

Aspect Oriented Programming and Game Development

Ernest Criss
Department of Information and Computer Sciences
University of Hawaii at Manoa
Honolulu, HI 96822
criss@hawaii.edu

Abstract

Aspect Oriented Programming could possibly be the next phase in the evolution of game development. This case-study will attempt to explore this new and exciting programming paradigm that promises to be as significant to the software development world as Object Oriented Programming. We will explore utilizing AOP facilities by extending an existing game called Invaders[1]—a game that bares resemblance to the classic Space Invaders[2].

1 Introduction

The development of video games have undergone significant changes over the years. Each change brought about improvements that empowered developers to create larger, more sophisticated, higher quality games. The earliest games, written in assembly language, were complex and difficult to maintain, yet were limited in functionality. As processing power increased so did the proliferation of the use of high level languages, namely the C programming language, in video game development. With C, code was much more humanly readable and with the ability to separate concerns into modular elements, gave way to more manageable, reusable software.

One of the most significant examples of the usefulness of reusable modules in video game development is the OpenGL Application Programming Interface (API) which is used in numerous commercial quality games today. With the advent of the Object Oriented Programming (OOP) paradigm, reusability and manageability was taken to whole new level. OOP allowed game specific concerns such as: collision detection, resource management and components that handle artificial intelligence to be encapsulated into separate entities called classes.

AOP offers a means of extending an existing game by plugging in additional behaviors as well as replacing existing behaviors without modifications to the source code. This capability has some very powerful applications. Some example applications are game patches. Game patches are regularly released to provide bug fixes or introduce additional features. In this paper we provide a patch to introduce a demo mode to an existing game called Invaders. Our implementation of the demo mode puts a limit in the number of levels the player is allowed to access. The demo mode would need to meet the following requirements:

- Ability to disable when given a validation code
- Monitor the players level progression and determine the maximum level allowed by the demo has been reached
- Display the “Game Over” screen with an added overlay that informs the player how to purchase the full version of the game
- Should not incur a significant overhead

2 The Demo Aspect

To address the above requirements, we defined the following three pointcuts:

- `readActivationCode`
- `observeSceneChange`
- `endOfDemo`

```
pointcut readActivationCode(Invaders game) :
    call(void Invaders.initWorld()) && target(game);

before(Invaders game) : readActivationCode(game) {
    try {
        BufferedReader br = new BufferedReader(new FileReader("activation"));

        String code = br.readLine();
        if ( code.equals(VALID_CODE) ) {
            activated = true;
        }
    } catch ( Exception e ) {
        activated = false;
    }
}
```

Figure 1. `readActivationCode` pointcut and it’s corresponding `before` advice

2.1 Addressing requirement 1

We placed the `readActivationCode` pointcut on a call to the `initWorld` method—where all components of the game are initialized. The `before` advice attached to the `readActivationCode` pointcut reads a file named `activation` and checks to see if both the file exists and also if the code within the file is valid(for this study, we opted to keep the validation routine rudimentary). If the activation file exists and contains a valid activation code, then the demo mode is not enabled.

```

pointcut observeSceneChange(Invaders game) :
    call(void Invaders.updateWorld(*) && target(game));

after(Invaders game) : observeSceneChange(game) {
    if ( !activated && game.currentSceneIndex == LAST_ALLOWED_SCENE ) {
        game.paintWorld();
        game.gameOver();
    }
}

```

Figure 2. observeSceneChange pointcut with after advice

2.2 Addressing requirement 2

An after advice was placed on the observeSceneChange pointcut that keeps track of the currentSceneIndex. The currentSceneIndex is an integer that contains the index of the current scene. The currentSceneIndex is incremented in the updateWorld method after each a scene is cleared. The after advice compares the currentSceneIndex to the LAST_ALLOWED_SCENE introduced constant which contains the maximum allowed level the player is allowed in the demo. When the maximum allowed level is reached, the game is over.

```

pointcut endOfDemo(Invaders game) :
    call(void Invaders.paintGameOver()) && target(game);

after(Invaders game) : endOfDemo(game) {
    if ( !activated ) {
        Graphics2D g = (Graphics2D)game.strategy.getDrawGraphics();
        g.setColor(Color.YELLOW);
        g.setFont(new Font("Arial",Font.BOLD,20));
        g.drawString("Buy Invaders Now",Stage.WIDTH/2-70,100);
        g.drawString("Only 499.99",Stage.WIDTH/2-50,120);
        game.strategy.show();
    }
}

