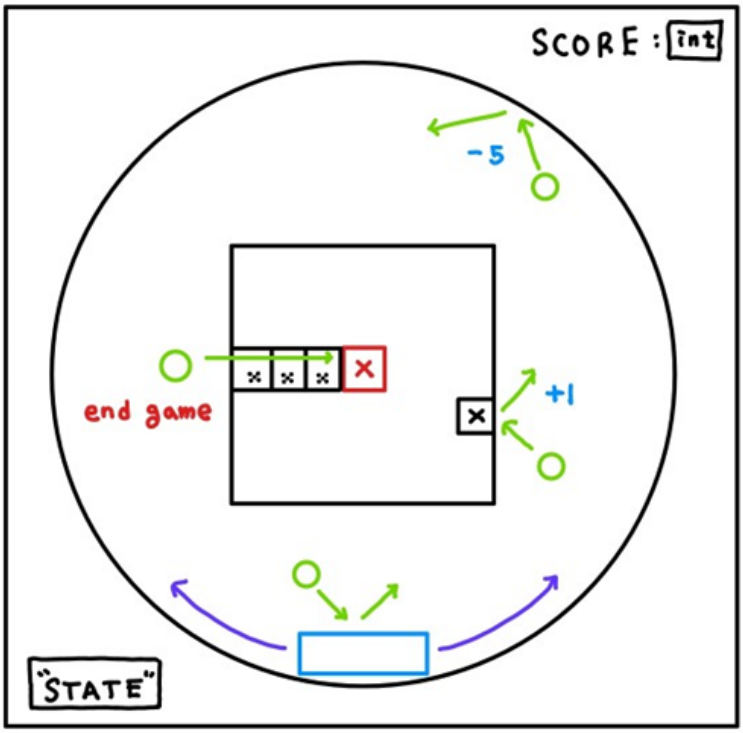
돌려돌려, 볼록 깨기 프로젝트

▼ 게임 설명 [만든이: 20182113 오세현]



- 키보드 입력을 통해 중심 축을 기준으로 Bar 회전
- 블록을 부수면 +1점, 외부 경계에 부딛치면 -5점, Red 블럭에 부딛히면 게임 종료

▼ 목표

RED 블록 및 외부 경계를 피해 최대한 블록을 부수자 !

▼ 게임 설계 과정

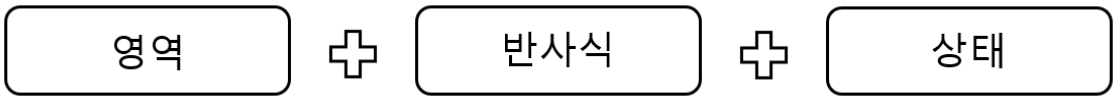
1. 처음 상태일 때 ‘S’를 누를 시 게임 시작
2. 블록을 부수면 +1점, 외부 경계에 부딛치면 -5점
3. 중심의 RED 블록을 부술 시 게임 종료
4. 게임 종료 상태일때 ‘D’를 누를 시 처음 화면으로 이동

▼ 구현해야 할 과정

1. 블록 및 외부 경계 Bar 영역 설정
2. 공이 블록에 부딛치는 위치에 따른 속도 변화
3. 공의 외부 경계에 부딛칠 때 속도 변화
4. 공이 Bar에 부딛칠 때 속도 변화
5. 키보드 입력에 대한 Bar 회전
6. 현재 상태를 화면에 출력 [처음 화면, 게임 중, 끝]

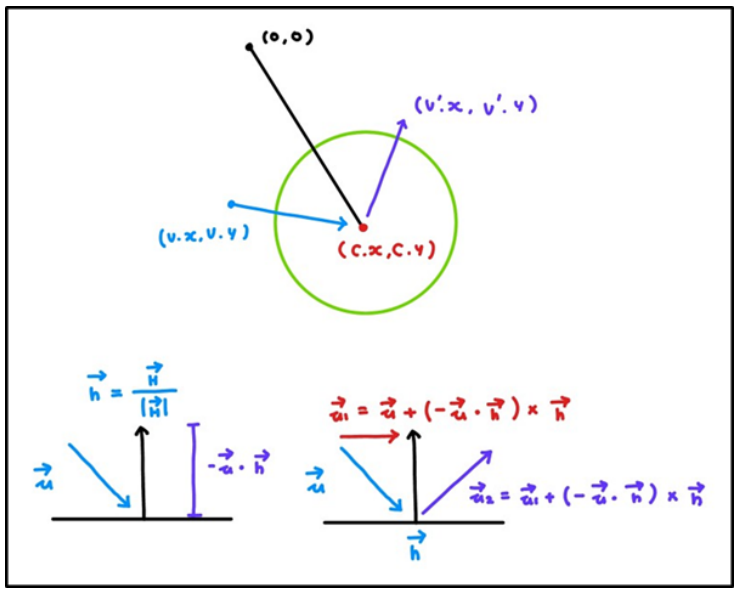
7. 공이 블록, 외부 경계에 부딛칠 때 점수 변화

- 최종적으로 [영역 , 반사식 , 현재 상태]를 구현해야한다.



