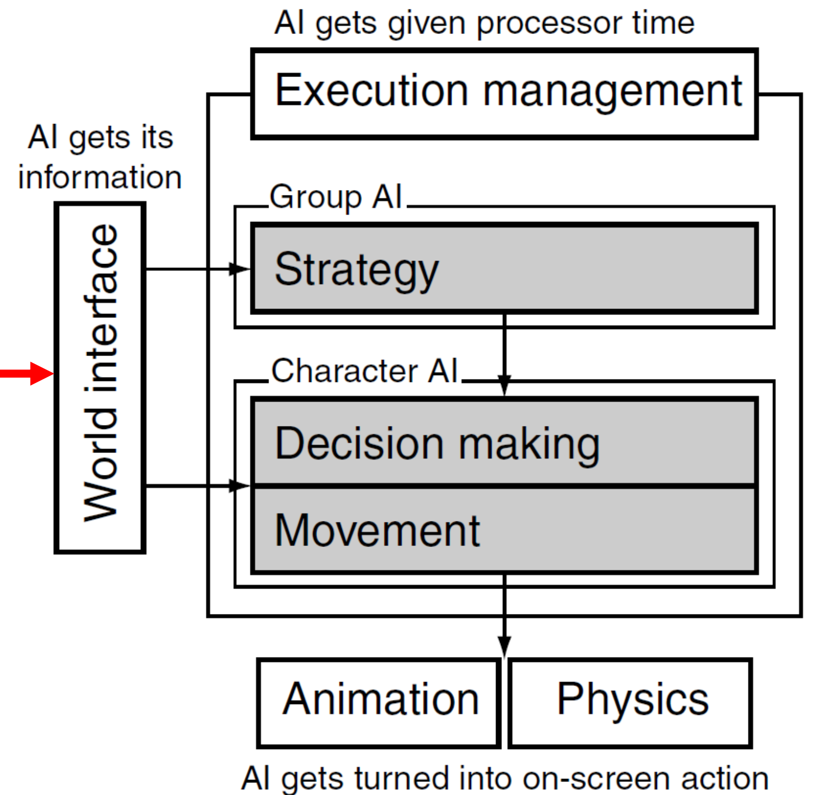
Artificial Intelligence

Lecture 06 – Sensor Systems


Edirlei Soares de Lima
<edirlei.slima@gmail.com>

Game AI – Model

- Pathfinding
- Steering behaviours
- Finite state machines
- Automated planning
- Behaviour trees
- Randomness
- **Sensor systems**
- Machine learning



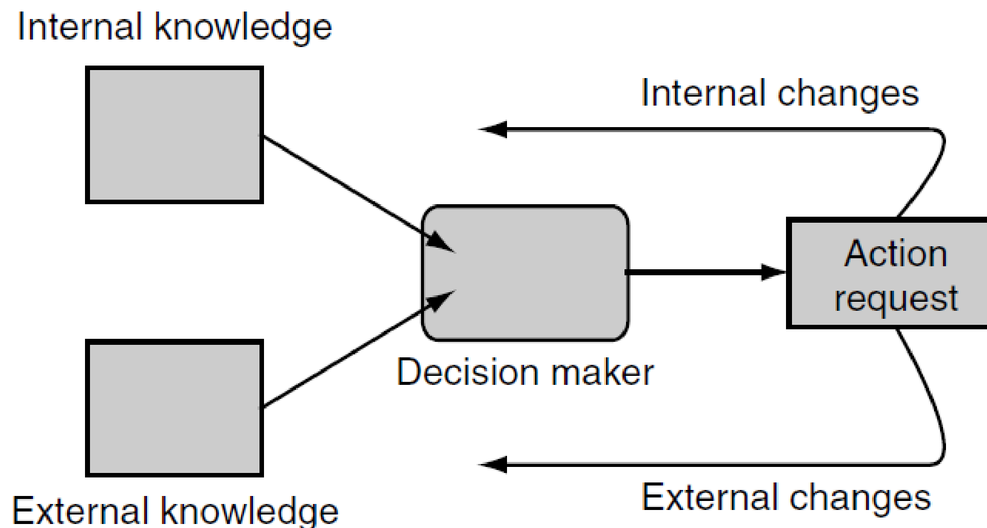
Decision Making

- In game AI, decision making is the ability of a character/agent to decide what to do.

- The agent processes a set of information that it uses to generate an action that it wants to carry out.
  - **Input:** agent's knowledge about the world;
  - **Output:** an action request;

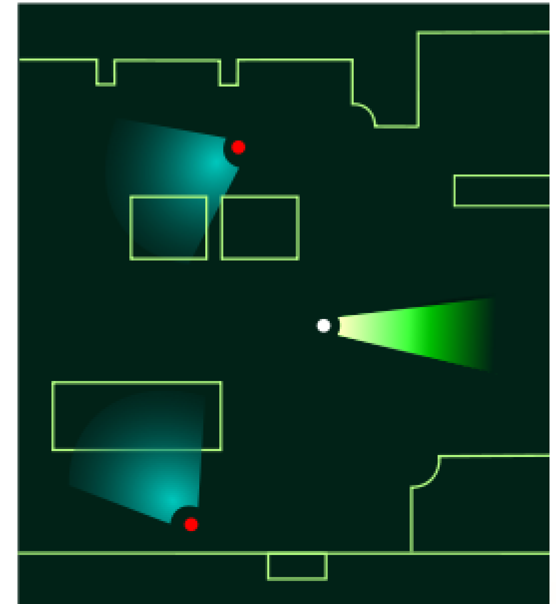
Decision Making

- The knowledge can be broken down into external and internal knowledge.
  - **External knowledge:** information about the game environment (e.g. characters' positions, level layout, noise direction).
  - **Internal knowledge:** information about the character's internal state (e.g. health, goals, last actions).



Sensing Information

- Game environments simulate the physical world, at least to some degree.
  - In the real world, a person gains information about its environment by using its senses.
- If a loud noise is generated in the game, all characters that are close to the event must "hear the sound".
  - However, a character across the other end of the level may not heard it, and neither would a character behind a soundproof window.



Sensing Information

- Is not possible to make a real simulation of how biological sensors work. Simplifications are indispensable.
  - There is no point in simulating the way that sound travels from the headphones on a character’s head down its ear canal.

