

2023학년도 프로그래밍방법론및실습 프로젝트 최종보고서

[뉴진스 마리오 (Angry Mario (feat. New Jeans))]

제출일자 : 2023. 12. 22

팀원 1:

팀원 2:

목 차

1. 프로젝트 개요	3
가. 프로젝트 소개	3
나. 개발 범위 및 내용	3
다. 개발 목표	3
2. 세부 개발 내용	3
가. 개발 과정 및 추진 체계	3
나. 빌드 환경 및 플랫폼	3
다. 소스코드 구조	4
라. 주요 알고리즘 및 구현 소개	4
3. 동작 설명	5
가. 프로그램 동작 화면	5
나. 개발 목표 달성도	5
4. 결론	5
5. 참고자료	5

1. 프로젝트 개요

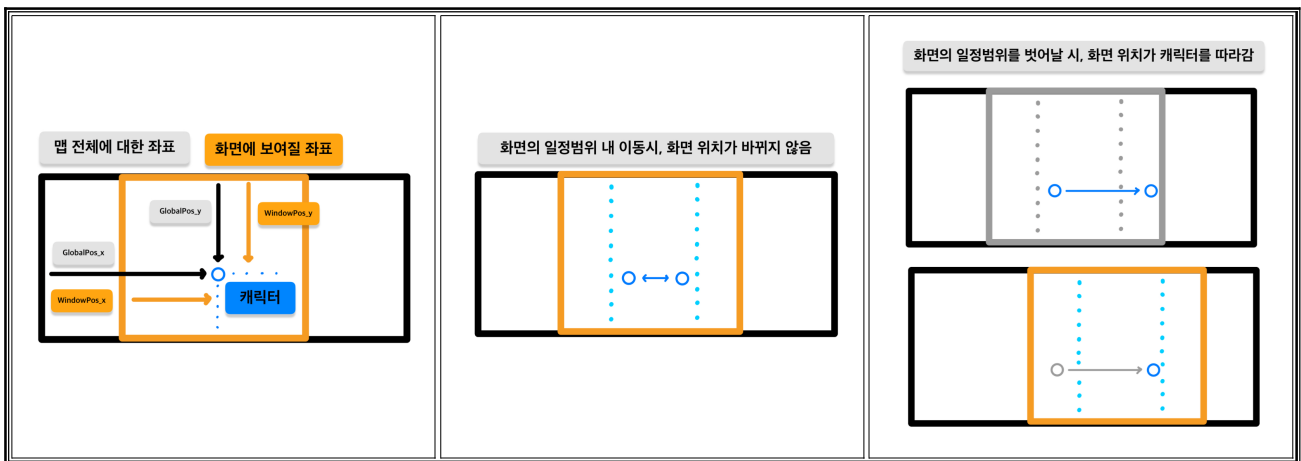
가. 프로젝트 소개 및 배경 지식 설명

- SDL2 라이브러리를 이용해 슈퍼 마리오와 유사한 플랫폼 게임을 구현하였다. 게임의 목표는 가시, 지뢰 등의 장애물을 피해 맵 끝의 아이템을 먹은 후, 다시 처음 위치로 돌아오는 것이다.
- 사용자는 키보드의 화살표를 통해 캐릭터를 움직일 수 있으며, 캐릭터의 점프, 낙하를 이용해 적을 밟거나 화면 내 블록을 부술 수 있다.
- 만약 캐릭터가 적에게 부딪히면 캐릭터는 사망하며, 이에 대한 메시지를 보여준다. 캐릭터가 절벽에서 떨어졌을 경우에도 마찬가지이다.
- 만약 캐릭터가 적 머리를 밟게 되면 캐릭터는 윗 방향으로 조금 튕기게 되며, 적은 사망해 더 이상 화면에서 보이지 않게 된다.

나. 개발 범위 및 내용

○ (1) 캐릭터 구현과 캐릭터와 관련된 물리법칙

- 사용자의 입력으로 이동, 점프 등이 가능한 캐릭터를 구현한다. 캐릭터는 우리가 화면에 보여질 좌표와 맵 전체에 대한 좌표를 가지며, 우리의 입력 (키보드) 에 따라 값이 변화해야 한다.

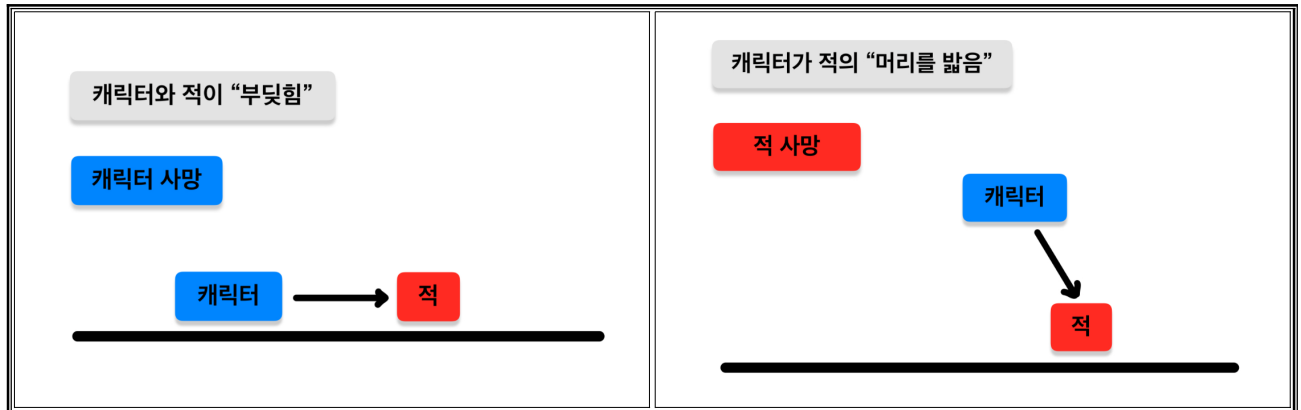


- 또한 캐릭터는 상시 중력을 받아 (블록에 가로막히거나 점프 도중이 아니라면) 아래 방향으로 떨어져야 한다. 만약 캐릭터가 (블록이 비어있어) 화면 아래로 떨어진 경우, 캐릭터는 즉사하고 이를 알려준다.

○ (2) 적 구현과 적과 관련된 법칙

○ 게임에 플레이어를 죽일 수 있는 다수의 적을 구현한다. 적은 특정 구간을 좌우로 순찰하고 있으며, 만약 캐릭터가 적과 만나게 되면 캐릭터 또는 적이 사망한다.

만약 캐릭터와 적이 부딪히면 캐릭터가 사망하고, 캐릭터가 적의 머리를 밟은 경우 적이 사망한다. 적이 사망한 경우, 캐릭터는 적을 밟았으므로 위로 조금 튕기게 되고, 적은 화면에서 더 이상 보이지 않게 된다.



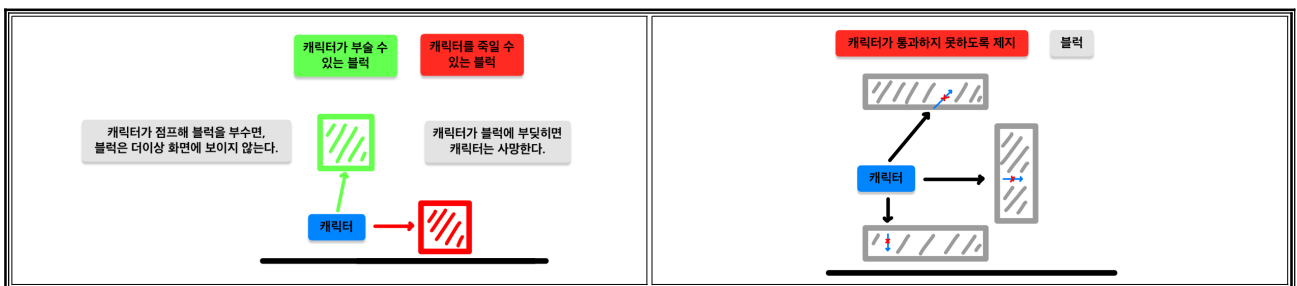
○ (3) 맵 구현 및 블럭 구현

○ 캐릭터가 돌아다닐 맵을 구현 또는 소스코드 내 이식해 둔다. 맵 정보에는 "어느 위치에 어느 블럭이 위치한다", "맵의 전체 크기는 얼마이다" 등의 정보가 포함되 있어야 한다. 또한 맵을 구성하는 블럭은 다수의 종류가 존재하며, "캐릭터가 부술 수 있는 블럭", "캐릭터를 죽일 수 있는 블럭" 등을 포함한다.

