Game Looping

Ken-Yi Lee

Game Programming, Fall 2020 @ National Taiwan University

FeisStu

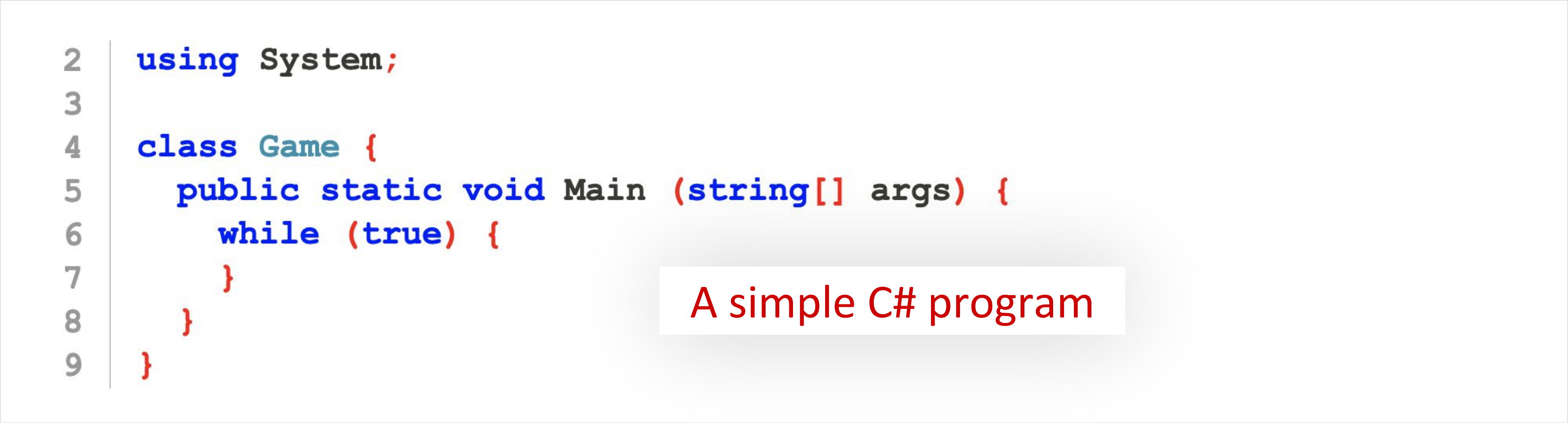
Game Programming

- Rendering
- Looping and control
- Math
- Behaviour and navigation (AI)
- Physics
- Animation and effects
- Networking

Game Programming

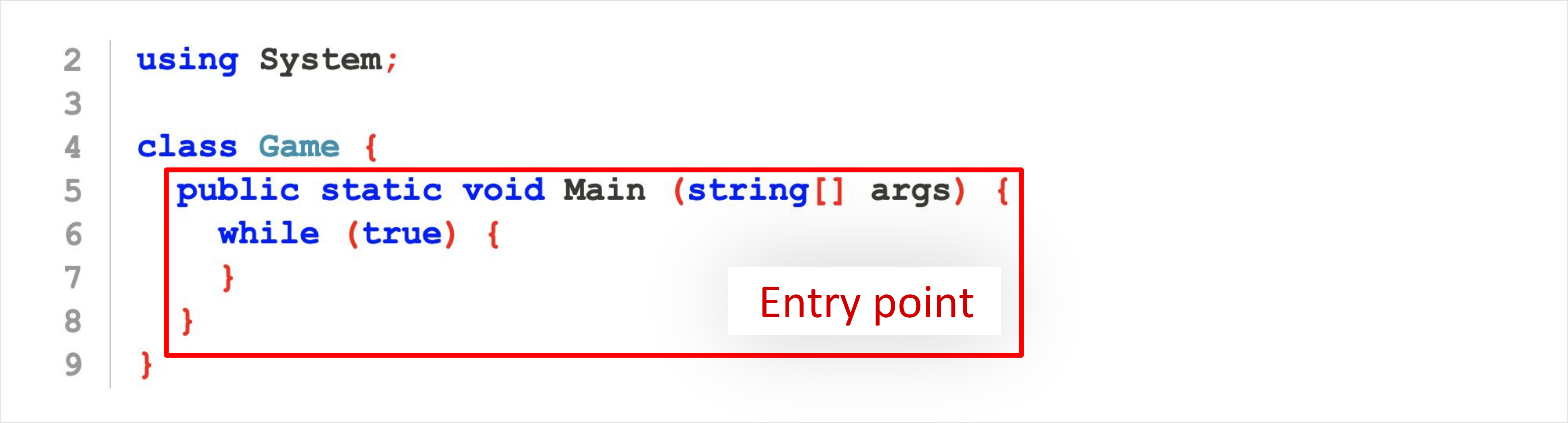
- Rendering
- Looping and control
- Math
- Behaviour and navigation (AI)
- Physics
- Animation and effects
- Networking

FeisStu



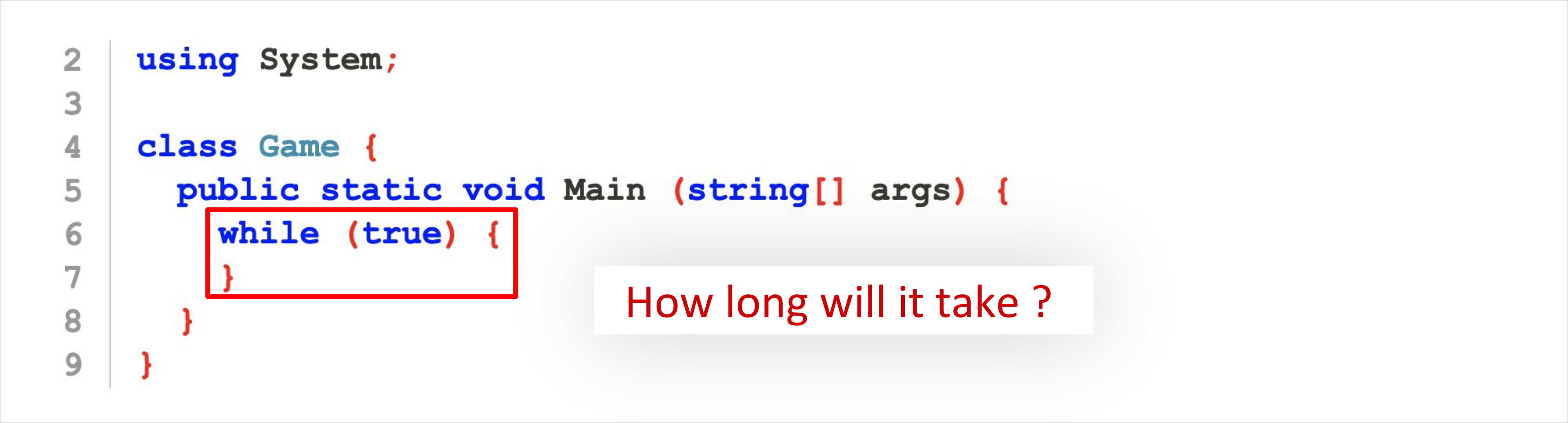
```
using System;

class Game {
  public static void Main (string[] args) {
    while (true) {
    }
  }
}
```

A simple C# program



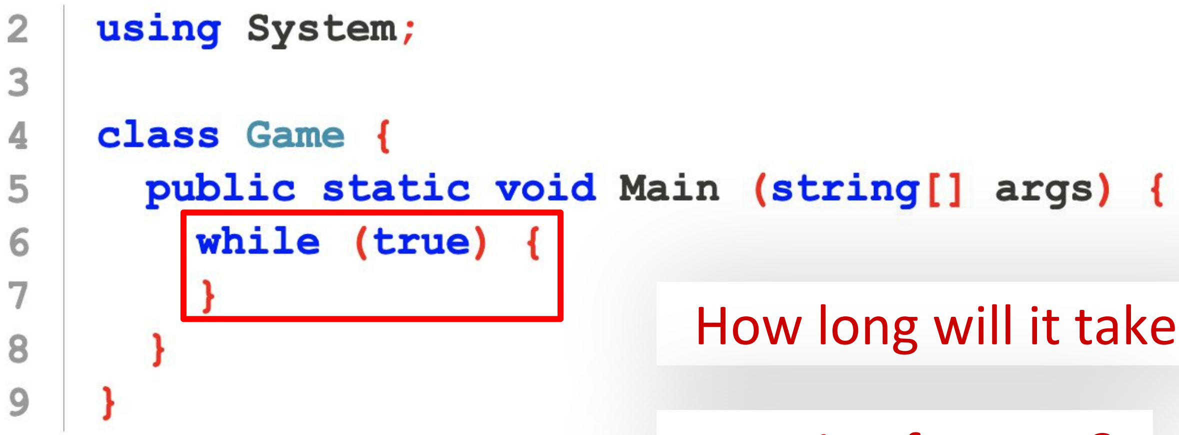
```
using System;

class Game {
  public static void Main (string[] args) {
    while (true) {
    }
  }
}
```

Entry point



```
using System;

class Game {
  public static void Main (string[] args) {
    while (true) {
    }
  }
}
```

How long will it take ?

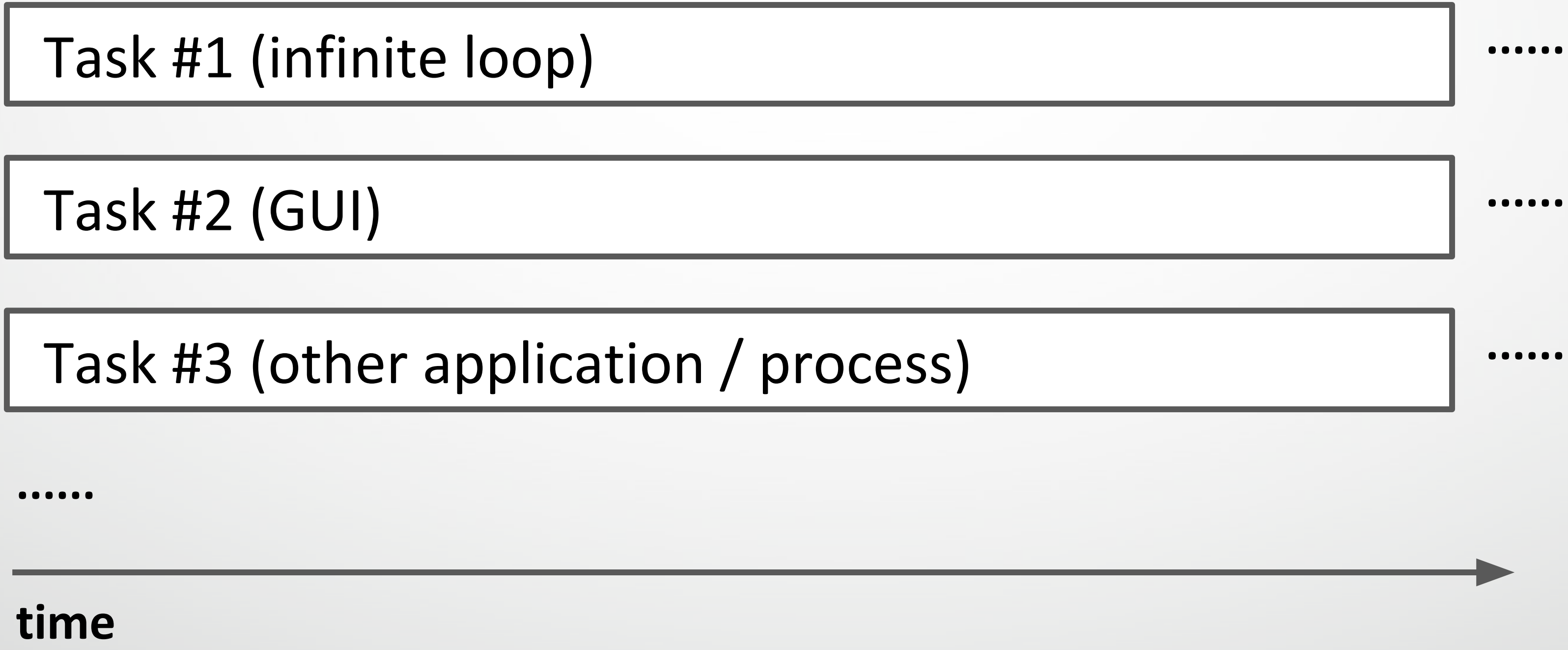


```
using System;

class Game {
  public static void Main (string[] args) {
    while (true) {
    }
  }
}
```

How long will it take ?

Hanging forever ?

Multitasking



Task #1 (infinite loop) ......

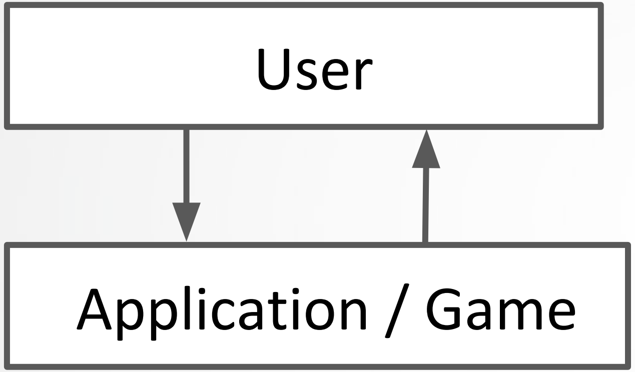
Task #2 (GUI) ......

Task #3 (other application / process) ......

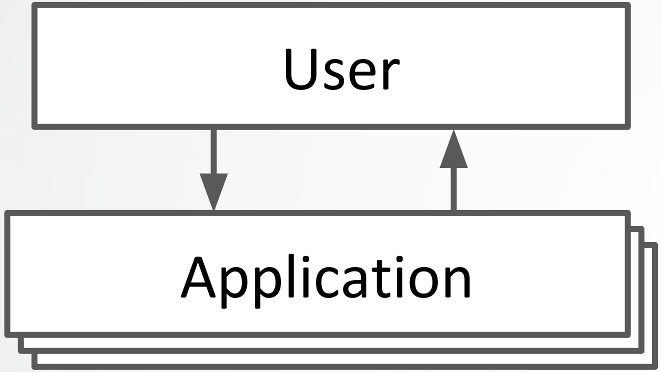
......

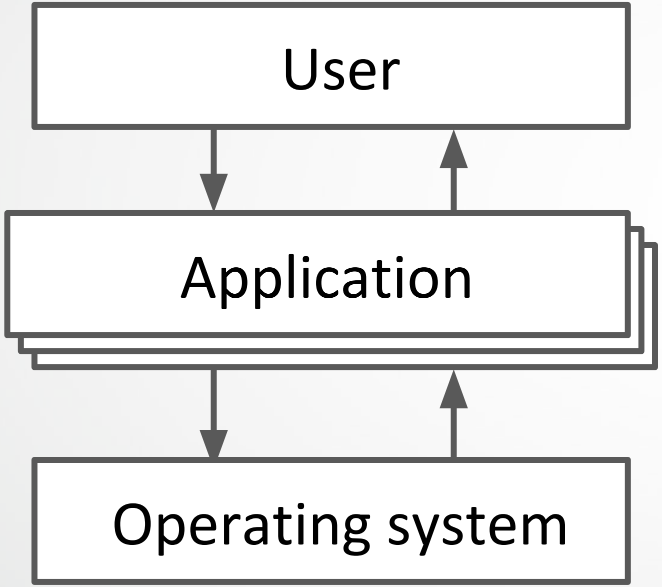
time

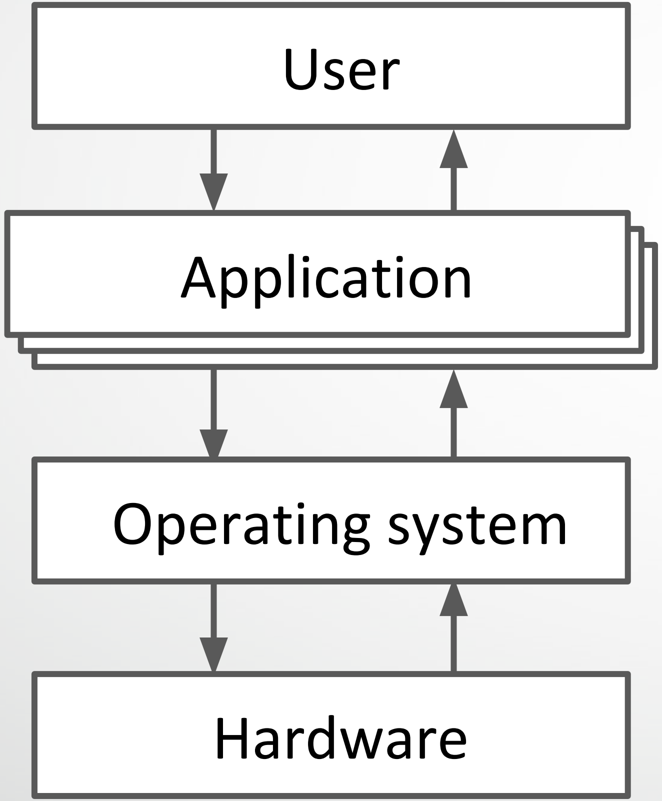
User

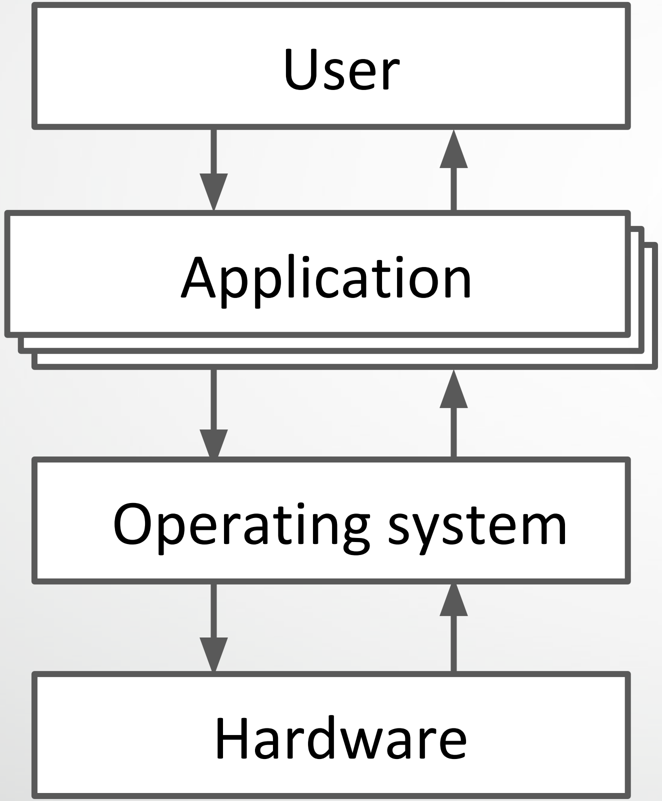

User

Application / Game

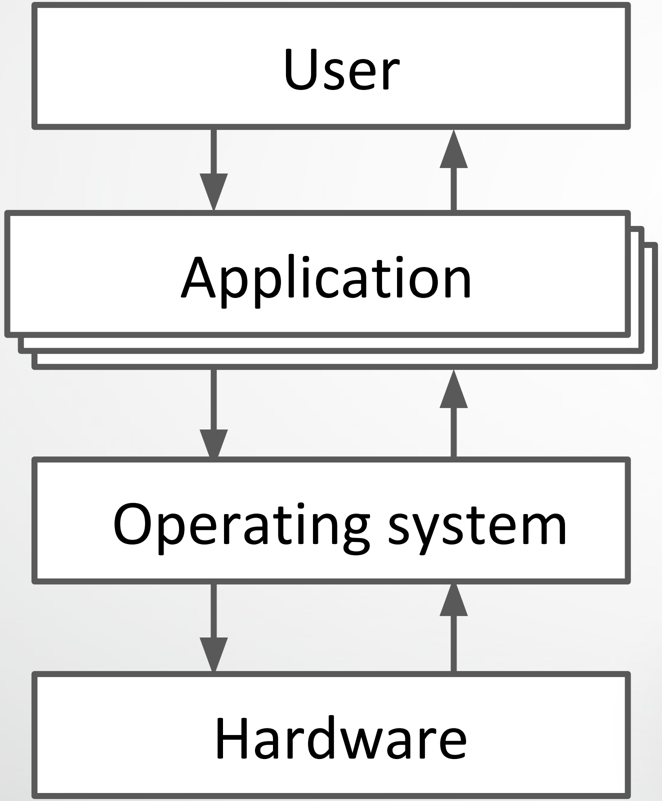

User
Application


User
Application
Operating system


User
Application
Operating system
Hardware



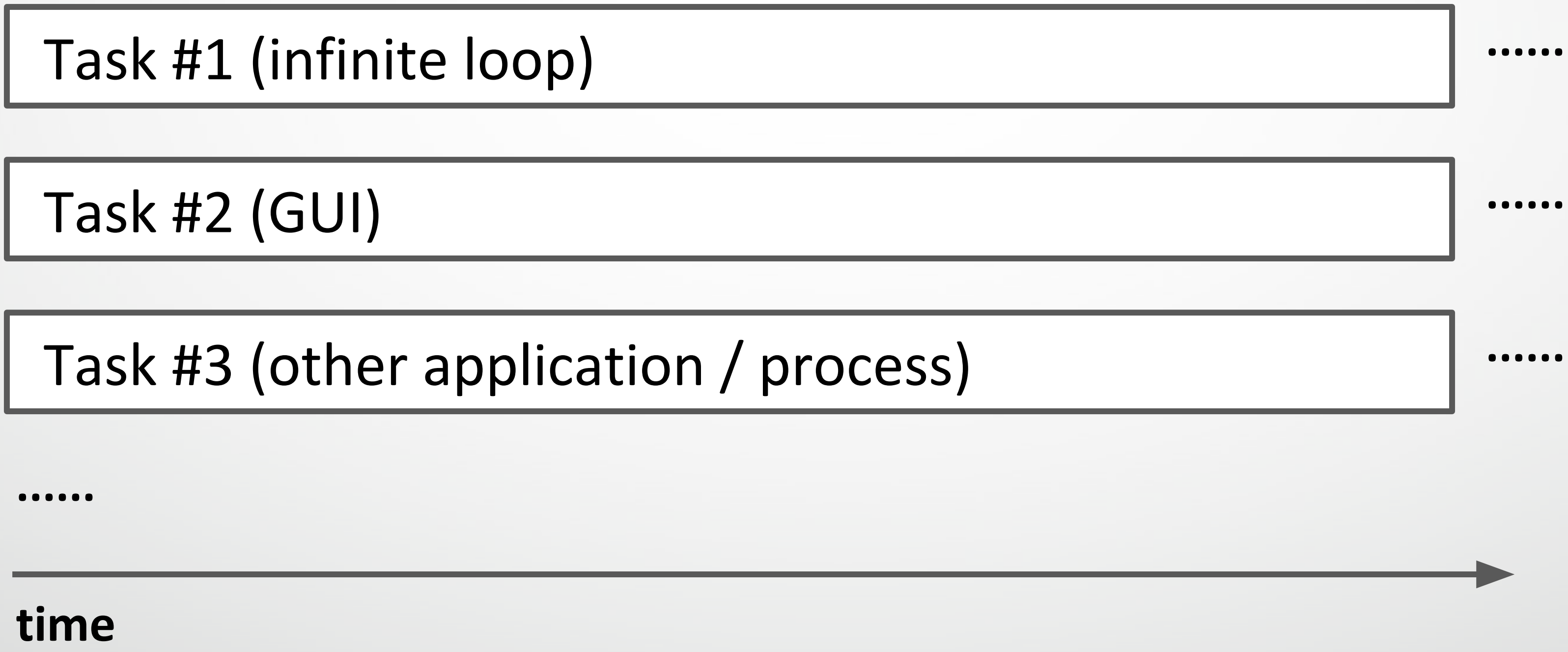
Multiple applications / processes / tasks



Multiple applications / processes / tasks

with "single" CPU / Core ?

Multitasking



Task #1 (infinite loop) ……

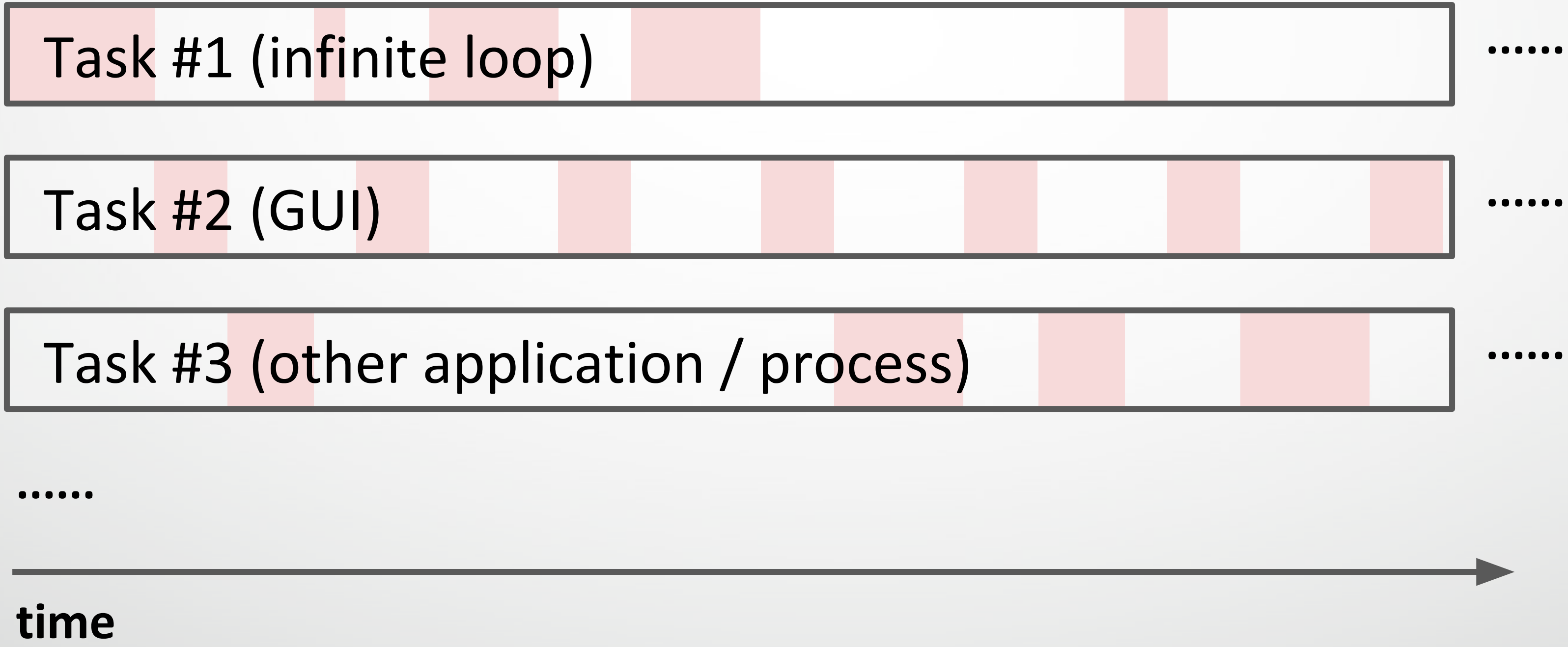
Task #2 (GUI) ……

Task #3 (other application / process) ……

……

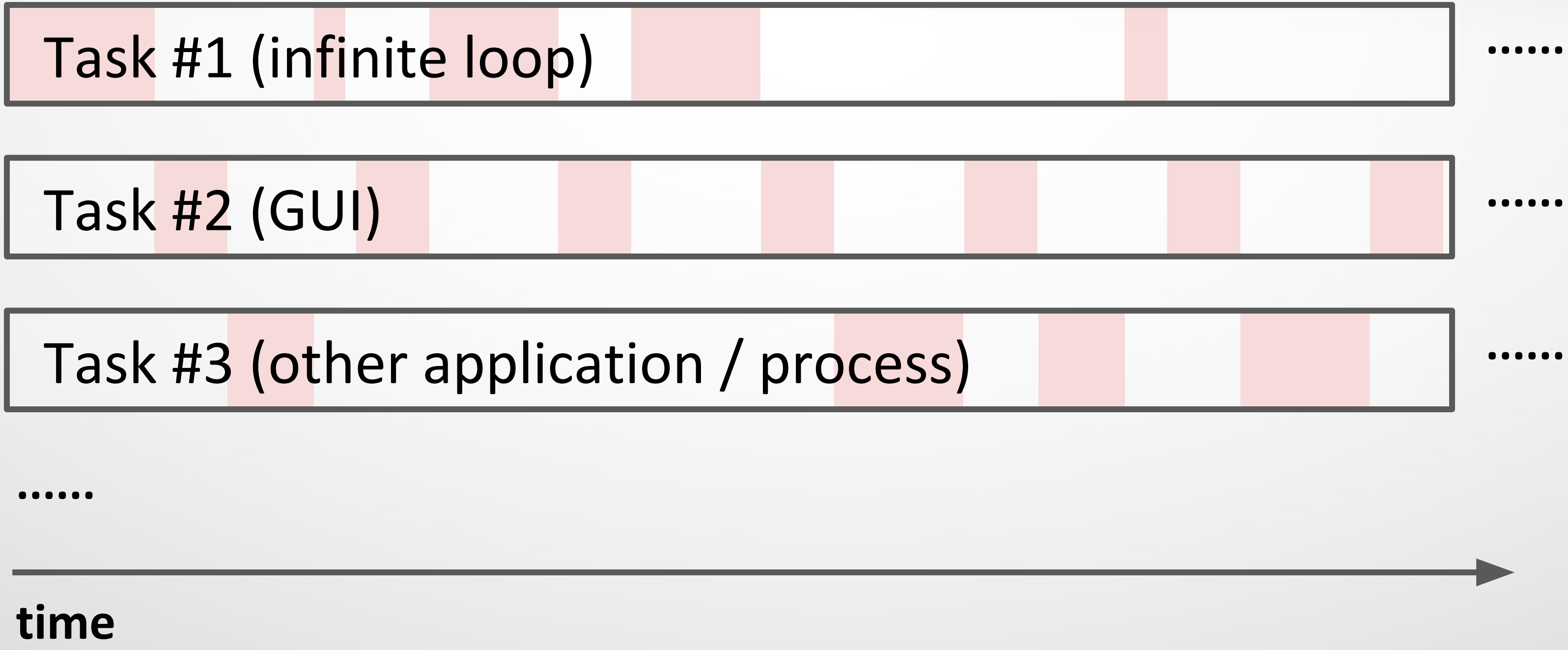
time

Multitasking



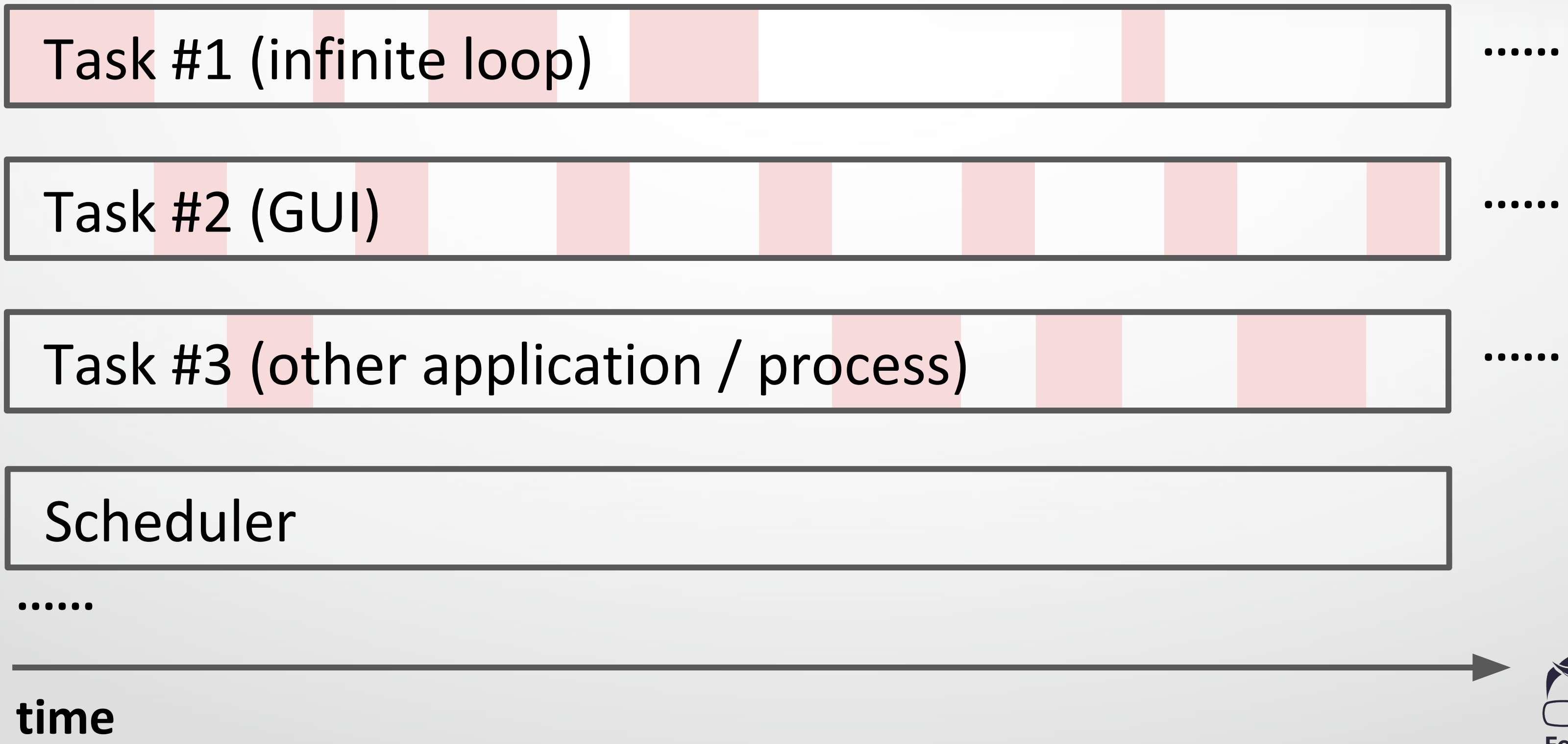
Multitasking

How to switch between tasks ?



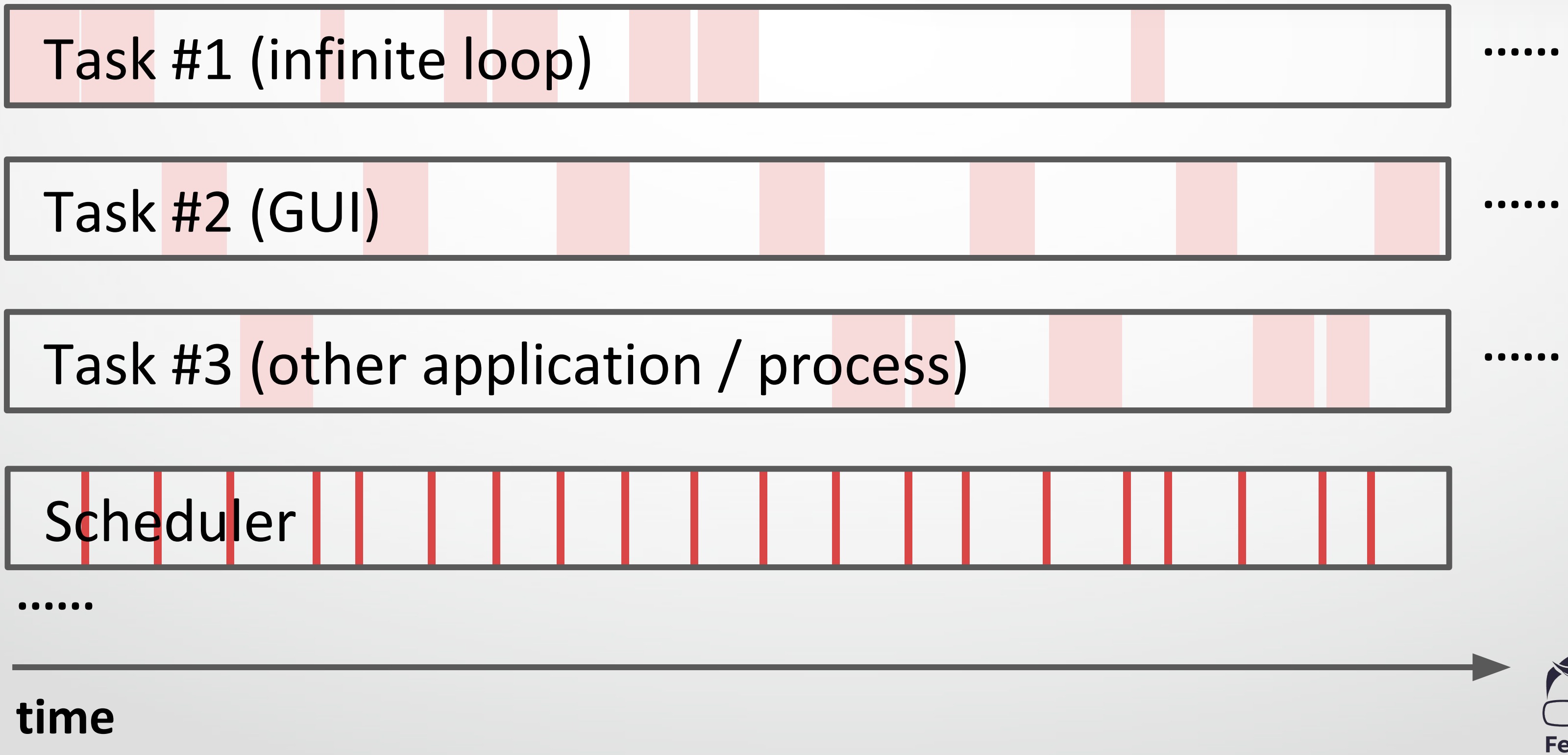
Multitasking

How to switch between tasks ?

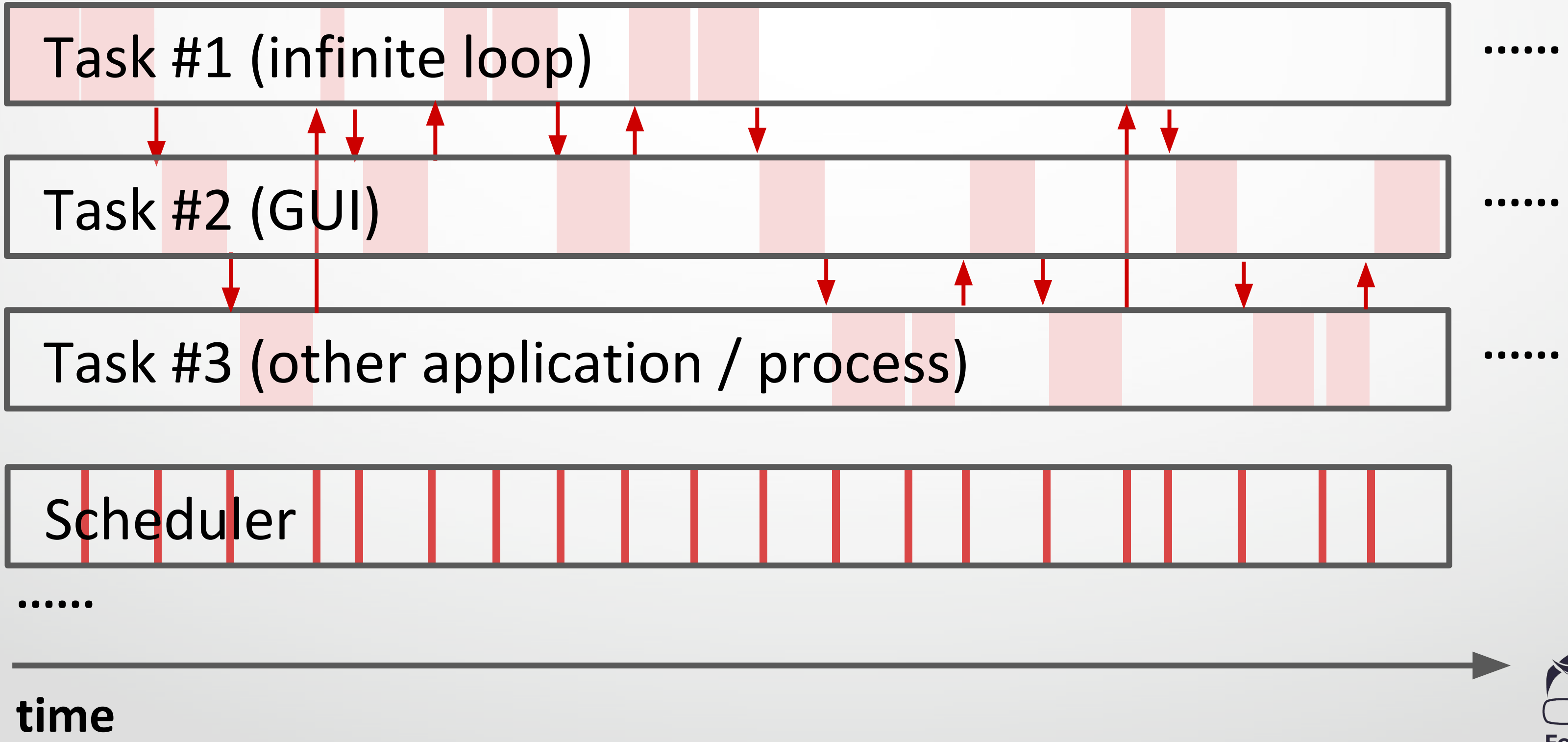


Multitasking

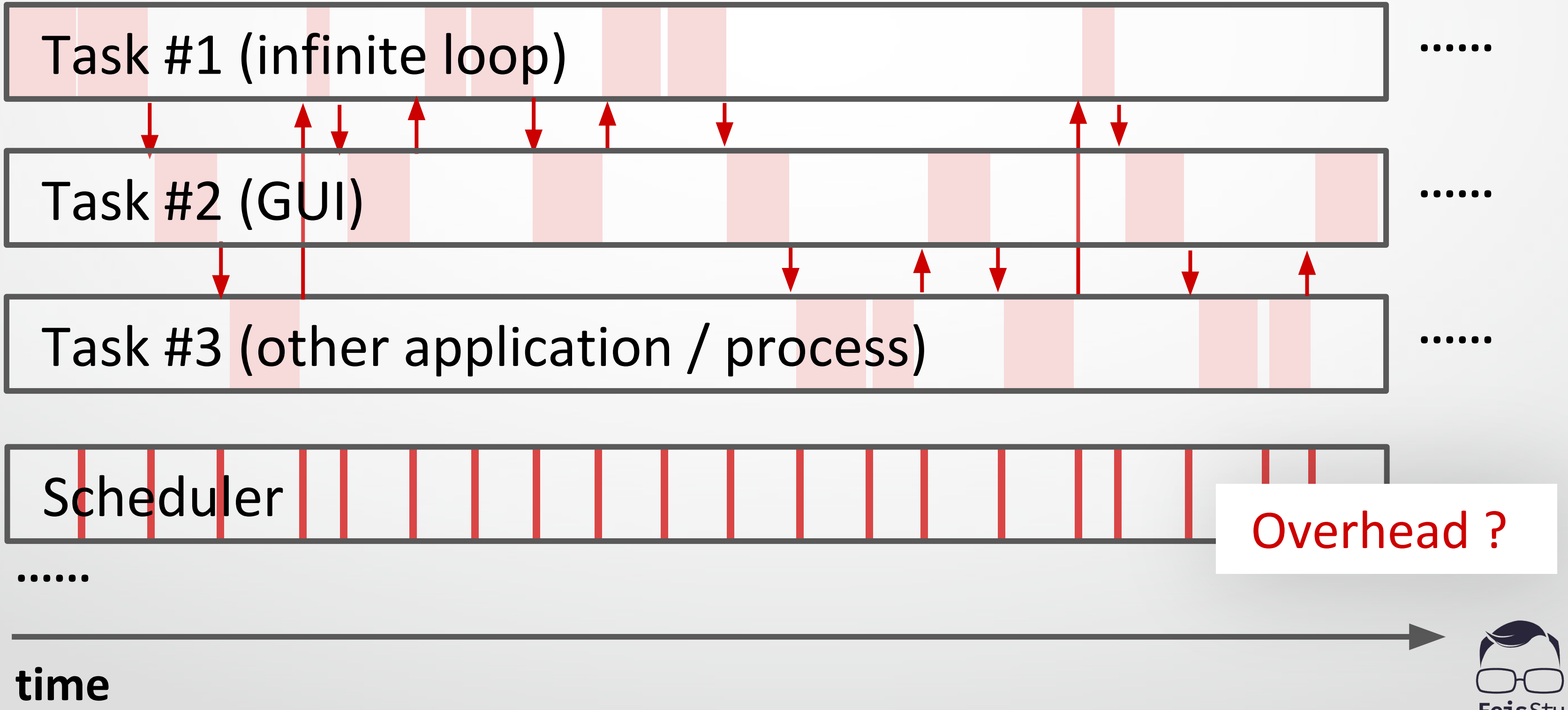
How to switch between tasks ?



Context switch



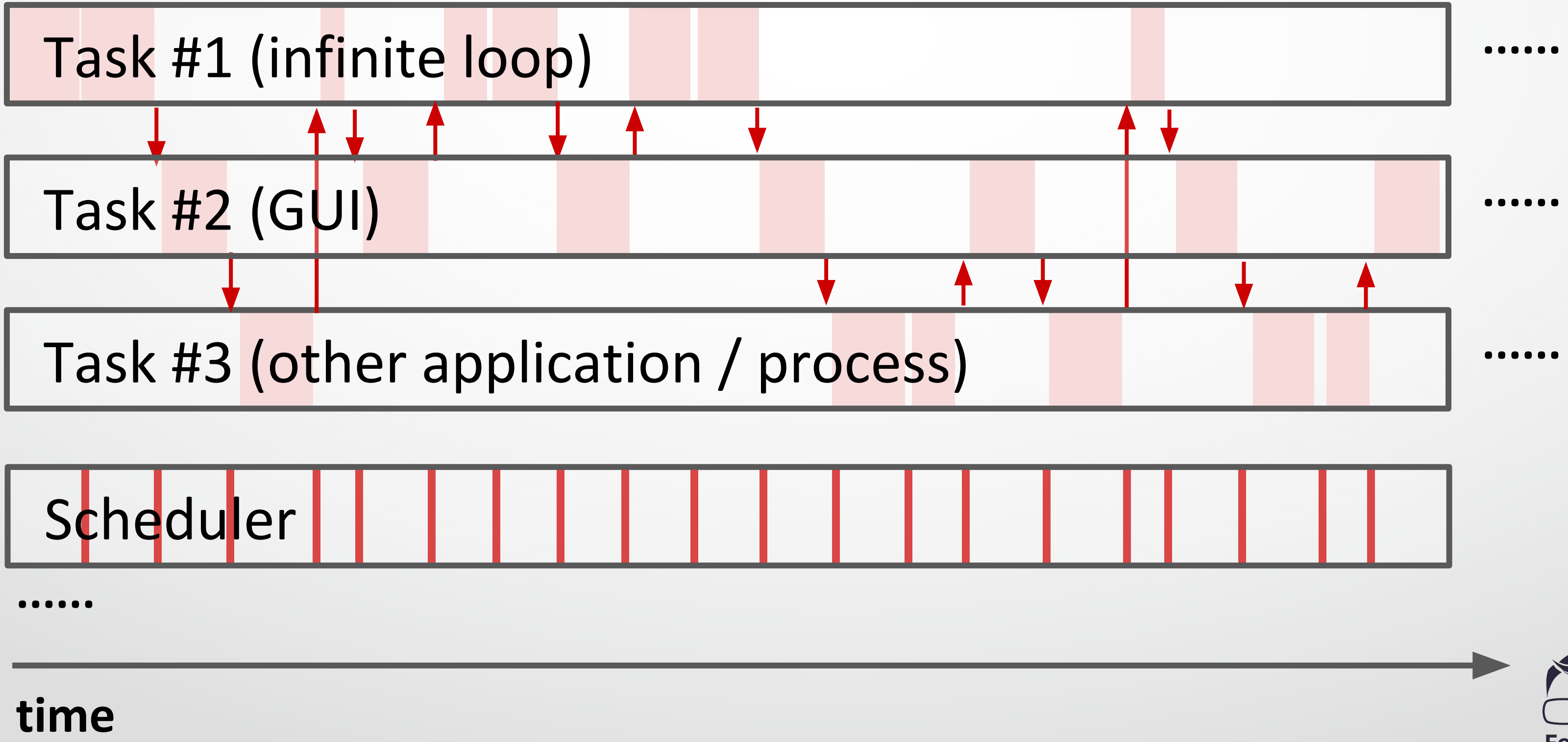
Context switch



Overhead ?

Context switch

Reasons to switch ?

