Dodge Game 구현 설명

신태욱

☞전체 게임 설명

1. 사방에서 나오는 적기와 해당 적기들이 발사하는 총알을 피하고, 위에서 떨어지는 운석, 일정 시간 지난 뒤 나타나는 유도형 총알을 피하는 게임입니다.

2. 이에 네 방향으로 이동할 수 있고, 네 방향으로 모두 공격이 가능합니다. 적기체는 공격하여 없앨 수 있으며, 점수 또한 얻을 수 있습니다.

3. 적기체의 경우 3가지의 패턴을 지니며, 아이템은 체력증가/파워증가/쉴드생성 3가지를 만들었습니다.

3.1. 추가적으로 점수 2000점 당 폭탄 아이템 1개씩을 지급하며, 폭탄 아이템은 사용시 전지역의 적개체, 총알을 삭제하게됩니다.

4. 0~100초 까지 20초를 단위로 난이도의 증가를 삽입하였고, 100초를 버텼을 경우에는 보스로 진입을 하게 됩니다.

5. 보스의 경우 4가지의 패턴을 지니고 있으며, 보스기체 명중에 따라 점수를 얻어 폭탄 아이템 또한 얻을 수 있도록 구현하였습니다.

6. 만일 죽거나, 클리어를 하게된다면 얻은 점수와 버틴 시간을 보여주게됩니다.

☞세부 설명

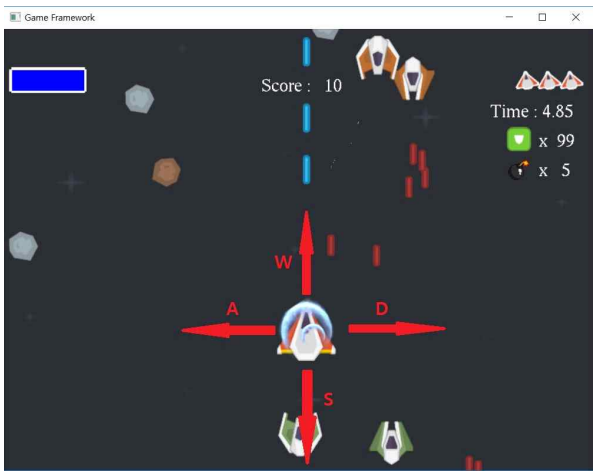
저는 이번 프로젝트에서 플레이어 객체의 움직임과 객체의 스킬 부분을 분담받아서 코드를 구현하도록 했습니다.

- 플레이어의 공격
- 플레이어의 특별 공격 기능 구현
- 키보드 입력에 따른 플레이어의 이동 및 공격 구현 및 동시 입력 기능 지원
- 플레이어의 방어막(실드) 구현
- 플레이어의 상태 화면 출력
- 기타 기능 구현

이렇게 부분으로 나누어서 설명드리도록 하겠습니다.

◎플레이어의 공격

교수님께서 '슈팅게임'으로 프로젝트를 제출하라고 하신 날부터 저희 조는 어떤 슈팅게임을 할까 고심한 끝에 조작하기 간편하고 쉽게 즐길 수 있는 게임을 만들기로 했습니다. 그리고 그렇게 생각을 하던 중 저는 곰플레이어에서 쉽게 할 수 있는 '닷지'라는 프로그램을 생각하게 되었고, 닷지를 업그레이드 시켜서 적 객체를 우리가 공격할 수 있도록 바꿔보기로 했습니다.
그렇게 닷지게임 같은 경우는 사방에서 적 객체가 날라오기 때문에 수업시간에 주어진 단방향 공격으로는 안되었기에 사방으로 공격을 할 수 있도록 방향을 파라미터로 넣었습니다.



```
void createPlayerBullet(int bullet_direction) //플레이어의 총알 생성함수
{
    GameObject* bullet_obj = newObject(); // 총알 객체 생성

    if ((bullet_obj) && (player_bullet_count <= 5)) { // 플레이어의 현재 총알 객체 수를 5개로 제한
        player_bullet_count += 1; //총알 객체가 생성될 때마다 객체수 1 증가
        bullet_obj->type = PLAYER_BULLET; //총알 타입
        bullet_obj->bitmap = createBitmap("laserBlue03.png"); //총알 이미지
        //EAST
        if (bullet_direction == EAST) { //만약 동쪽(오른쪽)으로 발사 할 때
            bullet_obj->direction = bullet_direction;
            bullet_obj->width = 20.0f * power * 0.5f;
            bullet_obj->height = 40.0f;
            bullet_obj->x = player_obj->x + player_back_obj->width / 2 + bullet_obj->width / 2;
            bullet_obj->y = player_obj->y;
            bullet_obj->vx = BULLET_SPEED * 2.0f; //총알 속도 설정
            bullet_obj->vy = 0.0f;
            bullet_obj->a = 270.0f;
            player_back_obj->a = 270.0f; //플레이어 객체 또한 동쪽으로 향하도록 한다.
        }//WEST
        else if (bullet_direction == WEST) { //만약 서쪽(왼쪽)으로 발사 할 때
            bullet_obj->direction = bullet_direction;
            bullet_obj->width = 20.0f * power * 0.5f;
            bullet_obj->height = 40.0f;
            bullet_obj->x = player_obj->x - player_back_obj->width / 2 - bullet_obj->width / 2;
            bullet_obj->y = player_obj->y;
            bullet_obj->vx = -BULLET_SPEED * 2.0f;
            bullet_obj->vy = 0.0f;
            bullet_obj->a = 90.0f;
            player_back_obj->a = 90.0f; //플레이어 객체 또한 서쪽으로 향하도록 한다.
        }//SOUTH
        else if (bullet_direction == SOUTH) { //만약 남쪽(아랫쪽)으로 발사 할 때
            bullet_obj->direction = bullet_direction;
            bullet_obj->width = 20.0f * power * 0.5f;
            bullet_obj->height = 40.0f;
            bullet_obj->x = player_obj->x;
            bullet_obj->y = player_obj->y - player_back_obj->height / 2 - bullet_obj->height / 2;
            bullet_obj->vx = 0.0f;
            bullet_obj->vy = -BULLET_SPEED * 2.0f;
            bullet_obj->a = 180.0f;
            player_back_obj->a = 180.0f; //플레이어 객체 또한 남쪽을 향하도록 한다.
        }//NORTH
        else if (bullet_direction == NORTH) { //만약 북쪽(위쪽)으로 발사 할 때
            bullet_obj->direction = bullet_direction;
            bullet_obj->width = 20.0f * power * 0.5f;
            bullet_obj->height = 40.0f;
            bullet_obj->x = player_obj->x;
            bullet_obj->y = player_obj->y + player_back_obj->height / 2 + bullet_obj->height / 2;
            bullet_obj->vx = 0.0f;
            bullet_obj->vy = BULLET_SPEED * 2.0f;
            bullet_obj->a = 0.0f;
            player_back_obj->a = 0.0f; //플레이어 객체 또한 북쪽으로 향하도록 한다.
        }
        return;
    }
    deleteObject(bullet_obj);
}
```

