3D게임2 프로그래밍

과제1

게임공학과 3학년 2020182028 이규원

목차

1. **과제에 대한 목표**

* 바운드 스페어를 통한 충돌처리 구현 및 적 회피
* 2D 텍스처를 활용한 시작 화면 구현
* 각 객체를 별도 클래스에서 구현.
* 스카이 박스, 지형 구현

1. **구현 내용**

* 시작 모드
* 인 게임 모드
* 플레이어
* 적
* 카메라
* 지형
* 리소스 관리

1. **과제에 대한 목표**

* 바운드 스페어를 통한 충돌처리 구현 및 적 회피

플레이어와 적을 바운드스페어로 감싸고, 충돌 상황을 처리해서 그에 따른 회피 매커니즘을 구현하고자 한다.

* 2D 텍스처를 활용한 시작 화면 구현

게임 시작 시, 2D 텍스처를 활용한 직관적인 시작화면을 제공해 조작법을 보기 쉽게 제공하는 것을 목표로 한다.

* 각 객체를 별도 클래스에서 구현.

플레이어, 적, 카메라, 지형 등을 하나의 클래스 혹은 헤더파일에 모두 구현이 아닌, 객체 지향적인 구현을 목표로 했다. 동시에 객체 간 상호작용이 자연스럽게 구현하고자 한다.

* 스카이 박스, 지형 구현

게임의 배경을 구현하기 위해 스카이 박스와 지형을 추가하고자 한다. 높이맵을 통해 지형의 높낮이를 표현하여 더 자연스럽게 표현하고자 하고, 스카이 박스를 통해 배경을 구현하고자 한다.

1. **구현 내용**

* **시작 모드**

게임 실행 시 초기 모드를 설정하기 위해, StartMode에서 enum으로 설정했다. 초기 모드에 따라 메인 화면, 인 게임 화면으로 진입할 수 있다.

2D그래픽으로 텍스쳐를 띄우는 클래스는 Guide.h에 들어있다.

Render에서 이미지의 크기를 받아오고, Scale을 통해 이미지 크기를 조정해 화면에 출력할 수 있다.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

* **인 게임 모드**

인 게임 모드로 전환되면 게임을 시작하면서 필요한 객체를 Start 함수에서 Scene에 추가 한다.

Scene.cpp는 카메라, 캐릭터 제어, 적, 지형 객체 등을 초기화해 모든 객체 업데이트와 렌더링을 다 담당하는 프레임워크다.

KeyboardController, MouseMotionController, MouseController 함수를 통해 플레이어의 키보드, 마우스 입력을 scene에 전달하는 역할을 한다.

야외, 하늘, 사람, 구름이(가) 표시된 사진

자동 생성된 설명

* **플레이어**

Control.h 파일에서 플레이어인 헬리콥터 클래스를 관리한다.

자료구조 : 플레이어의 위치, 크기, 회전율, 객체의 충돌 범위, 헬리콥터의 방향 벡터를 포함하고 있다.

InputKey : 키보드 입력에 따라 MoveForward, MoveBackward, MoveLeft, MoveRight 변수를 설정하여 객체 이동 방향을 지정한다.

InputMouseMotion : 플레이어가 클릭하면 마우스를 감추고 화면을 회전할 수 있게 구현했다.

float cxDelta = (float)(mouse.CurrentPosition().x - MotionPosition.x) / 5.0f;

식으로 마우스의 현재 위치와 이전 위치간의 차이를 구해 회전 변화의 방향을 결정한다.

Update : 플레이 동안 갱신되는 값들을 처리하기 위한 값으로, 플레이어가 바라보는 방향으로 입력한 방향키에 따라 이동할 수 있도록 계산했다.

Position.x += Vec.Look.x \* FT \* 40;

Position.y += Vec.Look.y \* FT \* 40;

Position.z += Vec.Look.z \* FT \* 40;

바라보는 방향에 따라 거리를 구하기 위해 벡터의 룩벡터를 통해 구현 했다.

Render : InitMatrix 및 Transform 함수를 사용해 헬리콥터의 몸통의 크기, 이동, 회전을 렌더링 할 수 있게 구현했다.

헬기의 머리는 헬기 몸통의 변환을 그대로 가져와 몸통보다 위로 움직이고, 회전시켜 마치 헬기의 날개가 돌아가 나는 것처럼 구현했다.

추가로, 스카이 박스 메시와 텍스처를 통해 스카이 박스를 구현했지만, 세 면이 어둡게 보이는 버그가 있었고 아직 문제 원인 파악을 하지 못해 해결하진 못했다.

바운드 스페어를 OOBB로 구현해 Scene에 등록되어 있는 enemy의 정보를 읽어와 플레이어와 적의 바운드 스페어의 충돌을 CheckCollision으로 검사해 충돌 검사를 하고 충돌 시 플레이어가 enemy에게 접근하지 못하도록 구현 했다.

* **적**

Enemy.h 파일에서 적의 클래스를 관리한다.

기본적으로 플레이어와 구성과 마찬가지다.

update를 통해 행동을 갱신하며, 위치에 따라 해당 위치에 렌더링 된다.

적 객체는 따로 사용자의 입력을 받아 움직이지 않고 자동적으로 좌우로 움직이게 구현했다.

* **카메라**

CameraController.h는 게임 내 플레이어의 카메라를 제어하는 클래스를 정의한다.

카메라의 회전을 부드럽게 하기 위해 lerp 함수를 사용 했다.

CamRotation.x = std::lerp(CamRotation.x, DestCamRotation.x, FT \* 15); 식을 통해 현재 카메라 각도와 목표각도를 선형보간을 통해 부드럽게 맞춰 자연스럽게 구현했다.

CameraUtil.cpp에서 카메라의 이동, 회전, 추적, 투영 설정 등을 정의하는 camera 클래스를 구현했다.

TrackWithOffset함수를 만들어 카메라가 특정 객체를 추적할 수 있도록 구현했다. Pos, Up, Right, Look값을 입력 받아 해당 방향을 바라볼 수 있도록 구현했고, Offset의 값을 입력 받아 해당 객체에서 원하는 만큼 떨어져 볼 수 있도록 구현했다.

* **지형**

Terrain.h에서 게임 내 지형 객체를 정의하는 클래스를 구현하고 있다.

Terrain을 생성할 때, x,z값을 입력받아 지형을 어디에 생성할 지 구현할 수 있게 만들었다. 지형의 크기를 맞추기 위해 적당히 크기를 늘리고, 다른 클래스와 다르게 지형은 렌더링만 구현되고 있다.

* **리소스 관리**

메시, 셰이더, 텍스처는 모두 GameResource.cpp에서 받고 있다.

셰이더 생성: CreateShaderResource 함수는 게임 객체 렌더링과 바운드박스 렌더링을 위한 셰이더 파이프라인을 구현한다.

메쉬 생성: CreateMeshResource 함수는 게임에 필요한 다양한 메쉬 구조를 생성하여, 화면에 출력할 그래픽 구조를 구현한다.

텍스처 생성: CreateTextureResource 함수는 이미지 파일을 텍스처로 변환한다.