별의 커비 모작



구현할 기능.

1. 커비의 변신기능 그리고, 변신에 따른 다양한 모션
2. 헬퍼 알고리즘과 몬스터 알고리즘
3. 헬퍼 시스템 (2인 플레이 전환 과정)
4. 맵툴을 이용한 스테이지 저장 및 읽기 과정 (이거 힘들 것 같다.)

// 리소스 자체가 타일맵 파일이 거의 없음. 대부분이 통맵으로 이루어짐.

1. 스테이지 모작을 통한 보스 패턴 구현.

해야하는 것 (위의 번호와 매칭)

1. 커비의 다양한 변신을 위한 리소스를 준비한다. 포토샵 편집이 필수.

참고로 커비 이미지는 원본의 5배로 설정했다. (DS 크기의 이미지가 원본임)

1. 게임을 플레이하면서 헬퍼의 행동을 보고, 어떤 식으로 플레이어를 도와주고,

몬스터를 처리하며, 어떤 알고리즘을 통해서 몬스터가 움직이는지,

각 몬스터마다 움직임 패턴 등을 분석하여 적용.

1. 헬퍼 알고리즘을 구성하면 자연스럽게 전환할 수 있는 부분이라 생략함.
2. 기본적인 맵툴의 편집기능을 갖추고, 해당 타일마다 어떠한 정보를 추가적으로 가지도록 저장하여 이것을 읽어서 사용할 수 있도록 만드는가. 충돌과 관련된 정보를 포함하는 것은 물론이고, 해당 스테이지에서 나오게 할 몬스터에 대한 정보도 심을 수 있다면 좋을 것 같다. 결국에는 맵툴뿐만이 아니라, 몬스터에 대한 정보까지 포함할 수 있는 툴을 제작하는 것이 목표.
3. 보스 패턴을 구현하는 것은 어렵지 않을 것 같다. 다만, 기존의 커비는 너무 난이도가 쉬운 편이니 더 어렵고 힘들게 구현하는 것이 좋겠다.

메모장.

현재 수업 프레임워크에서는 Monster에게 InputComponent를 줌으로써, (캐릭터 클래스 상속) 몬스터를 플레이어로 정하고 싶은 경우에 바로 옮겨줄 수 있는 처리가 되어있다.

나도 이런 시스템이 필요하기는 하다. Helper가, 몬스터와 동일하게 생겼고,

그럼에도 몬스터와 다른 점은 UI추가, 알고리즘… 이런 부분이다.

기본적인 베이스를 몬스터에 깔고 간다. 리소스 부분이나, 알고리즘 부분이.

쉽게 말해서 몬스터의 적이 몬스터로 바뀌고 플레이어 따라다니는 개념이 헬퍼이다.

따라서 몬스터에게 캐릭터클래스를 상속하고 여기서 여러 몬스터를 구현. (상속상속상속)

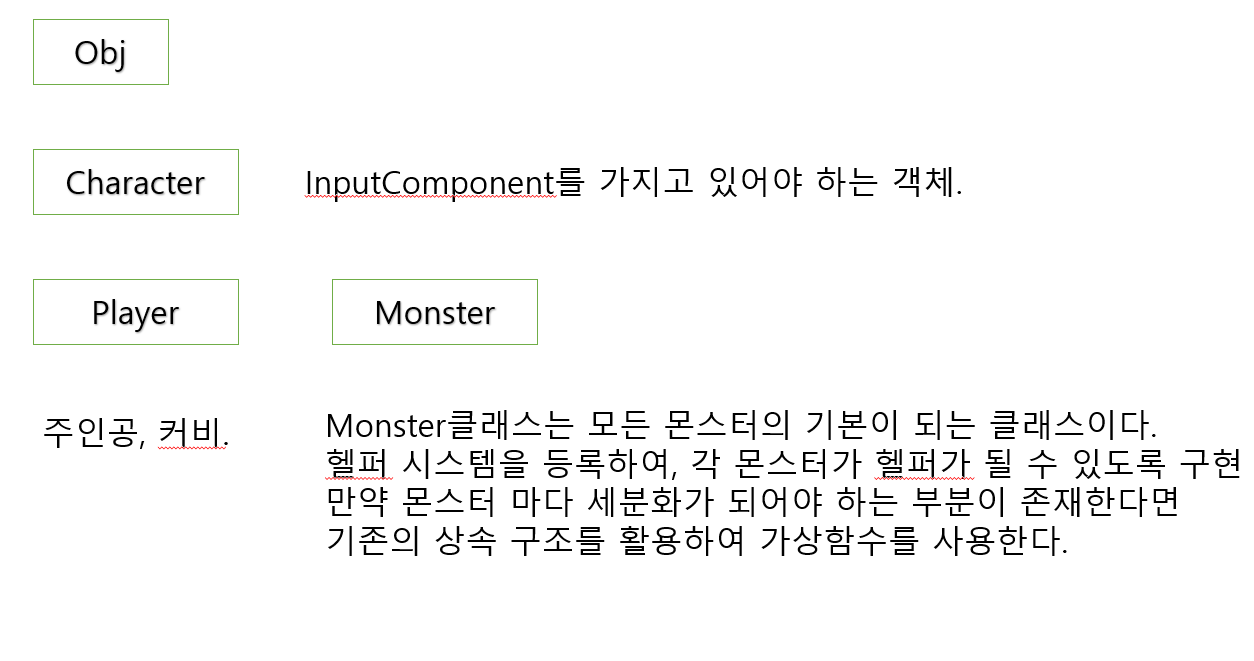
몬스터에 헬퍼 시스템을 구현하여, 만약 이 몬스터가 헬퍼라면, 플레이어에게 도움을 줄 수 있는 알고리즘을 추가하자.

그러면, 몬스터 클래스를 기준으로 하여 헬퍼시스템을 가지고 있고 (여기서 몬스터의 리소스를 가져다 쓰기 좋을 것이다.) 몬스터는 기본적으로 헬퍼가 될 수 있다.

// 각 몬스터마다 조금씩 다른 처리를 해줘야 하는데 이런 것은 상속으로 해결하되,

헬퍼시스템은 몬스터에 추가하여, 모든 몬스터들에게 일괄적으로 적용하는 시스템이 있으면 좋을 것이다.

만약 이 부분에서 세분화가 필요하다면 상속시스템을 활용하면 될 것이다.

모

몬스터나, 헬퍼는 뛰는 모션이 없다. 즉, 뛰는 것은 플레이어의 고유한 성질이다.

커비 리소스를 보면, 한쪽 다리가 긴 부분을 볼 수 있는데, 이 부분은 경사진 면에서 사용하는 리소스이다. 즉, 경사진 면에 따라 애니메이션이 달라지는 부분이다.

근데, 이런 것 하나하나 다 처리하면 굉장히 복잡하니까, 나는 기본 평면의 상태만 처리를 하도록 한다.

**Debug**

기본 커비의 애니메이션을 추가했다. 일단, 출력은 했는데, 밀려서 짤리고 움직인다.

이것을 최적의 값으로 바꾸는 처리를 …

일단, 디버깅을 모드에서 애니메이션을 교체하는 기능을 만들어서 처리해보도록 하자.

리소스를 추가하는 빈도수가 많다. (게임의 특성상)

나는 디버그모드에서 이 텍스처의 출력 위치를 변경하는 키를 등록하여 사용할 것이다.

컨트롤 + -> / 컨트롤 + <- : 출력하는 x좌표를 이동.

컨트롤 + 위 화살표 / 컨트롤 + 아래 화살표 : 출력하는 y좌표를 이동

출력하는 사이즈를 바꾸는 키.

Shift + -> / Shift + <- : 출력하는 클립의 x 사이즈를 변경

Shift + 위 화살표 / Shift + 아래 화살표 : 출력하는 클립 y사이즈 변경

출력하는 애니메이션을 바꾸는 키

ALT + -> / ALT + 위 화살표 : ++방향.

Alt + <- / ALT + 아래 화살표: -- 방향

새로운 입력시스템. 가변인자 템플릿을 활용한 간단한 입력시스템으로 해결했다.

여기에 추가적으로 애니메이션의 FrameTime을 0.1초씩 증가시키거나 감소시킬 수 있는 키를 만들었으며, 전체시간을 0.f로 만드는 입력키(P)를 만들었다.

FrameTime에 손을 대는 키는 +, - 키로 구성이 되어있다. 현재 돌리고 있는 애니메이션에 대해서 0.1s 증가 / 감소로 구성.

// 상태가 이상한 것.

AttackLeft

// 처리 완료.

RunRight

IdleRight

이미지가 앞으로 쏠린다는 의미는.

.이미지 보다 Frame의 크기가 뒤쪽에 있다는 의미

(앞으로)

이미지가 뒤로 쏠린다는 의미는

이미지 보다 Frame의 크기가 앞쪽에 있다는 의미

(뒤로)

**애니메이션 편집툴 제작**

애니메이션 편집툴의 주요 기능.

1. 한 오브젝트의 애니메이션 src이미지를 모두 출력한다.(화면영역만)
2. 이 src의 이미지에서 키보드를 통해 src의 출력 영역이 변하고, 이렇게 변한 좌표를 토대로 현재 애니메이션의 Frame을 표현해준다.
3. 편집 관리자는 Coord vector 정보를 가지고 있어서, 옮긴 Frame의 좌표에서

마우스를 떼면 좌표가 들어가고, Backspace 키를 누르면 좌표 하나를 버린다.

1. 이렇게 추가한 좌표를 토대로 현재 돌리고 있는 애니메이션의 좌표로 설정하여 보여준다.
2. 오른쪽 상단에 현재 편집하고 있는 애니메이션을 보여줌으로써 쉽게 좌표를 추가할 수 있도록 한다.
3. Save, Load기능을 만들어서, 특정 애니메이션의 vector정보로 사용할 수 있다면 좋을 것 같다.
4. 단축키 기능

W, A, S, D 카메라 이동

F1 / F2 카메라 속도 증가/감소

F3 충돌 활성화 / 비활성화

ESC 좌표 모두 삭제

**추가적 요구사항**

모든 프레임의 크기는 동일하다. 출력 위치가 다를 수는 있지만, 프레임의 크기는 동일하다. 따라서, 처음 프레임을 추가하는 경우에 FrameSize를 정해서 다음에 추가하는 프레임 정보를 제한하도록 한다. ㅇㅇ

(F3키를 눌러서 FRAME을 제한하는 기능을 만든다. 만약 어떤 프레임도 추가되어 있지 않은 경우에는 무시. 하나만 추가된 경우에 그걸로 셋팅, 둘 이상이면 마우스의 위치를 토대로 셋팅 아니면 무시)

그리고 이후에 이 설정을 다른 프레임의 정보에도 고정.ㅇㅇ

현재 마우스의 위치에서 이전에 추가한 프레임이 있었을 때 비율적으로 가운데를 누르면 단순히 위치를 이동하는 기능을 만든다. 혹은 해당 Coord정보를 바꿀 수 있도록 인덱스를 가지고 있는 것도 좋다. ㅇㅇ

이렇게 현재 클릭한 Coord정보의 위치를 바꾸는 기능을 만든다.ㅇㅇ

CTRL Z 없앴던 프레임을 다시 복구하는 기능 (순서는 보장하지 않음) ㅇㅇ

여러 프레임을 한번에 옮기는 기능 (키보드ㅇㅇ / 마우스 ㅇㅇ)

여러 프레임을 한번에 사이즈를 바꾸는 기능 ( 키보드 ㅇㅇ/ 마우스 ㅇㅇ)

이미 선택한 프레임을 다시 선택하면 풀어버리는 기능. ㅇㅇ

터무니없이 큰 값, 터무니없이 작은 값. ㅇㅇ

여러 개의 프레임을 선택할 수 있는 기능을 만든다. ㅇㅇ

툴을 조작하는 방법을 만들었으니, 이제 이렇게 설정한 프레임 좌표를 애니메이션에 적용해서 돌려주는 처리가 필요하다.

1. 오른쪽 상단에 현재 추가한 프레임의 애니메이션을 보여준다. ㅇㅇ
2. VECTOR값의 영향을 실시간으로 받도록 구성한다. (출력 영역 + 좌표 처리)ㅇㅇ
3. 좌표를 저장할 수 있는 기능을 제작한다. (애니메이션에서 필요한 정보를 통째로 저장하는 기능이 있다면 더욱 좋겠다.
4. 그 외 애니메이션 교체 단축키, 애니메이션 스피드 단축키, ㅇㅇ

애니메이션 저장하기 단축키 / 불러오기 단축키

1. 기존에 추가된 애니메이션 클립 정보를 읽어오는 기능. Load

초기화할 때, 해당 vector정보가 있다면 가져오기. ㅇㅇ

프레임의 추가가 일어나는 경우에서는 크게 문제가 될 부분은 없지만,

삭제에서는 마지막 프레임의 접근에서 문제가 발생할 여지가 있다.

그리고 단순히 벡터만 넘겨주면 끝나는 것이 아니라, 다른 업데이트 값이 있다면 그것도 업데이트 해야 한다.

근데 이 처리는 프레임이 추가 삭제 되는 경우에 해줘야 한다.

* 프레임의 개수가 달라졌다.
* 단순히 프레임 위치 수정은 전혀 문제가 되지 않음.

iFrame (바뀌면 그냥 0으로 주자) ㅇㅇ

fTime(0.f) ㅇㅇ

fPlayTime … fFrameTime 이거는 단축키를 통해 조정할 예정.

option (편집에서는 사용하지 않음) pass

그 외, FrameSize의 영향을 받도록 프레임의 사이즈를 고정시키는 순간

이 프레임 사이즈로 출력 영역을 정하는 것으로 한다. ㅇㅇ

// 현재 대상의 클립에 뭔가 저장이 되어 있을 가능성이 있다.

m\_vecCoord = m\_pEditObject->GetvecCoord();

// @@ 이 작업은 클립을 바꾸는 경우에도 진행해야 한다.

저장의 경우에는, 애니메이션 클립의 정보자체를 읽어오는거니까..

애니메이션 클립에서 필요한 정보를 모두 셋팅하는 시스템을 만들거나.

아니면, 단순히 좌표와 이름정도만 저장하는 시스템을 구성하거나…

멋있는건 전자.

string strName; string 클래스를 저장할 수는 없으니, 문자열로 변환하여 저장.

ANIMATION\_OPTION eOption; // 옵션 저장

(근데, 에디터를 통해서 옵션을 설정하는 부분이 아직 없넴. (이거 MessageBox로 물어봐도 ? ㄴㄴ

그냥, 키 하나 잡고서 그걸로 셋팅해서 저장하는 것으로 한다.)

// 당연히 edit에서는 loop로 돌리고…) (나중에 불러온 데이터를 당연히 옵션값 바꿀 수 있도록 한다)

ANIMATION\_TYPE eType; // 그냥 아틀라스주자. 편집이 아틀라스.

class CTexture\* pTexture; // 이거는…

unsigned int iColorKey; // 저장

bool bColorKey; .// 저장

int iFrame

; // 굳이 저장해야 하나 pass 0데입.

// vector의 size저장. 이후에 각 프레임의 좌표저장.

float fTime; // pass 0.f

float fPlayTime; // 저장.

float fFrameTime // 저장.

float fOptionTime; // pass 0.f

float fOptionPlayTime // 옵션 시간 저장.

vector<ClipCoord> vecCoord;

읽어온 정보는.. 일단 구조체에 기본정보로 셋팅하고서, 위에서 읽어온 순서대로 그대로 값에 데입해주면 될 것 같다. ㅇㅇ

// 애니메이션에 미리 텍스처의 경로와, 이름을 셋팅하여 생성만 해두면, 클립정보는 읽어와서 데입하는 시스템을 만들어뒀다.ㅇㅇ

편집모드에서 돌리는 모든 애니메이션은 AO\_LOOP옵션이여야만 하므로, 초기화할 때 설정하는 애니메이션을 그렇게 적용했고 또 다른 애니메이션으로 바꾸는 경우에 이 처리를 해주고 있다.(잘했네) ㅇㅇ

이제 우리는 저장을 하는 경우에 CTRLS 옵션값을 지정해서,ㅇㅇ 설정해주는 시스템을 구현하고 이것을 읽어오는 기능을 만든다. ㅇㅇ만약에 옵션값을 설정을 하지 않은 상태라면 저장을 막고,(messagebox) 아무튼 여기서 설정한 값을 옵션값으로 하여 저장을 한다.ㅇㅇ 인게임에서는 편집모드가 아니라 불러온 이후에 옵션이 그대로 들어가겠지만, 만약 편집모드에서 들어온 경우에는 상관없이, 루프로 설정된다.ㅇㅇ

결론 : 편집모드에서는 불러와도 loop 아무튼, 읽어오는 것만 보면 loop가 아닌 값으로 읽어올 수 있다.ㅇㅇ

// 참고로 불러오는 경우에는 현재 돌리고 있는 클립의 이름으로 읽어온다. 파일이 없는 경우에는

처리 fail ㅇㅇ

// 저장시에는 현재 클립의 이름 + 날짜로 저장해버린다.ㅇㅇ

저장을 하는 경우에는, 현재 클립의 정보를 저장하고, 불러오기를 하는 경우에는 파일이름에 따라 읽어온 파일의 데이터가 달라지며, 해당 클립의 정보를 찾아서 데입하는 시스템이다. ㅇㅇ

물론, 불러오는 클립의 정보와 읽어오는 클립은 같은 것으로 설정할 것이다. ㅇㅇ

이렇게 애니메이션을 바꿔가면서 따로따로 저장/불러오기 작업을 거쳐서 파일을 만든 이후에

Init에서 해당 부분을 Load를 이용해서 수정하면 끝이다.



CtrlV를 통해서 현재 선택한 프레임을 복사하는 기능을 만든다.

선택한 프레임이 있는 상태에서 CTRL V 키를 누르면, 근처 위치에 선택한 프레임을 복사해주고

기존에 선택했던 프레임은 편집상태에서 풀리게 되고 새로 생긴 프레임만 새로 편집상태가 된다.

(이후에 이것을 가지고 바로 처리할 수 있는 시스템이다) ㅇㅇ

다른 작업(마우스의 위치로 판단)을 하는 경우에는 모든 입력처리를 막도록 했다. ㅇㅇ

저장하는 시스템까지 만들어 두었으니, 이제 애니메이션을 추가하는 작업을 진행해보자.

현재 애니메이션 시스템은 어떤 좌표도 잡혀 있지 않고, 단순히 초기값만 있는 상태에서의 동작을 정의하지 않았다. 따라서, 이 부분에 대한 해결책이 2가지가 있는데,

하나는 vector, Animation 에서 절대로 터지지 않도록 예외적인 case를 만들거나

혹은 아무런 값도 추가되지 않은 상황에서 기본으로 값을 주는 것이다.

우리는 2번째 방법을 사용할 것이다.

아니 왜 이렇게 실수 많이 하냐.. case에 break 바로 뒤에 쓰고 있어서 처리 안 되고, 이름 잘못 써서 처리 안 되고 끔찍하네.

아무리 봐도 false인 값이 true가 되어 있다 ? -> 어디서 만지고 있따.

**애니메이션을 추가하는 작업의 순서.**

**저장하기(프레임 따는 작업)**

1. 각 오브젝트의 Init부분에서 Clip을 생성해준다. (일단, 클립생성)

// 그렇다고, 하드코딩 좌표를 바로 빼면, 어떤 부분을 해야 할지 모르지 않을까..?

1. 해당 클립에 텍스처를 지정해준다. (SetClipTexture)
2. 컬러키를 지정해준다(필수)
3. 프레임을 따서 저장한다.
4. 저장된 파일의 이름을 교체하여 불러올 수 있는 상태로 만든다.

**불러오기.**

1. 각 오브젝트의 Init부분에서 Clip을 생성해준다. (일단, 클립생성)
2. 해당 클립에 텍스처를 지정해준다. (SetClipTexture)
3. 클립을 Load한다. // 이 과정에서 컬러키, 플레이시간, 옵션, 좌표 등의 정보를 추가함)

애니메이션을 바꾸는 경우에 기존에 들어 있던 편집 coord정보에 접근하여 인덱스 오류가 났음. 해결.

아, 속도 조절하다가 애니메이션 시간 때문에 터졌다.. 예외처리하자

폴더 개념을 추가하여 저장위치변경 기능. ㅇㅇ

+ 애니메이션 클립 시간 변경 시 최솟값을 정한다. ( 음수값으로 가면, loop생성) ㅇㅇ

애니메이션의 컬러키를 미리 지정해두지 않으면 컬러키가 없는 상태로 데이터가 저장된다.

CtrlA 키. 전체 선택.ㄴ

9 STATIC\_STATE 설정. ㅇㅇ 키/ renderㅇㅇ / 적용 ㅇㄴㅁㄴㅇㅈㅇㄴㅁㅇㅁ

0 을 FrameAdj 활성화

이것을, on off처럼 해서 적용한 이후부터 처리를 해줄 것인지 아니면, 적용하기 이전에 상태만 바꾸면 일단 바꾸고 (바로 적용)할 것인지에 대한 여부를 정해야 한다.

(이것을 사용하는 경우는 새로운 애니메이션의 프레임을 넣는 경우일 것이다. 따라서, 적용 이후에 고정을 시키는 것이 맞을 것 같다.)

좌표를 고정시킨다는 의미는 결국 프레임의 SIZE를 제한한다는 의미인 것도 있다.

근데, 이 시스템 자체가 프레임의 사이즈를 고정시킨 체로 X/Y 좌표를 고정, 간격 조정을 해서 함께 이용하는 시스템이다.

고정된 X, Y값을 그려주는 것이 어떨까 ? ㅇㅇ

.

EX: 프레임 사이즈 맞추고, Y고정, X 간격을 설정하여 톡톡 (X 축으로 맞춰진 경우)

EX: 프레임 사이즈 맞추고, NONE 상태, 간격 없음 (맞는 것이 없는 경우)

EX: 프레임 사이즈 맞추고, X고정, Y간격 설정하여 톡톡 (Y축으로 맞춰진 경우)

// 프레임 사이즈를 맞추기 위해서 특정 프레임을 선택하여 맞추는 기능이 필요하므로 뺄 수 없다. F3’

Static X가 On인 상태로 X좌표를 움직인다. -> return; Y는 OK

Static Y가 On인 상태로 Y좌표를 움직인다. -> return; X는 OK

키보드 이동 : 막힌 상태가 아니라면 이동가능, 다만 간격이 켜진 경우 모두 다 이동한다.

마우스 이동: 막힌 상태 상관없이 이동 불가능. (XY다 움직임)

키보드를 통한 프레임 크기 : 프레임사이즈가 고정이면 처리 X(원래 있던 것)

정적 상태인 축에 대해서는 크기 증가 불가능. 아니라면 가능한데 간격이 켜진 경우 처리 X

마우스를 통한 프레임 크기 : 하나라도 막혀 있다면 처리 X

CtrlV 시스템. 일단, 세 가지로 나눈다.

NONE : 어느 것도 제한이 없는 상태이다. 그 자리에 그대로 넣어주고 기존의 편집 프레임을 풀고, 새로 생긴 프레임을 넣어준다.

// 미리 충돌나거나 none인 것을 거른다.

StaticX :ㅇㅇ

StaticYㅇㅇ

// 세 경우 모두 마우스와 충돌이라면 해당 좌표에 추가한다는 것이다.

Y고정이 되어 있는 경우에는 X의 좌표를 움직인다고 생각해보자… m\_bFrameAdj가 활성화되어 있지 않은 경우에는 선택한 프레임만 이동시키면 된다. 활성화되어 있다면, 하나를 움직여도 모두 처리해야 한다. ㅇㅇ

F10 프레임 간격 설정 true / false 키ㅇㅇ / Render ㅇㅇ/ 적용 ㅇㅇ

활성화/ 비활성화 기능이 있는 이유 : 아틀라스 이미지가 개판이면 false로 처리해야 함.

CtrlZ의 수명이 esc, 애니메이션 전환에 영향을 받지 않도록 구성한다. ㅇㅇ

F11 현재 추가된 프레임의 위치 좌우 반전 (순서 거꾸로).

(좌우 반대 전용 애니메이션 추가 용도)

마우스의 위치에 따라서, ctrl v를 누르면, 자동으로 해당 간격에 추가해주는 기능.

Ex : 선택한 프레임 (1개)에서 마우스가 해당 프레임의 내부에 있으면, 그냥 그 자리에 추가.

왼쪽에 있다면 왼쪽에 추가. 오른쪽에 있다면 오른쪽에 추가.

(적어도 대각선이동은 없다고 가정한다.)

y축도 따져서 y축이 안쪽에 있으면 고정.

Home / end y축 간격 증/감

Pg down / delete x축 간격 증/감 ㅇㅇ

Adj Frame을 변경하는 상황 예시.

X고정 / Y고정 / none  
Y간격/ X간격/ 간격설정 X

사용하는 경우는 이렇게 3가지가 있다.

따라서 NONE상태일 때는 간격설정을 막는 처리를 해주자.

(물론 마우스의 위치를 토대로 처리함)

키 ㅇㅇ, Render ㅇㅇ, 적용

// 예시. y축이 고정이 되어 있는 상태로 x의 간격을 증가시켰다.

// 이런 상황에서 한 프레임을 기준으로 하여 CtrlV키를 누른 경우 마우스의 위치 기준(좌우상하) 중에서

// y축 고정이므로 좌 충돌 우, 이렇게 3가지가 있는데 충돌이면 그 자리 그대로 복사

// 좌인 경우에는 편집하는 프레임의 begin좌표를 기준으로 Adj.x + fFrameX만큼 빼서, 새로운 프레임의 begin으로 삼는다

// 이후에 Frame값을 이용하여 end를 구할 것이다.

// 오른쪽에 있던 경우에는 fFrameX + Adj.x 만큼 더해서 그곳을 begin으로 삼아서 처리한다.

// 같은 논리로 Y축 고정에 대해서도 진행하게 된다.

// 위의 작업을 다 끝내긴 했는데 이것이 도움이 크게 될지는 모르겠다 .. ㅜㅜ

아무튼 간격을 가지고 쉽게 추가하는 방식도 추가하긴 했다. (좀 꼬이긴 했는데.. ㅎ)

원래 프레임의 애니메이션 옵션을 보여주는 기능을 제작한다.

편집 모드에서는 기본적으로 TIME\_DESTROY와 같은 옵션을 설정하여 돌리지 않고, loop로 설정해서 돌리고 있다. 그런데 이것의 문제점은 기존의 애니메이션의 옵션을 보여줄 수 있는 시스템이 없다는 것이다. 따라서, 애니메이션 옵션을 루프로 설정하기 이전에 미리 기존에 저장된 값을 받아둔 후에 이것을 출력해주는 시스템을 만드는 것이 좋겠다. 설정하기 전에 미리 받아서 이 값에 따른 출력을 보여준 후에 다시 애니메이션을 교체하는 순간에 기존에 받아 두었던 이 값을 다시 주는 것이다. (왜냐면 원래는 loop가 아닌데 루프로 설정이 되었을 확률이 있으니까) ㅇㅇ

{" "", ", ", "

{", "

ㅇㄴㅁ

"" }

-> 근데 이게 직선으로 맞는 모션이랑 많이 비슷함.

(다할 수는 없으니까 Hit부분은 날라가는 모션으로 하자), 이게 맞는 위치에 따라서 좀 달름.ad

날라가고 회전하는것만.

모션 추가가 제대로 안 된 경우(저장했는데) 오타오타,…

iOnceLoopFrame 변수하나 넣고 저장하고, 여기서 또 잘못 읽어서 엄청 시간 썼네 …. ㅜ

==== 시간까지 같다

JumpRight/ JumpLeft 1.56 once return

BigJumpLeft, BigJumpRight once return 1.54

" KirbyBigWalkLeftsdww",loop .1sa KirbyBigWalkR, ight"",

"KirbyEatLeft", once return 1.2s "KirbyEatRight",

", "KirbyVomitRight","""KirbyVomitLeft",

"KirbyTakeRight once return 0.76 KirbyTakeLeft",

KirbyAttackLeft",""KirbyAttackRight",

KirbyDownLeft KirbyIdleRight",

"KirbyDownRight" KirbyIdleLeft",

"KirbyWalkLeft 1s----"KirbyWalkRight" f

"KirbyFlyUpLeft KirbyFlyUpRight"

, "KirbyRunRight", "KirbyRunLeftw

KirbyBigHitLeft" "KirbyBigHitRight

KirbyHitLeft "KirbyHitRight",

KirbyFlyLeft", KirbyFlyLeft", KirbyFlyLeft",

KirbyFlyLeft", KirbyFlyRight",awdadwadwssswssssssssssssss

"KirbyPreventLeft", "KirbyPreventRight",

"KirbySlideLeft", once return 1.56s KirbySlideRight",

Adwa=dd애니메이션의 프레임 타임을 설정하는 것이… 프레임을 삭제하고, 추가하는 작업에서 분명 시간의 영향을 받는데, 이것을 업데이트해주고 있지 않다.

(차이가 심해지기 전에 미리 잡아야 한다.)

새로운 프레임을 추가하는 경우. fFrameTime \* vecSize() -> fPlayTime

기존의 프레임이 삭제되는 경우 fFrmaTime \* vecSize() -> fPlayTime

이런 업데이트 처리를 해야한다.

그리고 작업할 때 이름 좀 잘 보면서 하자

애니메이션 중에는 프레임의 구간 중에 일부를 돌리고 나서 loop구간이 생기는 애니메이션이 존재한다. 특정구간을 한 번 플레이 하고, 일부는 계속 돌려줘야 한다.

즉, 구간반복 옵션이 필요한 것이다.

옵션에 once loop 옵션을 주고, 해당 프레임의 경우에는 한번 돌린 이후에 해당 구간만 반복할 수 있도록 어느 위치를 반복하는지에 대한 정보를 저장할 수 있도록 한다.

Ex : 커비의 빨아들이기 애니메이션은 한번 끝까지 애니메이션을 돌리고 이후에 특정 구간이 반복된다.

옵션추가, 해당 옵션을 이용할 수 있는 변수를 추가하면 된다. iLoopFrame

근데 지금 현재 추가한 애니메이션이 꽤 되는 상황에서 이 변수에 대한 파일입출력을 진행하면,

분명 터질 것이다. 따라서, 일단 애니메이션을 다 만들고, Load에서는(해당 부분) 부르지 말고, Save에서만 저장해. 그리고 이 파일을 다시 수정해서 Load하도록 만들어.

다른 방법 : 그냥 애니메이션 통째로 저장하는 시스템 구현해서 save를 일단 처리하고, 이후에 그 파일만 Load하는 거지. (굿)

Once Loop 애니메이션 업데이트 방식..

일단, 기존의 애니메이션 시스템과 동일하게 한번은 돌리도록 한다. 그 이후에 옵션값에 따라서 조금씩 다른 처리를 해주고 있는데, 다른 옵션에서는 0을 주는 상황에서 once loop는 iFrame으로 FrameBegin값을 주는 것이다. 이러면, Begin의 값부터 end까지 반복이 된다.

애니메이션 업데이트에서 처리했고, 편집모드에서 보여주는 처리 및 적용, SAVE부분은 일단 생략

AO ONCE LOOP 옵션으로 설정하는 경우에는 이것의 옵션대로 보여주고,

그것이 아니라면, LOOP 로 보여주고 있다.ㄴㅇㅇㅇㅇㅇ

이제 이런 식으로 보여주는 옵션도 만들었으니… 일단, 다 만들고나서, Once Loop는 생각하지 않고 모두 만든다. 그리고 이것을 모두 불러올 수 있는 시스템을 구현. (애니메이션 클래스의 Load)

이후에 여기서 셋팅-> Save가 가능하도록 Save를 고치고 Load고치고… 끝.

현재 기본적인 몬스터 3마리와, 커비 움직임에 대한 리소스가 준비가 되어 있는 상태이다.

여기에 일단, 배경을 추가하여, 화면 스크롤이 되도록 한다.

현재 프레임워크의 구조상, 맨 먼저 출력하는 Layer가 BackGround인데, 여기에는 오브젝트를 넣어서 출력하는 개념이다. 따라서, Stage 클래스를 제작하여 여기서 배경을 설정할 수 있도록 구성하고, 충돌체에 대한 내용도 여기서 처리하는 것으로 한다.

// 일단 Stage가 배경의 사이즈를 받아서 처리하는 시스템을 구성한다

배경 이미지는 큰 리소스 파일에서 분할이 되어 있는 상황인데 이것을 사용하려면,

소스에서의 시작위치를 알아야 하고, 이 정보를 이용해서 따로 구해내야 하며, .

스테이지 3개 정도가지고, 일반 몹 배치하고, 여기서 여러가지 능력을 가진 몬스터를 추가하여 변화하는 모습을 모여주고, 가볍게 보스전으로 끝내면 좋을 것 같다.

// DDD (구현이 쉬움.) -> 소스 구해놓기

// 메타나이트 (간지 캐릭터) 많이 구현하더만 .. ㅋㅋ

동작 소스 + 배경.

일단 배경을 구했으니, 배경을 스크롤하는 방법에 대해서 알아보자. 일단 stage 자체의 size는 foreground에게 맞추도록 하고, background는 크기가 작거나, 혹은 비슷하거나 할 텐데.

Foreground는 플레이어의 좌표에 따른 플레이어의 위치가 그대로 들어가게 되고, Background는 플레이어가 움직인 경우에, 스크롤 속도만큼 이동하도록 한다. (이러면, 분명 background의 끝에 도달하게 될 것인데… 이 경우에는 처음 부분을 붙여서 스크롤이 되도록 한다. (플레이어의 포인터를 가져와서 사용한 이후에, stop을 확인하면서 입력이 들어왔으면 fTime을 이용해서 레이어의 속도만큼 이동하도록 구현한다. OK!!

일단, 백그라운드부터 카메라의 스크롤이 되도록 한다.

카메라를 들고 있는 쪽에서, 스테이지의 크기를 가지고 있어야 하는데, 내 구조상 SceneManager가 이것을 카메라를 들고 있다. 따라서, SceneManager에 화면의 크기를 등록하는 시스템을 구현해놓는다.

가상의 rect를 생각해보자. 플레이어는 이 가상의 rect에서 움직인다. 우리는 플레이어의 위치를 기반으로 카메라의 위치를 업데이트해준다. (여기서 필요한 정보가. 스테이지의 크기인 것이다.

여기에서 스테이지의 시작 위치도 받아두어야 할 것이다. (그러니까 스테이지가 리소스의 중간에 존재하는 식인데, 여기서 카메라 좌표를 뜯어내고, ….

일단, 카메라는 스테이지의 크기를 알고 있도록 한다. (SceneManager에서 가지고 있어야 한다)

(0,0)의 좌표에서 (Size.x, Size.y) 에서의 좌표까지를 이동할 수 있는 상태이고, 이것을 기준으로 카메라의 위치(l, t)을 구해낸다.

이렇게 구해낸 카메라의 시작 좌표를 가지고, 이것을 소스 좌표에 적용시키는 것인데, 소스에서 해당 스테이지의 위치가 0,0부터 시작이 된다고 하면, (이미지 자체가 통맵/ 나눠진 것이 아니라) 우리가 구한 카메라의 좌표가 src의 출력위치가 되면 된다. 하지만 그렇지 않은 경우에는

여기에서 src의 시작 위치(소스에서의 0,0)에서 카메라의 좌표를 더해서(src에서의 출력위치이자, 실제 출력이 되어야 하는 부분) 이 위치에서 출력을 해줘야 한다.

결론을 내면, 카메라의 연산은 0,0 을 기준으로 하여 스테이지의 size를 토대로 구해 놓고, 이것을 실제 src의 시작 좌표를 토대로 출력위치를 구하는 작업인 것이다.

SceneManager는 Stage의 크기를 알아둔다. (이래야 0,0 을 기준으로 하여 스테이지의 크기를 넘어갔는지, 혹은 카메라가 0, 0 각각 비교하여 넘어가는 상황인지 알 수 있다.)

Scene에서는 스테이지 이미지의 src에서의 좌표(시작, 끝)을 알고 있으면 된다. (그래야 출력 위치를 구하지)

결론 : Scene에 src의 시작과 끝 좌표를 주면, 이것을 기반으로 SceneManager의 좌표를 설정하는 것으로 하자.

Scene에서는 자신이 출력할 이미지의 크기를 알고 있을 것이고, 이것을 기반으로 Init에서 초기화를 해둔 이후에 SceneManager에서 셋팅하는 방식도 괜찮다.

// 스테이지의 크기를 설정한다. BackGround말고, ForeGround에 해당하는 이미지의 크기(지형)

위에서 Scene에서 이 정보가 필요한 것이 아니라… Stage에서 해당 src와 dest의 위치가 필요한 것이다. Stage의 정보 자체는 어떤 Scene이냐에 따라서 초기화가 달라지는 부분이니, 이 처리를 Scene의 Init에서 하자는 의미였다. 즉, 스테이지의 정보를 scene에서 설정해주고 ㅇㅇ 한다.

오브젝트의 피봇처리를 하는 것이 좋을까 ? 일단, 피봇 처리를 하든 안 하든, Center값이 필요하긴 하다. 피봇 개념을 도입해서, 아니다. 그냥 Center만들자.여기서 Center는 애니메이션이 돌아가는 부분의 가운데를 말한다. 즉, x는 m\_tPos.x + m\_tSize/ 2.f

오브젝트에 카메라의 좌표가 적용이 되었으니, 배경을 출력해보도록 한다. 디디디와 싸울 때를 생각해보면, background가 애니메이션이 돌아가고 있다.

현재 구조상, 스테이지를 기준으로 Size를 잡아서 카메라의 좌표를 고정시키기 때문에, 스테이지에 해당하는 출력인 foreground는 잘 출력이 될 것이다.

KEYPRESS 의 방식을 사용하지 않고, 지금 해당 방향키를 누르는 중인지를 판단할 수 있을까 ? 간단하게는 Stage(해당 키를 눌렀는지 판단하고 싶어하는 클래스)에 bool bLeftArrow, bRightArrow 같은 변수를 준 후에… 왼쪽 누르고 있으면 true 아니면 false값을 프레임마다 던져주면…

(함수 호출시 true / 매프레임 당 기본적으로 false를 주는 처리)

왼쪽 화살표, 오른쪽 화살표를 눌렀는지 판단이 가능하다.

근데 이러면 함수 등록하고 조금 과정이 복잡한 편이다. 따라서, 그냥 쉽고 간편한 KEYPRESS를 사용하고자 한다.

배경의 무한스크롤이 Speed값의 조절과, 막힌 공간에서 움직일 수 없도록 하는 처리로 인해서 보통은 배경 크기 따라가지 않을까 생각한다.

자, 이제는 KEYPRESS를 통한 스크롤이다.

어제, 지형에 해당하는 FOREGROUND에 대한 스크롤을 해주었는데, 플레이어의 위치를 기준으로 하여 우우리가 설정한 PIVOT을 기준으로 하여 플레이가 해당 PIVOT의 위치에 존재하도록 하면서, 마지막에 SIZE를 넘거나, 0보다 작아지는 순간에서 카메라를 움직이지 않게 하고, 우리는 이 카메라의 위치를 가져와서, SRC의 위치를 기준으로 출력하는 작업을 해주고 있다. 아무튼, 플레이어의 위치를 기준으로 출력한다. 여기서 플레이가 점프를 하는 경우에 Y축의 위치가 달라지고, 움직인 플레이어의 위치를 토대로 카메라의 위치를 수정하여 SRC의 출력위치가 달라지는 처리까지 되고 있다.

그런데, 이제 BACKGROUND를 처리를 하려고 한다. 일단, X축에 대해서는 우리가 설정한 무한 스크롤의 방식으로 처리할 것이다. 근데 Y축은 ? Y축의 경우에도 우리가 점프를 하면, 이것의 영향을 받아서 움직여야 할 것이다. (Y축 또한 무한 스크롤 방식이 아니라면 말이다.) 우리는 위에서 아래로 움직이는 맵을 구성하지 않을 것이다. 따라서 주로 X축을 향해서 움직이는 것이 대부분인데, 그렇다고 하더라도 플레이어의 Y축의 위치에 따라서 BACKGROUND또한 위로 올라가는 처리를 해줘야 한다.

OK ! 그러면, BACKGROUND의 X축은 입력에 따른 무한 스크롤 방식으로 하고, Y축에 대해서는 플레이어의 좌표를 받아서 처리하는 것으로 하자. 여기에서 Y축에 대한 PIVOT을 처리하고 싶다면 나중에 추가하여 처리하도록 한다.

일단 출력 위치는 0,0 에서 RS\_WIDTH, RS\_HEIGH(SIZE) 로 고정.

X축에 대한 무한 스크롤 방식.

일단, SRC의 위치를 기준으로 출력하려는 좌표가 시작 좌표보다 작은지, 큰지를 알아내기 위해서, SRC에서 Background의 시작위치와 끝 위치를 받아 두는 시스템을 구축해야 한다. 마치, foreground의 출력을 위해서 우리가 시작 좌표, 끝 좌표를 받아 두었던 것처럼 + Size를 받아서 편리하게 이용한다

일단, 현재 출력하려는 위치가 0, 0, size.x, size.y 를 넘어가고 있는지를 확인하자.

포지션 정보가 2가지가 더 필요한데,

3가지 상황이 있다.

일단, 넘어가지 않은 상황과 그렇지 않은 상황으로 나누어서 1, 2로 나눌 수 있다.

넘어가지 않은 상황은 그리 문제가 되는 부분이 아니기 때문에 넘긴다.

넘어가는 상황.

1. 오른쪽 끝에 있다가 오른쪽으로 이동이 해서 범위를 넘어가는 경우

출력하는 위치의 순서를 제대로 규정해야 하는데, 일단, 오른쪽에서 남아있는 부분이 1번.

그리고 넘어가서 출력되는 부분(왼쪽의 부분)이 2번이다.

1. 왼쪽 끝에 있다가 왼쪽으로 이동해서 범위를 넘어가는 경우

오른쪽에서 새로 생긴 부분이 1번. 왼쪽에서 남아있던 부분이 2번.

이렇게 달라지는 좌표를 기준으로 출력을 따로 2번 해주어야 한다는 의미이다.

그러면 우리가 begin end 묶음이 2개가 필요한데, 이것을 ClipCoord 구조체를 활용하기로 하고,

이름은 상대적인 위치로 잡지 말고, 먼저 출력하는 front와 back으로 나누도록 한다.

출력 범위를 지정하는 것 !

일단, 범위가 넘어가지 않은 경우에는 2번 출력이 일어날 필요가 없다.

일어나더라도 Front와 Back중에서 하나는 무효화되어 처리가 될 것이다.

게다가 정보를 생각해보면, 일단 범위가 넘어가지 않는 경우를 기준으로 처리할 좌표를 가지고 있다가 이것이 넘어가는 경우에 Front와 Back을 설정하고 이것들의 좌표를 토대로 처리하는 시스템이 더 쉽고 간단하지 않을까 ? 그러니까 우리가 필요한 정보는 Position정보 하나 더, (이 위치를 기준으로 RS\_WIDTH 만큼 출력할 것이다.) front와 back 두 가지 정보를 이용해서 출력할지 여부를 가지고 있을 bool 변수 하나.

기본적인 처리 방식은, 일단, BackGroundPos를 가지고 출력할 위치를 움직인다.

여기서 스피드에 따른 Position의 변화를 주면서 범위를 벗어났는지 확인한다.

범위를 벗어난 경우에는 범위를 벗어났다는 표시로 OutOfRange를 true로 주고,

벗어난 범위를 front와 back에 계산을 해둔다.

그리고 이렇게 벗어난 범위를 front와 back을 가지고 출력한다.(여기서 OutOfRange를 이용)

OutOfRange의 값과는 상관없이 일단은 BackGroundPos를 이동시킬 것인데, front와 back으로 나누어질 수 없는 범위까지 이동을 하게 되면, 다시 OutOfRange 값을 false로 주고, 반대편으로 넘겨주는 처리가 필요하다. 그러니까, 왼쪽으로 벗어나서, RS\_WIDTH 만큼을 벗어난 경우. 오른쪽에서 다시 좌표를 잡아주고 OutOfRange = false 이런 처리를 하자는 소리다. 2방향에 대해서만 처리를 해주면 될 것이다. 그러니까 OutOfRange가 true인 상황에서 계속 움직이다가 다시 벗어나지 않는 경우가 되는 경우를 말하고 싶은 것이다.