○ (4) 캐릭터와 지형지물간 상호작용

○ 캐릭터와 지형지물간 상호작용을 구현한다. 앞서 말한 캐릭터가 "부술 수 있는 블럭" 을 부순 경우, 해당 블럭은 화면에서 더 이상 보이지 않아야 하며, "캐릭터를 죽일 수 있는 블럭" 을 만난 경우, 캐릭터가 사망해야 한다.

또한 캐릭터가 통상적인 블럭을 통과하면 안되므로, 캐릭터가 그러한 블럭에 부딪히거나 통과하려 할 경우 이를 제지한다.





다. 개발 목표

평가 항목	개발 목표	우선 순위
소스코드의 모듈화	소스코드의 역할과 목적에 따라 모듈화를 진행하여 가독성과 개발 효율성을 높인다.	1
기능구현 정확성	구현한 모든 기능들이 정상적으로 작동하도록 한다. 여기서 말하는 기능은 앞서 보인 개발 범위 및 내용을 포함한다.	1
유지보수 용이성	추후 프로그램에 버그가 일어났을 때, 이를 추적하여 어느 부분이 문제인지 확인하기 쉽도록 한다.	2
소스코드 확장성 및 이식성	추후 프로그램에 기능을 추가하기 위해 필요한 refacotring 을 최소화 할 수 있도록 한다. 또한 여러 SDL 버전에서도 실행 가능하도록 한다.	3

2. 세부 개발 내용

가. 개발 과정 및 추진 체계

팀원	역할
	게임 아이디어 도출, SDL 레퍼런스 탐색, 프로그램 시나리오 구성, 그래픽 리소스 제작, 그래픽 이미지 좌표 확보, 블록-캐릭터 충돌 메커니즘 구현, 최종 보고서 일부 작성
	게임 아이디어 도출, SDL 레퍼런스 탐색, 프로그램 시나리오 구성, 기본 함수 정의, 캐릭터 움직임 구현, 게임 BGM 및 효과음 적용, 소스코드 병합 및 refactoring, 최종 보고서 일부 작성

세부 개발 내용	담당자	개발 기간								비고
		11월				12월				
		1주	2주	3주	4주	1주	2주	3주	4주	
게임 아이디어 도출	동반									
SDL 레퍼런스 탐색	동반									
그래픽 리소스 작업										
캐릭터 움직임 구현										
맵 구현										
적-캐릭터 충돌 메커니즘 구현										
블록-캐릭터 충돌 메커니즘 구현										

게임 BGM 및 효과음 적용										
소스코드 병합 및 refactoring										
최종 보고서 작성	동반									
발표 영상 제작	동반									

나. 빌드 환경 및 플랫폼

항목	사양
운영 체제	Ubuntu 20.04
개발 언어	C
컴파일러	gcc 9.4.0
사용 라이브러리	SDL2 2.0.10

필요 라이브러리 설치

```
$ sudo apt install -y gcc git libsdl2*-dev xorg-dev
```

```
$ git clone https://github.com/jbw9964/Programming_methodology_project.git
```

프로젝트 빌드

```
$ cd cd Programming_methodology_project/src/
```

```
$ gcc main.c -o main -ISDL2 -ISDL2_image -ISDL2_mixer
```

게임 실행

```
$ ./main
```

다. 소스코드 구조

폴더 / 파일			설명
asset	image		캐릭터, 블록 등의 이미지가 저장된 폴더
	sound	bgm	게임의 배경음악이 저장된 폴더
		object	게임의 캐릭터, 적, 블록 등에 관련된 음악이 저장된 폴더
		optional	캐릭터의 죽음 등 캐릭터 조건에 관련된 음악이 저장된 폴더
src	GameObject	Player.h	캐릭터와 관련된 소스코드의 헤더파일
		Player.c	캐릭터 (Player 구조체) 와 관련된 소스코드
		Enemy.h	적과 관련된 소스코드의 헤더파일
		Enemy.c	적 (Enemy_Object 구조체) 와 관련된 소스코드

	Init	Map.h	맵과 관련된 소스코드의 헤더파일
		Map.c	맵 (Map 구조체) 와 관련된 소스코드
		Init.h	Init.c 의 헤더파일
		Init.c	게임을 실행을 위한 초기화 함수가 포함된 소스코드
	Utils	Utils.h	Utils.c 의 헤더파일
		Utils.c	편의를 위한 함수가 포함된 소스코드
	def.h		소스코드의 모든 매크로 정의, 함수가 들어간 헤더파일
	main.h		main.h 의 헤더파일
	main.c		직접 게임이 구동되는 소스코드

라. 주요 알고리즘 및 구현 소개

- 전체 소스코드는 협업한 Github 에 기재되어 있으며 아래 링크를 통해 확인할 수 있다.
(https://github.com/jbw9964/Programming_methodology_project)
- 소스코드를 그대로 보며 설명하는 것 보다 작동 방식을 나타낸 다이어그램이 더 좋을 것 같아 최대한 그림으로 나타내어 설명하였다.
- 해당 보고서에서 설명할 내용은 크게 3 가지로, 첫째 캐릭터를 구현하고 키보드로 이를 움직이는 방식. 둘째 적 구현 및 적이 움직이는 방식. 셋째 캐릭터와 게임 내 물체간의 상호작용에 관한 부분이다.

