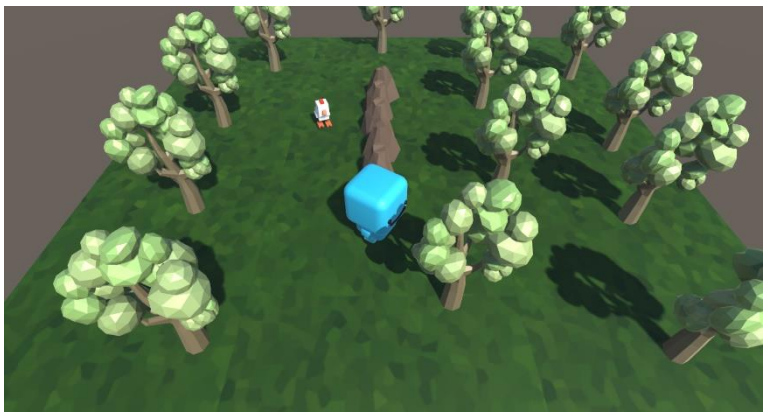
Anexo 8

Escena de demostración de máquinas de estados: el chico contra la gallina

En esta escena de demostración de máquinas de estados se presenta a un chico (muñeco azul) que podrá ser controlado por el usuario, indicándole el punto al que quiere que se mueva mediante el botón izquierdo del ratón. El muñeco se moverá a esa posición correctamente a menos que la gallina se acerque demasiado, en ese caso empezará a correr despavorido y el usuario perderá el control sobre él.



La gallina se irá moviendo aleatoriamente por el mapa cada cierto tiempo, pero si el chico entra en su campo de visión ira detrás de él persiguiéndole, hasta alcanzarle o perderle de vista.

Figura 29. Chico huyendo de la gallina

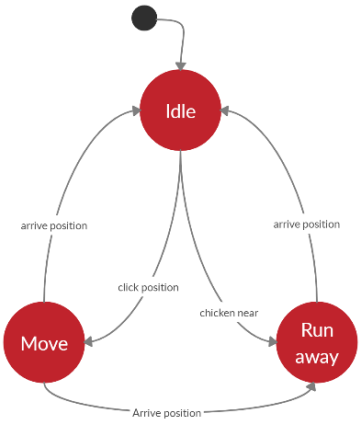


Figura 30. Máquina de estados del chico

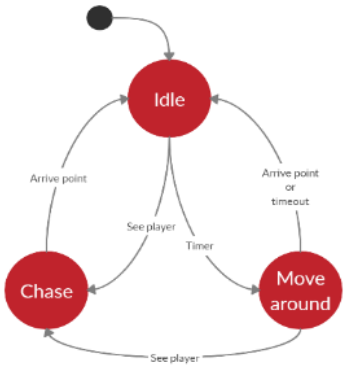


Figura 31. Máquina de estados de la gallina

Fragmentos de código

```
private void CreateStateMachine()
{
    // Perceptions
    Perception click = testBoyFSM.CreatePerception<PushPerception>();
    Perception clickOnMoving = testBoyFSM.CreatePerception<PushPerception>();
    arriveToDestination = testBoyFSM.CreatePerception<ArriveToDestination>(new ArriveToDestination());
    Perception timeToStopRunning = testBoyFSM.CreatePerception<TimerPerception>(7);
    Perception stopRunningAway = testBoyFSM.CreateOrPerception<OrPerception>(arriveToDestination, timeToStopRunning);
    Perception chickenNear = testBoyFSM.CreatePerception<ValuePerception>(() => distanceToChicken < minDistanceToChicken);

    // States
    State idleState = testBoyFSM.CreateEntryState("Idle");
    State movingState = testBoyFSM.CreateState("Moving", Move);
    State runAwayState = testBoyFSM.CreateState("Run away", RunAway);

    // Transitions
    testBoyFSM.CreateTransition("mouse clicked", idleState, click, movingState);
    testBoyFSM.CreateTransition("get to destination from moving", movingState, stopRunningAway, idleState);
    testBoyFSM.CreateTransition("chicken near from idle", idleState, chickenNear, runAwayState);
    testBoyFSM.CreateTransition("get to destination from run away", runAwayState, stopRunningAway, idleState);
    testBoyFSM.CreateTransition("chicken near from moving", movingState, chickenNear, runAwayState);
    testBoyFSM.CreateTransition("change destination", movingState, clickOnMoving, movingState);
}
```

