State Machines

CI410

Sequencing events

- Typical game flow
- Logo
- Press play
  - Level 1
  - Level 2 …
- Game End
  - Win/Loose
- Back to 1

States

- We can think of our game as being in a number of states
- The way the game behaves will vary by state
  - While Playing, player controls the ship
  - When Game over the player can restart
  - When level is finished, game needs to spawn more Asteroids

State transitions

- For each state
  - Something many need to happen on Entry
  - Something many need to happen During
  - Something many need to happen on Exit
- From each state, the game may go to a different state either by itself or under user control

Enter the State Machine

- Treats game play as a sequence of states
- They may all behave quite differently

```
28 references
public enum GameStates
{   //State the game is in
    None        //Pre start
    , Init
    , Startup
    , PressPlay
    , Play
    , Playing
    , NextLevelText
    , NextLevelPlay
    , GameOver
}

GameStates mCurrentState = GameStates.None;    //Pre First State in initialisation
//State machine handler
9 references
static public GameStates GameState {   //This may call itself recursivly
    private set {
        if (value != sSingleton.mCurrentState)
        {  //Only change state if different from last one, or its first time its used
            sSingleton.ExitState(sSingleton.mCurrentState);  //Exit last state
            GameStates tNextState = sSingleton.EnterState(value); //Enter new state
            if (value == tNextState)
            { //If return state is final state, set it
                sSingleton.mCurrentState = tNextState;
            }
            else
            {
                sSingleton.mCurrentState = value;  //State we are in now
                GameState = tNextState; //If not we need to change state again, until we reach the final one
            }
        }
    }
    get {
        return sSingleton.mCurrentState;   //Get Current State
    }
}
```

EnterState()

- Enters a new state, running specific code for each state
- May ask to enter another state on completion

```
private GameStates EnterState(GameStates vState)
{
    Debug.LogFormat("Enter State {0}", vState);
    switch (vState)
    {
        case GameStates.Init:   //Initialise game, and put up press play
            Debug.Assert(GameOverText != null && PressPlayText != null && NextLevelText != null, "Please set Text fields");
            GameOverText.SetActive(false);  //Turn off all GameState UI Text
            NextLevelText.SetActive(false);
            PressPlayText.SetActive(false);
            GameClear();    //Remove old GameObjects
            return GameStates.PressPlay;    //Also trigger new state on exit

        case GameStates.PressPlay:
            PressPlayText.SetActive(true); //Show Press play
            SpawnAsteroids(StartingAsteroids);
            break;

        case GameStates.Play:
            SpawnPlayer();
            mPlayerDead = false;
            return GameStates.Playing;

        case GameStates.NextLevelText:
            mLevel++;
            mCountDown = 1.0f;
            NextLevelText.SetActive(true); //Show Next Level Message
            break;

        case GameStates.NextLevelPlay:
            SpawnAsteroids(Mathf.Min(mLevel + StartingAsteroids-1, 30));   //More Asteroids each level, but limit at 30
            return GameStates.Playing;

        case GameStates.GameOver:
            GameOverText.SetActive(true);  //Turn Game Over
            break;

        default:    //No Action
            break;
    }
    return vState;  //Default just return state we entered
}
```

Code Walkthrough

```
private GameStates EnterState(GameStates vState)
{
    Debug.LogFormat("Enter State {0}", vState);
    switch (vState)
    {
        case GameStates.Init:   //Initialise game, and put up press play
            Debug.Assert(GameOverText != null && PressPlayText != null && NextLevelText != null, "Please set Text fields");
            GameOverText.SetActive(false);  //Turn off all GameState UI Text
            NextLevelText.SetActive(false);
            PressPlayText.SetActive(false);
            GameClear();    //Remove old GameObjects
            return GameStates.PressPlay;    //Also trigger new state on exit

        case GameStates.PressPlay:
            PressPlayText.SetActive(true); //Show Press play
            SpawnAsteroids(StartingAsteroids);
            break;

        case GameStates.Play:
            SpawnPlayer();
            mPlayerDead = false;
            return GameStates.Playing;

        case GameStates.NextLevelText:
            mLevel++;
            mCountDown = 1.0f;
            NextLevelText.SetActive(true); //Show Next Level Message
            break;

        case GameStates.NextLevelPlay:
            SpawnAsteroids(Mathf.Min(mLevel + StartingAsteroids-1, 30));   //More Asteroids each level, but limit at 30
            return GameStates.Playing;

        case GameStates.GameOver:
            GameOverText.SetActive(true);  //Turn Game Over
            break;

        default:    //No Action
            break;
    }
    return vState;  //Default just return state we entered
}
```

ExitState()