1) 캐릭터 구현 및 키보드 입력을 통해 움직이기

가) 캐릭터 구현

- 게임 내 캐릭터는 여러 정보를 포함하고 있다. 아래 그림은 캐릭터를 구현한 구조체이다.

```

/** ...
typedef struct Player
{
    SDL_Texture *Tex;
    SDL_Rect Src_rect[NUM_OF_PLAYER_STATE];
    Player_State Current_state;
    Keyboard Key;

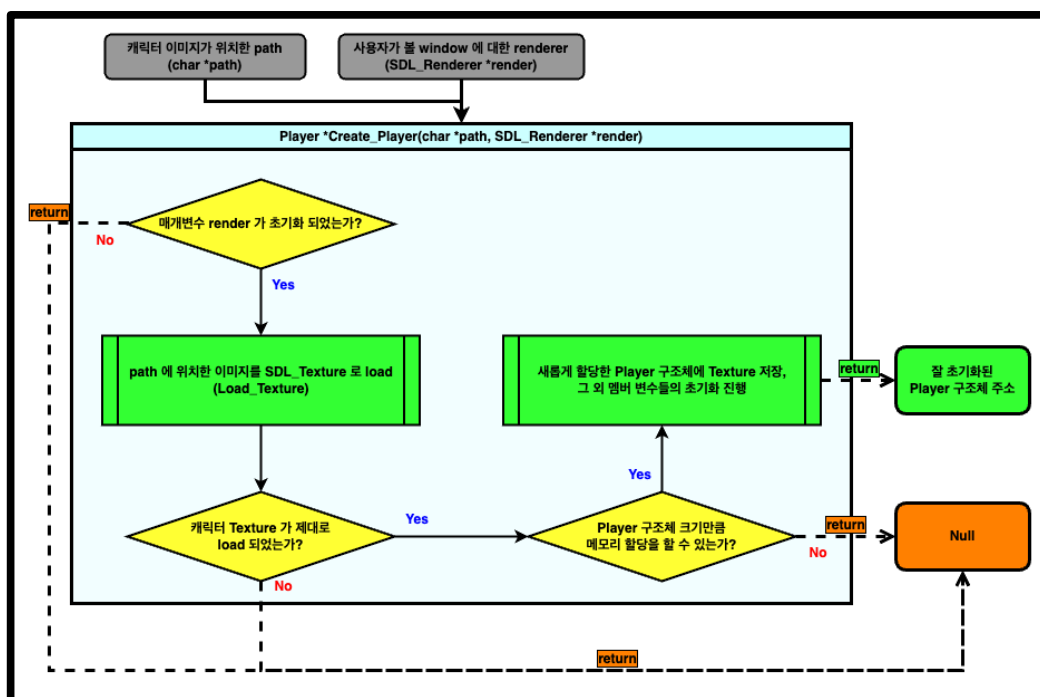
    float GlobalPos_x;
    float GlobalPos_y;
    float WindowPos_x;
    float WindowPos_y;
    float Speed_x;
    float Speed_y;
    bool is_dead;
    bool is_clear;
} Player;

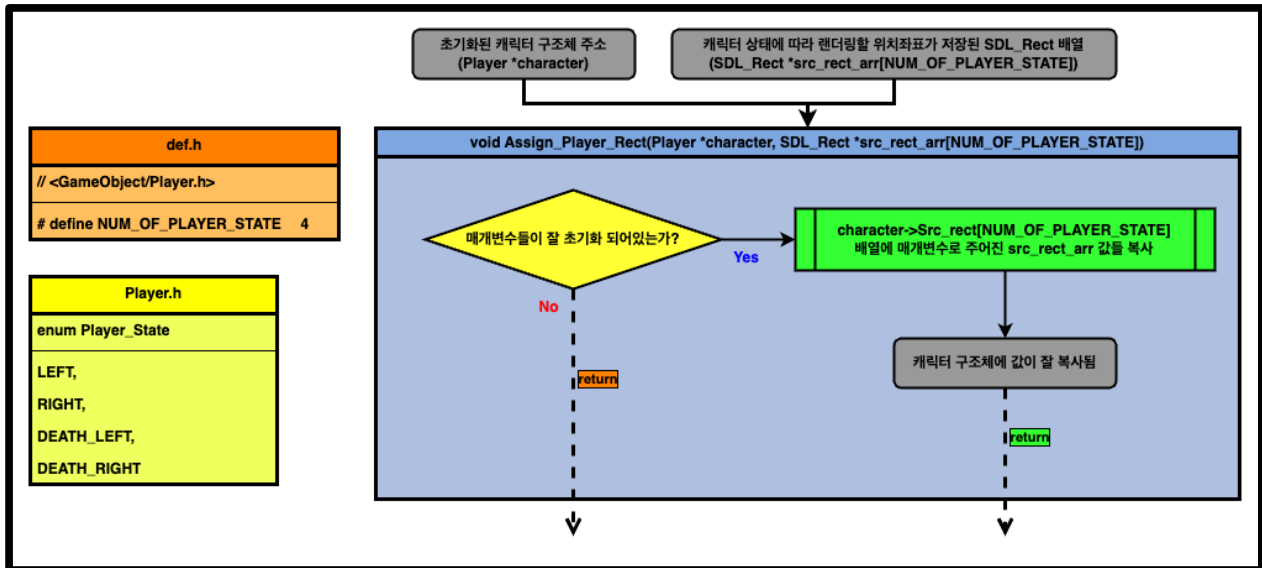
```

○ 캐릭터 구조체 멤버에 대한 설명은 다음과 같다.

식별자	타입	설명
Tex	SDL_Texture	캐릭터 전체 이미지를 불러와 저장시켜둔 SDL_Texture
Src_rect	SDL_Rect [NUM_OF_PLAYER_STATE]	캐릭터의 상태에 따라 불러온 Texture의 위치를 지칭하는 SDL_Rect 배열
Current_state	Player_State	현재 캐릭터상태를 나타내는 enum 변수
Key	Keyboard	사용자가 키보드에 입력한 상태를 저장하는 구조체
GlobalPos_*	float	전체 맵 내 캐릭터가 위치한 좌표
WindowPos_*	float	사용자가 보는 화면 내 캐릭터가 위치한 좌표
Speed_*	float	현재 캐릭터의 속도
is_dead	bool	현재 캐릭터가 사망했는지를 지칭하는 변수
is_clear	bool	캐릭터가 목표를 달성하고 해당 스테이지를 클리어 했는지 지칭하는 변수

○ 캐릭터 구조체를 사용하기 위해선 2 번의 초기화 과정이 필요하다. 첫째는 Create_Player 함수를 통해 Player 구조체를 초기화하는 부분, 둘째는 Assign_Player_Rect 함수를 통해 구조체의 Src_rect 멤버를 초기화 하는 부분이다. 두 함수는 모두 Player.h, Player.c 에 선언, 정의되어 있으며, 각 함수의 구동 다이어그램은 아래 그림과 같다.





나) 키보드 입력을 통해 캐릭터 움직이기

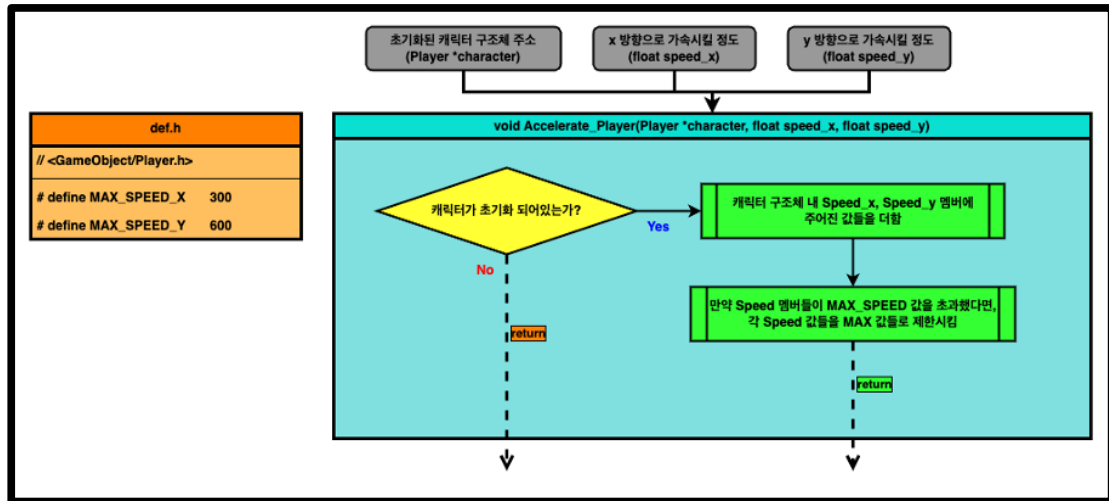
- 캐릭터의 움직임이 어떻게 이뤄지는지 알기 위해 먼저 Keyboard 구조체에 대해 알아야 한다. 다음 그림은 이를 나타낸 그림이다.

```

/**
 * @brief   A structure that record keyboard input
 * @param   `directions`   if user pushes keyboard, structure will store the status
 */
typedef struct Keyboard
{
    bool Up, Down, Left, Right, Restart;
} Keyboard;

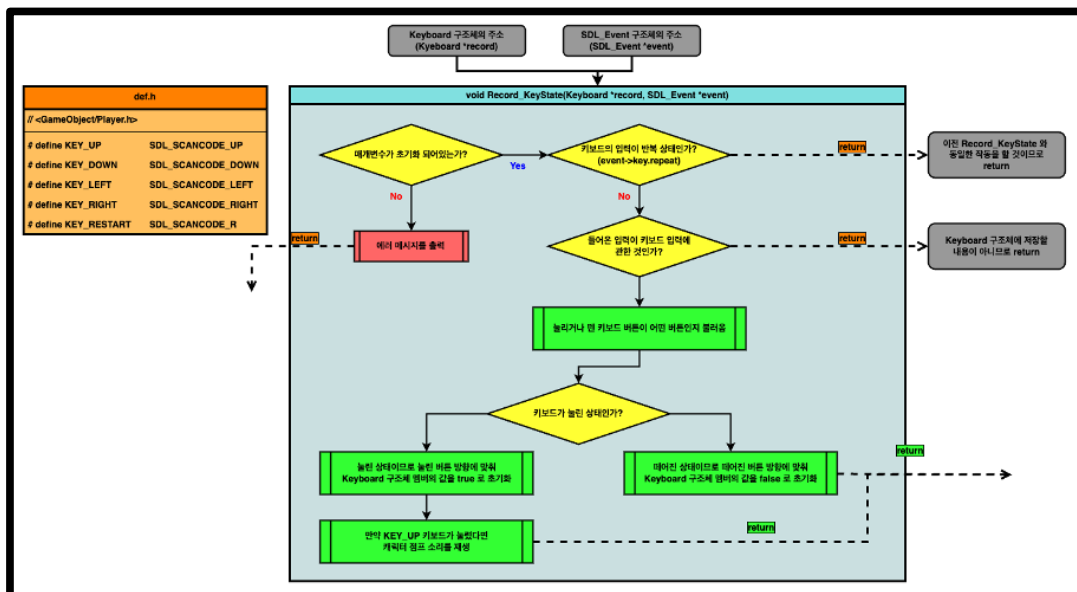
```

