Assignment 1 Report

Team Name: Endgame Duo
Team Members:

Name: shcha1220@gmail.com
- **Department:** CSE
- **Student ID:** 20200120
- **HEMOS ID:** carprefer

Name: SOONHO (snow kim)
- **Department:** CSE
- **Student ID:** 20200703
- **HEMOS ID:** rlatnsgh0708

Technical Details

Development Environment:

- **IDEs:** Visual Studio Code
- **Framework Versions:**
  - OpenGL: 4.2 (Compatibility Profile) Mesa 23.2.1
  - GLEW: 2.2.0-4
  - freeglut: 2.8.1-6
  - GLM: 1.0.1
- **Git:** https://github.com/csed451/graphics.git
- **Other Tools:**
  - GCC(g++), WSL(Ubuntu 22.04)

Implementation Details

- **Program Features Outline:**
  - **Object Structure**:
    - 게임의 각 객체에 대하여 Object라는 기본 클래스 만들어 변환(translate, rotate, scale) 및 렌더링 로직을 공통으로 관리하도록 구현했다.
  - **Object Pooling**:
    - Player의 Attack과 Enemy의 Bullet처럼 자주 생성되고 소멸되는 객체들을 **ObjectPool 패턴**을 활용하여 관리했다. 이를 통해 동적 할당/해제로 인한 성능 저하를 최소화했다.
  - **Game State**:
    - GameState 열거형(enum)을 통해 Playing, GameOver 상태를 구분하고, state에 따라 게임이 동작하도록 구현했다.
  - **Player 무적 상태 및 Heart 시각화**:
    - Player가 피격 시, 일정 시간 동안 반투명 상태가 되는 **Recovery 상태**가 된다.
    - Player의 남은 생명을 화면 좌측 하단에 Heart모양의 객체로 표시했다.
  - **Enemy Bullet Patterns 및 Health Bar**:
    - Enemy는 두 가지 패턴이 번갈아 나타나는 나선형 탄막을 생성하도록 구현했다.
    - 화면 상단에 적의 현재 체력을 비율로 표시하는 UI를 구현했다.
  - **Bounding box:**
    - 화면을 통해 보여지는 world 좌표를 계산하여 player가 그 밖으로 나가지 못하게 했다.
  - **충돌 처리(hitbox):**
    - 각 object 객체마다 radius를 관리하여 두 객체 사이의 거리를 계산하는 식으로 구형태의 hitbox를 구현하였다.
  - **게임 재시작 및 종료**:
    - 게임 종료 시 'R' 키로 게임을 재시작하거나 'Q' 키로 종료할 수 있는 기능을 추가했다.

- **Detailed Description of Additional Requirements:**
  - **Complex shape Rendering:**
    - Enemy를 복잡한 모양으로 구성했고, Player에도 부스터를 추가하여 그렸다.

  - **Primitive rendering optimization:**
    - 기존의 glBegin()과 glEnd()를 사용하는 Immediate Mode 대신, 정점 데이터를 배열에 저장하고 한 번에 GPU에 넘겨주는 vertex array 방식을 사용하여, 엔딩 장면의 별들을 그리는 데 사용하였다. 두 방식을 비교해본 결과 확연한 성능 차이를 확인할 수 있었다.
    - 이외에도 화면 밖의 object들을 그리지 않는 식으로 최적화를 진행했다.

  - **Global camera animation:**
    - 게임이 종료될 때, Player가 별들이 있는 우주로 날아가는 듯한 카메라 움직임을 추가했다.
    - 이를 위해 projection matrix를 perspective를 통해 설정하였고, cameraMatrix를 따로 관리하여 object의 model view matrix를 계산할 때 앞에 곱해주었다.

- **Rationale Behind Program's Design:**
  - **Object Oriented Programing:**
    - Player, Enemy, Bullet 등을 객체로 정의하고, 이 객체들이 공통적으로 사용하는 변수나 함수들을 **Object**라는 Abstract Base Class로 정의하여 사용했다.

  - **Object Pooling Pattern**:
    - 본 게임의 특성 상 Bullet이나 Attack 객체들은 짧은 시간 동안 빠르게 생성되고 소멸된다. new와 delete를 반복적으로 호출하면서 프로그램이 느려지는 것을 확인하고, 이후 방법을 모색하다가 ObjectPool 방식을을 도입했다.
    - Player와 Enemy 객체가 새롭게 할당될 때, 각 클래스 내부 Constructor에서 대략적으로 필요한 각각의 객체를 미리 생성해두고, 필요할 때마다 활성화/비활성화하여 사용했다. 구체적인 예시로는 객체가 화면을 벗어나거나 충돌 시 비활성화(release)하여 Pool에 반납하는 형태로 시스템 부하를 줄였다.

