PBL크래프톤 Lv.1 Mission 1

dsohn, hopark

목표

슈팅게임 개발

- 방향키를 입력하면 플레이어가 움직여야함
- 스페이스 바를 입력하면 총알을 발사해야함
- 플레이어가 공격해야하는 적이 있어야함

개발 일지

				dsohn		hopark		ALL		
				월	화	수	목	금	토	일
3/7 ~ 3/13	분류	task	담당자	3/7	3/8	3/9	3/10	3/11	3/12	3/13
	협업 환경 세팅	개발일정 & 리소스배분	ALL							
		이슈 트래킹방법 조사 및 선정	ALL		보고서					
	작업 환경 세팅	개발에 사용할 개발 언어 리서치 및 선정	ALL		보고서					
		통합 개발 환경 리서치 및 선정	ALL		보고서					
		버전 관리 툴 리서치 및 선정	ALL		보고서					
	게임 엔진 구현	게임 매니저 구현	dsohn							
		SDL 기반 화면 출력	dsohn							
		게임 오브젝트 객체 구조	dsohn							
	입출력	키보드 입력 처리를 위한 메서드 선정 및 구현	hopark			mac ->SDL				
	객체 관리	객체 구현(player)	dsohn							
		객체 구현(enemy)	hopark							
		자료구조 리서치	hopark							
		메모리관리	dsohn							
	보고서 작성	이슈 별 개발 일지 작성	ALL							
		최종 보고서 수정 및 마무리	ALL							

이슈 트래킹

	Github Issues	Trello	Jira
요금	무료	무료	10명까지 무료
칸반보드	없음	있음	있음
사용 난이도	쉬움	어려움	아주 복잡함
사이트 속도	빠름	빠름	느림

복잡한 프로젝트가 아니고 인원이 많은 팀도 아니다 보니, 복잡한 기능보다는 간단한 이슈트래킹 툴을 쓰는 것이 다 좋을 것이라 생각했다.

또한 github에서 바로 쓸 수 있기 때문에 편의성도 매우 높다고 판단했다.

개발 언어 비교

	c++	c#	java
속도	빠름	느림	느림
생산성	나쁨	좋음	좋음
메모리관리	수동 (빠름)	자동 (느림)	자동 (느림)
지원 라이브러리	보통	많음	많음
다중상속	가능	불가	불가

그 외
- c : 여러 장점이 많지만 절차지향언어라 게임 개발에 적합하지 않다
- python : 속도가 너무 느리고 병렬처리에 제약이 많다.

결론 : 속도가 빠르고 다중상속이 가능한 C++ 언어를 선택하기로 했다.

통합개발환경 (선정언어 : c++)

1. Visual Studio
   - 기능이 많지만 무겁고 MacOS에선 C#만 지원한다.
2. Clion
   - 기능이 매우 많지만 **유료**다.
3. Dev c++
   - 업데이트가 끊겨 노후되었다.
4. Online IDE
   - 인터넷에 의존성이 높아 인터넷속도가 속도에 영향을 끼친다
5. **Vscode + makefile**
   - 속도가 빠르고 가볍다
   - 여러 환경에서 사용이 용이하다.

버전관리 툴

	Git	SVN	Perforce
관리 구조	분산관리	중앙관리	
접근 관리	저장소 별 permission 할당	파일 또는 디렉토리 별 permission 할당	
속도	commit, diff, merge는 빠르지만 많은 개발자의 push, pull은 느리다	느림	commit, diff, merge는 느리지만 push, pull은 빠르다.
비용	무료	무료(서버 필요)	부분유료(서버 필요)

자주 사용되는 git과 perforce중 프로젝트가 인원이 적은 프로젝트이고, 버전관리 서버 구축이 필요 없는 git을 사용하기로 했다.

키보드 입력 메서드(1)

(1) <termios.h >를 이용한 터미널 제어와 read를 이용한 구현

- 터미널상에 출력을 제어
- 버퍼의 크기를 1로줄여
- 한 글자씩 키를 받아온다.

```c
#include <termios.h>
#include <unistd.h>
#include <stdio.h>

int main(void)
{
    char c;
    struct termios term;

    tcgetattr(STDIN_FILENO, &term);
    term.c_lflag &= ~ICANON;      // non-canonical input 설정
    term.c_lflag &= ~ECHO;        // 입력 시 터미널에 보이지 않게
    term.c_cc[VMIN] = 1;          // 최소 입력 버퍼 크기
    term.c_cc[VTIME] = 0;         //버퍼 비우는 시간 (timeout)
    tcsetattr(STDIN_FILENO, TCSANOW, &term);

    while (read(0, &c, sizeof(c)) > 0)
    {
        printf("input: %c\n", c);
    }
}
```

키보드 입력 메서드(2)