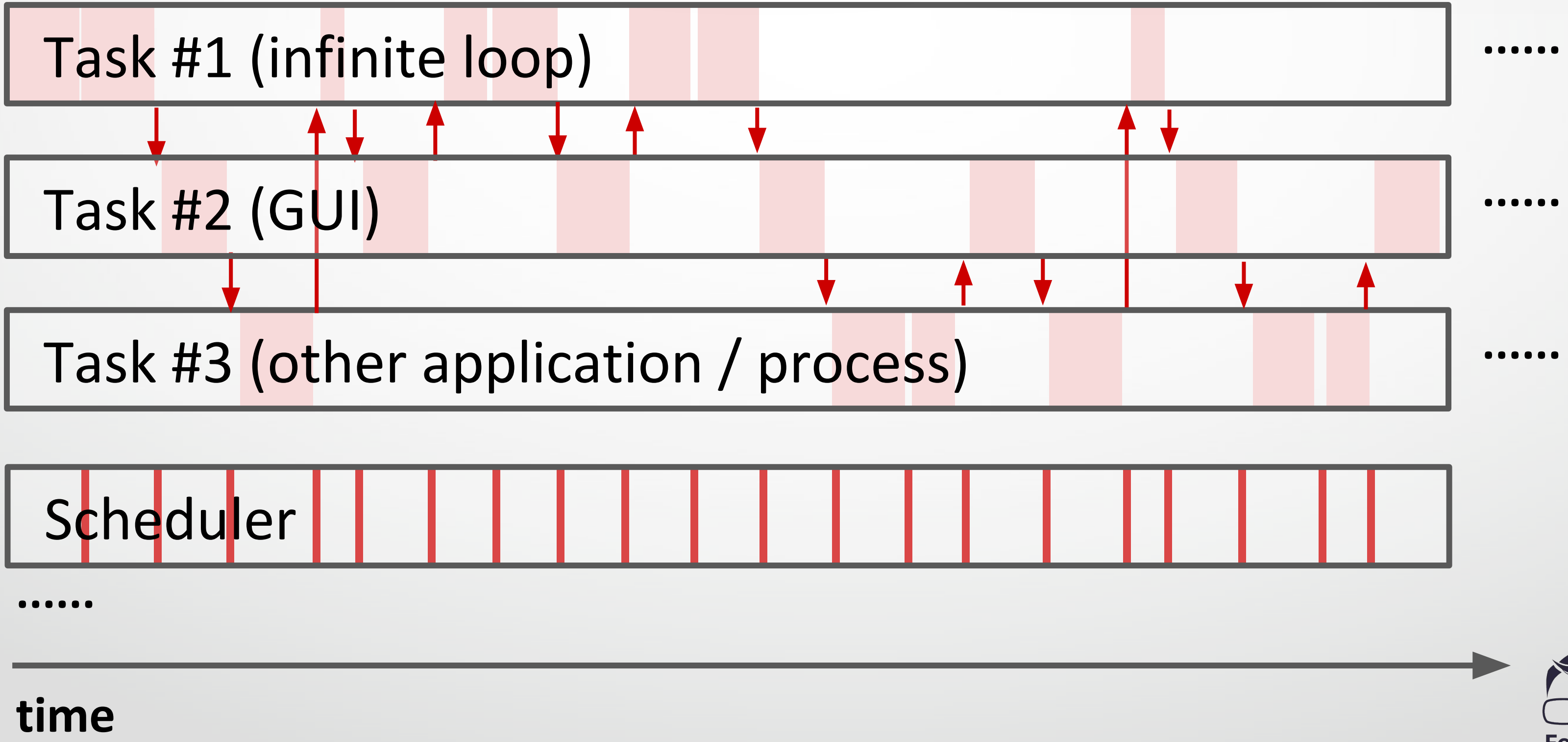


Context switch

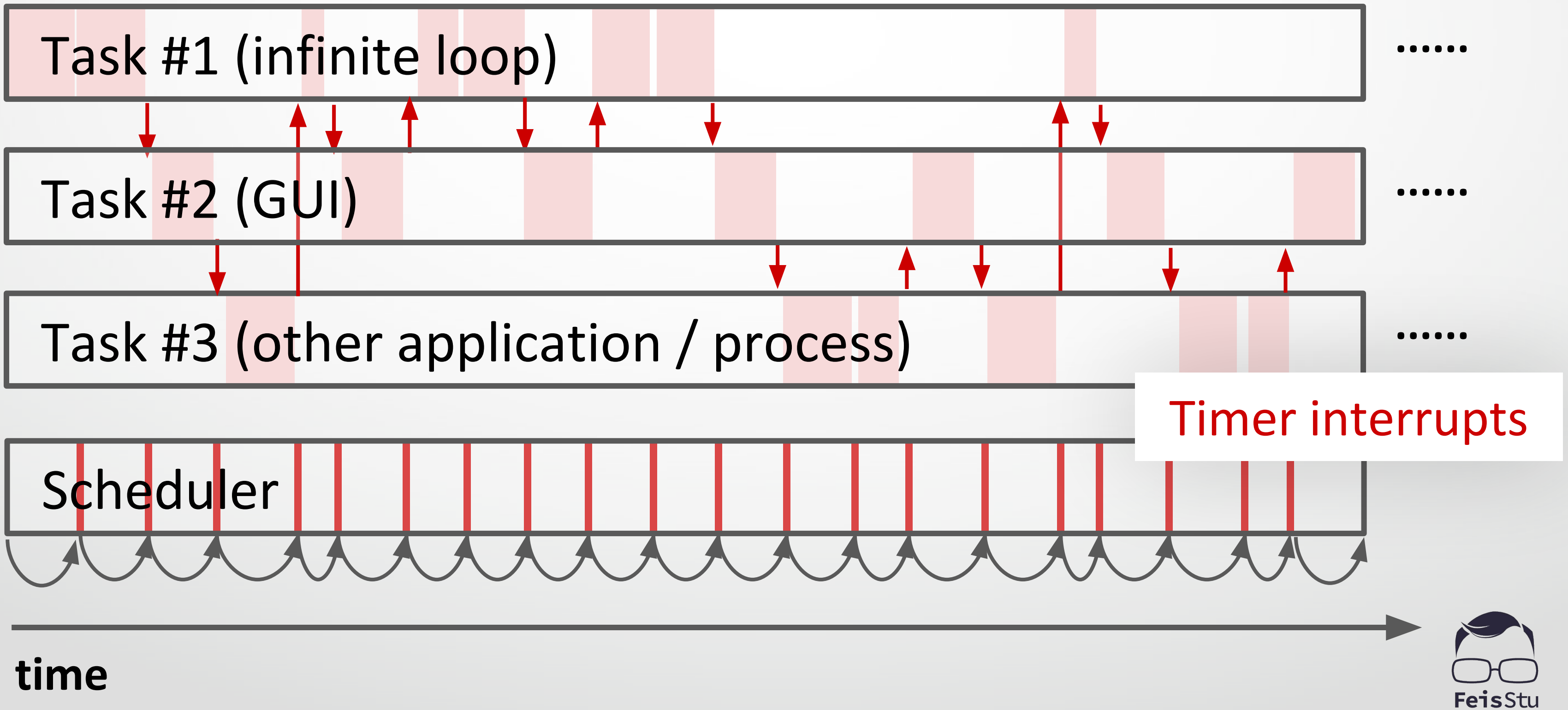
Finished ?

Waiting I/O ?

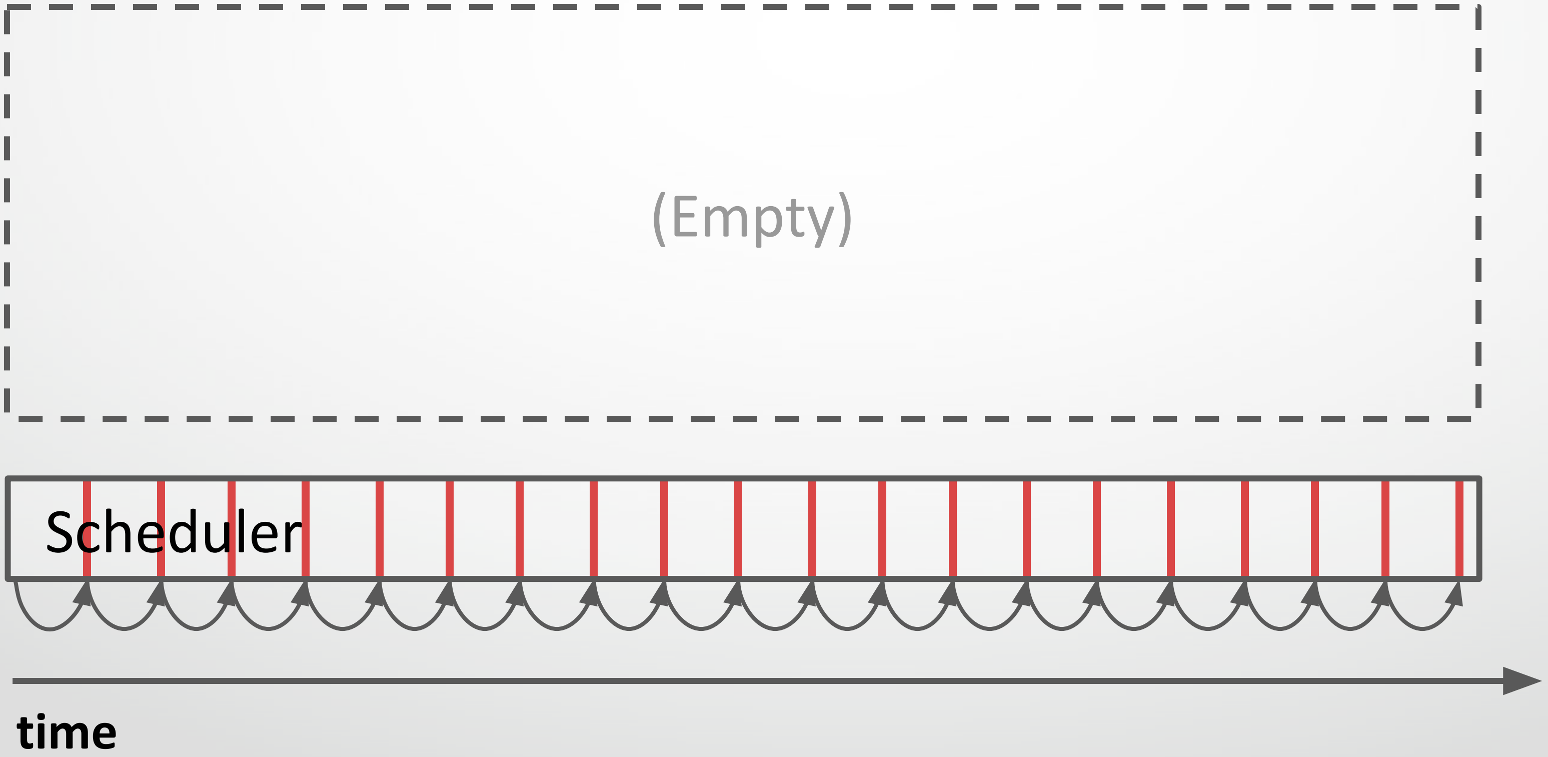
Preempted



Hardware interrupt



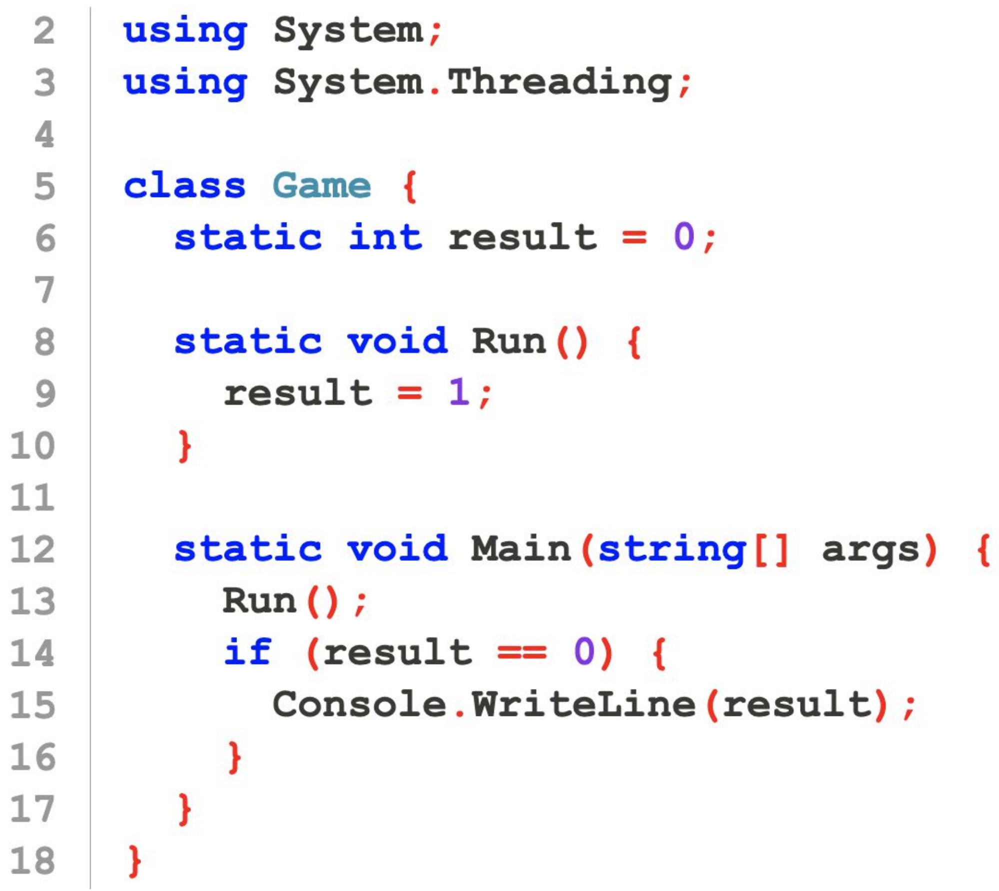
"Finally I can sleep."



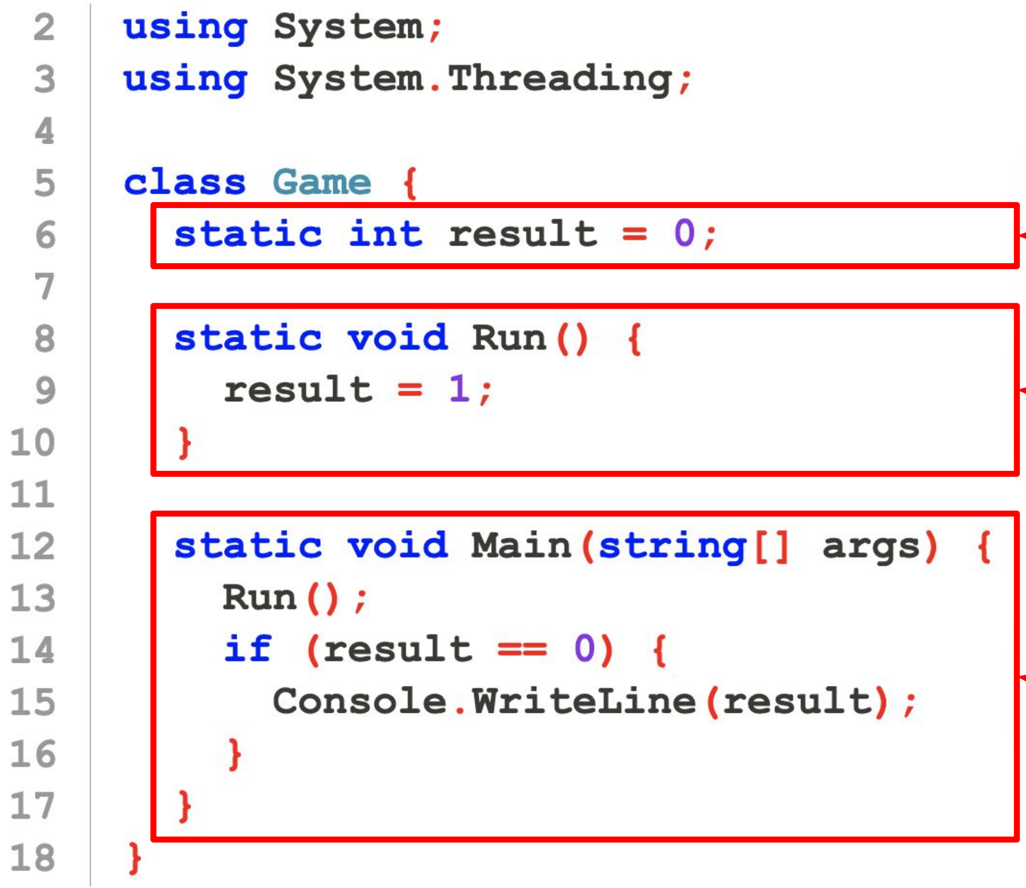
(Empty)

Scheduler

time



```csharp
using System;
using System.Threading;

class Game {
  static int result = 0;

  static void Run() {
    result = 1;
  }

  static void Main(string[] args) {
    Run();
    if (result == 0) {
      Console.WriteLine(result);
    }
  }
}
```

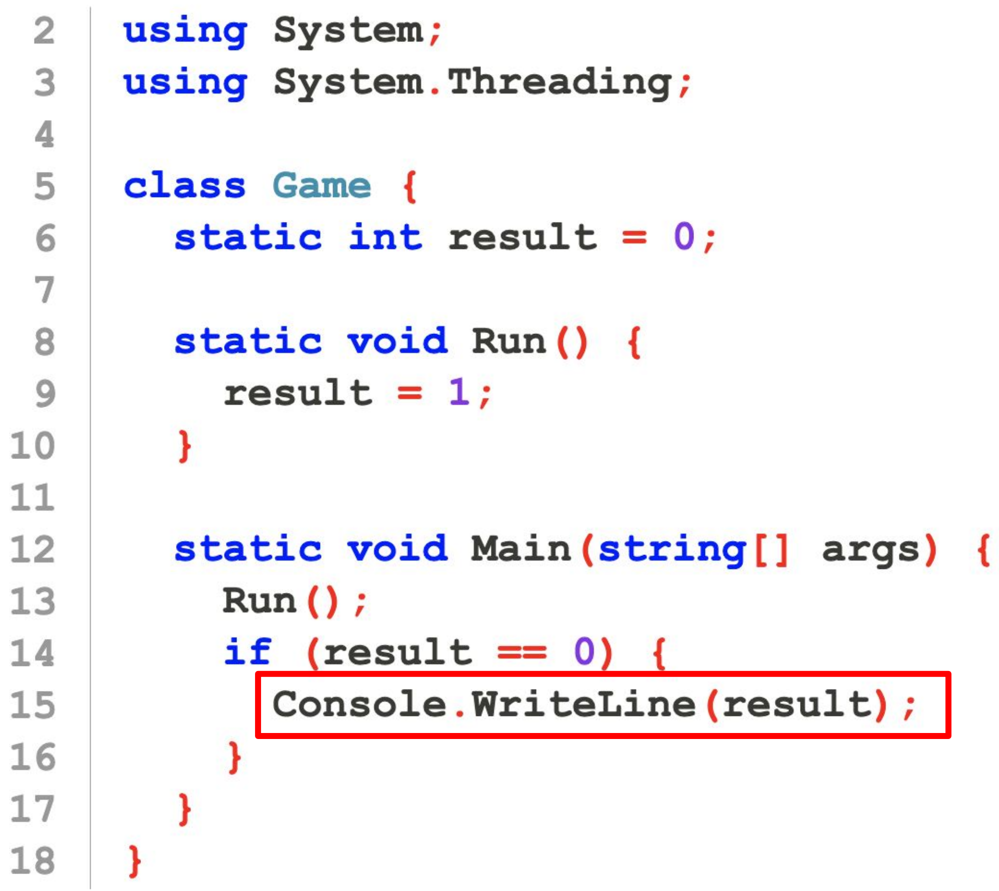


```
using System;
using System.Threading;

class Game {
  static int result = 0;

  static void Run() {
    result = 1;
  }

  static void Main(string[] args) {
    Run();
    if (result == 0) {
      Console.WriteLine(result);
    }
  }
}
```

Shared data

Function

Function



```
using System;
using System.Threading;

class Game {
  static int result = 0;

  static void Run() {
    result = 1;
  }

  static void Main(string[] args) {
    Run();
    if (result == 0) {
      Console.WriteLine(result);
    }
  }
}
```

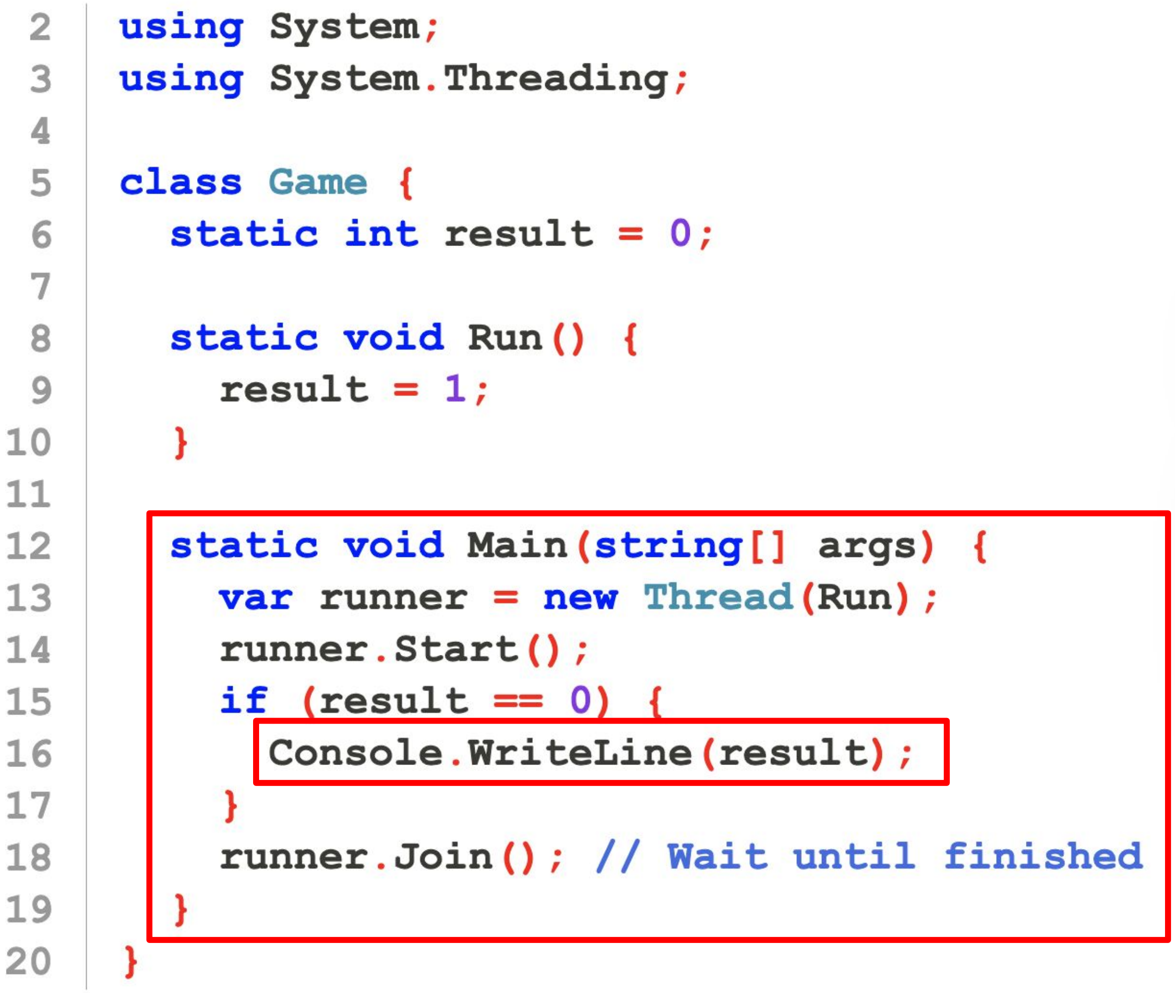
Output ?

1 ?

0 ?

(empty) ?

```csharp
using System;
using System.Threading;

class Game {
  static int result = 0;

  static void Run() {
    result = 1;
  }

  static void Main(string[] args) {
    var runner = new Thread(Run);
    runner.Start();
    if (result == 0) {
      Console.WriteLine(result);
    }
    runner.Join(); // Wait until finished
  }
}
```



```
using System;
using System.Threading;

class Game {
  static int result = 0;

  static void Run() {
    result = 1;
  }

  static void Main(string[] args) {
    var runner = new Thread(Run);
    runner.Start();
    if (result == 0) {
      Console.WriteLine(result);
    }
    runner.Join(); // Wait until finished
  }
}
```

Output ?

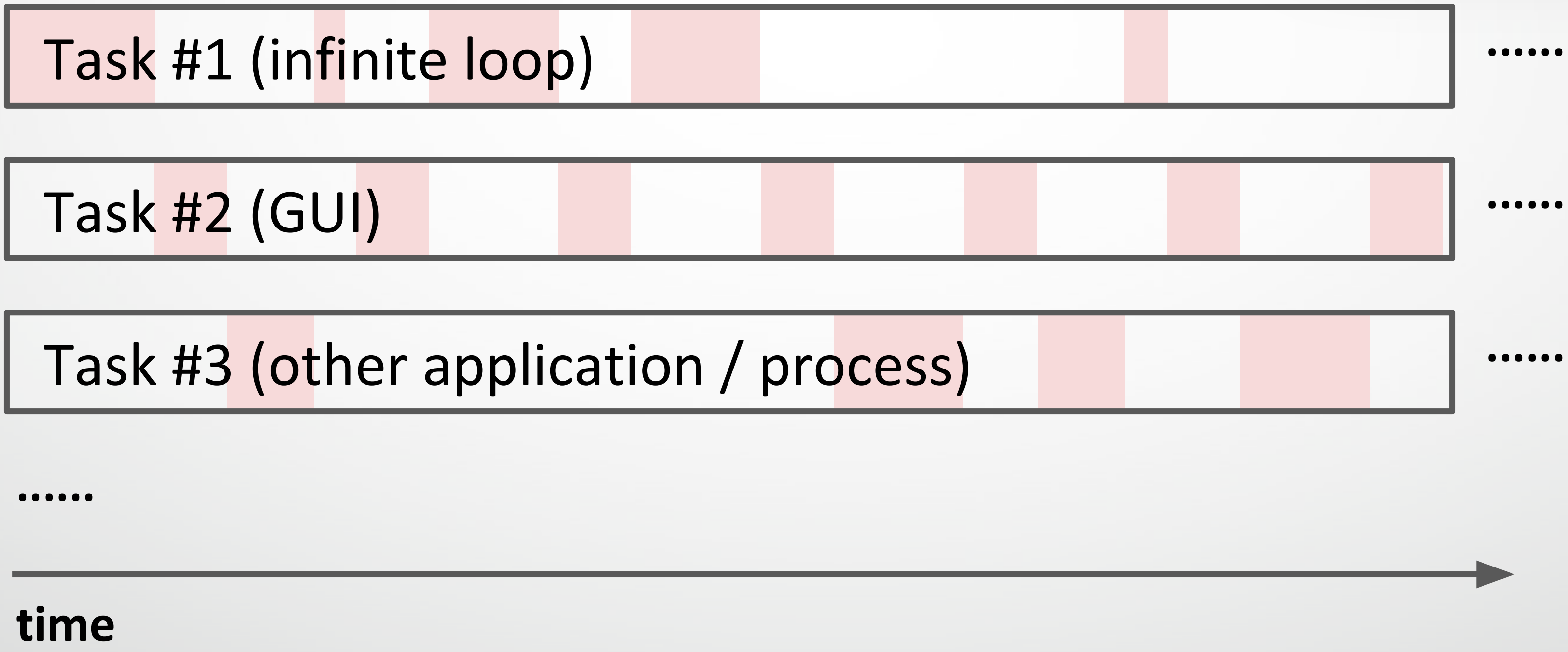
1 ?

0 ?

(empty) ?

Multitasking

Thread-safe ?



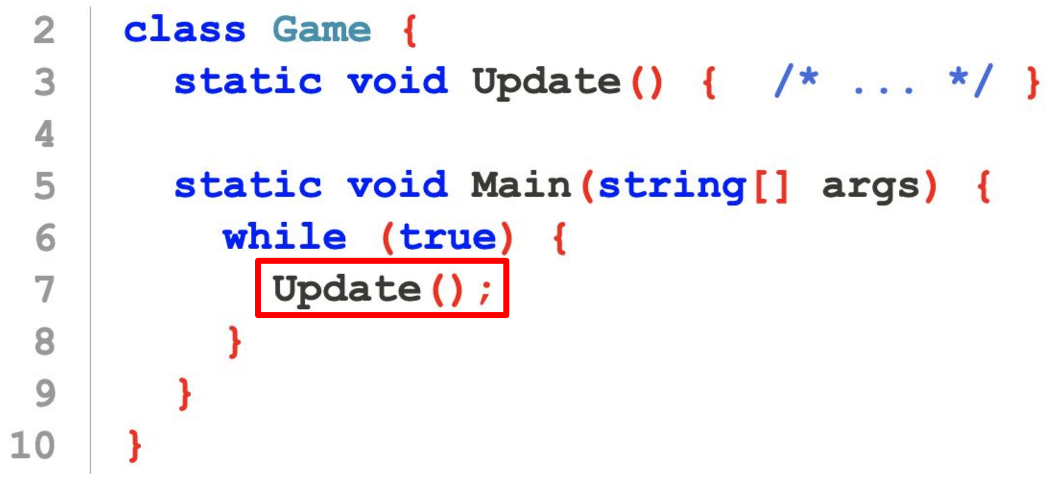
Task #1 (infinite loop) ......

Task #2 (GUI) ......

Task #3 (other application / process) ......

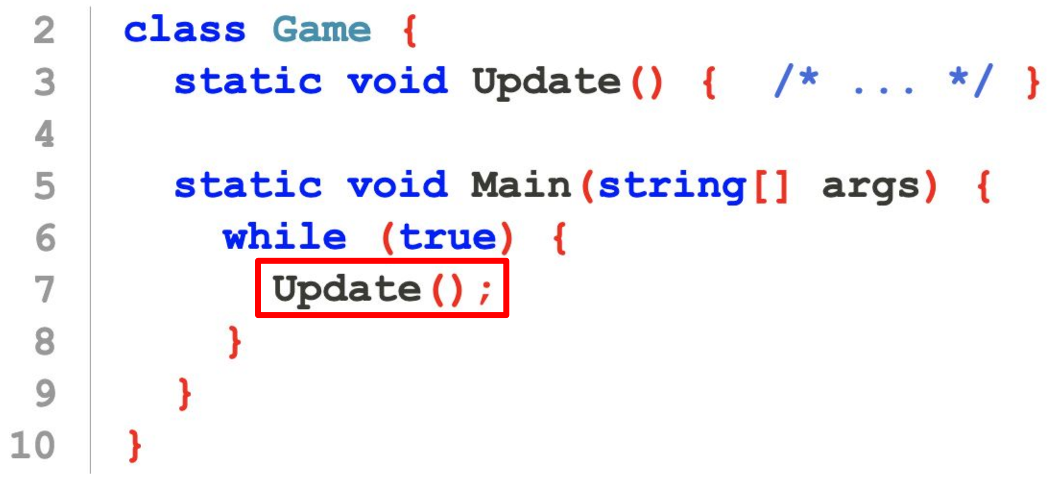
......

time



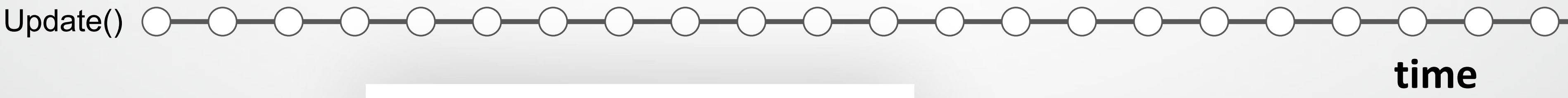
```
class Game {
  static void Update() {  /* ... */ }

  static void Main(string[] args) {
    while (true) {
      Update();
    }
  }
}
```

How about in a game ?

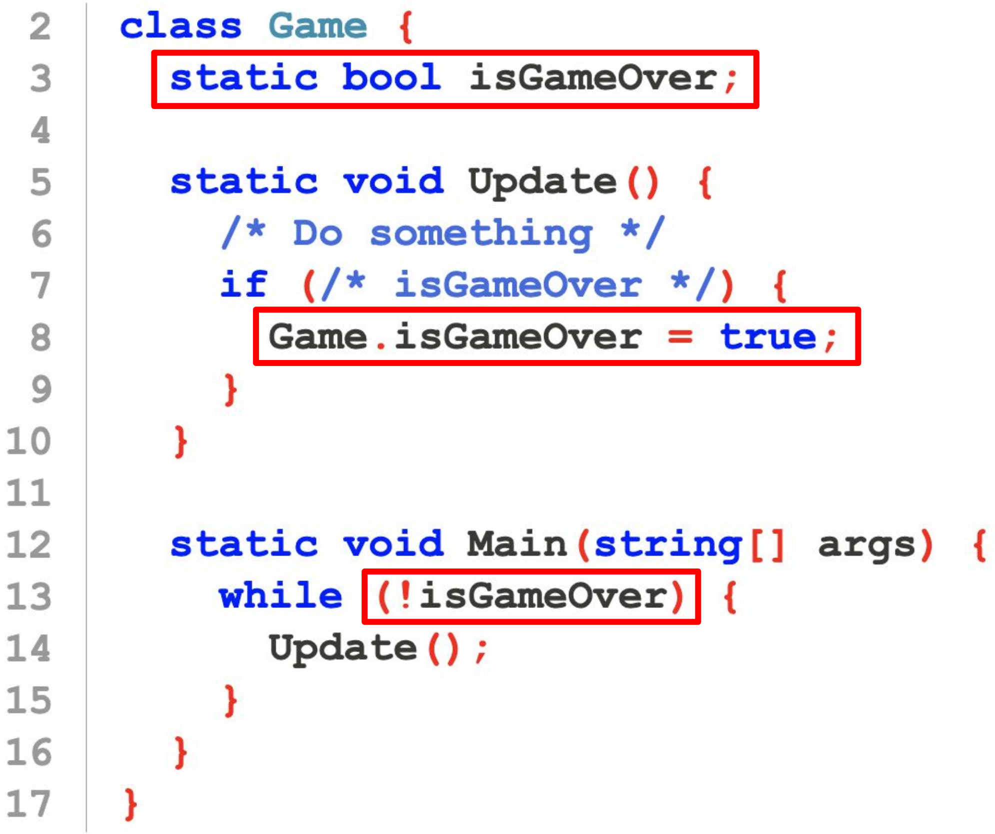


```
class Game {
  static void Update() {  /* ... */ }

  static void Main(string[] args) {
    while (true) {
      Update();
    }
  }
}
```

How about in a game ?



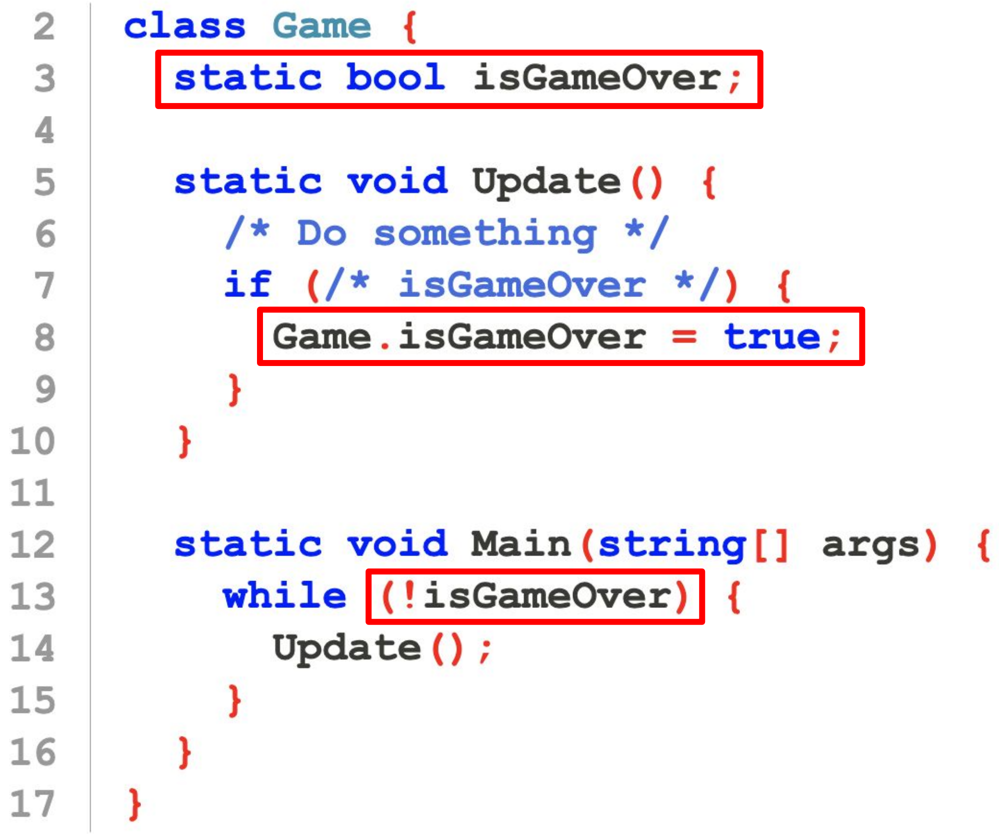
Continuous or discrete ?



```
class Game {
  static bool isGameOver;

  static void Update() {
    /* Do something */
    if (/* isGameOver */) {
      Game.isGameOver = true;
    }
  }

  static void Main(string[] args) {
    while (!isGameOver) {
      Update();
    }
  }
}
```

How about in a game ?

Shared data

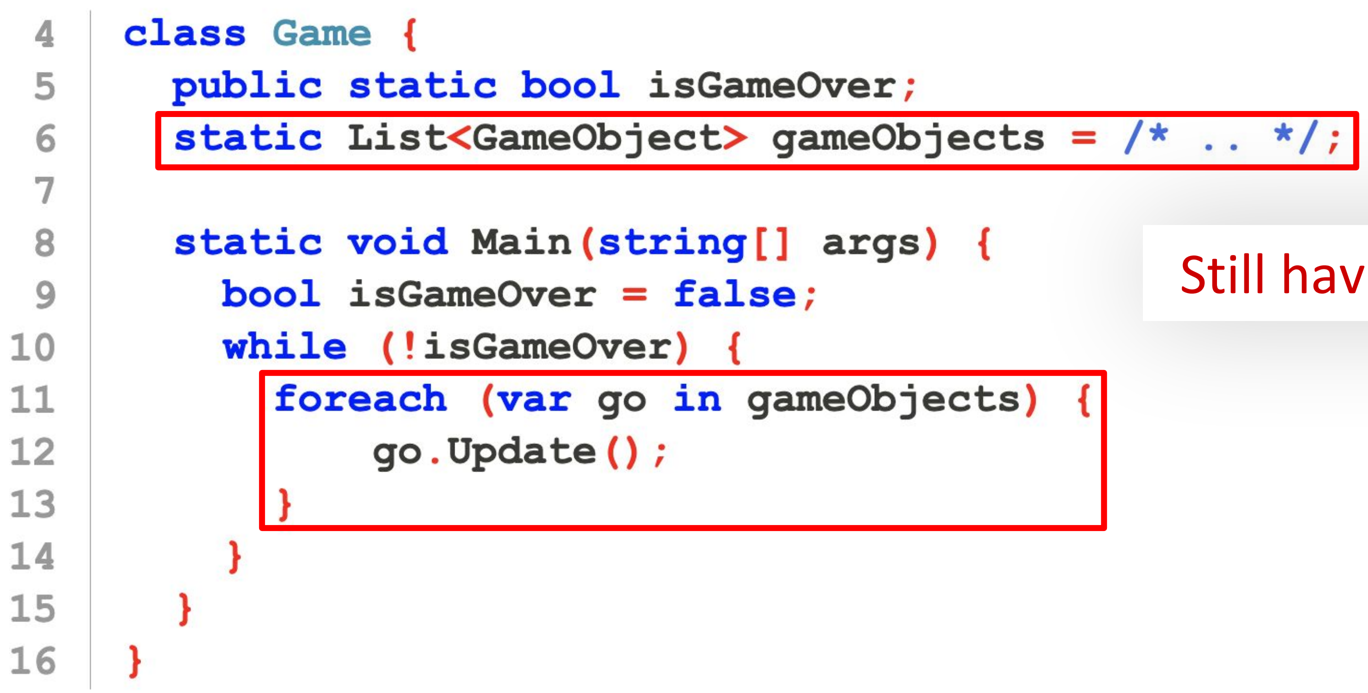


```
class Game {
  static bool isGameOver;

  static void Update() {
    /* Do something */
    if (/* isGameOver */) {
      Game.isGameOver = true;
    }
  }

  static void Main(string[] args) {
    while (!isGameOver) {
      Update();
    }
  }
}
```

How about in a game ?

Shared data

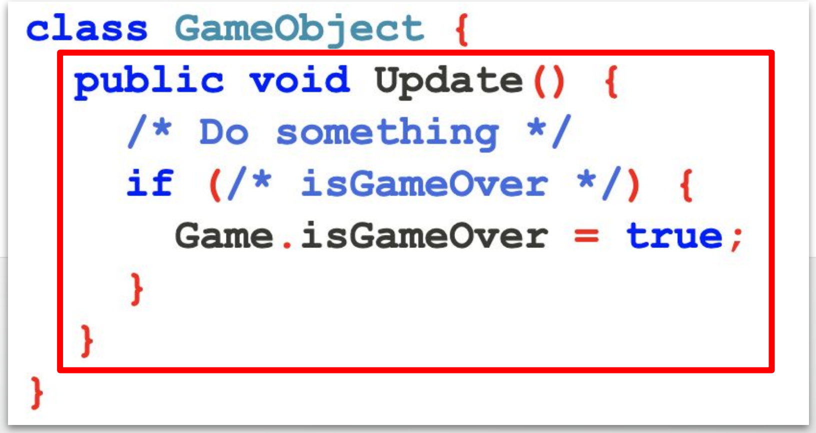
Thread safe ?



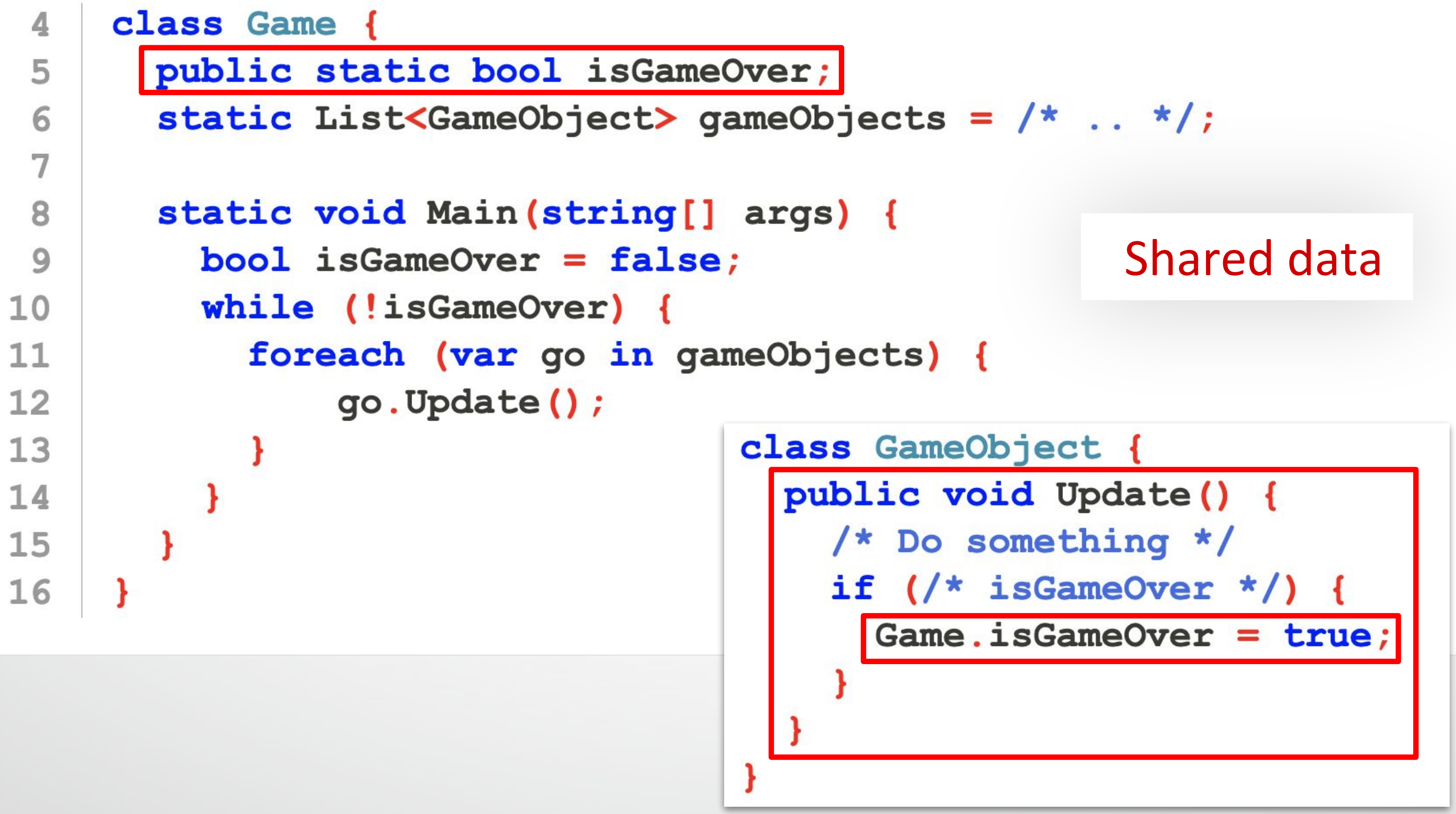
```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
    }
  }
}
```

Still have many objects

```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
    }
  }
}
```



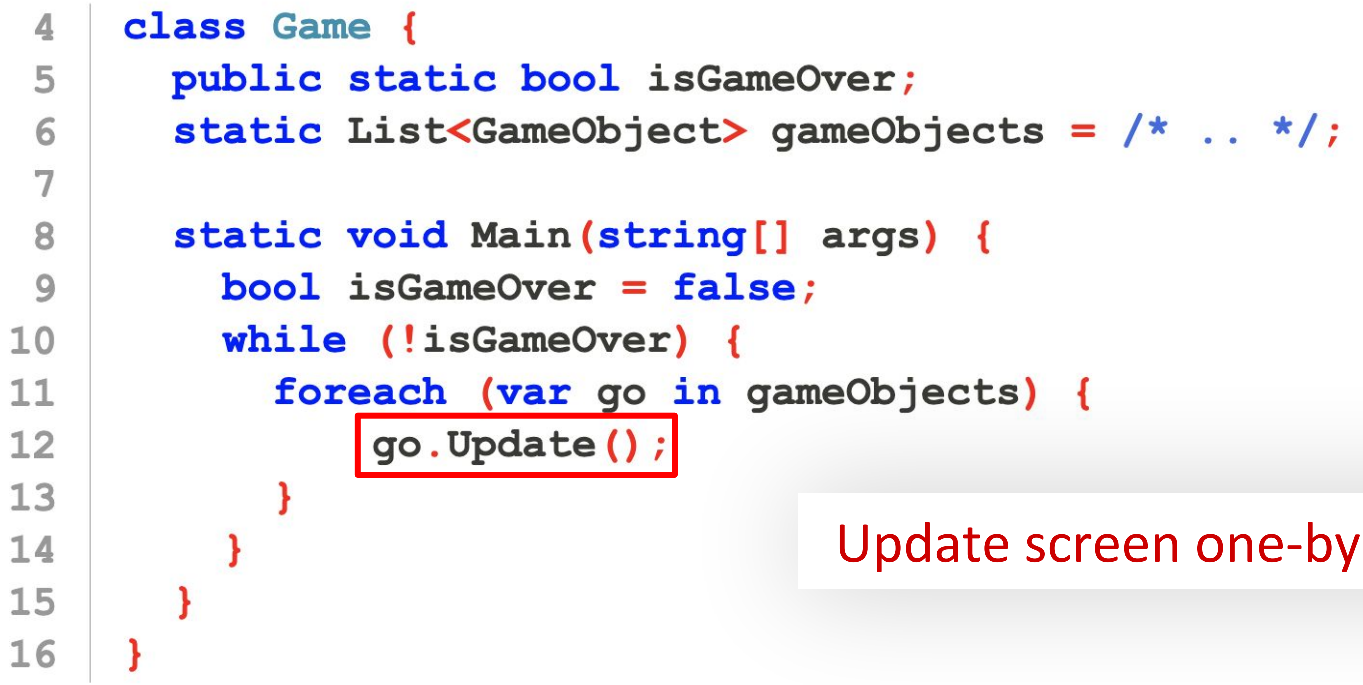
```
class GameObject {
  public void Update() {
    /* Do something */
    if (/* isGameOver */) {
      Game.isGameOver = true;
    }
  }
}
```



```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
    }
  }
}
```

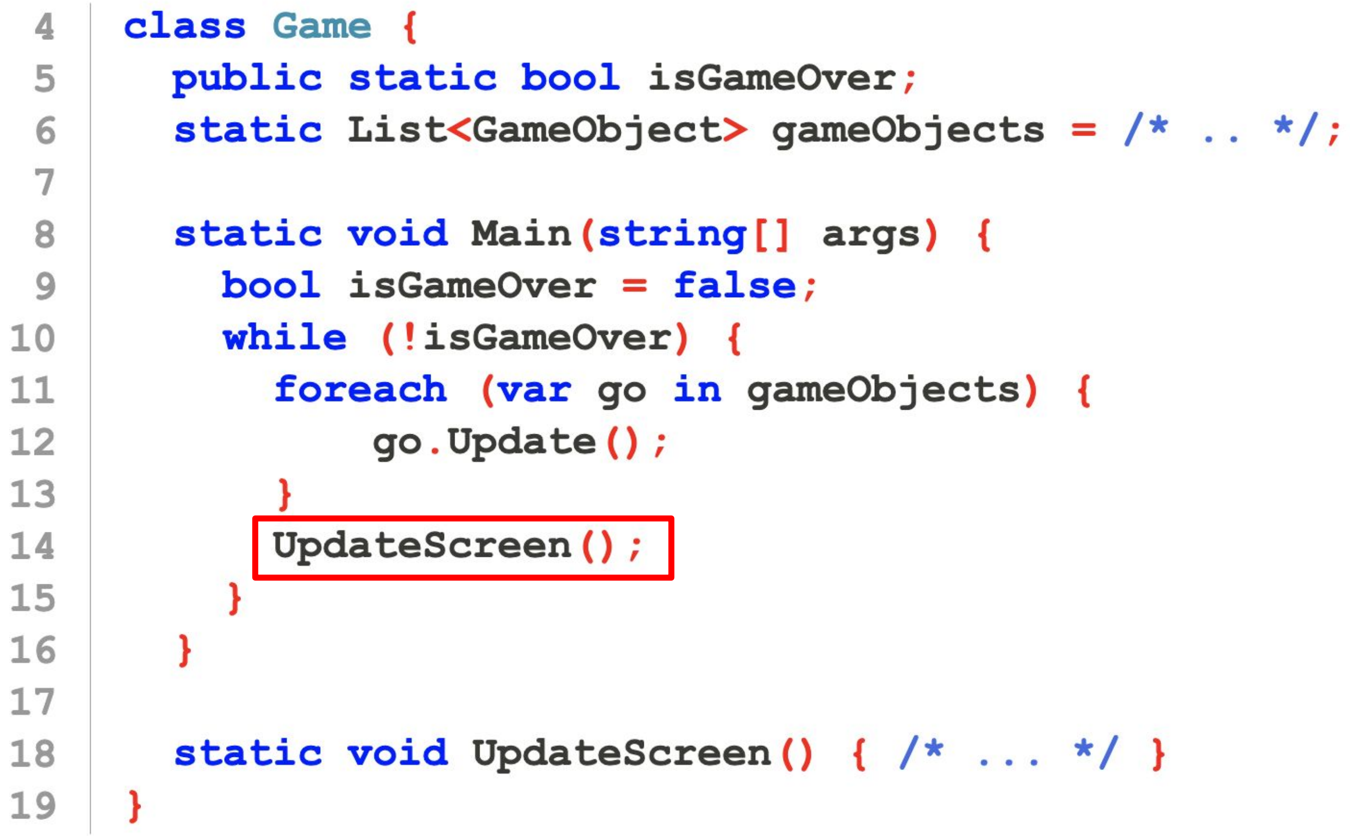
Shared data

```
class GameObject {
  public void Update() {
    /* Do something */
    if (/* isGameOver */) {
      Game.isGameOver = true;
    }
  }
}
```

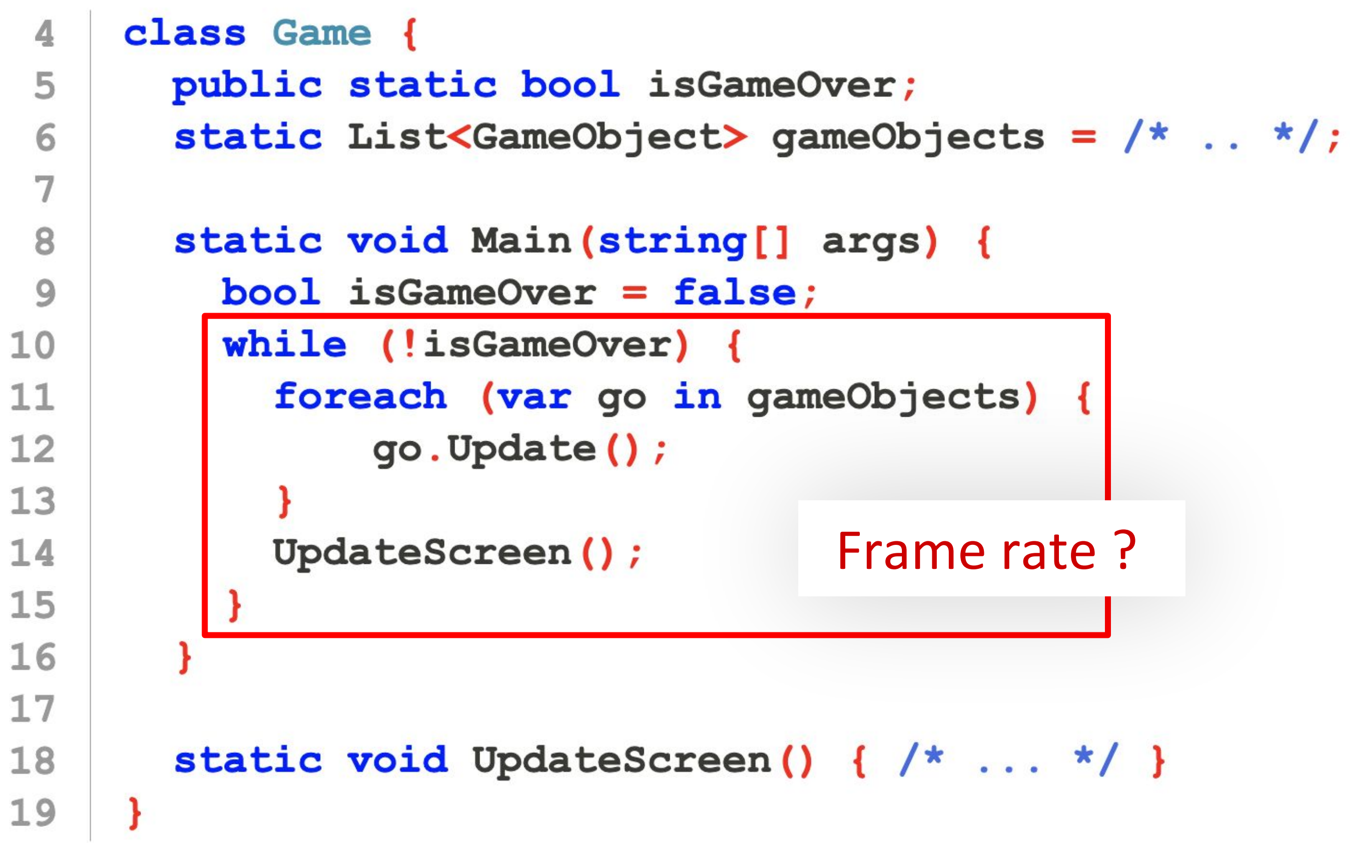


```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
    }
  }
}
```

Update screen one-by-one ?