- **How Design was Implemented:**
  - **class object 구현:**
    - 이 클래스는 modelMatrix를 이용한 위치, 회전, 크기 정보부터 isActive, isVisible 같은 상태 정보를 관리한다. 또 draw_shape() 함수와 같이 일부는 pure virtual function로 선언하여, 각 하위 클래스에서 반드시 구현하도록 했다.

  - **class objectPool 구현:**
    - Object 클래스를 상속 받아 정의한 모든 하위 클래스에 대해서 사용할 수 있도록 구현했다. Enemy와 Player 클래스는 각각 멤버 변수로 ObjectPool<Bullet>, ObjectPool<Attack>을 가지고 있고, shoot()이나 update()와 같은 맴버 함수 내부에서 acquire()나 release()를 통해 Pool에 요청 및 반환하도록 구현했다.

  - **하위 클래스 구현:**
    - entities/ 디렉토리 내에 Player, Enemy, Bullet 등 Object를 상속받는 게임 객체 클래스들을 구현했다. 각 클래스는 draw_shape()를 반드시 오버라이드하여 정의하도록 했고, update() 함수 내부에서 이동이나 발사, 충돌 처리 등을 동작하도록 했다.

  - **게임 상태 관리:**
    - main.cpp의 timer() 콜백 함수가 주기적으로 update() 함수를 호출한다. update() 함수는 전역 변수인 gameState를 확인하고 현재 게임 상태에 맞는 동작을 실행한다. Player나 Enemy의 체력(heart)이 0이 되면 gameState를 GameOver로 변경하고 재시작 및 종료 화면이 출력되도록 했다.

End-User Guide

- **How to Run and Operate Your Program:**

  1. **컴파일 (Compilation):**
     먼저 `src` 디렉토리로 이동한 후, 터미널에 아래 명령어를 입력하여 소스 코드(`main.cpp`)를 컴파일한다.

     ```
     >> g++ main.cpp base/*.cpp entities/*.cpp -o main \
           -I. -Ibase -Ientities -I../../include \
           -lGL -lGLEW -lglut
     ```

     컴파일이 완료되면 `src` 디렉토리 내에 `main`이라는 실행 파일이 생성된다.

  2. **실행 (Execution):**
     생성한 파일을 실행하면 게임이 시작된다.

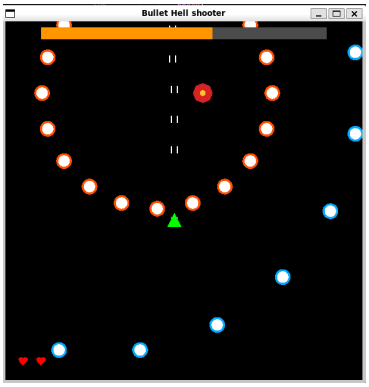
     ```
     >> ./main
     ```

  3. **조작법 (Controls):**
     - **이동:**
       - ↑ (위 방향키): 위로 이동
       - ↓ (아래 방향키): 아래로 이동
       - ← (왼쪽 방향키): 왼쪽으로 이동
       - → (오른쪽 방향키): 오른쪽으로 이동
     - **공격:**
       - `Space Bar`: 누르고 있는 동안 총알 발사
     - **게임 종료 시:**
       - `R` 또는 `r`: 게임 재시작
       - `Q`, `q`, 또는 `ESC`: 게임 종료

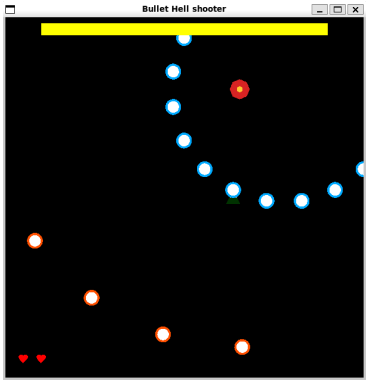
  이 외에도 Assn1의 README를 참고하여, 직접 git clone하고 컴파일 및 실행하는 방법이 있다.