- Tidy up when we exit state

```
//Used to clear up after a state is exited
1 reference
private void ExitState(GameStates vState)
{
    Debug.LogFormat("Exit State {0}", vState);
    switch (vState)
    {
        case GameStates.PressPlay:
            PressPlayText.SetActive(false); //Turn of User Prompt
            break;

        case GameStates.NextLevelText:
            NextLevelText.SetActive(false); //Remove UI
            break;

        default:    //No Action
            break;
    }
}
```

GameStateCoRoutine()

- Runs in background, allowing things to take place during a state

```
//Run this as a CoRoutine every 10th of a second as running it in Update() would be overkill
1 reference
IEnumerator GameStateCoRoutine()
{
    do
    {
        DebugText.text = GameState.ToString();
        switch (GameState)
        {

            case GameStates.PressPlay:
                if (Input.GetKey(KeyCode.Space))
                {
                    GameState = GameStates.Play;    //Go to new state
                }
                break;

            case GameStates.Playing:
                {
                    RockBase[] tAsteroids = FindObjectsOfType<RockBase>();  //Get all the asteroids in the scene
                    if (tAsteroids.Length == 0)
                    {
                        GameState = GameStates.NextLevelText;   //Next Level Text
                    }
                }
                break;

            case GameStates.NextLevelText:
                if(mCountDown>0)
                {
                    mCountDown -= Time.deltaTime;
                    NextLevelText.GetComponent<Text>().text = string.Format("Level {0}\nGet Ready {1:f1}", mLevel + 1,mCountDown);
                }
                else
                {
                    GameState = GameStates.NextLevelPlay;    //Play
                }
                break;
```

CoRoutines

- Will run code alongside other code
  - It's a kind of event queue type of multitasking but its not multithreaded
  - In order for other stuff to run the CoRoutine must yield, if you don't yield Unity will hang

  - Needs a special return type called IEnumerator
  - Has to be run with StartCoroutine(GameStateCoRoutine());

```
yield return new WaitForSeconds(0.1f);  //Wait for a 10th of a second before runnign again, lets other stuff process
```

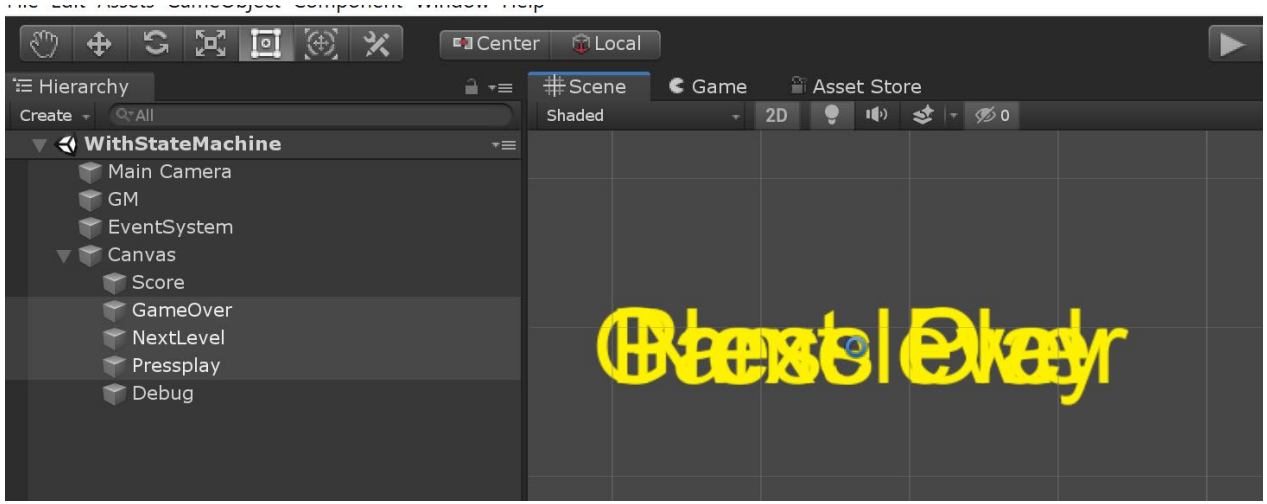
House keeping

1. It would be useful to have all rocks be derived from a common base class

➢Why?

2. We will need player messages to inform

➢How could we do this?



Turn object off to hide

```
[SerializeField]
private GameObject GameOverText;    //Set in IDE

[SerializeField]
private GameObject PressPlayText;    //Set in IDE

[SerializeField]
private GameObject NextLevelText;    //Set in IDE
```

```
GameOverText.SetActive(false);
NextLevelText.SetActive(false);
PressPlayText.SetActive(false);
```

Double Workshop

THIS WEEK & NEXT WEEK

Implement the state machine

- Make new RockBase.cs, empty base class
  - Make all the Rocks & UFO inherit from this
- Make the 3 Game State UI Objects for Press Play, New Level and GameOver
- Add a UI text field to display the current State for debug
- Update your GM.cs to include a state machine, use the Gist to guide you
- If stuck ask for help