Assignment 2 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:**
  - **Orbiting entities**:
    - Object 클래스를 상속받아 Orbit 클래스를 만들고, rotate_world() 함수를 사용하여 해당 entity가, parent 객체인Player를 중심으로 회전하도록 구현했다.
    - 그리고 이를 player의 목숨 개수와 연동하였다.
  - **Enemy class**:
    - Assign 1에서 이미 구현한 class Enemy를 확장하여 main()함수 내부에서 instance를 생성하는 것만으로 사용 가능하도록 구현했다.
    - 이동 로직은 천천히 위에서 아래로 내려가고 다시 올라가는 방식으로 구현했다.
  - **Hierarchical animation**:
    - 각 Enemy에 Upper arm, Lower arm, Hand를 Hierarchical modeling을 통해 연결하였다. 이를 위해 object 클래스의 parent 멤버를 사용하였다.
    - Object 클래스 내부적으로 가지고 있는 model_Matrix 앞에 parent의 final_Matrix를 곱하는 식으로 최종 Current Matrix를 계산하도록 구현했다.

      Current_Matrix = (Parent) final_Matrix * (Self) model_Matrix

      위 동작을 연쇄적으로 일어나 항상 Root까지의 각 연산이 반영된다.
    - 이를 **Model View Matrix**로 load해서 draw를 진행하였다.
    - 간단하게 parent만 지정해줘도 hierarchical modeling이 가능해진다.
  - **Smooth player movement**:
    - 이전 상태를 저장하는 변수를 추가하여 player의 움직임을 더 부드럽게 구현했다.

- **Detailed Description of Additional Requirements:**
  - **Multi-level hierarchical animation:**
    - multi-level hierarchical modeling을 통해 enemy의 upper arm, lower arm, hand를 구현하였고, 각각의 회전을 위해 object 클래스의 rotate_local 함수를 사용하여 animation을 구현했다.

  - **Parametric shaping of entities:**
    - Enemy의 가장 하위 클래스인, Class Hand의 update() 함수에서 object 클래스의 정의한 scale_local() 함수를 기반으로 시간에 따라 hand의 크기를 변화시키는 parametric shaping을 구현했다.

- **Rationale Behind Program's Design:**
  - **Object class:**
    - Assign 1 부터 Object 클래스 내 modelMatrix를 정의하여 따로 저장 및 관리했다.
    - 기존의 pushMatrix, popMatrix 함수를 사용하면 model view matrix를 직접 수정하기 힘들다. glTranslatef, glRotatef 같은 함수들로 model view matrix의 오른쪽에 행렬을 곱하는 식으로 수정이 이뤄지는데, 이는 world(parent) 기준 translate, rotate를 적용하기 불편하다.
    - 따라서 model view matrix를 직접 계산해서 클래스마다 관리하고, 이를 load해서 사용하는 식으로 구현했다. 이를 통해 model view matrix의 왼쪽에 행렬을 곱하여 world(parent) 기준 translate, rotate를 적용할 수 있도록 하였다.

- **How Design was Implemented:**
  - **Object class 사용:**
    - Assign 1에서 구현한 Object 클래스 내부 Method 기능을 최대한 활용하였다.
    - Hierarchical 관계를 Object 클래스의 parent 멤버를 지정하는 식으로 구현하였고, 부모 object의 draw()와 update()에서 자식 클래스의 draw와 update를 담당하는 방식으로 구현했다.
    - 부모 기준 회전(orbit)을 구현할 때는 rotate_world() 함수를, 객체 본인 기준 회전(lower, upper, hand)를 구현할 때는 rotate_local() 함수를 사용하였다.

End-User Guide

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

1. **컴파일 및 실행 (Compilation & Execution):**
   이번엔 Make 명령어를 사용하여 실행할 수 있도록 코드를 추가했다. 먼저 `src` 디렉토리로 이동한 후 터미널에 아래 명령어를 입력하여 게임을 실행한다.

   ```
   >> make (or make all)
   ```

   위 명령어를 입력하면 `src` 디렉토리 내에 `main`이라는 실행 파일이 생성된다.

2. **실행 ():**
   생성한 파일을 실행하면 게임이 시작된다. make run을 입력하면 빌드 및 실행까지 한 번에 할 수 있다. make 명령어를 통해 이미 실행파일을 생성했다면, 간단히 ./main을 통해서도 실행가능하다.