(2) GetAsyncKeyState 함수

- <window.h> 헤더에서 볼 수 있듯이 윈도우에서 사용할 수 있는 함수
- keyboard나 mouse클릭상태를 반환한다

```c
#include<stdio.h>
#include<windows.h>
void gotoxy(int x, int y) {
    COORD Pos = { x - 1, y - 1 };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Pos);
}
void main() {
    int x = 1;
    int y = 1;
    gotoxy(x, y);
    while (1) {
        if (GetAsyncKeyState(VK_LEFT) & 0x8000) { //왼쪽
            x--;
        }
        if (GetAsyncKeyState(VK_RIGHT) & 0x8000) { //오른쪽
            x++;
        }
        if (GetAsyncKeyState(VK_UP) & 0x8000) { //위
            y--;
        }
        if (GetAsyncKeyState(VK_DOWN) & 0x8000) { //아래
            y++;
        }

        system("cls");
        gotoxy(x, y);
        printf("♥");
    }
}
```

키보드 입력 메서드(3)

(3) SDL 라이브러리 이용하기

- C 프로그래밍 언어로 짜여진 크로스플랫폼 멀티미디어 라이브러리
- 비디오, 이벤트, 디지털 오디오, CD-ROM, 사운드, 스레드, 공유 객체 불러오기 등을 관리할 수 있다.

```
while (!close) {
    SDL_Event event;
    // Events management
    while (SDL_PollEvent(&event)) {
        switch (event.type) {
        case SDL_QUIT:
            // handling of close button
            close = 1;
            break;
        case SDL_KEYDOWN:
            // keyboard API for key pressed
            switch (event.key.keysym.scancode) {
            case SDL_SCANCODE_W:
            case SDL_SCANCODE_UP:
                dest.y -= speed / 30;
                break;
            default:
                break;
            }
        }
    }
```

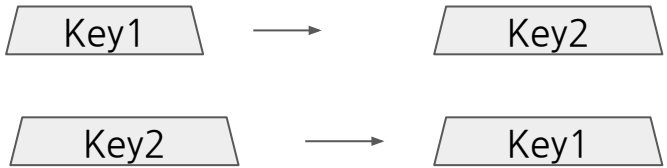
키보드 입력 메소드 비교

	〈termios.h 〉	GetAsyncKeyState	SDL Library
멀티플래폼 지원	window 지원 x	mac os 지원 x	o
코드의 간결성	간결한편	매우 간결하다	보통
복합키 입력 가능한가?	불가	가능	가능
더블 입력이 가능한가?	가능	가능	가능
래퍼런스가 많은가?	적다	보통	많다

결론 : 대부분 플래폼에서 쓸 수 있고 멀티키 입력구현이 가능한 SDL Library를 채택하였다.

키 입력 처리

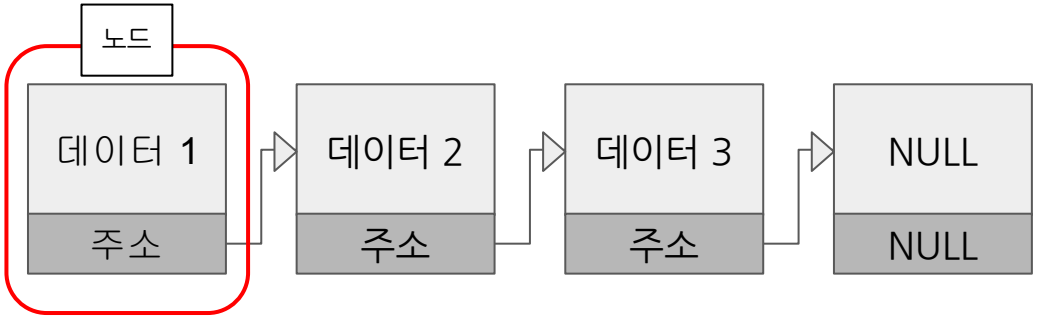
- 현재 keyboard의 입력 상태를 확인해주는 SDL_GetKeyboardState함수를 이용하여 상태를 저장하고 키보드의 상태변화를 비교하며 구현하였다.

KeyDown	KeyUP
 Key가 안 눌린 상태 -> Key가 눌러짐	 Key가 눌러진 상태 -> Key가 눌러지지 않음
DoubleKey	**MultiKey**
 Key1(2)가 눌러진 상태 -> Key2(1)가 눌러짐	 키를 눌렀는데 이전에 마지막으로 키누른 시간과의 시간 차가 임의의 시간 보다 짧음