이렇게 동서남북 방향 4방향으로 나누어서 각각 총알객체를 구현하도록 했으며
원래 수업시간에 구현했던 슈팅게임에서는 총알 객체가 poolsize상에서는 계속 만들어지도록 되어있었지만 저희는 과도한 총알 객체 생성으로 인한 버벅거리는 증상을 줄이기 위해서 화면 상에 있는 총알 객체를 5개로 제한 시키도록 했습니다.
그리고 각각 총알을 발사하는 방향에 따라 플레이어 객체또한 방향을 틀도록 구현했습니다.

```
void movePlayerBullet(GameObject* bullet) //플레이어객체의 총알을 이동시키는 함수
{
    if (bullet)
    {
        bullet->x += bullet->vx; //총알 객체를 생성할 때 설정했던 bullet->vx의 속도로 x좌표 이동
        bullet->y += bullet->vy; //총알 객체를 생성할 때 설정했던 bullet->vy의 속도로 y좌표 이동

        //EAST
        if (bullet->vx > 0) { //만약 동쪽(오른쪽)으로 총알 방향을 설정했을 때
            if (bullet->x + bullet->width / 2 > WIN_WIDTH) //총알이 화면 벗어난 경우
            {
                deleteObject(bullet); //총알 객체 삭제
                player_bullet_count--; //총알 객체 갯수 1 감소
            }
        }//WEST
        else if (bullet->vx < 0) { //만약 서쪽(왼쪽)으로 총알 방향을 설정했을 때
            if (bullet->x - bullet->width / 2 < 0.0f) //총알이 화면 벗어난 경우
            {
                deleteObject(bullet); //총알 객체 삭제
                player_bullet_count--; //총알 객체 갯수 1 감소
            }
        }//NORTH
        else if (bullet->vy > 0) { //만약 북쪽(위쪽)으로 총알 방향을 설정했을 때
            if (bullet->y - bullet->height / 2 > WIN_HEIGHT) //총알이 화면 벗어난 경우
            {
                deleteObject(bullet); //총알 객체 삭제
                player_bullet_count--; //총알 객체 갯수 1 감소
            }
        }//SOUTH
        else if (bullet->vy < 0) { ////만약 남쪽(아랫쪽)으로 총알 방향을 설정했을 때
            if (bullet->y + bullet->height / 2 < 0.0f) //총알이 화면 벗어난 경우
            {
                deleteObject(bullet); //총알 객체 삭제
                player_bullet_count--; //총알 객체 갯수 1 감소
            }
        }
    }
}
```

이렇게 createPlayerBullet()함수에서 받은 vx, vy값으로 총알의 속도를 지정해서 movePlayerBullet()함수에서는 그 총알 속도에 따라 총알 객체를 움직이도록 설정했습니다.

◎플레이어의 특별 공격 기능 구현

이렇게 4방향으로 공격을 구현한 뒤에 보통 슈팅게임 같은 경우 플레이어 객체가 쓸 수 있는 필살기가 하나씩은 있기 마련입니다.

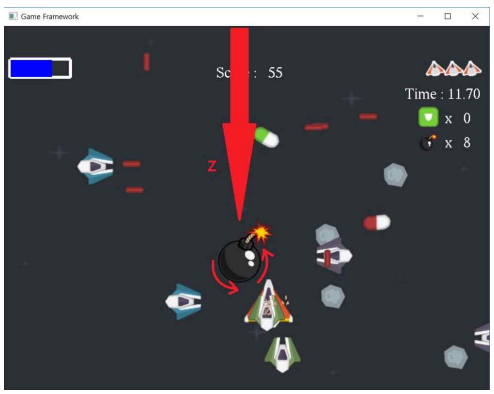
그래서 제가 생각해 낸 것은 게임을 하다보니 적 객체, 돌덩이들(메테오), 그리고 적 객체가 발사한 총알 객체들이 너무나도 많아서 그 객체들에게 모두 타격을 줄 수 있는 전체적으로 데미지를 줄 수 있는 필살기를 만들고자 했습니다.

