|  |  |  |  |
| --- | --- | --- | --- |
| 도둑들 | | | |
| **5월 3주** | **2023. 5. 14 ~ 2023. 5. 20** | **작성자** | **정극훈** |
| **이번 주**  **목표** | **정극훈: 캐릭터 앞에 손전등을 사용한듯한 빛효과 구현, 게임로직 구현**  **이도영: 충돌체크 마무리**  **김혁동: FBX 인스턴싱 문제 해결, 서버 게임로직 구현, UI 그래픽 제작** | | |
| **5/14**  **일** | **이도영:** 충돌처리 oriented bounding box로 다시 제작  OBB AABB와 다르게 가상의 선에 투영하여 투영시킨 영역을 이용해서 충돌되었는지를 감지하는 알고리즘   * 움직이고 있는 사물의 방향까지 고려하기 때문에 aabb보다 더 충돌체크를 정확하게 할 수 있다.   도표, 라인, 스케치, 평면도이(가) 표시된 사진  자동 생성된 설명  원리   * A중점과 B중점의 거리는 T라고 하면 T또한 projection이 가능 * T를 A에서 B로 가는 방향성이 가진 벡터로 가정하면 L과 T를 내적 하여 구하면 충돌의 범위가 나오게 됨. * 그 다음 ra와 rb가 projection된 범위 T, L내적보다 작으면 충돌이 발생하지 않고 있다는 뜻이다. 반대로 T,L내적보다 ra rb합이 더 크면 충돌이 발생했다는 뜻이다. 수식으로 나타내면 T L<= ra+rb(충돌)   ra와 rb를 구하는 방법은  도표, 라인, 스케치, 삼각형이(가) 표시된 사진  자동 생성된 설명  분리 축 벡터 L과 벡터 a1+a2의 내적 값 rb도 마찬가지이다.  이와 같은 원리를 직육면체를 계산하기 위해 서로 평행한 것을 제외하고 하나씩만 골라내면 면 법선 벡터 3개 변 3개 3 + 3 (3 \* 3) = 15개가 된다. 그러므로 15번의 검사로 충돌 여부를 알아낼 수 있다.  정리, OBB vs OBB 충돌은 두 물체 간의 접촉 유무를 확인하기 위해선 분리축 후보 15개를 조사하여 판단할 수 있음. 그 15개의 분리축을 각각에 두 직육면체를 투영시켜 그 투영범위가 겹치는가를 조사하면 됨. 충돌하는 경우 15번만에 충돌결과를 알 수 있지만 충돌하지 않는 경우는 그 전에 분리 축이 발견함.  이를 토대로한 OBB와 OBB의 충돌을 처리하는 BoxBoxintersection 함수를 제작. | | |
| **5/15**  **월** | **정극훈:** 어두운 백화점에서 손전등으로 앞을 밝힌다는 게임 컨셉을 구현하기 위해 FlashLightScript를 생성했다. 이 스크립트는 플레이어 객체에 컴포넌트로 추가하며 플레이어의 앞을 비추는 손전등에 관련된 코드를 구현할 것이다. 손전등의 이동과 관련된 코드는 기존의 카메라 스크립트 코드를 참고하여 넣었다. 이동은 바로 구현이 됬으나 손전등의 회전에서 문제가 발생했다. 빛을 구현하는 Light class는 빛의 회전을 GetTransform에서 SetLocalRotation이 아닌 LookVector를 이용해 SetLightDirection을 사용해 회전한다. 그런데 이 함수를 사용할려면 매개 변수로 LookVector를 전달해줘야 하는데 기존 코드는 Light의 LookVector를 구하는 함수가 없다. 그래서 스크립트 내에서 벡터를 계산해서 값을 바로 넣으려 했는데 손전등이 회전을 하지 않는다. 아무래도 계산식을 바꾸던지 Light의 벡터를 구하는 함수를 추가하던지 해야겠다.  **이도영:** 충돌하는 박스에 대한 CollisionBox 정보를 저장할 CBox 헤더파일과 CBox cpp 파일을 만듬.  CBox에 들어갈 정보는  텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진  자동 생성된 설명  Center axis(right up look 벡터저장) extent translation   * 이 값들을 통해 충돌처리를 함.   MapManager에서 loadmap을 통해 모든 맵을 받아온 뒤 MapCBox에 push\_back 하여 생성자를 통해 맵의 데이터를 저장합니다.  