- Keyboard 구조체는 사용자가 키보드 입력을 주었을 때, 그 상태를 기억해 저장해놓기 위한 구조체이다. 구조체 내 Up, Down, Left, Right 멤버는 캐릭터의 이동 방향을 지칭하기 위한 변수이고, Restart 멤버는 캐릭터가 죽었을 때, R 키를 누른 상태를 기억하기 위한 변수이다.
- 앞서 캐릭터 구조체 내부에는 Keyboard 타입의 Key 멤버가 있었다. 이를 통해 캐릭터는 매 게임 loop 마다 키보드 입력을 감지하고, 그에 맞춰 Key 멤버를 업데이트하여 움직일 방향을 정하게 된다.
- 이에 사용되는 함수는 크게 4 가지인데, 키보드 입력 및 캐릭터의 속도를 조절하는 함수 3 가지, 설정된 캐릭터 속도에 따라 위치를 변화시키는 함수 1 가지이다.
- 먼저 키보드 입력 및 캐릭터 속도를 조절하는 함수들을 보자. 구동과정을 쉽게 이해하기 위해 캐릭터의 속도를 조절하는 함수를 먼저 설명하겠다.



○ Accelerate_Player 함수는 매개변수로 주어진 캐릭터를 주어진 속도만큼 가속시키는 함수이다. 이 때 만약 캐릭터가 설정된 최대 또는 최소 속도를 벗어나게 된다면, 캐릭터의 속도를 제한하는 특징을 가졌다.

○ 다음은 키보드 입력에 따라 Keyboard 구조체를 수정해주는 함수이다.



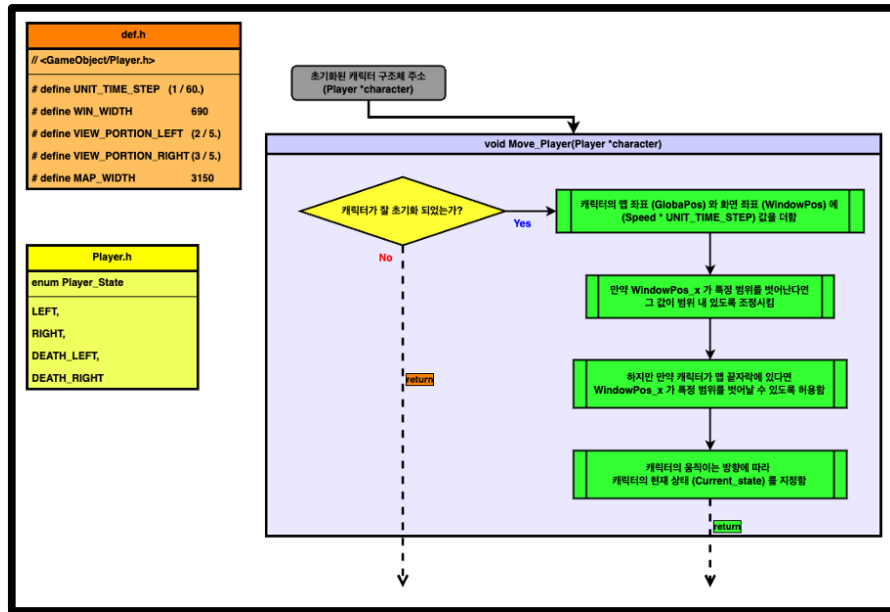
○ Record_KeyState 함수는 키보드 입력에 따라 구조체 멤버의 값을 초기화 시켜주는 함수이다.

○ 이제 앞서 두 함수 Accelerate_Player, Record_KeyState 를 이용해, 키보드 입력에 따라 캐릭터 속도를 변화시키는 함수를 보자.



○ Receive_Keyboard_input 함수는 복잡해 보이지만 핵심만 놓고 보면 그렇지 않다. 함수는 SDL_PollEvent 함수를 통해 사용자의 입력을 불러오고, 불러온 입력에 맞춰 Record_KeyState 함수를 호출해 캐릭터 멤버인 Key 를 업데이트한다. 그런 후 업데이트 된 Key 에 맞춰 캐릭터의 속도를 증감시키는 것이다.

○ 앞서 설명 함수들로 캐릭터의 속도가 조절되는 것을 확인하였다. 이제 그 속도를 통해 캐릭터가 직접 움직이는 함수를 보자.



- Move_Player 함수는 현재 캐릭터의 속도에 따라 캐릭터의 좌표를 업데이트 시키는 함수이다. Move_Player 함수의 특징은 캐릭터의 맵 전체 좌표 (GlobalPos) 뿐만 아니라, 사용자 화면에 보일 화면 좌표 (WindowPos) 또한 업데이트 시킨다.
- 위 함수를 통해 사용자 입장에서 캐릭터의 움직임이 더 부드러워지고, 랜더링되는 화면 위치 또한 제한시켜 화면의 끝자락 또한 볼 수 있게 해준다.

2) 적 구현 및 적의 움직임 구현

가) 적 구현

- 다음 그림은 게임 내 적을 구현한 구조체를 나타낸 그림이다.

```

/* ...
typedef struct Enemy_Object
{
    SDL_Texture *Tex;
    SDL_Rect Src_rect[NUM_OF_ENEMY_DIRECTION];
    Enemy_Direction Current_direction;

    float GlobalPos_x;
    float GlobalPos_y;
    float WindowPos_x;
    float WindowPos_y;

    float Moving_speed;
    bool is_dead;
} Enemy_Object;

```

- Enemy_Object 구조체와 Player 구조체를 비교해보면 많은 부분이 비슷한 것을 볼 수 있다. 때문에 적 구조체를 사용하기 위한 함수도 유사한데, 아래 그림은 이를 나타낸 그림이다.

```

/**
 * @brief Create `Enemy_Object` structure with given arguments.
 *
 * @param `path` path to load PNG image
 * @param `render` `SDL_Renderer` * to render image
 * @param `global_~` the global coordinate to spawn enemy
 *
 * @return `Enemy_Object` * : allocated memory
 */
Enemy_Object *Create_Enemy_Object(char *path, SDL_Renderer *render, float global_x, float global_y);

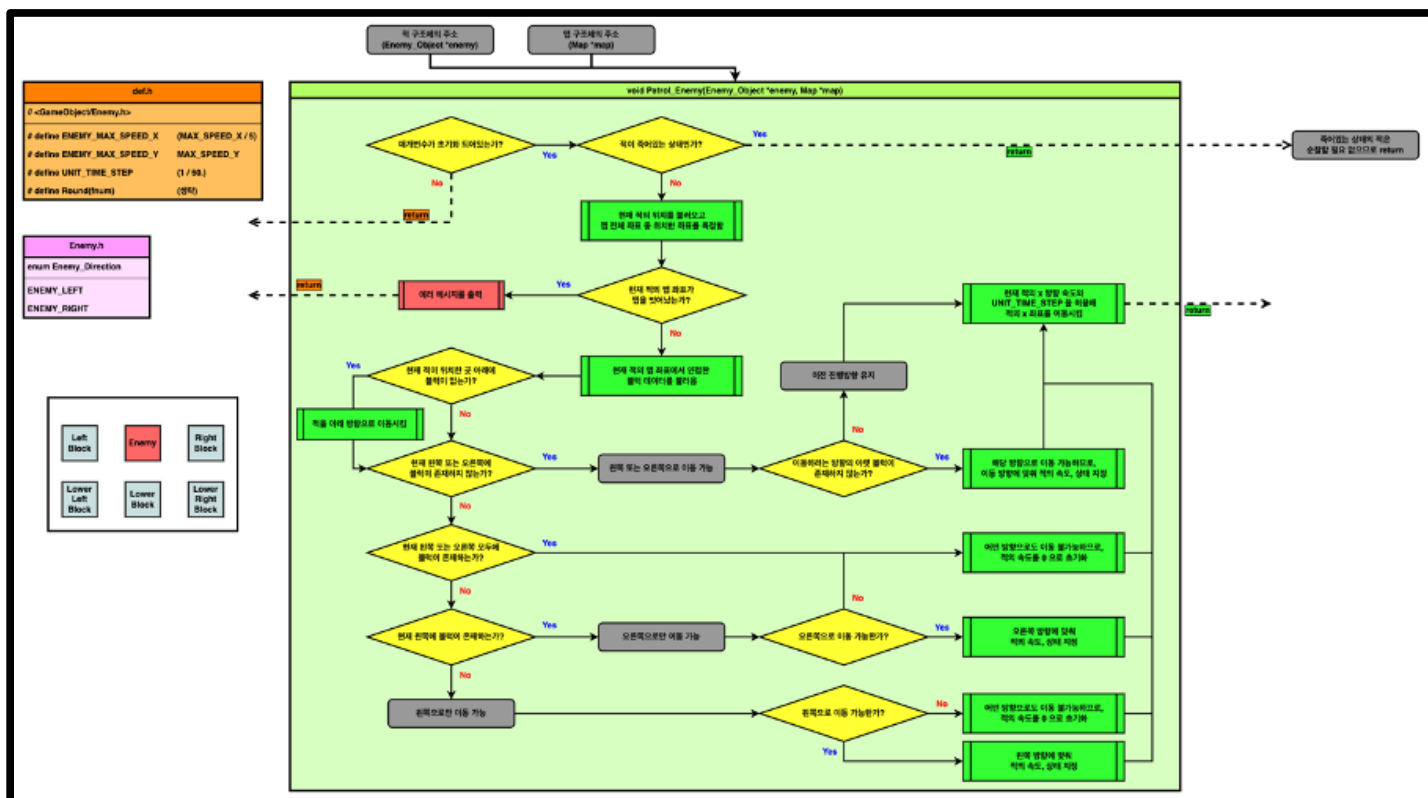
// @brief Assign `enemy->Src_rect[]` with each `Enemy_Direction`.
void Assign_Enemy_Rect(Enemy_Object *enemy, SDL_Rect *src_rect_arr[NUM_OF_ENEMY_DIRECTION]);

```

- 캐릭터 구조체와 마찬가지로 적 구조체를 사용하기 위해선 2 번의 초기화 과정이 필요하다. 적 구조체를 초기화시키는 Create_Enemy_Object 함수, 적 구조체 내 Src_rect 멤버를 초기화 시키는 Assign_Enemy_Rect 함수이다.
- 위 두 함수는 앞서 설명한 Create_Player, Assign_Player_Rect 함수와 구동 방식이 매우 유사하다. 때문에 위 두 함수에 대한 설명은 생략하겠다.

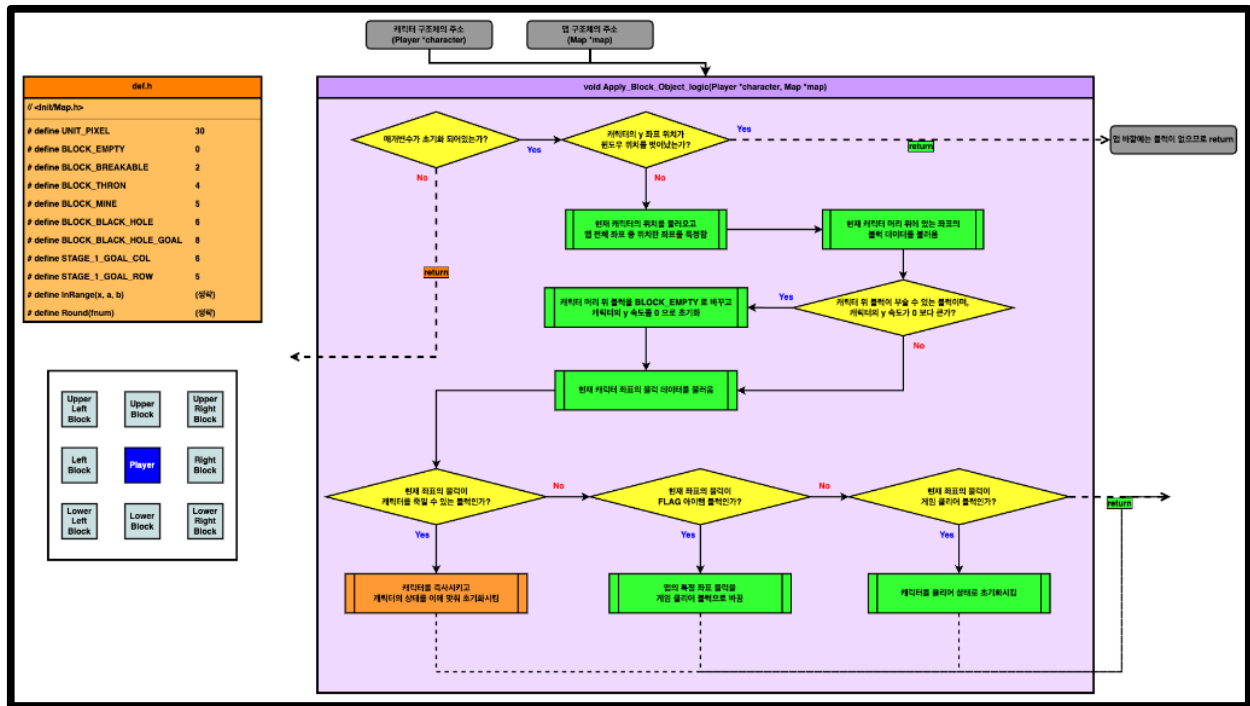
나) 적의 움직임 구현

- 이제 적이 지정된 좌표를 좌우로 순찰하는 기능을 알아보자. 해당 기능은 Partol_Enemy 함수를 통해 구현하였다. 해당 함수는 Enemy.h, Enemy.c 파일에 선언, 정의되어있다.



로 이동하다 가로막히는 경우이다.

