과목명	컴퓨터그래픽스I
교수님	김형석
학 과	응용소프트웨어공학과
학 번	20203218
이 름	임미선
날 짜	2022.06.01

東義大學校
DONG-EUI UNIVERSITY

《 목차 》

1. 프로젝트 소개

1-1. 디자인 컨셉

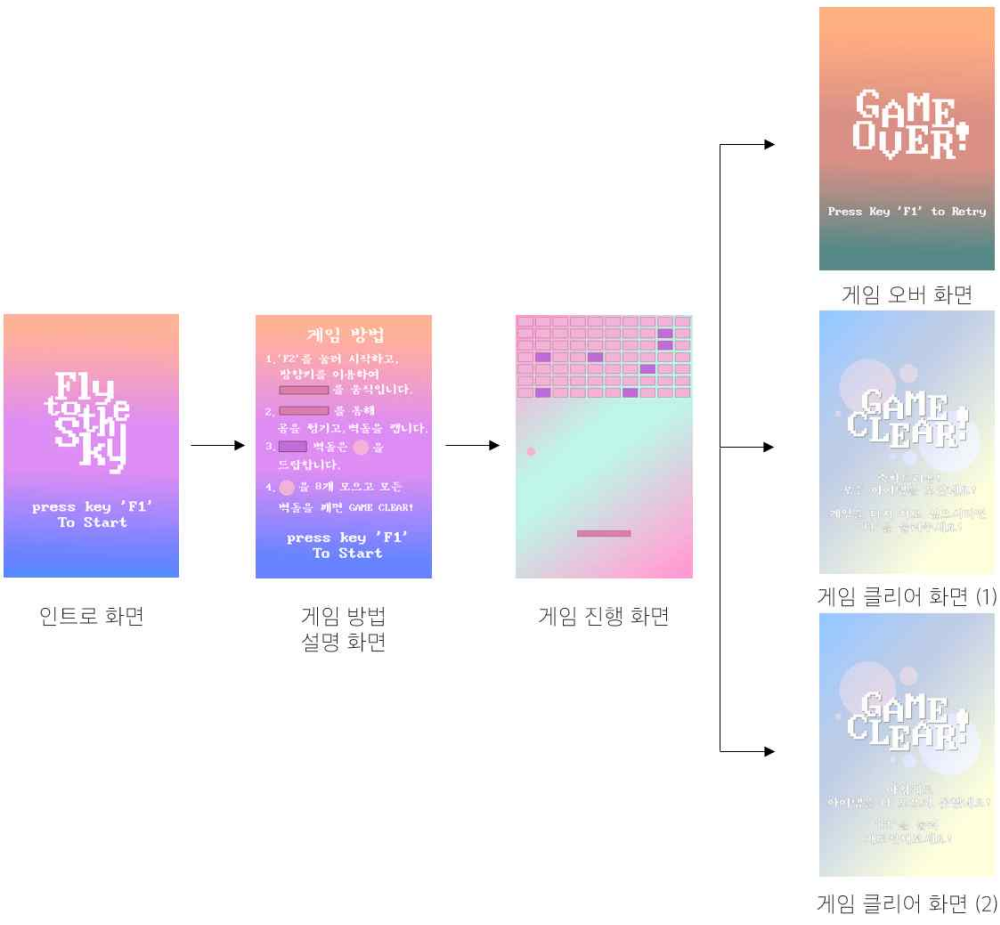
- 분홍색, 하늘색 계열 등의 최대한 부드러운 파스텔 색감을 이용하였다.
- 도트 게임의 느낌이 나는 게임 화면 이미지를 다양하게 제작하여 적용하였다.

