Battle Ship
(2인용 슈팅게임)

과목명	윈도우즈 프로그래밍
담당교수	김동근 교수님
학과	컴퓨터공학부 컴퓨터공학전공
학번	201401932
이름	노 명 환

목 차

Ⅰ. 서론

처음 프로젝트에 대하여 생각을 하였을 때, 가장 먼저 떠오른 것은 어린 시절 인터넷에서 플레이하던 간단한 Flash game이 생각이 났습니다. 이번 학기에 윈도우즈 프로그래밍을 배우면서 스스로 Flash game을 비슷하게 구현할 수 있을 것 같다는 확신을 가지게 되었고, 이번 프로젝트를 통하여 직접 구현해보고자 마음을 먹게 되었습니다. 여러 가지 게임이 떠오르고 구상을 하였고, 고민 끝에 포물선 운동에 중점을 두고 게임을 설계하였습니다.

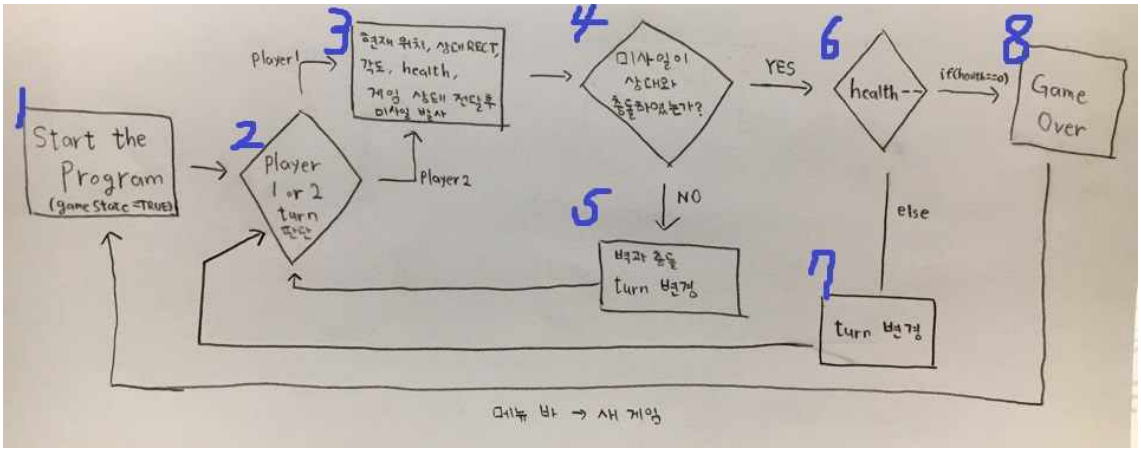
Battle Ship은 2인용 플레이에 적합하게 설계 되어있는 슈팅 게임입니다. 각 Player는 시작과 동시에 화면에 자신의 배를 선택하여 게임을 플레이하게 됩니다. 각 배는 같은 health를 가지고 있으며, 미사일을 발사하여 명중하게 된다면 health가 1씩 감소합니다. 먼저 상대방의 health가 0에 도달하게 만드는 Player가 승리하게 되며 이를 위하여 미사일을 적중시키는 것이 중요합니다.

각 배는 구조체를 통하여 설계하였습니다. 구조체 내에는 배의 POINT 중심 좌표, RECT 배의 영역 좌표, 미사일을 발사할 때 사용하는 degree 변수, 자신의 turn을 체크하여주는 bool turn 변수, 마지막으로 체력을 나타내는 health 변수를 가지고 있습니다.

프로그램 내에서는 최초 WM_CREATE 내에서 각 배의 기본 좌표, 각도, health, turn 등 구조체 내의 변수들을 모두 초기화 시켜주며 게임 내에서는 WM_KEYDOWN을 통하여 구조체 내의 변수들을 변경할 수 있습니다. 각 플레이어는 화살표를 통하여, 자신의 배의 위치를 이동할 수 있습니다. 배의 위치는 map 밖으로 벗어날 수 없으며, 이는 전체 창의 좌표를 RECT map으로 저장하여 충돌판정을 통하여 위치를 제한하였습니다. 또한, 보다 전략적인 플레이를 위하여 매 turn마다 move 변수(10칸)만 움직일 수 있도록 움직임에 제한을 두었습니다. 위치를 지정한 이후, END(Degree up), HOME(Degree down)를 통하여 각도를 설정하고 SPACE BUTTON을 통하여 미사일을 발사할 수 있습니다. 미사일은 throwBall 함수를 통하여 구현하였는데, throwBall 함수에서는 자신의 배의 위치(미사일 발사 위치), 각도, 시간(미사일 위치 갱신), 상대 플레이어 정보, 맵 등을 받아 미사일을 비트맵으로 출력시킵니다. 미사일의 궤적은 포물선 운동을 기반으로 하여, 삼각함수를 이용하여 표현하였습니다. 만약 미사일이 map을 벗어나게 된다면, PAINT문이 종료되며, turn이 상대방에게 넘어가게 됩니다. 또한 미사일이 상대방에게 명중하였을 경우, 상대방의 health가 1이 감소하며 health가 0이 되었다면 Player의 승리가 되며, gameState가 FALSE가 되어 WM_KEYDOWN이 if문 검사를 통하여 호출을 하지 못하게 됩니다. 게임이 끝났을 때 새로운 게임을 시작하는 것은 메뉴 바에서 새 게임을 추가하여 구현하였습니다. 새 게임을 시작하였을 경우에는 WM_CREATE 문 내의 구조체 초기화를 다시하며 gameState를 TRUE로 변경하여, 새로운 게임을 시작하게 됩니다.

