위피 개발자를 위한 아이폰 게임 애플리케이션 개발 방법론※

유현성°, 이대웅*
상명대학교
대학원 컴퓨터과학과 게임학전공 박사과정°, 소프트웨어대학 디지털미디어학부*
binarybard@naver.com, rhee219@smu.ac.kr

An iPhone Game Application Development Methodology for WIPI Developer

Hyun Seong Yu°, Dae Woong Rhee*
SangMyung University
Ph.D. Course, Dept of Computer Science, Graduate School°,
Division of Digital Media, College of Computer Software & Digital Media*

요 약

위피의 탑재 의무 폐지는 국내 위피 개발자들이 스마트폰 개발 환경으로 진출하는 계기를 만들었다. 특히 아이폰은 게임 애플리케이션의 새로운 판매 시장으로 가능성이 매우 크나, 표준적이고 명확한 게임 개발 방법론의 부재로 국내의 위피 개발자들이 진입하기에는 어려움이 있다. 그러므로 국내의 위피 개발자들이 효과적으로 아이폰 게임 애플리케이션 개발을 할 수 있는 방안이 필요하다. 본 논문에서는 위피와 아이폰의 애플리케이션 개발 방법론을 비교 분석하고, 이를 토대로 위피의 구조가 적용된 아이폰 게임 애플리케이션 개발 방법론을 제시한다. 이를 통해 국내의 위피 개발자들이 아이폰 게임 애플리케이션 개발을 용이하게 할 수 있다.

ABSTRACT

The abolition of WIPI installation duty makes domestic WIPI developers get a chance to enter into smartphone development environment. Specially, iPhone has a big possibility for a new game application market but traditional domestic WIPI developers have difficulties to get in because there are no standard game development methodology. Therefore, a method is necessary to develop iPhone game application effectively for traditional domestic WIPI developers. In this paper, we compare and analyze development methodology of WIPI and iPhone application and suggest a new methodology for iPhone application development with adaptation of WIPI architectures. Using this methodology, traditional domestic WIPI developers can develop iPhone game applications with ease.

Keyword : Smartphone, iPhone, Mobile Game Development Methodology

접수일자 : 2009년 10월 20일
심사완료 : 2009년 12월 07일
※ 이 논문은 상명대학교 부설연구기관 2009 선발과제 연구비에 의해서 수행되었음
○ 제1저자 : 유현성(binarybard@naver.com) * 교신저자 : 이대웅(rhee219@smu.ac.kr)

1. 서 론

그 동안 국내 모바일 플랫폼에서 의무 탑재가 요구되어 왔던 위피(WIPI)가 2009년 4월부터 폐지가 되었다. 이로 인해 다양한 기능을 갖춘 외산 스마트폰들이 국내에 새로운 인프라를 형성할 수 있는 계기가 마련되었다. 이로 인해 국내의 위피 개발자들도 스마트폰 애플리케이션 개발에 진입하려는 움직임이 예상된다[1].

이러한 환경 하에서 애플(Apple)의 아이폰(iPhone) 출시 및 앱스토어(App Store)의 런칭은 모바일 애플리케이션 시장에 새로운 패러다임을 제시한다. 특히 단일 플랫폼으로써의 개발 환경을 제공하여 애플리케이션 생산력을 극대화 시킬 수 있게 되었다. 이로 인하여 기존의 단말기 중심의 패러다임이 아이폰 OS와 같은 스마트폰 플랫폼 중심으로 전환되고 있다[2].

이것을 반증하듯, 아이폰은 오픈 마켓인 앱스토어를 통해 54억 달러 정도의 시장 규모를 형성함으로써 스마트폰 애플리케이션 시장에서 가장 높은 성장률을 보이고 있다. 앱스토어에서 가장 인기 있는 소프트웨어는 게임 프로그램으로, 현재 6천 5백여건에 이르고 있다[3].

이와 같은 상황에 비추어 볼 때, 국내의 위피 개발자들도 아이폰 게임 애플리케이션 시장에 진출해야 할 필요성은 매우 크다고 할 수 있다. 현재 스마트폰의 오픈마켓 중 애플 앱스토어의 규모가 가장 크고, 위피의 탑재 의무화 폐지로 인해 국내에서 위피를 주로 다루던 많은 모바일 개발자가 다른 시장으로의 진출이 필요하기 때문이다.

위피 개발자들의 아이폰 애플리케이션 개발에 진입하기 위한 채널로 대표적인 것은 애플 공식 개발자 포럼에서 제공하는 튜토리얼이다. 그러나 앱스토어에서 가장 큰 비중을 차지하는 게임 애플리케이션의 표준화된 개발 방법론에 대한 내용은 전무한 상황이다. 게임 개발의 요소라고 할 수 있는 그래픽이나 사운드, 그리고 입력 처리에 대한 내용은 개별적으로 존재하나, 그것을 게임 애플리케이션의 형태로 개발하기 위한 표준화 자료는 없다[4]. 앱스토어에는 많은 게임 애플리케이션이 등록되어 있는데, 이는 각 개발업체가 자신들만의 각기 다른 방식으로 개발을 하거나, 유료 게임 엔진을 이용한다. 이러한 개발 방법론은 기업 노하우로써, 대체적으로 공개가 되지 않았고, 아이폰 애플리케이션의 개발 방법론 또한 기본적으로 위피와는 많은 부분에서 차이점이 있다. 이로 인해 국내의 위피 개발자들이 아이폰 게임 애플리케이션 개발에 적응하는 데에 어려움이 있다.