플레이어의 위치가 변화 할 때에 마다 플레이어의 center axis extent translation 값을 받아와서  텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진  자동 생성된 설명  CheckCollision 함수로 이동합니다.  텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진  자동 생성된 설명  이 함수에서는 CBox에 멤버함수인 충돌 검사하는 BoxBoxIntersection으로 player의 정보를 전달하여 map의 모든 객체와 충돌을 검사하게 됩니다.  BoxBoxIntersection은 위에서 학습한 이론을 토대로 작성한 OBB와 OBB의 충돌을 검사하는 함수입니다.  투영하였을 때의 중심 사이의 거리를 비교하여 큰지 작은지를 비교하여 충돌하지 않을 시 true를 리턴 15개의 비교 값이 모두 false이면 충돌한 결과로 나오게 된다.  이를 토대로 충돌 처리하는 함수를 완성하였다.  하지만 맵에서 충돌이 잘되는 부분이 있는가 하면 충돌이 잘 되지 않는 부분이 있기도 하다. 빈 땅에서 충돌 처리되어 충돌이 아예 되지 않는 경우가 발생함.  **김혁동:** 우리 게임에는 아직 데이터를 로딩할 때에 로딩을 하고 있다는 것을 알려주는 **로딩 창(로딩 신) 기능이 존재하지 않음**. 실제 게임에서 로딩창이 존재하지 않는다면 게임이 멈춘 것처럼 보이므로 이 부분에 대한 개선이 필요할 것으로 보였음. 이 생각을 바탕으로 **로딩 Scene을 생성하기 위해 대략적인 계획을 작성**함  텍스트, 화이트보드, 친필, 실내이(가) 표시된 사진  자동 생성된 설명  **<Loading Scene 구조에 대한 고찰>**  위는 현재 우리 게임 코드의 구조에서 어떤 부분을 변경해야 로딩 신을 구현할 수 있을 지, 어떤 부분에 대해 조심해야 할지에 대해 대략적으로 그리고 써본 내용임. 아래의 내용을 대략적으로 설명해 보자면, **SceneManager에서 Loading을 시작했다는 것을 캐치하면 Loading Scene을 임시로 사용**하도록 하고, Loading Scene을 루프 시키다가, 뒤에서 **데이터 로드가 모두 끝나면 다시 사용 Scene을 변경하여 적용**하도록 하였음.  해당 부분을 작성하는 경우, thread를 나누어 **Loading Scene을 표시하는 부분을 기존의 메인 스레드**로 둔 채로, 데이터를 로딩하는 **Loading Thread를 생성하여 로드**를 하는 것이 가장 쉽고 효율적으로 보였음. Thread를 생성한 뒤, join이 아닌 **detach 함수를 사용하게 되면, 해당 스레드를 기다리지 않고 넘어가게 되므로**, 자연스럽게 **Loading이 끝난 뒤, 스레드는 파괴**되며 스레드가 **파괴되기 직전에 로딩된 데이터를 넘기고, 신을 변경**하면 되는 것으로 보였음.  구조 계획 자체만으로는 어떤 문제가 있을 지 아직 알기 어려워, **직접 구현해 본 뒤에 문제점에 대해 생각**해보기로 하였음. | | |
| **5/16**  **화** | **김혁동:** 위에서 **계획한 내용을 바탕으로 로딩신을 구현**하였음.  텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  위와 같이, LoadScene 함수가 불리는 경우 로딩 함수를 곧바로 부르는 것이 아니라, **일단은 LoadingScene으로 변경**하게 됨. LoadingScene은 그리 무거운 데이터를 넣지 않거나, 게임을 기동할 때에 데이터를 미리 읽어 두고 해당 데이터를 표시하는 방식으로 진행하면 굉장히 빠른 속도로 바뀌게 될 것임.  일단 **LoadingScene을 로딩한 뒤**, LoadingScene도 Scene이므로, Scene에 대한 **Awake함수와 Start함수를 호출**하여 준 뒤, 목표로 하고 있던 Scene의 로딩을 하기 위해 **thread를 생성하고, 해당 신을 만들어진 thread에서 로드**하도록 함. 이를 바탕으로 **현재 Scene은 Loading Scene이므로 Loading Scene을 표시**하게 되고, **새로 생성한 다른 스레드에서 로딩을 시작**하도록 함.  텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진  자동 생성된 설명  로딩 스레드에서 **로딩이 끝나기 직전에, 로딩을 한 데이터를 Scene으로 저장**하고, **ChangeToLoadedScene이라는 함수를 호출**하도록 하였음.  텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  **ChangeToLoadedScene** 함수에서는 **데이터 로드가 완료된 Scene을 현재 Scene으로 변경**하고, 해당 Scene이 깨어날 때 해야하는 **Awake와 Start 함수를 실행**해 주도록 하였음.  위의 코드들을 바탕으로, **로딩이 시작될 때에는 Loading Scene을 표시하고, 로딩이 완료된 뒤에 로딩한 Scene을 표시하는 형태의 코드가 완성**되었음.  그러나 위에서의 문제점은 **Loading Scene이 로드 되고 난 뒤, 생각보다 다음 Scene의 로드가 빨랐을 경우, Loading Scene의 Awake나 Start가 불리기도 전에 Loading이 끝나 순서가 어긋날 수 있다는 문제**가 있음. 이는, Loading Scene**에서 자신의 Load 함수가 끝나기 전에는 thread가 대기하고 있도록 설정을 해 주어야 할 것**으로 보임. | | |
| **5/17**  **수** | **정극훈:** 해상도 변경 옵션을 추가하기로 했다. 일단 프로그램이 해상도 변경이 잘 되는지 확인했다. 프로그램을 킨 후 해상도를 변경했더니 창 자체는 정상적으로 크기가 변경됐다. 하지만 로그인 화면에 배치한 이미지나 UI, Text의 위치가 바뀌지 않았다. 처음 객체들을 배치할 때 좌표를 해상도와 관계없이 절대값으로 넣어 놔서 그렇다. 그래서 이 좌표들을 해상도에 따라 상대적으로 바뀌도록 구현했다. 그런데 로그인 화면의 이미지가 이상하게 변경되는 문제가 있었다. 현재 로그인 화면의 이미지는 플레인으로 생성했고 이걸 카메라로 보는 방식인데 여기서 이미지의 크기를 건드리는 부분이 문제가 되는 것 같다. 그래서 2D이미지의 경우 이전에 구현했던 direct11on12를 이용해서 텍스트를 생성했던 것처럼 2dbitmap이미지를 생성해서 출력하려고 한다.  **이도영:** 맵이 많이 비어 보인다는 의견이 있어 맵을 재기획. 맵 제작에 앞서 맵에서 가장 중요하게 생각하고 있는 부분은 다이아 방으로의 유도 및 엄폐에 대한 레벨디자인이라고 생각. 아무리 엄폐를 하더라도 다른 한 방향에서는 보일 수 있도록 기획하여 한 곳에서 대기하여 숨어있거나, 싸우지 않고 도망쳐 게임의 재미를 떨어트릴 수 있는 행위를 없애려 함. 그와 더불어 맵의 이동에 조금은 자유도를 두며 다이아방으로 인도할 수 있도록 기획하고자 함.  현재 우리의 맵의 문제점은 캐릭터의 크기에 비해 맵안에 있는 객체의 크기가 너무 큰 문제점과 맵 안에 비어 있는 문제점을 해결하기 위한 대안   * 추가 에셋 구매 후 배치 * 우리의 생각과 부합하는 맵 객체 제작   두 가지의 방법을 생각하였음.  넘을 수 있는 벽을 맵에 배치해 엄폐와 자유도를 동시에 줄 수 있게끔 생각도 하였으나 이 부분은 혁동과 대화가 필요하여 주말 회의 때에 다시한번 이야기할 예정 | | |
| **5/18**  **목** | **정극훈:** 이미지를 출력하기 위해 DrawBitmap를 사용하려고 했더니 함수에 매개 변수로 전달할 Bitmap 객체가 필요했다. 처음 생각했던 것은 파일의 이동경로를 이용해서 이미지를 생성하려 했는데 다른 방식이 필요한 듯하다. 그래서 Bitmap 객체를 만드는 함수를 살펴봤는데 CreateBitmapFromWicBitmap이란 함수를 통해 생성한다는 것을 알았다. 그런데 비트맵 생성과 관련된 코드와 요소가 매우 많았다. 새로 추가해야 할 코드가 매우 많아서 작업량이 생각보다 많아 우선 손전등 코드를 구현하고 이후에 구현해야겠다.  **이도영:** 맵의 중앙 섹터에서는 충돌 처리가 잘되나 그 중앙 섹터를 나가 이외의 곳으로 갈 시에  맵의 빈공간에서도 충돌체크가 되어지는 버그수정을 위한 디버깅 과정에서 몇몇 오류발견   * 맵의 모든 데이터를 비교하지 않음. * Extent에 쓰레기 값이 저장되는 경우   예외처리르 통해 모든 데이터를 비교함.  