|  |  |  |  |
| --- | --- | --- | --- |
| 도둑들 | | | |
| **5월 1주** | **2023. 4. 30 ~ 2023. 5. 6** | **작성자** | **정극훈** |
| **이번 주**  **목표** | **김혁동: FBX 인스턴싱 문제 해결**  **이도영: 서버 동기화**  **정극훈: 파티클 코드 개선, FBX 인스턴싱 문제 해결** | | |
| **4/30**  **일** | **이도영:** 8시 팀원과 모여서 merge 후 클라에 대한 설명  각 객체를 화며에 띄운 뒤 객체 별 움직이는 것을 화면으로 확인 위한 코드 작성.  현재 플레이어의 움직임 상태를 받아서 어떠한 움직임을 하고 있는지 알려줘야 됨.  플레이어의 현재 움직임 상태를 나타내는 type 넣기  - Protocol PL\_ACTION 관련 타입 추가   Move, jump, none, attack   Player 객체에 get set 함수 추가.   플레이어 움직임 관련 send\_manager에 추가  Action type으로 통해 이 id가 무슨 상태인지 ID 값에 따라 저장한 뒤 id에 따라 client에 뿌려준다.  Packet이 이벤트가 있을 때 모든 아이디값의 정보를 읽고 전달.  자기자신의 아이디  점프에 관한 것  Action\_type enum을 따로 줄지 jump에 관한 패킷을 따로 만들지 고민  Jump에 관한 패킷을 따로 만듬, 우리의 move 패킷은 x, z 좌표만을 갖고 있기에 y좌표 만을 관리하는 점프 패킷을 따로 만들기로 결정함.  패킷의 수가 늘어나는데 좋은 것인가? jump패킷은 계속 전송되는 것이 아닌 jump만 할 때 전송 하므로 바틀넥에 관한 것은 걱정할 필요 없음. 문제점은 jump패킷을 보내고 스페이스르 떼어도 jump하는 동안은 계속해서 움직여 주어야 함. -> 패킷을 게속 전송해야한다.  그레서 계속 move 관련 패킷을 보내주어야 함. Move에 y 좌표를 추가함.  Move 할 때에 x값과 z값만 받아오고 있음. y값을 받게 되면 jump 할 때에 오류가 발생하는 경우가 잦아 x z 값만 받아왔는데 이럴 경우 y 좌표 값이 변경되었을 때 다른 id 값에 전송할 때에 바틀 넥이 발생함. x, z 만 보내는 것이 y좌표까지 합쳐서 보낸 뒤 타입에 따라 플레이어의 type을 변경해주려 함.  또한 점프 계산식에 관련하여 오류가 발생 점프의 계산이 끝난 뒤의 값들이 제각각인 경우가 많음 y 좌표가 0이 되어야 하는데 -되는 오류 +되는 오류 점프가 제대로 이루어 지지 않음.  예외처리로 해결하려 함. State enum을 넣어 점프 종료시 점프 종료 패킷과 함께 y좌표를 0으로 변경하여 서버에서 클라이언트로 받을 때 각 packet\_manager에서 각 state의 상태를 변경해 주며 좌표 또한 변경하게 끔. 근본적인 계산 문제를 해결하지 않는 이상 이루어 지지 않는 해결책  Id 값 별로 어떤 id값의 데이터를 보내주는 지 나타낼 예정  충돌처리를 위한 map관련 서버 코딩  Map object를 저장하는 map\_object  Map정보를 불러오고 map대한 정보를 관리하는 map\_manager  Map에 대한 정보를 저장하는 map\_info 생성  Map\_manager의 LoadMap 함수에서 맵을 불러와 center position과 scale을 vector에 저장한 뒤  텍스트이(가) 표시된 사진  자동 생성된 설명  이러한 map data를 map\_object에 emaplace\_back 시킨다.    텍스트이(가) 표시된 사진  자동 생성된 설명  Map object에는 이와 같은 정보가 저장되며 is blocking에 의해 갈 수 있는 지형인지 아닌지 판단하며 OBJ\_TYPE을 통해 대략적인 어떤 지형인지 나타낸다.    