Assignment #1: 2D Drawing

CSED451 Computer Graphics (Spring 2021) Assignment #1

콤퓨타화상처리

최은수 (컴퓨터공학과, 20180050, ches7283),
권민재 (컴퓨터공학과, 20190084, mzg00)

개요

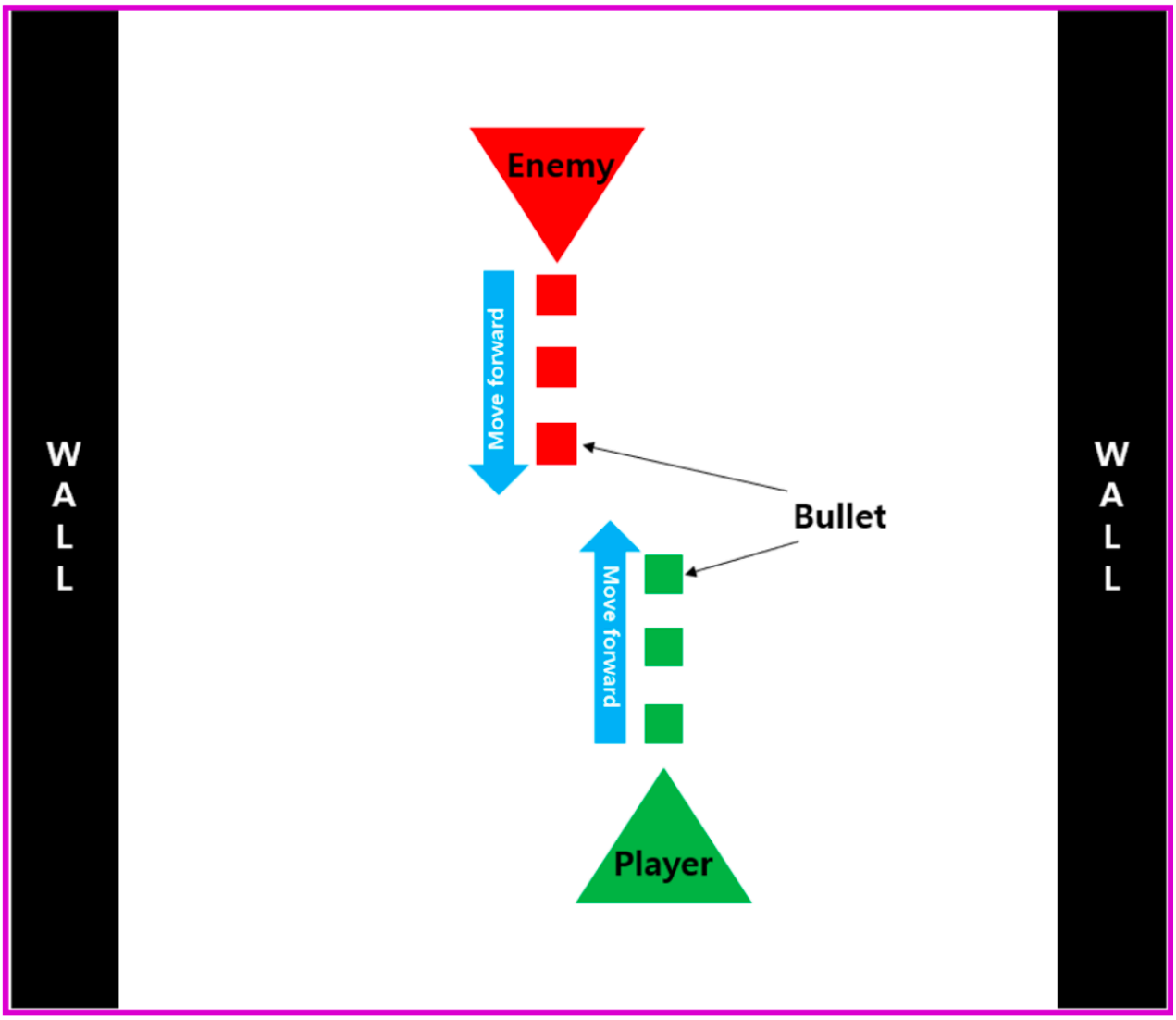
이 과제에서는 OpenGL을 활용하여 "Firi Games"의 "Phoenix 2(2016)"에 기반한 슈팅게임을 구현해야한다. 여기서 적과 플레이어의 조건은 아래와 같다.

