**<10.28>**

**FrameSkip**

정확하게 구현을 해야하는 부분이므로 중점적으로 투자를할 것을 권장한다.

**CframeSkip 클래스**

**우리가 희망하는 프레임을 정확하게 일치시키기 위한 용도**

우리는 50프레임으로 만들 것. (1초에 50번의 로직이 돌게끔)

**느린컴퓨터에서는 느리게돌고, 빠른컴퓨터에서는 빠르게 도는일이 온라인게임에서 일어나면 안되므로 속도를 일괄적으로 맞춰야한다.**

그래서 프레임스킵이라는 클래스는, 50프레임으로 맞추면 1초에 50번의 로직이 돌게끔 할 것이다.

애니메이션과 프레임의 갱신이 느리더라도 이동속도는 같아야 한다.

**이때 느린컴퓨터의 경우 렌더를 포기하고 따라잡으며, 로직은 눈에보이지 않지만 내부에서는 돌고있게 된다.**

( 이렇게해서 속도를 똑같이 맞추고 있음 )

사실상 3D게임과 2D게임의 가장 큰차이는 이것이다.

**3D게임은 애초에 애니메이션데이터 자체가 프레임개념이 없고 중간중간에 키프레임을 잡아놓고 그 중간을 보간으로 계산한다.**

**( 보간 : 보간**은 두 점을 연결하는 방법을 의미한다. 여기서 말하는 연결은 궤적을 생성한다는 뜻이다. 보간이 필요한 이유는 정보를 압축한 것을 다시 복원하기 위함이다 **)**

키프레임만 잡아놓고 중간에 대한 부분은 보간처리해서 계산하여 애니메이션을 그때그때 만들어 내는 것.

따라서 컴퓨터가 빠르면 부드럽게 움직일것이고, 느리면 버벅댈 것.

2D게임 같은 경우는 이미 그래픽디자이너가 각각의 프레임을 다 만들어 놓은 상태이다.

(애니메이션이 모두 일치하게끔 다 만들어놨음)

**우리입장에서 보간처리를 하지않고, 정해진대로 프레임이 나와야만 한다**.

3D에서는 희망사항대로라면 ~정도 원했지만, 해봤더니 기대에 못미친다면 … (ex 실제로 한 프레임이 도는시간에 다섯프레임을 소비해버리는 등)

이때 키프레임과 키프레임사이에 애니메이션을 계산해서 그리는 것

**프레임이 2D에서는 다 정해져있기 때문에 애초에 프레임스킵이라는 개념이 상관이 없다.**

지금까지는 캐릭터가 좌표가있다면 한프레임이 지날 때 x축으로 +2를합니다 라는 식으로 우리가 로직을 짰었다.

이 같은 경우 위와같이 프레임이 너무 느려서 어느컴퓨터에서는 20프레임밖에 안나온다면 속도자체가 떨어지는 현상이 일어나게 될 것.

**이러한 문제 때문에 3D게임에서는 델타타임을 구한다.**

**(델타타임:이전프레임에서 지금프레임까지의 소비된 시간.)**

만약 시간이 0.2초가 지났다면 속도에다 해당 시간을 곱해서 계산을 한다. 시간이라는 개념으로 모든 것이 설계가 이뤄진다.

**애초에 픽셀이란 개념이 없고 월드상의 좌표로 이루어진다.**

**모니터의 픽셀이라는 것은 카메라의 위치에 따라 달라지므로 기준이 될수없으며, 따라서 픽셀이란 개념을 적용시킬수없다.**

이런 절대값 개념이 없다보니 먼저 1초에 속도를 구한다. 만약에 1초에 속도가 3이라는 길이를 가는것으로 정했다면, **우리는 델타타임을 구해서 내가 정한 속도(스피드)에다가 곱해버린다.**

유니티 같은 경우 전역으로 객체가 존재하게 되는데, 객체의 델타를 구하게 되면 이전프레임에서 지금프레임의 시간을 뱉어준다. ( 50프레임이면 1초에 50번 돈다. )

**50프레임 -> 프레임당 20m/s의 시간**

**100프레임 -> 프레임당 10m/s의 시간**

**프레임이 낮을수록(성능 나쁨) 1프레임당 이동하는 속도가 상대적으로 높음**

**프레임이 높을수록(성능 좋음) 1프레임당 이동하는 속도가 상대적으로 적음**

**(1000m/s기준으로 1초에 50번 도는것이므로.)**

**그럼 이 델타를 내가정한 속도에 곱해버린다.**

**그리고 이것을 x에다가 반영시키는 것.**

기획자는 1초에 3이라는 수치를 가게끔 희망했고, 델타타임이 1초에 50m/s이 나왔다고 한다면 여기에 x (Speed) 한 결과값을 덧셈하면 이게 1초에 50번나오면 3이 갈 것.

이런식으로 계산을 할 것. 초단위의 속도를 구하고, 델타타임을 구해서 그 델타타임의 여기가 1초라는 기준으로 나오기 떄문에 1초기준 0.0몇으로 나온다.

아니면 진짜컴퓨터가 느렸다 0.1 = 10프레임일경우 3에다가 0.1곱한값으로 이동을 하는 것.

컴퓨터가 느려서 프레임이 떨어져도 이동속도가 계산이 되었기 때문에 해당 속도만큼을 가게될 것.

**결론적으로 1초에 3이라는 수치를 가게될 것. 그러다보니 프레임스킵이라는 개념을 쓰지않는다.**

**하지만 2D게임에서는 픽셀이라는 개념으로 가기 때문에 여기서도 픽셀개념으로 가게된다.**

2D에서는 모든개념을 프레임으로 가기때문에 애니메이션,이동로직도 프레임을 기준으로 한다.(1프레임에 얼마를갑니다)

만약 2D에서도 프레임을 따르고싶지 않다면 이동 로직만큼은 바꿀수있다.

(한프레임에 2픽셀을 간다는 것은 1초에 100픽셀을 가게 된다.)

그렇다면 프레임에 대한 델타타임을 구해서 100을 곱해서 그픽셀을 이동시켜도 되긴하지만, 완벽하게 정확하지는 않을 것.

**이경우 프레임이란 개념이 없고 항시 최선을 다해 도는것이므로 애니메이션이 2D에서 애매해진다.** (여기서 애니메이션은 스프라이트라는 프레임애니메이션을 뜻함)

3D에서는 그 기반으로 됐기 때문에 가능하다. **그런데 우리는 애니메이션이 프레임이라는 것으로 정해져 있기 때문에 만약 내가 돌리면서 이동을 부드럽게 했다고 해도 애니메이션이 달그락 거릴 것.** 그러다보니 애니메이션과 이동의 개념만큼은 프레임개념으로 간다.

그외의 로직부분은 프레임을 쓰지않을 것.

**프레임단위를 사용해야할 것 : 애니메이션, 또는 프레임이하로 낮게 측정이 되야되는것들.**

**프레임보다 낮은 단위로의 시간체크는 불가능하다.**

20m/s라는 시간마다 1프레임이 간다고 가정하고 15m/s마다 어떤 작업을 한다고 로직을 짰다면 (timeGettime 등을이용)

이경우는 우리가 체크조차 할 수 없는 위치이며 프레임단위로 애니메이션을 넘어가기로 규칙을 정했고 그럼우리는 한프레임이 루프자체가 1초에 이간격으로 도는 구조이다.

때문에 15m/s를 체크하려고 해봤자 여기즈음에 아예로직이 오지않는다. 돌지않는다.

왜냐하면 로직자체가 20m/s간격으로 돌고있기 때문이다.

