# 

# 

# 

# 

PEC 2 – IA para juegos de estrategia y rol

Joaquín López Soriano

Inteligencia artificial para videojuegos

Máster Universitario en Diseño y Programación de Videojuegos

Universidad Oberta de Cataluña

# 

Índice

Contenido

[Herramientas 3](#_Toc153144567)

[Actividad 1: colmena de insectos 4](#_Toc153144568)

[Código 5](#_Toc153144569)

[Actividad 2: árboles de comportamiento 10](#_Toc153144570)

[Código 11](#_Toc153144571)

[Actividad 3: agentes tai-chi 16](#_Toc153144572)

[Video 16](#_Toc153144573)

[Repositorio 18](#_Toc153144574)

[Bibliografía 18](#_Toc153144575)

[Assets 18](#_Toc153144576)

[Flocking 18](#_Toc153144577)

[Árboles de comportamiento 18](#_Toc153144578)

[Movimientos de formación 18](#_Toc153144579)

# 

# 

# 

# 

# Herramientas

Para realizar esta actividad, se han utilizado las siguientes herramientas:

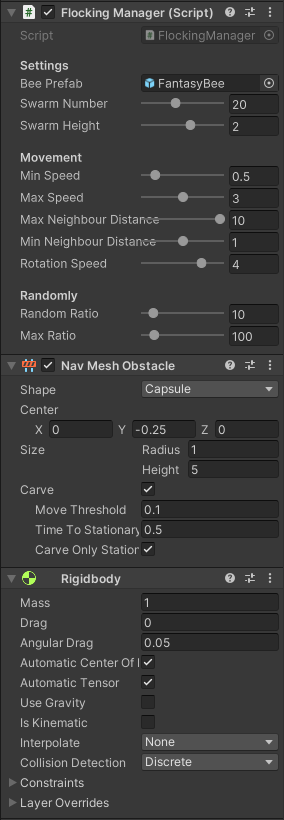
* ***Unity v2022.3.11f1***: plataforma de desarrollo de videojuegos
* ***JetBrains Rider v2022.3.3***: IDE para programar en el lenguaje de programación C#.
* ***AI Navigation v2.0.0***: Sistema inteligente que permite trasladar personajes en un escenario.
* ***Mixamo***: empresa que ofrece personajes 3D y animaciones.
* ***Hithub***: plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git.
* ***Microsoft Word v2018***: software de tratamiento de textos.

# Actividad 1: colmena de insectos

Para realizar está actividad, se ha implementado el algoritmo *flocking* definido en un [tutorial](https://learn.unity.com/tutorial/flocking) en la plataforma de aprendizaje de Unity. Se han creado los *scripts*:

* *FlockingManager*: gestiona la colmena. Crea los insectos que la integran y define una posición a la que acuden periódicamente.
* *Flock*: gestiona cada insecto de la colmena, su movimiento y rotación dentro de la colmena.

Tras implementar las clases, se ha buscado un modelo 3D de un insecto, en este caso el [recurso](https://assetstore.unity.com/packages/3d/characters/animals/fantasy-bee-135487) de una abeja en *Unity Asset Store*. A continuación se ha importado al proyecto.

Por último se ha implementado la colmena de insectos en el proyecto, para ello se ha creado un objeto *Swarm*, y se le ha agregado:

* *script FlockingManager* definiendo sus parámetros para que se ajusten lo mejor posible a un enjambre de abejas.
* componente [*Nav Mesh Obstacle*](https://docs.unity3d.com/2021.3/Documentation/Manual/class-NavMeshObstacle.html)para definir al objeto como un obstáculo en el [Sistema de Navegación de Unity](https://docs.unity3d.com/es/2019.4/Manual/nav-NavigationSystem.html). Se selecciona la opción *carve* para definir un obstáculo estático.

Se ha agregado el script *Flock* al *prefab Bee* para definir el insecto que forma parte de la colmena.

Tras ello, se ha rehecho la superficie válida para los agentes, creándose una zona no válida más grande en los árboles asignados con una colmena de insectos.  
No ha sido necesario agregar el componente Rigidbody para que el sistema de Navegación detecte al árbol con una colmena de insectos como obstáculo.

|  |  |
| --- | --- |
|  |  |
| *árbol con colmena* | *árbol sin colmena* |

## Código

El código está dentro de la carpeta *Script/Flocking*.

