**AI Development Assignment 1 – Scripts**

# **Task 1 – State System**

## **StateDrivenBrain0**

using UnityEngine;  
using System.Collections;  
  
public class StateDrivenBrain0 : BasicAIController0{  
    *// Declares the constant state names*  
    public enum ChopperStates { Idle0, TakeOff0, Hover0, Forward0, Land0 };  
    *// Sets up state machine variable*  
    public FSM<ChopperStates> chopperStateMachine;  
    *// Puts a 0.4s interval on the state machine*  
    protected float thinkInterval = 0.4f;  
    *// Initialises chopperStateActive to true*  
    public bool chopperStateActive = true;  
    *// Sets up transform variables for these GameObjects*  
    public Transform HoverLocation1;  
    public Transform HoverLocation2;  
    public Transform LandingPad;  
    public Transform RotationPoint;  
  
    *// Initialises two boolean variables to false*  
    public bool Landed = false;  
    public bool HoverPosition = false;  
  
    protected void Awake()  
    {  
        *// Sets up a new FSM*  
        chopperStateMachine = new FSM<ChopperStates>();  
        *// This adds in all the new states into the state machine using variables declared*  
        chopperStateMachine.AddState(new Idle0<ChopperStates>(ChopperStates.Idle0, this, 0f));  
        chopperStateMachine.AddState(new TakeOff0<ChopperStates>(ChopperStates.TakeOff0, this, 0f));  
        chopperStateMachine.AddState(new Hover0<ChopperStates>(ChopperStates.Hover0, this, 0f));  
        chopperStateMachine.AddState(new Forward0<ChopperStates>(ChopperStates.Forward0, this, 0f));  
        chopperStateMachine.AddState(new Land0<ChopperStates>(ChopperStates.Land0, this, 0f));  
        *// Sets Idle as first state to be in once plays been activated*  
        chopperStateMachine.SetInitialState (ChopperStates.Idle0);  
  
        *// These are all the transitions of the states I will require within this program*  
        chopperStateMachine.AddTransition (ChopperStates.Idle0, ChopperStates.TakeOff0);  
        chopperStateMachine.AddTransition (ChopperStates.TakeOff0, ChopperStates.Hover0);  
        chopperStateMachine.AddTransition (ChopperStates.Hover0, ChopperStates.Forward0);  
        chopperStateMachine.AddTransition (ChopperStates.Forward0, ChopperStates.Hover0);  
        chopperStateMachine.AddTransition (ChopperStates.Hover0, ChopperStates.Land0);  
        chopperStateMachine.AddTransition (ChopperStates.Land0, ChopperStates.Idle0);  
    }  
  
    *// These Guards determine the conditions states need to meet to be initialised*  
    public bool GuardIdle0ToTakeOff0(State<ChopperStates> currentState)  
    {  
        *// Instantiates once delay ends and if landed is false*  
        return (delay <= 0.0f && Landed == false);  
    }  
    public bool GuardTakeOff0ToHover0(State<ChopperStates> currentState)  
    {  
        *// Changes state when helicopter reaches GameObjects Y position*  
        return (transform.position.y >= HoverLocation1.position.y);  
    }  
    public bool GuardHover0ToForward0(State<ChopperStates> currentState)  
    {  
        *// Instantiates once delay ends and when helicopters Z position is not equal to the GameObject*  
        return (delay <= 0.0f && transform.position.z != HoverLocation2.position.z);  
    }  
    public bool GuardForward0ToHover0(State<ChopperStates> currentState)  
    {  
        *// Instantiates when the boolean is true*  
        return (HoverPosition == true && transform.position.Equals(HoverLocation2.position));  
    }  
    public bool GuardHover0ToLand0(State<ChopperStates> currentState)  
    {  
        *// Instantiates once delay ends*  
        return (delay <= 0.0f);  
    }  
    public bool GuardLand0ToIdle0(State<ChopperStates> currentState)  
    {  
        *// Changes state when the helicopters Y coordinate is the same as the GameObjects*  
        return (transform.position.y <= LandingPad.position.y);  
    }  
  
