Exploration of Design Patterns in Game Development

Technical Work Term Report (Work Term 3)

APSC 310
University of British Columbia

Muchen He
Associate Developer, BioWare Edmonton
Student Number: 44638154
November 21, 2018

Contents

Preface & Foreword

Purpose

The purpose of this report is to explore and observer different kinds of theoretical methods nad patterns and how they're used ot develop components that make up a game.

As well to consolidate my knowledge as a whole in the industry as design patterns are software development concepts and potentially applicable elsewhere in the tech industry.

Background

(Working at EA-Bioware) (Associate Developer (Programmer) for game Anthem) (Using modern game engine Frostbite)

Scope of Coverage

(The scope of this report)(Covers many design patterns outlined in "Gang of Four")(Will not cover detailed breakdown, explaination, and specification of existing proprietary tech; i.e. extended code from the game or Frostbite engine)
The technical coverage cited in this report is limited because the co-op work term position is working in UI/UX. Thus I lack understanding in other areas of the game such as character handling, world rendering, etc. Many of the workflows I describe in this report may only apply to UI/UX development within this project.

Contributors

(Tim Gibson - senior software lead, manager: contribution by supervising the report and giving advice)

Summary

List of Figures

List of Tables

1 Introduction

This enters the discussion portion of the report. We will analyze the current processes in game development for UI/UX at BioWare. Then look at some of the more theoretical, templated solutions known as design patterns in later sections and draw comparisons. We will also look into how the design patterns would be used in a general sense in game development (not specific to BioWare or Bioware's current project, Anthem).

1.1 Game Development Overview

A major portion of the video game market consists of AAA (Triple-A) video games. These are titles with very high production value, large budget, and typically consists of a team of hundreds of developers. These developers consist of artists, scripters, programmers, managers, etc. Many of these roles are further grouped into sub-teams such as rendering, physics/simulation, AI, UI/UX, sound design, music production, networking, etc. Individual developers are specialized to work on one part of the game, with the exception of leads, managers, and other executives.

1.1.1 Game Engine

Developers will use a game engine so that many people can work on the project at once. The game engine is generally highly optimized for graphics rendering, physics simulation (such as collision detection), and object handling.

The Frostbite engine is the game engine widely adoped at Electronic Arts (EA) studios. It was originally developed by Digital Illusions CE (DICE).[1][2] As expected, the game engine is a piece of software, thus it is prone to software development problems that design patterns are ought to solve.

1.2 Game Building

Programmers make primitive entities such as an abstract UI widget, as well as the system that manages these entities.

The scripters and artists then use these entities in *schematics*, which are blocks of game content that contains logic and output, given some input.

These schematics are in game levels, UI widgets, and prefabricated logic blocks (LogicPrefab). Thousands of these make up functionalities in a game, and are all handled by the game engine.

1.2.1 Role of Programmers

At BioWare, most game programmers work with code (in C++ and C#). Programmers are specialized to develop in-game entities, or tools for scripters, designers, and artists. Programmers also integrate systems together, such as making mouse inputs interact with hitzones widgets, which interacts with graphical widgest or animation timelines.[1]

1.3 Design Pattern

Design patterns are reusable, general, abstract solutions to common problems in software developent.[3] Design patterns are meant to be used when there no problems actually exist – doing so would cause excess undesired complexity.

[1] Animation timelines are timeline that describe how a particular object will animate. Example: a fade in.

1.3.1 Antipatterns

Software written without care are difficult to maintain, prone to bugs and errors, and hard to collaborate. Software written with these "hacks" are known as *antipatterns*, where there is more negative consequences than the benefits of its solution. Antipatterns are counter parts to correctly implemented design patterns.[4]

Avoid antipatterns as much as possible. Exploring the negative effects of antipatterns is not part of the scope of this report.

1.4 Gang of Four

In software development, the phrase "Gang of Four" refers to the four authors of the Design Patterns book. The book features the most common design patterns widely in use today.

2 Creational Design Patterns

Creational design patterns involves with object creation.[5] This is especially useful in game development, such as when we need to create many trees or bushes in a level of a game. These patterns become more useful when the objects we're trying to create have relationships, such as *inheritance*, to other objects.

2.1 Builder

In object oriented programming, the construction of an instance of an object takes parameters. It is possible that we need a lot of parameters to satisfy more variants of the class. This introduces the telescoping constructor antipattern,[6] where there are numerous variations of constructor methods all delegate to the default constructor. The problem is having this many constructor methods with slight variations is hard to maintain and difficult to use. When creating a new instance (invoking the constructor), it's hard to tell what the actual paramters are if they have the same type.

The builder pattern encapsulates the parameters into a single structure. The structure itself is then passed into the constructor method, where we can access the data inside the structure again. The members of the structure can be accessed and modified by name, thereby eliminating the ambiguity of telescoping constructor.

This is useful in games. For example, a customizable character would have many parameters to be varied (Figure 1).

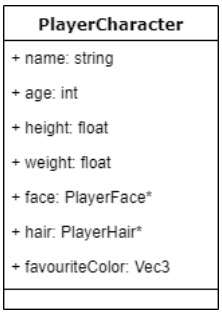

Figure 1: PlayerCharacter class with various desired parameters

2.2 Factories

2.3 Object Pool

2.4 Prototype

2.5 Singleton

In a way, they're just glorified global variable.

3 Behaviour Design Patterns

4 Structural Design Patterns

5 Architectural Patterns

5.1 Entity-Component-System

5.1.1 Composition over Inheritance

5.2 Model-View-Controller

Mostly used in UI development.

6 Conclusions

References

[1] "Frostbite." https://www.ea.com/frostbite, 2018. Online; accessed 2018.

[2] "Frostbite (game engine)." https://en.wikipedia.org/wiki/Frostbite_%28game_engine%29, 2018. Online; accessed 2018.

[3] SourceMaking, "Design Patterns." https://sourcemaking.com/design_patterns, 2018. Online; accessed Nov 2018.

[4] SourceMaking, "AntiPatterns." https://sourcemaking.com/antipatterns, 2018. Online; accessed Nov 2018.

[5] SourceMaking, "Creational Patterns." https://sourcemaking.com/creational_patterns, 2018. Online; accessed Nov 2018.

[6] "The Telescoping Constructor (Anti)Pattern." http://www.captaindebug.com/2011/05/telescoping-constructor-antipattern.html#.W_XGGUxFwis, May 21 2011. Online; accessed Nov 2018.

A Sample Code