

[오픈소스SW개론] 과제5

Snake Game OSS 최종 보고서

-

2019.03.26 ~ 2019.05.31

학과	이름	이메일 주소
소프트웨어학과	김경남	kkyy0126@naver.com
소프트웨어학과	진수빈	subinjin22@gmail.com

목차

1. 프로젝트 개요	2
1-1 프로젝트 주제	
1-2 주제 선정 이유	
1-3 프로젝트 목표	
1-4 프로젝트 범위	
1-5 프로젝트 환경	
2. Snake Game OSS	6
2-1 선택한 Snake Game 오픈소스 소개	
2-2 다운로드 및 출처	
2-3 라이선스	
3. 프로젝트 수행 내용	10
3-1 프로젝트 규칙 및 버전 기준 정하기	
3-2 기존 코드의 오류 개선하기	
3-3 가독성을 기준으로 clean 코드로 개선하기	
3-4 기능 추가 및 기능 오류 개선	
3-5 Git을 이용하기	
3-6 GitHub을 이용하기	
3-7 문서화 작업하기	
4. 역할 분담	17
4-1 코드 분석, 개선, 기능추가 역할 분담	
5. 추진 방법 및 수행 일정	18
6. 회의록	20
7. 참고문헌 및 사이트	26

1. 프로젝트 개요

1-1. 프로젝트 주제

해당 프로젝트는 오픈소스를 활용한 프로젝트로, 하나의 오픈소스를 복제 및 수정하여 2차파생물을 일정 규칙에 준수하여 도출해 내는 데에 의의가 있다. 프로젝트의 진행을 위해 선별된 주제는 Snake Game으로, 대표적인 고전 아케이드 게임이다. Snake Game은 화면에 나타나는 뱀(사용자의 캐릭터)을 방향키로 움직여 랜덤위치에 생성된 먹이를 먹으며 뱀의 크기를 늘려나간다는 방식을 기본으로 한다.



(출처: coolmathgames <https://www.coolmathgames.com/0-snake>)

1-2. 주제 선정 이유

다양한 게임들 중 Snake Game을 주제로 선택한 이유는, 남녀노소 누구나 이해하기 쉬운 룰로 간단하고 빠른 플레이가 가능하고, 이로 인해 다양한 변형이 가능하기 때문이다. 또한 오래된 고전 게임이기 때문에 그동안 축적된 여러 Snake Game의 오픈소스들을 참고하여 효과적인 구현 방식이나 적합한 자료구조 등을 적용시켜 볼 수 있다는 점에서 Snake Game을 주제로 선정하게 되었다.

1-3. 프로젝트 목표

해당 프로젝트의 목표는 코드 실행에 있어 오류를 최소화하여 게임의 버그를 줄이는 것과 새로운 기능을 추가하여 사용자가 Snake Game을 좀 더 재미있고 신선하게 플레이 하도록 하는 것이다. 또한 코드를 가독성의 측면에서 재검토 및 수정하여 재사용성이 높은 소스 코드로 완성시키는 것 역시 목표로 삼고있다.

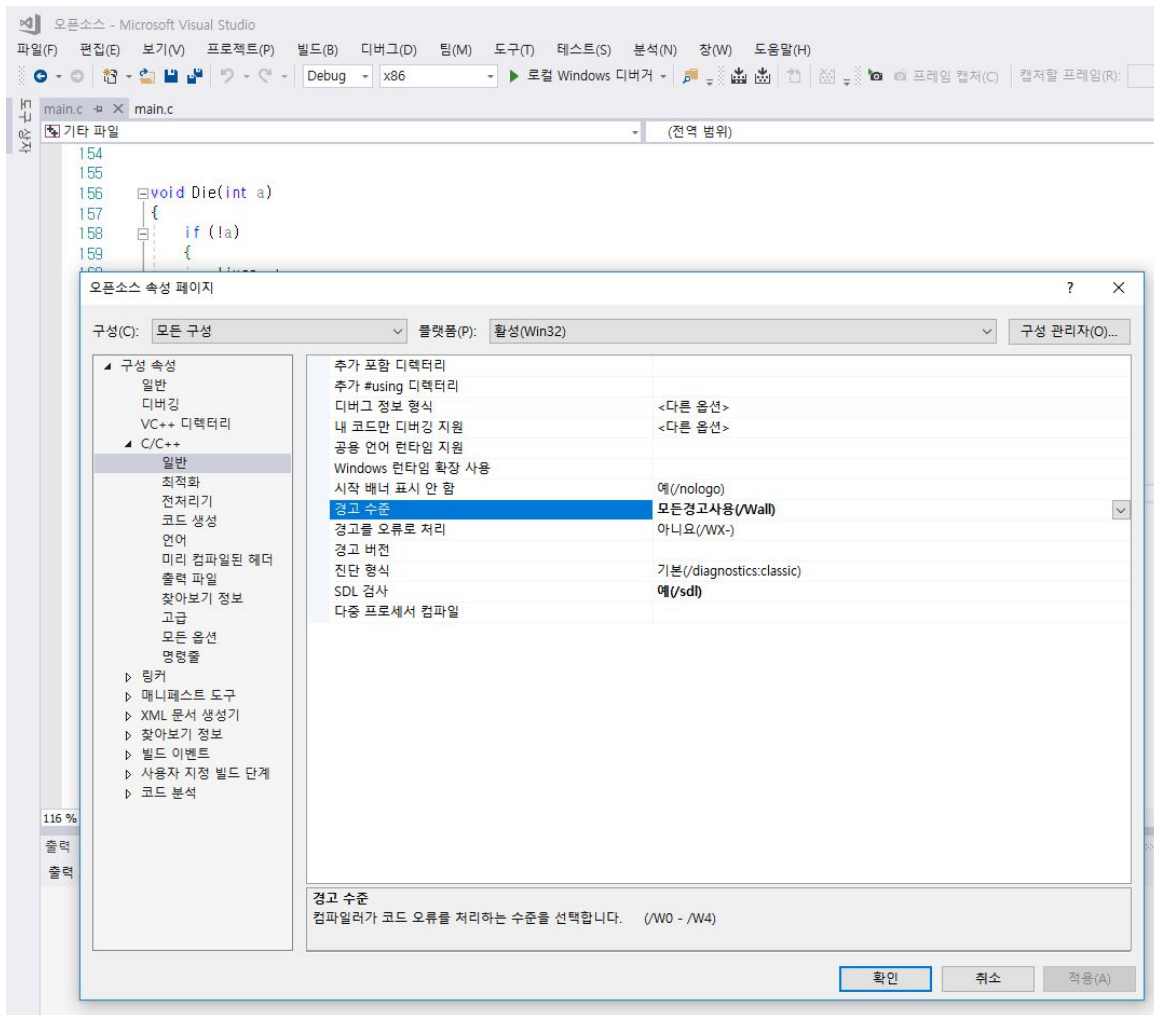
1-4. 프로젝트 범위

소스코드를 가독성의 측면에서 수정하고, 이식성 향상을 위한 코드로 변경하여 코드의 재사용성을 높이는 것까지 진행한다. 또한 해당 프로젝트는 오픈소스 기반이므로 GitHub를 통해 재배포하는 것 역시 프로젝트의 범위에 해당한다.

1-5. 프로젝트 환경

1-5-1. 로컬개발환경

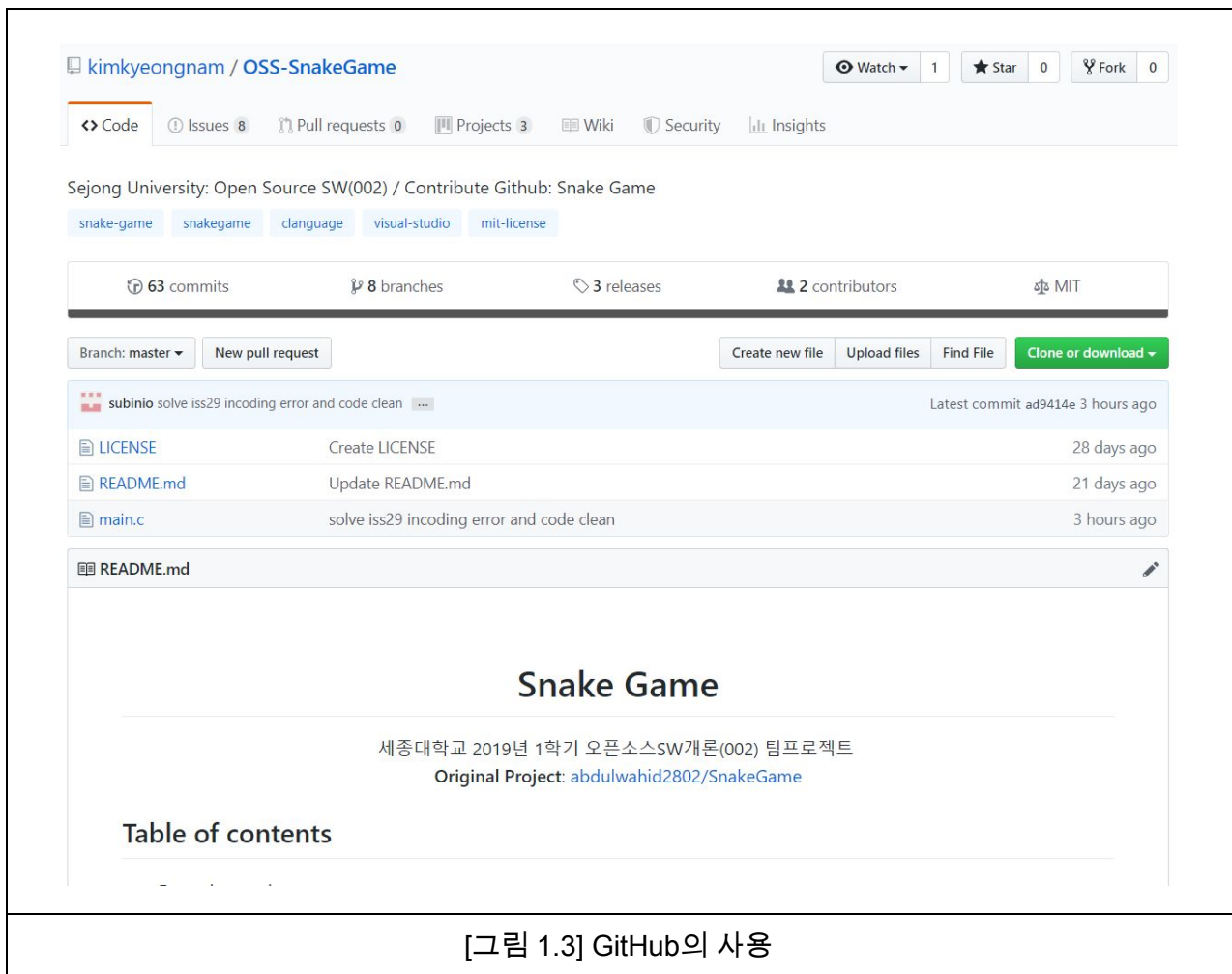
해당 프로젝트는 Windows 10 환경에서 진행되었고, 언어는 C(버전 99), 개발환경툴은 Microsoft Visual Studio 2017을 사용하였다. Visual Studio 2017에서 코드를 컴파일 할 때, 컴파일 환경은 Warning Level을 “모든 경고 사용(/Wall)”으로 설정 한 후 코드 수정을 진행하였다. Warning Level은 ‘속성 > C/C++ > 일반 > 경고 수준’에서 수정이 가능하다.



[그림 1.2] Warning Level 설정

1-5-2. 버전 관리 시스템

해당 프로젝트는 총 2명에서 협업하여 개발하는 프로젝트이다. 그러므로 협업자들의 변경사항을 시간에 따라 저장할 수 있고, 특정 시점 코드의 복원을 가능하게 하는 버전 관리 시스템을 활용하고자 한다. 여기서 사용한 버전 관리 시스템은 Git이고, Git을 지원하여 Git 리포지토리를 인터넷 상에 제공해 주는 서비스인 GitHub를 이용할 것이다.