프로그램의 설계와 구현은 가장 처음에 포물선 운동에 관하여 정보를 찾아보고 이후에는 모두 직접 구현하였습니다. 막히는 부분이 있을 때는 소스 코드를 수정하며 여러 번의 테스트를 반복하였고, 필요한 함수는 교재를 참고하였습니다. 구상 단계에서 필요한 함수들을 설계하였고 객체지향적인 프로그래밍을 위하여 노력하였습니다.

Ⅱ. 본론



1. Start the Program (gameState = TRUE)

WM_CREATE에서 최초로 두 Player의 중심 좌표를 설정하여준다. 이 중심 좌표를 중심으로 미리 정의해놓은 size 크기만큼 RECT 구조체의 좌표를 설정하여준다. 또한 각 Player의 체력(health)을 최초 3으로 설정하여주며, turn 변수를 설정하여 각 플레이어의 턴을 구분하여주며 맵의 크기를 GetClientRect를 통하여 받아와 map 변수에 저장한다. 마지막으로, gameState를 TRUE로 설정하여 게임을 시작한다.

2. Player 1 or 2 turn 판단 + 움직임 및 각도 조절

bool 변수를 통하여 Player의 turn을 판단하고 WM_KEYDOWN을 통하여 입력받은 키를 분석하여 Player의 상태를 변경한다.

이 때, Player가 움직일 때는 move라는 변수만큼 매 turn마다 움직일 수 있다.

KEY	기 능
LEFT	Player의 중심 좌표를 step만큼 왼쪽으로 이동시켜준다. 중심 좌표가 map.left보다 작다면, 원래의 위치로 되돌린다.
RIGHT	Player의 중심 좌표를 step만큼 오른쪽으로 이동시켜준다. 중심 좌표가 map.right보다 크다면, 원래의 위치로 되돌린다.
UP	Player의 중심 좌표를 step만큼 위쪽으로 이동시켜준다. 중심 좌표가 map.top보다 작다면, 원래의 위치로 되돌린다.
DOWN	Player의 중심 좌표를 step만큼 왼쪽으로 이동시켜준다. 중심 좌표가 map.bottom보다 크다면, 원래의 위치로 되돌린다.
END	Player의 Degree를 15만큼 증가시킨다. 이 때, 최대 Degree는 80을 넘을 수 없다.
HOME	Player의 Degree를 15만큼 감소시킨다. 이 때, 최소 Degree는 30보다 작을 수 없다.
SPACE	nowDraw를 True로 만들어주며, throwBall 함수를 호출한다.

3. 미사일 발사 (SPACE 버튼)

Space Button을 누를 시 nowDraw가 TRUE가 되며 throwBall 함수를 호출하게 된다.

int throwBall (HDC hdc, int x, int y, int degree, int time, RECT map, bool &nowDraw, RECT player, int &health, bool &state)	
hdc	그리기 정보 전달
int x, y	현재 turn에 해당되는 Player의 좌표를 전달하여 미사일을 발사할 때, 미사일의 출발 좌표로 사용
int degree	미사일의 궤적을 그릴 때 사용되는 각도
int time	Timer가 호출될 때마다 증가하는 변수이다. Timer를 설정하여 WM_Timer가 호출될 때마다, time을 증가시켜 미사일의 궤적을 time에 근거하여 설정하였다. time이 증가할 때마다 미사일이 날아가며 그에 따라 미사일을 Bitmap으로 보이도록 한다.
RECT map	전체 map의 좌표를 가지고 있는 변수이다. map의 좌표를 받아와 미사일이 map의 left, top, right, bottom 좌표에서 벗어나는지를 체크하여 벗어났다면 nowDraw의 상태를 FALSE로 변경시킨다.
bool &nowDraw	미사일을 그리는 여부를 설정하는 변수이다. 미사일이 map 밖으로 벗어나거나 상대 player와 충돌할 시, nowDraw를 FALSE로 변경하여 WM_PAINT문을 종료시킨다.
RECT player	상대 Player의 좌표를 가지고 있는 변수이다. 상대 Player의 좌표를 받아와 미사일이 상대 Player와 충돌하는지 검사할 때 사용한다.
int &health	상대 Player의 health 변수. 상대 Player와의 충돌이 일어날 시 상대 Player의 health를 감소시킨다.
bool &state	만약 상대 Player의 health가 0이 된다면, 게임의 상태를 변경