    public void Start()  
    {  
        *// Calls to Think function*  
        StartCoroutine(Think());  
    }  
  
    public void Update()  
    {  
        base.Update ();  
        *// Refers to all act callbacks within other scripts*  
        if (chopperStateActive)   
        {  
            chopperStateMachine.CurrentState.Act ();  
        }  
    }  
  
    protected IEnumerator Think()  
    {  
        yield return new WaitForSeconds(thinkInterval);  
        *// Checks state machine*  
        chopperStateMachine.Check();  
        *// Calls to Think function*  
        StartCoroutine(Think());  
    }  
  
    void OnTriggerEnter(Collider col)  
    {  
        *// Once box collider has been entered set HoverPosition to true*  
        HoverPosition = true;  
    }  
}

## **BasicAIController0**

using UnityEngine;  
using System.Collections;  
  
*// This script sets up the delay that the chopper waits for before entering it's next state*  
  
public class BasicAIController0 : MonoBehaviour {  
  
    *// This float variable sets delay's initial value to 2.5*  
    public float delay = 2.5f;  
  
    void Start ()   
    {  
  
    }  
  
    *// Time.deltaTime needs to be updated constantly and also as it's inherited in other scripts*  
    public void Update ()   
    {  
        *// Subtracting Time.deltaTime from delay starts a countdown in seconds*  
        delay -= Time.deltaTime;  
    }  
}

## **AIState0**

using UnityEngine;  
using System.Collections;  
  
*// The AIState script inherits from the State Machine*  
public class AIState0<T> : State<T> {  
    *// Sets a variable to enable referring to StateDrivenBrain script as brain*  
    protected StateDrivenBrain0 brain;  
  
    *// Sets up FSM*  
    public AIState0(T stateName, StateDrivenBrain0 brain, float minDuration): base(stateName, brain, minDuration)   
    {  
        this.brain = brain;  
    }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void OnStateTriggerEnter(Collider collider)   
    {  
  
    }  
  
    public override void Monitor()   
    {  
  
    }  
  
    public override void Act()   
    {  
  
    }  
}

## **Idle0**

using UnityEngine;  
using System.Collections;  
  
*// The idle state is for when the Helicopter has no functions to carry out*  
  
*// The Idle class inherits from the AIState script*  
public class Idle0<T> : AIState0<T> {  
  
    *// Sets up FSM*  
    public Idle0(T stateName, StateDrivenBrain0 controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// No instructions are needed as the helicopter is idle*  
    }  
}

## **TakeOff0**

using UnityEngine;  
using System.Collections;  
  
*// The TakeOff state is for upwards movement of the Helicopter*  
  
*// The TakeOff class inherits from the AIState script*  
public class TakeOff0<T> : AIState0<T> {  
  
    *// Sets up FSM*  
    public TakeOff0(T stateName, StateDrivenBrain0 controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// This float variable sets UpThrust's initial value to 5.0*  
    public float UpThrust = 5.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// UpThrust \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move = UpThrust \* Time.deltaTime;  
        *// This transforms the helicopter to the HoverLocation1 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.HoverLocation1.position, move);  
        *// resets delay to 5.0 for next state*  
        brain.delay = 5.0f;  
    }  
}

## **Hover0**

using UnityEngine;  
using System.Collections;  
  
*// The Hover state is for when the Helicopter needs to maintain standby flying*  
  
*// The Hover class inherits from the AIState script*  
public class Hover0<T> : AIState0<T> {  
  
    *// Sets up FSM*  
    public Hover0(T stateName, StateDrivenBrain0 controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// This line transforms the helicopters rotation to match the RotationPoint GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.RotationPoint.rotation, 0.4f);  
    }  
}

## **Forward0**

using UnityEngine;  
using System.Collections;  
  
*// The forward state is for moving the Helicopter from one point to another across the sky*  
  
*// The Forward class inherits from the AIState script*  
public class Forward0<T> : AIState0<T> {  
  