적

- 서로 다른 색깔의 적이 5번 등장하고 일정 시간마다 총알을 발사하고 무작위로 이동해야한다.

플레이어

- 상하좌우 이동 가능하며 총알을 발사하여 적을 모두 제거하면 승리한다.
- Mode 변경을 통해 (All-pass / All-fail) 모드를 선택할 수 있다.



프로그래밍 환경

개발 환경

- Visual Studio 2019

라이브러리 버전

- OpenGL 4.1
- freeglut 3.2.1
- glew 2.2.0
- glm 0.9.9.9

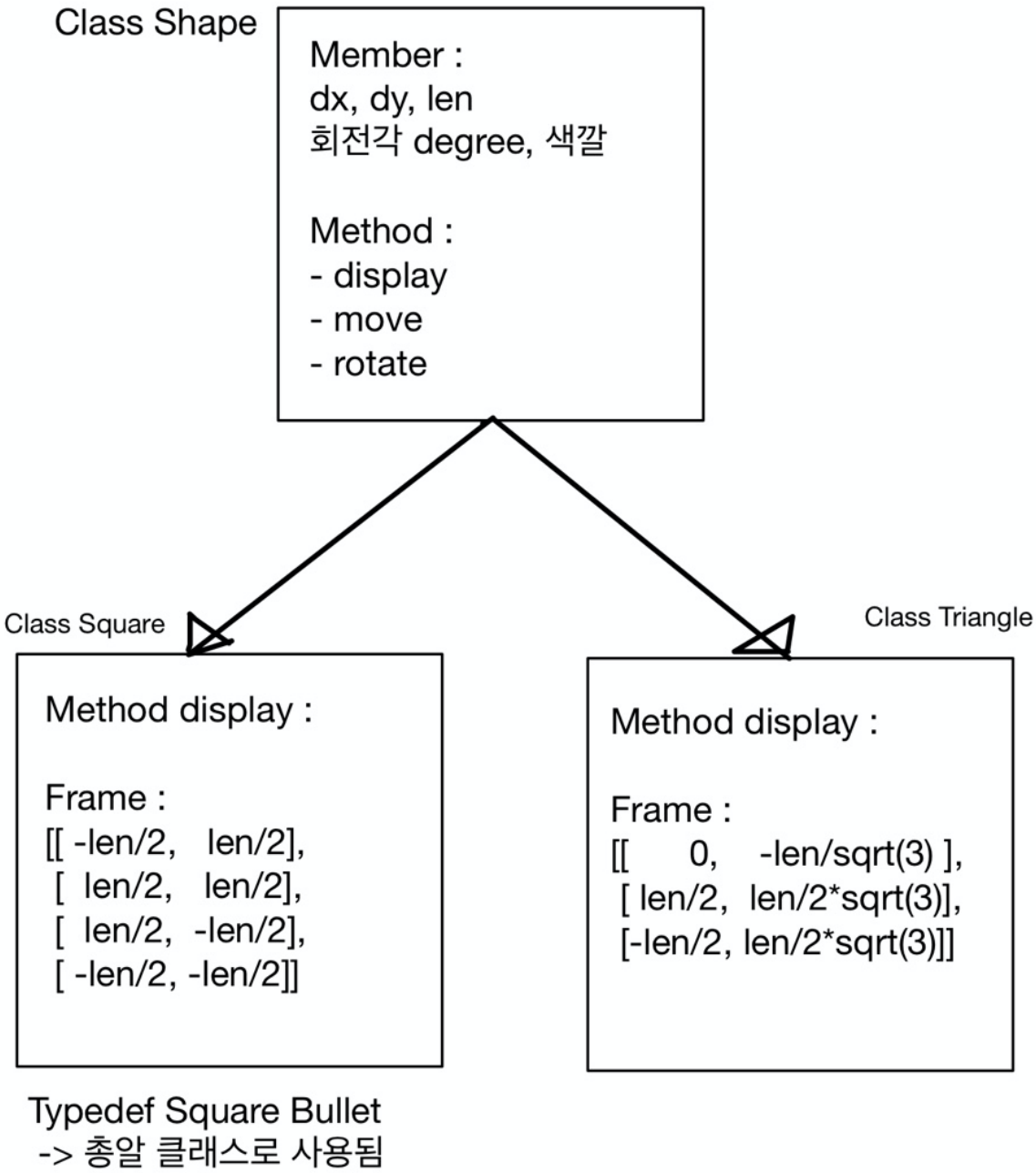
프로그램 설계 및구현

Class

Game

- game mode (all-pass / all-fail) 변경 및 관리
- Player & Enemy instance 생성 및 삭제와 관리
- Infomation display
- Game over/win 체크 후 해당 루틴 실행