국내 위피 개발자들이 스마트폰 애플리케이션 시장에 진입할 필요가 있는 상황에서, 아이폰을 비롯한 스마트폰 게임 애플리케이션 개발에 쉽게 적응할 수 있는 방안이 필요하다. 본 논문에서는 이에 대한 시작 연구로써, 기존의 국내 위피 플랫폼 개발자들이 아이폰 게임 애플리케이션 개발에 쉽게 접근할 수 있는 개발 방법론을 제안한다. 이를 위해 위피의 게임 개발 방법론을 분석하고 그 구조를 이용하여 아이폰에 적용시킬 수 있는 방안을 모색하고자 한다.

본 논문은 다음과 같이 구성된다. 2장에서는 아이폰의 일반적인 애플리케이션을 개발하는 방법론에 대해 기술하고, 3장에서는 위피에서 게임 애플리케이션을 개발하는 방법론에 대해 정리한다. 이를 통해 4장에서는 게임 개발적 측면에서 아이폰과 위피의 개발 방법론을 비교 정리하여, 5장에서 위피의 구조를 적용한 아이폰 게임 개발 방법론에 대해 기술한다. 6장에서는 본 논문에서 제안한 방법론을 적용하여 게임을 구현하고, 마지막으로 결론과 향후 연구 과제를 제시한다.

2. 일반적인 아이폰 애플리케이션의 개발 방법

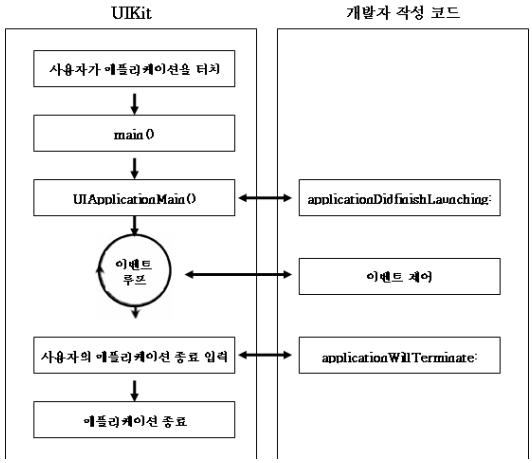
프로그램 제작 관점에서 볼 때의 게임 애플리케이션은 사용자가 화면에 출력되는 이미지를 입력 장치를 통해 조작하면서 진행하는 소프트웨어라고 할 수 있다[5]. 즉, 게임 프로그램 개발 방법론에는

그래픽과 사운드, 그리고 입력에 관련된 처리와 이런 요소들을 실행 및 제어하는 구조가 포함된다는 것이다. 아이폰과 위피의 개발 방법론은 위와 같은 요소 등에서 여러 차이점이 있다. 특히 아이폰의 경우에는 MVC(Model-View-Controller) 디자인 패턴 및 Delegation과 같은 개발 방법론이 추가되어서 위피에서의 클래스 중심 개발 방법과는 큰 차이를 보인다. 또한 그래픽을 처리하는 방식도 외부에 있는 이미지를 코드에서 등록하여 사용하는 위피의 방식과는 달리, 아이폰은 View라는 윈도우를 이용하여 그래픽을 처리하는 방식을 이용한다.

위 요소들을 중심으로 볼 때, 아이폰이 위피를 비롯한 기존의 모바일 게임 개발 방법론과 가장 크게 다른 점은 애플리케이션의 실행 제어 구조와 그래픽 처리 방식이다. 실행 제어 구조는 크게 애플리케이션이 시작되서 종료될 때까지의 라이프 사이클과 클래스 간의 구조, 그리고 애플리케이션을 계속 지속시키며 이벤트를 처리하는 루프 구조로 구분될 수 있다. 그래픽 처리 방식의 경우에는 실제 그래픽을 화면에 출력하는 방법과 그것을 움직여서 애니메이션을 처리하는 방법으로 구분된다. 2장과 3장에서는 위에서 언급한 내용들을 중심으로 아이폰과 위피 애플리케이션의 개발 방법론을 정리 및 비교한다.

2.1 아이폰의 실행 제어 구조

사용자가 터치 입력을 통해 어플리케이션을 실행시키면 UIKit 프레임워크의 main()이 먼저 호출된다. main() 에서는 애플리케이션에서 사용할 메모리 공간을 확보하는 역할을 한다. 이어서 UIApplicationMain()이 호출되는데, 이는 자동으로 applicationDidFinishLaunching: 메시지를 전송한다. applicationDidFinishLaunching: 는 개발자가 작성하는 코드 부분으로써, 애플리케이션에서 사용할 nib 파일을 메모리에 적재하는 등의 초기화 작업을 실행하게 된다. 이후부터는 이벤트를 처리하는 루프를 실행하게 되는데, 이 루프를 통해 이벤트를 주고 받으면서 애플리케이션이 동작된다. 이벤트를 받는 일을 하는 것은 UIApplicationMain()이 담당하고, 이벤트에 대한 응답은 개발자가 작성한 코드에 의해 이루어진다. 애플리케이션이 실행중인 상태에서 사용자가 종료 처리를 하면 applicationWillTerminate: 메시지를 전송하는데, 이 부분에 개발자는 메모리 해제와 같은 종료 처리 코드를 작성할 수 있다. [그림 1]은 이와 같은 아이폰 애플리케이션의 전반적인 실행 제어 구조를 보여준다.



