Motor de Jogos e Arquitetura

Input system e design patterns

Slides por:
Gustavo Ferreira Ceccon (gustavo.ceccon@usp.br)

Este material é uma criação do
Time de Ensino de Desenvolvimento de Jogos Eletrônicos (TEDJE)
Filiado ao grupo de cultura e extensão Fellowship of the Game (FoG), vinculado ao ICMC - USP

Objetivos

- ➔ Mostrar diferentes formas de input system
- ➔ Como usar o Input da Unity
- ➔ O que são design patterns e como usar
- ➔ Como buildar seu jogo

Índice

1. Input
2. Design Patterns

1. Input

1. Input

- ➔ Como receber eventos e processar eventos?
  - ◆ O sistema operacional oferece diferentes formas de suporte aos eventos, geralmente uma das duas formas:
    - Callbacks e event listeners
    - Event polling e input states

1. Input

- Callbacks e event listeners
  - Implementação de funções especializadas em receber certos parâmetros
    - Keyboard, mouse, joystick
  - O sistema operacional chama todas as funções implementadas “cadastradas”
  - Sistemas operacionais mobile geralmente implementam essa arquitetura (Android, Java ME)

1. Input

Exemplo Android

1. Input

- Event polling e input states
  - São armazenadas mensagens ou eventos pelo sistema operacional, gerado pelas interrupções
  - Quando necessário processar o evento, é dado um poll que retorna a lista de eventos que aconteceu desde o último poll
  - Também é possível implementar um buffer de estados que é dado um reset todo poll
    - Acessível através de funções e membros estáticos

1. Input

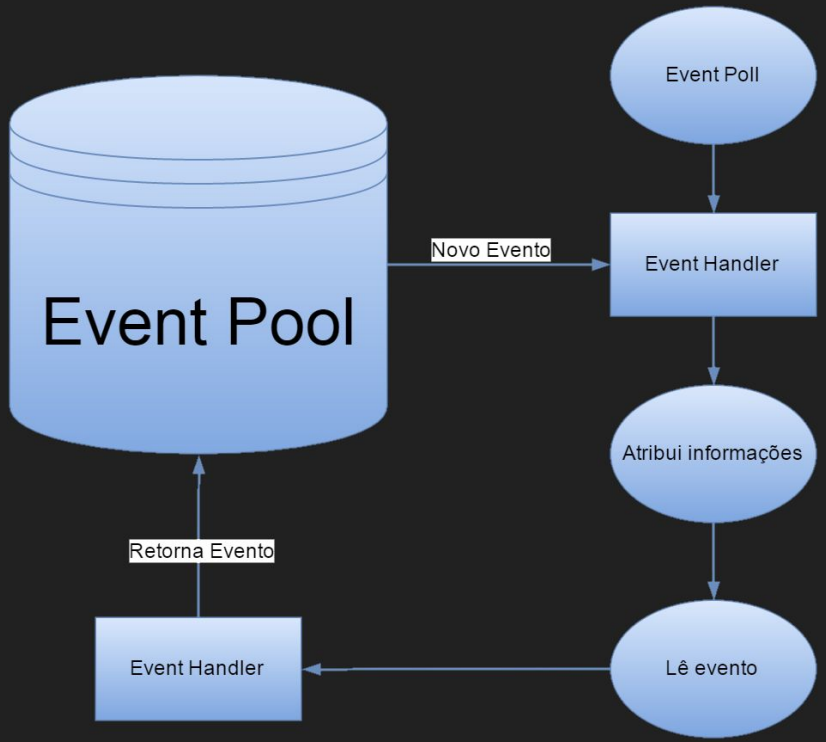
Exemplo SDL

Exemplo Unity

1. Input

- Event pool
  - Pool ≠ Poll
  - Parecido com thread pool, você cria uma piscina de objetos que representam eventos, para diminuir a quantidade de alocação toda vez que processar os eventos

