Produção de Jogos 4 - Apresentação

Design Patterns (Padrões de Projeto)

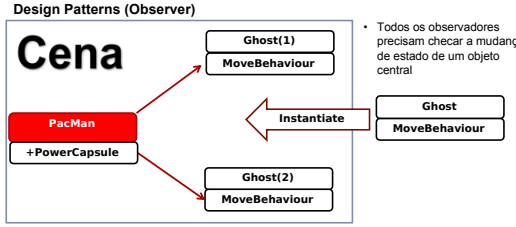
Criação	Estrutura	Comportamental
Singleton	Component	**Strategy**
Factory Method	Flyweight	**State**
Builder	Bridge	**Observer**
Abstract Factory	Decorator	Command
Prototype	Composite	Iterator
	Proxy	Mediator
	Adapter	Memento
	Facede	Interpreter
		Template Methode
		Visitor
		Chain of responsibility

Produção de Jogos 4 - Observer

Descrição: Define uma dependência **um-para-muitos** entre objetos de modo que, quando **um objeto muda de estado** , **todos** os seus dependentes **são notificados** e atualizados automaticamente.
Uso: (Um avisa muitos) **Um objeto central deve "avisar" outros objetos** de algo ou chamar uma função nesses objetos.
Ex.: Os fantasmas do Pac-man são observadores, quando o player come uma capsula de força, todos devem ser avisados para mudar seu comportamento (fugir). Quando o efeito termina mudar para perseguir novamente. Evento de elementos de interface (botões) possuem Listener.
Prática:
Crie uma interface para o notificador (Subject) com métodos para registar, excluir e notificar os Observadores.
Crie uma interface para o observador (Observer) com método para receber a mensagem.
Crie uma classe concreta notificador onde há uma lista de Observadores e métodos para adicionar, remover e atribuir dados. O Subject deve percorre a lista de Observadores chamado o método update de todos sempre a alterar seu estado.

Produção de Jogos 4 - Observer

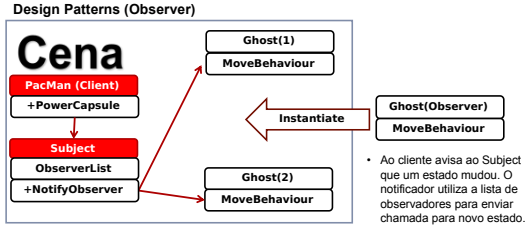
Design Patterns (Observer)



- Todos os observadores precisam checar a mudança de estado de um objeto central

Produção de Jogos 4 - Observer

Design Patterns (Observer)



- Ao cliente avisa ao Subject que um estado mudou. O notificador utiliza a lista de observadores para enviar chamada para novo estado.

Produção de Jogos 4 - Observer

Design Patterns (Observer)

ObserverInterface.cs

```
using UnityEngine;

public interface ObserverInterface {
    public void NotifyObserver();
}
```

ObserverNPC.cs

```
using UnityEngine;

public class ObserverNPC : MonoBehaviour, ObserverInterface {
    void Start(){
        SubjectPlayer.instance.AddObserver(this);
    }

    public void NotifyObserver(){
        Debug.Log("Avisado:" + gameObject.name);
    }
}
```

Produção de Jogos 4 - Observer

Design Patterns (Observer)

Player.cs

```
using UnityEngine;

public class Player : MonoBehaviour {

    void Update() {
        if (Input.GetKeyDown(KeyCode.Space)){
            SubjectPlayer.instance.NotifyObservers();
        }
    }
}
```

SubjectPlayer.cs

```
using System.Collections.Generic;
using UnityEngine;

public class SubjectPlayer : MonoBehaviour
{
    public static SubjectPlayer instance;
    public List<ObserverInterface> list;

    void Awake(){
        instance = this;
        list = new List<ObserverInterface>();
    }

    public void AddObserver(ObserverInterface obs){
        list.Add(obs);
    }

    public void NotifyObservers(){
        foreach(ObserverInterface obs in list){
            obs.NotifyObserver();
        }
    }
}
```

Programação Funcional

Action, Function, Delegate

using System;

Action<T1, ... T12>:

Function<T1,... T10, R>

Delegate void Prototipo();
Prototipo myDelegate

```
using System;

public class Program {

        public static Action<string> myAction;
        public static void Main(){
                myAction = MyMethod;
                myAction("teste");
        }

        public static void MyMethod(string s){
                Console.WriteLine(s);
        }
}
```

Programação Funcional

Action, Function, Delegate

using System;

Action<T1, ... T12>:

Function<T1,... T10, R>

Delegate void Prototipo();
Prototipo myDelegate

```
using System;

public class Program {

        public static Func<string, int> myAction;
        public static void Main(){
                myAction = MyMethod;

        Console.WriteLine(myAction("dois"));
        }
        public static int MyMethod(string s){
                if (s == "um") return 1;
                if (s == "dois") return 2;
                return 0;
        }
}
```

Programação Funcional

Observer com Delegate

SubjectPlayer.cs

```
using UnityEngine;

public class SubjectPlayer : MonoBehaviour {
    public static SubjectPlayer instance;
    public delegate void MyDelegate();
    public MyDelegate notify;

    void Awake(){
        instance = this;
    }

    public void NotifyObservers(){
        notify();
    }
}
```

Player.cs

```
using UnityEngine;

public class Player : MonoBehaviour {

    void Update() {
        if (Input.GetKeyDown(KeyCode.Space)){
            SubjectPlayer.instance.NotifyObservers();
        }
    }
}
```

ObserverNPC.cs

```
using UnityEngine;

public class ObserverNPC : MonoBehaviour, ObserverInterface {
    void Start(){
        SubjectPlayer.instance.notify += NotifyObserver;
    }

    public void NotifyObserver(){
        Debug.Log("Avisado:" + gameObject.name);
    }
}
```

SendMessage / BroadCast

SendMessage

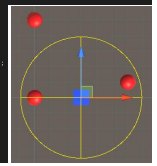
Envia mensagem para uma lista de objetos

ObjBroadCastMessage.cs

```
using UnityEngine;

public class ObjBroadCastMessage : MonoBehaviour {
    void ApplyDamage(float damage) {
        Debug.Log(gameObject.name + " " + damage);
    }
}
```

PlayerObserver.cs

```
using UnityEngine;

public class PlayerObserver : MonoBehaviour {

    public LayerMask layer;
    public float radiuns;

    void Update() {
        if (Input.GetKeyDown(KeyCode.Space)){
            Collider[] enemies = Physics.OverlapSphere(transform.position, radiuns, layer);
            foreach(Collider c in enemies){
                c.SendMessage("SetDamage");
            }
        }
    }

    void OnDrawGizmos() {
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(transform.position, radiuns);
    }
}
```



SendMessage / BroadCast

BroadcastMessage

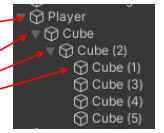
Propaga chamada de método em todos os filhos de um objeto

PlayerObserver.cs

```
using UnityEngine;

public class PlayerObserver : MonoBehaviour {

    void Start(){
        BroadcastMessage("ApplyDamage", 5.0f);
    }
}
```

ObjBroadCastMessage.cs

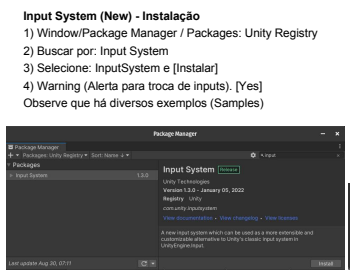
```
using UnityEngine;

public class ObjBroadCastMessage : MonoBehaviour {

    void ApplyDamage(float damage) {
        Debug.Log(gameObject.name + " " + damage);
    }
}
```

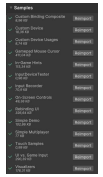


Novo Input Systen

Input System (New) - Instalação

1) Window/Package Manager / Packages: Unity Registry
2) Buscar por: Input System
3) Selecione: InputSystem e [Instalar]
4) Warning (Alerta para troca de inputs). [Yes]
Observe que há diversos exemplos (Samples)