1-2. 게임 구성

- 화면 내에서 많은 충돌 처리를 수행하며 공이 튄다.
- bar(긴 막대기) 위에 공이 오도록 하며 공이 튀는 방향을 조절한다.
- 분홍색 벽돌 : 공과 충돌함 -> 깨져서 사라짐
- 보라색 벽돌 : 공과 충돌함 -> 깨져서 사라짐 -> 아이템을 drop
- 게임 클리어(엔딩)
    1) 모든 벽돌을 깨고, 모든 아이템을 획득함
    2) 모든 벽돌을 깨고, 아이템을 모두 획득하지 못함

2. 프로젝트 구현 방법

2-1. 화면 구성



인트로 화면 | 게임 방법 설명 화면 | 게임 진행 화면 | 게임 오버 화면 | 게임 클리어 화면 (1) | 게임 클리어 화면 (2)

▼ 화면 전환 변수 선언

```
// 현재 화면을 판단할 bool 변수
bool window_intro;// 인트로 화면
bool window_game_solution; // 게임 방법 화면
bool window_game_now;// 게임 진행 화면
bool window_game_over; // 게임 오버 화면
bool clear;// 클리어 화면
```

- 현재 화면을 판단할 bool 변수를 통해 6개의 화면을 전환할 수 있도록 구성함.

▼ Key 입력에 따른 화면 전환

```
// 게임 활성화 'F1' Key
case GLUT_KEY_F1:
        if (window_intro) { // 인트로 화면일 때 F1을 누르면
                game_start = true; // 게임 시작 활성화
                window_game_solution = true; // 게임 방법 화면 출력
                window_intro = false; // 인트로 화면 비활성화
        }
        else if (window_game_solution) { // 게임 방법 화면일 때 F1을 누르면
                window_game_solution = false; // 게임 방법 화면 비활성화
                window_game_now = true; // 게임 화면
        }
        else if (game_over) { // 게임 오버 화면일 때 F1을 누르면
                game_over = false; // 게임 오버 비활성화
                init(); // 게임 retry
        }
        else if (clear) { // 클리어 화면일 때 F1을 누르면
                clear = false; // 클리어 비활성화
                init(); // 게임 retry
        }
        break;
```

- 'GLUT_KEY_F1' 입력에 따라 화면이 재구성되도록 화면 전환을 구현하였음.
- moving_ball(움직이는 공), bar 좌표 또한 GLUT_KEY를 이용하여 화면이 렌더링될 때마다 좌표가 변환되도록 하였음.

▼ 'stb_image' 헤더를 이용한 Texture Mapping

```
// 이미지 파일 열기
unsigned char* LoadMeshFromFile(const char* texFile) {
        int w, h;
        GLuint texture; // 텍스쳐 버퍼
```

```
	glGenTextures(1, &texture);
	FILE* fp = NULL;
	if (fopen_s(&fp, texFile, "rb")) {
		printf("ERROR : No %s. \n fail to bind %d\n", texFile, texture);
		return (unsigned char*)false;
	}
	int channel;
	unsigned char* image = stbi_load_from_file(fp, &w, &h, &channel, 4);
	fclose(fp);
	return image;
}

void intro_image_texture(char a[]) {
	GLuint texID;
	unsigned char* bitmap;
	a = (char*)a;
	bitmap = LoadMeshFromFile(a);

	glEnable(GL_TEXTURE_2D);
	glGenTextures(1, &texID);
	glBindTexture(GL_TEXTURE_2D, texID);

	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, bitmap);
	free(bitmap);
}

void intro_image_texture_blend(char a[]) {
	GLuint texID;
	unsigned char* bitmap;
	a = (char*)a;
	bitmap = LoadMeshFromFile(a);

	glEnable(GL_TEXTURE_2D);
	glEnable(GL_BLEND);
	glBlendFunc(GL_DST_COLOR, GL_ZERO);

	glGenTextures(1, &texID);
	glBindTexture(GL_TEXTURE_2D, texID);
	glBlendFunc(GL_ONE, GL_ONE);
	glBindTexture(GL_TEXTURE_2D, texID);
```

```
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, bitmap);
        free(bitmap);
}
```