▼ 통합 반사 코드

▼ 반사 식



▼ 구현 코드

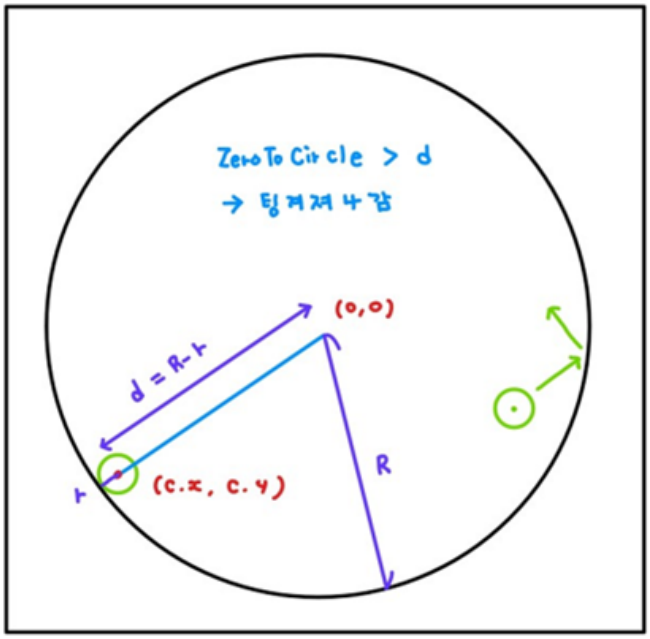
```
void Collision_Detection(void) {

    float nx = 0.0;
    float ny = 0.0;
    float ndx = 0.0;
    float ndy = 0.0;
    float speed = 0.0;
    float distance = 0.0;
    float a = 0.0;

    speed = sqrt(circle_velocity.x * circle_velocity.x + circle_velocity.y * circle_velocity.y);

    distance = sqrt(circle.x * circle.x + circle.y * circle.y);

    nx = -circle.x / distance;
    ny = -circle.y / distance;

    ndx = -circle_velocity.x / speed;
    ndy = -circle_velocity.y / speed;

    a = ndx * nx + ndy * ny;

    if (Collision_Detection_to_Bar(bar_a, bar_b, bar_c, bar_d, circle) == 1) {
        circle_velocity.x = (2.0 * a * nx - ndx) * speed;
        circle_velocity.y = (2.0 * a * ny - ndy) * speed;
    }

    if (Collision_Detection_to_Circle(0.0, 0.0, circle) == 1) {
        circle_velocity.x = (2.0 * a * nx - ndx) * speed;
        circle_velocity.y = (2.0 * a * ny - ndy) * speed;
        score -= 5;
    }

}
```

▼ 설명

입력된 속도 벡터에 대해 반사 속도 벡터를 구하기 위해서는 내적에 대한 이해가 필요하다. 단위 법선 벡터와 입력 속도의 내적 값을 단위 법선 벡터에 곱한 후 입력 속도 벡터에 두 번 더하면 반사 속도 벡터를 구할 수 있다.

▼ 작은 공이 큰 원에 부딛치는 이벤트가 발생하는 영역 지정

▼ 이벤트 영역 식



▼ 구현 코드

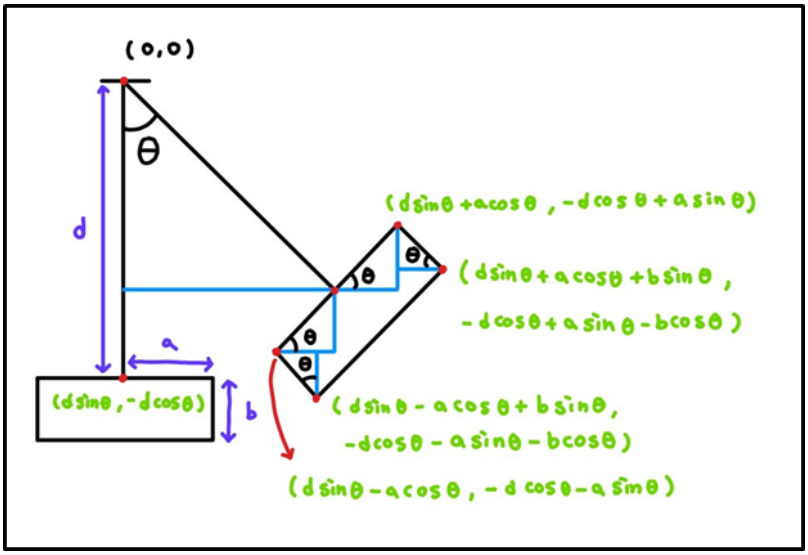
```
int Collision_Detection_to_Circle(float a, float b, c_Point c) {
    float distance_c = 0.0;
    distance_c = sqrt((a - c.x) * (a - c.x) + (b - c.y) * (b - c.y));

    if (distance_c < 520)
        return 0;
    else
        return 1;
}
```