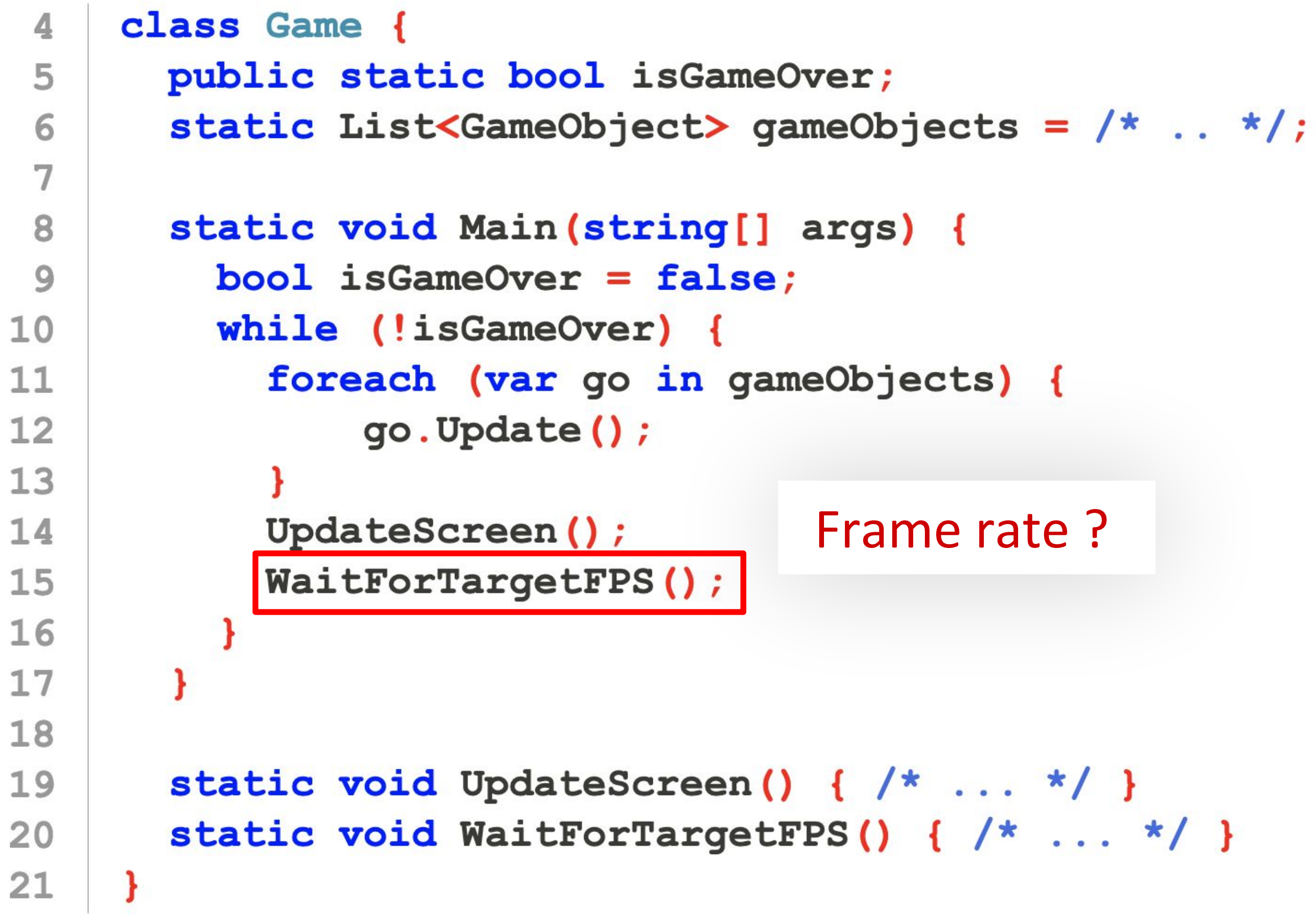


```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
    }
  }

  static void UpdateScreen() { /* ... */ }
}
```



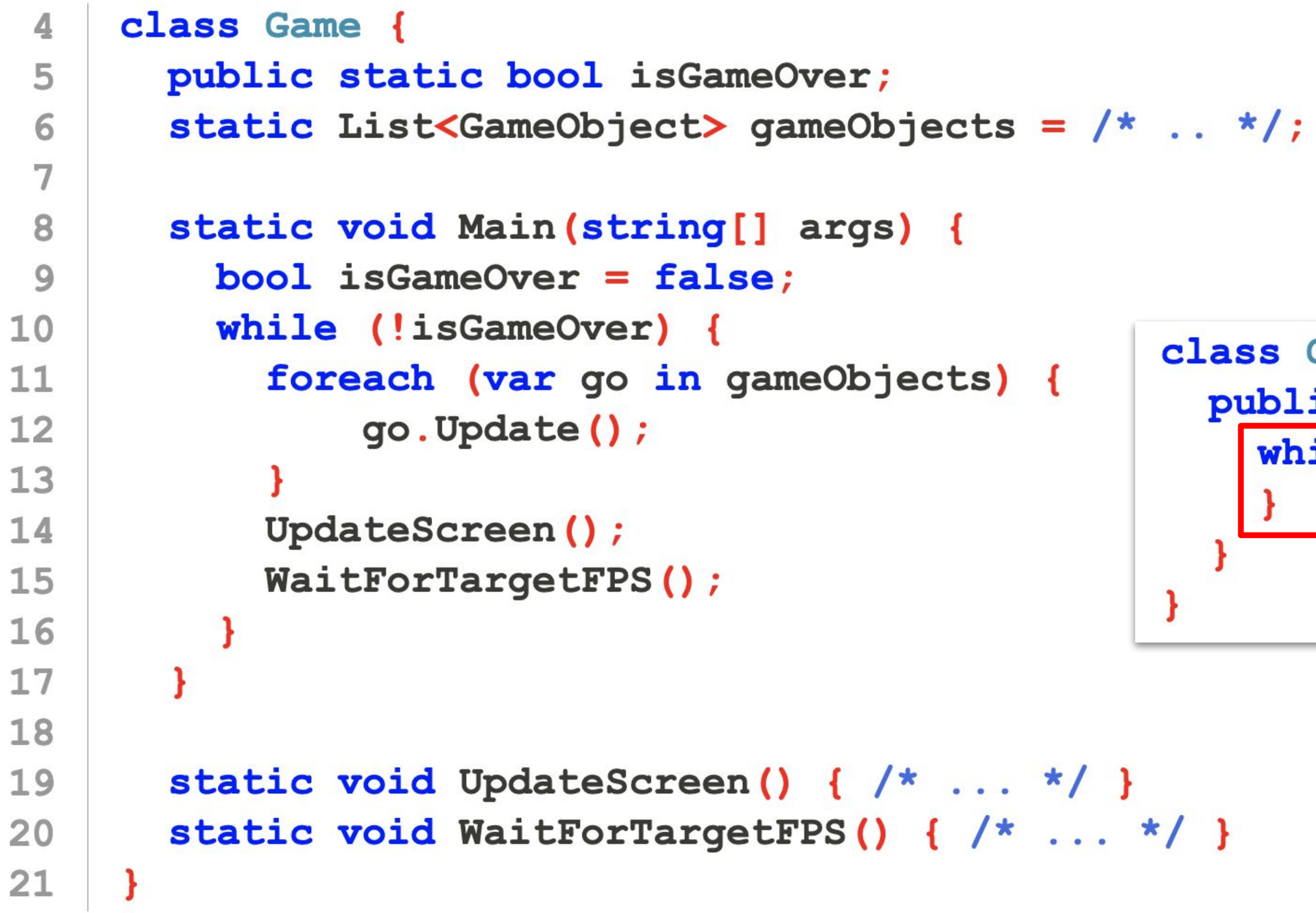
```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
    }
  }

  static void UpdateScreen() { /* ... */ }
}
```

Frame rate ?

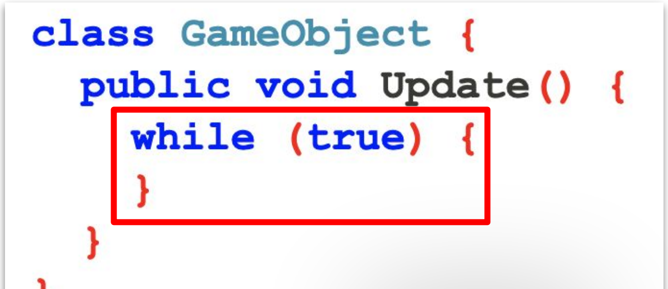


```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }

  static void UpdateScreen() { /* ... */ }
  static void WaitForTargetFPS() { /* ... */ }
}
```

Frame rate ?



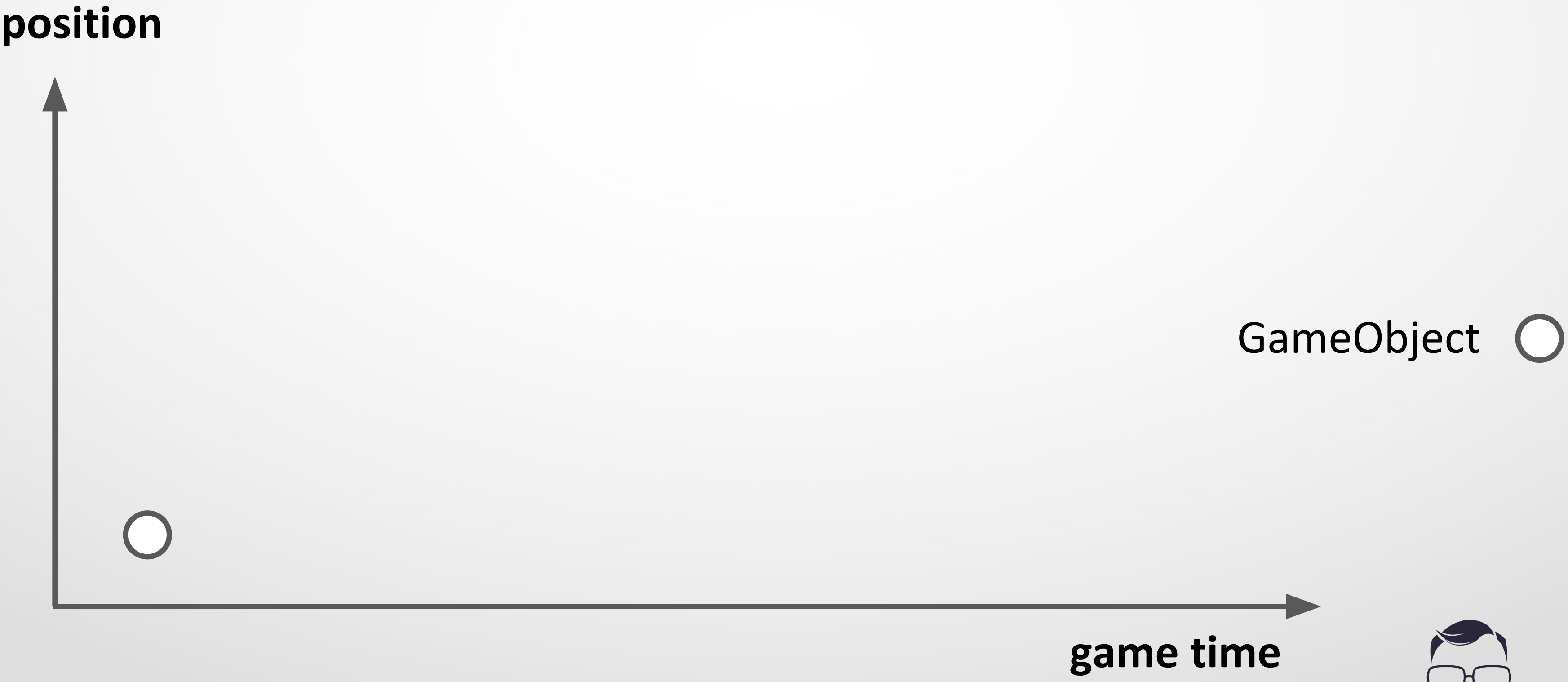
```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }

  static void UpdateScreen() { /* ... */ }
  static void WaitForTargetFPS() { /* ... */ }
}
```



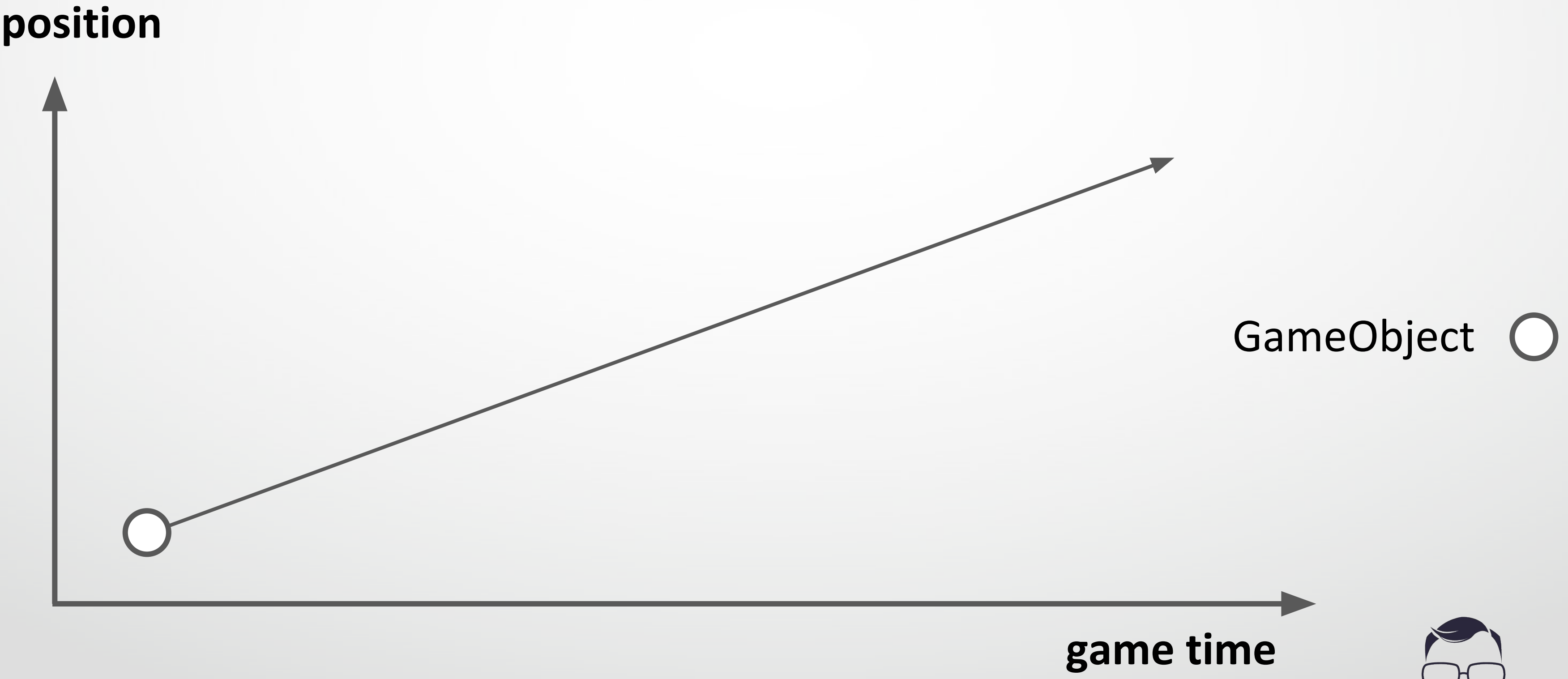
```
class GameObject {
  public void Update() {
    while (true) {
    }
  }
}
```

Hang ?

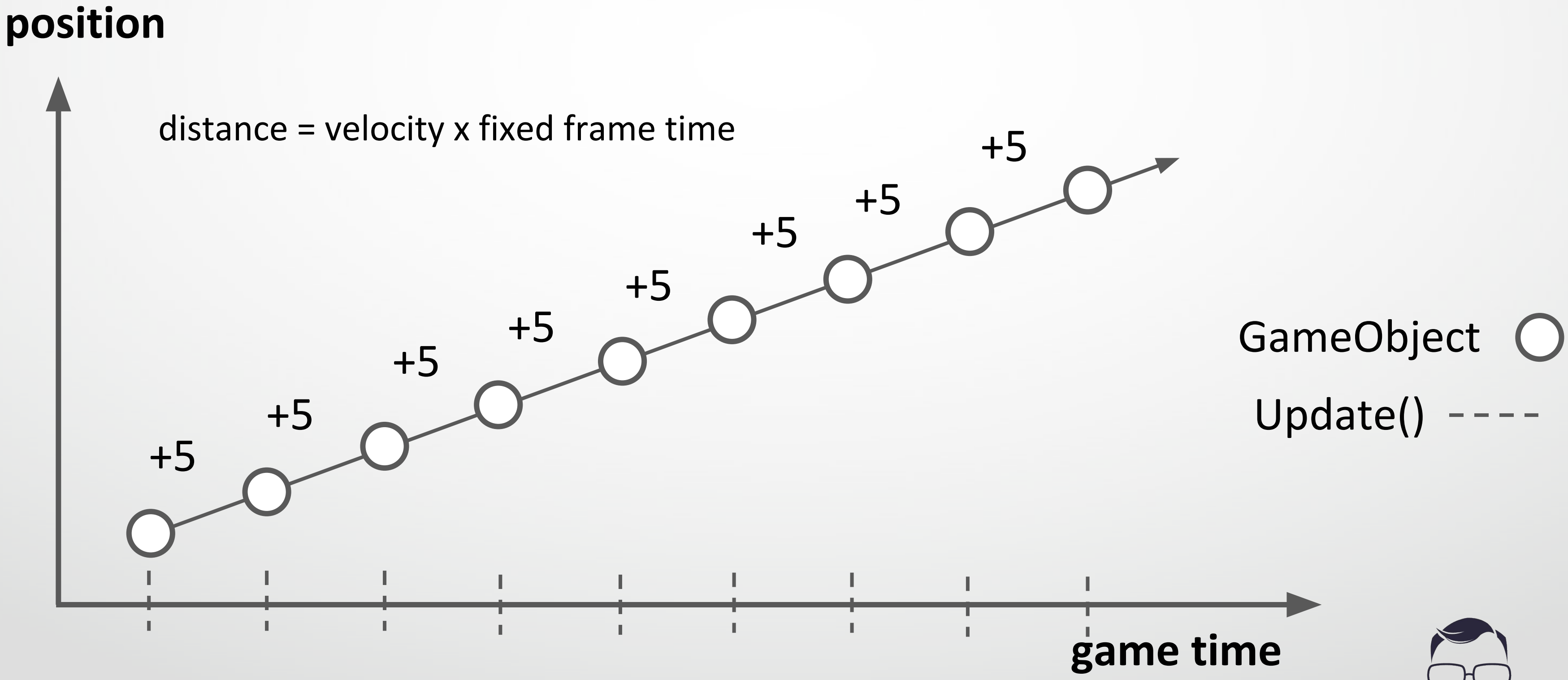
Object movement



Object moving with constant velocity

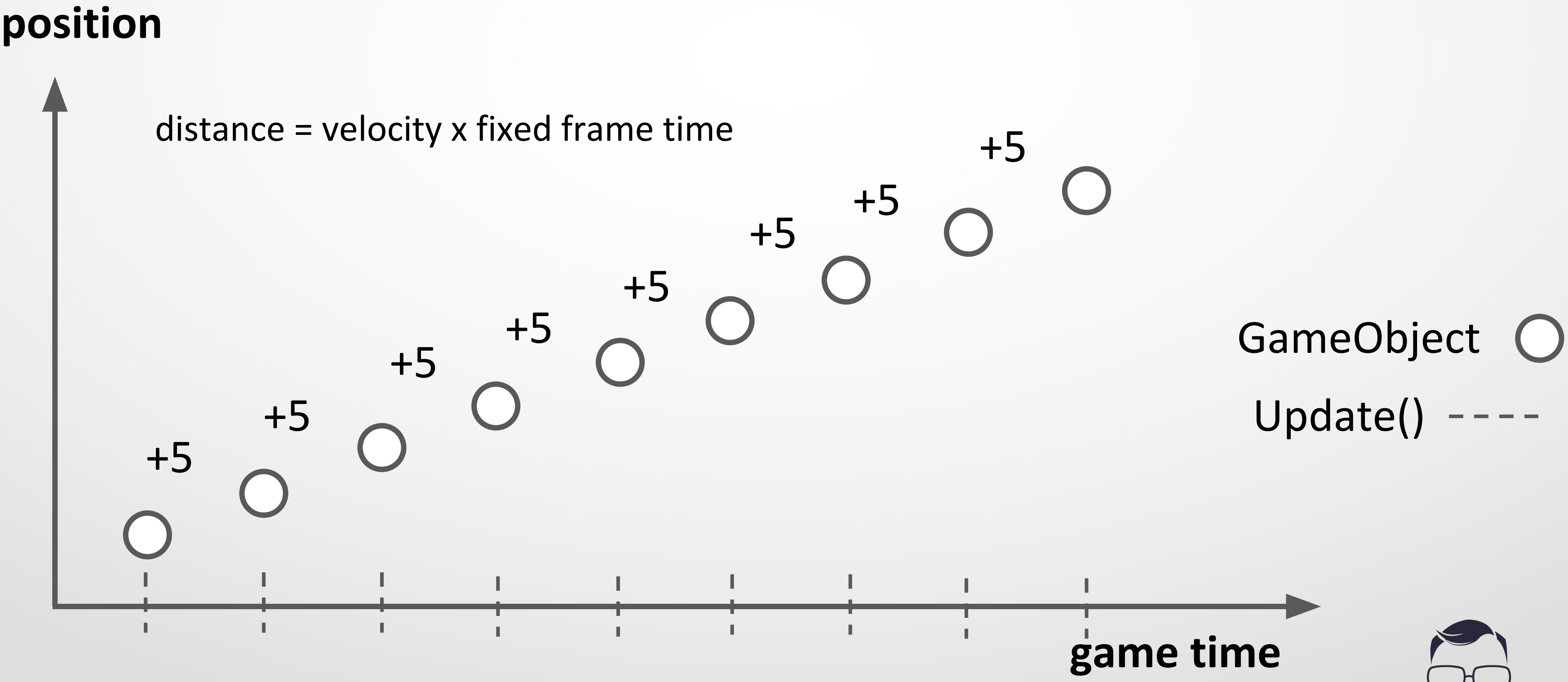


Object moving with constant velocity

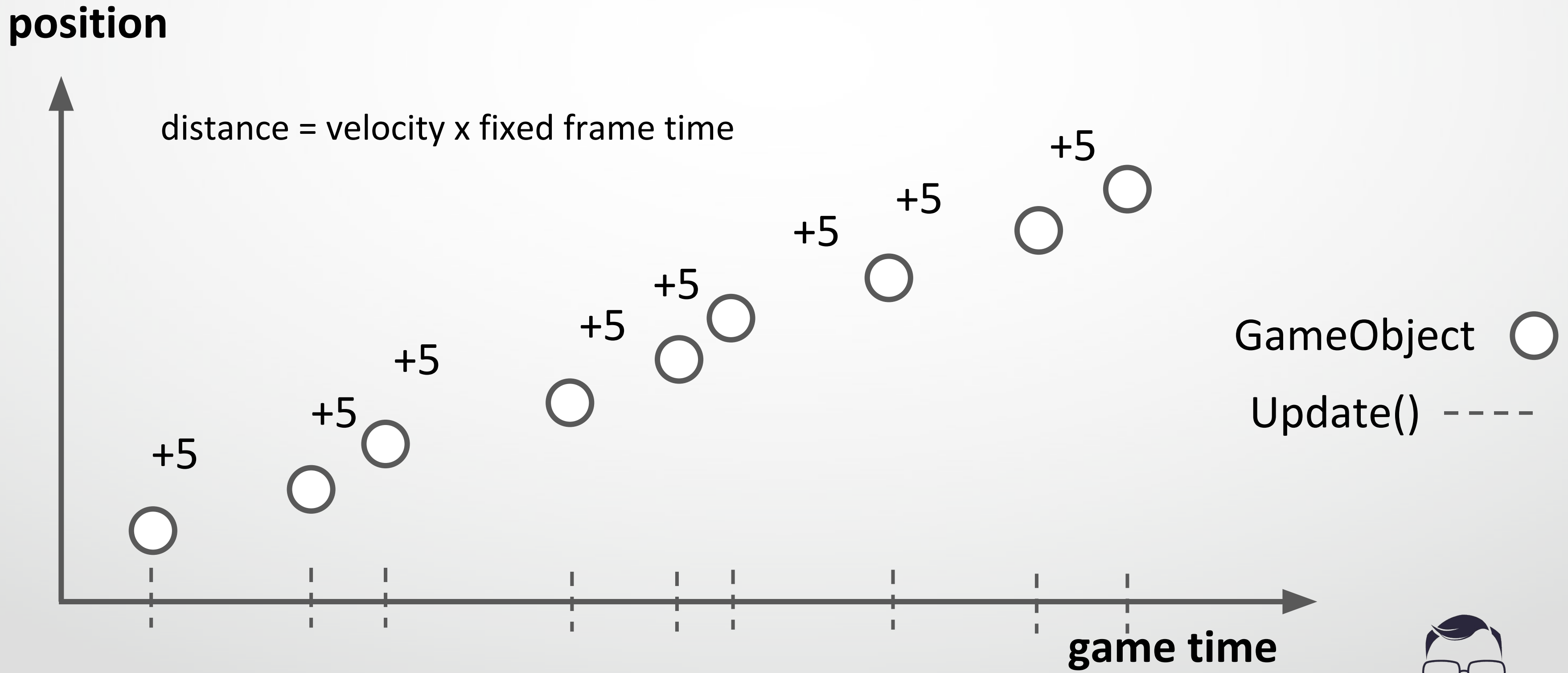


distance = velocity x fixed frame time

Object moving with constant velocity

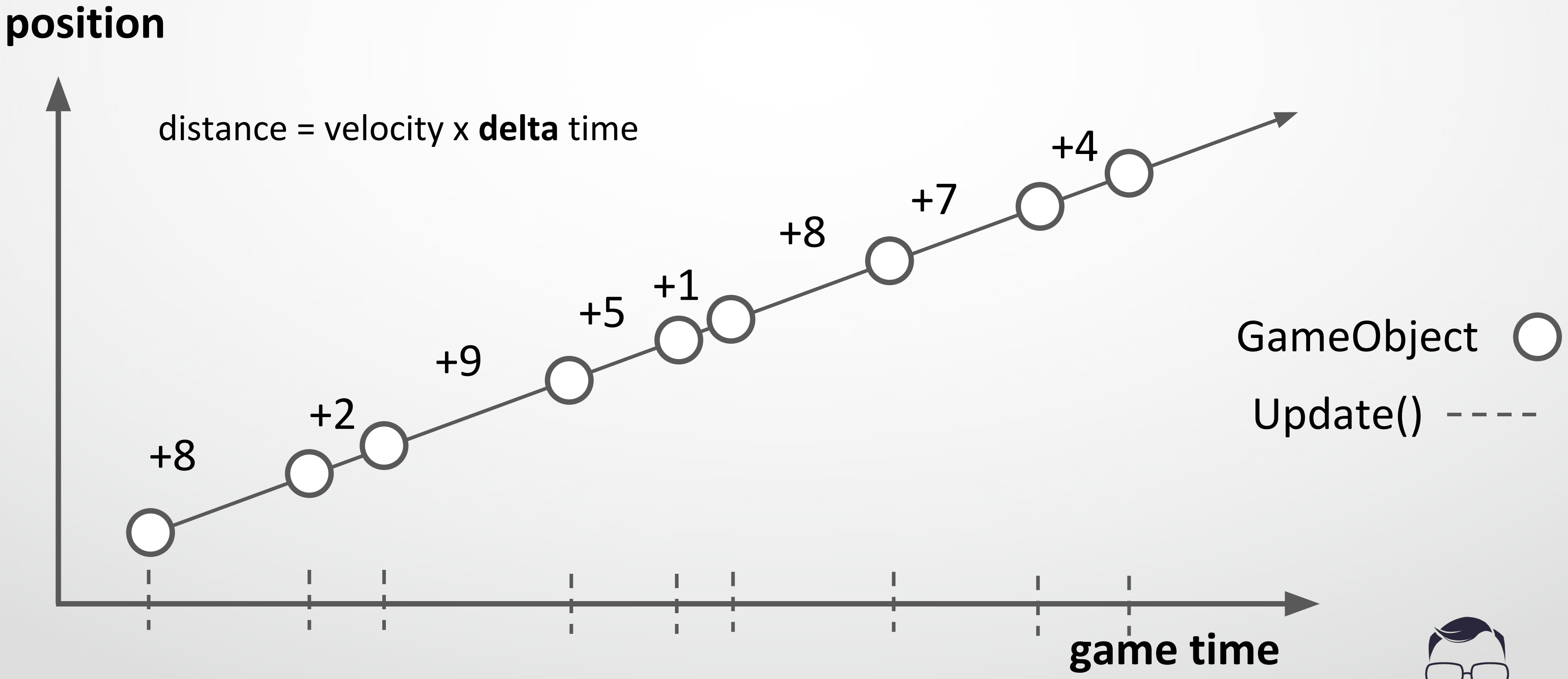


Object moving with constant velocity ?

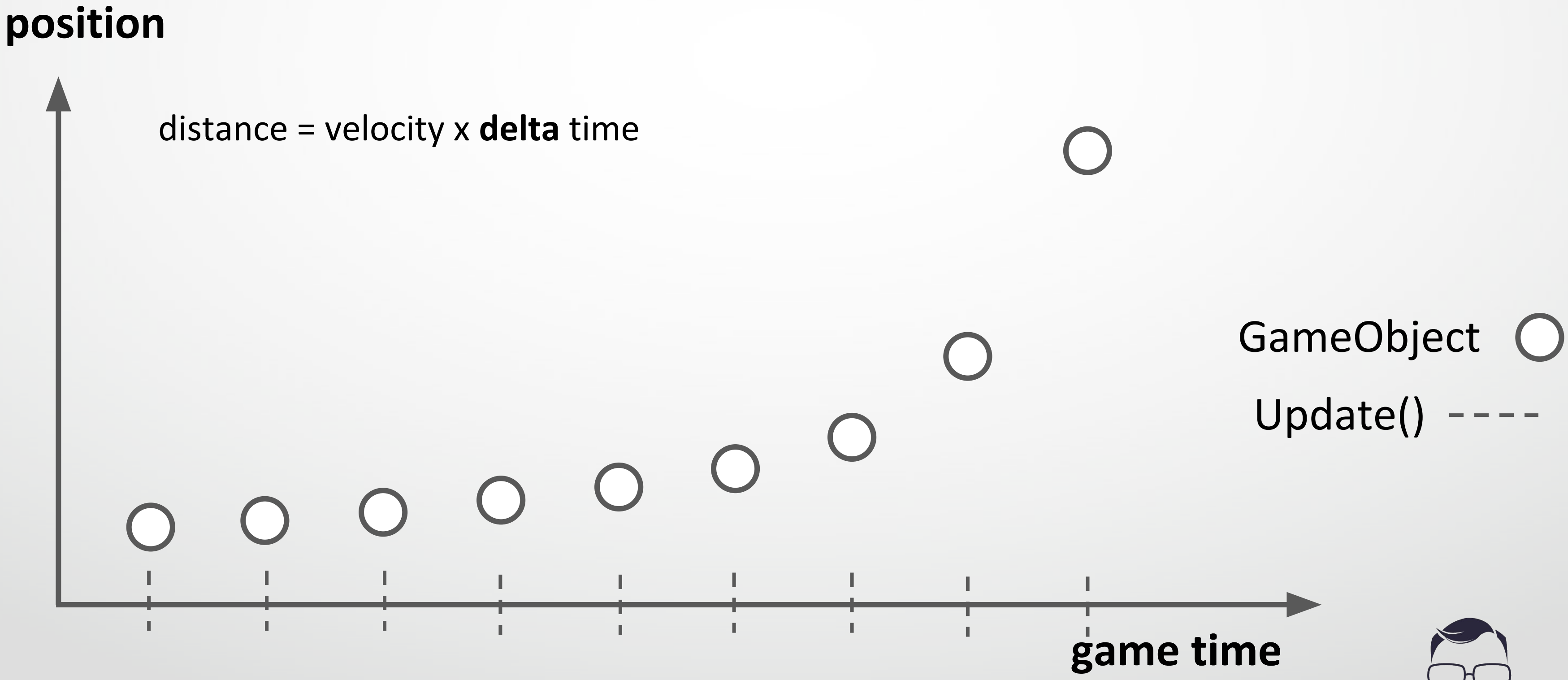


FeisStu

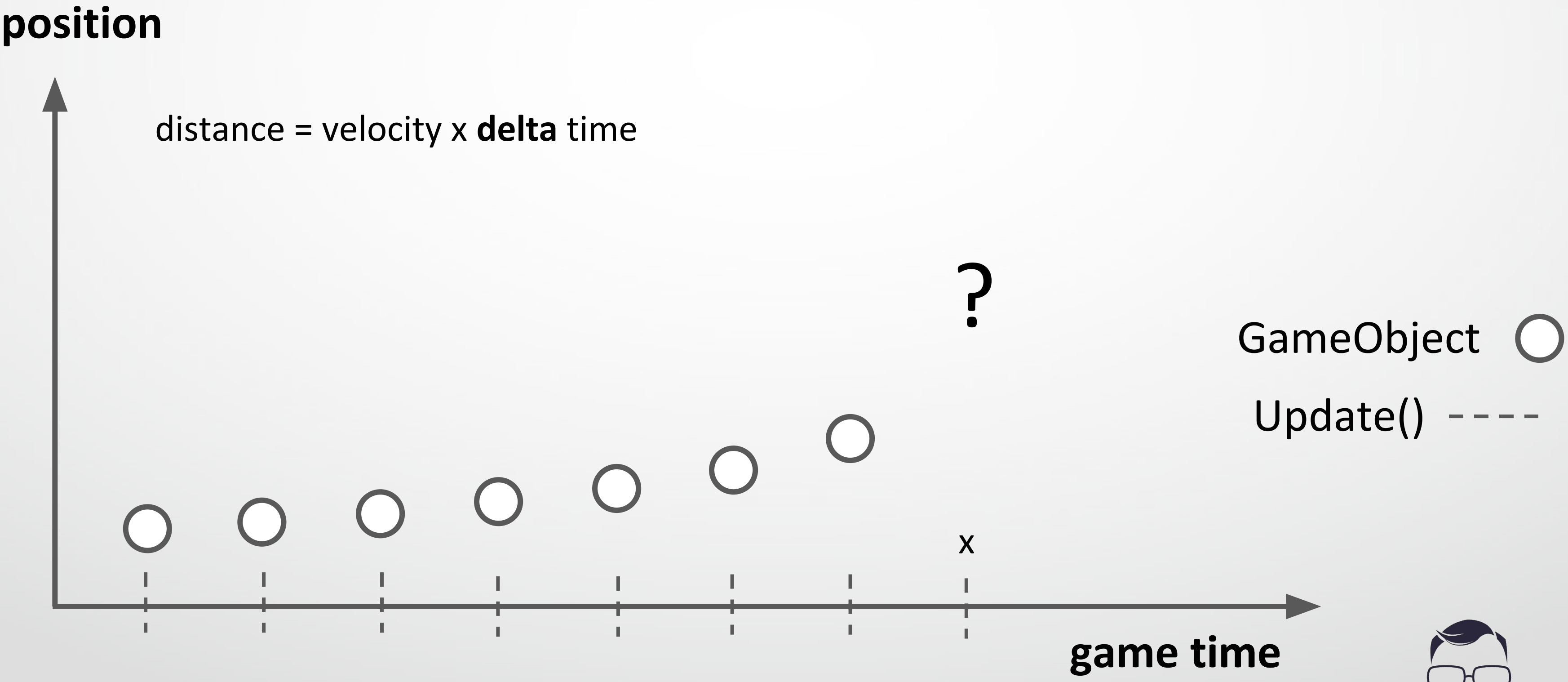
Object moving with constant velocity



Object moving with uniform acceleration



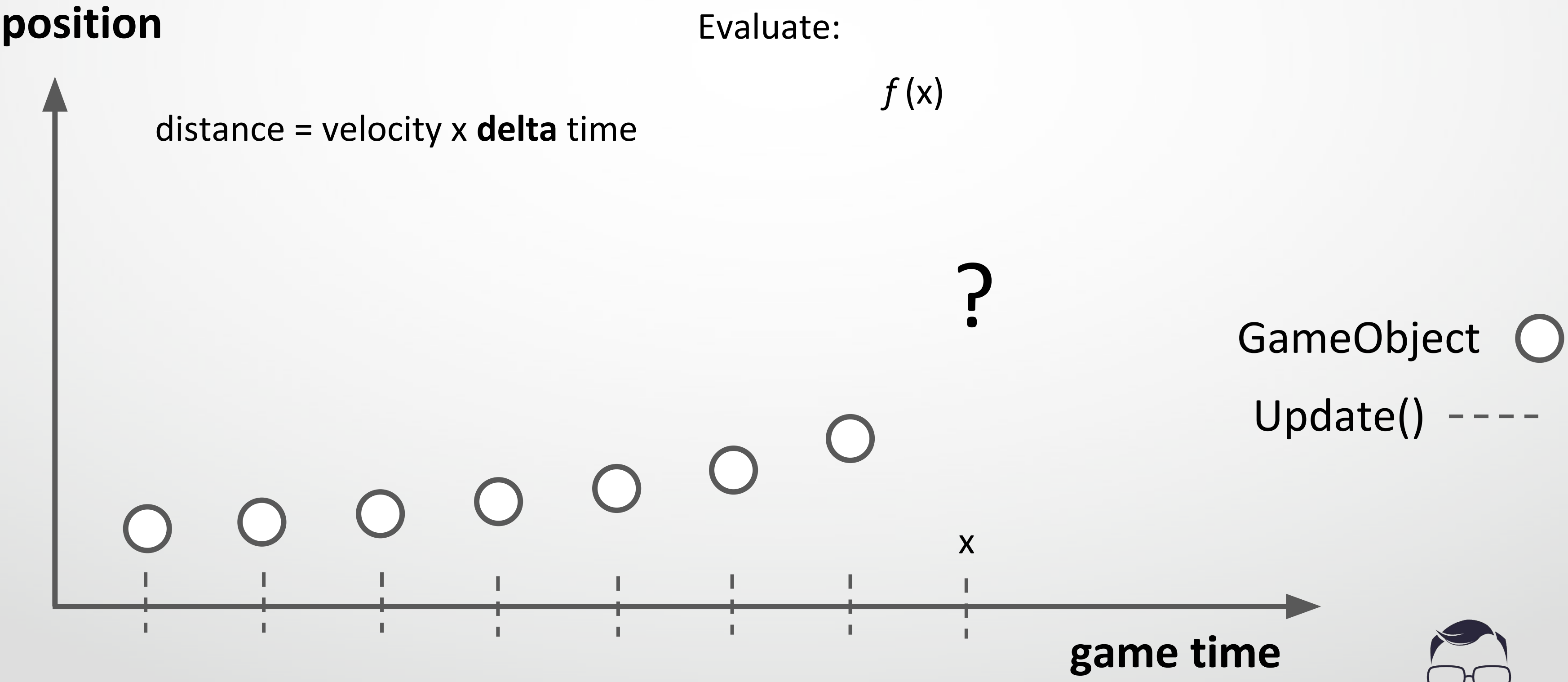
Object moving with uniform acceleration



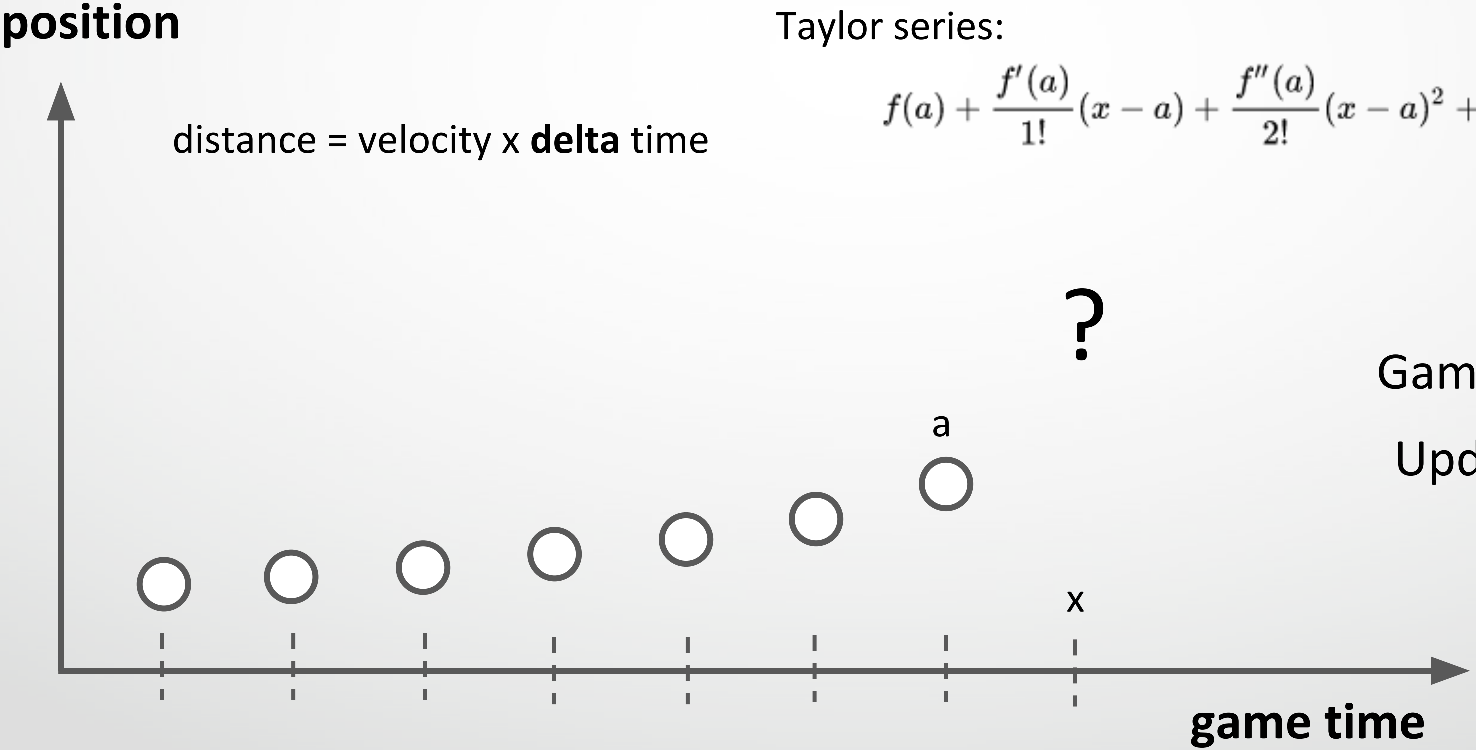
Object moving with uniform acceleration

Evaluate:

$f\,(x)$



Object moving with uniform acceleration



distance = velocity x **delta** time

Taylor series:

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots,$$

GameObject ○

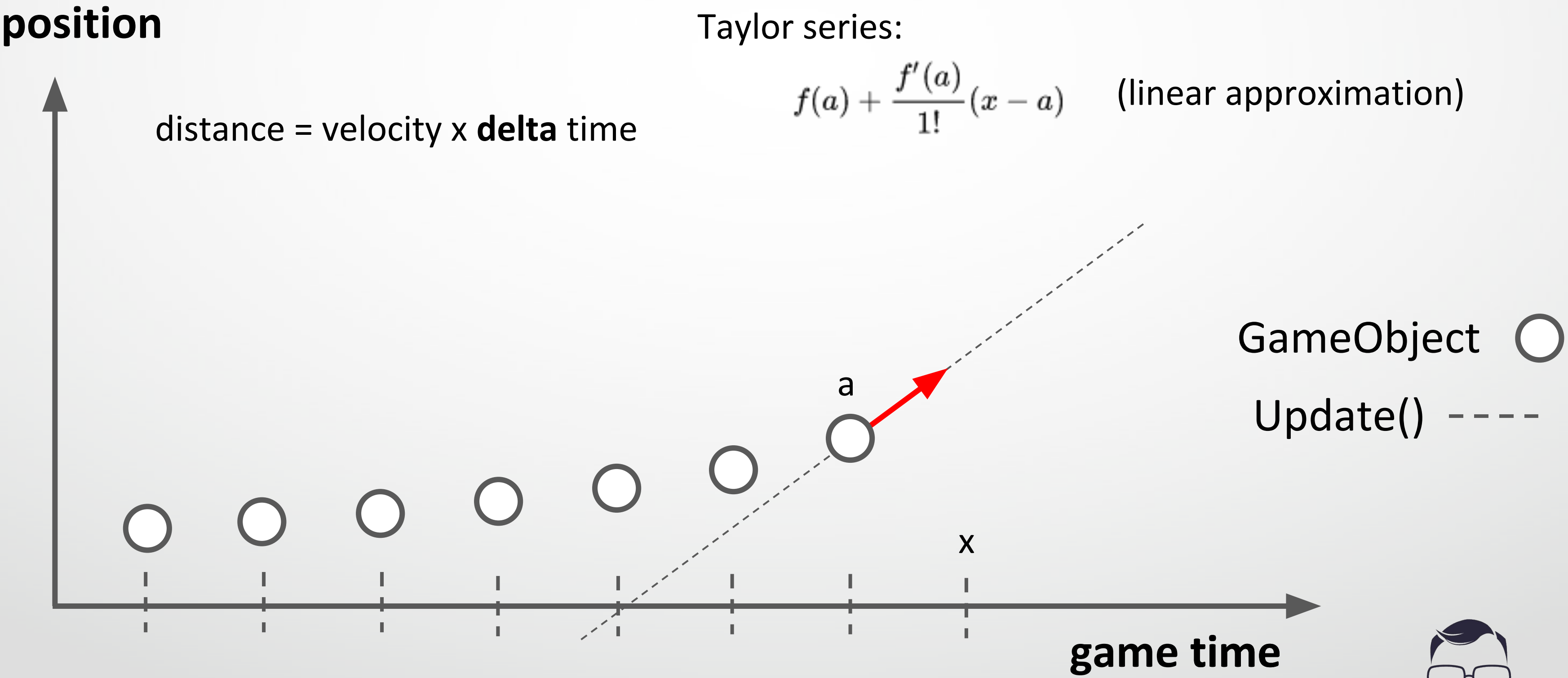
Update() - - - -

Object moving with uniform acceleration

Taylor series:

$$f(a) + \frac{f'(a)}{1!}(x-a)$$

(linear approximation)



