2D Game // Breakout

-지금까지 OpenGL의 기본, 빛, 심화 내용, 심화 빛, PBR, 디버깅 방법, Text 렌더링 등 에 대해서 배웠음

-배운 내용을 토대로 간단한 2D 게임을 만드는 튜토리얼을 진행할 것임

-만들려는 게임 : 2D Arcade Game Breakout // 벽돌 깨기

-여기서 배울 수 있는 것 중, 가장 필요하다고 생각한 점이 바로 충돌처리임

-게임에서는 충돌 처리가 아주 빈번하게 필요로 하기 때문에 배우고 싶어서 진행할 예정

Breakout 메커니즘

-작은 패들을 플레이어가 제어할 수 있으며, 좌우로 움직이고 특정 영역을 넘어갈 수 없음

-공이 화면을 가로질러 이동하며, 공이 패들에 충돌한 위치에 따라서 날아가는 방향이 다름

-공이 패들의 중심에서 멀어질수록 방향이 많이 틀어짐

-공이 벽돌, 화면의 경계에 충돌하여도 충돌한 위치에 따라서 날아가는 방향이 다름

-공이 패들을 넘어 하단의 화면으로 넘어가게 된다면 플레이어는 라이프를 잃거나 게임오버

-벽돌은 공이랑 충돌할 시, 파괴됨

-모든 벽돌이 파괴되는 즉시 플레이어는 승리

OpenGL Breakout

-고전적인 breakout에서 OpenGL에서 그래픽 렌더링을 해서 추가 기능을 넣을 예정

-추가내용

-그래픽

-파티클

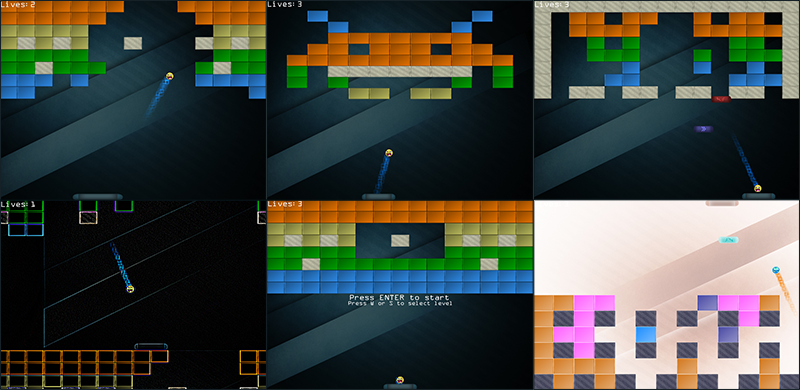
-텍스트 렌더링

-파워 업(능력)

-후처리 효과

-여러 레벨

-예상 결과물



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

Breakout // Setting up

-간단한 게임 프레임 워크가 필요함

-이 게임은 또한 여러 라이브러리를 사용함

-새로운 라이브러리가 필요할 때 소개할 예정

-먼저 모든 게임에 대한 렌더와 게임 코드에 관란 uber 게임 클래스를 정의함

-게임 클래스를 만드는 이유는 게임 코드를 구성하면서, 동시에 window를 띄우는 코드랑 분리하기 위해서임

-게임 클래스 함수는 초기화, 업데이트, 입력, 렌더링 함수를 호스트함

|  |
| --- |
| class Game  {  public:  // game state  GameState State;  bool Keys[1024];  unsigned int Width, Height;  // constructor/destructor  Game(unsigned int width, unsigned int height);  ~Game();  // initialize game state (load all shaders/textures/levels)  void Init();  // game loop  void ProcessInput(float dt);  void Update(float dt);  void Render();  }; |

-클래스는 게임에서 기대할 수 있는 호스팅을 함

-게임 해상도로 게임을 초기화, 쉐이더, 텍스처, 모든 플레이 상태를 초기화

-프로세스를 호출해서 입력을 처리

-업데이트에 게임 플레이 이벤트를 입력, 업데이트

-렌더를 불러서 게임을 렌더링(움직임 로직이랑 렌더 로직이랑 나눈 것을 확인)

