n8 Game Framework Design Document

Nate Ebel

February 26, 2014

Contents

1 Design and Use Patterns

1.1 Command

Object oriented replacement for callbacks that encapsulates a request within an object. Benefits and uses include:

- Allows easy redo/undo functionality if do and undo methods are implemented and some type of history is kept
- Can be made general enough to operate on different entities such as a "jump" command that works for both user controlled and computer controlled entities
- Can become a sandbox if many common operations are included with the base class
- If command is stateless, a single instance of the command may be used for the entire program
- Can be easily mapped to new inputs

1.2 Observer

Allows passing of information between loosely coupled components through the use of notify, and onNotify methods. Objects can be either subjects that notify others when certain events happen, or they can be observers and respond to received notifications from observed objects. Observers register an interest in an object so it may receive notifications from that object. This pattern allows dissimilar objects to know about each other. For example, a GUI object may want to update a life meter when a user entity is hit by an enemy. The user class doesn't need to know about the GUI object, it just needs to broadcast that its life total has changed.

1.3 Subclass Sandbox

1.4 Singleton

1.5 Service Locator

1.6 Components

2 Client Use/Interaction

Client should instantiate a single instance of a Game object.

3 Resource Management

4 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.

5 Window Management

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

6 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.

7 Entities

8 Components

9 Entity States

10 Input

11 Events

12 Rendering

12.1 Rendering operations

- Render single entity
- Render single entity with a camera object (Camera: see Section 13)

12.2 Static sprite

12.3 Animated sprite

13 Camera

14 Operations on or using entities

AI physics

15 Game levels

16 Globally Available Services

Possible options for commonly available services include:

- Singletons
- Service Locator system. GameObject base class could store registered services

- Subclass sandbox where common operations are defined in GameObject base class

16.1 Logging

Logging should be available to any aspect of the system.

16.2 Audio

17 Audio

18 System wide values and enums

19 Configuration data/files

20 Use Cases

20.1 Input changes game state

20.2 Input changes player state

walking → flying

20.3 Input moves player

20.4 Entity destroyed and animation is played

20.5 Timed event activated

20.6 Timed event destroyed

20.7 Entity collision

20.8 AI player must make a decision

20.9 Notify an achievement system when 5 enemies are destroyed

20.10 Remap an action's key