Figura 32. Fragmento de código para la creación de la máquina de estados del chico

```
private void CreateStateMachine()
{
    // Perceptions
    WatchingPerception seePlayer = chickenFSM.CreatePerception<WatchingPerception>(new WatchingPerception(gameObject, target, visionCollider));
    arriveToDestination = chickenFSM.CreatePerception<ArriveToDestination>(new ArriveToDestination());
    Perception moveTimeout = chickenFSM.CreatePerception<TimerPerception>(5);
    Perception getDestinationOrTimeout = chickenFSM.CreateOrPerception<OrPerception>(moveTimeout, arriveToDestination);
    Perception timer = chickenFSM.CreatePerception<TimerPerception>(15);

    // States
    State idleState = chickenFSM.CreateEntryState("Idle", Idle);
    State chaseState = chickenFSM.CreateState("Chase", Chase);
    State moveState = chickenFSM.CreateState("Move around", MoveAround);

    // Transitios
    chickenFSM.CreateTransition("see the player from idle", idleState, seePlayer, chaseState);
    chickenFSM.CreateTransition("move randomly", idleState, timer, moveState);
    chickenFSM.CreateTransition("get to destination from chase", chaseState, arriveToDestination, idleState);
    //chickenFSM.CreateTransition("keep chasing", chaseState, seePlayer, chaseState);
    chickenFSM.CreateTransition("get to destination from move around", moveState, getDestinationOrTimeout, idleState);
    chickenFSM.CreateTransition("see the player from move around", moveState, seePlayer, chaseState);
}
```

Figura 33. Fragmento de código para la creación de la máquina de estados de la gallina

Anexo 9

Escena de demostración de máquinas de estados jerárquicas: el radar

En esta escena vemos un radar en una autopista por la que van pasando diversos coches a distintas velocidades. El radar irá detectando la velocidad a la que van los distintos coches, si esta velocidad supera el límite (120) aparecerá una luz roja, si es correcta aparecerá una luz verde como en la imagen y si se estropea aparecerá una luz amarilla intermitente y los coches irán más rápido.

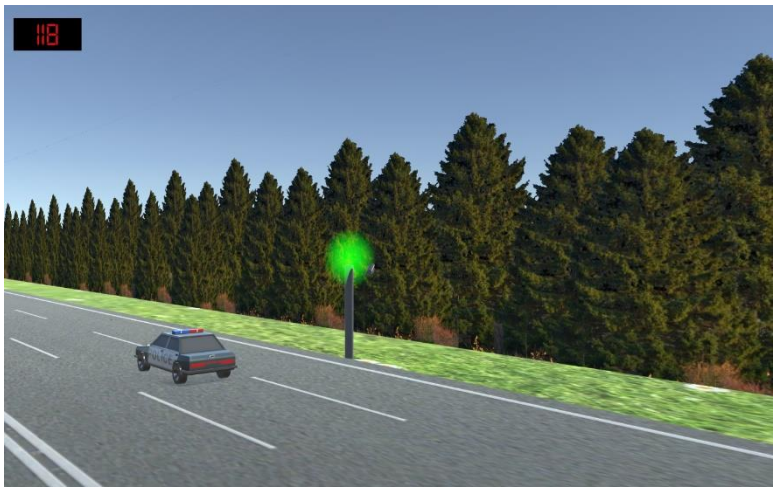


Figura 34. Captura de pantalla de la escena. La luz indica si es correcta o no la velocidad y arriba a la izquierda muestra la velocidad