[그림 1.3] GitHub의 사용

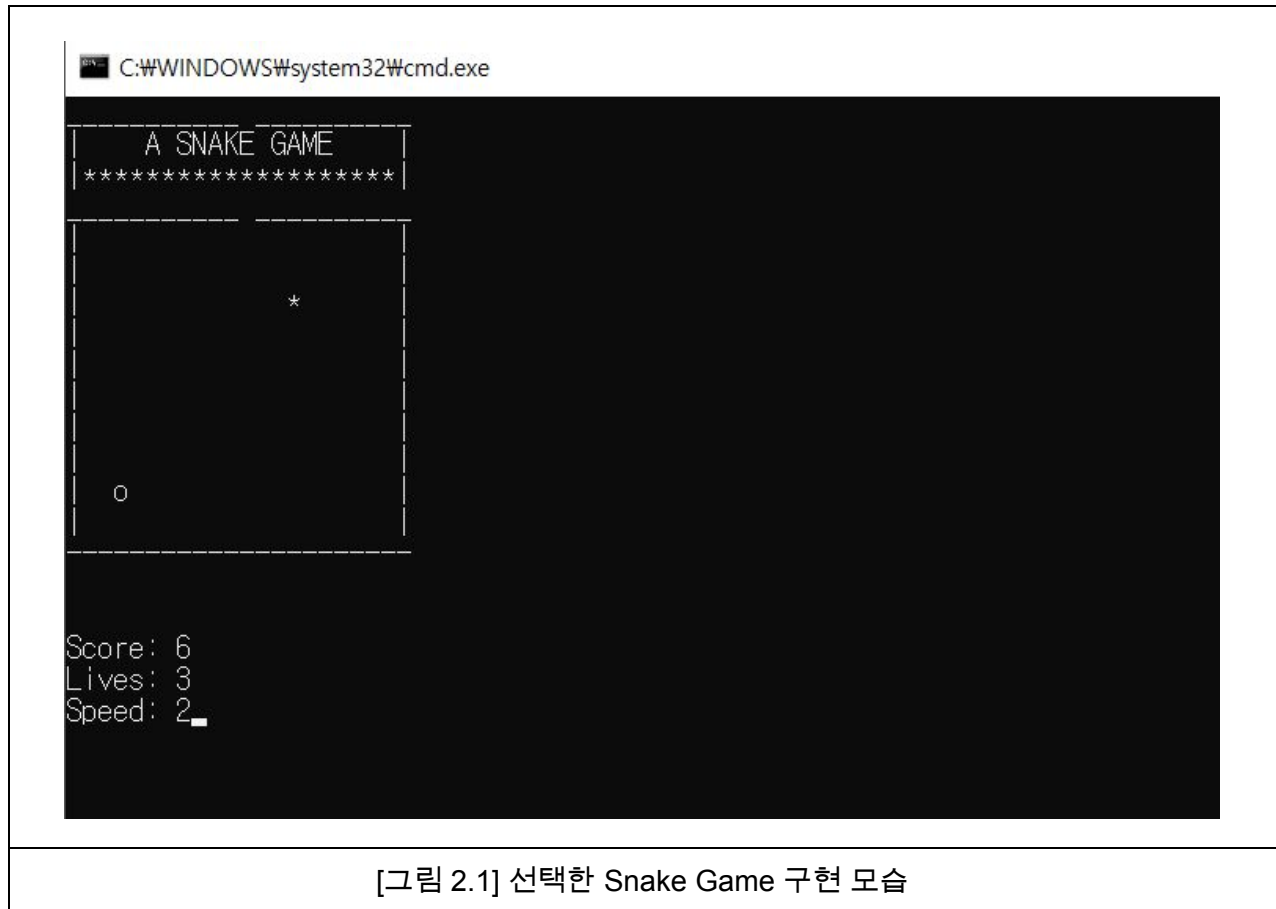
1-5-3. 라이선스 - MIT License

라이선스란 OSS개발자와 이용자 간에 사용방법 및 조건의 범위를 명시한 계약으로 OSS를 사용하기 위해서는 OSS개발자가 만들어 놓은 사용방법 및 조건의 범위에 따라야 한다. 우리가 사용할 Snake Game 오픈소스는 MIT라이선스를 따르고 있으며, MIT라이선스는 실행환경에서 반드시 준수해야하는 규칙은 따로 없고, 소스코드 재배포 시, 라이선스와 저작권 관련 명시를 지켜주면 된다.

2. Snake Game OSS

2-1. 선택한 Snake Game 오픈소스 소개

GitHub를 통해 Snake Game 오픈소스를 가져오게 되었고, 선택한 Snake Game은 C언어로 작성되어 [그림 2.1]과 같이 cmd창에서 구현된다.



실행과 동시에 뱀을 나타내는 '*'이 움직이며 사용자는 방향키를 눌러 먹이쪽으로 뱀('*')의 방향을 움직인다. 점수가 올라갈수록 뱀이 움직이는 속도(Speed)는 증가한다. 총 목숨 수는 3으로, 벽에 3번 이상 부딪히면 게임은 종료된다.

기존의 Snake Game은 먹이를 먹으면 뱀의 몸통 길이가 늘어나는게 눈에 보이지만, 선택한 Snake Game 오픈소스는 눈에 보이는 뱀의 길이는 늘어나지 않고, Score점수만 증가하여 먹이를 먹었다는 것을 표현한다.

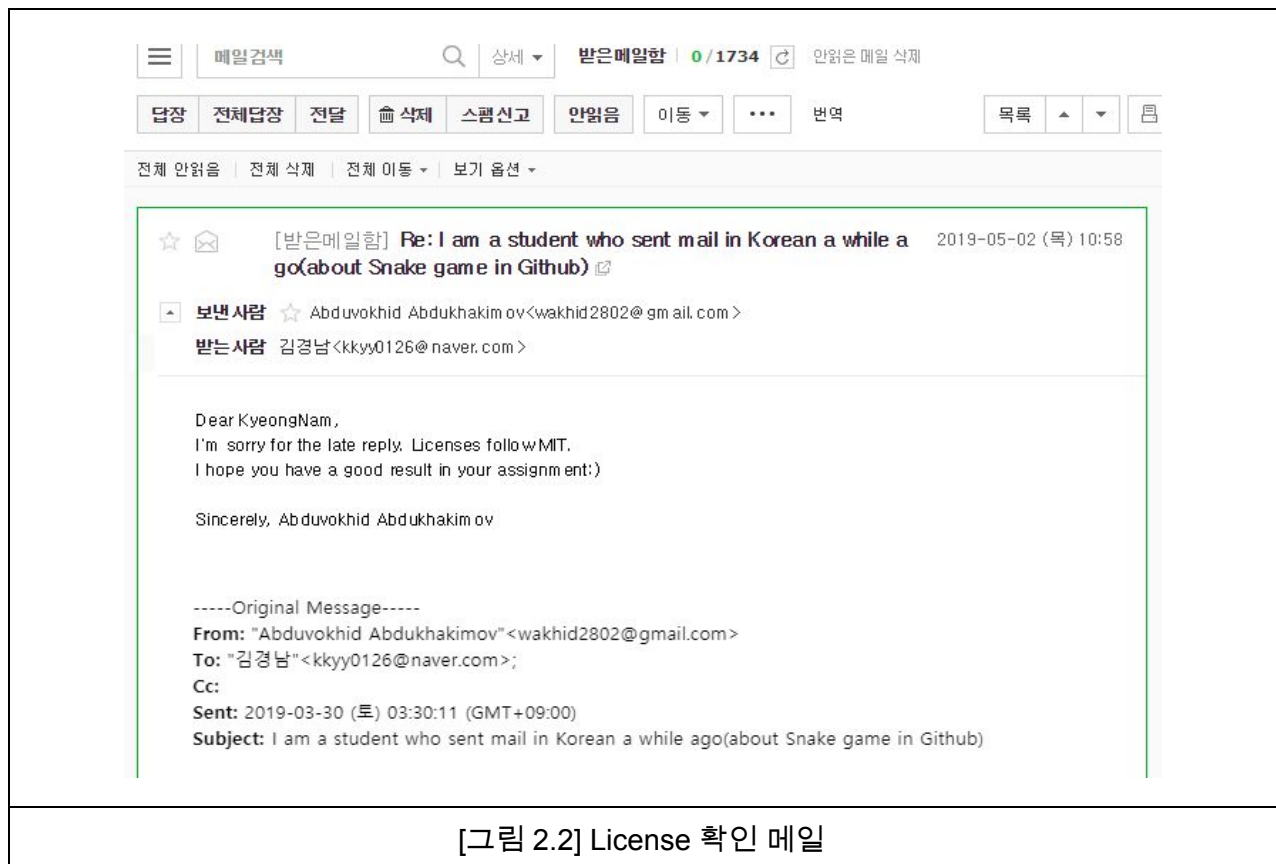
2-2. 다운로드 및 출처

프로젝트를 위해 선택한 Snake Game 오픈소스의 원작자는 Abduvokhid Abdukhakimov (GitHub ID : abdulwahid2802)로, 원작자의 GitHub에서 Snake Game 오픈소스를 가져오게 되었다. 오픈소스를 가져와 코드의 복제 및 수정을 위해 원작자의 Snake Game 리포지토리를 포크(Fork)한 후, Clone하여 팀장의 GitHub에 새 리포지토리를 생성하였다. 오픈소스 원본 출처와 생성된 새 리포지토리의 출처는 아래와 같다.

- Snake Game 원작자(원본) 출처 : <https://github.com/abdulwahid2802/SnakeGame>
- 개발을 위한 새 리포지토리 출처 : <https://github.com/kimkyeongnam/OSS-SnakeGame>

2-3. 라이선스

원작자의 GitHub를 방문했을 때, Snake Game의 라이선스가 표시되어 있지 않아 원작자에게 메일을 보내 확인 해본 결과, 해당 Snake Game 오픈소스는 MIT라이선스를 따른다는 것을 알 수 있었다.



MIT 라이선스(MIT License)는 미국 매사추세츠 공과대학교(MIT)에서 해당 대학의 소프트웨어 공학도들을 돕기 위해 개발된 라이선스이다. MIT 라이선스는 SW의 소스 코드를 공개하지 않아도 되는 대표적인 오픈소스SW 라이선스이다. 즉, MIT 라이선스를 따르는 소프트웨어를 개조한 제품을 반드시 오픈소스로 배포해야 한다는 규정이 없어 다른 엄격한 라이선스들에 비해 자유로운 편이다. MIT 라이선스의 준수사항은 다음과 같다.

- SW에 대한 자유로운 사용, 복제, 배포 허용
- SW를 배포하는 경우, 저작권을 반드시 표시
- SW를 배포하는 경우, 라이선스 사본을 반드시 첨부
- SW를 배포하는 경우, 보증의 부인과 책임의 제한에 대한 내용을 반드시 표시
- 수정 프로그램에 대한 소스코드의 공개를 요구하지 않으므로 상용 SW에 무제한 사용가능
- 소스코드 재배포를 위해서는 저작권 표시와 문구, 면책조항을 유지하여야 함
- MIT 라이선스에 따른 프로그램을 바이너리 형태로 재배포 하는 경우에는 소스코드를 배포하는 경우와 마찬가지로 배포판과 함께 제공되는 문서 또는 그 밖의 매체에 저작권 표시와 문구, 면책조항을 담아야 함.
- 공개 범위 : MIT 라이선스는 의무사항만 준수한다면 소스코드를 공개하지 않아도 됨.

3. 프로젝트 수행 내용

3-1. 프로젝트 규칙 및 버전 기준 정하기

3-1-1. 변수 작성 방법

- 1) 변수는 한 목적으로만 사용할 것.
- 2) 변수는 항상 소문자로 시작하고, 다음단어는 대문자로 구분할 것.
- 3) 함수의 매개변수 역시 그 의미를 명확히 표현하는 단어로 작성할 것.

3-1-2. 함수 작성 방법

- 1) 함수명은 항상 대문자로 시작하고, 다음단어는 대문자로 구분할 것.
- 2) 변수명과 함수명을 같게 하지 말 것.
- 3) 자주쓰는 함수의 접두사를 공통으로 사용할 것.

함수의 접두사	설명
Avr	평균값
Cnt	데이터의 개수를 얻을 때
Get	어떤 값을 얻어낼 때
Set	값을 설정할 때
Is	무엇인가 물을 때
Key	데이터 중 키값만 얻을 때

3-1-3. Branch 규칙

- 1) Branch 생성 기준

코드개선, 버그수정, 기능추가를 기준으로 두고 각각의 기준마다 새로운 내용에 대해 브랜치를 생성한다.

예) 꼬리생성에 대한 기능 추가일 때, 인코딩에 대한 버그 수정일 때 브랜치 생성.

2) Branch 작성 방법

- ■/issNum_△ 의 방식으로 작성
- ■ : clean(코드 개선), bug(버그 수정), new(기능 추가)
- issNum : 관련 이슈 번호
- △ : 구체적인 키워드
- 예) new/iss17_tail : issue17번의 꼬리 생성에 관한 기능 추가
- bug/iss29_draw : issue 29번의 출력과 관련한 버그 수정

3-1-4. Commit 규칙

1) Commit 기준

각각의 로컬 저장소에서 branch를 생성하여 각자가 맡은 기능에 대한 구현이 어느 정도 실현 되었으면 Commit을 진행한다. 단, 해당 코드가 돌아가지 않거나 도중에 에러가 나는 (디버깅이 되지 않은) 코드는 Commit을 진행해서는 안된다.

