

Unity로 다수의 인원이 개발할 때 알아두면 좋은 9가지

* Git과 Source Tree, Unity 4.5 개발 환경 기준

Unity로 다수의 인원이 개발할 때 알아두면 좋은 9가지

1. 버전 관리 시스템에 대해 알아두자
2. 프로젝트 설정 시 주의점
3. .meta 파일이 갱신되는 경우에 대해 알아두자
4. 커밋 시 주의할 점
5. 푸시 및 풀에 실패하는 경우를 극복하자
6. 충돌이 일어났을 경우의 대처법을 알아두자
7. 기능 추가 플로어
8. 문자 코드에 주의할 것
9. 어셋 서버 및 SVN, Git에 연결이 되지 않는 경우의 대처 방법

출처: <http://tsubakit1.hateblo.jp/entry/20140613/1402670011>

번역: [이근수](#), Assistant Producer, [Maverick Games](#)




오역이 발견되시면 코멘트(Ctrl+Alt+M; Mac에선 CMD+Alt+M)를 달아주시거나, gslee@maverickgames.co로 알려주세요.

1. 버전 관리 시스템에 대해 알아두자

게임 잼이나 게임 개발 시 Git과 같은 버전 관리 시스템을 쓰지 않는 사람이 있을 경우 꽤 협업이 어렵다. 특히 게임 잼과 같은 개발 속도가 빠르고 팀원들에게 여유가 없는 상황일 경우, 버전 관리에 참여할 수 없는 사람 만든 그래픽이나 시스템은 최종적으로 게임에 넣지 못하는 일이 많다.

SourceTree는 [여기](#)에서 다운로드할 수 있으며, 다음 5가지를 알아두자.

| |
|------------|
| 커밋(commit) |
|------------|

자신의 디바이스로 만든 신규 데이터 등록 () 이나 변경 () , 삭제 () 를 진행한다. 대부분은 커밋할 파일을 선택하고 커밋 버튼을 누르면 된다. 또는 스테이지(커밋 대상 일람)에 파일을 등록하고 커밋 버튼을 누른다. AssetServer나 SVN의 경우에는 함께 푸시(업로드)가 실행되기도 한다.

리셋

자신의 변경 내역을 지우고 파일을 마지막으로 커밋한 상태로 되돌린다. 신규 아이템에는 리셋이 아닌 삭제가 진행된다. AssetServer에서는 discard.


푸시(upload)

커밋한 내용을 업로드하여 저장소를 갱신한다. 다른 누군가가 푸시를 한 경우에는 실패가 뜬다. 그런 경우에는 일단 풀(업데이트)로 자신의 환경을 최신 상태로 만든 다음 다시 푸시한다.

풀 (Update)

푸시로 만들어진 프로젝트의 변경 내용을 다운로드하여 로컬에 반영한다. 풀로 갱신 중인 파일을 자신의 로컬에서 갱신하려할 경우 실패가 뜬다. 풀로 갱신된 파일을 이미 커밋해버린 경우, 충돌이 발생한다

머지 (해결)

풀로 다른 사람의 커밋과 자신의 커밋이 충돌() 한 경우 누구의 커밋을 적용할 지 결정하는 조작. 서로의 변경점을 비교해주는 경우도 있다. 프로젝트 관리 시 일어나는 대부분의 문제가 바로 이것.

버전 관리와 무관하게 파일을 주고 받을 경우에는 unitypackage를 사용하는 편이 좋다. unitypackage에는 .meta 정보가 포함되므로, 이 파일 포맷으로 전송하면 참조 내용이 빠지는 일은 없다. 어셋스토어에 올라온 어셋도 unitypackage로 등록되므로, 팀원 모두가 같은 어셋을 다운로드하면 문제 없이 사용할 수 있다.

unitypackage가 아닌 다른 방법으로 파일을 주고 받을 경우 받은 사람이 반드시 커밋을 해야 한다. 실수로라도 여러 사람에게 보내면 안된다. 메타 파일이 중복 생성되어 프로젝트가 혼란에 빠지게 된다.

unitypackage로 파일을 공유하며 프로젝트를 진행할 경우, dropbox와 같은 것으로 관리를 하면 '최신'이 어떤 것인지 알기 쉬우므로 상당히 좋다. 버전 관리보다 강제적인 방법이기도 하나, '최신 상태 유지'를 우선적으로 할 수 있다. 추천은 하지 않지만.

2. 프로젝트 설정시 주의점

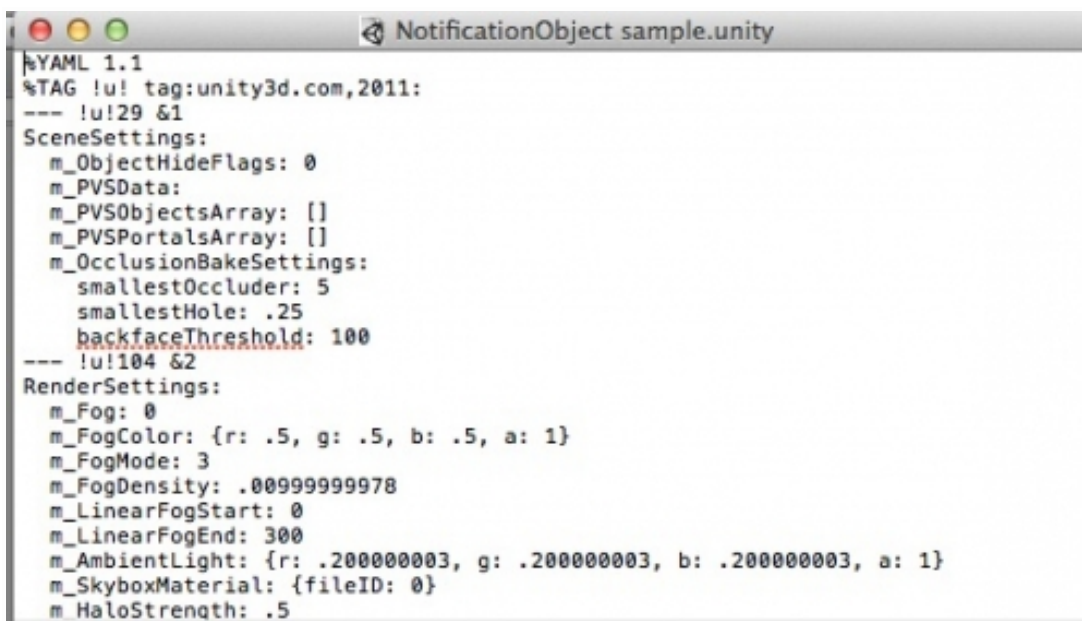
Unity로 프로젝트를 만들 때 해야 할 2가지.

Unity 버전을 (마이너 버전 포함) 통일할 것과 meta 포맷을 'force text'로 설정하는 것이다.

일단 **Unity 버전을 통일하는 것이** 대단히 중요하다. Unity는 어느 정도 상위/하위호환은 되지만 버전 차이에 따라 변환을 해야하는 경우도 생긴다. 특히 **씬이나 메타 파일의 파라미터는 에디터의 버전에 따라 미묘하게 달라질 수 있으므로, 버전을 맞추지 않으면 불필요하게 변환하는 경우가 생긴다.** 어쨌거나 여럿이서 개발을 할 경우 각기 다른 버전을 사용함으로써 발생하는 리스크가 상당히 크다. API도 다르고, 이상한 설정 파일이 생기기도 한다.

한가지 더, **meta 포맷을 force text로 설정하는 것**. 이렇게 함으로서 prefab이나 scene과 같은 바이너리 포맷 파일을 텍스트로 확인할 수 있다. 씬을 머지하는 것이 상당히 어렵기는 하지만 아예 안되는 것보다는 낫다. prefab 머지는 의외로 어떻게든 된다.

force text 설정은 [Menu] -> [Edit] -> [Project Settings] -> [Editor]를 연 다음 [Asset Serialization]를 [Force Text]로 바꾸면 된다. 이 설정은 베이스가 될 프로젝트에만 해두어도 된다. 참고로 **ForceText는 최근 Unity 무료판에서도 사용할 수 있게 되었다**.



```
YAML 1.1
%TAG !u! tag:unity3d.com,2011:
--- !u!29 &1
SceneSettings:
  m_ObjectHideFlags: 0
  m_PVSDData:
  m_PVSObjectsArray: []
  m_PVSPortalsArray: []
  m_OcclusionBakeSettings:
    smallestOccluder: 5
    smallestHole: .25
    backfaceThreshold: 100
--- !u!104 &2
RenderSettings:
  m_Fog: 0
  m_FogColor: {r: .5, g: .5, b: .5, a: 1}
  m_FogMode: 3
  m_FogDensity: .009999999978
  m_LinearFogStart: 0
  m_LinearFogEnd: 300
  m_AmbientLight: {r: .200000003, g: .200000003, b: .200000003, a: 1}
  m_SkyboxMaterial: {fileID: 0}
  m_HaloStrength: .5
```

git으로 관리하는 경우에는 다음 항목을 ignore에 추가해야한다. 이 항목은 뭔가 바뀔 때마다 갱신되는 데다가 .meta와 소스 코드만 있다면 문제없이 돌아가는 부분들이다.

```
[L]ibrary/
[T]emp/
[Oo]bj/
# Autogenerated VS/MD solution and project files
```

```
/*.csproj
/*.unityproj
/*.sln
/*.suo
/*.user
/*.userprefs
/*.pidb
/*.booproj
```

3. .meta 파일이 갱신되는 경우에 대해 알아두자

meta 파일은 Unity 프로젝트 임포트 시 생성되며 각 파일에 대한 정보를 가진다. 예를 들어 Texture 압축 포맷, 모델 설정 등등.

또한 프로젝트에서 파일을 참조하는 데 필요한 UUID (유니크한 ID) 를 가진 것도 이것이다. 따라서 **meta 파일이 재생성되어 UUID 값이 바뀔 경우, Unity 프로젝트에서 참조를 잃게 된다.** 메타 파일이 생성, 파기되는 패턴은 다음과 같다.

Unity Project가 아닌 방법으로 파일을 옮기거나, 리네임한 경우

Finder나 Explorer로 파일을 조작한 경우 Unity 에디터는 '파일이 사라졌다'고 판단, 현재 meta 파일을 삭제하고 새로운 메타 파일을 작성한다.

meta 파일이 없는 경우

예를 들어 Assets에 어떤 파일을 넣었을 때, 파일명+.meta 파일이 없을 경우 자동적으로 파일명+.meta 파일이 만들어진다. 따라서 meta를 커밋하지 않고 푸시해버리면 팀원 전체가 meta를 생성하는 참극이 벌어진다.

meta 파일의 원본 파일이 없을 경우

메타파일이 참조하는 원본 파일이 없을 경우, 메타 파일은 지워진다. 이 판단은 「***.meta」의 ***와 같은 이름의 파일 유무에 따라 진행된다. 따라서 메타 파일만 남기고 파일을 갱신하면 파일에 덮어쓰기가 될 가능성이 있다.

의도치 않게 메타 파일이 생성, 삭제되면 문제가 생기므로, 파일 덮어쓰기용 이외에는 **Project 뷰에서 파일을 조작하는 것이 정석이다**. 어쩔 수 없이 파일을 옮겨야 할 때에는 .meta 파일도 함께 옮길 것.

반대로 메타 파일이 갱신되는 패턴은 단순하다. **Unity 에디터에서 파일 파라미터가 변경될 경우 메타 데이터가 갱신된다**.

예를 들어 텍스처 설정을 Texture에서 Sprite로 바꾸면 메타 데이터도 갱신된다. 이러한 경우, 메타 데이터를 갱신하지 않고 풀한 사람 쪽에서는 제대로 작동되지 않을 수 있다.

메타 파일이 덮어쓰기되는 타이밍은 **실행 저장 및 Save Project로 프로젝트를 저장하거나, Unity Editor를 종료하는 타이밍**. 따라서 이 항목을 실행하지 않은 상태로 에디터가 다운되어버릴 경우 .meta 데이터가 갱신되지 않을 가능성이 있다(경우에 따라서는 프리팹 내용이 사라진다). 이런 경우를 대비해 에디터 확장으로 자동 갱신되는 빈도를 조절할 수 있다. 예를 들면 실행 자동 백업 기능을 사용하면 일정 시간마다 메타 파일이 갱신된다.

참고로 **파일을 덮어쓰기 갱신하고 싶은 경우 Finder, Explorer로 직접 파일을 덮어쓰면 된다**.

이것은 어째서인지 Unity의 Project 뷰에서는 파일 덮어쓰기가 되지 않는 문제에 대한 대책의

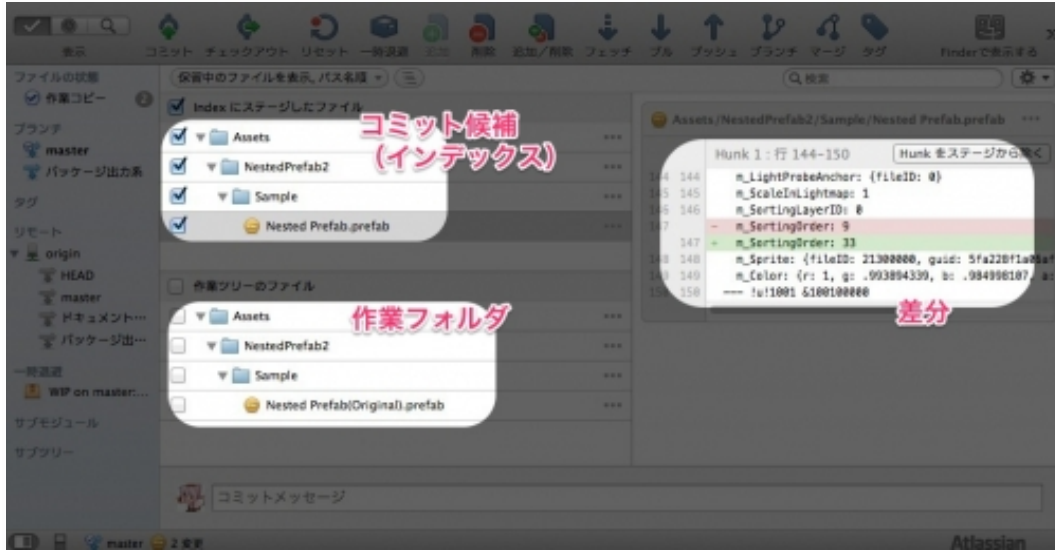
한 가지이며 **meta 파일과 함께 같은 이름의 파일이 있을 경우 meta가 새로 생성되지 않는 점**을 이용한 꼼수다. 물론 같은 이름의 파일을 드로그&드랍으로 추가할 경우 덮어쓰기를 실행하는 스크립트를 만들면 해결할 수도 있으나, 숫자로 끝나는 파일에는 사용할 수 없는 등 문제가 있어 Finder 사용을 추천하는 바이다.

한 가지 더. git의 버전 컨트롤 설정 시 여러 사람의 메타 파일 설정이 다를 경우, 풀을 할 때마다 매번 다른 내용이 돌아오는 일이 생긴다. 예를 들면 ForceText로 설정한 사람과 Binary로 설정한 사람이 있을 경우, 메타 파일이 자동으로 바뀌어 버린다. 또한 Version Control 설정이 meta인 사람과 asset server인 사람이 있을 경우도 그러하다. 이 역시 메타 파일이 바뀌어 버린다(멋대로 메타 파일이 바뀐다고 생각하는 사람들의 경우 대부분 이 문제. 이게 아니면 Unity 에디터 버전 문제다) .

기본적으로(에디터 확장으로 메타 파일을 갱신하는 기능이 없는 한) 메타 파일이 멋대로 바뀌는 일은 없으므로, 이 점을 주의하도록 하자.

4. 커밋 시 주의할 점

커밋을 할 때, 기본적으로 **자신의 작업물을 반영하는 데 필요한 파일 만을 커밋**해야한다. 의외로 자주 있는 일이 작업 전체를 커밋하는 경우인데, 팀에 이런 사람이 있는 경우 잦은 충돌이 일어나 문제가 생긴다. 커밋을 되돌리는 것은 상당히 귀찮은 작업이므로, 커밋은 신중하게 하도록 하자. 참고로 소스 커밋의 경우에는 행 단위로도 할 수 있다.



커밋 시 특히 주의할 점은 파일을 커밋할 때 메타 파일(.meta)을 함께 커밋하는 것이다. .meta 파일은 Assets 폴더 아래로 파일을 배치할 때 파일명+.meta 형식으로 별도 생성된다. 메타 파일은 파일 참조나 파라미터를 관리하는 파일이므로 반드시 커밋해야 한다.

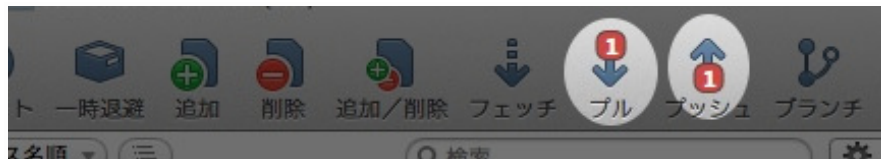


만약 프로젝트를 풀한 다음 .meta가 생성되었다면 누군가가 .meta를 커밋하지 않았다는 것이다. 그런 경우에는 해당 작업자가 메타 파일을 커밋하도록 지시해야 한다. 다른 작업자들은 메타 파일을 리셋해서 없던 것으로 처리하는 것이 가장 좋은 방법이다. 무슨 일이 있더라도 신규 .meta 파일을 커밋해서는 안된다.

또 하나가 남았다. 커밋 전에 싹을 저장하거나 에디터를 닫고 project를 저장하기를 권한다. 이유는 위에 쓴 것과 같이 프리팹 파일 갱신 타이밍이 그 때이기 때문이다. 따라서 이렇게 하지 않으면 프리팹이나 메타 데이터가 갱신되지 않는 경우가 생긴다.

5. 푸시 및 풀에 실패하는 경우를 극복하자

개발 중 푸시에 실패하는 경우가 있다. 대부분의 원인은 **풀을 해야하는 데 푸시를 하는** 경우다. 버전 관리 시스템에서는 동일한 브랜치의 경우 항상 최신 상태로 개발을 진행하게 되므로, 풀을 할 수 있는 항목이 있는 경우에는 푸시를 할 수 없다.



또 한 가지 가능성이 있다면 **푸시할 권한이 없는** 경우다. 그럴 때에는 풀 리퀘스트를 보내거나 권한을 받도록 한다. 원인에 대한 에러 메시지가 표시되므로 잘 확인하도록 하자.

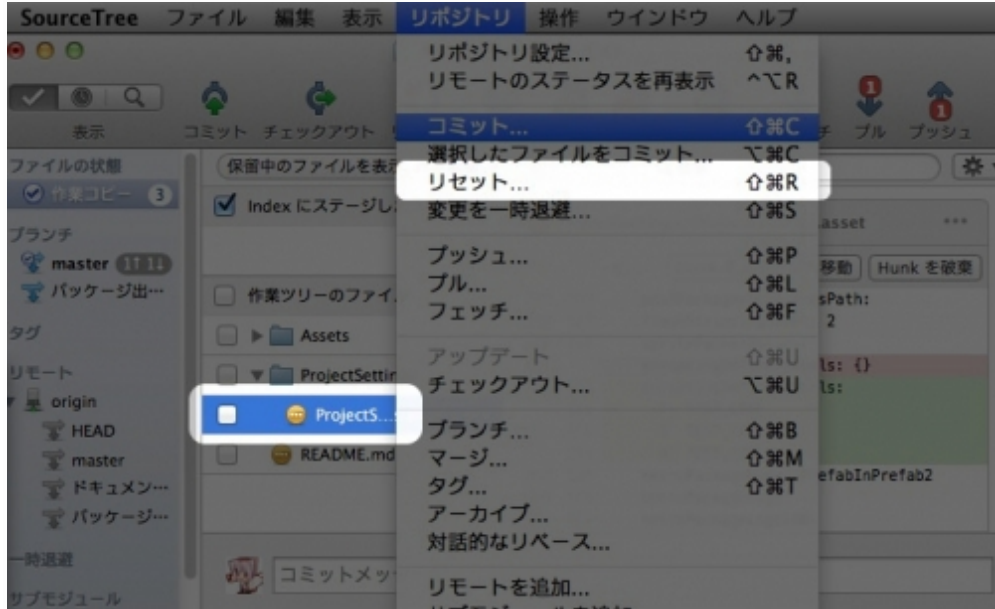
풀을 할 수 없는 경우는 대체로 **풀을 통해 갱신될 파일을 로컬에서 갱신하고 있는** 경우다. 예를 들어 아래와 같이 내가 README.md 변경을 푸시하고 있는 중에 다른 사람이 README.md를 바꾸고 있다면, **충돌이 일어난 파일 목록을 받아와 리셋(discard)하여 충돌을 해결**할 수 있다.

```
ブランチ "master" を "origin" からプルしています
キャンセル

git -c diff.mnemonicprefix=false -c core.quotepath=false -c credential.helper=sourcetree fetch
origin
From https://github.com/tsubaki/NestedPrefabUni
756a0d1..3ef7626 master -> origin/master

git -c diff.mnemonicprefix=false -c core.quotepath=false -c credential.helper=sourcetree pull --
no-commit origin master
From https://github.com/tsubaki/NestedPrefabUni
* branch      master      -> FETCH_HEAD
error: Your local changes to the following files would be overwritten by merge:
  README.md
Please, commit your changes or stash them before you can merge.
Aborting
Updating 756a0d1..3ef7626
Completed with errors, see above

閉じる
```

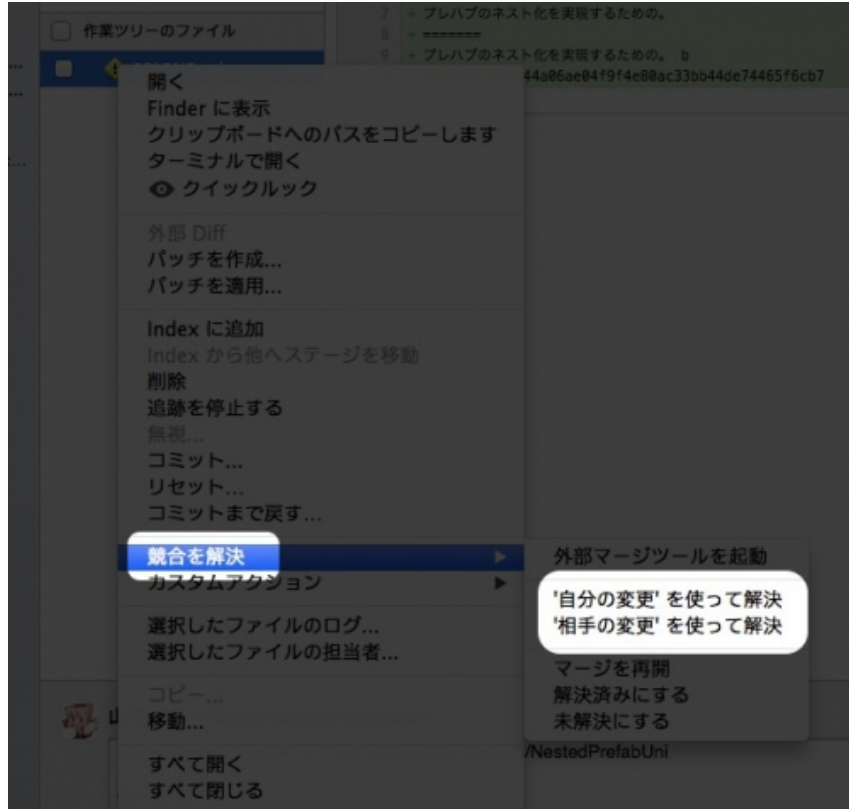


만약 자신의 변경 내용을 리셋하고 싶지 않다면 변경 내용을 일시 취소한 다음 나중에 풀하면 풀을 진행할 수 있다. 그 뒤 취소 내용을 다시 돌리면 충돌이 일어나므로 머지해야 한다. 신규 아이템은 취소할 수 없으므로 삭제해야 한다.

외부 머지툴(Win merge 등)이 없을 경우에는 머지할 수 없으므로 상대방의 변경 내용을 적용시키도록 한다.

6. 충돌이 일어났을 경우의 대처법을 알아두자

다수의 커밋이 충돌하게 되는 경우, 작업을 계속하기 위해서는 머지를 해야 한다. 머지를 하려면 **컴플릭트한 파일을 선택해 충돌 해결 -> 상대방의 작업을 변경해 충돌을 해결한다...** 는 방법이 있다. 어쨌거나 외부 머지 툴로도 해결이 어려운 경우에는 이것이 기본이다. 외부 머지 툴을 사용하면 좀 더 세부적으로 머지를 할 수 있다.



씬을 머지하는 건 상당히 귀찮은 일이다. 단순한 구조나 파라미터 차이 정도라면 머지를 할 수 있지만, 부자관계 변경이나 복잡한 참조가 끼어들게 되면 난이도가 높아진다.

일단 어셋 등에서 씬을 머지하거나 리플렉션으로 씬 구조를 취득, 나누기를 작성해 적용할 수는 있지만, 상당히 번거로우므로 관리 주체가 담당하는 쪽이 가장 편하다. 이 부분은 아직 연구 중이므로 결과가 나오면 공개하겠다.

프리팹 머지는 씬 머지보다는 간단하나 계층이 깊어질 수록 이론 상으로는 가능해도 현실적으로는 어려우므로, nested prefab 등 방법으로 계층의 깊어지지 않게 하거나 외부 파라미터가 자동 조정되도록 해두자.

7. 기능 추가 플로우

기능을 추가할 경우에는 다음과 같은 순서로 작성한다.

1. 기능 테스트용 씬 작성
2. 1에서 작성한 씬 안에 기능을 만든다
3. 게임용 씬에 넣는다
4. 커밋 (& 푸시)

게임용 씬이 대단히 복잡하거나 스테이지 등 장시간 조절이 필요한 경우에는 **씬을 분할한 다음 실행 시에 결합하도록 하자.**

다수의 인원이 Unity 프로젝트를 공동 개발하는 방법의 일안

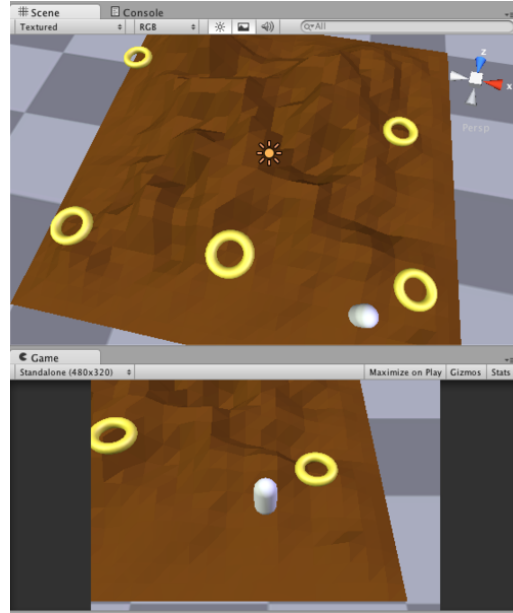
다수의 인원으로 Unity 프로젝트를 개발할 경우 가장 문제가 되는 것이 씬의 공동 편집이다.

Unity 씬 파일은 머지할 수 없는 바이너리 형식이므로 다수의 인원이 동시에 편집을 하면 반드시 누군가의 편집 내용을 버려야만 하게 된다.

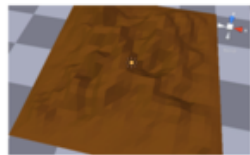
이 문제를 해결하는 방법을 몇 가지 생각한 결과 개인적으로 가장 관철을 못한 것이, 씬을 세부적인 단위로 나눈 다음 **씬 추가 읽어들이기**를 사용해 통합시키는 방법이다.

씬 추가 읽어들이기는 현재 씬 위에 지정한 씬을 겹쳐서 읽어들이는 기능이다.

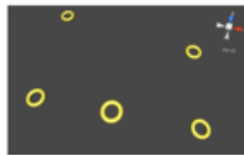
예를 들어 이런 프로젝트가 있다고 하자. 지형 위에 동그라미가 몇 개 배치되어 있고 그 위에 조작할 수 있는 흰색 캡슐이 움직이고 있다.



이것을 "BG"와 "Level"과 "System" 총 3가지로 나눈다.



BG

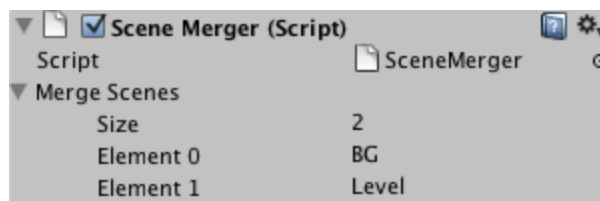


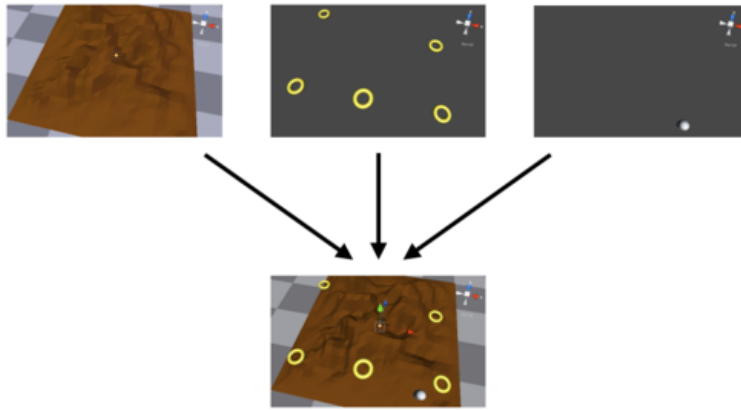
Level



System

그리고 각각의 씬에 SceneMerger라는 스크립트를 추가한다. 이 스크립트는 게임 개발과 함께 다른 2개 씬을 추가로 읽어들이는 스크립트다.



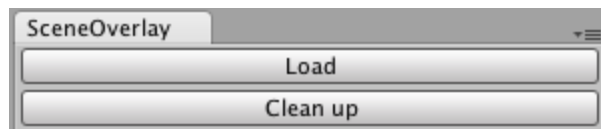


이제 어떤 씬을 편집하더라도 게임을 시작하면 같은 상태가 된다.

이렇게 씬을 담당자 숫자만큼 나누어 각 담당자가 자기가 맡은 씬만 편집하도록 하면 충돌은 피할 수 있을 것이다.

단, 이 방법은 게임을 시작하기 전에는 합쳐진 상황을 볼 수 없으며, 편집 작업을 하기에 어렵다. 가능하다면 통합한 상태로 편집을 하고, 저장할 때만 나뉜 상태로 되돌리는 기능이 생겼으면 한다.

이것이 가능하도록 에디터 스크립트를 만들어 사용해 보았다. "SceneOverlay" 윈도우 안에 그 기능을 모아두었다.



"Load" 버튼을 누르면 다른 씬을 추가로 읽어온다. 이에 따라 추가된 게임 오브젝트는 "_overlay"라는 이름의 게임 오브젝트의 자식 계층에 들어간다.

"Clean up" 버튼을 누르면 "_overlay"가 삭제되어 나뉘어 있던 원래 상태로 돌아간다.

또한 씬 저장 액션을 후크하는 스크립트도 만들어 씬 저장 시에 자동으로 나뉘게 했다.

이 방법을 사용하려면 스테이지에 배치된 기믹 각각이 독립적으로 움직이는 형태로 만들어야 한다. 이유는 씬 로드가 즉각 일어나는 것이 아니므로, 오브젝트 간 연계 타이밍이 꽤 힘들기

때문이다. 이런 경우에는 NotificationObject나 NotificationCenter와 같은 통지 센터를 활용하는 것이 좋다.

[Unity Wiki Tips - 메시지 통지 센터](#)

다수의 적 캐릭터를 만들어 충돌이 일어날 때마다 SendMessage를 사용한 코딩을 하면 나중에 코드를 정비하기 어려워진다.

Unity Wiki에서 NotificationCenter라는 좋은 해결책을 찾았으므로 여기에 소개한다.

C# 스크립트는 링크 가장 하단에 있으므로 카피해두는 걸 추천.

<http://wiki.unity3d.com/index.php?title=CSharpNotificationCenter>

사용 방법

이것은 자동으로 인스턴스화된 싱글톤 클래스로 오브젝트에 추가할 필요 없이 프로젝트 어셋에 넣어 DefaultCenter에 호출만 한다. 이 static 메소드의 호출에 의해 게임 오브젝트 작성, NotificationCenter 컴포넌트 추가, 컴포넌트 참조 작성이 자동으로 진행된다.

PostNotification()를 통한 통지 투고(Post) 가능 :

```
NotificationCenter.DefaultCenter.PostNotification(this, "OnBumperCollision");  
NotificationCenter.DefaultCenter.PostNotification(this, "OnBumperCollision", anyObject);  
NotificationCenter.DefaultCenter.PostNotification(new Notification(this, "OnBumperCollision");  
NotificationCenter.DefaultCenter.PostNotification(new Notification(this, "OnBumperCollision", anyObject));
```

PostNotification() 메소드는 몇 가지 호출 방법이 있다. 필수 파라미터는 단 2가지(수신자, 통지명). 옵션에서 데이터 오브젝트도 파라미터로 수신하게 할 수 있으며 송수신되는 것만 있으면 어떤 것이든 괜찮다. 예를 들어 OnBumperCollision 통지가 Collision 관련 정보를 송신해도 되며, Collision 오브젝트 자체를 보낼 수도 있다. 많은 경우에는 데이터 오브젝트는 해쉬 테이블로 사용하며 데이터 오브젝트 키와 한 쌍으로 여러가지 정보를 담을 수 있게 한다.

유의할 점은 PostNotification()는 각각의 파라미터를 넘길 수 있으며, Notification 오브젝트를 넘길 수도 있다. 통지 오브젝트는 서브클래스화할 수 있다. 통지 데이터 세트를 만들 경우. 추가로 Notification 오브젝트는 필수 파라미터는 2개(송신자, 통지명) 뿐이므로 옵션에서 데이터 오브젝트도 파라미터로 수신할 수 있다.

addObserver 메소드 호출로 통지를 수신할 수 있게 등록 가능 :

```
NotificationCenter.DefaultCenter.AddObserver(this, "OnBumperCollision");
```

DefaultCenter () 는 디폴트 NotificationCenter의 인스턴스를 돌려보내는 static 메소드이며, 존재하지 않을 경우에는 자동으로 인스턴스화된다. AddObserver () 통지는 2개의 파라미터가 필요하다(수신하려는 오브젝트명, 등록할 통지명). 등록할 통지명은 이벤트를 실행하는 함수명이다.

RemoveObserver () 메소드로 옵저버 삭제 가능 :

```
NotificationCenter.DefaultCenter.RemoveObserver(this, "OnBumperCollision");
```

RemoveObserver () 는 지정된 통지 이벤트의 옵저버 리스트에서 오브젝트를 삭제한다.

통지를 수신하기 위해 통지명을 가진 메소드를 넣는다 :

```
using UnityEngine;
```

```
using System.Collections;
```

```
public class Test : MonoBehaviour {
```

```
    // Use this for initialization
```

```
    void Start ()
```

```
    {
```

```
        NotificationCenter.DefaultCenter.AddObserver(this, "OnBumperCollision");
```

```
    }
```

```
    void OnBumperCollision ()
```

```
    {
```

```
        // do something...
```

```
    }
```

```
}
```

게임 매니저 등 어떤 클래스에서도 접속하는 것은 프리팹화하여 해당 항목을 머지하는 쪽이 좋다. 그 경우 프리팹 충돌에 주의해야 한다.