Object moving with uniform acceleration

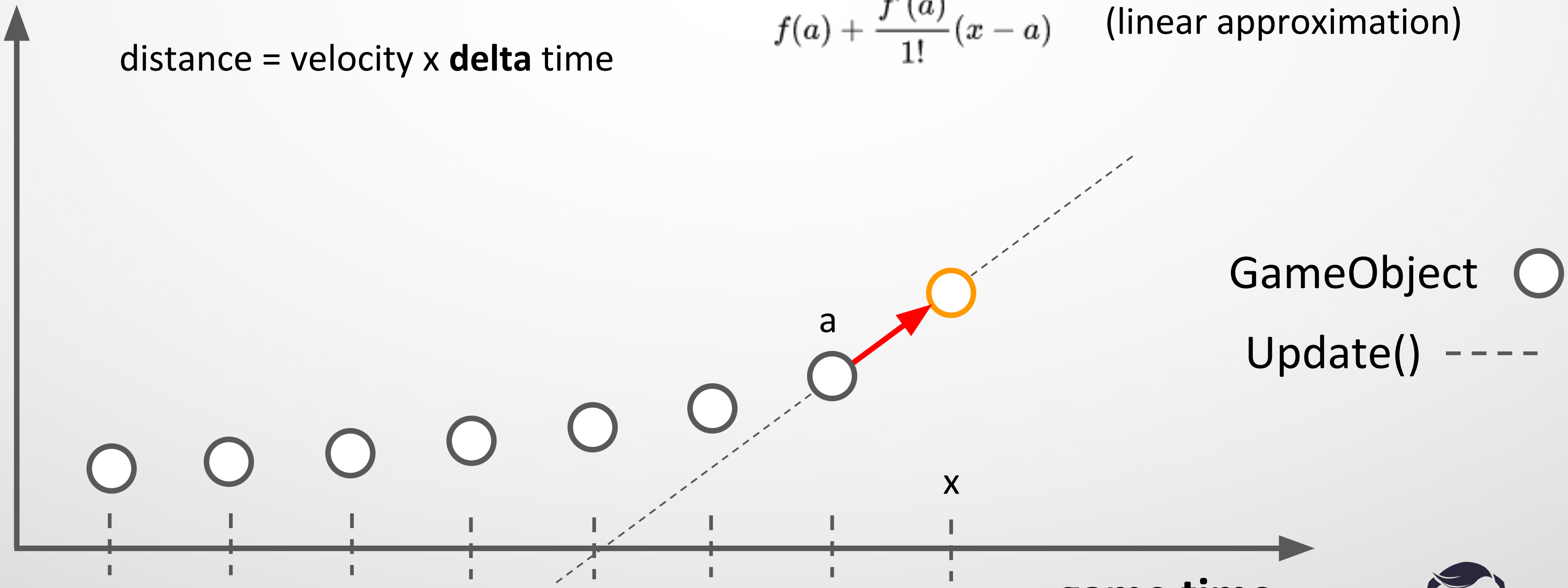
position

distance = velocity x **delta** time

Taylor series:

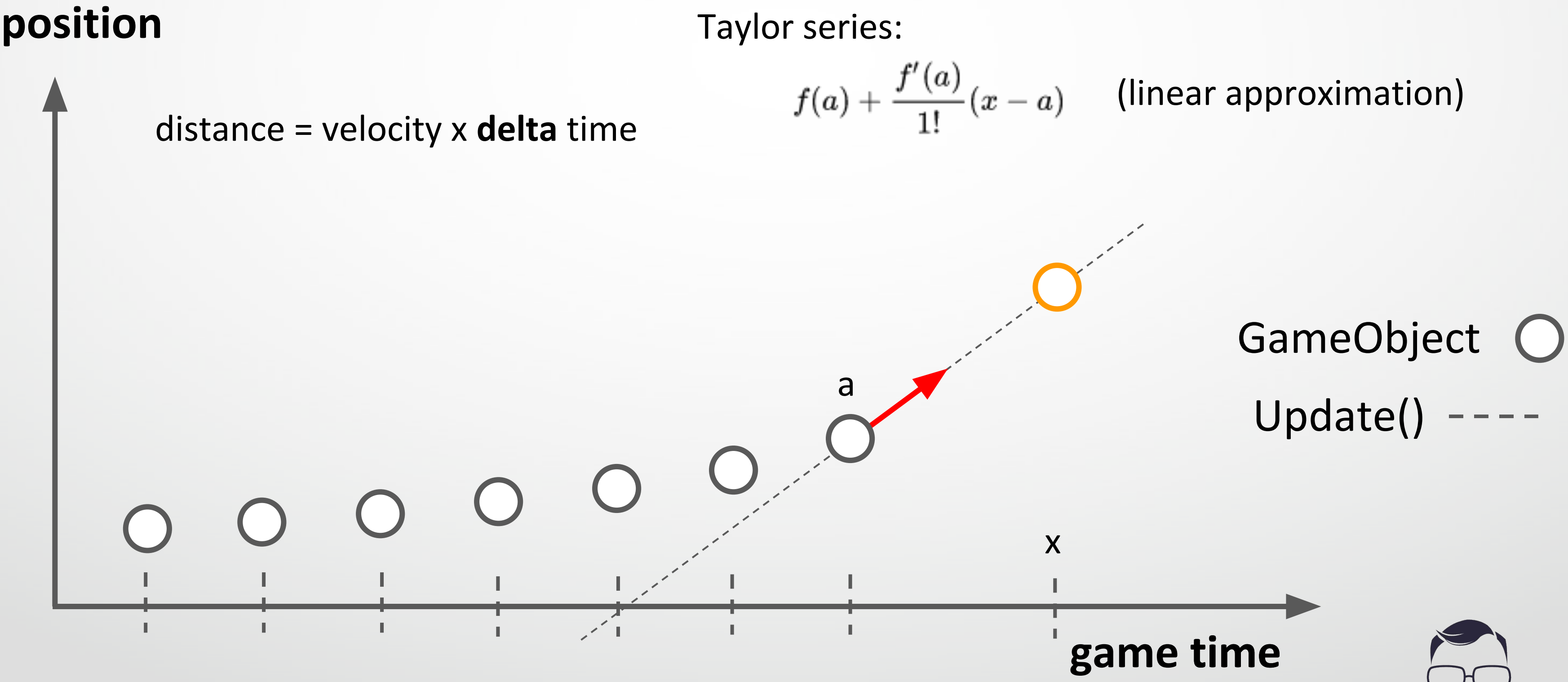
$$f(a) + \frac{f'(a)}{1!}(x - a)$$

(linear approximation)



game time

Object moving with uniform acceleration

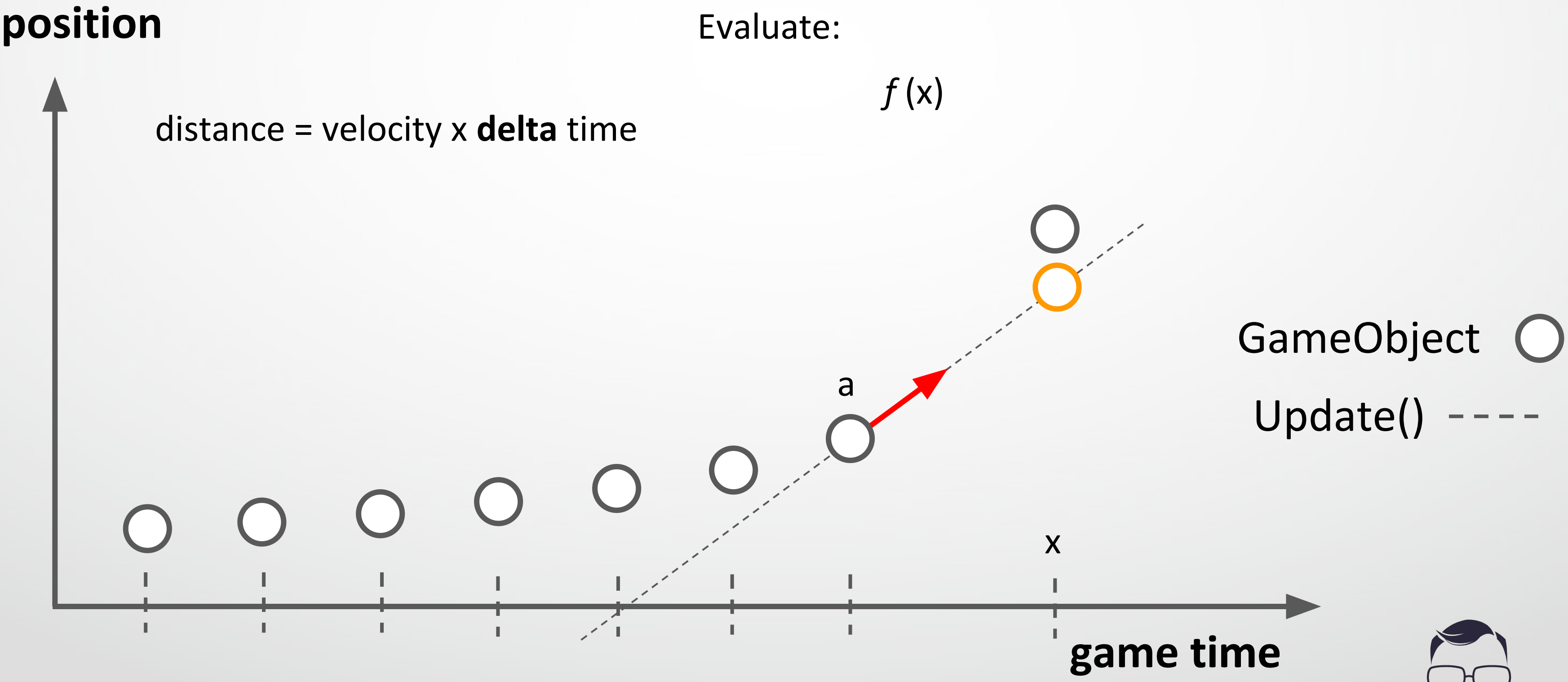


distance = velocity x **delta** time

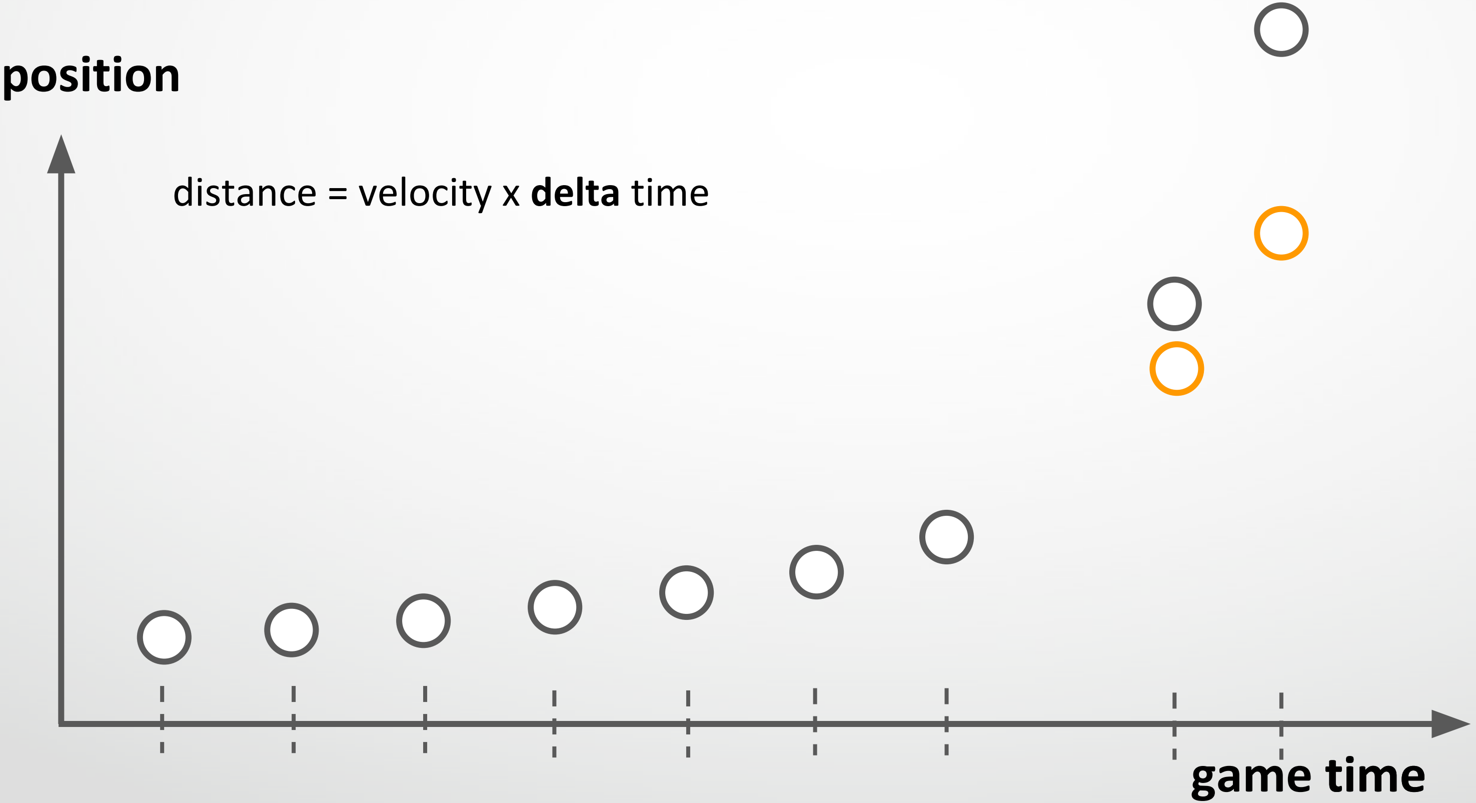
Taylor series:

$$f(a) + \frac{f'(a)}{1!}(x-a)$$ (linear approximation)

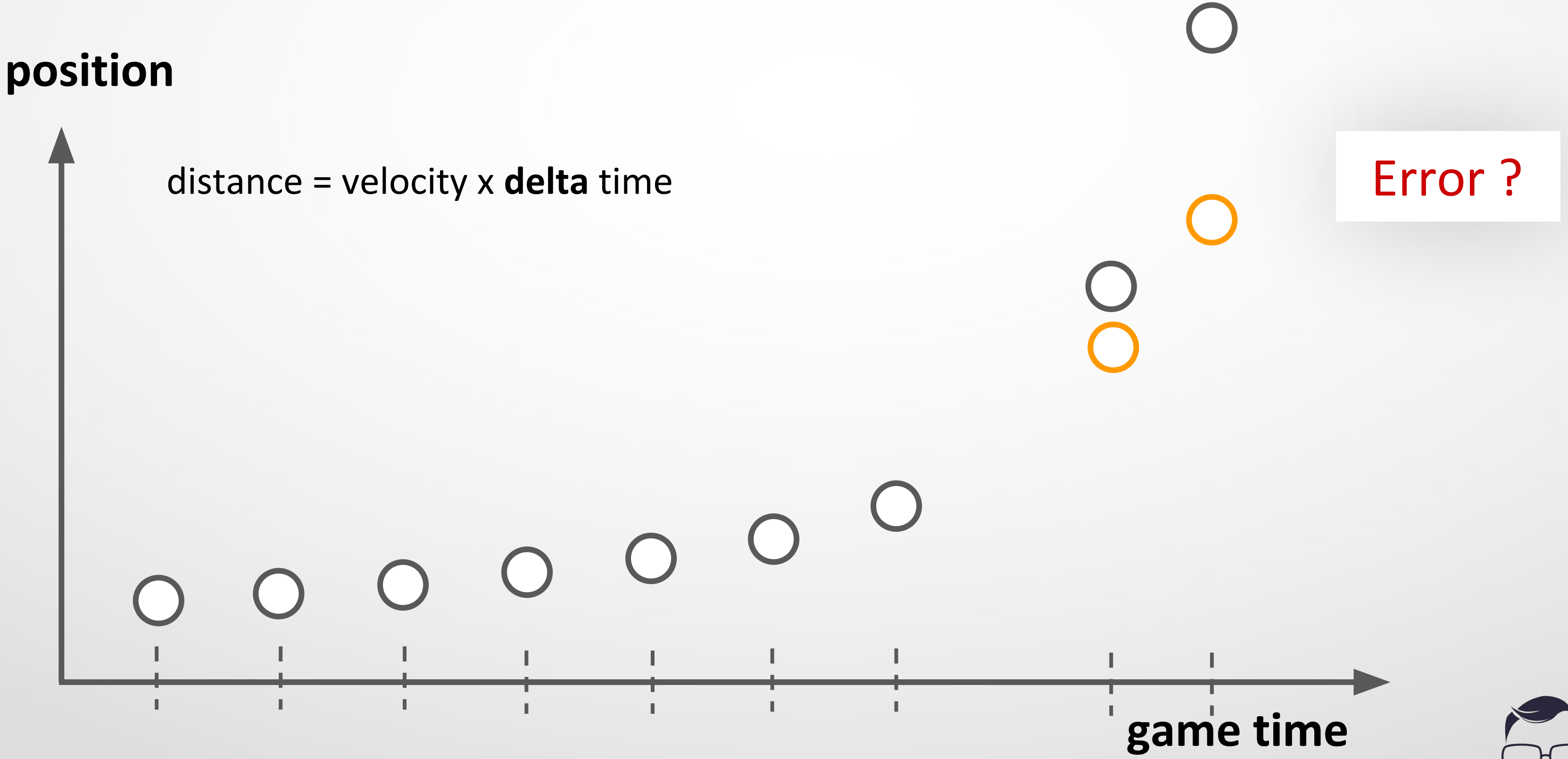
Object moving with uniform acceleration



Object moving with uniform acceleration



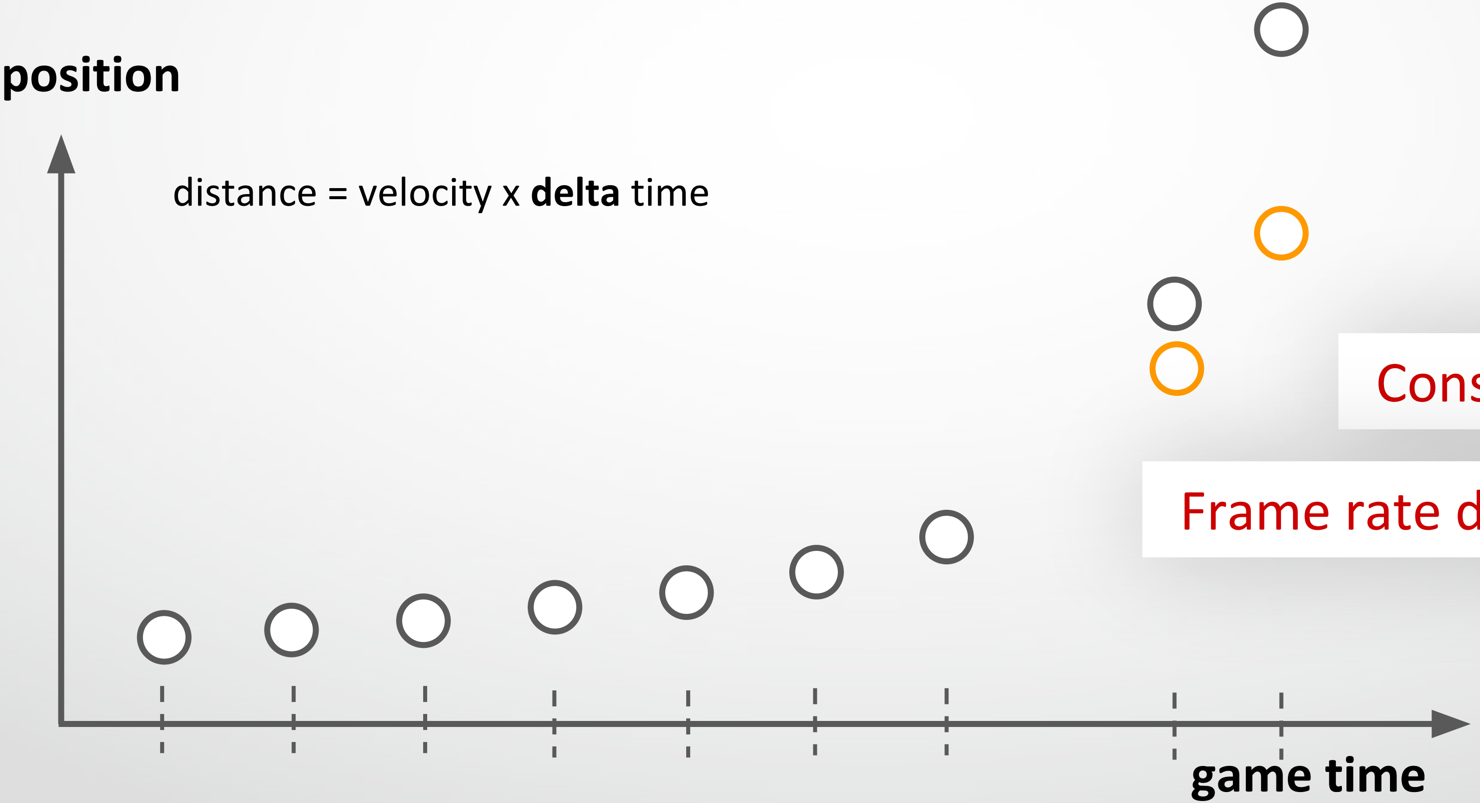
Object moving with uniform acceleration



distance = velocity x **delta** time

Error ?

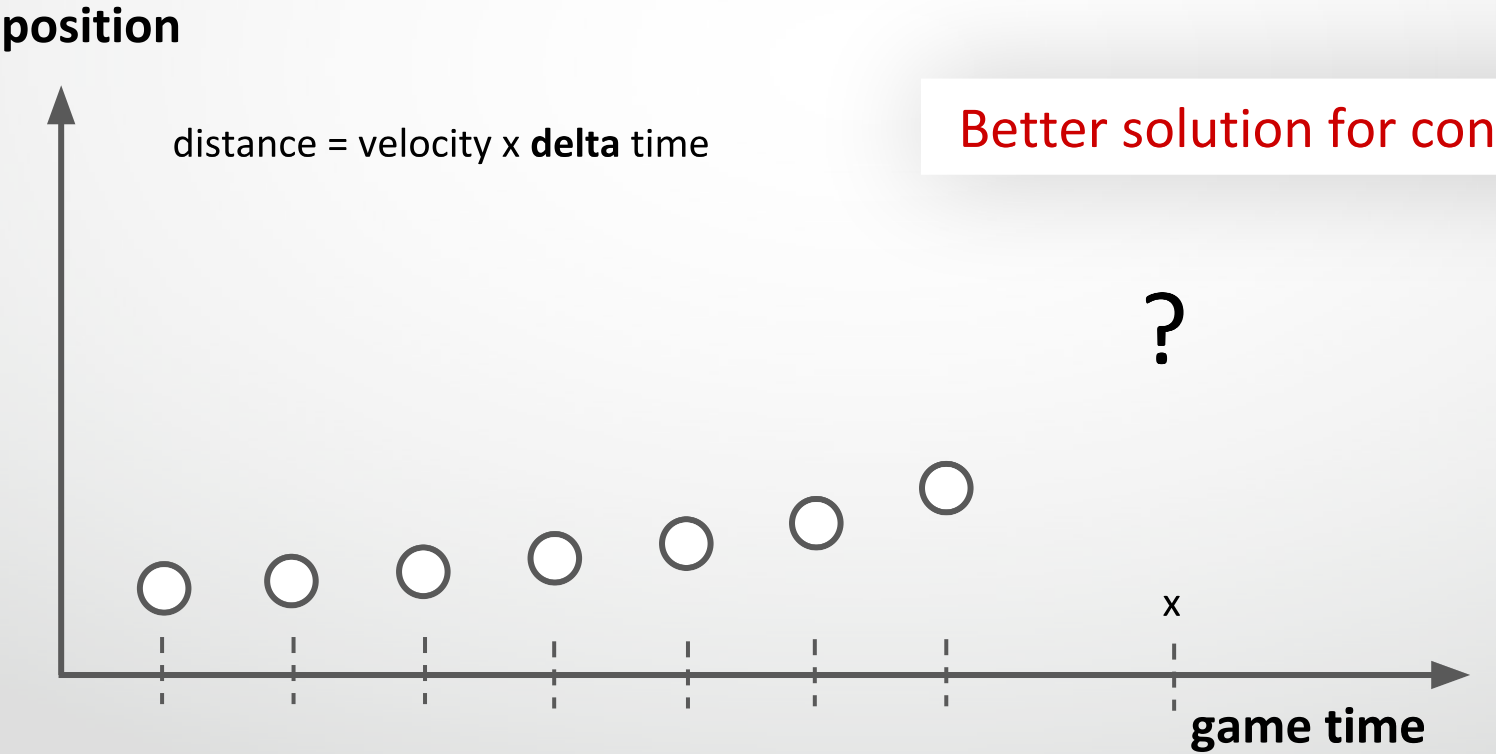
Object moving with uniform acceleration



Consistency ?

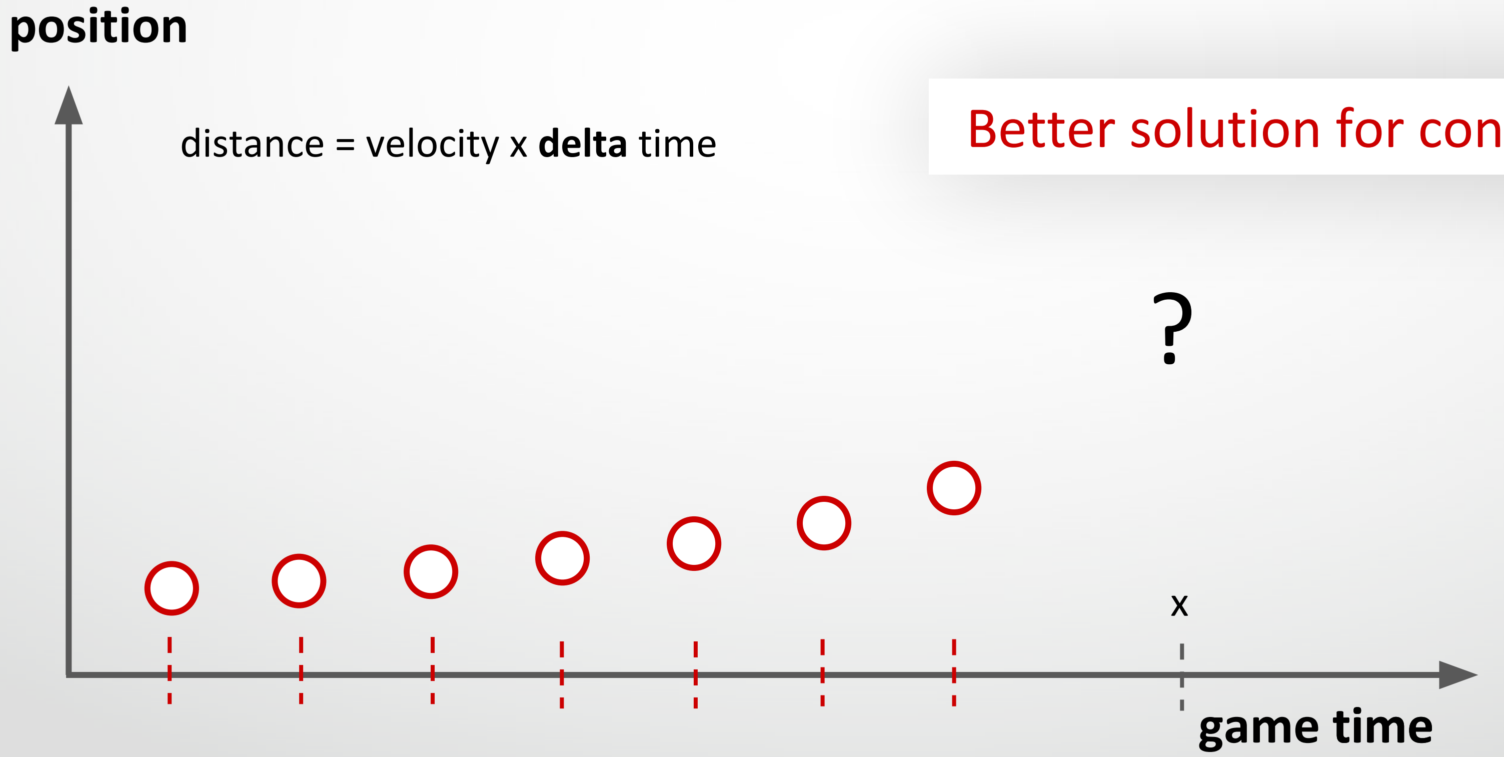
Frame rate dependent

Object moving with uniform acceleration



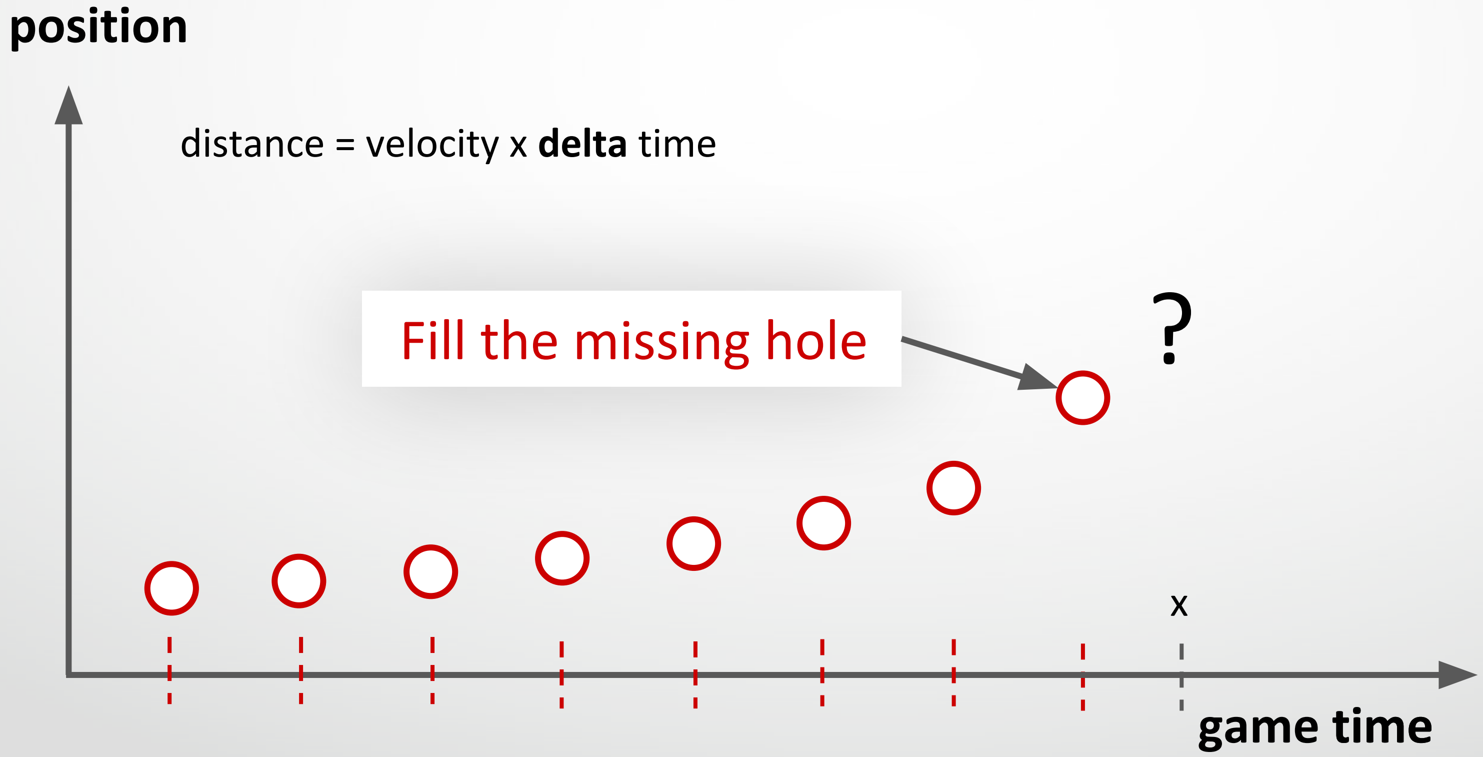
Better solution for consistency ?

Fixed update rate

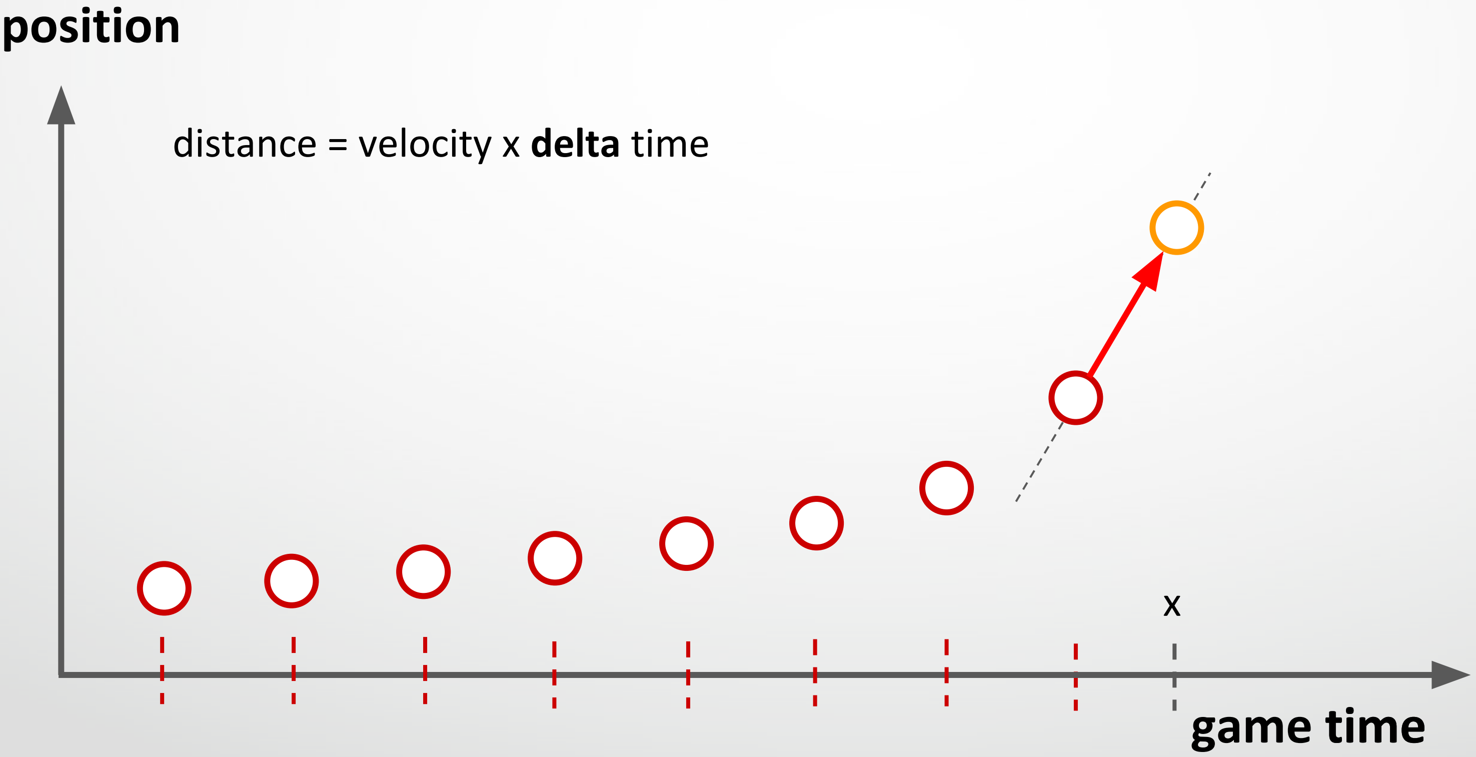


Better solution for consistency ?

Fixed update rate



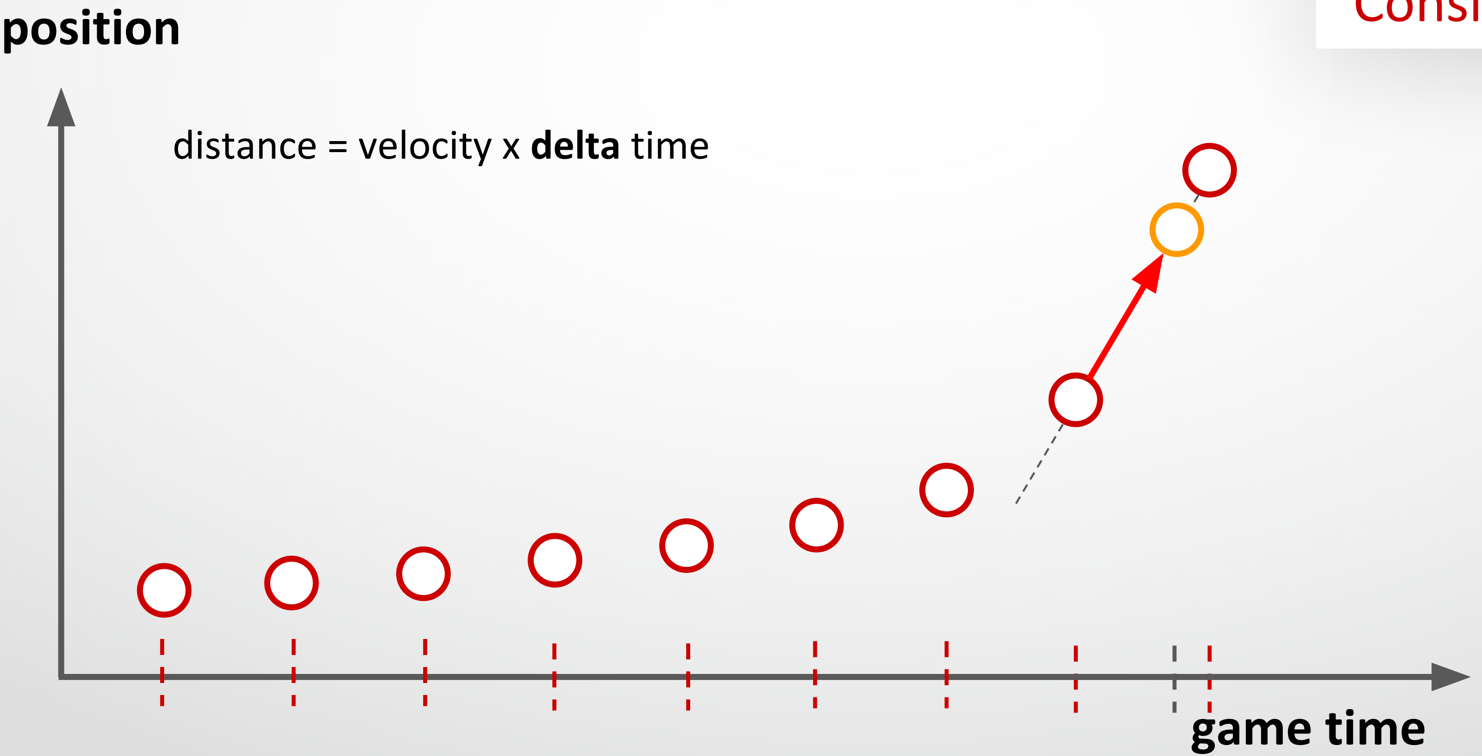
Fixed update rate



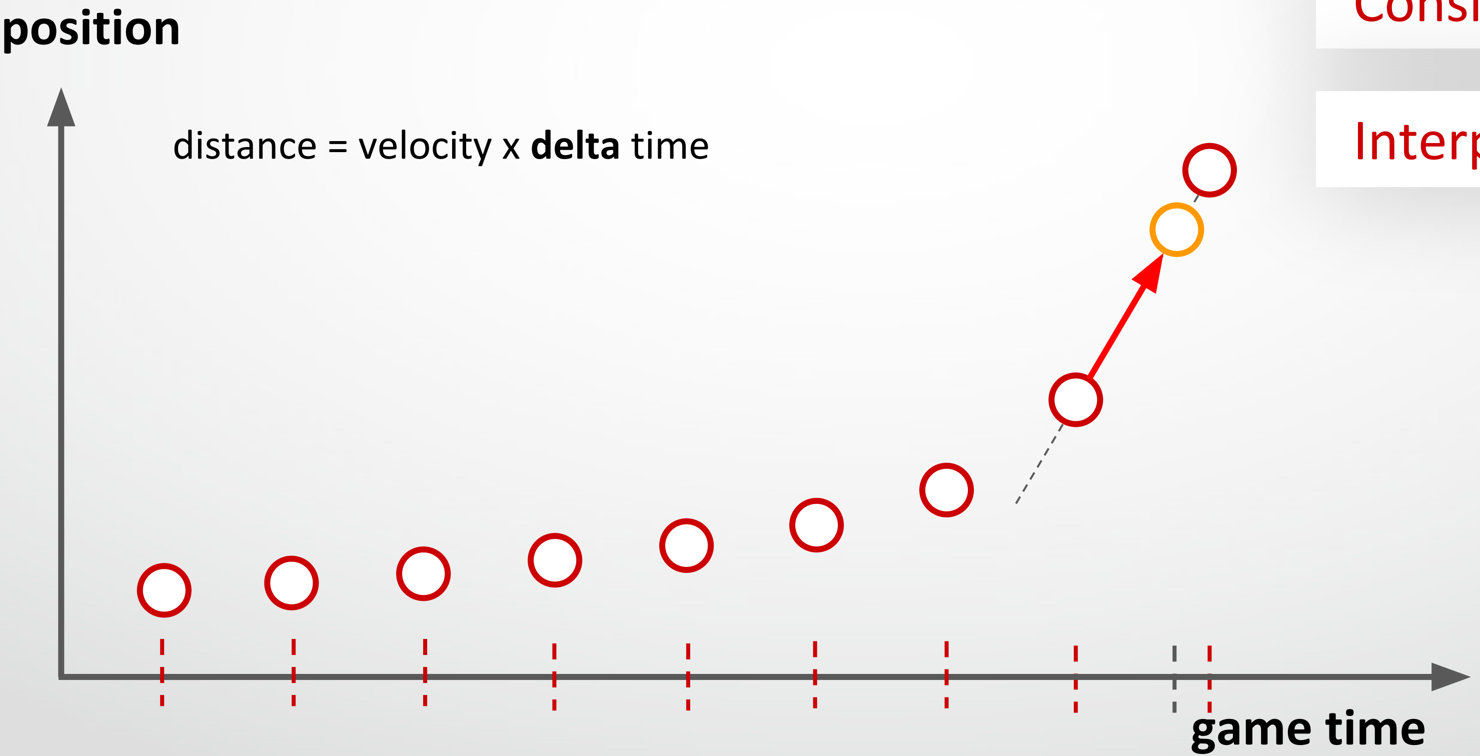
Fixed update rate

Consistent ?

distance = velocity x **delta** time



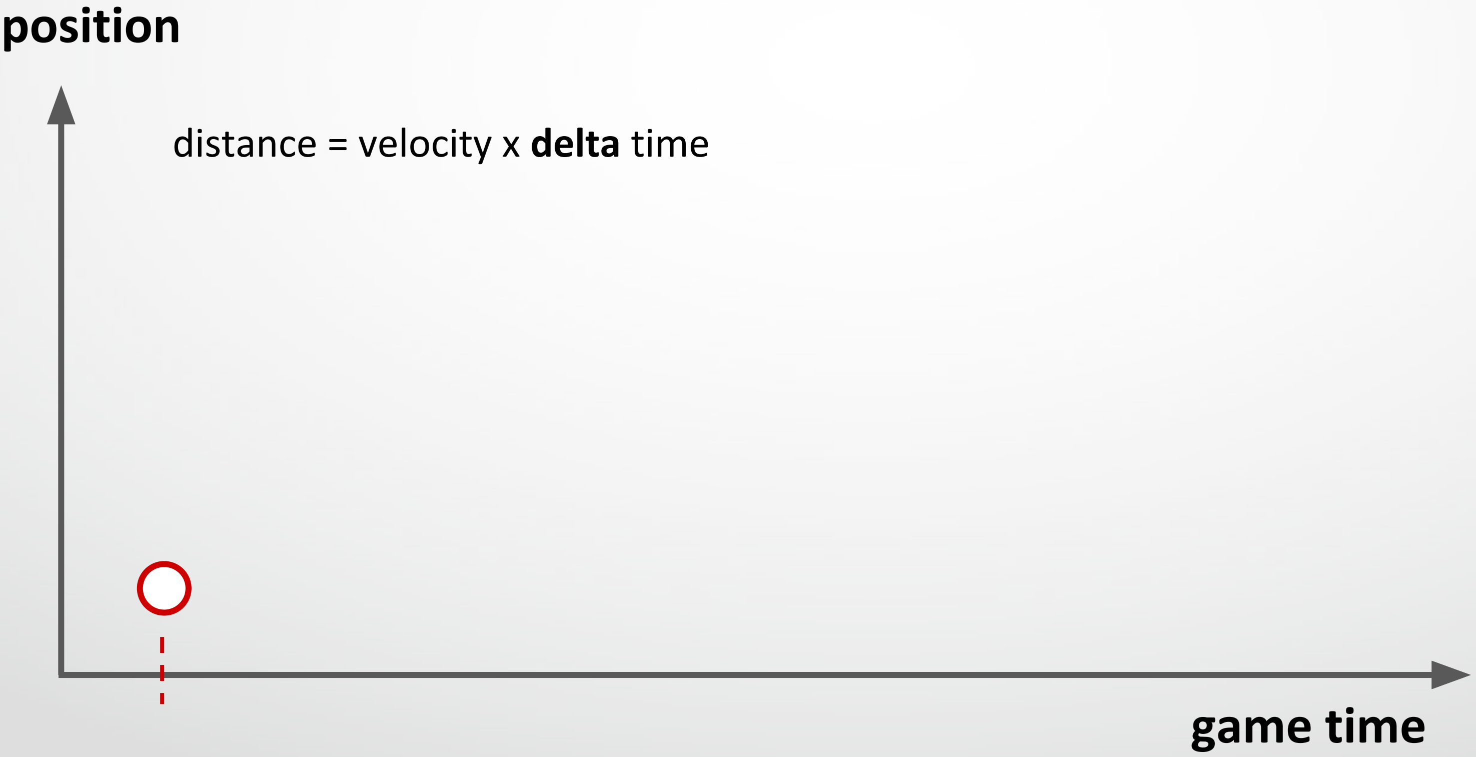
Fixed update rate



Consistent ?

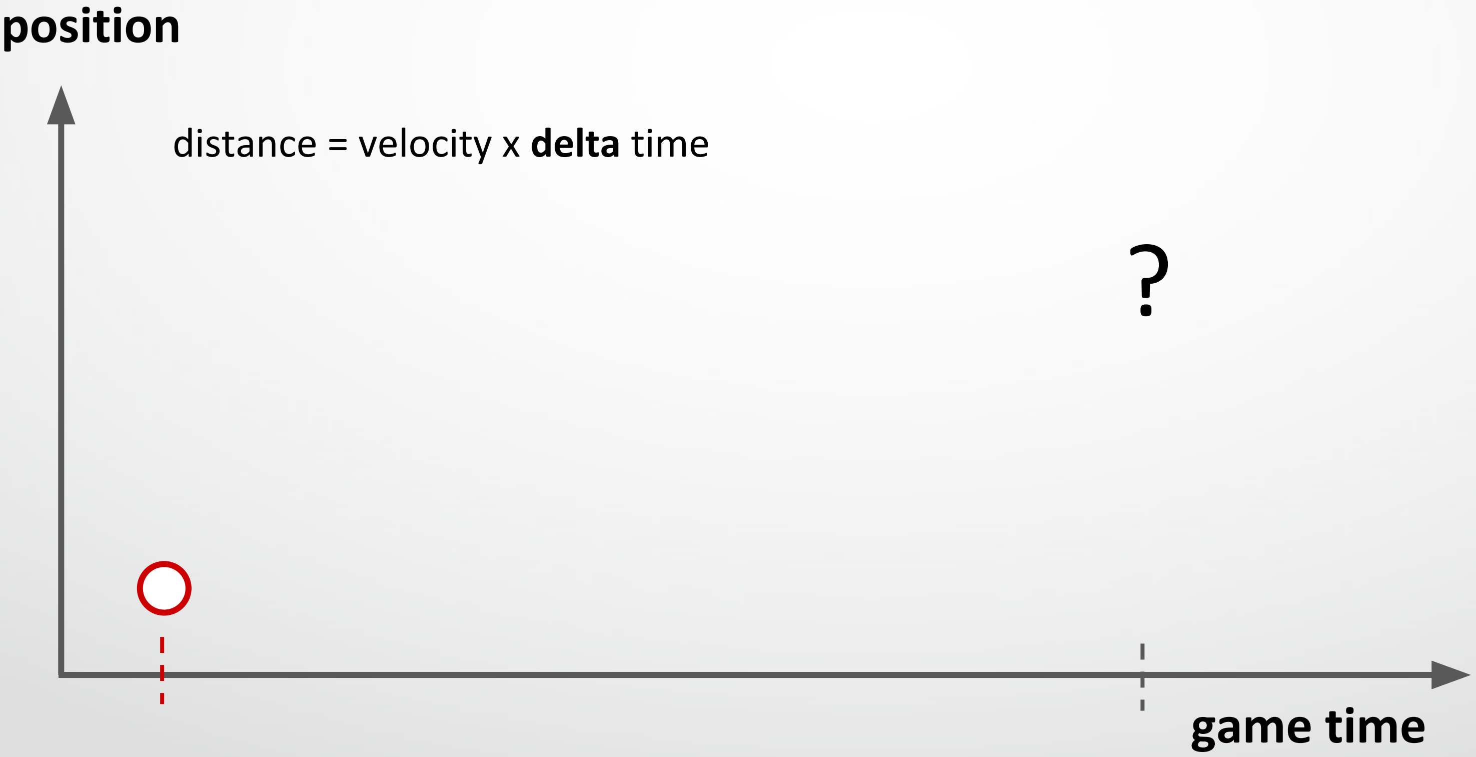
Interpolated

If the game is lagging ...