Shape

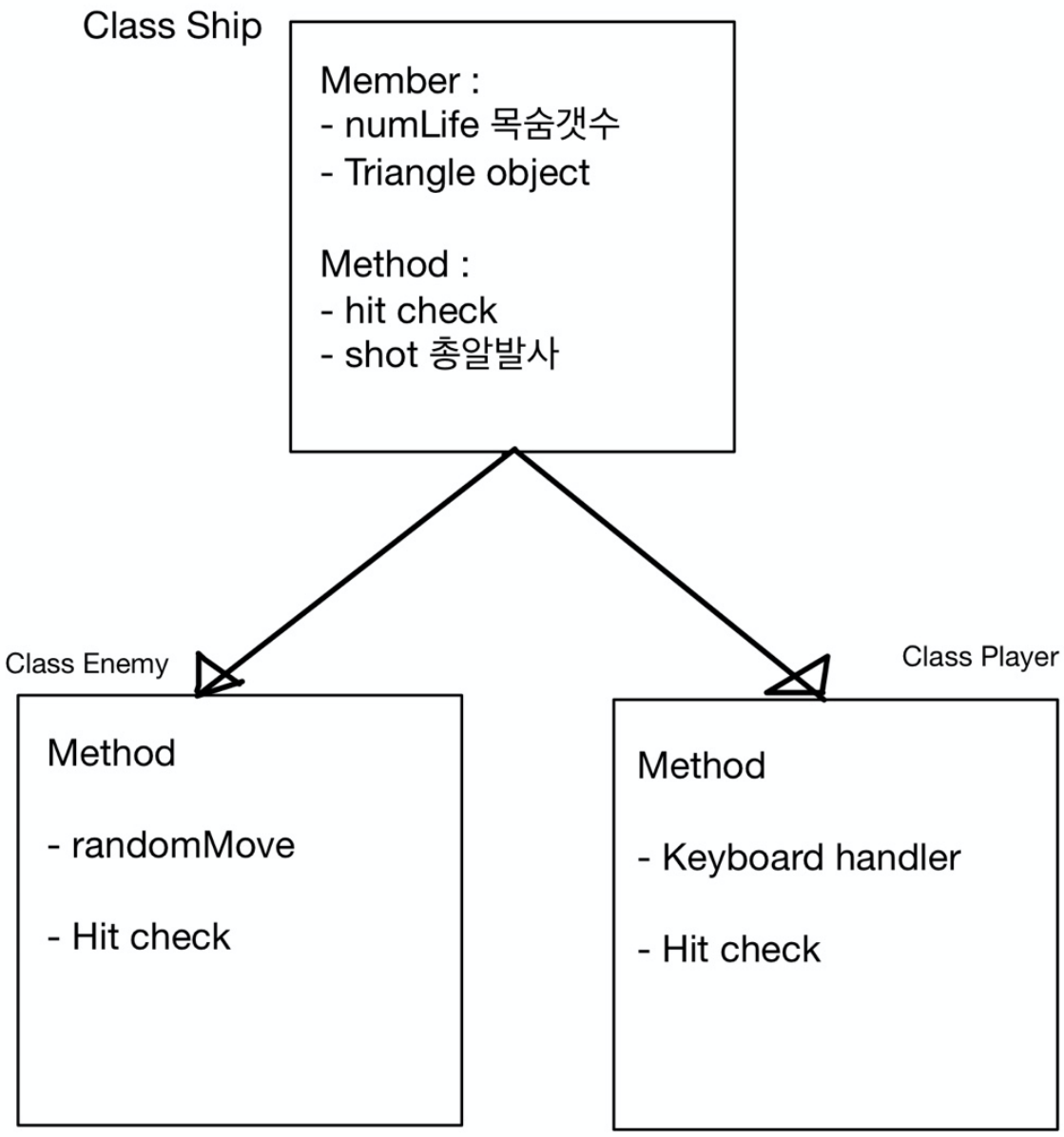


Display

각 class (square/triangle)의 model frame에서 transformation 함수를 사용하여 model-coordinate 값을 고정하고,
display할 때 world-coordinate에만 움직인 좌표로 보이게 한다.
아래 3개의 함수를 사용한다.

- glColor3fv
- glTranslatef(dx, dy, 0.0f)
- glRotatef(degree, 0, 0, 1);

Ship



Player::keyboard_handler

- Switch case 문을 이용하여 OpenGL keyboard handler로 부터 넘어온 정보를 활용하여 구현하였다.
- **상하좌우키** : window 밖을 넘어가는지 체크 한 후 triangle->move를 사용하여 이동
- **space_bar** : shot 함수를 호출하여 Bullet 생성 후 리턴

Hit check

- 총알이 우주선에 충돌하였는지 체크하는 루틴.
- 총알은 사각형이므로 네 점중 하나라도 우주선(삼각형) 내부에 존재하면 충돌로 간주한다.
- 삼각형의 한 점 T와 총알의 점 K가 삼각형의 나머지 두 점 A, B로 이루어진 직선너머 같은 쪽에 있는지를 모든 점에 대해 체크한다.
- $f(x,y) = (x-x1)(y1-y2) - (y-y1)(x1-x2)$ 에서 $f(T) \times f(K) > 0$인지 삼각형의 모든 점에 대해 체크한다.
- Enemy와 Player의 경우 Mode에 따라 목숨을 잃고, 아니고가 다르므로 각 class마다 method 정의가 조금 다르다.

Callback function

Keyboard Event