그 뒤 BlockTileMap 에서 map\_obj 데이터를 받아 BLOCK 인지 UNBLOCK인지 판단 한 뒤  저장한다.  m\_tile\_map은 map\_info에 선언 된 클래스로 2차원 평면상 좌표 값과 그 좌표값의 map에 있는 오브젝트 타입을 저장한다. 맵의 크기를 180x180으로 줄여 일단 그리드를 180x180 2차원 배열로 만든 뒤 위와 같이 정보를 저장시킴. | | |
| **5/1**  **월** | **정극훈:** 로그인 화면에서 2dtext가 출력될 때 한글이 포함되어 있으면 오류가 발생하거나 출력이 제대로 되지 않는 문제를 인지  처음 2dtext를 만들었을 때는 한글이 정상적으로 출력됬었는데 로그인 기능을 점검하는 도중에 한글 출력에 문제가 있다는 것을 알았다. 우선 화면에 출력할 문자열 정보가 들어가있는 wstring에 한글만 포함되있을경우 프로그램이 실행되지 않았다. 그리고 영어와 한글을 동시에 넣고 출력할 경우 한글 부분이 깨져서 출력됬다. 이 문제의 경우엔 Text class를 바꾼 적이 없으므로 비주얼 스튜디오 설정문제라 생각해 인터넷에 한글깨짐 문제로 검색해보았다. 검색결과 Text class를 유니코드로 인코딩하여 저장하는 해결방법이 있었고 해당 방법을 사용한 결과 한글이 정상적으로 출력되며 한글만 출력해도 프로그램이 정상적으로 실행되는 것을 확인했다. | | |
| **5/2**  **화** | **정극훈:** 기존에 버그가 있었던 Jump 코드를 수정  기존 Jump 코드는 프레임타임을 측정해 1초동안 플레이어의 y좌표가 올라가고 다시 1초동안 y좌표가 감소하는 방식으로 구현됬었는데 서버에 접속하니 점프가 제대로 작동하지 않는 문제가 발생했다. 플레이어가 점프가 안된다거나 공중에서 잠깐 멈칫한다던가 하는 문제가 계속 발생해서 Jump 코드의 로직을 바꾸었다. 일단 Physics class를 플레이어에 적용해 중력의 영향을 받도록 하였다. 현재는 충돌처리가 없으므로 y좌표가 0이 되면 그대로 멈추도록 하였다. 중력의 영향을 주면 Jump 코드에서 하강하는 부분을 따로 구현할 필요가 없어진다. 그래서 처음에 상승속도를 주고 프레임마다 상승속도를 줄이는 방법으로 코드를 구현했다. 이러한 방법을 사용하면 속도가 +일때는 올라가고 -가 되면 자연스럽게 내려가게 된다. 이러면 플레이어가 공중에서 멈칫하지 않고 점프가 되자마자 바로 내려가는 문제도 해결된다. | | |
| **5/3**  **수** | **김혁동:** 제대로 된 서버 동기화를 위해 **서버 코드를 확인**함. 도영이가 짠 코드는 그냥 하나의 클라이언트가 서버와 연결만 할 수 있을 정도로만 코드가 짜여 있음. **네트워크를 연결하기 위해서는 모든 네트워크 객체를 관리하고, 그 네트워크 객체들 중에서 자기 자신에 대한 입력과 다른 플레이어에 대한 입력을 모두 받아올 수 있지 않으면 안됨**. 그러므로, 네트워크 오브젝트가 추가되는 경우 해당 네트워크 오브젝트에 대해 접근하여 데이터를 저장하고 변경하는 방식으로 진행하지 않으면 안 됨. 현재는 ThievesPacketManager라는 클래스에 **멤버 변수로 현재 클라이언트의 데이터를 그냥 받아와서 적용하는 방식으로 적용**이 되고 있음. 이 부분을 NetworkObj라는 네트워킹 오브젝트를 관할하는 클래스에서 관리하게 하고, 그 클래스의 데이터에 접근하여 데이터를 변경하는 방식으로 진행하지 않으면 안됨.  텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진  자동 생성된 설명  <현재는 패킷만 관리해야 할 패킷 매니저 안에 멤버 변수로 받은 위치를 주고 있음>    <이미 패킷 매니저 안에 네트워크 오브젝트의 데이터를 저장하는 변수가 있으므로, 해당 데이터안에서 접근하는 것이 맞을 것으로 보임>  도영이의 서버 코드를 분석하여 느낀 점은, **전체적인 서버의 구조가 굉장히 난해하고 유지 보수가 어렵게 되어 있는 것으로 판단**되었음.  먼저, 현재 단계에서는 **클라이언트 플레이어의 위치를 SceneManager에서 선언하여 사용하고 있음**. 이는 SceneManager에 플레이어가 소속되어 있는 것이 되므로, **인 게임 신에서 생성되어야 할 플레이어의 데이터를** **SceneManager 단계에서 미리 가지고 있다가 이용**하는 것이 됨. 물론 이렇게 할 수도 있겠지만, 그렇다면 현재 클라이언트가 아닌 다른 클라이언트로부터 받아올 데이터도 SceneManager에서 관리를 해야 하는 것이므로 이 부분은 **객체지향적으로 잘못된 부분**으로 보임.  이외에도, ThievesPacketManager가 **플레이어 하나를 위한 멤버 변수를 가지고 있는 등의 문제**도 있으므로 전반적으로 문제가 있음.  텍스트, 폰트, 스크린샷이(가) 표시된 사진  자동 생성된 설명  <SceneManager에 존재하는 플레이어 데이터. 여기에 있는 것은 구조 상 좋지 않음.>  텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진  자동 생성된 설명  <ThievesPacketManager에 존재하는 데이터. 패킷 매니저는 하나의 데이터만 관리하는 것이 아니므로 따로 이 곳에서 받아서는 안됨.>  현재 우리 졸업작품의 코드는 기본적으로 Component 구조로 이루어져 있으므로, **네트워킹을 총괄하는 컴포넌트를 하나 추가하여 네트워크의 전반적인 내용을 관리하도록 지정**하는 것이 좋아보임. 이 생각을 바탕으로 **대략적인 구조를 기획**함.  이미 서버 측에서 네트워킹으로 관리할 데이터 들의 초기 정보를 OBJ\_INFO 패킷을 통해 제공해 주고 있고, 이를 클라이언트 측에서 저장하는 위에서 언급한 m\_obj\_map라는 컨테이너가 있음. 그러므로, 이 **컨테이너에 데이터를 담아두고, 이 컨테이너를 통해 데이터 관리**를 하도록 함. 그렇다고 한다면 이 **받은 데이터를 실제 게임 오브젝트에 적용해 주는 컴포넌트가 필요**할 것으로 보임.  그러므로, **NetworkSystem이라는 컴포넌트를 만들고 이 컴포넌트에서 패킷을 통해 받은 데이터들을 처리**하도록 함. 기본적으로 **NetworkSystem의 업데이트가 실행이 되면, 네트워킹 타입에 따라 자신이 해야 할 네트워킹을 실행**하도록 함. 네트워킹 타입은 지금 당장으로서는 **PLAYER (현재 클라이언트의 플레이어), OTHER\_PLAYER (다른 클라이언트의 플레이어), NPC의 세개의 타입으로 구성**하도록 함.  각각의 타입에 대해 다른 방식의 데이터 처리를 해주어야 함. 각각의 타입은 다음과 같은 네트워크 데이터의 적용을 진행하도록 함.  **PLAYER** -> m\_obj\_map에 저장된 데이터를 가져와, 해당 데이터를 바탕으로 자신의 데이터를 이동시킴. 저장된 값에서 Position 값과 애니메이션 값을 적용해 줌. **PLAYER가 보는 방향인 Rotation 값은 클라이언트에서 계산하도록 되어 있으므로 해당 값은 적용에서 제외**함.  **OTHER\_PLAYER** -> m\_obj\_map에 저장된 데이터를 가져와서 그 데이터를 바탕으로 자신의 데이터를 이동시킴. 저장된 값에서 Position 값과 애니메이션 값을 적용해 줌.  **NPC** -> m\_obj\_map에 저장된 데이터를 가져와서 그 데이터를 바탕으로 자신의 데이터를 이동시킴. 