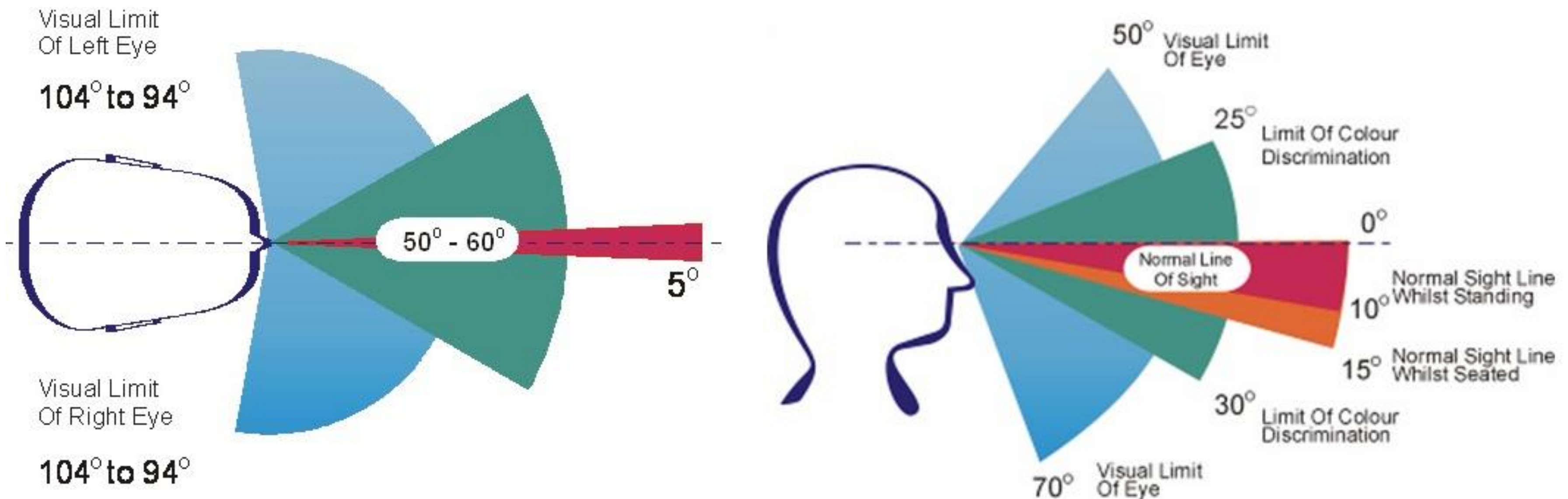
- **Example:**
  - A character might be constantly looking at the position of the player.
  - When the player gets near enough, the character suddenly engages its “chase” action.
  - It appears to the player as if the character couldn’t see the player until he got close enough.

Sensory Modalities

- Four natural human senses are suitable for use in a game (in roughly decreasing order of use):
  - **Sight:** the ability of character to see. Players can easily tell if it is being simulated badly.
  - **Touch:** the ability of character to sense direct physical contact. Easily implemented using collision detection.
  - **Hearing:** the ability of character to hear sounds. The real world sound propagation process is complex and usually is simplified in games.
  - **Smell:** the ability of character to smell gases. Is a relatively unexplored sense in games.

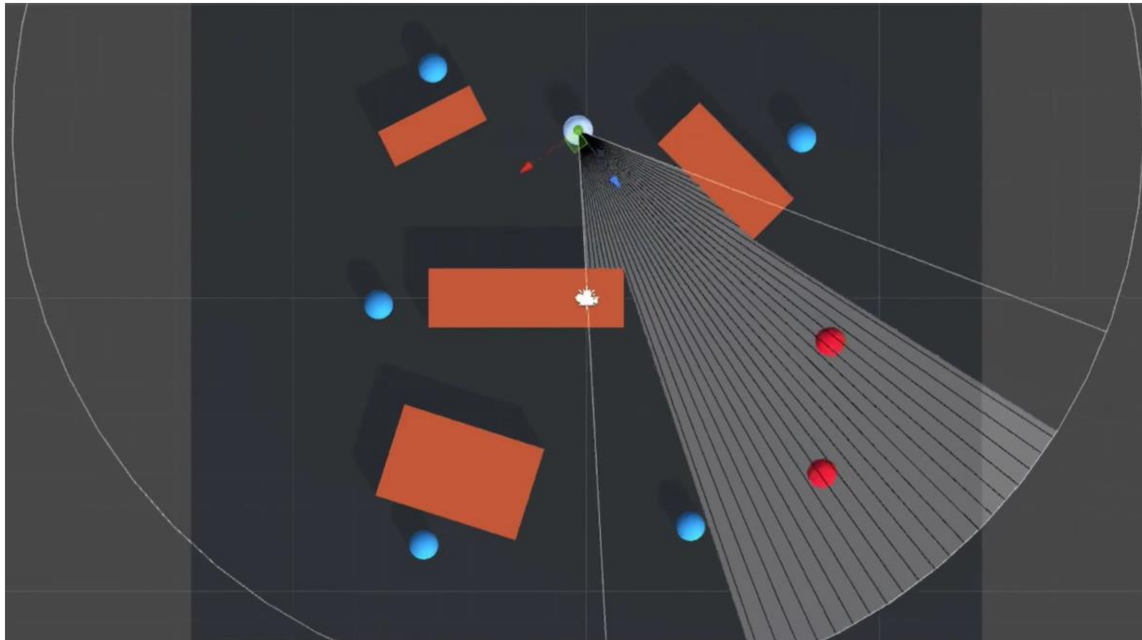
Sight

- There are several factors that can affect a character's ability to see something:
  - Sight Cone: cone shape that limits the view angle.



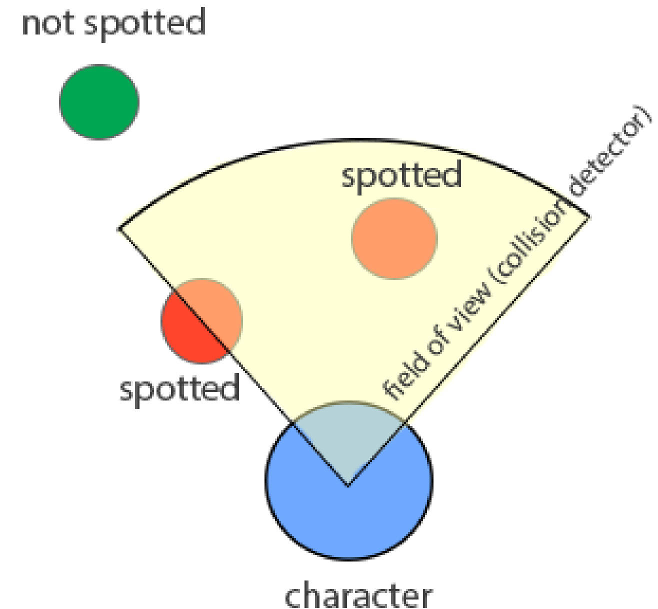
Sight

- There are several factors that can affect a character's ability to see something:
  - Line of Sight: to see an object, the character needs a direct line of sight with the object.



