n8 Game Framework Design Document

Nate Ebel

March 9, 2014

Contents

1 Current Ideas and Things to Think About

- Using a more "pure" entity-component system with systems operating on components and the "entity" or game object just acts as an id to relate components loosely to one another
- Systems can be treated as services that are globally available through a service locator
- ... more ideas and full new design for ECS in notes

2 Design and Use Patterns

2.1 Command

2.2 Observer

2.3 Subclass Sandbox

2.4 Singleton

2.5 Service Locator

2.6 Components

3 Client Use/Interaction

Client should instantiate a single instance of a Game object.

4 Resource Management

5 Game Loop

Takes place within Game and uses the Game object's GameTimer instance to control the loop. Within the loop the current GameState is processed at each iteration.

6 Window Management

Should be a window object. This object should be stored by the Game class. Should support renaming and resizing.

7 Game States

A GameState is whatever is happening in the forefront of the game. GameStates will be maintained on a stack within a GameStateManger. GameStateManager acts as a bridge between the game loop within Game and the currently active GameState. Should support passing of entities between states, and each GameState will maintain a list of its entities. GameStates will include 3 key methods of responding to user input, updating, and rendering. The Update method will receive the current time as an argument, and Render will receive the current window canvas as an argument. These 3 methods will be called in the game loop within the Game object. Pause and resume methods can be overridden to handle changes to and from the GameState.

8 Entity-Component System

8.1 Entities

Entities will be constructed using an entity-component system. Entity will be a general class that can store different components. Different behaviors will then come from which components an entity has. Entity will have a broadcast method to send messages between components.

8.2 Components

Functionality can be added to entities through components. A component could be a position, a sprite, player information, or anything else that helps differentiate an entity or added to an entity's behavior. Components should take a GameObject pointer as an argument to its methods so s single instance of a component could potentially be used for many entities if the component is stateless.

8.2.1 Component Communication

8.2.1.1 Shared State Entity data that is cross-domain and will be used be multiple components should just be maintained in the entity class. This is a good candidate for data such as x and y position.

8.2.1.2 Direct Reference Components that are closely related could store direct references to one another. For now, this approach is not supported.

8.2.1.3 Messaging Components can broadcast and receive messages to one another. A component can then be wired to handle a particular message type. For example, physics component could detect and collision and broadcast a collision event which the audio component handles by playing a sound and the player info component handles be decreasing the player's health. These messages could be as simple as an int or they could be a message object containing more information.

9 Input

Input will be handled by an central input manager that tracks which keys have been pressed down or are up. Game states can then query the input manager and define state specific behaviors for inputs.

10 Events

Events/messages between components are handled by broadcasting messages to other components. Events/messages between an entity/component and a higher level system such as GUI or achievement system will use the observer pattern.

11 Rendering

11.1 Rendering operations

- Render single entity
- Render single entity with a camera object (Camera: see Section 12)

11.2 Static sprite

11.3 Animated sprite

12 Camera

Camera can be used to center drawing operations around a specified entity. This entity will be registered with the camera as an observable object. When the camera receives a "position changed" event from the entity, it will update its position accordingly. The camera will be created with the height and width of the current game world. When the world size changes,

the camera will need to be updated. The camera will be stored within a GameState. A camera object can be passed in to an entity's render method to have it draw based on the camera's current position.

13 Operations on or using entities

AI physics

14 Game levels

15 Globally Available Services

Use globally available service locator. The service locator should support registering and unregistering of services, and retrieval of those services. Ensuring that a service is available before be used will be the job of client code.

15.1 Logging

Logging should be available to any aspect of the system through the ServiceLocator.

15.2 Audio Player

The audio service should be available to any aspect of the system through the ServiceLocator. The audio service should maintain a queue of sounds that have been signaled to play. Sounds should be played on a separate thread.

16 System wide values and enums

17 Configuration data/files

18 Use Cases

18.1 Input changes game state

18.2 Input changes player state

walking $\rightarrow$ flying