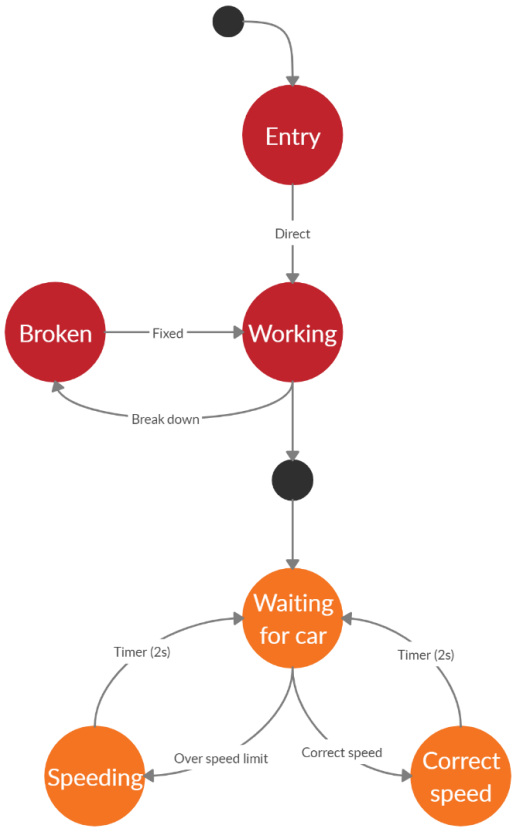


Figura 35. Diagrama de estados de la máquina de estados y submáquina de estados del radar

Fragmentos de código

```
private void CreateStateMachine()
{
    radarFSM = new StateMachineEngine(false);

    // Perceptions
    Perception direct = radarFSM.CreatePerception<PushPerception>();
    Perception breakDown = radarFSM.CreatePerception<TimerPerception>(30);
    Perception fix = radarFSM.CreatePerception<TimerPerception>(15);

    // States
    State entryState = radarFSM.CreateEntryState("Entry State");
    State workingState = radarFSM.CreateSubStateMachine("Working", workingSubFSM);
    brokenState = radarFSM.CreateState("Broken", Broken);

    // Transitions
    radarFSM.CreateTransition("Direct", entryState, direct, workingState);
    workingSubFSM.CreateExitTransition("To broken", workingState, breakDown, brokenState);
    radarFSM.CreateTransition("To working", brokenState, fix, workingState);

    radarFSM.Fire("Direct");
}
```

Figura 36. Fragmento de código para la creación de la máquina de estados principal del radar

```
private void CreateSubMachine()
{
    workingSubFSM = new StateMachineEngine(true);

    // Perceptions
    detectCar = workingSubFSM.CreatePerception<DetectCar>(new DetectCar(gameObject, pointToLook));
    Perception carOverSpeed = workingSubFSM.CreatePerception<ValuePerception>(() => detectCar.GetCarSpeed() > 20);
    Perception carOnSpeed = workingSubFSM.CreatePerception<ValuePerception>(() => detectCar.GetCarSpeed() <= 20);
    Perception overSpeedLimit = workingSubFSM.CreateAndPerception<AndPerception>(detectCar, carOverSpeed);
    Perception onSpeedLimit = workingSubFSM.CreateAndPerception<AndPerception>(detectCar, carOnSpeed);
    Perception timeout = workingSubFSM.CreatePerception<TimerPerception>(2);

    // States
    State waitingForCarState = workingSubFSM.CreateEntryState("Waiting for car", OnWaitingForCar);
    State speedingState = workingSubFSM.CreateState("Speeding", OnSpeeding);
    State correctSpeedState = workingSubFSM.CreateState("Correct speed", OnCorrectSpeed);

    // Transitions
    workingSubFSM.CreateTransition("Car over speed limit", waitingForCarState, overSpeedLimit, speedingState);
    workingSubFSM.CreateTransition("Car on speed limit", waitingForCarState, onSpeedLimit, correctSpeedState);
    workingSubFSM.CreateTransition("To waiting for next bad car", speedingState, timeout, waitingForCarState);
    workingSubFSM.CreateTransition("To waiting for next good car", correctSpeedState, timeout, waitingForCarState);
}
```

Figura 37. Fragmento de código para la creación de la submáquina de estados del radar

Anexo 10

Escena de demostración de árboles de comportamiento: entrar en la casa

En esta escena de árboles de comportamientos el muñeco azul intentará entrar en su casa. El jugador podrá modificar, mediante el selector de arriba a la derecha, el estado de la puerta. Teniendo tres opciones que modificarán las acciones del muñeco.

La primera opción de 'puerta abierta', el muñeco llegará a la puerta y la abrirá sin problemas pudiendo entrar en la casa. En la segunda opción de 'puerta candada', el muñeco llegará a la puerta verá que no puede entrar y entonces irá a por la llave para volver a la puerta y poder abrirla. Y, por último, la última opción de 'puerta cerrada' el muñeco irá a la puerta verá que no puede entrar y que no hay ninguna llave por lo que decidirá explotar la puerta para entrar, situaciones desesperadas requieren de medidas desesperadas.