참고로 초기 상태는 0 , 0에 둘 것이기 때문에 벗어나지 않는다.

좌표를 움직이는 처리를 할 과정에서, tBackGround의 경우에는 src에 해당하는 좌표이다. 즉, left top은 이미 있으니 right bottom을 표현할 값만 있으면 되는 것이다.

BackGround의 출력을 위해서 설정해야 하는 정보.

Texture, BackGroundSrcPos, BackGroundSpeed, BackGroundStartPos(선택사항) , Size(자동),

Y축은 일단은 0에서 RS\_HEIGHT 만큼으로 고정하여 출력하는 것으로 한다.

// 8.6

Y축을 수정해야 하는 상황이 왔다.

Y축은 플레이어의 위치를 기반으로 하여, 어느 정도의 PIVOT을 기준으로 잡을지 생각을 해줘야 하는데, 굳이 이렇게 복잡하게 해줘야 하나 이런 생각도 들고, 그러니까 굳이 해줄 필요가 있는가 ? 보니까, 이 BACKGROUND의 경우에는 플레이어의 위치에서 꽤나 멀리 있다고 판단이 되도록 해야 하는데, FOREGROUND에 해당하는 지형을 움직이고 있다고 해서, 뒤에 있는 구름까지 급격한 변화가 생긴다 ? 굳이 그럴 필요가 없다. 더군다나, FOREGROUND로 인해서 충분히 지형이 변화가 일어나고 있으니, 지금도 충분히 자연스럽게 표현되고 있다.

Y축 수정을 하지 않는 것으로 한다.

와우 하나도 힘들지 않아 ! 앞으로 몇 시간은 풀로 커비 모작 작업을 할 수 있을 것만 같은 걸?

눈에 보이는 것은 커비랑 몬스터 몇 마리 움직이는 것뿐이지만, 이거라도 해낸 게 어디야 ? 솔직히 아직 5일째라고, 커비에서 그거 몇 마리 만들어뒀으면 거의 다 한 거지 인정 ? 인정 ? 인정 ? 인정 ? 인정 ?

8.6 작업 목록

1. StartScene을 만들어서 여기에 버튼 3가지 추가.

2. 능력이 있는 몬스터의 리소스를 따는 작업.

3. 플레이어의 기본 공격모션. 먹고 밷고, 날리고 ? 하는 모션과 이펙트를 적용하는 작업을 해야 한다.

# 1 . StartScene

현재, 마우스와 충돌이 일어나는 유일한 Scene인 EditScene을 보면, 자체적으로 마우스와 충돌한 프레임을 검사해주는 처리를 하고 있는데, EditScene의 경우에는 오브젝트가 여러 개의 충돌체를 가진 것으로 처리하고 싶지는 않아서 따로 처리를 하도록 구현한 것이고, 나머지 다른 장면의 경우에는 상관없이 처리하면 된다. 따라서, 마우스의 충돌체에 대한 초기화를 명확히 해주어야 하며,

그와 충돌이 일어날 물체 또한 제대로 생성해주어야 한다.

+ 장면 전환 처리 !

EditScene 장면 전환이 없음.

MainScene 장면 전환이 빈번

MapEditScene 잘 모르겟다ㅣ.

현재 프레임워크의 구조상, UI가 겹치는 경우를 처리해주기 위해서, 마우스가 하나의 UI와 충돌이 일어났으면 더 이상의 충돌처리를 하지 않고 있다. 나의 게임에서 이 부분이 필요한지 생각해보자.

1. 타일맵을 만드는데, 여기에 이미지의 편집기능을 추가할 것이라면 당연히 UI중복처리를 막아야 한다.
2. 타일맵을 만들기는 하는데 단순히 갈 수 있는 곳 못 가는 곳으로 처리할 것이라면 처리할 필요가 없다.

에이 근데 솔직히 할거면 당연히 1번이지 ! 근데 커비에서 타일맵(괜찮아 있어) OK

나중을 위해서 따로 처리하는 것으로 하자.

현재 버튼과, 충돌처리, 마우스 선택에 따른 모드 전환을 구현해둔 상태이다.

이제, StartScene에서 버튼을 클릭한 이후에 처리가 제대로 되는지를 확인해야 한다.

버튼 그냥, ㄴㄹㅇㄹㄴㅇㄹㄴㅇㄴㅇ아니 생각해보니 map툴할때도 버튼 쓰겠구나.. ㅎㅎ

아무튼, 나중에 상속구조로 바꿔서 처리하는 것으로 하자. ( 기본 Init부분 )

채널 리스트가 초기화가 안 되어 있다. -> 채널 설정에서 조건이 틀려먹었다.

보고 적는 것도 못하네;;

현재, 버튼이 제대로 사라지지 않고서 list에 남아있는 현상이 생겨서 오류가 터짐.

오류 해결 바람.

버튼 자체는 제대로 없애고 있는데, 그 안에 있던 Collider에서 문제가 생기는 것 같다.

Ref = 2인 상황에서 오브젝트가 사라져서 1이 되고, 남은 1은 충돌관리자에서 처리를 해줘야 하는데, 이 처리가 없다. 즉, ChangeScene을 하는 순간에 기존에 남아있던 Collider는 기존 장면의 것이므로 날려주는 처리가 필요한데 이런 처리가 없다는 것이다.

다음 장면이 생성이 되는 순간에서 다음 장면에 추가되는 오브젝트를 어디에 넣고 다니는지를 파악하면 되는데… Scene의 경우에는 따로 참조값을 증가하지 않았고, 이후에 우리가 날리는 것도 전혀 문제가 되지 않는 부분이다. 새로운 장면이 만들어지고, 이후에 그 장면에 새로운 오브젝트를 추가하는 것은 문제삼을 부분이 아니다. 단지, 충돌 관리자에서 소멸시키지 않은 것이 문제니까, 충돌관리자의 측면에서 생각을 해보면 장면이 생성이 되면서 오브젝트에 대해서는 Start함수가 호출되는 순간에 충돌체를 충돌관리자에게 넘겨주고 있다. 이 말은, 장면을 전환하는 과정에서 어느 정도의 Term이 있다는 것인데, ( 장면이 교체가 되고 WorldObjList를 봤더니 어 있네 하고서 배치하는 과정에서 Start함수를 호출하니까.)

아무튼 내가 하고 싶은 말은, 장면이 교체되는 바로 그 순간에 다음 장면에 생성되는 오브젝트의 충돌체가 들어가 있는 상황은 아닐 것이라는 말이다. 즉, 충돌관리자에서는 충돌체를 담을 때 참조 값을 증가시켰고, 이제 이 충돌체들은 사용하지 않으니, Safe Release VecList함수를 통해서 기존장면의 충돌체들을 모두 날려주는 것이 맞다.

기존에 있던 충돌체에 대한 오류는 해결을 했는데 새로운 문제가 터진다. 오브젝트 하나가 장면 생성 이후에 바로 죽는 것이다. 보니까, 기본 애니메이션이 맞아 죽는 걸로 되어 있어서 바로 죽는 것… 수정완료

현재 애니메이션 편집 모드로 이동하는 경우 오브젝트 배치가 되지 않고 있다. 이러면, 즉, 객체를 생성해서 각 객체의 소속은 지정해주는데 직접적으로 Start함수를 호출하면서 넣어주는 처리가 없는 것이다. 이렇게 오브젝트를 넣어주지 않게 되면 Scene에서 참조하는 것은 상관이 없지만 Layer에도 배치가 되지 않게 된다. 따라서, Layer에서 배치된 오브젝트들을 날려주는 순간에 해당오브젝트들이 들어가 있지 않아서 결국에는 사라지지 않는 현상이 생기는 것이다.

오브젝트의 생성 방식을 보면 미리 1로 참조를 만들어 놓고서 그 상태로 Layer에 넣는 작업을 거치는데, 현재 상황의 문제점은 Start함수를 호출하지 않아서 이것이 Layer에 배치가 되지 않고 따라서 오브젝트가 삭제되지 않는 다는 것이다. 따라서 SetWorld함수를 강제로 호출하여 Layer에 배치가 된다면 나중에 Layer가 날라가는 순간에 Object를 정리하게 될 것이다.

OK ! 다 잘 되구요 !

8.7 작업 목록

1. StartScene을 만들어서 여기에 버튼 3가지 추가. ㅇㅇ

2. 능력이 있는 몬스터의 리소스를 따는 작업.

3. 플레이어의 기본 공격모션. 먹고 밷고, 날리고 ? 하는 모션과 이펙트를 적용하는 작업을 해야 한다.

4. 맵 에디터 작업 ( 나중에 해도 된다. )

중력 및 모션 처리 -> 능력 있는 몬스터와의 전투(로 하거나 아니면 능력을 얻을 수 있는 다른 형식을 줘도 된다. 솔직히 이렇게 구현하면 더 쉽긴 함. ㅋㅋ 편법이다 ㅋㅋ 근데 다 보여줄 수 있는 부분이고 호호)

* 이 방식대로 했을 때의 이점. 따로 몬스터를 더 뜯을 필요가 없다는 것.
* 완전 꿀 인정 ? 인정합니다. 호호
* 보스 리소스 뜯기 작업 및 제작

커비가 A키를 눌러서 공격을 하면, 빨아들이는 이펙트를 보여줘야 하고, 회오리(?)의 충돌체가 커지는지는 모르겠지만, 아무튼 충돌이 일어나는 경우에 커비의 입을 향해서 각도로 움직이도록 하여야 한다. 그리고, 몬스터를 사라지게 하는 처리. + 커비는 Big상태가 된다.

이펙트와 공격모션을 보여주는 상황에서 일단, 충돌체의 영역 안에 들어가서, 빨아들이게 되는 순간 애니메이션이 끊기지 않고 Eat애니메이션까지 연결되고 있다. 충돌체의 크기가 일정하게 고정이 된 상태에서 공격이 일어나는 것으로 보인다.

여기서 2가지 처리가 일어나야 하는데, 플레이어의 공격 충돌체를 미리 달고 다니고, 이 충돌체에 대한 처리를 하는 순간은… 해당 방향을 바라보고 있는 경우이다.

즉, 몬스터와 공격 충돌체의 충돌중인 상태가 첫번째 조건.

그 상태에서 해당 공격 충돌체의 방향으로 공격(A)키를 누르고 있는 경우

몬스터를 빨아들이는 처리가 되는 것이다.

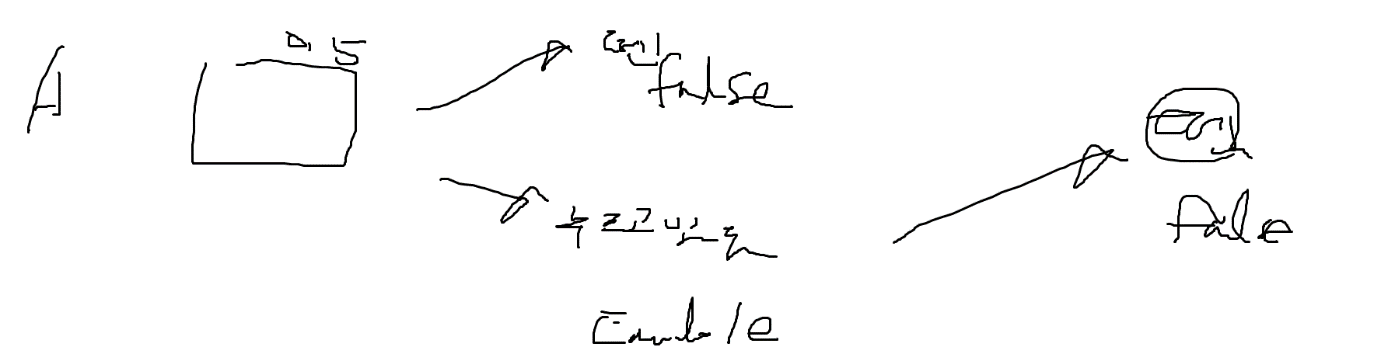
2가지 방법이 있다. 일단, 해당 방향으로 공격을 하는 경우에는 활성화를 true로 만들어주는 경우.(이때 맞으면 바로 처리하면 되는 부분)

맞지 않아도 계속 충돌처리를 하되 만약에 맞은 경우 들어오는 함수에서 공격을 하고 있었는지 등의 여부를 확인하는 방식이다.

매번 다른 오브젝트와 충돌을 검사하며, 예외처리까지 하는 것보다는 확실하게 공격을 하고 있는 경우에만 활성화를 주는 것이 더 나을 것 같다. 충돌체가 많지 않기 때문에 활성화 비활성화가 그렇게 어려운 부분도 아니다. 오히려 꼭 처리해야 하는 상황이 아닌데 이것을 매 프레임마다 검사하는 것이 훨씬 느리다. OK 그러면, 충돌이 필요한 순간에 true 활성화를 주는 것으로 하겠다.

일단, 몬스터에게 충돌체를 심도록 한다. 위치 차이를 주도록 하고.

플레이어에도 공격 충돌체를 추가한다.

공격 시작에서 Enable -> 이후에 0.5s 동안은 무조건 활성화 상태 유지 이후에 0.5s가 지난 시점부터 계속 검사를 해서 뗀 경우에 false 계속 누르고 있었다면 뗀 순간부터 false처리를 해주고 있따.

* 비활성화 충돌체는 흰색으로 출력해주고 있다.

잘 처리가 된다. 이제, 우리는 이 해당 충돌체가 overrap상태일 때 호출하는 함수를 이용해서

몬스터를 빨아들이는 시스템을 구현해야 한다. 몬스터에 변수하나를 둬서 (이거 보스 몬스터는 해당이 안 되니까 Normal몬스터 클래스에만 두는 것으로 하자. 이 bool 변수가 true가 되는 순간 해당 오브젝트에서 자체적으로 하던 처리를 막아버리고, 무조건 일정한 속도로 커비의 입속으로 들어오도록 한다.

모션이 끊기지 않도록 해야 하는데, 이 부분은 커비가 누군가와 충돌이 일어난 경우에 무조건, 모션이 진행되도록 예외처리를 해주면 될 것 같다. 예를 들면, 먼가 빨아들인 순간에 업데이트 처리를 통해서 AttackTime을 0.f . 이런 식으로 주면 계속 모션이 돌아가게 된다.

몬스터의 경우에는 플레이어에게 빨려먹혀지는 상황. 이 상황에서 해당 몬스터의 스피드를 0으로 둬서 업데이트 처리가 되지 않도록 하거나 .. 아무튼 내부적으로 막아주고, Hit모션을 계속 보여주면서 커비의 입을 향해서 날라간다.

이런 처리는 어떨까 ? 몬스터가 빨려들어가는 경우를 판단해서 빨려들어가는 경우라면 HIt모션을 호출(이걸 가상함수로 처리해!) 그리고 스피드 꺼버리는 처리. 일단, 빨아먹는 경우에는 계속 모션이 진행중 (충돌중) 계속 속도를 끄고, HIt모션을 띄워주며, 일정한 속도로 날라가도록 처리해줄 수 있음. 어느 한 쪽에서. 모두 다 처리하는 것이다.

굳이 양쪽을 모두 더럽힐 필요가 없는 방식인 것도 맞으며, 굳이 오브젝트쪽에서 이런 저런 ? 처리도 필요없는 ? ㅇㅇ 아마도 일단 해보자.,

충돌 상태를 정의한다. 일단, 처음 충돌이 된 경우에 Eat을 true로 주어야 한다.

충돌중에서는 몬스터가 끌려올 수 있도록 함수를 구성해줘야 한다. (그냥 이때 계속 true주자)

충돌이 끝난 경우(몬스터의 소멸)에서는 Eat을 풀어준다.

(현재의 구조상 충돌하던 물체(몬스터)가 갑자기 사라져도 Release함수는 호출해주고 있다)

이건 여담인데 굳이 left right 나눠야 되냐 ? 짜피 하고 있는 처리는 똑같은데 ? .. ok 버리자.

먹는 처리가 잘 되고 있다. 여기서 몬스터에 적절한 destory값을 주게 되면 아라서 사라지게 된다.

이것들을 충분한 값을 줘서 처리하도록 하고, 플레이어의 Center의 위치와 어느 정도의 거리가 차이가 나면 먹는 애니메이션을 호출해주고, 몬스터를 없애주면 될 것 같다.

\*처리를 하는 충돌체가 반드시 몬스터여야 한다.

Big애니메이션을 적용했다. Walk Run Jump 다 잘 되는 것 같다. ㅇㅋ

이제 별을 뱉는 것을 한번 해보도록 하자.

별은 회전을 이루면서 날라가는데 귀찮으면 그냥 별로 하고.. ㅇㅇ

날라가는 별 이펙트를 하나 제작한다. 회전이 자연스럽게 이루어지도록.

큰 별이 하나 날라가고 그 뒤를 조그만 별들이 같이 따라가는 형식

Effect 제작 목록. (기본 Kirby)

1. **커비 별을 토하면, 날라가는 별(이펙트는… 아니지) ㅇㅇ**
2. **달리면 날리는 먼지 ㅇㅇ**
3. **빨아들일 때 나오는 먼지… ㅇㅇ**
4. **입김 이펙트 ㅇㅇ**

**떨어짐의 별은 넘긴다.**

**Big이 아닌 Down상태일 때 가능한 동작. 다운 + S(점프) -> 슬라이드**

**(몬스터와 만나면 넉백효과) (몬스터 날라감 효과)**

**보니까 처리가, 그냥 우선순위에서 밀려서 따로 아무 처리를 하지 않는 것 같다.**

**그러니까 hold 상태인 경우에 애니메이션을 출력해주고 떨어진 경우에 false처리 하는 단순함이 보인다.**

삼키는 애니메이션 잘 나오는데 너무 느리다. 수정이 필요할 듯 하다.

애니메이션 편집 모드 키셋팅.

// 애니메이션의 속도를 변경하는 키를 만든다.

GET\_SINGLE(CInput)->AddNormalKey("--FrameTime", VK\_OEM\_MINUS);

GET\_SINGLE(CInput)->AddNormalKey("++FrameTime", VK\_OEM\_PLUS);

// 애니메이션 교체.

GET\_SINGLE(CInput)->AddNormalKey("++Animation", VK\_F5);

GET\_SINGLE(CInput)->AddNormalKey("--Animation", VK\_F6);

// Camera Move

GET\_SINGLE(CInput)->AddNormalKey("D", 'D');

GET\_SINGLE(CInput)->AddNormalKey("W", 'W');

GET\_SINGLE(CInput)->AddNormalKey("A", 'A');

GET\_SINGLE(CInput)->AddNormalKey("S", 'S');

// 속도 증가 감소.

GET\_SINGLE(CInput)->AddNormalKey("F1", VK\_F1);

GET\_SINGLE(CInput)->AddNormalKey("F2", VK\_F2);

// 현재 활성화된 프레임으로 크기를 고정시킴. 여러개가 잡혀있는 경우에는 처리하지 않음.

// 이렇게 설정한 이후에 이전에 잡아둔 모든 프레임 정보가 프레임에 맞게 수정됨.

GET\_SINGLE(CInput)->AddNormalKey("F3", VK\_F3);

// Save Option Change

GET\_SINGLE(CInput)->AddNormalKey("F4", VK\_F4);

// Once Loop Begin Frame

GET\_SINGLE(CInput)->AddNormalKey("F7", VK\_F7);

// reset

GET\_SINGLE(CInput)->AddNormalKey("F8", VK\_F8);

// Static State

GET\_SINGLE(CInput)->AddNormalKey("9", '9');

// Frame Adj Enable

GET\_SINGLE(CInput)->AddNormalKey("0", '0');

// Collision Enable

GET\_SINGLE(CInput)->AddNormalKey("Q", 'Q');

// all Delete

GET\_SINGLE(CInput)->AddNormalKey("ESC", VK\_ESCAPE);

// COORD KEY

GET\_SINGLE(CInput)->AddNormalKey("CoordDelete", VK\_BACK);

// Coord Move & Resize Keyboard

GET\_SINGLE(CInput)->AddNormalKey("Right", VK\_RIGHT);

GET\_SINGLE(CInput)->AddNormalKey("Left", VK\_LEFT);

GET\_SINGLE(CInput)->AddNormalKey("Top", VK\_UP);

GET\_SINGLE(CInput)->AddNormalKey("Bottom", VK\_DOWN);

// Coord Move Speed

GET\_SINGLE(CInput)->AddNormalKey("++MoveSpeed", '1');

GET\_SINGLE(CInput)->AddNormalKey("--MoveSpeed", '2');

// Coord Move Mouse

GET\_SINGLE(CInput)->AddNormalKey("E", 'E'); // Pos

GET\_SINGLE(CInput)->AddNormalKey("R", 'R'); // Size

// EditCoord clear

GET\_SINGLE(CInput)->AddNormalKey("F", 'F');

// ex : E + 키보드 위치이동.

// E + 마우스 위치이동

// R + 키보드 크기변경

// R + 마우스 크기변경

// Most Newly Added Coord Revive

GET\_SINGLE(CInput)->AddNormalKey("CtrlZ", 'Z', VK\_LCONTROL); // DELETE : 가장 최근 좌표를 없앤다.

// 근데 해당 좌표가 여러개일 가능성이 있으므로 vector로 받아둔다.

// 저장하기

GET\_SINGLE(CInput)->AddNormalKey("CtrlS", 'S', VK\_LCONTROL);

// 불러오기

GET\_SINGLE(CInput)->AddNormalKey("CtrlL", 'L', VK\_LCONTROL);

// 편집 프레임 붙여넣기

GET\_SINGLE(CInput)->AddNormalKey("CtrlV", 'V', VK\_LCONTROL);

// 모든 프레임 편집모드 선택.

GET\_SINGLE(CInput)->AddNormalKey("CtrlA", 'A', VK\_LCONTROL);

// Frame 간격 설정.

GET\_SINGLE(CInput)->AddNormalKey("++FrameAdjX", VK\_NEXT); // x 증가

GET\_SINGLE(CInput)->AddNormalKey("--FrameAdjX", VK\_DELETE); // x 감소

GET\_SINGLE(CInput)->AddNormalKey("++FrameAdjY", VK\_HOME); // y 증가

GET\_SINGLE(CInput)->AddNormalKey("--FrameAdjY", VK\_END); // y 감소

// 애니메이션 프레임 넘기기. 보통 P(정지) 상태로 사용할 예정

GET\_SINGLE(CInput)->AddNormalKey("3", '3'); // Next

GET\_SINGLE(CInput)->AddNormalKey("4", '4'); // Prev

// vector 순서 뒤집기

GET\_SINGLE(CInput)->AddNormalKey("5", '5'); // Reverse

// 편집하는 오브젝트 변경.

GET\_SINGLE(CInput)->AddNormalKey("ObjectChange", VK\_TAB);

커비가 몬스터를 먹는 부분은 자연스럽다 이 이후에 커지는 상태가 되어야 한다.

Bool 변수를 통해서 처리하도록 한다. 그리고 출력해주는 모션을 big에 맞도록 고친다

big상태인 경우에는 방어를 할 수가 없다..

오늘은, 중력을 구현하여 적절한 수치를 구해보도록 한다.

중력이 되면, 그 이후에는 다양한 이펙트들을 만들어보는 것을 목표로 한다.

그 이전에 삼키는 애니메이션이 조금 안 맞는 것 같으니 (속도나, 모션이나) 이것을 먼저 수정하는 것으로 한다. 모션은 같은데 속도가 안 맞다. 처리완료!

중력처리

중력 처리는 일단, Character오브젝트에 Gravity 변수를 추가하고, 이것의 on/off에 따라서 중력처리를 하는 것으로 한다. 커비에서의 중력처리는 일정하게 빠른 속도로 떨어지는 것으로 보이는데,

해당 bool 변수를 가지고 스테이지의 오브젝트와 충돌하는 경우. 충돌한 영역만큼을 다시 옮기고, gravity 변수를 false로 만드는 처리를 하는 것이다. (이후에 더 이상 떨어지지 않도록)

그 이후에 다시 release가 되는 순간에 true를 줘서 중력의 영향을 받도록 구성한다.

근데, 이런 식으로 구현하면 안 될 것 같다. 그냥, 중력의 영향을 받을지만 flag로 처리하고, 스테이지 충돌체와 충돌한 경우에 해당 Intersect의 위치만큼 빠져나오도록 하는 것이다.

Start 나중에 다시 튀어오르기를 할 때 필요할 것 같은 느낌이 들어서 일단 만든다.

일단, 직선 충돌은 따로 해줘야 한다. 근데, 단순히 Rect충돌을 가지고 처리하는 것이 매우 골치아플 가능성이 높다. 왜냐하면, 이미 잘 만들어진 TOOL기능에서 방향성에 따른 처리를 할 수 있도록 되어 있는데, (여기서 위치와 방향을 모두 따져서 어느 위치에서 날라와서 어떤 식으로 처리를 해줘야 하는지에 대한 부분이 모두 포함이 되어 있다) 이것을 포기한다면 나는 이 골치아픈 vector의 구현을 다시 구현해야 함은 물론, 좀 다른 식의 툴까지 만들어야 한다는 것이다.

성우쌤이 짜준 맵툴을 사용하면 vector 구현, map툴 모두 가져다가 쓰면 된다.

그걸 쓰지 않고 단순히 Rect충돌 처리를 한다면, 위에서 구현한 vector를 다시 구현해야 한다.(Rect를 가지고) 물론 보고 할테지만, 이미 만들어져 있던 것을 생각하면 굳이 그렇게 해야 하는가 하는 생각이 든다. 또한 이것뿐만이 아니라 맵툴까지 만들어져 있는 상태라서 내가 이것을 포기하면 map Tool + Vector 구현까지 포기하는 것.

솔직히 이거 가져다가 쓰면 시간 이득에서 5일 이상이다. 쉽고 간편하고 맵툴까지 … 여기서 발전을 시켜야 겠지만 아무튼 VECTOR구현이 되어 있는 상태에서 다시 만들 필요까지는 ….

OK 가져다가 쓰자. 대신에 확실하게 이해하는 것으로 하자.

대신에 여기에 살을 붙여야 한다 !

커비의 행동 중에 하나가 높은 곳에서 떨어졌을 때 조금 튀어오르는 것인데, 이것은 Big상태가 아닌 경우에만 실행된다. 또한 중간에 떨어지는 모션이 변경되는 경우에 나오지 않는다.

오늘 따로 하는 것이 없지만 괜찮아. 지금 내가 구상하고 있는 시스템상 이거 이해하고 활용만 해도 5일치 시간번다.

일단, 스테이지의 정보들 중에서 Tile에 해당하는 정보만 읽고 저장할 수 있는 시스템이 만들어졌다. 이제 이것을 불러오고 읽어올 수 있도록 window 창 부분만 가져오자.

8.9 작업

1. 스테이지 맵툴 제작. 저장하고 읽어오는 시스템까지. ㅇㅇㅁㄴ

2. 만들어진 스테이지를 가지고 충돌처리(프레임워크에 구현이 되어있는 상태) 적용 ㅇㅇ

3. 중력을 적용하여 다양한 동작을 구현.

4. 위의 동작에 이펙트를 추가.

맵툴 제작

기본적인 기능은 프레임 워크에서 그대로 따서 옮겼다. 이제 여기서 나만의 기능을 하나씩 추가하여 편리하게 이용할 수 있는 맵툴로 제작하자.

예를 들면, 마우스가 선택한 타일의 인덱스를 받아서 이것을 일괄적으로 편집하는 기능 일괄 삭제 기능 등.. (그냥 셋팅하면 된다.)

일단, 타일에 뿌려주는 Rect가 굉장히 마음에 들지 않는다. 이것을 아주 작게 표시하고 싶다.

그냥 가자. xx

맵툴의 저장, 읽어오기 다 잘 되는데, 여기에 기능을 추가하려고 한다.

내가 선택한 타일의 인덱스를 받아 두고 이것을 모두 삭제하는 기능. ESC .. ㅇㅇ

// 구체적으로는 Normal로 바꾸는 것이다. XX

// 그리고 이것을 다시 복구하는 기능 ctrl Z XX

내가 마우스로 선택한 인덱스를 다시 삭제하는 기능. -> 이거는 … Normal로 다시 클릭하면 되잖아.

좌클릭을 하면 타일 하나만 추가가 되는데, 드래그를 하면서 그리는 Rect에 대하여 타일을 추가하는 기능. ㅇㅇ

간단한 기능인데, 벽인 부분을 편집 IndexVector에 담는 기능이다.ㅌㅌ

# 추가 기능

옵션모드에서 선택한 타일의 인덱스를 담아두고, 이 타일을 다시 선택하면, NORMAL상태로 바꾸는 기능 ㅇㅇ

WALL 상태에 있는 모든 타일을 다시 Normal로 변경하는 ESC기능. ㅇㅇㅇ

드래그를 통해서 그리는 Rect를 보여주고 이것과 충돌하는 타일을 WALL/ Normal 로 바꾸는 기능

(일괄 편집기능) ㅇㅇ

출력 기준으로 하면, Screen이고, Tile기준으로 하면 월드좌표. 어느 한 쪽을 선택하든 변환은 필수적. ㅇㅇ

// 출력기준으로 ClickPos를 잡는다. ㅇㅇ

기존에 있던 EditIndex Vec를 없앴다. (사용하는 곳도 없고 너무 비효율적인 작업을 하고 있기에)

2가지의 경우로 나누어서 타일의 옵션을 설정하는 것으로 한다.

일단, 현재 설정이 block인 상태 / block이 아닌 상태

Block 인 상태에서는 block인 물체를 클릭하면 normal로 변경.

Normal인 물체를 클릭하면 당연히 block 으로 변경

Normal인 상태 에서 normal 클릭 (변화 X -> 어떻게 보면 덮어 씌우는 개념이니 변화)

Normal인 상태에서 block클릭시 normal로 변화.

현재 맵툴에서 타일 기반의 충돌은 되어 있는데, Line충돌을 처리하는 방법이 나와있지 않다.

Line충돌을 구현하고 이것이

일단, 타일맵을 만들었고, 이것의 충돌처리 로직도 있으니 한번 돌려본다.

아니, 잘 되는데 ResourceManager에서 계속 터지는데 ?

일단, 직선 충돌 만들어보자.

직선 충돌을 다른 오브젝트가 가지고 다니는 경우가 있나 ? 있나 ? 있나 ?

ㄴㄴ 솔직히 Stage의 정보. 그러면 Relative정보를 설정할 필요가 있을까 ?

그리고 이것을 설정해도 애매한 점은 두 점에 대해서 처리하는게 좀..

# 게임에서 사용할 기본적인 맵툴작업은 끝났다. (이미지 타일은 사용하지 않을 것 같으니…)

Line충돌에서 살짝 잘못 처리한 부분이 있는데, 현재 구조에서 src의 위치를 Rect의 좌상단으로 잡고 오른쪽 아래를 dest의 좌표로 잡고 있는 것이다. (근데 우연히 src의 위치를 왼쪽 상단에 주고 dest의 위치를 오른쪽 하단에 주고 있더라는…)

분명 위치에 따라서 달라질 수 있는 부분인데, 미스.

정보를 셋팅하는 순간에서 위치를 토대로 Rect를 만들어서 정보로 넣어주고 이것을 충돌처리 하는 상황에서 가져와서 사용한다.

예외 처리 부분에서 사용하고, 내부적으로 사각형의 4변과 Line충돌을 하는 부분에서는 딱히 문제가 없다. (내부적으로 직선의 방정식을 구해서 좌표를 찍어주고 있으니)

처리 완료. -

선분 Vs 사각형 충돌을 완성했다. 이제 올려주는 처리만 적용해주면 될 것 같다.

PlayerRect에 적용하면 다른 충돌처리를 하는 Rect를 검사해줘야 한다.

Stage에 적용하면, 해당 StageLine에 적용할 수 있어서 검사할 필요없이 처리하면 된다.

문제 한 가지 더, 플레이어에 적용한다는 의미는 다른 중력의 영향을 받는 오브젝트들에게도 똑같은 처리를 한다는 것을 의미한다.

누구의 함수를 등록하지 ? 그냥 MainScene의 함수를 등록해주자.

Line과 충돌이 일어나는 경우에 해당 함수가 호출이 된다. 그러면 여기서는 오브젝트의 위치를 바꿔주는 처리를 해야 하는데, 오브젝트의 중심에 해당하는 좌표를 넣어주고 그때의 Y값으로 위치를 설정하는 것이다. 현재, Rect의 하단 Line과 충돌이 이루어지고 있는 상황이다. 그러면, 왼쪽편의 영역이 더 많은 상태에서 오른쪽 하단에서 충돌이 난 경우에도 해당 함수가 들어오고, 왼쪽 하단에서 충돌이 난 경우에도 바로 해당 함수가 호출되는 것이다.

즉, 충돌이 난 시점에서 바로 위치를 조정하면 안 된다는 말을 하고 싶다. 그래서 내가 생각한 Logic은 Line의 GetX, GetY함수를 만들어 둔 상태이다. 여기서 GetX를 호출하여 Y값을 넘겨주는 상황에서 만약 해당 선분에서 정의할 수 없는 Y의 값이 들어오는 경우 -1을, GetY에서 X의 값이 정의할 수 없는 값이라면 -1.f 을 return하고 있다.

(생각해보니, RectInfo의 정보를 Line에 추가해둔 까닭은 Src의 위치와 Dest의 위치관계가 항상 고정된 것이 아니기 때문이었다. 위치는 가변적으로 바뀔 수 있기 때문에 Dest의 y값, Src의 y값 누가 더 큰 값을 가지는지 명확하게 규정할 수 없다. 따라서 상황에 따라 위치를 표현할 수 있는 Rect를 정의해두는 개념이 도입되었고, 생각을 확장하다보니 자연스럽게 충돌에서 예외처리부분도 교체해주게 된 것이다.)

미리 정의한 LineRect의 값은 두 점에 의해서 그려진 Rect를 정의하고 있다. Src와 Dest의 위치를 토대로 정의하고 있기 때문에 x의 범위 / y의 범위를 확인하는 용도로 사용할 수 있다.

GETX GETY함수에서 예외 처리부분으로 사용이 되었다. 윈도우 좌표계의 특성상 map에 그리는 선분에서 GetX GetY함수를 호출하는 경우에 정상적으로 -1.f가 나오는 경우는 없다.

0,0 보다 낮은 범위에서 논다면 가능한데, 그런 경우는 없다.

-1.f는 예외상황을 나타내는 절대적인 값으로 사용이 가능하다.

위에서 하던 것을 계속하면, 충돌이 일어나는 상황에서 오브젝트의 x위치를 넣어주는 것이다.

그러면 피봇 0.5가 적용된 위치로 가운데 좌표로 사용이 가능한데, 이때 넣어준 x의 값에 대한 y의 반환 좌표를 가지고 오브젝트의 좌표로 주는 것이다. Line과 충돌을 했으니, 직선의 방정식을 이용해서 해당 Y좌표를 구해내고 이것을 오브젝트한테 적용하여 오브젝트가 있어야 하는 Y의 위치를 정의하는 방식. 아무튼, 여기서 위에서 말한 끄트머리 충돌시에도 해당 함수가 호출이 되기 때문에 중심의 위치를 넣어주더라도 정의할 수 없는 부분이 나오는 경우에는 처리하지 않는 것이다. (만약 이 부분에 대한 처리를 하지 않았더라면… 두 점에 의해서 그려지는 직선의 방정식에 대한 처리를 하고 있을 것. 그러면 가상의 보이지 않는 선에 의한 위치 선정이 되는 마법을 보게 되었겠지)

자 이제 완성이 되었으니 확인해보자.

잘 올라가는데, 중간에 가다가 경사가 급해지면 왜 탈출이 안 되니 …

보니까 타일 때문이네… 여기서 생기는 구멍은 Line하나 바닥에 그려서 막을 수 있다.

현재 Line은 바닥에 대한 처리로 대각선, 직선 상관없이 이용할 수 있는 형태이다.)

(이거 로직 구현해야… 겠다. 두 직선이 x축 혹은 y축에 평행하는 형태.

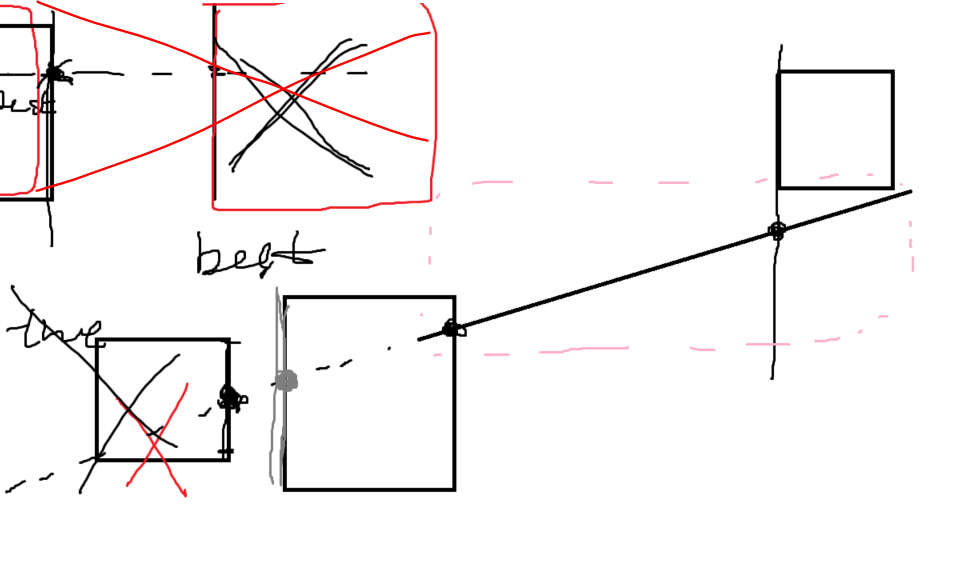
솔직히 말해서 안 그래도 타일위치에서 바닥이 안 맞는 상황 혹은 벽에 안 맞는 상황이 있어서 고통스러웠는데 잘 되었다.

아니다 다 구현해주자.

구현하면서 정말 신기한 부분은 내가 정확하게 모르고서 짠 로직인데도 이것에 대해 더 파다보면 왜 그런지 이해하게 되는 점이다. Line에 의해서 생기는 Rect와 사각형을 충돌처리하여 충돌이 나지 않은 경우는 절대로 Line과 충돌일 수 없다는 예외처리의 의미가 무엇인지 깊이 느끼고 있다.

해당 경우를 미리 배제해줌으로서 이후에 생기는 잡다한 문제를 생각하지 않아도 된다.

아니 근데, 이거 예외가 또 있네 ㅋㅋㅋㅋ



현재, 선분의 직선의 방정식을 구해서, 사각형의 변이 알려주는 X or Y 의 좌표를 이용해서 교점을 구하고, 이 교점이 사각형의 변 위의 점인 경우 충돌이라고 판단하고 있다.

(이 처리 이전에 선분에 의해서 그려지는 사각형과 Rect를 충돌처리해서 true인 경우만 처리를 해주고 있다)

나는 Rect에서의 예외 처리로 인해서 완전히 오류가 배제될 것이라고 생각했는데, 아니었다.

사각형의 충돌이 일어나는데도 불구하고, 충돌로 인식해서는 안 되는 경우가 존재한다.

(하나씩 건드려보지 않으면 발견도 하지 못했을 오류로 보인다)

회색 부분을 보면, 선분이 만드는 직선의 방정식의 교점을 구한 것을 볼 수 있다.

분명 Rect의 예외처리를 통과한 상태이고, 좌표를 검사해서 변 위의 점이라고 판단하고 있는 상황이다. 이때 이것을 충돌이라고 판단해주고 있는 것이다. 오른쪽 변을 생각하면 당연히 충돌이긴 하지만 지금 검사하고 있는 것은 왼쪽 변에 대한 처리이다. 이 부분을 고려할 때 분명 이 처리는 잘못된 것이다.

결론적으로는 변 위의 점인지만 판단을 해서는 안 되고, 좌표의 범위도 같이 판단하여 예외 처리를 해야 한다.

(아니면 미리 위에서 막아주는 것도 좋다.) (이걸로 간다)

이미 직선 충돌에 많은 시간을 썼고, 이미 탈탈 털리고 있지만, 이거 끝내면 더 이상 어려운 처리는 없다. 단지 구현일 뿐이지. 근데 그것보다 이게 더 어렵기 때문에 쉬운 문제다. 하하하하하

아니 근데, 물론 이거 고치긴 하겠는데, 솔직히 평행하는 직선을 바닥에 까는 처리는 꼭 해야하는 거야 ? 그거 vector개념 넣어서 처리하는 것도 굉장히 어렵고, 또 이걸 하는 이유가 단순히 바닥이랑 좀 안 맞는 것 같아서 인데, 그런 경우라면 프레임을 30을 찍더라도 타일 크기를 줄여서 정교하게 맞추면 되는 부분 아니야 ? 그치 …. (너무 무리한 짓을 시도했다.)

흐음. 그러면, 타일 크기를 줄여서 정교하게 (프레임은 망했네) 처리하는 것으로 하고, 이거 예외처리만 좀 해보자.

아니, 이거 예외 처리하면서 왜 이렇게 무서운 부분이 하나씩 보일까 ?

Src와 Dest가 있는데, Src와 Dest위치를 거꾸로 넣어주면, (현재 src가 왼쪽에 있다고 가정하고 충돌처리가 진행되고 있다) 예외처리 부분이 깨지게 되는 것이다. 이건 고려하면 더 복잡하다.

내가 만드는 게임이니 당연히 src는 왼쪽에 있고 dest 는 오른쪽에 있다. (X에 대해서는…)

그렇다면 Y는 어떨까 (예외처리를 안 하던 부분이 추가되어서 이런 정보가 필요하다)

아니 그럴 필요도 없이 누가 더 왼쪽에 있고, 오른쪽에 있는지 확인해주면 되는 부분인가…

그렇네….

OK 예외 처리 했다. 확인은 일부러 하지 않는 이상 보이지 않겠지만…

# 직선 Vs 사각형 충돌 예외처리 끝.

오브젝트가 붕 뜨는 느낌이 드는 것은 타일 크기를 줄이든 충돌체 크기를 줄여서 어떻게든 처리하는 것으로 한다. (이걸 다시 vector를 구현한다는 것은 시간을 퍼붓겠다는 소리니까 그만하자)

# 정리.

일단, 직선 vs 사각형 충돌을 구현을 해둔 상태이다. 오브젝트가 뜨는 느낌이 드는 것은 충돌체와 타일의 정교한 처리로 어떻게든 해결할 문제이다. 그러니까 지금 처리할 필요는 없다.’ ㅇㅇ

그럼 다음 작업. 이거 지금 배경 스크롤이 무조건 키보드 누르는 경우에 나가고 있는데

타일 혹은 벽에 해당하는 부분에 부딪힌 경우에 스크롤이 되지 않게 하는 방법이 필요하다.)

! 좋은 생각 났다.

플레이어의 좌표를 움직이는 상황에서 만약에 현재 방향키가 눌려있는 상황인지를 확인해서,

눌려있는 상황이라면, flag를 false로 두고, 매 프레임마다 true로 주면 어떨까 ?

해당 업데이트를 하는 곳은 어디인가 ? Stage

타일 충돌을 하는 상황에서 Stage를 가지고 오니 나쁘지 않다.

Stage에서 변수를 들고 있는 것으로 하자.

X축으로 커비가 밀리는 경우에만 false처리를 해서 BackGroundUpdate를 막아주었다.

Collision 이전에 Update 처리를 미리 해버린 까닭에 프레임 쪽에서 문제가 있는 것 같아서.

일단, Render쪽으로 빼서 처리했다. Collision이후에 막히게 되어서 일단 생각처럼 잘 된다.