▼ 설명

게임을 구현하기 위해서는 작은 공은 큰 원 안을 자유로이 움직이나 큰 원 밖을 넘어가면 팅겨져 나가는 이벤트가 필요하다. 이를 위해서 큰 원의 내부와 외부 영역 지정이 필요하다. 큰원의 중심 점과 작은 원의 중심 점이 R-r 보다 작으면 원 내부에 위치해 있으나 R-r 보다 큰 경우 외부에 위치에 있음으로 정의한다.

▼ Bar의 움직임 구현

▼ 회전 좌표를 이용한 Bar 이동 구현



▼ 구현 코드

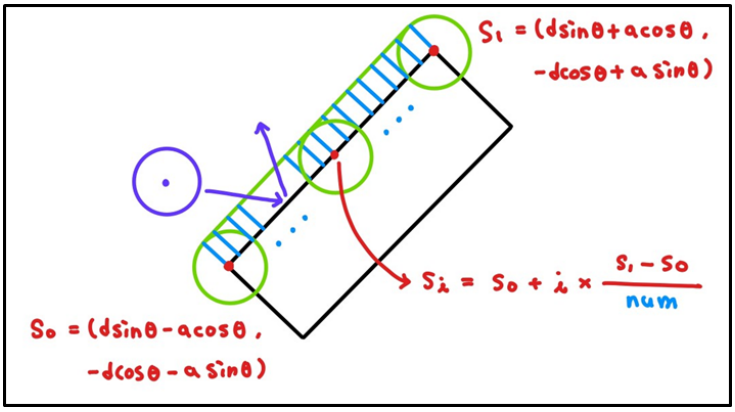
```
void Modeling_Stick(void) {

    if (theta < 0)
        theta += 2 * PI;

    if (theta > 2 * PI)
        theta -= 2 * PI;

    bar_a.x = 500 * sin(theta) + 50.0 * cos(theta);
    bar_a.y = -500 * cos(theta) + 50.0 * sin(theta);

    bar_b.x = 500 * sin(theta) + 50.0 * cos(theta) + 25.0 * sin(theta);
    bar_b.y = -500 * cos(theta) + 50 * sin(theta) - 25.0 * cos(theta);

    bar_c.x = 500 * sin(theta) - 50.0 * cos(theta) + 25.0 * sin(theta);
    bar_c.y = -500 * cos(theta) - 50.0 * sin(theta) - 25.0 * cos(theta);

    bar_d.x = 500 * sin(theta) - 50.0 * cos(theta);
    bar_d.y = -500 * cos(theta) - 50.0 * sin(theta);

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
    glVertex2f(bar_a.x, bar_a.y);
    glVertex2f(bar_b.x, bar_b.y);
    glVertex2f(bar_c.x, bar_c.y);
    glVertex2f(bar_d.x, bar_d.y);
    glEnd();
}
```

▼ 설명

키보드로 theta를 입력 받음을 전제로 한다. Bar가 회전축에 대해 theta만큼 이동하였을 때 네 모서리 좌표를 보정하고 그를 바탕으로 새로 Bar을 그린다. 이 과정이 즉각적으로 이루어 질 시 중심 축에 대한 Bar의 회전을 구현할 수 있다.

▼ 작은 공이 Bar에 부딪치는 이벤트가 발생하는 영역 지정

▼ 이벤트 식



▼ 구현 코드

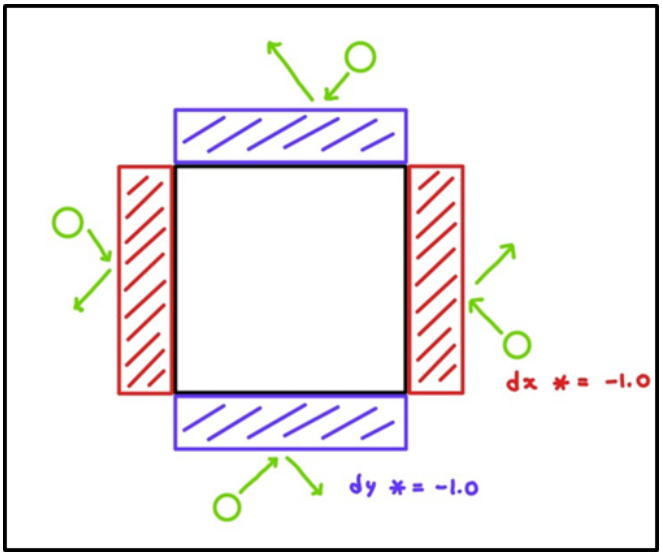
```
int Collision_Detection_to_Bar(Point a, Point b, Point c, Point d, c_Point circle) {

    float dist[101];

    float start_x = a.x;
    float start_y = a.y;
    float delta_x = (d.x - a.x) / 100;
    float delta_y = (d.y - a.y) / 100;

    float total_x = 0.0;
    float total_y = 0.0;

    for (int i = 0; i <= 100; i++) {
        total_x = start_x + i * delta_x;
        total_y = start_y + i * delta_y;

        dist[i] = sqrt((circle.x - total_x) * (circle.x - total_x) + (circle.y - total_y) * (circle.y - total_y));
    }

    for (int i = 0; i <= 100; i++) {
        if (dist[i] < radius)
            return 1;
    }

    return 0;
}
```