    *// Sets up FSM*  
    public Forward0(T stateName, StateDrivenBrain0 controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// This float variable sets speed's initial value to 15.0*  
    public float speed = 10.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
        brain.delay = 5.0f;  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move = speed \* Time.deltaTime;  
        *// This transforms the helicopter to the HoverLocation2 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.HoverLocation2.position, move);  
        *// This line transforms the helicopters rotation to match the HoverLocation2 GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.HoverLocation2.rotation, 0.4f);  
    }  
}

## **Land0**

using UnityEngine;  
using System.Collections;  
  
*// The Land state is for downwards movement of the Helicopter*  
  
*// The Land class inherits from the AIState script*  
public class Land0<T> : AIState0<T> {  
  
    *// Sets up FSM*  
    public Land0(T stateName, StateDrivenBrain0 controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public float UpThrust = 5.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// UpThrust \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move = UpThrust \* Time.deltaTime;  
        *// This transforms the helicopter to the LandingPad GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.LandingPad.position, move);  
        *// This rotation reverts the helicopter to its original rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.LandingPad.rotation, 0.4f);  
        *// Boolean that's becomes true at the end of this state*  
        brain.Landed = true;  
    }  
}

# **Task 3 – Weapon System**

## **StateDrivenBrain**

using UnityEngine;  
using System.Collections;  
  
public class StateDrivenBrain : BasicAIController{  
    *// Declares the constant state names*  
    public enum ChopperStates { Idle, TakeOff, Hover, Forward, KillSoldier1, KillSoldier2, DropBomb, Back, Land };  
    *// Sets up state machine variable*  
    public FSM<ChopperStates> chopperStateMachine;  
    *// Puts a 0.4s interval on the state machine*  
    protected float thinkInterval = 0.4f;  
    *// Initialises chopperStateActive to true*  
    public bool chopperStateActive = true;  
    *// Sets up transform variables for these GameObjects*  
    public Transform HoverLocation1;  
    public Transform HoverLocation2;  
    public Transform LandingPad;  
    public Transform RotationPoint;  
    public Transform AttackLocation1;   
    public Transform AttackLocation2;  
    public Transform AttackLocation3;  
    public Transform AttackLocation4;  
    public Transform Rocket1;  
    public Transform Rocket2;  
    public Transform Bomb;  
    public Transform Soldier1;  
    public Transform Soldier2;  
    public Transform BombDestination;  
    public GameObject DestroyRocket1;  
    public GameObject DestroyRocket2;  
    public GameObject DestroyBomb;  
    public GameObject SoldierA;  
    public GameObject SoldierB;  
    public GameObject Fire;  
    public GameObject Fire1;  
    public GameObject Fire2;  
    public GameObject RocketExplosion1;  
    public GameObject RocketExplosion2;  
    public GameObject BombExplosion;  
    public GameObject Smoke;  
    public GameObject Smoke1;  
    public GameObject Smoke2;  
    public GameObject Smoke3;  
    public GameObject RocketBang;  
    public GameObject BombBang;  
    public GameObject BulletRelease;  
    public GameObject BombRelease;  
  
    *// Initialises two boolean variables to false*  
    public bool Landed = false;  
    public bool HoverPosition = false;  
  
    protected void Awake()  
    {  
        *// Sets up a new FSM*  
        chopperStateMachine = new FSM<ChopperStates>();  
        *// This adds in all the new states into the state machine using variables declared*  
        chopperStateMachine.AddState(new Idle<ChopperStates>(ChopperStates.Idle, this, 0f));  
        chopperStateMachine.AddState(new TakeOff<ChopperStates>(ChopperStates.TakeOff, this, 0f));  
        chopperStateMachine.AddState(new Hover<ChopperStates>(ChopperStates.Hover, this, 0f));  
        chopperStateMachine.AddState(new Forward<ChopperStates>(ChopperStates.Forward, this, 0f));  
        chopperStateMachine.AddState(new Land<ChopperStates>(ChopperStates.Land, this, 0f));  
        chopperStateMachine.AddState(new KillSoldier1<ChopperStates>(ChopperStates.KillSoldier1, this, 0f));  
        chopperStateMachine.AddState(new KillSoldier2<ChopperStates>(ChopperStates.KillSoldier2, this, 0f));  
        chopperStateMachine.AddState(new DropBomb<ChopperStates>(ChopperStates.DropBomb, this, 0f));  
        chopperStateMachine.AddState(new Back<ChopperStates>(ChopperStates.Back, this, 0f));  
        *// Sets Idle as first state to be in once plays been activated*  
        chopperStateMachine.SetInitialState (ChopperStates.Idle);  
  