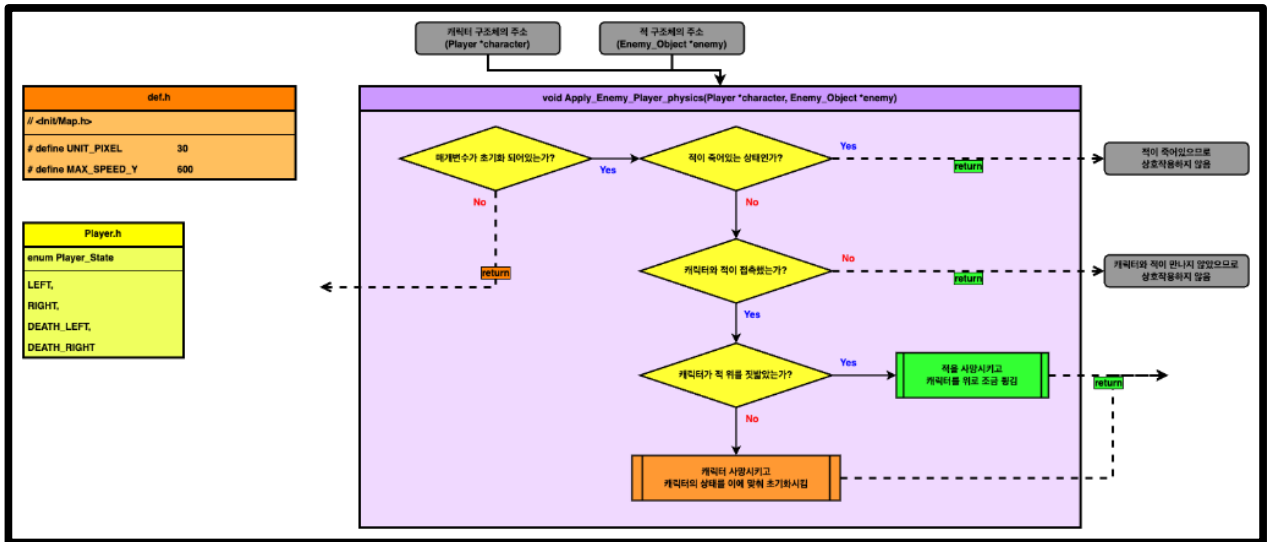
- 만약 인접한 블록이 캐릭터가 통과할 수 없는 블록이고, 해당 방향으로 캐릭터가 이동중 (캐릭터 속도를 통해 판단) 이라면, 캐릭터의 움직임을 제한하고 위치를 교정한다.
- 두번째 상호작용은 Apply_Block_Object_logic 함수를 통해 구현하였다. 해당 함수는 Player.h, Player.c 파일에 선언, 정의되어 있다.



- Apply_Block_Object_logic 함수는 이전 함수에 비해 간단하다. 만약 캐릭터 머리 위의 블록이 부술 수 있는 블록이고, 캐릭터가 점프 상태이면, 해당 블록을 바꿔치기하고 캐릭터의 속도를 조절한다. 만약 캐릭터가 캐릭터를 죽일 수 있는 블록을 만나면, 캐릭터를 사망시키고, FLAG 블록을 만나면 이에 맞는 처리를 진행한다.

나) 캐릭터와 적 간의 상호작용

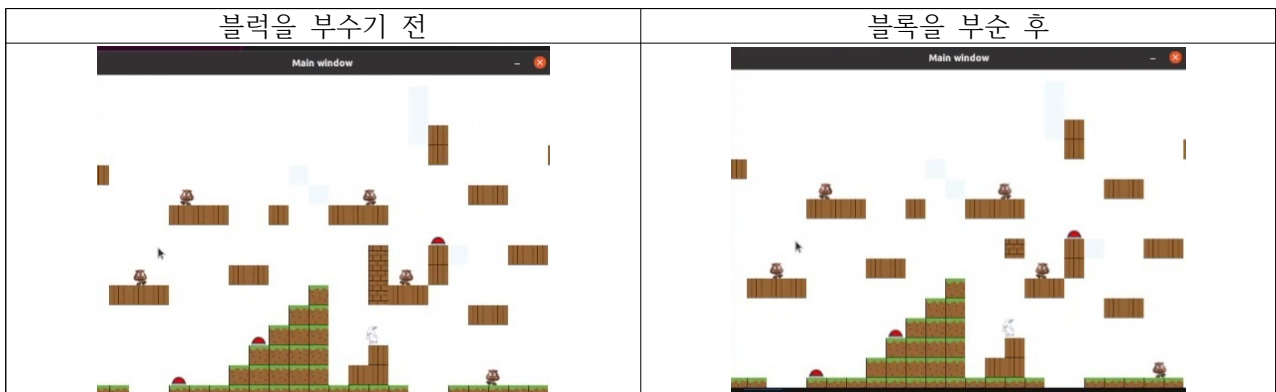
- 캐릭터와 적 간의 상호작용은 생각보다 단순하다. 캐릭터와 적이 만났을 때, 둘 중 하나는 사망하면 되기 때문이다. 해당 기능은 Apply_Enemy_Player_physics 함수를 통해 구현하였다. 함수는 Enemy.h, Enemy.c 파일에 선언, 정의되어 있다.



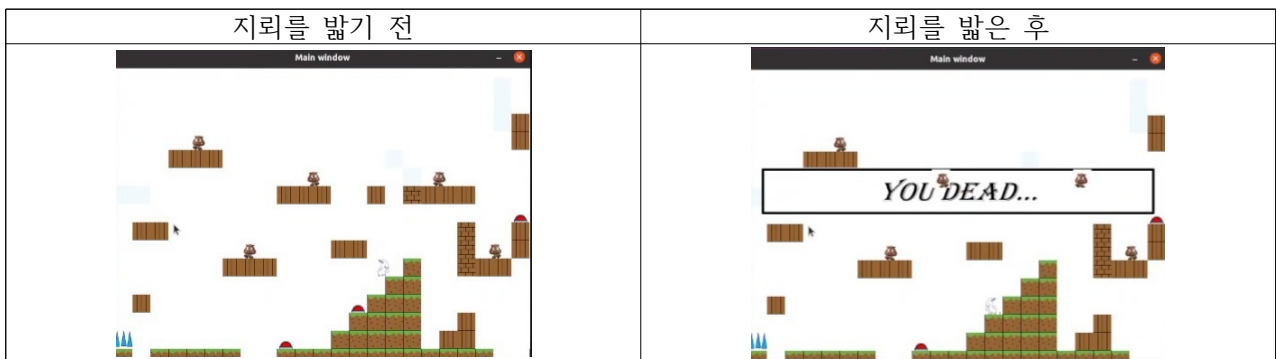
3. 동작 설명 (약 2페이지)