Figura 38. Captura de pantalla de la escena. Muñeco azul yendo a buscar la llave para entrar en la casa

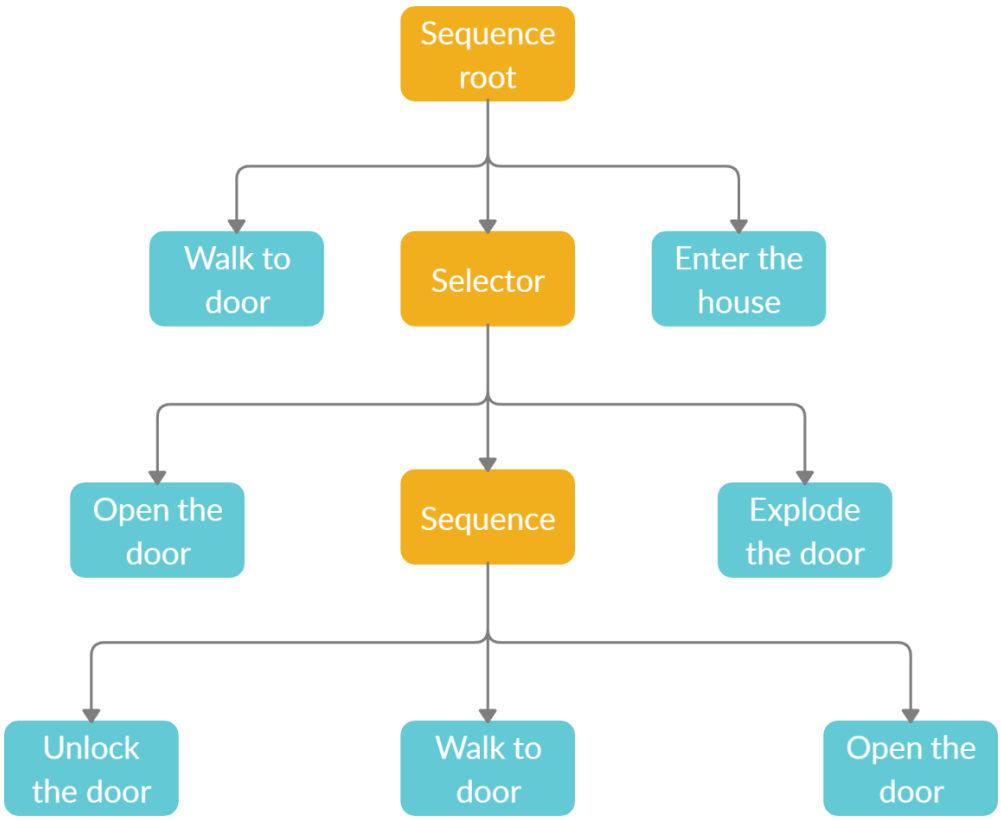


Figura 39. Diagrama del árbol de comportamientos del muñeco azul

Fragmentos de código

```
private void CreateBehaviourTree()
{
    behaviourTree = new BehaviourTreeEngine(false);

    rootSequence = behaviourTree.CreateSequenceNode("Root", false);
    selectorDoor = behaviourTree.CreateSelectorNode("Selector door");
    sequenceDoor = behaviourTree.CreateSequenceNode("Sequence door", false);
    walkToDoor1 = behaviourTree.CreateLeafNode("Walk to door 1", WalkToDoor, ArriveToDoor);
    walkToDoor2 = behaviourTree.CreateLeafNode("Walk to door 2", WalkToDoor, ArriveToDoor);
    enterTheHouse = behaviourTree.CreateLeafNode("Enter the house", EnterTheHouse, HasEnteredTheHouse);
    openDoor1 = behaviourTree.CreateLeafNode("Open the door 1", OpenDoor, DoorOpened);
    openDoor2 = behaviourTree.CreateLeafNode("Open the door 2", OpenDoor, DoorOpened);
    unlockDoor = behaviourTree.CreateLeafNode("Find key", UnlockDoor, IsTheDoorUnlocked);
    smashDoor = behaviourTree.CreateLeafNode("Explode door", SmashDoor, DoorSmashed);

    rootSequence.AddChild(walkToDoor1);
    rootSequence.AddChild(selectorDoor);
    rootSequence.AddChild(enterTheHouse);

    selectorDoor.AddChild(openDoor1);
    selectorDoor.AddChild(sequenceDoor);
    selectorDoor.AddChild(smashDoor);

    sequenceDoor.AddChild(unlockDoor);
    sequenceDoor.AddChild(walkToDoor2);
    sequenceDoor.AddChild(openDoor2);

    behaviourTree.SetRootNode(rootSequence);
}
```

Figura 40. Fragmento de código para la creación del árbol de comportamientos del muñeco azul

Anexo 11

Escena de demostración de nodos decoradores en un árbol de comportamientos: ¡De pesca!