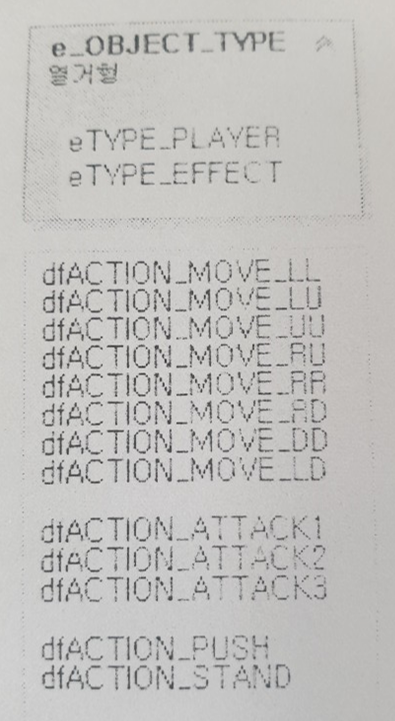
그러다보니 이런거나 25m/s, 55m/s같은 경우는 사용할수없음.

**굉장히 짧은시간은 프레임단위로 가는 것이 맞고, 초단위같이 긴 것은 시간측정으로 하는 것을 권장.**

우리가하는 2D게임에서는 50프레임을 맞출것이고, 이동에대한 로직은 프레임에대한 개념으로 가겠다는 것. 그 두개만 기억하면된다.

그외적인 부분은 시간이라는 개념을 적용하여 로직을 짜도 상관없음.

**Object\_Type**



**네트워크 할 때 액션에대한 메시지번호를 일괄적으로 쓸것이기 때문에 값을 똑같이 맞춰야한다.**

( 키보드입력이 됐을 때 왼쪽키를 눌렀다면 무브 LL 그러면 액션1번으로 보겠다는뜻. )

8방향에대한 액션번호와 어택123에 대한 번호

PUSH와 STAND는 생략한다. (애니메이션도 없음)

#define dfACTION\_MOVE\_LL 0

#define dfACTION\_MOVE\_LU 1

#define dfACTION\_MOVE\_UU 2

#define dfACTION\_MOVE\_RU 3

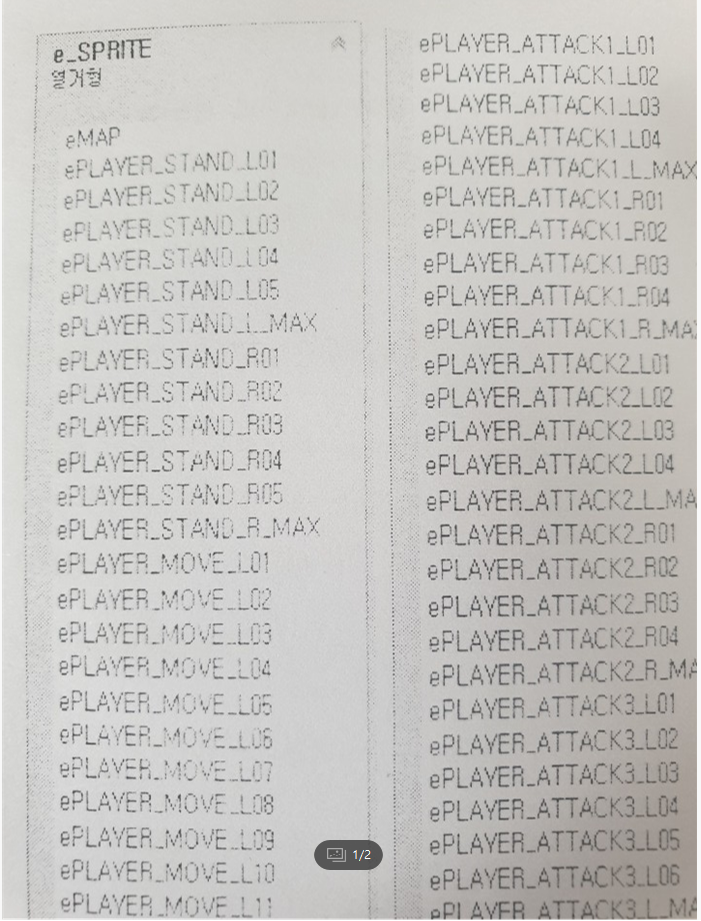
#define dfACTION\_MOVE\_RR 4

#define dfACTION\_MOVE\_RD 5

#define dfACTION\_MOVE\_DD 6

#define dfACTION\_MOVE\_LD 7

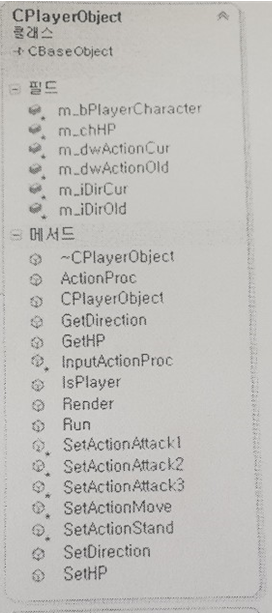
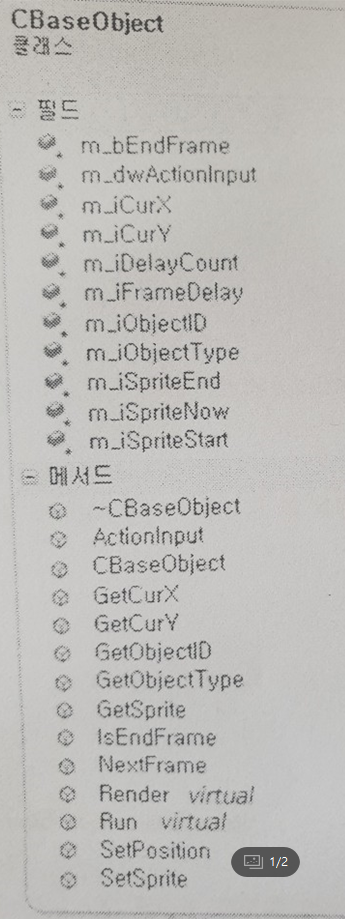
**e\_SPRITE**



스프라이트를 enum으로 번호를 배열로 매긴 것.

(중간중간 MAX로 끊어놓은 이유는 결국 애니메이션 루프를 돌려야 하기 때문에 이해하기 쉽게 하기위함)

**실제 컨텐츠 오브젝트에 대한 객체**



**Player와 Effect오브젝트는 모두 BaseObject를 public상속받는다.**

(위 클래스의 구조나 멤버변수, 함수들은 참고사항일 뿐이며 그대로 따라서 만들 필요는 없음.)

내 캐릭터와 적군 (다른 유저) 캐릭터의 클래스 구분은 없다. 모든 유저의 캐릭터는 CplayerObject를 사용한다.

모든 Object는 애니메이션을 포함하고있다고 전제되어있기 때문에 애니메이션기능을 포함시켜버린다. BaseObject에 애니메이션 기능이 포함이 됐기 때문에 상속받는 플레이어쪽에서는 애니메이션에 대해 고민할 필요가 없다.

**CbaseObject::**

**bool IsEndFrame()**

**멤버변수인 m\_bEndFrame을 반환하며, 이 멤버는 오브젝트는 애니메이션을 1회 가동시킨 후 끝에 도달했으면 true로 바꿔줄 것.**

이는 우리가 애니메이션이 1회가 끝났는지 판단할 일이 생기기 때문인데,

공격애니메이션 같은 경우 애니메이션이 끝나면 정지로 돌아가야한다.