저장된 값에서 Position 값과 애니메이션 값을 적용해 줌.  **설계한 대로 코드를 작성**함. 물론, 서버 동기화가 완료된 것이 아닌 기존 클라이언트와 서버의 연결에 대한 부분만을 작성을 완료한 것이므로 실제 서버 동기화를 완료하기 위해서는 이 구조를 바탕으로 게임 오브젝트에 NetworkSystem 컴포넌트를 추가하여 OTHER\_PLAYER 타입을 적용하는 것이 필요함.    <추가된 NetworkSystem 컴포넌트>  텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  <NetworkSystem 컴포넌트 코드 중 PLAYER 타입의 처리>  그러나, 위의 코드를 작성하는 도중 **서버에 존재하는 잘못된 코드들을 발견**하였음.  첫번째로, **sc\_packet\_obj\_info 패킷**에서 type 변수가 먼저 선언되어야 하는데 **변수의 순서가 바뀌어 패킷을 읽을 때에 타입을 제대로 읽지 못하는 현상**이 일어난 것을 발견하였음. 기본적으로 패킷을 읽어 올 때에 패킷을 size를 읽고 해당 사이즈만큼 가져온 뒤, size 다음에 있는 type 변수를 통해 어떤 패킷인지 구별하여 패킷의 처리를 진행하고 있음. 그러나, 패킷에 저장된 변수의 순서가 바뀌면서 id를 type으로 인식하여 패킷이 제대로 전달되지 않았음. 이 부분은 두 변수의 위치를 변경해 주는 것으로 바로 해결하였음.  텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진  자동 생성된 설명  <type과 id의 순서가 바뀌어 있다>  두번째는 **sc\_packet\_sign\_in\_ok 패킷이 오고 있지 않은 것을 인지**함. sc\_packet\_sign\_in\_ok 패킷의 경우 자신의 네트워크 상의 ID를 가져오도록 하기 때문에 **이 패킷이 전달되지 않으면 자기 자신의 네트워크 ID를 알 수 없으므로 초기화로 설정된 ID인 -1을 자신의 ID로 인식**하게 됨. 그렇기 때문에 서버로부터 받은 데이터를 네트워크 오브젝트에 넣어도 해당 **데이터를 꺼내 올 방법이 없게 됨**. 이 패킷은 cs\_packet\_sign\_in을 클라이언트에서 전달해 주지 않으면 주지 않는 것으로 보이므로 해당 데이터를 전달해 주도록 변경함. 물론, **이 부분은 실제로는 유저의 ID와 비밀번호를 받아 해당 유저가 DB에 존재하는 지 판단하기 위한 내용이 있을 것** 이므로, 이후에 **DB와 연결을 하게 된다면 변경**을 해 주어야 함.  텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진  자동 생성된 설명  <인게임 신을 로드 할 때에 cs\_packet\_sign\_in 패킷도 전달해 주도록 함>  현재 이동 처리 등을 간편하게 하기 위해 서버에는 **Look Vector를 전달하여 저장하게 하는 방식으로 Rotation의 처리**를 하고 있음. 그러나 이 값을 플레이어의 실제 Rotation 즉, **회전 축 기준의 각도 값으로 변경하기 위해서는 회전 행렬에 대한 처리가 필요**함. **원래라면 Up, Right, Look 세개의 벡터를 구해 회전 행렬을 적용**하거나 쿼터니언을 사용하여야 하지만, 우리 졸업작품에서는 **Y축의 회전(Yaw) 만을 얻어 오면 되도록 되어 있으므로 Look Vector의 x요소와 z요소만을 가져와 atan2**(180도에서 -180도의 360도 각도를 모두 얻어야 하므로 atan2를 사용하는 것이 바람직함)**를 하는 것으로 Y축의 각도를 얻어 올 수 있음**. 플레이어의 Rotation은 **Y축의 각도를 Radian 값으로 저장하고 있으므로 atan2함수를 사용한 그대로 사용**하면 됨. 