메모장에 몇몇 오타가 들어가 있는 것의 삭제를 통해 오류 수정  **김혁동:** 서버 코드에서 Room 안에서 **시간을 저장하는 방식이 이상했음**.  기존에는 **time point에다가 계속해서 시간을 더하여 현재 시간을 만드는 이상한 방식**을 택하고 있었는데, 이 부분을 **설정한 시간으로부터 얼마나 지났는 지를 받아오는 함수로 변경**해 주었음.  텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진  자동 생성된 설명  <새로 변경된 GetRoundTime 함수>  도영이에게 충돌처리를 모두 맡기고 내가 게임 로직을 전담하기로 하였으므로, 게임 로직을 바꾸기 위해 다시 한번 **서버 코드를 분석**하기 시작하였음.  **현재 코드는 로그인 성공 패킷을 전해줌과 동시에 게임의 시작을 전달하여 클라이언트에서 게임으로 바로 들어가도록 설정**해 두었음. 그러나, 실제로는 **로그인**을 하면 **로비로 이동**을 하고, **룸을 선택하여 접속**한 뒤, **룸에서 레디**를 하고 모든 사람이 레디가 된 뒤에 **게임 스타트**를 눌러야 실행이 되는 형식임. 즉, **구현된 코드에는 생략된 부분이 매우 많고**, 게임 내에서 전달해 줘야 하는 공격, 상호작용, 페이즈 변경 등의 패킷까지 생각하게 되면 **너무나도 많은 부분이 구현이 안 된 것**을 알 수 있음.  일단은 게임 중의 패킷은 난이도가 낮을 것으로 판단하여, **로그인부터 게임 시작까지의 패킷에 대해 구현을 먼저 시작**하기로 결정하였음.  서버 코드를 분석하여 본 결과 많은 부분에서 문제가 있다는 것을 확인하였음. 먼저 **플레이어가 상속받는 것이 NPC나 움직이는 객체 들에 대한 데이터를 제어하는 MoveObjectManager를 상속받는 것을 확인**했음. Player는 항상 움직이는 객체가 되는 것이 아니고, Lua 등에 의해 제어되는 것도 아니므로, MoveObj가 아니라, Object만을 상속하여 아예 다른 객체로서 있는 것이 합당한 것으로 보임. 그러나, **MoveObj를 상속함으로 인해 의미 없는 데이터를 사용하게 되고, 메모리의 낭비와 더불어 탐색에서도 불리한 부분이 있을 것**으로 보임.  또한, **Room에서 플레이어의 데이터를 저장할 때에 Vector<int>로 저장하는 부분이 문제**가 있는 것으로 보였음. 물론 Room만 존재할 때에는 8명 이상의 플레이어가 존재하지 않으므로 문제가 되지 않을 수 있지만, Lobby 화면이 존재하는 경우 해당 Room의 데이터를 표시해야 하는 경우가 생길 때에 탐색에 시간이 걸리게 될 수 있음. BinarySearch를 하게 되면 탐색 속도가 빨라질 수도 있으나, 그 부분은 결국엔 플레이어 ID를 정렬해야 한다는 이야기가 되므로 플레이어의 숫자가 많아지게 되면 서버의 부담이 될 것으로 보임.  **이 부분들을 고려하여 다시 한번 서버 코드를 리팩토링 하게 되면 엄청난 작업량이 될 것**으로 보임. 그러므로 이 부분도 제대로 계획을 세워 진행해야 할 것으로 보임.  일단, 코드의 어떤 부분을 바꿔야 하는지에 대해 정리해 보았음.  클라이언트에서 **로그인을 하는 것과 동시에 게임을 시작하는 부분을 아래와 같이 변경**하여야 함.  로그인을 할 때, 로그인이 허가된 경우(로그인 허가는 DB 관련이므로 나중으로 미룸)에는 **로비로 이동**해야 함. 이 부분은 로비에 플레이어에 대한 데이터를 추가하거나 하는 것으로 충분할 것으로 보임.  로비에서 **로그아웃** 부분을 추가하여야 함. 이미 존재하는 플레이어를 지우는 것으로 끝날 것으로 보임.  로비에서 룸을 선택하는 경우, **룸으로 이동**하도록 함. 이는 Room 클래스의 코드에 존재하므로 변경할 부분이 크게 존재하지 않을 것임.  룸에서는 **레디와 레디 해제**가 있어야 함. 이는 플레이어의 State를 변경하는 것으로 괜찮을 것으로 보임.  룸에서 **로비로 나갈 수 있어야 함**. 이 부분도 Room 클래스에 구현되어 있으므로 해당 함수를 활용하면 될 것으로 보임.  **룸에서 GameStart**를 선택하면 모두 레디가 되었는지 확인한 뒤에 게임으로 이동하여야 함.  위의 부분을 간단하게 추가하면 되는 것으로 판단하였으나, 위에서 언급한 **구조적 문제를 해결하면서 바꿔야 하므로** **상당한 시간이 소요될 것**으로 보임. | | |