300.f 속도로 설정하니 나쁘지 않네. ㅇㅇ

아 그리고 지형이 좀 흔들리네요. 들들 볶아. 플레이어의 위치가 갑자기 충돌처리로 변하니까 이걸 카메라가 잡지 못하는 것 같네. ( 카메라 업데이트 호출로 해결 ) ㅇㅇ

# 8월 11일(작업할 것)

3. 중력을 적용하여 다양한 동작을 구현.

4. 위의 동작에 이펙트를 추가.

5. 터지는 것 좀 어떻게 해봐. ㅇㅇ

# 마우스 텍스처 터지는 문제(해결)

마우스 애니메이션에 있는 Texture가 ref값이 맞지 않는 상황이다.

마우스의 애니메이션 생성 ref = 1

Input에서 마우스 애니메이션을 참조하고, Ref = 2

해당 애니메이션에 클립을 넣는 과정으로 파일이름 여러 개를 vector에 저장하여 SetClipTexture함수를 호출.

그러면 해당 이름을 가지고 텍스처를 Load하여 //

아니, 잠깐 현재 리소스가 이미 있는 경우에 해당 리소스를 설정하고, ref값 하나만 증가하면 되는 상황이다. 그런데, 이것을 확인하는 곳이 LoadTexture이다.

그렇다면 LoadTexture를 통해서 해당 이름을 가진 텍스처를 확인해보는데 여기서 FindTexture를 사용.

2가지 경우가 있다. Texture가 있는 경우 / 없는 경우.

있는 경우라면 FindTexture로 증가시킨 참조 값을 감소시키고 return false를 하는 것이 맞다.

근데 지금 그냥 나오고 있네… ㅡ.ㅡ

해당 텍스처가 있는 경우에는 해당 함수에서 증가시킨 ref 값을 일단 감소하여 아무 일도 없던 것처럼 만들어 두고, 이후에 이 Load함수가 끝나고 해당 텍스처를 읽어오는(ResourceManager로부터 읽어오는 상황에서 FindTexture함수를 이용하여 ref값을 증가 시켜 두는 것이 맞다.

아무튼 여기서 문제가 생길 여지가 있었던 것 같다.

반면에 새로 생성하는 경우에는 resourceManager에 하나가 들어가 있도록 하고, 이것을 참조하는 다른 곳에서 ref를 증가시켜 두는 것이 맞다.

현재 터지는 것은 이미 있던 리소스를 해제하는 상황은 아니다.

마우스 ref = 1 ㅇㅇ

마우스 충돌체 ref = 2 ㅇㅇ

마우스의 애니메이션 생성 ref = 1

Input에서 마우스 애니메이션을 참조하고, Ref = 2 (마우스 날라가면서 1, Input에서 1개) ㅇㅇ

해당 애니메이션에 클립을 넣는 과정으로 파일이름 여러 개를 vector에 저장하여 SetClipTexture함수를 호출.

LoadTexture 해당 리소스가 없기 때문에 하나를 생성

이것이 리소스 매니저에서 들어가게 된다.. Ref = 1

이렇게 생성한 애니메이션 텍스처를 기본 마우스 클립에서 참조하여 Ref = 2

마우스의 Start함수를 통해서 Texture로 셋팅하여 Ref = 3

ClearScene함수를 통해서 Texture Ref = 2

// 여기서 문제가 생긴다. 리소스 매니저 1개, 기본 텍스처 1개, 애니메이션의 클립 텍스처 1개

// 이렇게 3개를 없애야 하는데, 그 전에 ClearScene을 통해서 ref 값을 하나를 줄여버린 것.

DESTROY\_SINGLE(CSceneManager);

DESTROY\_SINGLE(CInput)

GET\_SINGLE(CCollisionManager)->Clear();

DESTROY\_SINGLE(CColliionManager);

InputManager의 소멸자가 호출되어 마우스, 마우스 충돌체, 마우스 애니메이션을 날려줌.

보통의 충돌체라면 CollisionManager에 의해서 ColliderList에 참조되어 있었겠지만, 마우스에 대한 처리는 따로 이루어지므로 상관이 없다. Ref = 1인 상태로 사라지게 된다.

마우스가 ref = 1인 상태였다가 지우게 되어 소멸된다.

그러면 마우스의 충돌체 ref = 1 Input에서 소멸된다.

애니메이션 소멸. ㅇㅇ

텍스처의 경우에는 오브젝트 날려서 1, 애니메이션 날려서 1, (여기까지가 0인 상태로)

ResourceManager에서 날리는 처리. ( ERROR)

그러면 마우스의 텍스처의 경우 ClearScene의 영향을 받지 않도록 처리하면 되겠다.

그러면 해결이 되야 하는데, 그 전에 터지는 것을 보면 다른 리소스도 댕글링 포인터가 되어 버린 까닭이다.

일단, 위에 텍스처의 경우에는 ClearScene의 영향을 받지 않아서 ref = 3인 상태로 유지되고

이후에 오브젝트의 텍스처 1, 애니메이션 참조로 인한 텍스처 1, 리소스 매니저가 가지고 있던 텍스처1 를 통해서 0이 되고 잘 삭제가 될 것이다. ㅇㅇ

# 다음 문제 (해결)

BackBuffer 에서 문제가 생기는 것 같은데 한번 없애보도록 하겠다.

일단 ResourceManager의 Init함수에서 해당 텍스처를 읽어오고 있다. 생성이 되어

Map에 들어가고 ref = 1 인 상태이다

이후에 ClearScene에서 처리가 일어나는데 변동은 없다.

GameManager에서 해당 텍스처를 가져오는데 바로 release함수를 호출한다. 변동 X

Ref = 1

GameManager에서 SAFE\_RELEASE를 통해서 소멸.

이렇게 이미 사라진 BackBuffer는 ResourceManager가 가지고 있어서 Release함수를 호출하게 됨

(ERROR)

해결 방법.

여러가지가 있지만, 참조의 정석대로, GameManager에서 해당 텍스처를 참조하는 것이니까, ResourceManager에 넣어둔 상태로 ref = 1 GameManager 에서 참조하여 ref = 2로둔다.

이후 리소스를 정리하는 상황에서 GameManager에서 하나 지워주고 1, 나머지 하나는 ResourceManager가 정리한다. Ref = 0

오류를 정리한 것은 좋은데 StartScene에서만 안 터지는 것은 아닌지 확인해보자.

ㅋㅋㅋ

# MainScene 에서 Scene 터지는 문제.

Burt 오브젝트를 삭제하는 상황에서 이미 Scene이 사라져 있는 상황이 발생.

일단 장면이 생성되면서 ref = 1

장면관리자에서 날려주는 처리로 소멸.

은 잘 되는데, CollisionManager에서 오브젝트 정리하는 과정에서 문제가 발생하는 것으로 보인다.

아마도, WorldObjList에서 미리 받아둔 객체가 있어서 소멸이 안 되겠지. 근데 SceneManager는 이미 날라가서 해당 오브젝트가 날라가는 순간에 Scene을 찾을 방법이 없다.

SceneManager가 장면을 지워둔 상태로 CollisionManager의 WorldObjList를 소멸하는 순간 여기에 남아 있던 오브젝트가 사라지게 되면서, 해당 장면(댕글링 포인터)을 참조하는 상황

일단 오브젝트가 날라가는 상황에서 CollisionManager가 필요한지 확인해보자.

오브젝트가 날라간다. -> Collider가 날라간다. -> Collider는 자신과 충돌이 일어났던 물체와 충돌 상태를 정리해주기 위한 처리를 소멸자에서 하고 있다. (여기서 충돌관리자를 참조하게 된다.)

즉, 오브젝트는 충돌관리자를 참조한다.

오브젝트가 날라가는 순간에 Scene을 필요로 한다.

즉, 오브젝트는 SceneManager의 소멸자에 의해서 Scene이 날라가기 이전에 없어져야 한다.

그러면 CollisionManager 와 SceneManager의 소멸순서를 바꾸면 해결될 문제인지 보자.

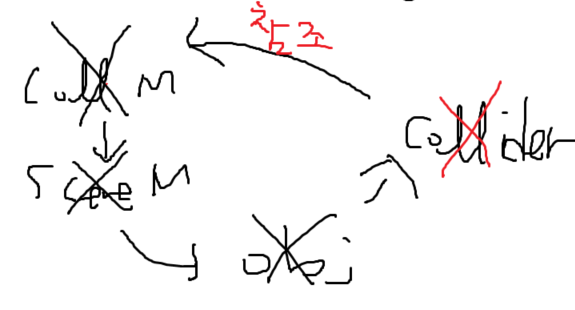
지금의 순서 : SceneManager -> CollisionManager

(결국 오브젝트는 CollisionManager에서 소멸이 일어나고 Scene에 접근할 수 없어서 문제가 발생)

바꾼 순서 : CollisionManager -> SceneManager

일단, CollisionManager를 먼저 호출하여 오브젝트가 일부는 날라갈 것임. 이후에 인접행렬을 없애주는 처리를 하고 있는데, SceneManager에서 오브젝트가 소멸이 되는 상황에서 해당 인접행렬을 이용하고 있음. (ERROR)

또한, 이미 CollisionManager의 소멸자가 호출된 상황에서 CollisionManager를 다시 호출하는 상황이 발생함.



즉, 순서를 바꾸는 것으로는 처리되지 않는다. WorldObjList를 날리는 함수를 CollisionManager에서 따로 빼서 이것을 SceneManager이전에 호출하든, 하면 되겠다.

SceneManager의 소멸자에서 미리 WorldObjList를 소멸하는 함수를 호출해주고 있다.

그러면, Scene 이 사라지기 이전에 모든 오브젝트가 정리가 되고, 오브젝트가 정리되면서 CollisionManager에 접근을 하는데, 아직 해당 정보는 남아있기 때문에(SceneManager에서 소멸하는 상황이니까) 참조하는데 문제가 되지 않고, 이후에 CollisionManager에서 남은 정보를 소멸하게 될 것이다.

돌려보자. OK 문제 없다.

다른 버튼을 누른 다음에도 확인을 해보는 것으로 한다. 오류 다 잡았다.

# 터지는 것 좀 어떻게 해봐. 끝

중력을 적용하여 다양한 동작을 구현.

일단 활성화 개념을 사용하여 오브젝트들에게 중력을 주는 것으로 하고, 플레이어의 점프를 만들어보자.

현재 타일 충돌이 일어나고 있는 상황이기에, 바닥으로 꺼지지는 않을 것이다.

오브젝트가 바닥에 부딪히고 나서 움직이질 못한다. Why ?

중력에 의해서 아래로 내려간다. 그러면 오브젝트는 가만히 있더라도 아래를 향해 이동한 것이 된다. 그러면 타일에 의한 충돌처리에 의해서 다시 올라간다. 이게 반복이다. 뭔가 바닥에 닿은 경우에 중력의 영향을 받지 않을 Flag처리가 필요하다고 판단이 된다.

타일과 충돌이 일어난 경우 즉, Y축에 의해 위로 튕겨나간 경우에 Flag를 false로 주고,

Release함수를 통해서 오브젝트가 Stage인 경우에 다시 Flag를 true로 주면 되지 않을까 ?

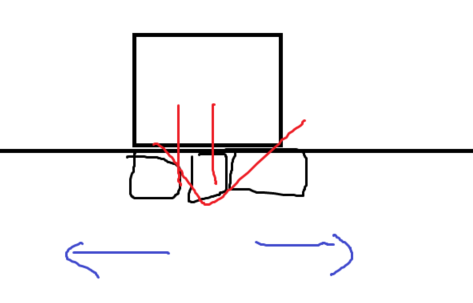
근데, release함수에서 단순히 중력을 true로 주는 상황으로 처리하는 것이 과연 맞을까 ?

여기서 또 처리를 나눠서 해야 하는 것 아닌가 ?

아니 애초에 위로 올리는 경우에 충돌영역 만큼 올리고, 조금 더 띄워주는 처리가 되어 있는 상황이다. 그렇다는 소리는 ..

근데, 다시보니까 이게 문제가 아닌 듯 하다. 현재, 타일의 크기는 커비이미지의 1/3 정도 된다.

이 상태에서 물체를 아래로 끌어당긴다고 생각해보자.



이 상황에서 좌우 이동키를 누르고 있는 상황이다. 그러면, X/Y의 이동량이 둘다 나오게 된다.

그렇다면 이동 방향이 왼쪽을 향하는 대각선, 오른쪽을 향하는 대각선이 나오게 될 것이다.

그러면 X축에 대한 처리를 먼저 할 때 이것이 바깥에서 파고든 것으로 인식하게 될 것이다.

그러면 여기서 파고든 만큼을 다시 이동시키는 것이다. 여기서 좌표가 멈추게 된다.

수업 코드에서 이런 현상이 발생하지 않은 까닭.

거기서는 시간을 중첩시켜서 떨구는 처리를 하고 있는데 Y축에 대한 처리를 하는 상황에서 시간값을 초기화시킨다. 당연히 Y의 Move값이 미세해질 것이다. Y축에 대한 변화가 없거나 아주 미세한 상황에서 X축을 이동시킨다. 그러면 Y 이동값을 없는 상태로 X만 움직인다고 판단하지 않을까 ? Y에 대한 처리는 당연히 없고. X에 대한 처리를 하겠지만, Y가 변동이 없다면 파고드는 상황이 아니다.

결론적으로 뭐랄까, 떨어지는 값을 조정하여 Y값이 바뀌지도 않을 처리를 해줘야 한다는 것인데..

나는 일정한 속도로 떨어지게 하고 싶다. Y 축을 먼저 처리를 해보는 것은 어떨까 ? 아니 상관없어. 그냥 시간 값에 대한 처리를 하는데, 매우 변동이 없도록 처리하면 안 되나 ?

아니 이상할 거 같아. 속도가 달라지잖아.

그러면, 바닥에 붙은 상황과, 그렇지 않은 상황을 구분해서 중력에 따른 처리를 하면 될 것 같다.

Y축에 대한 처리를 했다. 그러면 밟았다는 소리니까. False

그러면 다음 프레임에서 Update처리를 할 때, Y에 대한 Move값은 없다. 따라서 이동이 가능하다.

그럼 다시 true가 되는건 어느 상황이냐 ! 하면 , Late Update에서 true를 주면 되지 않을까 ?

이게 프레임이 순서상 밀리기는 하는데 그리 큰 문제는 되지 않을 것이라고 생각이 된다.

일단 기본적으로 LateUpdate에서 true(중력처리)를 주는데, 이후에 Collision쪽에서 Y축에 대한 충돌처리가 생겼다면 False. 그러면 다음 프레임에서 중력에 대한 처리가 없어지고 우리는 움직일 움직인다. (Update) 여기서 LateUpdate함수를 호출하면서 다시 true로 만들고, 이 상태에서 Y축의 이동이 없었으므로 당연히 충돌이 아니다. True인 상태로, 다음 프레임에서는 충돌처리를 할 것이고, 이러면 여기서는 이동한 만큼을 빠꾸.

그냥, 시간 같은 값을 하나 입력 받아서, 초기화시켜버리는 처리를 통해서 다음 프레임의 YMove값을 날리고, 이 상황에서 좌우로 이동하는 경우에는 … 한번 해보자.

이번 프레임에서는 움직이는 것이 되고 다음 프레임에서는 살짝 미세하게 YMove값이 존재하여 대각선을 향해 이동하는 처리가 될 가능성이 있는데, 아 모르겠다 일단 해보자.

스테이지를 WorldObjList에서 제외한다. 사용하기 위해서 참조하고 지워주는 작업을 하고 있는데, 스테이지 오브젝트에 대해서는 이것을 하지 않는 것뿐이다.

보자. 위에서 언급했던 것처럼 한번 이동을 하고 나서, 시간이 중첩되어 미세하게 Y축으로 이동이 생기는데 이것이 타일에 파고드는 정도의 양은 아닌지. 따로 문제가 생기지 않는다. (애초에 0.1f 값을 더 올려서 그런가 ..)

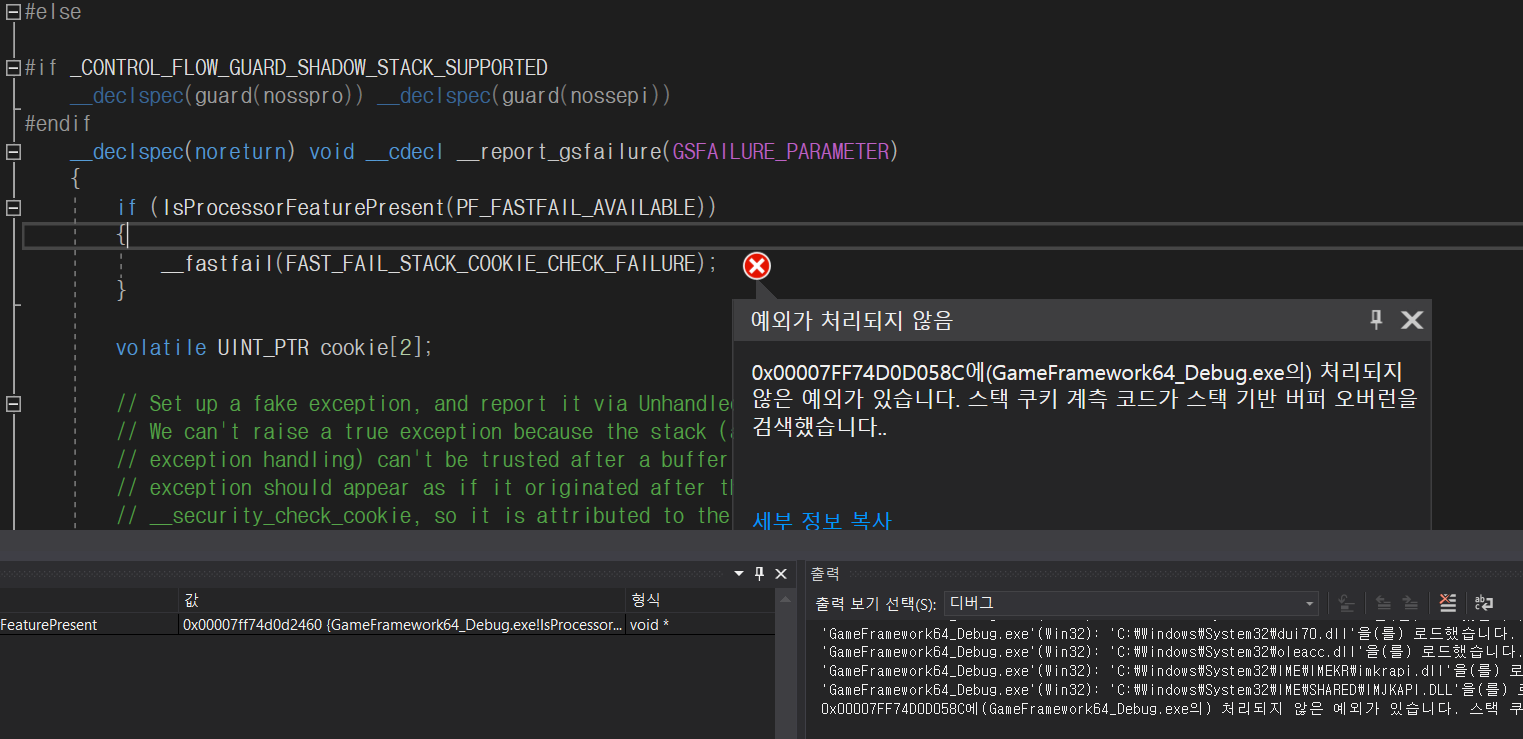
정확하게는 모르겠다. 아무튼 직선에서도 해당 처리를 해주는 것으로 하자.

시간 기반의 처리를 이용하면서 동일한 위치를 이동하게 하고 싶다면 그냥 FLAG처럼 이용해.

시간은 중첩시키되 0.f가 아니면 동일한 속도로 떨어지게. (근데 자세히 보니까 속도가 다르네)

0.f초가 아니면 730 + gravityTime \* 10.f 만큼 fTime의 속도로 내려가는 것으로 했다.

# 메모리 릭이 생기긴 하는데, 가끔씩 생기는 현상… ?!



BindAxis 할 때 처음 보는 오류로 터진다. ??

중력은 어떻게든 적용을 해봤으니.. 이제 점프를 해보도록 하자.

일단 점프는 중력을 이기는 힘을 가지고 있는데, 이 힘이 양의 값을 가지면 위로 올라가도록 하고, 음의 값을 가지면 역으로 내려가도록 하는 처리로 구현할 것이다.

일단, 점프가 필요한 물체에는 CCharacter물체에 두는 것이 좋을까 ? 아니면 보스 처리할 떄 사용하긴 하겠지만, 각 오브젝트 특유의 스킬 느낌 나도록 처리하는 것으로 할까 ?

일단 따로 따로 구현한다.

# 슬라이드

슬라이드 커맨드는 down + jumpKey(S)이다. 슬라이드가 켜진 경우에 다른 처리를 할 수 없도록 막아야 한다. Stop을 키고서 몇 초 이후에 다시 끄는 식으로 구현하면 될 것 같다.

슬라이드를 구현은 했는데, 속도를 적절하고 조절해야 하지만, LimitPixel을 정해두는 것이 좋겠다.

그냥 파워 값으로 처리하는 것이 좋으려나

이렇게 만들어 둔 파워값을 점프에도 사용하는 것이 좋겠다. 같이 사용하는 것은 아니니까

슬라이드 이거 하나에 너무 신경 많이 쓴다 적당히 해.. ㅇㅋ 끝.

원작과 비슷한 느낌의 점프, 떨어짐의 속도를 지정했다.

이제 점프 중에 할 수 없는 처리를 막아주도록 하자.

# 모션의 자연스러움.

점프 중에 공격이 켜진 경우. 원작에서는 살짝 앞으로 나가주고 있는데, 이 처리를 어떻게 할까.

점프 중에 A키를 누름. -> Flag를 킴. -> 방향을 받아놓고 처리하면 됨.

처리를 하기는 했는데 가만히 있다가 점프를 한 경우에 대해서는 예외 처리가 필요하다.

방향키를 눌러서 이동이 들어간 경우에 Move = true 인 상태가 된다.

매 프레임마다 Move의 값을 false로 주고 있다. 일단, Input에서 입력이 들어오면 move = true.

이후에 이 값을 update에서 사용하고, render에서 false값으로 줘버리면. 된다.

데이트에서 해당 값을 이용해서 false인 경우에는 아무 처리도 안 하고 가만히 있게 한다.

문제는 공격키를 누른 다음에 처리를 못한다는 것이다. (Move가 안 들어오니까.)

점프 + 공격을 한 시점에서 Move 값을 이용해서 m\_ePrevDir 의 값을 flag처리하면 되겠다.

단순히 가만히 있는 경우를 flag에 추가하면 되는 것.

Move.x 값이 0 인 경우에는 아무 처리도 안 하는 식으로 하면 된다.

(이 값은 Dir, 점프에서 공격을 하는 순간에만 받아 두는 값이니까 문제 없다.) ㅇㅇ

공격을 하던 중에. 바닥에 닿은 경우 해당 모션을 끊고서 다시 실행함. -> 흡입모션 예외처리.

현재 옵션이 Attack Left 가 아니고 Attack Right가 아닌 경우에만 DefaultClipChange. ㅇㅇ

그리고 StageLine 먹지 말라고..sa ???

현재 Input의 시점에서 Move값을 받아오고 있는데, 이것이 정확하지 않은 것 같다… 공격을 시작한 시점이라는 것이 이미 꽤나 오래전 이야기라는 듯이. Move값은 0이 나오고 있다.

(이거 그냥 매번 처리하면 되는 문제 아님 ?) ㅇㅇ

아니다, 너무 원작과 똑같이 가려고 노력하고 있었네. 그냥 내가 처리하고 싶은 대로 처리하면 그이잖아…

A 누른 상태에서 화살표 왼쪽 누르면 왼쪽으로 가고, 오른쪽.. 안 누르면 가만히 있는 것으로 했다.

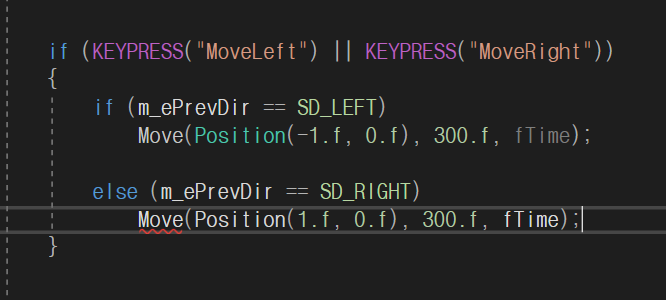
단순히 이런 식으로 처리를 하니 왼쪽 오른쪽 번갈아 누를 수 있게 되었다. 따라서, 이전 방향을 받아 두고 그거에 맞는 경우에만 처리하도록 구성했다.

왼쪽을 누르는데 왼쪽에서 시작한 경우 왼쪽처리.

오른쪽을 누르는데 오른쪽을 보며 시작한 경우 오른쪽 처리.

즉, 왼쪽에서 시작했는데 오른쪽키를 누른 경우는 처리하지 않는다.

여기에 기능을 추가하여, 짝이 맞지 않는 경우에도 일단 키를 눌렀다면 해당 방향으로 가도록 처리해야 겠다. ㅇㅇ

와우, 뭔가 문제인지 1분 이상 고민했다.

# 디테일

오른쪽으로 점프하다가 왼쪽으로 턴하는 순간 모션이 풀려버림.

가장 이상적인 처리를 말해봄. 왼쪽에서 점프하다가 오른쪽으로 턴하는 그런 경우에 반대쪽 Clip으로 바꿔버림. 근데 이것의 시간을 넘겨주는 방식으로 하면, 모션 연결되고 정말 자연스러움.

(는 귀찮으니까 나중에 하든 말든 하자. 굳이 이런 디테일한 부분은 나중에 해도 되잖아.)

앞으로 디테일 관련은 (처리 안 한 것) Tag붙임.

// 해결된 문제

그리고 슬라이드 왜 사라짐..ㄴ 점프와의 연결고리 문제.

Jump를 기준으로 Down상태면 처리를 막는다.(Slide해야 하니까)

Down기준으로 Jump 상태라면… 당연히 막아야지. 점프중에 아래키 누른다고 그거 처리하면 모션 이상하지..

그냥 downup에서만 jump Limit을 풀면 될 것 같다. 아니야..

Jump를 하고 싶은데 Down이 true라서 안 되는 것이다. 그렇다면 다른 쪽에서 false로 만드는 처리를 하면 자연스럽게 처리될 문제네. 이걸 하면 절대 이 동작은 안 된다 하는 그런 동작에서 처리하면 된다. 근데 안 움직이면 안 되서 두 처리 다했다. 근데 왜 안 되는데 .. 슬라이드가 되는 경우 false로 주니 잘 된다. ㅇㅇ

# 앞으로의 문제 12일 작업.

커비 : Collider 잘못 먹는 상황. ㅇㅇ

커비 : 선분에서 처리가 안 됨. 점프 등. ㅇㅇ

커비 : 이거 Run / Walk 모션 계속되는 것 어떻게 없애봐. ㅇㅇ

Default클립에서 구멍이 난 것이다. PrevDir을 이용해서 Run을 시작한 시점에 사용하도록 한다.

모션을 기본으로 바꿔줘야 하는 경우를 인식할 때 사용한다

왼쪽이나 오른쪽 키를 더블 클릭하는 경우 Run이 true가 되고 있다. 그리고 해당 키를 up하는 경우.

‘

점프 중에 좌/우 키를 누른 경우에는 무시. (이 때 키를 막 누른다고 한들 Run상태는 사라지지 않음) 반대로 점프 중에 달리려고 하는 경우도 당연히 무시 된다.

점프가 아닌 상황에서 달리는 중에 키를 up한 경우 달리기를 풀어버린다.

걷는 경우에서도 마찬가지로 처리한다.

커비 : 바닥에서 내려가는 상황에서 Slide가 풀려야 멋있어. (디테일)

커비 : Down 상태일 때 BackGround 움직이는 거 가능. (고치자) ㅇㅇ

커비 : 기본 속도가 빠르며 달리는 속도와 다른 점이 없음. ㅇㅇ

아니, 조금 있으면 방학 끝나는데, 4시 다 된 상태로 오면 어떻게. 포폴 망하면 누구 탓하게

좀 일찍 오자 친구야.

커비가 Line에 서있는 상태로 점프가 들어오면, 즉, Line충돌시, 커비가 Jump 상태인 경우에 위치를 바꾸지 않는 처리를 하고 있다. 근데, 이렇게 하면 생기는 문제점이 Line위에 있다가 점프를 한 경우와, 기존에 점프를 했다가 떨어지는 경우를 구분하지 못하는 것이다. 그래서, 나는 충돌함수에서 S키를 확인을 해서 처리를 하도록 하겠다.

S키를 Down한 경우라면, 라인 위에서 점프키를 누른 경우로 그냥 넘기고,

S키를 누르지 않은 경우. (떨어지는 상태라면,

로 하려고 했는데 좋은 생각이 났다.

점프를 하는 순간에 m\_bJump = true 로 주고 이후에 넘겨준 파워 값을 설정하고 0.1s 마다

파워 값을 줄여주고 있는데, 그러면 이 파워 값의 상태에 따라서 점프를 시작한 시점인지 /

점프를 하면서 떨어지는 상황인지를 파악할 수 있다.

1000.f 정도를 기준으로. 파워 값이 1000.f 이하면 점프를 시작한 이후라는 것이고, 그게 아니라면 점프를 시작하는 시점이라는 것이다.

ㅇㅇ

커비가 Big상태일 때 별을 토하는 애니메이션을 추가한다.

풍선 상태일 때 입김을 토하는 애니메이션을 추가하고 입김 오브젝트를 만든다.

입김 후에는 떨어지는 모션으로 바뀌는데 모션 추가 따로 안 해뒀으니

(Idle모션으로 하고) 입김애니메이션으로 바꾼다. ㅇㅇ

점프 상태에서 다시 점프를 하는 경우. 풍선 상태가 되고 이 풍성 상태로는 중력의 중첩을 없애주는 처리를 한다. 점프를 계속 누르는 경우에 파워 값을 이용한 일시적인 점핑 효과가 존재한다.

사용하는 float 변수로 점프를 사용해도 무방

날아다니는 것은 구현했다.

위의 동작에 이펙트를 추가.

1. **커비 별을 토하면, 날라가는 별(이펙트는… 아니지) ㅇㅇ**
2. **달리면 날리는 먼지 / 슬라이드 시에도 나온다. ㅇㅇ**
3. **빨아들일 때 나오는 먼지… ㅇㅇ**

**미리 Effect포인터를 하나 가지고 있도록 하여, Loop애니메이션을 돌리다가, 공격이 끝나는 순간 삭제해주는 것으로 한다. ㅇㅇ**

1. **입김 이펙트 (충돌체 필요함) ㅇㅇ**

**떨어짐의 별은 넘긴다. ㅇㅇ**

1. **몬스터 터지는 이펙트 별 !! (프로토타입은 제작, 적용은 이후에)**

**입김에 대한 예외처리 추가. ㅇ**

**점프 중에 방향이 바뀌는 경우 애니메이션을 반대로 출력해주는 기능.**

**Ballon상태에서도 반대로 출력해주는 기능. (디테일)**

**점프UP 도중에 풍선상태가 되는 것을 막았다. ㅇㅇ**

**달리면 날리는 먼지의 경우에는 Loop 애니메이션으로 설정이 되어 있다.**

**일단, 이펙트 또한 오브젝트에 속하니까 비활성화를 해줄 수 있다. 그렇다면, 게임이 시작되는 순간에 미리 양쪽 1개씩 생성을 해두고, 이것을 활성화 하는 식으로 하면 어떨까 ?**

**오른쪽을 향해 달리고 있는 경우에 오른쪽에 해당하는 Effect를 켜주고, 꺼지는 순간 다시 풀어주고**

**아니면 달리고 있는 상황에서 몇 초 간격으로 플레이어를 따라다니는 이펙트를 생성해주고, 생성해주는 이펙트는 방향에 따라 달라지도록 설정하는 것이다.**

먹은 상황에서 Run상태를 풀어줘야 한다.

먹은 상태에서는 날 수 없다. ㅇㅇ

충돌체에 의해서 빨아들이는 처리를 하고 있는 중에 오브젝트가 사라지지 않아서 다시 끌려오는 문제가 생기는데 이 부분에 대해서는 나중에 오브젝트 삭제를 할 예정이라 일단은 넘기자.

남은 작업

1. 보스 몬스터 관련.
2. 스테이지 관련 (오브젝트 생성 및, 배치)
3. 커비 변신
4. 툴 프로그래밍 수정 ㅇㅇ 끝.
5. 전투 시스템 구현 (+UI) -> 공격 모션 & 넉백 효과 등등.

일단은 변신을 위한 오브젝트 시스템부터 만들어보자. 변하기 위한 아이템. ? ㅇㅇ

해당 아이템은 살짝 위쪽에 충돌체를 가지고 있어서 그곳과 충돌하면 플레이어가 변신하게 되며,

이후에는 해당 부분이 없어지고 기둥만 남는다. 일정 시간이 지나면 해당 부분이 다시 생기게 되고 충돌이 가능해진다.

충돌이 된 상태에서 커비의 상태를 비교하여 자신과 다른 상태라면 교체한다.

그리고 이후에 모든 처리를 해당 상태에 따라서 처리한다.

상태에 따른 공격 / 상태에 따른 효과 등등

변신 구현 목록

고려 사항 : 변신할 수 있는 경우가 많을 수록 구현해야 하는 양은 2~3배가 뛰는 것이다.

꼭 구현하고 싶은, 혹은 쉽게 구현할 수 있는 능력을 선택하고 동작이 다양하면 이 다양한 동작을 다 일일이 처리할 것을 생각해서 간략하게 표현할 생각을 해야 한다.

솔직히, 변신 쉬운 것 위주로 몇 개만 구현하고 안전하게 포트폴리오 만들고 싶다.(시간에 찌들지 않게 안전하게 ..)

하기 쉬운 것 정말 쉬운 목록 : 일회성 커비.

Cook : 솔직히 애니메이션이 대부분이다. 몬스터 튕겨 들어오는 것은 간단하게 구현하면 될 문제고 솔직히 능력중에서 이 친구만큼 쉬운 것도 없다. (좌우 구분도 없음)

요리 모음 아이콘.

솔직히 커비 상태였다가, 공격하기만 하면 한번 애니메이션 호출하고 끝임. (개이득)

(들어가면 재밌을 것 같긴 하다) 넣자.

Crash : 이것도 거의 아무것도 없음. 보여줄 것도 없는게 흠. Pass

Mike : 구현 쉽고, 나름 괜찮음. 넣자. (리소스의 마이크로 사용) (위와 같은 이유로 개이득)

Sleep : 넣자. 이거 정말 구현 쉬움. (능력 습득과 동시에 잠에 빠짐 개이득)

(체력 회복 효과있음)

Stone : 능력이 정말 단순함 (그냥 공격하면 돌 되는 것 밖에 없음) 동작 교체 어렵지 않은 편.

합격.

Sword : 솔직히 진짜 어려운 능력인데, 이거 있으면 다크나이트 등장이 가능해짐.

그리고 솔직히 제일 멋있는 능력이라 빼놓기는 좀. 합격.

// 변신 합격 목록

Cook / Mike / Sleep

Stone, Sword(애도 나중에 하자)

변신은 이렇게만 정해두고 시간 있으면 더 추가하는 것으로 하겠다.

Mike(좌우) Sleep(좌우) Cook 부터 리소스 뽑아보자.

능력이 바뀌는 순간마다 사용하는 충돌체를 활성화 비활성화 상태로 관리하는 것도 좋겠다.

일단 리소스에서 사용할 이름을 전역변수로 사용해서 만들어 두었다.

Easy 애니메이션은 상관이 없지만, Ability 애니메이션의 경우에는 비슷한 애니메이션이 반복되는 시스템이다. 따라서, 기본 커비 애니메이션의 옵션값, 시간 정보 등 여러 정보를 알고 있고 이것을 통일시켜주는 것이 깔끔할 것이다.

필요한 게임의 영상은 따로 찾아서 추가하도록 하자.

(편집하기 전에 애니메이션 툴 좀 어떻게 처리하자)

13일 마지막 작업 애니메이션 툴 수정.

**툴 기능 편집**

너무 작은 프레임 (점에 가까울 정도의) 삭제. ㅇㅇ

딱 점 정도만 막을게. ㅇㅇ

프레임 선택 시 여러 개를 한번에 선택이 되는 기능 삭제 (하나만 처리가 되도록) (가장 위의 것.)

ㅇㅇ (강화)

Static 기능을 없애고 마우스의 위치에 따라서 처리할 수 있도록 구성.(이건 솔직히 괜찮네 다시보니까, 그냥 쓰자) ㅇㅇ

출력 위치를 이미지의 크기의 영향을 받도록 구현하자. 여윳 값을 충분히 주고, 짤리더라..

ㅇㅇ

Collision Enable 키는 이제 무의미했다. ㅇㅇ 아래로 교체.

새로운 기능 : 다중 충돌처리. 위의 키를 대신하는 기능을 넣는다. ㅇㅇ

이 키를 누르면 다중 충돌이 활성화되고, 꺼진 상태라면 하나씩 선택이 되는 것이다.

만약에 여러 프레임과 충돌이 일어나는 경우에는 가장 위에 것을 선택하는 것으로 하자.

활성화 비활성화ㅇㅇ / Render ㅇㅇ / Editvector ㅇㅇ에 넣는 과정만 확인해보면 된다.

오케이 좋았고 너무 좋네 ㅎㅎ ㅇㅇ

수정 완료.

보스

1. 디디디 : 쉬워서 간단해서 메인 캐릭터
2. 다크나이트 : 멋져서 솔직히 멋짐 (하지만 시간 없으면 버린다.)
3. Mike -> sleep -> stone-> stage(monster, ability) -> ddd(boss) // (안전선) -> Add작업.

마이크 너무 만만하게 봤는데, 화면 흔들림에다가 번쩍임 효과까지 장난 아니네 ?

ㅋㅋㅋㅋㅋ

일단 뜯자 !a

14일 애니메이션 뜯고 적용하는 작업 위의 순서대로 시작.

(애니메이션 타임 뜯기는 나중에.)

내일 아침에 일찍오자. 1시 각이야 !!

14일. 2시 26분 도착.

아니 왜 컬러키가.. 설정을 했는데 이후에 변하니 ?

아니, 이전에 호출되어야 하는 함수가 이후에 출력이 되고 있었네 하.. 1시간 와.

마이크 1.17 초 once return 1 2 3(1.4s)

Sleep 1 3 4 5 (678) \* 3 ~ ㅇㅋ

모자, 방울은 따로 이펙트로 따주자.ㅈㅇ 7초.

좌우 이미지의 좌표를 넣는 것을 왜 따로 할 생각을 했는지 이 부분은, 왼쪽 기준이든 오른쪽 기준이든 일단 좌표를 넣고서 이 좌표를 다른 쪽의 방향으로 복사를 해온 다음에 Y축에 대하여 회전을 하면 되는 것이다.

근데 이 Y축이라는 것은 이미지의 중심을 의미한다.

해당 알고리즘을 잘 생각해서 추가해준다면, 왼쪽 프레임만 따주고서 오른쪽 스프라이트는 그냥 거저먹는 방법을 구현할 수 있다.

그림을 기준으로 생각해보자.

이걸 만들어야 앞으로 리소스 작업의 시간을 1/2 로 줄여버릴 수 있다. 무조건 구현.

일단, Y축의 좌표는 그대로이다.

2개의 좌표가 필요하다. 왼쪽 상단과 오른쪽 하단을 의미하는 A와 D

그런데, 중심 X에 대해 대칭이동을 하는 경우에 기준이 되는 좌표가 변하는 것이다.

그래서 우리는 변환이 일어난 좌표를 기준으로 하여 왼쪽 상단 오른쪽 하단의 좌표를 구해내야 한다.

A - > A’를 구하는 방법. (Y좌표는 그대로.)

X좌표의 경우에는 중심축과 X거리를 구해서, 중심축에서 이 거리를 더하면 A’의 X좌표를 구할 수 있다.

A’의 X = mid.x + abs(mid.x – A.x)

A’의 Y = A의 Y

이렇게 좌표를 구하고, A와 D에 대해서 A’ , D’의 좌표를 구한다.

LR변환이든 RL변환이든 구해지는 좌표는 오른쪽 상단, 왼쪽 하단의 좌표이기 때문에

이것을 왼쪽 상단 / 오른쪽 하단의 좌표로 변환해야 한다.

// 좌우 동일.

기존에 Begin에 해당하는 좌표는 변환이후에 오른쪽 상단으로 이동이 된다.

따라서 이렇게 오른쪽 상단으로 이동한 좌표를 왼쪽 상단으로 변환하는 작업을 거친다.

프레임x 크기 만큼 빼면 된다.

기존에 End에 해당하는 좌표는 변환이후에 왼쪽 하단으로 이동이 된다.

프레임x 크기 만큼 더하면 된다.

그러면 우리는 변환한 좌표에서 Coord정보를 구해낼 수 있다.

근데, 이거 이왕할거면 통 크게 기능 몇 가지만 더 추가하자.

사본이 되는 클립을 받아둔 상태로 애니메이션 복사를 처리하는 부분.

(현재, 애니메이션의 정보를 읽어와서 처리하는 시스템이 완성이 되어 있는 것으로 기억한다.

그러니까 읽어와서 좌표 정보를 가지고 있겠지.)

그러니까 사본이 되는 객체를 저장하고 이것을 붙여 넣는 작업.

U on

i 사본 Save.

o 사본 붙여넣기. (여기서는 좌표에 대한 변환까지 모두 포함하는 작업이다.)

이미지의 중심을 기준으로 반전을 시키는 코드를 완성했다. 나는 이제껏 하던 바보짓을 안 해도 되는 것이다.

이미지를 반대로 돌리는 기능도 있지만, 그렇지 않은 기능을 추가하여, 좌표가 같으면 좋겠지만,

좌표가 같지 않더라도 단순히 옵션값을 복사해서 사용하는 용도로 쓸 수 있도록 한다.

구체적 말하자면, 좌표가 아닌 단순히 애니메이션 정보를 따오는 기능을 추가하자는 것이다.

그러면 굳이 내가 저렇게 정보를 기억하지 않더라도 쉽게 처리할 수 있다. ^^

이 기능을 추가한 이후로 정말 빠르게 리소스를 추가할 수 있게 되었다. 기존의 방식이

멋모르고 사용하는 호구에 가까웠다면 이것은 새로운 돌파구

근데근데근데근데근데근데 애니메이션을 바꾸는 방식이 매우매우매우매우매우 귀찮다.

순서가 주소순으로 들어가서 매우 불편.

따라서, 이것을 EditManager에서 string vector를 하나 받아두고, 내가 원하는 형태로 추가해놓고

ChangeClip을 하면 이것을 기반으로 바꾸는 것으로 하자 (그러면 완전히 정렬되는 개념이다)

가상함수로 AddClipName함수를 만들고 이것을 오브젝트가 바뀌는 순간마다 호출해주는 것으로 했다 그러면, 해당 vector의 인덱스를 교체해주고 이러면서 해당 인덱스에 맞는 string으로 ChangClip을 하는 것이다. 따라서 Left Right 다른 모션 이런 식으로 애니메이션이 변경이 될 것이다.

오브젝트를 바꾸는 상황에서는 이것을 다시 넣어준다.

Ok 내일 와서 나머지 스톤 리소스를 따는 것으로 하자.

로드를 하면, 저장된 클립의 정보를 이용하여, (Clip의 이름을 저장해둔 상태이다)

애니메이션을 돌리는데 필요한 정보들을 가져와서 복사한 이후에 이 값을 미리 생성해둔 클립(같은 이름)에 복사하는 것이다.

아무튼, 저장한 정보를 바탕으로 클립을 찾아서 복사해주는 시스템이 구현이 되어 있다.

미리 생각을 해두자, 왼쪽을 기준으로 애니메이션을 만들어서 저장을 했다, 이것을 사본으로 하여 오른쪽이 기준은 클립에 복사를 하는 것이다. 일단, 저장된 옵션 정보. (한쪽을 기준으로 미리 저장을 한 상태이니 생각할 필요가 없다.)