-게임 state를 위한 Enum 생성

|  |
| --- |
| enum GameState {  GAME\_ACTIVE,  GAME\_MENU,  GAME\_WIN  }; |

-게임의 상태를 알 수 있게 진행 중, 메뉴, 이겼을 때 렌더링, 또는 처리를 조정할 수 있음

Utility

-대규모의 프로그램을 만들려면 텍스처나 쉐이더같은 여러 OpenGL 게념들을 자주 사용할 것임

-shader class를 항목에 대해서 더 사용하기 쉬운 인터페이스를 만드는 것이 바람직함

-shader class는 2개 또는 3개(기하학 추가)일 경우에 쉐이더를 생성하는 것을 정의

-uniform 값을 설정할 유틸리티 함수

-byte 배열과 주어진 너비, 높이에서 2D 텍스처 이미지를 생성하는 texture class 정의

-shader, texture에 대한 헤더와 클래스 파일을 생성하였음

Resource management

-shader, texture 클래스는 잘 작동하나, 초기화를 위해 바이트 배열이나 문자열 목록이 필요함

-파일 읽기 코드를 class에 넣어도 되지만, 이것은 single responsibility principle 단일 원칙 책임에 위배됨

-따라서 이러한 자원을 로드하기 위한 단일 entity는 종종 리소스 매니저라고 불림

-여러 방법이 있으나, 이번에는 정적 특성으로 프로젝트 전체에서 항상 사용할 수 있는 싱글톤 정적 리소스 매니저를 사용, 로드된 모든 리소스와 관련 로드 기능을 호스팅함

-싱글톤 정적 리소스 매니저

-단점 : 주로 OOP의 손실과 구성/파괴에 대한 제어가 줄어듬

-장점 : 비교적 규모가 작은 프로젝트의 경우 작업하기가 쉬움

-리소스 매니저 헤더와 코드 생성

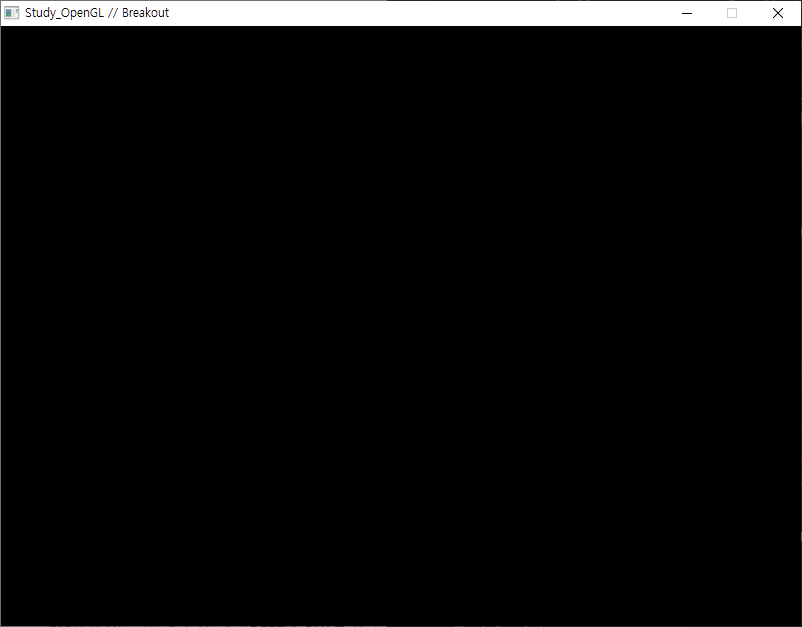
-사용방법

|  |
| --- |
| Shader shader = ResourceManager::LoadShader("vertex.vs", "fragment.vs", nullptr, "test");  // then use it  shader.Use();  // or  ResourceManager::GetShader("test").Use(); |

Program

-게임을 위한 window창, OpenGL 상태를 초기화

-2D이기 때문에 깊이 테스트를 사용할 수 없고, 모든 z값은 동일하게 정의됨

-breakout 게임을 위한 창

-GLFW window생성

-콜백 함수 등록

-game 객체 생성

-관련 기능을 game class에 전파