2) Commit 메세지 작성 기준

- 제목과 본문을 한 줄 띄워 분리
- 제목은 50자 이내로, 첫글자는 대문자로 작성
- 제목 끝에 '.'(마침표) 사용 금지
- 제목은 명령조로 작성
- 본문은 '어떻게'보다 '무엇을', '왜'에 맞춰 작성
- 본문 작성 시 변경 내용 마다 번호를 붙여 작성

3-1-5. Merge 규칙

1) Merge 기준

각자가 맡은 작업을 수행한 후 Pull request를 진행하여 다른 팀원의 피드백 (Merge 찬성 혹은 개선 내용 추가)을 바탕으로 Merge를 시행하거나 해당 코드를 다시 수정한다.

3-1-6. 버전 기준

x.y.z.를 기준으로 각각의 규칙에 맞춰 작성한다.

x		y	z	
0	기존코드에 클린코드로 수정	commit 시 작성했던 설명 중 제일 처음 나오는 함수 번호로 작성 0: 아무런 작업도 하지 않음 1: header 수정 2: main() 3: InitInfo() 4: InitSnake() 5: DrawSnake() 6: DelSnake() 7: Display() 8: MoveSnake() 9: GetInput() 10: EatFood() 11: ChkSnakeDie() 12: DrawTitle() 13: DrawScore() 14: DrawMap() 15: DrawFood() 16: SetCurrentCursorPos() 17: RemoveCursor() 18: RelocateFood() 19: RelocateSnake() 20: IsCollisionTopBorder() 21: IsCollisionBottomBorder() 22: IsCollisionLeftBorder() 23: IsCollisionRightBorder() 24: AddTail() 25: DelAllTail() 26: ChkOppositeDir() 27: FreeAllMemories() 28: IsAlive() 29: GameStart() 30: SelectMenu() 31: ShowIntro() 32: Choice()	0	버그를 수정하지 않거나, 버그가 없을 시
1	기본적인 기능 추가			
2	최종본 및 최종버그 수정			
3	OSS강의 이후의 업데이트		1	버그 수정 시

3-2. 기존 코드의 오류 개선하기

3-2-1. 컴파일 환경 Warning Level 오류 개선

컴파일 환경의 Warning Level을 최상으로 했을 때의 빌드 오류를 수정하여 준다.

```
s:\10\include\10.0.16299.0\us\winioc\h(7548): warning C4668: '_WIN32_WINNT_WIN10_RS1'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(7553): warning C4668: '_WIN32_WINNT_WIN10_TH2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(7557): warning C4668: '_WIN32_WINNT_WIN10_TH2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(7564): warning C4668: '_WIN32_WINNT_WIN10_RS2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(7574): warning C4668: '_WIN32_WINNT_WIN10_RS2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(7579): warning C4668: '_WIN32_WINNT_WIN10_TH2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(7583): warning C4668: '_WIN32_WINNT_WIN10_RS3'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(7777): warning C4668: '_WIN32_WINNT_WIN10_RS2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(12232): warning C4668: '_WIN32_WINNT_WIN10_RS2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(12249): warning C4668: '_WIN32_WINNT_WIN10_RS3'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(12266): warning C4668: '_WIN32_WINNT_WIN10_RS3'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(12285): warning C4668: '_WIN32_WINNT_WIN10_RS2'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(12404): warning C4668: '_WIN32_WINNT_WIN10_RS3'은(는) 전처리기 매크로로 정의되어 있지 않음
s:\10\include\10.0.16299.0\us\winioc\h(35): warning C4820: 'timespec64': '4'바이트 채움 문자가 데이터 멤버 'tv_nsec' 뒤에 추가되었음
s:\10\include\10.0.16299.0\us\winioc\h(42): warning C4820: 'timespec': '4'바이트 채움 문자가 데이터 멤버 'tv_nsec' 뒤에 추가되었음
sgame\소스.c(52): warning C4255: 'InitMap': 함수 프로토타입을 입력하지 않았습니다. '()'에서 '(void)'로 변환됩니다.
sgame\소스.c(64): warning C4255: 'DrawMap': 함수 프로토타입을 입력하지 않았습니다. '()'에서 '(void)'로 변환됩니다.
sgame\소스.c(117): warning C4255: 'GetInput': 함수 프로토타입을 입력하지 않았습니다. '()'에서 '(void)'로 변환됩니다.
sgame\소스.c(126): warning C4255: 'EatFood': 함수 프로토타입을 입력하지 않았습니다. '()'에서 '(void)'로 변환됩니다.
sgame\소스.c(170): warning C4255: 'main': 함수 프로토타입을 입력하지 않았습니다. '()'에서 '(void)'로 변환됩니다.
sgame\소스.c(173): warning C4242: '합수': 'time_t'에서 'unsigned int'('오')로 변환하면서 데이터가 손실될 수 있습니다.
s:\10\include\10.0.16299.0\us\winioc\h(946): warning C4710: 'int printf(const char *const, ...)': 함수를 인라인화하지 못했습니다.
s:\10\include\10.0.16299.0\us\winioc\h(946): note: 'printf' 선언을 참조하십시오.
sgame\소스.c(114): warning C4715: 'DrawMap': 모든 제어 경로에서 값을 반환하지는 않습니다.
s:\10\source\repos\SnakeGame\Debug\SnakeGame.exe
```

[그림 3.1] Warning Level 빌드 오류



1) 함수 프로토타입 오류 개선 (warning C4255)

매개변수가 존재하지 않는 함수들은 반드시 함수 프로토타입을 void로 작성해 주어야 한다.

<pre>void GetInput() { /**/ if (_kbhit()) { direction = _getch(); } }</pre>	<pre>void GetInput(void) { int getchar; if (_kbhit() != 0) { if (_getch() == DIR_KEY) { getchar = _getch(); } if (IsOppositeDir(getchar)) { return; } else if (IsSameDir(getchar)) { return; } else { Head->dir = getchar; MoveSnake(Head->dir); } } }</pre>
[그림 3.2] 기존 코드 오류	[그림 3.3] 코드 개선 후

2) int 형 데이터 손실 오류 개선 (warning C4242)

time()함수가 unsigned int로 무리하게 변환되면서 데이터의 손실을 가져올 수 있으므로 이에 유의하여 미리 unsigned int형으로 직접 변환시켜 준다.



 <code>srand(time(0));</code>	 <code>srand((unsigned int)time(0));</code>
[그림 3.4] 기존 코드 오류	[그림 3.5] 코드 개선 후

3) DrawMap()함수 경로 별 return값 오류 개선 (warning C4715)

DrawMap()함수에서 Switch문이 사용되었는데, 모든 case를 작성한 후, default문을 작성하지 않아 default의 경우에 리턴값이 없어 모든 경우의 리턴값이 없다고 오류가 나타났다. 그러므로 Switch문 마지막에는 default문과 그 리턴값을 작성하여 오류를 개선하였다.

3-2-2. 조건문 if-else문 예외 처리 오류 개선

조건문 작성 시, if문의 처리 후 else문이 없다면 예외 처리에 실패할 수 있기 때문에 반드시 if문에는 else문을 작성하여 예외 처리를 진행하여야 한다.

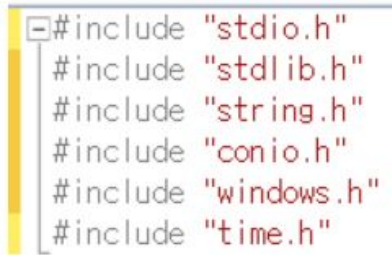
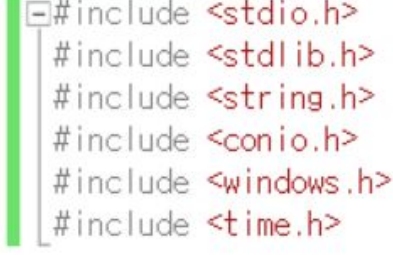
 <pre>if (hits == 5) { wait = 200; speed++; } else if (hits == 7) { wait = 100; speed++; } else if (hits == 9) { wait = 0; speed++; }</pre>	 <pre>void ControlSpeed(void) { if (score == 5) { wait = 100; speed++; return; } else if (score == 7) { wait = 80; speed++; return; } else if (score == 9) { wait = 50; speed++; return; } else { return; } }</pre>
[그림 3.6] 기존 코드 오류	[그림 3.7] 코드 개선 후

3-2-3. 반복문의 return 처리 오류 개선

조건문과 마찬가지로 반복문에서도 항상 return을 잘 활용하여 무한루프에 빠지지 않게 하여야 하고, 함수일 경우에는 return타입에 맞게 return값을 잘 정의해 주어야 한다.

3-2-4. 헤더파일 작성 방식 오류 개선

해당 Snake Game 오픈소스 코드는 헤더파일 작성 방식에 오류가 있다. 헤더파일을 include할 때, 사용자 지정 헤더가 아니므로 “”(큰따옴표)가 아닌 <>로 작성해 주어야 한다.

 <pre>#include "stdio.h" #include "stdlib.h" #include "string.h" #include "conio.h" #include "windows.h" #include "time.h"</pre>	 <pre>#include <stdio.h> #include <stdlib.h> #include <string.h> #include <conio.h> #include <windows.h> #include <time.h></pre>
[그림 3.8] 기존 코드 오류	[그림 3.9] 코드 개선 후

3-3. 가독성을 기준으로 clean 코드로 개선하기

3-3-1. 같은 의미를 갖는 변수들을 구조체로 선언하기

Xpos변수, Ypos변수는 뱀의 위치를 나타내고, direction은 뱀의 방향을 나타내므로 뱀을 표현하는 구조체 안의 멤버변수로 만들어 주었다. Xfood변수와 Yfood변수 역시 먹이의 위치를 나타내는 함수이므로, 뱀을 표현하는 구조체를 선언하여 구조체 안의 멤버 변수로 적용 시켰다.

 <pre>static int Xpos; static int Ypos; static int Xfood; static int Yfood; static int direction = 80; //down static int hits = 0; static int wait = 500; static int lives = 3; static int speed = 1;</pre>	 <pre>typedef struct Snake { COORD pos; int dir; struct Snake * pre; struct Snake * next; }Snake; typedef struct Food { COORD pos; }Food;</pre>
[그림 3.10] 기존 코드	[그림 3.11] 코드 개선 후

3-3-2. 전역 변수의 사용을 줄이기

맵을 그릴 때만 사용하면 되는 MAP 배열을 전역변수로 선언하기 보단, 맵을 그리는 역할을 하는 drawMap()함수안에 지역변수로 작성해 주었다.

<pre>static char HEADER[4][25] = { "-----", " A SNAKE GAME ", " ***** ", "-----", }; static char FOOTER[25] = { "-----" }; static char MAP[15][25] = { " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " "</pre>	<pre>int DrawMap() { char HEADER[4][25] = { "-----", " A SNAKE GAME ", " ***** ", "-----", }; char FOOTER[25] = { "-----" }; char MAP[15][25] = { " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " " }; switch (direction) { case UP: if (Ypos <= 0) return 0; else Ypos--; break; case DOWN: if (Ypos >= 9) return 0; else Ypos++; break; case LEFT: if (Xpos <= 1) return 0; else</pre>
<p>[그림 3.12] 기존 코드</p>	<p>[그림 3.13] 코드 개선 후</p>

3-3-3. 하나의 함수에는 하나의 기능을 하도록 설계하기

EatFood()함수를 각각의 기능에 맞게 EatFood(), ControlSpeed(), RelocateFood()로 분해하였다.

<pre> void EatFood() { if (Ypos == Yfood && Xpos == Xfood) { hits++; Xfood = (rand() % 20) + 1; Yfood = (rand() % 8) + 1; MAP[Yfood][Xfood] = 'o'; if (hits == 5) { wait = 200; speed++; } else if (hits == 7) { wait = 100; speed++; } else if (hits == 9) { wait = 0; speed++; } } } </pre>	<pre> void EatFood(void) { if (Head->pos.Y == food.pos.Y && Head->pos.X == food.pos.X) { score++; AddTail(); RelocateFood(); ControlSpeed(); } } void ControlSpeed(void) { if (score == 5) { wait = 100; speed++; } else if (score == 7) { wait = 80; speed++; } else if (score == 9) { wait = 50; speed++; } } void RelocateFood(void) { food.pos.X = (rand() % (BOARD_WIDTH - 2)) + LEFT_BORDER + 1; food.pos.Y = (rand() % (BOARD_HEIGHT - 2)) + TOP_BORDER + 1; DrawFood(); } </pre>
<p>[그림 3.14] 기존 코드</p>	<p>[그림 3.15] 코드 개선 후</p>

3-3-4. 의미있는 상수를 매직넘버화 하기

<pre> void InitMap() { Xpos = (rand() % 20) + 1; Ypos = (rand() % 8) + 1; Xfood = (rand() % 20) + 1; Yfood = (rand() % 8) + 1; MAP[Yfood][Xfood] = 'o'; } </pre>	<pre> /* Size of GameBoard */ #define BOARD_WIDTH 45 #define BOARD_HEIGHT 23 /* Game Border */ #define TOP_BORDER 5 #define BOTTOM_BORDER 27 #define LEFT_BORDER 5 #define RIGHT_BORDER 49 void RelocateFood(void) { food.pos.X = (rand() % (BOARD_WIDTH - 2)) + LEFT_BORDER + 1; food.pos.Y = (rand() % (BOARD_HEIGHT - 2)) + TOP_BORDER + 1; DrawFood(); } </pre>
[그림 3.16] 기존 코드	[그림 3.17] 코드 개선 후

3-3-5. 조건식의 함수화 하기1

조건식을 변수로 표현하기보단 함수로 표현하여 코드의 가독성을 높인다.