▼ 설명

이 식은 작은 공이 bar에 부딛침 판정을 받는 영역을 지정한 코드이다. 공과 부딛히는 한 면을 기준으로 좌표를 구하고 면의 좌표와 공의 좌표 사이의 거리가 r보다 작으면 부딛침 판정을 받도록 한다. 이 코드와 반사 식 코드가 같이 사용되어 부딛침 판정을 받았을 때 반사 식 코드가 작동하는 형태로 되어있다.

▼ 블록 영역 및 반사 코드

▼ 블럭이 공이 부딛칠 때 영역 및 블럭 방향에 따른 반사 식



▼ 블럭 영역 구현 코드

```
int Collision_Detection_x(c_Point ball, float ball_r, c_Point square) {
    if (ball.y + ball_r < square.y - 25)
        return 0;
    else if (ball.y - ball_r > square.y + 25)
        return 0;
    else if (ball.x < square.x - 25)
        return 0;
    else if (ball.x > square.x + 25)
        return 0;
    else
        return 1;

}

int Collision_Detection_y(c_Point ball, float ball_r, c_Point square) {
    if (ball.y < square.y - 25)
        return 0;
    else if (ball.y > square.y + 25)
        return 0;
    else if (ball.x + ball_r < square.x - 25)
        return 0;
    else if (ball.x - ball_r > square.x + 25)
        return 0;
    else
        return 1;

}
```

▼ 블럭 반사 구현 코드

```
void Collision_Detection_to_Block(void) {

    circle.x += circle_velocity.x;
    circle.y += circle_velocity.y;

    for (int i = 0; i < total; i++) {
        if (block[i].collision == 0) {
            if (Collision_Detection_x(circle, radius, block[i]) == 1) {
                block[i].collision = 1;
                circle_velocity.y *= -1.0;
                score++;
                break;
            }
            else if (Collision_Detection_y(circle, radius, block[i]) == 1) {
                block[i].collision = 1;
                circle_velocity.x *= -1.0;
                score++;
                break;
            }
        }

    }
}
```

▼ 설명

블록의 경우 이전의 원 혹은 Bar에서 발생했던 반사식과 다르게 구성되어있다. 블럭의 경우 4면이 공에 의해 부딛힐 수 있는데 좌, 우면에서 공을 부딛힐 시 속도 x가 반전되고 상, 하에서 부딛힐 시 속도 y가 반전된다.

▼ 텍스트 화면 출력

▼ GLUT Tutorials 사이트를 참고하여 텍스트 화면 출력 구현

비트맵폰트

비트맵폰트는 2D 폰트를 말합니다. 비트맵폰트를 3D 상에서 표현하지만 굵기가 없고 회전 및 크기변환을 할 수 없습니다. 오로지 이동만 할 수 있습니다. 그리고 이 폰트는 빌보드처럼 항상 관측자 쪽으로 향하고 있습니다. 이 점이 안 좋은 것 같지만 다르게 생각해보면 폰트의 방향에 대해서 신경 쓰지 않아도 되는 것입니다. 폰트는 항상 관측자 쪽으로 향해있으니까요.

이번 장에서는 비트맵형식의 폰트를 화면에 출력하는 GLUT 함수에 대해서 알아보겠습니다. 우선, glutBitmapCharacter 함수가 필요합니다. 다음은 이 함수의 설명입니다.

```
void glutBitmapCharacter(void *font, int character)

인자 설명:
font - 사용할 폰트의 이름입니다.(아래에 사용가능한 값들이 있습니다.)
character - 렌더링할 대상입니다. 문자, 기호, 숫자 등등...
```