Sight

- There are several factors that can affect a character's ability to see something:
  - Distance: although humans have no distance limitation to their sight, most games add a distance limit.
    - Atmospheric effects (such as fog) and the curve of the earth also limit the ability to see very long distances.



Sight

- There are several factors that can affect a character's ability to see something:

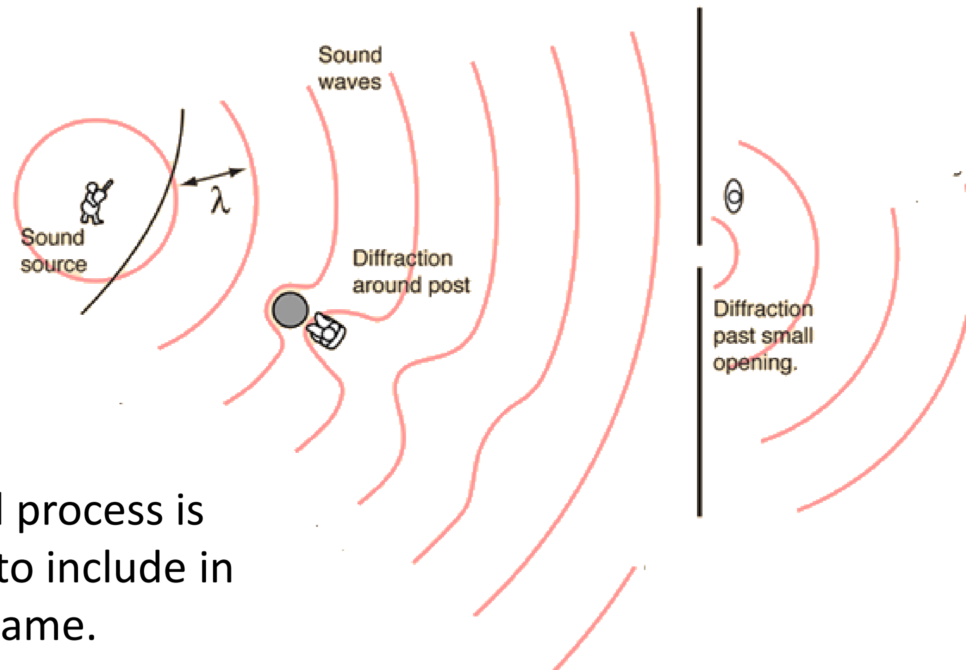
  - Brightness: its notoriously difficult to see in dim light or dark rooms.

Sight

- There are several factors that can affect a character's ability to see something:
  - Differentiation: its difficult see objects that do not contrast with their backgrounds. Camouflage works based on this principle.

Hearing

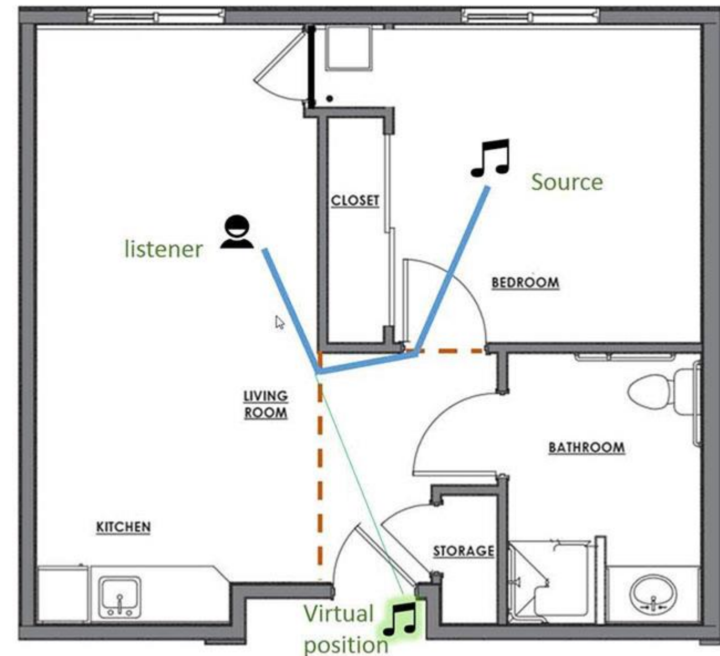
- The real world <u>sound propagation</u> process is complex:
  - Sound travels as compression waves through any physical medium.
  - The wave takes time to move, and as it moves it spreads out and is subject to friction. Both factors serve to diminish the intensity (volume) of the sound with distance.



The physical process is to complex to include in the AI of a game.

Hearing

- Games usually simplify the sound propagation process: the sound uniformly reduce its volume over distance until it pass below some threshold.
  - Sound travels through air around corners without a problem.
  - Typically all materials are divided into two categories: those that do not transmit sound and those that do.
  - The speed of sound is often fast enough not to be noticed.