1. Input


Event Poll
Event Pool
Novo Evento
Event Handler
Atribui informações
Retorna Evento
Event Handler
Lê evento

1. Input

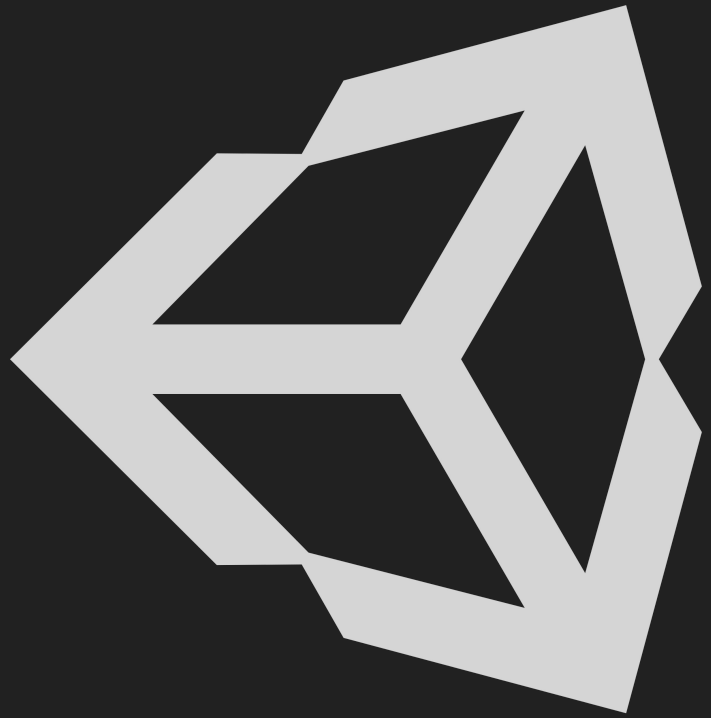
- Input Manager (antigo)
  - É horrível (e todo mundo concorda)
  - Axis são usados para representar entradas de ponto flutuante
  - Tudo é mapeado através de “nomes” na configuração do projeto, incluindo joystick
  - Input System, input refeito

1. Input

➔ Controle (Joystick)

- Existem diferentes plugins para fazer o bind de controles, já que o original da Unity não funciona (direito)
- É preciso mapear os botões, identificar eventos (conexão) etc.
- Depende do modelo e do driver

UNITY TIME !!!! - Platformer



2. Design Patterns

2. Design Patterns

- Problemas comuns de programação resolvidos, como restrição de acesso, de criação, comportamento etc.
- Não se deve forçar um padrão sobre um problema, deve entender as aplicações e quando usar
- Geralmente voltados a problemas de programação orientada a objeto, devido a herança, escopo, polimorfismo
- Tipos: criação, estrutural, comportamental, concorrente

2. Design Patterns

- Padrões de projetos
  - Singleton (criação)
  - Observer (comportamental)
  - Flyweight (estrutural)
  - Prototype (criação)

2. Design Patterns

- Singletone
  - Exemplo: um game manager que contém informações sensíveis e que devem ser únicas. Ele controla coisas dentro de um jogo e múltiplas instâncias podem causar problemas

2. Design Patterns

- ➔ Solução
  - ◆ Deixar o construtor como privado
  - ◆ Criar uma função global/estática (acessível) que retorna a instância única
    - A instância pode ser criada sempre no início
    - A instância pode ser criada na primeira chamada
      - ○ Lazy initialization (outro padrão!)