- 출처: https://sungcheol-kim.gitbook.io/glut-tutorials/chapter13

▼ 구현 코드

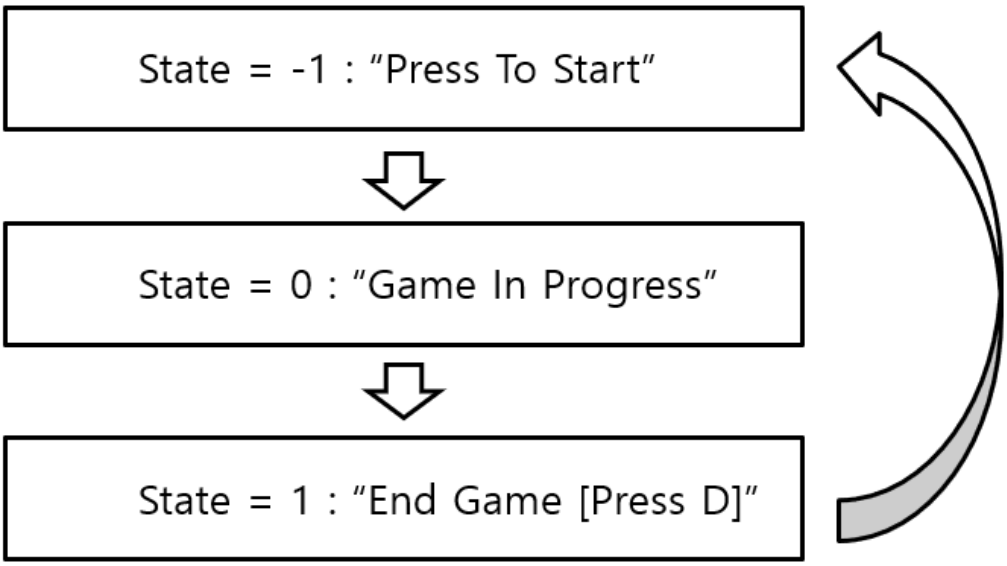
```
void renderBitmapCharacterString(float x, float y, void* font, char* string) {
    char* c;
    glRasterPos2f(x, y);

    for (c = string; *c != '₩0'; c++)
        glutBitmapCharacter(font, *c);
}

void renderBitmapCharacterInt(float x, float y, void* font, int score) {
    char c[100];
    sprintf_s(c, "%d", score);
    glRasterPos2f(x, y);

    for (int i = 0; c[i] != '₩0'; i++)
        glutBitmapCharacter(font, c[i]);
}
```

▼ 설명

사이트에 나온 비트맵 폰트 함수를 하용하였다. glutBitmapCharacter()함수의 경우 문자열만 입력 가능하여 숫자형을 출력하고 싶은 경우 sprintf_s() 함수를 통해 숫자형을 문자형으로 변환하는 과정을 거치었다.

▼ 상태 메시지 구현

▼ 이벤트에 따른 처음 화면, 게임 시작 화면, 게임 종료 화면 설계

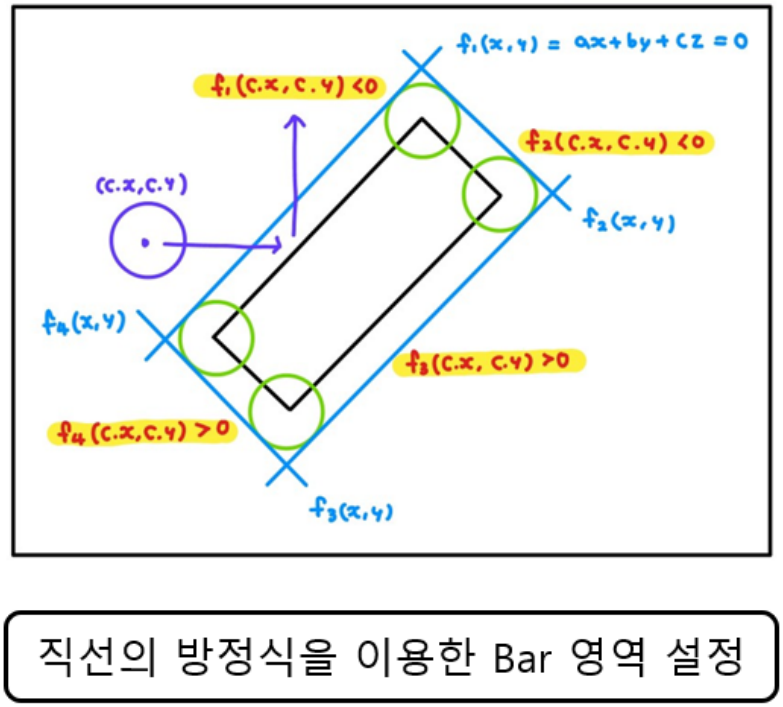


▼ 설명

게임의 처음, 중간, 끝을 설계하기 위해서는 각 상태에 따른 수치를 지정하고 이벤트가 발생하면 다음 수치로 넘어가도록 코드를 구성한다. 이벤트는 지금 현재 상태와 키보드 입력, 블록의 부숨 유무를 조합하여 발생된다.