**FlockingManager.cs**

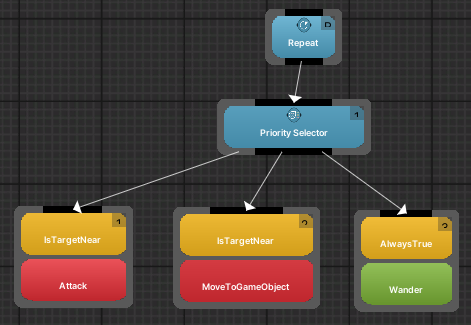
using UnityEngine;  
using UnityEngine.AI;  
using Random = UnityEngine.Random;  
  
namespace Script.Flocking  
{  
*/\*   
 \* Handles the behaviour of a swarm  
 \* The swarm bounds are set at the \_swarmLimits variable and defined with:  
 \* height -> as parameter  
 \* width and depth -> the nav mesh obstacle 3D geometry   
 \*/* public class **FlockingManager** : MonoBehaviour  
 {  
 [Header("Settings")]   
 [SerializeField] private GameObject **beePrefab**; *// the model use in the swarm* [SerializeField][Range(0, 50)] private int **swarmNumber** = 20; *// total number of elements in the swarm* [SerializeField][Range(0.5f, 3)] private float **swarmHeight**= 1.5f; *// height of the swarm* [Header("Movement")]   
 [SerializeField][Range(0, 4)] private float **minSpeed**= 0.5f; *// minimum speed of the elements in the swarm* public float MinSpeed => minSpeed;  
   
 [SerializeField][Range(1, 5)] private float **maxSpeed** = 3; *// maximum speed of the elements in the swarm* public float MaxSpeed => maxSpeed;  
   
 [SerializeField][Range(0, 2)] private float **minNeighbourDistance** = 1; *// minimum distance between the elements in the swarm* public float MinNeighbourDistance => minNeighbourDistance;  
   
 [SerializeField][Range(1, 10)] private float **maxNeighbourDistance** = 10; *// maximum distance between the elements in the swarm* public float MaxNeighbourDistance => maxNeighbourDistance;  
   
 [SerializeField][Range(1, 5)] private float **rotationSpeed** = 2; *// rotation speed of the elements in the swarm* public float RotationSpeed => rotationSpeed;  
   
 [Header("Randomness")]   
 [SerializeField][Range(0.1f, 100)] private float **randomRatio** = 10; *// ratio of an number included in a gap* [SerializeField][Range(50, 500)] private float **maxRatio** = 100; *// maximum value of the random gap   
   
   
 // private variables* private GameObject[] \_swarm; *// list of the elements in the swarm* public GameObject[] Swarm => \_swarm;  
   
 private Vector3 \_swarmLimits; *// bounds of the swarm* public Vector3 SwarmLimits => \_swarmLimits;  
   
 private Vector3 \_goalPosition = Vector3.zero; *// position to reach of the elements in the swarm* public Vector3 GoalPosition => \_goalPosition;  
  
 private NavMeshObstacle \_navMeshObstacle; *// obstacle to avoid on the navigation system, bounds set with a 3D geometry  
  
 // Initializes values* private void **Awake**()  
 {  
 \_navMeshObstacle = GetComponent<NavMeshObstacle>(); *// get the NavMeshObstacle component* if (swarmHeight > \_navMeshObstacle.height) *// checks if the height of the swarm is in bounds, is less than the NavMeshObstacle height 3D geometry* {  
 swarmHeight = \_navMeshObstacle.height; *// if the swarmHeight value is greater, set to the 3D geometry height* }  
 \_swarmLimits = new Vector3(\_navMeshObstacle.radius, swarmHeight, \_navMeshObstacle.radius); *// set the swarm bounds related to the NavMeshObstacle 3D geometry* }  
  
 *// Start is called before the first frame update* private void **Start**()  
 {  
 \_swarm = new GameObject[swarmNumber]; *// creates an array with a lenght of the swarmNumber* for (int i = 0; i < \_swarm.Length; i++) *// creates an element of the swarm in a random position inside the swarm bounds* {  
 Vector3 position = transform.position +  
 new Vector3(Random.Range(-\_swarmLimits.x, \_swarmLimits.x),  
 Random.Range(-\_swarmLimits.y, \_swarmLimits.y),  
 Random.Range(-\_swarmLimits.z, \_swarmLimits.z));  
 GameObject bee = Instantiate(beePrefab, position, Quaternion.identity);  
 bee.GetComponent<Flock>().FlockingManager = this;   
 bee.transform.parent = transform; *// set the swarm handler the parent of the element* \_swarm[i] = bee;  
 }  
  
