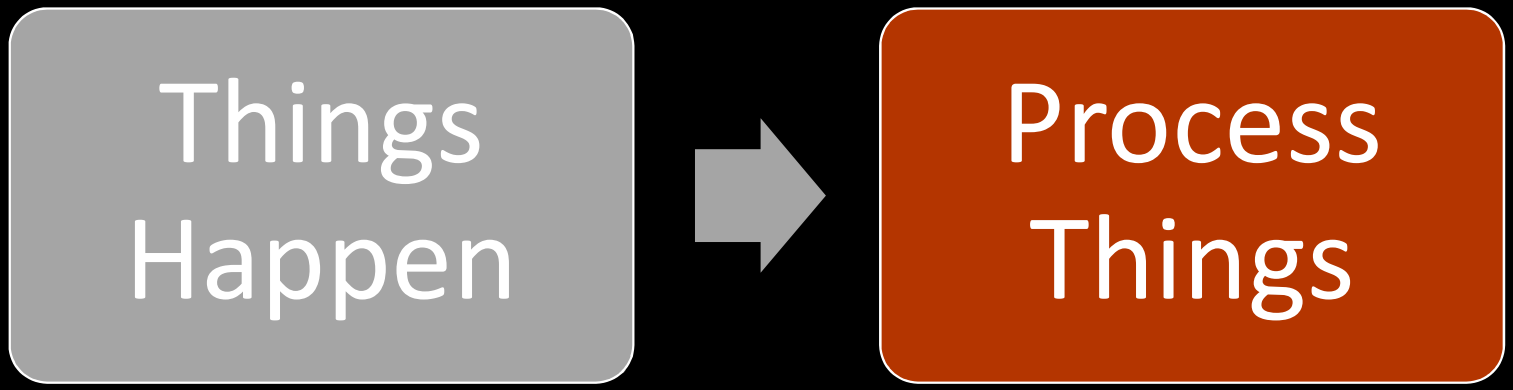
Publishers, Subscribers and Events

Common Game Loop Processing

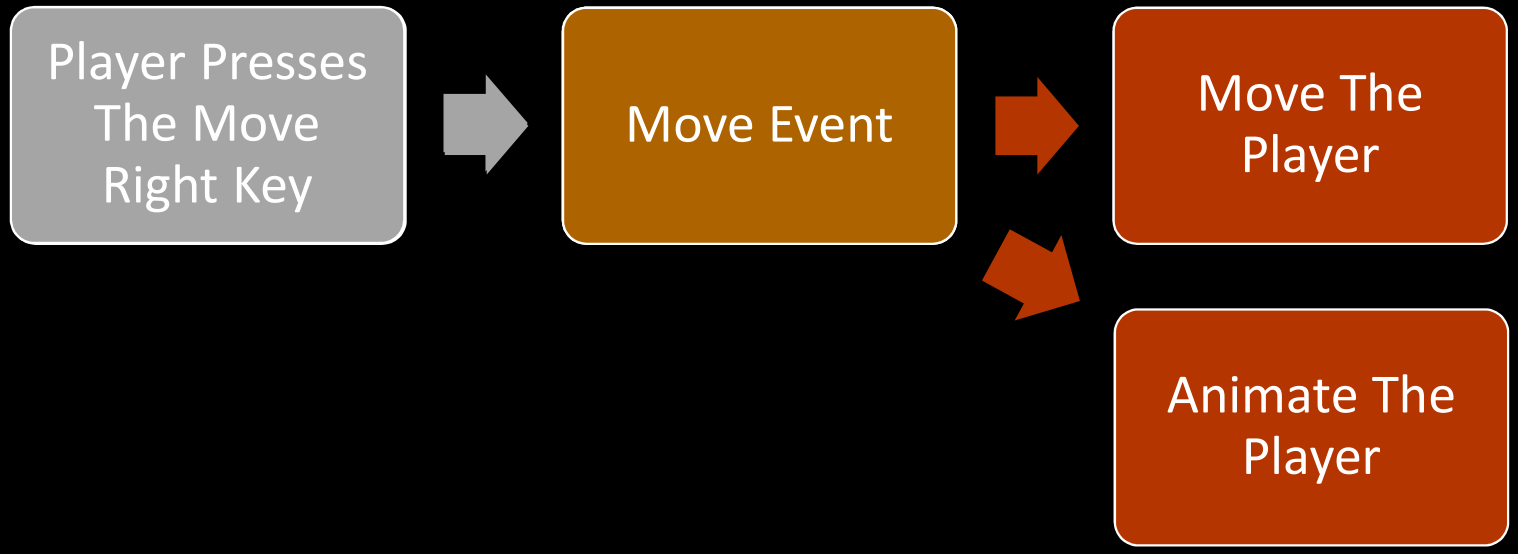


Publishers, Subscribers and Events

Common Game Loop Processing

Publishers, Subscribers and Events

Common Game Loop Processing



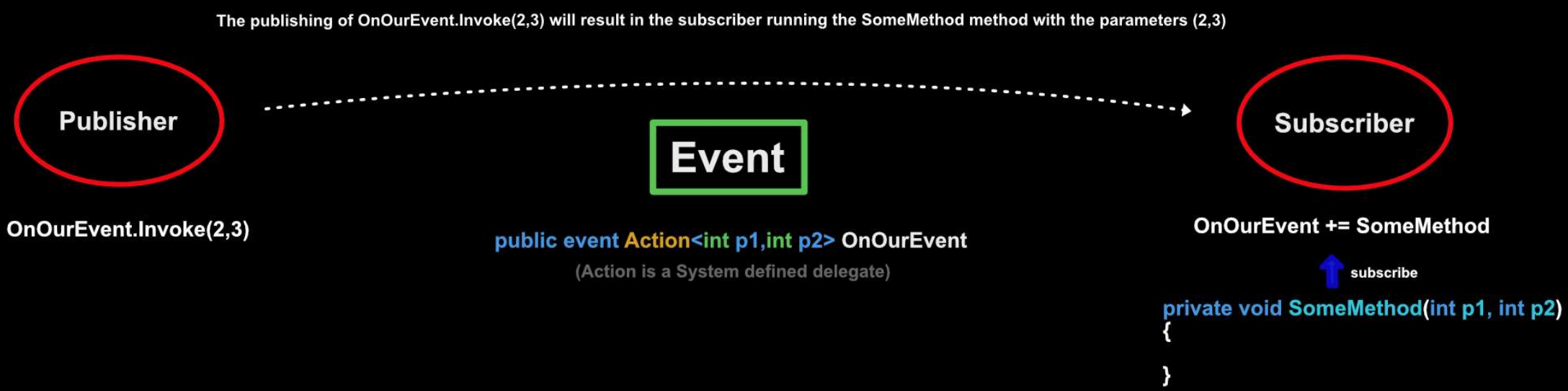
Publisher Subscriber Design Pattern

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern



Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Alternatives

```
public delegate void OurDelegate (int p1 , int p2)
public event OurDelegate OnOurEvent
```

OR

```
public delegate void EventHandler (object sender, EventArgs e)
public event EventHandler OnOurEvent
```

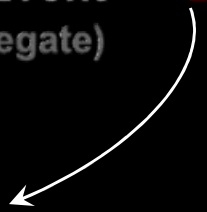
(EventHandler is a System defined delegate)