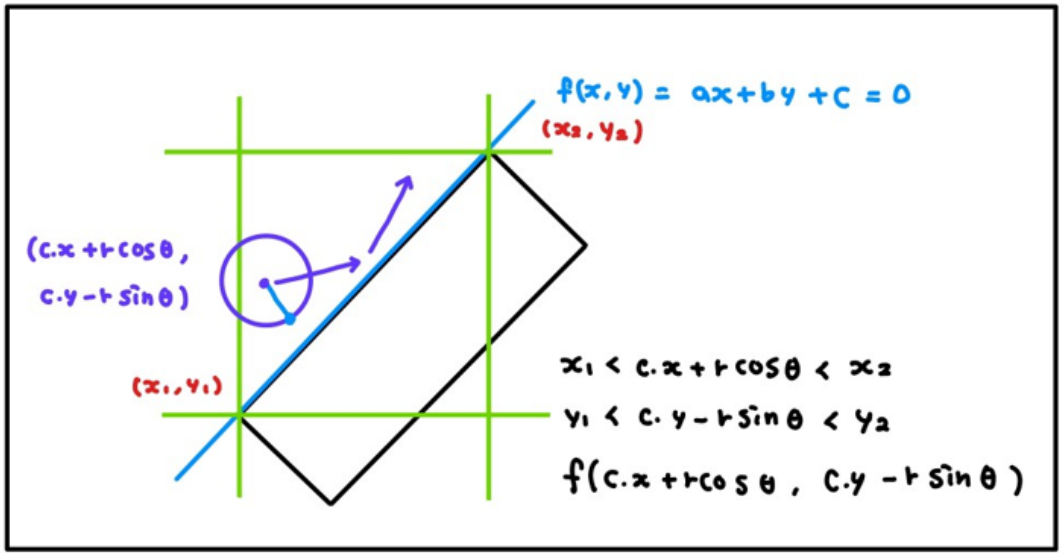
▼ 시행 착오

▼ 직선의 방정식을 이용한 Bar 영역 지정 실패



직선의 방정식을 이용한 Bar 영역 설정

▼ 범위 지정과 직선의 방정식을 이용한 Bar 영역 지정 실패



범위 지정과 직선의 방정식을 통한 Bar 영역 설정

▼ 현 코드의 문제점

▼ Bar 영역 중첩 문제

현재 코드의 경우 작은 공과 bar의 충돌영역을 bar의 한 면에만 적용하였다 그결과 다른 면에 공이 부딛힐 경우 공이 부자연스럽게 움직이는 모습을 확인하였다.

▼ 블록과 공의 부자연스러운 충돌

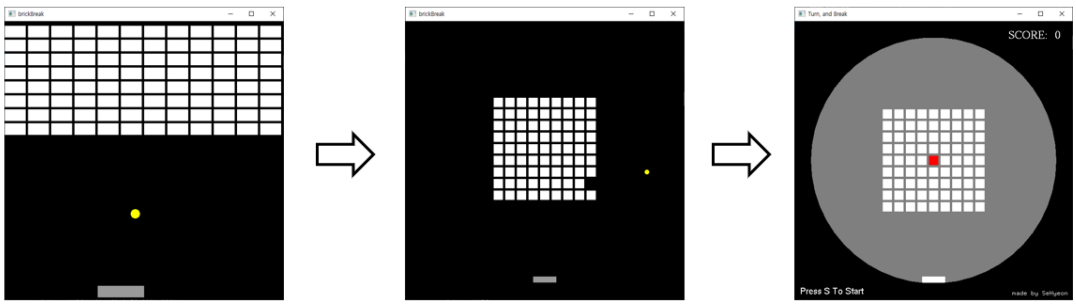
블럭이 공에 부딛히는 영역은 직선적으로 이루어져 있다. 그래서 블럭의 꼭짓점에 공이 부딛힐 시 부자연스러운 반사가 발생한다.

▼ 이번 프로젝트를 진행하며 느낀점

▼ 느낀점

- GLUT를 이용한 프로젝트는 처음이라 힘들었지만 재미있었다.
- 무언가를 만들어 결과물을 도출하는 과정에서 자신감을 얻었다.