 \_goalPosition = transform.position;  
 }  
  
 *// Update is called once per frame* private void **Update**()  
 {  
 if (IsRandomValue()) *// set a random position inside the swarm bounds* { *// to reach to all the elements in the swarm* \_goalPosition = transform.position +  
 new Vector3(Random.Range(-\_swarmLimits.x, \_swarmLimits.x),  
 Random.Range(-\_swarmLimits.y, \_swarmLimits.y),  
 Random.Range(-\_swarmLimits.z, \_swarmLimits.z));  
 }  
 }  
  
 *// checks if a random value is included in the gap* public bool IsRandomValue()  
 {  
 return Random.Range(0, maxRatio) < randomRatio;  
 }  
 }  
}

**Flock.cs**

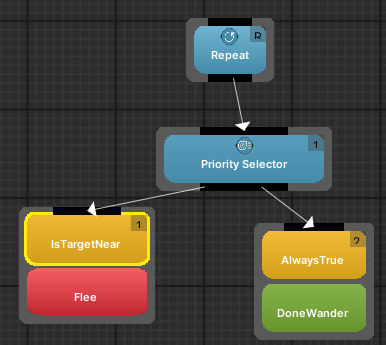
using Script.Game;  
using UnityEngine;  
using Random = UnityEngine.Random;  
  
namespace Script.Flocking  
{  
 */\*  
 \* Handles the behaviour of each element in the swarm  
 \*/* public class **Flock** : MonoBehaviour  
 {  
  
 private FlockingManager \_flockingManager; *// swarm handler* public FlockingManager FlockingManager  
 {  
 set => \_flockingManager = value;  
 }  
 private Animator \_animator; *// animator* private float \_speed; *// element current speed* private Bounds \_bounds; *// element's bounds* private readonly int \_boundsFactor = 2; *// bound's multiplier factor* private void **Awake**()  
 {  
 \_animator = GetComponent<Animator>();  
 }  
  
 *// Start is called before the first frame update* private void **Start**()  
 {  
 \_speed = Random.Range(\_flockingManager.MinSpeed, \_flockingManager.MaxSpeed); *// set randomly the initial speed  
 // set the element's bounds, its position is the handler swarm position, its bounds are the swarm bounds plus a multiplier factor* \_bounds = new Bounds(\_flockingManager.transform.position, \_flockingManager.SwarmLimits \* \_boundsFactor);   
 }  
  
 *// Update is called once per frame* private void **Update**()  
 {  
 if (!\_bounds.Contains(transform.position)) *// if the element's position is not in its bounds,* { *// redirect its orientation and movement to its bounds* Vector3 direction = \_flockingManager.transform.position - transform.position;  
 transform.rotation = Quaternion.Slerp(transform.rotation,   
 Quaternion.LookRotation(direction),   
 \_flockingManager.RotationSpeed \* Time.deltaTime);  
 }  
 else  
 {  
 if (\_flockingManager.IsRandomValue()) *// change the element's speed randomly* {  
 \_speed = Random.Range(\_flockingManager.MinSpeed, \_flockingManager.MaxSpeed);  
 }  
  
 if (\_flockingManager.IsRandomValue()) *// apply the flocking algorithm randomly* {  
 ApplyRules();  
 }  
 }  
 transform.Translate(0, 0, \_speed \* Time.deltaTime); *// move the element* \_animator.SetFloat(Constant.Animation.**SPEED**, \_speed); *// animate the element* }  
  
 *// flocking algorithm to the orientation on a swarm element  
 // orientation = group orientation + avoid orientation + group position* private void ApplyRules()  
 {  
 GameObject[] gameObjects = \_flockingManager.Swarm;  
  