- `glutKeyboardFunc`, `glutKeyboardUpFunc`, `glutSpecialFunc`, `glutSpecialUpFunc`을 사용하여 키보드 이벤트를 처리하였다.
- Special Key
  - 방향키 입력을 일정한 캐릭터로 변환해서 플레이어 클래스에게 해당 키가 down 혹은 up 되었음을 알린다.
- Normal Key
  - 스페이스 입력을 일정한 캐릭터로 변환해서 플레이어 클래스에게 해당 키가 down 혹은 up 되었음을 알린다.
  - 모드를 변경하는 키인 `c`나 `f`가 입력되었을 경우에는 모드를 표시하는 전역 변수를 수정한다.

TimerFunc

BulletMoveHit

- 0.3초마다 실행되며, enemy와 player가 쏜 bullet들을 일정 단위길이만큼씩 이동시킨다.
- 이동시킨 bullet이 window밖을 나갈 경우 삭제한다.
- 각 bullet들이 Enemy와 Player에 충돌했는지 체크하고 충돌한 bullet은 삭제한다.

BulletEnemyShot

- 3초마다 실행되며, enemy의 randomMove가 호출되며 좌/우/제자리 중 하나의 랜덤한 움직임을 한다.
- param value로 (value+1)%7을 넘겨줌으로써 움직임 7번 중 1번씩 bullet을 발사한다.

프로그램 실행 방법

우주선 이동

- 키보드의 **상하좌우** 키를 누르면 그에 따라 우주선이 상하좌우로 이동한다.
- 우주선은 선체의 일부분도 window 밖으로 나갈 수 없으며 적이 있는 위치까지 제한없이 이동 가능하다.

총알 발사

- 키보드의 **space bar** 를 누르면 삼각형의 위쪽 꼭짓점을 통해 총알을 발사할 수 있다.
- 발사 가능한 총알 개수에는 제한이 없으며 이동과 동시에 총알 발사도 가능하다.