En esta escena el muñeco azul pasará la tarde pescando. Cada cierto tiempo pescará dos cosas: una bota vieja o un pez. Si pesca la bota vieja la devolverá al río, en cambio, si pesca un pez lo guardará en su cesta.

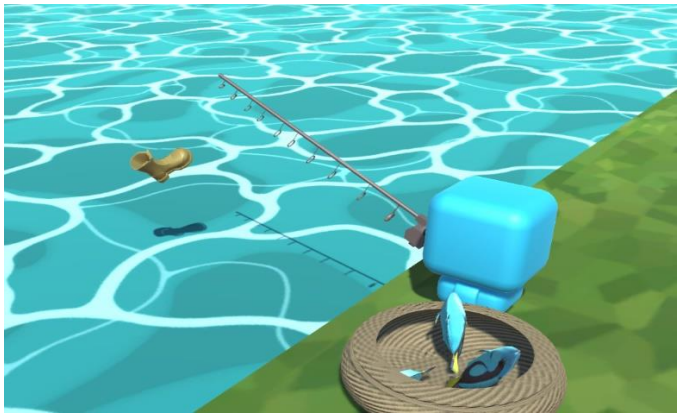


Figura 41. Captura de pantalla de la escena. El muñeco azul pescando una bota vieja ¡Qué mala suerte!