<pre> while (lives) { Die(DrawMap()); GetInput(); EatFood(); system("cls"); } </pre>	<pre> bool isAlive() { return(lives > 0); } while (isAlive()) { Die(DrawMap()); GetInput(); EatFood(); system("cls"); } </pre>
[그림 3.18] 기존 코드	[그림 3.19] 코드 개선 후

3-3-6. 조건식의 함수화 하기2

조건식을 비교연산자로 표현하기 보단 함수로 표현하여 코드의 가독성을 높인다.

<pre> switch (direction) { case UP: if (Ypos <= 0) return 0; else Ypos--; break; case DOWN: if (Ypos >= 9) return 0; else Ypos++; break; case LEFT: if (Xpos <= 1) return 0; else Xpos--; break; case RIGHT: if (Xpos >= 20) return 0; else Xpos++; break; } </pre>	<pre> bool IsCollisionTopBorder(void) { return (Head->pos.Y <= TOP_BORDER); } bool IsCollisionBottomBorder(void) { return (Head->pos.Y >= BOTTOM_BORDER); } bool IsCollisionLeftBorder(void) { return (Head->pos.X <= LEFT_BORDER); } bool IsCollisionRightBorder(void) { return (Head->pos.X >= RIGHT_BORDER); } switch (Head->dir) { case UP: Head->pos.Y--; if (IsCollisionTopBorder() IsCollisionBody()){ return 0; } else{ return 1; } break; case DOWN: Head->pos.Y++; if (IsCollisionBottomBorder() IsCollisionBody()){ return 0; } else{ return 1; } break; case LEFT: Head->pos.X--; if (IsCollisionLeftBorder() IsCollisionBody()){ return 0; } else{ return 1; } break; } </pre>
<p>[그림 3.20] 기존 코드</p>	<p>[그림 3.21] 코드 개선 후</p>

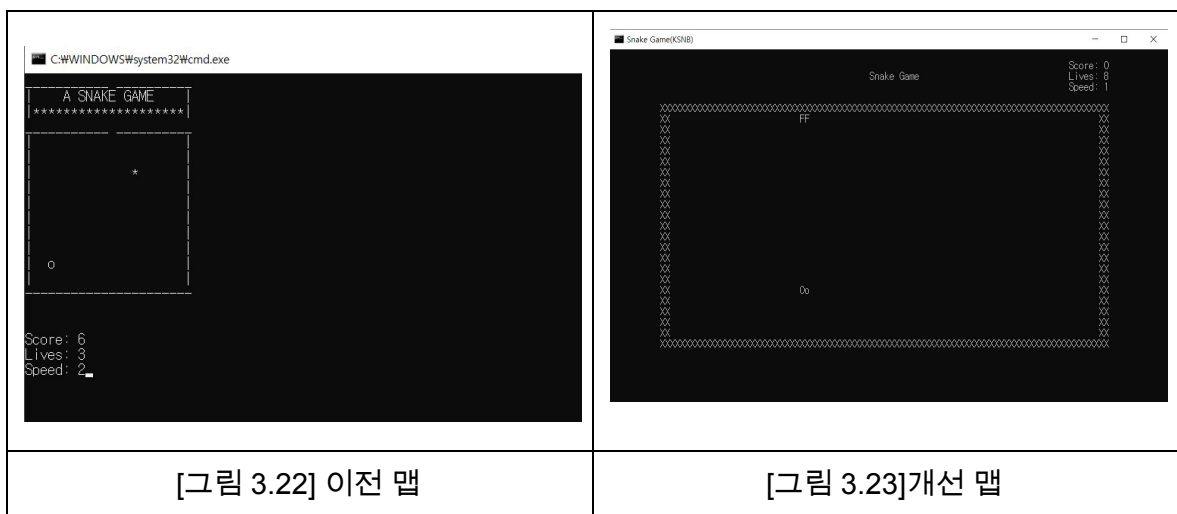
3-4. 기능 추가 및 기능 오류 개선하기

3-4-1. 맵 크기 변경하기

1) 내용

기존의 맵은 너무 작았고, 배열화 되어 있지 않기 때문에 맵을 그리는데 비효율적이었다. 또한 맵이 너무 작았던 탓에 뱀의 속도를 빠르게 하는 데에 어느 정도 제한이 있었다. 그러므로 맵의 크기를 변경하고자 하였다.

2) 비교 (크기 차이)



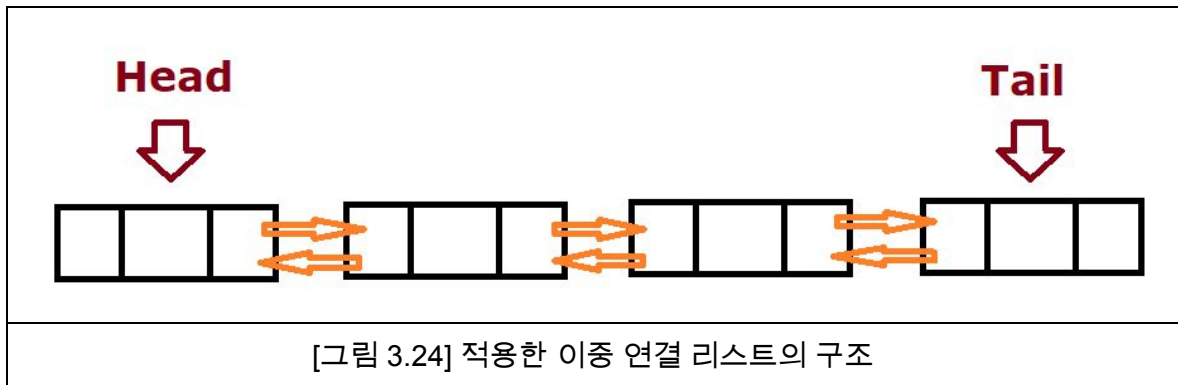
3-4-2. 꼬리 생성 기능 추가하기

1) 내용

fork 해 온 기존의 코드에서는 꼬리가 캐릭터에 직접적으로 생성되지 않았고, 단지 점수가 증가함에 따라 꼬리가 늘어난다는 것을 알 수 있었다. 하지만 고전의 Snake Game은 먹이를 먹을 때 마다 꼬리가 계속해서 증가하고 뱀의 길이가 길어져 벽에 부딪히거나 자신의 몸과 부딪히면 죽는다는 룰을 반드시 적용시킨다. 그러므로 fork 해 온 Snake Game 오픈소스 코드를 수정하여 꼬리가 눈에 보이도록 생성하는 기능을 추가하고자 하였다.

2) 사용한 자료구조

뱀의 꼬리가 먹이를 먹으면 늘어나고, 아이템을 먹으면 줄어든다는 특징이 있기 때문에 뱀을 표현하기 위한 자료구조에 대한 첫번째 아이디어는 “스택”이었다. 그러나 스택은 메모리의 할당부분에서 애초에 많은 메모리를 할당해야 했기 때문에 적합하지 않은 자료구조였다. 이후 메모리에 대한 낭비를 신경쓰지 않아도 되는 “단일 연결리스트”를 자료구조로 선택하여 진행하고자 하였지만, 꼬리를 삭제하기 위해서는 이중포문이 돌아가야 했기 때문에 가독성의 측면과 효율성의 부분에서 적합하지 않은 자료구조였다. 따라서 최종적으로 선택한 자료구조는 “이중 연결 리스트”로, 해당 프로젝트 범위인 가독성의 측면에서 매우 적합하고 코드의 효율성 또한 높아 우리가 개발하고자 하는 Snake Game에 가장 적합한 자료구조라고 생각하였다.



3) 자료구조 사용 방법

Head와 Tail 포인터를 선언 해 준 뒤 뱀의 꼬리가 하나씩 증가 할 때마다 노드를 할당받아 와 Tail의 next에 계속 연결시켜 준다.

3-4-3. 뱀을 정반대 방향으로의 움직이지 못하게 하기

기존 Snake Game 과 마찬가지로 뱀이 현재 움직이고 있는 방향의 정 반대 방향의 방향키를 누른다면 그 키는 적용되면 안된다. 따라서 해당 문제를 IsOppositeDir()함수와 IsSameDir()함수를 사용하여 해결하였다.

3-4-4. 사용한 메모리 해제하기

이중연결리스트를 사용하다 보니 게임이 종료된 후에 그냥 게임을 닫아버리면 동적할당 받은 메모리는 여전히 남아있다. 그러므로 반드시 더이상 쓰지 않는 메모리는 free를 사용하여 해제해주어야 한다. 따라서 해당 코드에서는 FreeAllMemories()의 함수를 통해 사용하였던 메모리를 전부 해제시켜 종료 전에 시행하게 하였다.

3-4-5. 방향키 이외의 키 입력 못하게 하기

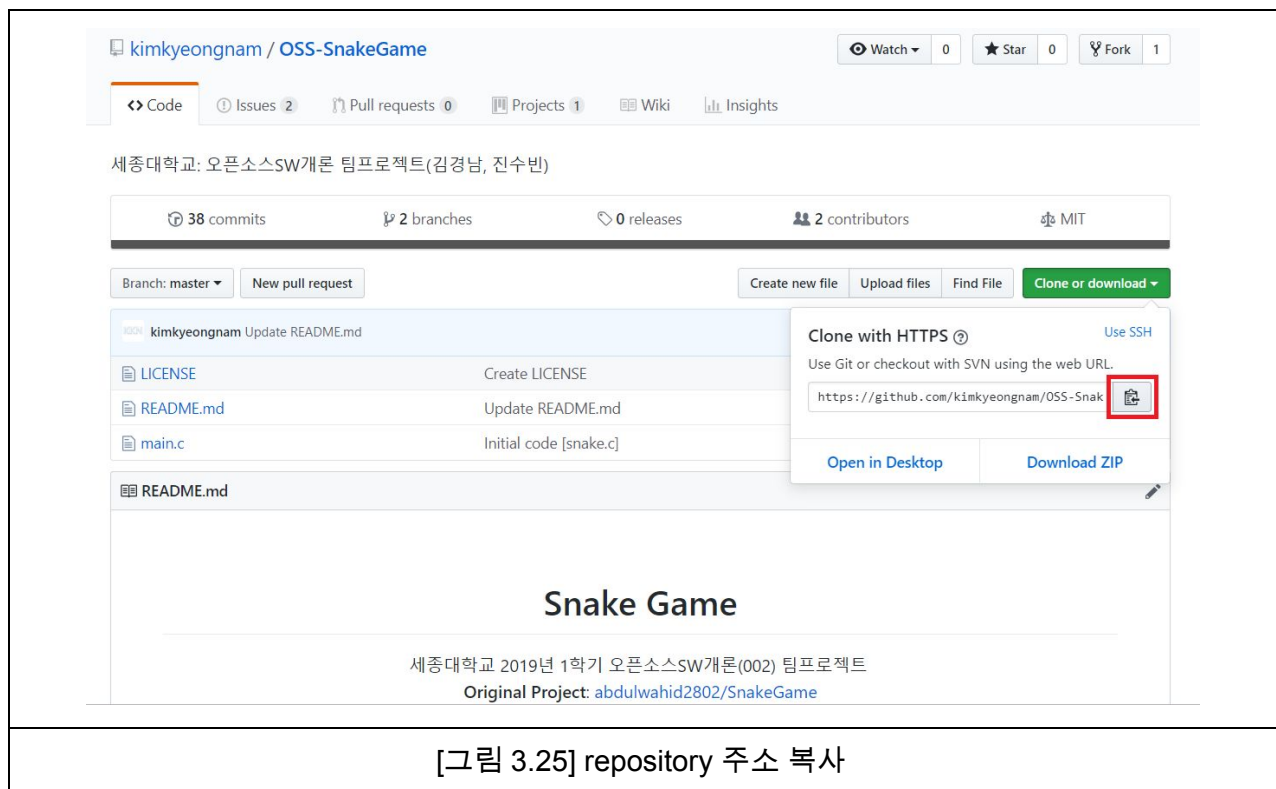
DIR_KEY 224를 define 해 주어 방향키 이외의 키를 입력 못하도록 제어 하였다.