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Extend EventArgs

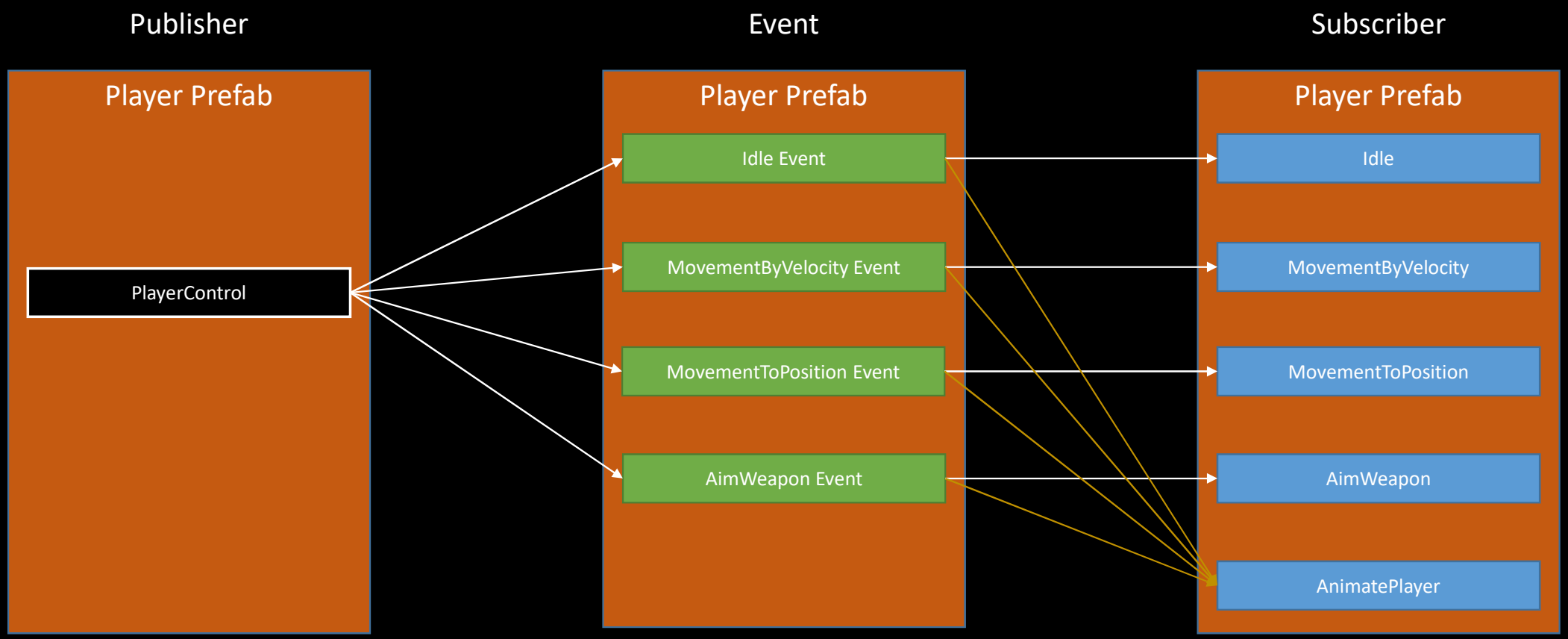
```
public delegate void EventHandler (object sender, EventArgs e)
public event EventHandler OnOurEvent
(EventHandler is a System defined delegate)
```



```
public class MyEventArgs : EventArgs
{
    public int p2
    public int p2
}
```

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern



Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Publisher

Player Prefab

PlayerControl

Part Of The Player Roll

```
player.movementToPositionEvent.CallMovementToPositionEvent
(targetPosition, player.transform.position,
movementDetails.rollSpeed, direction, isPlayerRolling);
```

Event

Player Prefab

```
public class MovementToPositionEvent : MonoBehaviour
{
    public event Action<MovementToPositionEvent,
      MovementToPositionArgs> OnMovementToPosition;

    1 reference
    public void CallMovementToPositionEvent(Vector3
      movePosition, Vector3 currentPosition, float moveSpeed,
      Vector2 moveDirection, bool isRolling = false)
    {
        OnMovementToPosition?.Invoke(this, new
            MovementToPositionArgs() { movePosition =
            movePosition, currentPosition = currentPosition,
            moveSpeed = moveSpeed, moveDirection = moveDirection,
            isRolling = isRolling });
    }
}

5 references
public class MovementToPositionArgs : EventArgs
{
    public Vector3 movePosition;
    public Vector3 currentPosition;
    public float moveSpeed;
    public Vector2 moveDirection;
    public bool isRolling;
}
```

Subscriber

Player Prefab

```
public class MovementToPosition : MonoBehaviour
{
  private void OnEnable()
  {
      // Subscribe to movement to position event
      movementToPositionEvent.OnMovementToPosition +=
        MovementToPositionEvent_OnMovementToPosition;
  }
// On movement event
2 references
private void MovementToPositionEvent_OnMovementToPosition(MovementToPositionEvent
  movementToPositionEvent, MovementToPositionArgs movementToPositionArgs)
{
    MoveRigidBody(movementToPositionArgs.movePosition,
      movementToPositionArgs.currentPosition, movementToPositionArgs.moveSpeed);
}

/// <summary>
/// Move the rigidbody component
/// </summary>
1 reference
private void MoveRigidBody(Vector3 movePosition, Vector3 currentPosition, float
  moveSpeed)
{
    Vector2 unitVector = Vector3.Normalize(movePosition - currentPosition);

    rigidBody2D.MovePosition(rigidBody2D.position + (unitVector * moveSpeed *
      Time.fixedDeltaTime));
}
}
```