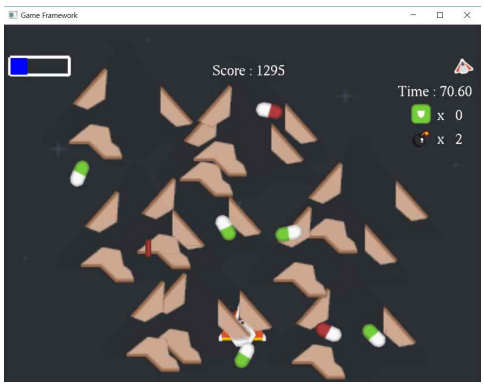
```
void createPlayerBomb() //폭탄을 생성하는 함수
{
    GameObject* bomb_obj = newObject(); //폭탄 객체 생성

    if (bomb_obj) {
        bomb_obj->type = PLAYER_BOMB; //폭탄 타입 설정
        bomb_obj->bitmap = createBitmap("bomb.png"); //폭탄 비트맵 생성

        bomb_obj->direction = SOUTH; //폭탄이 떨어지는 방향 남쪽으로 설정
        bomb_obj->width = 100.0f;
        bomb_obj->height = 100.0f; //폭탄 객체의 너비와 높이 설정
        bomb_obj->x = WIN_WIDTH / 2;
        bomb_obj->y = WIN_HEIGHT; //폭탄 객체 생성 위치설정(화면 상단부 중간에서 생성됨)
        bomb_obj->vx = 0.0f;
        bomb_obj->vy = -10.0f; //폭탄 객체 이동 속도
        bomb_obj->a = 0.0f;

        return;
    }
    deleteObject(bomb_obj);
}
```

먼저 폭탄 객체를 생성하도록 했습니다.
폭탄 객체가 생성되면 화면의 중간 상단부에서부터 떨어지도록 설정했습니다.

```
void createBombparticle() //폭탄 객체가 터질 때의 particle 생성함수
{
    int i, j;
    for (i = 0; i < 10; i++)  //particle객체 10개 생성
    {
        GameObject* particle = newObject();

        if (particle)
        {
            particle->type = BOMB_PARTICLE;
            particle->bitmap = createBitmap("particle3.png");
            particle->width = WIN_WIDTH / 3.0f;
            particle->height = WIN_HEIGHT / 3.0f;
            particle->x = WIN_WIDTH / 2.0f + randomFloat(-200.0f, +200.0f);
            particle->y = WIN_HEIGHT / 2.0f + randomFloat(-200.0f, +200.0f); //파티클 객체 생성 위치는 랜덤으로 설정
            particle->a = 0.0f;

            for (j = 0; j < POOL_SIZE; j++)
            {
                GameObject* obj = getObject(j);
                if (obj->type == ENEMY1 || obj->type == ENEMY2 || obj->type == ENEMY3
                    || obj->type == ENEMY4 || obj->type == BOSS
                    || obj->type == ENEMY_BULLET || obj->type == RANDOM_PARTICLE || obj->type == RANDOM_BULLET)
                { //파티클객체를 생성 할 때 생성되어있는 적 객체, 보스 객체, 적 객체의 총알 등이 있다면

                    obj->health -= 4; //각각 -4씩 체력을 닳게 한다 (결국 particle객체수(10) × 4 = 40 → 객체의 체력 40이 감소)
                    if (obj->health <= 0) { //만약 적 객체의 체력이 0보다 적다면
                        if (obj->type == ENEMY1) score += HIT_SCORE;
                        else if (obj->type == ENEMY2) score += HIT_SCORE * 2;
                        else if (obj->type == ENEMY3) score += HIT_SCORE * 3;
                        else if (obj->type == ENEMY4) score += HIT_SCORE * 4;
                        deleteObject(obj); //점수를 올리고 적객체를 삭제합니다.
                    }
                }
            }

        }
    }
}
```

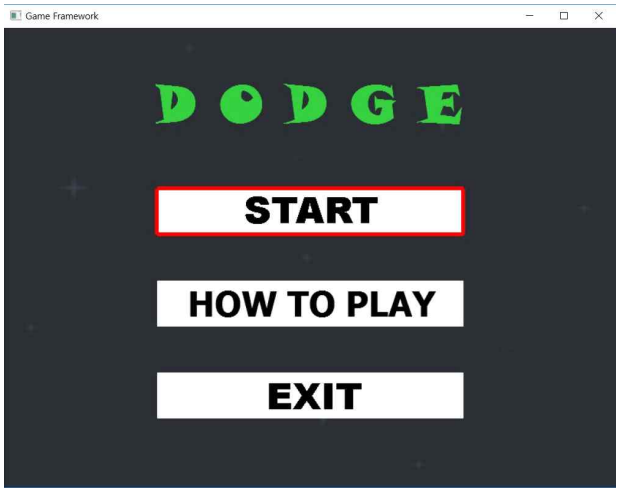
그리고 폭탄이 터지면(폭탄객체가 삭제될 때) particle을 10개 생성하도록 했습니다.
화면에서 랜덤으로 10개의 particle이 나타나게 됩니다.
그래서 그 각각의 particle마다 현재 생성되어있는 적 객체, 보스 객체, 적 객체의 총알에 -4씩 데미지를 주도록 했으며, 10개의 파티클이 총 -40의 데미지를 주게됩니다.
그리고 폭탄으로 인해 적객체가 죽었을 경우에도 스코어를 얻을 수 있도록 설정했습니다.