이를 컨텐츠 차원에서 판단하기는 곤란하기 때문에 애니메이션의 마지막프레임에 도달했다면 true로 바꿔주는 형식으로 확인한다.

또는 이펙트 같은 경우에도 사용한다.

(1회재생 후 파괴된다 🡺 EndFrame을 체크하여 자기자신을 파괴하는 등의 로직에 사용)

애니메이션이 바뀔 때 false로 초기화된다.

애니메이션 재생은 무한루프를 기본원칙으로 한다.

하나의 애니메이션이 셋팅이되면 반복재생을 하되 마지막프레임에 처음 도달했을 때 EndFrame의 플래그를 true로 바꿔놓고 반복재생을 하는 방식을 택한다.

경우에 따라 반복재생시 액션이 추가적으로 바뀌거나, 이 객체가 파괴되거나 해야하는 오브젝트들이 존재한다.

**BaseObject::ActionInput()**

키보드 조작 또는 네트워크를 통해서 해당 객체의 액션 (컨트롤) 을 입력한다.

원칙적으로 따진다면 큐가 되는 것이 맞으나,

우리가 하고있는 프로젝트에서는 동시 키입력이 되는 구간이 없다.

( 공격 + 이동을 동시에 하는 것은 불가능하며, 대각선이동역시 따로 메시지가 존재한다 )

때문에 0x8000과 같은 bitflag 도 필요없음.

**이와 같은 한프레임에 하나의 메시지만 들어올수 있는 방식은 네트워크 메시지도 마찬가지인데 상대방이 메시지를 여러 개 쐈다면 가장마지막 것만 처리하고 나머지는 버리는 구조로 간다.**

(Push가 여러 번되면 그냥 덮어쓰는 구조)이기 때문. 덮어써도 문제가 없으며 오히려 저장된 행위들을 순간적으로 모두 수행하는 것이 더 이상함.

외부에서 메시지를 던지는 역할.

키보드입력이 되어도 액션인풋, 메시지가 들어와도 액션인풋만 할 것.

**GetCurX(), GetCurY()**

현재좌표

Old는 필요없음 (옛날좌표는 필요없다)

**DelayCount()**

2D에서는 애니메이션이 프레임단위.

**이 애니메이션을 1초에 50프레임으로 돌리면 어마어마하게 빠를 것이기 때문에 프레임을 늦춰야 한다.**

프레임당 20m/s일때 60m/s의 시간을 기준으로 늦춘다면 계산이 애매해진다.

그리고 약간의 시간차이로 인해서 애니메이션이 덜그럭 거릴 것.

60단위로 돌아온다는 보장이 없기 때문에.

((

이번에는 (20m/s + 39m/s)59가 나왔다면, 다음 애니메이션이 될 차례가 아닌데 (1m/s남았기 때문에 그냥 넘기는게 차라리 나을 것이다. 하지만 이것을 구분할 방법이없음.)

59m/s가 나와서 이번프레임을 한번 더찍었는데 사실은 다음프레임까지 넘어가려면 실제로는 20에다가 61이 더해져야 다음프레임으로 넘어갈 것.

원래대로라면 60m/s단위로 가야하는데 중간에 걸려버릴경우 (넘어가야하는데 넘어가지 못했을 경우) 해당애니메이션이 더 길게출력되게 되므로 전체적으로 봤을 때 애니메이션자체가 달그락 거릴 것

빨라졌다 느려졌다...

))

**때문에 시간이라는 개념으로 처리하는 것이 아니라 프레임으로 처리하게된다.**

그리고 프레임을 60프레임으로 맞춘다.

프레임딜레이에 3,3,3,2같은 것을 넣고, 딜레이A카운터는 카운팅용이다.

0,1,2,0,1,2하면서 프레임딜레이가 도달할때까지 반복으로 현재 애니메이션을 찍겠다는 것

이구조를 똑같이 따라할 필요없음.

한가지 예시일 뿐임

**GetObjectID()**

**이 아이디는 서버가 부여하는 고유값이다.**

지금은 없지만 서버가 달라붙으면 서버가 고유한 아이디를 부여할 것. 실제로는 회원번호가 되야하는데 지금은 회원가입이라는 개념이 없으므로 이렇게 가정하고 가게된다.

ObjectType이 이펙트냐 플레이어냐. 지금우리 오브젝트는 딱 두개밖에없음.

내것도 플레이어 남것도 플레이어.

**SpriteEnd(), SpriteNode(), SpriteStart()**

는 **애니메이션의 범위이다**. 스타트에서 앤드까지 애니메이션을 시킬것이고 나오는 현재 애니메이션.

**NextFrame()**

다음프레임으로 넘겨주는 기본행동

각 모든 오브젝트의 기본 Run 파트에서 이 NextFrmae을 호출시키게될 것.

**Virtual Run()**

메인 Update에서 호출해 줄 객체의 Run함수(애니메이션 처리와 ActionProc를 호출)

**CplayerObject::**

**IsPlayer()**

해당 객체가 플레이어인지 판단하는 플레그값.

내 플레이어인지 아닌지 알아야할 시기가 나오게될 것. 지금당장은 몰라도 된다.

그냥 외부에서 전역변수로 포인터를 들고있으면 된다. 그리고 키보드 눌릴때마다 그 객체한테 꽂아주면 된다.

나중에 네트워크가 붙게되면 객체자체가 내가 플레이어냐 아니냐를 알아야 하는 시점이 있다. 애니메이션이 되는 것을 처리할 때.

그떄 다시고민해볼 것

**dwACtionCur, ddwActionOld**

**이 액션에 Cur, Old가 있는 이유도 네트워크인데, 네트워크에다 보낼때는 변화만 보낼 것.**

내가 아무런행동을 하지않는다면 아무것도 보내지 않을것이고, 이동을 시작하면 이동만 보낸 후 정지할 때 다시보낼 것.

이처럼 액션의 변화를 봐야하기 때문에 Old, Current비교가 되어야 변화가 있는지 없는지 확인할 수 있는거니까.

이 멤버는 알아서 판단하여 넣을 것.

**Player::ActionProc()**

Run()함수 내부에서 호출 해주는 실제 객체 처리 부.

내 base멤버에있는 actionInput 여기에있는 메시지를 수행하겠다.

**Player::InputActionProc()**

ActionInput을 통해서 들어온 액션 값을 실제로 처리하는 부분. ActionProc()내부에서 호출 당함

**SetAction~ 함수들**

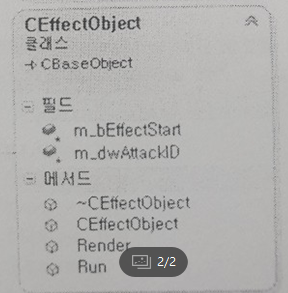
애니메이션을 실제로 변경하는 함수들

이안에서 어택을 지정 하면 스프라이트 범위를 몇번에서 몇번으로 셋팅을하고 그리고 프레임딜레이와 같은 것들을 안에서 셋팅을 할 것. (내부모쪽 함수를 사용해서)

**ActionProc(), InputActionProc()**

ActionProc는 Run안에서 호출시키는 함수이고, ActionProc안에서 InputActionProc를 호출시키는 형태.

반드시 위의 예제와 똑같이 만들 필요는 없으나 혼동을 방지하기 위하여 언급한다.



**CeffectObject::**

만약 플레이어가 Attack을 수행했다고 한다면 해당 애니메이션을 수행하고 데미지가 들어갈 것.