모드 변경

- **C** : 키보드의 'c' 키를 누르면 올패스 모드로 전환된다.
- **F** : 키보드의 'f' 키를 누르면 올패일 모드로 전환된다.
  모드 변경 키가 한 번 더 눌릴경우 다시 노말모드로 변경된다.
  해당 모드는 모두 화면에 표시되므로 시각적으로 확인할 수 있다.

예제

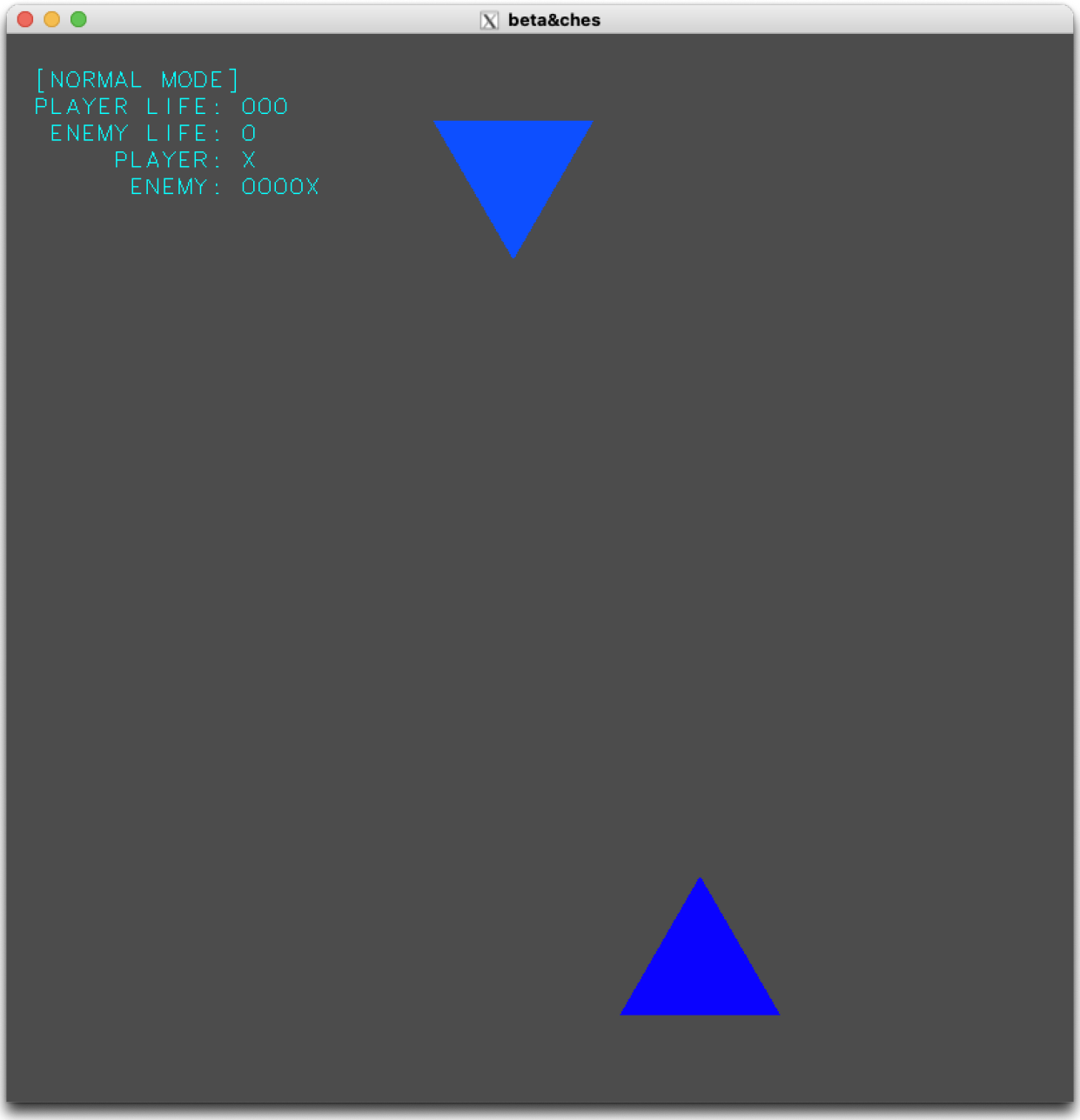
실행 시작 Normal mode, enemy 목숨1, player 목숨3