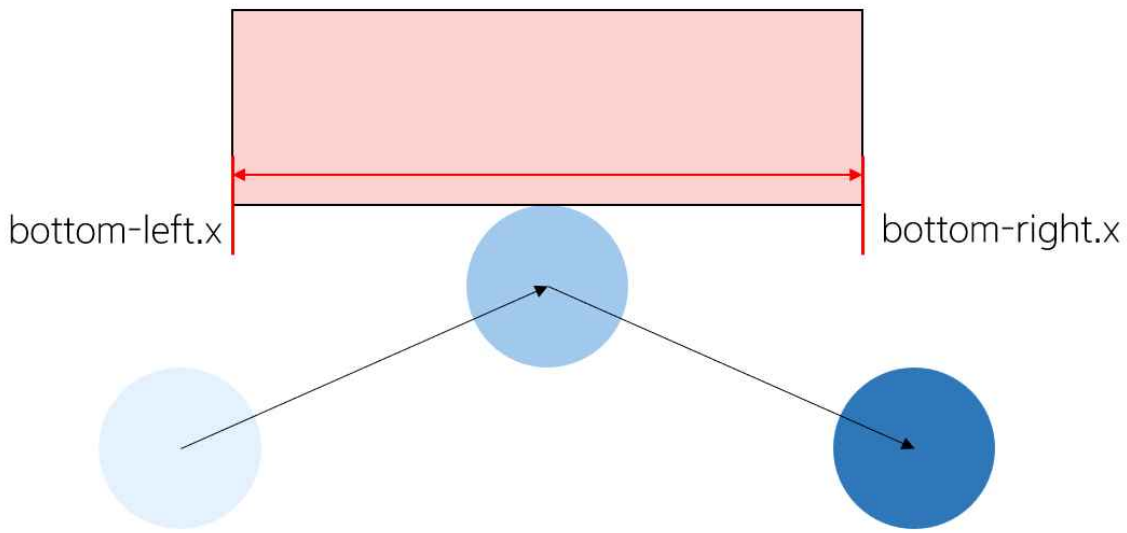
- moving_ball(움직이는 공), bar 좌표 또한 GLUT_KEY를 이용하여 화면이 렌더링될 때마다 좌표가 변환되도록 하였음.
- 6개의 화면을 각 화면 전환 bool 변수가 true일 경우에만 도형에 Texture Mapping이 적용되도록 함.

2-2. 충돌처리

a. 상하좌우

- 주의할 점 : 모든 충돌 처리에서 예외적인 경우를 방지하기 위해서 벽돌과 공의 중심까지의 거리가 공의 반지름보다 작거나 같은 경우에만 충돌 처리 구문을 수행하도록 하였음.
- 충돌 전 : 충돌 처리 조건문을 통해 벽돌 배열을 반복 탐색함.
- 충돌 후 : 벽돌을 그리는 꼭짓점 좌표 4개를 모두 (0, 0)으로 보내어 충돌 위치에 존재하는 벽돌이 지워지도록 함.

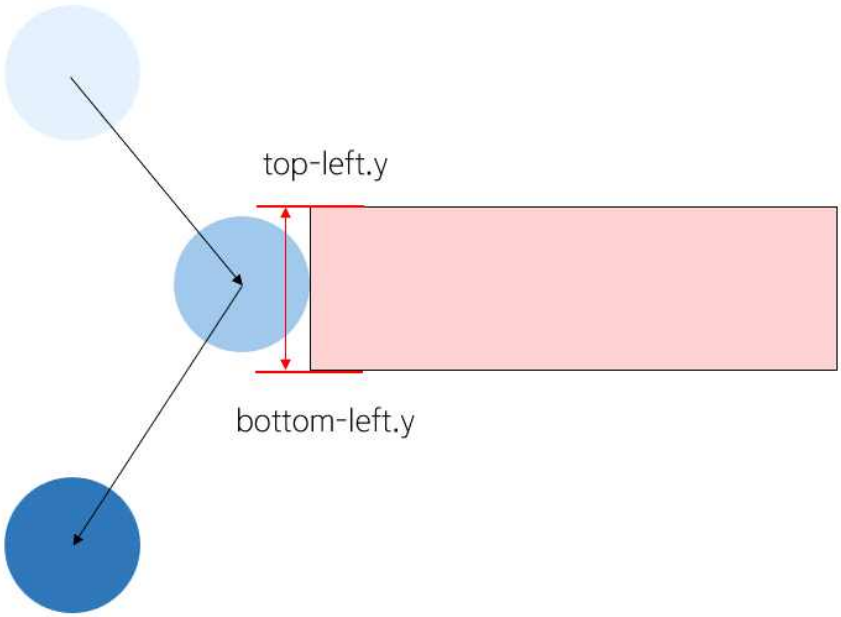
▼ Top Collision, Bottom Collision (x축 좌표 이용)




```
if (brick_array[i][j].rectangle[0].x <= moving_ball.x
                                    && brick_array[i][j].rectangle[3].x >= moving_ball.x)
```