3-4-6. Intro 적용하기

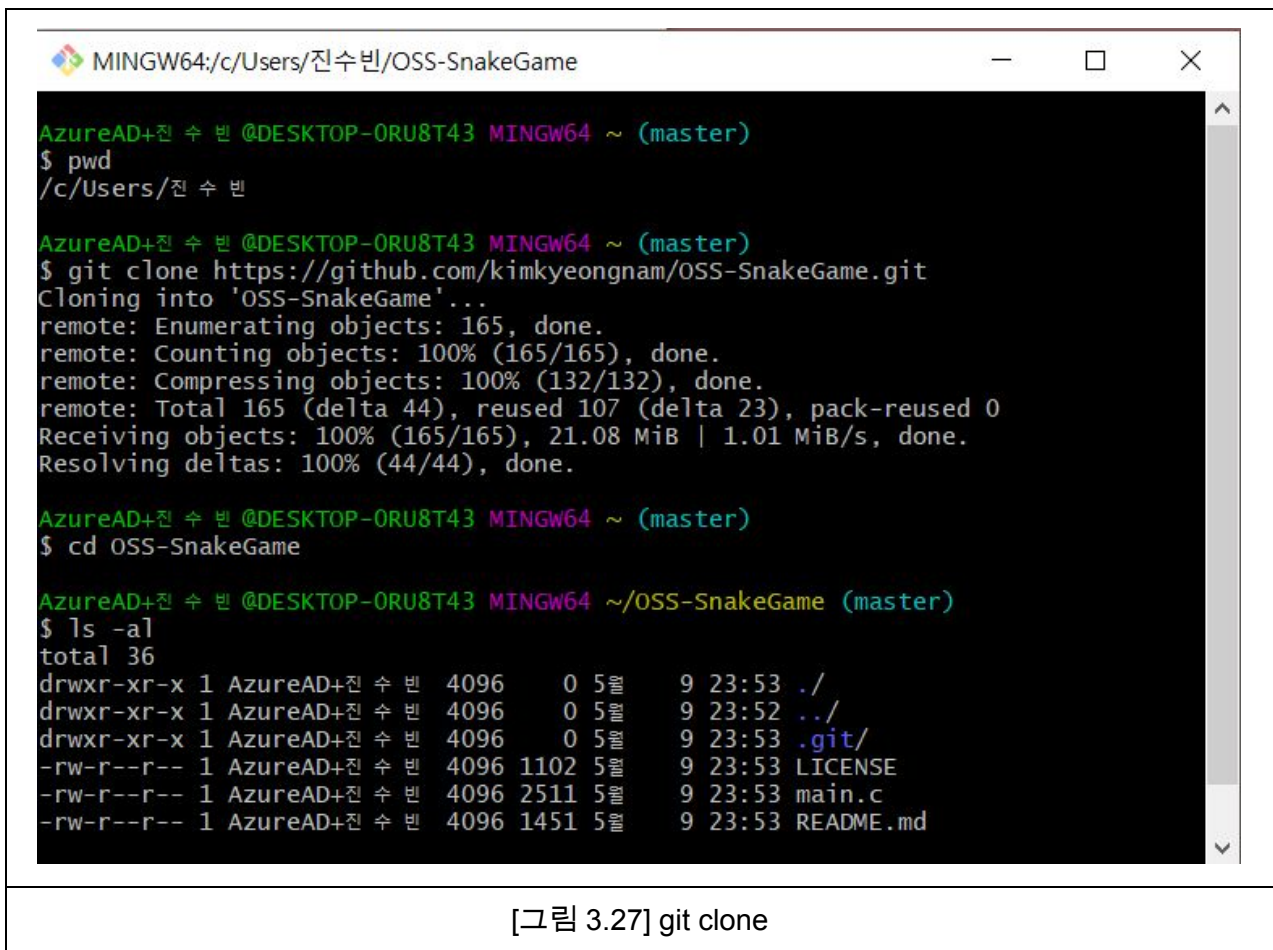
기존코드에는 Intro가 없었으므로 게임시작과 종료를 선택할 수 있는 인트로를 생성하였다.

3-5. Git을 이용하기

3-5-1. git clone



[그림 3.25]와 같이 팀장이 새로 생성한 repository에 들어가 해당 repository의 주소를 복사한다.

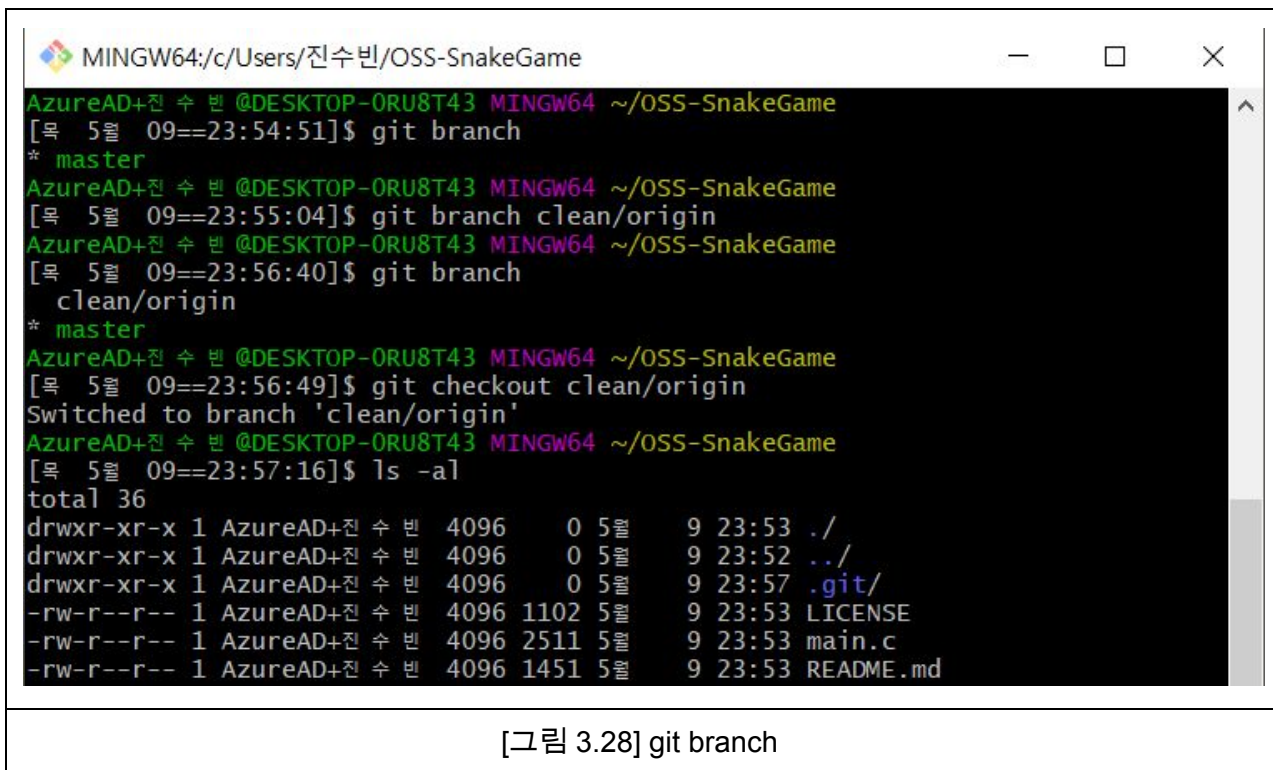


[그림 3.27] git clone

git bash를 열고 pwd명령어를 입력해 현재 경로를 확인한다. 이때, cd명령어를 통해 원하는 경로로 이동할 수 있다. 원하는 경로로 이동했다면 git clone “repository_주소” 를 입력하여 로컬저장소에 원격저장소의 코드들을 가져오게 된다. 여기서 cd “repository_이름” 을 통해 해당 파일로 이동하고 숨겨져있는 모든 파일까지 다 보여주는 ls -al이라는 명령어를 통해 .git이라는 파일이 있는지 확인하면 로컬저장소가 잘 생성된 것이다.

3-5-2. git branch & git checkout

이제부터 본격적으로 git을 사용할 것이다. git branch “원하는_브랜치_이름” 명령어를 입력하여 브랜치를 생성한다. git branch 라는 명령어를 통해 로컬저장소의 모든 브랜치와 현재 위치해 있는 브랜치의 정보 등을 알 수 있다. 이후 git checkout “이동하고싶은_브랜치_이름” 을 입력하여 원하는 브랜치로 이동한다. 이동한 브랜치에서 역시 ls -al 명령어를 통해 브랜치에 존재하는 파일들을 볼 수 있다.

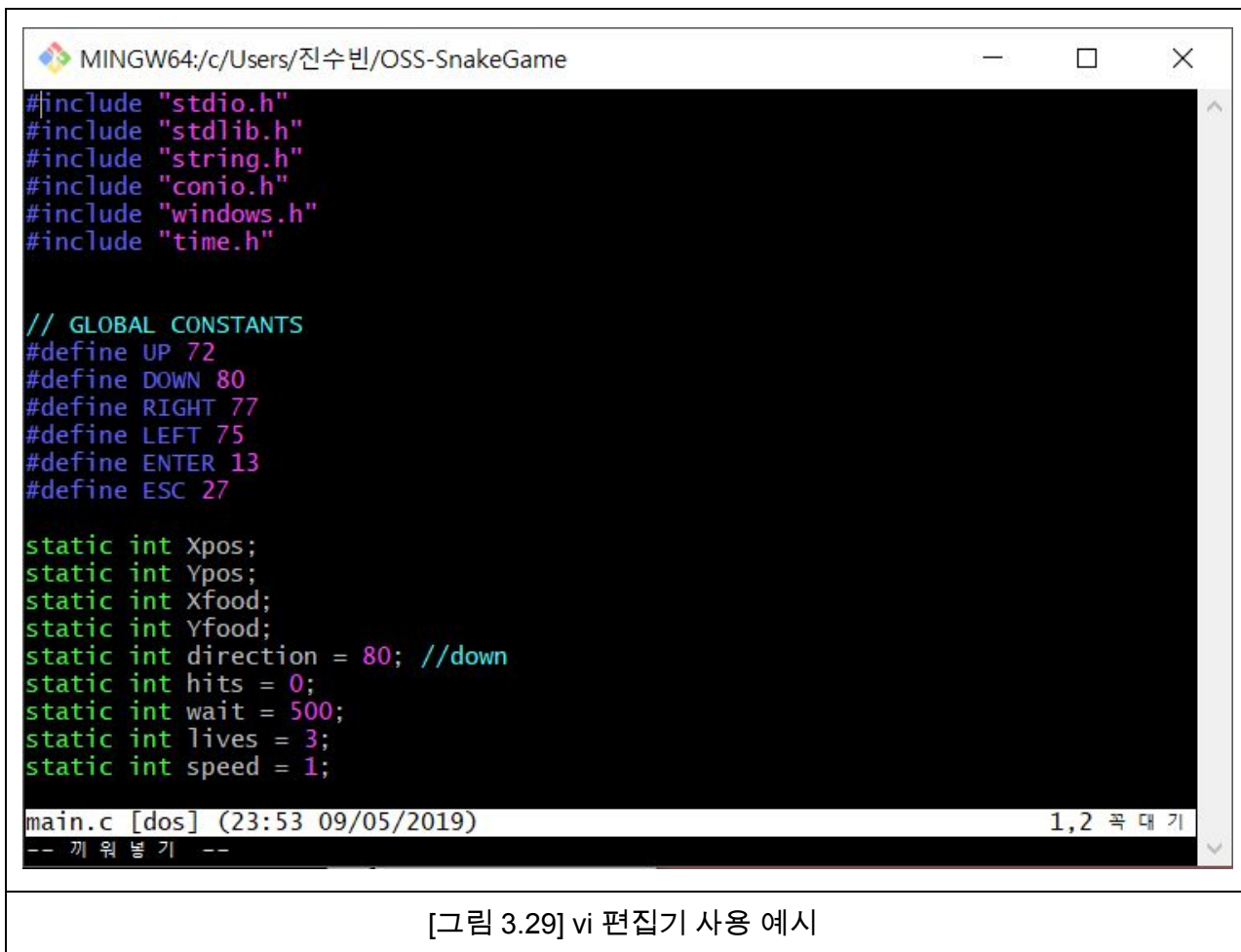


3-5-3. vi 편집기

브랜치의 작업이 끝났다면 본격적으로 코딩을 시작하여야 한다. vi “열고싶은 파일명.확장명”을 입력한 후 해당코드를 수정할 수 있다. vi에서는 컴파일 할 수 없기 때문에 여기서 우리는 vi에 열린 코드를 복사하여 Visual Studio2017 환경에서 컴파일 한 후 다시 vi 창으로 붙여넣기 하였다.