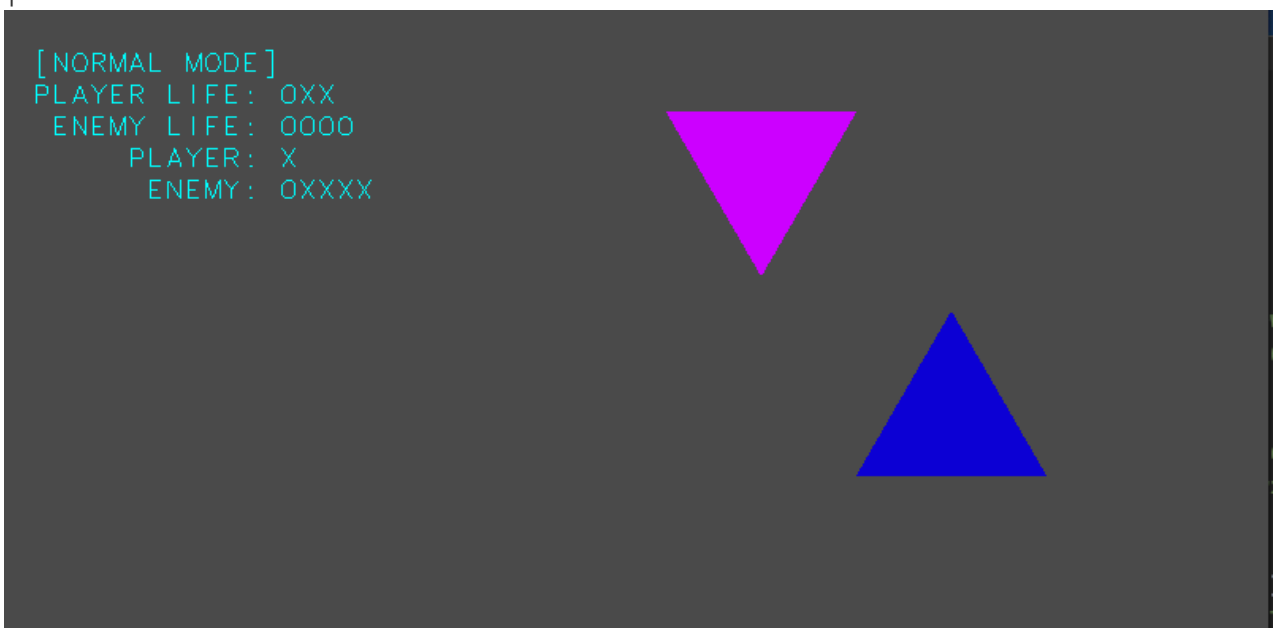
스코어보드 좌측 상단에서 enemy와 player의 남은 목숨과 총 갯수, mode를 확인할 수 있다.

이동 상하좌우 키보드 입력을 통해 이동

목숨

- enemy : 총 5마리로 뒤에 나올 때마다 목숨이 +1 더 많으며 각각 모두 색이 다르다.
- player : 목숨이 한개씩 줄어들 때마다 색깔이 변한다.