그리고 애니메이션 정보(당연히 좌우 옵션이 같으므로 생각해줄 필요가 없다)

따라서 한 쪽에 대한 애니메이션 클립을 복사해온 이후에는 그대로 붙여서 저장하면 되는 것이다.

현재 스톤 능력의 리소스와 스워드 능력의 리소스를 따둔 상태이다.

그 결과, 리소스의 양이 무지막지하게 많아 졌고, 이것을 가지고 다니는 배열의 차이가 생겨, 이것을 호출해주는 시스템을 만들어 두면 정말 편할 것이다.

예상이 되는 방법 중에 하나는 Tag를 넣어주면 해당 리소스에 해당하는 배열의 주소를 반환해주는 방식이고, 또 다른 하나는 그냥 모든 정보를 다 때려 박으면 출력해주는 방식이다.

대신에 맵핑을 통해 간단하게 표현하는 함수를 만들어서 사용했다. (생각하는 방식으로 하는 것이 매우 어려움) (내가 잘 몰라서)

이제, 기존의 시스템. Kirby만 생각하던 시스템에서 -> 다른 종류의 커비를 생각하는 설계가 되어야 한다. 이 부분은 반드시 Normal상태에서 처리를 할 것이라 든지.

아무튼, 키입력은 동일하니까, 따로 사용할 수 있는 시스템을 구현해보자.

Attack (복잡한 부분은 내일 16일에 하는 것으로) 함수.

커비 종류마다 사용하는 충돌체를 모두 초기화하고 이걸 기본적으로 비활성화하여, 해당 커비로 전환이 되는 순간 활성화를 시켜주는 시스템을 구현할까 한다. 아무튼, 일단,은 PlayerAttack 부분을 예외로 처리했다.

모션 적용 잘되고, 이제 공격하는 부분을 수정하면 될 것이다. (내일)

8월 16일 2시 45분 도착.

오늘은, 어제 적용한 커비 리소스를 직접 전투에 적용할 수 있도록 모션 작업부터 할 것이다.

가장 간단하고 쉬운 Stone부터 처리를 하도록 하자.

일단, 기본적으로 커비는 점프상태에서 다시 점프키를 누른 경우에 Balloon상태가 되는 것은 동일하다. 따라서 이 부분은 리소스만 변경해주면 될 것이다.

일반적인 커비의 경우에는 2가지의 공격이 있는데, 일단,

점프공격 -> 점프 상태로 흡입공격의 시작

기본공격 -> 흡입공격의 시작

먹고 뱉기 -> 별을 쏘는 동작으로 일반적인 커비만 가지고 있는 경우이다.

대시 공격 -> 흡입 공격.

근데, 다시 생각해볼 부분은, 기본적인 커비는 a키를 누르고 있는 경우인데 능력이 있는 커비는 단순히 A키를 누른 경우에 처리를 해야 한다는 것이다.

그래서 다른 함수를 만들어서 (키는 동일하게 사용하되, 능력이 있는 커비에만 적용이 되도록 하는 시스템을 구현해야 할 것 같다)

키가 같아서 함수는 동일하게 들어옴 but 실질적인 처리는 막는 것.

아니면 완전히 따로 처리를 하도록 할까 ? 막 스톤 커비, 무슨 무슨 커비 이런거 마다 각각 함수 따로 지정하고 사용하는 개념이지. (많이 복잡하면 그렇게 해도 되지.)

일단, 기본적인 모션에서는 동일하니까 막아주는 부분에서는 동일하게 처리하고,

일반 커비에서만 별쏘기가 가능하도록 빨아들이는 모션이 가능하도록 구성.

그리고 기본적인 구현 부분에 해당하는, 부분은 그냥 냅둬서 모든 커비가 리소스를 다르게 같이 사용하도록 했다. 문제가 하나 있노라면, 이런 식으로 구현을 해버리면, 일관성이 떨어진 다는 것이다.

그래서 나는 각 상태마다 공격 관련 함수를 따로 지정을 해놓고 사용할 것이다. 그리고 상태가 다르다면 그냥 호출이 되지 않도록 처리하면 간단하다. 같은 코드를 여러 번 사용하더라도 코드를 보기에는 정말 깔끔하게 처리할 수 있다. (가독성이 100% 증가한다면 이것을 당연히 해야지)

그러면 각 커비의 상태마다 공격부분은 따로 처리하는 것으로 하고, 기본적인 구현은 기본커비에서 구현했던 것을 많이 참고하도록 하자.

그리고 이 Attack 부분을 제외하고는 따로 처리를 해줘야 하는 부분이 없다. 커비의 기본적인 모션은 동일하고, 직접적으로 차이가 나는 부분은 공격에서 드러나기 때문이다.

기본적인 커비에 대한 공격처리는 이미 구현이 되어 있으므로 생각하지 않아도 된다.

일단, 능력 있는 커비에 대해서는 풍선 상태에서 공격이 일어난 경우에만 처리를 했으니 테스트해보자. ㅇㅇ

* 일단은 바닥과 충돌하는 경우에 애니메이션을 바꾸는 부분 또한 다르게 처리해줘야 하므로 함수를 만들어보자. ( 상황을 따로 잡아줘야 하니까) ㅇㅇ
* 이 부분에 대해서 처리를 했고, 더 이상 볼 부분은 따로 없다고 판단된다. (왜냐면 단순히 풍선상태에서 모션 바뀌는 것, Run상태 풀리는 것, 클립 바꾸는 처리 이런 것 밖에 없으니까) 아무튼 애니메이션과 상태의 변화가 자연스러운지 확인해보자. ㅇㅇ

자연스러우니 이제는 각 커비별로 공격하는 것을 만들어보자.

---- 아무튼 이 부분의 처리는 단순히 리소스만 생각해서 바꿔주면 되는 부분이다 ---

능력이 있는 커비라면 …

점프 공격 ->

기본 공격 ->

대시 공격 ->

스톤 : 자나깨나 돌덩어리로 변신. 점프 상태가 되든 뭐가 되든지 간에 돌덩이리로 변신하는데, 이 상태가 되면 바닥으로 떨어져야 한다. 키 입력은 하나만 처리한다. Stone 상태를 푸는 키.

점프공격 : 돌

기본 공격 : 돌

대시 공격 : 돌

구현을 하는데, 기존의 시스템을 이용해야 하는 부분이 어느 정도 있다. 이것을 적용할 수 있는 상태로 변경을 해둔 상태이다.

우리에게 필요한 것은 세분화된 업데이트처리. ㅇㅋ

상태에 따른 flag도 따로 두어야 하는 걸까 ? 글쎄… ㅇㅇ 아마 그럴 것 같다.

각 커비의 상태마다 Static Size 당연히 달라지는 부분이고, (소스 크기상)

충돌체 또한 달라지는 부분이다. 사거리가 다르고, 사용하는 충돌체가 다르다.

일단, 여러 충돌체를 다 생성하고, 각 상태마다 사용할 충돌체를 따로 두는 것이 좋을 것 같다.

그리고 특정 커비가 활성화되는 순간에 이것을 활성화/비활성화 하는 처리.

이것이 필요한 이유가, 우리가 활성화시킨 충돌체(즉 상태에 따른 충돌체에) 직접 상태에 따른 함수를 적용할 수 있다는 것이다.

생각해보자, 일반 커비의 충돌체에서는 빨아들이는 처리를 하기 위한 코드가 작성이 되어 있는데, 이 코드위에서 예외처리를 계속 난발하며 코드를 더럽히는 것보다는… 차라리 상태에 따른 충돌체, 상태에 따른 충돌체 등록 함수 스타일이 훨씬 낫다.

주의 : 커비를 바꾸는 상황에서 충돌체 비활성화 / 활성화.

다시 사용할 수 있는 충돌체를 생각해보고, 이것이 가능하다면 Size변경을 이용해보도록 하자.

안 되는 것은 따로 생성하는 걸로.

일단, 현재 충돌체를 가진 물체는 모두 타일을 뚫을 수 없도록 되어 있지 않나 ?

이건 나중에 처리해보자.

KirbyBody -> 재생가능 Size변경 필수

KirbyAttackRight, Left 재생 불가능한 충돌체

스톤 커비. : 그냥 Body로 다 처리하면 되겠는데 ?

바꿀 필요도 없겠는데.. ? 함수이름 정도만 ?

스톤커비는 stone상태인 경우에 몬스터와 충돌하면 몬스터가 다치도록 한다.

그게 아니라면 커비가 아프겠지.

돌 상태인 경우에는 움직일 수 없고, 모든 입력을 받지 않도록 한다.

다시 공격키를 누르면 상태가 풀린다.

현재 바닥과 붙는 경우에 대한 처리(애니메이션 변경을 수정해야 한다)

Stone Jump Left(이상 없음) / Stone Walk ㅇㅇ / Sword Walk Left ㅇㅇ/ Stone Attack ㅇㅇ시간 줄이기.

Sword run right

나는 몰랐는데, 애니메이션 반전하는 과정에서 몇 픽셀이 빗나가서 살짝 이상하게 짤리는 부분이 있다. 심지어는 Pen의 기본 3px 정도의 값 때문에 보이지 않는 부분도 있다는 점.. gee

Stone 이펙트까지 지정을 해봤으니, 스워드에 대한 처리를 해야 한다. 일단 밥부터 먹고,

스워드의 경우에는 콤보 시스템이 존재하기 때문에 내부적으로 이것을 처리해주도록 한다.

상태에 따른 처리가 필요해서 enum문을 하나 써야한다.

공격을 시작하고 난 이후에 공격모션이 나오고, 이 모션이 나오는 동안은 커비는 움직일 수 없다.

이게 3콤보로 반복이 되는데, 공격키를 누른 경우에 일정 시간이 지나기 이전에 해당 키를 눌렀는지에 대한 처리가 필요하다. 예를 들면, 처음 키를 누른 이후에 0.3s 이후에 다시 눌러야 두 번째 공격키로 인식을 한다는 식의 ..

애니메이션에서 현재 사용하고 있는 클립의 진행상황을 저장하고 이것을 얻어오는 시스템이 있다면 멋지겠는데 ? (나중에 추가하자, 디테일)

공격을 시작한다. 그러면 0.5s간 Stop 다른 입력을 못 받는 상태가 된다. 이 상태로 0.3s 이전에 키를 다시 누른 경우 / 0.3s 이후에 누른 경우. (초기 상태가 되어서 처음 공격모션으로 처리된다)

이전에 누른 경우에는 one상태일 것인데 해당 모션을 출력한다. 근데 만약에 0.3s 이전에 키를 눌렀다고 해보자, 그러면 이전에 출력한 애니메이션을 버리고 새로운 애니메이션을 출력할 것인 것 이 처리가 맞는가 ? 아니, 틀리다. 기존의 애니메이션의 시간을 보장해준 상태로 다음 애니메이션을 처리해야 하는 것이다. 그러니까, 기존에 입력의 처리를 받았다고 하더라도 ChangeClip을 통해서 바꿔버리는 처리는 하면 안 된다. 현재 해당 공격 클립은 모두 once return으로 처리가 되어 있으니 Default클립을 설정하는 것은 어떨까 ?

근데 이렇게 처리를 하여도 연속적으로 키를 난사하면 디폴트 클립 값이 덮어씌워지므로 문제가 생길 여지가 다분하다.

그러면 이것은 어떨까 ?

Attack 부분에서는 단순히 키를 미리 입력 받는 용도로 사용하는 것이다. 직접적인 처리는 Update에서 따로 옮겨서 하는 거지. 그리고 이것을 마구 누르는 경우에도 처리가 되도록, 그냥 키만 저장하는 것이다. 예를 들어 2번의 입력이 있었다면, none->one->two 의 상태로 이동했을 것인데, 이것을 직접적으로 처리하는 곳은 Update이고 우리는 two애니메이션까지 출력해주면 되는 것.

짱구 열심히 굴리고 있어 !

처음에 공격을 시작하면

m\_eSwordAttack = SA\_ONE; // ONE 애니메이션까지 호출한다는 의미이다.

m\_eSwordState = SA\_ONE; // 현재 호출하는 애니메이션이 SA ONE에 해당한다는 의미이다.

SwordKirbyChangeClip(SW\_ATTACK1); // 위의 상황에 맞게 ONE

SwordKirbyDefaultClip(SW\_IDLE);

이런 처리를 하고 있지.

3번. ~-> m\_eSwordAttack = SA\_THREE (호출해야 하는 애니메이션은 3번째까지이다.)

One 돌리는 중-> one 끝. 현재 돌려야 하는 애니메이션은 3까지라서 커서

상태를 2로 바꾸고 2애니메이션을 호출. Default idle설정.

2 돌리는중 -> 2 끝 -> 현재 돌려야 하는 애니메이션이 3보다 큰지(다음 것을 돌려야 하는지 확인) -> 2까지인 경우에는 현재 클립(idle로 설정하고 SwordAttack 을 NONE으로 줘서 초기값으로 만듬) -> 더 큰 경우 3을 셋팅. 3애니메이션 시작.

3애니메이션의 경우에는 끝난 경우에 초기상태로 돌리고 끝냄.

현재 구조상 키를 난사한다고 하더라도 출력해야할 애니메이션이 고정이 되어 있는 상태이고

해당 애니메이션까지의 호출을 끝내야지 초기상태가 들어가기 때문에 여러 번 누른다고 해서 중복으로 처리가 되지는 않는다.

중간에 다른 애니메이션으로 교체가 되는 경우에는 해당 애니메이션 클립(공격모션)이 끝났다고 인식하여 none값을 줘버리고 default로 처리하고 있다.

Stop 처리의 경우에는 우리가 호출하길 원하는 애니메이션이 끝나는 시점에서 false를 주는 것으로 했다. (이로써 공격때 자동으로 멈추게 된다)

대쉬를 하면, 대쉬 어택을 true로 줘서 이펙트를 생성하지 않도록 하고, 끝나면 false로 준다.

그러면 다음 대쉬를 하는 경우에는 다시 Dash대로 처리.

이후에 끝나고, Slide의 경우에는 슬라이드에 맞게 이펙트를 생성한다. (false상태니까)

자 일단, 스톤, 스워드에 대해서 모든 애니메이션을 적용시켰고, 모션 적용까지 다 해줬다.

내일 17일은 능력 있는 몬스터의 소스를 따는 작업을 하고 몬스터를 움직이는 처리를 하며

플레이어의 오프셋을 맞추는 작업을 진행하도록 하자.

이거좀 잡고 가자, 왜 커비 가운데 출력 아니야 ? 이거 고치고 간다.

8.17 1시 17분 도착

오늘은 몬스터를 살아서 움직이도록 하기 이전에 화면 흔들기부터 구현을 해보도록 하겠다.

카메라를 가지고 있는 SceneManager에 카메라를 흔들지에 대한 여부를 만들고 이 여부를 true로 주는 것이다. 그리고 보통 몇 초간 흔들지를 정할 것이니 이것에 대한 시간 변수를 받으면 좋을 것 같다.

외부에서 카메라 흔들기를 적용하는 것은 get\_single을 이용해서 시간을 주면 해당 시간에 맞춰서 bSway 변수를 true로 준다(내부적으로) 시간을 중첩하는데 이용할 변수도 필요함.

카메라에 값을 주면 일정시간 흔든 이후에, 다시 카메라 흔들기가 종료된다.

카메라 흔들기 완성.

자 이제 Line 부분을 고쳐볼건데, Line과 충돌이 일어났는지에 대한 여부를 받아놓고 위에서 해당 부분이 true라면 즉, Line충돌인 경우에는 x축으로 팅겨나가는 현상을 막아버릴 것이다.

라인의 끝부분에 서서 문제없이 처리하기 위함이다.

이 부분은 타일 사이즈 줄여서 맞추는게 답인 것 같다. ㅇㅇ

메모리 릭 나중에 (디테일)

스테이지 릭 남는 현상

Stage 생성 1

sceneManager 등록 1

scene manager -1

Layer -1

Stone Walk Left 애니메이션 수정

값을 올려주는 처리를 할 때, 수정…. 수정 수정…

Input -> update -> Collision - > render

// Collision에서 true값을 주고, 다음 프레임에서 update가 영향을 받아서 처리가 되지 않도록 하는 것. 그리고 중력에 대한 처리를 한 이후에 다시 풀어준다.

Obj -> Update (Collision = false) -> virtual Update -> (false)

이전 프레임에 Line충돌이 일어났었는지 확인하고, Collision에서 일어났었더라면, 무조건 Y축에 대한 처리를 하는 것으로 하자. 이거 안 끝나겠다. 여기서 그만하자 일단.

현재, LineCollision을 하면서 모든 처리를 이곳에 맡기기 때문에 실제로 Y축에 대한 처리가 안 되고 있다는 것이다. (빠지는 느낌을 준다)

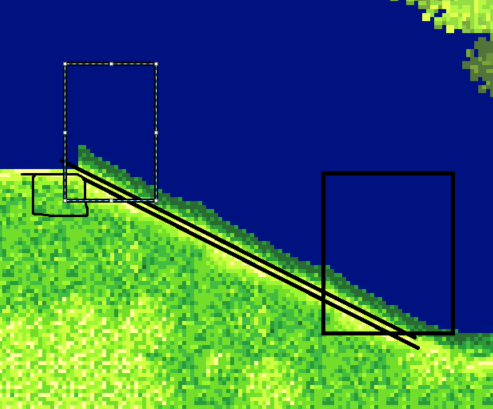
Middle Collision을 false로 주고, 만약 Y 충돌이라고 하더라도,

즉, MiddleCollision(Line충돌이면서) 이 false인 경우에 Y에 대한 처리가 필요하다.

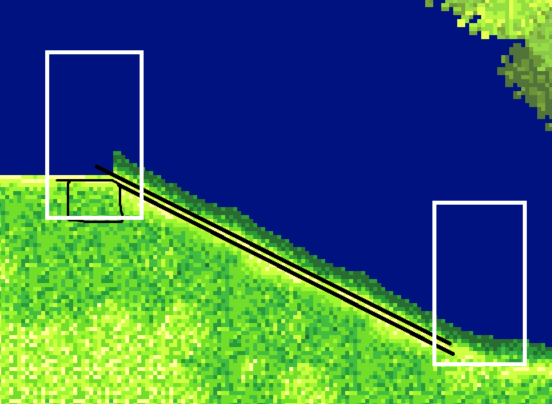
충돌이면서 MiddleCollision이 true라면 이 경우는 Y축에 대한 처리가 필요하지 않다.

X 에 대한 처리 : 직선과 충돌이 일어나면 타일의 충돌처리는 무시해버린다.

직선과 충돌이 일어나지 않은 경우에는 당연히 기본적으로 처리한다.]

Y에 대한 처리 : 일단 현재 충돌이면서 MiddleCollision이 true인 상태. 

이런 상태에서는 직선을 따라가야 하므로 타일에 대한 충돌처리를 하지 않도록 했다.

다만, 이것을 벗어나서 즉, 중심이 직선 밖에 존재하는 경우. 

이런 경우에는 타일의 충돌로 인해서 올라가야 하므로 타일의 충돌을 만들어두었다.

(일단, 충돌이 일어나서 X축에 대한 처리는 하지 않기 때문에 위로 올라가는 수밖에)

+ 현재 충돌이 아니고 이전에 충돌이 일어난 경우에 파고드는(이미 파고들어간 상태가 되어) 경우가 생겨서 예외처리로 살려줬다.

(근데 이건 위의 경우에서 제대로 처리하면 없어지는 부분아닌가 ?)이 아니였다.

보니까 이전에 충돌이 일어난 경우에는 한 프레임에 대해서 올려주는 예외처리가 되어 있어서

이후부터는 문제가 생기지 않는다(충돌이 아니고 올려줬으니까)

현재 라인과 충돌이 일어난 경우에 타일에 대한 충돌은 해주지 않기 때문에 Intersect +- 30.f

만 인식하여 처리를 하여도 큰 문제는 생기지 않는다.

타일의 모서리에 충돌이 일어난 경우에는 직선과 충돌이 되는 순간.(return 의 경우에)

일단 LineCollision을 인식해주고, 그러면 타일에 대한 충돌처리가 막히게 되나 ?

X축 막힘 , Y축 LineColl + Middle False -> 충돌을 하는 것으로 되니까. 이 경우에도 Middle함수를 true로 호출해주기로 하자.

그렇다면 아예 직선에 대한 충돌처리도 안 하고, 타일에 대한 충돌처리도 막히게 되어서,

중력에 의해 떨어져 중간에 대한 처리가 될 것이다.

(이 부분은 귀찮아서 그냥 타일을 파버렸다. 이렇게 해도 문제는 없으니)

다음 작업하자. 다음 작업 다음 작업 !!

몬스터의 이동

몬스터는 자신의 패턴에 따라서 움직임이 다르다. 고양이의 경우에는 뛰어다님.

Burt의 경우에는 화면 밖을 넘어가는 순간 사라지게 해야 하며,

Grizoo는 바닥을 돌아돌아다니다가, 벽에 만나는 경우 반대로 돌아간다.

Twizzy 이 분은 계속 점프하면서 다니거나 날아다님.

enum OBJECT\_TYPE

{

OT\_NONE,

OT\_NORMAL\_MONSTER,

OT\_PLAYER,

OT\_PLAYER\_ATTACK,

OT\_MONSTER\_ATTACK,

OT\_BOSS,

OT\_ABILITY\_MONSTER,

OT\_UI,

OT\_STAGE

};

몬스터를 복사해. 그러면 기본적으로 Layer의 ObjList에 들어가면서 ref = 1이 되는거지.

그런데, 이 상황에서 타일 충돌을 할 가능성이 있는 오브젝트들을 추가하기 때문에 ref = 2 가 된다.

즉, 우리는 타일 충돌을 하면서 이것을 없애주는 처리를 해줘야 한다는 소리.

Burt의 경우에는 타일과 충돌이 일어나지 않으므로 처음부터 추가하지 않도록 하고,

Ref = 1인 상태에서 화면 벗어나면 죽는다.

여기서 프로토타입은 생성 1 -> Layer 배치 그대로.

Gizoo -> world Obj 1 / Layer 1

삭제.

이제껏 충돌체만 삭제해주고 있었다.

빨아들이는 처리의 경우에는… 어떤 식으로 진행해야 할까 ?

플레이어가 먹어대는 몬스터를 List로 가지고 있고, 이 친구들을 다 일괄적으로 처리하는 것이 좋을 것 같다. (일단 따로 업데이트로 처리를 해야지 이건 아니다)

내일은 기존의 몬스터가 움직이도록 만들고, 충돌처리 및, 삭제 등을 구현한다.

스테이지에서 생성되는 시스템까지 구현하고 시간이 남는다면 리소스를 뽑는 것으로 하자.

몬스터 Spawn 시스템

일단, Spawner 클래스를 하나 만들어서, 특정 몬스터가 생성되는 시점을 이 클래스에서 관리하는 것으로 한다.

해당 클래스는 생성할 몬스터의 Tag값을 가지고 있고, 몬스터의 생성 조건을 처리하는 함수가 등록이 되어 있다. 몬스터의 생성위치는 해당 Spawner의 위치로 한다.

몬스터가 생성이 되는 조건은 단순한데, 카메라의 Rect안에 스포너의 위치가 들어가 있다면, 생성이 된다. 그러니까 구체적으로는 false -> true가 되는 상황에서 생기는 것이다.

다음 프레임에서는 true -> true인 상태가 될텐데, 이 경우에는 생성하지 않는다.

즉, 스포너가 나오지 않은 상태였다가 나온 경우에 몬스터가 생성이 되는 시스템이다. 나온 상태였다가 지속적으로 나온 상태일 때는 처리하지 않는다.

이러면, 자연스럽게 플레이어가 와리가리 하는 상황에서 몬스터가 출몰하게 된다. 한번 구현해보도록 하자.

몬스터의 생성 지점은, 스테이지마다 가지고 있어야 할 정보이다. 따라서, 우리는 스테이지가 해당 스포너 List를 들고 있도록 하여, 매번 스테이지에서 처리하는 시스템을 만들고, 스테이지를 통해서 간접적으로 우리가 생성할 오브젝트를 설정하는 기능을 만들어주면 좋겠다.

일단, 스포너 클래스를 제작한다.

스포너 클래스를 제작하였으니, 이제 스테이지에서 해당 클래스를 가지고 있는 형태로 만들어두겠다. (해결 2개씩 먹도록 고쳤다)

한번에 커비가 3개를 먹는 상황 혹은, 너무 빨리 먹기 시작하는 상황을 없애고 싶다.

뭔가를 먹은 경우에 take를 true로 준다. 이 take가 true가 된 상황에서는

M\_fPower 및, m\_fPowerTime 을 사용하는 곳.

Jump / Vomit / Balloon / Slide / breath

변수를 재활용하는 취지는 좋은데, 상황이 겹친다면 문제가 생길 여지가 있다.

Jump, Slide, Balloon 은 같이 사용해도 문제가 생길 여지는 없다.

Vomit, breath의 경우에는 위의 3가지 경우와 같이 사용될 여지가 있으니 바꿔주는 것이 맞다.

Ex : balloon + vomit / balloon + breath / Jump + breath

M\_fVomitTime m\_fBreathTime

두 변수를 통해서 Vomit Breath 에 해당하는 Time변수를 따로 두었으니, 이제 공통된 변수에 접근하여 생기는 문제는 없을 것이다. (점프가 이상하다든지..)

해결한 것으로 보인다.

스포너 클래스를 보면, 현재 플레이어의 위치와는 상관없이 무작정 고정된 위치를 생성을 해주고 있는데, 처리는 2가지 정도가 있을 수 있다.

첫째, 그냥 플레이어의 위치를 토대로 (지금 좌우 구분없이 카메라 좌표에 충돌했는지만 확인하는 시스템으므로 플레이어의 위치를 토대로 셋팅을 해버린다)

둘째, 플레이어의 위치를 보고, 왼쪽이라면 왼쪽에 존재하는 경우만 생성을 한다.

오른쪽이라면 오른쪽에서 오고 있던 경우만.

왼쪽이라면 왼쪽에서 오고 있던 경우만. (십중 팔구 이거, 원작 스타일)

원작 스타일대로 구현하겠다. (구현 OK)

Grizoo / Twizzy Move

Grizoo의 경우에는 MovePixel이 있어서 해당 Move값을 이동하면 방향이 자동으로 바뀌게 할 것이다. 스포너는 스폰 타입을 다르게 가지고 있어서, 이 타입에 따라서 처리가 되도록 만들자.

아니다, 그냥 만들자.

그리주는 중력을 주고, Line / Tile 충돌주고, 스피드 설정하고 좌우로 이동하는 것으로 했다.

이러면서 돌아다님

이 친구 계속 타일 날라오면서 타일충돌 때문에 바닥 스크래치 내는 현상이 일어났는데, 처음에 로직 구성할 때 타일 충돌을 안 하는 물체는 넣지 않게 되어 있는데, 이 친구의 경우에는 처음에는 추가하고 나중에 꺼버리기 때문에 이것을 처리해야 한다.

그리고 몬스터에 대한 Line충돌 처리는 따로 안 해주고 있는데 비슷한 로직이니까 복붙하도록 하자.

몬스터에 대한 처리는 DisEnable 형식으로 교체해뒀고, 이제 Effect만 적용하면 되겠다.

아니, 이걸, 하기 쉬운 방향으로 정의를 해야지.. 친구야.. 일일히 고통스럽게 만들면 어떻게..

지금이라도 다시 return 하자.

기본적으로 Tile충돌은 다 하도록 한다. (하는 오브젝트가 많으니까)

Line충돌은 false

중력 false

중력 처리하는 물체들.

플레이어, 몬스터

모두 적용해줬다. 진짜 바보 같은 짓했네..

Grizoo가 Line과의 충돌에서 이전 충돌을 확인하고 있는지 봐야 한다.

(근데 이건 충돌체가 뒤에 있어서 그런 것 같다) 고쳐주자.

좌우에서 벽 만나면 방향 바꾸도록 grizoo 잘 모르겠다.. 계속 들어오는거 보니까 둠칫둠칫 심각하네

Grizoo의 경우 최대 생성 개수가 제한이 되도록 한다. (나중에 만들자)

좌우로 둠칫둠칫 하는 것을 보니, 중력이 있음을 의미한다.

즉, Stage충돌에 대한 처리를 하고 있는 경우에는 중력의 처리를 하지 않도록 되어 있는데 이것이 제대로 처리가 되지 않는 것이다. 커비의 경우에는 내부적으로 아래로 떨군 경우에 대한 처리를 적용해주고 있었는데 여기서는 다른 오브젝트에 대해서는 적용하지 않으니 커비를 제외한 오브젝트들(대표적으로 grizoo)은 결국 계속 바닥으로 떨어지는 처리가 추가되었을 것이다.’

이 친구는 움직이는데, 충돌로 인해서 대각선으로 빠진다는 말이지 그러면 X축에 대해서 처리를 하는 상황에서 파고 들어갔다고 인식이 되어서 둠칫둠칫 동작이 일어나게 되는 것이다.

이것을 수정한 이후에는 아래로 빠지지 않으니 좌우 충돌을 명확하게 축으로서 판단할 것이다.

제대로 안 되니, 이 부분에 대해서는 문제가 없을 것 같은 부분에 배치를 하도록 하자. ^^

이제 스포너에 몬스터의 개수를 제한하는 기능을 만든다.

아니면, 시간에 대한 처리 여부를 담아서, 2가지 조건을 같이 판단하는 것이다.

충돌이든 아니든, 상관하지 않고 시작을 재도록 할까 ? 그러도록 하자.

생성을 한 이후로 시간을 재면서, 시간이 풀리는 순간

그냥 생성 개수 제한을 주도록 하겠다.

Burt는 그냥 막 생성하고, gizoo는 1개로 제한하자. (스포너에서의 제한.

스포너가 아닌 경우에는 그냥 생성이 될 것임, 치트키)

Twizzy의 경우에는 플레이어 보면서 짹짹 거리는 타입이 있고, 하늘 날아서 돌아다니는 타입이 있는데 둘다 만들어주도록 하겠다.

짹짹이라면, 일정 시간을 기준으로 점프인 상황 / 아닌 상황으로 나뉘는데,

점프가 되는 경우라면 파워와 파워 시간을 초기화한다.

트위지 fly는 계속 날아다니고,

트위지는 생성되면 점프인 상황 점프가 아닌 상황으로 나뉘어서 계속 뛰어나닐 것이다.

1초마다 상태가 변하는데, 방향은 커비를 계속 바라볼 것이다.

점프인 상황에서는 점프를 하고, 점프가 아닌 상황이라면 점프를 하지 않는다.

트위지에 대한 FLY / 일반 구분해서 잘 만들어 주었다. 이제 먹으면 전투를 구현해보자.

일단, 이펙트의 경우에는 tile과 충돌하면 죽지 않는가 ? 근데 너무 일찍 죽어서 좀 그렇네..

사망 시간 미리 정해놓을까..

플레이어의 공격에 충돌하면, 데미지를 먹고, 죽거나 혹은 넉백이 되어야 한다.

일단은 Hit애니메이션을 주는 것으로 하자.

일단, 뭔가와 충돌이 되면 넉백이 되는 기능을 구현해보자.

character쪽에 구현하는 것으로 하고, character가 넉백이 온이 되면, 위로 몇 픽셀을 올리고,

자신이 가리키는 방향의 반대 방향으로 날려주면 되지 않을까 ?

그리고 죽을 때 이펙트도 팡팡 터지는 거 필요하다..

커비 너무 할 거 많아.. ㅋㅋㅋㅋㅋ

이펙트 하나를 넣긴 했는데, 하아, 여기서 넉백하고, 튕겨나가게 해주면 뭐 거의 다 한거지 ㅇㅇ

이제 집가자.

넉백 효과, X Y 분리해서 만들도록 하고, 알파블렌딩 효과로 깜빡이도록 만든다. (무적 상태의 표현)

알파 블렌딩 효과가 어떻게 이루어지는지는 이해했다. 이미지를 그대로 복사해서 알파 값에 따라 출력하는 것은 컬러키를 지정하는 방식과는 맞지 않다.

Transparentblt(colorkey 지정)함수와 AlphaBlend 함수를 같이 사용할 수 없어서 생기는 문제다.

따라서 이 과정을 메모리 DC를 하나 생성해서 원본 출력DC에서 해당 자리를 복사해온 상태로, (배경복사) 여기에 AlphaBlend함수를 이용하여 컬러키를 뺀 캐릭터를 출력, 그러면 우리가 설정한 배경 + 캐릭터가 나오게 되는데 이것을 알파블렌딩 적용하여 원본 DC에 출력하면 되는 것이다. 그러면, 배경의 경우에는 100 % + 50 %로 100%와 같은 상태. 캐릭터의 경우에는 50%만 적용된 상태가 되는 것.

개념은 이해했는데, 일단, 오프셋을 맞추기 위해서 RenderPos의 개념을 넣어야 할 것 같다.

RenderPos는 Pos를 기준으로 출력하는 위치가 달라지는 상대적인 위치를 말한다.

이것으로 출력할 위치를 제대로 처리해주는 시스템을 구현해야 한다.

오프셋 맞추는 작업이다.

현재, pivot이 잡혀 있는 상황을 고려하여 왼쪽 상단의 좌표를 구해주고 있는 상황이다.

우리는, 어느 위치에서 RenderPos를 조정할지는 상관없는데, 아무튼 Pos를 기준으로 몇 픽셀을 옮기든, LeftTop 출력위치를 기준으로 Offset을 조정하든 해야하는 것이다.

그리고 이 RenderPos는 출력하고자 하는 애니메이션에 따라서 오프셋이 달라질 수 있는 부분이다. 그래서 애니메이션에 맞춰서 적용이 되어야 하는데, 모든 정보를 하드코딩할 것이 아니라면, 이 오프셋을 조정해주는 시스템도 구현을 해야 할 것이다.

근데 이 부분의 경우에는 아무리 보아도 각 클립의 정보로 셋팅이 되어야 하는데, 무서운 점은 저장되는 이름이다.

그러면, 새로 이 정보를 저장하는 것으로 애니메이션 툴을 수정하고, 하는 김에 Rect도..

또한 해당 정보를 읽어올 수 있는 시스템을 구현하며, 이 정보를 따로 읽어 오는 시스템을 구현해야 하는 것이다.

해당 클립의 Offset을 지정하여 조정할 수 있는 시스템을 만들어둔다. 일단은, 저장 및 로드는 뒤로 미루고, 애니메이션 툴에서 해당 정보를 저장하는 기능부터 만들어야 한다.

즉, Static Size의 위치를 보여주고 이 위치를 기준으로 오프셋을 맞추는 개념.

클립에 Offset 정보를 추가했다.

EditManager에 옵션 정보를 두고 offset을 설정하는 모드를 사용하여 키를 같이 쓰는 것으로 하자.

솔직히, 프레임당 오프셋 잡는 건 좀 오바. 왜냐면, 이건 애니메이션 프레임을 잡는 것으로 맞추는 부분이니까.

모드 FrameAdj Mode와 Frame Offset 모드를 둬서 버튼을 같이 이용하도록 하자.

간격을 TRUE인 상태로 두면, ADJ모드가 활성화되어서 이것을 맞출 수 있는 상태가 되고,

FALSE인 상태라면, OFFSET모드가 활성화 되어서 해당 키를 통해서 오프셋을 설정할 수 있는 것이다. 같이 사용하는 경우가 있는지만 보면 된다.

아니면, 화살표를 가지고 처리를 하는데, 아무 프레임도 누르지 않은 상태로 화살표를 누르면, 이 값을 조정하는 것으로 하는 것이 어떨까 ? 아니라면 다시 복귀시키면 되잖아.

아니 그냥, FLAG 하나 더 추가해서 7번키로 이거 조정할지 말지 상태를 받는 거야. 그리고 해당 키가 TRUE가 된다면, 이제 마우스와 키보드를 이용해서 OFFSET만 조정할 수 있는거지.

COORD\_STATE에 offsetDoing flag 추가,

Offset change 상태로 CCS\_OFFSET 추가.

일단 출력부터.ㅇㅇ / Save Load / 바꾸는 과정에서 offset 읽어오기 ㅇㅇ / 움직임 적용.

오프셋을 설정하는 순간에 static 사이즈에 대한 처리도 해줘야 한다.

FLAG가 굳이 필요한가 ? 그냥 ON 되는 경우에 마우스와 키보드에 대한 처리를 OFFSET 지정에 맞춰버리면 되는거 아니야 ? 그렇지 ㅇㅇ

Offset이 true가 되는 경우 프레임을 선택해서 움직이고 마우스도 이동하고 사이즈를 증감하는 그런 처리가 모두 막히는 것이다. 애초에 선택자체가 안 되도록 하는 것이 좋지 않을까 ? ㅇㅋ

오프셋이 켜져 있는 상황에서는 기존의 충돌을 활용하여 선택해서 움직이는 처리의 모든 과정과,

사이즈를 재설정하는 기능을 모두 처리하지 않는다. (애초에 충돌처리도 막아버리고, EditCoord도 날리고 상태정보 또한 날려버리니) 할 수 있는 것은 없다.

이 상태로 마우스와 키보드를 활용하여 오프셋을 조정해보자. 처리가 잘 막히는지도 확인하고,

ㅇㅇ 잘 막히는 것 같다. 근데 마우스는 정말 필요 없는 기능이라고 판단된다.

없앴고, 이제 offset이 on이 되는 경우 기존에 처리하던 동작은 다 막히게 된다.

(잘 막았다)

Save Offset은 계속 출력이 되고 오프셋 모드인 경우 이것을 출력해주도록 하자. 잘 된다.

이제 우리가 이렇게 설정한 오프셋을 출력화면에 보여주는 것으로 하자.

일단, Offset이 On인 경우에 static Size를 출력해준다. 현재, 중심위치를 기준으로 하여(pivot)

Static Size를 통해서 충돌체가 잡혀 있는 상태이다. (모두 그렇다)

그러니까 우리는, 이 StaticSize를 이용해서 출력을 해야 한다는 것인데…

이것을 게임에 적용하는 것과 동일하게 보여주어야 한다는 것이다. But how ?!

애니메이션 툴에서는 보여주는 애니메이션 자체를 옮기기는 좀 그렇다.

따라서, 애니메이션의 위치가 아닌 보여주는 프레임의 위치를 다르게 주는 것으로 한다.

이렇게 하면 우리가 옮긴 좌표는 RenderPos를 기준으로 아래로 내리고 오른쪽으로 이동하면서

출력이 될 것이다. 일단 이것을 확인한다.

저장을 하는 경우에는 이것의 상황이 반대로 되는 것인데, 우리가 옮기는 것은 실제로는 애니메이션의 출력위치가 된다. 따라서, 우리는 이 좌표를 반대로 넣어주면 된다는 것이다.

저장하는 상황에서는 좌표를 반대로 넣어준다. 기억하면 된다.

우리는 애니메이션 툴을 통해서 좌표를 잡아줄 때, static Rect의 출력 위치를 가지고 돌아다니면서 확인을 하였다, 하지만, 우리가 실제로 움직이길 원하는 건 static Rect가 아닌 애니메이션의 위치이다. 그래서 반전이 필요하다. 애니메이션을 올려야 하니까.

ㅇㅋ. 그러면, 오프셋 시스템을 구현하였으니, 이제는 오프셋을 제대로 설정하고, 이 오프셋을 토대로 Save / Load하는 시스템을 만들어보도록 하자.

일단, 기존의 클립 정보를 백업 해둔다. (LoadClip의 정보가 완전히 변하는 거니까)

Load / Save ㅇㅇ부터 기능부터 손을 댄다.

모두 저장하는 기능을 만든다.

오프셋을 일단, 저장하고 하, 이거 Save Option값 때문에 애매하네.. ㅋㅋㅋㅋㅋ

그러면, 간단하게 이렇게 하자. 애니메이션이 기존의 옵션만 들고 있으면 되는 거지… 그치

애니메이션 클립마다 자신의 이전 정보를 가지고 있게 하는 것은 뭐가 아니야 한번만 이 지랄할건데.

모든 애니메이션에 대해서 기존옵션 그대로 저장을 하며, (오프셋의 정보를 가진 체로 저장하도록 한다.)

일단은.. 그렇다. 다 해주자. 모든 애니메이션 정보에 대해서. 이렇게 해서 정보를 모두 잘 저장하면 그 이후부터 이 파일을 읽어오며, 클립 마다 오프셋을 설정하는 것으로 한다.

일단은, 백업을 하나 해두고, 이것을 하나씩 오프셋을 맞춰가면서 저장해본다.

그렇게 모든 파일을 다 저장하고서, (어짜피 오프셋 맞춰야 하잖아.)

이것을 읽어와서 OFFSET포함하여 읽어오는 기능을 구현한다. 솔직히 이거 구현해도 많이 한거야.

일단은 클립에서 하나씩 저장을 하기 이전에 이름을 붙여주는 방식을 없애버린다. (너무 많으니까 귀찮아)

그리고, New 폴더에 저장을 하게 한다.

오프셋은 반대로 지정하게 하여 저장한다.

몇 개를 저장해보고 이것을 확인해보면 터지려나. 아니야 이 부부은 그냥 확인 안 해도 충분히 할 수 있는 부분이지.

오브젝트에서 오프셋의 영향을 받도록 해보자. 적용 완료.

그러면, 모든 정보에 대해서 오프셋을 지정을 받도록 저장을 해보도록 하자. (하나씩 수정하면서 Offset 작업을 해보자는거지)

아니, 이거 왜 넘겨줬지.. 그냥 싱글톤써서 받으면 더 쉬운데… 히익. (바보바보)

하나만 확인해보고 오프셋 작업 들어간다. 잘 되네요.

자 하나씩 맞춰보자 !!

애니메이션은 옵션, coord 이런 정보는 다 뽑아두었으니, offset만 새로 맞춘다 !

오프셋의 상태를 고정시키는 기능을 만들어두자.

5번 offset copy & offset paste 6번

아무튼 오늘 오프셋 다 잡고 간다.

지금 설정하는 것은 Static Size를 기준으로 한 출력 위치이다. (Body의 위치)

이거 가지고 바닥 충돌처리하는 걸 기억해)

왼쪽 애니메이션이랑 오른쪽 애니메이션이랑 다를 수 있지. 이 클립은 반전을 만들기 이전의 클립이고, 따라서 우리는 이것을 오프셋의 개념으로 맞추려는거잖아 오케이 ?

와 진짜 감동이다. 와 이거 좌우 애니메이션 위치가 확 바뀌는걸 한번에 처리를 해버리네, ㅋㅋㅋㅋ

하드 코딩 안 하고 이렇게 오프셋을 맞춰버린다고 ? 믿혔네

아니 이제껏 저장 안 되고 있었구나.. 아니,

내가 원하는 로직 : 1 2 3 4 : saved -> 1 2 3 4

지금, 옵션값이 달라지는 현상이 일어났다. 어디서 틀려먹은 건지 엑엑. 믿혔구나 하하

기존 백업 정보를 이용해서 클립이름 + 옵션 정보만 저장을 하는 기능을 만들어서 이것을 토대로 한번만 노가다 해보자.

하, 좀 맞자 ㅋㅋㅋㅋ

ㅋㅋㅋㅋㅋ

아니 이거 왜 저장 제대로 안 해서 또 힘들게 만들어… 진짜 너무하네 적당히 해야지.. ㅋㅋㅋㅋ

자 멘탈 잡고, 원래 이런 날도 있는 거지 ㅋㅋㅋ

옵션이 준비가 된 클립들이 있어 이 클립들을 저장하면 ,offset 정보 / 옵션 정보까지 다 잘 저장해.

그러니까, 한번만 고생하자. 한번만, 이제 확실히 저장하고, 확실하게 Load하니까… 마지막이야.

두번 다시 하고 싶지 않은 노다가 3번 믿힌 욕나오네 진짜 ㅋㅋ 해냈다.