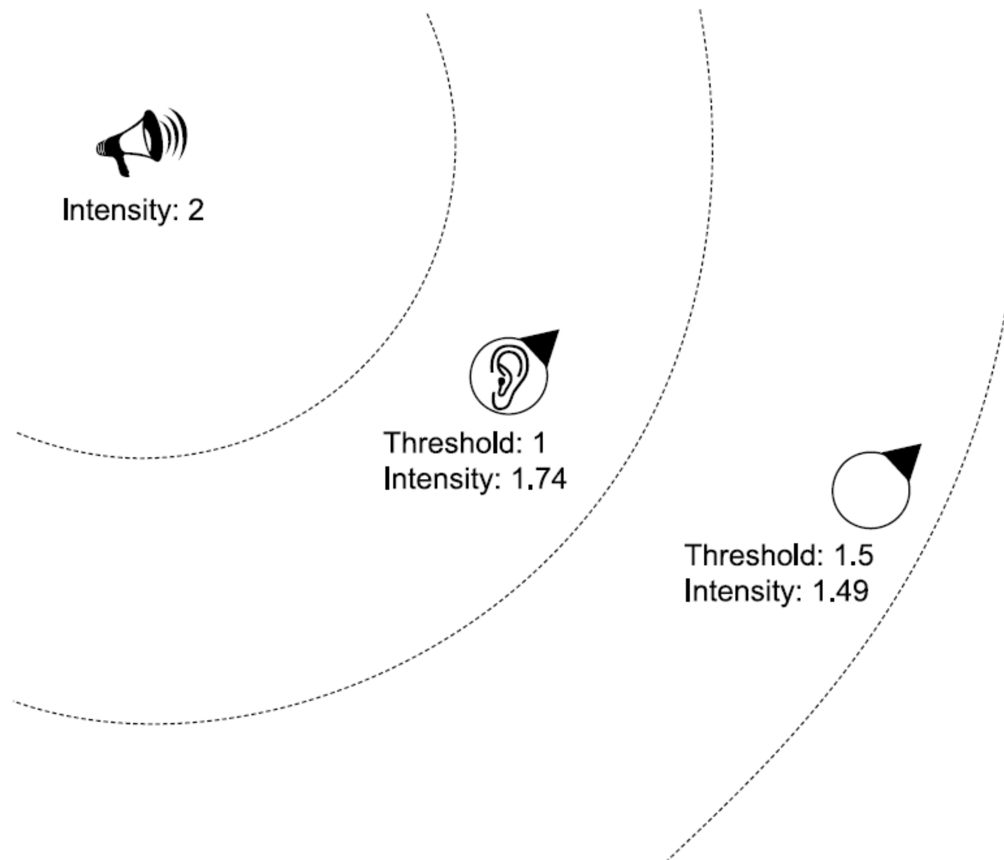


Sense Management Algorithm

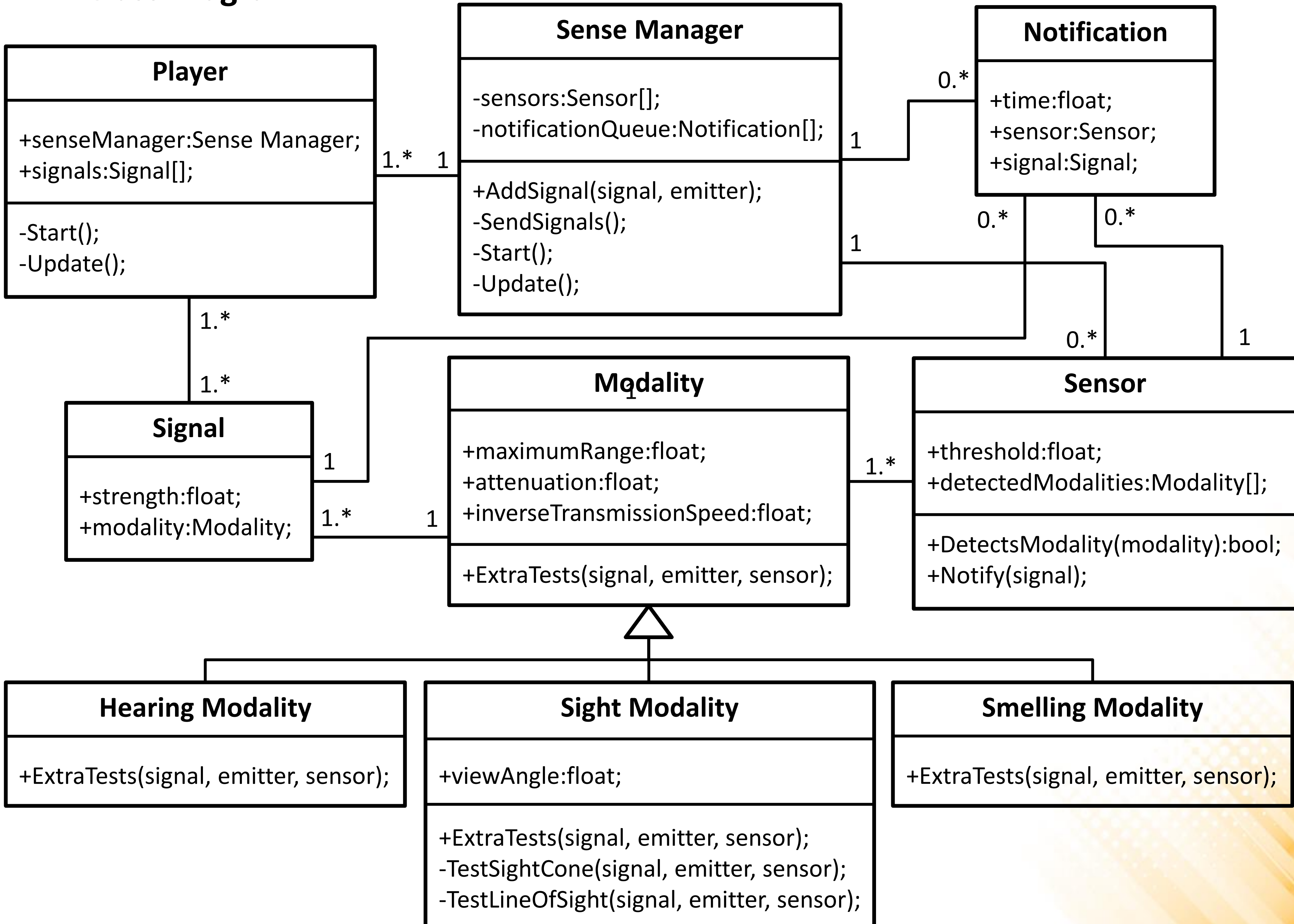
- **Algorithm:**
  1. Potential sensors (e.g. characters) are found and registered;
  2. The sense manager accepts signals: messages that indicate that something has occurred in the game level.
     - Signal information: signal modality, intensity at its source, and the position of the source.
  3. When a signal is sent to the sense manager, it finds all sensors within the maximum radius of the corresponding modality.
  4. For each sensors, it calculates the intensity of the signal when it reaches the sensors and the time when that will happen.
  5. If the intensity test passes, then the algorithm may perform additional tests (e.g. line of sight).
  6. If all tests pass, then a request to notify the character is posted in a queue.

Sense Management Algorithm

- Attenuation in action:



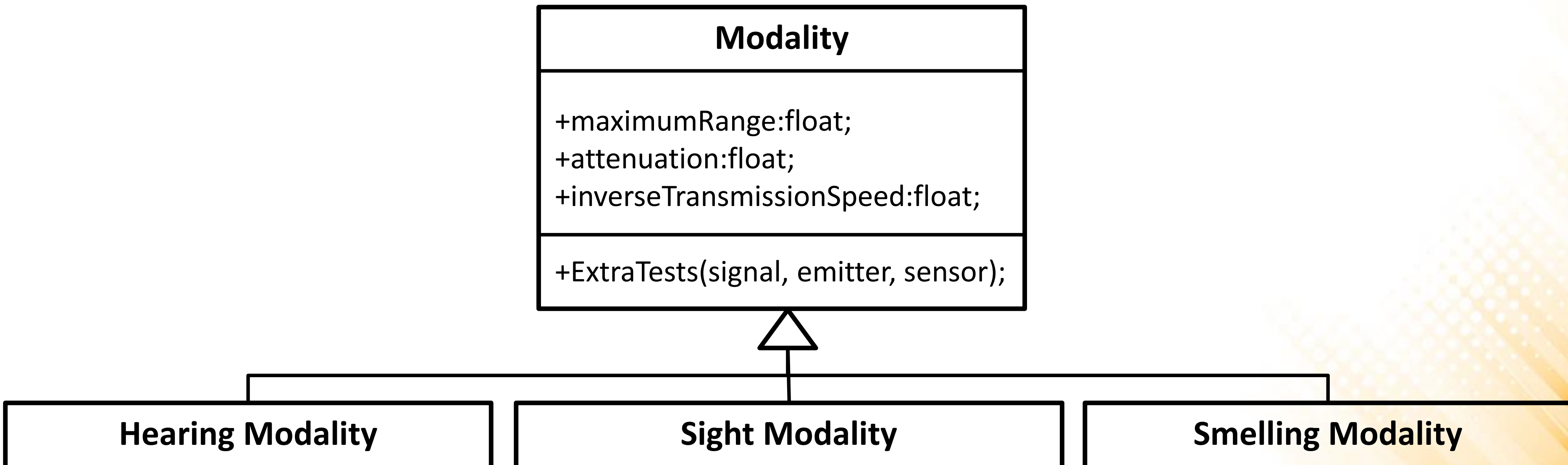
Class Diagram



Modality Classes

- **Modality Class:**

```
public abstract class Modality : ScriptableObject{
    public float maximumRange;
    public float attenuation;
    public float inverseTransmissionSpeed;
    public abstract bool ExtraTests(Signal signal, Transform emitter,
                                    Sensor sensor);
}
```



Modality Classes

- **Hearing Modality Class:**

```
[CreateAssetMenu(menuName = "Sense System/Modality/Hearing")]
public class HearingModality : Modality{
  public override bool ExtraTests(Signal signal, Transform emitter,
                                  Sensor sensor){
    return true;
  }
}
```

- **Smelling Modality Class:**

```
[CreateAssetMenu(menuName = "Sense System/Modality/Smelling")]
public class SmellingModality : Modality{
  public override bool ExtraTests(Signal signal, Transform emitter,
                                  Sensor sensor){
    return true;
  }
}
```

Modality Classes

- **Sight Modality Class:**

```
[CreateAssetMenu(menuName = "Sense System/Modality/Sight")]
public class SightModality : Modality
{
  public float viewAngle;

  public override bool ExtraTests(Signal signal, Transform emitter,
                                  Sensor sensor){
    if (!TestSightCone(signal, emitter, sensor))
      return false;
    if (!TestLineOfSight(signal, emitter, sensor))
      return false;
    return true;
  }

  ...
```

Modality Classes

```
  private bool TestSightCone(Signal signal, Transform emitter,
                             Sensor sensor){
    Vector3 targetDir = emitter.position - sensor.transform.position;
    float angle = Vector3.Angle(targetDir, sensor.transform.forward);
    if (angle < viewAngle)
      return true;
    return false;
  }

  private bool TestLineOfSight(Signal signal, Transform emitter,
                               Sensor sensor){
    Vector3 targetDir = emitter.position - sensor.transform.position;
    RaycastHit hit;
    if (Physics.Raycast(sensor.transform.position, targetDir,
        out hit, 10)){
      if (hit.transform.name == emitter.tag)
        return true;
    }
    return false;
  }
}
```

Signal and Notification Classes

- **Signal Class:**

```
[CreateAssetMenu(menuName = "Sense System/Signal")]
public class Signal : ScriptableObject{
  public float strength;
  public Modality modality;
}
```

Signal
+strength:float; +modality:Modality;

- **Notification Class:**

```
public class Notification {
  public float time;
  public Sensor sensor;
  public Signal signal;
  public Notification(float t, Sensor sen, Signal sig){
    time = t;
    sensor = sen;
    signal = sig;
  }
}
```

Notification
+time:float; +sensor:Sensor; +signal:Signal;

Sensor Class

- **Sensor Class:**

```
public class Sensor : MonoBehaviour {
  public float threshold;
  public Modality[] detectedModalities;

  public bool DetectsModality(Modality modality){
    foreach (Modality mod in detectedModalities){
      if (mod == modality)
        return true;
    }
    return false;
  }

  public void Notify(Signal signal){
    if (signal.modality is HearingModality)
      Debug.Log(name + " heard a sound!");
    if (signal.modality is SightModality)
      Debug.Log(name + " saw someone!");
  }
}
```

Sensor
+threshold:float; +detectedModalities:Modality[];
+DetectsModality(mod):bool; +Notify(signal);

Sense Manager Class

- **Sense Manager Class:**

```
public class SenseManager : MonoBehaviour
{
  private List<Sensor> sensors = new List<Sensor>();
  private SimplePriorityQueue<Notification> notificationQueue = new
                                  SimplePriorityQueue<Notification>();

  void Start()
  {
    Sensor[] allSensors = Object.FindObjectsOfType<Sensor>();
    foreach (Sensor sensor in allSensors)
      sensors.Add(sensor);
  }

  void Update()
  {
    SendSignals();
  }

  ...
```

Sense Manager
-sensors:Sensor[]; -notificationQueue:Notification[];
+AddSignal(signal, emitter); -SendSignals(); -Start(); -Update();

Sense Manager Class

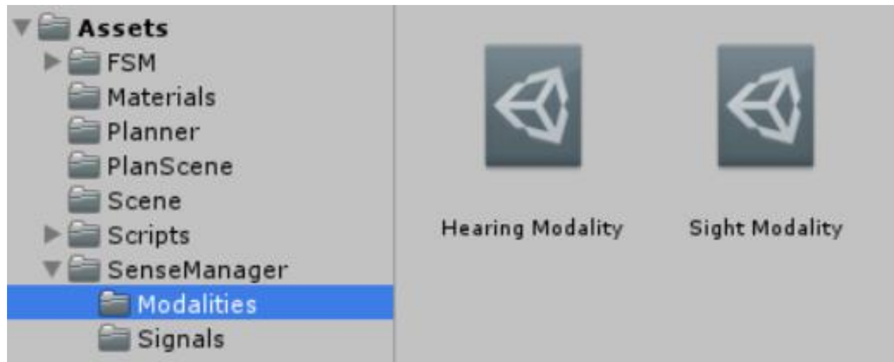
```
public void AddSignal(Signal signal, Transform emitter){
  foreach(Sensor sensor in sensors){
    if (!sensor.DetectsModality(signal.modality))
      continue;
    float distance = Vector3.Distance(emitter.position,
                                      sensor.transform.position);
    if (signal.modality.maximumRange < distance)
      continue;
    float intensity = signal.strength *
                Mathf.Pow(signal.modality.attenuation, distance);
    if (intensity < sensor.threshold)
      continue;
    if (!signal.modality.ExtraTests(signal, emitter, sensor))
      continue;
    float time = Time.time + (distance *
                       signal.modality.inverseTransmissionSpeed);
    Notification notification = new Notification(time, sensor, signal);
    notificationQueue.Enqueue(notification, notification.time);
  }
  SendSignals();
}
...
```