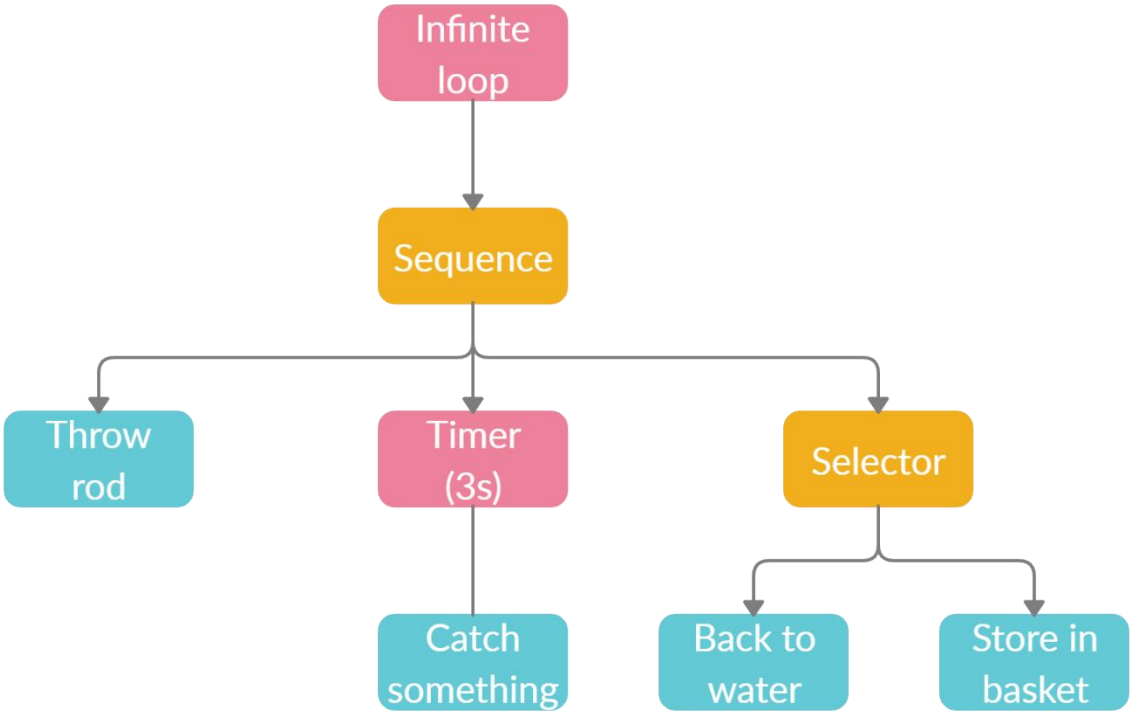


Figura 42. Diagrama del árbol de comportamientos del muñeco azul

Fragmentos de código

```
private void CreateBehaviourTree()
{
    behaviourTree = new BehaviourTreeEngine(false);

    LeafNode throwRod = behaviourTree.CreateLeafNode("Throw rod", ThrowRod, RodThrown);
    LeafNode catchSomething = behaviourTree.CreateLeafNode("Catch something", Catch, SomethingCatched);
    LeafNode returnToWater = behaviourTree.CreateLeafNode("Return to water", () => Invoke("ReturnToWater", 2), ReturnedToWater);
    LeafNode storeInBasket = behaviourTree.CreateLeafNode("Store in the basket", () => Invoke("StoreBasket", 2), StoredInBasket);
    TimerDecoratorNode timerNode = behaviourTree.CreateTimerNode("Timer node", catchSomething, 3);
    SelectorNode selectorNode = behaviourTree.CreateSelectorNode("Selector node");
    selectorNode.AddChild(returnToWater);
    selectorNode.AddChild(storeInBasket);
    SequenceNode sequenceNode = behaviourTree.CreateSequenceNode("Sequence node", false);
    sequenceNode.AddChild(throwRod);
    sequenceNode.AddChild(timerNode);
    sequenceNode.AddChild(selectorNode);
    LoopDecoratorNode rootNode = behaviourTree.CreateLoopNode("Root node", sequenceNode);

    behaviourTree.SetRootNode(rootNode);
}
```

Figura 43. Fragmento de código para la creación del árbol de comportamientos

Anexo 12

Escena de demostración de un árbol de comportamientos dentro de una máquina de estados: entrar en la casa custodiada por un enemigo

En esta escena habrá dos muñecos de distinto color, rojo y azul. El muñeco rojo tan solo dar vueltas alrededor de la casa, protegiéndola. Mientras que el muñeco azul intentará entrar en la casa: para ello primero se dirigirá hacia la puerta, una vez descubra que está cerrada irá a por la llave y cuando la tenga volverá a la puerta para abrirla y entrar en la casa. Pero si en algún momento, el muñeco rojo pasa cerca del azul este correrá a un punto aleatorio del mapa y no volverá a reanudar su rutina hasta que no esté lo suficientemente lejos.

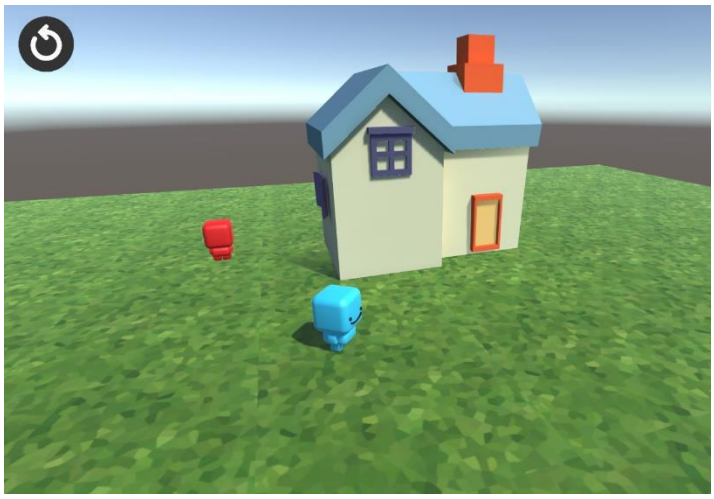


Figura 44. Captura de pantalla de la escena. El muñeco azul ya ha recogido la llave y huye del muñeco rojo, después se dirigirá hacia la puerta