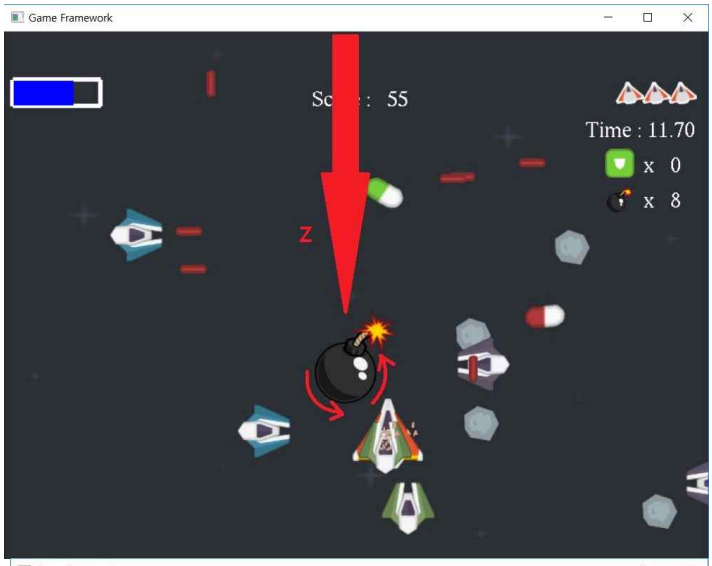


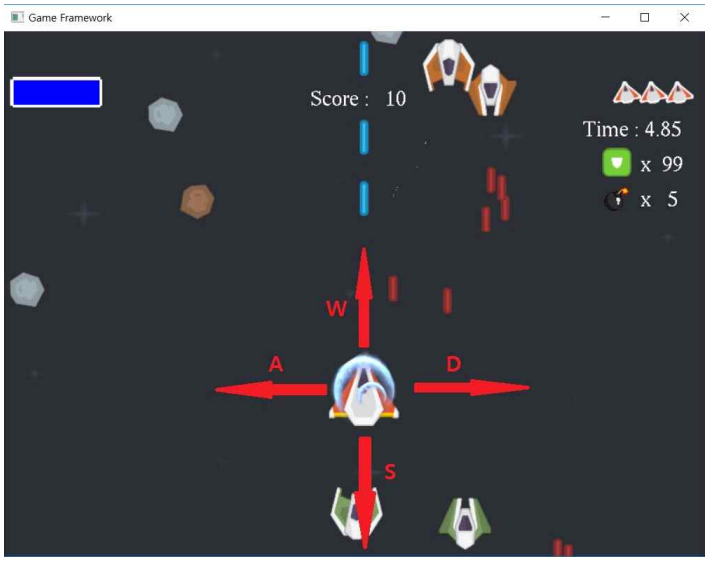


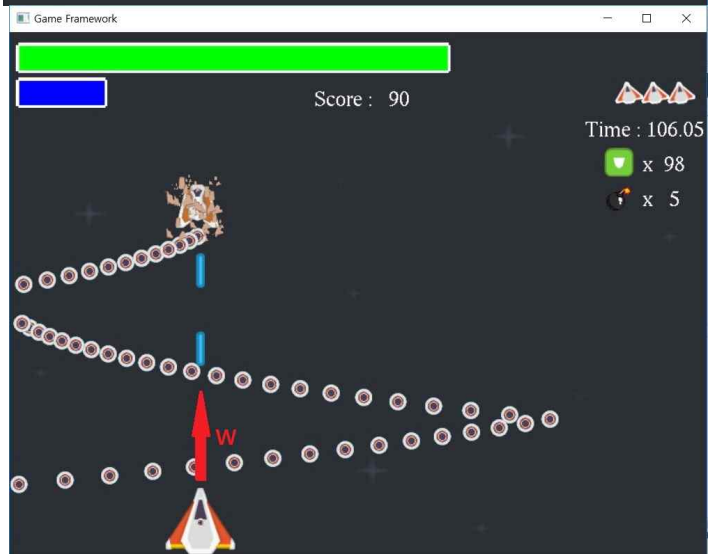
◎키보드 입력에 따른 플레이어의 이동 및 공격 구현 및 동시 입력 기능 지원

```
void handleGeneralKey(unsigned char key, int x, int y) //키보드 입력에 따른 기능 출력 함수
{
    switch (key)
    {
    case 13:        // enter를 눌렀을 때
    {
        if (game_mode == PRE_GAME) // pre-game상태(메뉴 고를 때의 상태)
        {
            if (menu_selection == 0) //start 선택 했을 경우
            {
                changeGameMode(READY_GAME);
            }
            else if (menu_selection == 1) //how to play 선택 했을 경우
            {
                changeGameMode(EXPLAIN_GAME);
            }
            else if (menu_selection == 2) //exit 선택 했을 경우
            {
                finalize();
                exit(0);
            }
        }
        else if ((game_mode == POST_GAME) || (game_mode == CLEAR_GAME) || (game_mode == EXPLAIN_GAME))
        // 만약 중간에 game over되었을 경우, clear했을 경우, 그리고 how to play 메뉴에 들어가있을 경우
        {
            changeGameMode(PRE_GAME); //엔터를 누르면 메뉴 선택하는 모드로 바꿉니다.
        }
    }
    break;

    case 68:    // D //EAST
    case 100:   // d
    {
        if (game_mode == IN_GAME) { //게임 중인 상태에서
            createPlayerBullet(EAST); //동쪽으로 총알을 생성합니다.
        }
    }
    break;
    case 65:    // A //WEST
    case 97:    // a
    {
        if (game_mode == IN_GAME) { //게임 중인 상태에서
            createPlayerBullet(WEST); //서쪽으로 총알을 생성합니다.
        }
    }
    break;
    case 83:    // S //SOUTH
    case 115:   // s
    {
        if (game_mode == IN_GAME) { //게임 중인 상태에서
            createPlayerBullet(SOUTH); //남쪽으로 총알을 생성합니다.
        }
    }
    break;
    case 87:    // W //NORTH
    case 119:   // w
    {
        if ((game_mode == IN_GAME) || (game_mode == IN_BOSS)) { //만약 게임 중이거나 보스가 출현한 상태에서는
            createPlayerBullet(NORTH); //북쪽으로 총알을 생성합니다.
        } //보스가 출현했을 때는 한방향(북쪽)으로만 총알이 생성가능하도록 했습니다.
    }
    break;
    case 90:   //Z
    case 122:   //z