        *// These are all the transitions of the states I will require within this program*  
        chopperStateMachine.AddTransition (ChopperStates.Idle, ChopperStates.TakeOff);  
        chopperStateMachine.AddTransition (ChopperStates.TakeOff, ChopperStates.Hover);  
        chopperStateMachine.AddTransition (ChopperStates.Hover, ChopperStates.Forward);  
        chopperStateMachine.AddTransition (ChopperStates.Forward, ChopperStates.KillSoldier1);  
        chopperStateMachine.AddTransition (ChopperStates.KillSoldier1, ChopperStates.KillSoldier2);  
        chopperStateMachine.AddTransition (ChopperStates.KillSoldier2, ChopperStates.DropBomb);  
        chopperStateMachine.AddTransition (ChopperStates.DropBomb, ChopperStates.Back);  
        chopperStateMachine.AddTransition (ChopperStates.Back, ChopperStates.Hover);  
        chopperStateMachine.AddTransition (ChopperStates.Hover, ChopperStates.Land);  
        chopperStateMachine.AddTransition (ChopperStates.Land, ChopperStates.Idle);  
    }  
  
    *// These Guards determine the conditions states need to meet to be initialised*  
    public bool GuardIdleToTakeOff(State<ChopperStates> currentState)  
    {  
        *// Instantiates once delay ends and if landed is false*  
        return (delay <= 0.0f && Landed == false);  
    }  
    public bool GuardTakeOffToHover(State<ChopperStates> currentState)  
    {  
        *// Changes state when helicopter reaches GameObjects Y position*  
        return (transform.position.y >= HoverLocation1.position.y);  
    }  
    public bool GuardHoverToForward(State<ChopperStates> currentState)  
    {  
        *// Instantiates once delay ends and when helicopters Z position is not equal to the GameObject*  
        return (delay <= 0.0f && transform.position.z != HoverLocation2.position.z);  
    }  
    public bool GuardForwardToKillSoldier1(State<ChopperStates> currentState)  
    {   
        return (delay <= 0.0f && transform.position.Equals(AttackLocation1.position));  
    }  
    public bool GuardKillSoldier1ToKillSoldier2(State<ChopperStates> currentState)  
    {   
        return (delay <= 0.0f && transform.position.Equals(AttackLocation2.position));  
    }  
    public bool GuardKillSoldier2ToDropBomb(State<ChopperStates> currentState)  
    {   
        return (delay <= 0.0f && transform.position.Equals(AttackLocation3.position));  
    }  
    public bool GuardDropBombToBack(State<ChopperStates> currentState)  
    {   
        return (delay <= 0.0f && transform.position.Equals(AttackLocation4.position));  
    }  
    public bool GuardBackToHover(State<ChopperStates> currentState)  
    {  
        *// Instantiates when the boolean is true*  
        return (HoverPosition == true && transform.position.Equals(HoverLocation2.position));  
    }  
    public bool GuardHoverToLand(State<ChopperStates> currentState)  
    {  
        *// Instantiates once delay ends*  
        return (delay <= 0.0f);  
    }  
    public bool GuardLandToIdle(State<ChopperStates> currentState)  
    {  
        *// Changes state when the helicopters Y coordinate is the same as the GameObjects*  
        return (transform.position.y <= LandingPad.position.y);  
    }  
  
    public void Start()  
    {  
        *// Calls to Think function*  
        StartCoroutine(Think());  
    }  
  