너무 고통스러웠고 오늘 한거라고는 정말 오프셋 맞춰둔 것 밖에는 없지만, 그래도 잘했어. 포기하지 않고, 세심함이 부족해서 몸이 고생했을 뿐이야. 다음번에는 이런 파일 입출력 과정에서 실수하지 않도록 확실하게 저장하고, 확실하게 Load하는 습관을 들여야지…

아무튼, 오프셋 깔끔하게 잘 구현했고, 이제 피격 상태 알파블렌딩 효과를 적용시켜보자 ! !!!!

단순히 값이 켜진 상태로 몇 초간 지속 이런 식으로 구현이 되어 있는데, 우리는 이게 아니라, 몇 초를 간격으로 몇 번 깜빡일 건지를 정하는 것으로 하자.

1초 3번.

0.5s \* 6

SwordWalk Left

오늘은, 각 상태마다 플레이어의 충돌체를 활성화/ 비활성화 하는 기능을 만들고 이것에 따라서, 오브젝트를 충돌시켜보자. 아니면 넉백 시스템을 구현해둔다.

스워드 커비 점프공격 충돌체 사이즈 200 x 200

대시 공격은 기본 플레이어의 충돌체를 이용하면 된다.

기본 공격의 경우에는 좌우, 첫번째 공격모션에 대해서만 한 번 처리를 해주거나, 말거나 둘 중에 하나인데,

해도 그만, 안 해도 그만.

그러면, 공격 콤보에 대해서는 따로 처리를 하지 않고, 점프 공격에 대한 충돌체만 추가한다.

이런 식으로 점프에 대한 처리를 하는 위치를 바꿔준다면, 모션이 풀리지도 않을 것이고, 추가적으로 점프에 대한 처리도 하지 않을 것이다.

스워드 공격 중에 점프하는 것. 막기

공격 중에 Balloon상태가 되는 것 또한 처리를 미리해서 return 함으로서 처리를 막는다.

스워드 커비

기본공격 : 총알 나가는 것으로 처리

대시공격 : 기본 충돌체를 이용하여 처리

점프공격 : 원 충돌체 활성화를 통한 처리.

스톤 커비

돌로 변하는 공격 : 처리

마이크 커비

마이크의 경우에는 공격을 해서 진동이 울리는 순간, 화면 크기만한 충돌체를 둬서 처리하는 것으로 할까 아니면, 오브젝트를 하나 둬서 이것을 Blend 효과를 주면서 처리하는 것으로 할까

아니면, 플레이어의 위치를 기준으로 일정 거리 안에 들어오는 물체를 없애는 것으로 할까 ?

일단, 충돌체를 넣는 것은 너무 거추장스럽고, 커비의 위치를 기준으로 몇 px의 물체를 차례로 날리는 처리를 하면 좋을 것 같다.

일단, 클래스를 하나 제작하는 것으로 한다.

그래야 제대로 뭔가가 나올 것만 같아.

MikeObject 라는 클래스를 제작하여 해당 클래스가 생성이 되면, Blend효과를 이용하여 화면을 노란 색으로 블렌딩하도록 하고, 커비만 가장 앞으로 나오도록 한다. (커비를 제외한 오브젝트는 이전에 출력하고 그 다음에 MikeObject를 알파블렌딩하여 출력, 깜빡임 효과를 준다.

이 상태로, 진동이 끝나면 오브젝트가 죽도록 한다.(즉, 진동의 남은 시간을 기준으로 하여, DieTime을 설정한다)

스워드 워크 레프트 잔상.

대시 공격 중에 일반 공격가능 ㅋ 수정완료

마이크 오브젝트는 플레이어에 의해서 관리되어, 플레이어를 출력할 시점에서 미리 처리 -> 플레이어의 처리를 진행한다.

플레이어가 가지고서 있다가 필요할 때 사용하는 객체로 두는 것이다.

플레이어가 필요한 순간에 해당 오브젝트를 이용하여 AlphaBlend효과를 주고 해당 효과가 끝나면 자동으로 사라지도록 한다. 그리고 이 친구는 화면 크기의 Rect충돌체를 가지고 있어서 자신과 충돌이 된 물체를 없애도록 해야 한다.

그러면, 우리는 일단, 활성화 비활성화의 개념을 도입하여 물체를 조정해보자. 이 객체는 플레이어가 따로 가지고 다니기 때문에 Enable 변수를 사용하지 않으니 우리가 이용하기 좋다.

카메라를 흔드는 시간을 얻어오는 함수를 만들어보자.

커비를 Default Object에서 맨 위로 올리는 처리를 해주자.

Twizzy가 Line Collision이 되어서 Hit애니메이션이 호출되고 있다.

MikeObject는 자신의 오브젝트를 업데이트 해야 하는데, Size를 화면크기로 지정하고,

카메라의 좌표를 주는 것으로 한다.

일단, 여기까지 잘 되나 해보자. 텍스처는 Start 함수에서 SetSize가 되지 않는다.

즉, 애니메이션에 대해서만 기본값으로 설정이 되는 것이다.

갑작기 애니메이션을 돌리다가, 강하게 출력이 되는 이유 : Blend Time이 끝나면, 자동으로 원래 방식으로 돌아가게 된다.

따라서, 블렌드 효과가 없는 체로 출력이 되는 것.

시간을 충분히 줘서, 카메라의 흔들림이 먼저 끝나게 된다면, 문제가 없을 것 같다.

오케이 처리 완료 이제는, 몬스터 사라지는 처리.

마이크 공격에 대한 처리를 함수를 등록해줬으니 한번 보자.

함수를 호출해서, 모든 오브젝트, 충돌처리를 해야 하는 오브젝트들에 대해서 화면의 화면 Rect와 충돌했는지 확인하고 처리를 한다.

확인해보자.

이제는 충돌처리가 다 되었으니, 넉백 기능을 구현해보자.

너가 생각하는 것이 뭔지는 잘 알 것 같아. 근데, 이 부분은 시간이 나면 해야 하는 부분이다.

지금 하면 좋겠지만, 시간이 나면 하는 것이 더 좋아. 일단 미룬다. (디테일)

# 보스전 (DDD) (리소스만)

일단, 디디디에 해당하는 리소스를 뜯도록 하자.

리소스를 만드는 과정이다. 왼쪽을 기준으로 오프셋까지 다 맞춰두고 이것을 복사한다.

그러면, 옵션 정보는 그대로므로 신경을 써주지 않아도 되고, 오프셋만 한번 보고 저장하면 된다.

애니메이션 정보 출력 위치 이미지의 크기를 기준으로 설정 완료.

리소스 따는 작업만 해둔 상태.

# UI 이미지 따는 작업.

커비의 경우에는 간단하게 on / off 스위치로 된 체력바를 사용하는 것도 좋다.

(다른 사람들 많이 하듯이)

# 스테이지간 연결. (Door 생성)

일단, MainScene에서 구현해둔 것을 많이 활용할 것이다. 적어도 3 스테이지에서 ( 원래 스테이지 구성을 많이 해야 게임이 있어 보이겠지..)

아무튼 StartScene -> Stage1 -> Stage2 -> DDD Scene 정도로 생각해보고 있다.

별개 : MapEditScene / AnimationEditScene

디디디 장면의 경우에는 애니메이션을 계속 출력해야 하는 배경이 있다. 이걸 어떤 식으로 구현해야할까 ?

일단, 뒷 배경에 애니메이션을 까는 개념인데, 애니메이션과 스크롤이 같이 진행되는 것은 존재할 수 없다. 화면이 멈춰있는 상태에서 애니메이션이 돌아가거나, 아니면 텍스처 스크롤이거나, 반드시 둘 중에 하나만 있는 기능이라는 것이다.

따라서, 우리는 애니메이션을 호출하여, 계속 보여줄 수 있는 시스템을 따로 추가하고, (이것을 계속 애니메이션을 돌리면 된다. 스테이지는 ForeGround만 처리하는 것으로 한다.

그러면, 풀 애니메이션과 같은 부분을 비슷하게(물론 따로 객체를 만들어야 하겠지만) 처리할 수 있겠다. 어쨌든 우리는 스테이지에서 백그라운드의 처리를 하지 않는 상황까지 고려해서 구현을 해야 한다

Falling 상태가 되어서 입김을 부는 순간에 공격에 대한 대기 시간을 주자. 그러면, 입김 생성 이후에 바로 공격을 할 수가 없어서 자연스러운 처리가 될 것이다.

현재 처리가 마음에 들기는 하는데, 왜 좌우로 안 움직이는지 확인하고 싶다.

그냥 하자.

그러면 넉백 애니메이션 후에도 제대로 되는지 확인해보도록 하자.

ㅇㅋ 다른 커비로도 확인해보자.

일반 커비의 경우 슬라이드를 제외하고는 일단 처리 완료

스톤 커비.. 날라간 이후에 공격이 막히기는 하는데,

기본 공격 : 돌

점프 공격 : 돌

점프공격 돌 :

Balloon 상태에서 처리가 안 되는 현상.

각 커비의 상태마다 넉업종료 시간을 다르게 줘서 처리해야 하는 상황이 생겼다.

단순히 1초로 해버리면 커비의 상태에 따라서 조금 처리가 이상해진다.

따라서, 기본값을 1초로 두고, 이것을 설정할 수 있도록 한다.

근데 점프 상황인 경우에는 이런 처리도 상관이 없지 않나 ?

점프에 대해서는 잘 처리하니까, 풍선상태만 고려하여 값을 주는 것으로 한다.

문제가 하나 생겼는데, 만약에 충돌이 되서 넉업이 되는 상황에서 왜 충돌이 되었는데 입력이 가능한지.. 아마, 시간이 다 지나서, 넉업, 넉백이 풀리는 상황이 일어난다. 그러면 여기서 스파게티 소스 방식으로 처리해서 넉백 혹은 넉업이 되는 상황에서 적어도 이 시간 동안은 입력을 막아버린다면, 날라가는 처리에 대해서 손 쉽게 입력을 막을 수 있을 것 같다. 근데, 이 처리의 경우에는 Balloon은 예외로 둔다. (이 때는 false로 값을 던져주는 처리도 필요하다)

그냥 충분한 시간을 주고, 바닥과 충돌한 경우에 true 상태로 만들어주는 것도 좋겠다.

풍선 상태의 경우에는 날라가더라도 입력을 받을 수 있도록 예외처리.

하나 더 생각해봐야 하는 부분이. 단순히 KnockUp / KnockBack을 InputDisEnable로 대체할 수 있는지를 확인해봐야 한다.. 그냥 x축에 대해서만 충돌이 일어나는 경우에 바닥과 계속 충돌이므로해당 처리가 바로 풀리게 된다.

결론적으로는 이것도 처리하고 저것도 처리해야 한다.

현재, InputEnable에 대한 처리를 해줘서, 기본적인 입력을 모두 막았고, Balloon상태에서 올라가는 처리는 풀어야 하므로 이 부분만 예외처리를 해두었다.

지금 처리를 모든 넉백의 경우에 그냥 비활성화 처리를 해주고 있는데, balloon상태로 충돌이 되는 경우에는 인풋처리를 풀지 않고, 나머지의 경우에는 모든 처리를 막으면 그만이다.

즉, 점프의 경우에도 예외처리로 다 막아버리고, 우리는 단지 balloon상태를 제외하고는 입력을 꺼버리면 되는 것. 확인하면 된다. 이제

StoneKirby 공격시간이 너무 느리다. Balloon상태에서 다시 공격하는 시간이 너무 느림. 수정

스톤 커비는 처리했다.

스워드 커비의 경우에는 애니메이션에 콤보가 적용되어 있어서 각각 모션이 끝나는 순간에 처리를 AttackEnd 처리를 해줬다.

점프 공격을 하고 있는 경우에 다시 점프 공격하는 것을 막아버린다.

스워드 점프공격중에 방향키를 바닥에서 난사하면 풀리는데, 이것이 flag가 갑자기 바뀌는 순간이 있는 것 같다.

아무튼 여기를 공략한다.

이것의 문제점은 애니메이션이 끝나는 순간에 누르고 있던 방향키가 인식되어 (근데 왜 풀리지 ?)

이해가 안 되는데 ? 뭐지 AttackEnd로 막는 것이 좋을 것 같다. 잘 안 되니까, 그냥 bool 변수 하나만 더 두자. 그러면 될 것 같다. 해당 애니메이션이 호출되고 있는 경우라면 true 아닌 경우에는 false를 넘겨준다. 그러면 확실하게 이 순간을 알 수 있지 않을까 /

그냥 스워드 커비는 해당 부분의 처리를 하지 말자. 이거 그냥 빼면 더 처리가 쉬울 것 같다.

순간적으로 none값이 들어오는 상황에서 처리가 일어나는 듯하다.

점프 공격에 대한 처리를 막았다. SideUp 에서 문제가 생기는 것이 었다.

이제, Sword 커비에 대한 넉백 처리를 확인해보자. 잘 된다. 이제 맞으면 몬스터가 넉백되는 것을 구현해보자. 일단, 모든 커비에 대해서 슬라이드에 대한 처리를 적용한다.

공통 동작

점프, balloon, 슬라이드 막기, 입김 뱉기

커비만의 동작 : 빨아들이기 먹은 것 뱉기

Grizoo

Twizzy

TwizzyFly

Burt

DDD(추가 예정 ㅇㅇ or Not)

몬스터가 날라가는 방향과 반대로 방향을 바꿔줘야 하지 않을까

왼쪽으로 날라간다. 그러면 오른쪽을 보게 해야지.

오른쪽으로 날라간다. 그러면 왼쪽을 보게 해야지.

시간 초기화 하고 맞은 방향과 반대되는 위치로 이동을 하도록 했다.

그리고 뒤 보면서 걸어가는건.. 넉백이 너무 긴거 아니야 ?

맞고 맞은 방향으로 Hit -> 같은 방향으로 Move (멈춰있는 시간 동안 자연스럽게 Hit애니메이션)

몬스터에서 잠깐 동안 Move멈추는 키를 만들어두자. 그리고 이게 false가 되면, 시간 재서 일정시간 이후에 다시 움직이도록 처리한다.

근데, 기존의 처리 방식이나, 단순히 Move가 false이면 시간을 잰다는 식으로 처리를 하면, 기존의 처리 (마이크에 의해서 가만히 있어야만 하는 오브젝트 같은.. 아무튼 이런 처리가 무시되므로

따로 flag 변수와 시간변수를 둬서 처리를 하는 것으로 하자.)

현재 구조상 Move값이 false면 뭘 하든지 false이다. 즉, 아무 처리도 하지 않는다. 따라서 Monster클래스에서 이것을 처리해주면 좋을 것 같다.

이제, 몬스터가 Stop의 처리를 하는 상황에서 따로 움직이지 않게 된다. 우리는 넉백시간으로 이것과 동일하게 해주면 넉백 끝나도 다시 움직이는 처리가 깔끔하게 될 것이다.

해당 시간이 끝나면 다시 원래하던 처리를 이어서 하도록 !

플레이어의 모션에 따라서 좀 처리를 다르게 처리해야 한다.

Down 이면, run 을 푼다.

바뀐 방향에 대해서 애니메이션을 다시 셋팅해야한다.

이제는, 애니메이션 방향에 문제가 생기는 경우는 없을 것이다.

Prevent에 대한 몬스터의 넉백을 처리했으니 (이건 모든 커비에 대해서 공통인 사항이다.)

Slide에 대한 처리를 해보도록 하자. ㅇㅇ

입김 오브젝트에 대한 처리도 해보도록 하자. 몬스터 클래스에서 일괄적으로 처리하면 되는 부분 인정 ?

모든 오브젝트의 넉백에 해당되는 말이다. 몬스터가 공격을 받고서 살아있다면 위로 뛰운다.

근데, 일정한 수치를 주고, 죽이는 DieTime (사형선고)를 주면, 그냥 단순히 넉백인데,

Slide는 그냥 쭉 밀고, 입김 또한 그렇게 밀어준다. 그런데,,,

대상이 점프를 하고 있는 경우에는 위로 밀어주는 처리가 진행되고 있다.

플레이어가 던지는 물체. 별, 입김, 총알 이런 물체이다. 해당 물체들에 공격력을 주는 처리를 나중에 해주겠지만, 무적 상태와 넉백에 대한 처리는 동일하게 처리하면 된다.

입김에 대한 처리 넉백 ㅇㅇ.

별쏘기(는 그냥 죽도록 처리한다) ㅇㅇ , 돌 ㅇㅇ, 총알 ㅇㅇ에 대한 처리를 한다.

// 맞으면 Enable = false로 주고, 날리는 처리.

근데 이 부분은 데미지를 주고 데미지에 따라서 처리해야 하는 부분이다. 나중에 추가.

각 이펙트마다 넉백이 되는 힘을 정해두는 것으로 한다. 기본 값은 0, 복사.

일단, 커비로부터 넉백 or 넉업에 대한 처리는 수동으로 조절하고, 커비가 만들어내는 오브젝트와 충돌이 일어나는 경우에는 파워값을 이용한다.

별을 쏘는 이펙트의 경우에는 하나의 물체와 충돌이 된 이후에 다음 물체와 충돌을 하지 않을지 충돌을 진행할 것인지에 대한 여부가 필요한데, 이건 단순히 flag를 하나 줘서 이 값을 활성화 시켜주는 것으로 하겠다. 그렇다면, 기본적으로는 대부분이 한번의 충돌을 하고 다른 물체와 충돌이 일어나지 않는다. 그래서 기본값을 false로 두고, 계속 충돌이 일어나야 하는 물체. 별쏘기 이것만충돌을 시킨다.

커비가 만드는 물체에 대한 공격은 자신의 방향과 동일하게 해당 방향으로 날라가면 될 것 같다.

오브젝트에서 방향을 구해내는 로직을 만들어야 하는데 현재, 이걸 Effect에서만 사용하니까 이것을 Effect에서만 처리를 해주자.

별이나, 입김 공격에 대해서는 PlayTime을 지정해줘도 그냥, 괜찮은데, Bullet의 경우에는 사라지도록 처리를 하고 싶다. Bullet (Sword)애 대해서만 함수를 지정해주면 될 것 같다.

스타트 함수에서 복사되는 총알에 대한 함수 포인터를 지정해서 아라서 없어지는 처리를 해주고 있다.

스테이지(직선, 타일)에 대한 충돌처리로 맞는 순간 DieTime을 수정해서 처리를 해주고 있다. 이후에 이펙트 생성.

몬스터와 충돌이 일어나는 경우를 정의해야 하는데,

모든 이펙트에 대해서 넉백에 대한 처리는 몬스터 클래스에서 해주고 있으니, 모든 이펙트에 대해서 맞으면 사라지는 처리를 해주고, 이펙트까지 생성해준다. 일단, 없애주는 처리를 하기는 해야 하는데, 이펙트를 생성할지에 대한 여부를 미리 받아두는 것으로 하는 것이 좋겠다.

모든 오브젝트에 대해서 처리하고 기본값은 true

이펙트가 생성이 될지에 대한 여부는 좀 보면서 확인을 해봐야 할 것 같다.

Time을 셋팅하는 순간, flag를 설정을 해주는데, 기본값을 설정하는 처리를 한다. 근데 이게 단순히 기본값으로 처리를 하면, 기존에 해둔 설정이 무너질 가능성이 있기 때문에 이것을 하지 않고 따로 구성한다.

이펙트에 해당하는 오브젝트가, Stage(Line, Tile)에 대해서 충돌이 일어난 경우에는 이펙트를 생성하고, 그게 아니고, 몬스터와 충돌이 일어난 경우에는 이펙트를 생성해주지 않는다.

(몬스터에서 이펙트를 생성해주니까)

이펙트가 생성이 되는 것은 좋은데, 너무 빠르게 사라지는 것이 문제…. DieTime …. 이 너무 작다.

기본적으로 0.5f의 시간을 가지고 DieTime으로 처리하되, Bullet에 대한 처리를 따로 해주는 것으로 한다. ㅇㅇ

다른 오브젝트의 경우에는 스테이지 충돌시 0.5f

몬스터 충돌시 그대로 (끝날 때까지)

스워드 총알이면, 따로 시간을 약하게 주는 것으로 한다.

이런 상황에서 약하게 넉백 효과를 준다.

이펙트를 생성하기 이전에 죽여서 문제가 생긴다. 따라서, 일단 Destroy옵션을 빼버린다.

(애초에 이것을 PlayTime으로 조절하는 상황…)

이제 대시 공격에 맞은 적을 띄워주는 것으로 하자. ㅇㅇ

점프공격 ! 의 경우에는 충돌체 자체가 달라져서 처리를 다른 곳에서 해줘야 한다. ㅇㅇ

넉백은 다 한 것 같으니 무적 시간을 줘서 충돌처리를 하지 않는 것으로 하자.

// 구체적으로는 충돌 시작 시점에서 처리를 해주고 있는데, 다음 프레임에서 해당 오브젝트와 다시 충돌 처음의 상태가 되도록 서로 모르는 상태로 만들어버리는 것이다. 동시에 Blend 시간과 무적시간을 동일시 하여 이 무적 시간에는 충돌 처리를 하지 않도록 한다.

Blend 시간이 끝나면, 충돌하던 물체를 없던 것으로 한다. 그러면 이 시간이 끝나고 기존에 충돌을 계속하던 물체도 처리가 될 것이다.

일단, 문제가 생길 수 있는 영역(따로 처리하는 부분부터 처리를 하고 상황을 보자)

무적 시간이 너무 길어 ! -> Blend시간을 짧게 주면 된다.

커비의 충돌체와 연결이 되어 있는 경우에는 무적 시간을 부여하지만, 커비의 충돌체가 아닌 커비의 총알로 처리를 하는 경우에는 이것이 처리가 안 되므로 우리는 따로 이것을 처리해줘야 한다. 모든 커비의 종류에 대해서 처리가 잘 되고 있다. 오케이

그러면 스워드 커비에 대해서만 콤보 중에 애니메이션이 끊어지는 경우 (공격 먹은 경우에 대한 예외처리<총알 생성 풀어버리고… 애니메이션 끝내는 처리..)를 해준다. ㅇㅇ 처리 완료.

# 커비 변신계기 마련 (몬스터 or 아이템)

나는 몬스터를 선택했다. (그게 더 구성이 알차기 때문에 ㅋㅋ

Rocky : 막 점프를 하는데, 이거 구현하기 귀찮으니까, (힘들고 힘들고 귀찮고 힘든 작업)

그냥 움직이면서 돌아다니는 것만 처리하겠다.

Walk -> 제자리 -> 반복. 다른 상황 : Hit ㅇㅇ

Sleep : 걷기 -> 자기 반복 .. ㅇㅇㅇ

위의 두 몬스터는 그냥 이렇게 구현하고..

Sword : 걸어다니기 -> 멈추기 // 공격 -> 반복 Hit

플레이어가 일정 거리에 있다면, 이것을 인식하여 플레이어를 향해서 공격을 날린다.

(이때 이 공격은 좌우로 할까 아니면 각도로 할 까…) 좌우로 하자.

근데, 공격을 날리는 순간에 플레이어의 위치를 확인하기 때문에 뭔가 동작이 매우 이상하다.

움직이는 경우도 마찬가지 이다. 매 프레임당 이것을 확인하는 것은 아니기 때문에 일단, 거리가 끝나면 확인을 해주고 있다… ㅇㅇ

Mike : 걸어다니다가 세 방향을 향해서 음표를 발사한다.

공격을 하는 시점에서 기본 값을 그냥 자신의 이동방향의 반대로 공격을 하는 것인데,

플레이어가 일정한 거리에 있다면 이를 인식하고 플레이어의 방향으로 (좌, 우) 공격을 한다.

기준 angle을 주면 여기서 += 30도로 공격이 일어난다고 하자.

블레이드 나이트와 다른 점은 공격하는 시점이 반드시 고정이 되어 있다는 것이다. 블레이드 나이트의 경우에는 돌아다니다가 플레이어가 일정 거리에 오면 공격을 시작한다.

아니면 그냥 단순하게 플레이어가 범위에 들어있는 경우와 그렇지 않는 경우로 나눠서, 아니다.

마이크에 플레이어의 위치를 기준으로 하여 공격 모션을 출력해주었다. 여기서, Default클립을 주지 않아서 생기는 문제점 또한 없앴다. 이 로직을 블레이드 나이트에도 그대로 적용하도록 하여,

자연스러운 모션이 나오도록 한다.

블레이드 나이트 어택 모션이 짤려있음 (칼 부분) (짤린 거 아님)

Sleep, Rocky는 stop -> move 반복의 단순한 처리로 해결한다.

몬스터는 대부분 다했는데, 일부 공격에 대한 이펙트 처리가 적용이 안 되어 있다. 잘 된다.

이제, 마이크와 Sword에 대해서 총알을 설정해준다.

총알에 대해서는 채널과 프로파일에 대한 처리를 MonsterAttack으로 하면 되는데 총알과 충돌이 되어서 ot\_player에 해당하는 태그를 가지고 있다면, 그것은 몬스터의 총알이라고 할 수 있다.ㄴ

총알 생성을 보고서, 총알 쏘는 시간도 조절.

24일 작업 : 몬스터 추가 / DDD 패턴 (보스 추가)

몬스터는 일단, 추가 했다. 수치 조정은 나중에 하기로 하고, 능력을 어떤 식으로 표현할 것인지 구상을 해보도록 하자. 해당 몬스터를 빨아들였을 경우에 이 몬스터가 능력을 가진 몬스터인지 아닌지를 알 수 있어야 하고, 능력을 가졌다면 어떤 능력을 가졌는지. 그런 Tag 값이 필요하다.

태그는 추가해주었고, 이제 몬스터 로직이 제대로 돌아가는지 확인을 해보도록 하자.

ㅇㅇ

블레이드 나이트를 먹으면 터지는 현상이 발생. ref값의 문제로 터지는 것으로 생각한다.

생성 (Layer) – 1

타일 충돌용 – 1

커비 벡터 1

커비가 총알을 먹었어. 근데, 이 총알이 사라졌어. 그래서 터진거지.

몬스터 총알과 플레이어는 충돌처리를 진행한다.

여기서 총알을 먹어버린 것이다.

기존에 몬스터 총알이 없었기 때문에 이런 문제가 생겼다. 아무튼 몬스터만 먹도록 구성한다.

플레이어와 몬스터가 충돌할 수 있다.

플레이어와 몬스터의 공격이 충돌할 수 있다.

그런데, 플레이어를 기준으로 몬스터의 공격을 생각하지 않고 캐스팅하는 부분이 있다.

따라서 이 부분을 막아줘야 한다.

정리를 해보자면, 반드시 몬스터만 들어올 것이라고 생각하고 짠 부분을 몬스터의 공격이 들어올 수 있다는 가정을 가져야 한다.

이 부분은 나중에 따로 처리해야 하고 일단 잘 나오나 확인 조정..

애니메이션의 프레임 유지 및, 시간 유지 좌우에 대한 처리 (마이크, 디테일)

몬스터를 추가했으니, 이제 먹은 몬스터의 상태에 따라서, 처리를 해주는 것으로 하자.

(커비의 변신 !)

먹고서 취할 수 있는 행동으로는 뱉는거, 먹는것이 있다.

(원작에서는 먹고 있는 것을 뺏기지 않는다, 단순히 이미 먹은 능력을 빼앗길 뿐.)

뱉거나 ㅇㅇ, 먹는 경우에 normal\_monster로 바꿔준다. (먹는 경우에는 당연히 해당 능력으로 변하는 처리를 포함해야지.)

Down의 상태에서는 일단, 바닥에서 별 문제없이 붙어있는 상태이다. 따라서, 이 상태에서 down을 누른 경우에는 Conversion을 진행해도 좋다.

몬스터가 날라가서 Line과 충돌이 되는 경우에 바로 처리를 안 해버리니.. 문제가 심각하다.

충돌을 하는데 계속 날라가는거 아닌가 ? 혹은 충돌처리를 막았거나.

충돌이 된 경우에 넉업이 켜져있는 상태라면… 충돌처리를 하지 말아야 하지 않을까 ?

더 올라가야 한다는 소리니까. 넉백의 경우에는 꺼버리자.

중력의 처리는 Line 이 충돌되는 순간 받지 않는데, 뭔가 다른 쪽에서 내리고 있는 상황.

Blend에서 Line에 대한 처리를 하지 않고 꺼버려서 문제가 생겼다. ㅇㅇ

몬스터 총알 충돌처리를 해야 한다.

# UI (체력바, 상태정보 출력) ㅇㅇ

어떤 이미지를 따도록 할 것인지 해야 한다.

이미지 + 글자 -> 능력 표시. Or not NOT (단순한 것이 멋이다)

체력 표시 ㄴ: 하트로 표시. (6개, 하트, 빈 하트)

좌 하단에 능력과, 하트 6개를 찍는 것으로 하자.

능력 이미지 + 글자. 이것을 편집을 통해서 가지고 있도록 한다.

능력 이미지와, 글자를 땄다. 우리는 이것을 가지고 해당 이미지를 출력해줄 것인데, 클래스를 구분하여 이름… 클래스 Image 클래스… Heart Class.. 를 만들어서 출력하는 것으로 하자

UI가 플레이어를 알고 있도록 구성하든가, 아니면, 플레이어가 UI를 알고 있도록 구성해야 한다.

나는 UI가 플레이어를 알고 있도록 할 것이다.

AbilityImage의 경우에는 KirbyType의 값을 받아서 이 값을 가지고 출력을 한다.

마이크의 경우에는 X값의 변화가 필요하다. 170 \* 170 간격.

AbilityName의 경우에는 커비타입만 가지고 처리하면 된다. 212 \* 100

Heart는 자신이 출력해야 하는 위치에서 6개의 하트를 그리는데 일정한 간격을 유지하며 그리고

플레이어의 남은 체력을 가지고 처리한다. 100…

UI를 추가했으니 이제 잘 되는지 확인해보도록 하자.

잘 안 되면 오프셋좀 설정해주는 처리도 해야 한다.

참고 : 장면이 바뀌는 순간 계속 남아있어야 하는 정보. UI, Kirby 등의 REF값을 높여주는 처리를 해서 장면을 옮기는 것이다.

타겟을 잡아서 해당 타겟의 정보를 얻어서 출력하는 클래스를 만들고 여기서 세분화시켜서 처리를 하는 것으로 하자.

커비 UI를 적용했는데, 위치를 어디로 해야 마음에 들지는 잘 모르겠다.

아무튼, 어느 위치로 하든 마음에 안 들 것 같다. ㅋㅋㅋㅋㅋ

내일 25일은 .. UI를 적용하고, 데미지 개념을 추가한다. (몬스터와의 전투처리를 완료, 스타튕기는 처리를 통해서 능력을 뱉는 것을 구현한다, 스테이지 연결)

26일은 …. Bullet을 맞는 경우 Blend 상태인데도 체력이 까지는 상황(무적을 먹지 않는건가 ?)

스타튕기는 처리를 통해서 능력을 뱉는 것을 구현한다, 스테이지 연결) 보스 제작..

# 튕기는 별에 대한 처리와 가만히 있는 별. (능력을 가진…)

해당 이펙트는 이펙트 리소스를 가지고 2개를 겹쳐서 사용하는데, 이펙트는 플레이어의 상대적인 위치를 잡아서 따라다니는 기능이 이미 구현이 되어 있다. 따라서, 이것이 Target과 어떤 의미를 가지는지 생각을 해봐야 하는데… 아 혼동했네..

이펙트는 게임에서 사용하는 오브젝트에 해당하고 UI의 경우에는 TargetUI라고 해서 해당 오브젝트로부터 정보를 가져오는 용도로 사용한다. 아무튼 이펙트와 UI는 다른 클래스이다. 아무 문제가 없다.

정리를 하자면, 별 애니메이션을 호출하면서 그 뒤쪽에 BlueBall 이펙트를 달아두는데 한쪽에만 충돌체를 담아두고, 블루볼 이펙트는 별을 따라가게 한다.

생각을 해보자. 커비가 별을 먹고서 뱉는 것은 공격이지. Bullet.. 당연히 Damage처리해야지.

근데, 가만히 있는 파란 별이나, 점프하면서 돌아다니는 파란 별은 Bullet인가 ? 아니..

그냥 돌아다니면서 점프하는 Effect이다. 즉, Bullet이 아닌 다른 처리를 해줘야 한다는 것이다.

여기서 두 클래스는 공통된 처리를 할 것 같다. 예를 들면, MoveTime… 이런 처리를 할 것 같으니 부모 클래스를 둬서 깔끔하게 처리를 하도록 하자.

생각해보니 부모 클래스가 없어도 문제는 없을 것 같다는 생각이 들었다. 아무튼, 원래 미리 부모 클래스를 정의 하는 것이 나중에 필요한 상황이 있을 수 있기에 하는 것이 좋다.

아무튼 따로 하자.

JumpStar의 경우에는 Jump Flag를 가지고 있어서 점프를 하고 다시 바닥에 붙으면 false처리를 해주고 이 상황에서 다시 Jump를 하고, 아무튼 이런 식으로 처리를 하면서 자신의 방향대로 나아가다가 결국에는 터지도록 하면 된다.

MoveStar의 경우에는 일정시간 동안 자신의 방향대로 움직이도록 하고, 그 위치에서 정지한 상태로 있도록 한다. 그리고 이후에 건드리지 않으면 사라진다.

좋은 생각났다. 해당 오브젝트에서 왼쪽으로 움직이는 MoveStar와 오른쪽으로 움직이는 MoveStar를 따로 만들어서 생성하는 코드까지 만들어 두고, 이 상태에서 일정 시간동안 움직이도록 하고, MoveAngle 몇 초가 지난 이후에는 바로 m\_bAngle Flag만 꺼버리면 될 것 같다. ㅇㅇ

Jump Star의 경우에는 한번 점프를 하는 상황에서 위쪽에 타일이 있다면 어떻게 처리가 되어야 할까 ? 중력과 같은 방향의 힘으로 다시 내려와야 하지 않을까 ? 아니면, 점프 자체를 꺼버리고 기존에 중첩 시킨 중력 시간을 가지고 그냥 바닥으로 떨어지게 처리하면 될 것이다. (즉, 그냥 그대로의 처리.) 단지, 위쪽과 부딪힌 경우에도 점프를 꺼버리면 된다는 것. 근데. 이건 플레이어도 마찬가지 … 네..

아니면 파고 들었다고 인식해서 밀어주고 있으니까 그냥 둬도 될 것 같고, 아무튼 점프를 통해서 계속 튀어오르는 처리를 구현해보자.

JumpStar와 MoveStar 클래스를 구현했으니, 이제 BlueBall 이펙트와 연동하여 이것을 따라다니는 것으로 처리를 해보자.

아니다 일단 BlueBallEffect를 띄우는 것으로 하자.

KirbyLayerSort 함수는 “Default” 레이어에서 “Kirby”를 올리는 함수이다

따라서 이 함수를 LayerUp으로 고쳤다면, “Default”와 “Kirby”를 인자로 넣어주면 된다.

이것의 활용도를 넓혀주었으니, 이제 Blue Ball 이펙트를 생성하여 따라가는 처리를 하도록 한다.

일단 생성만 하고 잘 따라가는지 결과를 확인하도록 하자.

오케이 괜찮게 잘 나오고, 이제 이렇게 푸른 빛을 내는 스타를 MoveStar 또는 JumpStar 클래스를 통해서 만들어보도록 하자.

우리에게 필요한 것은 Left/ Right 각각 Move / Jump 해서 4개가 필요하다.

충돌 관계를 하나 더 설정해야 하는데, Monster의 것도 아니고, Stage (?)의 것도 아닌 StarProfile

StarProfile의 경우에는 플레이어와 충돌이 일어나도록 할 것이다.

(이것을 이렇게 따로 둬야 하는 이유는 StageProfile이라고 하면, Stage와 충돌을 시키지 못하기 때문이다.)

아무튼 StarProfile의 경우에는 스테이지와 충돌을 시키고, 플레이어와 충돌을 시켜야 한다.

그것 외에 몬스터나, 플레이어의 공격, 몬스터의 공격과는 전혀 관계가 없다.

별을 기준으로 하여 스테이지와 충돌을 시켰고, 스테이지를 기준으로 스타를 추가.

플레이어를 기준으로 스타를 스타를 기준으로 플레이어를 추가하여

별은 플레이어와 스테이지와 처리가 된다. 그리고, 우리는 이 값을 태그를 이용해서 처리를 하면 되는 것이다.

무브 스타의 경우에는 1초간 움직이고 나머지의 시간은 움직이지 않는다. 즉, 5초에서 1초 동안은 움직이고 나머지 4초는 멈춰있는 처리를 하는 것이다.

왼쪽과 오른쪽을 나눠서 미리 각도까지 설정해서 그 상태로 두는 것으로 하자.

같은 리소스(별이 돌아가는 리소스)를 사용하되 처리는 다르게 하도록 클래스를 따로 만들어서 두는 것이다. 리소스는 같이 사용하고…. (텍스처 이미지)

아무튼, 우리는 여기서 이렇게 처리가 되는 별을 따라가는 블루 스타에 대한 처리를 내부적으로 생성하여 연결하고… 그런 처리를 해주는 것이다.

MoveStar에 대한 초기값은 미리 설정을 해두었으니, Jump Star 에 대한 처리를 해준다.

아마 커비의 중심에서 튕겨져나오는 별을 구현을 해야할 것이다.

아무튼 우리는 초기 위치를 설정해둔 이후에 이것이 바로 위로 뛸 수 있도록 하는 것이 중요하다.

그래서 우리는 초기 파워를 설정해두고 이것을 복사하여 사용할 것이다.

클래스는 모두 추가했으니, 이제 능력을 벗고 던지는 처리를 해보자. (지금 적용할 수 있는 처리)

드디어 Throw 처리를 하는 것이다.

Kirby Conversion키에서 사용하던 조건을 그대로 사용하면 어떻게든 될 것이다.

현재 점프가 꺼지는 상황을 구현하지 않은 상태이다. 따라서, 점프가 꺼지는 상황을 일단은 만들어 두려고 한다. LineCollision과 TileCollision(바닥)에 해당하는 상황에서 우리는 Jump를 false로 만들어 주어야 한다.

프로파일과 채널을 통한 충돌 처리가 갑자기 추가가 되었으므로 이것에 대한 예외 처리를 꾸준히 해줘야 할 것이다. (기본적인 틀은 잡혀있으니 우리는 objectType에 따른 처리만 해주면 되지 않을까 ? )

근데, 이것이 점프 스타인지 아니면 MoveStar인지 구분을 못하네 ? 그냥 뭔 짓을 하든 flag로 처리를 할 수 밖에 없는 상황이다. 둘 중 하나 택일. ENUM choice

MoveStar가 Line과 충돌이 되는 경우가 있는지는 잘 모르겠는데, 만약에 있다면, 뭔 처리를 해줘야 하나 ?

그냥 이 친구는 MoveTime까지는 계속 움직이는 처리를 해줄 것이고, Line과 충돌을 하면 올라가면 그만 아니야 ? 그렇지 … (아니면 그냥 멈춰있게 하던가)

멈추게 해주자.

점프 스타의 경우에는 Line위에서 Jump가 계속 일어나야 하는데, 충돌이랍시고, 지속적으로 Line에서 땡겨버리면 곤란하다. 따라서 플레이어 잡아 땡기고, 벗어나도록 해주는 예외 처리를 똑같이 적용해서, 한 절반 정도 힘을 기준으로 하면, 아무 문제 없이 점프를 통해서 올라가는 상황과 그렇지 않은 상황을 구분할 수 있을 것이다.

이렇게 상황에 따라서, Line에 대한 위치 선정을 할지를 결정한다면 …

점프를 통해서 위로 올라가는 상황에서는 잡아당기지 않을 것이고, 내려오는 상황에서만 Line에 대한 처리를 해줄 것이다. 따라서, 같은 방향을 움직이며 점프 / 점프 / 점프가 반복이 되는데 아무튼, 이런 상황을 처리하고 있다.

JumpDisEnable 에 대한 처리를 해줘야 한다.

점프를 하고 떨어지는 상태에서만 Line에 대한 충돌처리를 해주고, 여기서 점프 상태를 풀어줌으로써 다시 점프. 계속 충돌이 들어오는 상황일 것인데, 점프 다운 상태는 아니기 때문에 따로 처리를 하지 않고 계속 올라간다.

그리고, Tile (바닥)에 대한 DisEnable 처리.

MoveStar와 Tile의 충돌… 그냥 바닥으로 박히면 막아주고, 들어가면 밀어주고 처리를 할 필요가 없다. (어짜피 미니까…)

JumpStar의 경우에는 바닥과 충돌이 이루어진 경우에 Jump = false로 주고, 위치를 선정한 다음에

다시 위로 뛰도록 해주어야 할 것이다.

클래스도 추가했고, 뱉어주는 처리도 했는데, 이것에 대한 처리… 타일, LineCollision에 대한 처리가 제대로 이루어 지는지 확인해보도록 하자.

MoveStar 중력 false ㅇㅇ, Tile True ㅇㅇ Line true ㅇㅇ

JumpStar 중력 true ㅇㅇ Tile true ㅇㅇ Line True ㅇㅇ

//

전투가 일어나는 상황에서 블렌드 상태에서 공격을 맞고 쓰러지는 상황을 처리한다.

플레이어가 공격을 맞는 경우 ㅇㅇ/ 몬스터가 공격을 맞는 경우. ㅇㅇ

몬스터가 공격을 맞는 경우에는 몬스터에게 넉백처리를 해주면서 데미지를 넘겨주고 있다. 근데 이 상황에서 만약 몬스터가 무적상태라면 이미 조건문에서 걸려서 처리가 되지 않는다.

확인을 해봤는데, 실제로 무적 상태에서는 공격처리가 되지 않으며 무적 상태가 풀린 경우에만 처리를 해준다.

이걸 같이 Blend 처리를 해주면 좋을 것 같고(둘다), 또한, MoveStar 는 아니지만, JumpStar의 경우에는 방향이 바뀌는 처리도 해줘야 할 것 같다.

점프를 한 횟수를 기준으로 하여 떨어진 상황에서 하나씩 카운트를 세준다.

처음에 올라가는 상황을 1이라고 인식하고 ㅇㅋ 잘 했다.

이제, 이것을 먹는 처리와, 몬스터에 대한 충돌은 안 하지.. 아무튼 Monster임을 확신하고 짠 부분에 대한 예외 처리를 해줘야 한다.

일단, 스타와 충돌이 되는 물체에 대해서 생각해보도록 하자. Stage(Tile, Line) ㅇㅇ / Player

플레이어의 충돌체 함수와 자신에게 걸린 충돌체 함수. ㅇㅇ

플레이어의 충돌체에 대한 처리를 해주었고, 빨아들이는 충돌체에서만 처리를 하도록 만들었다. 다른 경우에는 retrurn을 호출하기 때문에… 그리고, JumpStar에 다양한 변수타입을 추가하여, 이것이 가지고 있는 능력에 따라 처리를 하도록 만들었다. (MonsterType과 동일한 처리이므로 같은 flag를 사용했다.) 아무튼 플레이어의 충돌체에 대해서는 모두 처리가 끝났다. 자신의 호출함수는 없으므로 생각하지 않아도 좋다.

능력을 뱉고 자신의 능력은 사라져야 한다. 자신의 능력을 별한테 넘겨주는 처리를 해주고 있다.

자 그러면 테스트.

충돌이 잘 막히고 있는지 ㅇㅇ , 이것과는 충돌이 되는지 안 되는지 ㅇㅇ 먹는 처리는 잘 되는지 안 되는지. ㅇㅇ

몬스터가 있어야 뭐든 테스트를 해볼 수 있을 것 같다고 판단된다.

슬립이 끝난 이후에는 체력이 꽉 채워져야 한다. ㅇㅇ

남은 작업 총정리

# 스타 튕기는 처리 ㅇㅇ , 먹는 처리 ㅇㅇ , 뺏기는 처리 dd

뺏기는 처리의 경우에는 해당 능력을 먹은 상태에서 2번의 공격을 맞았는지 확인하고 2번의 공격을 맞은 경우에 상태를 별이 자동으로 나오도록 처리를 하는 것으로 하겠다.

그리고 능력이 바뀐 경우에 이 처리는 초기화를 해줘야 한다.