    {
        if ((game_mode == IN_GAME) || (game_mode == IN_BOSS)) { //만약 게임 중이거나 보스가 출현한 상태에서는
            if (bomb > 0) { //가용 폭탄이 있다면
                createPlayerBomb(); // 폭탄 객체를 생성 하고
                bomb--; //가용 폭탄 갯수를 줄입니다.
            }
        }
    }
    break;
    }
}
```









handlegeneralkey()함수 구현 부분

- 메뉴 고를 때 start, how to play, exit칸에 대해서 '엔터'를 누를 경우 각 메뉴 실행

- game over, game clear, how to play의 상황에서 '엔터'를 누를 경우 메뉴창으로 다시 돌아가기

- 'd', 'a', 's', 'w' 각각 동, 서, 남, 북쪽으로 플레이어 객체가 총알을 발사하도록 합니다.

- 보스 객체가 발생한 이후부터는 'w'키를 이용한 위쪽으로의 공격만 가능하도록 구현했습니다. (보스 객체가 나타나고 보스 객체와 싸울 경우에 보스 객체가 총알을 발산하는 일정 패턴이 있는데 그 패턴에 피해서 옆이나 뒤에서 공격하는 꼼수를 방지하기 위함입니다.)

- 'z'키를 누르면 가용 폭탄이 있을 경우 폭탄 필살기를 쓸 수 있습니다.

```
void handleSpecialKey2() //동시입력 지원 함수(만약 공격중(w,a,s,d키를 누르는 중)이더라도 방향키 입력을 받아서 처리)
{
    if ((game_mode == IN_GAME) || (game_mode == IN_BOSS)) { //게임중이거나 보스 출현 상태에서
        if ((GetAsyncKeyState(VK_UP) & 0x8000) && !(player_obj->y + player_obj->height / 2 >= WIN_HEIGHT)) { //만약 ↑키를 누른상태에서 플레이어가 화면을 벗어 나지 않았을때
            player_obj->y += PLAYER_SPEED;
            player_back_obj->y += PLAYER_SPEED; //플레이어 객체를 위로 이동시킨다.
        }
        if ((GetAsyncKeyState(VK_LEFT) & 0x8000) && !(player_obj->x - player_obj->width / 2 <= 0)) { //만약 ←키를 누른상태에서 플레이어가 화면을 벗어 나지 않았을때
            player_obj->x -= PLAYER_SPEED;
            player_back_obj->x -= PLAYER_SPEED; //플레이어 객체를 왼쪽으로 이동시킨다.
        }
        if ((GetAsyncKeyState(VK_RIGHT) & 0x8000) && !(player_obj->x + player_obj->width / 2 >= WIN_WIDTH)) { //만약 →키를 누른상태에서 플레이어가 화면을 벗어 나지 않았을 때
            player_obj->x += PLAYER_SPEED;
            player_back_obj->x += PLAYER_SPEED; //플레이어 객체를 오른쪽으로 이동시킨다.
        }
        if ((GetAsyncKeyState(VK_DOWN) & 0x8000) && !(player_obj->y - player_obj->height / 2 <= 0)) { //만약 ↓키를 누른상태에서 플레이어가 화면을 벗어 나지 않았을때
            player_obj->y -= PLAYER_SPEED;
            player_back_obj->y -= PLAYER_SPEED; //플레이어 객체를 아랫쪽으로 이동시킨다.
        }
    }
}
```

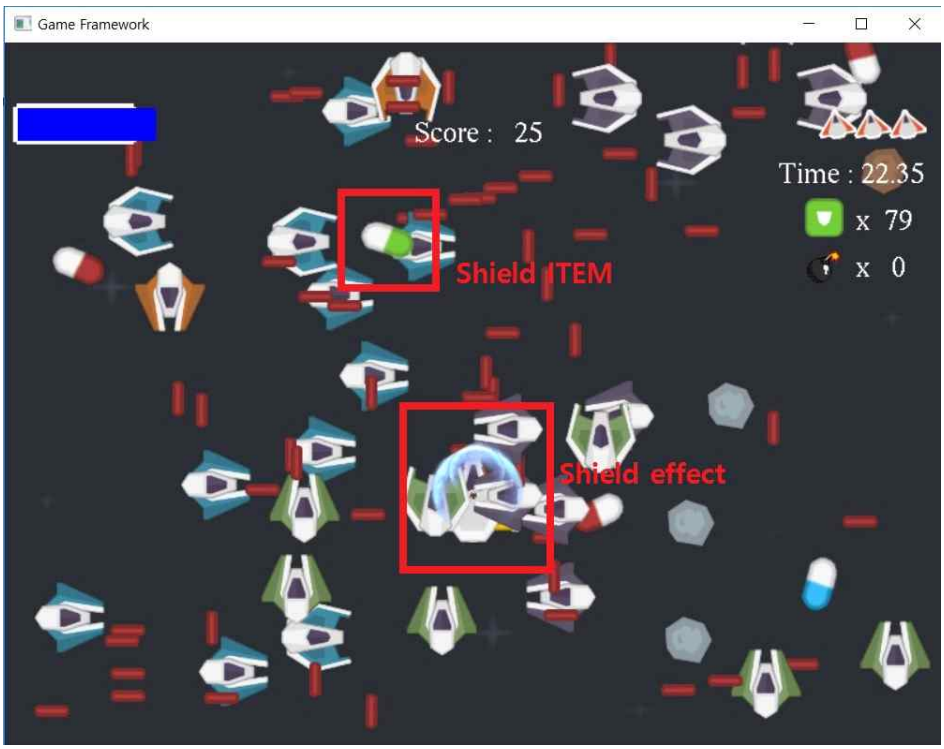
handlespeciallkey()함수

GetAsyncKeyState()함수를 통해서 현재 방향키를 누르고 있는지 여부를 받아서 방향키의 방향에 따라 플레이어 객체를 이동시킬 수 있도록 했습니다.
또한 방향키 이외에 다른 키를 누르면서 동시에 방향키를 누르더라도 인식을 할 수 있으므로 이동과 동시에 공격의 구현이 가능해졌습니다.

◎플레이어의 방어막(실드) 구현

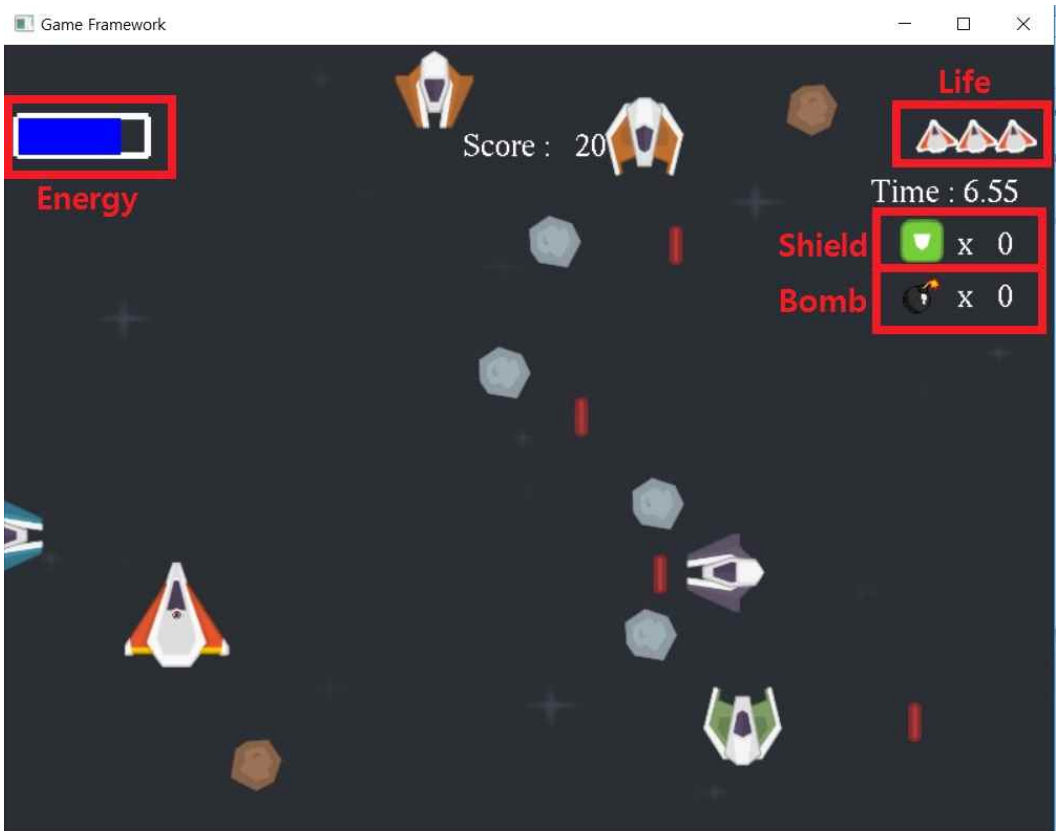
```
void createShieldParticles(GameObject* obj) //실드 particle 생성하는 함수
{
    int i;
    for (i = 0; i < 2; i++) { //실드 particle 2개 생성
        GameObject* shield_particle = newObject();

        if ((shield) && (shield_particle)) {
            shield_particle->type = PARTICLE;
            shield_particle->bitmap = createBitmap("Shield_particle.png");
            if (i == 0) { //첫번째 particle은
                shield_particle->width = shield_particle->height = 120.0f; //크게 생성
                shield_particle->x = player_obj->x;
                shield_particle->y = player_obj->y;
            }
            else { //나머지 particle은
                shield_particle->width = shield_particle->height = randomFloat(30.0f, 80.0f); //랜덤 크기로 생성
                shield_particle->x = obj->x + randomFloat(-25.0f, +25.0f);
                shield_particle->y = obj->y + randomFloat(-25.0f, +25.0f);
            }
            shield_particle->a = 0.0f;
        }
    }
}
```