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Event

Player Prefab

```
public class MovementToPositionEvent : MonoBehaviour
{
    public event Action<MovementToPositionEvent,
      MovementToPositionArgs> OnMovementToPosition;

    1 reference
    public void CallMovementToPositionEvent(Vector3
      movePosition, Vector3 currentPosition, float moveSpeed,
      Vector2 moveDirection, bool isRolling = false)
    {
        OnMovementToPosition?.Invoke(this, new
          MovementToPositionArgs() { movePosition =
          movePosition, currentPosition = currentPosition,
          moveSpeed = moveSpeed, moveDirection = moveDirection,
          isRolling = isRolling });
    }
}

5 references
public class MovementToPositionArgs : EventArgs
{
    public Vector3 movePosition;
    public Vector3 currentPosition;
    public float moveSpeed;
    public Vector2 moveDirection;
    public bool isRolling;
}
```

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Publisher

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Subscriber

Player Prefab

```
public class MovementToPosition : MonoBehaviour
{
    private void OnEnable()
    {
        // Subscribe to movement to position event
        movementToPositionEvent.OnMovementToPosition +=
            MovementToPositionEvent_OnMovementToPosition;
    }
    // On movement event
    2 references
    private void MovementToPositionEvent_OnMovementToPosition(MovementToPositionEvent
        movementToPositionEvent, MovementToPositionArgs movementToPositionArgs)
    {
        MoveRigidBody(movementToPositionArgs.movePosition,
            movementToPositionArgs.currentPosition, movementToPositionArgs.moveSpeed);
    }

    /// <summary>
    /// Move the rigidbody component
    /// </summary>
    1 reference
    private void MoveRigidBody(Vector3 movePosition, Vector3 currentPosition, float
        moveSpeed)
    {
        Vector2 unitVector = Vector3.Normalize(movePosition - currentPosition);

        rigidBody2D.MovePosition(rigidBody2D.position + (unitVector * moveSpeed *
            Time.fixedDeltaTime));
    }
}
```

Publishers, Subscribers and Events

Publisher Subscriber Design Pattern

Publisher

Player Prefab

PlayerControl

Part Of The Player Roll

```
player.movementToPositionEvent.CallMovementToPositionEvent
(targetPosition, player.transform.position,
movementDetails.rollSpeed, direction, isPlayerRolling);
```

Event

Player Prefab

```
public class MovementToPositionEvent : MonoBehaviour
{
    public event Action<MovementToPositionEvent,
      MovementToPositionArgs> OnMovementToPosition;

    1 reference
    public void CallMovementToPositionEvent(Vector3
      movePosition, Vector3 currentPosition, float moveSpeed,
      Vector2 moveDirection, bool isRolling = false)
    {
        OnMovementToPosition?.Invoke(this, new
            MovementToPositionArgs() { movePosition =
            movePosition, currentPosition = currentPosition,
            moveSpeed = moveSpeed, moveDirection = moveDirection,
            isRolling = isRolling });
    }
}

5 references
public class MovementToPositionArgs : EventArgs
{
    public Vector3 movePosition;
    public Vector3 currentPosition;
    public float moveSpeed;
    public Vector2 moveDirection;
    public bool isRolling;
}
```

Subscriber

Player Prefab

```
public class MovementToPosition : MonoBehaviour
{
  private void OnEnable()
  {
      // Subscribe to movement to position event
      movementToPositionEvent.OnMovementToPosition +=
        MovementToPositionEvent_OnMovementToPosition;
  }
// On movement event
2 references
private void MovementToPositionEvent_OnMovementToPosition(MovementToPositionEvent
  movementToPositionEvent, MovementToPositionArgs movementToPositionArgs)
{
    MoveRigidBody(movementToPositionArgs.movePosition,
      movementToPositionArgs.currentPosition, movementToPositionArgs.moveSpeed);
}

/// <summary>
/// Move the rigidbody component
/// </summary>
1 reference
private void MoveRigidBody(Vector3 movePosition, Vector3 currentPosition, float
  moveSpeed)
{
    Vector2 unitVector = Vector3.Normalize(movePosition - currentPosition);

    rigidBody2D.MovePosition(rigidBody2D.position + (unitVector * moveSpeed *
      Time.fixedDeltaTime));
}
}
```