가. 프로그램 동작 화면

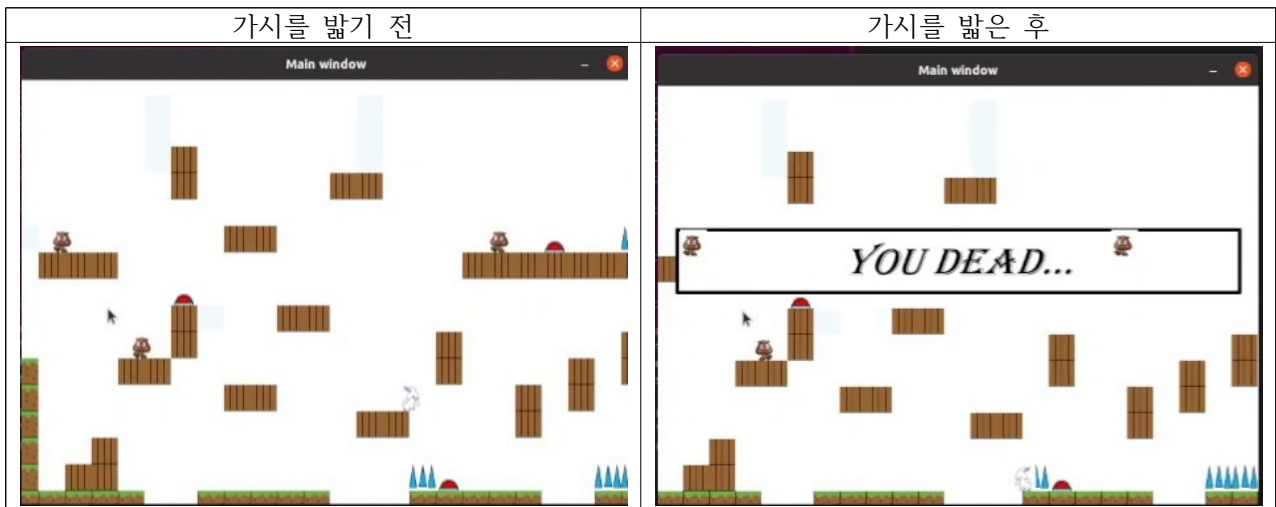
1) Breakable Block 을 밟아 없애는 과정



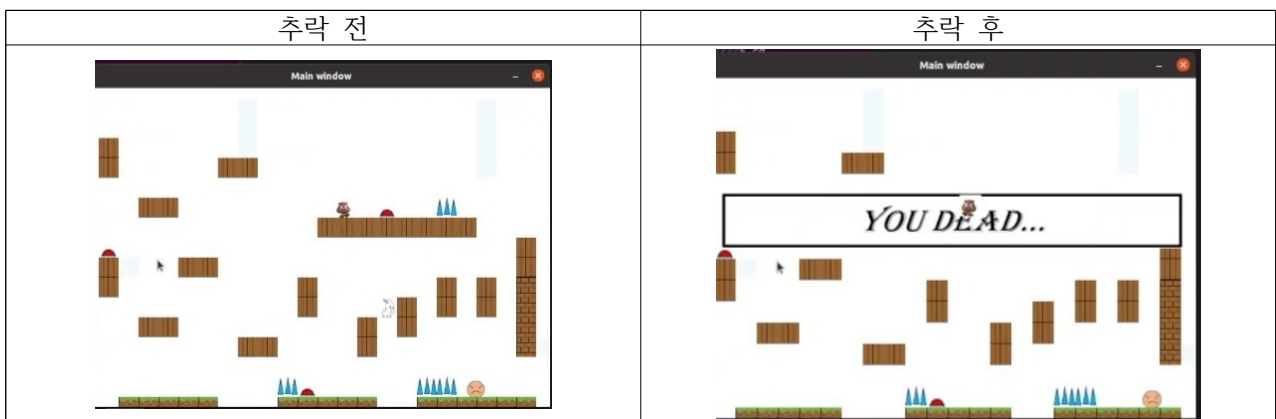
2) Mine Block 을 밟고 캐릭터가 사망하는 과정



3) Thron Block 을 밟고 캐릭터가 사망하는 과정

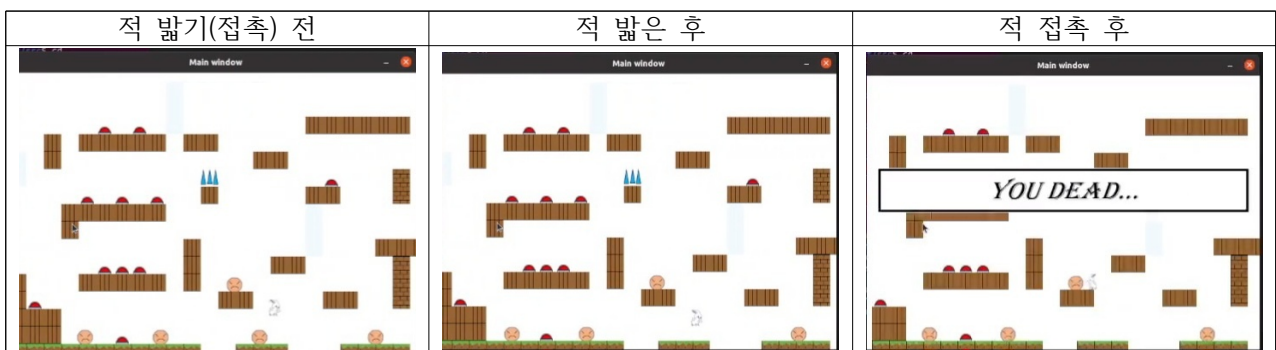


4) 캐릭터가 절벽으로 떨어져 사망하는 과정



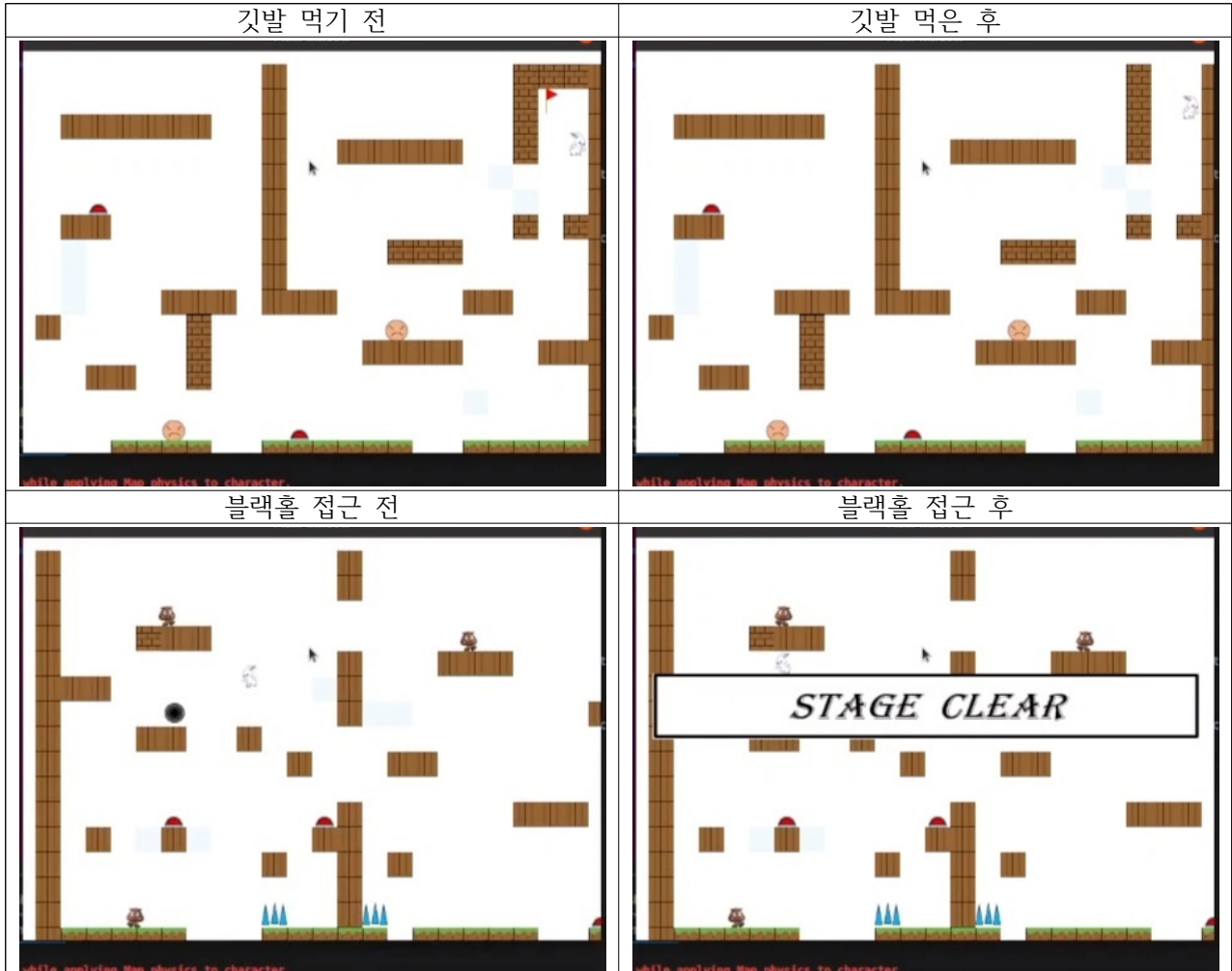
5) 캐릭터가 적을 밟아 죽이거나, 적과 접촉해 사망하는 과정

○ 이 때 적은 갈색 버섯이나 주황색 원형 형상을 의미한다.



6) 게임 클리어 조건 실행 과정

○ 깃발을 획득 한 후, 처음 위치 부근의 블랙홀에 들어가야 한다.