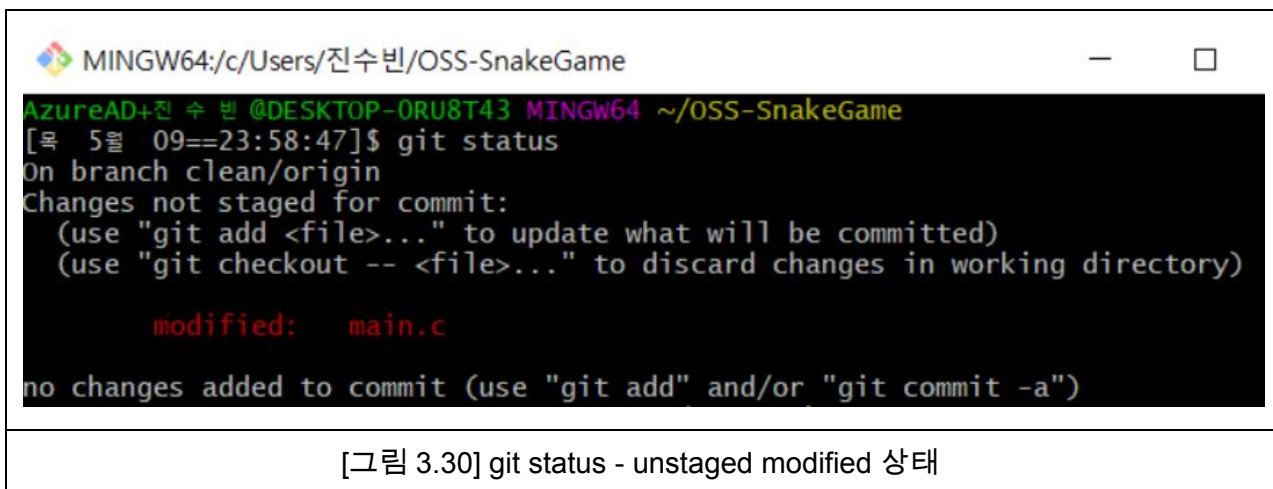
vi에서 사용한 주된 명령어는 다음과 같다.

- 명령모드에서 a : 입력모드로 전환 (현재 커서 다음자리 입력)
- 명령모드에서 Shift +v +g : 전체선택
- 명령모드에서 y : 복사
- 명령모드에서 p : 붙여넣기
- 명령모드에서 d : 삭제
- 마지막 행 모드에서 :wq! : 저장 후 마치고
- 마지막 행 모드에서 :q! : 그냥 마치고



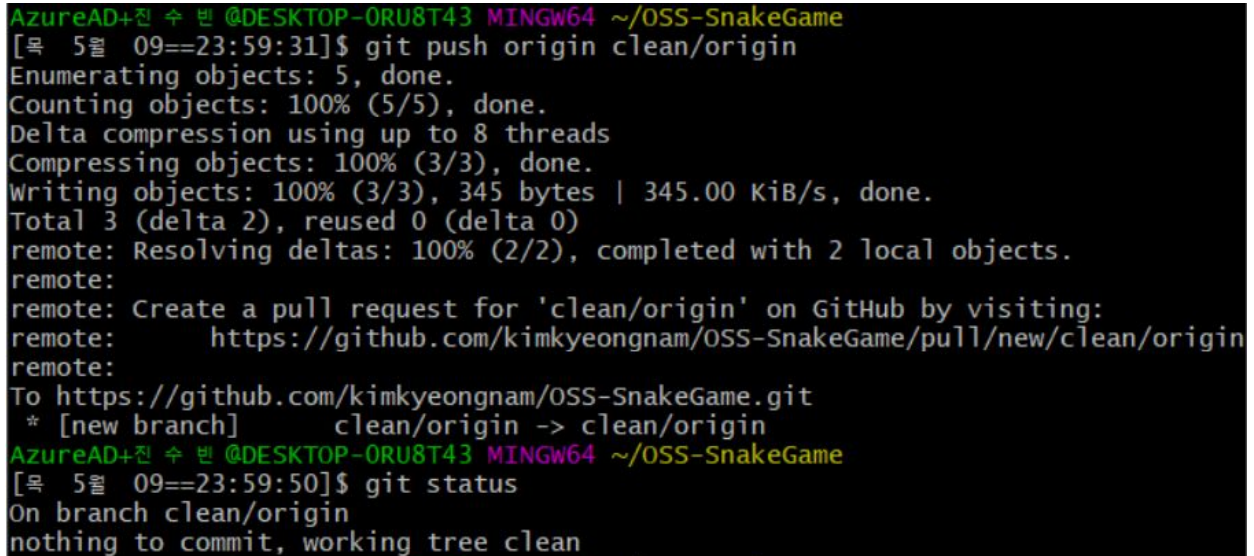
3-5-5. git status

git status 명령어를 통해 파일의 상태를 알 수 있다. untracked, modified, staged, committed의 상태를 알 수 있다.



3-5-8. git push

git push는 로컬저장소에서 원격저장소로 저장하는 명령어이다. 그러므로 매우 신중하게 사용하여야 한다. 만약 branch에만 push를 원한다면 git push 해서는 안되고, [그림 3.33] 처럼 git push origin “브랜치명”을 입력하여야 한다. 여기서 origin은 로컬 저장소를 의미한다.

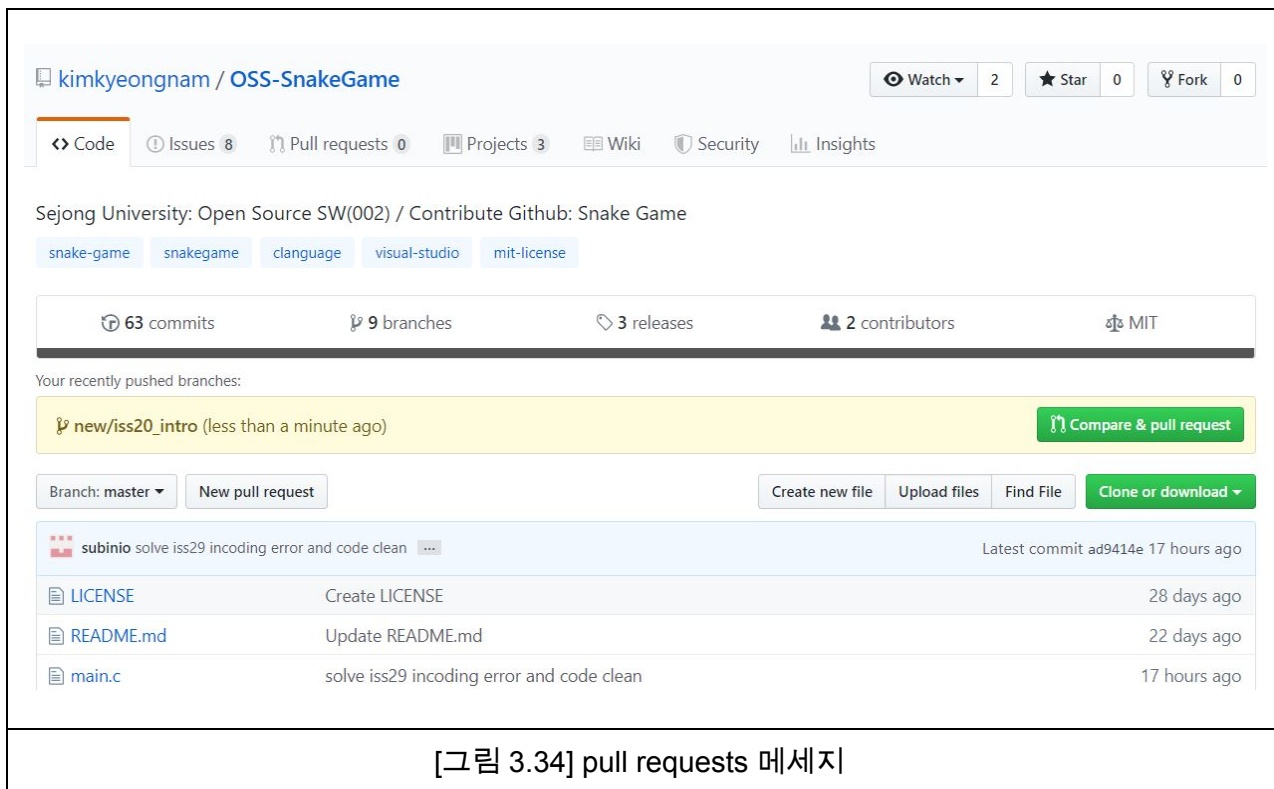


```
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[목 5월 09==23:59:31]$ git push origin clean/origin
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 345 bytes | 345.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'clean/origin' on GitHub by visiting:
remote:   https://github.com/kimyeongnam/OSS-SnakeGame/pull/new/clean/origin
remote:
To https://github.com/kimyeongnam/OSS-SnakeGame.git
 * [new branch]      clean/origin -> clean/origin
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[목 5월 09==23:59:50]$ git status
On branch clean/origin
nothing to commit, working tree clean
```

[그림 3.33] git push origin 브랜치

3-5-9. git merge

git merge란 브랜치끼리의 통합을 말하며 각자가 개발하던 브랜치를 마스터 브랜치에 통합하고 싶을 때도 사용한다. 위의 그림처럼 git push를 브랜치에 했다면 원격저장소인 github에서는 [그림 3.34]와 같은 pull requests 메시지가 뜬다. 그럼 해당 버튼을 클릭하고, 팀원들의 피드백을 기다린다.



팀원들의 과반수 이상의 코드리뷰를 듣고 merge를 시작한다. merge는 로컬 저장소에서 실행이 가능하므로 git bash에서 진행한다. 이 경우에는 a라는 브랜치를 master 브랜치에 merge하여야 하므로 우선 master 브랜치로 checkout 해주어야 한다. 그 이후 master 브랜치에서 git merge a(머지할 브랜치 이름)을 입력하면 [그림 3.35]와 같이 머지에 성공한다. 마지막으로 git push를 master에 실행시키면 자신의 코드가 통합된 코드가 원격저장소에 올라가게 된다.


```
MINGW64:/c/Users/진수빈
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 10==01:23:50]$ git branch
* clean/origin
  master
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 10==01:23:59]$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 10==01:24:12]$ git branch
  clean/origin
* master
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 10==01:24:15]$ git pull
Already up to date.
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 10==01:27:36]$ git merge clean/origin
Merge made by the 'recursive' strategy.
  main.c | 106 ++++++
  1 file changed, 62 insertions(+), 44 deletions(-)
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 10==01:29:04]$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
AzureAD+진 수 빈 @DESKTOP-0RU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 10==01:29:17]$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 280 bytes | 280.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/kimkyeongnam/OSS-SnakeGame.git
  0d0a891..78c4533  master -> master
```

[그림 3.35] git merge

3-5-10. git fetch

git fetch란 로컬 저장소를 원격저장소와 현재 시점을 기준으로 동기화 하는 명령어로 협업에 있어 매우 중요하다. git fetch는 로컬 컴퓨터에서 자신의 작업을 시작하기 전에, push하기 전에 반드시 시행해 merge 시 충돌을 최소화 해주어야 한다. git fetch 순서는 다음과 같다. 우선 git remote -v 명령어를 통해 로컬저장소와 원격 저장소의 연결을 확인하고, git fetch --all 명령어를 통해 모든 파일에 대한 fetch를 실행한다. git branch -vv를 입력하여 추적상황을 확인하고 마지막으로 merge시켜 로컬저장소와 원격저장소를 동일 시 할 수 있다.

```

AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[월 5월 19==21:56:25]$ git remote -v
origin https://github.com/kimkyeongnam/OSS-SnakeGame.git (fetch)
origin https://github.com/kimkyeongnam/OSS-SnakeGame.git (push)
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[월 5월 19==21:56:43]$ git fetch --all
Fetching origin
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 15 (delta 5), reused 6 (delta 2), pack-reused 6
Unpacking objects: 100% (15/15), done.
From https://github.com/kimkyeongnam/OSS-SnakeGame
  78c4533..7a5237e  master      -> origin/master
* [new branch]      clean/origin-kn -> origin/clean/origin-kn
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[월 5월 19==21:56:57]$ git branch -vv
  clean/origin 8378969 modify global variable
* master       78c4533 [origin/master: behind 5] Merge branch 'clean/origin'
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[월 5월 19==21:57:09]$ git merge origin/master
Updating 78c4533..7a5237e
Fast-forward
 main.c | 193 ++++++
 1 file changed, 116 insertions(+), 77 deletions(-)