게임의 기능을 업그레이드 하라고 하신 2번째 과제 부분에서 구현했던 실드 함수를 그대로 가져와 사용하도록 했습니다. 실드 파티클을 2개 생성하며 첫 번째 파티클은 크게 생성하고 2번째 파티클은 랜덤크기로 설정해서 생성토록 했습니다.



◎플레이어의 상태 화면 출력

```
void drawPlayerStates() //화면에 플레이어의 상태를 알려주는 객체들을 만들어서 띄어놓는 함수

{
    // energy
    setColor(1, 1, 1); //에너지 상태 틀을 흰색으로 설정
    strokeRectangle(60, WIN_HEIGHT - 70, 100, 30, 0); //사각형의 에너지 상태 틀 생성
    if (player_obj) {
        if (player_obj->health <= 20) //플레이어 객체의 체력이 20 이하면 에너지바를 빨간색으로 설정
        {
            setColor(1, 0, 0);
        }
        else //20 보다 크다면 파란색으로 설정
        {
            setColor(0, 0, 1);
        }
        fillRectangle(60 - (100 - player_obj->health) / 2, WIN_HEIGHT - 70, player_obj->health - 2, 28, 0); //남은 체력치 만큼 에너지바 채웁니다.
    }
    // life
    if (num_life != 0) { //만약 남은 life가 0이 아닐때
        if (num_life >= 1) { //life가 1이상
            drawBitmap(life_image1, WIN_WIDTH - 120 + 3 * 30, WIN_HEIGHT - 70, life_image1->width, life_image1->height, 10);
            if (num_life >= 2) { // life가 2이상
                drawBitmap(life_image2, WIN_WIDTH - 120 + 2 * 30, WIN_HEIGHT - 70, life_image2->width, life_image2->height, 10);
                if (num_life >= 3) { // life가 3이상
                    drawBitmap(life_image3, WIN_WIDTH - 120 + 1 * 30, WIN_HEIGHT - 70, life_image3->width, life_image3->height, 10);
                } //life는 이미지 객체를 생성하여 나타냈습니다.
            }
        }
    }
    // shield
    char shield_str[15]; //현재 보유하고 있는 실드의 갯수를 나타냅니다.
    sprintf(shield_str, " x %3d", shield);
    setColor(1, 1, 1);
    drawBitmap(shield_image, WIN_WIDTH - 100, WIN_HEIGHT - 150, shield_image->width, shield_image->height, 0);
    drawText(WIN_WIDTH - 80, WIN_HEIGHT - 160, GLUT_BITMAP_TIMES_ROMAN_24, shield_str);

    // bomb
    char bomb_str[15]; //현재 보유하고 있는 폭탄의 갯수를 나타냅니다.
    sprintf(bomb_str, " x %3d", bomb);
    setColor(1, 1, 1);
    drawBitmap(bomb_image, WIN_WIDTH - 100, WIN_HEIGHT - 190, 30, 30, 0);
    drawText(WIN_WIDTH - 80, WIN_HEIGHT - 200, GLUT_BITMAP_TIMES_ROMAN_24, bomb_str);
}
```



drawPlayerState()함수를 통해서 플레이어 객체의 Life, Energy, Shield, Bomb 보유 상태를 화면에 출력시키도록 했습니다.

◎기타 기능 구현

-플레이어 객체가 죽었을 때(Life가 닳았을 때)

```
void createDieParticles2() //플레이어 객체가 죽었을 때 나오는 particle 함수
{
    int i;
    for (i = 0; i < 10; i++) //particle 객체 10개 생성
    {
        GameObject* particle = newObject();

        if (particle)
        {
            particle->type = PARTICLE;
            particle->bitmap = createBitmap("particle2.png");
            particle->width = WIN_WIDTH / 2.0f;
            particle->height = WIN_HEIGHT / 2.0f;
            particle->x = WIN_WIDTH / 2.0f + randomFloat(-100.0f, +100.0f);
            particle->y = WIN_HEIGHT / 2.0f + randomFloat(-100.0f, +100.0f); //파티클 생성 위치를 랜덤으로 잡았습니다.
            particle->a = 0.0f;
        }
    }
}
```