 Vector3 center = Vector3.zero; *// relative position of the swarm elements* Vector3 avoid = Vector3.zero; *// position to avoid the neighbours swarm elements* float groupSpeed = 0.01f; *// relative speed of the swarm elements* int groupSize = 0; *// swarm elements near it* foreach (GameObject go in gameObjects) *// iterate through all the swarm elements to get the relative position and speed* {  
 if (go != gameObject) *// not take into account for the calculus itself* {  
 float neighbourDistance = Vector3.Distance(go.transform.position, transform.position); *// get the distance to the element* if (neighbourDistance <= \_flockingManager.MaxNeighbourDistance) *// check if the element is too far* {  
 center += go.transform.position; *// take the element's position into account to move along with it* groupSize++; *// increase the counter* if (neighbourDistance < \_flockingManager.MinNeighbourDistance) *// check f the element is too near* {  
 avoid += (transform.position - go.transform.position); *// take the element's position into account to avoid it* }  
  
 groupSpeed += go.GetComponent<Flock>().\_speed; *// take the element's speed into account to move along with it* }  
 }  
 }  
  
 if (groupSize > 0) *// it the are swarm elements near it, calculate its new orientation* {  
 *// get the related position taking into account its near elements, the goal position and its position* center = center / groupSize + (\_flockingManager.GoalPosition - transform.position);   
 \_speed = groupSpeed / groupSize; *// get the related speed taking into account its near elements* if (\_speed > \_flockingManager.MaxSpeed) *// checks if the speed exceeds its maximum speed* {  
 \_speed = \_flockingManager.MaxSpeed; *// set its speed to its maximum value* }  
  
 Vector3 direction = (center - avoid) - transform.position; *// get the related direction taking into account its near elements* if (direction != Vector3.zero)  
 { *// update its direction* transform.rotation = Quaternion.Slerp(transform.rotation,   
 Quaternion.LookRotation(direction),   
 \_flockingManager.RotationSpeed \* Time.deltaTime);  
 }  
 }  
  
 }  
 }  
}

## Video

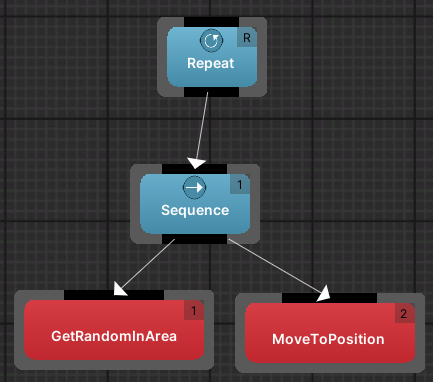
# Actividad 2: árboles de comportamiento

 Para esta actividad he representado a un monstruo que va tras una ciudadana. Para ello, he definido 2 árboles de comportamiento, uno para el monstruo donde se ha diseñado el siguiente árbol de comportamiento.

Por defecto deambula por el escenario mediante la acción *Wander*. Si se encuentra cerca de la ciudadana a una distancia de 15 unidades comprobada mediante la condición *IsTargetNear*, la empieza a perseguir mediante la acción MovetoGameObject. Y cuando está más cerca todavía, a una distancia de 2 unidades comprobada mediante la condición *IsTargetNear*, la ataca mediante la acción Attack hasta que la ciudadana se escapa de su distancia.

Para el comportamiento de la ciudadana, se ha diseñado el siguiente árbol de comportamiento. Es parecido al anterior, ya que tiene el mismo comportamiento por defecto, deambular por el escenario.

Cuando el monstruo está cerca a una distancia de 2 unidades comprobada mediante la condición *IsTargetNear,* se ejecuta la acción Flee, gritando y huyendo del monstruo.

Por último, el árbol de comportamiento DoneWander que realiza dos acciones de forma secuencial, la primera acción *GetRandomInArea* obtiene una posición aleatorio del escenario y la segunda, *MoveToPosition* hace que se mueva hacia la posición generada en la acción anterior.

Tras llegar al destino, se vuelve a ejecutar la primera acción.

Estas dos acciones se repiten de forma infinita.

# 

## Código

El código está dentro de la carpeta *Script/Monster* y *Script/CitizenFlee*

**AttackOnce.cs**

using BBUnity.Actions;  
using Pada1.BBCore;  
using Pada1.BBCore.Tasks;  
using UnityEngine;  
  
namespace Script.Monster  
{  
 [Action("MyActions/Attack")]  
 [Help("Periodically attacks the target. This action never ends.")]  
 public class AttackOnce : GOAction  
 {  
 *// Event raised when fled citizen is near.* public static event System.EventHandler OnAttack;  
 *// Define the input parameter delay, with the waited game loops between attacks.* [InParam("delay", DefaultValue=30)]  
 public int delay;  
   
 *// Game loops since the last attack.* private int elapsed = 0;  
   
 *///<value>Input Target Parameter to to check the distance.</value>* [InParam("target")]  
 [Help("The target that is close")]  
 public GameObject target;  
   