```

[그림 3.36] git fetch

3-5-11. git stash

git stash란 commit 하지 않고 나중에 다시 돌아와 작업을 진행하는 경우에 사용하는 명령어이다. 예를 들어 새로운 브랜치에서 새로운 기능을 개발하고 있던 도중 급하게 기존의 버그 문제를 해결해야 한다면 하던 개발을 저장하지 않고 나가거나, 개발이 완료되지 않은 상태에서 commit하여야 한다. 이런 경우, 개발을 멈추고 해당 파일을 git add명령어를 통해 staged 상태를 만든 후 git stash 명령을 사용하면 커밋하지 않아도 작성하고 있던 코드가 저장된다. 따라서 그 이후에는 브랜치를 옮겨 다른 작업을 진행해도 상관이 없다. 다시 stash했던 브랜치로 돌아가 git stash list 명령을 실행하면 된다.

```
MINGW64/c/Users/진수빈
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:08:25]$ git checkout new/iss20_intro
Switched to branch 'new/iss20_intro'
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:09:31]$ git branch
  bug/iss29_draw
  clean/origin
  master
  new/iss17_tail
* new/iss20_intro
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:09:37]$ vi main.c
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:11:18]$ git status
On branch new/iss20_intro
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   main.c

no changes added to commit (use "git add" and/or "git commit -a")
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:11:23]$ git add main.c
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:11:56]$ git stash
Saved working directory and index state WIP on iss20_intro: ad9414e solve iss29
incoding error and code clean
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:16:37]$ git status
On branch new/iss20_intro
nothing to commit, working tree clean
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:17:03]$ git checkout new/iss17_tail
Switched to branch 'new/iss17_tail'
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:17:56]$ git branch
  bug/iss29_draw
  clean/origin
  master
* new/iss17_tail
  new/iss20_intro
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:17:59]$ vi main.c
AzureAD+진 수 빈 @DESKTOP-ORU8T43 MINGW64 ~/OSS-SnakeGame
[금 5월 31==15:18:21]$ |
```

[그림 3.37] git stash

3-5-12. git tag

태그는 소프트웨어 버전을 릴리즈할 때 매우 용이하다. 변경 이력의 주요 지점이 표식을 남김으로써 해당 버전의 소스코드를 얻을 수 있다. git tag -a “태그명” “commit 번호 앞의 7자리”를 입력하면 해당 commit에 설정했던 태그명으로 태그가 생성된다. git show “태그명”을 통해 해당 태그의 commit 히스토리를 볼 수 있다. 태그를 원격저장소에 올리려면 git push --tags를 입력하면 된다.

```
[목 5월 30==21:50:50]$git tag -a v0.0.0 92ced4d
[목 5월 30==21:52:52]$git tag -a v0.1.0 1957777
[목 5월 30==21:53:55]$git tag -a v1.14.0 28ff573
[목 5월 30==21:54:35]$git tag
v0.0.0
v0.1.0
v1.14.0
```

[그림 3.38] tag 추가

```
[목 5월 30==21:54:49]$git show v0.0.0
tag v0.0.0
Tagger: kimkyeongnam <kkyy0126@naver.com>
Date: Thu May 30 21:52:32 2019 +0900

Upload original code

commit 92ced4d11b50db8a4d766b4f6a5614e86c1b3b5f (tag: v0.0.0)
Author: kimkyeongnam <kkyy0126@naver.com>
Date: Mon Apr 15 22:10:36 2019 +0900

    Initial code [snake.c]

diff --git a/.gitignore b/.gitignore
deleted file mode 100644
index bafe145..0000000
--- a/.gitignore
+++ /dev/null
@@ -1,2 +0,0 @@
-*.exe
-
diff --git a/README.md b/README.md
deleted file mode 100644
index 1a2b3c4..0000000
```

[그림 3.39] tag 기록 확인

```
[목 5월 30==21:55:43]$git push --tags
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 457 bytes | 152.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/kimkyeongnam/OSS-SnakeGame.git
 * [new tag]          v0.0.0 -> v0.0.0
 * [new tag]          v0.1.0 -> v0.1.0
 * [new tag]          v1.14.0 -> v1.14.0
```

[그림 3.40] tag 업로드

3-6. GitHub을 이용하기

3-6-1. Issue 활용하기

Issue를 활용하여 팀원간의 지켜야 할 규칙이나 알아두어야 할 점을 기록한다. 또한 협업을 하면서 풀리지 않는 문제나 개선점 등, 프로젝트를 하면서 필요한 모든 내용에 대해 의논해야할 경우에도 Issue를 통하여 자유롭게 의견을 주고받을 수 있다.

Issue에는 Label과 Milestone이라는 기능을 제공하고 있어 프로젝트 관리를 좀 더 용이하게 할 수 있다.

The screenshot shows the GitHub interface for the repository 'kimkyeongnam / OSS-SnakeGame'. The 'Issues' tab is selected, showing 8 open issues. The issues are listed with their titles, labels, and creation dates. The labels used include 'FAQ', 'New', 'Bug', 'Improvement', 'Documentation', and 'Rule'. A 'ProTip' at the bottom suggests using checkboxes to edit multiple issues at once.

Issue Title	Labels	Created	Comments
FAQ	FAQ	#38 opened a day ago by kimkyeongnam	0
[기능추가]items	New	#36 opened a day ago by subinio	1
[Ver.] 태그(버전) 작성	Bug, Improvement, New, Tag	#33 opened 2 days ago by kimkyeongnam	3
[문서화] Wiki	Documentation	#32 opened 2 days ago by kimkyeongnam	4
[문서화] Readme	Documentation	#30 opened 2 days ago by kimkyeongnam	2
[기능 추가] Outro 화면	New	#22 opened 3 days ago by kimkyeongnam	2
[기능 추가] Intro 화면	New	#20 opened 3 days ago by kimkyeongnam	3
프로젝트 수행 시 필요한 규칙	Rule	#4 opened on 15 Apr by kimkyeongnam	8

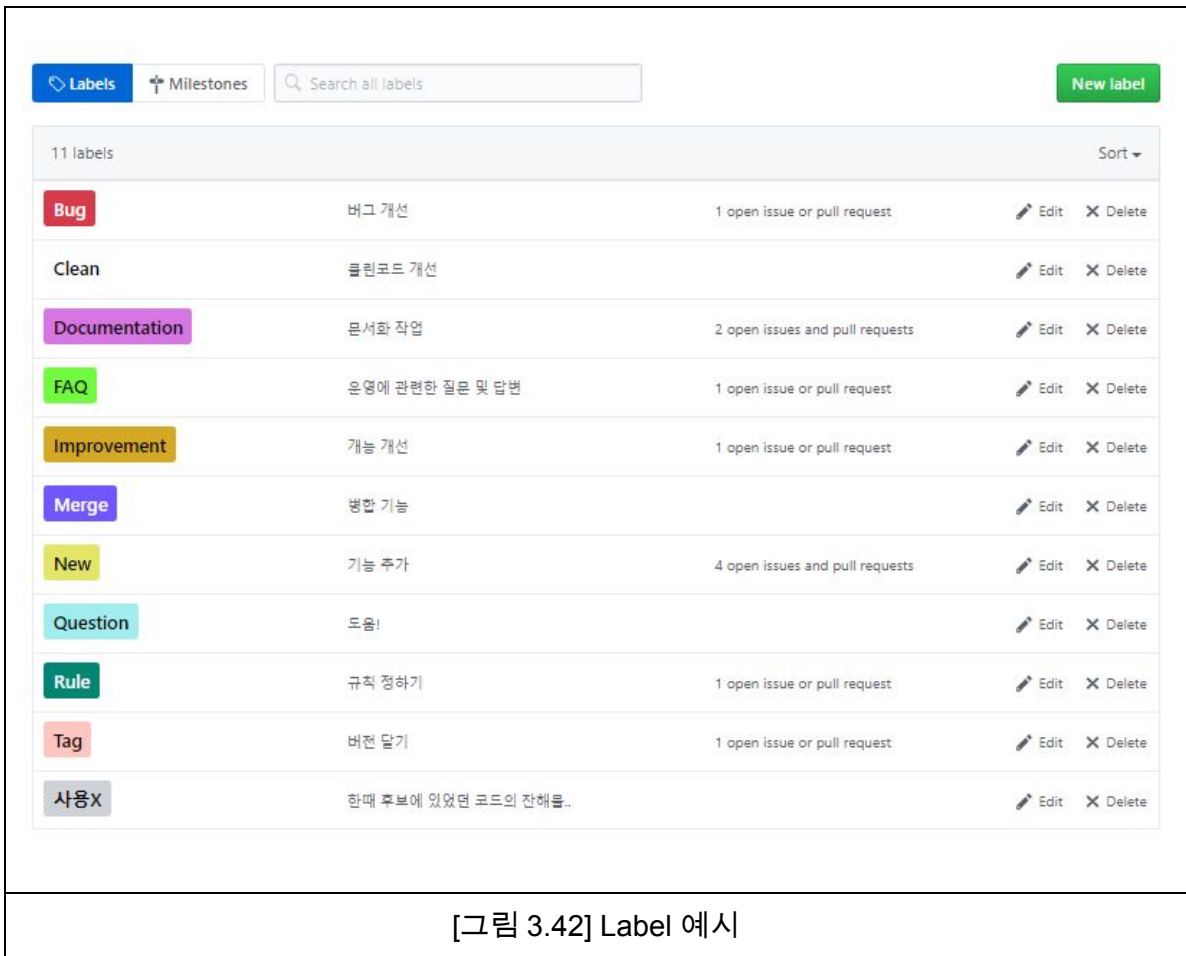
ProTip! Click a checkbox on the left to edit multiple issues at once.

[그림 3.41] Issue 활용 예시

1) Label

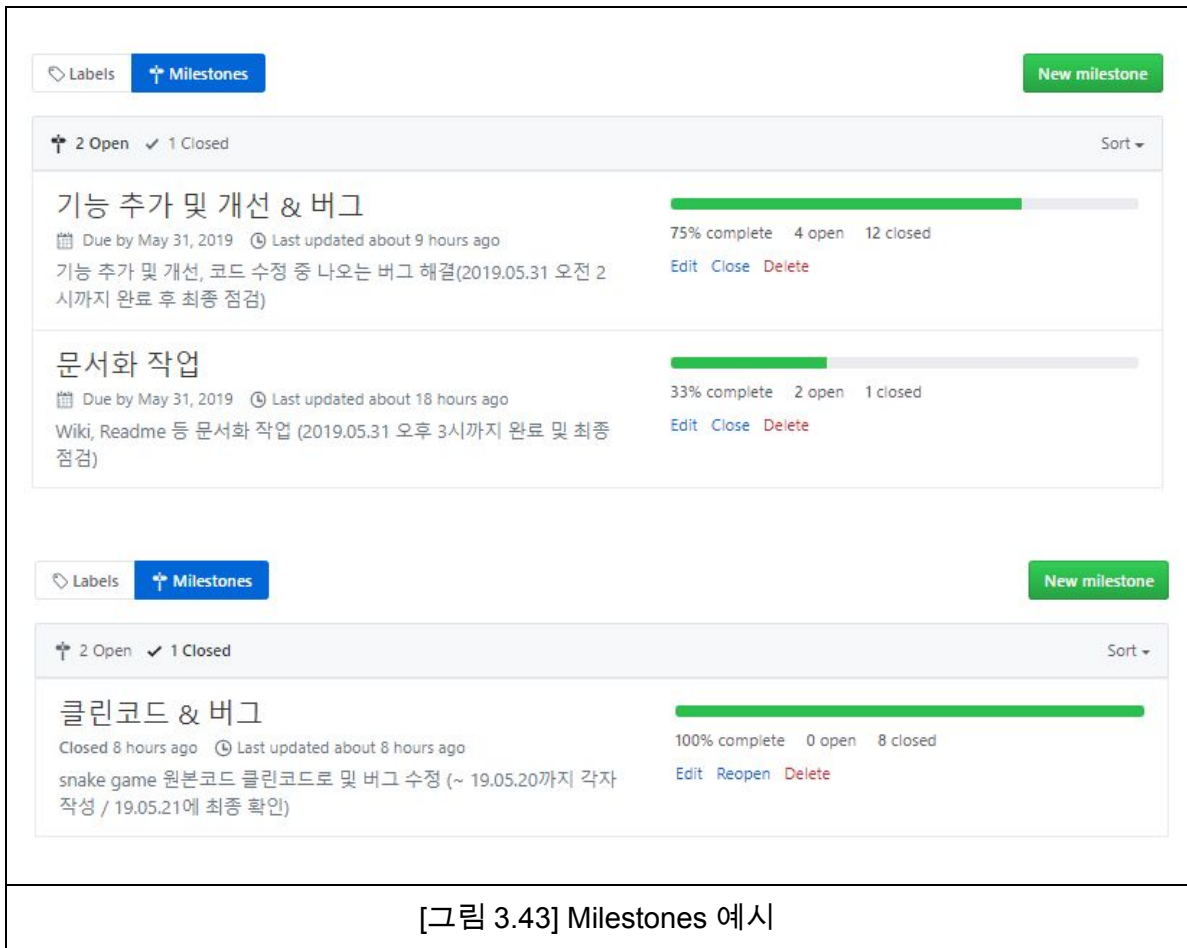
해당 이슈의 주요내용이 무엇인지 구분해준다. 이번 프로젝트는 기존의 코드를 클린코드로 작성하여 기능을 추가 및 개선했기 때문에 Clean, Improvement, New라는 label을 생성했다. 또한 이러한 과정을 거치면서 발생하는 오류나 합병해야 하는 경우를 위해 Bug, Merge라는 label을 만들었다.