죽은 경우에 초기화. ㅇㅇ

능력이 변한 경우에 초기화. ㅇㅇ

맞아서 뺏긴 상태에서는 초기화. ㅇㅇ

일반 커비에서는 상태 변화 없음. ㅇㅇ

일부러 상태를 푸는 경우에는 능력이 있는 경우 ㅇㅇ , 그것을 풀 상황이 되는 경우 ㅇㅇ.

공격을 맞아서 상태가 풀리는 경우에는 능력이 있는 경우 ㅇㅇ , 그것이 풀릴 상황이 되는 경우(2번 타격) ㅇㅇ

적용 OK. 일단, 플레이어가 맞는 경우에 해당 함수를 호출해준다. 그리고 같은 능력을 가진 체로 3번의 공격을 맞은 경우에 능력이 풀리는 것으로 했다. 문제는 아마 맞고 있는 상태로 애니메이션이 호출될 것이라는 건데, 여기서 Idle 모션으로 셋팅하는 점은 예외 처리를 해주도록 하자.

일단, 상태를 보자.

먹고 있는 경우에 공격을 맞으면 먹고 있던 것이 풀리는 것으로 하겠다. 모션이 매칭이 안 되서 너무 부자연스럽다. ..

지금 총알 맞고 날라가는 처리가 없네 믿힌. 공략 공략.

몬스터와 충돌이 일어나면, 넉업 / 넉백 효과가 생기고 있다. 몬스터가 날린 총알에 대해서는 어떤 식으로 처리할 것인지 생각해보자. 일단, 넉백 효과는 주는 것이 맞을 것 같고, 점프 상태에 대한 넉업 효과는.. 이것도 주는 것이 맞을 것 같은데…

그러면 해당 처리를 분리하여 처리를 하는 것으로 하자. 처리하는 부분을 함수로 나눠서 처리를

간편하게 했다. 몬스터의 공격을 맞은 경우에도 몬스터와 충돌이 일어났을 경우와 똑같이 처리를 해서 이제 잘 되는지 확인해보자. ㅇㅇ

맞으면 능력이 사라지는 처리는 Damage 함수에서 해주고 있으니 문제는 없다. ㅇㅇ

맞는 순간 빨아들이는 모션이 꺼지지 않는 것을 보니.. 따로 처리를 해줘야 할 것 같다.

누른 것이 꺼지는 상황을 추가해야 한다는 것이다. ㅇㅇ

스워드 커비의 경우에는 일반 공격을 하던 중에 충돌이 일어나면 바로 상태가 풀려버리고, 점프 공격, 대시 공격에는 무적 상태이기 때문에 문제 없다. ㅇㅇ

Sway Time에서는 생성하지 않도록 하자. (스포너) ㅇㅇ

# 시간 동기화 (프레임 제한기법) 어쩐지 잘 안 되더라 ㅡㅡ

(구현 하긴 했는데 안 쓸 것 같다..)

# 넉백, 넉업 효과 적용 ㅇㅇ (무적 처리까지 끝났다)

넉백의 효과의 경우에는 파워 값을 사용하여 처음에는 큰 값으로 움직이고 점점 더 값이 떨어지는 처리를 한다. 이것은 점프에서도 했던 처리로 거의 뭐 복붙하면 된다.

if (m\_bJump)

{

// 점프중에 A 공격키를 누르고 있지 않다면 점프의 처리를 한다.

m\_fJumpTime += fTime;

m\_fPowerTime += fTime;

if (m\_fPowerTime >= 0.1f)

{

m\_fPowerTime -= 0.1f;

m\_fPower -= 200.f;

}

Move(Position(0.f, -1.f), m\_fPower, fTime);

우리가 필요한 정보는 PlayTime을 처리할 변수, 시간 값을 조정할 변수. powerXTime powerYTime

Power값.

시간 기반 처리의 절대적인 문제점.

프레임이 저하되는 순간 게임이 터진다.

따라서, 차라리 제한되는 픽셀을 잡아두고 이것을 몇 초를 가지고 처리할 것인지 받아서 처리하는 것이 낫지 않을까 ?

예를 들어서 1초에 500px을 올린다고 한다면

기본 커비 넉백은 적용했으니, 다른 커비에도 적용하고 충돌 시스템완전히 구현해주자.,

일단 체력의 개념을 도입해서, 몬스터의 체력이 다된 경우에 DieTime을 셋팅해서 몇 초간 Hit애니메이션을 호출해주고, 없애주는 처리를 하는 것으로 했다. 더불어 움직이지 못하도록 구성했다.

플레이어의 경우에는 죽는 처리를 하지 않도록 하자. 그냥(죽지 않도록…)

이펙트에는 데미지의 개념을 도입하여, 기본 값은 0으로 하고, 날리는 물체를 기준으로 데미지를 주는 것으로 하자. 그것 외에 직접 부딪히는 처리에 대해서는 하나하나 처리를 하면 되지 않을까 ?

플레이어가 때리는 공격은 던지는 물체에 따라서 데미지가 달라지는 부분인데, 몬스터의 경우에는 반드시 1의 데미지를 입히도록 설정할 것이다. 그래서 최대 체력은 6이고, 플레이어의 공격은 각각 다르게 처리한다.

아무튼 이펙트에 일부 수치를 주도록 하고, 충돌에 의한 데미지는 따로.

결론 : 이펙트에 데미지의 개념을 도입할 것이다. (이펙트 또한 단순히 데미지를 입히는 경우가 있지 않은가 ? 플레이어의 총알, 몬스터의 총알을 제외하고는 바로 없어지는 처리는 하고 있지 않지만, 일단 맞추는 기능이 있는 Effect 클래스에 데미지의 개념을 추가해야 한다)

총알 vs 총알의 충돌이 있는가 ? 총알에 체력의 개념이 필요한지 묻는 것이다. 각 총알끼리의 충돌은 없으며, 하나의 오브젝트를 맞추고 사라질지의 여부는 총알마다 다른 부분이다. 아무튼 총알끼리의 처리는 없다.

Effect에 데미지의 개념을 추가했으니, 이제 이 데미지를 가지고 처리를 해보도록 하자.

몬스터는 이미 처리를 한 상태이고, 플레이어의 처리. 죽지 않도록 만드는 처리를 한다.

플레이어가 죽은 경우에는 체력을 복구 시키고, 무적으로 5초를 준다.

자 이제, 공격을 하거나, 공격을 먹는 것에 대한 처리를 해보도록 하자.

참고로 원작에서는 부딪혀서 공격을 입히는 방식이 있는데, (무적 상태에서)

내 게임에서는 무적 상태로는 부딪히지 않도록 한다.

체력을 설정해야 하는 것들.

Effect는 공격용이므로 필요가 없다.

커비 : 6dd

Burt : 2 dd

Grizoo : 5d

Twzzy : 1 dd

Blade : 3 d

Mike : 3 d

Rocky : 4d

Sleep : 3d

데미지를 설정해야 하는 것들. (커비)

보통 커비 : 빨아들이기(보스를 제외하고는 한대 컷, 설정 필요없음),

별 뱉기 : 보스를 제외한 오브젝트의 경우에는 한방. 보스의 경우에는 치명적인 데미지 정도.

몬스터 체력의 최대값이 5로 설정. ㅇㅇ ㅋㅋ

공통의 기술 : Slide(2) ㅇㅇ ㅋㅋ, 입김(2) ㅇㅇㅋㅋ , Prevent (1) ㅇㅇ ㅋㅋ

스워드 커비 : 일반 공격 (각 총알에 대한 처리)

: 한발. 3 (Ani) ㅇㅇㅋ

: 여러발 1 ㅇㅇㅋ

, 점프 공격2 ㅇㅋ , 대시 공격 3 (원작에서는 맞은 시간을 기반으로 처리하는 것 같다) ㅇㅋ

스톤 커비 : 돌. (3) ㅇㅋ

막기 상태이거나, 돌 상태인 경우에 몬스터 총알에 대한 처리를 하면 이상하다.

몬스터 기준의 데미지 설정은….

반드시 1로 고정하면 된다. ㅇㅇㅋ

밀치는 데미지는 1로 고정. (내부적 처리)

뭔가를 던져대는 경우. 해당 데미지를 1로 고정. ㅇ ㅇㅋ

(Blade Knight, Mike) ㅇㅇ

데미지 설정 및, 체력 설정은 ㅇㅇ 로하고 적용은 ㅋㅋ로 한다.

이제 밀치는 데미지를 처리할 것인데, 커비가 밀치는 경우가 있는지 생각해보자. (공격동작을 제외하고는 밀치는 처리가 없다) 그리고, 공격 동작에 대해서는 이미 처리를 했다. (오브젝트로 날리는 처리)

몬스터에 대한 데미지 설정을 해야 한다.

몬스터의 총알 설정. ㅇㅇ ㅋㅋ

몬스터의 밀치기 설정 ㅇㅇㅋㅋ

지금 몬스터의 총알에 대한 플레이어 충돌이 구현이 안 되어 있는 것 같다.

총알은 몬스터와, 스테이지, 플레이어에게 충돌이 되는데,

스테이지의 경우 잘 처리하고 있고, 몬스터와 플레이어로 구분한다.

자신이 플레이어의 총알이라면 몬스터와 충돌을 할 것이고,

몬스터의 총알이라면 플레이어와 충돌이 일어날 것이다. (프로파일과 채널의 구성상 그렇다)

따라서 오브젝트 타입을 검사하는 경우에 맞은 상대가 플레이어라면, this는 몬스터의 총알이라는 것이다.

몬스터가 맞는 처리를 하는 것은 몬스터에서 처리하고 있다. 플레이어는 Bullet의 공격을 받는 경우 어디서 처리를 해줄까. (플레이어에서)

즉, 총알에서는 자신이 없어지는 처리를 하고 플레이어의 호출함수에서는 데미지 처리와, Blend. ㅇㅇㅋ

일단, Blend 상태에서의 몬스터 총알 충돌 (플레이어) ㅇㅇ

Blend 상태에서의 플레이어 총알 충돌(몬스터) ㅇㅇ

제대로 막히고 있는지 확인하고, 플레이어의 경우에는 세분화해서 Prevent상태인지 확인하여 처리한다. ㅇㅇ 스워드 점프공격, 대시 공격 상태에서도 무시.

스워드커비로, 점프 공격 or 대시 공격을 하고 있는 상황에서 몸통과 총알이 부딪히는 경우.

해당 총알을 바로 없앤다. 당연히 충돌처리는 하지 않도록 한다.

그리고, 대시 공격의 경우에는 CollisionStart 함수를 바로 사용해서, 이런 식으로 충돌이 된 경우에 없애 버리는 처리를 바로 해야 하지만, 점프 공격의 경우에는 그 이전에 원 충돌체에서 먼저 충돌이 되는 경우가 생길 가능성이 충분하다. 이것에 대해서도 없애주는 처리를 진행해야 한다.

했다. 그러면, 대시 공격이거나, 점프 공격인 상황에서 몸통에 총알이 맞더라도 무시하며,

맞는 순간 총알을 없애주는 처리를 해준다.

대시 공격의 경우에는 BodyCollisionStart 함수를 통해서 해당 총알을 없애주고 있으며,

점프 공격의 경우 BodyCollision 함수를 통해서도 없애주고 있지만, 원에 대한 충돌체에 대해서도 없애주고 있다.

대시 공격이나, 점프 공격을 하고 있는 상황에서 몬스터의 총알과 충돌이 일어난다면, 몬스터의 총알을 없애는 처리를 해주고 해당 부분에 이펙트를 생성해준다.

즉, 점프 공격에 의해서 사라지는 몬스터 총알에 대한 이펙트 팡팡

대시 공격으로 돌진하면서 나오는 이펙트 팡팡.

점프 공격 중에 자신의 충돌체와 부딪히는 경우에도 팡팡.

충돌에 관한 처리는 다 했고, 이제 확인하고 UI를 적용해봐야 할 것 같다.

ㅇㅇ 잘 된다.

이제 마이크에 대한 처리.

마이크 공격 동작시 카메라와 충돌하는 오브젝트가 있다면, 해당 오브젝트의 gravity를 false로

죽는 시간을 흔드는 시간으로 주며, 다 흔들린 다음에는 죽도록 한다.

마이크 공격에 총알이 맞는 경우에 DieTime을 설정해서 죽도록 해준다. 이펙트 생성까지.

Speed 를 0.f로 줘서 못 움직이도록 처리한다.

마이크 동작 시에 충돌이 일어날 수가 있는가 ? 없다. 이전에 충돌이 일어났으면 일어난 거지

공격 시작과 동시에 충돌은 일어나지 않는다. (속도가 0이니까, 플레이어도 제자리고)

처리 하지 않아도 되는 부분. ㅇㅇ

Sleep 커비에 대해서는 별로 처리를 안 해줬는데, 일단, 상태를 나눠서 처리를 해보자.

Sleep 커비의 상태로 몬스터와 충돌이 일어난 경우. (플레이어가 공격을 한 것으로 처리한다) ㅇㅇ

몬스터의 총알과 충돌이 일어난 경우 없애는 처리를 한다. ㅇㅇ

데미지는 2로 처리.

//

새로운 문제가 2가지 생겼는데…일단, 아까 마이크 공격 처리를 하면서, 프레임 once loop로 처리를 해주다보니, 몬스터가 공격을 맞고서, return이 안 되고 있다.

Mike와 Rocky의 경우에는 Stop이 켜진 상태면 잠깐의 시간을 센 이후에 바로 애니메이션을 바꿔주는 처리를 하면 될 것 같다. 그러면 아무 문제없이 처리가 되겠지.

(근데 이 부분은 어느 쪽을 택해도 예외처리가 필요한 부분. 아예 애니메이션 클립에 정보를 추가해놓지 않는 이상은 …)

안 그래도 이 부분에서 파생된 문제가 조금 있었다. Rocky가 공격 맞은 이후에 Stop 애니메이션 상태로 돌아다니는….

일단 지금은 마이크와 Rocky에 대해서만 처리를 하는데, 따로 보니까 다른 친구들 또한 이 처리가 필요할 것 같다. 모션과 처리가 안 맞는다는 생각이 든다면 따로 예외처리를 해주도록 하자.

아마 Mike와 Rocky만 할 것 같다. 왜냐면, 다른 애들 같은 경우에는 once return의 옵션을 가지고 있어서, 기본으로 Idle로 이동을 하는데, 이 친구들의 경우에는 옵션 자체가 loop이기 때문에 이러 현상이 나왔기 때문이다. (즉, 다른 애들이 모션이 이상할 확률은 낮다.)

생각을 해봤는데, 그냥 맞고 나서, 상태를 풀어주는 것이 어떨까 ? 초기의 상태로 돌려버리는 것이다. 이 처리를 한다면, 모든 몬스터에 대한 처리를 한번에 할 수 있고, 애니메이션이 맞지 않아서 이상하게 보일 이유도 없다.

아무튼 Stop 몬스터 클래스를 공략하는 것으로 하자. (Mike와 Rocky에 대한 처리를 없애두었다.)

(일단, 이것만 처리하면 된다. 주위 보지 말고)

몬스터에 대해서 Stop 을 시작한 이후로 0.5s가 지나면, 초기상태로 돌려서 애니메이션을 기본 상태로 두도록 했다. 따라서 Mike나, Stone이 애니메이션 Loop가 되는 일은 없을 것이고, 더 나아가서 몬스터의 흐름이 깨지는 경우도 없을 것이다. 다시 초기 상태로 만들어버리니까… ㅇㅇ

//

슬라이드를 하고 있는 경우에 무적 처리를 해줘야 하는거 아닌가 ? 그뤄치..

대쉬 공격과 마찬가지로 몸통에 대한 처리로 공격을 해주고 있다. 따라서, 우리는 AttackLeft, Right / JumpAttack Collider 같은 것과는 전혀 상관이 없다. 단지 Sword Dash Attack에서 해주던 처리를 동일하게 해주면 되는 것이다.

이렇게 처리를 했는데, Slide || Prevent …

그러면 Dash 공격을 하는 경우(Slide On)으로 처리하는 것이고,

모든 커비에 대한 Slide에 대한 처리도 동일하게 진행되는 것이다.

점프 공격과 같은 기존의 처리는 동일하니 문제도 없다.

그냥 단순하게 DashAttack 을 Slide랑 묶어서 같이 처리해버린 것.

아무튼 스워드 커비의 DashAttack, JumpAttack 모든 커비의 Slide 부분이 처리가 될 것이다. ㅇㅇ

//

다른 문제. 플레이어의 Attack 충돌체(흡입용)과 몬스터 총알의 충돌이 된다.

즉, 여기에 충돌이 일어나서 멈추는 것이 문제다. Bullet의 처리를 막아주어야 한다. ㅇㅇ

커비의 충돌체와 몬스터 총알을 처리하면 안 되는 경우.

일반 커비의 흡입 충돌체와는 충돌을 처리하면 안 된다.

흡입이 일어나는 경우에 활성화하는 충돌체에 대한 flag를 받아 두고 이것을 토대로 현재 흡입중인지 파악한다. Bullet에서 이것을 파악하여 충돌처리를(DieTime) 할지에 대한 여부를 검사하여 처리한다. 생각해보니 이렇게 처리를 하면 안 될 것 같다.

예외 처리의 순서. 일단, Bulllet에 대한 AttackLeft Right 의 충돌체와 절대 충돌처리를 하지 않도록 구성했고, 나머지 충돌체에 대해서만 상대의 무적 여부를 검사하는 것으로 했다. ㅇㅇ 해결.

// Sword Kirby Jump Attack 하는 중에 Down 처리로 풀리는 상황이 있다. ㅇㅇ

프레임이 말도 안 되게 드랍이 되는 현상이 생겼는데, 아마도, 델타타임을 중첩시키고 이것을 제한하는 방식이 아니라, 그냥 프레임이 잘 뜨는 상황에서도 드랍이 나오는 현상이 생겨서 없애주었다. TileRender를 할 때 프레임 시간을 멈추는 처리를 했다. ㅇㅇ

시간을 멈췄는데 좌우로 이동하는 현상이 발생. 근데, 시간이 멈추면 아예 업데이트 같은 처리도 안 해야 하는 것이 맞지 않나 ? 아예 키입력을 받지 않도록 ?

문제는 Input자체에서 함수가 호출이 되는 것이 문제다. 내부적인 처리를 안 한다고 생각할 수 있지만, 함수 호출로서 어느 정도 처리가 되고 있다.

AxisKey / ActionKey를 Pause상태에서 입력을 받지 않도록 처리하면 될 것 같다.

BindAxisKey 혹은 BindActionKey는 키에 직업 함수가 등록된 형태인데, Pause 상태의 경우에는 해당 처리를 하지 않도록 했다.

일단, StartMode 에서는 정지가 없고, MapEditMode에서는 Pause를 조절할 수 있고, 여기서 키와 등록된 함수를 사용할 수 없지만, 아무튼 정지 상태로 키입력을 막는 것도 나쁘지는 않다. ㅇㅇ

Animation Edit Mode의 경우에는 이미 마우스가 떠나면 처리를 하지 않는다. 애초에 Pause의 영향을 받는 Bind Key를 등록하지도 않은 상태이다. 처리를 막을 만한 것이 없는 단순한 입력으로 이루어져 있어서 변함이 없다고 볼 수 있다. ㅇㅇ

이제 Play Mode를 확인해보면 되는데, 프레임이 갑자기 들어오는 경우에 대한 처리도 막아주어야 할 것 같다. TileRender를 키는 순간 pause가 켜져서 시간에 대한 처리를 하지 않으며, 등록된 함수 역시 호출하지 않는다. 그러면, pause가 true인 상태에서는 Render만 해주고 있으며(Tile 때문에) Update 처리를 막고 함수도 막는다 ㅇㅇ 좋다.

이제 우리는 갑자기 빠지는 처리를 막기만 하면 될 것 같다.

타일에서 갑자기 빠지는 처리가 되는 것은 프레임의 비약적인 상승 때문에 모아두었던 시간에 대한 처리가 한번에 이루어지기 때문이다. 따라서 우리는 Pause상태가 풀린 경우에는 Stop Time을 설정해서 잠시 동안 멈춰주는 처리를 통해서 정상화를 준비하는 것이 어떨까 ?

빠진다. 일단, 문제는 기존에 프레임이 딸려서 하지 못했던 처리가 갑자기 이루어져서 생기는 문제라고 판단된다. 현재 Default Layer에 있는 중력의 처리를 받는 물체가 이미 바닥 혹은 Line과 충돌이 된 상태에 (빠질 가능성이 있는 상태) 있다면, 지금까지 모아두었던 중력시간을 0.f로 초기화를 한다.

충돌이 지속적으로 난 상태인지 어떻게 알 수 있는가 ? 그냥 퍼즈 상태에서 충돌 처리까지 해주고 있다. 따라서, 우리는 지속적으로 밀어주는 처리가 진행이 되고 있어서 빠지는 문제는 생기지 않을 것이다.

충돌이 되었는지 여부만 따지만 따지면 되는 것 아닌가 ?

만약에 Line OR Tile과 충돌을 안 하는 물체라면, 당연히 충돌 처리도 되어있지 않은 상태고, 중력으로 떨어지는 처리를 하는 상황도 아닐 것이며

Render와 동시에 Collision도 진행하면서 Tile의 밀어주는 처리를 하는 것으로 하자.

Render가 꺼진 시점에서 (프레임이 비약적으로 상승하는 순간)에 중력 시간을 초기화해준다.

빠진다. 충돌이 난 시점에서 true 인 상태로, Render에서 초기화. False인 상태로 함수를 들어와서 항상 처리하지 않을 것 같은데.

LineCollision || StageCollision( Tile ) 에 대한 충돌이 일어났던 경우에는 GravityTime을 0.f로 줘서 f1을 통한 모드 변경에서 갑자기 떨어지지 않도록 처리를 해주었다.

이것이 왜 잘 될까 생각을 해보면, 일단, LineCollision과 StageCollision을 false로 만드는 순간을 생각해보면 되는데, 충돌 직전에 false를 주고, 충돌이라면 true를 주는 방식이다. 이것이 아마 Collision에서 일어났나 ? 확인해본 결과 LateUpdate에서 (이것을 사용하지 않을 무렵)(주로 Update 에서 사용을 하니까) 이것을 false로 주고 있다.

그렇다는 소리는, 우리가 f1키를 눌러서 시간을 멈춘 시점에서 일단 충돌함수는 계속 돌아가고 있는 상태로 만약에 충돌이 된 상태라면 true값을 넘겨줄 것이다. 근데, 이것에 대한 덮어씌우는 처리가 없어서 그대로 충돌인 상태와 그렇지 않은 상태를 명확하게 구분할 수 있게 된다. (다시 덮어 씌우질 않으니까. )

아무튼, Tile혹은 Line과 충돌이 된 상황이라면 true값이 처리가 된 상태로 유지가 되고(LateUpdate를 진행하지 않으니까. 충돌이 되지 않은 상태라면 false인 상태가 유지가 되는 것이다.)

이 개념에는 충돌 활성화의 개념도 포함이 되고 있으니, 단순히 충돌을 했는지에 대한 여부만 판단하면 되는 것이다. 그래서 충돌이 된 오브젝트라면 중력시간을 0.f로 만들어버리고, 충돌이 아닌 경우라면 그냥 냅두는 처리를 통해서 Render를 재시동하는 순간 프레임의 상승으로 인한 처리를 막는 것이다.

몇 번의 검증을 통해서 확인해보는 시간을 갖도록 하자. ㅇㅇ 해결 .

# 보스전 (DDD) (리소스만)

보스 패턴 분석 및, 공격 리소스 추출.

DDD의 모션으로는 Idle / Walk / Jump(Attack) / Balloon Attack / 달려가서 내리찍기.

// Hit 모션 / 흡입 모션 (이때 별을 뱉도록 해도 좋지 않나 ? ㅋㅋㅋ)

일단,

Idle ->JumpAttack 2번 구현을 해보자.

현재 보스의 점프 상태 구현을 false처리를 해두지 않은 상태라서 점프를 제대로 안 한다.

점프 어택을 2번 실행한 이후에는 MoveStar를 생성하고 다시 Idle 상태로 돌아간다.

5초간 Idle모션.

일단, 이것을 적용해두었으니 확인을 해보도록 하자. 0.5f , 0.65f 로 설정을 해서, 잘 맞게 된 것 같다.

점프를 한 이후에 몇 초간의 간격을 줘서 다시 점프를 바로 점프를 할 수 없도록 처리를 하자.

‘

아니 상황 똑바로 나눠서 처리해..

현재, 점프를 하고서 바닥에 떨어지면 점프를 false로 주고 다시 점프를 하도록 처리를 하고 있다.

여기서 생기는 문제점은 점프가 끝난 이후에 바로 점프를 줘서 애니메이션이 맞지를 않는다는 것이다.

그렇다면, 점프를 한 이후로 점프 시간을 줘서 처리를 하려고 한다.

어떤 식으로 구현해야 할까 ?

일단 점프를 하고, 바닥에 닿는다. 그러면, Jump = false JumpTimeEnable = false인 상태이다.

그러면 단 한 번만, 이것에 대한 처리를 할텐데, 점프시간을 주는 것이다. 그리고 다시 enable 을 false로 둔다. (처리하지 않도록) count = 1

그리고 일정 시간이 지나면 점프를 한다.

다시 떨어진다 count = 2 그러면 이제 상태가 바뀌는 것이다.

근데 기존의 애니메이션과 동일한 처리가 이루어져서 애니메이션의 시간에 접근하여 초기화를 할 수 있는 그런 함수가 필요하다고 판단이 되는 바다.

나는 구현한 적이 없는데, count가 한번 더 세어지는 현상이 발생한다. 이 문제는 JumpDisEnable의 난무로 처리가 되는 것인데, 점프를 한번만 false로 줘야 1번의 점프로 패턴이 끝나는 현상이 나오지 않을 것이다. 즉, 닫는 한번만 false. 점프하는 순간 true.

점프 하는 순간에 충돌이 나서 false처리를 하는 경우가 생기는데, 이것은 Line에서 사용하던 방식 그대로 처리를 해야 한다. 즉, Down상태에서만 disenable

그러면 떨어지는 상황을 정확하게 인식하여, 점프를 풀어주기 때문에 무작정 false값을 줘서 count를 처리하는 경우는 없을 것이다.

잘 처리 된다.

구현된 처리 : Idle -> Jump Idle -> Walk ->

거리 기반 Walk 처리를 해주는 것으로 했다.

플레이어와의 거리가 300 픽셀 정도 차이가 난다면, 두번 점프해서 공격을 먹이는 것으로 한다.

거리가 500px정도 된다. 그러면, 걸어가서 거리를 좁히는 것으로 한다.

거리가 700정도 되면, 달려가서 때리는 것으로 한다.

Idle 모션을 진행한 이후에 어떤 식의 모션을 취할지 구하는데, 이것을 거리로 처리한다.

점프를 하는데, 플레이어의 거리에 따라서 X를 다시 설정하는 것으로 하자.

거리를 기반으로 해서, 이 속도로 이 만큼 움직여야 이 거리를 채울 수 있다는 시간을 정해서 그 만큼 이동하도록 한다.

내 속도가 400.f 인데 플레이어와의 거리가 100.f이다. 그러면. 몇 초간 이동을 해야 플레이어한테 다가갈 수 있을까 ?

1s -> 400.f

1/4 s -> 100.f

즉 정리를 하자면, 이동해야 하는 픽셀 / 초당 스피드 px -> 시간.

0.5s -> 200.f

지금의 비율을 유지하고 점프를 천천히 처리되도록 하자.

원작에서의 비율과 조금 차이가 나서 너무 아래쪽에 플레이어가 위치를 하고 있는데, 이것은

카메라의 피봇을 바꿔주면 어떻게든 처리할 수 있을 것 같다. 이건 BackGround의 위치와는 전혀 상관없는 처리니까..

현재 막아둔 처리가 갑자기 중력 값이 커져서 바닥으로 파고 들어가는 현상인데, 이것이 중력에 대한 처리를 막아야 하는 부분이지, 다른 부분까지 처리를 하지 않도록 하는 것이 맞는가 ? ㄴㄴㄴ

오직 중력에 대한 처리만 막아두는 것이 맞다. 그 외에 카메라 같은 부분에 이것의 영향이 간다면 곤란하다.

그러면, 중력에

장면을 변하는 과정에서 문제가 생길지 고려해보자. 장면 전환이 되었다. Scene에서 직접 Player를 셋팅해주었다. 그러면 참조횟수 감소에 대한 처리를 하고 업데이트를 통해서 정보를 가져오는데 아무런 문제가 없을 것이다. 터지지 않는다. 그리고 애니메이션 EditScene 의 경우에는 플레이어를 가져오지 않는다 이 상태로 애니메이션 클립만 생성을 해두고, 여기서 Update처리를 하지 않는다. (문제가 없다.) 단순히 애니메이션에 대한 업데이트 처리만 진행하므로 Animation Scene에서도 아무런 문제가 없을 것이다. ㅇㅇ

PlayMode 의 경우에도 미리 값을 셋팅을 해주기 때문에, 아무런 문제가 없을 것이다.

테스트. ㅇㅇ

근데 카메라의 문제가 그 부분이 아닌 것 같다… 띠용.

아니, 바닥과 보스가 충돌되고 있는 상태에서 계속 Sway를 주면 어떻게. ㅋㅋㅋ 이상할 수 밖에 없네.

한번의 임팩트 있는 점프 그리고 확실한 카메라 흔들림으로 해줄까..

아니면 시간을 재서 간격을 두도록 하는 것은 ?

흔들리는 간격을 지정할 수 있는 시스템이 있어야 혹은, 흔드는 와중에도 어느 정도는 플레이어의 위치를 잡아주는 시스템이 있다거나..

나는 Sway의 간격을 설정하는 시스템을 구현해서 주기는 고정(20배)이고, Scale 값을 통해서 움직이는 간격을 조정할 수 있도록 처리했다. 이것을 통해서 살짝 오래 흔들리는 처리도 가뿐히 할 수 있을 것 이다. 마음에 안 드는 부분도 수정 가능하고.

먼가 로직을 잘못 짠 것 같다. 처리가 이상해… ㅠㅠ

자 일단, 왜 진동이 안 되는 순간이 있는지 ?

잘 나오네 ? 이상하게

그러면 모션을 바꿔보는 처리를 해보자.

보스 패턴을 구현을 해보았는데, 일단, JumpAttack -> Idle

Walk -> Attak -> Idle

Run -> Attack -> Idle 거리에 따른 처리를 통해서 이렇게 3가지 패턴을 만들어 보았다.

여기서 더 많은 패턴, 빨아들이는 공격 / Balloon Attack 등을 처리할 수도 있지만,

지금의 패턴으로도 충분히 컨텐츠를 뽑아내는데 것에 문제는 없다. 따라서, 이 부분은 나중에 하고 싶으면 하는 것으로 하고, 따로 추가를 하지는 말도록 한다.

자 그러면, 이 상태에서 MoveStar를 생성해주는 코드를 작성하자. 잘 생성을 해주었으니, 이제 충돌 처리를 해야될 것 같다.

충돌이 되지 않았던 이유가 충돌에 사용하는 ColliderList와 Collider UI List에 해당 오브젝트의 Collider들이 들어가 있지 않기 때문이다. 아무튼 이것을 기존에 받은 UI 충돌체와, 일반 충돌체를 모두 release처리를 해주고 있다.

따라서, 우리는 이전에 사용하던 물체에 대한 충돌체들에 대한 정보도 날라가므로 다시 처리를 해줘야 하는 것이다. 결국에는 살아남은 오브젝트의 충돌체를 넣어주면 되는 부분.

이미 처리를 통해서 날리고, 살아남은 오브젝트에 대해서 다시 충돌체를 넣어주는 함수를 주면 된다. 그러면 이미 사라졌던 이전 장면의 충돌체가 다시 등록이 되는 개념. ㅇㅇ

이제 충돌처리를 해보자. 일단 자고 내일 하자.

8월 29일 4일…

오늘은 DDD클래스의 충돌에 대한 처리를 하는 것으로 한다.

원작과 동일하게 하든지, 아니면 다르게 하든지 그것은 마음대로 한다. 그러면 게임에서 존재하는 여러 동작을 생각해보도록 하자.

충돌과 관련된 처리의 동작만 생각해보자.

보스가 나왔는데, MIKE공격을 받는다. 그러면 죽는 것이 맞을까 ? (이상한데 이 처리는)

조금 강한 공격으로 인식하고, 죽을 정도의 처리는 하지 않는 것이 맞다.

현재 방식이 몬스터 클래스에서 DieTime이 잡혀있는 경우에 HitAnimtion을 계속 호출해주고 있는 상황인데, 이것을 없애고, 따로 HitTime을 잡아서 처리를 하는 것이 좋을 것 같다. (그럼 이 동안에는 무조건 HitAnimation을 호출하고 있는 것이다.)

혹시 Hit애니메이션을 호출하는 상황에서 아파서 계속 움직이는 오브젝트가 있는가 ?

그런 것이 아니라면 HitAnimation을 활성화 하는 상황에서 StopTime을 설정해주려고 하는데 ..

아니다 MoveDisEnable함수로 처리를 하는 것으로 하자.

MoveDisEnable을 Boss에서 처리하면, 넉백효과는 없는 상태로 단순히 맞은 위치를 기준으로 그곳을 바라보도록 하고, Hit애니메이션을 호출하며 이것이 끝나면 Idle상태가 된다.

그렇다면, 시간을 재면서 패턴이 시작되는 순간에 다시 자신의 처리를 하기 시작할 것이다.

MoveDisEnable 함수를 호출해주면, 충돌한 방향을 향해서 바라보고. Hit -> idle -> Stop Time이 끝나면 다시 Pattern 공격이 시작되는 것이다.

아무튼 이것은 나중에 확인을 해보는 것으로 하고 지금은 HitAnimation에 대한 처리를 하는 것으로 하자.

기본 공격의 경우에는 HitAnimation을 호출하고, 자동으로 Idle모션 까지 바꿔주는 처리 즉,

Hit -> Idle 처리가 되야 한다.

근데 마이크 공격의 경우에는 Hit -> Idle 이런 처리를 원하는 것이 아니라 마이크 공격이 일어나는 순간에는 계속 HitAnimation을 계속 호출해줘야 한다는 것이다.

그래서 나는 HitTime을 줘서 따로 이것이 켜진 시간 동안은 무조건 HitAnimation이 호출이 되도록 할 것이다.

그리고 사라지는 오브젝트의 경우 (일반 몬스터)에는 몇 초의 시간이 지나면 없어지는 처리가 똑같지만,

일단은 Hit시스템부터 구현. ㅇㅇ

기존의 일반 몬스터의 경우에도 HitAnimation을 덮어씌우는 처리가 없어졌으므로, 모두 설정을 해주도록 하자. (그리고 이 부분은 공통사항이니까)

일반 몬스터의 경우에는 잠시 동안, Gravity 시간을 멈춰두고, HitAnimation을 호출해주다가,

이 시간에 맞춰서 Mike공격이 끝나면 사라지게 되어있다.

Boss 의 경우에는 ?

아파하는 동안에는 GravityTime을 꾸준히 없애든지 처리를 해야 하며, 이 시간이 지나면 다시 중력처리를 시작… 그리고 다시 움직여야 한다.

HitAnimation을 셋팅하는 순간을 생각을 해보자. 만약에 몬스터와 플레이어가 충돌이 난 경우이거나, 플레이어의 공격과 몬스터가 충돌이 난 상태라면, HitAnimation함수를 셋팅하는 것이 아니라, StopTime을 셋팅해서 한번의 once return 애니메이션이 호출이 되도록 처리가 되어있다. 즉,

MikeAttack에 대한 HitAnimation처리는 이것으로 고정된다.

따라서 우리는 이 시간 동안에 GravityTime에 대한 처리를 같이 해줘도 상관이 없는 것이다.

같이 처리를 하니까..)

참고로 Gravity를 꺼버리면, GravityTime도 아라서 처리가 되므로 큰 문제는 없을 것이다.

(파고드는 현상은 있으려나 ? 있으면 이 부분은 수정을 하는 것으로 하자)

HitAnimation를 호출하는 순간(Mike) Gravity를 false로 주고, 이후에 HitAnimation이 끝나고, 다시 중력의 처리를 해주는 것으로 한다. ㅇㅇ

그러면 일반 몬스터의 경우에는 공격을 받고 중력의 처리를 받으면서 HitAnimation이 계속 호출이 되다가,

마이크의 공격을 맞은 경우에 한가지 처리를 추가해서, Move처리도 꺼버리는 것으로 한다.

그러면 이 동안 (맞는 동안은 움직이지 못하겠지..)

Move의 영향을 받는 처리가 막히는 것이다.

일반 몬스터 : Hit -> Move, gravity = false -> Die 팡. ㅇㅇ

보스 몬스터 : Hit -> Move, gravity = false -> Damage처리 -> 애니메이션 처리가 끝나고 다시 패턴으로 돌아간다. 중력 시간 / Move true ㅇㅇ

자 그러면 제대로 처리가 되는지 한번 확인을 해보도록 하자.

처리는 되서 애니메이션 호출하고 이 애니메이션이 끝나는 순간 다시 중력처리 true Move True를 줘서 다시 패턴으로 돌아가도록 처리를 했는데, HitAnimation이 2프레임이라서 문제가 생긴다. 이 부분은 Mike, Stone 에 대한 처리와 동일하게 처리를 하는 것으로 하자.

결국 2가지 중에 하나를 포기하는 것인데, 일반적인 충돌의 경우에는 0.5s가 지나면, Idle애니메이션으로 바꾸는 처리를 하고 있으므로, once loop Hit 옵션을 바꾸는 것으로 하자.

Once loop로 옵션을 바꾸었으니, 일반적인 충돌이 일어난 경우에는 idle로 자동 셋팅.

마이크 공격의 경우에는 loop로 처리가 될 것이다.

마이크 공격을 맞고서 갑자기 점프력이 상승하는 문제가 생기는데, 중력이 계속 false인 상태로 처리가 된 것이라고 생각된다. 즉 PrevGravity를 제대로 인식하지 못하는 것인데, DDD의 경우에는 Init에서 한번만 생성을 하니까, Prev값이 false라서 제대로 처리가 되지 않은 것이다. 따라서 SetGravity에서 이 값을 받아주면 될 것이다.

Init에서 미리 처리를 해주면, 이제 이 문제에 대해서는 잘 처리가 될 것이다.

그러니까, 공격을 맞고서, 기존에 하던 처리에 대해서 그대로 이어지는 처리가 되는 것이다.

ㅇㅇ 이제 점프를 조절을 해보자.

Mike 에서 HitAnimation이 제대로 출력이 되는지 / 끝난 이후에 이어서 잘 처리가 되는지.

잘 처리는 되는데, 이렇게 구현하기가 싫다. 따라서, falling 상태를 구현해서 이 상태인 경우에는 패턴의 처리를 막고 계속 떨어지도록 처리를 하자.

만약에 점프를 한 상태에서 마이크 공격을 맞은 경우에는 falling = true 상태가 되어서 업데이트 처리를 막을 것이다. (이 상황에서는 패턴에 대한 처리를 하지 않는다.)

그리고 Falling이 false가 되는 순간 -> 바닥에 떨어지는 순간부터 패턴을 다시 시작한다.

이후에 falling 을 true로 주는 경우는 없으므로 따로 처리를 하지 않고 있다가, 만약에 다시 점프를 한 상태에서 마이크 공격을 맞는 경우에는 다시 true처리가 되어서 다시 패턴에 대한 처리를 하지 않게 된다 떨어지면 패턴 초기상태로 돌려버린다.

점프를 하지 않은 상태라면, falling 은 계속 false인 상태이다. 따라서, 마이크 공격 시간이 끝난 이후에는 바로 Move = true 인 상태로 바로 패턴이 시작되는 것이다.

이것에 대해 확인해보도록 하자. ㅇㅇ 잘 된다. 살짝 시간 차이가 있는 것을 보니, 바닥에 붙은 상태에서부터 패턴에 대한 처리를 하고 있는 것이다.

그러니까, 내가 원하는 처리는, 점프를 하고 있는 상태에서 Idle처리를 하다가 갑자기 다 떨어지지도 않았는데 점프를 시작하지 않는 것. 해당 처리를 원했고, 점프를 하고 있는 상태에서 맞으면 이 시간 만큼을 처리하지 않고 있다가. 바닥에 붙은 경우에 처리를 하는 것으로 했고 붙어 있었던 경우에는 바로 처리를 시작하는 것으로 했다.

지금은 일반공격에 대한 처리는 확인할 수 없는데, 일단, 마이크 공격에 대해서는 hit애니메이션은 잘 출력해주고 있다.

다음 작업은 충돌연결.

Boss는 모든 넉백효과 넉업 효과를 받지 않는다. (무적 시간과 blend, Damage처리만 하는 것으로 한다)

일단 Body에 대한 처리를 했으니 다른 것을 막아본다.

원작에서 slide와 prevent에 대한 처리를 확인해보자.

만약 무적 상태에서 입김을 맞은 경우는 어떻게 될까 ? 그냥 사라지는 처리로 가는 것 같은데 ?

NORMAL : 별 뱉기 ㅇㅇㅋ / 흡입하기(처리X) ㅇㅇㅋ

MUTUAL : 입김 뱉기 ㅇㅇㅋ / 슬라이드 ㅇㅇ / Preventㅇ ㅇㅋ

슬라이드의 경우에는 충돌이 되는 경우에 역으로 이동되는 것으로 하자.

별 뱉기의 경우에는 데미지를 먹되 날라가지는 않도록 한다.

아니 애초에 보스가 날라가는 상황이 있나 ? 없잖아 이거 다시 풀어버리면 되지 않을까 ?

일단, 확인. Blend 상태에서 다시 공격이 일어날 정도의 처리가 없고, 그나마 SwordAttack 정도인데, 이 경우에는 처리를 하지 않아서, 단순히 이펙트가 사라지는 처리만 하게 되는 것이다. 아무튼, 전혀 문제가 없다.

STONE : 돌 떨어지는 상황에서만 하는 처리 ㅇㅇ

Stone 공격에서 그냥 몬스터가 부딪힌다고 바로 처리를 해버리지 않고, JUMP가 TRUE인 상태에서만 처리를 하는 것으로 하자.

근데, 단순히 그런 식으로 처리를 해버리면, 떨어지는 상태가 아니라면 충돌이 안 된다.

즉, 달려간 상태에서 돌로 변하거나 걸어가서 돌로 변하거나 전혀 충돌이 되지 않는 것이다.

떨어진 경우에는 처리. 걷거나, 달리다가 변한 경우에는 처리 X

그냥, 이거 디테일 신경 쓰다가, 너무 힘들 것 같으니, Boss에 대한 처리만 따로 하는 것으로 하자.

즉, Boss는 Jump가 true 인 상황에서 (돌로 떨어지는 상황에서만 충돌을 하는 것으로 한다)

달려가서 쓰거나, 걸어서 쓰거나, 처리를 무시. 점프에서 떨어지면서 돌로 변하는 경우에만 충돌하는 것으로 하겠다.

슬라이드를 하는 경우에 Stop을 사용하므로 이것도 없애주는 처리를 해야한다.

SWORD : 일반 공격 ㅇㅇ / 대시 공격 ㅇㅇ / 점프 공격 ㅇㅇ

대시 공격의 경우에는 보스가 어떤 상황이든 상관없이, 충돌이면 처리하는 것으로 한다.

근데, 점프 공격의 경우에는 좀 달라져야 한다.

일단, 공격시에는 처리를 막아버린다.

그리고, 내가 점프공격을 하는 상황에서 충돌이 난 경우에는 공격을 먹어야 할 것이다.

// 이 처리는 보스전에서만 !

MIKE : MIKE\_ATTACK (처리)

아예 충돌처리를 안 하는 것을 보니 설정이 잘못되어 있는 것 같다. 보니까 충돌체가 따로 없네…

근데 왜 타일 충돌은 되고 있을까 ? 타일 충돌은 기본적으로 true인 상태이다. 그리고 충돌체의 크기를 가지고 처리를 하는 방식이 아니다. 단순히 StaticSize를 가지고 처리를 하고 있다. 따라서 충돌체와는 전혀 상관이 없다.