    public void Update()  
    {  
        base.Update ();  
        *// Refers to all act callbacks within other scripts*  
        if (chopperStateActive)   
        {  
            chopperStateMachine.CurrentState.Act ();  
        }  
    }  
  
    protected IEnumerator Think()  
    {  
        yield return new WaitForSeconds(thinkInterval);  
        *// Checks state machine*  
        chopperStateMachine.Check();  
        *// Calls to Think function*  
        StartCoroutine(Think());  
    }  
  
    void OnTriggerEnter(Collider col)  
    {  
        *// Once box collider has been entered set HoverPosition to true*  
        HoverPosition = true;  
    }  
}

## **BasicAIController**

using UnityEngine;  
using System.Collections;  
  
*// This script sets up the delay that the chopper waits for before entering it's next state*  
  
public class BasicAIController : MonoBehaviour {  
  
    *// This float variable sets delay's initial value to 2.5*  
    public float delay = 2.5f;  
  
    void Start ()   
    {  
      
    }  
  
    *// Time.deltaTime needs to be updated constantly and also as it's inherited in other scripts*  
    public void Update ()   
    {  
        *// Subtracting Time.deltaTime from delay starts a countdown in seconds*  
        delay -= Time.deltaTime;  
    }  
}

## **AIState**

using UnityEngine;  
using System.Collections;  
  
*// The AIState script inherits from the State Machine*  
public class AIState<T> : State<T> {  
    *// Sets a variable to enable referring to StateDrivenBrain script as brain*  
    protected StateDrivenBrain brain;  
  
    *// Sets up FSM*  
    public AIState(T stateName, StateDrivenBrain brain, float minDuration): base(stateName, brain, minDuration)   
    {  
        this.brain = brain;  
    }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void OnStateTriggerEnter(Collider collider)   
    {  
          
    }  
  
    public override void Monitor()   
    {  
  
    }  
  
    public override void Act()   
    {  
          
    }  
}

## **Idle**

using UnityEngine;  
using System.Collections;  
  
*// The idle state is for when the Helicopter has no functions to carry out*  
  
*// The Idle class inherits from the AIState script*  
public class Idle<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public Idle(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
          
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// No instructions are needed as the helicopter is idle*  
    }  
}

## **TakeOff**

using UnityEngine;  
using System.Collections;  
  
*// The TakeOff state is for upwards movement of the Helicopter*  
  
*// The TakeOff class inherits from the AIState script*  
public class TakeOff<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public TakeOff(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// This float variable sets UpThrust's initial value to 5.0*  
    public float UpThrust = 5.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// UpThrust \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move = UpThrust \* Time.deltaTime;  
        *// This transforms the helicopter to the HoverLocation1 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.HoverLocation1.position, move);  
        *// resets delay to 5.0 for next state*  
        brain.delay = 5.0f;  
    }  
}

## **Hover**

using UnityEngine;  
using System.Collections;  
  
*// The Hover state is for when the Helicopter needs to maintain standby flying*  
  
*// The Hover class inherits from the AIState script*  
public class Hover<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public Hover(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// This line transforms the helicopters rotation to match the RotationPoint GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.RotationPoint.rotation, 0.4f);  
    }  
}

## **Forward**

using UnityEngine;  
using System.Collections;  
  
*// The forward state is for moving the Helicopter from one point to another across the sky*  
  
*// The Forward class inherits from the AIState script*  
public class Forward<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public Forward(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// This float variable sets speed's initial value to 15.0*  
    public float speed = 20.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
        *// When the chopper leaves this state it will reset the delay*  
        brain.delay = 6.0f;  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move = speed \* Time.deltaTime;  
        *// This transforms the helicopter to the AttackLocation1 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.AttackLocation1.position, move);  
        *// This line transforms the helicopters rotation to match the AttackLocation1 GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.AttackLocation1.rotation, 0.4f);  
    }  
}