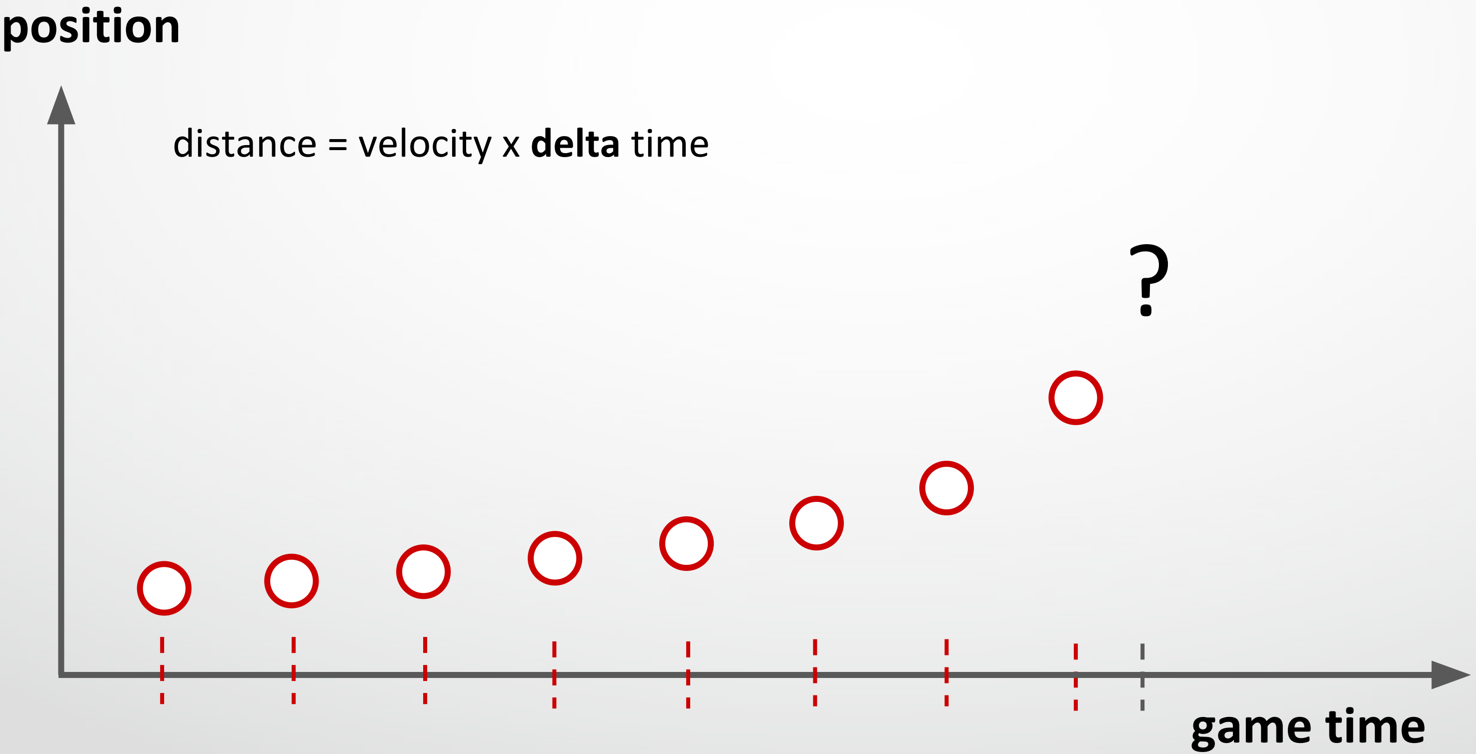


distance = velocity x **delta** time

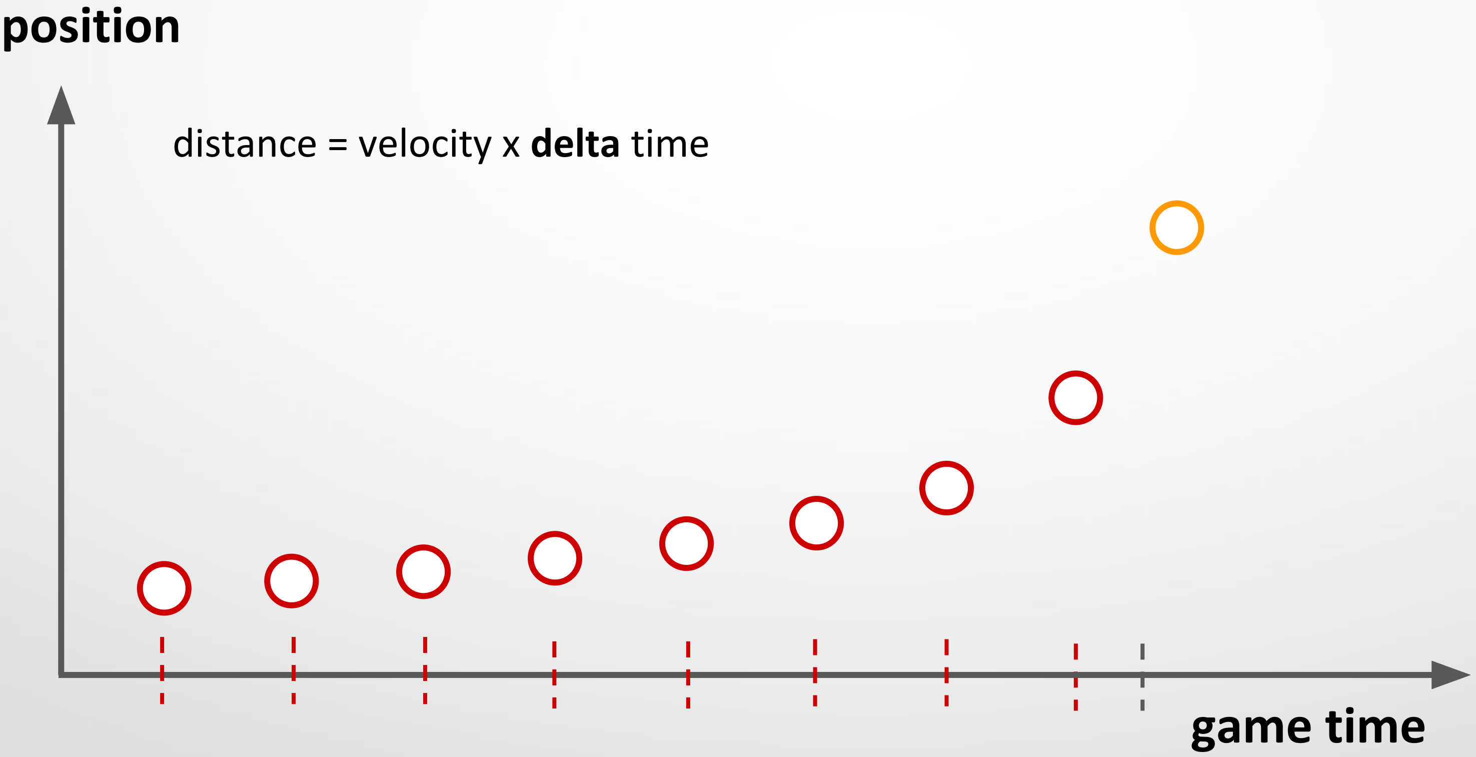
If the game is lagging ...



If the game is lagging ...



If the game is lagging ...

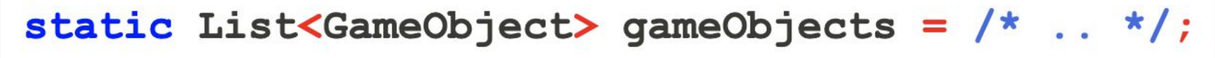


```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }

  static void UpdateScreen() { /* ... */ }
  static void WaitForTargetFPS() { /* ... */ }
}
```

```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      while (/* FixedUpdate */) {
        foreach (var go in gameObjects) {
            go.FixedUpdate();
        }
      }
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }

```

Inter-gameobject communication

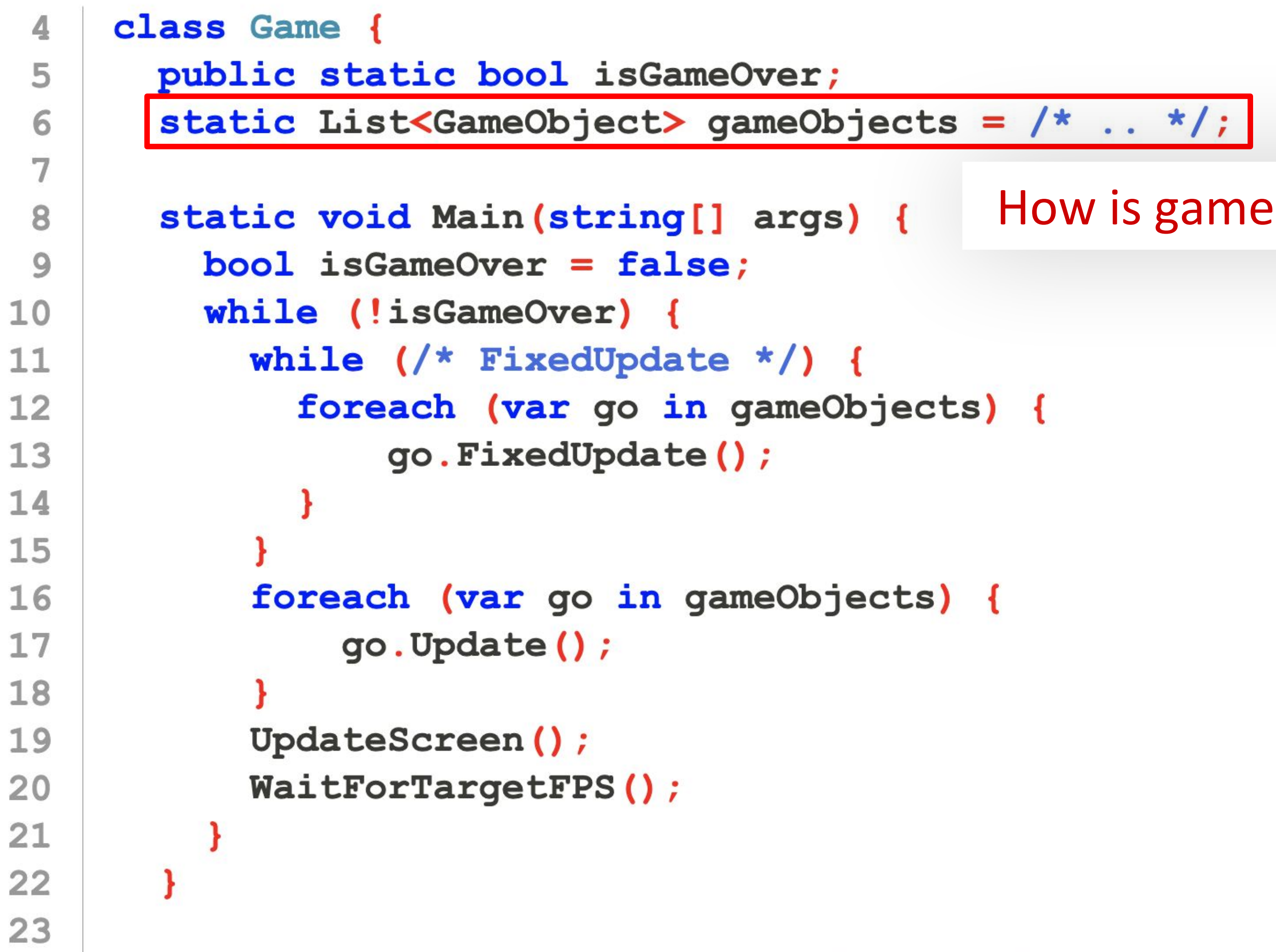
- Use singleton or service locator patterns
  - Global / static variables



```
static List<GameObject> gameObjects = /* .. */;
```

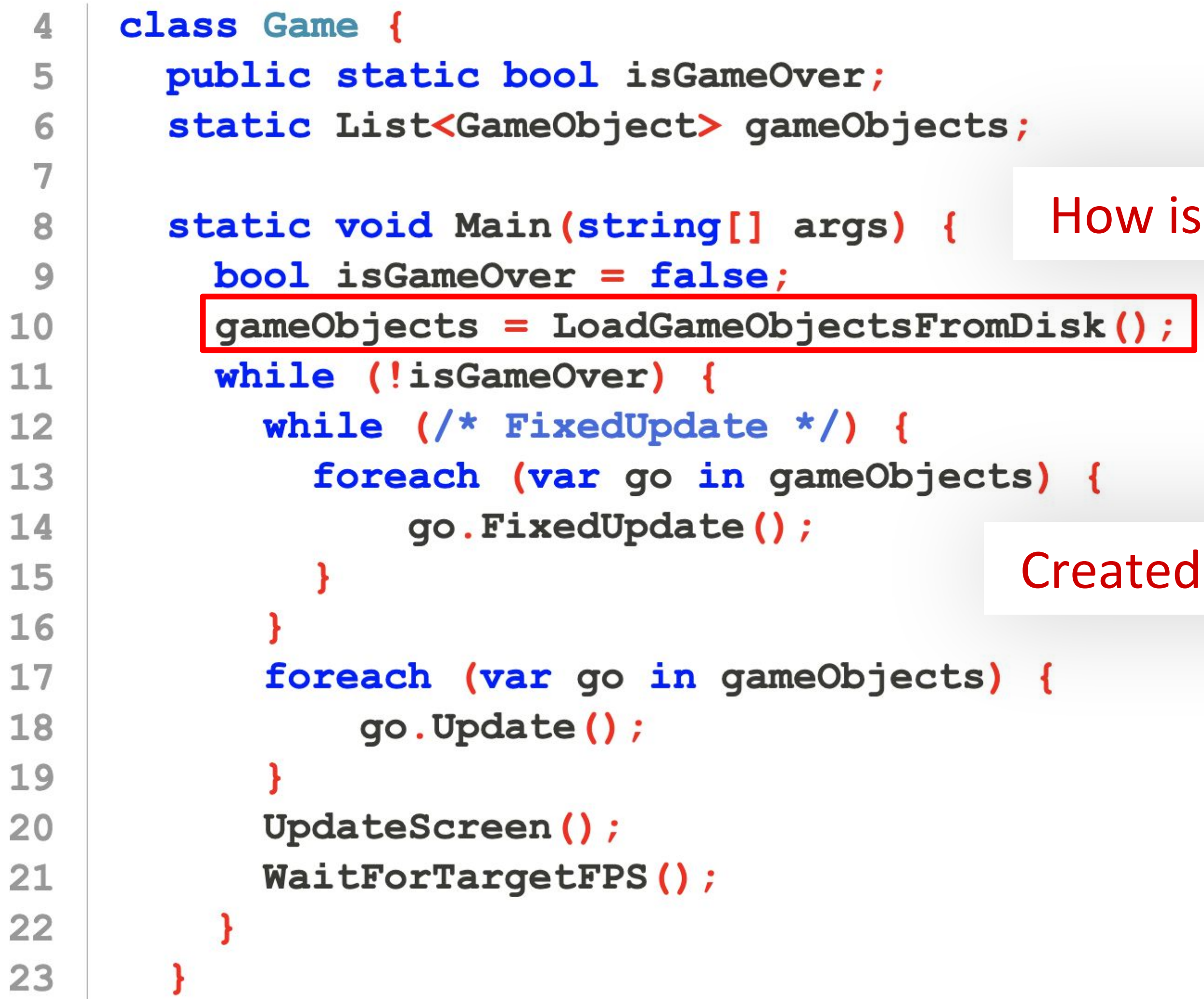
- Use observer pattern
  - Events
- Use dependency injection pattern

We will talk about this in "Game Control"



```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      while (/* FixedUpdate */) {
        foreach (var go in gameObjects) {
            go.FixedUpdate();
        }
      }
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }

```

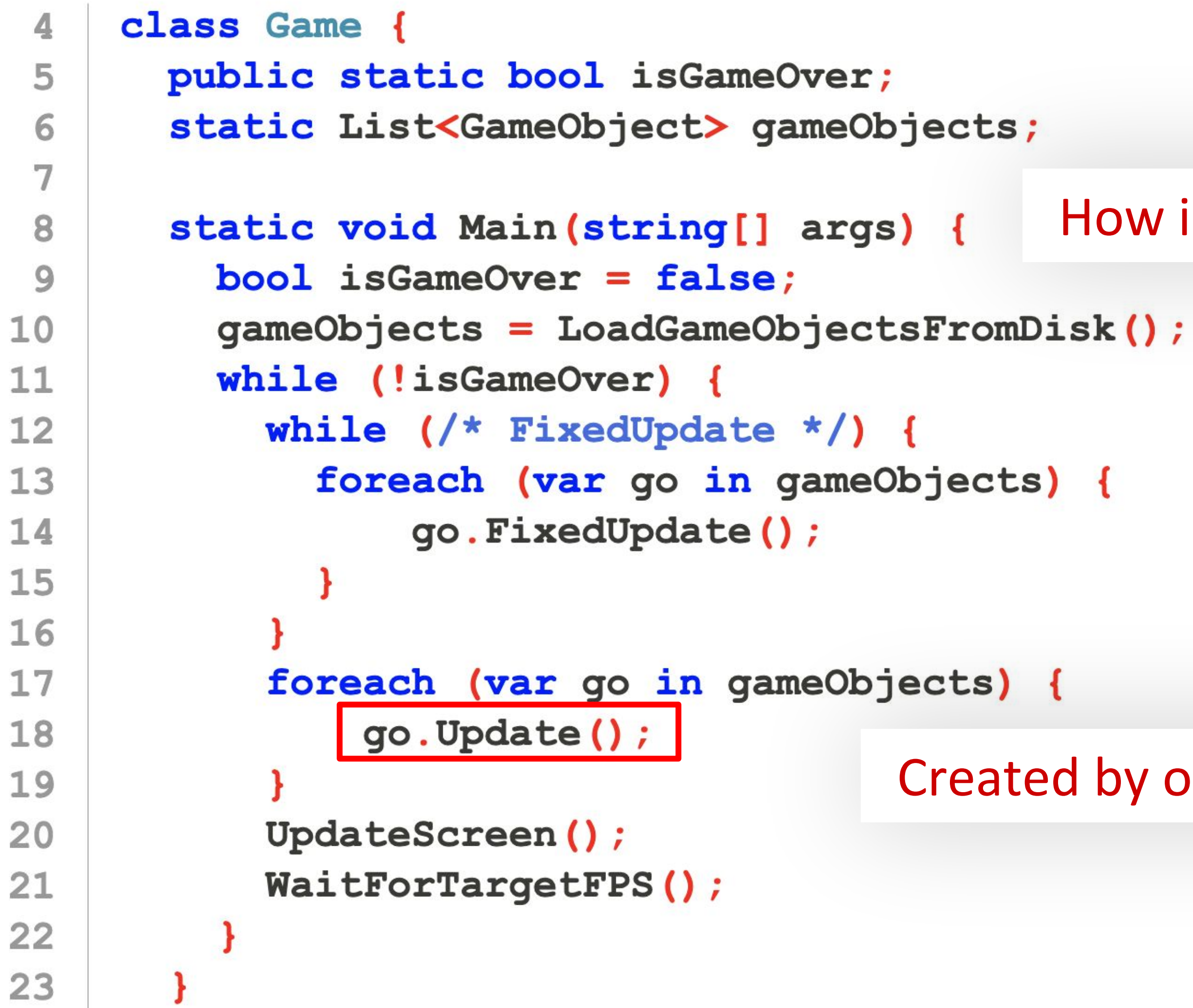
How is game object created ?



```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects;

  static void Main(string[] args) {
    bool isGameOver = false;
    gameObjects = LoadGameObjectsFromDisk();
    while (!isGameOver) {
      while (/* FixedUpdate */) {
        foreach (var go in gameObjects) {
            go.FixedUpdate();
        }
      }
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }
```

How is game object created ?

Created from serialized assets

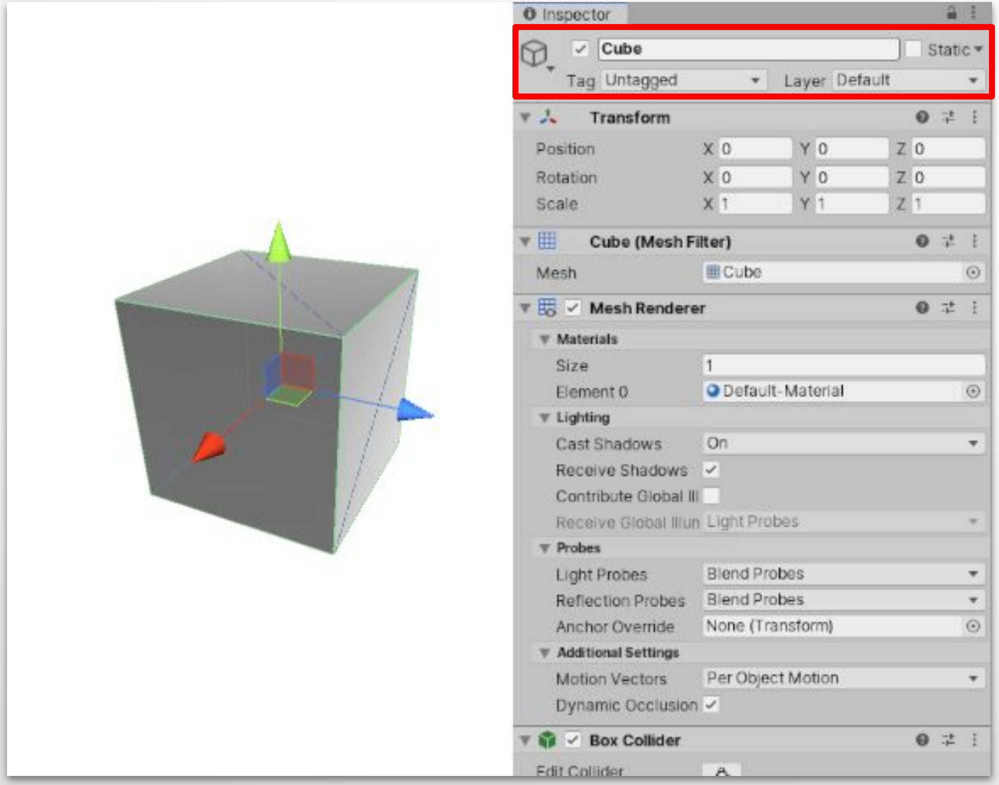


```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects;

  static void Main(string[] args) {
    bool isGameOver = false;
    gameObjects = LoadGameObjectsFromDisk();
    while (!isGameOver) {
      while (/* FixedUpdate */) {
        foreach (var go in gameObjects) {
            go.FixedUpdate();
        }
      }
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }
```

How is game object created ?

Created by other game object

GameObject



Inspector

☑ Cube □ Static

Tag Untagged Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Cube (Mesh Filter)

Mesh Cube

☑ Mesh Renderer

Materials

Size 1

Element 0 Default-Material

Lighting

Cast Shadows On

Receive Shadows ☑

Contribute Global Ill □

Receive Global Illum Light Probes

Probes

Light Probes Blend Probes

Reflection Probes Blend Probes

Anchor Override None (Transform)

Additional Settings

Motion Vectors Per Object Motion

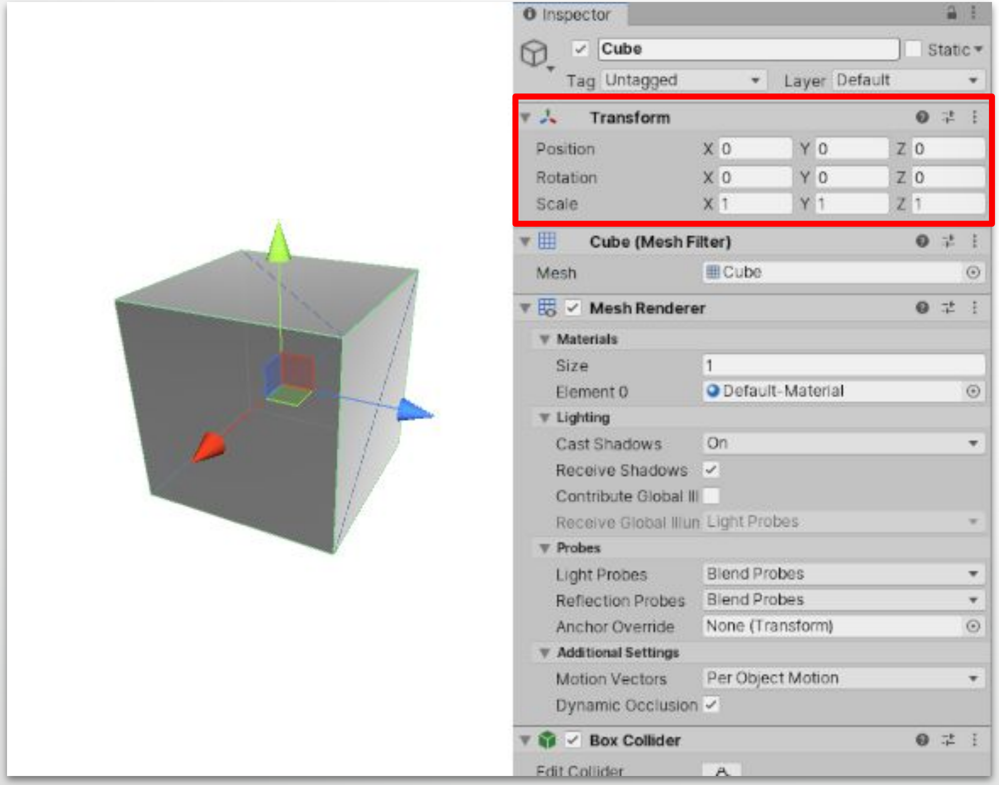
Dynamic Occlusion ☑

☑ Box Collider

Edit Collider

Properties: name, activeSelf

GameObject



Inspector

☑ Cube □ Static

Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

Cube (Mesh Filter)

Mesh Cube

☑ Mesh Renderer

Materials

Size 1

Element 0 Default-Material

Lighting

Cast Shadows On

Receive Shadows ☑

Contribute Global Ill □

Receive Global Illum Light Probes

Probes

Light Probes Blend Probes

Reflection Probes Blend Probes

Anchor Override None (Transform)

Additional Settings

Motion Vectors Per Object Motion

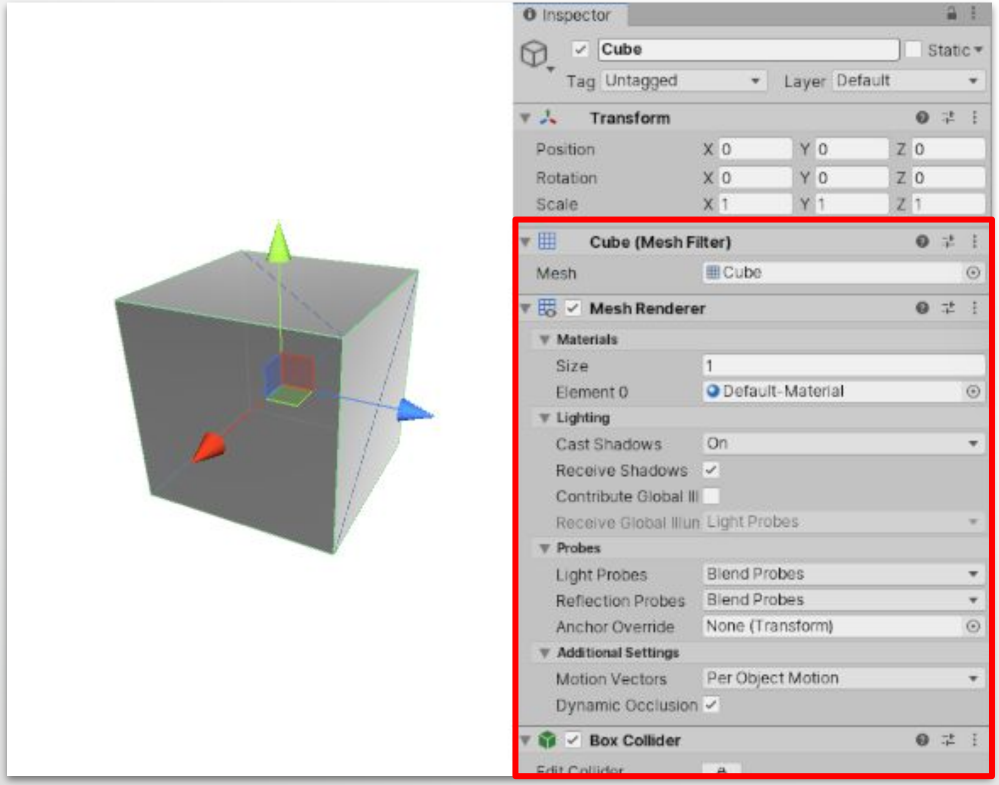
Dynamic Occlusion ☑

☑ Box Collider

Edit Collider

Transform component

GameObject



Inspector

☑ Cube □ Static

Tag Untagged Layer Default

Transform

Position X 0 Y 0 Z 0

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

Cube (Mesh Filter)

Mesh Cube

☑ Mesh Renderer

Materials

Size 1

Element 0 Default-Material

Lighting

Cast Shadows On

Receive Shadows ☑

Contribute Global Ill □

Receive Global Illun Light Probes

Probes

Light Probes Blend Probes

Reflection Probes Blend Probes

Anchor Override None (Transform)

Additional Settings

Motion Vectors Per Object Motion

Dynamic Occlusion ☑

☑ Box Collider

Edit Collider

Other components

Component pattern

GameObject

Component pattern

GameObject : A, B, C

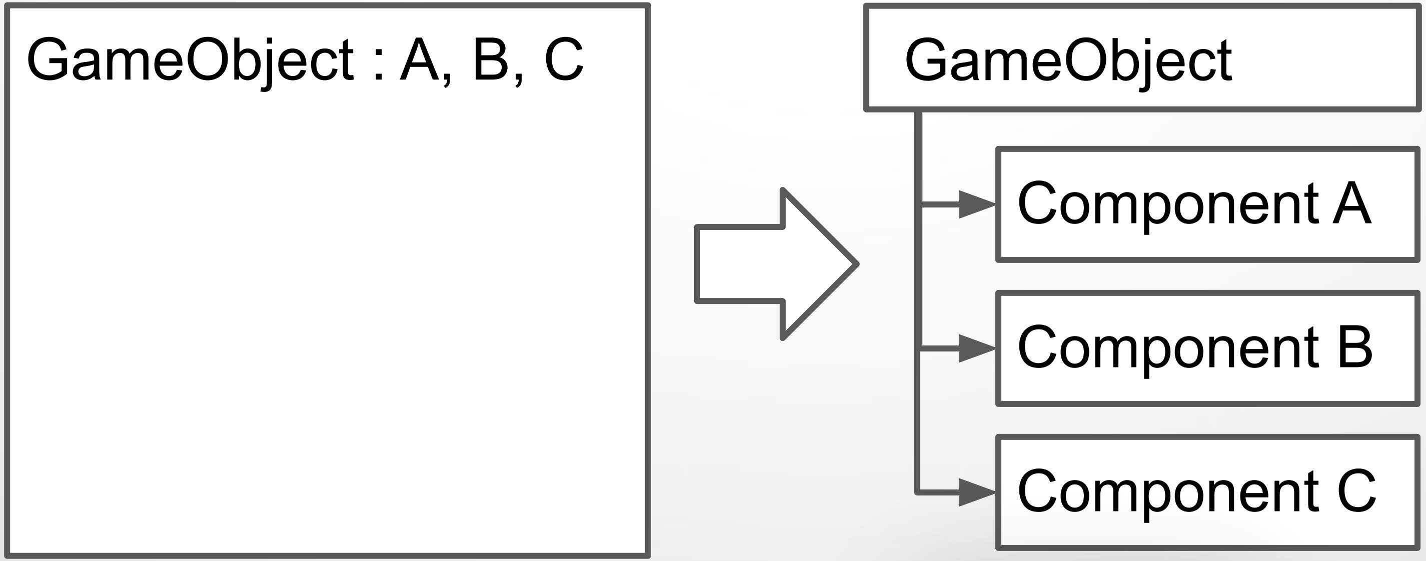
Inheritance

Component pattern

GameObject : A, B, C

Composition over inheritance

Component pattern

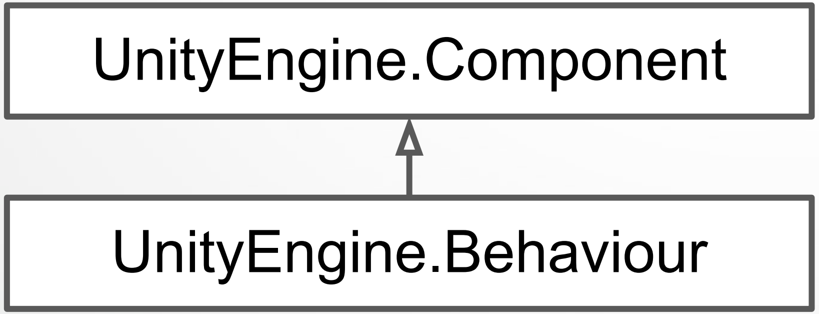


Decoupling

Component, Behaviour and MonoBehaviour

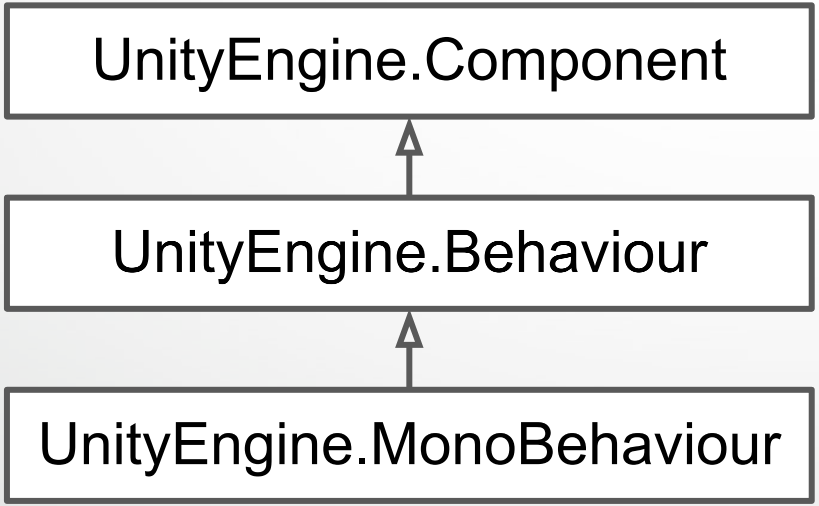
UnityEngine.Component

Component, Behaviour and MonoBehaviour



Diabled / Enabled

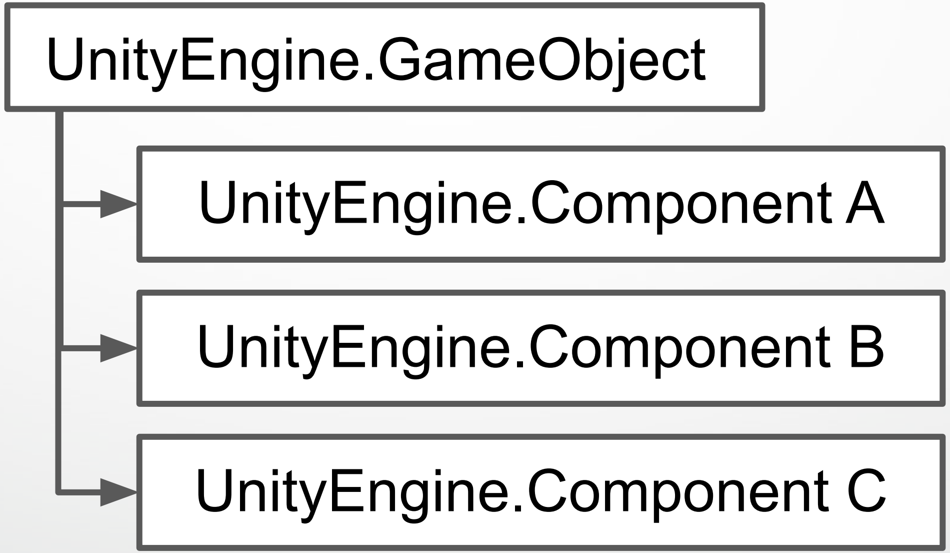
Component, Behaviour and MonoBehaviour



Diabled / Enabled

Custom script

GameObject composition



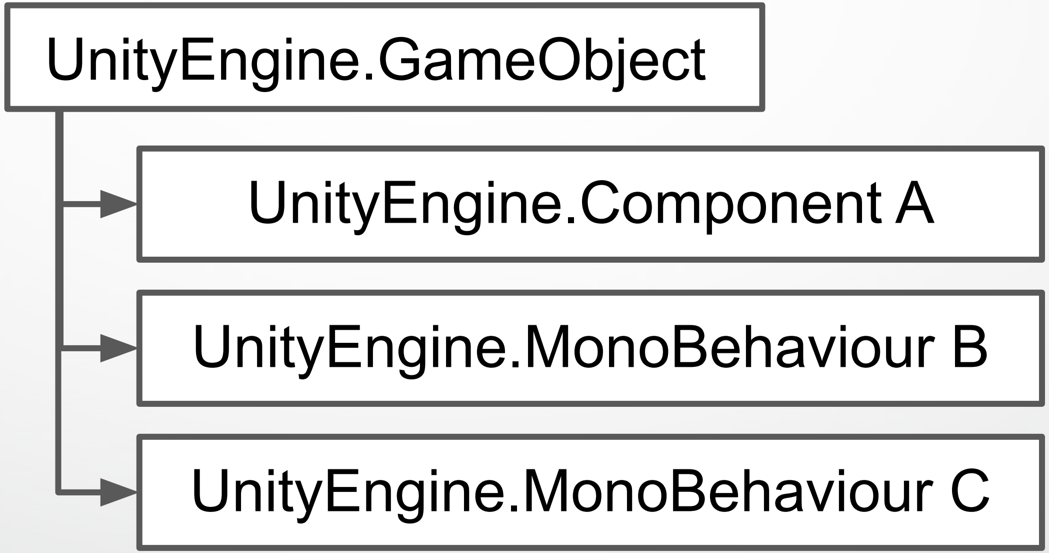
UnityEngine.GameObject

UnityEngine.Component A

UnityEngine.Component B

UnityEngine.Component C

GameObject composition


UnityEngine.GameObject

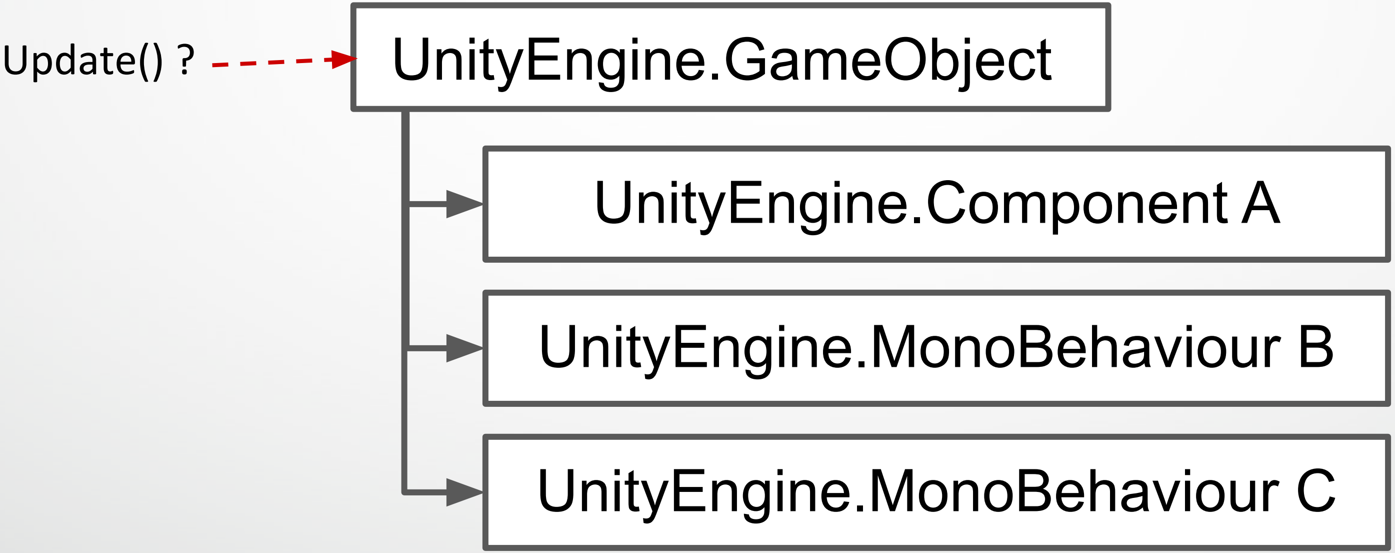
UnityEngine.Component A

UnityEngine.MonoBehaviour B

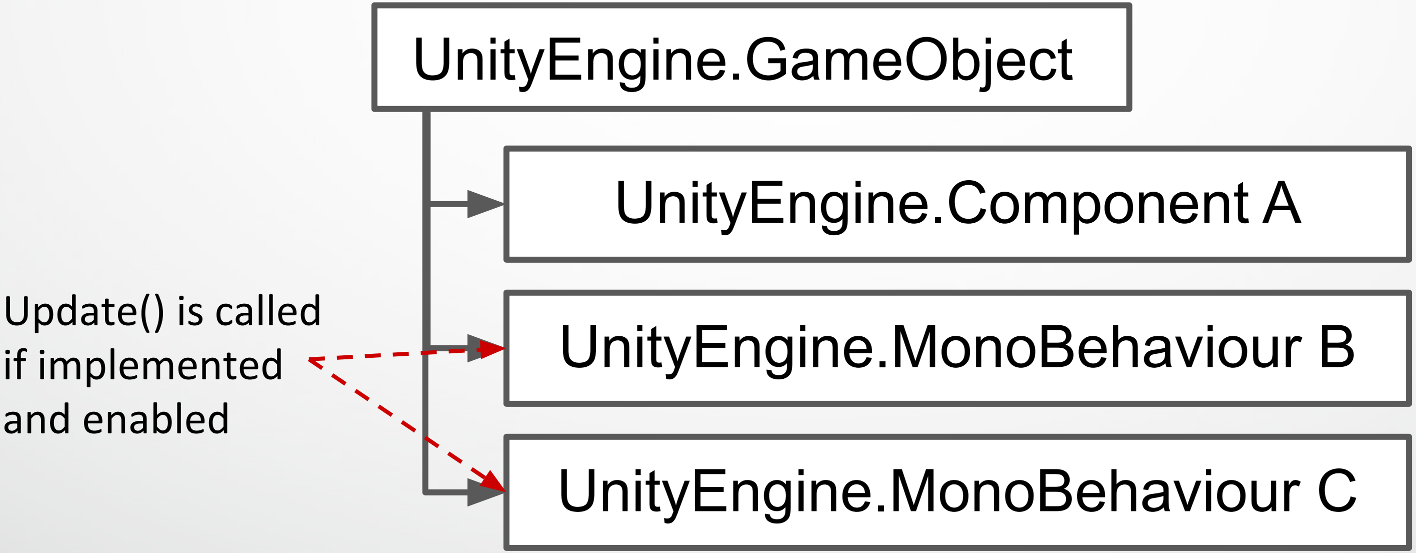
UnityEngine.MonoBehaviour C

FeisStu

Messages of MonoBehaviour

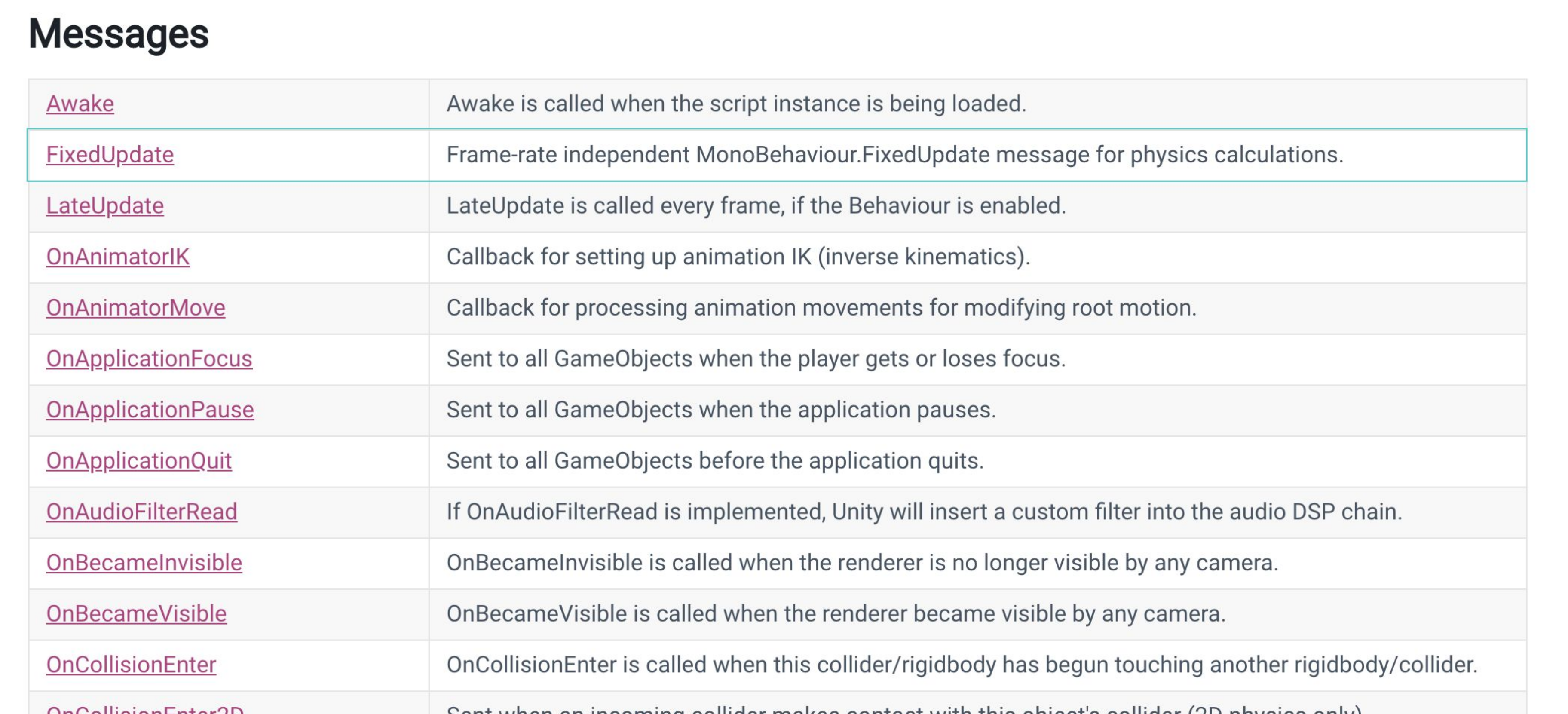


Messages of MonoBehaviour



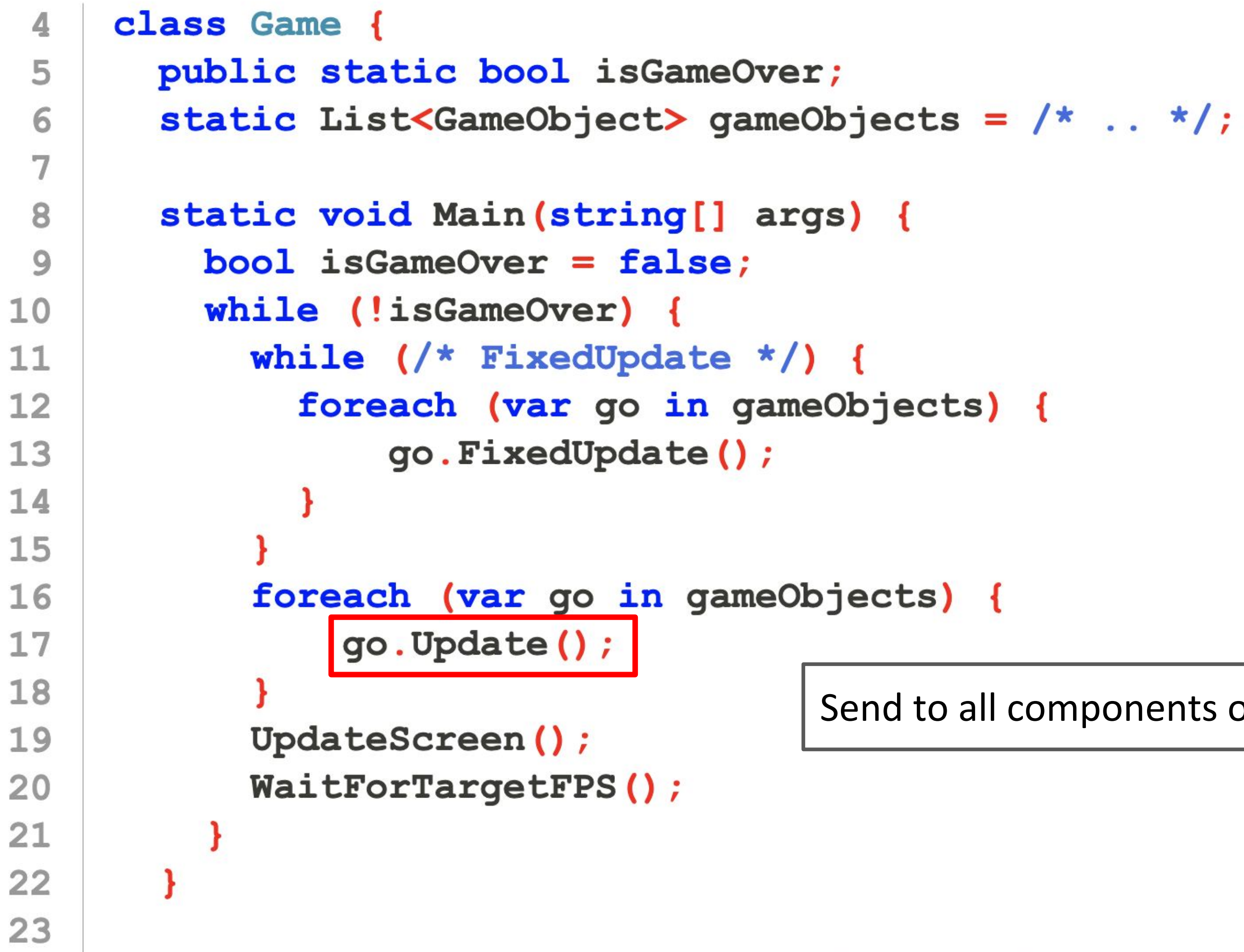
Update() is called
if implemented
and enabled