| **5/19**  **금** | **정극훈:** 손전등의 회전을 구현했다. 플레이어가 바라보는 방향에 따라 손전등이 비추는 영역이 회전한다. 해결방법은 이외로 간단했다. 플레이어의 LookVector를 받아온 후 SetLightDirection에 값을 전달하면 된다.  텍스트, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  현재는 Player 객체가 바로 값을 전달하는 것이 아닌 SceneManager로 값을 전달한 후 다시 스크립트로 전달한다. 이 부분은 스크립트에서 플레이어 객체를 검색한 후 바로 값을 전달하는 코드로 구현할 예정이다.  스크린샷, 3D 모델링, 그래픽 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진  자동 생성된 설명  **이도영:** 맵의 중앙 섹터에서는 충돌 처리가 잘되나 그 중앙 섹터를 나가 이외의 곳으로 갈 시에  맵의 빈공간에서도 충돌체크가 되어지는 버그수정을 위한 디버깅에 더불어 collision 관련 코드 리팩토리 아직 오류수정 안됨 주말에 회의 시간에 말한 뒤 collision data를 다시 받아서 진행해볼 예정 | | |
| **5/20**  **토** | **정극훈:** 보다 현실적인 손전등 효과를 위해 기존에 플레이어의 LookVector를 사용하는 대신 카메라의 LookVector를 사용하고 손전등의 위치도 적당히 수정했다. 스크린샷, 3D 모델링, 그래픽 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진  자동 생성된 설명  **도둑들 주간회의**  도둑들 맵 요소 추가 회의   * 이도영의 제안으로 데바데의 창틀과 같은 장애물을 추가하는 것이 어떤지 회의함 * 새로운 요소의 에셋은 이도영이 따로 구한다고 말함 * 새로운 요소를 추가하기 위해선 맵을 처음부터 새로 만들어야 한다고 김혁동이 의견을 제시함   클라이언트 회의   * 충돌처리 문제를 쉽게 확인하기 위한 OBB를 추가하기로 결정   김혁동의 게임로직 구상  현재  o를 눌러 사인 인 -> 바로 signinok를 받고 게임을 시작함  바꿔야 할거  o를 눌러 사인 인 -> 로비로 이동. (Robby에 플레이어 추가)  로비에서 선택 -> 룸으로 이동.  룸 선택  룸에서 해야 할 것  룸에서 나가기 -> 플레이어의 스테이트를 로그인으로 변경하고 룸에서 보유한 오브젝트에서 플레이어를 제거하기  룸에 들어가기 -> 플레이어의 스테이트를 인룸으로 변경하고 룸에서 보유한 오브젝트에 플레이어를 추가하기  룸에서 레디 -> 룸에서 레디한 플레이어를 설정하기  룸에서 레디 해제 -> 룸에서 레디한 플레이어를 설정해제 하기  룸에서 선택 -> 게임으로 이동.  서버 코드 회의  김혁동 의견   * 현재 Player 객체는 MoveObjManager라는 움직이는 객체 클래스에 상속을 받아서 생성됨 * 하지만 항상 Player가 움직이는 것은 아님 (ex 로비 씬) * 그러니 MoveObj가 아닌 Obj에서 상속을 받는 게 맞다고 생각 * 왜냐하면 Player가 필요한 데이터와 MoveObj에 필요한 데이터가 너무 다름 | | |
| **이번 주 문제점 및 해결방안** | **클라이언트: 하나의 FBX 메쉬를 여러 개 생성할 때에, 생성되지 않는 문제 수정중** | | |
| **다음 주 계획** | **정극훈: UI는 forward쉐이더로 생성하도록 변경, FBX모델의 그림자 구현, 콜라이더 데이터를 기반으로 OBB 추가**  **이도영: OBB 오류 수정, 슬라이딩 벡터 추가**  **김혁동: 게임 로직 구현** | | |
| **비고** | Github 도둑들 주소: <https://github.com/rmrgns/Thieves.git>  Github 졸업작품 회의록 주소: <https://github.com/rmrgns/gameproject_proceedings.git> | | |