## **KillSoldier1**

using UnityEngine;  
using System.Collections;  
  
public class KillSoldier1<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public KillSoldier1(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// These float variables set speed with an initial value*  
    public float speed = 15.0f;  
    public float RocketSpeed = 35.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
        *// Locates these two game objects at the start of this state*  
        brain.Rocket1 = GameObject.Find ("RedCardinalRocket").transform;  
        brain.Soldier1 = GameObject.Find ("Army 01 (1)").transform;  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
        *// When the chopper leaves this state it will reset the delay*  
        brain.delay = 8.0f;  
        *// When the chopper leaves this state it will deactivate these game objects*  
        brain.RocketBang.SetActive (false);  
        brain.BulletRelease.SetActive (false);  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move = speed \* Time.deltaTime;  
        *// This transforms the helicopter to the AttackLocation2 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.AttackLocation2.position, move);  
        *// This line transforms the helicopters rotation to match the AttackLocation2 GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.AttackLocation2.rotation, 0.4f);  
  
  
        *// If statement that instantiates when delay is equal to or less than 2.0*  
        if (brain.delay <= 2.0f)   
        {  
            *// The move float is the maxDistanceDelta for the transform*  
            *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
            float move1 = RocketSpeed \* Time.deltaTime;  
            *// This transforms the Rocket1 to the Soldier1 GameObject via move which adds the element of time*  
            brain.Rocket1.position = Vector3.MoveTowards (brain.Rocket1.position, brain.Soldier1.position, move1);  
            *// This is a specific game object with just the Bullet Release audio source on it*  
            brain.BulletRelease.SetActive (true);  
        }  
  
        *// This If statement is only accessable through the Bullet and Soldier meeting and setting off a trigger*  
        if (brain.Rocket1.GetComponent<BulletTrigger>().Check)   
        {  
            *// Here a group of game objects active states are changed to accomodate effects in the scene*  
            brain.DestroyRocket1.SetActive (false);  
            brain.SoldierA.SetActive (false);  
            brain.Fire.SetActive (true);  
            brain.RocketExplosion1.SetActive (true);  
            brain.Smoke.SetActive (true);  
            brain.RocketBang.SetActive (true);  
        }  
    }  
}

## **KillSoldier2**

using UnityEngine;  
using System.Collections;  
  
public class KillSoldier2<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public KillSoldier2(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// These float variables set speed with an initial value*  
    public float speed = 15.0f;  
    public float RocketSpeed = 35.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
        *// When the chopper leaves this state it will reset the delay*  
        brain.delay = 10.0f;  
        *// When the chopper leaves this state it will deactivate these game objects*  
        brain.RocketBang.SetActive (false);  
        brain.BulletRelease.SetActive (false);  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move1 = speed \* Time.deltaTime;  
        *// This transforms the helicopter to the AttackLocation3 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.AttackLocation3.position, move1);  
        *// This line transforms the helicopters rotation to match the AttackLocation3 GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.AttackLocation3.rotation, 0.4f);  
  
        *// If statement that instantiates when delay is equal to or less than 2.0*  
        if (brain.delay <= 2.0f)   
        {  
            *// The move float is the maxDistanceDelta for the transform*  
            *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
            float move2 = RocketSpeed \* Time.deltaTime;  
            *// This transforms the Rocket2 to the Soldier2 GameObject via move which adds the element of time*  
            brain.Rocket2.position = Vector3.MoveTowards (brain.Rocket2.position, brain.Soldier2.position, move2);  
            *// This is a specific game object with just the Bullet Release audio source on it*  
            brain.BulletRelease.SetActive (true);  
        }  
  
        *// This If statement is only true if Rocket2 and Soldier2's positions are equal*  
        if (brain.Rocket2.position.Equals(brain.Soldier2.position))  
        {  
            *// Here a group of game objects active states are changed to accomodate effects in the scene*  
            brain.DestroyRocket2.SetActive (false);  
            brain.SoldierB.SetActive (false);  
            brain.Fire1.SetActive (true);  
            brain.RocketExplosion2.SetActive (true);  
            brain.Smoke1.SetActive (true);  
            brain.RocketBang.SetActive (true);  
        }  
    }  
}