 *// Initialization method. If not established, we look for the shooting point.* public override void OnStart()  
 {  
 delay = 10;  
 }  
  
 *// Main class method, invoked by the execution engine.* public override TaskStatus OnUpdate()  
 {  
 if (delay > 0)  
 {  
 ++elapsed;  
 elapsed %= delay;  
 if (elapsed != 0)  
 return TaskStatus.**RUNNING**;  
 }  
   
 *// Trigger the attack* if (OnAttack != null)  
 OnAttack(this, System.EventArgs.Empty);  
 return TaskStatus.**RUNNING**;  
  
 } *// OnUpdate* }  
}

**FsmMonster.cs**

using Script.Citizen;  
using Script.Game;  
  
namespace Script.Monster  
{  
 */\*  
 \* Handle the behavior of the fled citizen  
 \* It inherit from FsmCitizen  
 \* Has the same behavior without the Destination State  
 \*/* public class **FsmMonster** : FsmCitizen  
 {  
  
 *// Start is called before the first frame update* private new void **Start**()  
 {  
 base.Start();  
 CurrentDestinationState = null;  
 AttackOnce.OnAttack += Attack; *// it trigger when is near a fled citizen* }  
  
 *// Update is called once per frame* private new void **Update**()  
 {  
 *// set the speed at the animator controller* Animator.SetFloat(Constant.Animation.**SPEED**, NavMeshAgent.speed);  
 CurrentMovementState.UpdateMovementState();  
 }  
  
 */\*  
 \* Triggers the attack animation at the animation controller  
 \*/* private void Attack(object sender, System.EventArgs e)  
 {  
 Animator.SetTrigger(Constant.Animation.ATTACK);  
 }  
  
 }  
}

**IsTargetNear.cs**

using System;  
using BBUnity.Conditions;  
using Pada1.BBCore;  
using UnityEngine;  
  
namespace Script.Monster  
{  
 [Condition("MyConditions/IsTargetNear")]  
 [Help("Checks whether the object is near another with a specific tag.")]  
 public class IsTargetNear : GOCondition  
 {  
 *///<value>Input maximum distance Parameter to consider that the target is close.</value>* [InParam("closeDistance")]  
 [Help("The maximum distance to consider that the target is close")]  
 public float closeDistance;  
   
 [InParam("targetName")]  
 [Help("The target name of the target")]  
 public String targetName;  
   
 *///<value>Input Target Parameter to to check the distance.</value>* [OutParam("target")]  
 [Help("The target that is close")]  
 public GameObject target;  
   
 public override bool Check()  
 {  
 *// loop for the game objects with a specific target* foreach (GameObject go in GameObject.FindGameObjectsWithTag(targetName))  
 {  
 if ((gameObject.transform.position - go.transform.position).sqrMagnitude <  
 closeDistance \* closeDistance) *// check if it is close* {  
 target = go; *// set the target* return true;  
 }  
   
 }  
 return false;  
 }  
 }  
}

**FleeOnce.cs**

using BBUnity.Actions;  
using Pada1.BBCore;  
using Pada1.BBCore.Tasks;  
  
namespace Script.Citizen.Flee  
{  
   
 [Action("MyActions/Flee")]  
 [Help("Triggers when is near the target.")]  
 public class FleeOnce : GOAction {  
   
 *// Event raised when sun rises or sets.* public static event System.EventHandler OnFlee;  
   
 *// Main class method, invoked by the execution engine.* public override TaskStatus OnUpdate()  
 {  
 *// Trigger the flee* if (OnFlee != null)  
 OnFlee(this, System.EventArgs.Empty);  
 return TaskStatus.**COMPLETED**;  
 }  
 }  
}

**FsmFled.cs**

using Script.Game;  
  
namespace Script.Citizen.Flee  
{  
 */\*  
 \* Handle the behavior of the fled citizen  
 \* It inherit from FsmCitizen  
 \* Has the same behavior without the Destination State  
 \*/* public class **FsmFled**: FsmCitizen  
 {  
 *// Start is called before the first frame update* private new void **Start**()  
 {  
 base.Start();  
 CurrentDestinationState = null;  
 FleeOnce.OnFlee += Flee; *// it trigger when is near a monster* }  
  
 *// Update is called once per frame* private new void **Update**()  
 {  
 *// set the speed at the animator controller* Animator.SetFloat(Constant.Animation.**SPEED**, NavMeshAgent.speed);  
 CurrentMovementState.UpdateMovementState();  
 }  
  