키 입력 처리

```cpp
bool InputManager::KeyDown(SDL_Scancode scanCode) {
    return(GetKeyState(scanCode).nowState && !GetKeyState(scanCode).lastState);
}

bool InputManager::GetKey(SDL_Scancode scanCode) {
    return(GetKeyState(scanCode).nowState);
}

bool InputManager::KeyUP(SDL_Scancode scanCode) {
    return(!GetKeyState(scanCode).nowState && GetKeyState(scanCode).lastState);
}

bool InputManager::DoubleKeyDown(SDL_Scancode scanCode) {

    if (KeyDown(scanCode) && GetKeyState(scanCode).lastUptime && _instance->_currentTime - GetKeyState(scanCode).lastUptime < _instance->_updateInterval)
        _instance->_keyState[scanCode].isDouble = true;
        return true;
    }
    return false;
}

bool InputManager::MulitKeyDown(SDL_Scancode scanCode1 , SDL_Scancode scanCode2) {
    return((GetKeyState(scanCode1).nowState && KeyDown(scanCode2))
            || GetKeyState(scanCode2).nowState && KeyDown(scanCode1));
}
```

객체 관리 자료구조 - 연결 리스트



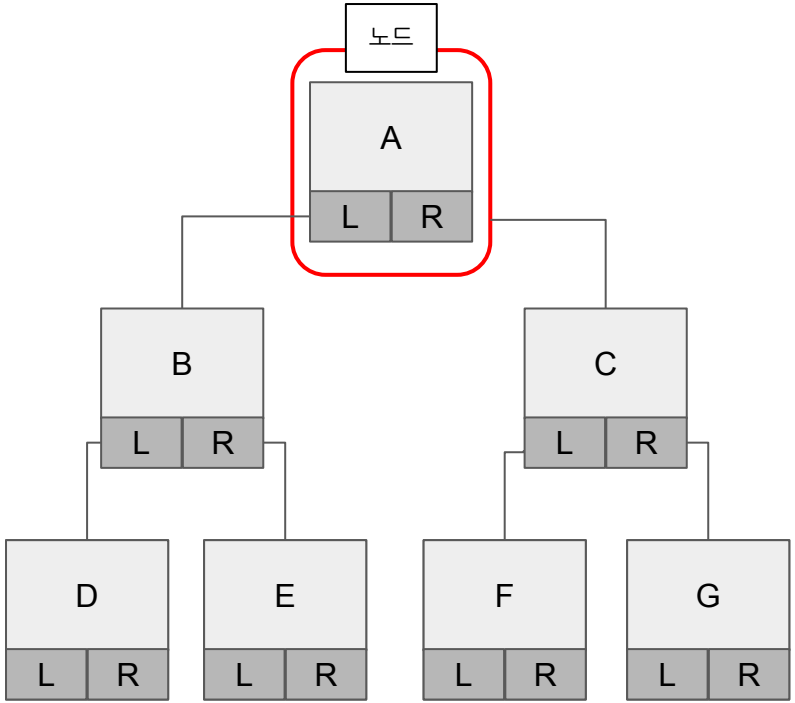
특징	
• 여러개의 노드(data + link)를 각각 연결한 선형 자료구조 • 포인터를 이용해 다음 데이터의 주소를 가지고 있음 • 연결선의 개수에 따라 이중연결리스트 / 단일 연결리스트로 구분 지을 수 있음	• 원소 추가 삭제가 빠름 • 추가 원소 이동 연산 불필요 • 인덱스로 탐색 할 시 편리 하나 데이터를 검색은 다른 자료구조에 비해 느림

객체 관리 자료구조 - 벡터

데이터 1	데이터 2	데이터 3	NULL
0	1	2	3

특징	
• 자료의 논리적 위치와 물리적 위치가 일치함 • 선형 자료 구조 • 데이터의 index를 알경우 매우 빠른 접근이 가능함	• 원소 추가 삭제가 매우 비효율적임 • 오브젝트 추가시 메모리 낭비가 발생할 수 있음 • 데이터 검색은 트리 구조에 비해 느림

객체 관리 자료구조 - 이진트리



특징

- 트리 : 부모 - 자식 관계로 연결된 비선형 자료구조이다.
- 이진트리 : 부모가 2개 이하의 자식을 가진 트리구조
- 사용 방법과 기준에 따라 탐색의 속도를 높일 수 있다.

메모리 관리 방식

	고정형 메모리 풀	가변형 메모리 풀	오브젝트 풀
시스템 콜 사용빈도	시작과 끝에 몰아서	시스템 콜 최소화	시작과 끝에 몰아서
메모리 최적화	메모리 낭비 심함	최적화율 높음	보통
릭 방지	O	O	O
범용성	O	O	종류별로 필요함

보통 많이 사용되는 오브젝트가 뚜렷한 경우 (지금 만드는 슈팅게임을 예로 들면 총알, 몬스터 등) 범용성 높은 메모리 풀을 사용하기보단 오브젝트 별 풀을 만들어 사용하는 경우가 많은 걸로 조사했다.

따라서 총알, 몬스터 등을 관리할 수 있는, BulletManager, MonsterManager등을 만들어 그 오브젝트에서 총알 몬스터 메모리를 관리해주는 방식으로 하는 것이 좋아보인다.

https://github.com/42Seoul-NULL/Terminvader