7) R 키를 눌러 처음 위치로 돌아오거나, 사망 후 R 키를 눌러 다시 살아나는 과정



나. 개발 목표 달성도

○ 1. 다. 에서 설정한 개발 목표 대비 달성도를 평가

평가 항목	개발 목표 대비달성도	우선 순위
소스코드의 모듈화	<p>달성도 : 100%</p> <p>기존의 개발 목표는 소스코드의 역할과 목적에 따라 모듈화를 진행하고 가독성과 개발 효율성을 높이는 것이었다. 우리 조는 부가적으로 필요한 자료 (asset) 과 기본이 되는 소스코드 (src) 의 구분부터 시작하여 각 기능별로 모듈화를 진행하였다.</p> <p>소스코드의 네이밍도 각 기능에 맞도록 Map, Enemy, Player등으로 구분지어 놓았고 함수의 선언과 정의를 분리해 소스코드를 제작하였다.</p> <p>각 함수나 필요한 상수의 네이밍도 각각의 기능에 맞도록 선언하여 헤더파일의 참조만 있다면 전체적으로 사용하며 인식하기 편하도록해 가독성과 개발 효율성을 모두 잡을 수 있었다.</p> <p>우선순위를 1순위로 둔 부분만큼 신경 써서 진행하였고, 충분히 이루었다고 생각한다.</p>	1
기능구현 정확성	<p>달성도 : 100%</p> <p>기존의 개발 목표는 앞서 구현한 모든 기능이 정상적으로 작동하도록 하는 것이었다.</p> <p>먼저 개발 환경을 구축하기 위해 또는 각각의 .c 파일을 개발하기 위해 주로 필요한 함수 및 매크로를 선언해 놓은 헤더파일들은 각각의 기능에서 필요한 모든 함수를 포함하고 선언되었다. 해당 기능들이 정확하게 구현이 되었기에 모든 .c파일들이 정확하게 기능한 것이다.</p> <p>그 다음 각각의 .c 파일들이다. 캐릭터와 관련된 소스코드인 Player.c 가 정확하게 구현되었기에 캐릭터의 이동과 이미지 구현이 자연스럽게 물리법칙을 따르는 것이다. 적에 대한 소스코드 Enemy.c 와 맵의 정보가 담긴 소스코드 Map.c 또한 정확히 구현되었기에 적의 움직임과 적의 기능, 맵에서의 블록들의 형상 및 기능이 정확히 구현된 것이다. 또한 해당 개발에 있어 편의를 위한 함수가 들어간 Util.c 가</p>	1

	<p>정확히 작동하고 게임의 직접적인 실행을 위해 필요한 main.c 가 정확하게 구현이 되었기 때문에 결국 main.c 가 성공적으로 빌드되며 게임이 정상적으로 작동되는 것이다.</p> <p>해당 부분도 우선순위를 1순위로 둔 부분만큼 신경 써서 진행하였고, 충분히 이루었다고 생각한다.</p>	
유지보수 용이성	<p>달성도 : 100%</p> <p>기존의 개발 목표는 추후 프로그램에 버그가 일어났을 때, 이를 추적하여 어느 부분이 문제인지 확인하기 쉽게 만드는 것이었다.</p> <p>해당 사항에 대해 높은 성취도를 얻기 위해 크게 설명해야 할 부분은 세 가지이다.</p> <p>첫 번째, Util.h에서 Shows_SDL_error함수를 선언하고 해당 .c 파일에서 에러에 대한 내용과 메시지, 에러가 발생한 코드상의 위치까지 확인이 가능하도록 만들었다.</p> <p>두 번째, malloc 함수를 사용해 메모리를 할당하거나 이미지를 렌더링 하는 함수를 사용할 경우에도 에러가 발생한 함수에 대한 정보와 메시지, 코드상의 위치까지 확인이 가능하도록 만들었다.</p> <p>세 번째, launch.json 파일을 다루며 여러 운영체제 및 환경에서 실행이 가능하도록 (Mac 과 Linux) 설정하였고, 해당 환경에서 디버깅까지 가능하도록 만들었다.</p> <p>해당 부분도 우선순위를 2순위로 둔 부분이지만 1순위로 둔 항목에 못지않도록 구현하였고, 충분히 이루었다고 생각한다.</p>	2
소스코드 확장성 및 이식성	<p>달성도 : 90%</p> <p>기존의 개발 목표는 추후 프로그램에 기능을 추가하기 위해 필요한 refacotring 을 최소화할 수 있도록 하고 여러 SDL 버전에서도 실행 가능하도록 하는 것이었다.</p> <p>우선 소스코드의 확장성은 소스코드의 모듈화와 유지보수의 용이성과 이어지는 부분이 많다.</p> <p>소스코드를 확장하여 더 많은 기능을 부여하기 위해서는 결국 새로운 기능을 담은 함수들이 선언되고 정의되어야 한다.</p> <p>우리 조는 모듈화와 유지보수 용이성에 대해서 충분히 구현되어졌다고 생각하기 때문에 기능확장을 생각한다면 기능의 구분에 따라서 기존에 작성된 헤더파일에서 다른 기능의 함수를 선</p>	3

	<p>언하고 .c 파일에서 명확하게 정의한다면 바로 적용가능 할 수 준이다. 따라서 확장성에 대해서는 충분히 안정적이라고 생각 한다.</p> <p>하지만 이식성에 대해서 약간의 아쉬운 점은 존재한다. 해당 아쉬운 점은 우분투 자체적인 SDL2 버전 호환 문제에서 기인한다. 우리가 수업시간에 자주 사용하던 Ubuntu 16.04에 환경에서는 SDL2(최대 버전 2.0.4)에 적용 가능한 함수들에 상 당히 제한사항이 많고 기능이 부족하여 가상환경에 Ubuntu 20.04를 추가적으로 설치하여 SDL2 (2.0.10)을 사용하였다. 해 당 사항에 있어서는 버전에 따른 이식성이 다소 부족하다는 아 쉬움이 남는다.</p> <p>성적 평가에 있어 해당 코드 빌드 및 실행에 필수적인 부분이 므로 명확하게 버전에 대한 설명이 필요하다고 생각해 다음과 같이 한 번 더 기술한 것이다.</p>	
--	--	--

4. 결론

- SDL2를 사용하여 게임의 그래픽 엔진을 구축했습니다. SDL2의 렌더링 기능을 활용하여 2D 스프라이
트, 애니메이션, 타일맵 및 파티클 효과를 구현하는 방법을 배웠습니다. 또한 SDL2의 이벤트 처리 기
능을 활용하여 키보드, 마우스 및 게임패드 입력을 처리하고 게임 캐릭터의 움직임 및 상호 작용을
구현했습니다.
- 배운 내용을 적극 활용하고자 SDL2를 사용하면서 메모리 관리와 최적화에 대한 고려 사항을 배웠습
니다. 텍스처 및 리소스를 효율적으로 로드하고 해제하는 방식을 적용해 봤습니다.
- 여러 가지 헤더파일과 소스 코드들을 만들다 보니 과정에서 발생한 버그를 식별하고 디버깅하는 기
술을 향상 시켰습니다. SDL2의 디버깅 도구를 활용하여 문제를 해결하는 법을 배웠습니다.
- 큰 규모의 개발을 진행하며 객체지향 프로그래밍이 어째서 각광 받았는지 몸소 체험한 것 같았습니
다. 또한 여러 참고자료를 접하며 각자의 구현 방식과 코딩 스타일을 엿볼 수 있었고, 궁극적으로 타
인이 작성한 코드를 더 잘 이해할 수 있게 되었습니다.
- 또한 가장 중요하게 느낀 점은 교수님께서 강조하신 "협업하여 개발하는 자세"에 대해 배울 수 있었
습니다.

5. 참고자료

[0] SDL2 설치 및 기본 내용 숙지

서울시립대학교 포털시스템 - 온라인강의실 - 프로그래밍 방법론 및 실습 (수업 배포 영상 및 자료)

[1] SDL2 라이브러리 조사

Wiki SDL : <https://wiki.libsdl.org/SDL2/FrontPage> (accessed on 2023-11-8).

[2] SDL2 라이브러리 학습

Lazyfoo net : <https://lazyfoo.net/tutorials/SDL/index.php> (accessed on 2023-11-15).

[3] SDL2 레퍼런스

Wikidocs : <https://wikidocs.net/161135> (accessed on 2023-11-05).

[4] Mario 레퍼런스

Github : <https://github.com/Luxon98/Super-Mario-Bros-game> (accessed on 2023-11-15)

[5] Cat Mario 레퍼런스(1)

Github : <https://github.com/akemin-dayo/OpenSyobonAction> (accessed on 2023-11-20)

[6] Cat Mario 레퍼런스(2)

Github : <https://github.com/weimzh/syobon> (accessed on 2023-11-20)