beta&ches
[NORMAL MODE]
PLAYER LIFE: OOO
ENEMY LIFE: O
PLAYER: X
ENEMY: OOOOX

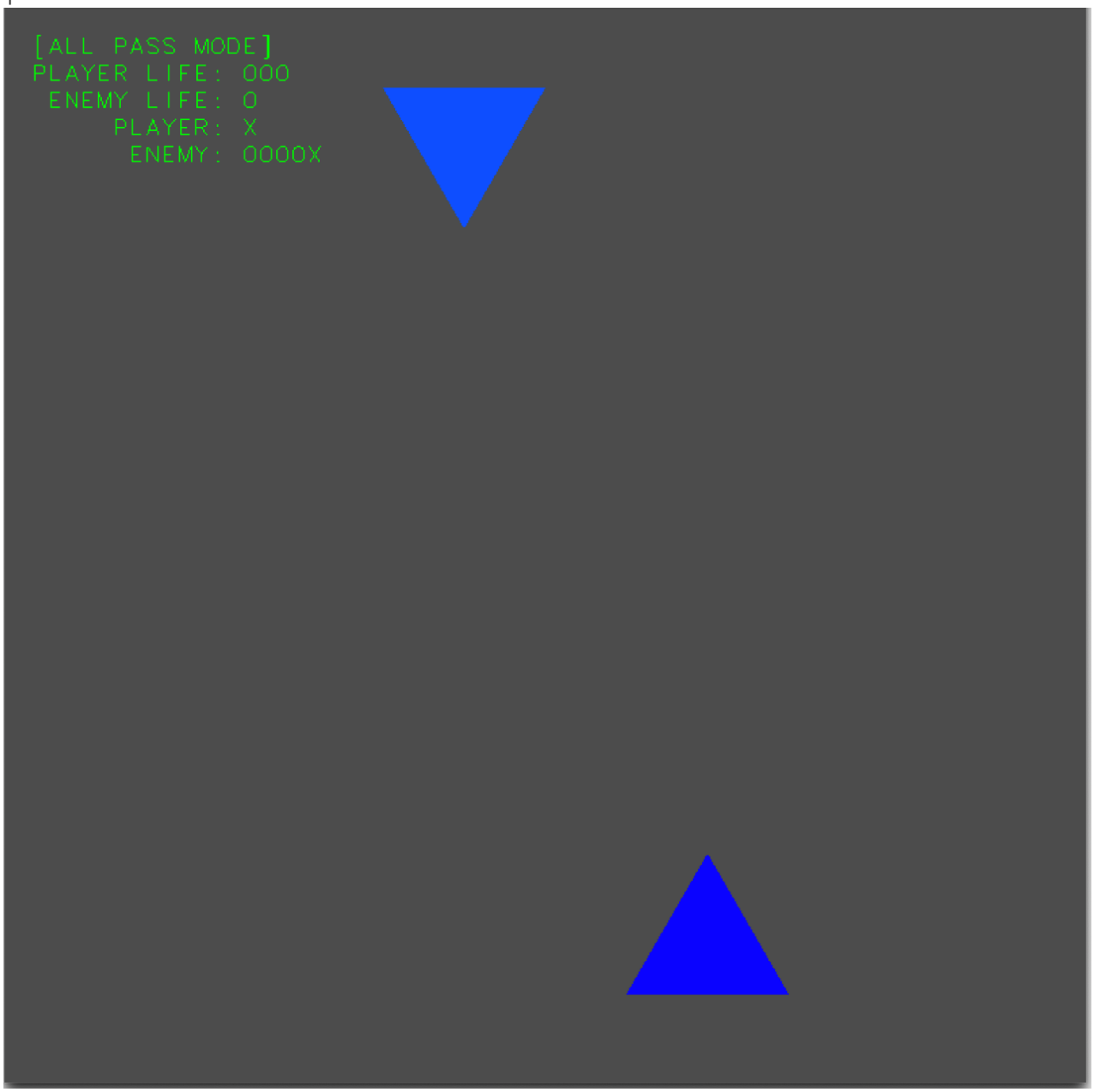

[NORMAL MODE]
PLAYER LIFE: OXX
ENEMY LIFE: OOOO
PLAYER: X
ENEMY: OXXXX

|
|--|--|

Mode 변경 주어진 키를 누를 경우에 all pass나 all fail 모드를 설정할 수 있고, 이는 좌측 상단의 스코어보드에서 확인 가능.

게임 종료

- win : 적을 모두 제거할 경우 승리
- Game over : 3개의 목숨을 모두 잃을 경우 패배.


[ALL PASS MODE]
PLAYER LIFE: OOO
ENEMY LIFE: O
PLAYER: X
ENEMY: OOOOX


WIN!


|
|--|--|

토론

callback 함수의 인자

- 해당 어싸인에서 본 팀은 구현을 위해 OpenGL의 여러 콜백함수 (`glTimerfunc` , `glDisplay` , `keyboard` 등) 를 사용하였다.
- openGL내부 콜백함수들은 외부에 선언되어있는 값들을 인자로 넘겨받을 수 없기 때문에 함수 내에서 우주선, 총알 등의 instance 들을 사용할 수 있는 방법에 대해 많은 고민을 하였다.
- 처음에는 JavaScript처럼 lambda 함수 리턴 을 활용하고자 하였지만 생각한대로 잘 구현되지 않아 외부에 static Global을 선언하여 활용하였다.
- 후에는 싱글턴방식을 차용하여 global 변수를 좀 더 객체지향적으로 포장하여 사용할 수 있을 것 같다.