아무튼, 충돌체가 없어도 기본적으로 타일에 대한 충돌처리는 진행이 되는 것이다. 일단 충돌체를 넣어주는 것으로 하자.

아 이걸 AddCollider로 설정한 것이 아니라, 따로 설정을 해버렸네. 상관은 없지만.

넉백 혹은 넉업 상태에서 입력을 받지 않도록 처리를 했는데, 이 상황을 깨야 하는 경우가 생긴다. 단순히 처리를 하고, 이 상황에서는 가능하도록 하는 flag가 필요해진 것이다.

보스와의 충돌과 관련된 처리를 하는 m-bBossCollision 변수를 두는 것으로 하자.

그리고 보스와 충돌이 된 상황에 true를 주고 그것이 아니라면 false처리를 둬서 이후의 입력처리를 막을지 막지 않을지 확인하는 것이다.

이 처리가 단순히 KnockBack에 대한 처리를 막는데 사용해야 하는가 ?

넉업을 하는 상황을 없을까 ? 있을 것이다. 그러면, 단순히 Boss와 충돌이 되는 상황에 flag를 true로 처리하고, 이 경우에만 예외 처리를 할 수 있도록 하자. 즉, 입력을 막아야 하는 상황이라면 !BossCollision && m\_bKnockBack 이 경우에 막는다.

반면 m\_bBossCollision의 경우(넉백이 되었지만, 보스에 의한 처리이라고 한다면) 조건을 false로 처리하여 입력을 받을 수 있도록 한다.

플레이어가 날라간 상황에서 기본적으로 강제 입력 처리를 false로 준다. 그리고 보스에 대한 처리 중에 입력을 받아야 하는 순간에 이것을 true로 주는 것이다. 무조건은 아니고, slide라던가, prevent 라던가 꼭 해줘야 하는 순간들이 있으니 이 상황에서 이 flag를 가지고 처리를 해주는 것으로 하자. 보통의 몬스터의 충돌 처리는 Input = false라서 처리가 막히고, 보스는 처리를 해줘서 입력처리를 하는 것이고, nockback처리가 끝난 상황에서 이것을 false로 주자.

보스와의 충돌에서 입력을 막지 않아야 하는 상황에서는 InputDisEnable의 값을 가지고 예외처리를 하도록 하자.

근데 이 처리를 우리가 필요한 상황에서만 처리하면 되는 것이다. 구체적으로 말하자면, preventUp 키에 대해서만 처리를 막는 것을 풀어주고자 하는 것이다.

그러면 다른 키 예를 들면 MoveSide 같은 키들은 처리가 되지 않는 것이다.

넉백이 되면서 Prevent UP 키를 통해서 해당 상태를 풀어버릴 수는 있지만, 좌우로 움직일 수는 없다.

아무튼 prevent는 끝.

보스의 Attack 시점에 충돌체를 사용하여 플레이어의 충돌처리를 해야한다.

자 이제, 보스의 패턴과 충돌 처리를 어느 정도 했으니, UI작업을 해보는 것으로 하겠다.

참고로 맞은 후에 0.5s가 지나면 다시 move = true 상태로 돌리고 있다. 그러면 이 상태에서 이전에 하던 처리를 계속 이어서 하게 되는데 이것은 부자연스럽다. 따라서 맞은 이후에는 초기 상태로 돌리는 것으로 처리했다.

case DS\_IDLE:

case DS\_WALK:

case DS\_RUN

case DSJUMP\_ATTACK:

case DS\_ATTACK:

이렇게 상태가 있는데, 맞는 애니메이션은 어느 상황에서 호출이 되는가 ?

Idle Walk Run

아니 몰라 해보자 . 점프하고 별 쏠지도 모르는 일임.

보니까, Hit 애니메이션 이후에 바로 Idle로 들어간다.

이 시간을 지켜주고 싶다면 우리가 마이크의 처리를 하던 것 처럼 하면 될 것이다. ㅇㅇ

이건 좋은데, Hit 상태에서 바로 Collider를 풀어버리는 처리를 해줘야 한다.

정상적인 처리를 통해서 애니메이션의 특정 프레임에 시작 / 종료에서는 처리를 해줘야 한다.

예외적인 상황은 맞는 상황인데 이 경우에는 애니메이션이 끝나고 충돌체를 비활성화 할 것이 아니라

맞는 그 순간에 바로 처리를 해주는 것이 맞다.

DDD의 Damage 함수를 오버라이딩 하여, 맞는 경우에는 충돌체를 바로 꺼버리도록 처리했다.

그렇다면 hit애니메이션이 끝나고 나서 처리를 하는 것이 아니라 충돌이 되자마자 바로 충돌처리를 하게 될 것이다. 확인. ㅇㅇ

왜 계속 스킵이 되는 느낌이 나지 ?

커비가 변하는 과정에서 충돌체를 꺼주자. ㅇㅇ

이제 보스의 체력바를 출력해주는 것으로 한다. ㅇㅇ

위치를 한번 설정해보자.

스포너의 출력과 Update가 안 되는데, 함수의 이름을 직접적으로 호출하는데 인식을 못한다. ee

아니 다시 디버깅모드로 시작하니까 되네 뭐지 ? gee ㅇㅇ

자 아무튼, 설정해준 오브젝트가 죽으면 Border와 HP를 활성화를 풀어버린다.

그리고 장면이 전환이 되면, 자동으로 이것을 세이브한다. 왜냐하면 UILayer의 소속이기 때문에

그러면 보스가 나오는 상황에서 타겟을 보스로 넣어주면 아라서 처리가 될 것이다. ㅇㅇ

이전에 생성했던 오브젝트를 찾아오는 방법이 있었던 것 같은데 뭐더라 ?

다음 장면의 Init순간에서는 다음 장면에 오브젝트를 추가한다. (이 상태로 FindObject함수를 호출하면, 다음 장면을 기준으로 오브젝트를 찾게된다.)

즉, 우리는 이전에 사용하던 오브젝트를 찾아오고 싶다면 이전의 장면에서 찾아와야 할 것이다.

적용. ㅇㅇ

보스의 패턴이 나오고 있는 상황에서 끊기면 안 되는 모션에서 공격이 맞는다면, 데미지 / 블렌드를 주는 것으로 하자. 즉, Hit은 처리를 하지 않는다. 그냥 단순히 맞았구나 처리만 주는 것이다.

0.2f 1번. 으로 설정했다. 충돌이 난 상태에서 칼날만 비비면, 안전하게 공격을 성공할 수 있는데,

너무 파고 들면 충돌처리가 되는 것이다. (크 원작스타일 좋네) ㅇㅇ

그러면, 보스가 죽으면 나오는 처리를 해주자.

8월 30일 6시에 출발에서 지옥 버스를 타고 7시 20분 정도에 와서 샌드위치를 사먹다.

아무튼 오늘은 빡세게 효율 3~4배 뽑는 것으로 하자 !!

오늘의 작업 : 남은DDD 처리 (사망쪽)\_해결하고 ㅇㅇ, 장면을 2개이상 제작한다.

Stage1 DDDScene 밖에는 없다. 따라서, 우리는 Stage2 Scene을 만들고, Stage3 Scene을 만들도록 한다.

장면 편집에서 너무 부실해. 그거 솔직히 뭔가 선택해야 하는데, 이상해. 메시지 박스는 좀 아닌거 같아. 그러면 장면 따로만들까 ? 장면 만들고 처리할 장면에 대해서 누를 수 있도록.

근데, 이걸 만들 수 있는데, 지금은 하지 말고 나중에. (디테일)

맵툴에서 Line에 대한 정보를 표시해서 출력해주는 것으로 하자.

Line에 대한 정보를 읽어 올 수 있도록 하고, Spawner까지 작성을 하는 것이다.

그러면, Tile + Line + Spawner 정보를 통한 삼위일체로 스테이지 제작은 정말 쉽게 할 수 있다.

(잔디는 Pass)

일단, Line과 Spawner를 저장하는 시스템을 구현한다. (디테일)

스포너는 나중에 하고, Line에 대한 정보만 어떻게 보면서 처리할 수 있도록 하자.

31일.

스포너의 정보에서 필요한 것을 저장하고 읽어올 수 있는 시스템을 개발한다. 그런데, 이것을 쉽게 수정할 수 있도록 txt 파일로 만드는 것으로 하자. 수정을 쉽게 할 수 없도록 만든다면 (Binary File) 툴을 통해서 읽어와서 고칠 수 있도록 해야 한다는 것인데, 그런 세세한 부분까지 고려하기에는 시간이 부족하다. 따라서, 파일을 통해서 수치만 바꾸면 바꿀 수 있는 형식인 텍스트 파일로 만들자.

Line은 읽어오는 시스템 OK (맵툴에서는)

스포너의 위치를 저장하는 시스템을 만든다.

//pStage->AddSpawner("BurtSpawner", 2000.f, 400.f, SD\_LEFT, "Burt");

//pStage->AddSpawner("TwizzyFlySpawner", 6000.f, 400.f, SD\_LEFT, "TwizzyFly");

//pStage->AddSpawner("TwizzySpawner", 5000.f, 400.f, SD\_LEFT, "Twizzy");

//pStage->AddSpawner("GrizooSpawner", 4000.f, 400.f, SD\_LEFT, "Grizoo", true, 1);

pStage->AddSpawner("MikeSpawner", 2000.f, 400.f, SD\_LEFT, "Mike", true, 1);

pStage->AddSpawner("BladeSpawner", 6000.f, 400.f, SD\_LEFT, "Blade", true, 1);

pStage->AddSpawner("RockySpawner", 5000.f, 400.f, SD\_LEFT, "Rocky", true, 1);

pStage->AddSpawner("SleepSpawner", 4000.f, 400.f, SD\_LEFT, "Sleep", true, 1);

일단, 기능은 Spawner Mode에서 내가 원하는 위치를 마우스로 클릭하면, 해당 위치를 기준으로

사각형이 잡히면서, 위치를 저장한다.

Spawner모드에서는 옵션 Change Key를 통해서 내가 원하는 몬스터를 선택할 수 있다.

생성 방향은 항상 SD\_LEFT 값으로 정해져 있으며, 우리가 설정하는 것은

현재 처리하고 있는 스포너의 위치는 어떻게 할까 ? 따로 출력하는 것으로 한다. (그냥 하지 말자 이거는 넣고 빼고 넣고 빼고 하면 되잖아)

스포너의 위치 ( 모든 스포너의 위치를 말하는데 이것도 모드에 따라서 다르게 출력하자,

Line모드에서는 Line을 … 스포너 모드에서는 스포너를 …) / 몬스터 ㅇㅇ/ 몬스터의 생성 방향ㅇ ㅇ , 카운트 ㅇㅇ 정도이다.

그러면 원하는 위치에 몬스터의 스포너를 생성하고 이것을 저장하는 기능을 만들어보자.

출력하는 기능까지 ! ㅇㅇ

모드에 따른 처리를 통해서 ESC / Delete / Save ! ㅇㅇ

파일의 저장기능도 만든다.

이거 그냥 따로 하면 안 되냐 ? 성우 쌤이 짜신 코드 그대로 복사하면 깔끔하게 저장할 수 있도록 할 수 있는뎅 ~ 아니면 지금처럼 시스템 정보를 활용하여 계속 저장하도록 해도 되고…

따로 해주는 것이 더 멋들어지지 않을까 ? ㅋㅋㅋ

ㅇㅋ 얼마 안 걸리니까 따로 해주자.

Tile : Save / Load / m\_pStage-> save load 함수 이용..

LineCollider & Spawner / SaveObjectPos / LoadObjectPos -> SaveObjectPosText LoadObjectPosText함수를 이용한다.

일단, 이거 함수 지정하고, 처리부터 해보자. 잘 되는지.

함수를 동시에 호출하되, 어느 한 쪽은 반드시 막히게 되어 있다. (왜냐면 모드는 중복이 나지 않으니까) ㅇㅇ

기본 디렉토리 동일하게.. 잘 읽어오네요 . ㅇㅇ

위치를 출력하되 카메라의 영향을 받도록 해야 한다.ㅇㅇ

생성 몬스터 / 위치 /

방향 / 숫자

현재 스포너에 대해서 정보를 저장하고 읽어오는 것(맵 에딧 씬에서)은 다 했는데,

설정하는 입력키나, 위치 설정 등의 기능이 없어서 새로운 무언가를 추가할 수가 없다.

Line / Tile Option 에 대한 마우스의 처리는 이미 되어 있으니, 스포너에 대한 마우스의 처리를 추가한다. 그냥 간단하게 점 찍는 처리를 해주면 될 것 같다. (이 정도면 괜찮네)

클릭하는 경우에 위치를 설정해주었고, 뗀 경우에 none상태로 바꾼다.

출력을 하다가 우클릭을 누르면 none상태로 바꾸고, 이것을 출력하지 않는다.

이 상태로 추가를 했다면, 본래의 스포너의 위치를 출력하지 않기 위해 0 0의 값을 준다.

그러면서 중복된 위치에서 출력이 되지 않도록 하는 것이다. (원래 추가하면 그 위치에 이거 띄우고 저거 띄우고 하니까)

현재 문제점 : 스포너의 위치가 제대로 출력이 되지 않는다. 그림도 그렇고 정보도 그렇다.

그리고 추가하는 상황에서는 타일의 격자 때문에 잘 보이지도 않는다. 따라서, 기능키를 통해서 Tile을 출력할지 말지를 결정하는 함수를 만들도록 하자.

(단순히 특정 모드에서 처리를 하지 않는 방식으로 하면 같이 봐야 하는 상황에서는 답이 없다)

경우를 생각해보자, TileRender가 true인 상태라면 그냥 처리를 하면되지. 근데, TileRender가 false인 상태라면 출력하지 않으면 된다. 그런데 만약에 false인 상태에서 Tile에 대한 수정을 가하려고 하는 순간 true값을 줘야 하는 것이 아닐까 ? 그렇지 ! 당연하지.

할거면, 모든 모드에 대한 처리를 하는 것이 맞지. (이거 끄고 저거 끄고 할 수 있도록 ㅋㅋ)

그게 재밌지 않나 ?

그러면, Tile에 대한 옵션을 끄고, 키면, 우리가 Tile에 대한 충돌을 키고 끌 수 있도록 구성하면 된다. ㅇㅇ

그리고 이 정보를 출력해주는 것으로 한다. ㅇㅇ

타일은 적용 / 라인 / 스포너 모두 적용.

보이지 않는 것은 넣지도 않는다. (이걸로 막아주는 것으로 하자. 보이는 것에 대한 처리를 하도록 한다. )

이제 일단 저장이 잘 되는지 확인해보도록 하자.

덧붙이면, 이전 생각에는 false인 상태에서 해당 구역의 처리를 하려고 하는 경우에 그냥 true로 만들어주는 처리를 하려고 했는데 ! 그냥 보이지 않는 것은 처리하지 않도록 하는 것이 좋을 것 같다. (그러니까 모드의 개념인데 여기에 보이냐 보이지 않냐의 개념까지 섞인 것이다.)

스포너의 위치를 제대로 출력해주자.

출력하는 위치가 달라지는 것은 무엇. 아마도, 실제 값에서 이것을 빼다가 문제가 생긴 것으로 생각한다. 즉, 출력위치는 분명 카메라의 위치에 따라 값이 달라지는 것이 맞는데, 실제 위치는 그대로 출력을 해주는 것이 맞다. ㅇㅇ 해결.

위치를 읽어오고 저장하는 시스템이 완성이 되었으니, 이것을 기반으로 이제 타일과 정보를 저장하고 읽어서 적용을 시켜보도록 하자.

저장하는 것에 Door의 정보까지 저장하는 시스템을 구현한다. (디테일)

입력을 통해서 옵션을 바꾸는 기능을 추가한다.

GET\_SINGLE(CInput)->AddNormalKey("3", '3'); // Dir

GET\_SINGLE(CInput)->AddNormalKey("4", '4'); // Count Switch

GET\_SINGLE(CInput)->AddNormalKey("5", '5'); // ++

GET\_SINGLE(CInput)->AddNormalKey("6", '6'); // --

스테이지에서 이 파일을 읽을 수 있는 시스템부터 구현한다.

타일 읽어오기 (구현이 되어있음) / LineCollision & SpawnerInfo 를 읽어오는 시스템을 구현한다.

LineCollider와 Spawner 정보를 저장하는 시스템. 그러면, 이것을 어떻게 구별을 할 것인가를 생각해보자. 모드가 Tile인 상태로 Load를 하는 경우에는 타일을.. Line / Spawner 상태로 Load를 하는 경우에는 해당 함수를 호출해주는 것으로 하자. !

옵션 정보에 따라서 Tag정보가 달라진다. 활성화 정보가 true라면 우리가 저장하고 읽어온 이후에 이것의 영향을 받을 것이요, 그렇지 않다면, 그냥 의미 없는 수치가 될 것이다. 따라서 굳이 조건을 걸 필요는 없는데, 깔끔하게 걸어주도록 하자.

Count flag가 false인 경우 0이 되도록.

생각을 해봤는데, 현재 출력하고 있는 위치 출력의 경우에는 Stage 자체에 띄워버리면 될 것 같다. 굳이 이걸, 화면에 띄울 이유가 전혀 없다… (좁기도 하고… ) (애니메이션 편집에서는 가치가 충분 했지만, 여기서는 아니라고 판단된다)

따라서, 나는 모든 배치하는 오브젝트를 스테이지를 기준으로 출력할 것이다.

라인의 경우에는 Line을 출력하고 Dest의 위치를 여기야 여기 스포너 위치 출력 부분

아직 입력은 안 했다.

자 오늘까지 간단하게 맵툴을 만들어서, 이것을 멋이 나도록 가져다가 쓸 수 있는 툴을 만들고,

내일 사운드 작업 + 오류 해결, 디테일한 부분을 만들도록 한다. 아무튼 오류가 우선임 ㅇㅇ

아무튼, 하던 것을 마저 하자.

맵툴에서는 공통된 기능이 하나가 있는데, 지금 편집하고 있는 무언가가 있고, 이것을 적용을 시키거나, 삭제하거나 하는 기능이다. 현란한 편집까지 구현할 시간은 없고, (그래서 txt파일로 만든거지 ㅇㅇ) …

그래서 Spawner모드에서도 현재 위치를 표시해주는 기능을 추가한다.

이것도 동일하게 처리해주자. 뭔가 마우스를 이용한 처리를 하다가 우클릭을 누른 경우에 초기화를 시켜버리는 것. ! 좋네 !

직선의 길이에 대한 예외처리 / 위치에 대한 예외처리.

enum TILE\_OPTION

{

TO\_NONE,

TO\_NORMAL,

TO\_WALL,

TO\_END

};

모드에 따라서, 변경하는 옵션이 달라지도록 구성을 하자. 예를 들어서, 현재 모드가 타일옵션이다. 그러면, 타일 옵션을 변경하는 것으로 하고, 현재 모드가 Spawner라면 SPAWNER옵션을 바꾸는 것으로 하자. ㅇㅇ

모드에 따른 출력 정보는 당연히 달라진다. (Text / Map)

모드에 따른 최근 정보 삭제 기능 / 모드에 따른 최근 정보 데입 기능

Spawner Dir을 바꾸는 기능. 스폰 모드인 경우에 특정 키를 누르면 변한다… 이런 식으로

생성 개수를 설정하는 기능. 푸는 기능.. flag

개수를 설정하는 기능 +-

마우스의 위치를 출력하는 시스템. 그리고 그것을 보여주는 시스템을 만들자.

만들었으니, 이제는 모드에 따라서 키를 다르게 처리를 할 수 있도록 한다.

Line 모드 좌클릭. Src / 우클릭 Dest 그 상태로 Enter키를 누르면 해당 좌표를 저장하고 다른 처리를 하도록 한다.

이거 시간 남으면 하는 것으로 하고 나중에 하자. (지금은 스테이지)

근데 이거 좌표를 보여주는 것은 필요해. N을 누르면 src 설정. M을 누르면 dest 설정.

이 상태에서 엔터키를 누르면 이 좌표를 저장하고 Render

델키를 누르면 가장 최근에 추가했던 좌표를 지우고 Render가 되지 않도록 처리.

입력, 출력

일단은 src dest를 출력하는 것으로 한다 입출 ㅇㅇ . 그리고 이것을 저장하는 기능(일단은 벡터ㅇㅇ에) 입출. 지우는 기능 입출.(벡터) ㅇㅇ

Enter 저장 / Delete 없애기 ㅇㅇ

B src / N dest / M src dest delete ㅇㅇ

Src와 dest를 통한 Line 보라색 ㅇㅇ

그 외에 나머지 Line 빨간색 ㅇㅇ

다 잘 되는데, b와 n을 통한 직선 긋기가 굉장히 마음에 안 든다. 차라리 마우스의 충돌을 하는 것이 백배천배는 낫다…. ㅎㄷㄷ 그거 ESC로 날리는 것도 따로 처리를 해주도록 하자.

모드 개념을 도입을 하는 것이 정신건강에 좋을 것 이라고 판단된다.

누르면 src 땡기면 dest하면서 render 업하면 NONE 상태로 LINE에 대한 처리를 클릭 상태 NONE 으로 처리를 한다. ㅇㅇ

Line 모드에서 우클릭을 누르면 지금 처리하고 있는 것을 삭제하며 enter 저장 / backspace 삭제

ㅇㅇ

현재의 문제점 하나를 짚어주면, 지금 추가해둔 라인을 다시 만드는 시스템이 없다.. (디테일)

우리가 라인 정보를 저장할 수 있는 시스템을 구현한다면, 금상천화이다. (하지만, 이것은 디테일의 부분)

즉, Line정보를 저장하고 읽어오는 시스템을 구현한다. (디테일)

현재 만들어진 내용을 토대로 직접 보면서 라인충돌체를 넣도록 한다. ㅇㅇ

Sound 분류에 들어간다.

31일 사운드를 2시간만에 모두 적용하고 보스를 하나 더 만드는 것으로 한다. (조아쓰)

// 필수

자기가 그리고 있는 것과 기존에 저장된 것을 다르게 출력하는 시스템을 구현한다.

텍스트를 가지고 출력을 하는 경우에는 바로 월드좌표에 출력이지. 그런데 우리가 출력을 하려고 하는 경우에는 ? 카메라의 영향을 받도록 해줘야 한다.

스테이지에 LineCollider를 추가하고, 스포너를 넣는 기능을 만들었다. (BackGround ForeGround는 디테일) 아쉽다 ㅠ

이제 제대로 데이터를 넣고 이것을 기반으로 스테이지 읽도록 해보겠다.

아니 이거 스테이지에서 테두리 한번 둘러치는 기능 없으니까 너무 힘들잖아 빨리 추가해 ㅇㅇ

라인과 스포너의 정보를 추가하고 이것을 읽어와서 Stage2Scene 에서 읽어오는 처리를 했고,

스테이지 1에서 문을 열고 들어가면 스테이지 2로 연결되도록 했다.

// 아무튼, 우리는 Stage2의 처리가 잘 되었는지 확인하자.

// 지형충돌이 자연스럽게 이루어 지는가 ? 스포너의 정보는 맞는가 ?

스테이지 정보 맞다. Line충돌 자연스럽다. 그냥 디테일한 부분에서 문제가 있는 것 뿐이지 스테이지 자체를 읽어오고 적용하는 것에는 전혀 문제가 없다.

상황을 보니까, 스포너의 처리에서 Y축에 대한 생성을 막아야 한다. 그렇지 않으면 점프할 때 마다 생성이 되는 현상이 일어난다.

그냥 적당히 올려주는 것으로 하자 너무 바닥에 붙으면 안 된다.

디테일 디테일 디테일 디테일 디테일 (맵 에딧 씬에서 오브젝트 위치 변경하는 부분 디테일 디테일)

Line 충돌체를 넣는 상황. Tip 써본다

올라가는 상황 즉, 이 시점에서 타일과 충돌을 원하는 상황에서는 살짝 올라가는 것으로 처리가 가능하다.

아래에서 올라가는 상황에서는 타일 밑으로 들어가면 곤란하다. (여기서 Tile의 충돌을 막아버리고 Line을 따라가는데, Line때문에 파고 드는 현상이 생긴다)

직선의 경우에는 조금 높게 잡아야 맞는다.

물체가 움직이지 않는 이유 : 그거 타일충돌 된거임.. ㅇㅇㅇ

키를 추가하는 부분이 장면 생성에서 들어가기도 한다.

어느 시점에서 플레이어를 올려줘야 할지… 계속 올릴 수도 없는 거고… 이전 오브젝트와 새로 추가된 오브젝트가 동시에 있는 시점.. 언제 일까 ?

그냥 flag 하나 넣고서, 장면이 바뀐 시점에서 몇 초가 지나면, 다시 올려주는 처리를 하면 안 되나 ? 아니면 몬스터가 생성되는 시점에서 그냥 올리거나.. 아니 근데, 이것도 아니야 ! 아니야

몬스터 생성 시점에 올려서 항상 플레이어가 나중에 출력되는 것도 괜찮지.

스포너가 있다 -> MainScene이라는 의미이다.

근데 이것도 이상하잖아.. 그러면, 장면이 바뀐 이후에 SceneManager에 변수 하나를 두고, 이것 이후에 출력해버리자. 근데 이게 시간까지는 필요가 없다. 왜냐하면 오브젝트를 담아놓고 WorldObjList 에서 배치를 하는 것은 1프레임이면 끝나는 작업이기 때문이다. 즉, 장면이 바뀌는 순간에 true를 만들어놓고, 다음 프레임(이미 추가했을 것이다.)에서 LayerUP에 대해서 처리를 하고 false!

자 그렇다면 호출되는 시점을 잘 생각해보도록 하자. 아니, 차라리 Grass Layer 를 따로 만들면 안 되냐 엄청 귀찮은데.. ㅇㅇ

그냥 너무 쉽게 해결해보림.

이걸로 하셈 ㅇㅇ 좋음

스테이지 몇 개 정도 추가하고 + 터널 하나 정도 추가한 이후에 보스 하고 끝내는 걸로 하자.

적당한 이미지 하나 잡아서 이걸 보면서 끝내는 것으로 한다.

일단 집에 가자. 밤새면서 스테이지 몇 개만 추가하고 자는 걸로…

9월 1일 내일 발표.

아직 사운드도 안 했는데에 ~

ㅋㅋㅋㅋㅋ

자, 내가 원하는 시스템은 일단, 플레이어의 시작 위치와 Door의 위치를 설정하는 기능이다.

아니 근데, 생각해보자. 시작 위치를 정해놓는 방식인데, 어느 위치에서 나와서 어느 위치로 가는게 그게 되나 ? 애초에 해당 장면으로 가면, Door를 통해서 위치 설정이 되는 것이 아니라, 단순히 해당 스테이지의 시작위치를 기준을 할 것이니까… 아니면 예외 처리로 이 경우에는 문으로 이동한다… 이런 식으로 하면 되는데 이걸 설정할 바에는 (내 게임에서도 사용하지 않을 것이니) 그냥 1개의 문으로 고정시키면 그만이다. 그리고, 미리 이동할 장면을 설정하거나, 따로 설정하거나 할 수 있으면 좋을 것이다.

1. 도어 정보에 이동할 장면을 scenetype을 미리 설정.
2. 아니다, 이건 바뀔 수 있는 부분이니까, 나중에 따로 설정한다.

우리는 2번으로 간다.

배치할 위치가 2개. 즉, 스테이지의 시작위치와, Door 위치를 설정할 수 있는 시스템을 구현한다.

Position Mode 해가지고, 시작 위치, Door 를 옵션으로 변경가능 하도록 하면 된다.

스포너, 라인충돌체처럼 여러 개가 필요한 정보인가 ? 아니다. 단지 하나만 있으면 된다. (하나로 고정할 것이고)

따라서 현재 처리하는 … 그런 정보는 필요가 없고, 그냥 바로 적용해주면 되는 부분이다.

그냥 해당 옵션에서 클릭하면 바로 바뀌도록 처리.. 그리고, Start위치와 Door부분만 따로 출력해주는 것으로 한다.

2가지 처리를 할 것인데, 하나는 왼쪽 상단에 고정 시키면서 출력할 정보이고, 나머지 하나는 해당 위치에서 TextOut을 통한 출력이다. ㅇㅇ

포지션 모드 출력. ㅇㅇ

입력 ㅇㅇ / 출력 ㅇㅇ / 옵션 변경. ㅇㅇ 변경한 옵션의 출력 ㅇㅋ

문이 어떤 식으로 출력이 되는지 사이즈를 알고 있어야 한다.

자 잘 출력이 되니까, 이제는 저장하는 기능을 만든다. ㅇㅇ

현재 마우스 입력이 잘 되어 있지 않다. 따라서, 이것에 대한 처리를 하도록 한다.

이거 현재 위치 개념 도입해야 겠네.. 실수로 클릭한 것을 움직여버리네..

Enter delete 개념을 추가한다. 다만 벡터에 넣는 개념이 아닐 뿐이지. 클릭하면 그 부분을 Render하여 보여주자. esc에도 적용하고.. ㅇㅇ save delete에도 적용했다. 출력도 해당 위치에 해주고 있고(옵션에 따라서) 개꿀.

확인해보고, 이것이 미리 입력을 하는 방식이라서, 안전하게 추가를 할 수 있을 것이다.

잘 했다. ㅇㅇ ㅇㄴㅁ

우클릭은 Pos의 위치를 바꾸는 것으로 한다. ㅇㅇ

도어 저장 / 출력/ 입력 / 시작위치 저장 시스템 구현 완료.

맵의 Fore, BackGround를 설정하고 카메라에 제한을 걸며 해당 크기를 읽어와서 처리하는 시스템을 구현한다.

즉, 미리 타일을 생성하지 않으며, 스테이지에 대한 BackGround or ForeGround를 설정하는 방식이 아니다. 이런 방식을 통해서 우리는 좀 더 맵툴 다운, 선택지가 많은 무언가를 만들어 낼 수 있으며, 좀더 멋지게 보일 수 있다. 현재는 그냥 출력만 해주고 있는데, 스크롤 속도를 적용하여 움직이는 시스템을 한번 구현해보도록 하자. (직접 볼 수 있도록 ~)

일단, enum문을 통해서 여러 맵의 종류를 추가하고 이것의 데이터를 어딘가에 저장하는 시스템을 구현한다. MapScene에서는 필요하고, MainScene에서도 필요하다. 아니, enum이 필요한가 ? 글쎄 ?

너무 많아서 필요가 없을지도.. 그냥 단순 수치를 저장하는 것으로 하자. 우리가 계속 저장하던 텍스트 파일에 추가적인 정보로 우리가 선택한 스테이지의 Rect, 우리가 고른 ForeGround의 Rect를 저장해야 하나 ? 그러니까, 어딘가에 저장을 해둬야 하는데, 뭔가 하드코딩스럽지 않게 만들기 위해서는 이 스테이지 정보, ForeGround정보도 텍스트 파일을 통해서 저장할 수 있도록 해야하고, 이것을 읽어서 게임이 시작되면 누군가가 들고 있도록 하다가 없애주면 된다.

MainScene에서 당연히 사용하고, MapScene에서도 사용하는 정보이다. 그렇다면 SceneManager 가 들고있도록 하는 것을 어떨까 ? 이렇게 하면, 어느 위치에서 있든 가져다가 쓸 수 있으며, 반대로 이 정보를 찾아오기도 편할 것이다.

정리를 하면 이렇다. SceneManager는 vector(더 많은 정보가 추가될 수도 있지 않은가)를 통해서 스테이지의 위치(정보를 알면 당연히 크기도 알 수 있으니 이것만) 정보와,(이게 ForeGround) /

BackGround의 위치정보를 저장하며, (근데 원하는 정보가 다르니 따로 저장하자.)

우리가 스테이지의 정보를 저장하는 곳에서는 이 정보를 모두 저장하여 Pos : x , y 이런 식으로 저장하는 것이 아니라, 단순히 인덱스를 저장하는 것이다.

텍스트에 추가한 순서대로 ForeGround정보든, BackGround정보든 들어가게 되는데, 우리는 이 인덱스를 활용하여, 벡터에 들어간 순서대로 바로 저장해버린다.

예시를 들어보자. (ForeGround vector의 정보이다)

1 : 0 0 100 100

2 : 100 100 200 200

3 : 300 300 400 400

이런 식으로 있다고 생각해보자. 텍스트에서도 이 순서대로 저장이 되어 있을 것이다.

그러면 우리는 텍스트 파일에 원하는 인덱스를 저장해두는 것이다.(즉, 벡터의 인덱스를)

Txt -> BackGround Number : 2

이런 식으로 …

우리는 텍스처에서의 위치 정보를 먼저 읽고나서 이것의 옵션값을 돌려가면서(옵션 값이라고 표현했지만, 수 많은(?) 까지는 아니지만, 추가될 수 있는 무한한 크기의 int형을 enum으로 표현하기는 뭐하기 때문에(나중에 추가되는 텍스처의 가능성을 고려한다) 단순히 int형을 이용해서 옵션을 변경할 수 있도록 할 것이다. 참고로, 우리는 이미 정해져 있는 위치를 읽어오는 방식을 선택했기 때문에, 마우스를 통한 위치 설정 뭐 이런 거랑은 전혀 관계가 없다. 단순히 증가 감소를 통한 선택 Index 변경 정도면 충분하다. (마우스 처리 못하게 막자)

BackGround 는 설정하고, ForeGround의 경우에는 스테이지의 크기랑 연관이 깊으므로 타일의 시작도 여기서부터이다.

스테이지의 크기를 설정하지 않으면(ForeGround) 타일도 없겠지..

자 그럼 Rect를 따는 작업부터 하자.

Texture에 조그맣게 글씨를 써놓자 ! ㅇㅇ 좋아.  
참고로 DDDScene같은 경우에는 리소스가 완전히 분리되어 있어서 따로 사용하는 것으로 한다.

(이건 스테이지 전용이지)

ForeGround :

1. (0, 0, 7920, 1100);
2. (23, 1715, 7300, 2921);
3. 1815 2413 3449 3439
4. 63 1225 1357 1933
5. 63 2001 1363 2703
6. 47 2745 1349 3457
7. 135 3529 1425 4249
8. 1557 3525 2851 4223
9. 2961 3507 4249 4219
10. 143 4287 1425 4969
11. 1559 4265 2851 4977
12. 2959 4251 4251 4971
13. 147 5021 1437 5707
14. 1561 5023 2849 5723
15. 2961 5011 4253 5735

처음에는 그림판에서 위치 하나씩 외워서 적다가, 오캠으로 영상찍어서 바로 좌표를 넣어버리는 내 모습을 보았다. 극한의 효율충.

자 이제 이 정보를 텍스트 파일로 만든다. 다 만들었다. ㅇㅇ

이제 이것을 SceneManager가 ClipCoord의 정보로 가지고 있으면 좋을 것 같다.

m\_vecForeGroundPos

m\_vecBackGroundPos

일단 무서우니까 저장한다.

== BackGround ==

10

src : (15, 85) / dest : (3376, 1163)

src : (1816, 1276) / dest : (3490, 2371)

src : (8059, 543) / dest : (9405, 2383)

src : (4313, 4025) / dest : (9341, 5241)

src : (2843, 3625) / dest : (4203, 5303)

src : (473, 5793) / dest : (9237, 6769)

src : (38, 3493) / dest : (1314, 4387)

src : (1423, 3458) / dest : (2695, 4475)

src : (51, 4551) / dest : (1325, 5539)

src : (1421 4539) / dest : (2689, 5529)

== ForeGround ==

15

src: (0, 0) / dest : (7920, 1100)

src : (23, 1715) / dest : (7300, 2921)

src : (1815, 2413) / dest : (3449, 3439)

src : (63, 1225) / dest : (1357, 1933)

src : (63, 2001) / dest : (1363, 2703)

src : (47, 2745) / dest : (1349, 3457)

src : (135, 3529) / dest : (1425, 4249)

src : (1557, 3525) / dest : (2851, 4223)

src : (2961, 3507) / dest : (4249, 4219)

src : (143, 4287) / dest : (1425, 4969)

src : (1559, 4265) / dest : (2851, 4977)

src : (2959, 4251) / dest : (4251, 4971)

src : (147, 5021) / dest : (1437, 5707)

src : (1561, 5023) / dest : (2849, 5723)

src : (2961, 5011) / dest : (4253, 5735)

BackGround :

1 : (15, 85, 3376, 1163);

2: (1816, 1276, 3490, 2371);

3 : 8059 543 9405 2383

4313 4025 9341 5241

5 : 2843 3625 4203 5303

6 : 473 5793 9237 6769

7 : 38 3493 1314 4387

8 : 1423 3458 2695 4475

9 : 51 4551 1325 5539

10 : 1421 4539 2689 5529

텍스트 파일은 구려서 문자열만 클릭하는 것이 안 되니 비쥬얼을 이용하면 빨리하겠네.

이제 vector를 이용하여 불러오는 함수가 제대로 실행됨을 확인했고 그 와중에 인덱스를 통해서 정보를 받아오는 시스템까지 구현했으니, 이제 모드를 통해서 백그라운드와, Fore그라운드를 설정할 수 있는 기능을 제작한다.

일단, 모드를 만들고 모드에 대한 옵션(정보는 없지만 index를 옮기는 것을 만들고)

근데, 외부에서 접근하는데 이것에 매번 계속 싱글톤 써서 함수 호출하고, 하는 것도 좋은가 ? 안 좋은가 ? 근데, 나는 이거 맵핑해주는 것이 귀찮고 워낙 빈도수가 높으므로 따로 메모리를 잡아서 복사를 하는 방식으로 할 것이다. 여기서 해당 벡터는 한번 추가된 이후로는 다시 처리하지 않으므로 이런 식으로 해도 크게 문제가 생길 것 같지는 않다.

Position 정보를 가져오는 것을 만들었으니, 이제 모드를 생성한다.

하나만 짚고 넘어가자. BackGround는 예외인데, ForeGround 가 없는 상황에서 Tile을 찍을 수 있는가 ? 아니… 그건 막아버려야지… 즉, 우리는 이것을 설정했는지에 대한 여부를 가져온다.

BackGround와 ForeGround가 설정이 되어있지 않은 경우가 있지 않지 친구야 왜냐면 기본으로 인덱스 0을 줄텐데..

초기 상태에 0번에 해당하는 백그라운드와 fore그라운드, 그리고 그것에 맞는 타일의 개수를 잡는다.

변수명 너무 길어서 짧게 바꿨다. 어짜피 의미는 충분하니까. ㅇㅇ 초기 상태를 설정하는 것으로 바꿨다.

이제 입력과 출력에 대해 생각을 해보도록 하자.

잠깐 동안 생각해봤는데 말이야… 이거 지금 계속 잡다가는 뭔가 위험할 수도 있을 것 같다는 생각이 든다는 말이지. 그래서, 적어도 이 정도를 했다고 보여주기 위해서는 (그러니까 보험으로 이 만큼은 해야 한다고…) Stage1 -> 2 -> DDD 까지는 적용을 해놓는 것이 좋아. 여기에 Sound까지..

일단은, 2 -> DDD로 넘어갈 수 있도록 설정을 하고서, 1스테이지의 Door 위치를 다시 설정하고, 사운드를 적용하면서 보험을 걸어두자.

DDD에서 패턴 간격이 1초 정도로 나오는 것은 정말 좋아. 근데, 초기에 시작하는 상황에서는 좀 가만히 서있게 하고 싶거든… 그래서 어떤 방식을 취하면 되냐면 그냥 초기 LimitTime을 크게 주면 쉽게 할 수 있어. ㅋㅋ

그러니까 초기에만 좀 길게 서있다가 패턴은 쭉쭉 나오는 방식이지. 아무튼 수정 가자.

보스 너무 맞기만 하는데 ? 이거 무적시간을 길게 주든지 아니면, 간격을 좁혀.

맞으면 무적시간이 0.7s인데, 맞고서 서있는 시간이 1초임 ㅋㅋㅋㅋ

계속 맞아 아니 ㅋㅋㅋ

0.5s 의 간격을 주는 것으로 했다.

아니 그래도 스워드 너무 쎄…

그러면, 보스만 처리를 다르게 하자. 호출한 객체가 보스 몬스터라면, Blend Time을 길게 주는 것이다. (나머지 Monster의 경우에는 그대로 0.7s로 하고)

pMonster->SetBlend(0.7f, 2);

아니야 됬어, 원래 원작에서도 걔 맞기만 한다구… 굳이 쎄게 해주고 싶으면 HP를 더 크게 주면 되잖아.. ㅠㅜ ㅇㅇ 체력 30 증가 / 간격 시간 0.3s (상향 0.2s 조정)

ㅇㅋ 좋아. 쎄졌네. 쪼금

아무튼, 이제 기본적인 보험Scene은 되었으니 보험의 대표, Sound를 넣어주도록 하자.

하 엔딩 만들어야 하는데 ㅋㅋㅋㅋ

아 내 게임은 스테이지가 많아야 답이 있는데 ㅇㄴㅁㄻㄴㅇ럼ㄴ아럼ㄴ아럼 ㅋㅋㅋㅋ

# 사운드 적용

지금 사운드 적용할 것을 추리고 있는데, LandCollison이 조금 특이한 게 Run상황에서 바닥과 충돌하면 처리를 안 하고, 이 처리는 커비만 한다는 거지.. 딱 한번 출력하고 마는데, 아무튼 선택해서 넣자.

사운드는 모두 구해놓았고, 더 이상의 Sound는 추가하지 않을 것이다.

아무튼, 사운드 매니저를 통해서 하나씩 적용을 해보도록 하자.

참고로 MainScene에서 프로토타입을 지키기 위해서 FindTexture를 했던 것처럼 이곳에서도 FindSound함수를 미리 호출해주는 과정이 있어야 ClearScene을 하더라도 계속 남아있게 될 것이다. 반면 StartScene 에서만 사용하는 Sound라면 장면 전환에서 없어지더라도 상관이 없다.

헷갈리니까, 하나씩 하나씩 위에서부터 처리하자.

MainScene에서 Load해야 하는 사운드 (즉, 프로토타입에 해당하는 사운드를 추가한다)

Scene에서는 사용하는 프로토타입을 각각 들고 있는 시스템이라서, 이것을 계속 생성하고, Texture에 대해서는 참조를 해주는 처리를 하고 있다. 사운드의 경우는 어떤가 ? 프로토타입 같은 개념이 있어서 이것을 날리는 처리를 하고 있는가 ? 아니다. 그냥 맵에서 해당 리소스를 들고 있으며, 이것을 찾아오고 날리는 작업만을 수행하고 있다. 그래서 내가 하고 싶은 말은, MainScene에서 이것을 매번 Load할 필요가 없고, Stage1Scene에서 한번만 Load를 한 다음에, 그 이후에는 SaveSound만 해주면 되는 것이다. 자 그럼 Stage1Scene에서 Sound를 추가해보자.

구현했다.

아무튼, 이제 배경음이 잘 뜨는지 확인해보도록 한다. Music 폴더는 끝났다. ㅇㅋ

스워드 커비에 대해서 하고 있는데 잘 적용되고, 좀 뭔가 부실해보이는 부분에 중복되는 사운드라도 추가하고 있다. ㅇㅇ

점프 힛에서 조금 아쉬움이 있긴 한데 뭐 그렇게 썩은 편은 아니야.. 있는 게 훨 나은 것 같으니까 일단 냅두자.

스톤 커비는 돌로 변하는 상황에서 지형과 충돌했는지 여부(최초)에 따라서 랜덤으로 소리를 출력해주는 것으로 했다. 를 했는데 1개만 못해서 1개로 바꿨다. 괜찮더라고 훨씬… ㅇㅇ

몬스터도 좋아요 !

마이크 커비 ! ㅇㅇㅋㅋ

커비 빨아들이는 것이 조금 문제다. 어느 시점에서 끄고 어느 시점에서 키는지 잘 생각해봐야 한다. 일단, 공격을 시작하면, 한번만 호출해준다.

그리고 그냥 공격을 종료하는 상황 / 먹고 강제로 종료가 되는 상황에서 이것을 빼주는 것이다.

그러면 자연스럽게 처리가 될 것이다.

소리도 너무 커서 줄여줬다.

아까 소리가 너무 컸던 이유가, 볼륨을 50%에 맞춰놓고 아 이 정도면 괜찮은데 이러고 있으니까,

당연히 인게임가서는 너무 컸던 거임.. ㅇㅇ 100%에 맞춰놓고 들으니까, 수치가 통일되서 소리가 큰지 분간이 되기 시작함.. ㅇㅇ 아무튼 끝내줬음. Attack.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 아아아앙아아앙

먼지 소리는 그거 없는게 더 낫다. 그리고 원작에서도 없더라고… 아무튼 제외.

좀 콤보라든지 아쉬운 부분에서 소리를 추가해주고 있긴 한데 시발 이거 언제 끝낼 수 있지 ? ㅠㅠ

커비 사운드 끝.

이제껏 본적도 없는 예외상황이 발생했는데, 커비의 AttackCollision을 가지고 직선이랑 충돌처리됨 ㅋㅋㅋ 개웃기넴 ㅋㅋㅋ

예외 처리로 BodyCollision이 아닌 경우에는 return으로 바꿨다. 해결.

이제, StarBullet에 대해 예외처리를 해서, 시간과, 이펙트가 동일하게 적용해준다.

아니면 그냥 해도 되고, 근데, 굳이 이렇게 예외처리를 해줘야 하나 ? 차라리 float형 변수 하나를 들고 있어서 이것을 얻어오는 것이 훨 낫지 않나 ? 아니 또 귀찮네 그냥 따로 처리하자.

일단, 왼쪽의 상태를 보고 나머지를 적용하도록 한다. 이 이펙트만 적용한 이후에 바로 맵툴작업 수정 들어가자. BACKGROUND, FOREGROUND모드가 있는데, 일단은 FOREGROUND와 TILE의 연관성을 어떻게 처리를 할지 생각해보자.

STAR의 태그를 가지고 있는 것이 오직 VOMIT STAR에만 적용이 되는지 확인해야 한다.

아니면 이름을 바꾸거나…

이름을 바꿔서 처리했으며 이제 이것이 사라지는 시간을 보고서 판단을 해보자.

마지막 이펙트를 넣고서 좀 자야겠다.

백그라운드 모드와 ForeGround모드를 어떤 방식으로 처리할까 ?

BackGround의 경우에는 그냥 적용해도 아무 문제가 없다.

ForeGround를 설정하는 경우는 2가지로 나뉘는데, 편집한 타일이 존재하는지,

그렇지 않은지이다. 만약에 편집한 파일이 존재한다면, 메시지 박스를 띄우고, 다시 새 타일을 생성할 것이라는 문구를 띄워준다. (타일의 개수는 스테이지의 크기에 귀속된다. 즉, ForeGround를 바꾸면, 이것에 따라서 타일의 개수도 달라져야 한다. (물론 그럼에도 불구하고, 따로 처리를 할 수 있지만.)

음 지금, mainScene을 제외하고는 배경음악을 바꾸지 않고 있는데, 나쁘지 않은 선택이다. 그거 되게 몽환적이니까.

Back : 15개 fore : 10개

자료가 많으니까, 2번 3번을 같이 써주는 처리를 하는 것으로 하자.

일단, 출력을 작업하고 있다. 모드의 변경에 따라 지금의 모드를 출력하고, 다른 모드들 처럼 옵션 값을 출력하는 개념이 아니라, 계속 지금 설정한 ForeGround와 BackGround의 인덱스를 보여준다.

우리가 해당 수치를 바꾸고 나서, 타일 편집을 하려고 하는 순간, 이전 스테이지의 사이즈를 받아놓고 이것을 기준으로 현재 스테이지의 사이즈와 비교하여, 다르다면 메시지를 출력하며, 타일에 대한 정보를 다시 입력을 할지 아니면, 말지를 결정하는 것이다.

일단, 키 입력을 통해서 변하도록 만드는 것으로 하자. 3번 4번을 이용해서 움직이도록 한다.

백그라운드의 출력을 변경하는 기능을 만들었으니, 키를 증가시키고 감소시키는 키를 만들어보자.

(입력 처리를 해보자) ㅇㅇ

마우스는 이미했고, 이제 키보드를 통한 변경. ㅇㅇ

이제, Save Delete 개념을 만든다. ESC는 어떻게 할까 ?

마우스 우클릭에 대한 처리도… 인덱스 0으로 ㅇㅇ

솔직히 말하면 별 의미는 없는데 통일성을 유지하기 위해서 ESC도 인덱스 0으로. ㅇㅇ

DELETE도 인덱스 0으로 ㅇㅇ

SAVE는 적용의 개념을 도입할 것인데, BackGround에서는 그냥 값만 바꾸면 이것이 적용이지 굳이 뭘 선택해서 enter를 누르고 이것이 적용이 되는 개념이 전혀 아니다. (처리를 안 한다.) ㅇㅇ

ForeGround는 타일의 적용처리를 한다. 현재 보여주는 텍스처에 대한 타일의 적용. ㅇㅇ

근데, ForeGround같은 경우에는 이걸 적용의 개념을 섞어서, 타일에 적용을 할지 여부를 의미한다.

ForeGround의 적용은 2가지의 의미가 있는데, 일단, 스테이지 사이즈로서의 적용 (Tile) 텍스처로서의 적용이 있다. (인덱스 기본) 우리가 그냥 바꿔주는 것은 Texture의 위치 그리고 타일은 enter의 여부에 따라서 적용을 해주는 것으로 하겠다. ( 적용 Enter 기능) ㅇㅇ

Esc를 누르면 초기상태로 변하는 것으로 했다. ㅇㅇ 근데, 여기서 타일의 적용여부 즉 enter는 어떻게 처리가 되어야 하는가 ? 지금 현재의 인덱스 값으로 타일이 적용되어야 할 것이다. 여기서 이 작업은 위험을 내포하므로 메시지 박스는 필수이다. (근데 그게 ESC랑 무슨 상관이라고 이렇게 써둔거임 ? …) 아무튼 나중에 적용시키자. ㅇㅇ

자 이제 위의 기능들을 한번 확인해보자.

ESC 초기상태로 ㅇㅇ / Delete ㅇㅇ / Enter 기능. (Back은 적용이 없고, Fore의 경우에는 보여지는 ForeGround Textrue에 대한 TileSet 생성) ㅇㅇ

ESC나 Delete 키를 통해서 초기 상태로 이동하는 상황에서는 보험을 걸어주는 것으로 했다.

근데, 우클릭도 쓰지 않나 ? 우클릭도 같은 처리를 하는데, 이것에 대해서 메시지 박스를 호출하여 보험을 걸어주는 처리를 했다.

이제, 만든 것을 확인해보자. 옵션을 변경하는 함수를 등록할 것이다. 즉, 옵션정보를 하나씩 뒤로 미루는 것이다. (지금보니까, 중요해서 ㅋㅋㅋ) 모드에 대해서 이 처리를 한다.

그거 꼭 필요한 부분아니야 나중에 고쳐도 된다.

지금은 스테이지를 뽑아낼 시간이야. 아무튼, 우리는 텍스트를 통해서 스테이지 파일을 만들 것인 것, 이것을 읽어오고 저장하는 시스템이 아직 구현이 되지 않았다. 따라서, 벡터의 인덱스를 저장하는 기능을 만든다. BackGround : %d ForeGround : %d 이런 식으로 ….

자 그러면 이것을 읽어오는 시스템을 구현한다.’

현재 구현 : Stage 1 (하드코딩)

Stage 2

Stage 3

DDDScene

Boss 죽는 사운드랑, Ending Music

스테이지 5까지 모두 추가했다. 엔딩 장면에서 노래한번 출력하고 마는 것으로 하자.

만들면서 신경을 써줘야 하는 정보…

1. BackGround / ForeGround 설정 그리고 이것을 토대로 타일 편집.
2. 스포너 정보
3. 라인 정보
4. 위치 정보 Door StartPos

1

장면을 적용할 때는 읽어온 데이터를 기반으로 LimitCamera / StageSize 등을 설정하며 Speed는 따로 처리를 해주자. (일단은..) 일단 스테이지 쪽에서 읽어오는 시스템을 구현. ㅇㅇ

제대로 나오는 것을 확인했다. 이제는 여러 스테이지를 추가하면서, 장면을 생성해보자.

지금은 현재장면의 Tag를 가져와서 노래를 다시 복구하는 짓을 하고 있는데 이것보다는 string 으로 설정을 해두고 처리하는 것이 훨씬 낫다. 바로 하자.

스테이지마다 기본적으로 호출해야하는 Bgm이 달라진다. 아니면 MainScene에서 해도 되는 부분이고, Stage 마다 자신의 기본 브금을 설정할 수 있는 기능을 만든다. 그리고 Init부분에서 이것을 처리해주면 그만이다. ㅇㅇ 확인.

새로 생성한 스테이지 3를 확인한다.

보스랑 싸우는데 보스 이펙트가 너무 부실하다. 매우 아쉬운 부분임 빨리 고치셈. 이거 몇 분 안 걸림.

사운드를 다른 것을 넣고 싶기는 한데, 어디서 ㅇ 구했다 빠르게 장면 만들고 적용하즈아 !

브금의 경우에는 한번만 쓰고 버리므로 그냥 Load 하면 된다.1

# 스테이지 추가 ㅇㅇ , 오브젝트 추가(디테일 ), DOOR 추가 dd

# 스테이지간 연결. (Door 생성) OK

문에다가 충돌체를 하나 두고, 이것의 텍스처만 지정하면 충분하다. 여기서 이펙트의 타겟을 잡아서 처리를 하는 것으로 하자. 그런데, MoveStar, JumpStar, Bullet과는 다르게 충돌체를 가지지 않아야 한다. 단순히 애니메이션만 Loop로 돌려주어야 한다. 아무튼, 기능은 Door쪽에서 구현한다.

충돌체를 가지지 않는 애니메이션을 복사해서 생성해올 (아니, 이게 필요한가 ? 그냥 장면에서 1,2 개만 추가해놓고 이후에는 사용하지 않는 오브젝트가 아닌가 ?)

그래도 여러 번 사용할 가능성이 있으므로 미리 프로토타입을 만들어주고, 이것을 복사해서 사용하도록 하자. 중간에 오브젝트가 추가되는 경우는 없다. 단지 맵 에디터에서 사용하기 위한 프로토타입이라고 할 수 있다.

아무튼, Door 클래스를 제작하고, 별 애니메이션만 들어있는 이펙트를 생성한다.

UI오브젝트의 경우에는 한번만 리소스를 읽어오면 되므로 Init에서 처리를 했지만, Door 클래스의 경우에는 따로 적용이 필요하다.

Init에다가 따로, Init처리를 하더라도 문제는 없는 것이 같은 이름의 텍스처가 이미 존재한다면 그것을 읽어오지 않고, 그냥 참조를 하니까. Init에서 처리를 해도 좋겠다.

오브젝트를 복사하는 상황에서는 해당 오브젝트의 복사생성자를 호출한다. 즉, Init함수를 호출하는 것이 아니다. 우리는 이것을 기반으로 생각하며 뇌절을 방지하자.

미리 연결되는 함수를 만들어두고, 이것을 연결하여 연결 장면을 골라두는 것으로 하자.

Door 클래스를 복사해서, 충돌체까지 복사를 해둔 상태로 ㅇㅇ, 어느 장면으로 이동을 시킬지 tag를 등록하고, 위치만 셋팅해주면 일단 뭐든지 될 것 같다.

동아리 시간에 사운드 작업을 하는 것으로 하자.

배경음악과, 효과음은 어느 정도 뽑아 두었고, 연결 하는 작업은 다 보스까지 만들고 나서 진행하도록 하자.

// NOT

# 풀 애니메이션 ㅇㅇ ( 수정 필요함 살짝 뜸… 디테일)

# 이펙트 적용 (사망, 피격) ㅇㅇ

# 몬스터와의 전투( 데미지 개념 추가 ) ㅇㅇ

# 흡입 애니메이션 스킵 ?! (디테일) ㅇㅇ

# 애니메이션 잔상 처리 (디테일)

# 장면 전환 OK

일단, 커비와 UI 오브젝트에 대하여 Ref 값을 미리 증가를 시켜둔 상태이다. 그렇다면 다음으로 무슨 처리가 있어야 할까 ?

일단 이렇게 처리를 한 이후에 다음 장면에 대한 처리를 하기 시작할 것인데, 스테이지의 타일과, Door 정보는 기본이고, 잡다한 오브젝트를 모두 설정해주어야 하며,

이 오브젝트들은 기본 오브젝트 생성값 + 연결 참조값(신경 쓸 부분은 아니다) + 1의 ref 값을 가지게 된다. 그렇다면 이전에 생성되었던 오브젝트는 SceneChange함수를 통해서 사라지는 처리가 되고,

충돌 관리자에서는 이전에 충돌이 되었던 상태의 물체들을 모두 날려주는 처리를 하고 있다.

이전 충돌에서 남아있던 오브젝트에 대한 참조도 풀어버리는 처리를 하고 있다. (여기서 참조값이 줄어들게 되는 것)

다음 장면을 위한 충돌 오브젝트의 경우에는 어떻게 되는가 ? 충돌 관리자에서 오브젝트를 처리하는 부분이 어떻게 되는지 묻고 있다.

일단, 다음 장면이 생성이 되면, 타일 충돌을 하기 위한 오브젝트를 바로 넣어주지는 않는다.

이전에 타일 충돌에 사용하던 오브젝트들은 충돌 관리자에서 참조를 풀어버리고, 그 결과 이전 오브젝트는 사라지게 된다. (커비 + UI를 제외하고는…) 이 상태에서 새로운 오브젝트가 추가되고 추가된 오브젝트는 ref = 1인 상태가 되는 것이다. 이 상태로 리소스 관리자에 의해서 ClearScene함수를 호출하지만, 이미 해당 텍스처를 사용하는 오브젝트가 남아있기에 사라지지는 않는다.

장면에서 사용하는 오브젝트들을 담아두는데, 아마, 새로 생성하는 오브젝트의 경우에는 SetWorld함수를 통해서 배치가 되는데… WorldObjList에 배치가 된다. 우리는 ObjList를 통해서 새롭게 생성된 오브젝트를 찾아올 수 있게 되는 것이다. 그리고, WorldObjList에 배치가 되면서, SetWorld (Start 함수 / Layer 추가)함수를 호출하게 되는데, 여기서 직접적으로 레이어에 존재하게 되는 것이다.

그러면 새로운 오브젝트의 경우에는 생성이 되어 ref = 1인 상태로 장면의 ObjList에 들어가 있게 되고 Star함수를 통해서 새로운 장면에 해당하는 오브젝트가 충돌 관리자의 손에 들어간다.

동시에 Layer에도 등록이 되는 개념이다.

기존의 오브젝트의 경우에는 일단, 참조 값을 증가시켜 둔 상태라서, Layer에는 그대로 들어가 있는 상태이다. 그런데, 장면이 바뀌어서 ObjList에는 들어가 있지 않는다. (문제) 이 상태로 당연히 WorldObjList에도 들어가지 않아서, Star함수를 호출하지 않는다. (따라서, 충돌관리자에 대한 정보도 들어가지 않는다)

후훗 내 예상대로 역시 장면이 달라지는 이유에서 ObjList에서 오브젝트를 찾아올 수 없는 현상이 일어나고 기존에 있던 오브젝트가 충돌관리자에 들어가지 않는 현상이 일어나게 되는 문제가 방생한다. 따라서 그냥 떨어지게 되는 것이다.

이 문제를 해결하기 위해서는 기존의 오브젝트에 대해서 새로운 장면을 통한 AddObject함수를 호출하면 될 것이다.

장면 전환은 아무 문제 없이 잘 된다. 근데, 이전 장면과 프레임 차이가 70정도 차이가 나게된 경우에 시간 동기화에서 터져버리는 문제… 크.. 망했구나 ㅋㅋㅋㅋㅋ

이거 프레임을 120정도로 고정시키고 처리를 하는 것으로 해야 겠다.

만약 1초에 120장을 그려야 한다면 … 1프레임 당 시간은 …. 1/120s

만약 1/120s 의 시간이 지나지 않은 경우에는 ? 처리를 하지 말아야 한다. 즉, 0s로 넘겨야 한다는 것.

아무리 봐도 이상한 것이 2번씩 처리가 되는 것이 정말 이상하다. ㅋㅋㅋ 로직에서 문제가 있는 것이 아니라 오브젝트라도 2개가 들어간 것 같은 느낌이다..

어, 진짜네 ㅎㅎ

띠용

현재 살아있는 오브젝트가 2개가 들어가는 현상이 발생한다. 분명 어디서 틀려 먹은 것은 확실한데 나는 아마도 Scene을 설정하는 부분에서 이미 살아있는 오브젝트들이 새로운 Scene으로 갈아탔는데, 이 상황에서 (없어지지 않을 것인데…) 여기에서 또 넣어준 것이 아닐까. ..

디버깅으로 확인을 해보았는데, 안 들어온다… (즉, Layer에서 해당 오브젝트가 사라지고 있지 않다는 소리가 된다…)

CurrentScene에서 있던 오브젝트들이 이미 소속이 다음 Scene으로 되어 있는 상황이 된 것이 아닐까 ? 이런 상황에서 Layer에 남아 있는 이전 장면 소속인 친구들을 날려주니, Kirby / UI들은 그대로 남아있고 이런 상황에서 AddObject 함수를 통해서 또 오브젝트를 넣어준 것이다.

자 보니까, AddObject 함수를 호출하는 순간 SetScene을 호출을 하고, 그러니까 구체적으로 말하자면, AddObject -> AddRef -> Layer에서 Release & Erase -> SetWorld함수를 통한 Layer배치, CollisionManager에 등록

해결..

왜, 리소스가 다 날라갈까 ? 이전에 사용하던 리소스가 있는데, 이것이 날라가는 이유는 ref 값의 문제이다. 즉, ClearScene에서 문제가 생기는 것으로 보이는데.

새로 추가가 되는 리소스의 경우에는 ref = 2라서, 아무 문제 없이 처리가 이루어 질 것이다.

Blue Ball을 보면서 원인을 확인해보도록 한다.

초기 생성 1 (맵) -> 클립이 참조 1 ㅇㅇ-> // clear scene -> 장면 1 -> (제일 처음)

Find -> rel // 클립이 참조 1 ->

Scene이 사라지면서 Prototype을 소멸… 애니메이션이 사라지면서 1감소 ㅇㅇ

Clear scene -> prototype save //

여기서 숫자 안 맞기 시작하쥬…

프로토 타입 없애면서 애니메이션 사라짐. – 1

프로토타입의 경우에서 텍스처에 대해서 생각해보자. 애니메이션도 있을 가능성이 있다.

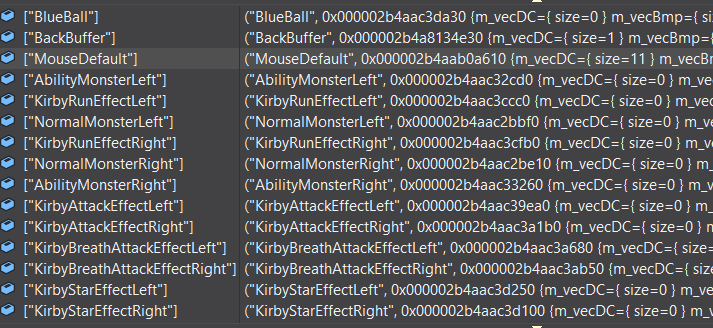
일단, 하나만 있는 경우. 둘 다 있는 경우.

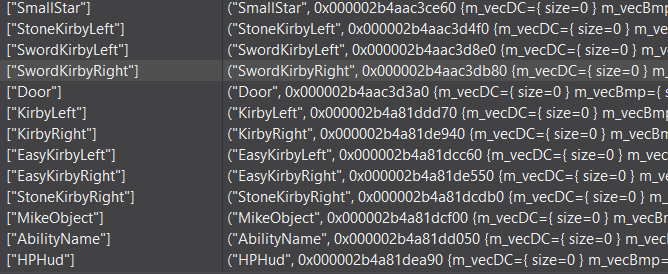
하나만 있는 경우에는 하나의 것을 추가해주면 그만이다. 하 너무 힘들어져서 텍스처에 변수 하나 넣는 것으로 했다.

IsPrototype 프로토타입이 아닌 물체에 대해서만 처리를 하도록 할 것이다. 즉, 기본값은 true.

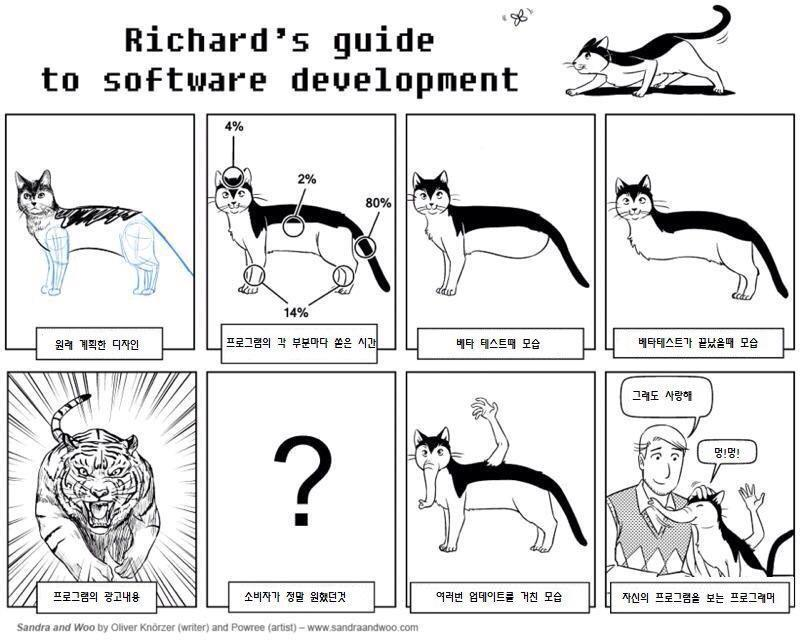
결론은 이미 있는 텍스처라고 하더라도, clearScene을 통해서 참조를 감소시켜서는 안 된다.

아니 근데, 처음에 넣는 경우에도 이런 식으로 처리가 된다. 문제는 같은 텍스처를 돌려서 쓰는 경우가 많은데, ㅋㅋㅋㅋ 이쪽에서 clearScene을 대비해서 참조를 한번 더 해준다. 근데 저쪽에서도 참조를 더 해주고 ㅋㅋㅋㅋㅋ 그냥 하드코딩 하는 게 마음 편하겠다.





하 ㅋㅋㅋㅋ 진짜, 화난다.



# 9월 3일 시연회 이후

맵툴에서 오브젝트의 위치를 바꾸는 기능은 만들지 않을 것이다. 이 작업을 굳이 하지 않아도 충분히 분량은 나왔다고 판단된다. 마지막으로, Stone 상태와 Mike 상태에서 생기는 오류(무적상태를 해주지 않는 점)에 대해서 고치고 ㅇㅇ , 보스와 싸우는 부분에서 여러 이펙트를 추가하는 것으로 마무리를 하려고 한다. 마지막 Ending 부분에서 마지막 띄워주고 노래 틀고 마무리하자.

보스가 점프한 이후에 떨어지면 뿍. ㅇㅇ

별맞으면 팎(크게)

망치로 때리면 작게 빡 ㅇ ㅇ

내 로직에서는 Idle, Walk, Run, JumpAttack, Attack(Hammer)

일단, EndingScene을 만들어준다. ㅇㅇ

보스전에서 사용하는 이펙트에 대한 소리는 다 넣어주었고, EndingScene에서 사용할 노래도 적용했고, 장면 전환 연결만 하면 된다.

학원가서 마무리 테스트..ㅇㅇ

엔딩씬에서는 StarObject와 충돌이 일어난 경우에 아 귀찮네 그냥 하지 말까.. ㅋㅋㅋㅋ

스타 이펙트의 힘과, 속도, 혹은 개수만 다르게 해줘도 멋질 것 같은데 ?

속도와 스피드를 랜덤으로 줘서 랜덤한 크기의 원을 그리도록 하고, 또한 플레이 시간도 랜덤으로 설정해줘서 규칙적이지 않을 것이다. 굿굿.

나쁘지 않다.

보스를 이긴 이후에 문을 생성해준다.

엔딩 씬에서는 문을 생성하지 않는다.

보스에서 출력할 문의 위치를 정한다.

위에 써둔 각 상황에 맞춰서 이펙트를 호출하게 해뒀다. 이제 엔딩에 관한 처리를 하려고 한다.

보스가 사라지면 특정 자리에 문을 주고서 이곳에 들어가면 Ending Scene이 나오도록 할까 ?

괜찮지 브금은 새로운 장면에서 호출해주면 되고, 그러면, EndingScene을 만들어보도록 하자.

근데 굳이 Ending Scene을 뻔하게 만들어줘야 하나 ? 남들 쉽게 하듯이 그냥 사진 한 장 띄우고 감사합니다… 너무 뻔한데..

근데 이 브금은 절대로 끊기는 로직을 작성하지 않을 것이고 (몬스터에 의한 사망효과가 절대 나오지 않도록 한다.) 거기서 이펙트 개념을 섞어서 막 멋나도록 ㅇㅇ

그러면, 나는 장면을 하나 생성해서 (Stage랑 동일하다) 이곳에서 팔짝팔짝 뛰어다닐 수 있도록 하고, 특정 부분을 밟으면 이펙트가 팡팡 터지도록 하는 시스템 그런 것들을 만들면서 그것을 엔딩 씬으로 하는 것으로 하자. 한번만 노래를 호출하면 되는 거니까 !

보스를 없앤 이후에 Door를 생성해주면 즉시 생성하지 않는다. 단순히 저장만 해둔 상태로 처리하면 된다. 그냥 하드코딩 한번 하자.

사운드는 나중에 확인을 하자.

간단하게, Mike부터 수정할 것인데, 마이크가 공격을 하는 상황에서 SetColliderState함수를 이용해서 비활성화 상태로 만들어주고, 이후에 마이크의 공격이 끝나는 시점에 (MikeObject를 가지고 판단할 수 있다.) 다시 true 상태로 만들어 줄 것이다.

커비의 충돌체가 4가지가 있는데, 첫번째로 기본 Body충돌체

왼쪽과 오른쪽 사거리를 표시하는 기본 커비 충돌체가 있다.

점프 공격(Sword)에서 사용하는 스워드 커비 충돌체.

이렇게 4가지의 충돌체가 있는데, Attack의 경우에는 기본 커비가 아니면 처리를 꺼버리나 ?

아니면 아예 막아버리나 그렇고.. ( 내부적으로 막았네 )

점프 공격 충돌체의 경우에는 특정 상황에서만 키고 끄는 거라 문제가 없고, 기본커비의 두 충돌체도 그렇다.

하나만 기본적으로 계속 충돌이 되는 것이다 따라서 이것만 처리하면 될 것이다. 확인.

참고로 이 현상은 무적상태로 마이크 공격을 시작한 경우에 발생한다.

문제는 움직이지 못하게 묶여있는 시간보다 적게 충돌을 막아주는 처리를 하여, 시간이 잘 맞지 않는다. 따라서 커비쪽에서 따로 구현을 해보도록 하자.

커비에서 따로 마이크 공격의 중력을 꺼버리는 처리를 하는 곳이 있는데, 여기서 잠시 멈춰있는 처리가 끝나고 나서 다시 Stop을 true 중력도 키고 하는 부분이 있는데, 이 상황에서 다시 충돌체를 true로 주었다.

그렇다면…. 공격을 시작하는 상황에서 State = false -> 정지시간 -> 정지 시간이 끝나고 true

그러면 정지 시간 동안은 false인 상태로 있기 때문에 충돌처리가 되지 않는다. 따라서 중간에 충돌이 일어나는 상황은 없을 것이다.

문제가 중력이 false인 상태로 넉백 / 넉업 처리를 진행하는 방식이다. 이 상황을 잘 생각해보면, 분명 넉백과 넉업에 대한 처리를 하는 과정에서 Stop 시간을 이용하고 있다. 즉, 이쪽에서도 사용하고 있고 저쪽에서도 사용하고 있어서 로직이 꼬일 것이다. 예상이 되는 상황은 지금 마이크 공격에 의해서 정지가 되어있는 상황에서 충돌처리가 시작된다. 그러면서 충돌처리 로직이 먼저 끝나지는 않을 것 같고 아마 아니지 먼저 끝났으면, false상태로 중력의 처리를 했겠구나…

아무튼 이 상태로 충돌처리 로직이 진행이 되어서 충돌처리에 사용하던 stop 변수(같은 거임 ㅇㅇ) 를 false로 만들어버리는 현상이 생기게 된다. 즉, 마이크 공격 시간 이후에 중력을 꺼버리는 현상은 사라지는 것이다 … 그러면서 중력에 대한 처리가 계속 false 상태였던 것.

이런 상태였는데, 이렇게 정지 시간인 상황에서 충돌처리를 할 수 없도록 만들어서 무조건, Mike의 정지시간이 끝난 이후에만 다시 충돌처리를 하도록 만들었다. 즉, 위에서 문제가 생겼던 부분 Mike공격 중에 충돌이 일어나는 현상은 사라지게 된 것이다. 따라서 자연스럽게 문제가 생기지 않게 되었다.

그런데 이 현상이 Stone에서도 일어났던 것 같다. 확인해보자. 안 보임.. 뭔지 잘 모르겠음

커맨드 만지다가 재밌는 상황을 보게 되었는데, 스톤이랑, Sword 상태에서 슬라이드 하면서 공격을 할 수가 있다. ㅋㅋㅋㅋㅋㅋ 이런 상황 처음보는데 ㅋㅋㅋ

스톤커비로 달려가면서 돌로 변하는거 가능.

스워드 커비 공격 모션 스킵하면서 빠르고 기묘한 이동처리를 받을 수 있음 (기존의 로직에서 부스트 효과를 받는 것 같다. )

이 부분은 수정할 것인데 재밌으니까 한번 찍어두려고 한다. 그냥 냅둘까 히든 커맨드처럼 ㅋㅋ

이 부분 수정해도 크게 달라질 것 없는 부분이기도 하고, 냅두는 것도 꽤나 재밌을 것 같으니까 (수정했다)

9월 7일

위에서 해둔 작업, DDDScene 사운드 작업을 마무리했고, EndingScene에 대한 것도 마무리 했다.

이제 점프중인 상황에서 타일에 막히는 것을 풀어보도록 하자.

근데, 이걸 하면, 일반 스테이지의 경우에는 점프UP상태라고 하더라도 올라가면 안 되고,

단순히 중간 점프 지점인 경우에는 JUMPUP상태에서 올라가야 하는데, 이걸 구분하기가 애매하다.

이걸 처리를 하려면 Tile의 옵션을 바꿔서 이것은 점프업 상태에서 올라갈 수 있는 타일이라고 인식하고 내려가는 상황에서만 충돌처리를 하면 되지만, 그런 식으로 타일을 추가하기에는 너무 귀찮은 점이 많다. 그래서 그냥 하지 않을 것이다.

타일로 충분히 경우를 나눠서 처리할 수 있지만, 귀찮으므로 패스…

스포너랑 일정 거리 이상일 때만 생성하도록 하자.

X거리 Y거리를 따로 둬서 처리했다.

그냥, 생성을 다시 하려고 하는 상황에서 플레이어와의 거리가 일정 거리 이하라면 처리를 하지 않는 것으로 하자.

그냥 일정한 조건 (카메라)에 의해서 생성이 되는데 이것이 무너지는, 점프 상황에서 이것이 깨지니까, 이것을 인식하도록 하면 되는 것 아니야 ? 점프 중인 경우에 생성 X, 즉 걷거나 뛰어오는 경우에는 생성하지만, 점프하는 경우에는 생성하지 않는다.

일단, 거리 조건은 무조건 하고, 점프하고 있는 경우에는… 막아야 하나 ? ㅇㅇ

단순히 점프를 하는 상황에서 막는다… 이거 정말 안 좋은 방식이다. 처리를 나누는 것이 좋을 것 같다.

X 거리 따로, Y거리 따로…

x거리에 대해서는 처리를 하지 않았고, Y거리가 300.f 이상인 경우에만 처리를 하는 것으로 바꿧다 점프랑은 상관없이.

카메라보다 더 넓은 위치를 이용하여 미리 생성을 하고(원래 원작에서도 미리 생성하더만..)

인식의 범위가 커졌기 때문에, 그 안에서 점프를 한다고 해서 카메라 충돌이 되어서 생성이 되지는 않을 것이다.(범위가 커지니까) 즉, 확실하게 바깥으로 갔다가 다시 온 경우가 아니라면 생성하지 않는 시스템인 것이다. ㅇㅇ

맞은 경우에 공격 소리가 계속 되는 오류를 없앴다. ㅇㅇ

별을 뱉는 도중에 입김 처럼 입력을 잠시 동안 막는 처리를 통해서 별을 뱉으면서 달리는 것과 같은 이상한 처리를 막아버렸다. ㅇㅇ

그러면, 생성을 하고서 제한 시간을 주는 것은 어떨까 ? 아니면, 범위를 넓게 주든지.

Y축에 대해서만 카메라 보다 넓은 범위를 준다면, 미리 생성을 하고, 내가 점프를 한 이후에도 (이미 충돌이라고 인식을 하기 때문에) 더 이상 생성을 하지 않을 것이다.

죽은 상황에서 스테이지의 처음 위치로 이동하는 것으로 했다. ㅇㅇ

나름 원했던 처리대로 되니까 그만 하겠다.

이제 입력 막는 처리. 마우스의 위치에 따라서.. 마우스의 위치를 파악하는 코드를 어디에 작성할까…

인풋에서 계속 업데이트 하도록 하고 이것을 얻어올 수 있도록 만들어서 여러 부분에서 사용하면 좋을 듯하다.

근데, 이걸 하나하나 적용을 하다가 생각이 든 건데… 마우스가 내부에 있지 않은 경우에 따로 처리를 막고 있잖아… 봐봐 함수를 등록을 했단 말이지.. 함수랑 입력을 같이 등록했지.. 그렇다면 애초에 입력의 업데이트를 막아버린다면 아무 문제도 없는 거 아님 ? 마우스가 이동을 하는데, 애초에 이동을 하기 위해서는 입력이 들어와야 하는 건데, 이것을 수동으로 막고 있다고, 인게임. 마우스가 바깥에 있는데 입력처리 할 필요가 없고, 맵툴에서도 그렇고, 애니메이션 툴에서도 그렇다. 이러면, 그냥 한번에 처리를 해버리면 그만 이잖아. 바깥에 있네 ? 그러면 다른 처리를 하고 있다고 인식하고 입력을 다 막아버리면 그만인데, 이걸 왜 이렇게 힘들게 했는지….

수정 했다. ㅇㅋ 소스코드에서 주석 같은 것은 그냥 둘거고… 파일 다했고… 정리 다 했고…

이제 영상 찍고 마무리 하면 될 것 같다.

이제 모든 작업이 끝났다. 따라서, 포트폴리오 문서화를 하기로 하자. ㅇㅇ 정리했다.

모든 문제 해결 ㅇㅇ

커비 변신 키는 막지 않도록 한다. 그 외 잡다한 키들도.. 그냥 냅둔다. ㅇㅇ

Warning이 더러워서 잡아주고 있다. (대부분 출력에서 좌표가지고 연산하다가 짤리는 경우이다.)

어짜피 짤릴 부분을 미리 짤라주는 처리를 하고 있다. 컴파일러에게 이 부분은 문제가 없으니까 넘어가라는 형변환의 표시를 주는 것. 죄다 이런 것뿐이라 문제가 터질 부분이 없다… ㅇㅇ

아무튼, 릴리즈에서 타일 출력 잘 되는지 확인하고 ㅇㅇ 바로 영상 찍는 걸로 하자 ㅇㅇ.

이제 영상을 찍어보도록 하겠다.

아니 애니메이션 기능이 뭐였더라 ㅋㅋㅋㅋ

여기에 적어놓겠다.

Map Tool

// Camera Move

GET\_SINGLE(CInput)->AddNormalKey("D", 'D');

GET\_SINGLE(CInput)->AddNormalKey("W", 'W');

GET\_SINGLE(CInput)->AddNormalKey("A", 'A');

GET\_SINGLE(CInput)->AddNormalKey("S", 'S');

// 속도 증가 감소.

GET\_SINGLE(CInput)->AddNormalKey("F1", VK\_F1);

GET\_SINGLE(CInput)->AddNormalKey("F2", VK\_F2);

// RENDER

GET\_SINGLE(CInput)->AddNormalKey("F3", VK\_F3);

GET\_SINGLE(CInput)->AddNormalKey("F4", VK\_F4);

GET\_SINGLE(CInput)->AddNormalKey("F5", VK\_F5);

// Edge Block

GET\_SINGLE(CInput)->AddNormalKey("F6", VK\_F6);

// Line Edit

GET\_SINGLE(CInput)->AddNormalKey("DeletePos", VK\_BACK);

GET\_SINGLE(CInput)->AddNormalKey("SavePos", VK\_RETURN);

// Spawner

GET\_SINGLE(CInput)->AddNormalKey("3", '3'); // Dir

GET\_SINGLE(CInput)->AddNormalKey("4", '4'); // Count Switch

GET\_SINGLE(CInput)->AddNormalKey("5", '5'); // ++

GET\_SINGLE(CInput)->AddNormalKey("6", '6'); // --

// Bind Key

GET\_SINGLE(CInput)->AddActionKey("EditModeNext", '1');

GET\_SINGLE(CInput)->AddActionKey("EditOptionNext", '2');

GET\_SINGLE(CInput)->AddActionKey("Save", 'S', ST\_CTRL);

GET\_SINGLE(CInput)->AddActionKey("Load", 'L', ST\_CTRL);

GET\_SINGLE(CInput)->AddActionKey("ESC", VK\_ESCAPE);

Animation Tool

// 애니메이션의 속도를 변경하는 키를 만든다.

GET\_SINGLE(CInput)->AddNormalKey("--FrameTime", VK\_OEM\_MINUS);

GET\_SINGLE(CInput)->AddNormalKey("++FrameTime", VK\_OEM\_PLUS);

// 애니메이션 교체.

GET\_SINGLE(CInput)->AddNormalKey("++Animation", VK\_F5);

GET\_SINGLE(CInput)->AddNormalKey("--Animation", VK\_F6);

// Camera Move

GET\_SINGLE(CInput)->AddNormalKey("D", 'D');

GET\_SINGLE(CInput)->AddNormalKey("W", 'W');

GET\_SINGLE(CInput)->AddNormalKey("A", 'A');

GET\_SINGLE(CInput)->AddNormalKey("S", 'S');

// 속도 증가 감소.

GET\_SINGLE(CInput)->AddNormalKey("F1", VK\_F1);

GET\_SINGLE(CInput)->AddNormalKey("F2", VK\_F2);

// 현재 활성화된 프레임으로 크기를 고정시킴. 여러개가 잡혀있는 경우에는 처리하지 않음.

// 이렇게 설정한 이후에 이전에 잡아둔 모든 프레임 정보가 프레임에 맞게 수정됨.

GET\_SINGLE(CInput)->AddNormalKey("F3", VK\_F3);

// Save Option Change

GET\_SINGLE(CInput)->AddNormalKey("F4", VK\_F4);

// Once Loop Begin Frame

GET\_SINGLE(CInput)->AddNormalKey("F7", VK\_F7);

// reset

GET\_SINGLE(CInput)->AddNormalKey("F8", VK\_F8);

// Offset Copy & paste

GET\_SINGLE(CInput)->AddNormalKey("5", '5');

GET\_SINGLE(CInput)->AddNormalKey("6", '6');

// Brush Color

GET\_SINGLE(CInput)->AddNormalKey("7", '7');

// Offset Mode

GET\_SINGLE(CInput)->AddNormalKey("8", '8');

// Static State (X, Y 사용빈도 떨어짐)

GET\_SINGLE(CInput)->AddNormalKey("9", '9');

// Frame Adj Enable

GET\_SINGLE(CInput)->AddNormalKey("0", '0');

// Overlap Collision Enable

GET\_SINGLE(CInput)->AddNormalKey("Q", 'Q');

// all Delete

GET\_SINGLE(CInput)->AddNormalKey("ESC", VK\_ESCAPE);

// COORD KEY

GET\_SINGLE(CInput)->AddNormalKey("CoordDelete", VK\_BACK);

// Coord Move & Resize Keyboard

GET\_SINGLE(CInput)->AddNormalKey("Right", VK\_RIGHT);

GET\_SINGLE(CInput)->AddNormalKey("Left", VK\_LEFT);

GET\_SINGLE(CInput)->AddNormalKey("Top", VK\_UP);

GET\_SINGLE(CInput)->AddNormalKey("Bottom", VK\_DOWN);

// Coord Move Speed

GET\_SINGLE(CInput)->AddNormalKey("++MoveSpeed", '1');

GET\_SINGLE(CInput)->AddNormalKey("--MoveSpeed", '2');

// Coord Move Mouse

GET\_SINGLE(CInput)->AddNormalKey("E", 'E'); // Pos

GET\_SINGLE(CInput)->AddNormalKey("R", 'R'); // Size

// EditCoord clear

GET\_SINGLE(CInput)->AddNormalKey("F", 'F');

// ex : E + 키보드 위치이동.

// E + 마우스 위치이동

// R + 키보드 크기변경

// R + 마우스 크기변경

// Most Newly Added Coord Revive

GET\_SINGLE(CInput)->AddNormalKey("CtrlZ", 'Z', VK\_LCONTROL); // DELETE : 가장 최근 좌표를 없앤다.

// 근데 해당 좌표가 여러개일 가능성이 있으므로 vector로 받아둔다.

// 저장하기

GET\_SINGLE(CInput)->AddNormalKey("CtrlS", 'S', VK\_LCONTROL);

// 불러오기

GET\_SINGLE(CInput)->AddNormalKey("CtrlL", 'L', VK\_LCONTROL);

// 편집 프레임 붙여넣기

GET\_SINGLE(CInput)->AddNormalKey("CtrlV", 'V', VK\_LCONTROL);

// 모든 프레임 편집모드 선택.

GET\_SINGLE(CInput)->AddNormalKey("CtrlA", 'A', VK\_LCONTROL);

// Frame 간격 설정.

GET\_SINGLE(CInput)->AddNormalKey("++FrameAdjX", VK\_NEXT); // x 증가

GET\_SINGLE(CInput)->AddNormalKey("--FrameAdjX", VK\_DELETE); // x 감소

GET\_SINGLE(CInput)->AddNormalKey("++FrameAdjY", VK\_HOME); // y 증가

GET\_SINGLE(CInput)->AddNormalKey("--FrameAdjY", VK\_END); // y 감소

// 애니메이션 프레임 넘기기. 보통 P(정지) 상태로 사용할 예정

GET\_SINGLE(CInput)->AddNormalKey("3", '3'); // Next

GET\_SINGLE(CInput)->AddNormalKey("4", '4'); // Prev

// 변환 모드 활성화.

GET\_SINGLE(CInput)->AddNormalKey("ConversionMode", 'Y');

// 사본저장.

GET\_SINGLE(CInput)->AddNormalKey("ClipCut", 'U');

// 사본 붙여 넣기. Reverse

GET\_SINGLE(CInput)->AddNormalKey("ClipCopy", 'I');

// 사본 붙여 넣기 NOT Reverse

GET\_SINGLE(CInput)->AddNormalKey("ClipCopy2", 'O');

// 편집하는 오브젝트 변경.

GET\_SINGLE(CInput)->AddNormalKey("ObjectChange", VK\_TAB);

// Size 변경

GET\_SINGLE(CInput)->AddNormalKey("Shift", VK\_SHIFT);

// 출력 위치 변경

GET\_SINGLE(CInput)->AddNormalKey("Ctrl", VK\_LCONTROL);