- 벽돌의 윗부분과 아랫부분 충돌은 벽돌의 가로 범위 내에 공의 중심이 존재할 경우를 공통 조건문으로 작성하였음.

▼ Left Collision, Right Collision (y축 좌표 이용)




```
if (brick_array[i][j].rectangle[0].y >= moving_ball.y
                                    && brick_array[i][j].rectangle[1].y <= moving_ball.y)
```

- 벽돌의 왼쪽부분과 오른쪽부분 충돌은 벽돌의 세로 범위 내에 공의 중심이 존재할 경우를 공통 조건문으로 작성하였음.

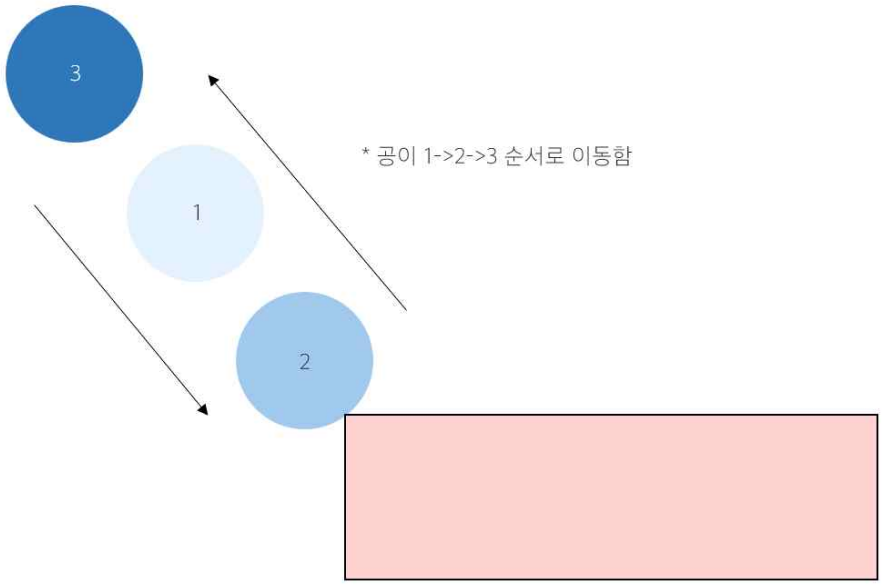
a. 모서리

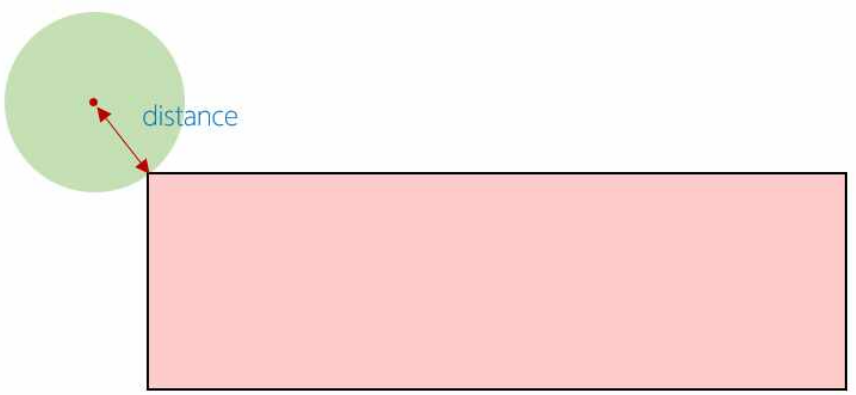
▼ 모서리 충돌

▼ 벽돌의 각 꼭짓점으로부터 공의 중심까지의 distance

- 벽돌의 각 꼭짓점으로부터 현재 공의 중심까지의 거리가 공의 반지름보다 작거나 같을 때, 3가지 충돌 방향의 경우를 나누어 모서리 충돌을 처리한다.
- 예를 들어, 좌상단(top-left) 모서리 충돌에는
  1) (공의 x 속력 > 0) and (공의 y 속력 < 0)
  2) (공의 x 속력 < 0) and (공의 y 속력 < 0)
  3) (공의 x 속력 > 0) and (공의 y 속력 > 0)