이런 식으로 현재 클라이언트의 플레이어가 아닌 다른 플레이어 들이 보고 있는 곳을 보여줄 수 있게 되었음. 이 말은 **플레이어의 Rotation값의 동기화가 완료**되었다는 것임.    <받아온 데이터를 저장할 때 atan2 함수를 사용하여 Y축의 각도를 계산해 주었음>  **이도영:** Collisioner추가  Collisioner 이 파일에선 Collisioner 관련 클래스를 선언한다. Collisioner, BoxCollision, SpherCollision이 있으며, BoxCollision, SpherCollision은 Collisioner를 상속받아 작동한다.  Collision\_checker 추가  이 파일에선 충돌했는지 체크하는 함수들이며 두 선분에 관련된 코드들을 작성하였다. | | |
| **5/4**  **목** | **김혁동:** 이전에는 플레이어가 키보드를 통한 인풋 데이터를 넣어 주었을 때만 플레이어 이동이나 회전 상태의 움직임을 전달해 주도록 하였음. 이 부분을 Update 루프에 맞추어 실시간으로 움직임을 전달해 주는 것과 동시에 애니메이션의 동기화도 같이 해주기 위하여 move 패킷의 내용을 변경하였음.  텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진  자동 생성된 설명  <변경된 패킷의 실제 내용>  먼저 기존에는 어떤 방향으로 이동을 시작했는지를 클라이언트에서 확인하여 해당 방향의 번호를 보내주는 방식을 택했었으나, 바뀐 패킷에서는 char타입의 1바이트에 존재하는 8비트에 대하여 각각의 비트에서 해당 키보드의 버튼이 눌려 있는지를 계산하는 방식으로 변경하였음. 이를 통해 지속적으로 Input 데이터를 받아 실시간으로 데이터를 주고받을 때에도 문제가 없어짐.  텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  <비트 연산을 통해 버튼이 눌려져 있는지를 확인함>  또한, 현재 유저가 어떤 애니메이션을 사용하고 있는지에 대한 데이터(action\_type)도 추가하여 해당 애니메이션을 재생하도록 하는 방식으로 변경하였음.    <서버에서 애니메이션 인덱스를 받아 저장해 줌>    **디스커넥트를 처리**함. 기존 서버에서는 디스커넥트 시 아무런 처리도 해주지 않았으나 **디스커넥트가 실행되었을 때 Room에서 데이터를 지우고 현재 Room에 있는 모든 플레이어에게 디스 커넥트를 알리고 플레이어의 데이터를 리셋**하도록 함.  텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  <실제로 작성한 코드>  디스커넥트를 작성하면서, **게임이 시작되지 않은 경우**에는 **Room 안에 있는 유저에 대한 데이터를 처리하지 않게 되어 있으나** 이 부분은 **실제 Room이 구현된 이후에는 상태의 단계 별로 변경해 주어야 할 것**으로 보임.  **이도영:** 위의 함수들을 다시 map manager에서 사용하여 충돌처리를 체크하였음. CheckInRange코드를 통해 충돌처리를 체크함. 맵데이터를 받아서 이 범위안에 데이터들이 들어오는 것은 체크하였으나 true여야 하는 데이터들이 false로 되는 오류가 발견됨. Range를 측정할 때에 pos 관련 자료형의 차이가 있어서 값이 달라지나 보았지만 모두 float으로 통일해도 데이터가 달라짐  디버깅 후 Emplace\_back 할 때에 값들을 확인 모든 값들이 정상  그러던 중 center\_pos 와 size의 숫자 차이가 있는 것을 확인 57개 58개로 서로 다른 사이즈를 갖고 있어. 메모장에서 띄어쓰기가 안되있는 부분 수정.  그 후 메모장의 모든 값들을 getline 한 줄로 모두 받아 올 수 있도록 map.txt 수정 | | |