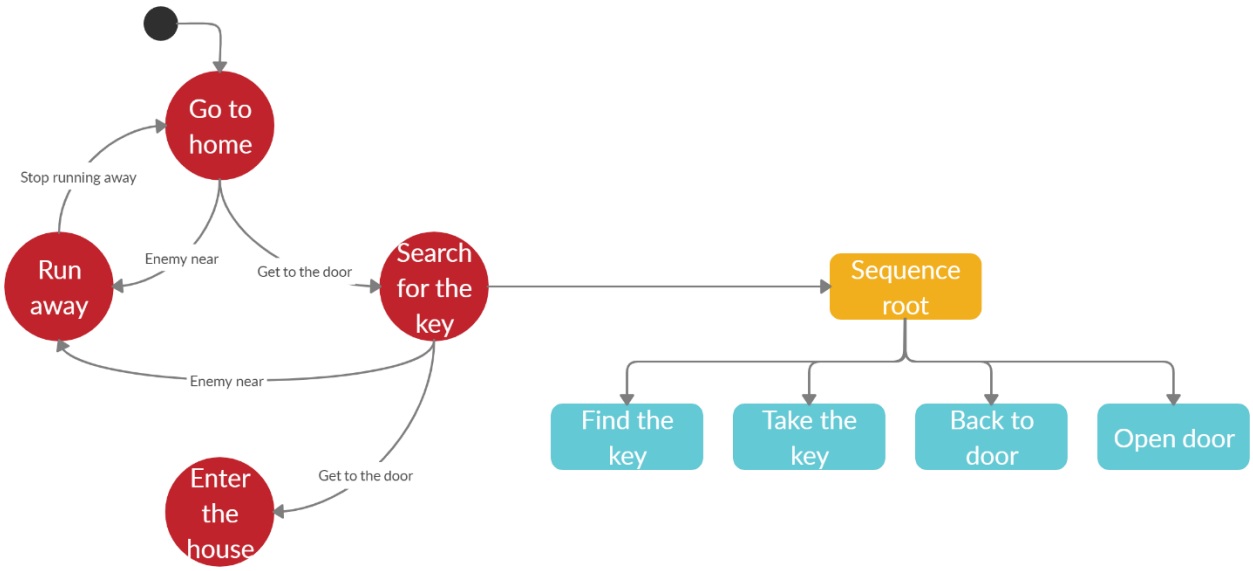


Figura 45. Diagrama de la máquina de estados y el subárbol del muñeco azul

Fragmentos de código

```
private void CreateFiniteStateMachine()
{
    stateMachine = new StateMachineEngine(false);

    arriveToHouse = stateMachine.CreatePerception<ArriveToDestination>(new ArriveToDestination());
    arriveToHouse.SetDestination(housePoint.position);
    ValuePerception enemyNear = stateMachine.CreatePerception<ValuePerception>(() => Vector3.Distance(transform.position, badBoy.transform.position) < 8);
    BehaviourTreeStatusPerception enterTheHouse = stateMachine.CreatePerception<BehaviourTreeStatusPerception>(behaviourTree, ReturnValues.Succeed);
    pointToRun = stateMachine.CreatePerception<ArriveToDestination>(new ArriveToDestination());

    State goToHouse = stateMachine.CreateEntryState("Go to house", ToHouse);
    State subBehaviourTree = stateMachine.CreateSubStateMachine("Sub behaviour tree", behaviourTree);
    State runAway = stateMachine.CreateState("Run away", RunAway);
    State enterHouse = stateMachine.CreateState("Enter in house", EnterHouse);

    stateMachine.CreateTransition("To enter the house", goToHouse, arriveToHouse, subBehaviourTree);
    stateMachine.CreateTransition("To run away", goToHouse, enemyNear, runAway);
    behaviourTree.CreateExitTransition("Run away", subBehaviourTree, enemyNear, runAway);
    behaviourTree.CreateExitTransition("Enter the house", subBehaviourTree, enterTheHouse, enterHouse);
    stateMachine.CreateTransition("Stop running", runAway, pointToRun, goToHouse);
}
```

Figura 46. Fragmento de código para la creación de la máquina de estados

```
private void CreateBehaviourTree()
{
    behaviourTree = new BehaviourTreeEngine(true);

    LeafNode toKey = behaviourTree.CreateLeafNode("To key", GoToKey, ArriveToKey);
    LeafNode getKey = behaviourTree.CreateLeafNode("Get key", GetKey, KeyTaken);
    LeafNode backToDoor = behaviourTree.CreateLeafNode("Back to door", ToDoor, ArriveToDoor);
    LeafNode openDoor = behaviourTree.CreateLeafNode("Open door", OpenDoor, DoorOpened);
    SequenceNode sequenceRoot = behaviourTree.CreateSequenceNode("Sequence root", false);
    sequenceRoot.AddChild(toKey);
    sequenceRoot.AddChild(getKey);
    sequenceRoot.AddChild(backToDoor);
    sequenceRoot.AddChild(openDoor);

    behaviourTree.SetRootNode(sequenceRoot);
}
```

Figura 47. Fragmento de código para la creación del subárbol de comportamientos

Anexo 13

Escena de demostración de una máquina de estados dentro de un árbol de comportamientos: ¡A cocinar!