이렇게 3가지 방향에서 공이 날아오는 경우가 있다.





따라서, 모서리 충돌 처리를 위해 공이 모서리에 부딪히는 순간의 x 방향 속력, y 방향 속력의 부호에 따른 3가지 경우로 나눠서 속력을 반전시키는 구문을 수행한다.

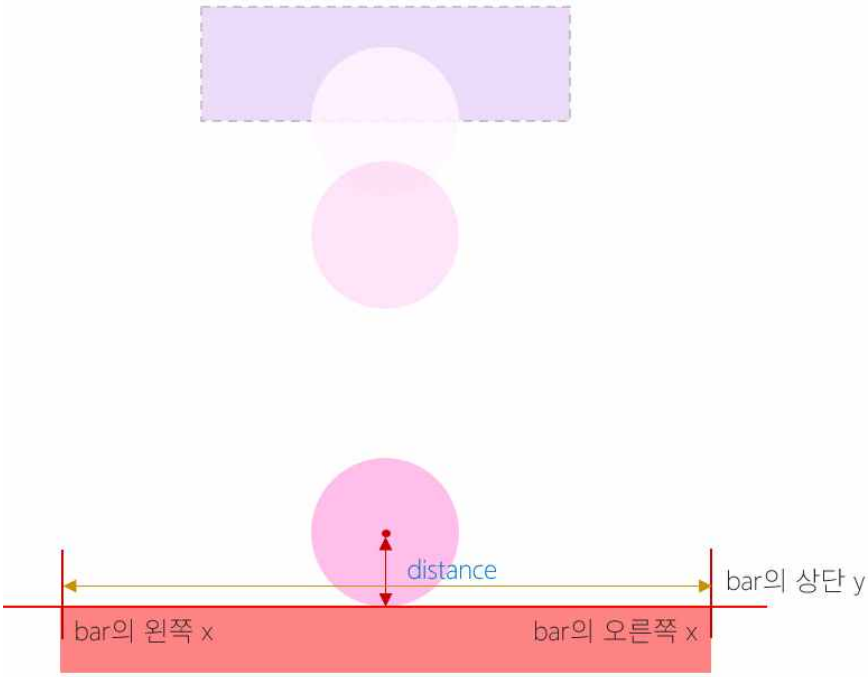
▼ 좌상단(top-left) 충돌 처리

```
// top-left collision
if (distance(Point(brick_array[i][j].rectangle[0]), Point(moving_ball)) <=
moving_ball_radius)
{
        if (velocity.x >= 0 && velocity.y <= 0) {
                velocity.x *= -1;
                velocity.y *= -1;
        }
        else if (velocity.x >= 0 && velocity.y >= 0) {
                velocity.x *= -1;
        }
        else if (velocity.x <= 0 && velocity.y <= 0) {
                velocity.y *= -1;
```

```
        }
}
```

2-3. 아이템 획득 기능

▼ 아이템 공과 bar 상단의 충돌 처리



- 보라색 벽돌을 깨면 분홍색 원 형태의 아이템이 drop된다.
- 아이템의 위치 지정 : 모든 벽돌의 위치 중 랜덤으로 행/열을 선택하여 8개의 아이템을 생성한다.
- 아이템 획득 조건

1) bar의 상단 y 좌표으로부터 아이템 공의 중심까지의 거리가 아이템 공의 반지름부터 작거나 같을 경우

2) bar의 가로 범위 내에 아이템 공의 중심이 위치하는 경우

이 두 조건을 만족하는 경우, 해당하는 아이템 공을 획득 처리한다.