## **DropBomb**

using UnityEngine;  
using System.Collections;  
  
public class DropBomb<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public DropBomb(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// These float variables set speed with an initial value*  
    public float speed = 15.0f;  
    public float RocketSpeed = 30.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
        *// When the chopper leaves this state it will deactivate these game objects*  
        brain.BombBang.SetActive (false);  
        brain.BombRelease.SetActive (false);  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move2 = speed \* Time.deltaTime;  
        *// This transforms the helicopter to the AttackLocation4 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.AttackLocation4.position, move2);  
        *// This line transforms the helicopters rotation to match the AttackLocation4 GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.AttackLocation4.rotation, 0.6f);  
  
        *// If statement that instantiates when delay is equal to or less than 4.0*  
        if (brain.delay <= 4.0f)   
        {  
            *// The move float is the maxDistanceDelta for the transform*  
            *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
            float move3 = RocketSpeed \* Time.deltaTime;  
            *// This transforms the Bomb to the BombDestination GameObject via move which adds the element of time*  
            brain.Bomb.position = Vector3.MoveTowards (brain.Bomb.position, brain.BombDestination.position, move3);  
            *// This is a specific game object with just the Bomb Release audio source on it*  
            brain.BombRelease.SetActive (true);  
        }  
  
        *// This If statement is only true if Bomb and BombDestination positions are equal*  
        if (brain.Bomb.position.Equals(brain.BombDestination.position))  
        {  
            *// Here a group of game objects active states are changed to accomodate effects in the scene*  
            brain.DestroyBomb.SetActive (false);  
            brain.Fire2.SetActive (true);  
            brain.BombExplosion.SetActive (true);  
            brain.Smoke2.SetActive (true);  
            brain.Smoke3.SetActive (true);  
            brain.BombBang.SetActive (true);  
        }  
    }  
}

## **Back**

using UnityEngine;  
using System.Collections;  
  
public class Back<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public Back(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// This float variable sets speed's initial value to 20.0*  
    public float speed = 20.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// speed \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move2 = speed \* Time.deltaTime;  
        *// This transforms the helicopter to the HoverLocation2 GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.HoverLocation2.position, move2);  
        *// This line transforms the helicopters rotation to match the HoverLocation2 GameObject, the float determines the speed of rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.HoverLocation2.rotation, 0.6f);  
    }  
}

## **Land**

using UnityEngine;  
using System.Collections;  
  
*// The Land state is for downwards movement of the Helicopter*  
  
*// The Land class inherits from the AIState script*  
public class Land<T> : AIState<T> {  
  
    *// Sets up FSM*  
    public Land(T stateName, StateDrivenBrain controller, float minDuration) : base(stateName, controller, minDuration) { }  
  
    *// This float variable sets UpThrust's's initial value to 5.0*  
    public float UpThrust = 5.0f;  
  
    public override void OnEnter()   
    {  
        base.OnEnter();  
    }  
  
    public override void OnLeave()   
    {  
        base.OnLeave();  
    }  
  
    public override void Act()  
    {  
        *// The move float is the maxDistanceDelta for the transform*  
        *// UpThrust \* Time.deltaTime is how many frames the helicopter moves per second*  
        float move = UpThrust \* Time.deltaTime;  
        *// This transforms the helicopter to the LandingPad GameObject via move which adds the element of time*  
        brain.transform.position = Vector3.MoveTowards (brain.transform.position, brain.LandingPad.position, move);  
        *// This rotation reverts the helicopter to its original rotation*  
        brain.transform.rotation = Quaternion.RotateTowards (brain.transform.rotation, brain.LandingPad.rotation, 0.4f);  
        *// Boolean that's becomes true at the end of this state*  
        brain.Landed = true;  
    }  
}

## **BulletTrigger**

using UnityEngine;  
using System.Collections;  
  
public class BulletTrigger : MonoBehaviour {  
  
    public bool Check = false;  
  
    public void OnTriggerEnter(Collider col)  
    {  
        if (col.name == ("Army 01 (1)"))  
        {  
            Check = true;  
        }  
    }  
}