- **Screenshots of Each Feature:**

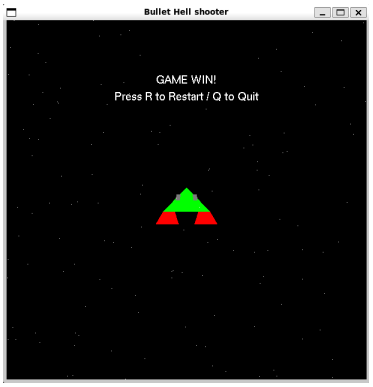
  1. 게임 플레이 어면

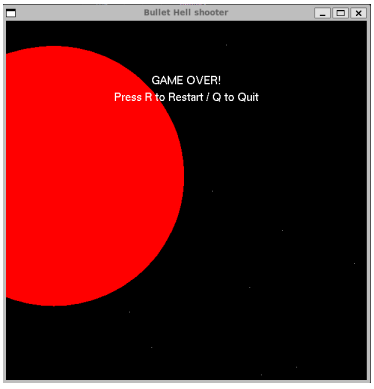
  2. Player 피격 및 무적 상태 (Player 색상이 바뀜)

  3. 게임 승리 화면

  4. 게임 패배 화면 및 이스터에그

Discussions/Conclusions

- **Obstacles Encountered and Resolutions:**
  - Object에 적용되는 좌표계가 헷갈렸다.
    - position을 직접 조절하는 방식 대신 model matrix를 업데이트하는 방식을 사용하면서, 화면 상에서 보여져야 하는 움직임과 코드 상 연산 순서와 방식에서 괴리감이 있었다.
- **Ideas for Future Program Improvement:**
  - Enemy가 발사하는 Bullet 패턴과 이동 동선을 더욱 복잡하고 다양하게 구성할 수 있을 것이다.
  - Canon 클래스를 Player와 독립적으로 구현하여, 추후 더 많은 Canon을 추가하고, 회전시키는 등 기능을 다양화할 수 있을 것이다.
  - Player가 피격되었을 때 camera를 조절하여 화면이 흔들리거나 반짝이는 효과를 넣을 수 있을 것이다.
  - 3D 좌표계 상에서 설계하여 추후 object들을 3D로 구성할 수 있을 것이다.
- **Learnings and Conclusions from Assignment:**
  - Object 클래스를 설계하면서, local 좌표계와 world 좌표계가 어떤 개념이고 그 작동 기작을 배울 수 있었다.
  - local 좌표계에서 화면 상 좌표계로 오기까지 곱해지는 model view matrix와 projection matrix가 각각 어떤 역할을 하고, camera를 어떤 식으로 조작할 수 있는지를 배울 수 있었다.

References

- **OpenGL Official Website** (https://www.opengl.org/)
- **OpenGL Extension Wrangler Library** (http://glew.sourceforge.net/)
- **freeglut** (http://freeglut.sourceforge.net/)
- **OpenGL Mathematics (GLM)** (https://glm.g-truc.net/0.9.9/index.html)

AI-Assisted Coding References

- **SunHo Cha**
  - **Tools Used:** Microsoft Copilot
  - **How Used:**
    - 특정 함수의 사용법을 익히기 위해 예시를 같이 제시해달라고 요구했다. 빠른 디버깅을 위해 에러 메세지를 넣고 어떤 문제인지 확인했다. c++의 문법을 복습할 때도 AI를 사용했다.
  - **Application Location:**
    - 플레이어 하트를 그릴 때 vertex 위치를 짜달라고 했다.
  - **Estimated AI Assistance Percentage:**
    - 전반적인 과정에 대략 30% 정도

- **SoonHo KIM**
  - **Tools Used:** Google Gemini, GitHub Copilot
  - **How Used:**
    - 여러 Mode에 대해서 Vertice를 찍어보며 이해에 도움이 되는 예제를 만들어달라고 했다.
    - 각종 라이브러리 함수가 어떻게 사용되는지 알아볼 때 자주 사용했다.
    - Vscode의 자동완성 기능을 사용하여 반복되는 코드 작성에 대한 시간을 단축할 수 있었다.
  - **Application Location:**
    - Bullet의 테두리를 구현할 때, 의도대로 동작하지 않는 코드에 대해 검토 및 수정해 달라고했다.
    - 게임 종료시 출력되는 Message 문구와 관련된 코드를 작성해달라고 했다.
  - **Estimated AI Assistance Percentage:**
    - 전반적인 과정에 대략 40% 정도