 */\*  
 \* Triggers the flee animation at the animation controller and doubles them speed to fred quicker  
 \*/* private void Flee(object sender, System.EventArgs e)  
 {  
 Animator.SetTrigger(Constant.Animation.FLEE);  
 NavMeshAgent.speed \*= 2;  
 }  
 }  
}

## Video

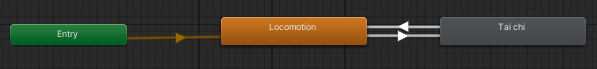
# Actividad 3: agentes tai-chi

Para realizar está actividad, se ha implementado el algoritmo *movimiento de formación* definido en una [presentación](https://www.cs.upc.edu/~escudero/gameAI/tactics/index.html#8) sobre estrategía y tácticas. Se han creado los *scripts*:

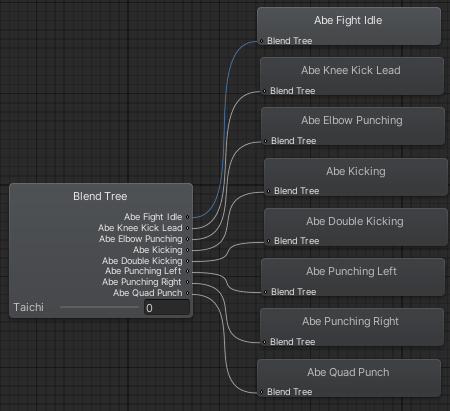
* *FormationManager*: gestiona la composición de la formación. Define el líder de la formación, a quién han de seguir los seguidores, así como el número de seguidores por fila y su número de filas.
* *Slot*: gestiona cada seguidor integrante de la formación. Su función es seguir al líder.

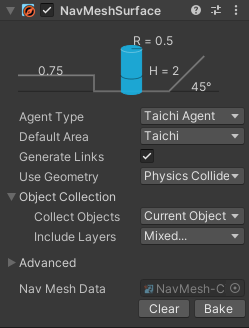
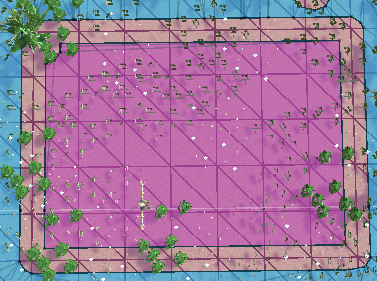
Tras implementar las clases, se ha buscado un modelo 3D y animaciones en la web [Mixamo](https://www.mixamo.com/). Se han agregado al proyecto y se ha definido el controlador de animaciones.

En el controlador de animación se ha agregado un árbol de animación llamado Taichi, como se muestra en la figura inferior



donde se definen todos los movimientos de taichi posibles.



También se ha definido una nueva área (‘Taichi’) y un nuevo agente (‘Taichi Agent’) en el [Sistema de Navegación de Unity](https://docs.unity3d.com/es/2019.4/Manual/nav-NavigationSystem.html) para definir el área válida de movimiento para estos agentes. En la foto inferior es el área morada. Esta área es válida solo para los agentes que practican taichi.

Por último, se ha definido el código para mover al líder por esta área válida heredando de la clase FsmCitizen y cada cierto tiempo hace una serie de posturas que los seguidores imitan.

## Video

# 

# 

# Repositorio

La URL del repositorio GIT es: <https://github.com/JoaquinLopezSoriano/IA-PEC2.git>

# Bibliografía

## Assets

* Abeja: <https://assetstore.unity.com/packages/3d/characters/animals/fantasy-bee-135487>
* Escenario: <https://assetstore.unity.com/packages/3d/environments/urban/city-package-107224>
* Modelos y animaciones: <https://www.mixamo.com/>

## Flocking

* Algoritmo: <https://learn.unity.com/tutorial/flocking#6317c5d8edbc2a2290a9e35e>
* Obstáculo: <https://docs.unity3d.com/2021.3/Documentation/Manual/class-NavMeshObstacle.html>

## Árboles de comportamiento

* <https://bb.padaonegames.com/doku.php>

## Movimientos de formación

* Algoritmo: <https://www.cs.upc.edu/~escudero/gameAI/tactics/index.html#8>
* Definir la máscara de área: <https://forum.unity.com/threads/navmesh-sampleposition-always-returns-false-why.1360879/>