Sense Manager Class

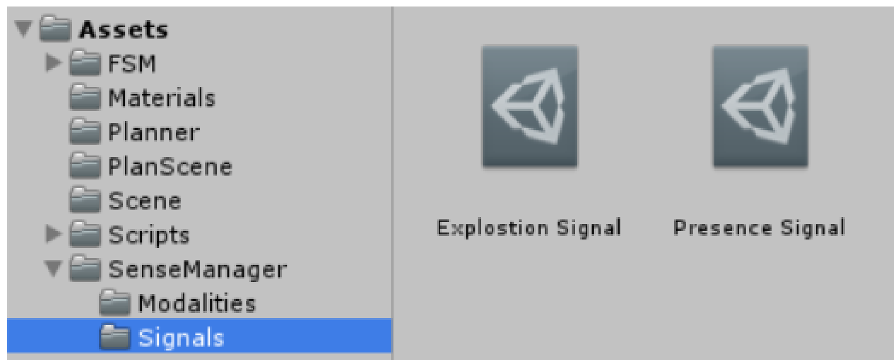
```
  private void SendSignals()
  {
    while (notificationQueue.Count > 0)
    {
      Notification notification = notificationQueue.First;
      if (notification.time < Time.time)
      {
        notification.sensor.Notify(notification.signal);
        notificationQueue.Dequeue();
      }
      else
      {
        break;
      }
    }
  }
}
```

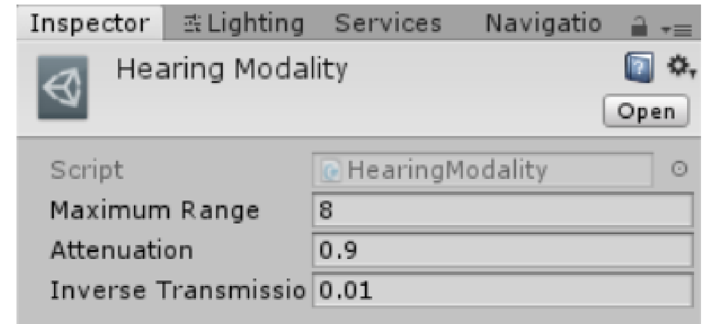
Sense Manager - Objects

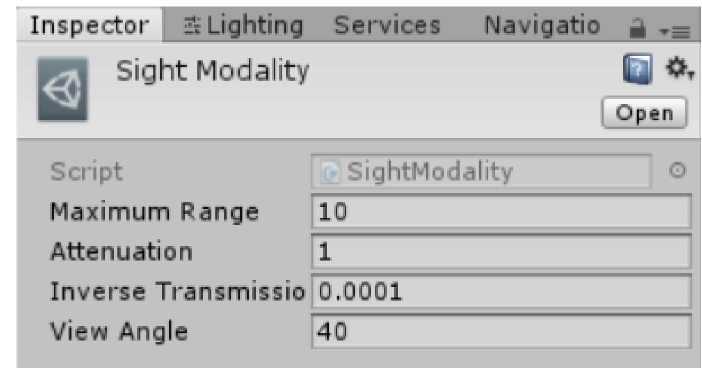
- Modalities:



- Signals:

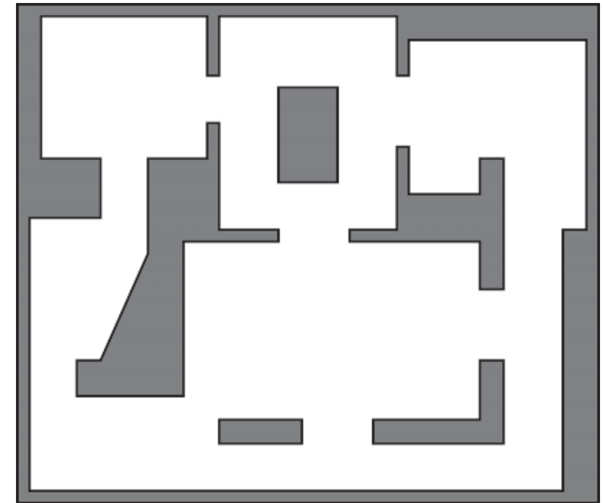






Exercise 1

1) Create a level geometry based on the illustrated map. Then:
   a) Add 10 NPCs to the level and implement a simple Finite State Machine (FSM) to control them (patrol, chase, kill);
   b) Add a controllable player character;
   c) Using the Sense Management system, allow the NPCs to see the player (modify the FSM to use the Sense Management system);
   d) Add 10 hidden bombs in the level. When the player touches a bomb, it explodes (producing sound). The NPCs that hear the explosion, will go to the explosion place to see what happened.



Level map

Exercise 2

2) An alternative approach to implement sense management in a game consists of using the physics engine to detect and trigger sense functions. For this task, Unity has great function to detect objects (sensors) within a sphere range:

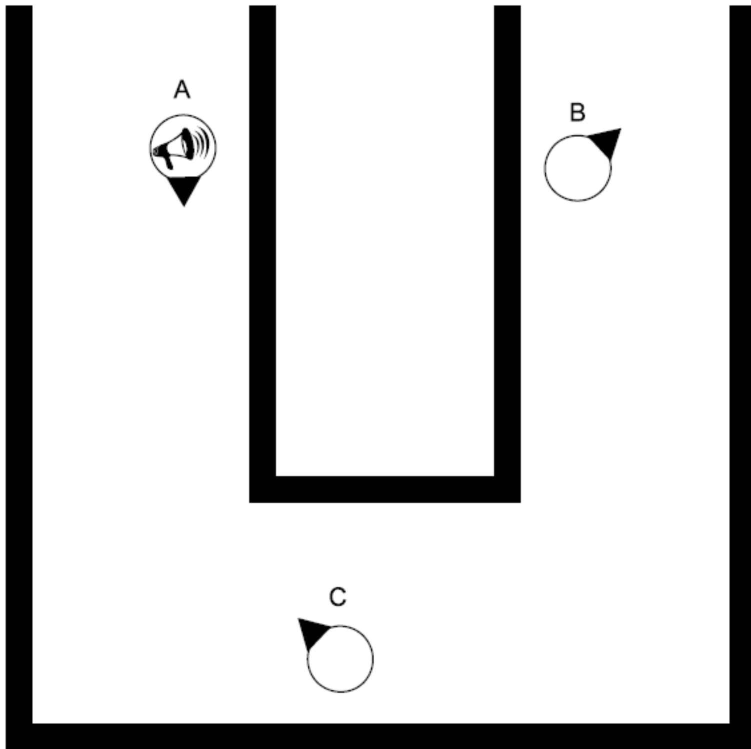
```
Collider[] Physics.OverlapSphere(Vector3 position, float radius,
                                 int layerMask = AllLayers);
```