Messages of MonoBehaviour



Messages

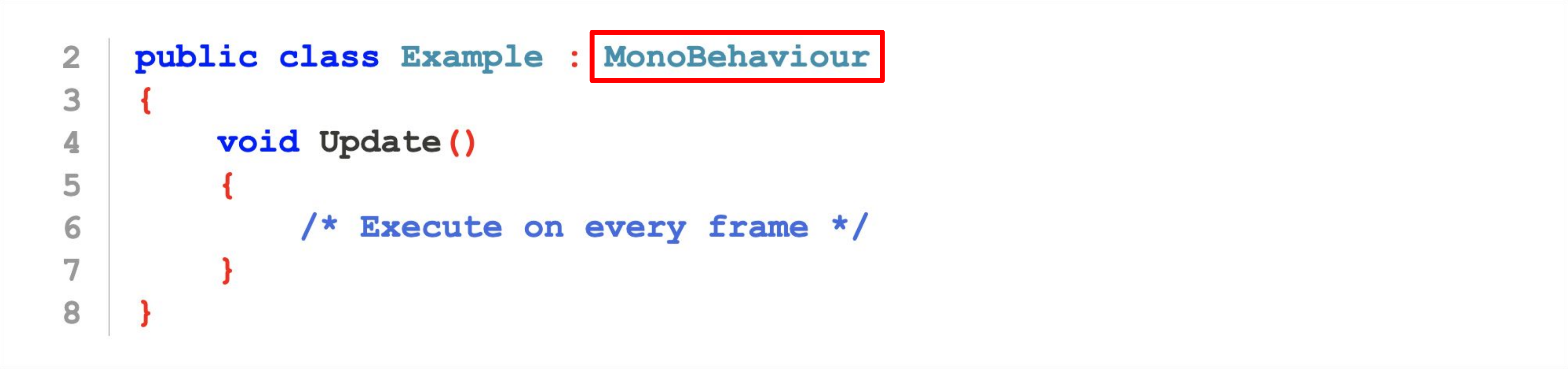
Awake	Awake is called when the script instance is being loaded.
FixedUpdate	Frame-rate independent MonoBehaviour.FixedUpdate message for physics calculations.
LateUpdate	LateUpdate is called every frame, if the Behaviour is enabled.
OnAnimatorIK	Callback for setting up animation IK (inverse kinematics).
OnAnimatorMove	Callback for processing animation movements for modifying root motion.
OnApplicationFocus	Sent to all GameObjects when the player gets or loses focus.
OnApplicationPause	Sent to all GameObjects when the application pauses.
OnApplicationQuit	Sent to all GameObjects before the application quits.
OnAudioFilterRead	If OnAudioFilterRead is implemented, Unity will insert a custom filter into the audio DSP chain.
OnBecameInvisible	OnBecameInvisible is called when the renderer is no longer visible by any camera.
OnBecameVisible	OnBecameVisible is called when the renderer became visible by any camera.
OnCollisionEnter	OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.
OnCollisionEnter2D	Sent when an incoming collider makes contact with this object's collider (2D physics only).



```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects = /* .. */;

  static void Main(string[] args) {
    bool isGameOver = false;
    while (!isGameOver) {
      while (/* FixedUpdate */) {
        foreach (var go in gameObjects) {
            go.FixedUpdate();
        }
      }
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }

```

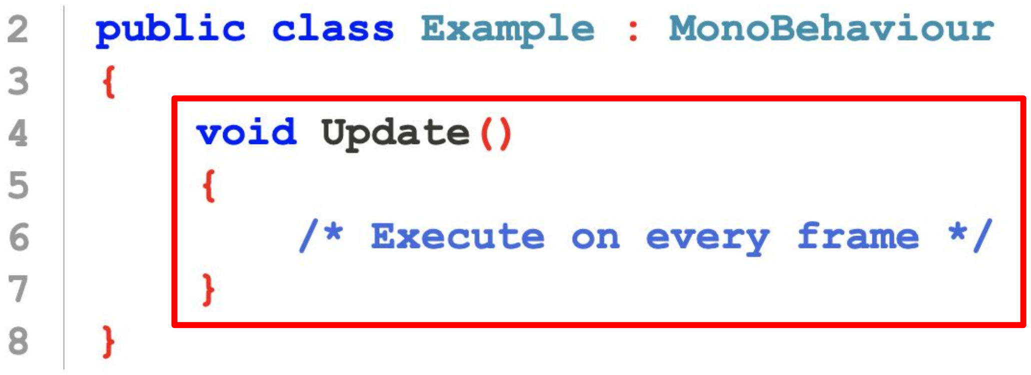
Send to all components of the game object

Custom script and MonoBehaviour



```
public class Example : MonoBehaviour
{
    void Update()
    {
        /* Execute on every frame */
    }
}
```

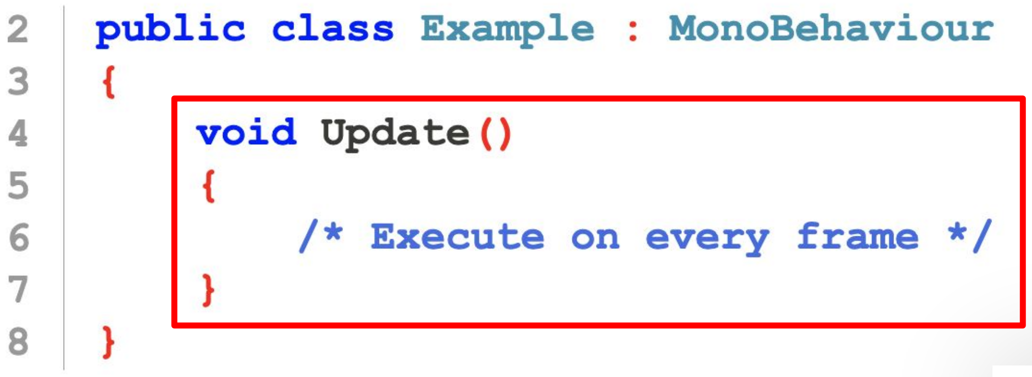
Custom script and MonoBehaviour



```
public class Example : MonoBehaviour
{
    void Update()
    {
        /* Execute on every frame */
    }
}
```

Public ?

Custom script and MonoBehaviour



```
public class Example : MonoBehaviour
{
    void Update()
    {
        /* Execute on every frame */
    }
}
```

Public ?

Message → Call by name

Messages of MonoBehaviour

- Awake(), OnEnable(), Start()
- FixedUpdate()
- Update(), LateUpate()
- OnRenderImage()
- OnGUI()
- OnDisable()
- OnDestroy()

Messages of MonoBehaviour

- **Awake**(), OnEnable(), **Start**()
- FixedUpdate()
- Update(), LateUpate()
- OnRenderImage()
- OnGUI()
- OnDisable()
- **OnDestroy**()

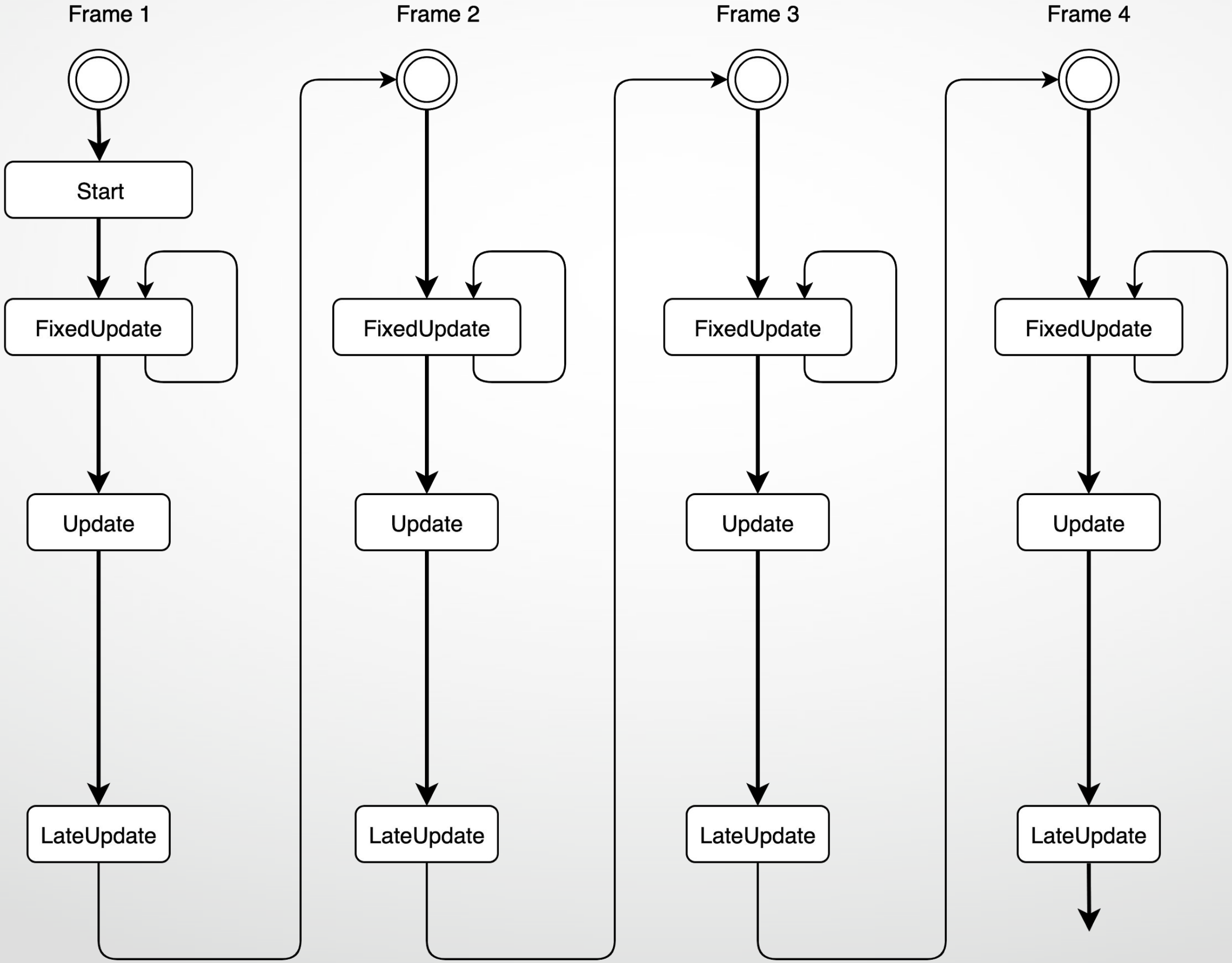
Happens at most once in its life

Messages of MonoBehaviour

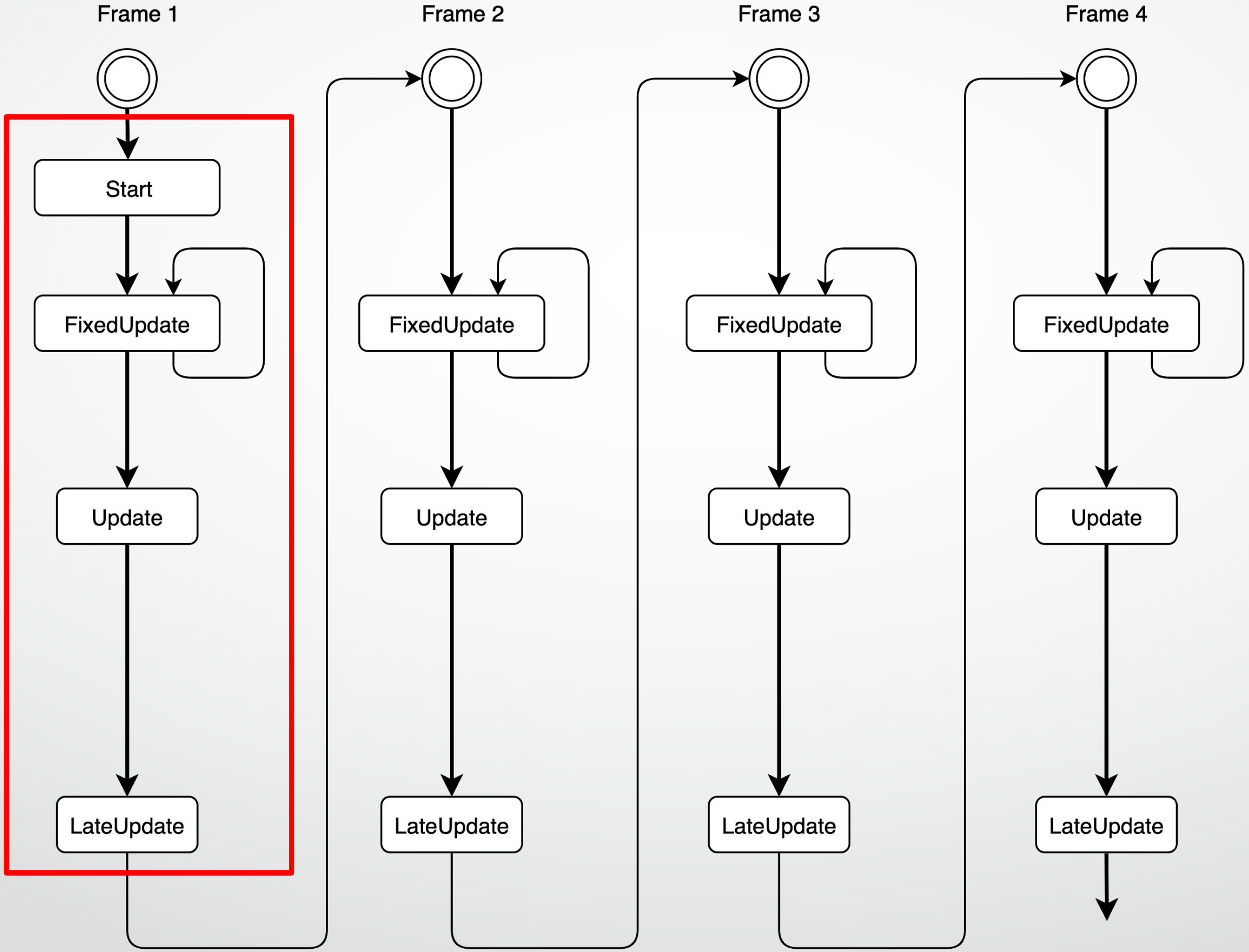
- Awake(), OnEnable(), Start()
- **FixedUpdate**()
- Update(), LateUpate()
- OnRenderImage()
- **OnGUI**()
- OnDisable()
- OnDestroy()

May happen more than once per frame

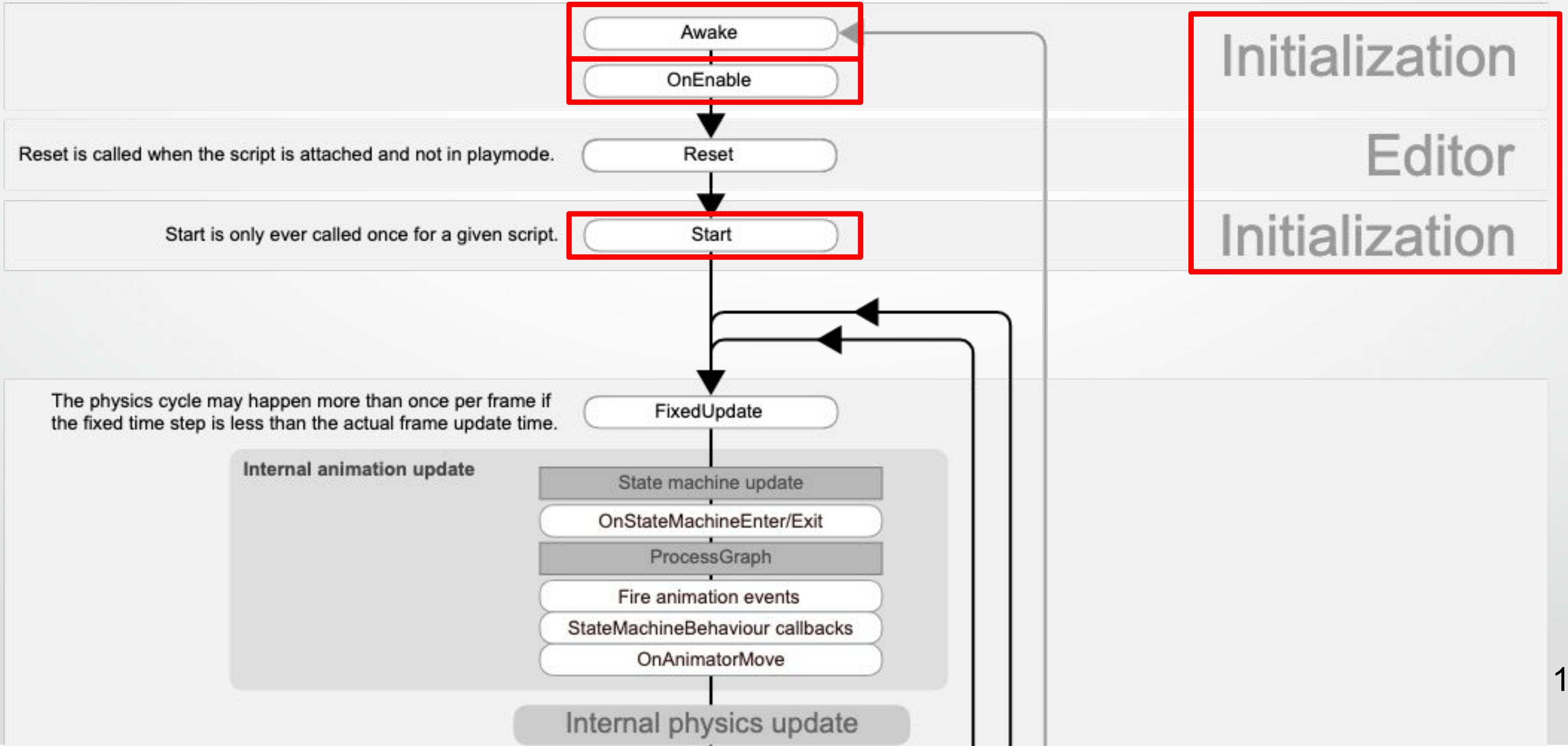
FeisStu


Frame 1
Frame 2
Frame 3
Frame 4
Start
FixedUpdate
Update
LateUpdate
FixedUpdate
Update
LateUpdate
FixedUpdate
Update
LateUpdate
FixedUpdate
Update
LateUpdate

FeisStu


Frame 1
Frame 2
Frame 3
Frame 4
Start
FixedUpdate
Update
LateUpdate
FixedUpdate
Update
LateUpdate
FixedUpdate
Update
LateUpdate
FixedUpdate
Update
LateUpdate

Order of Execution for Event Functions

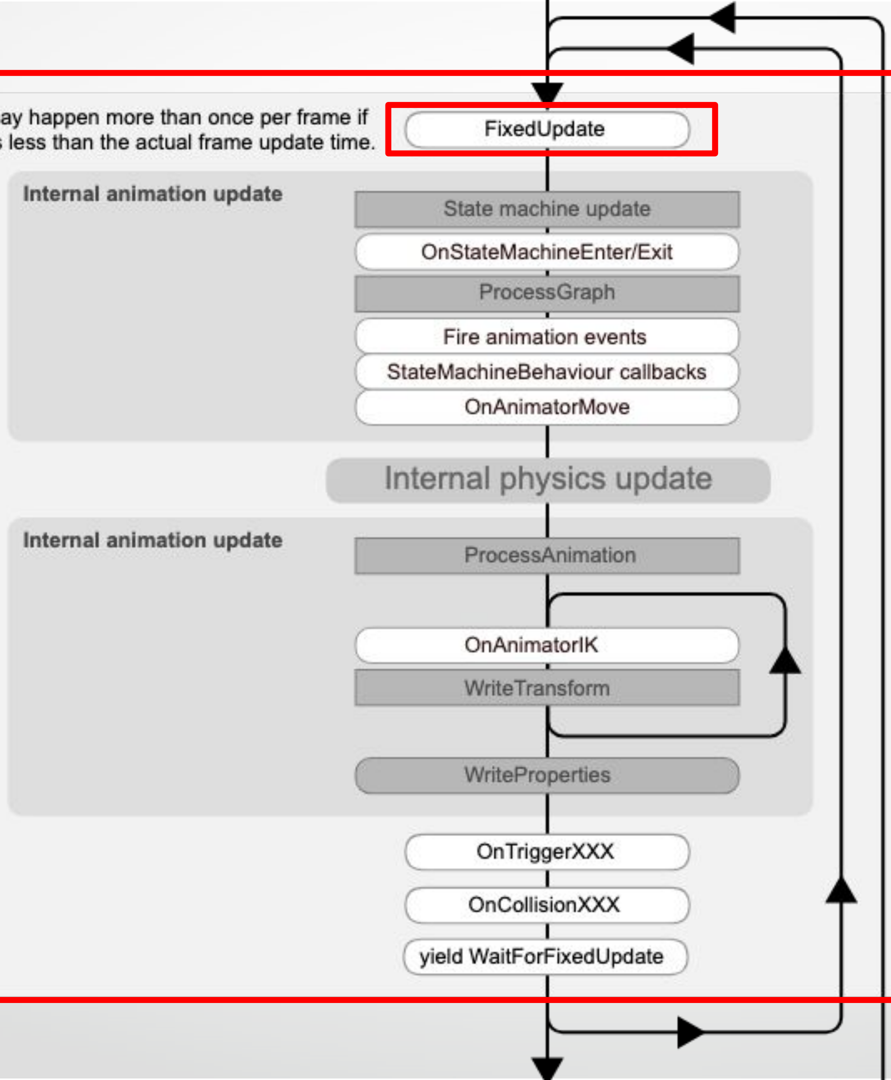


Reset is called when the script is attached and not in playmode.

Start is only ever called once for a given script.

The physics cycle may happen more than once per frame if the fixed time step is less than the actual frame update time.

Physics



The physics cycle may happen more than once per frame if the fixed time step is less than the actual frame update time.

FixedUpdate

Internal animation update

State machine update

OnStateMachineEnter/Exit

ProcessGraph

Fire animation events

StateMachineBehaviour callbacks

OnAnimatorMove

Internal physics update

Internal animation update

ProcessAnimation

OnAnimatorIK

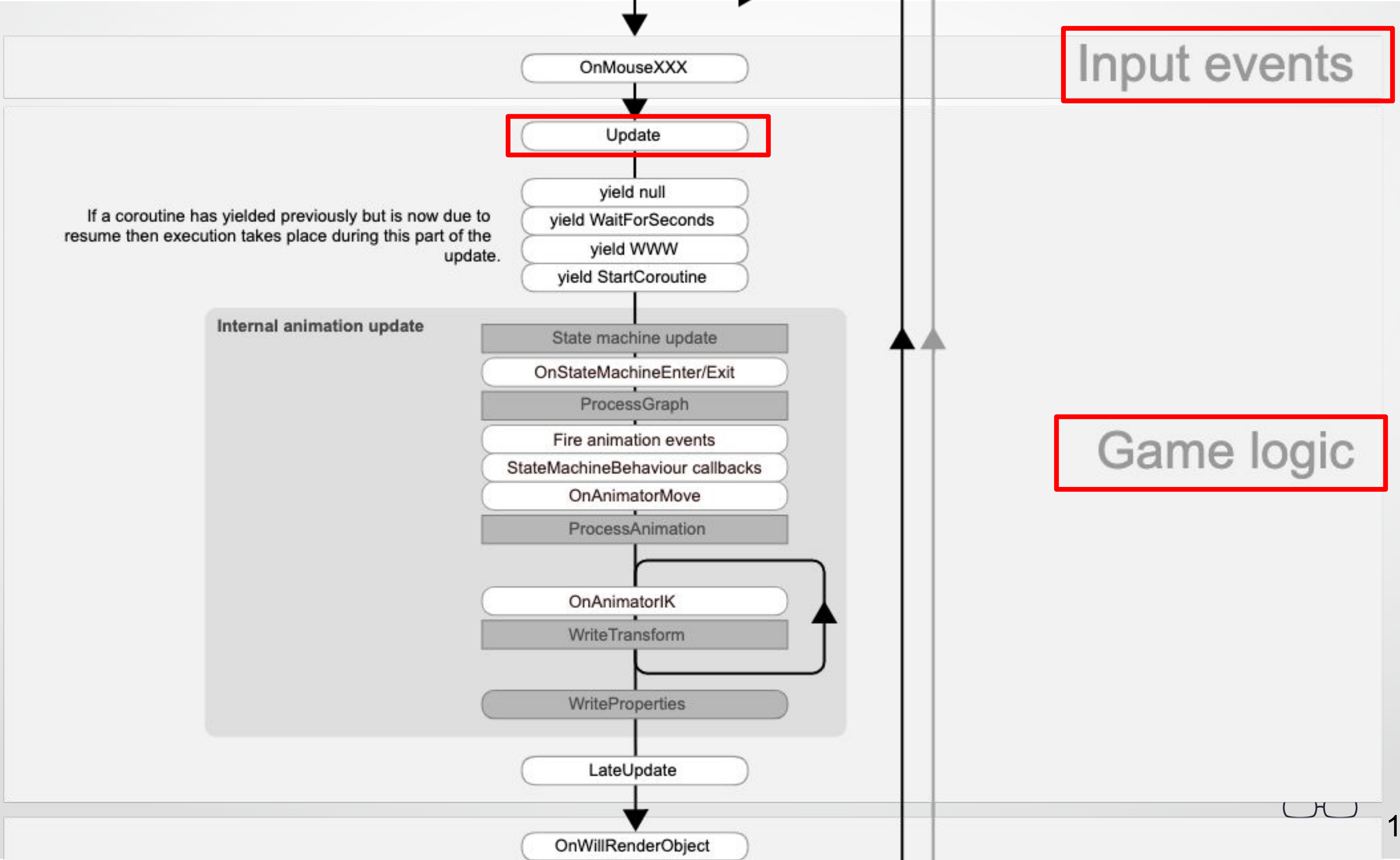
WriteTransform

WriteProperties

OnTriggerXXX

OnCollisionXXX

yield WaitForFixedUpdate



Input events

OnMouseXXX

Game logic

Update

If a coroutine has yielded previously but is now due to resume then execution takes place during this part of the update.

yield null

yield WaitForSeconds

yield WWW

yield StartCoroutine

Internal animation update

State machine update

OnStateMachineEnter/Exit

ProcessGraph

Fire animation events

StateMachineBehaviour callbacks

OnAnimatorMove

ProcessAnimation

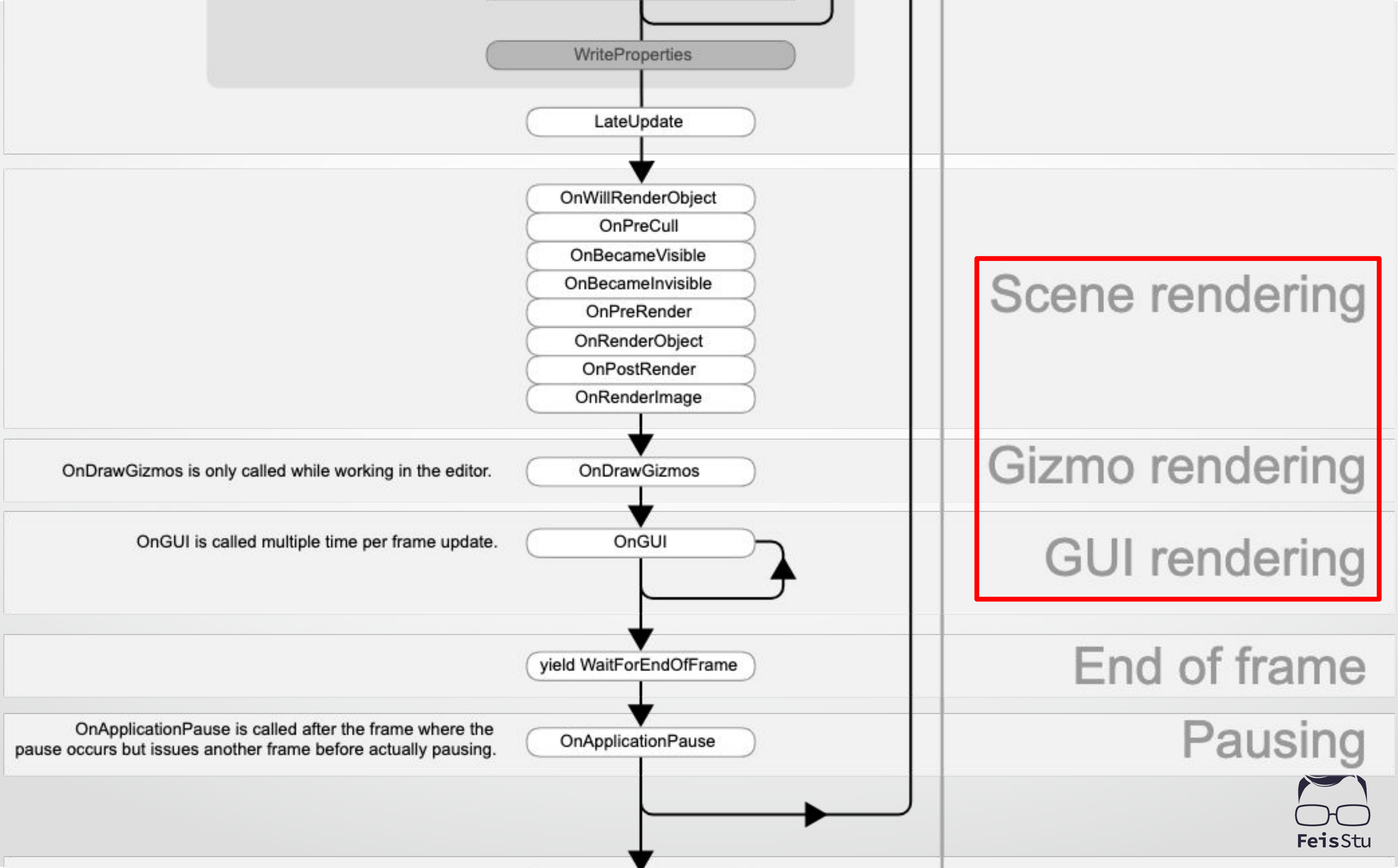
OnAnimatorIK

WriteTransform

WriteProperties

LateUpdate

OnWillRenderObject



WriteProperties

LateUpdate

OnWillRenderObject

OnPreCull

OnBecameVisible

OnBecameInvisible

OnPreRender

OnRenderObject

OnPostRender

OnRenderImage

Scene rendering

OnDrawGizmos is only called while working in the editor.

OnDrawGizmos

Gizmo rendering

OnGUI is called multiple time per frame update.

OnGUI

GUI rendering

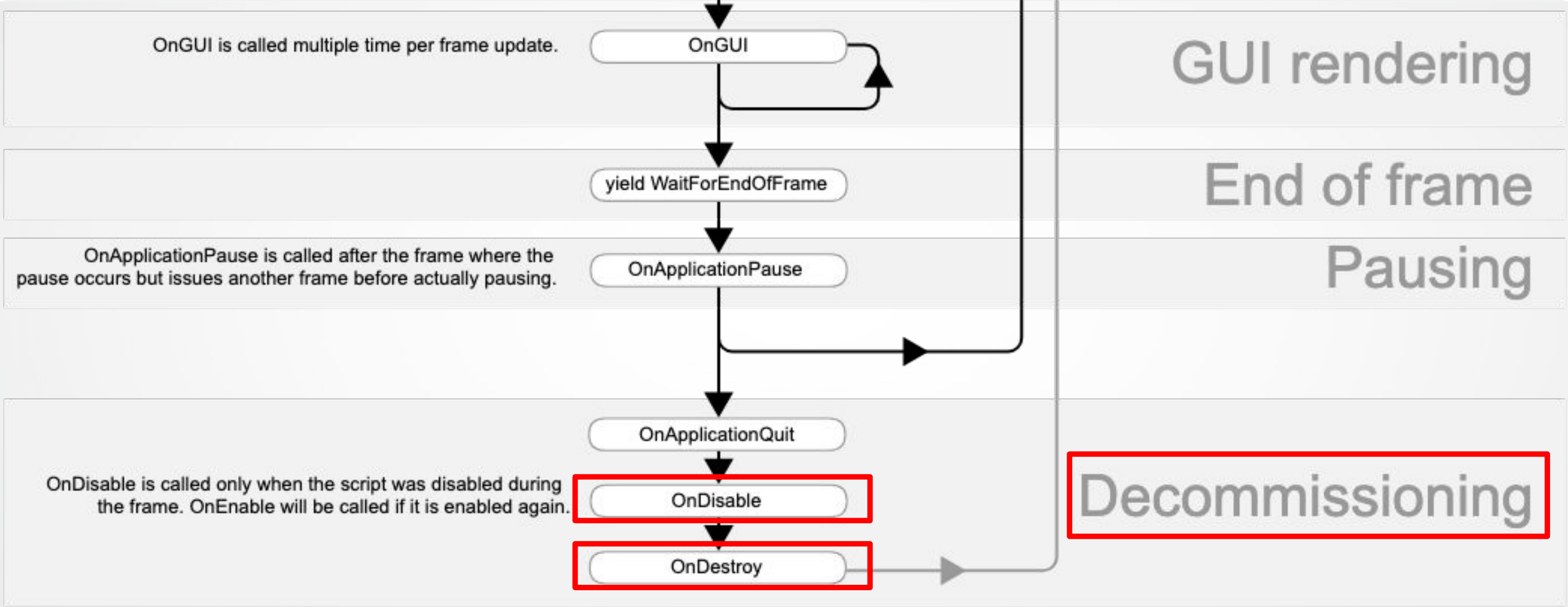
yield WaitForEndOfFrame

End of frame

OnApplicationPause is called after the frame where the pause occurs but issues another frame before actually pausing.

OnApplicationPause

Pausing



OnGUI is called multiple time per frame update.

OnGUI

GUI rendering

yield WaitForEndOfFrame

End of frame

OnApplicationPause is called after the frame where the pause occurs but issues another frame before actually pausing.

OnApplicationPause

Pausing

OnApplicationQuit

OnDisable is called only when the script was disabled during the frame. OnEnable will be called if it is enabled again.

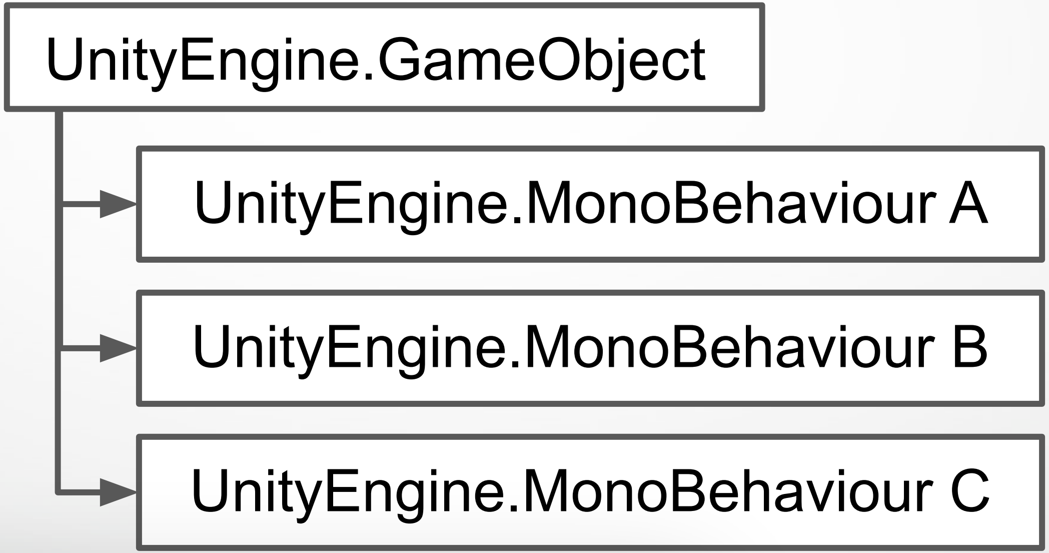
OnDisable

OnDestroy

Decommissioning

FeisStu

Script Execution Order



UnityEngine.GameObject

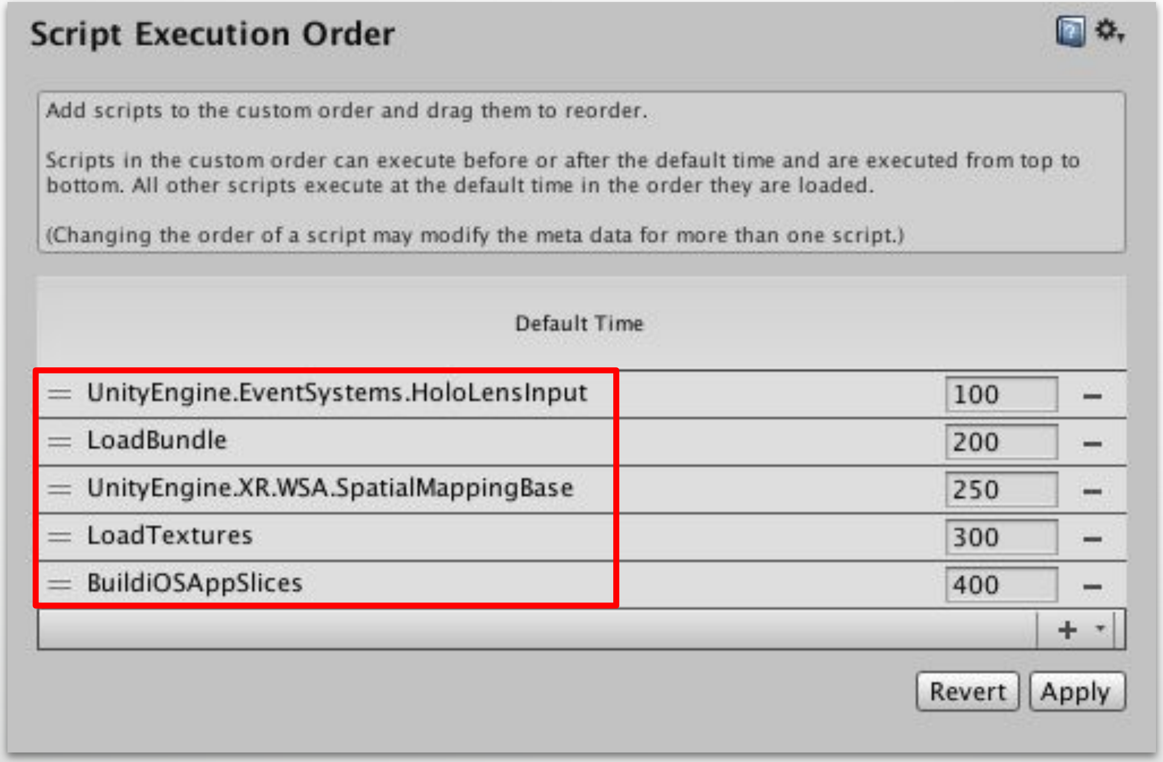
UnityEngine.MonoBehaviour A

UnityEngine.MonoBehaviour B

UnityEngine.MonoBehaviour C

Which component executes first ?

Script Execution Order settings



Script Execution Order

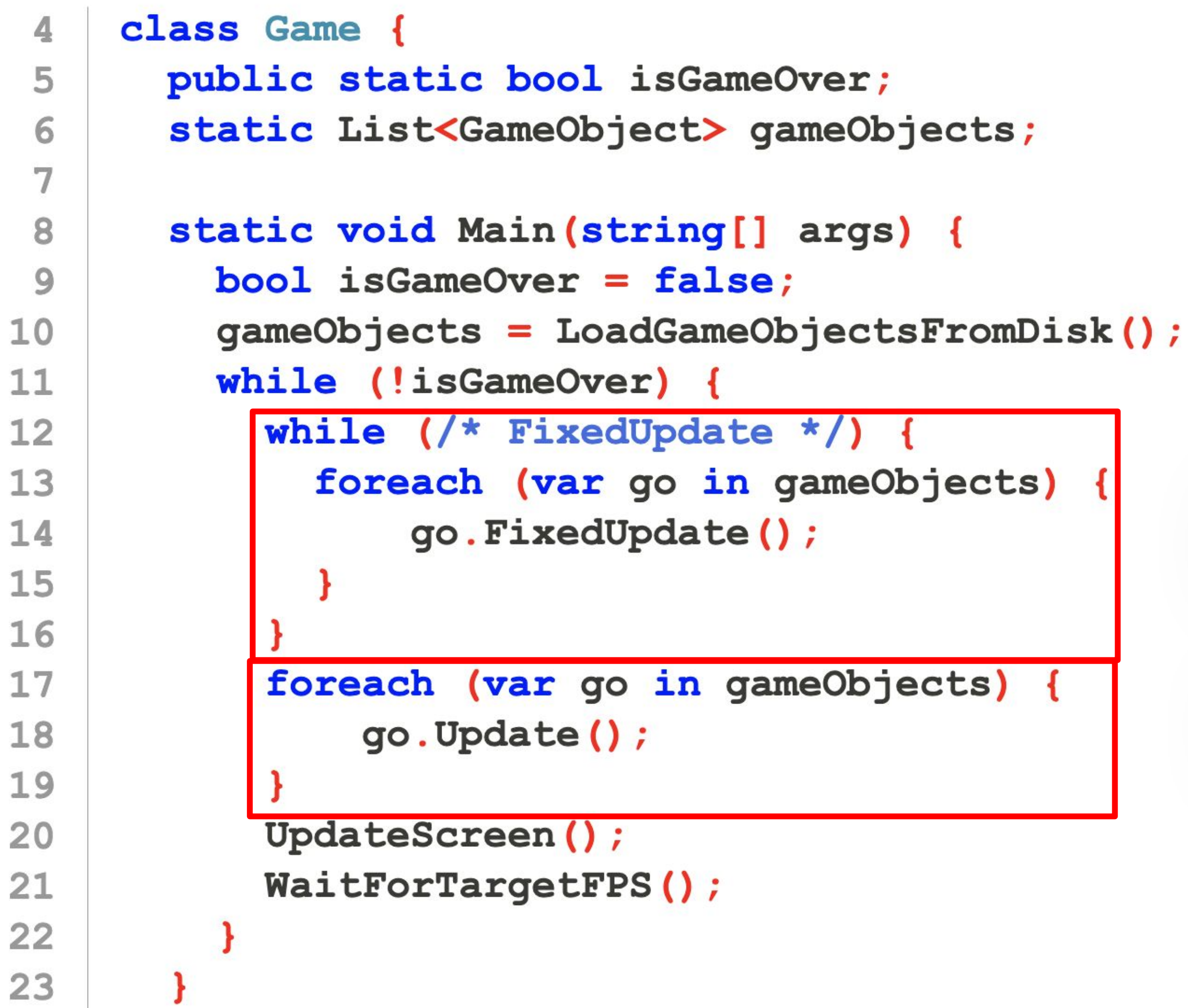
Add scripts to the custom order and drag them to reorder.

Scripts in the custom order can execute before or after the default time and are executed from top to bottom. All other scripts execute at the default time in the order they are loaded.

(Changing the order of a script may modify the meta data for more than one script.)

Default Time

UnityEngine.EventSystems.HoloLensInput	100
LoadBundle	200
UnityEngine.XR.WSA.SpatialMappingBase	250
LoadTextures	300
BuildiOSAppSlices	400



```
class Game {
  public static bool isGameOver;
  static List<GameObject> gameObjects;

  static void Main(string[] args) {
    bool isGameOver = false;
    gameObjects = LoadGameObjectsFromDisk();
    while (!isGameOver) {
      while (/* FixedUpdate */) {
        foreach (var go in gameObjects) {
            go.FixedUpdate();
        }
      }
      foreach (var go in gameObjects) {
          go.Update();
      }
      UpdateScreen();
      WaitForTargetFPS();
    }
  }
```

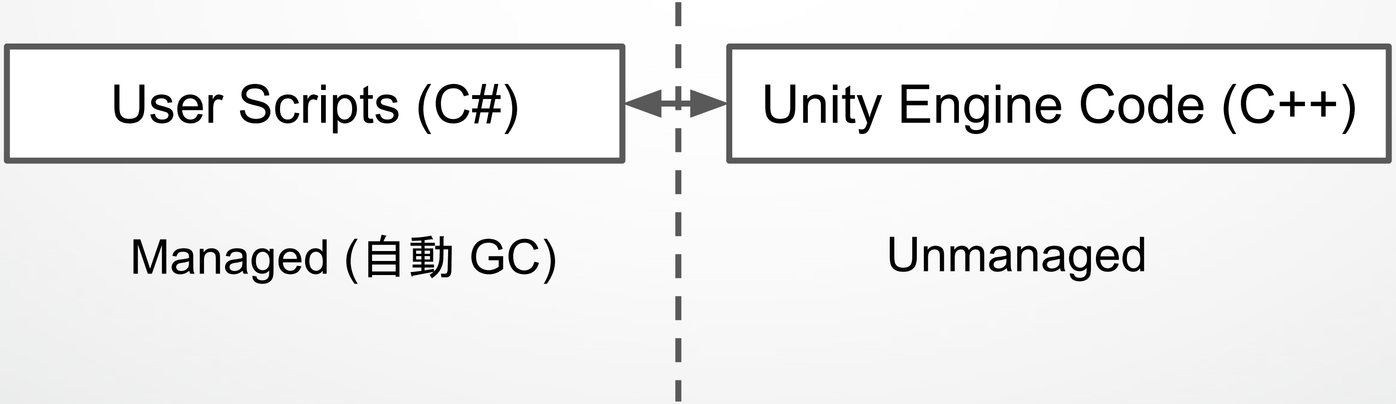
FixedUpdate()

Update()

Conclusion about order of execution

- Cooperative multitasking, single thread
- Order of Event Functions
  - https://docs.unity3d.com/Manual/ExecutionOrder.html
- Order of GameObjects ?
- Order of MonoBehaviours
  - Script Execution Order settings: https://docs.unity3d.com/Manual/class-MonoManager.html

Where is the Main function ?