**하지만 내부에서는 X키를 누르고 액션이 변경이 되자마자 서버에다 이미 Attack을 쏴버렸다.**

이후 **서버에서는 충돌처리까지 완료했고, 지금은 결과 (‘플레이어가 특정 오브젝트를 공격했다’라는 판정 )까지 와버린 상태.**

하지만 플레이어쪽에서는 아직 애니메이션도 들어가지않았다. 프레임이 변화가 없거나 1~2프레임정도만 지났기 때문이다.

이펙트는 주먹을 뻗어서 부딪히는 시점에 터져야 하는데, 이벤트가 수행되기전에 이미 결과 메시지가 나한테로 반환된다.

**이펙트는 애니메이션에 맞춰 동기화 되어야 하며, 이것은 서버가 맞춰줄수 있는 것이 아니므로 클라가 알아서 해야하는 일이다.**

**서버는 최대한 빨리 결과값을 줄 것이고, 이펙트-애니메이션 동기화는 클라쪽에서 알아서한다.**

여기서 생각해볼수있는 두가지 방법

**1. 공격성공이라는 메시지가 왔을 때 이펙트 객체를 생성시키는 방법**

**애니메이션이 보이면 안되므로 객체는 생성하되 가동이 되지않고 정지상태로 있는것.**

이 이펙트오브젝트가 어딘가 생성하고, 얘에 대한 애니메이션을 계속해서 판단하고 있는 것.

이펙트 오브젝트는 Run이라는 코드안에서 계속 판단하고 이펙트가 언제터질까 기다리다가 조건이 맞는다면 가동되는 것.

**2. 공격성공 메시지가 왔을 때 기억해두는 것.**

**공격성공에 대한 메시지를 받았으니까 내 주먹이 뻗을때쯤에 이펙트를 생성시키겠다고 Player::Run에서 이것을 판단하는 것**

어떤 방법을 써도 상관없음.

위 예제에서는 1번방식을 사용하고있으므로 **M\_dwAttackID** 와 **m\_bEffectStart**라는 플래그를 사용

**이 멤버를 통해서 해당 이펙트객체가 생성이 될 때 공격자를 알고있는 상태가 되었고 매번 Run 액션파트에서 공격자의 애니메이션 번호를 계속 확인하는 코드가 들어간다.**

그리고 자기이펙트를 언제터트릴지 본인이 계속 체크하고 있는 것

이런동기화는 클라쪽에서 알아서 맞춰줘야한다.

서버쪽에서 관여할 수 없음.

우리가 지금 테스트할 환경에서는,

공격키를 눌렀을 때 서버한테 그 공격메시지를 보내고 서버가 판정해서 결과를 주기까지가 내가 공격애니메이션이 끝나기 전까지 이미 와버릴것

왜냐하면 테스트 상황자체가 로컬네트워크고 또는 여러분의 컴퓨터한대, 또는 Local LAN. 네트워크상에 가장 좋은환경이다. 레이턴시는 거의없다. 못느낄정도.

**하지만 실제 서비스를 한다고 한다면 공격이 다끝난후에 판정이 오는경우가 생길 수있다.**

**이는 레이턴시가 길다는 얘기가 아니며 우리의 클라/서버 가 느린 것.**

실제로 서버를 우리가 만들었다고 한다면 수천/수만이 플레이한다면 느려지는 상황이 나올 것.

그렇다면 이것에 대한 동기화를 어떻게 맞출것인가?

**키보드로 조작을 한다고 한다면 내가 공격/이동을 누르는순간 액션이 들어가야 한다.**

( 마우스 조작이면 마우스 클릭을 했을 때 액션이 안들어가도 상관없다.

서버로부터 결과가 왔을 때 행동해도 어색함이 없음.키보드를 누를 때 키보드 포인터에 클릭했을 때 거기 클릭했다라는 이펙트는 뜨기 때문에, 유저입장에서는 이것을 두고 조작감이 떨어진다고 생각하지 않는다.

**따라서 조금늦게(50m/s, 100m/s정도) 움직여도 후에 움직여도 전혀 어색하지 않다** )

키보드의 경우 누르는순간 행위가 보여져야 하기 때문에, 서버로부터 늦은 결과를 받아서 행동한다면 한박자 느린느낌이 나게된다.

((

마영전에서는 이와 같이 한박자 느린현상을 해결하기 위해 **클라에서 먼저 확인해서 공격판정이 떨어지면 그 몬스터를 데미지받는 타격애니메이션에 프리징을 시킨다.** (이펙트는 터지지 않는상태) **그후 서버로부터 결과가와서 판정이 확인되었다면 이펙트가 터지게되고 타격애니메이션이 계속 재생된다.**

판정이 아니라고 결론이 난다면 miss를 띄우게될 것.

이경우 오히려 특수효과를 넣은듯한 타격감이 생기게된다.

이와 같이 기획적으로 접근하여 우회하는 방법을 사용하는 것이 일반적임.

))

**플레이어 객체의 생성**

**플레이어 객체의 포인터를 저장 해 둘 포인터 변수를 전역으로 하나 두도록 한다.**

**이는 플레이어 객체의 빠른 접근을 가능하도록 한다**. 모든 객체는 리스트로 관리 될 것이며, 플레이어 객체도 그 리스트 안에 들어있다. **어딘가에 플레이어 객체의 포인터를 저장해두지 않는다면 리스트를 매번 뒤져서 플레이어 객체를 찾아내야 한다.**

키보드 입력이 올때마다 네트워크로 오는 메시지는 결국 검색을 해야한다. 어떤Object의 Id와 Action이 같이 오기 때문에 항상 ID를 검색해서 찾아야하는데, 키보드 입력 같은 경우는 플레이어 자기 자체가 해야하므로 입력이 올때마다 찾아내서 검색하는 방법은 비효율적이다.

그냥 전역으로 포인터를 하나 두고서 키보드 입력을 할 것.

이것은 편법이나 잘못된 것이 아니라 원래 이렇게 한다.

CBaseObject \*g\_pPlayerObject;

현재는 스테이지 개념이 없으므로 게임 초기화 부분에서 임의의 좌표로 플레이어 캐릭터를 생성시켜 준다.

아직 적군은 없지만 테스트를 위해서 플레이어 객체 외에 여러명의 객체를 더 생성하여 리스트에 넣어놓고 테스트를 하는것도 좋다.

**InitailGame같은 곳에서 SpriteLoading을 한다음 약간의 플레이어 객체를 생성시켜서 리스트에 꽂아둔다.**

AI까지 넣어서 하면 좋긴하지만 그럴필요까지는 없다.

객체들을 이런식으로 구현한 것만 구현하고, 오브젝트 매니저의 유무 같은 경우는 무시할 것.

지금은 그게 중요한 것이 아님.

스프라이트 로드...

스프라이트 로드...

스프라이트 로드...

스프라이트 로드...

g\_pPlayerObject = CPlayerObject 생성;

g\_pPlayerObject -> 좌표 및 기타 정보, 능력치 셋팅..

g\_pPlayerObject -> 좌표 및 기타 정보, 능력치 셋팅..

g\_pPlayerObject -> 좌표 및 기타 정보, 능력치 셋팅..