   ```
   >> make run (or ./main)
   ```

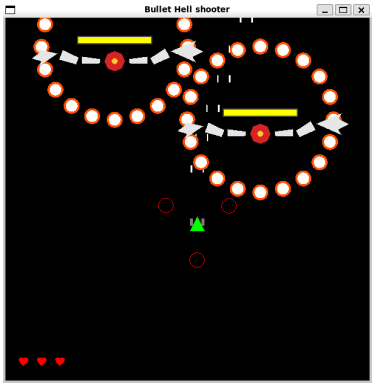
3. **실행파일 정리:**
   ```
   >> make clean (/build 폴더와 main 실행파일 삭제)
   ```

4. **조작법 (Controls):**
   - **이동:**
     - ↑, ↓, ←, →: 방향키를 사용하여 Player 이동
   - **공격:**
     - `Space Bar`: 누르고 있는 동안 총알 발사
   - **게임 종료 시:**
     - `R` 또는 `r`: 게임 재시작
     - `Q`, `q`, 또는 `ESC`: 게임 종료

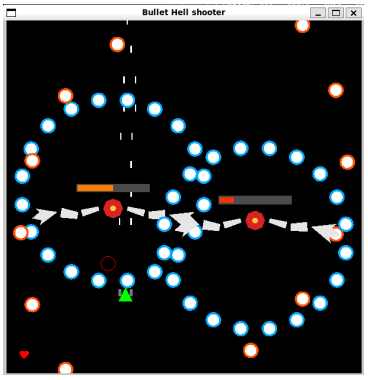
이 외에도 Assn2의 README를 참고하여, 직접 git clone하고 컴파일 및 실행하는 방법이 있다.

- **Screenshots of Each Feature:**

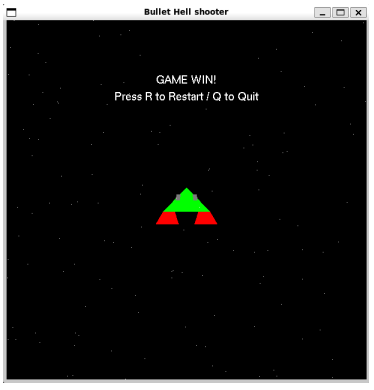
1. 게임 플레이 이미



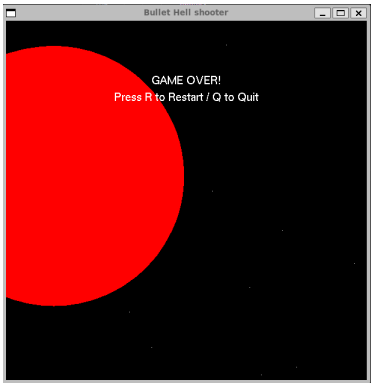
2. Player 피격 및 무적 상태 (Player 색상이 바뀜)



3. 게임 승리 화면



4. 게임 패배 화면 및 이스터에그



Discussions/Conclusions

- **Obstacles Encountered and Resolutions:**
  - Assign 1에서 추가적으로 다룬 내용들을 바탕으로 Assign 2를 쉽게 해결할 수 있었다.
- **Ideas for Future Program Improvement:**
  - Class Enemy의 instance화를 통해, 추가 enemy들을 더 생성할 수 있을 것이다.
  - Orbit을 attack의 한 종류로 설계할 수 있을 것이다.
  - Player에도 hierarchical modeling을 더 적용하여 복잡하게 구성할 수 있을 것이다.
  - 모든 연산과 modeling이 3D 상에서 구현되어 추후 확장에 용이할 것이다.
  - Upper, Lower, Hand 등의 각 객체에 반복해서 정의된 Method들을 Object Class로 옮겨 중복 코드 제거 및 최적화가 가능할 것이다.
- **Learnings and Conclusions from Assignment:**
  - Hierarchical modeling을 구현하면서, 효율적이고 간결한 구현 방법을 생각해 볼 수 있었고, local 좌표와 world 좌표 사이의 관계를 깊게 이해할 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:**
    - pdf의 설명을 이해하기 위해 사용했다. (예: parametric shaping에 scale 변환도 포함되는지 여부)
  - **Application Location:**
    - 이번 과제에는 AI의 도움을 받은 코드가 없다.
  - **Estimated AI Assistance Percentage:** 10%

- **SoonHo KIM**
  - **Tools Used:** GitHub Copilot
  - **How Used:**
    - 새롭게 생성할 클래스 형태를 제안받는 용도로 사용했다.
    - Class 파일이 많아지면서 Src 폴더 내 코드 Directory 재구성을 위해 사용했다.
  - **Application Location:**
    - Upper Class를 초기화 할 때, 보다 효율적인 방식이 있는지 제안 요청.
    - README 영어 문장으로 다듬을 때 사용.
    - MakeFile을 만들고, Warning을 분석 및 제거하는 데 사용.
  - **Estimated AI Assistance Percentage:** 30%