throwBall 함수 안에서는 최초 Player의 좌표를 시발점으로 하여, 미사일을 발사시킨다.

최초 미사일의 위치 좌표는 Player의 좌표로 입력받게 되며 dx, dy만큼 계속하여 위치가 변화하게 된다. dx와 dy는 Timer에서 time 변수를 계속 증가시키며, nowDraw가 FALSE가 되기 전까지 계속 증가하도록 구현하였다.

dx와 dy는 포물선 운동을 x와 y의 운동으로 각각 분해하여 구현하였다. 이 때, math.h 헤더 파일을 include하여 sin, cos 값을 계산하도록 하였으며 추가로 radian으로 변환시키는 함수를 추가하여 degree에 따라 radian으로 변환하여 좌표를 계산하도록 하였다.

미사일의 궤적은 ballx와 bally의 좌표를 기점으로 하여 ballsize를 define하여 DrawBitmap 함수를 통하여 미사일의 그림이 나타나도록 하였다.

4. 미사일이 상대와 충돌하였는가?

ⅰ) YES

미사일의 좌표가 RECT Player의 좌표 사이에 포함되어 있다면, 상대 객체의 좌표와 미사일의 좌표가 겹쳐있는 것이므로, 충돌한 것으로 판정한다.

ⅱ) NO

미사일의 좌표가 상대방의 좌표에 겹쳐있지 않고 계속 날아간다면, RECT map의 좌표와 만날 것이다. 이 때, 미사일의 좌표가 map의 좌표보다 크거나 작아진다면 map밖으로 벗어난 것이다.

5. 벽과 충돌, turn 변경

미사일의 좌표가 map의 좌표와 만난다면 벽과 충돌했을 상황이다. 이 때, nowDraw를 FALSE로 변경하여 PAINT문을 빠져나가게 만든다. 또한 turn을 변경하여 상대방의 차례로 변경한다.

6. health--

미사일이 상대 Player와 충돌하였다면 미사일이 명중하였으므로 상대방의 health를 하나 감소시킨다. 그 후, 상대방의 health가 0인지 아닌지를 검사하여 게임을 계속 이어나갈지 아니면 게임을 종료시킬지 State를 설정한다.

7. turn 변경

health가 0보다 크다면 게임이 아직 끝나지 않은 상황이다. 이 때, nowDraw를 FALSE로 변경하여 PAINT문을 탈출하고, turn을 변경하여 상대방이 미사일을 쏘도록 한다.

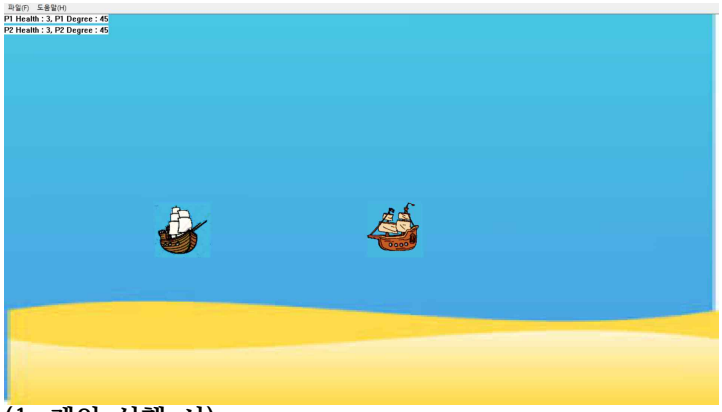
8. Game Over

health가 0이 되었다면 gameState가 0으로 변경이 된다.

gameState가 0인 상태에서는 winProc 내에서 WM_KEYDOWN이 작동하지 않으며, 이로 인하여 PAINT문 안으로 진입이 불가능하다. 화면이 멈춘 상태에서 아무것도 할 수 없는 상태가 된다.

새로운 게임을 시작하고 싶은 경우, 메뉴 바에 새 게임(N)을 선택하여 새로운 게임을 시작할 수 있다. 새로운 게임에서는 WM_CREATE에 있는 초기 설정들이 모두 들어가 있으며 gameState를 TRUE로 변경하여 처음부터 다시 시작할 수 있다.

Ⅲ. 실험결과

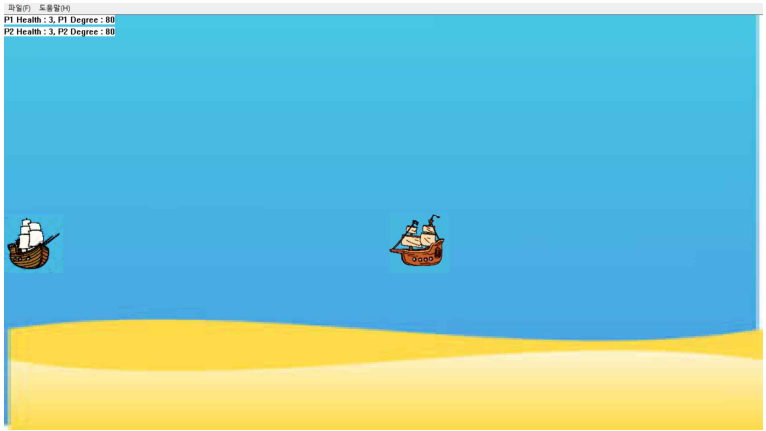


(1. 게임 실행 시)

최초 게임 실행 시 WM_CREATE에서 정의한 좌표로 ship1, ship2의 좌표가 설정되며 배경 및 배의 비트맵을 불러 와 화면에 표시된다.

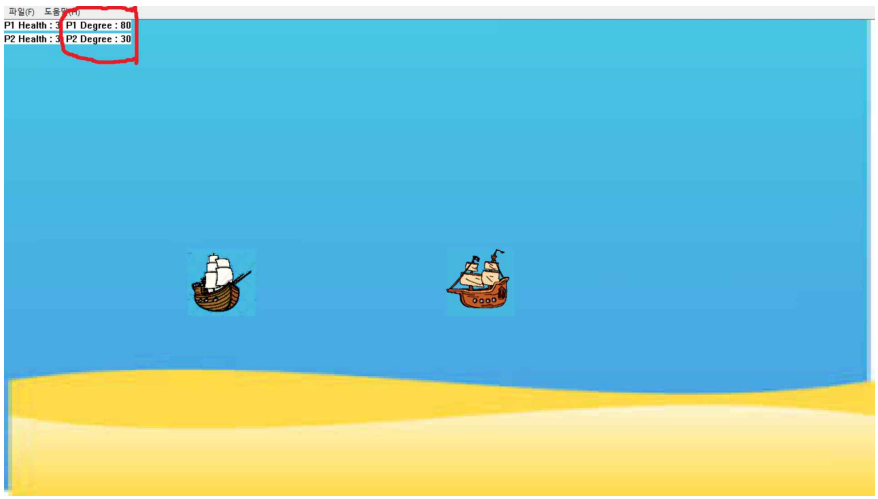
(2. 위치이동 시)

위치이동은 한 번의 turn에 10번까지 가능하며 이는 move 변수가 움직일 때마다 1씩 감소한다. 위치이동이 더 이상 불가능 할 때, SPACE를 통하여 미사일을 발사하는 것만이 가능하다.



(2-1. 배의 좌표가 끝까지 갔을 때)

그림과 같이 player.x가 map.x보다 작아질 경우, 배의 위치(player.x)를 자동으로 map.x로 설정시켜 화면 밖으로 벗어나지 않도록 설정한다.

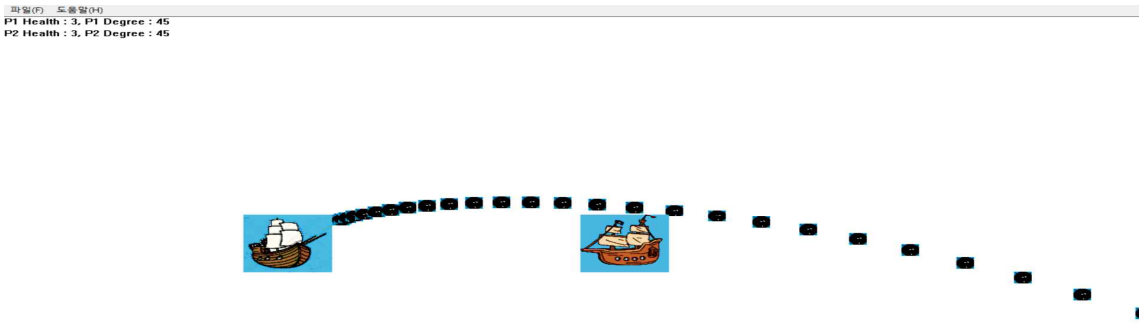


(3. 각도 변경 시)

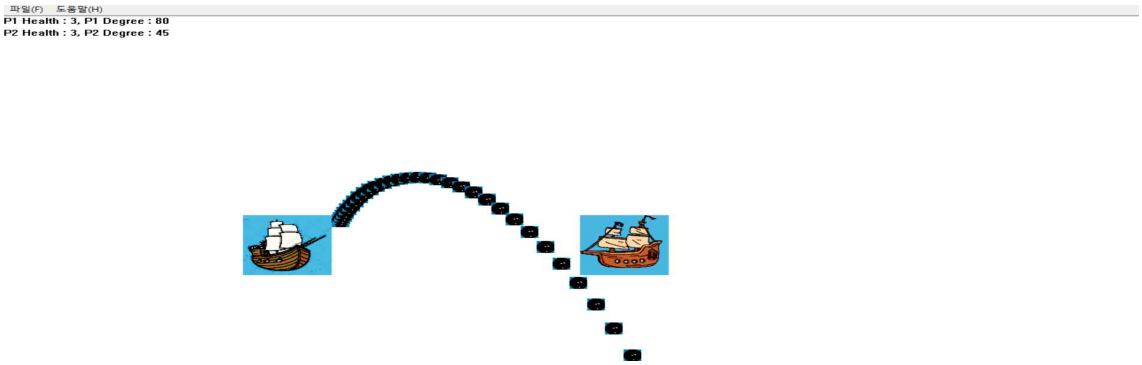
Player1 : END BUTTON을 통하여 각도를 증가

Player2 : HOME BUTTON을 통하여 각도를 감소

변화 된 각도는 화면에 출력시켜 인지할 수 있도록 한다.



(3-1. 미사일 궤적 45도)

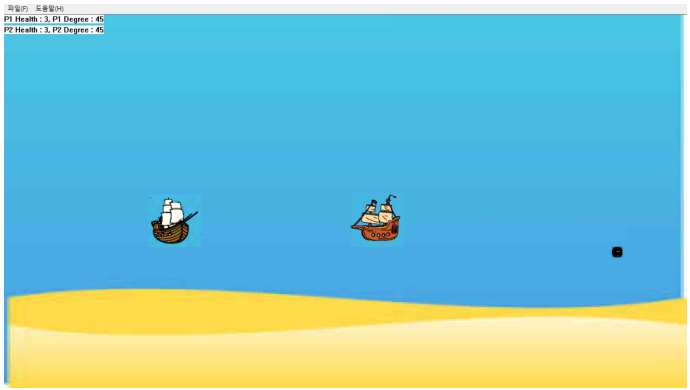


(3-2. 미사일 궤적 80도)

위의 두 그림은 각각 Degree가 45, 80 일 때의 미사일의 궤적을 나타낸 것이다.

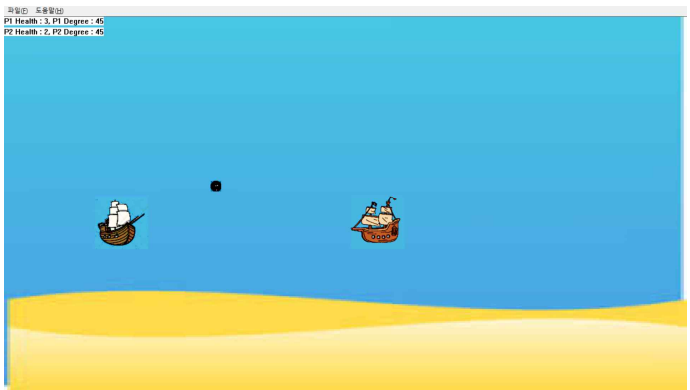
미사일의 궤적은 포물선 운동을 기반으로 하며, throwBall 함수 안에 삼각함수를 직접 구현하여 degree를 넘겨받아 궤적을 표현하였다.

그림에서 미사일의 간격이 점점 길어지는 것을 확인할 수 있는데, 이는 time이 점점 증가함으로써 이동거리가 점점 늘어나는 것을 표현한 것이다. 이는 미사일이 날아가며 가속도가 붙는 것으로 생각을 하여 구현하였다.



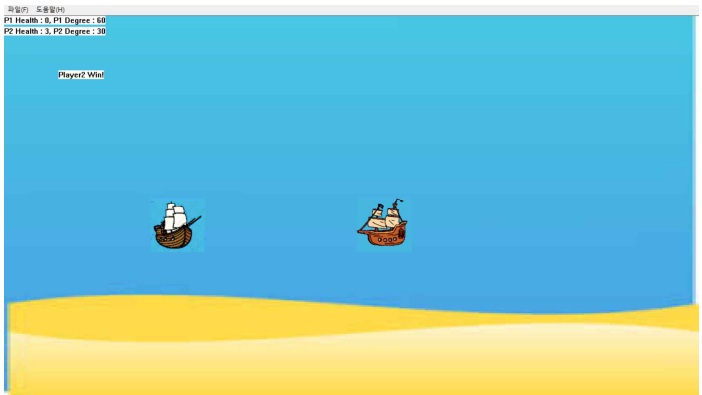
(4. 벽과 충돌 시)

위 그림은 미사일이 날아가 벽과 충돌하였을 시 ballx, bally가 map.right 혹은 map.bottom과 만났을 경우이다. 이를 if문을 통하여 검사하여 미사일이 벽과 만났을 때는 nowDraw가 FALSE가 되며, turn이 상대방에게 넘어가게 된다.



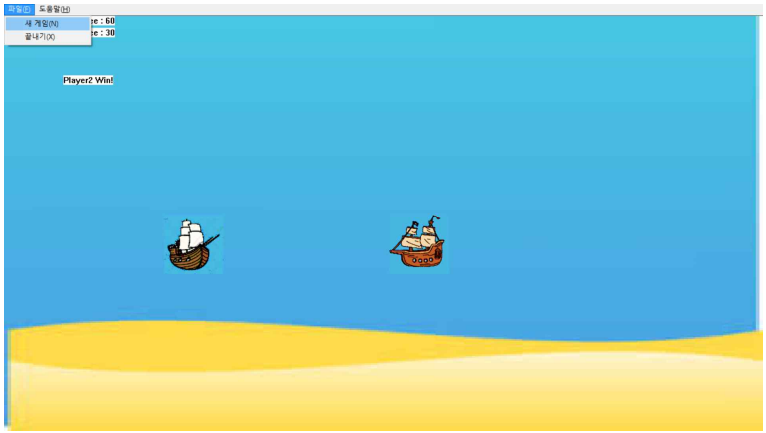
(5. 충돌 후 health 변화)

미사일이 상대 Player와 충돌하게 되면, 상대의 health에서 1을 감소시킨다.



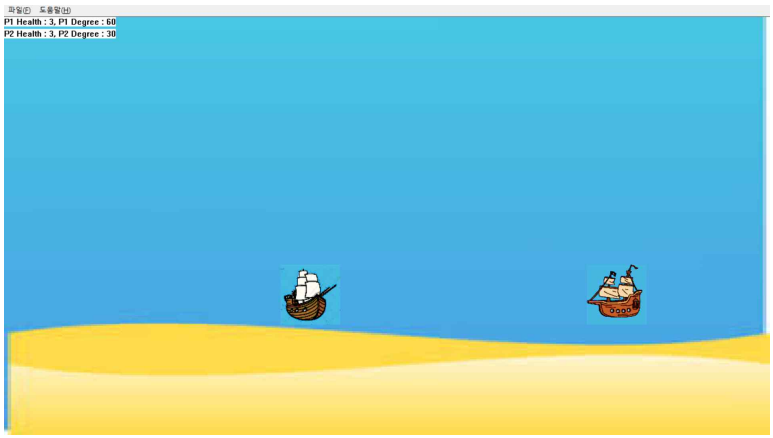
(6. 게임 끝)

상대방의 health가 0이 된다면 gameState=0이 되고, 더 이상 WM_KEYDOWN이 작동하지 않는다. 이로 인하여, 어떠한 키도 사용할 수 없는 상태가 된다.



(7. 새 게임 선택)

gameState=0이 되어 더 이상 게임이 진행하지 않을 때, 메뉴에서 새 게임(N)을 선택하면 새로운 게임을 진행할 수 있다.



(8. 새 게임 시작)

새 게임이 시작되면 WM_CREATE문 안의 좌표 설정을 다시 하며 gameState를 TRUE로 설정하여, 새로운 게임을 진행할 수 있도록 한다.

Ⅳ. 결론

처음 프로그램을 구상할 때, 게임을 만드는 일이 간단하고 쉬운 일이라고 생각하였습니다. 하지만, 첫 날 그래프를 그리면서 느낀 점은 아무것도 없는 백지에서 시작하여 프로그램을 하나씩 완성해 나가는 과정은 설계가 탄탄해야 한다는 점이었습니다. 프로그램 전반에서 사용하는 함수들을 미리 설계하지 않고 필요할 때마다 만들어서 사용하다보니 프로그램이 상당히 복잡하게 되었고 결국 프로그래밍을 끝낸 지금 소스 코드를 보면 500줄이 넘고 가독성도 많이 떨어지는 프로그램이 되었습니다. 다음부터 프로젝트를 하게 된다면 함수 설계부터 시작하여 조금 더 객체지향적인 프로그래밍을 하면 좋겠다는 생각이 들었습니다.

프로그램을 짜면서 가장 힘들었던 때는 가장 처음이었던 것 같습니다. 가장 처음에 포물선 운동을 구현하기 위하여 삼각함수를 만들고 궤적을 그렸으나 생각했던 것과는 달리 이상한 궤적을 나타내며 날아가는 함수들을 보면서 생각대로 잘 되지 않는다는 것을 느꼈습니다. 특히, y좌표를 윈도우즈 프로그래밍에서는 -로 주어야 제대로 그려진다는 것을 1시간동안 열심히 고민하여 깨달았을 때, 기초가 탄탄해야 한다는 것을 다시 한 번 느끼게 되었습니다.

이번에 윈도우즈 프로그래밍을 하면서 가장 큰 성과는 역시 간단한 콘솔 프로그램에서 벗어나서 윈도우즈라는 틀에서 스스로 프로그램을 설계한 점이라고 생각합니다. 스스로 생각하고 설계한 것이 실제로 표현되고 출력될 때마다 결과의 산물이 나타나 프로그래밍에 흥미를 크게 돋아주었습니다. 좌표계라는 곳에서 여러 가지 객체를 출력 및 충돌 알고리즘 등을 구현, 그리고 포물선 궤적 설계를 위해 삼각함수를 사용하는 알고리즘을 설계하면서 수학 공부에 대한 필요성을 다시 한 번 느끼기도 하였습니다.

수업시간에 배우는 내용은 단순히 소스 코드를 분석하고 이해하는 과정이라면 프로젝트를 통하여 스스로 프로그램을 설계하고 구현하는 것은 커다란 의미를 가지고 있다고 생각합니다. 앞으로도 다양한 프로젝트를 통하여 직접 문제를 해결하는 과정을 많이 거치면 좋을 것 같습니다. 스스로 하는 공부가 진짜 공부라는 다시 한 번 느끼게 해주는 프로젝트였습니다.