**리스트 삽입 ( g\_pPlayerObject )**

// 테스트용 객체들 생성

CBaseObject \* pObject;

pObject = CPlayerObject 생성

pObject -> 좌표 및 기타 능력치 셋팅

리스트 삽입(pObject)

CBaseObject \* pObject;

pObject = CPlayerObject 생성

pObject -> 좌표 및 기타 능력치 셋팅

리스트 삽입(pObject)

CBaseObject \* pObject;

pObject = CPlayerObject 생성

pObject -> 좌표 및 기타 능력치 셋팅

리스트 삽입(pObject)

**게임의 Update함수**

void Update\_Game(void)

{

if (g\_bActiveApp)

KeyProcess();

Action();

Draw();

g\_cScreenDib.DrawBuffer(g\_hWnd);

Sleep(0);

}

//메모장

void KeyProcess(void)

{

**이 함수는 전역에 있는 플레이어 객체 포인터 g\_pPlayerObject에게 명령을 주는 부분이다.**

그러므로 혹시나 g\_pPlayerObject가 NULL포인터는 아닌지 확인 후 처리하도록 한다.

... 키보드 상태에 따른 방향별 이동, 공격, 정지 동작의 확인...

... 키보드 상태에 따른 방향별 이동, 공격, 정지 동작의 확인...

... 키보드 상태에 따른 방향별 이동, 공격, 정지 동작의 확인...

... 키보드 상태에 따른 방향별 이동, 공격, 정지 동작의 확인...

... 키보드 상태에 따른 방향별 이동, 공격, 정지 동작의 확인...

// 플레이어 객체에게 입력된 동작을 알려준다.

g\_pPlayerObject->ActionInput(dwAction);

}

정지 같은 경우 Stand라는 액션이 들어온 것. 키가 안눌리면 StandAction. 하지만 애니메이션은 계속 돌고있는 상태이다.

이동 같은 경우도 이동이 계속들어오면 애니메이션이 이어져야한다. 그런데 여기서 뛰는 경우 즉각정지하고 애니메이션이 바뀌는 현상이 나타난다.

공격도 마찬가지로 자동으로 끝까지 다 애니메이션이 출력되고 정지로 돌아야 할것이다.

여기서 주의해야 하는 부분은, **이 함수에서 PlayerObject의 동작, 프레임등을 확인하여 상황에 따른 처리하는 로직이 들어가면 안될 것.**

**KeyProcess에서는 오직 키 입력에 대한 정보 생성 후 이를 플레이어 객체에게 전달 해주고 마무리가 되는것이 좋다.** ( 이 입력에 대한 처리는 총 8방향이 되어야 할 것.)

**만약 네트워크가 달라붙은 경우에는 내 플레이어도 존재하지 않을 수있다. 예외처리가 되어야 할 것. 네트워크로 서버에서 객체를 움직이라고 했지만 해당 객체가 존재하지 않을 수도 있다. 항상 고려해야 하는 부분!**

void Action(void)

{

...객체 리스트 전체를 돌면서 모든 객체에게 Run을 호출한다.

...리스트의 객체들을 Y좌표 기준으로 정렬한다.

...이펙트는 가장 뒤로 밀어버린다.

}

Action함수의 마지막에서는 리스트의 객체들을 Y좌표 기준으로 정렬을 해줘야한다. 그래야 Draw부분에서 순서대로 찍히게 될것. 또한 이펙트 종류의 객체들은 가장 뒤로 밀어 주어야 한다.

Y축이 클수록 뒤에 렌더링이되어야 할 것.

Effect역시 항상 맨앞에 터져야 하기 때문에 가장 뒤로 밀 것.

void Draw(void)

{

//출력 버퍼 포인터

BYTE\* bypDest = g\_cScreenDib.GetDibBuffer();

int DestWidth = g\_cScreenDib.GetWidth();

int DestHeight = g\_cScreenDib.GetHeight();

int DestPitch = g\_cScreenDib.GetPitch();

... 맵출력 ...

현재 맵은 별도의 객체나 클래스를 만들지 않는다.

간단하게 만들고자 하는 경우는 여기서 SpriteDib.DrawImage함수로 그냥 (0,0)에 맵을 찍어주면 될 것.

제대로 배워보고 싶다면 맵관리 클래스를 만들어 전역이나 싱글톤을 두고 여기서 찍어주면 된다.

(이후 맵을 화면에 하나가 아닌 6400x6400 으로 열배로 늘려서 스크롤로 왔다갔다 할것인데, 이때는 맵클래스가 필요하게 될 것)

... 모든 객체 출력 ...

... 리스트의 모든 객체들에게 Render 함수를 호출 ...

pObject->Render(bypDest, DestWidth, DestHeight, DestPitch);

}

플레이어의 렌더안에서 게이지바, 캐릭터, 그림자가 모두 이루어진다. 이런것까지 모두 객체로 뺄 필요는 없다.

**로직(Action)과 렌더(Draw)는 반드시구분!!**

프레임이 느리다면 렌더를 생략한다. 때문에 렌더에서 멤버에게 변화를 주는 행위가 하나라도 들어간다면 코드가 완전히 꼬이게 된다.

**객체의 애니메이션 관련함수**

이제 모든 객체는 스프라이트 애니메이션이 가능해야 하므로 스프라이트는 모든 객체에게 공통적인 부분이다.

그러므로 스프라이트 프레임관리는 BaseObject에서 해주도록 한다.

void CBaseObject::SetSprite(int SpriteStart, int SpriteMax, int FrameDelay)

{

m\_SpriteStart = SpriteStart;

m\_SpriteMax = SpriteMax;

m\_FrmaeDelay = FrameDelay;

m\_SpriteNow = SpriteStart;

m\_DelayCount = 0;

m\_bEndFrame = FALSE;

}

int CBaseObject::GetSprite(void)

{

return m\_SpriteNow;

}

int CBaseObject::IsEndFrame(void)

{

return m\_bEndFrame;

}

void CBaseObject::NextFrame(void)

{

if (0 > m\_SpriteStart)

return;

//---------------------------------------------------

// 프레임 딜레이 값을 넘어야 다음 프레임으로 넘어간다.

//---------------------------------------------------

m\_DelayCount++;

if (m\_DelayCount >= m\_FrameDelay)

{

m\_DelayCount = 0;

m\_SpriteNow++;

//-----------------------------------------------

// 애니메이션 끝으로 이동되면 처음으로 되돌린다.

//-----------------------------------------------

if (m\_SpriteNow >= m\_SpriteStart)

{

m\_SpriteNow = m\_SpriteStart;

m\_bEndFrame = TRUE;

}

}

}

**애니메이션 변경**

액션이 변경되어 스프라이트 애니메이션이 변경되는 순간 외부에서 CBaseObject::SetSprite()함수를 호출하여 애니메이션을 변경하여 준다.

스프라이트 범위(Start, Max)와 애니메이션의 속도 조절을 위한 Delay값을 입력받는다.

**애니메이션 처리**

위 설계에서는 CbaseObject::NextFrame 함수를 통해서 다음 프레임 스프라이트로 넘어갈 수있다. **NextFrame함수는 각 객체의 Action에서 호출되며 매 프레임마다 한번씩 무조건 호출이 되어야 하는 함수이다.**

**NextFrame은 애니메이션 처리에 대한 부분이긴하지만, 렌더에 들어가지않는다. 애니메이션도 멤버변수에 변경을 가하고있기 때문에 로직파트에 들어간다.**

void CBaseObject::NextFrame(void)

{

// 스프라이트 자체가 셋팅되지 않았다면 무시.

if (0 > m\_SpriteStart)

return;

//---------------------------------------------------

// 프레임 딜레이 값을 넘어야 다음 프레임(스프라이트)으로 넘어간다.

//---------------------------------------------------

m\_DelayCount++;

if (m\_DelayCount >= m\_FrameDelay)

{

m\_DelayCount = 0;

//-----------------------------------------------

// 애니메이션이 반복재생되는 것이 디폴트 이므로,

// 애니메이션 끝으로 이동되면 처음으로 되돌린다.