User Scripts (C#)

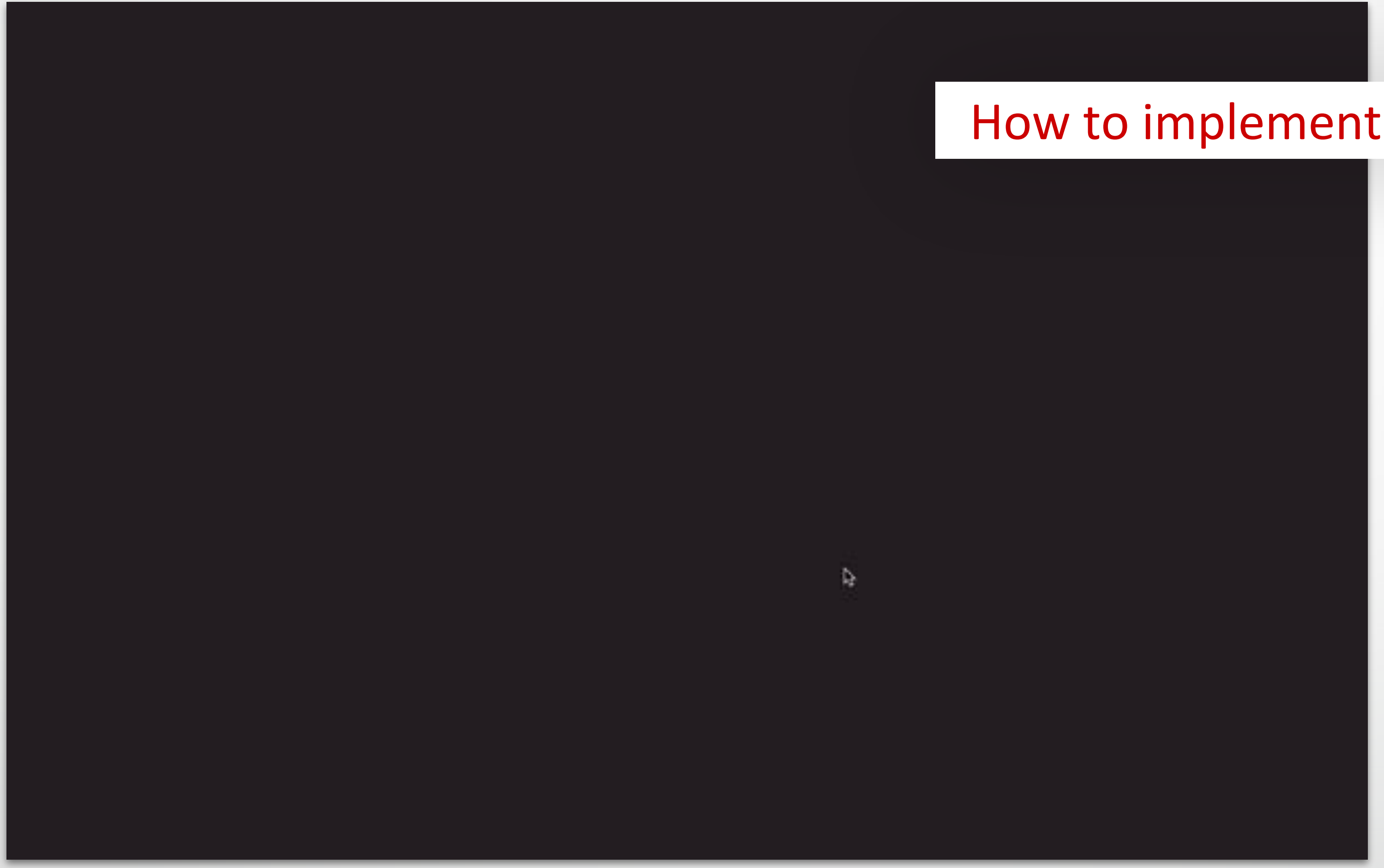
Unity Engine Code (C++)

Managed (自動 GC)

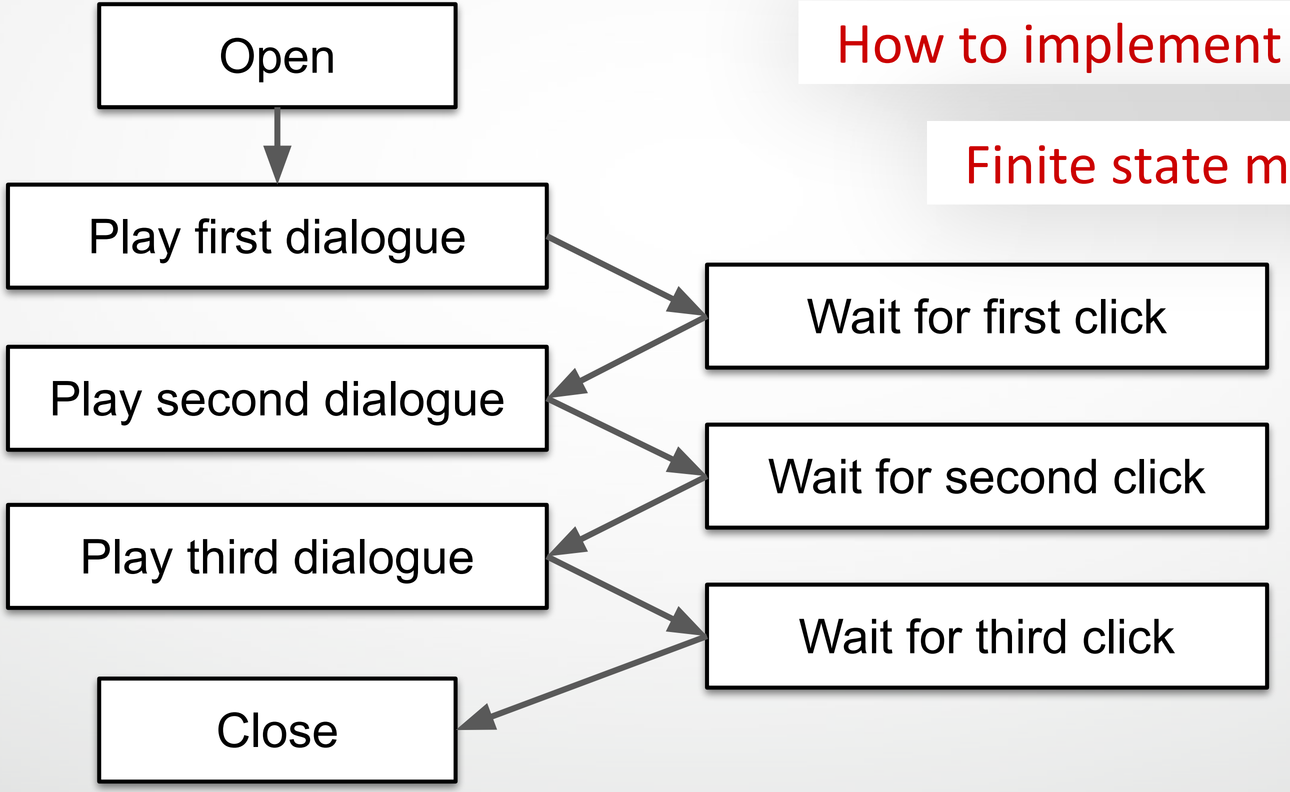
Unmanaged

Q & A

FeisStu

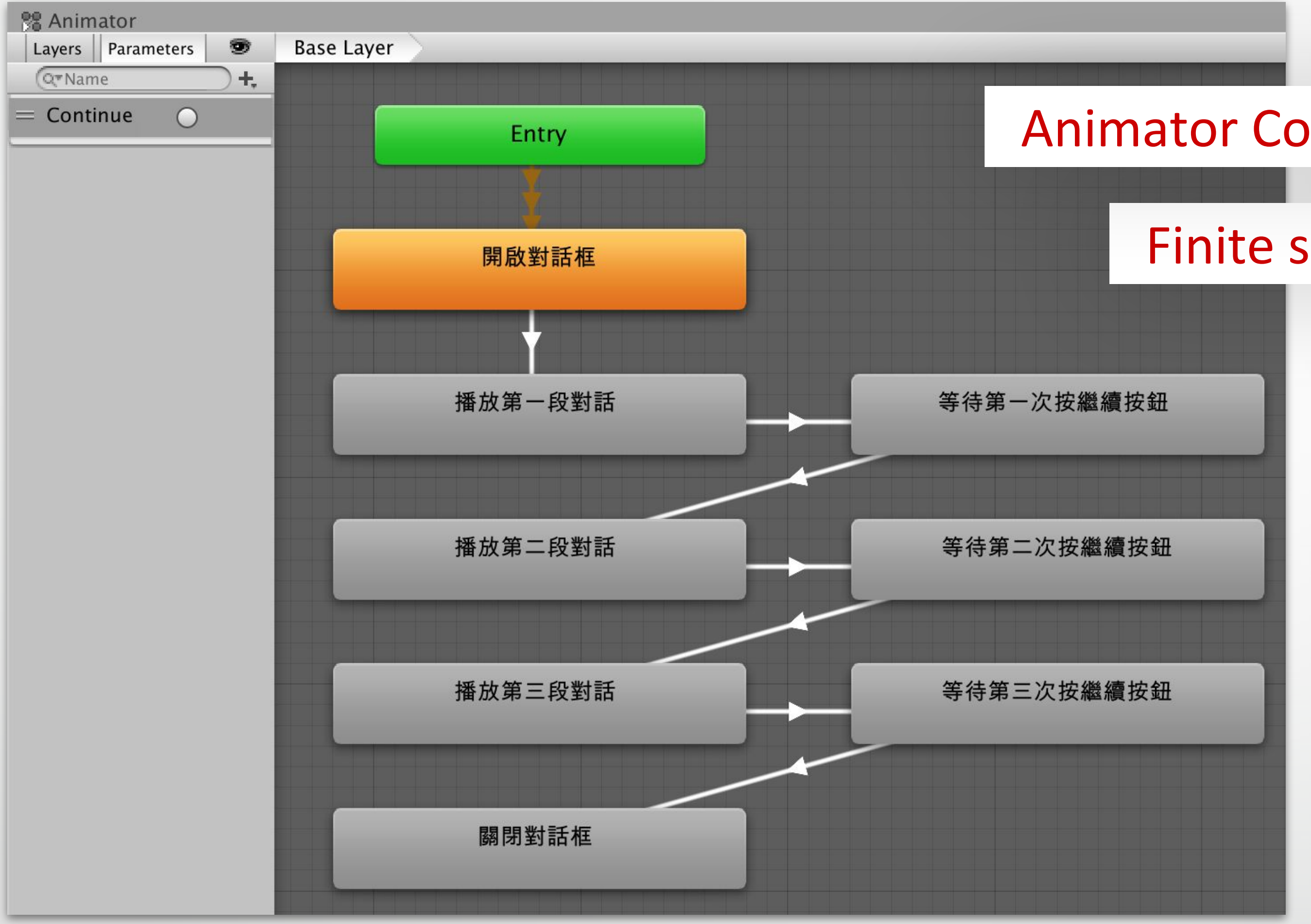


How to implement this ?



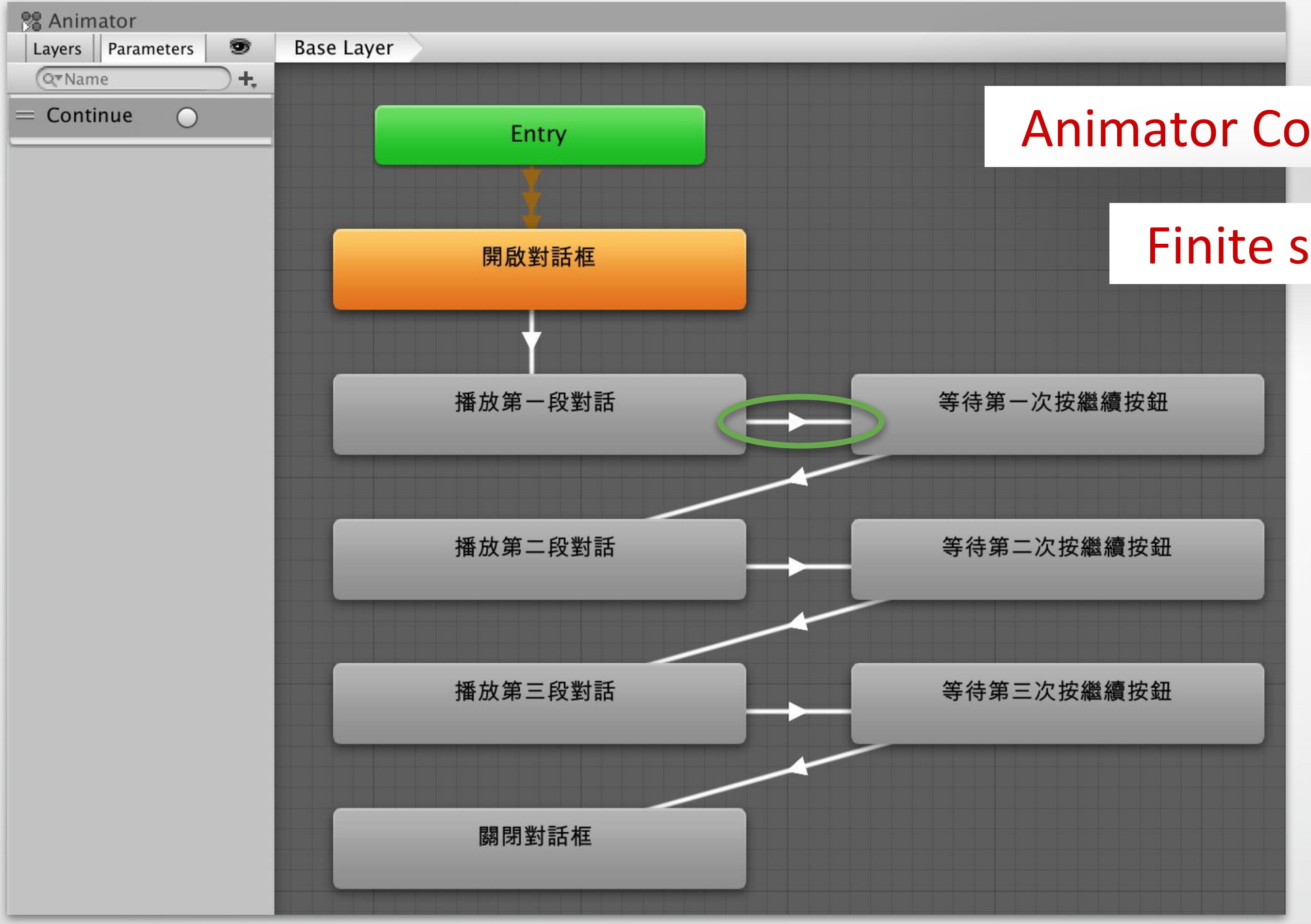
How to implement this ?

Finite state machine



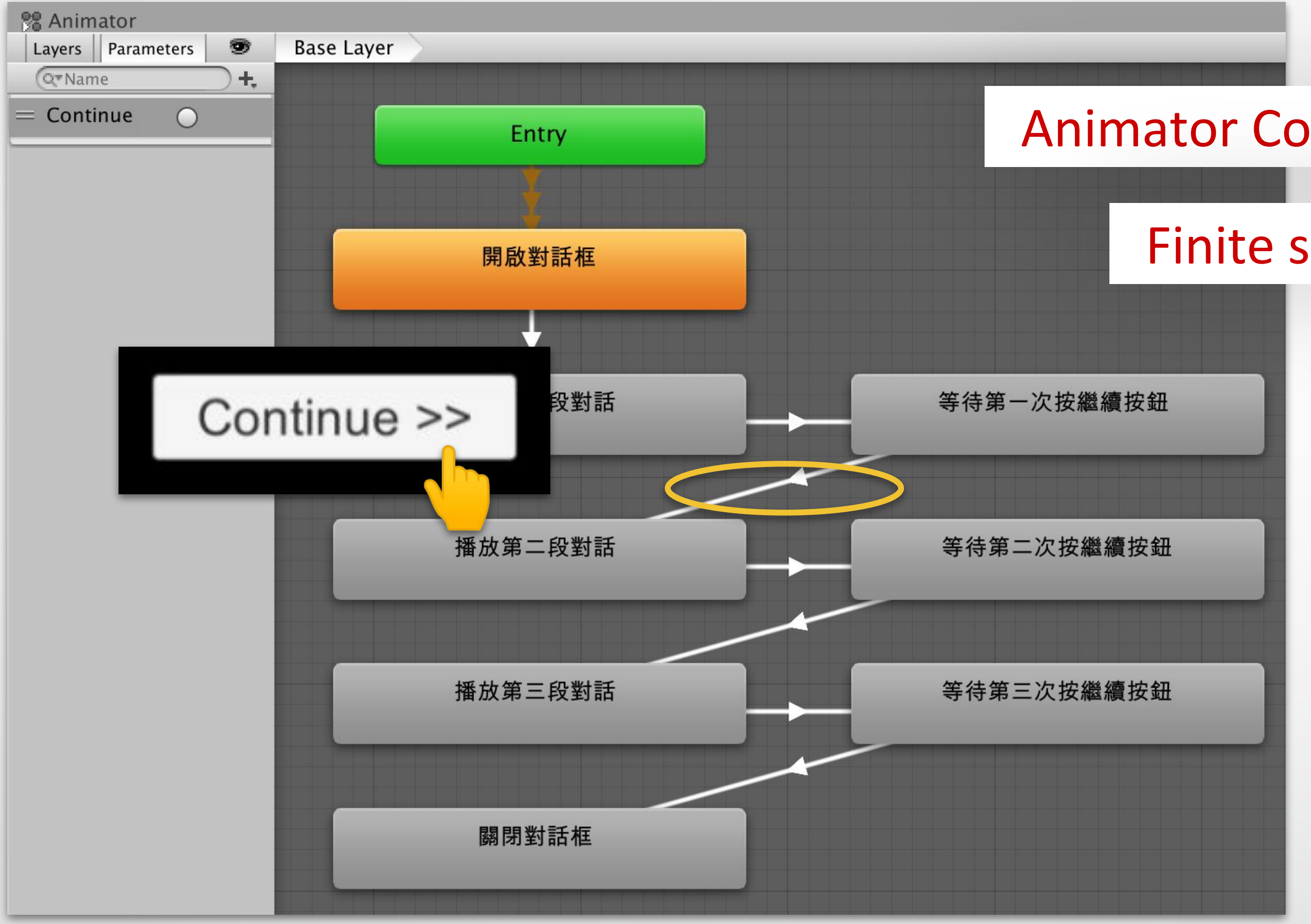
Animator Controller

Finite state machine



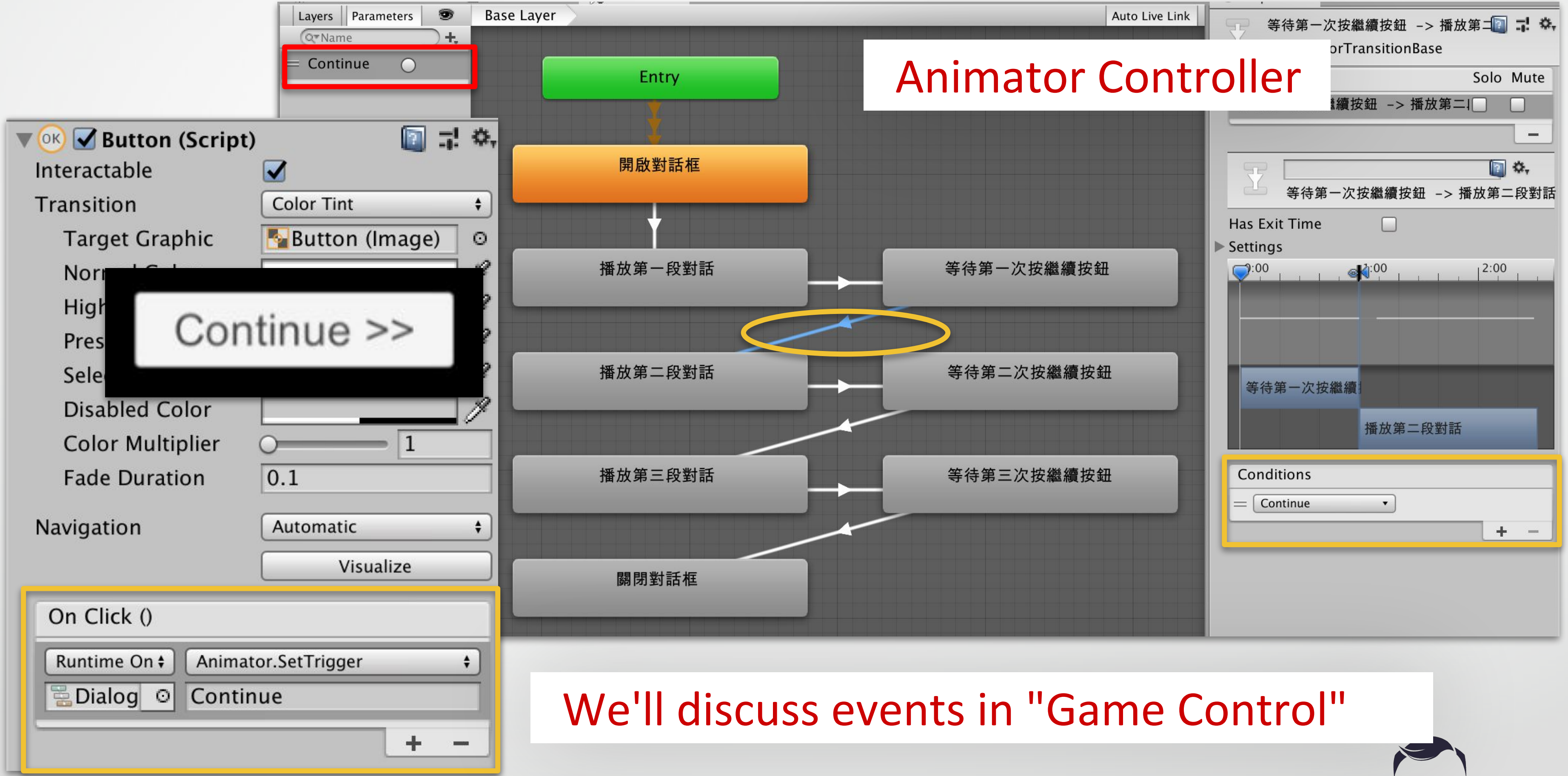
Animator Controller

Finite state machine



Animator Controller

Finite state machine



Animator Controller

Layers | Parameters | Base Layer | Auto Live Link

Continue ○

Button (Script)

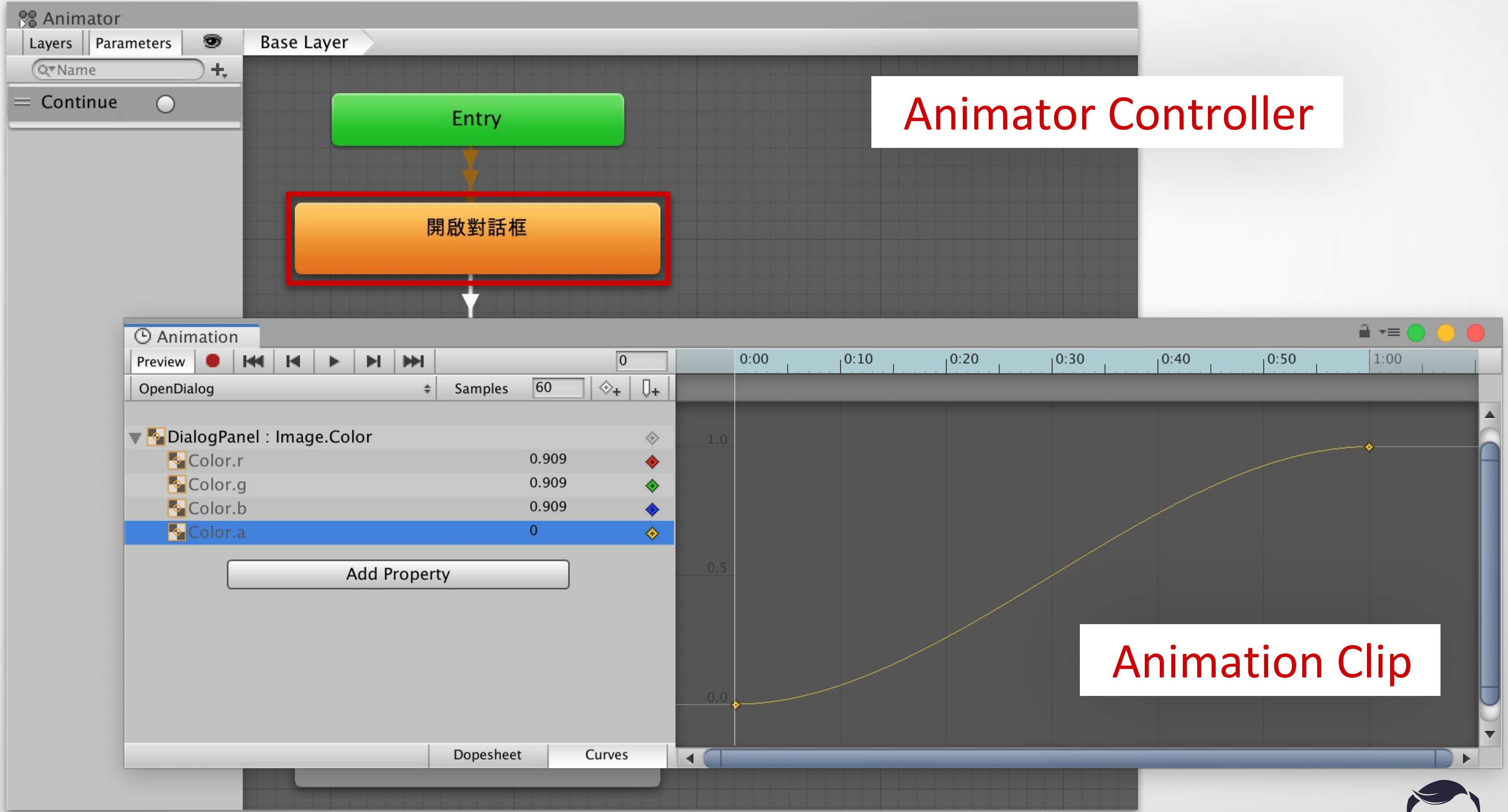
Interactable ☑
Transition: Color Tint
Target Graphic: Button (Image)
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
Visualize

On Click ()
Runtime On | Animator.SetTrigger
Dialog | Continue

等待第一次按繼續按鈕 -> 播放第二段對話
Has Exit Time □
Settings

Conditions
Continue

We'll discuss events in "Game Control"

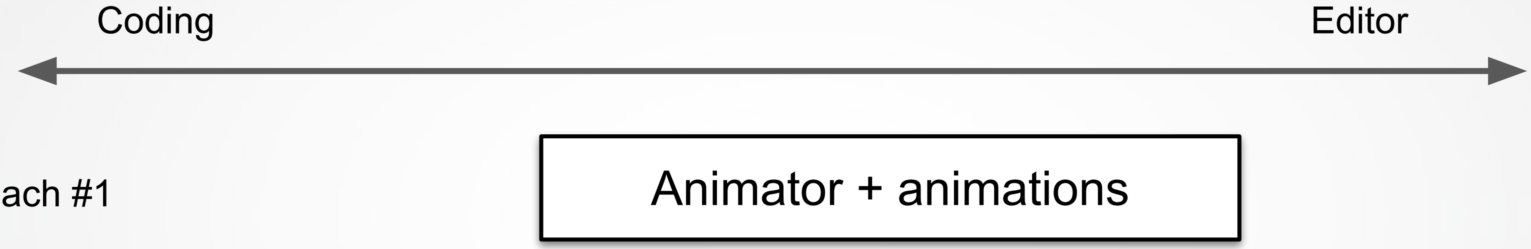


Animator Controller

Animation Clip

OpenDialog　Samples 60

DialogPanel : Image.Color	
Color.r	0.909
Color.g	0.909
Color.b	0.909
Color.a	0

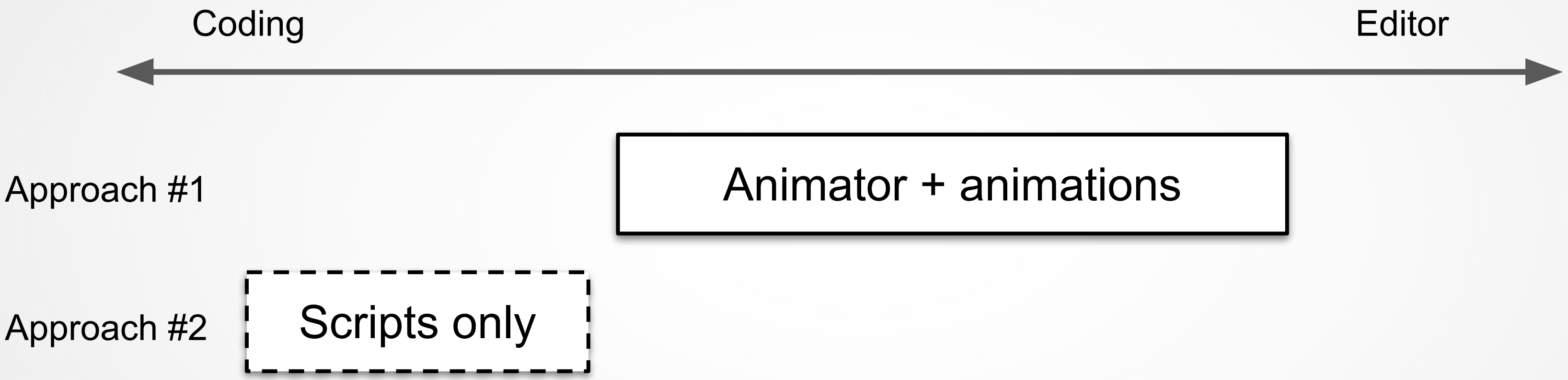


Coding

Editor

Approach #1

Animator + animations



Coding

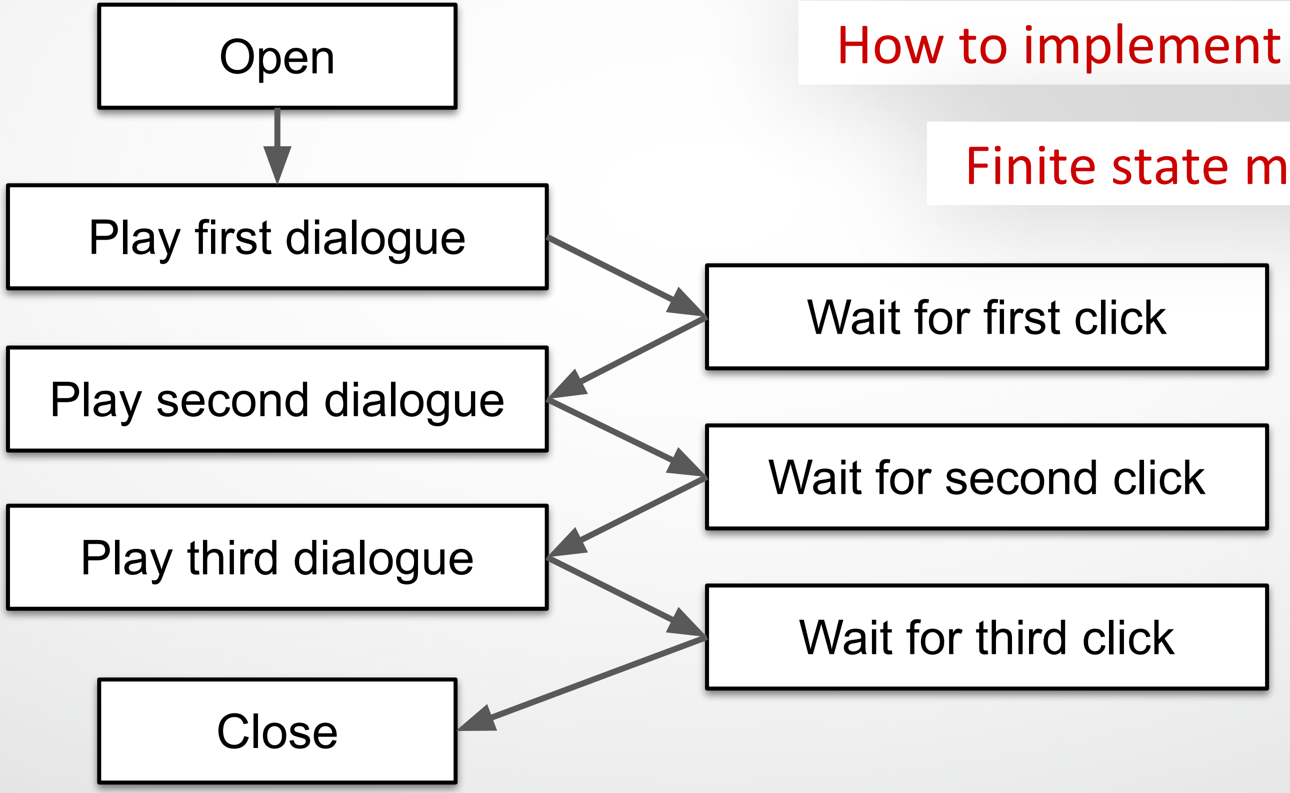
Editor

Approach #1

Animator + animations

Approach #2

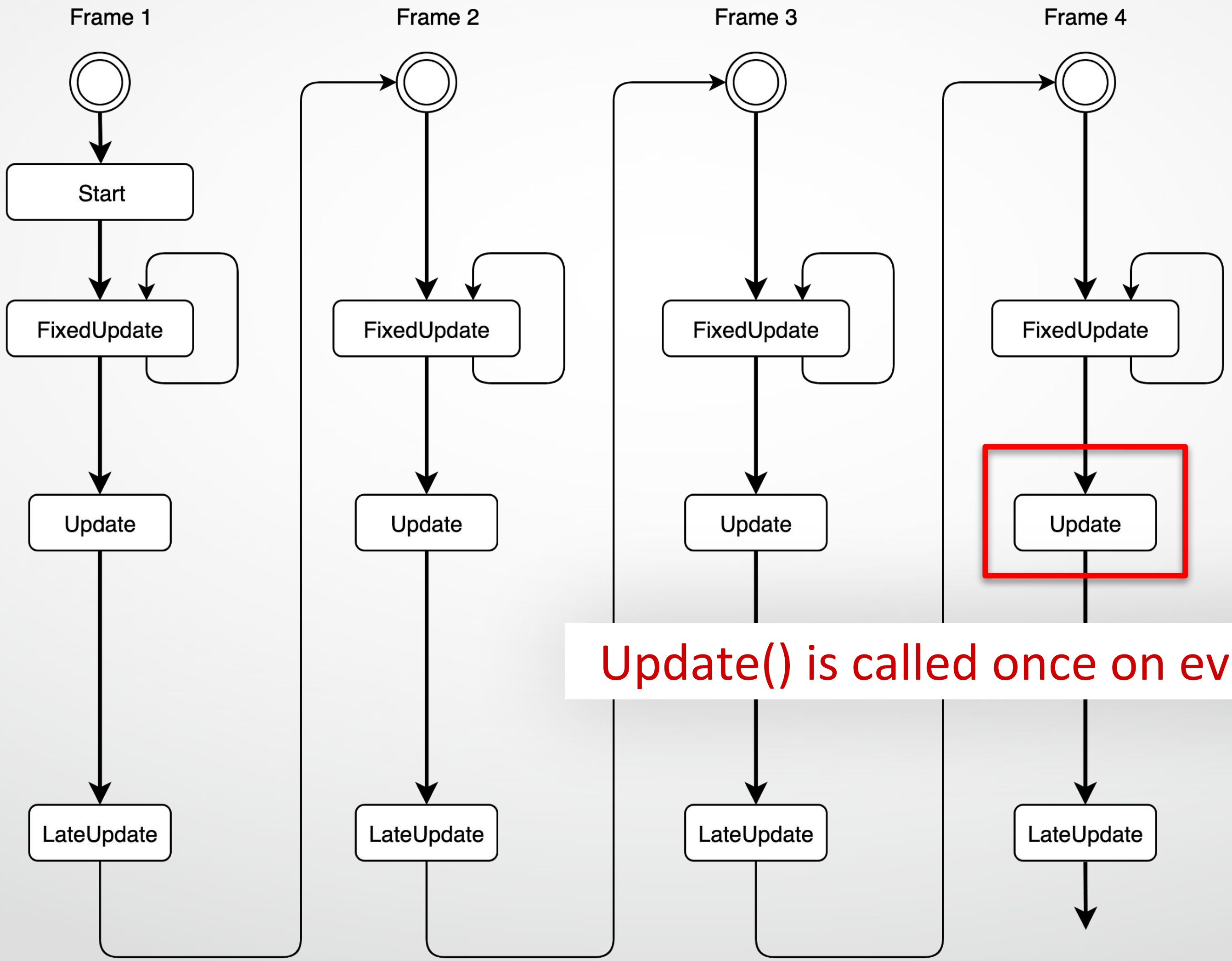
Scripts only



How to implement this ?

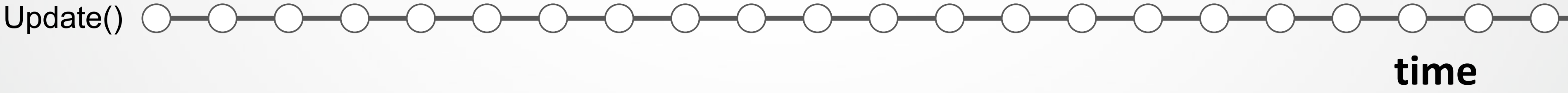
Finite state machine

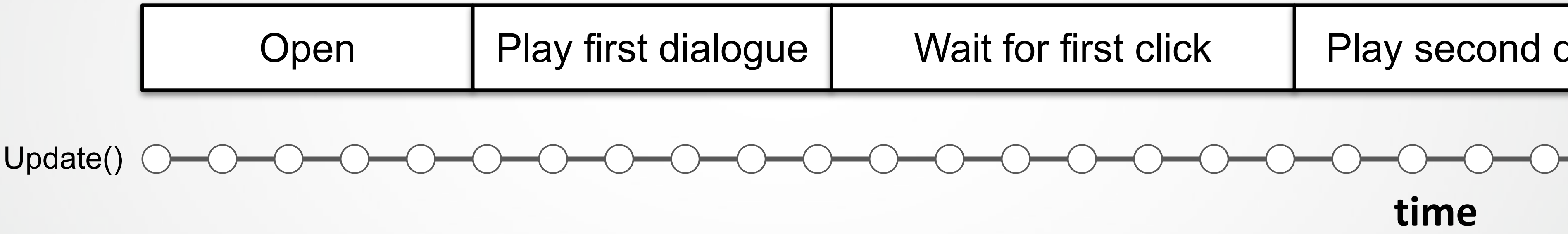
FeisStu



Update() is called once on every frame

time


Update()
time



Open

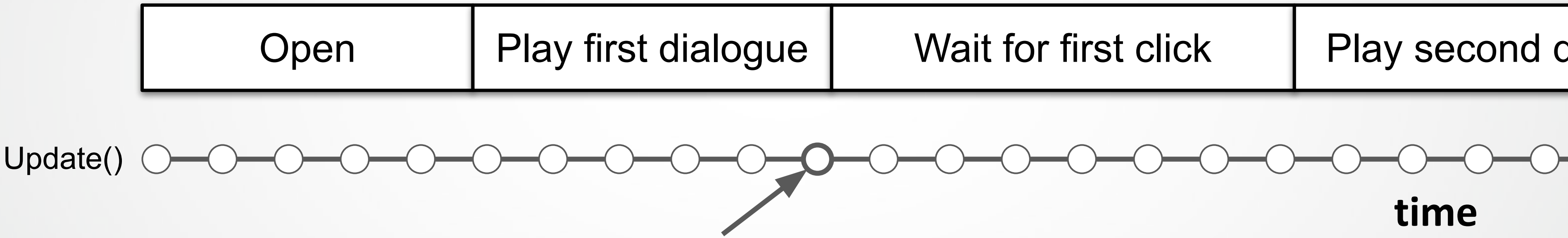
Play first dialogue

Wait for first click

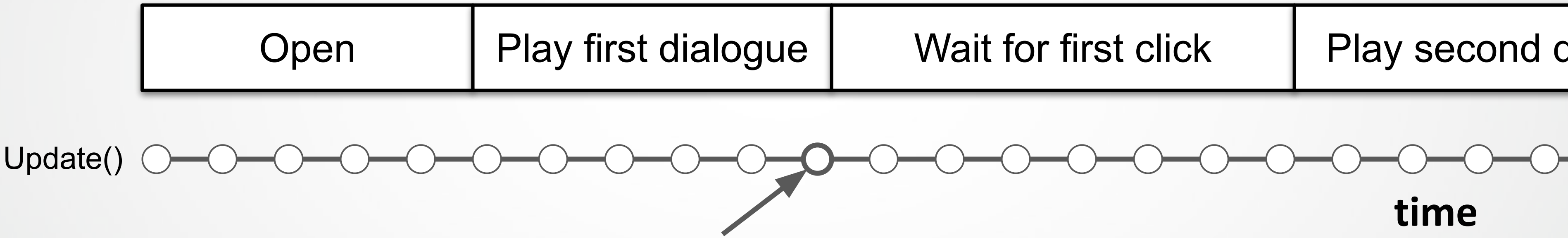
Play second d

Update()

time



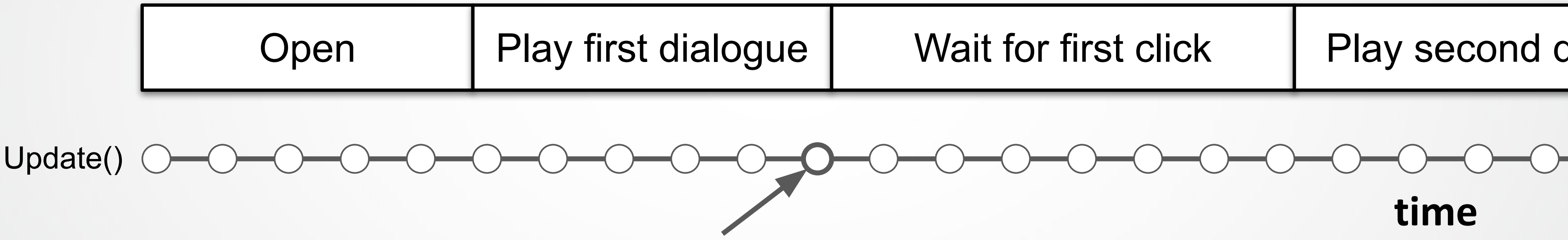
What to do on this frame ?



What to do on this frame ?

```
string currentState
int animationFrameIndex
bool isContinueButtonClicked
```

State



Open

Play first dialogue

Wait for first click

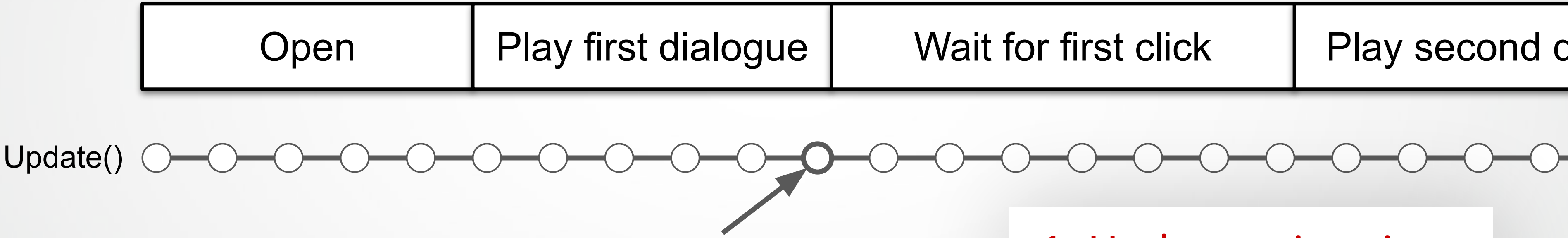
Play second d

Update()

time

What to do on this frame ?

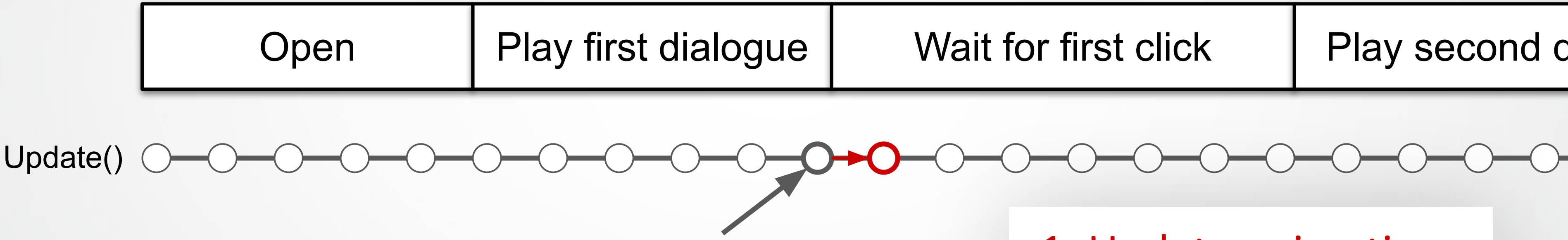
```
string currentState            :  "Play first dialogue"
int animationFrameIndex        :  6
bool isContinueButtonClicked   :  (Don't care)
```



What to do on this frame ?

1. Update animation

```
string currentState            :  "Play first dialogue"
int animationFrameIndex        :  6
bool isContinueButtonClicked   :  (Don't care)
```

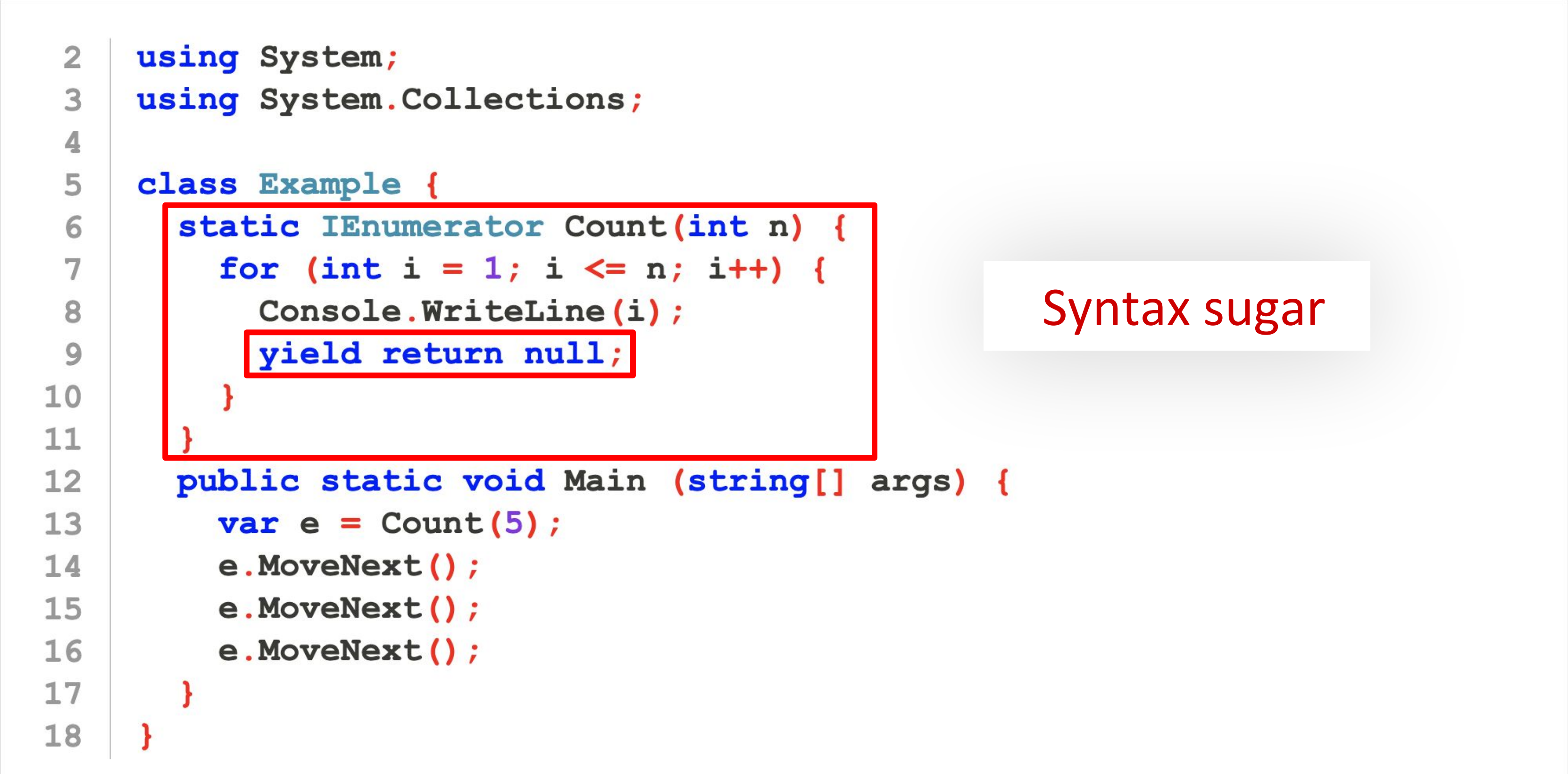


What to do on this frame ?