Ⅴ. 참고문헌

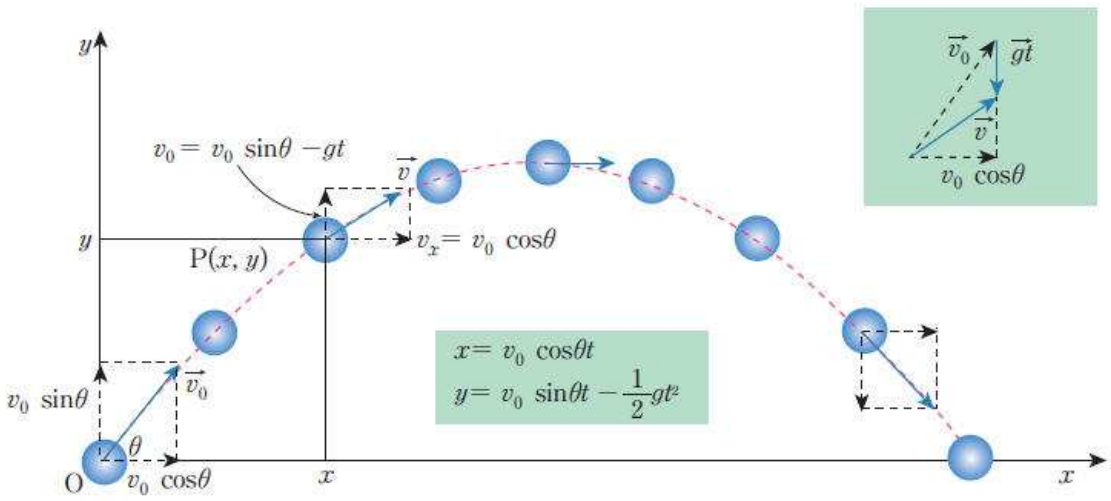


*Image From Google

Ⅵ. 부록

미사일 발사 함수

```
int throwBall(HDC hdc, int x, int y, int degree, int time, RECT map, bool &nowDraw, RECT player, int &health, bool &state)
{
	state = FALSE;
	static int ballx = x;
	static int bally = y;
	double dx, dy;

	if (time < 2)
	{
		ballx = x;
		bally = y;
	}
	if (x < player.left)
		dx = 4 * cos(radian(degree))*time;
	else
		dx = 4 * cos(radian(degree))*time*-1;
	dy = (80 * (sin(radian(degree))*time) - ((gravity*time*time) / 2)) / 50;
	//MoveToEx(hdc, ballx, bally, NULL);
	//LineTo(hdc, ballx + dx, bally - dy);
	DrawBitmap(hdc, ballx, bally, ballsize, ballsize, LoadBitmap(hInst, MAKEINTRESOURCE(IDB_Bomb)));
	ballx = ballx + dx;
	bally = bally - dy;

	//맵 밖으로 나갔을 때
	if (ballx > map.right)
	{
		nowDraw = FALSE;
		state = TRUE;
	}
	else if (bally > map.bottom)
	{
		nowDraw = FALSE;
		state = TRUE;
	}
	else if (ballx < map.left)
	{
		nowDraw = FALSE;
		state = TRUE;
	}
	else if (bally < map.top)
	{
		nowDraw = FALSE;
		state = TRUE;
	}

	//다른 객체와의 충돌
```

```
	if (player.left <= ballx + ballsize / 2 && ballx + ballsize / 2 <= player.right
		&& player.top <= bally + ballsize / 2 && bally + ballsize / 2 <= player.bottom)
	{
		health--;
		nowDraw = FALSE;
		state = TRUE;
	}
	return health;
}
```

배 구조체 정의

```
struct Tank
{
	RECT player;
	POINT center;
	int degree = 45;
	bool turn;
	int health;
};
```

위로 움직이는 경우

```
case VK_UP:
	move--;
	if (move > 0)
	{
		InvalidateRect(hWnd, NULL, TRUE);
		if (t1.turn)
			{
				t1.center.y -= step;
				if (t1.center.y <= map.top)
					t1.center.y = map.top + size;
				t1.player.top = t1.center.y - size;
				t1.player.bottom = t1.center.y + size;
				InvalidateRect(hWnd, &t1.player, TRUE);
			}
			if (t2.turn)
			{
				t2.center.y -= step;
				if (t2.center.y <= map.top)
					t2.center.y = map.top + size;
				t2.player.top = t2.center.y - size;
				t2.player.bottom = t2.center.y + size;
				InvalidateRect(hWnd, &t2.player, TRUE);
			}
			break;
```

스페이스

```
case VK_SPACE:
		if (nowDraw == FALSE)
			nowDraw = TRUE;
		time = 0;
		t1.turn = !t1.turn;
		t2.turn = !t2.turn;
		break;
```

페인트문

```
case WM_PAINT:
{
		hdc = BeginPaint(hWnd, &ps);
		DrawBitmap(hdc, 0, 0, map.right, map.bottom, LoadBitmap(hInst,
MAKEINTRESOURCE(IDB_Background)));
		DrawBitmap(hdc, t1.player.left, t1.player.top, size * 2, size * 2, LoadBitmap(hInst,
MAKEINTRESOURCE(IDB_Ship1)));
		DrawBitmap(hdc, t2.player.left, t2.player.top, size * 2, size * 2, LoadBitmap(hInst,
MAKEINTRESOURCE(IDB_Ship2)));
		if (nowDraw)
		{
			if (!t1.turn)		//t1의 미사일
				t2.health = throwBall(hdc, t1.player.right, t1.player.top, t1.degree, time,
map, nowDraw, t2.player, t2.health, gameState);
			if (!t2.turn)		//t2의 미사일
				t1.health = throwBall(hdc, t2.player.left, t2.player.top, t2.degree, time, map,
nowDraw, t1.player, t1.health, gameState);
			move = 11;
		}
		wsprintf(str, TEXT("P1 Health : %d, P1 Degree : %d"), t1.health, t1.degree);
		wsprintf(str2, TEXT("P2 Health : %d, P2 Degree : %d"), t2.health, t2.degree);
		TextOut(hdc, 0, 0, str, lstrlen(str));
		TextOut(hdc, 0, 20, str2, lstrlen(str2));
		if (t1.health == 0)
		{
			wsprintf(str3, TEXT("Player2 Win!"));
			TextOut(hdc, 100, 100, str3, lstrlen(str3));
			gameState = FALSE;
		}
		else if (t2.health == 0)
		{
			wsprintf(str3, TEXT("Player1 Win!"));
			TextOut(hdc, 100, 100, str3, lstrlen(str3));
			gameState = FALSE;
		}
		EndPaint(hWnd, &ps);
}
```