2. Design Patterns

- Observer
  - Exemplo: uma HUD precisa saber se o jogador perdeu o não para mostrar a tela de game over, porém não é uma boa ideia chamar uma função a partir do player

2. Design Patterns

- ➔ Solução
  - Criar um evento de callback na HUD
  - Adicionar um listener do script da HUD ao observer do game manager
  - Quando o jogador perder, o game manager ativa o observer / evento, chamando a função de callback

2. Design Patterns

- Flyweight
  - Exemplo: precisamos renderizar um conjunto de objetos iguais (árvores) com o mesmo modelo e em posições diferentes

2. Design Patterns

- ➔ Solução
  - Compartilhamos o mesmo objeto de modelo e textura para a árvore e criamos apenas objetos com a referência ao mesmo
  - Criamos novas instâncias apenas para a posição da árvore

2. Design Patterns

- Prototype e Factory
  - Exemplo: precisamos criar um spawner de monstros que tem o mesmo comportamento e atributos

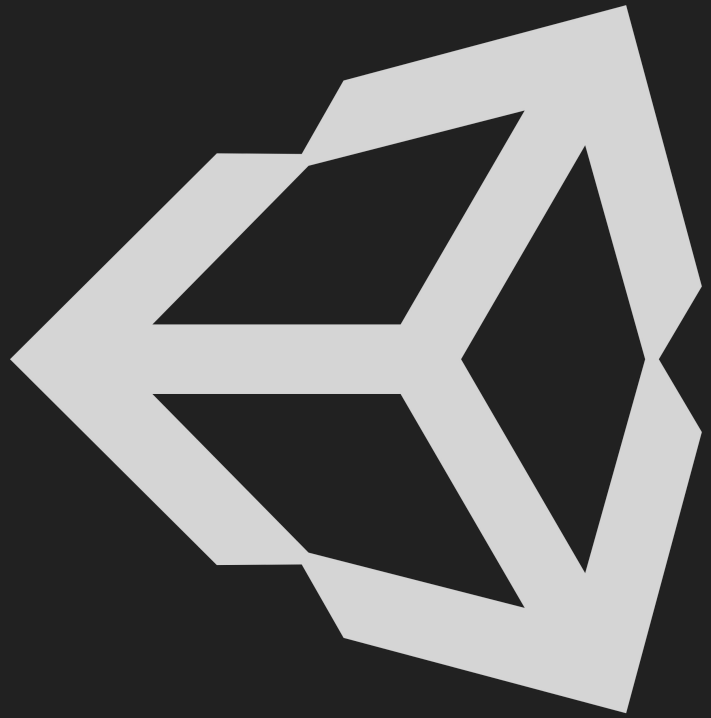
2. Design Patterns

- ➔ Solução 1 (Prototype)
  - ◆ Criamos um prefab, ou seja, um modelo do monstro
  - ◆ Criamos uma classe spawner que recria vários clones desse objeto
    - Shallow cloning
    - Deep cloning

2. Design Patterns

- Solução 2 (Factory)
  - Criamos uma fábrica, que sabe fazer apenas monstros
  - Cada vez que criarmos um monstro, pedimos para fábrica instanciar um novo objeto e recebemos uma entidade abstrata
  - Abstract Factory é mesma coisa, só que aceita implementação de diferentes fábricas

UNITY TIME !!!! - Spawner



Dúvidas?

Referências

Referências

[1] Jason Gregory-Game Engine Architecture-A K Peters (2009)
[2] Game Coding Complete, Fourth Edition (2012) - Mike McShaffry, David Graham
[3] David H. Eberly 3D Game Engine Architecture Engineering Real-Time Applications with Wild Magic The Morgan Kaufmann Series in Interactive 3D Technology 2004
[4] http://gameprogrammingpatterns.com/
[5] http://gafferongames.com/
[6] http://docs.unity3d.com/Manual/index.html
[7] http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/
[8] https://en.wikipedia.org/wiki/Software_design_pattern
[9] https://www.youtube.com/user/BSVino/videos
[10] https://www.youtube.com/user/thebennybox/videos
[11] https://www.youtube.com/user/GameEngineArchitects/videos
[12] https://www.youtube.com/user/Cercopithecan/videos
[13] http://www.glfw.org/docs/latest/input_guide.html
[14] http://lazyfoo.net/tutorials/SDL/index.php
[15]
[16]
[17]