| **5/5**  **금** | **정극훈:** 로그인 화면에서 ID와 Password를 입력받아 올바른 정보를 입력하면 Game 씬으로 전환하는 코드 구현  입력은 Input class에서 키보드 입력을 받고 KEY\_STATE를 변경시키는 부분에 받은 키보드 입력을 \_userID, \_userPassword에 넣는다. 처음에 \_userID를 입력받고 엔터키를 누르면 \_checkIDorPassword 변수를 변경해 다음 입력부터 \_userPassword를 입력받는다. \_userPassword 입력이 끝나고 다시 엔터키를 누르면 LoginInput 스크립트에서 기존에 저장된 정보와 현재 입력받은 ID, Password를 비교해 일치하면 씬을 전환, 틀리면 \_userID와 \_userPassword를 초기화한다. 현재는 서버에 ID와 Password 정보가 없기 때문에 클라이언트에서 임시로 지정한 값을 입력하면 씬이 전환되도록 만들었다. 현재 입력받은 ID와 Password를 확인할 수 있도록 입력한 정보가 2dtext로 화면에 출력되도록 코드를 구현했다. | | |
| **5/6**  **토** | **정극훈**: 김혁동이 부탁했던 조명의 On/Off 기능을 구현  조명의 조작을 위해 LightEffect라는 새로운 스크립트를 만들었다. 추후에 추가할 빛과 관련된 조작은 이 스크립트를 통해 동작할 예정이다. 우선 기존에 만들어놨던 light코드를 점검했다. 기존에 LoadScene부분에서 주석처리해놨던 light코드들은 정상적으로 작동했다. 이제 조명을 on/off하는 스위치 기능을 구현하기 위해 Light class에 \_lightState 변수와 관련 함수들을 추가했다. \_lightState는 라 해당 빛의 렌더링의 여부를 결정한다. 그리고 LightEffect 스크립트에서 키를 입력받고 조명의 스위치 기능을 구현했다. 처음에는 특정 키를 누를 경우 this를 사용해 해당 빛을 조작했으나 이 경우엔 스크립트가 적용된 모든 빛이 동일한 조작의 영향을 받게 된다. 즉 내가 원하는 빛만 조작하는 것이 아닌 스크립트가 적용된 모든 빛이 일관되게 바뀐다. 이러한 코드는 추후에 문제가 발생할 수 있을 것이라 생각해 light를 생성할 때 SetName을 통해 이름을 생성하고 LightEffect 스크립트에서 특정 키를 누르면 내가 원하는 이름을 가진 빛만 조작되도록 코드를 구현했다.  처음 구현한 코드    수정한 코드    현재 수정한 코드는 모든 게임오브젝트를 루프를 통해 검사한다. 나중에 최적화에 문제가 생긴다면 스크립트를 여러 개 만들어서 this를 통해 바로 빛을 조작하는 방식을 사용해야 할 수도 있다.  아무 조명도 끄지 않은 상태    Directional Light를 끈 상태    Spot Light와 Point Light를 끈 상태 | | |
| **이번 주 문제점 및 해결방안** | **클라이언트: 하나의 FBX 메쉬를 여러 개 생성할 때에, 생성되지 않는 문제 수정중** | | |
| **다음 주 계획** | **정극훈: 캐릭터 앞에 손전등을 사용한듯한 빛효과 구현, 게임로직 구현**  **이도영: 충돌체크 마무리**  **김혁동: FBX 인스턴싱 문제 해결, 서버 게임로직 구현, UI 그래픽 제작** | | |
| **비고** | Github 도둑들 주소: <https://github.com/rmrgns/Thieves.git>  Github 졸업작품 회의록 주소: <https://github.com/rmrgns/gameproject_proceedings.git> | | |