게임용 씬과 합치는 것은 팀 리더 등 특정한 1인이 전담하는 방법도 있다. 이 경우에는 기존 씬과 합치기 좋도록 기능 테스트용으로 작성된 씬은 편집하기 쉽게 만드는 편이 좋다(컴퍼넌트 복사가 가능하거나 프리팹화할 수 있는 등).

또한 SVN이나 VSS와 같은 파일을 잠글 수 있는 시스템을 버전 관리 시 사용하는 경우에는 **싹을 편집할 때 잠글 수 있도록함으로서 게임용 싹에 넣는 작업 중 다른 사람의 싹 편집을 방지**할 수 있다. 이 경우 에디터 확장으로 어떤 서비스나 파일이 연결되어있는 지 확인할 수 있게 하는 편이 좋을 것이다.

첫 번째 단계에서 만든 '테스트용 싹'을 작업자 이름을 폴더명으로 지정해 관리한다면 프로젝트를 혼선없이 비교적 심플하게 저장할 수 있다. 이것은 브랜치를 끊어도 문제 없고 여기에서만 사용할 수 있는 스크립트가 있는 경우에는 싹 이름과 같은 폴더를 만들어 관리하면 알기 쉽다. 또 라벨을 붙여 관리하는 것 역시 좋은 선택이다.

8. 문자 코드에 주의할 것

개발팀 안에서 Windows와 Mac을 함께 쓰고 있다면 문자 코드에 주의해야 한다. 문자 코드가 UTF-8 BOM이 아닌 경우, Windows의 경우에는 일본어(한국어) 코멘트 뒤에 「. (닷)」을 부여하지 않으면 다음 행이 무시되는 버그가 있다.

Unity에서 C# 신규 작성 파일 UTF-8 BOM으로 만드는 명령어(Mac용)이나 스크립트파일 UTF-8변환(Windows)을 사용해 프로젝트 내부 문자 코드를 처리하거나 「. (닷)」을 일본어(한국어) 코멘트 뒤에 붙이도록 하자.

UTF-8 BOM으로 만드는 명령어(Mac용)

다음 명령어를 터미널에서 실행하면 C# 서식 파일이 수정된다. 중복 실행하면 BOM 데이터가 추가되므로 반드시 1번만 실행하자.

```
cd /Applications/Unity/Unity.app/Contents/Resources/ScriptTemplates;fn=81-C#
```

```
Script-NewBehaviourScript.cs.txt;fo=${fn}.org;mv "$fn" "$fo";echo -en '\xef\xbb\xbf' > "$fn"; cat "${fo}" >> "$fn"
```

실수로 중복 실행했을 경우에는 아래 파일을 바이너리 에디터로 열어 앞 단의 6byte만 0xEF 0xBB 0xBF 0xEF 0xBB 0xBF 이 되게 만든다. 혹시 7byte 이후 0xEF 0xBB 0xBF의 3byte 페어가 있을 경우에 삭제해야 한다.

```
/Applications/Unity/Unity.app/Contents/Resources/ScriptTemplates/81-C#\ Script-NewBehaviourScript.cs.txt
```

JavaScript나 Boo 서식인 경우에는 BOM 없는 UTF-8도 딱히 문제가 없으므로 이러한 명령어를 만들지 않았다.

스크립트파일 UTF-8변환 방법(Windows)

신규 스크립트 생성 시 사용되는 스크립트 템플릿 파일이 다음 폴더에 있다.

```
C:\Program Files (x86)\Unity\Editor\Data\Resources\ScriptTemplates
```

이 파일을 UTF-8(BOM 사용)으로 바꿔도 Unity에서 클래스명을 설정하고 덮어쓰기 저장하면 us-ascii로 되돌아가버린다. 그동안 스크립트 신규 생성 시에는 일반 텍스트 에디터로 열어 저장해왔었으나, AssetPostprocess로 편집하면 이 문제가 없다는 점을 깨닫고 이것저것 만들어보았다.

```
AssetPostprocessUTF8Encode.cs
```

<https://github.com/sharkattack51/Unity-EditorScript/blob/master/AssetPostprocessUTF8Encode.cs>

파일 encode를 받아오는 부분은 다음을 참조했다.

<http://dobon.net/vb/dotnet/string/detectcode.html>

9. 어셋 서버 및 SVN, Git에 연결이 되지 않는 경우의 대처 방법

어셋 서버가 연결되지 않을 경우에는 대상 서버 문제인 경우가 많다. ping 등의 방법으로 문제가 없는 지 확인하자.

대학교나 PC방 등 동일한 게이트웨이를 사용하는 경우에도 접속이 되지 않는 경우가 있다. 특정 PC(우선 LAN이 연결된)을 호스트기로 지정하고 다른 PC를 연결하면 된다.

또는 Bitbucket 같은 서비스를 사용하면 되나 SSL 설정 등이 귀찮다. Git의 경우에는 대부분 SSL 문제다.