[그림 1] 아이폰의 실행 제어 구조

아이폰 애플리케이션에서 제공하는 클래스의 구조는 Delegation과 MVC 디자인 패턴이다. 먼저 Delegation은 우리 말로 '위임'으로 해석할 수 있는데, 하나의 객체가 위임자로 지정된 다른 객체로 주기적으로 메시지를 보내서 해당 메시지를 처리하라고 요구하는 패턴이다. 위에서 언급한 applicationDidFinishLaunching: 메시지가 그 예라고 할 수 있다.

MVC 디자인 패턴은 Model, View, Controller로 구분하여 애플리케이션을 처리하는 것을 말한다. Model은 애플리케이션 데이터를 저장하고, 이 데이터를 조작하는 로직을 정의한다. View는 윈도우나 버튼과 같은 사용자 인터페이스 객체들의 집합으로 정의한다. 마지막으로 Controller는 위의 모

델과 View를 조율하는 역할을 한다. 즉, 모델이 바뀌었을 때 View가 제대로 갱신되도록 하는 역할을 하는 것이다.

UIApplicationMain()에서 제공하는 이벤트 루프는 사용자의 입력에 대응하는 메소드들의 실행을 관장한다. 그러나 화면에서의 그래픽 출력이나 애니메이션과 같이 지속적으로 실행을 해야 하는 메소드를 처리하는 방식에 대해서는 정의되어 있지 않다[6]. 그렇기 때문에 해당 기능을 수행할 수 있는 방식이 추가되어야 한다.

2.2 아이폰의 그래픽과 애니메이션 처리

아이폰 애플리케이션은 기본적으로 View라는 객체를 통해 사용자에게 화면을 보여준다. 사용자의 입력을 받아 그 결과를 보여주는 것도 View이다. 아이폰에서 View를 처리하는 순서는 다음과 같다. ① View를 사용자에게 표시하고 ② 사용자의 터치 입력을 받은 뒤에 ③ 터치 입력에 따른 처리를 수행하고, ④ 처리 결과를 다시 View로 반영한다[7].

UIView는 View를 그리고 이벤트를 처리하는 상위 클래스로, 화면을 갱신하는 메소드인 drawRect()를 오버라이드하여 그래픽을 처리한다. drawRect()는 View의 이동과 같은 변화가 있으면 자동으로 호출된다.

[그림 2] View를 이용한 그래픽 처리

[그림 2]에서 보는 것처럼 View를 이용한 그래픽 처리는 버튼이나 텍스트 박스와 같은 하이레벨 컴포넌트 혹은 이에 준하는 단순한 그래픽의 처리에 적합하다. 아이폰에서는 게임과 같이 복잡하고 정교한 그래픽 처리를 위해 Quartz 라이브러리를 제공한다.

Quartz는 Window API의 DC와 같이 CGContextRef라는 그래픽 컨텍스트를 이용하여 이미지를 그린다. 아이폰의 일반적인 애플리케이션에서 View와 Quartz를 사용하는 방식은 현재 상황에 맞는 그래픽 컨텍스트를 생성하며 [그림 3]과 같은 형식의 코드로 접근한다.

```
-(void)drawRect: (CGRect)rect {
  CGContextRef context
                = UIGraphicGetCurrentContext();
  CGPoint drawPoint = CGPointMake(100, 100);
  [testImage = drawAtPoint:drawPoint];
}
```

[그림 3] 아이폰의 그래픽 처리 코드

아이폰에서 이미지를 움직이는 애니메이션 처리는 UIView에서 제공하는 애니메이션을 이용한다. UIView 애니메이션은 View를 갱신할 때, 그 변화를 지정된 시간 동안 보여주는 동시에 부드럽게 이어주는 기능을 한다. 즉, UIView 애니메이션은 뷰의 현재의 상태와 앞으로 바뀔 미래의 상태 사이의 변화를 시작적으로 연결해주는 것이다. UIView 애니메이션은 블록 단위로 동작하며, 블록의 시작은 beginAnimations 메서드로 시작하고, 끝맺음은 commitAnimations 메서드로 끝낸다. 이 두 개의 메서드 호출 사이에 애니메이션하고자 하는 View의 형태를 지정해주어 이미지의 움직임을 제어할 수 있다[8].

3. 위피 게임 애플리케이션 개발 방법

3.1 위피의 실행 제어 구조

위피의 모든 프로그램은 Jlet을 상속받는다. Jlet

은 자바로 만든 위피 애플리케이션을 시작부터 종료까지의 흐름을 제어하기 위한 추상 클래스로써, 위피에서 제공하는 라이프 사이클 메서드인 startApp(), pauseApp(), destroyApp(), resumeApp()로 구성된다.

[그림 4] 위피의 실행 제어 구조

[그림 4]는 Jlet이 3개의 상태를 가지며, 4개의 라이프 사이클 메서드를 통해 애플리케이션 시작부터 종료까지의 상태를 전이하는 과정에 대해 표현한 그림이다.

먼저 startApp()을 통해서 애플리케이션을 Active 상태로 만든다. 이때가 애플리케이션이 실행되고 있는 상태이며 사용자의 입력을 받거나 특수 상황에 이르게 되면 애플리케이션이 잠시 멈추는 Paused 상태가 된다. 혹은 애플리케이션이 종료되는 Destroyed 상태로 바뀔 수도 있다. 물론 각 상태가 바뀔 때에는 해당되는 라이프 사이클 메서드가 호출된다. 이처럼 Jlet 클래스는 애플리케이션의 전반적인 실행 제어 구조를 담당한다.