Novo Input Systen

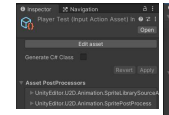
Input System (New) - Instalação

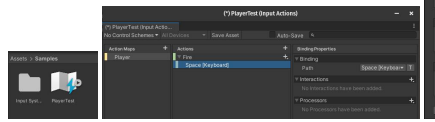
- Action Maps
- Actions
- Binding
- Properties: Binding / Interactions / Processor
- Player Input (Component)
  - Behavior: SendMessage / BroadcastMessage / Invoke Unity / Invoke C#
  - Divice Lost
  - Device Regained
  - Controle Changed
  - Events
- Modo 1 - Via Player Input - Callbacks
- Modo 2 - Via Script - Context, Delegate, Lambda Function, Programação Funcional
- Alterar o Esquema atual
- Rebainding – Mapear teclas

Novo Input Systen

Input System (New) - MODO 1

1) Na pasta Assets / Criar Input Actions
2) Editar
3) Criar
   - Actions Maps (organizar inputs)
   - Actions (ações e vincular teclas)
   - Binding properties (mapear)
4) Adicione o componente (Player Input) e atribua a Callback

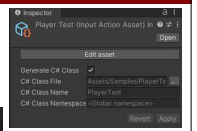




Novo Input Systen

Input System (New) - MODO 2

1) Clique no InputAction
Marque [Generate C# Class] e clique em [Apply]
2) Será criado um arquivo Input.Actions.cs
Não é preciso editor ou entender o conteúdo do arquivo



PlayerNewInput.cs

```
Assets > Samples > PlayerNewInput.cs > ...
1   using UnityEngine;
2   using UnityEngine.InputSystem;
3
4   public class InputNewInput : MonoBehaviour {
5
6       //Não é necessário ter o component PlayerInput
7       void Start(){
8           PlayerTest teste = new PlayerTest();  //Utilize a classe que foi criada acima
9           teste.Player.Enable();   //Habilite as Callbacks
10          teste.Player.Fire.performed += Fire;   //adicione com delegate
11      }
12      public void Fire(InputAction.CallbackContext context){
13          //controla a fase do contexto que podem ser: started, performed, canceled
14          if (context.performed)
15              Debug.Log("Jump: " + context.phase);
16      }
17  }
```

Novo Input Systen

Input System (New) - MODO 3

```
using UnityEngine;
using UnityEngine.InputSystem;
public class Player : MonoBehaviour {
    PlayerControls constrols;
    public float horizontal;
    void Awake() {
        controls = new PlayerControls();
        controls.GamePlay.Fire.performed += ctx => Fire();
        controls.GamePlay.Move.performed += ctx => horizontal = ctx.ReadValue<float>();
        controls.GamePlay.Move.canceled += ctx => horizontal = 0;
    }
    void OnEnable(){
        controls.GamePlay.Enable();
    }
    void Update(){
        transform.position += Vector3.left   * horizontal  * 6.0f * Time.deltaTime;
    }
    void Fire(){
        Debug.Log("Fire");
    }
    void OnDisable(){
        constrols.GamePlay.Disable();
    }
}
```

Novo Input Systen

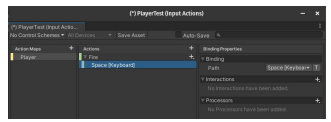
Rebinding

```
playerInputAction.Player.Disable();
playerInputAction.Player.Fire.PerformInteractiveRebinding()
    .OnComplete(callback => { Debug.Log(callback);
    callback.Dispose;
     playerInputAction.Player.Enable();
}).Start();
```

Novo Input Systen

Esquemas

No canto superior esquedo. <No Control Scheme>. Escolha o tipo de esquema.
Add Control Scheme... Keyboard – Salve. Assim o esquema atual será Teclado. Maque para cada binding Use Keyboard.
Podem ser adicionados mais esquemas para a mesma ação aplicando binds diferentes. Por exemplo: GamePad e configurar outra um botão para mesma ação.
Ao conectar o controle o esquema será alterado.



Novo Input Systen

Esquemas

```
if (esquema == 0){
    playerInput.SwitchCurrentActionMap( “Player”);  //”player”, etc...
    playerInputAction.Player.Enable();
    playerInputAction.UI.Disable();
} else if (esquema == 1){
    playerInput.SwitchCurrentActionMap( “UI”);  //usa “UI” no lugar do esquema do player
    playerInputAction.Player.Disable();
    playerInputAction.UI.Enable();   //deixa de mexer com o player e mexe com interface

}
```