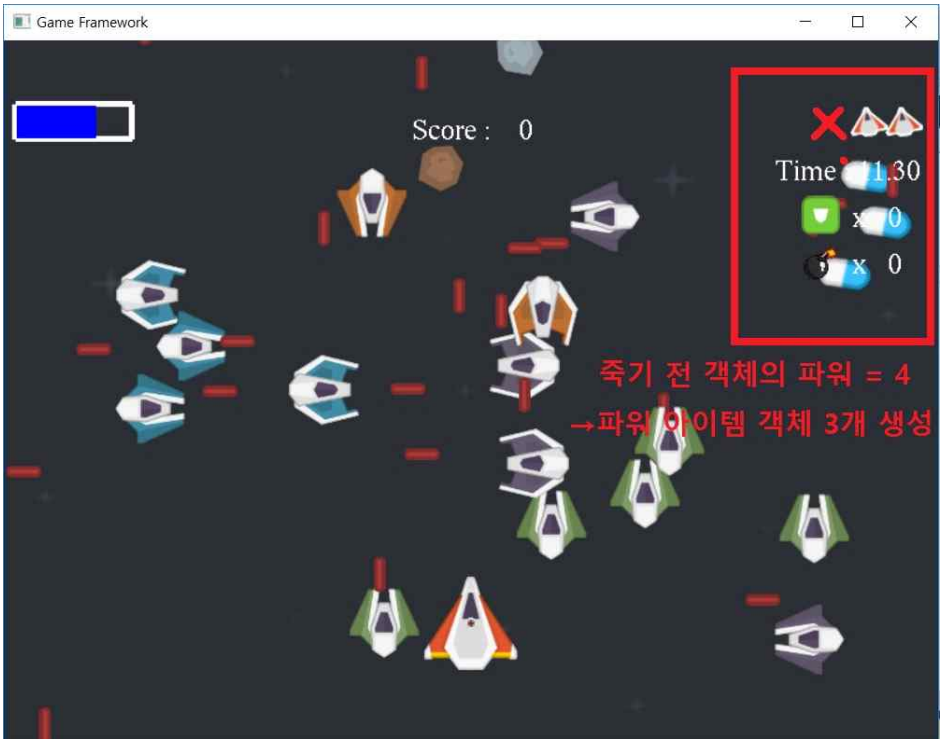
플레이어 객체가 죽었을 경우 particle을 화면 중앙에 발생시켰습니다.

```
void diemotion_createItemPower() { //플레이어 객체 죽었을 때 파워아이템 생성 함수

    for (int i = 1; i < power; power--) { //죽기 직전 power-1갯수 만큼 power 아이템 생성
        GameObject* item = newObject();
        if (item) {
            item->type = ITEM_POWER; //파워 아이템 타입 설정
            item->bitmap = createBitmap("power.png");
            item->x = player_obj->x;
            item->y = player_obj->y + player_obj->height / 2.0f; //아이템 생성 위치 설정
            item->vx = randomFloat(0, 1) + ITEM_POWER_SPEED;
            item->vy = randomFloat(0, 1) + ITEM_POWER_SPEED; //아이템 이동 속도 설정
            item->width = 40.0f;
            item->height = 40.0f; //아이템 객체 크기 설정
            item->a = 75.0f; //아이템 객체 각도 설정
            item->age = 500;
        }
    }
}
```

그리고 그 후에는 죽기 직전의 power-1만큼의 개수로 power아이템을 생성하도록 했습니다.

```
if ((game_mode == IN_GAME) || (game_mode == IN_BOSS)) {
    createDieParticles2();
    for (j = 0; j < POOL_SIZE; j++) {
        GameObject* enemybullet = getObject(j);
        if (enemybullet->type == ENEMY_BULLET)
            deleteObject(enemybullet);
    }
    diemotion_createItemPower();
    if (game_mode == IN_GAME)
        changeGameMode(READY_GAME);
    else
        changeGameMode(READY_BOSS);
}
```

왼쪽의 코드는 checkCollision()함수 인데
충돌을 감지하여 만약 체력이 0이하로 떨어지면
화면에 파티클을 생성시키며
파워 아이템을 생성 시킵니다.



-보스모드로 진입시

```
void movePlayer_BeforeBoss(int time) { //보스를 만났을 때(100초 경과했을 때) 플레이어가 생성 위치로 돌아가도록 하는 함수

    if (player_obj) {
        //생성 위치와 현재 플레이어의 사이 거리 구하기
        float width = WIN_WIDTH / 2.0f - player_obj->x; //플레이어생성 위치의 x좌표 - 현재 플레이어 위치의 x좌표
        float height = 100 - player_obj->y; //플레이어생성 위치의 y좌표 - 현재 플레이어 위치의 y좌표
        if (!(width == 0) || !(height == 0)) {
            player_obj->vx = width / time;
            player_obj->vy = height / time;
            player_obj->x += player_obj->vx;
            player_obj->y += player_obj->vy;

            player_back_obj->vx = width / time;
            player_back_obj->vy = height / time; //플레이어가 생성 위치로 돌아갈 속도 설정
            player_back_obj->x += player_back_obj->vx;
            player_back_obj->y += player_back_obj->vy; //플레이어 객체의 위치 설정
        }
        //거리 = 시간 * 속도,-> 속도 = 거리/시간
    }
}
```

게임을 하다가 100초가 경과되면 보스모드로 진입하게 됩니다.
보스모드로 진입 시에 플레이어 객체의 위치를 다시 플레이어 객체 생성 위치로 돌아가도록 하는 함수입니다.