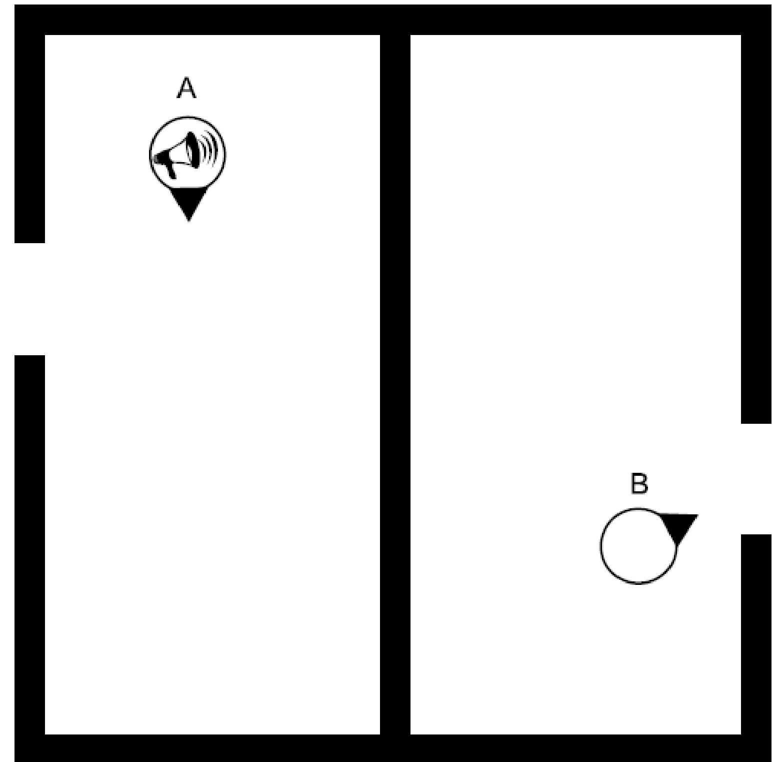
Implement and test the classes of a sense management system using the physics engine.

- Tips:
  - Some possible classes: Sensor (objects that receive senses), Emitter (objects that generate senses to all sensors that are within a sphere range of the emitter);
  - An explosion generates a single OverlapSphere call when the explosion occurs;
  - An object that can be seen by sensors need to call the OverlapSphere function every frame.

Sense Management

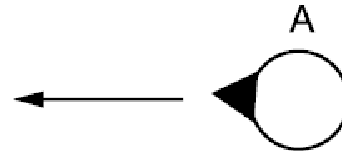
- **Weakness 1:** the system doesn't take level geometry into consideration, other than for line-of-sight tests.

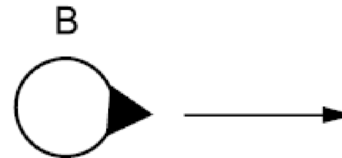




Sense Management

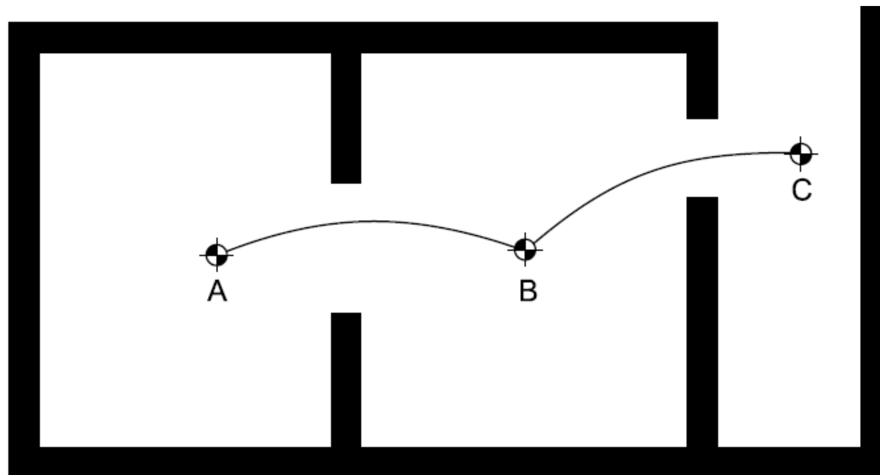
- **Weakness 2:** timing discrepancy for moving characters.





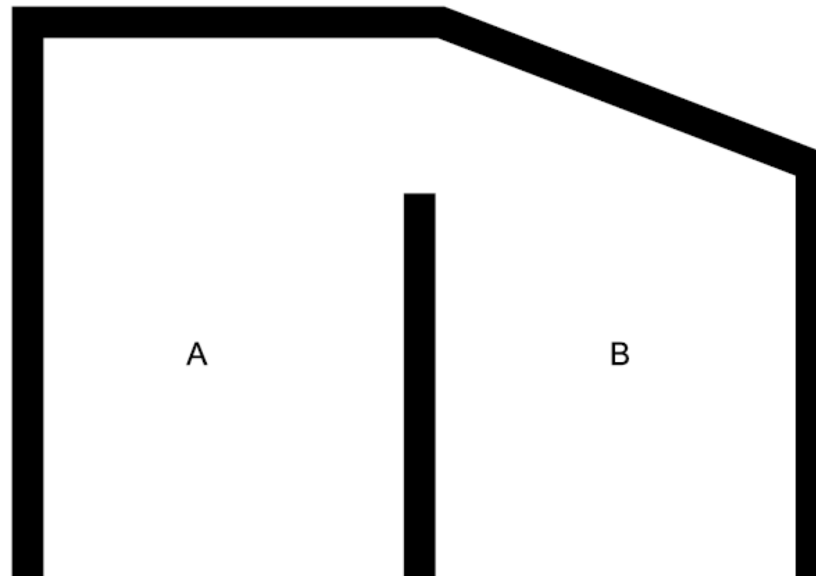
Sense Graph

- When the weaknesses of the Sense Management system are really a problem, there is a solution: **Sense Graph**.
  - The game level geometry is transformed into a directed graph for sense management (similar to the pathfinding graph).
  - Each node in the graph represents a region of the game level where signals can pass around.
  - For all modalities, the node contains an attenuation value that indicates how a signal decays for each unit of distance it travels.