▼ 아이템 공이 존재하는 벽돌의 깨짐 여부와 아이템 공의 drop 여부 검사

```
for (int j = 0; j < 8; j++) {
        tmp = 0;
        tmp_x = item.upper_brick_num[j].x;
        tmp_y = item.upper_brick_num[j].y;
        for (int i = 0; i < 4; i++) {
                // item 벽돌이 깨져있는지 검사
                if (brick_array[tmp_x][tmp_y].rectangle[i].x == 0
```

```
                        && brick_array[tmp_x][tmp_y].rectangle[i].y == 0) {
                        tmp++;
                // 벽돌의 모든 좌표(4개)가 원점으로 가있다면 item flag를 true로 만듬
                        // item flag가 true : item falling 수행
                        if (tmp == 4 && item.flag[j] == false) {
                                item.flag[j] = true;
                        }
                }
        }
}
```

▼ 해당 아이템 공 획득 처리

```
// item falling
for (int i = 0; i < 8; i++) {
        // item 벽돌이 깨진 경우 수행
        if (item.flag[i] == true) {
                Modeling_Circle(item_radius, item.item_text[i]);
                // item falling 속도 조절
                item.item_text[i].y -= 3.0;
                // bar의 아랫부분 y 좌표보다 내려갈 경우 (충돌 처리)
                if (item.item_text[i].y <= bar.rectangle[1].y) {
                        item.flag[i] = -1; // 획득 실패 처리
                }
                item_got_it(); // 아이템 획득 처리 -> 아이템 공의 모든 좌표를 원점으로
        }
}
```

3. 프로젝트 소감

3-1. 문제 해결 방식

a. 벽돌 충돌 처리 예외 발생

- 벽돌의 상하좌우 충돌 처리는 문제없이 수행되었지만, 벽돌의 꼭짓점에 정확히 공이 충돌할 경우 상하좌우 충돌 처리 조건에 만족하지 않아서 공이 벽돌을 통과하는 예외 문제가 발생하였다.

∴ 벽돌 모서리 충돌 조건을 추가하여 해결

b. 화면 전환 구성

- 게임이라면 인트로, 게임 오버, 클리어 화면이 필요하다고 생각하여 이미지 추가를 통해 화면을 추가하려고 했으나, 방법이 쉽게 떠오르지 않았다.

∴ 현재 화면의 여부를 알기 위해 bool 변수를 이용하여 해결

c. 충돌 처리 코드 작성 중, 정상 코드인지 확인하기가 어려움

- 공을 이리저리 튀게 하면서 벽돌 충돌 처리가 정상적으로 이루어지고 있는지 확인하기가 어려웠다.
∴ 게임 진행 화면 중, 'F2' Key와 'END' Key를 입력하면 공을 'F3', 'F4', 'F5', 'F6' Key들을 통해 공을 움직여볼 수 있는 Debug Mode에 진입할 수 있도록 하였음.

3-2. 아쉬운 점

- 여러 이미지를 불러오거나 벽돌, 공 등의 도형을 그리는 과정이 계속해서 화면이 재구성되며 호출된다. 이는 프로그램의 작동 속도와 여부에 영향을 끼치지만 마땅한 해결 방법이 떠오르지 않아서 고칠 수 없었다는 점이 아쉬웠다. (오랫동안 프로그램 가동 시, 강제로 프로그램이 종료되는 문제 발생)

3-3. 소감

- openGL을 이용하여 간단한 도형만을 그려보다가, 화면을 계속 재구성할 수 있는 glutIdelFunc 함수를 통해 게임을 만들 수 있겠다는 생각을 했었고, 실제로 이렇게 벽돌깨기 게임을 만들어볼 수 있어서 좋았다. 게임 내에서는 '공이 이동함', '공과 벽돌이 충돌함', '벽돌이 깨져서 사라짐' 등의 단순한 동작만이 이루어진다. 하지만 이들을 직접 다 구현한다는 점이 정말 쉽지 않다는 것을 깊게 깨달을 수 있었고, 개발에 있어서 생각의 흐름이 어떻게 이루어져야하는지 알 수 있었다.