En esta escena el muñeco azul se pone el gorro de chef para cocinar todas las recetas de pizza que le vayan llegando. Se encargará de: mirar la receta, poner la masa, el tomate y los ingredientes correspondientes. Una vez hecho todo esto la llevará al horno.

Figura 48. Captura de pantalla de la escena. El muñeco chef acaba de terminar una pizza y la lleva al horno

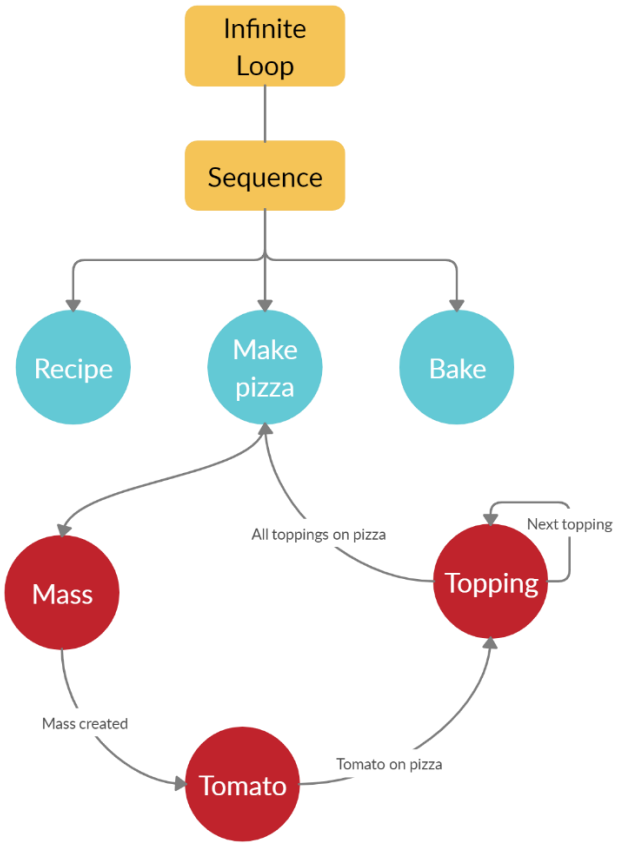


Figura 49. Diagrama de estados del árbol de comportamientos junto a la submáquina de estados del chef

Fragmentos de código

```
private void CreateBehaviourTree()
{
    LeafNode lookRecipe = behaviourTree.CreateLeafNode("Look recipe", CreateRecipe, IsRecipeCreated);
    LeafNode bake = behaviourTree.CreateLeafNode("Bake pizza", BakePizza, IsPizzaBaked);
    SequenceNode makePizza = behaviourTree.CreateSequenceNode("Make a pizza", false);
    makePizza.AddChild(lookRecipe);
    makePizza.AddChild(subFSM);
    makePizza.AddChild(bake);
    LoopDecoratorNode loopNode = behaviourTree.CreateLoopNode("Loop root", makePizza);

    behaviourTree.SetRootNode(loopNode);
}
```

Figura 50. Fragmentos de código para la creación del árbol de comportamientos

```
private void CreateStateMachine()
{
    // Perceptions
    massPutted = stateMachine.CreatePerception<TimerPerception>(1);
    tomatoPutted = stateMachine.CreatePerception<TimerPerception>(1);
    nextTopping = stateMachine.CreatePerception<TimerPerception>(1);
    allToppings = stateMachine.CreatePerception<PushPerception>();

    // States
    State putMass = stateMachine.CreateEntryState("Put mass", PutMass);
    State putTomato = stateMachine.CreateState("Put tomato", PutTomato);
    State putTopping = stateMachine.CreateState("Put topping", PutTopping);

    // Transitions
    stateMachine.CreateTransition("Mass putted", putMass, massPutted, putTomato);
    stateMachine.CreateTransition("Tomato putted", putTomato, tomatoPutted, putTopping);
    stateMachine.CreateTransition("Next topping", putTopping, nextTopping, putTopping);

    // Super-node of the Behaviour Tree and exit transition
    subFSM = behaviourTree.CreateSubBehaviour("Sub-FSM", stateMachine, putMass);
    stateMachine.CreateExitTransition("Back to BT", putTopping, allToppings, ReturnValues.Succeed);
}
```

Figura 51. Fragmento de código para la creación de la submáquina de estados