```

Figure 3. endOfDemo pointcut with after advice that displays modified “game over” message

2.3 Addressing requirement 3

The after advice attached to the endOfDemo pointcut displays a message that includes purchase information of the game. The pointcut itself is placed on an invocation of the paintGameOver method which handles the painting of the “Game Over” message which is displayed when either the game has been beaten or the player has lost.

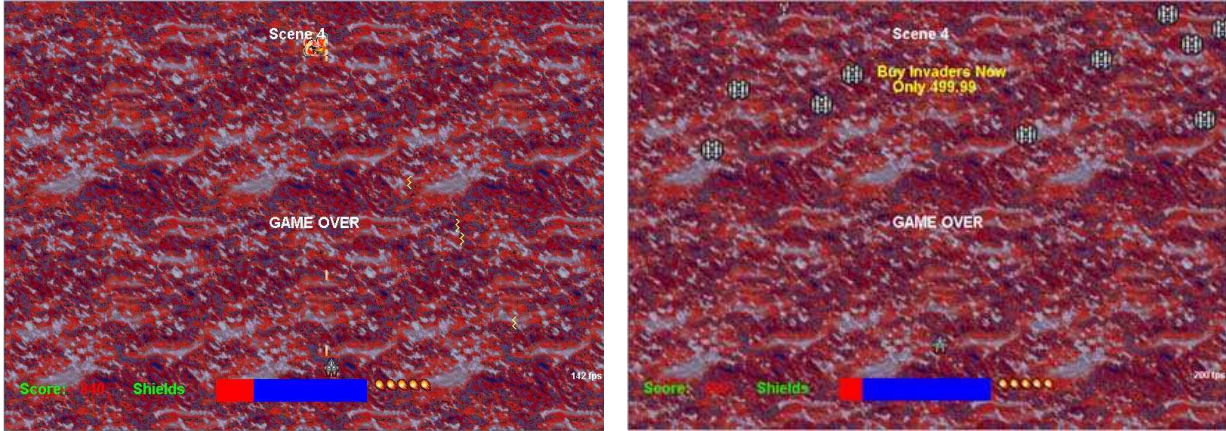


Figure 4. The normal “Game Over” screen when the demo mode is disabled (left), and the “Game Over” screen when the demo mode is enabled (right)

3 Performance

Average Frames Per Second	
With Aspect	~145 FPS
Without Aspect	~145 FPS

Figure 5. Comparison of average Frames Per Second

To determine if the fourth requirement was satisfied, we compare performance by creating another version of Invaders with the demo mode coded in. The games were then run on a computer with the following specifications:

- Windows XP Professional
- AMD Athlon XP 1400
- 728 MB DDR RAM
- ATI Radeon 7500

The performance was measured by the number frames per second that the game displays. The higher the frame rate the better. The results were fairly identical—the aspect version of the demo incurred little to no performance overhead compared to the version without aspects. This is due to the fact that Aspect/J injects bytecodes according to the portion of code specified by the pointcut and the type of advice. So the end result is practically the same from a bytecode perspective.

4 Conclusion

In this paper, we evaluated the viability of AOP within the context of game development. We have demonstrated an application of AOP by creating a game patch that provided a

demo mode without modifying the games source code. The end result provided a complete solution without incurring a significant performance overhead.

5 References

[1] Invaders – Developed by Planetalia

<http://www.planetalia.com/>

[2] Space Invaders – Developed by Taito Corporation

<http://www.taito.co.jp/eng/>