추가 구현

스코어보드

플레이어가 현재 게임의 상황을 편리하게 인식할 수 있도록 스코어보드를 제작하고, 게임이 종료되었을 때 승리 여부를 출력할 수 있도록 하였다. 스코어보드에는 게임 모드 (`Normal` , `All-pass` , `All-fail`), 플레이어의 남은 목숨과 남은 함선 수, 그리고 적의 남은 목숨과 남은 함선 수를 표시했다. 게임이 종료된 이후에 플레이어가 이겼다면 WIN!을 출력하고, 그렇지 않은 경우에는 `GAME OVER` 를 출력했다.

스코어보드는 `Game` 클래스의 `displayInfo` 를 이용하여 출력한다. 스코어보드에 텍스트를 출력하기 위해서 우선 출력하고자 하는 정보들의 문자열들을 만든 이후에 일정한 Y좌표 간격을 두고 문자열 별로 한 글자마다 `glutStrokeCharacter` 를 이용하여 출력하였다. `glutStrokeCharacter` 가 `glutBitmapCharacter` 보다 transform 하기에 더 용이하다고 판단하였기 때문에 스크로크 방식의 글자 출력을 이용하였으며, 이 프로그램에서 C++ `string` 을 이용하는데 반해 `glutStrokeString` 은 C 스타일의 문자열 `char *` 을 인자로 받고 있었기 때문에 `glutStrokeCharacter` 로 한글자씩 출력하는 방식을 선택하였다.

멀티 키 입력

glut의 키보드 이벤트를 단순하게 처리할 경우에는 멀티 키 입력을 처리할 수 없다. 즉, player가 우주선을 움직이면서 총알을 쏠 수 없다. 그렇기 때문에 일반적인 게임과 같이 플레이이어가 우주선을 움직이면서 총알을 쏠 수 있게 할 수 있도록 하기 위해 멀티 키 입력을 구현하였다.

멀티 키 입력을 위해 `Player` 클래스에 키 버퍼 `inputKey`를 만들어주었다. 기본적으로 `inputKey`의 값은 `false`이다. 이때, 키가 눌리는 이벤트 (key down)이 발생하면, `inputKey`에서 눌린 키 인덱스의 값을 true로 만들어준다. 키 입력이 끝나는 이벤트 (key up)이 발생하면, `inputKey`에서 입력이 끝난 키 인덱스의 값을 false로 만들어준다. 이후 Key down 이벤트가 발생할 때 마다 플레이어의 키보드 핸들러를 호출하여 해당 핸들러가 `inputKey`에서 true인 모든 키에 대해 검사하도록 처리해서 멀티 키 입력을 구현하였다.

결론

1. OpenGL gl* 함수들을 통해 vertex 생성과 색깔 변경, view point등의 활용을 익힐 수 있었다.
2. GLUT 를 이용하여 **Window 생성**, **Keyboard 입력 이벤트 핸들링**, **타이머 기능** 등의 활용을 익힐 수 있었다.
3. 추가구현을 포함하여 멀티키 입력까지 적용하는 성공적인 슈팅게임을 완성하였다.

개선 방향

우주선 회전 추가

- 충돌 체크와 총알 발사를 하는 함수에서 우주선의 각 vertex를 구할 때 rotate된 각도도 고려하여 계산한다.
- 그래서 현재는 우주선이 상하좌우로만 이동하는데 특수한 아이템 획득 또는 키보드 입력을 통해 내심을 중심으로 회전할 수 있도록 추가하면 좋을 것 같다.
- 또한 총알이 나가는 꼭짓점은 항상 고정이기 때문에 회전하면 총알의 이동도 그에 따라 방향/각도가 달라져서 훨씬 게임의 난이도가 상승할 것이다.

참고문헌

- OpenGL Docs (https://www.khronos.org/opengl/)