▼ 진행 과정



Version_1 Version_2 Version_3

▼ 전체 구현 코드

```
#include <Windows.h>
#include <gl/glut.h>
#include <gl/glu.h>
#include <math.h>
#include <stdio.h>
#include <string.h>

#define width           600
#define height          600
#define polygon_num   50
#define total           81
#define PI              3.141592

int   left = 0;
int   bottom = 0;

float theta = 0.0;


int score = 0;

char state[64];
char text[] = "SCORE:";
char made_by[] = "made by SeHyeon";
char state1[] = "Press S To Start";
char state2[] = "Game In Progress";
char state3[] = "End Game [ Press D To Menu ]";

typedef struct _Point {
    float x;
    float y;
} Point;

Point stick, circle_velocity, bar_a, bar_b, bar_c, bar_d;

float stick_velocity = 0.1;

float radius = 10.0;

float block_x = -200.0;
float block_y = -250.0;

struct c_Point {
    float x, y;
    int collision;
};

c_Point* block;

c_Point circle;

void init_Block(void) {
    block = new c_Point[total];
    int num = 0;
    for (int i = 0; i < total; i++) {
        if (i % 9 == 0) {
            num = 0;
            block_y += 50;
        }
        block[i].x = block_x + 50 * num;
        block[i].y = block_y;

        block[i].collision = 0;

        num++;
    }
}

void init() {
```

```
    circle.x = -200.0;
    circle.y = -400.0;
    circle_velocity.x = 0;
    circle_velocity.y = 0.;

    strcpy_s(state, state1);

    circle.collision = -1;

    init_Block();
}



void MyReshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluOrtho2D(-width, width, -height, height);

}

void Modeling_Circle(c_Point c, float r) {
    if (c.collision == 0) {
        float delta = 2.0 * PI / polygon_num;
        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 0);
        for (int i = 0; i < polygon_num; i++) {
            glVertex2f(c.x + r * cos(delta * i), c.y + r * sin(delta * i));
        }
        glEnd();
    }
}

void Modeling_Big_Circle(float x, float y, float r) {

    float delta = 2.0 * PI / polygon_num;
    glBegin(GL_POLYGON);
    glColor3f(0.5, 0.5, 0.5);
    for (int i = 0; i < polygon_num; i++) {
        glVertex2f(x + r * cos(delta * i), y + r * sin(delta * i));
    }
    glEnd();
}

void Modeling_Square() {
    glBegin(GL_POLYGON);
    glColor3f(0.0, 0.0, 0.0);
    glVertex2f(-600, -600);
    glVertex2f(600, -600);
    glVertex2f(600, 600);
    glVertex2f(-600, 600);
    glEnd();
}

void renderBitmapCharacterString(float x, float y, void* font, char* string) {
    char* c;
    glRasterPos2f(x, y);

    for (c = string; *c != '\0'; c++)
        glutBitmapCharacter(font, *c);
}

void renderBitmapCharacterInt(float x, float y, void* font, int score) {
    char c[100];
    sprintf_s(c, "%d", score);
    glRasterPos2f(x, y);

    for (int i = 0; c[i] != '\0'; i++)
        glutBitmapCharacter(font, c[i]);
}


void Modeling_Stick(void) {

    if (theta < 0)
```

```
        theta += 2 * PI;

    if (theta > 2 * PI)
        theta -= 2 * PI;

    bar_a.x = 500 * sin(theta) + 50.0 * cos(theta);
    bar_a.y = -500 * cos(theta) + 50.0 * sin(theta);

    bar_b.x = 500 * sin(theta) + 50.0 * cos(theta) + 25.0 * sin(theta);
    bar_b.y = -500 * cos(theta) + 50 * sin(theta) - 25.0 * cos(theta);

    bar_c.x = 500 * sin(theta) - 50.0 * cos(theta) + 25.0 * sin(theta);
    bar_c.y = -500 * cos(theta) - 50.0 * sin(theta) - 25.0 * cos(theta);

    bar_d.x = 500 * sin(theta) - 50.0 * cos(theta);
    bar_d.y = -500 * cos(theta) - 50.0 * sin(theta);

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
    glVertex2f(bar_a.x, bar_a.y);
    glVertex2f(bar_b.x, bar_b.y);
    glVertex2f(bar_c.x, bar_c.y);
    glVertex2f(bar_d.x, bar_d.y);
    glEnd();
}

void Modeling_Block(void) {
    for (int i = 0; i < total; i++) {
        if (block[i].collision == 0) {
            glColor3f(1.0, 1.0, 1.0);
            if (i == total / 2)
                glColor3f(1.0, 0.0, 0.0);
            glBegin(GL_POLYGON);
            glVertex2f(block[i].x + 20, block[i].y + 20);
            glVertex2f(block[i].x + 20, block[i].y - 20);
            glVertex2f(block[i].x - 20, block[i].y - 20);
            glVertex2f(block[i].x - 20, block[i].y + 20);
            glEnd();

        }

    }

}

int Collision_Detection_to_Bar(Point a, Point b, Point c, Point d, c_Point circle) {

    float dist[101];

    float start_x = a.x;
    float start_y = a.y;
    float delta_x = (d.x - a.x) / 100;
    float delta_y = (d.y - a.y) / 100;

    float total_x = 0.0;
    float total_y = 0.0;

    for (int i = 0; i <= 100; i++) {
        total_x = start_x + i * delta_x;
        total_y = start_y + i * delta_y;

        dist[i] = sqrt((circle.x - total_x) * (circle.x - total_x) + (circle.y - total_y) * (circle.y - total_y));