Jlet 클래스와 더불어 위피의 클래스 구조를 형성하는 Card 클래스에서는 이미지를 이용한 화면 출력 및 키 입력에 따른 이벤트 처리, 그리고 게임을 처리하기 위한 루프 구조의 내용을 가진다. 즉 Jlet 클래스와 Card 클래스의 역할이 명확하게 구분되어져 있다고 할 수 있다[9].

애플리케이션의 지속적인 실행 및 이벤트 처리를 위한 루프 구조는 Card 클래스에서 제어한다. 위피에서는 애플리케이션의 루프를 제어하기 위한 스레드의 생성을 제공하며, 개발자는 루프를 실행시키기 위해 스레드의 start()를 호출해야 한다. start()는 실제 루프에서의 처리 부분을 담당하고 있는 run()을 개발자가 지정한 타이밍에 맞게 지속적으로 실행되도록 한다. [그림 5]의 코드는 이와 같은 루프의 구조에 대해서 보여준다. 생성자를 통해 스레드를 생성하고, 여기에서 타이머 메서드를 등록하여 지속적인 게임 루프를 제어하는 것이다.

```
public GameCard() {  // 생성자
  ...
  mThread.start();     // 루프(스레드) 시작
}

public void run() {
  while (mTherad != null)
  {
    ...
    repaint();  // 화면에 이미지 출력 실행
    try {
      Thread.sleep(100) // 0.1초마다 실행
    }
    catch(InturruptedException ex) {
    }
  }
}
```

[그림 5] 위피의 게임 루프 처리 방식

3.2 위피의 그래픽과 애니메이션 처리

위피에서의 그래픽 처리는 Card 클래스에서 담당하고 있다. 그래픽을 출력하는 paint()는 해당 객체가 그려지도록 요청 받았을 때 자동적으로 호출된다. 중요한 것은 paint()가 개발자가 직접 호출하는 메소드가 아니라, 시스템에서 필요할 때마다 호출하는 메소드라는 점이다. 개발자는 직접 화면에 출력하고자 하는 내용을 paint() 내에 오버라이드 해야 한다.

게임에서의 그래픽 처리는 이미지 파일을 리소스로 다룬다. 위피에서 이미지를 사용하기 위해서는 다음과 같은 단계를 거쳐야 한다. ① 이미지 생성 및 이미지 변수를 만들고, ② createImage()를

호출하여, 이미지를 생성하여 준비된 변수에 할당한 뒤에, ③ drawImage()를 이용하여, 이미지를 원하는 곳에 그린다[10]. [그림 6]의 코드는 위피의 paint()를 처리하는 과정에서, 특정 키를 눌렀을 때 해당 그래픽을 출력하는 방식을 기술한 것으로, drawImage()의 활용에 대해 알 수 있다.

```
public void paint(Graphics g) {
  this.g = g;
  switch(gameState) {
    case MENU:
      showMenu();
      switch(m_PressKeyValue) {
        case EventQueue.DOWN:
          ...
          g.drawImage(...);
          break;
      }
    case GAME:
      showGame();
      ...
      break;
      }
  }
}
```

[그림 6] 키를 눌렀을 때 그래픽 처리 방식

위피에서 애니메이션을 처리하는 방식은 주로 프레임에 따른 애니메이션 카운트 변수를 이용한다. [그림 7]의 코드는 위피에서 이미지를 애니메이션 시키는 형태를 보여준다.

```
public void draw_Ani (Graphics g)
{
   // 애니메이션 카운트 증가
   AniCnt++;
   // 애니메이션 카운트가 5 이하이면
   if (AniCnt < 5)      {
      // 0부터 5프레임까지의 이미지 출력 내용
   }
   else         {
      // 5프레임 이후부터의 이미지 출력 내용
   }
}
```

[그림 7] 위피의 애니메이션 처리 방식

위의 코드에서 나타난 함수는 게임 루프에서 설정한 시간에 맞춰서 매번 호출이 된다. 예를 들어 게임 루프를 1초당 20프레임씩 출력하는 것으로 설정했다면, 위의 함수는 1초에 20번씩 호출이 된다. 호출이 될 때마다 애니메이션 카운트에 따라서 각기 다른 이미지를 화면에 출력하면서 애니메이션을 표현하게 된다[11].

4. 게임 개발에서의 아이폰과 위피 비교 분석

4.1 실행 제어 구조에서의 차이점

위피의 실행 제어 구조는 게임 제작에 있어서 기능적인 분류를 제공한다. 클래스의 구조를 전반적인 제어에 대한 관리 부분과 그래픽이나 게임 루프 처리 등의 게임 데이터 제어 부분으로 구분되었다. 이로 인해 설계나 유지보수의 측면세어 높은 가독성이나 이해도를 보장받을 수 있다. 애플리케이션을 지속시키는 루프 방식도 게임 처리 전용의 스레드를 이용하여 지속적으로 애플리케이션 실행에 대한 제어를 할 수 있다. 개발자는 루프 제어에 대한 변경이 필요할 때 run() 부분만을 수정하면 된다. 전반적으로 위피의 이러한 구조는 이전 국내에서 많이 사용되었던 GVM이나 Brew와 같은 모바일 플랫폼과 유사하므로 기존 위피 개발자 입장에서 접근이 용이하다는 장점이 있었다.

이에 반해 아이폰의 실행 제어 구조의 경우에 MVC 디자인 패턴에서의 View나 Controller의 개념 및 사용법, 그리고 Delegation 패턴에 의해 처리를 다른 메서드에게 위임한다는 개념 등은 국내 위피 개발자들에게는 익숙하지가 않다. 애플리케이션의 루프 형태 역시 게임에 적합한 화면 출력이나 입력 이벤트 처리에 대한 스레드나 타이머를 설정하는 방법론이 애플 개발자 포럼이나 튜토리얼에는 구체적으로 나와 있지 않다[12].

4.2 그래픽과 애니메이션 처리 방식의 차이점

위피에서는 Card 클래스에서 외부의 이미지 리소스를 이용하여, 이미지 변수를 생성하고 drawImage()를 통해 원하는 위치에 이미지를 그려주는 과정을 거친다.

아이폰에서는 UIWindow의 위에 다양한 UIView를 올려놓고, 화면 전환은 View 단위로 움직이며 진행한다. View를 통해 그래픽을 출력하는 drawRect()는 위피의 paint()와 같이 OS에 의해 자동으로 호출이 된다. UIView에서 기본적으로 제공하는 그래픽 처리 함수는 UIKit 프레임워크에 내장된 것으로서, 간단한 도형이나 View의 이미지를 그리는 데 사용된다[13]. 즉, 이미지가 게임 애플리케이션에 비해 상대적으로 적게 들어가는 일반 애플리케이션의 그래픽 처리에 쓰인다.

본 논문에서는 2D 게임 애플리케이션의 그래픽을 전문적으로 처리하는 그래픽 라이브러리인 Quartz를 이용한다. 그러나 Quartz에서는 위피와 같이 한 번의 호출로 그래픽을 쉽게 그려주는 메소드는 포함되어 있지 않다. 그렇기 때문에 아이폰에서 이미지를 그리기 위해서는 위피에 비해 그 방식이 복잡하고 코드 또한 길어지게 된다.

위피의 애니메이션 처리는 각 프레임에 따라 적절한 위치를 지정해주는 방식을 취하고 있다. 그러나 아이폰에서는 OS에서 지원하는 전용 애니메이션 메서드들을 통해 애니메이션을 처리한다. 이러한 메서드들은 개발자의 입장에서 사용 방법이 간단하고, 이미지를 부드럽게 움직일 수 있다는 장점이 있다. 그러나 게임에서 표현되는 이미지의 세밀한 움직임을 처리하는 데에는 어려움이 있다.

[표 1]은 이제까지의 아이폰과 위피를 비교한 내용을 정리한 것으로, 게임 개발의 측면에서 볼 때 위피보다는 아이폰에서의 개발이 더 불편한 것을 확인할 수 있다.

국내 위피 개발자가 지속적인 아이폰의 게임 애플리케이션을 개발하고자 한다면 이에 대한 내용을 차후 숙지해야 한다. 그렇지만 초기에 아이폰을 대상으로 개발을 할 때에는 많은 어려움을 겪을 수 있다. 다음 장에서는 이를 해결할 수 있는 방안으로, 위피의 구조를 적용한 아이폰 게임 애플리케이션의 방법론을 제안한다.

[표 1]

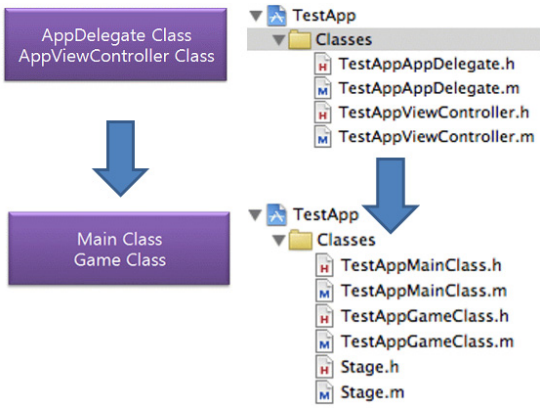
플랫폼	아이폰	위피
실행 제어 구조	MVC 패턴과 Delegation을 이용해 데이터를 제어	Jlet 및 Card 클래스가 기능적으로 분리 되어 데이터를 제어함
	게임 상태에 따라 처리할 이벤트 루프 구조가 기본적으로 제공되지 않음	run() 스레드를 통해 타이머를 이용한 게임 루프 있음
그래픽 ,애니메이션 처리 방식	View와 Quartz를 이용하여 그래픽 데이터를 관리	외부 이미지를 직접 이용해 변수 생성, Card 클래스를 통해 이를 관리함
	이미지를 처리하는 코드의 양이 많고 복잡함	이미지를 원하는 좌표에 그릴 수 있는 메서드 있음
	애니메이션 설정 및 사용이 용이함, 그러나 게임에서의 세밀한 움직임을 표현하는 데 어려움이 있음	프레임에 따라 원하는 위치에 이미지를 그려 세밀한 움직임 처리가 가능함

5. 아이폰 게임 애플리케이션 방법론 제시

5.1 실행 제어 구조에서의 변경

아이폰의 클래스 구조는 Delegation을 명시하는 클래스와 ViewController를 명시하는 클래스로 분류가 된다. 이는 아이폰의 Delegation 개념과 MVC 패턴을 구현한 것으로, 위피의 Jlet과 Card 클래스의 구조와는 다른 형태를 가진다. 또한 게임 데이터를 처리하는 구분이 뚜렷하게 되어있지 않다. 그러므로 위피의 클래스 구조와 같이 게임을 흐름을 제어하는 클래스와 게임 데이터 및 그 외 기능들을 처리하는 클래스로 분류를 하도록 한다.

[그림 8]과 같이 본 논문에서는 아이폰의 기본 클래스 구조인 Delegation 클래스와 ViewController 클래스를 위피와 같이 게임 흐름 제어 클래스와 게임 데이터 처리 클래스로 분류하였다.



[그림 8] 아이폰 실행 제어 구조(클래스 구조) 변경

MainClass는 기존의 Delegate 클래스를 변경한 것으로, 애플리케이션의 시작 및 전반적인 실행 제어를 담당한다. 이는 위피의 Jlet 클래스와 유사한 역할을 한다. GameClass는 ViewController Class를 변경한 것으로 위피의 Card 클래스와 유사한 역할을 하며, 이미지를 화면에 그리거나 사용자 입력을 처리한다.

아이폰은 UIApplicationMain()에 의해 터치 입력에 따라 이벤트를 처리하는 루프를 가진다. 그러나 이것은 기존의 게임 애플리케이션에서 나타나는 스레드와 타이머를 이용한 게임 루프 방식을 가지고 있지 않다. 위피에서는 스레드를 이용한 Run()에 타이머 메서드를 추가하여, 설정한 프레임에 따라 게임 메서드를 호출하는 방식을 사용한다. 본 논문에서는 이러한 위피의 구조를 차용하여, UIApplicationMain()의 초기화 수행 중 nib 파일을 로드할 때 호출되는 메서드, awakeFromNib()을 스레드로서 활용하고, 이 내부에 타이머를 설정한다. awakeFromNib은 객체를 할당하고 초기화를 한 후에 nib 파일에 있는 모든 오브젝트들에게 보내지는 메시지로써, View를 이용하는 초기화를 하지 않을 때에 사용할 수 있다. [그림 9]는 awakeFromNib()에 스레드 및 타이머 함수를 설정하는 내용을 가진다.

```
-(void) awakeFromNib {
  mainTimer =
  [[NSTimer scheduledTimeWithTimeInterval:0.05
                  target:self
                selector:@selector(TimerMethod:)
                userInfo:nil
                 repeats:YES] retain];
}
```

[그림 9] awakeFromNib 메서드

5.2 그래픽과 애니메이션 처리 방식의 차이점

아이폰은 View 기반 위에서 이미지를 그리거나 외부에 있는 것을 로드하여 사용한다. 그러나 View에서 제공하는 그래픽 처리 함수로는 다양한 게임 그래픽을 처리하기에 성능이 부족하고 다양한 활용도를 가진 API를 제공하지 않는다. 그렇기 때문에 본 논문에서는 아이폰의 그래픽 전용 라이브러리인 Quartz를 이용한다. 아이폰에서 그래픽을 처리하는 방식은 drawRect() 내에 이미지를 그리는 코드를 작성하는 것으로써, View에 의해 자동으로 호출이 된다. 그러나 setNeedsDisplay()에 의해 명시적으로 drawRect()를 호출하는 방식을 사용할 수도 있으며, 이는 위피에서 그래픽을 처리하는 paint() 메서드를 호출하는 방식과 유사하다. 본 논문은 위피의 구조를 아이폰에 적용시키는 연구이므로 타이머 메소드 내에서 setNeedsDisplay()를 호출시키는 형태를 이용하여 위피와 동일한 처리 방식을 사용했다.

위피에서는 Card 클래스를 통해 repaint()와 같은 형태로 이미지를 처리 및 관리를 한다. 이에 반해 아이폰에서는 drawRect()가 동일한 기능을 하는데, 이 때에 사용되는 윈도우와 라이브러리가 View와 Quartz이다. 기본적인 아이폰 프로그램에서는 이 두 가지를 이용하여 그래픽을 처리하고 관리하지만, 본 논문에서는 위피와 마찬가지로 이미지 변수를 생성하고 이것을 처리하는 방식으로 수정하였다.

또한 아이폰에서 그래픽을 처리하기 위한 메소드의 사용 방식이 위피와는 다르기 때문에, 이와 관련한 설정을 해 줄 필요가 있다. 그러므로 본 논문에서는 아이폰의 그래픽 출력에 대한 일련의 과정을 하나의 메서드로 생성하였다. 해당 메서드는 위피에서 그래픽을 출력하는 drawImage()와 유사한 형태로 사용 방식을 정의하였기 때문에 위피 개발자들이 쉽게 사용할 수 있다는 장점이 있다. [그림 10]은 본 논문에서 아이폰의 Quartz를 이용해 생성한 메서드로, 아이폰의 그래픽 출력 방식을 하나의 모듈로 구성한 것이다. 이것을 이용하면 개발자는 위피와 유사한 방식으로 그래픽을 처리하는 것이 가능하다.

```
-(void) drawImage:(double)x y:(double)y
                    image:(CGImageRef)image {
  CGImageRef imageRef = image;
  CGSize imageSize;
  imageSize.width = CGImageGetWidth(imageRef);
  imageSize.height = CGImageGetHeight(imageRef);

  CGContextDrawImage(context, CGRectMake(0, 0,
        imageSize.width, imageSize.height), imageRef);
}
```

[그림 10] 위피와 유사한 기능의 그래픽 출력 메서드

아이폰에서 제공하는 애니메이션 메서드는 사용 방법이 쉬우나, 이미지의 세밀한 움직임을 표현하는 데에는 어려움이 있다. 그렇기 때문에 위피에서 이미지를 애니메이션 시키는 방법을 이용하여 아이폰의 애니메이션 처리를 수행하도록 한다. 본 논문에서는 게임 루프를 통해 지속적으로 애니메이션을 수행하는 메소드를 호출하고, 해당 메소드는 애니메이션 카운트를 프레임 수에 맞춰 증가시키면서 이미지의 좌표를 조금씩 이동시키는 기능을 수행하며 애니메이션을 처리한다. [그림 11]은 아이폰에서 위피의 애니메이션 처리 방식을 이용하여 애니메이션을 처리하는 코드의 기초 형태로써, 본 논문에서 생성한 drawImage()를 통해 이미지의 움직임을 제어한다.

```
-(void) drawAnimation {
  frame++;

  [drawImage:0 y:0 image-bg];
  [drawImage:frame y:frame image->character];
}
```

[그림 11] 위피 방식으로 아이폰 애니메이션을 처리하는 코드

6. 실제 게임 적용 사례

본 논문에서 제시한 아이폰 게임 애플리케이션 방법론을 적용하여 터치 인터페이스를 활용한 숫자 연산 교육용 게임을 제작하였다. 이를 통해 위피의 구조가 적용된 아이폰 게임 애플리케이션 방법론의 효과성을 검증하고자 한다. 제작한 게임은 간단한 숫자를 이용하여 연산 교육을 진행하는 게임으로 총 3개의 스테이지로 이루어져 있다. 스테이지를 클리어해야 다음 스테이지로 진행할 수 있는 방식이며, 모든 스테이지를 클리어 했을 때에 게임의 엔딩이 나타난다.

```
- (void) awakeFromNib:
              (UIApplication *)application
{
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];

    srandom(time(NULL));

    mainTimer =
    [[NSTimer scheduledTimerWithTimeInterval:0.05
                                      target:self
                                    selector:@selector(onTimer:)
                                    userInfo:nil
                                     repeats:YES] retain];
    g = [[Util alloc] init];
    img = [[InitImage alloc] init];
    numberCalculation = [[NumberCalculation alloc] init];
    gunshooting = [[GunShooting alloc] init];
    train = [[Train alloc] init];

    [self initGame];
    [train InitTrain];
}
```

[그림 12] 게임에서 구현한 awakeFromNib()

애플리케이션이 실행되면 UIApplicationMain()이 시작되면서 객체를 생성하고 위임 담당 객체 및 이벤트 루프를 설정한다. 그리고 제어권을 awakeFromNib()에게 이전시키면서 스레드 및 타

이며 메서드를 정의하고 등록한다. [그림 12]는 이를 구현한 코드로써, 애플리케이션의 객체 생성 및 초기화, 그리고 타이머를 정의 및 등록하고 있다.

타이머를 통해 애플리케이션이 종료될 때까지 지속적으로 onTimer()가 호출되어 루프시켜야 할 메소드를 실행하며 게임을 진행한다. [그림 13]과 같이 onTimer()는 setNeedsDisplay()를 호출하는데, 이는 아이폰에서 이미지 출력을 담당하는 drawRect()의 호출을 유도한다.

```
- (void) onTimer:(NSTimer *)aTimer
{
    [self setNeedsDisplay];
}
```

[그림 13] 이미지 출력을 루프시키는 onTimer()

drawRect()는 setNeedsDisplay()에 의해 지속적으로 호출이 되며, 이를 통해 게임의 현재 상태를 체크하여 관련되는 화면을 출력하게 된다. [그림 14]의 코드는 제작한 게임의 drawRect()의 일부로, switch문을 통해 게임의 각 상태에 따라서 해당되는 화면을 출력하는 메서드를 호출한다. 이러한 형태는 위피에서의 게임 프로그래밍에서 현재 게임 상태에 따라 다른 그래픽을 출력하는 방식을 차용한 것이다.

```
- (void)drawRect:(NSRect)aRect
{
    context = UIGraphicsGetCurrentContext();

    srand(time(NULL));

    switch (gameState)
    {
        case LOGO:
            [self drawLogo];
            break;
        case INTRO:
            [self drawIntro];
            break;
        case MENU:
            [self drawMenu];
            break;
        case GAMEMENU:
            [self choiceGame];
            break;
```

[그림 14] 위피 방식으로 처리한 drawRect()

[그림 14]에서 drawRect()가 호출하는 메서드들은 그래픽을 출력하는 내용으로써, 위피 구조가 적용된 이미지 메서드와 애니메이션 처리 방법을 적용하고 있다. [그림 15]는 이 중 drawIntro()의 내용 일부로써, 본 논문에서 생성한 이미지 메서드를 이용한 애니메이션 처리에 대해 보여준다.

```
- (void) drawIntro
{
    frame++;
    [drawImage:0 y:0 image:titleBG];
    if (frame == 7)
        [drawImage:0 y:110 image:superman];
    else if (frame == 8) {
        [SetImageAlpha:3];
        [drawImage:60 y:110 image:superman];
    }
    else if (frame >= 9 && frame <= 15) {
        [SetImageAlpha:0];
        [drawImage:135 y:110 image:superman];
    }

    ...
```

[그림 15] drawIntro()의 코드 일부

[그림 16] 제작한 게임의 스크린샷 [그림 16]은 아이폰의 API 및 본 논문에서 설계한 게임 개발 방법론과 추가 메서드를 활용하여 제작한 게임의 화면이다.

[그림 16] 제작한 게임의 스크린샷

6. 결 론

본 논문은 극내의 위피 개발자들이 어떻게 하면 쉽게 아이폰 게임 애플리케이션 개발에 참여할 수 있을까하는 발상에서부터 출발하였다. 이에 아이폰의 애플리케이션 개발 방법과 위피의 게임 애플리케이션 개발 방법을 실행 제어 방식과 그래픽 및 애니메이션 처리 방식으로 구분하여 비교하였다. 현재 아이폰에서 제공하고 있는 개발 방법론에 대한 자료나 가이드라인은 주로 아이폰의 화면 단위인 View를 통해 실행 구조를 제어하고, 그래픽 및 애니메이션을 처리하는 방식에 대한 내용이다. 아이폰의 MVC 패턴이나 Delegation에 대한 개념, 그리고 View나 Quartz를 통한 이미지의 사용 방식은 기존에 주로 위피를 다루어오던 국내 개발자들에게는 매우 생소한 것이다. 그렇기 때문에 위피의 실행 제어 방식을 차용하여, 클래스를 기능별로 구분하고 게임 루프 방식을 적용하였다. 그래픽과 애니메이션의 처리 방식은 위피에서 사용되는 이미지 처리 메서드와 유사한 형태의 메서드를 추가 제작하여, 해당 처리를 쉽게 구현할 수 있도록 하였다.

본 논문에서 위피의 구조를 이용한 아이폰 개발 방법론 적용에 중점을 둔 결과 다음과 같은 한계사항이 있었다. 먼저 아이폰과 위피의 입력 처리 방식이 다르기 때문에 이에 대한 차이점에 대해 분석하는 내용을 다루지 못했다. 아이폰은 터치 입력 인터페이스를 이용하지만, 위피는 키 버튼 방식을 이용하기 때문에 이에 대한 차이점에 대해서 정리할 필요가 있다. 또한 아이폰의 새로운 인터페이스인 가속도 센서나 GPS 등을 이용한 게임 개발 방법론에 대해서도 추가적인 연구가 필요하다.

아이폰의 개발 방식은 애플리케이션 개발의 새로운 패러다임을 제공하면서 창조적이고 신선한 방식의 애플리케이션 제작에 유용하다. 하지만 국내의 위피 개발자들에게는 익숙하지 않은 형태의 개발 방법론이기 때문에 접근하기가 쉽지 않고, 현재 게임 개발 방법론 형태 역시 뚜렷하지 않은 상태이다. 본 논문에 국내 위피 개발자가 아이폰 개발 시장에 진입할 때에 도움이 될 수 있도록 하나의 방법론을 제시하는 데 목적이 있다고 할 수 있다. 또한 향후 아이폰 게임 애플리케이션을 개발할 때에 하나의 방법론으로써 가이드라인을 제시하는 데 그 의의가 있다.

본 논문에서 제시한 게임 개발 방법론은 이러한 의의에 맞게 연구가 된 하나의 가이드라인이며, 앞으로 지속적으로 연구되어야 할 대상이다. 현재는 아이폰 뿐만 아니라 다른 스마트폰에서의 게임 개발 방법론에 대한 논의가 부족하나, 시장에서의 지속적인 요구를 통해 활발한 연구가 시작될 것이라 예상할 수 있다. 향후에는 가속도 센서나 멀티 터치 인터페이스, 아이폰 플랫폼의 독특한 인터페이스와 기능들을 이용한 게임 애플리케이션을 개발 연구할 예정이다.

참고문헌

[1] 김기종, "모바일 플랫폼 위피 폐지 논의와 국내 휴대폰 시장의 영향", 산은조사월보, 633호, pp.52-79, 2008.

[2] 윤민홍, "글로벌 모바일 단말 소프트웨어 플랫폼 동향", 한국전자통신연구원, 23권, 1호, pp. 44-53, 2008

[3] http://www.zdnet.co.kr/ArticleView.asp?artice_id=20090303154249

[4] http://developer.apple.com/iphone/

[5] 송영덕, "게임 기술", 대림, pp272-274, 2004.

[6] 하기룡, "아이폰 프로그래밍 가이드", 프리렉, pp392-397, 2009.

[7] 애플, iPhone Application Programming Guide", 애플, pp.15-18, 2008.

[8] Jeff Lamarche, "Beginning iPhone Development", APRESS, pp203-211, 2008.

[9] 김석구, "위피 모바일 프로그래밍", 영진닷컴, pp.99-102, 2004.

[10] 김인교, "위피 모바일 게임 프로그래밍", 대림 pp.101-102

[11] http://www.iphonedevsdk.com/forum/iphone-sdk-development/35225-animation-uiimageview.html

[12] http://www.idevgames.com/forum/showthread.php?t=15301
[13] Paul Zirkle, "iPhone Game Development", O'REILLY, pp41-43, 2009.

유 현 성(Hyun Seong Yu)

2006년 상명대학교 소프트웨어학부 이학사
2008년 상명대학교 게임학과 게임학석사
2005년 10월~현재 (주)로직게임 개발팀장
2008년 3월~현재 상명대학교 컴퓨터과학과 게임학전공 박사재학중
2007년 3월~현재 용인송담대학 컴퓨터게임정보과 겸임교수

관심분야 : 게임 기획, 스마트폰 애플리케이션

이 대 웅(Dae Woong Rhee)

1996년 서울대학교 대학원 계산통계학과 이학박사
1990년~현재 상명대학교 소프트웨어대학 디지털 미디어학부 교수
2008년~현재 상명대학교 일반대학원 대학원장

관심분야 : 게임 기획, CT, 게임 프로그래밍