|  |  |  |  |
| --- | --- | --- | --- |
| [Cranding(크랜딩)] | | | |
| **주차** | **2023.12.24 ~ 2023.12.30** | **작성자** | **정경서** |
| **주간 회의** | | | |
| **회의 날짜** | | 2024.01.03 | |
| <캐릭터 별 애니메이션 지정>   1. 얼음 지형 보스 스킬  * 1번 스킬 → spin Attack * 2번 스킬 → Icicle Attack Ground * 기본 공격 → Fly Forward In Place 속도 2배정도로 조절하여 사용  1. 얼음 지형 일반 외계인  * 모든 애니메이션 사용  1. 불 지형 보스 스킬  * 왼손, 오른손 공격 * cast spell로 불 뿜기(파티클로 효과 적용)  1. 불 지형 일반 외계인  * 왼손, 오른손 공격 * 평소에는 걷기 애니메이션 * 플레이어 발견 시 뛰기 애니메이션  1. 자연 지형 보스 몬스터  * 점프 애니메이션 → 하늘에서 돌 떨어지도록 구현 * head attack  1. 자연 지형 일반 외계인  * 기본은 spin 애니메이션으로 이동 * head attack, projectile attack  1. 플레이어 애니메이션은 다음주에 정하기 | | | |
| **이번주 한 일** | | | |
| 1. 신동엽(서버)   2주차  2주차  C++ 복습  객체지향..   * 여지 껏 코딩해온건 절차지향이라 봐도 무방하다. * 객체지향 : 클래스 , 객체, ( 모든 오브젝트 , ex : 플레이어, 몬스터, Gameroom..등 ) * 클래스는 하나의 설계도라고 생각하는게 좋을 것 같다. * 예를 들어 Player를 설계한다고 가정하였을 때 속성으론 HP, Att, Pos등 이 있고, 기능(동작)으로는 Move, Attack, Die 와 같은 것들이 있다. 위에서 본 속성을 멤버변수로, 아래와 같은 동작들을 멤버 함수로 두어 설계해보자   클래스   * 굉장히 특별한 함수 2종, 생성자와 소멸자 [시작]과 [끝]을 알리는 함수들 * 주의 : 생성자와 소멸자를 통해서 만들어지고 소멸이 진행되는 것이 아닌 그 시점에서 실행되는 함수일 뿐이다. * 생성자에는 암시적 생성자와 명시적 생성자가 존재 * 생성자 종류에는 기본생성자, 복사 생성자, 기타 생성자 , 타입변환 생성자 등 * 암시적 형변환 : 컴파일러가 알아서 형변환을 진행, 아무런 행동을 하지 않아도 바꾼다. * 명시적 형변환 : ex int n; float f = (float)n; 이렇듯 직접적으로 우리가 변환을 주문하는 것이 명시적 * 암시적 형변환으로 인한 문제가 발생할 수 있다. 동작하면 안되는 것들을 컴파일러가 알아서 형변환을 진행해서 집어넣기 때문에 오히려 문제를 찾기가 더 힘들어 질수도. * 암시적 형변환을 막는방법 explicit * Explicit는 명시적인 용도로만 사용할 것이라고 알리는 것   상속성   * 공통적인 데이터들을 묶어 주기 위한.. 서버 프레임워크 제작시 상속을 적극 활용해보자 * 상속 ( inheritiance) : 부모에게 유산을 받는다고 생각 * 상속 받은 클래스의 생성자는 부모 생성자가 호출? 아니면 상속 클래스 생성자가 호출? * 결과는 둘다 호출 되지만 명확한 순서가 존재한다. 무조건 부모 생성자가 먼저 호출 소멸은 역순으로   은닉성   * 캡슐화(Encapsulation) 안전하게 하기 위해 접근을 제한한다. * 몰라도 되는 것은 깔끔하게 숨기겠다는 아이디어로 출발 * Public (모두가 접근), protected( 자식 클래스에게만 허락) , private (오직 나만 사용가능)   다형성   * 다형성(Polymorphism) * 오버로딩 : 함수 중복 정의 , 함수 이름의 재사용 * 오버라이딩 : 재정의, 부모 클래스의 함수를 자식 클래스에서 정의 * 바인딩(Binding)   + 정적 바인딩 ; 컴파일 시점에 결정     - 일반 함수는 정적바인딩을 사용   + 동적 바인딩 : 실행 시점에 결정     - 가상함수 (virtual function)       * 가상 함수 테이블 ( vftable )       * 사용해야 하는 이유? : 원본 객체에 따라가지고 다르게 동작해야 되는 경우에 가상함수를 사용한다. 우리가 만드는 게임에는 굉장히 만힝 그럴만한 일이 존재한다. 하다못해 객체들의 움직임을 제어하는 함수 또한 그렇게 구현해보면 괜찮을 것 같다.   서버 프레임워크 설계  기본적인 네트워크 부분부터 클래스 별로 설계를 해봤다.    Network.cpp   * 윈속 초기화 하는 부분과, accept하는 부분과 WorkerThread 함수를 구현 * 이전에 게임서버 프로그래밍 텀프로젝트에선 파일분할과 클래스 활용을 아예 하지 않았을 정도로 미흡하여 이번에 졸업 작품을 진행하면서 많이 신경써서 해봐야 할 것 같다.   ( C++ 꾸준히 )   * network에서의 멤버변수들에 대해 getter와 setter들을 다 만들어 변수에 대한 직접적인 접근을 막았다. * 클라이언트를 관리하는 컨테이너는 우선 array로 만들었지만 , 사실상 최대 인원수가 정해져 있지 않기 때문에 array보단 다른 컨테이너를 사용하는 것도 괜찮을 것 같다는 생각이 들었다. * 각 쓰레드들은 vector로 관리를 해주었다. * Overlapped 또한 필요한 정보들을 추가해주기 위해 Over\_exp라는 클래스로 새로 구현 * 게임 오브젝트 클래스에선 기본적으로 오브젝트 들이 가지고 있어야 할 공통적인 속성과 기능을 추가 * Player는 Gameobject를 상속받고 클라이언트에 필요한 변수와 함수들을 추가하였다.   게임서버 공부   * Atomic   + All- or – Nothing   + 멀티 쓰레드 환경에서 데이터레이스를 막기 위한 방법 중 하나.   + 아토믹으로 코드를 범벅을 한다면?   + 아토믹 자체가 연산이 많이 느리기 때문에 꼭 필요한 곳에서만 사용해야 한다. * 결론 :   + IOCP에 대해 더 많은 공부가 필요   + C++ 오버로딩과 오버라이딩( 가상함수) 적극 활용   + 클라이언트 네트워크 X  1. 정경서(클라이언트)  * 참조자   C++에서 참조자 (Reference)란 변수에 대한 또 다른 이름을 제공하는 기능입니다. 이를 통해 변수를 간접적으로 조작할 수 있습니다.  참조자는 선언 시에 초기화되어야 하며, 한 번 초기화된 후에는 다른 변수와 연결될 수 없습니다.  포인터는 선언 시에 초기화가 되지 않아도 사용 가능합니다.  C++ 문법 상 상수 리터럴을 일반적인 레퍼런스가 참조하는 것은 불가능하다. 하지만 const를 통해 선언 한다면 가능하다.  -----------------------------------------------------------------------------------------------------------------------  int a, b;  int& arr[2] = {a, b};  위와 같은 레퍼런스의 배열은 불가능  -----------------------------------------------------------------------------------------------------------------------  int arr[3] = {1, 2, 3};  int(&ref)[3] = arr;  위와 같은 배열의 레퍼런스는 가능하다. 이때 배열의 크기는 반드시 명시해주어야 한다.  이유는 레퍼런스는 몇몇 경우를 제외하고 메모리 상에서 공간을 차지하지 않기 때문이다  -----------------------------------------------------------------------------------------------------------------------  int& function() {  int a = 2;  return a;  }  int main() {  int b = function();  b = 3;  return 0;  }  function() 함수는 참조자를 리턴하는 함수이다. 이때 a는 참조자를 리턴하는 동시에 사라지게 된다.  따라서 이미 사라진 a의 값을 통해 b의 값을 선언하려는 문장이 된다.  이와 같이 레퍼런스는 있지만 참조하던 값이 사라진 레퍼런스를 Dangling 레퍼런스라고 한다.  \*\*레퍼런스를 리턴하는 함수에서 지역 변수의 레퍼런스를 리턴하지 않도록 조심하자\*\*  -----------------------------------------------------------------------------------------------------------------------  int& function(int& a) {  a = 5;  return a;  }  int main() {  int b = 2;  int c = function(b);  return 0;  }  위와 같은 경우에서는 function 함수에서 인자로 받은 레퍼런스를 리턴해주고 있다. 따라서 function(b)를 실행하면 a는 b를 참조하게 된다. 따라서 값을 리턴한 뒤에도 문제가 발생하지 않는다.  이러한 경우 레퍼런스가 참조하는 타입의 크기와 상관 없이, 주소값 복사 한번으로 값 전달이 끝나게 된다.  -----------------------------------------------------------------------------------------------------------------------  #include <iostream>  int function() {  int a = 5;  return a;  }  int main() {  const int& c = function();  std::cout << "c : " << c << std::endl;  return 0;  }  위 경우는 값을 리턴하는 함수를 참조자로 받는 코드의 예제이다.  만약 const 없이 int& c로 값을 받았다면 위에서 살펴본 예제와 같이 함수의 리턴값을 참조하기 전에 값이 사라지기 때문에 댕글링 레퍼런스가 된다.  하지만 const를 통해 선언 된 참조자라면 가능하다.  씹어먹는 C++ 페이지에서 좋은 표도 훔쳐봤다.     * 애니메이션 관련 코드 분석 * 유니티에서 모델 추출 후 다렉 프로젝트에서 적용 * 이때 특정 모델에서 애니메이션이 되지 않는 오류 발생. 이를 해결하기위해 여러가지 방법을 시도하고 있습니다. * 또한 모델의 애니메이션 속도가 너무 빠른 경우가 있어서 이를 조절하기위해 단순히 애니메이션 클립에서 속도를 조절하는 방법 말고 다른 방식에 대해 공부해 보았습니다  1. 홍예나(클라이언트)  * 키 변경에 따른 애니메이션 변경 관련된 코드 분석 * fLength를 이용하여 키가 눌리지 않은 경우 run 애니메이션에서 idle 애니메이션으로 변경된다.   -----------------------------------------------------------------------------------------------------------------------  // 플레이어가 움직이면 애니메이션 바꿔주는 부분  void CTerrainPlayer::Move(DWORD dwDirection, float fDistance, bool bUpdateVelocity)  {  if (dwDirection)  {  m\_pSkinnedAnimationController->SetTrackEnable(0, false);  m\_pSkinnedAnimationController->SetTrackEnable(1, true);  }  CPlayer::Move(dwDirection, fDistance, bUpdateVelocity);  }  void CTerrainPlayer::Update(float fTimeElapsed)  {  CPlayer::Update(fTimeElapsed);  if (m\_pSkinnedAnimationController)  {  float fLength = sqrtf(m\_xmf3Velocity.x \* m\_xmf3Velocity.x + m\_xmf3Velocity.z \* m\_xmf3Velocity.z);  if (::IsZero(fLength)) // 이동을 멈춘 경우  {  m\_pSkinnedAnimationController->SetTrackEnable(0, true); //다시 idle 애니메이션 활성화  m\_pSkinnedAnimationController->SetTrackEnable(1, false);  // 애니메이션트렉위치 변경 함수, 달리는 애니메이션 트렉의 가장 첫 부분으로 다시 옮김  // ->달리다가 멈춘 후 다시 달릴 때, 달리는 애니메이션이 멈췄던 지점이 아닌 애니메이션의 첫 부분부터 다시 실행되도록 설정한 것  m\_pSkinnedAnimationController->SetTrackPosition(1, 0.0f);  }  }  }  -----------------------------------------------------------------------------------------------------------------------   * C++ 상속 공부   상속이란?  다른 클래스의 정보를 물려받아 사용하는 것  ex.  -----------------------------------------------------------------------------------------------------------------------  class Base {  std::string s;  public:  Base() : s("기반") { std::cout << "기반 클래스" << std::endl; }  void what() { std::cout << s << std::endl; }  };  -----------------------------------------------------------------------------------------------------------------------  class Derived : public Base {  std::string s;  public:  Derived() : Base(), s("파생") {  std::cout << "파생 클래스" << std::endl;  // Base 에서 what() 을 물려 받았으므로  // Derived 에서 당연히 호출 가능  what();  }  };  -----------------------------------------------------------------------------------------------------------------------  → Derived 클래스가 Base 클래스를 상속받겠다는 의미  → Base : 부모 클래스, Derived : 자식 클래스    이러한 모습이 됨  따라서 Derived에서 Base의 what 함수를 호출할 수 있게 됨  이를 오버라이딩(overriding)이라고 한다. (오버로딩(overloading)과 다른 개념)  -----------------------------------------------------------------------------------------------------------------------  public:  Derived() : Base(), s("파생") {  std::cout << "파생 클래스" << std::endl;  -----------------------------------------------------------------------------------------------------------------------  Derived a; 라는 코드를 실행하게 되면  Base()를 통해 부모 클래스의 생성자를 먼저 호출 후, Derived 생성자 호출됨  <접근 지시자 private, protected, public>  - `private` : (부모님들한테 안가르쳐 주는) 자신만의 비밀번호  - `protected` : 집 현관문 비밀번호 ( 가족들은 알지만 그 외의 사람들은 접근불가)  - `public` : 집 주소 (가족 뿐만이 아니라 다른 사람들도 알 수 있음)  <접근 지시자를 이용한 상속>  ex. `class Derived : public Base`,  `class Derived : private Base`,  `class Derived : protected Base`  - `public` 형태로 상속 하였다면 기반 클래스의 접근 지시자들에 영향 없이 그대로 작동함. 즉 파생 클래스 입장에서 기반 클래스의 `public` 은 그대로 `public` 이고, `protected` 는 그대로 `protected` 이고, `private` 은 그대로 `private`.  - 만일 `protected` 로 상속하였다면 파생 클래스 입장에서 기반 클래스의 `public` 은 `protected` 로 바뀌고 나머지는 그대로 유지됨.  - 만일 `private` 으로 상속하였다면 파생 클래스 입장에서 기반 클래스의 모든 접근 지시자들이 `private` 가 됨. | | | |
| **다음주 할 일** | | | |
| 1. 신동엽(서버)  * 클라이언트 네트워크 클래스 * 서버 프레임워크 Player 클래스, 패킷 정하기 Thread manager * 유니티를 통해 맵 제작 + 리소스 모으기 * C++ : 연산자 오버로딩 , 동적 할당 , 타입변환 , Array, Vector, map  1. 정경서(클라이언트)  * 적 애니메이션 구현 * C++ 공부 * Fbx 공부  1. 홍예나(클라이언트)  * 애니메이션 블렌딩 공부 * fbx 공부 * 적 애니메이션 구현 * C++ 상속 추가로 공부 | | | |
| **비고** | | | |
|  | | | |