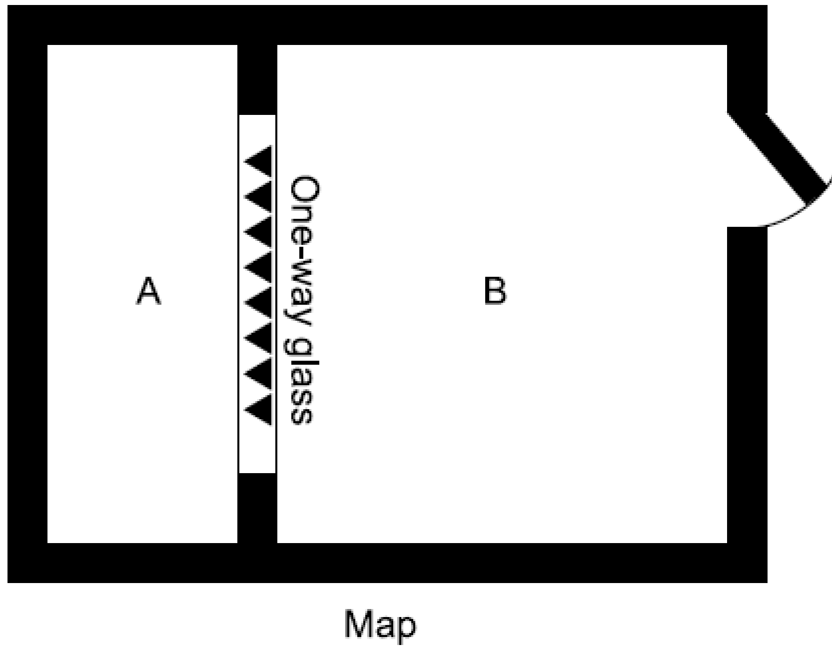
Sense Graph

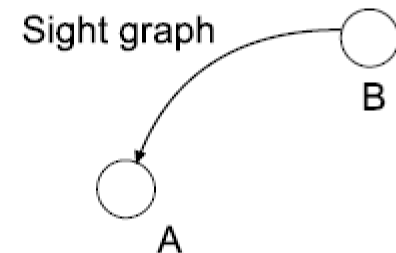
- The Sense Graph takes into consideration the level geometry:



Sense Graph

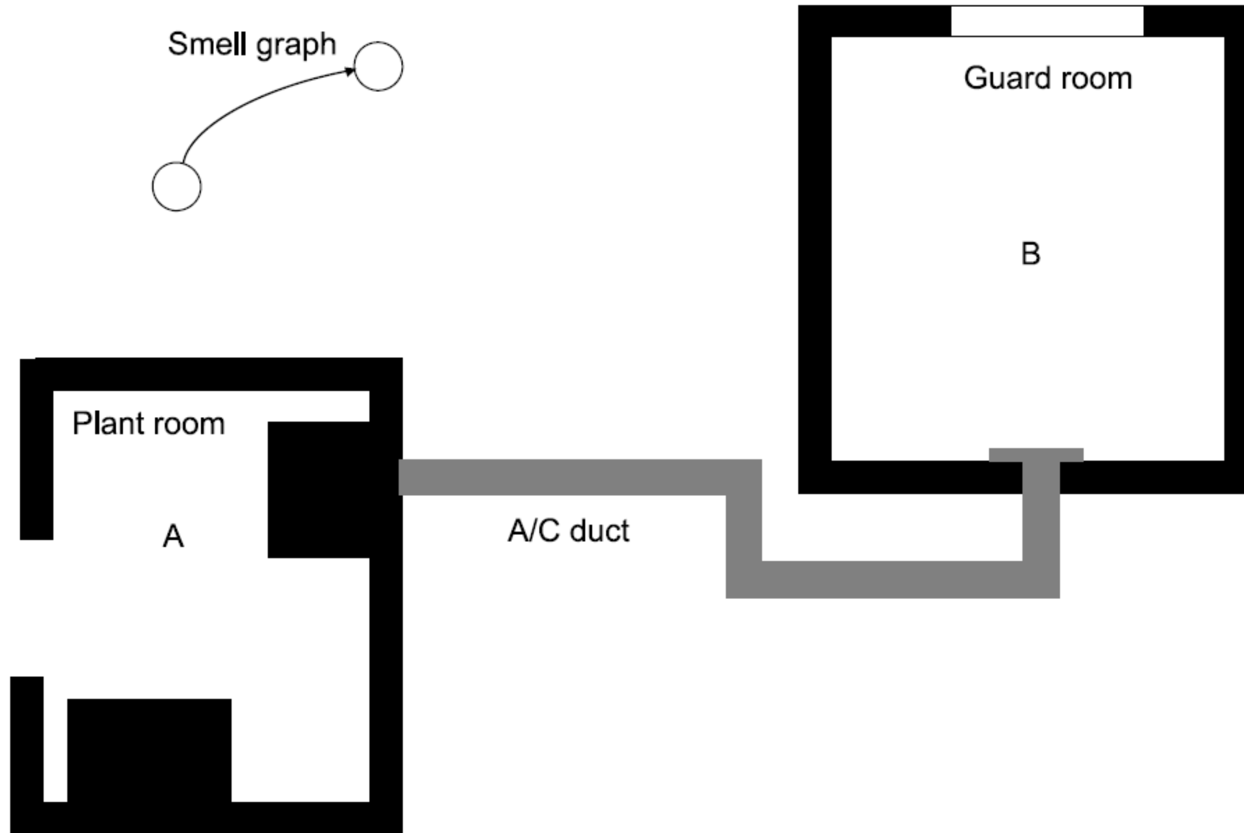
- Sense graph for one-way glass:





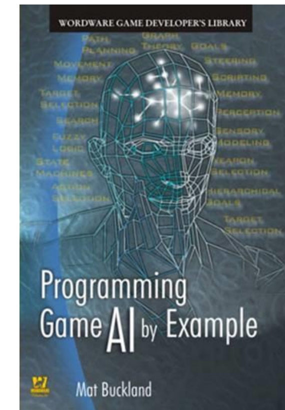
Sense Graph

- Air-conditioning in a sense graph:



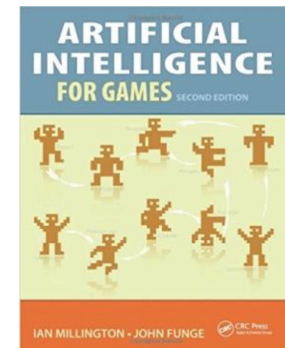
Further Reading

- Buckland, M. (2004). **Programming Game AI by Example**. Jones & Bartlett Learning. ISBN: 978-1-55622-078-4.



  – **Chapter 8: Practical Path Planning**

- Millington, I., Funge, J. (2009). **Artificial Intelligence for Games** (2nd ed.). CRC Press. ISBN: 978-0123747310.



  – **Chapter 10.5: Sense Management**