1. Update animation
2. Decide next state

`string currentState`	:	"Play first dialogue"	"Wait for first click"
`int animationFrameIndex`	:	6	(Don't care)
`bool isContinueButtonClicked`	:	(Don't care)	false

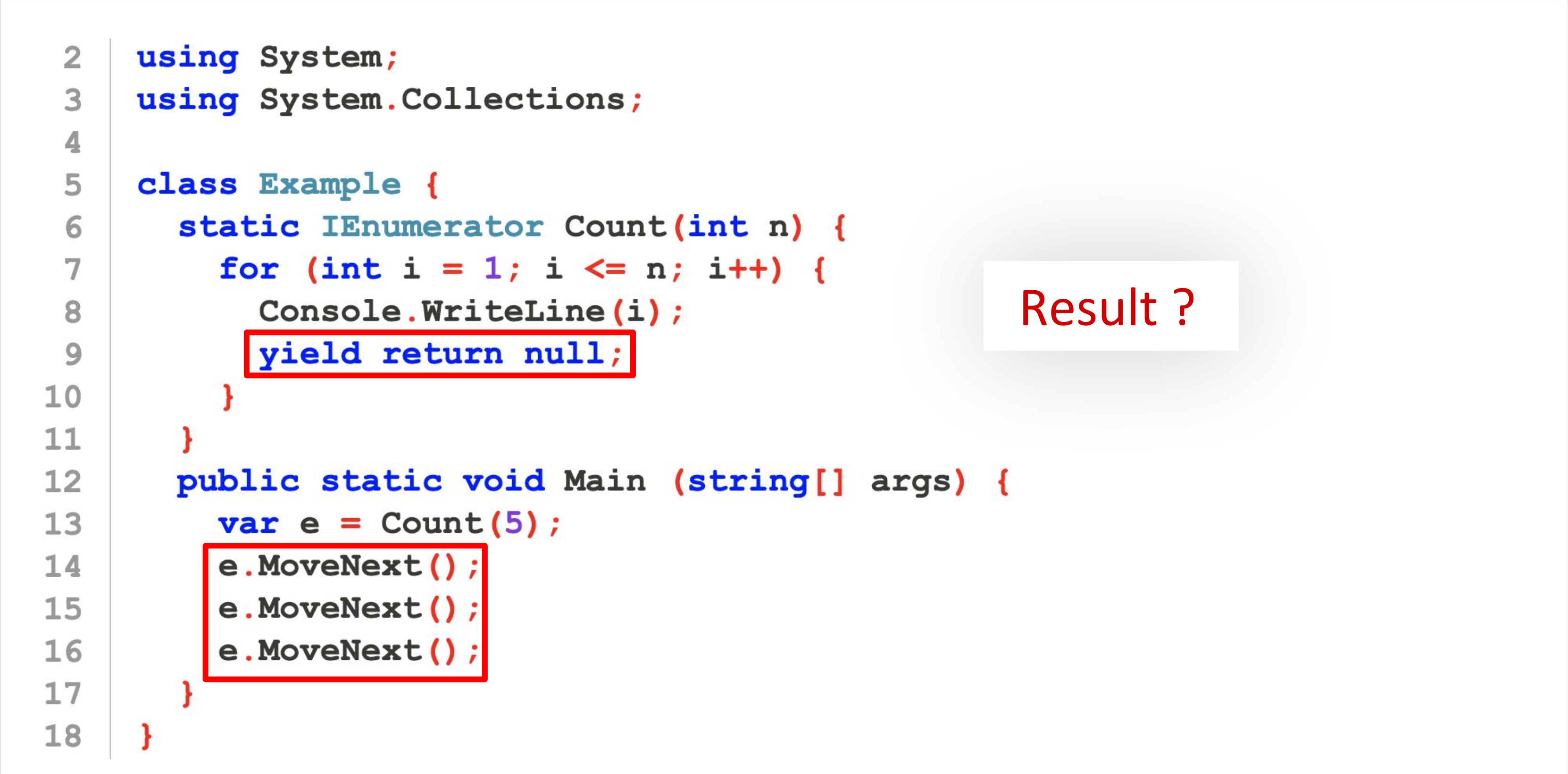
Iterator pattern



```
using System;
using System.Collections;

class Example {
  static IEnumerator Count(int n) {
    for (int i = 1; i <= n; i++) {
      Console.WriteLine(i);
      yield return null;
    }
  }
  public static void Main (string[] args) {
    var e = Count(5);
    e.MoveNext();
    e.MoveNext();
    e.MoveNext();
  }
}
```

Syntax sugar

Iterator pattern

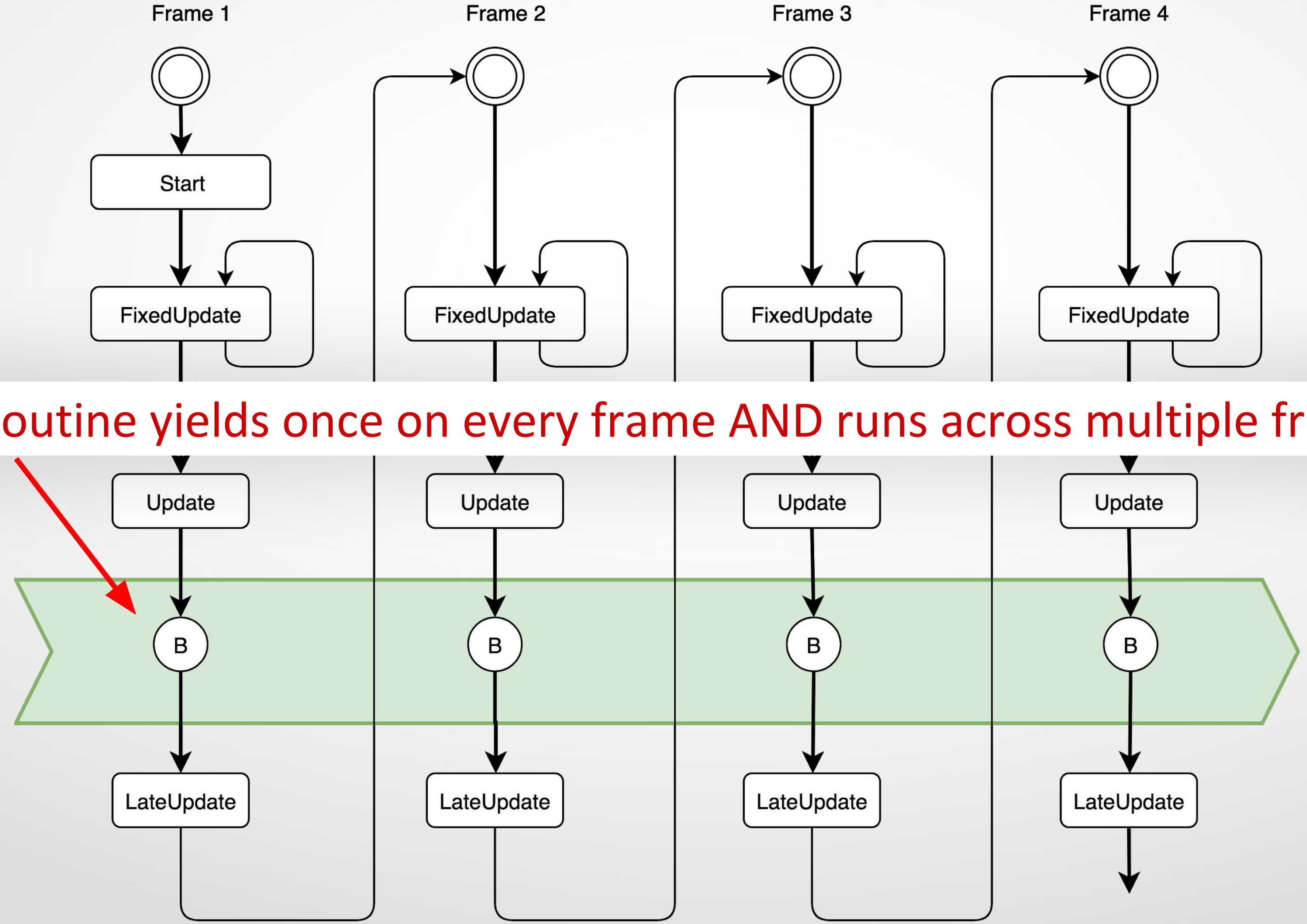


```
using System;
using System.Collections;

class Example {
  static IEnumerator Count(int n) {
    for (int i = 1; i <= n; i++) {
      Console.WriteLine(i);
      yield return null;
    }
  }
  public static void Main (string[] args) {
    var e = Count(5);
    e.MoveNext();
    e.MoveNext();
    e.MoveNext();
  }
}
```

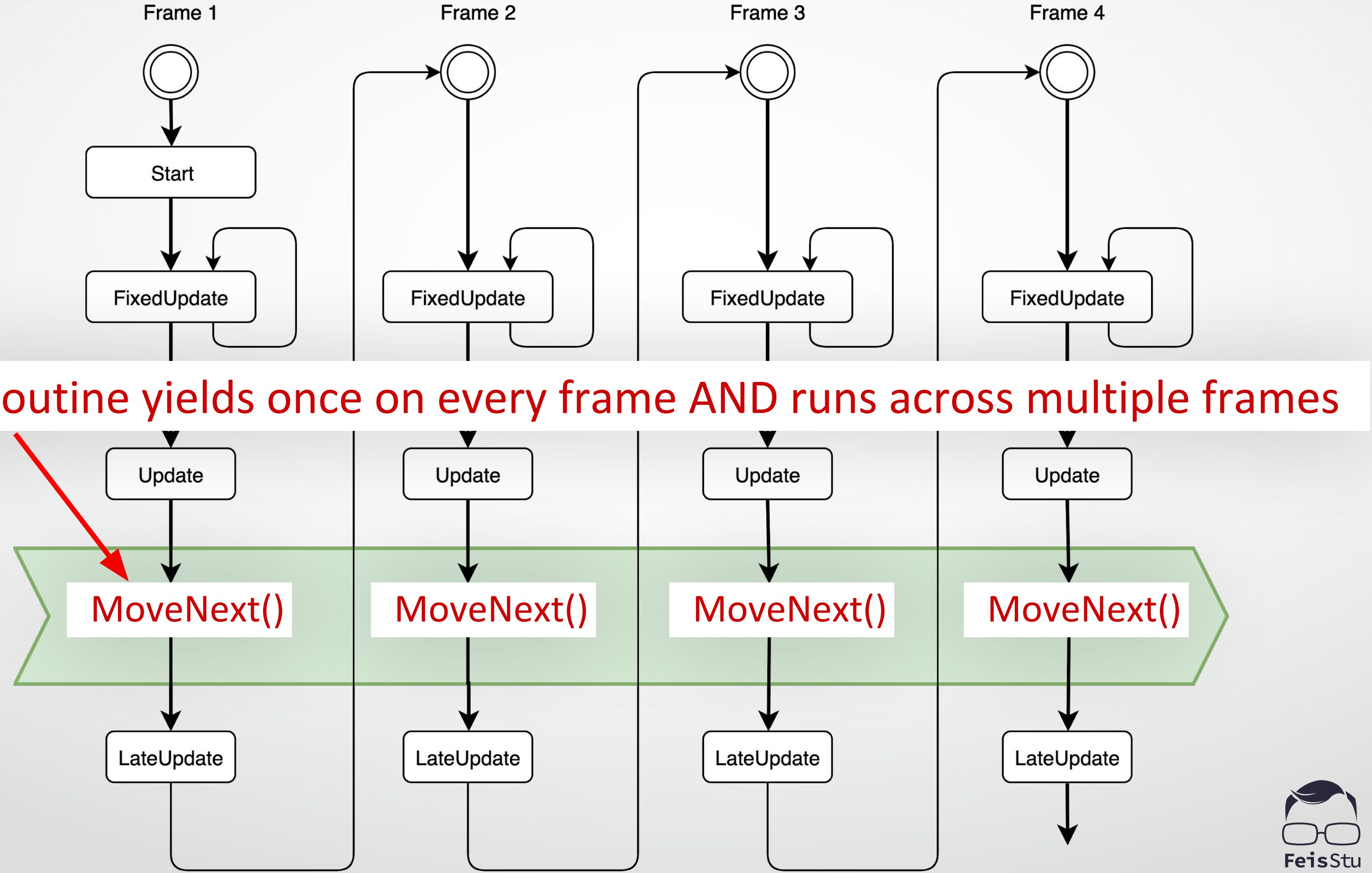
Result ?

UnityEngine.Coroutine

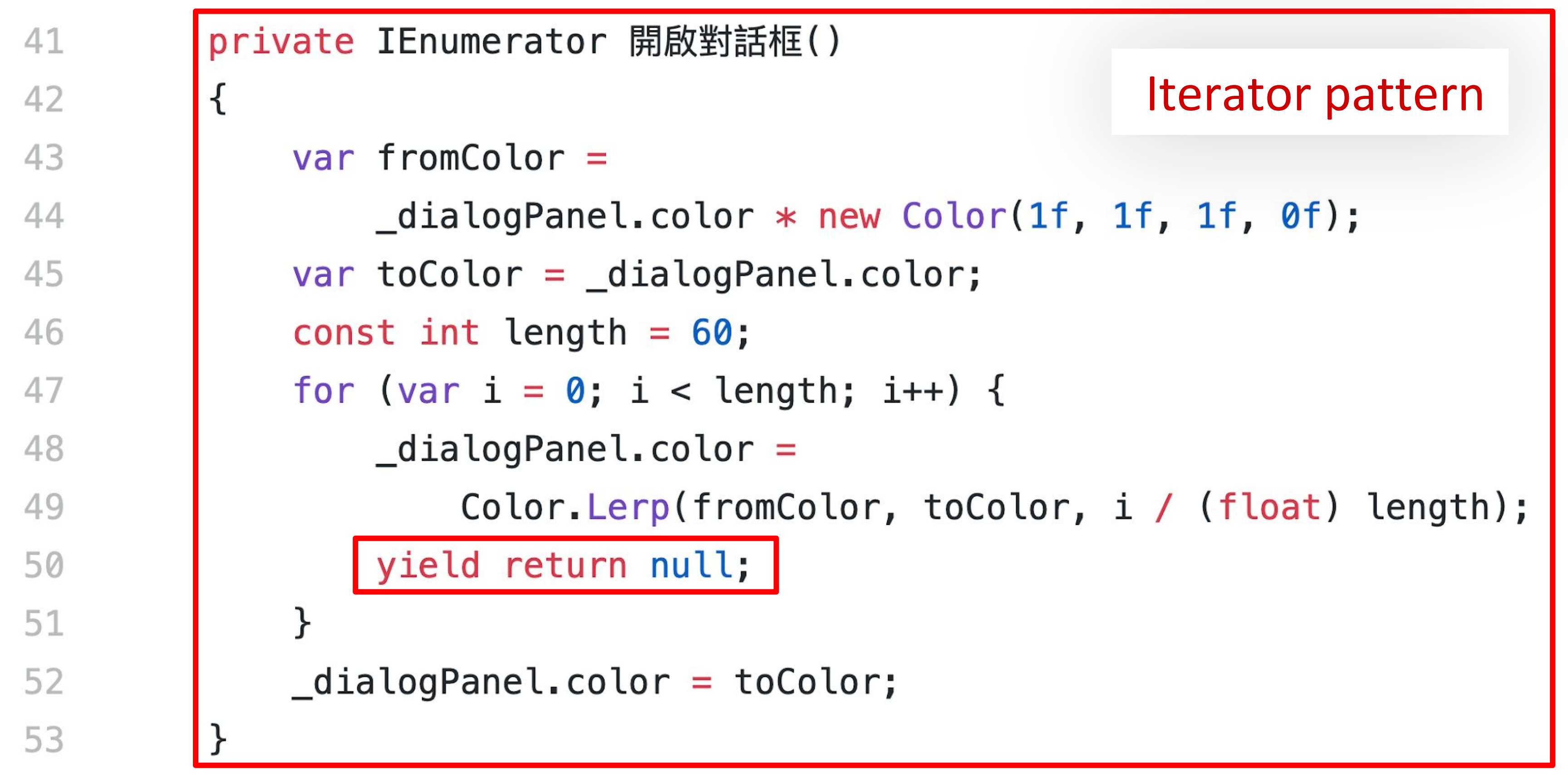
- Create a UnityEngine.Coroutine by calling MonoBehaviour.StartCoroutine()
  - The parameter is an IEnumerator object
- StartCoroutine(Count(5))



Coroutine yields once on every frame AND runs across multiple frames

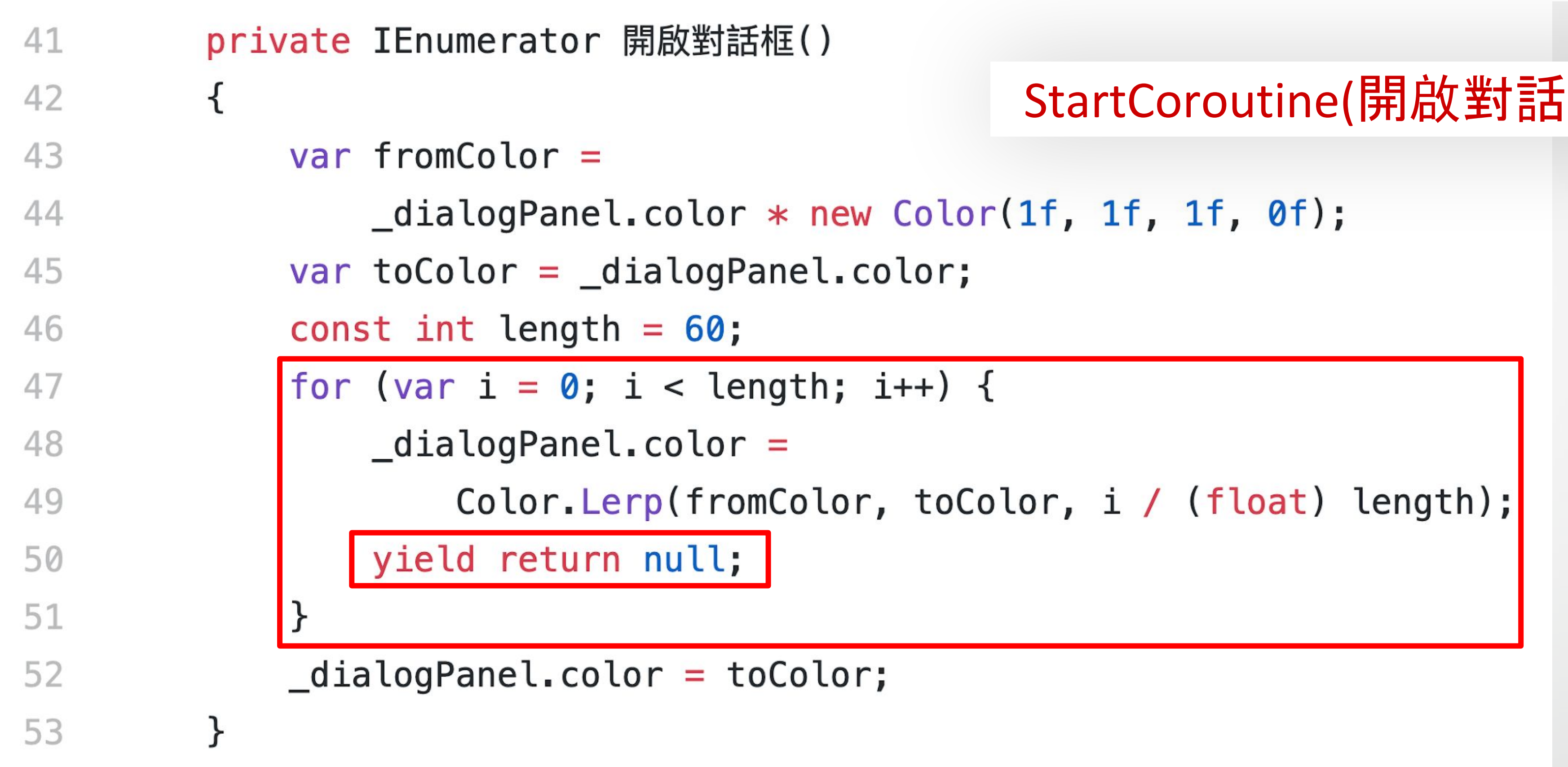


Coroutine yields once on every frame AND runs across multiple frames



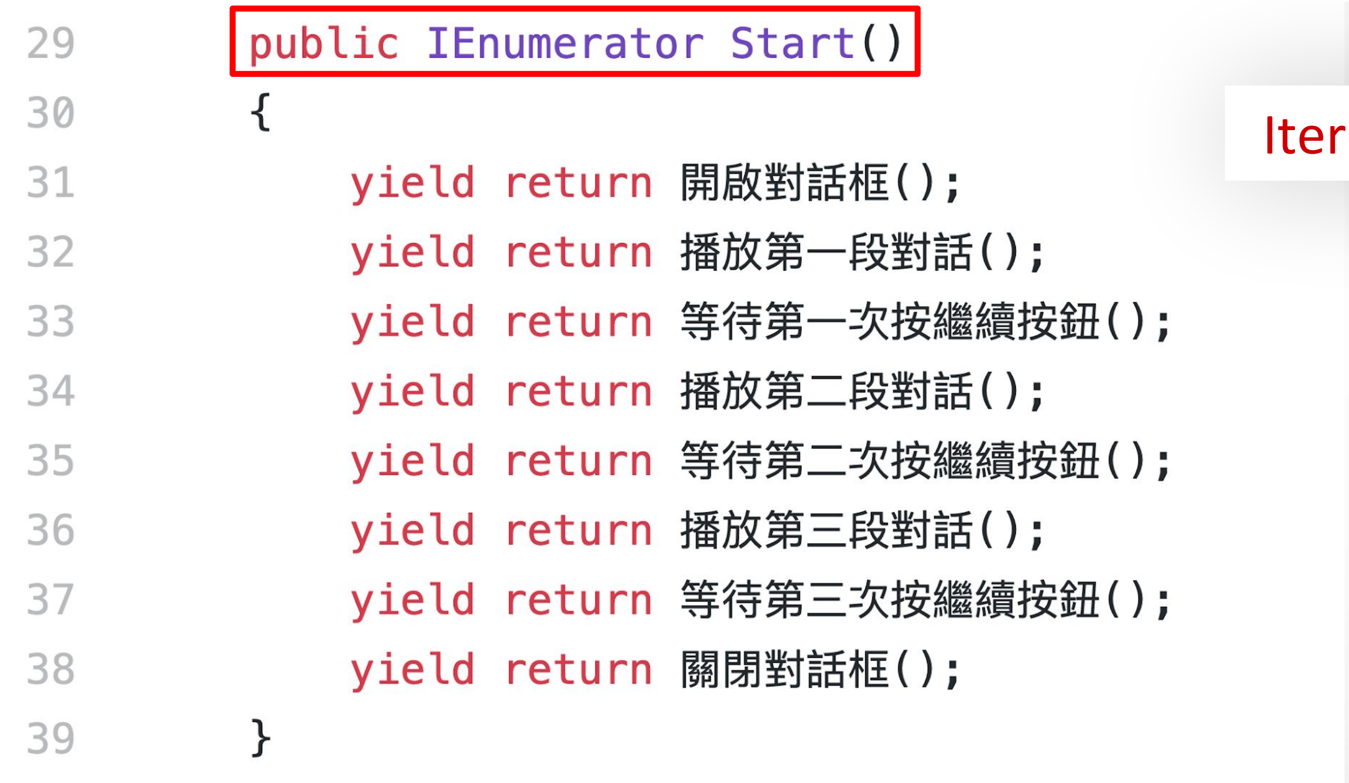
```
private IEnumerator 開啟對話框()
{
    var fromColor =
        _dialogPanel.color * new Color(1f, 1f, 1f, 0f);
    var toColor = _dialogPanel.color;
    const int length = 60;
    for (var i = 0; i < length; i++) {
        _dialogPanel.color =
            Color.Lerp(fromColor, toColor, i / (float) length);
        yield return null;
    }
    _dialogPanel.color = toColor;
}
```

Iterator pattern



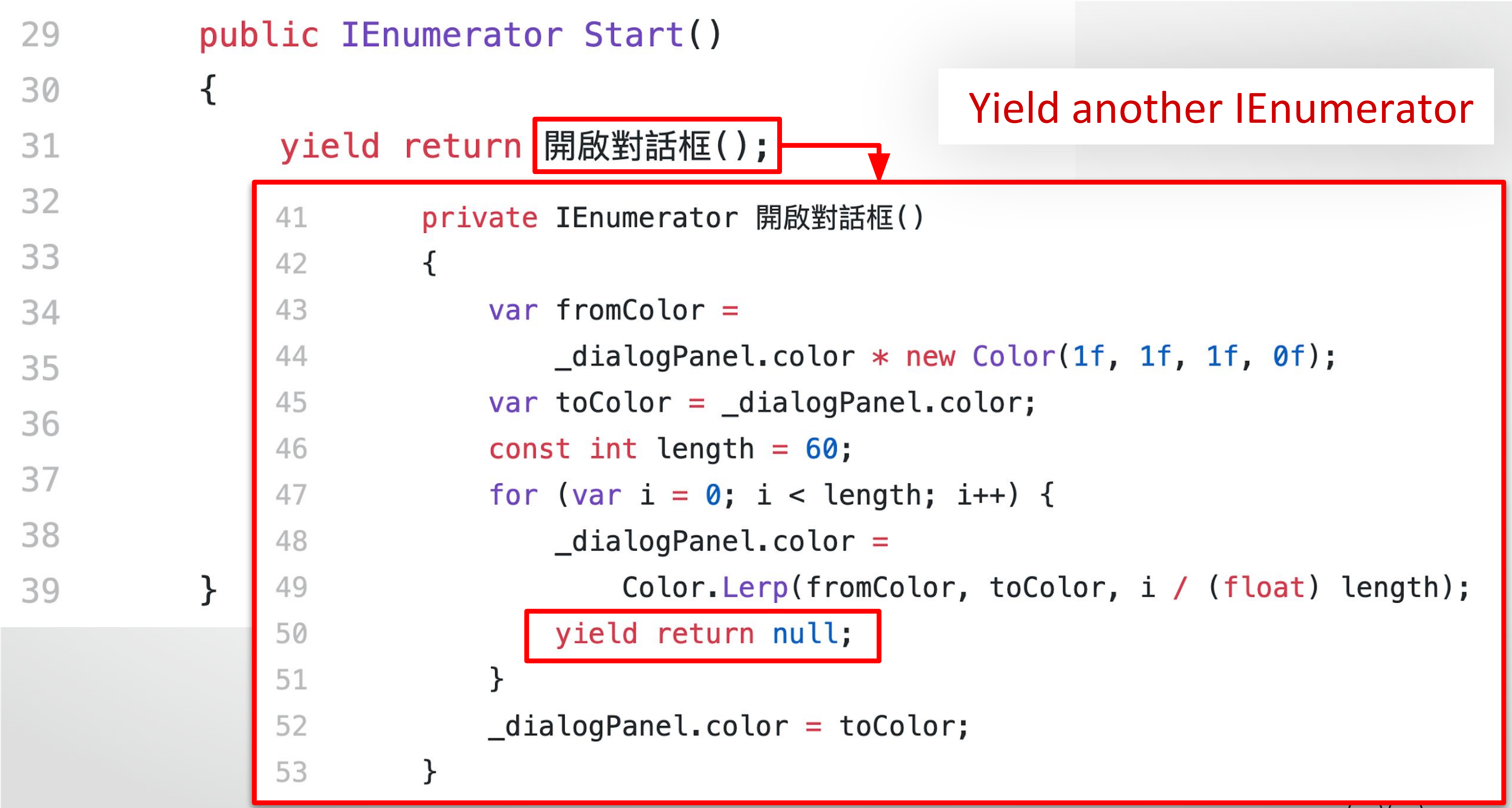
```
    private IEnumerator 開啟對話框()
    {
        var fromColor =
            _dialogPanel.color * new Color(1f, 1f, 1f, 0f);
        var toColor = _dialogPanel.color;
        const int length = 60;
        for (var i = 0; i < length; i++) {
            _dialogPanel.color =
                Color.Lerp(fromColor, toColor, i / (float) length);
            yield return null;
        }
        _dialogPanel.color = toColor;
    }
```

StartCoroutine(開啟對話框)



```
public IEnumerator Start()
{
    yield return 開啟對話框();
    yield return 播放第一段對話();
    yield return 等待第一次按繼續按鈕();
    yield return 播放第二段對話();
    yield return 等待第二次按繼續按鈕();
    yield return 播放第三段對話();
    yield return 等待第三次按繼續按鈕();
    yield return 關閉對話框();
}
```

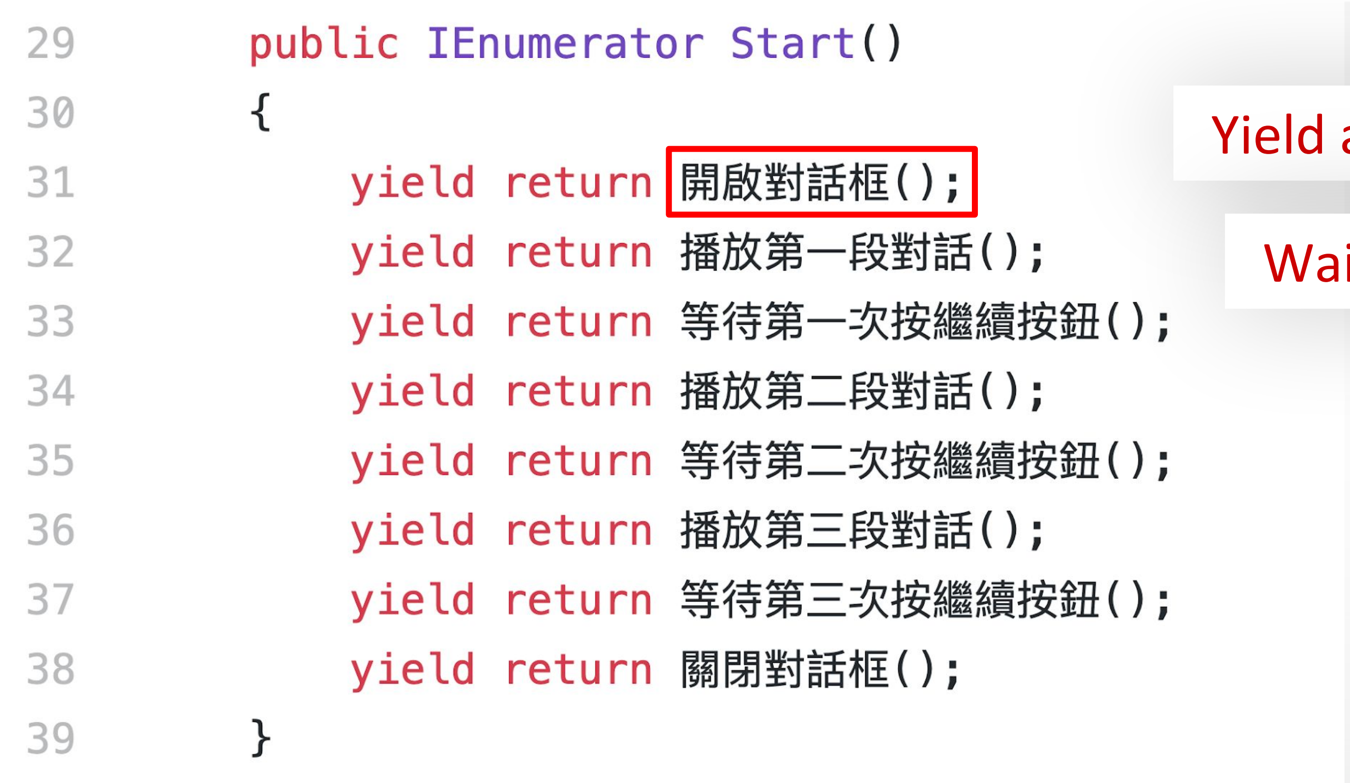
Iterator pattern



Yield another IEnumerator

```
public IEnumerator Start()
{
    yield return 開啟對話框();



}
```

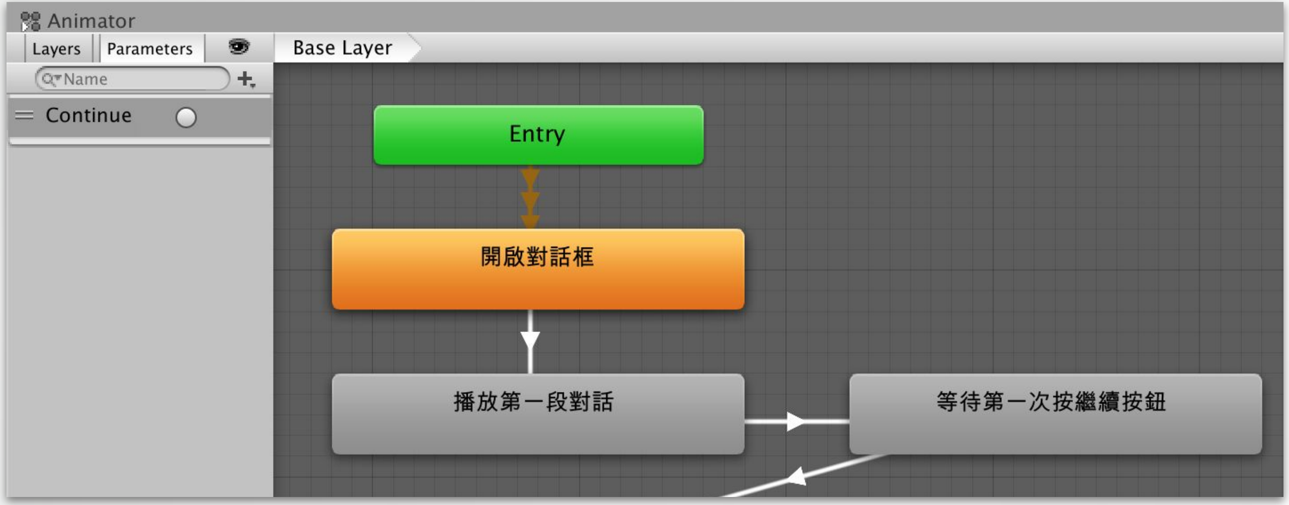
```
private IEnumerator 開啟對話框()
{
    var fromColor =
        _dialogPanel.color * new Color(1f, 1f, 1f, 0f);
    var toColor = _dialogPanel.color;
    const int length = 60;
    for (var i = 0; i < length; i++) {
        _dialogPanel.color =
            Color.Lerp(fromColor, toColor, i / (float) length);
        yield return null;
    }
    _dialogPanel.color = toColor;
}
```



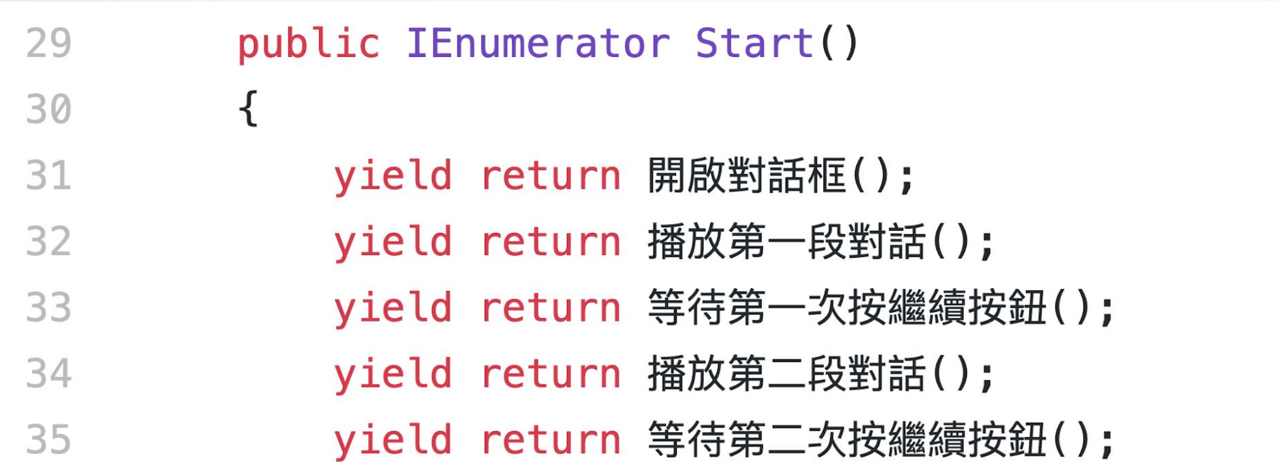
```
    public IEnumerator Start()
    {
        yield return 開啟對話框();
        yield return 播放第一段對話();
        yield return 等待第一次按繼續按鈕();
        yield return 播放第二段對話();
        yield return 等待第二次按繼續按鈕();
        yield return 播放第三段對話();
        yield return 等待第三次按繼續按鈕();
        yield return 關閉對話框();
    }
```

Yield another IEnumerator

Wait until finished



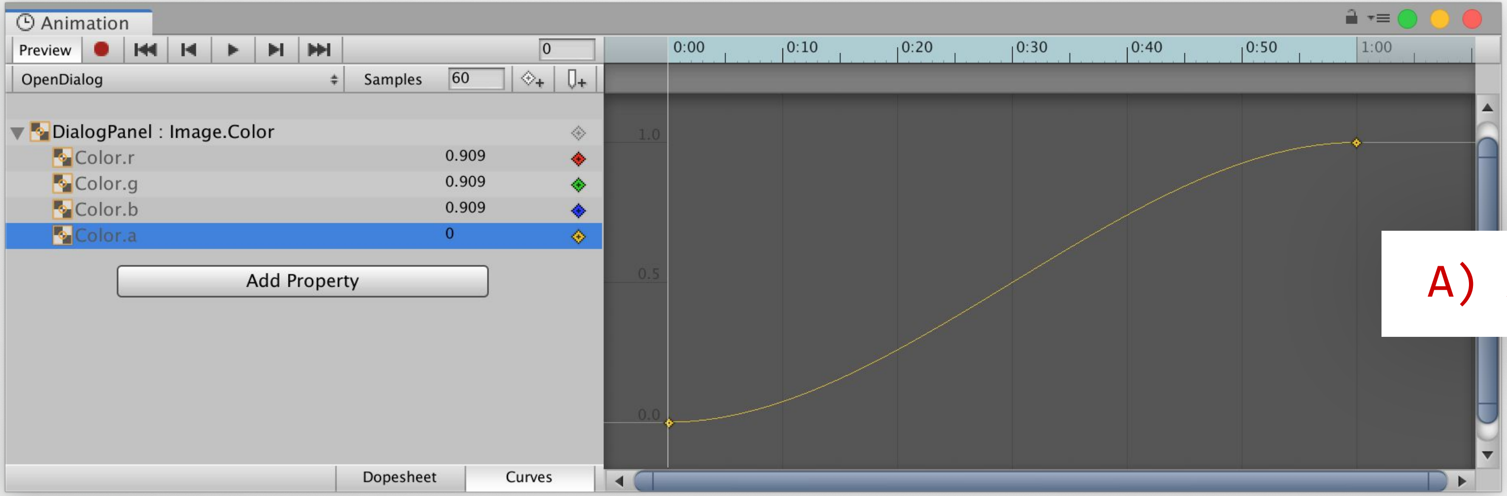
A) Animator



```
        public IEnumerator Start()
        {
            yield return 開啟對話框();
            yield return 播放第一段對話();
            yield return 等待第一次按繼續按鈕();
            yield return 播放第二段對話();
            yield return 等待第二次按繼續按鈕();
```

B) Script

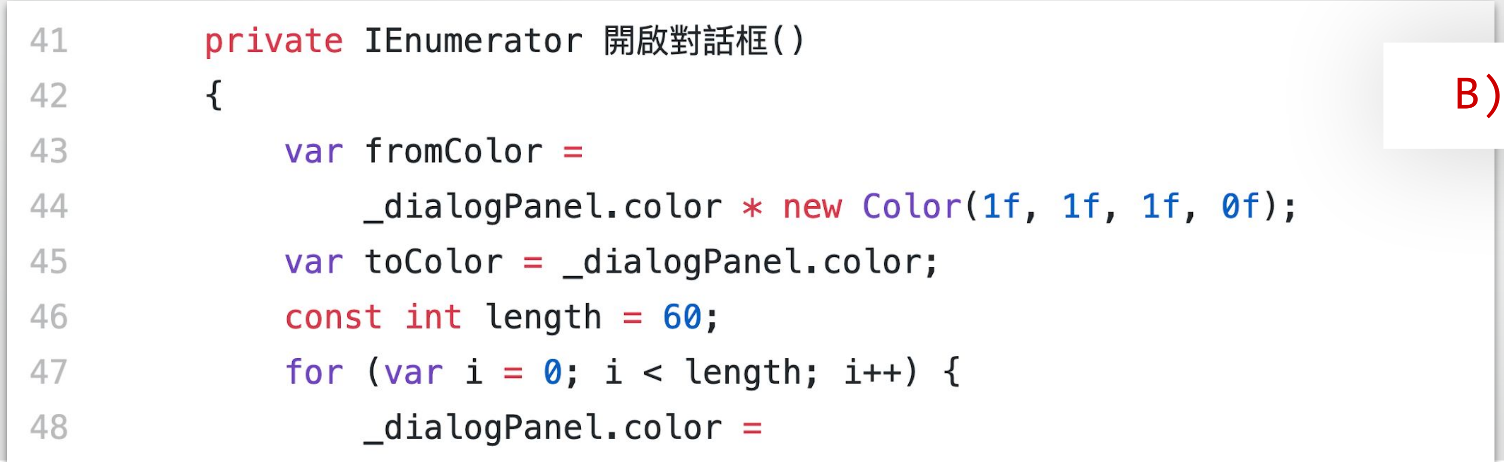
FeisStu



DialogPanel : Image.Color

Color.r	0.909
Color.g	0.909
Color.b	0.909
Color.a	0

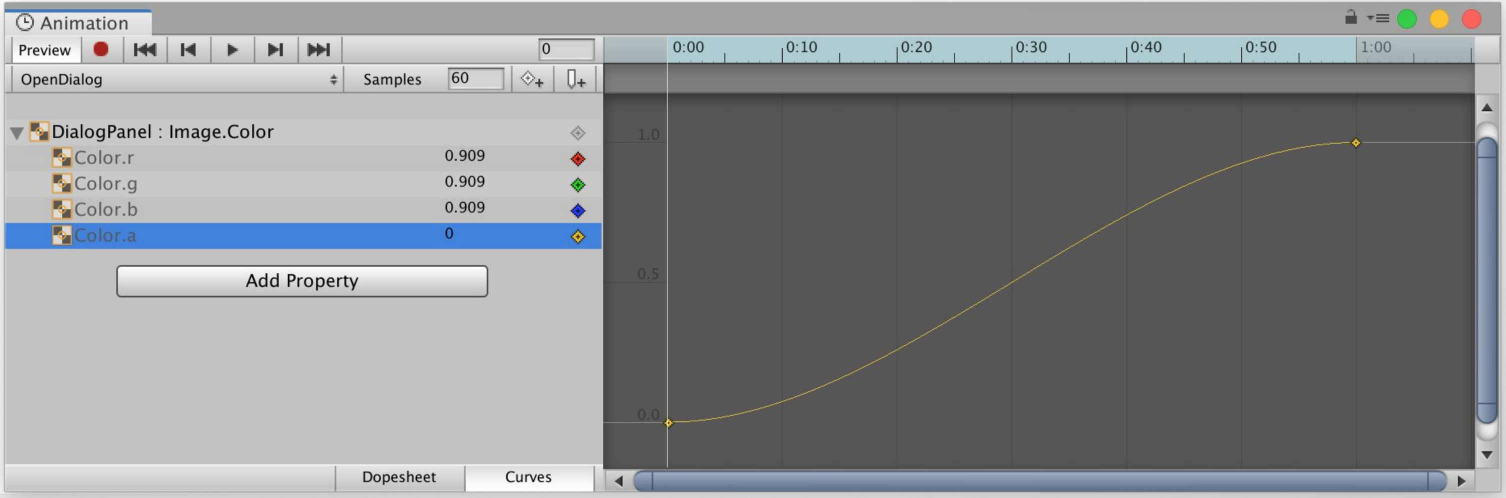
A) Animation



```
    private IEnumerator 開啟對話框()
    {
        var fromColor =
            _dialogPanel.color * new Color(1f, 1f, 1f, 0f);
        var toColor = _dialogPanel.color;
        const int length = 60;
        for (var i = 0; i < length; i++) {
            _dialogPanel.color =
```

B) Script

```
    public IEnumerator Start()
    {
        yield return 開啟對話框();
        yield return 播放第一段對話();






    }
```

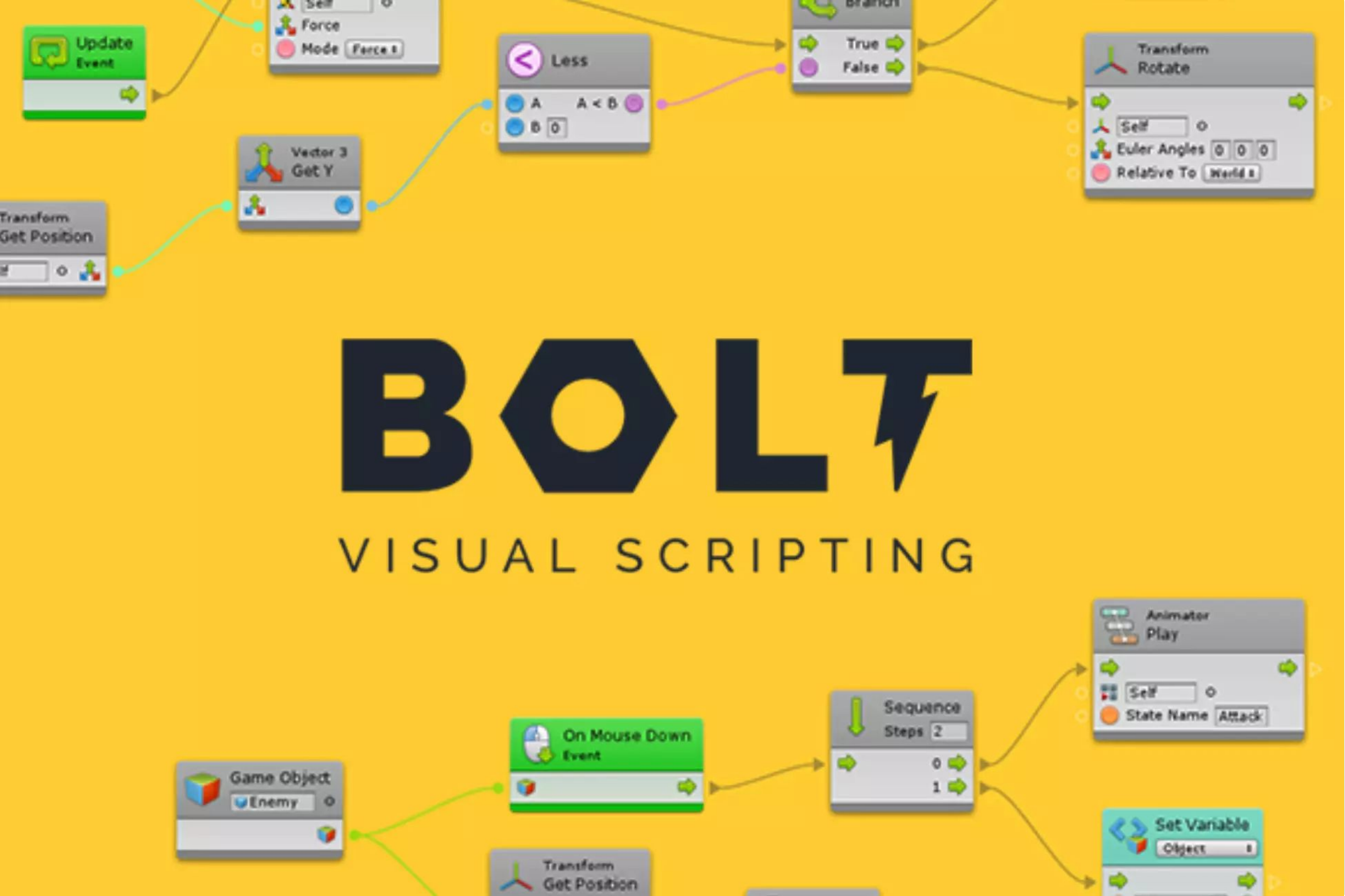


Samples 60

0

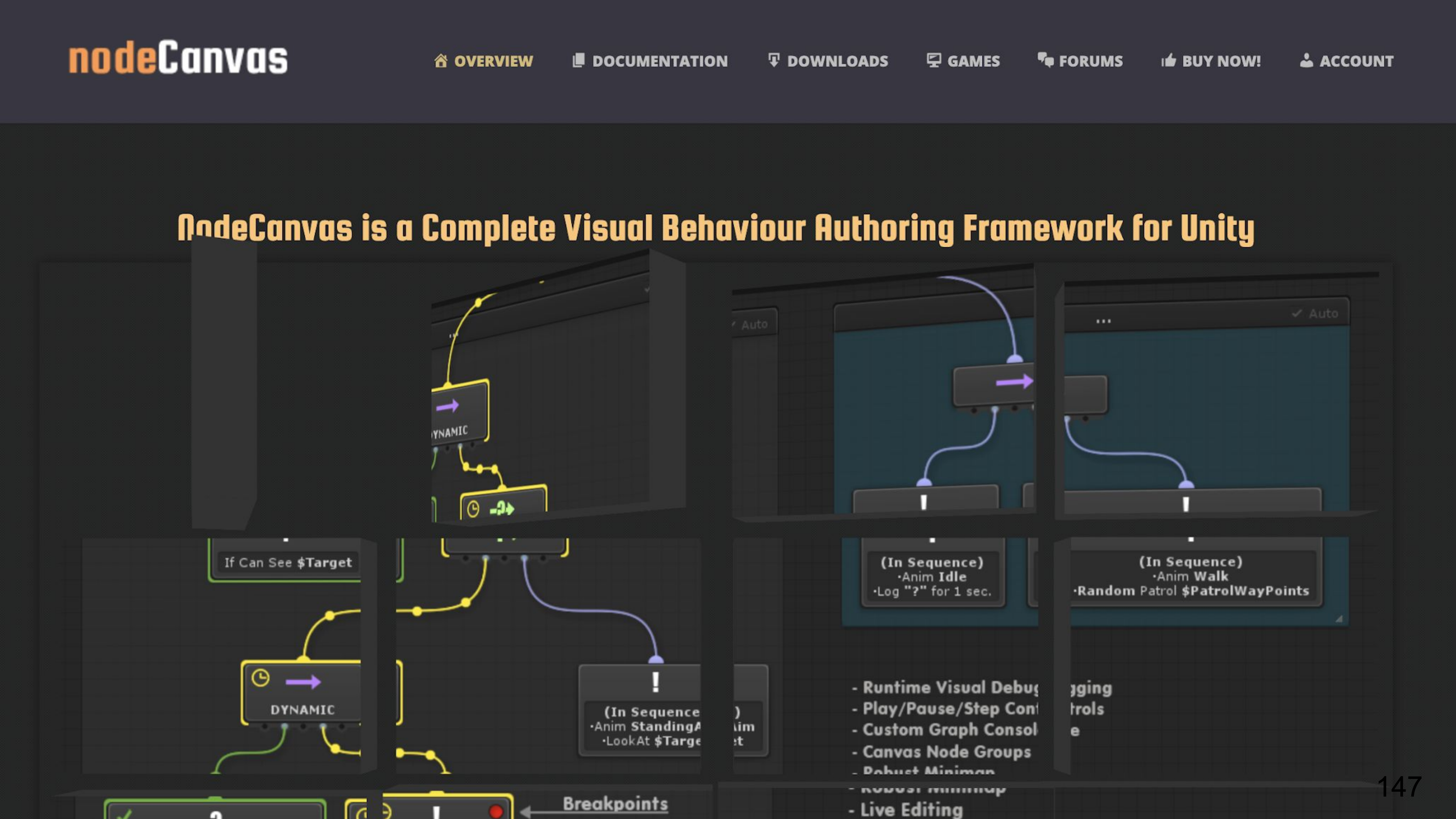
DialogPanel : Image.Color	
Color.r	0.909
Color.g	0.909
Color.b	0.909
Color.a	0

Hybrid approach ?


BOLT
VISUAL SCRIPTING
Update Event
Force
Mode
Force
Less
A
A < B
B
0
Branch
True
False
Transform Rotate
Self
Euler Angles
0
0
0
Relative To
World
Vector 3 Get Y
Transform Get Position
Animator Play
Self
State Name
Attack
Sequence
Steps
2
0
1
On Mouse Down Event
Game Object
Enemy
Set Variable
Object
Transform Get Position

Q & A

FeisStu



nodeCanvas

NodeCanvas is a Complete Visual Behaviour Authoring Framework for Unity

- Runtime Visual Debugging
- Play/Pause/Step Controls
- Custom Graph Console
- Canvas Node Groups
- Robust Minimap
- Live Editing