//-----------------------------------------------

m\_SpriteNow++;

if (m\_SpriteNow >= m\_SpriteMax)

{

m\_SpriteNow = m\_SpriteStart;

m\_bEndFrame = TRUE;

}

}

}

위 현재 멤버변수에는 DelayCount라는 멤버 변수가 있으며 이는 애니메이션의 프레임 딜레이를 위해 존재하는 변수이다.

m\_DelayCount = 3 인 경우는 하나의 스프라이트가 3프레임을 소비하고 다음 스프라이트로 넘어간다. 이는 SetSprite() 함수에서 설정되고 있고, 이는 필수가 아닌 옵션사항이다.

m\_SpriteNow는 현재의 스프라이트 인덱스를 가지며, 이 변수가 화면에 출력 할 스프라이트의 인덱스 이다. M\_SpriteMax값까지 +1씩 올라가며 Max에 도달한 경우 m\_bEndFrame을 true로 설정 후 다음 프레임으로 돌아간다.

if (m\_DelayCount >= m\_FrameDelay)

{

m\_DelayCount = 0;

//-----------------------------------------------

// 애니메이션 끝으로 이동되면 처음으로 되돌린다.

//-----------------------------------------------

m\_SpriteNow++;

if (m\_SpriteNow >= m\_SpriteMax)

{

m\_SpriteNow = m\_SpriteStart;

m\_bEndFrame = TRUE;

}

}

마지막 프레임에 도달 했을 시 m\_bEndFrame값이 TRUE로 변경 되어 애니메이션이 끝났다는 것을 확인하는 용도로 사용하게 된다.

공격애니메이션이 진행되는 동안에는 키가 눌려도 무시가 될 것.

그래서 Attack 1,2,3같은 경우에는 EndFrame이 되었을 때 Stand 애니메이션을 변경하는 작업만 진행하고 나가버리는 것. 그외에 행동의 경우에는 외부에서 들어온 사용자 입력을 처리하겠다는 것이다.

**외부에서의 액션입력은 단순 멤버변수의 셋팅만 해준다는 것이고 키보드 입력에서 ActionInput은 멤버변수 셋팅, 메시지전달 이라는 것**

**동작의 끝 확인**

외부에서 현재 동작이 끝났는지 확인을 해주어야 하는 상황이 생기곤 한다.

------------------------------------------------------------------------------------

1회성 애니메이션 출력 후 파괴되어야 하는 객체. Ex) 폭파 이펙트 등..

1회성 애니메이션 출력 후 다른 동작으로 변경되어야 하는 액션 Ex) 공격액션

------------------------------------------------------------------------------------

이런 액션의 경우 끝 프레임에 도달여부를 확인 할 수 있어야 하며, 현재 구조에서는 m\_bEndFrame멤버변수로 확인이 가능하게 된다.

**객체의 드로잉**

이제 모든 객체의 그래픽 출력은 메모리에 출력을 해야한다.

CBaseObject::Render(BYTE\* bypDest, int DestWidth, int DestHeight, int DestPitch)

그래서 최상위 객체의 Render가상 함수는 위와 같은 형식이 되었다.

이는 메모리상의 이미지에 출력을 위한 최소한의 정보 [ 출력대상 포인터 / 너비 / 높이 / 피치 ]이다. 각각의 객체들은 자신의 그림을 각자 자신의 Draw함수에서 해줄 수 있다.

DWORD CPlayerObject::Render(BYTE\* bypDest, int DestWidth, int DestHeight, int DestPitch)

{

// 그림자 출력

g\_Sprite.DrawSprite50(eSHAOW, CurX, CurY, bypDest, DestWidth, DestHeight, DestPitch);

// 플레이어 객체 출력

g\_cSprite.DrawSprite(GetSprite(), CurX, CurY, bypDest, DestWidth, DestHeight, DestPitch);

// HP게이지 바 출력

g\_cSprtie.DrawSprite(eGUAGE\_HP, CurX - 35, CurY + 9, bypDest, DestWidth, DestHeight, DestPitch.GetHP());

return TRUE;

}

이처럼 해당 객체에 관련된 그림은 해당 객체 내부에서 자신의 좌표를 기준으로 출력 해 줄 수 있다.

**객체 내부의 일부 참고용 코드**

DWORD CPlayerObject::Action(DWORD dwParam)

{

//-----------------------------------------------

// 다음 프레임으로 애니메이션

//-----------------------------------------------

NextFrame();

ActionProc();

return FALSE;

}

void CPlayerObject::ActionProc(void)

{

//----------------------------------------------------------------

// 몇몇 동작시 (공격)의 경우 강제적으로 해당 동작 처리를 완료해야한다.

//----------------------------------------------------------------

switch (m\_dwActionCur)

{

//----------------------------------------------------------------

// 공격동작 또는 데미지 동작은 애니메이션이 끝날 때까지 강제적으로 애니메이션이

// 되어야 하며 애니메이션이 끝난 후 기본동작으로 자동으로 돌아가야한다.

//----------------------------------------------------------------

case dfACTION\_ATTACK1:

case dfACTION\_ATTACK2:

case dfACTION\_ATTACK3:

if (IsEndFrame())

{

SetActionStand();

//-----------------------------------------------------------------------

// 공격이 끝났었더라면, 액션을 바꿔줘서 연속으로 공격을 할 때 전송이 // 가능하도록 한다.

//-----------------------------------------------------------------------

m\_dwActionInput = dfACTION\_STAND;

}

break;

default:

//------------------------------------------------

// 이외의 경우에는 사용자 입력 처리를 해준다.

//------------------------------------------------

InputActionProc();

break;

}

}

외부에서의 액션 입력은 단순 멤버변수에 셋팅만 해준다.

**액션게임으로 즉각 반응을 해야 하기 때문(한프레임에 동시에 두개의 입력이 들어올상황이 없기때문)에 큐방식은 사용할 필요가 없음.**

**혹시나 여기에서 직접스프라이트 변경, 애니메이션 변경, 좌표이동 등을 해주어선안되며 액션 변경에 대한 실질적인 처리는 Action함수에서만 하도록 한다.**

void CBaseObject::ActionInput(DWORD dwAction)

{

m\_dwActionInput = dwAction;

}

**FrameSkip에 대한 추가설명**

스타크래프트 같은경우 특정 플레이어의 프레임이 떨어진다면 화면이 흑백으로 변하면서 아예 멈춰버리고, 해당 플레이어를 쫓아낼건지 투표한다.

MO의 경우는 일부 유저가 모여서 게임을 하는 형식이기 때문에 이러한 방식을 채택해도 상관이 없었다.

MMO같은 경우 해당 플레이어는 끊기는듯한 현상과 함께 서버와 동기화를 맞춰나갈 것

만약 해당 프로젝트에 네트워크가 붙게된다면 좌표가 틀어질경우 서버가 끊어버릴것이다.

스피드핵(게임자체를 빠르게 돌리는 핵을 말함) 같은 경우,

**싱글게임에서는 통하는 핵이지만 클라-서버구조를 가진 온라인 게임 같은 경우는 서버에서 이동을 하는 것이 원본데이터이기 때문에 스피드핵을 사용한다고 해더라도 서버와 동기화된 좌표로 롤백되어버린다.**

(나혼자 빨리갔다가 다시 뒤로 튕겨나는 현상)

지금하고 있는 프로젝트의 경우에는 아예 접속을 끊어 버릴것이기 때문에 클라와 서버의 속도를 수시로 똑같이 일치하게 맞추지 않는다면 끊어질 것.