    }

    for (int i = 0; i <= 100; i++) {
        if (dist[i] < radius)
            return 1;
    }

    return 0;
}

int Collision_Detection_to_Circle(float a, float b, c_Point c) {
    float distance_c = 0.0;
    distance_c = sqrt((a - c.x) * (a - c.x) + (b - c.y) * (b - c.y));
```

```
    if (distance_c < 520)
        return 0;
    else
        return 1;
}


void Collision_Detection(void) {

    float nx = 0.0;
    float ny = 0.0;
    float ndx = 0.0;
    float ndy = 0.0;
    float speed = 0.0;
    float distance = 0.0;
    float a = 0.0;

    speed = sqrt(circle_velocity.x * circle_velocity.x + circle_velocity.y * circle_velocity.y);

    distance = sqrt(circle.x * circle.x + circle.y * circle.y);

    nx = -circle.x / distance;
    ny = -circle.y / distance;

    ndx = -circle_velocity.x / speed;
    ndy = -circle_velocity.y / speed;

    a = ndx * nx + ndy * ny;

    if (Collision_Detection_to_Bar(bar_a, bar_b, bar_c, bar_d, circle) == 1) {
        circle_velocity.x = (2.0 * a * nx - ndx) * speed;
        circle_velocity.y = (2.0 * a * ny - ndy) * speed;
    }

    if (Collision_Detection_to_Circle(0.0, 0.0, circle) == 1) {
        circle_velocity.x = (2.0 * a * nx - ndx) * speed;
        circle_velocity.y = (2.0 * a * ny - ndy) * speed;
        score -= 5;
    }

}


int Collision_Detection_x(c_Point ball, float ball_r, c_Point square) {
    if (ball.y + ball_r < square.y - 25)
        return 0;
    else if (ball.y - ball_r > square.y + 25)
        return 0;
    else if (ball.x < square.x - 25)
        return 0;
    else if (ball.x > square.x + 25)
        return 0;
    else
        return 1;

}

int Collision_Detection_y(c_Point ball, float ball_r, c_Point square) {
    if (ball.y < square.y - 25)
        return 0;
    else if (ball.y > square.y + 25)
        return 0;
    else if (ball.x + ball_r < square.x - 25)
        return 0;
    else if (ball.x - ball_r > square.x + 25)
        return 0;
    else
        return 1;

}

void Collision_Detection_to_Block(void) {

    circle.x += circle_velocity.x;
    circle.y += circle_velocity.y;
```

```
    for (int i = 0; i < total; i++) {
        if (block[i].collision == 0) {
            if (Collision_Detection_x(circle, radius, block[i]) == 1) {
                block[i].collision = 1;
                circle_velocity.y *= -1.0;
                score++;
                break;
            }
            else if (Collision_Detection_y(circle, radius, block[i]) == 1) {
                block[i].collision = 1;
                circle_velocity.x *= -1.0;
                score++;
                break;
            }
        }

    }
}


void ClearGame(void) {
    if (block[total / 2].collision == 1) {
        circle_velocity.x = 0;
        circle_velocity.y = 0;
        circle.collision = 1;
        strcpy_s(state, state3);
    }
}


void RenderScene(void) {
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);

    Modeling_Square();
    Modeling_Big_Circle(0, 0, 530);
    Modeling_Stick();
    Modeling_Block();
    Modeling_Circle(circle, radius);

    renderBitmapCharacterString(325, 525, GLUT_BITMAP_TIMES_ROMAN_24, text);
    renderBitmapCharacterString(340, -580, GLUT_BITMAP_8_BY_13, made_by);
    renderBitmapCharacterInt(525, 525, GLUT_BITMAP_TIMES_ROMAN_24, score);
    renderBitmapCharacterString(-575, -575, GLUT_BITMAP_HELVETICA_18, state);

    Collision_Detection_to_Block();
    Collision_Detection();
    ClearGame();

    glFlush();
    glutSwapBuffers();

}

void Special_Key(int key, int x, int y) {
    switch (key) {
    case GLUT_KEY_LEFT: theta -= stick_velocity; break;
    case GLUT_KEY_RIGHT: theta += stick_velocity; break;
    }

    glutPostRedisplay();
}

void My_Key(unsigned char key, int x, int y) {
    if (circle.collision == -1) {
        switch (key) {

        case 's': circle_velocity.x = -3;
            circle_velocity.y = -3;
            circle.collision = 0;
            strcpy_s(state, state2);
            break;

        }
    }
    if (circle.collision == 1) {
```

```
        switch (key) {

        case 'd':    circle.x = -200.0;
            circle.y = -400.0;
            circle.collision = -1;
            strcpy_s(state, state1);
            score = 0;
            for (int i = 0; i < total; i++)
                block[i].collision = 0;
            break;


        }
    }
}

void main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Turn, and Break");
    init();
    glutDisplayFunc(RenderScene);
    glutIdleFunc(RenderScene);
    glutReshapeFunc(MyReshape);
    glutKeyboardFunc(My_Key);
    glutSpecialFunc(Special_Key);
    glutMainLoop();
}
```