Introduction au Pattern Entity-System

Cours Serious Games

November 18, 2015

Introduction

Objectif:

- Etude du pattern design Entity System
- Exemple de framework EntitySystem
- Discussion sur une méthode de traduction des acteurs vers EntitySystem

Contexte

- Phase de conception GDD => OK
- Choix des technologies du moteur de jeu
- Choix des pratiques/patterns
- Phase de prototypage informatique
- Modélisation des acteurs du jeu en "game objects"
- Ajout de gameobjects utiles mais pas visibles...

Première approche avec l'OO

- Concepts de base de l'OO
  - Class
  - Attribut
  - Operation
  - Objet
  - Héritage entre classes

Difficultés avec l'OO...

- Problème avec les longues chaines d'héritage ...
- Ambiguïté des opérations (pb de l'héritage multiple)
- Difficulté à définir clairement l'état du processus (au sens OS) du jeu : pb de la sauvegarde
- Réutilisation inter projets de jeux

Le design pattern Entity/System

- Ses principes
  - Data-Driven
  - Composition over Inheritance
- Pourquoi?
  - Flexibilité dans la définition des acteurs
  - Réutilisation logicielle
  - Meilleur contrôle des dynamiques et des interactions
- Prix: Performance

Concepts de base

- Entity
  - Identifiable
  - Un simple "Container"
  - Peut définir une hiérarchie: sous entités
- Component
  - Contient les données
  - Comportements associés aux données
  - Ajouter/Retirer (dynamiquement) à des entités

Concepts de base v2

- Introduction du concept de system/processor
- Entity
- Component
  - Contient uniquement des données !
- System/Processor
  - Code lié à un aspect particulier
  - Chaque système est "indépendant"
  - Itérer sur les entités qui possèdent les composants liés à l'aspect traité

Simple implémentation de d'E/S

- Entity
  - id:Integer
  - components:List
- ComponentType: Enum
- EntityManager
  - ComponentType->[Entity]
- Component
  - type: ComponentType
  - data:Map
- System/Processor
  - onUpdate()

Amélioration 1

- Amélioration: Communication inter systemes
- Interaction entre systèmes
  - Modification des données des components
  - Avec des primitives de communication
  - Pattern: Observer
    - Enregistrement d'un système auprès d'une source d'événement
    - Déclenchement d'un événement
    - Handle de l'événement pas les systèmes
  - Pattern: Message passing
    - Envoyer explicitement un message
    - Destinataire du message Délégué Component Système

Amélioration 2

- Amélioration: Machine à états
- Les composants sont ajoutés/retirés avec une FSM
- L'entité sera dynamiquement traitée par des systèmes différents selon son état
- Gestion facile des facettes (aspects différents) d'une entité !

Implémentations

Implémentations:

- Moteur de jeux
- Unity
- Unreal Engine (avec héritage entre components)
- Ash (JS)
- Artemis (Java)
- ECS (Python)
- QTEntity(C++)

Exemple Version Simplifiée d'Asteroid

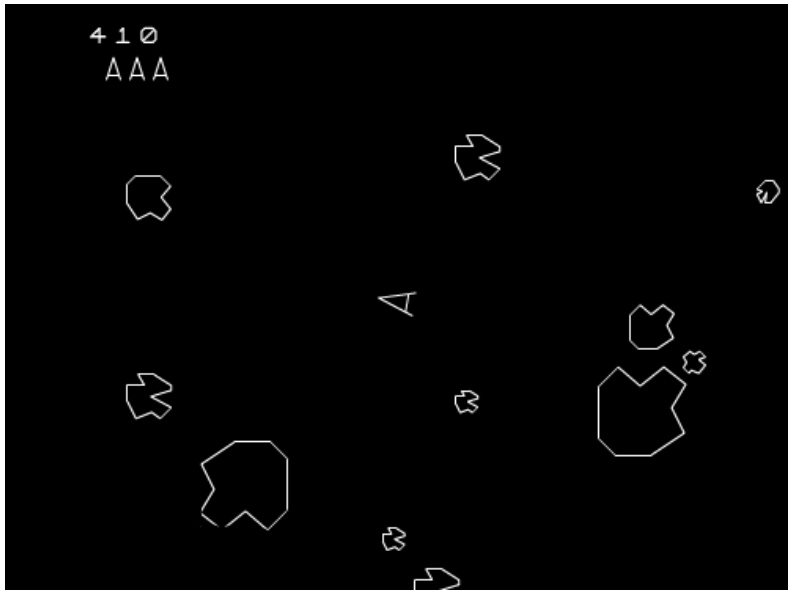

Figure : Asteroid Game

Components

- Components
  - GameStateComponent: lives, level, hits, isPlaying
  - Position: point rotation
  - Collision: radius
  - DeathLag: time
  - Display: viewToDisplay
  - Motion: velocity, angularVelocity
  - MotionControl: left, right, acceleration
  - AnimationComponent
  - AsteroidComponent: fsm
    - alive -> Motion, Collision, Display
    - destroyed -> DeathLag, Animation
  - SpaceShipComponent fsm
    - playing -> Motion, MotionControl, Collision, Display
    - destroyed -> Display, DeathLag, Animation
  - HudComponent: hudView
  - Audio sound
  - WaitForClickComponent

Exemple: les entités

- Entités
  - AsteroidEntity
    - AsteroidComponent
    - Position
    - Audio
  - SpaceshipEntity
    - SpaceshipComponent
    - Position
    - Audio
  - GameEntity
    - GameState
    - Hud
    - Display
    - Position

Exemple: les systèmes

- Systèmes
  - AudioSystem
  - CollisionSystem
  - DeathLagSystem
  - HudSystem
  - MotionControlSystem
  - MovementSystem
  - RenderSystem
  - GameDynamicsSystem