그렇다면 속도를 어떻게 맞출것인가?

**로직이 우리가 원하는 초당프레임보다 빠를경우**

**한프레임이 돌아서 Run과 Render까지 끝난 후 우리가 목표한 초당 프레임보다 시간이 남았을경우에는 Sleep으로 쉬면된다.**

로직이 빠를 때 느리게맞추는 것은 간단하다.

while (true)

{

//키보드입력

//로직

//렌더

Sleep(x);

}

해당 위치는 꼭 시작이나 끝이 아니어도 상관없으며,

**어디를 돌든 지정한 지점까지 돌아왔을때가 1프레임이 될 것.**

**우리가 앞으로 로직을짤때는 아무생각없이 루프를 도는 행위는 절대 없어야 하며 이는 모든 스레드, 코드에 해당된다.**

코드는 무조건 할일이 있으면 돌고, 없으면 쉬어야 한다.

할일이 없다는 조건이 Sleep이 될수도 있는 것이며 이는 개발자의 의견에 따라 다르다.

Sleep(10)으로 중지하면서 루프를 돌리겠다고 의도하였다면 그게 맞는 것.

**무한루프를 돌면서 단순하게 시간을 체크해서 렌더를 할까말까를 확인하는 코드는 잘못된 코드!**

(현재는 스레드를 쉬게하는 Sleep함수밖에 모르지만, 스레드 프로그래밍쪽 에서는 Sleep말고도 많이 존재한다. 스레드를 블락상태로 바꿀수있는 함수들.)

이 얘기를 우리 프로젝트에 적용한다면,

**초당 50프레임으로 돌린다는 조건을 맞췄기 때문에 20m/s마다 1프레임을 돌리는 조건을 맞출것이고 만약 빠르게 소비되었다면 Sleep으로 쉬게하겠다는 의미**.

프레임간의 간격이 20m/s를 거의 채웠거나 초과했다면 쉬지않고 넘어갈 것.

이경우 코어를 모두먹게되더라도 로직을 돌리게 된다.

클래스를 두든, static이나 전역함수를 두든 이전 프레임에 대한 타임인 **OldTime**과 현재시간인 **NowTime**을 할당해놓고 계산한다면 프레임당 몇 m/s인지 알 수 있을 것.

(Nowtime의 경우 TimegetTime()으로 갱신)

**NowTime – OldTime = DeltaTime**

델타타임이 20m/s보다 작다면 ( = 로직이 빠를경우) Sleep으로 원하는만큼(20m/s) 쉬게된다.

**우리가 원하는 것은 초당 50프레임 ㅡ> 프레임당 20m/s**

(만약 DeltaTime이 15m/s가 나왔다면 5m/s를 쉬게될 것)

Sleep()이후 return true를 반환하여 Render를 패스하지않고 돌리도록하고,

Oldtime을 갱신시키면 될 것.

**여기서 Timebeginperiod를 1로 맞춘다고 하더라도 필연적으로 오차가 발생하게된다.**

코드자체에는 문제가 없으며, 대부분 이 형식을 채택함에도 불구하고

프레임당 50m/s가 아니라 47,48정도가 나오게된다.

**이는 Sleep과 timeGetTime에서 발생하는 오차가 원인이다.**

우리가 2m/s를 쉬게한다고 해서 2m/s를 쉬게할 수가 없다.아무리 timebeginperiod를 1로 맞췄다고 하더라도 2m/s 이상이다.

Sleep이란 행위는 이 스레드를 2m/s후에 깨우라고 지정 후 내 스레드를 블락상태로 빼버리는 것.

**그럼 블락상태 있는 스레드 중 타이머를 기준으로 깨고자하는 스레드를 따로 관리하게 되며, 커널이 따로 스레드를 관리도중 깨어날 스레드의 유무를 스케쥴링할때마다 체크하게 된다.**

이후 깨워야 되는 순간에 레디큐로 올리면서 수행을 기다리고, 이후 러닝된다.

**2m/s가 지나자마자 정확하게 깨워냈다고 하더라도 2m/s이상이 나올수밖에 없다.**

**timebeginPeriod()를 1로 맞췄다고 하더라도 이는 체크하는 간격을 1로 바꾸는 것이며,**

**Sleep(2)이후에 깨어나도 바로 수행될수있는 것이 아니라 수행되기를 기다려야하기 때문이다.**

이경우 대부분이 2m/s가 아니라 3m/s쯤 쉬게되며,

**윈도우 10쪽에서는 거의 1m/s간격안으로 나오지만 윈도우 서버쪽에서는 2m/s가 나온다.**

**(Sleep(1)을 넣어도 2m/s가 나오게 된다)**

슬립이라는 행위가 정교하지 않기 때문인데, 이는 슬립이 나쁘거나 성능이 떨어져서 정교하지 않다는 것이 아니라 정교하게 기다릴 방법이 없는 것을 뜻한다.

**결국 우리는 어떤 방법을 동원해도 우리가 원하는 m/s에 정확하게 깨어날 수 없다.**

**무조건 그 이상이 될 것.**

**timegetTime()의 경우도 마찬가지인데, 기본적으로 윈도우는 타이머 인터럽트 갱신에 따라서 가지고 올 변수를 업데이트 시키고 있는 상태이다.**

timeGetTime은 커널내부에있는 타이머 변수, timeGetTime변수에서 가져오는 것.

값을 가져오는 시점에 오차 1m/s가 나게되는 것이다.

이과정에서 1m/s의 오차만 났다고 하더라도 1초에 50번이므로 50m/s의 오차가 생기게 되고, 이는

두프레임에서 세프레임정도가 빠지게된다는 뜻(50m/s -> 47~48m/s)이된다.

**이 같은 오차를 해결하기위해서,**

**델타 계산이 끝나고 Sleep(1)이 끝나고 난후에 OldTime을 TimeGetTime으로 업데이트 하는것이 아니라 20m/s을 더해준다.**

**실제로는 위와 같은 오차 때문에 20m/s이상이 나왔을 것이지만, OldTime에 단순히 20m/s를 더해주게 된다면 이후 프레임에 NowTime – Oldtime으로 델타를 계산할 때 넘어간 오차값을 소비시간으로 받아들이고 계산하게 될것이기 때문이다.**

(오차가 1m/s라고 가정한다면, 다음프레임에 15m/s에 프레임이 끝났다고 하더라도 delta값이 16m/s이 나오므로 Sleep(4)를 하게될 것)

**이 같이 오차를 무시하고 시간을 밀어넣는다면 1초에 50번씩 나는 오차값을 마지막에 딱 한번 나는나는 것 줄일 수 있다.**

**마지막계산은 계속 넘어가고있기 때문에 내가 보고있는 마지막 오차는 날수밖에 없음.**

QueryPerformanceCount라면 조금이나마 timegettime의 오차를 줄이겠지만 Sleep에서는 해결이 안되기 때문.

**로직이 우리가 원하는 초당프레임보다 느린경우**

**렌더파트를 지나지않고 뛰어넘을 것.**

클라이언트 게임에서는 부하의 70%이상이 렌더가 차지하고 있음.

(이는 렌더와 로직부를 나눠야되는 이유이기도 하다.)

**FrameSkip()에서 false를 리턴하여 렌더를 스킵한다면 다음프레임은 비정상적으로 빨리 끝나게 될 것이고, 빨리끝났으므로 Sleep을 해버리게 될것이다.**

**이경우에 렌더->렌더생략->렌더->렌더생략 이 반복되기 때문에 프레임이 반토막이 날 것.**

**때문에 렌더 스킵은 초과된시간이 한프레임의 시간을 넘었을 때 실시한다.**

(20m/s기준으로 돌리고 싶은데 프레임이 40m/s이상이 되었을 때)

초과한 시간을 모아뒀다가 20m/s에 도달했을 때를 기준으로 렌더를 스킵하고 초과분을 지울 것이다.

