ICE TASK1 code explanation

Project link  
https://github.com/JoshuaMiller-Pio/AI\_FiniteStatemachine.git

public enum FiniteStateMachine  
{  
 **Idle**,  
 **Chase**,  
 **Attack**}

Within my finite state machine there are 3 different states as declared above: Idle, chase and attack.

void **Start**()  
{  
 currentState = FiniteStateMachine.**Idle**;  
 agent = gameObject.GetComponent<NavMeshAgent>();  
}

Upon starting the program the current state is set to Idle and the navmesh agent is declared.

void **Update**()  
{  
 switch (currentState)  
 {  
 case FiniteStateMachine.**Idle**:

isIdle();

break;

case FiniteStateMachine.**Chase**:  
 isChasing();

break;

case FiniteStateMachine.**Attack**:

isAttacking();

break;  
 }  
}

to handle switching of states there is a switch statement constantly applying the current state within the update method.

*//applies the idle state*public void isIdle()  
{  
 Debug.Log("idle");  
 if (agent.remainingDistance == 0)  
 {  
   
 Vector3 destination = new Vector3(agent.transform.position.x +Random.Range(-5, 5), agent.transform.position.y,  
 agent.nextPosition.z + Random.Range(-5, 5));  
   
 agent.SetDestination(destination);  
   
 }  
   
}

Within the idle state the nav mesh agent will select a random point within a -5 to 5 distance away from it on the X and Z axis and move around constantly to simulate a wondering patrol system.

*//applies the chasing state*public void isChasing()  
{  
   
 if (DistanceToPlayer(agent.transform.position.x, player.transform.position.x,agent.transform.position.y, player.transform.position.y ) > 2)  
 {  
 agent.SetDestination(player.gameObject.transform.position);  
   
 }  
 Debug.Log("chasing");  
 if (DistanceToPlayer(agent.transform.position.x, player.transform.position.x,agent.transform.position.y, player.transform.position.y ) <= 1.5f)  
 {  
 currentState = FiniteStateMachine.**Attack**;  
 }  
   
}*//checks to see if player is within range and switches to chase mode*private void **OnTriggerEnter**(Collider other)  
{  
   
 if (other.gameObject.tag == "Player")  
 {  
 currentState = FiniteStateMachine.**Chase**;  
 agent.SetDestination(player.gameObject.transform.position);  
 }  
}

When the player enters the trigger around the agent, they enter into the chase state as defined within the OnTriggerEnter method, then the method isChasing is called, the method does a distance check between the agent and the player to determine whether it is close enough to attack or if it should move towards the player if it is close enough to attack it changes states to the attack.

*//applies the attacking state* public void isAttacking()  
 {  
 if (DistanceToPlayer(agent.transform.position.x, player.transform.position.x,agent.transform.position.y, player.transform.position.y ) > 1.4f)  
 {  
 currentState = FiniteStateMachine.**Chase**;  
 }  
 Debug.Log("attacking");  
 }

Within the attacking state it calls another distance check if it is not within range it will set itself back to chase state, else it will display the message attack.

*//used to calculate the distance to the player*private float DistanceToPlayer(float x1, float x2, float y1, float y2)  
{  
 float distance = (float)Math.Sqrt(math.pow((x2-x1),2) + math.pow((y2-y1),2)) ;  
 return distance;  
}

The distance to player method takes in 4 variables and calculates the distance between itself and the player, this could be cleaned up by constantly updating the players co-ordinates in the update method, however I felt that calling the variables and passing them over only when needed would be a better use of resources especially if there are multiple AI agents in a larger project.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated