# 인 게임 상태 전환 로직 구현

프로젝트를 제작하기에 앞서 Update 함수에서 하드코딩으로 각 상태를 일일이 구현하는 것보다 해당 상태일 때 필요한 로직을 상태에서만 구현하는 방법이 없는지 고민하다가 우연히 [읽은](https://book.naver.com/bookdb/book_detail.nhn?bid=2094425) [서적](https://book.naver.com/bookdb/book_detail.nhn?bid=7453748)에서 FSM을 생각하게 되었고 프로젝트에 활용해보고자 했습니다.

**프로젝트에 사용한 FSM 구조**

|  |  |  |
| --- | --- | --- |
| NFSM | | |
| InGameEntry | InGamePlay | InGameResult |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| InGameEntry | | InGamePlay | | InGameResult |
| InGameEntrySingle | InGameEntryPvP | InGamePlayMove | InGamePlayBattle | None |

|  |  |  |  |
| --- | --- | --- | --- |
| InGamePlayMove | | InGamePlayBattle | |
| InGamePlayMoveSingle | InGamePlayMovePvP | None | InGamePlayBattlePvP |

**설정 함수**

|  |  |
| --- | --- |
| Initailize | 최초 한번만 실행, FSM을 초기화하는 함수 |
| Start | Update 함수를 호출 하기 전에 미리 설정하고자 할 때 호출하는 함수 |
| Update | 지속적으로 상태를 감지하고 특정 조건에 의해 End 상태를 설정하는 함수 |
| End | 해당 FSM을 종료하고 다음 FSM을 호출하기 전에 정리하는 함수 |

**FSM 관리자 함수**

|  |  |
| --- | --- |
| StartFSM | FSM을 시작하는 함수 |
| NextFSM | 다음 FSM을 실행하거나 종료하는 함수 |

각 FSM에는 설정 함수를 통해서 해당 상태에서 조건에 의해 초기화, 갱신, 완료를 반복하고 상태가 종료되면 다음 FSM을 실행하도록 각 FSM 관리자에게 요청합니다. 요청받은 FSM 관리자는 해당 상태에 따라 다음 FSM을 실행할지, 종료할지 결정하고 관리자의 관리자에게 요청하는 구조입니다. 모든 FSM이 종료되면 인 게임 관리자에게 게임 종료를 요청하고 결과 처리를 합니다.

# 소녀전선식 진형 버프 구현

소녀전선에는 각 각의 진형 버프가 존재합니다. 예를 들면 다음과 같습니다.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

인 게임에서 이와 같은 형식의 진형 버프 기능을 구현해달라고 요청이 왔습니다. 처음 작업할 때 3 \* 3의 범위를 토대로 작업을 하였으나 캐릭터의 위치가 이동이 될 경우 3 \* 3 범위로는 구현할 수가 없었습니다. 그래서 3 \* 3 범위가 아니라 9 \* 9 범위로 확장시켜서 생각을 해보았습니다.

|  |  |  |
| --- | --- | --- |
| 01 02 03  04 05 06  07 08 09 | => | 01 02 03 04 05 06 07 08 09  10 11 12 13 14 15 16 17 18  20 21 22 23 24 25 26 27 28  30 31 32 33 34 35 36 37 38  40 41 42 43 44 45 46 47 48  50 51 52 53 54 55 56 57 58  60 61 62 63 64 65 66 67 68  70 71 72 73 74 75 76 77 78  80 81 82 83 84 85 86 87 88 |

여기서 3 \* 3 범위가 9 \* 9 범위에 해당되는 영역은 다음과 같습니다.

|  |  |  |
| --- | --- | --- |
| 01 02 03  04 05 06  07 08 09 | => | 33 34 35  43 44 45  53 54 55 |

처음 캐릭터가 배치되는 진형 인덱스를 9 \* 9 범위로 변환시킨 다음에 해당 캐릭터가 주는 진형 버프 간격을 계산하여 변형된 진형 범위 안에 캐릭터가 존재하면 버프를 부여하고 그렇지 않으면 버프를 부여하지 않는 방식으로 구현했습니다.

# View 전환 시스템

해당 뷰에서 유저가 행동을 끝나면 다른 뷰로 이동하거나 로비로 돌아옵니다. 뷰가 활성화/비활성화 부분을 일일이 체크 및 새로운 뷰가 추가될 때마다 추가적인 작업을 해야 합니다. 이런 부분을 원활하게 하기 위하여 만든 것이 View 전환 시스템입니다.

**주요 스크립트**

**ViewManager**

View를 관리하기 위한 초기화, Push, Pop, Peek을 처리하는 스크립트입니다. View로부터 요청이 들어오면 요청에 따라서 처리를 합니다.

|  |  |
| --- | --- |
| Initialize | 최초 한번만 실행, 기반이 되는 뷰와 초기화하는 함수 |
| Push | View를 Push하는 함수 |
| Pop | View를 Pop하는 함수 |
| Peek | View를 Peek하는 함수 |

**CView**

|  |  |
| --- | --- |
| SOpen | 외부로부터 뷰의 활성화를 시작하는 함수 |
| Open | 외부로부터 접근할 수 없는 함수로 SOpen을 호출하면 내부에서 호출하는 함수 |
| Opening | 뷰를 활성화하기 전에 사전에 해야하는 기능을 처리하는 함수 |
| Opened | 뷰가 완전히 활성화가 되었으며 스택에 Push하는 함수 |
| SClose | 외부로부터 뷰의 비활성화를 시작하는 함수 |
| Close | 외부로부터 접근할 수 없는 함수로 SClose을 호출하면 내부에서 호출하는 함수 |
| Closing | 뷰를 비활성화하기 전에 사전에 해야하는 기능을 처리하는 함수 |
| Closed | 뷰가 완전히 비활성화가 되었으며 스택에 Pop하는 함수 |
| Show | Push없이 단순히 뷰를 활성화하는 함수 |
| Hide | Pop없이 단순히 뷰를 비활성화하는 함수 |

**현재 남은 과제**

**팝업 창에 대한 작업이 현재 연구 중**

* 팝업 창이 한 번 활성화된 다음 비활성화 되기 전에 다시 팝업 창이 활성화되는 경우를 처리하지 않음

# 무한 스크롤

수집을 하는 게임들은 인벤토리가 거의 필수적으로 등장합니다. 하지만 유저가 소유하고 있는 아이템을 모두 생성하면 개수에 따라 지연 현상이 발생할 것이며 저 사양 폰의 경우 메모리 부족으로 어뷰징 현상이 발생될 수도 있습니다. 이를 방지하기 위한 기능이 무한 스크롤입니다.

작업은 NGUI의 스크롤 뷰를 분석하여 상속받아서 재사용을 했습니다.

**생성 작업**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| InLine1 | In아이템1 | In아이템2 | In아이템3 | In아이템4 |
| InLine2 | In아이템1 | In아이템2 | In아이템3 | In아이템4 |
| OutLine3 | Out아이템1 | Out아이템2 | Out아이템3 | Out아이템4 |

유저가 소유한 아이템을 모두 생성하지 말고 InLine1 ~ InLine2는 게임에서 보이는 영역과 유저에게 보이지는 않지만 드래그를 했을 경우 그 대비를 해두기 위해서 미리 한 Line(OutLine3)을 생성을 해둡니다.

**드래그 작업**

드래그를 하려면 [**드래그의 최종 영역 설정], [단순 드래그], [순간 드래그]**를 고려해야 합니다. 단순 드래그의 경우 가볍게 드래그를 하는 정도이며, 순간 드래그는 드래그를 강하게 함으로써 머신 슬롯처럼 여러 개의 항목을 지속적으로 갱신시켜주는 드래그를 말합니다. 이 둘을 구분하는 이유는 처리 방식이 다른 점이 있기 때문입니다.

지금부터 설명하는 스크롤 뷰의 방향은 Vertical이며 드래그 방식은 Top & Bottom입니다.

**드래그의 최종 영역 설정**

유저가 소유한 아이템 개수를 행 수, 열 수를 구한 다음 드래그 Bounds를 설정합니다.

**단순 드래그**

드래그를 하면 드래그 좌표가 행 하단 좌표보다 클 경우 다음과 같은 진행이 이루어집니다.

InLine1이 범위에서 벋어 납니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| InLine1 | In아이템1 | In아이템2 | In아이템3 | In아이템4 |
| InLine2 | In아이템1 | In아이템2 | In아이템3 | In아이템4 |
| InLine3 | In아이템1 | In아이템2 | In아이템3 | In아이템4 |

InLine1은 InLine3의 밑으로 이동하고 Line4의 정보로 초기화합니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| InLine2 | In아이템1 | In아이템2 | In아이템3 | In아이템4 |
| InLine3 | In아이템1 | In아이템2 | In아이템3 | In아이템4 |
| OutLine4 | Out아이템1 | Out아이템2 | Out아이템3 | Out아이템4 |

위와 같은 과정을 Top, Bottom 방향으로 드래그할 때 계속해서 반복합니다.

**순간 드래그**

기본적인 원리는 단순 드래그와 비슷합니다. 하지만 순간적으로 주는 강도가 사용자마다 다르며 deltaTime 역시 기기마다 다르기 때문에 Update 문에서 스크롤 뷰의 좌표가 일정하지 않습니다. 따라서 스크롤 뷰가 이동 중일 때 좌표를 가지고 와서 해당 좌표가 어떤 Line 인지 파악하여 라인을 기준으로 모든 아이템을 갱신하도록 합니다.

**후 처리 상속받은 Grid의 Reposition**

Grid의 Reposition 함수를 호출하면 아이템이 재배치가 되는데 위의 방식으로 스크롤 뷰를 변경하다 보니 Grid 역시 상속받은 후 재정의를 하였습니다. 재정의 후 재배치를 적용시켰으나 다음과 같은 문제가 발생했습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 최초 시작 라인들 |  | 마지막 라인들 |  | Reposition함수 실행 |
| InLine1 |  | OutLineN – 2(^^b) |  | OutLineN - 2 |
| InLine2 | => | OutLineN – 1 | => | OutLineN – 1 |
| OutLine3 |  | OutLineN |  | OutLineN(-\_-??????) |

마지막 라인들에서 Reposition을 실행하면 마지막 라인인데 대비용 라인을 하나 생성하게 됩니다. 따라서 마지막 라인의 경우 발생하는 부분을 고려하여 후 처리 작업을 진행해야 합니다.

**현재 남은 과제**

**아이템에 대한 MVC 모델을 적용**

M의 경우 인벤토리 아이템간 형식이 비슷하고 사용특성에 맞게 구현하면 되므로 추상화 및 구현(팩토리 패턴)을 도입하여 설계

V의 경우 아이템은 이미지만 출력하면 되므로 이미지 출력 기능들만 설계

C의 경우 M 과 V의 중간다리 역할을 하도록 설계

# [각 스토어 서비스 & 인앱 결제] 전환 시스템

이 전환 시스템은 프로젝트를 작업 진행 중 돌연 연구를 해보자는 의도로 구현하게 되었습니다.

해당 프로젝트인 미래구원자의 경우 다음과 같은 서비스를 진행하였습니다.

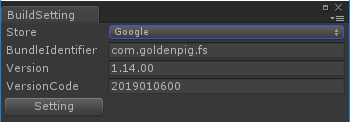
* 구글(미래구원자)
* IOS(The Savior of future:Tank War)
* 원스토어(미래구원자)

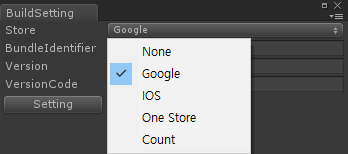
각 스토어의 프로세스는 다음과 같은 유사점이 있습니다.

* 로그인
* 업적
* 리더보드
* 인앱 결제

등이 있습니다. 하나의 프로젝트에 특정 스토어를 설정한 다음 해당 스토어의 제공자를 생성하여 처리하면 스토어 별 프로젝트를 따로 만들 필요가 없다는 생각이 들어서 구현을 시작하게 되었습니다.

 클릭



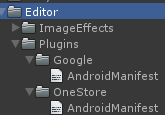


None과 Count는 테스트를 하기 위해 설정한 것을 지우지 않아서 나타나는 것임.

해당 정보의 입력이 완료되면 Setting 버튼을 클릭

서비스할 플랫폼이 Google일 경우 구글용 안드로이드 매니페스트 파일로 교체

원 스토어일 경우에는 원 스토어 안드로이드 매니페스트 파일로 교체



**스토어 관리 스크립트**

|  |
| --- |
|  |

**SocialAchievement**

* 스토어마다 사용되는 업적들의 공통된 기능을 정의한 스크립트. 추상 클래스

**PlatfromStoreManager**

제공자(Provider)를 생성하고 관련 기능 호출 시 제공자와 연결하는 스크립트

* 설정된 스토어의 설정에 따라 제공자(Provider)를 생성
* 제공자를 성공적으로 생성한 뒤 Operation을 통해 Service Type, Work Type, Callback 함수, 보조 값을 설정을 제공자에게 넘겨줌

**PlatfromProvider**

PlatfromStoreManager에게서 넘겨받은 매개 변수에 따라 처리를 함

* Service Type : SocialService, Billing
* Work :

1. 서비스 타입이 SocialService 일 경우 다음과 같은 타입이 존재함

SLogin, SLogout, SLoadAchivement, SOpenAchievement, SUpdateAchievement, SOpenLeaderBoard, SUpdateLeaderBoard,

SUserName, SUserIconURL, SEmail

1. 서비스 타입이 Billing 일 경우 다음과 같은 타입이 존재함

BInitialize, BPurchase

* Action : 작업 완료 이후 받을 콜백함수
* extraData : 해당 작업에 필요한 추가 값

**PlatformIAPService**

* 서비스 타입이 Billing일 경우 해당 스토어의 IAPService를 생성함. 현재 Onestore 전용.

**PlatformGameService**

* 서비스 타입이 SocialService 일 경우

**PlatformBilling**

* 결제 처리용으로 만들었으나 사용되지 않음.

**AchievementKey**

* 업적에 관련된 키들을 정의한 스크립트

**스토어 별 생성 스크립트**

|  |  |  |
| --- | --- | --- |
| **Google** | **IOS** | **OneStore** |
|  |  |  |

**GPGService 및 IOSGameService**

* 로그인, 업적, 리더보드 등에 관한 처리를 하는 스크립트

**GoogleBilling**

* 현재 유니티에서 제공해주는 IAP 플러그 인에 의해서 사용되지 않음.

**GoogleAchievement 및 IOSAchievement**

* 업적을 테이블화해서 불러오는 것이 아닌 스토어에서 제공해주는 기능을 통해서 관련된 정보를 가져와서 불러오는 방식을 선택함. 각 스토어 Service 스크립트와 스토어에서 제공해주는 업적간의 중계자 역할을 하는 스크립트

**IAPManager**

* 인앱 결제를 위한 목적으로 만든 스크립트. 초기 목적은 구글, IOS 포함이었으나 유니티에서 위 스토어의 IAP를 제공해주기에 현재는 원 스토어 용으로만 사용되고 있음.

**OneStoreIAPManager**

* 원 스토어 인앱 결제 서비스를 하기 위한 스크립트

# 애드브릭스 버그 해결

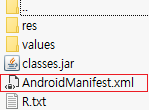
밑의 설명은 현장에서 겪었던 것을 개인적으로 재현하여 설명한 것 입니다. 2019-02-23이전 기준으로 애드브릭스를 포함하여 빌드하면 다음과 같이 앱이 2개가 설치되는 버그가 있습니다.

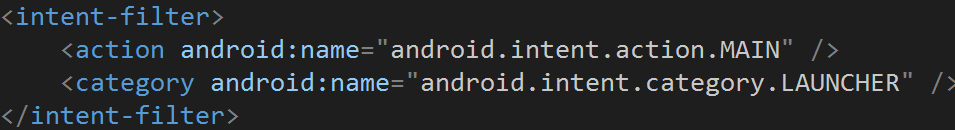


하지만 유니티에서 [AndroidManifest]을 검색해보면 하나밖에 없습니다.



애드브릭스의 aar파일들을 살펴보니 abx-unity.aar에서 다음과 같이 [AndroidManifest]를 포함하는 것을 알 수 있었으며, 이 매니페스트 파일를 열어서 실행해보니 다음과 같은 로직이 포함되어 있었습니다.





해당 로직을 제거하고 aar파일을 다시 묶어서 빌드를 하면 제대로 설치가 됩니다.