하지만 렌더를 한번 생략한다고 해서 완전히 맞춰지는 것이 아니다.

한프레임에 대한 로직이 20m/s인 것이지 렌더에대한 시간이 20m/s인 것이 아니기 때문.

**로직이 3m/s, 렌더가 17m/s 를 소비할때 렌더를 생략하면 3m/s가 다시 초과분에 저장 되야 할 것.**

(이때 3m/s를 신경쓰지않고 저장된 20m/s를 모두 날려버린다면 프로젝트 자체에 큰 문제는 없으나 권장하지 않는다. )

**여기서 초과분을 저장하는 기준을 1프레임단위(20m/s)로 하는 것은 하나의 기준일 뿐이며 적정선을 찾아 볼 것.**

만약 1프레임단위인 20m/s가 아닌 15m/s의 초과분이 생길 때 렌더를 스킵한다고 가정했는데 렌더하는 시간이 15m/s보다 빠를 경우 계산이 애매해진다.

다시 Sleep할수는 없는노릇.

위와 같은 경우를 방지하기 위해서 한프레임을 기준으로 한 것.

이때 이 프로젝트를 최종적으로 완성을 했을 때 게임을 돌리다가 다른프로그램(크롬 등)을 띄우면 일시적으로 멈추다시피 한다.

(학원컴퓨터는 듀얼밖에 안되다보니까 성능이 떨어져서 일어나는 현상)

이 경우 내 게임쪽에서 평소에는 50프레임이 나오다가 갑자기 어느순간 30프레임으로 떨어진다.

그럼 20프레임에 대한 부족분이 나왔기 때문에 언젠가는 다시 70프레임이 나와서 부족분을 따라 잡아야한다.

때문에 렌더 프레임과 로직프레임 두개를 따로 구해서 모니터링이 가능하게끔 한다.

(보통 타이틀바에 출력)

((

**우리의 목표는 로직프레임이 항시 50이 되는 것을 희망하는것이고 렌더는 0까지 떨어져도 상관없다. 렌더도 50이되면좋지만 떨어져도 상관없음. 다만 로직은 항시 50이 되어야 한다.**

근데만약에 렌더프레임은 10인데 로직프레임이 40이 됐다면 잘못만든 것. 로직이 50에 도달하지 못했는데도 렌더파트를 수행했기 때문.

로직이 부족한데 렌더를 돌 상황이 아니다!! 뭔가 계산이 잘못된 것.

만약하다가 로직이 40까지 떨어졌다면 언젠가는 60으로 보정을 시켜야한다. 렌더는 0이됐든 뭐가됐든 상관없음.

))

다만 프레임을 모니터링할 때 항상 50프레임이 나오게끔 하는 것은 불가능하며,

-+ 1정도 오차가 있을수있어서 49~51을 왔다갔다 할 것.

**프레임을 기준으로 맞추는 게임은 지금시대에서 콘솔을 제외하고는 없으며, 대부분 델타타임을 기준으로 간다. (**2D도 3D로 만들어버리는 추세)

**서버도 프레임이란 개념이 들어가지만 서버에서 프레임이란 시간을 똑같이 일치하게끔 맞출 이유가 없다. 서버 역시 시간이란 개념으로 이동을 하기때문이다.**

픽셀이란 개념이 존재할 이유가없다.

서버는 그리드형태 / 타일형태로 만들어진다고 언급했다. 그럼 대부분 타일과 타일의 이동시간을 정할 것. 이를 200m/s로 가정한다면 서버에서는 200m/s가 지났다면 한칸이동.이런식.

만약에 오차가 발생하면 우리는 그냥 렉을 느끼고 느리게 가는 것. 좌표가 날아다니면서 동기화가 일어나기도 한다. 서버기준으로 가는것이기 때문.

**콘솔은 하드웨어적으로 프레임을 맞춰 화면주사율과 똑같이 맞추는 것을 원칙으로 가기 때문에 무조건 하드웨어 스팩에 정해진 프레임에 완벽하게 동기화를 한다.**

60프레임이라면 무조건 모든 스레드자체를 60프레임에 완벽하게 동기화를 하는 구조로 만들어지고있다. ( 플레이스테이션, 엑스박스 모두해당됨 )

**( 어플리케이션이 아예돌지않기 때문에 컨텍스트 스위칭도 없을 뿐더러 코어마다 배정한 스레드로만돌기 때문에 완전히 맞추는 것이 가능 )**

우리입장에서는 이렇게 완벽하게 맞추는 것이 불가능하기 때문에 이정도 느낌으로 로직을 최대한 따라잡는 형태로 만들어 보겠다는 겁니다.

=====================================================================================

게임프레임 : 50fps

화면사이즈 : 640x480

컬러 비트 : 32bit

# 화면 이동영역 ---------------------------

#define dfRANGE\_MOVE\_TOP 50

#define dfRANGE\_MOVE\_LEFT 10

#define dfRANGE\_MOVE\_RIGHT 630

#define dfRANGE\_MOVE\_BOTTOM 470

위 좌표에 가지 못하게 해야하며,

해당 좌표에 닿는 경우 움직임을 멈추어야 함.

(작거나 같으면 멈춤)

예) 왼쪽 상단으로 이동 중 왼쪽 범위에 걸리면

위로 타고 이동하는게 아니며, 그자리에 멈춰야 함.

왜냐면!! 서버에서 멈출것이기 때문에.. 서버랑 똑같이 하기 위함임.

# 스프라이트 중점 정보 ----------------------------

캐릭터 스프라이트 - 71, 90

이펙트 스프라이트 - 70, 70

HP 게이지 - 0, 0

그림자 - 32, 4

# 이동 픽셀

X 축 - 3 Pixel 초당 150 픽셀

Y 축 - 2 Pixel 초당 100 픽셀

# 애니메이션 프레임 딜레이 ------------------------

#define dfDELAY\_STAND 5

#define dfDELAY\_MOVE 4

#define dfDELAY\_ATTACK1 3

#define dfDELAY\_ATTACK2 4

#define dfDELAY\_ATTACK3 4

#define dfDELAY\_EFFECT 3

각 액션별로 해당 애니메이션 루프 시 위 수치만큼 프레임을 넘긴 후

다음 애니메이션 스프라이트로 넘어가야 함.

예) Stand . 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 으로..

프레임딜레이를 어택1번이 3딜레이로 했으므로 1 1 1 2 2 2 3 3 3 ..(MAX?) 이런식으로 넘어간다는 뜻.

**얘를 한프레임당 20m/s로 봤을 때 얘는지금 60m/s \* 3 후에 다음정지동작이 된다는 뜻이므로,**

**이 180m/s동안에는 다른키는 입력받지 않아야 한다.**

내가 주먹키를 연타로 누른다고해서 또다시 주먹이 나가면 안될 것.

해당 액션이 끝나야 이동을 할수도있고 끝나야지 다른공격을 수행할수 있을 거잖아요.

만약 네트워크 상태에서 다른사람 컴퓨터와 내 컴퓨터에서 액션에 대한 애니메이션에 대한 속도가 달라진다면 문제가 생길수도있음.

쟤는 공격을 두번했는데 나는 한번밖에 못하는 현상이 나올수있기 때문이다.

이부분도 맞추기를 권장한다.!