이 외에도 문서화작업에 필요한 Documentation, 운영에 관련된 질문 및 답변을 하며 Github사용자와 소통하는 FAQ, 해당 프로젝트를 진행하면서 생기는 문제점에 대해 의논하는 Question과 규칙을 정하는 Rule, 버전에 대한 내용을 관리하기 위해 Tag라는 label을 추가했다.



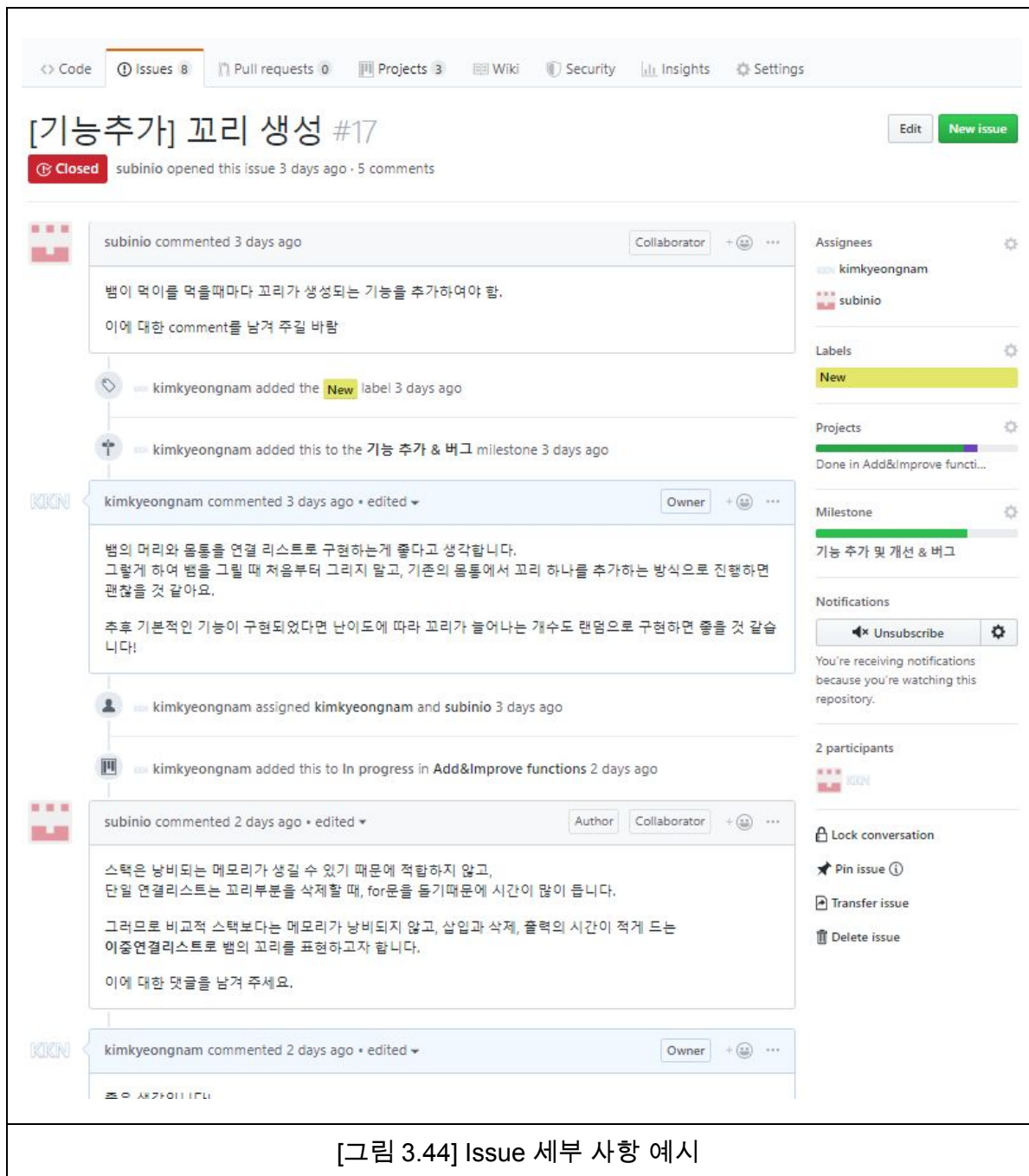
2) Milestones

프로젝트 일정을 관리할 수 있는 기능이다. 이번 프로젝트에서는 크게 클린코드&버그, 기능 추가 및 개선&버그, 문서화 작업 총 3가지로 구분하여 프로젝트 세부 일정을 구성하였다.



3) Issue 생성 및 토론

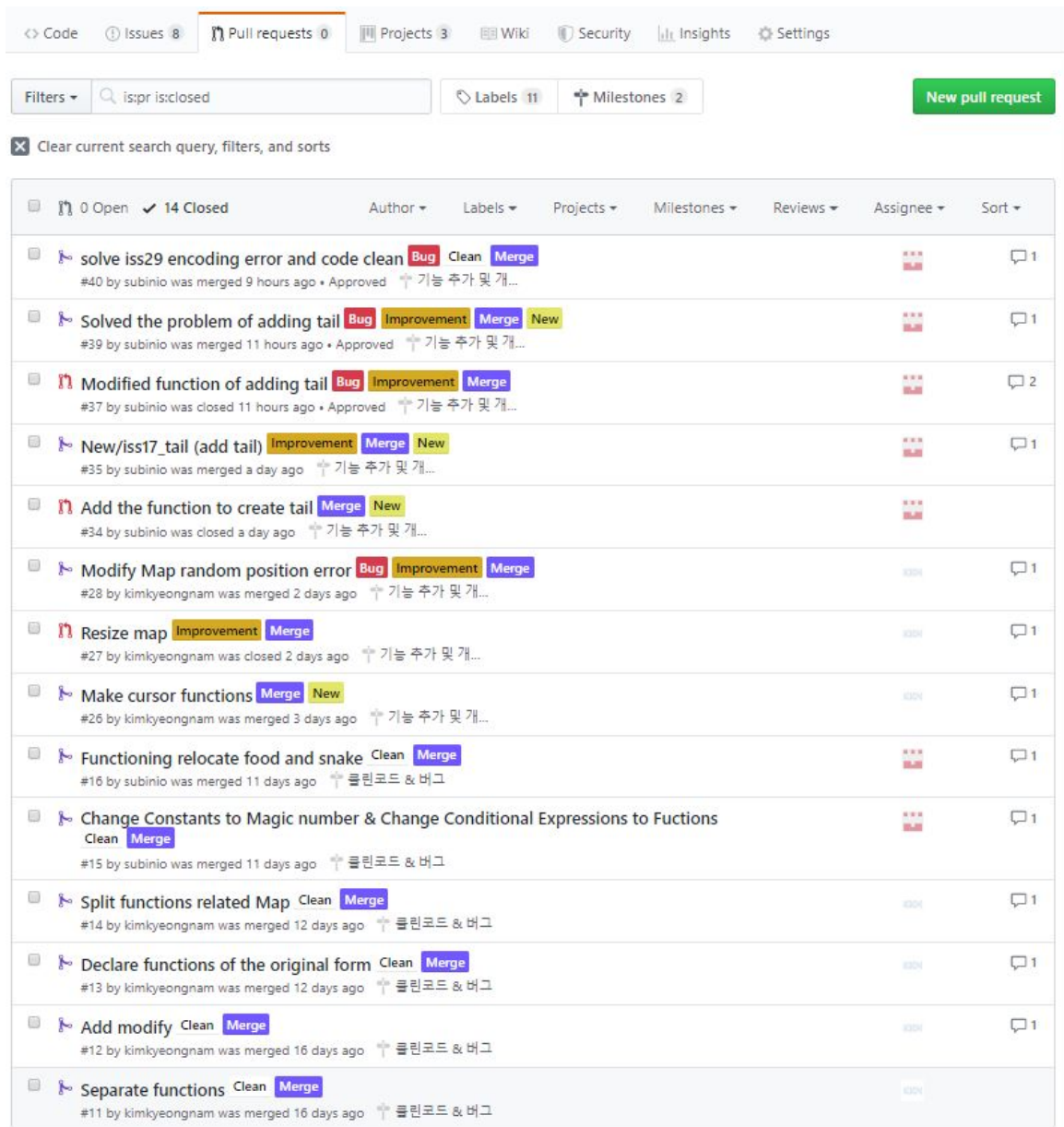
Issue를 생성하면 [그림 3.44]과 같은 화면을 볼 수 있다. 오른쪽 사이드바를 보면 Assignees, Labels, Projects, Milestone이라는 메뉴가 보이는데, 이를 통해 프로젝트의 일정을 공유하고 해당 Issue의 주요 내용이 무엇인지 한 눈에 파악할 수 있다. Assignees를 통해 보고 있는 Issue에 누가 참여했는지 알 수 있다. 의견은 댓글을 통해 교환할 수 있다.



[그림 3.44] Issue 세부 사항 예시

3-6-2. Pull requests 활용하기

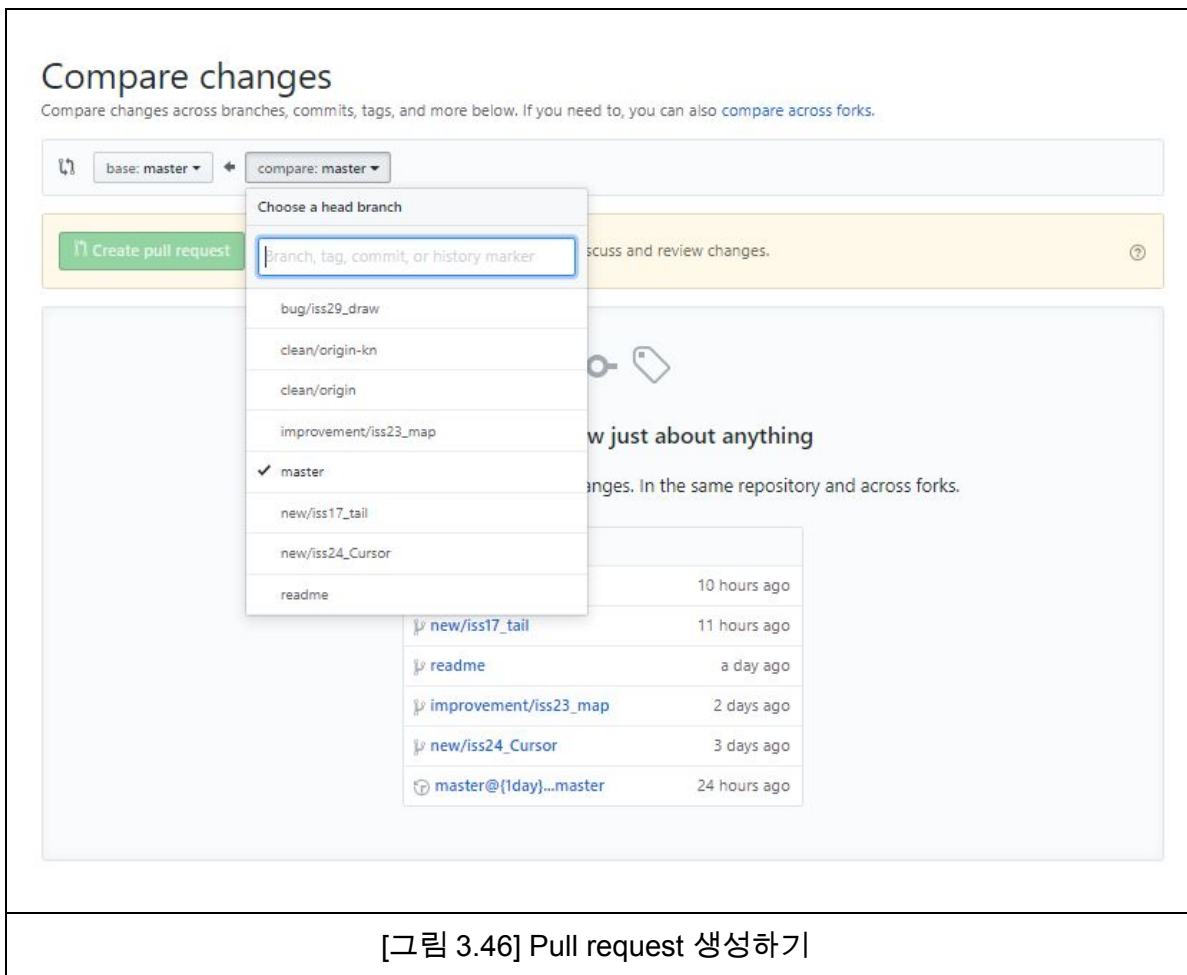
프로젝트를 진행하면서 어떤 branch를 특정한 branch에 합병하여 코드를 개선시키고자 할 때, Pull requests를 이용한다.



[그림 3.45] Pull requests 활용 예시

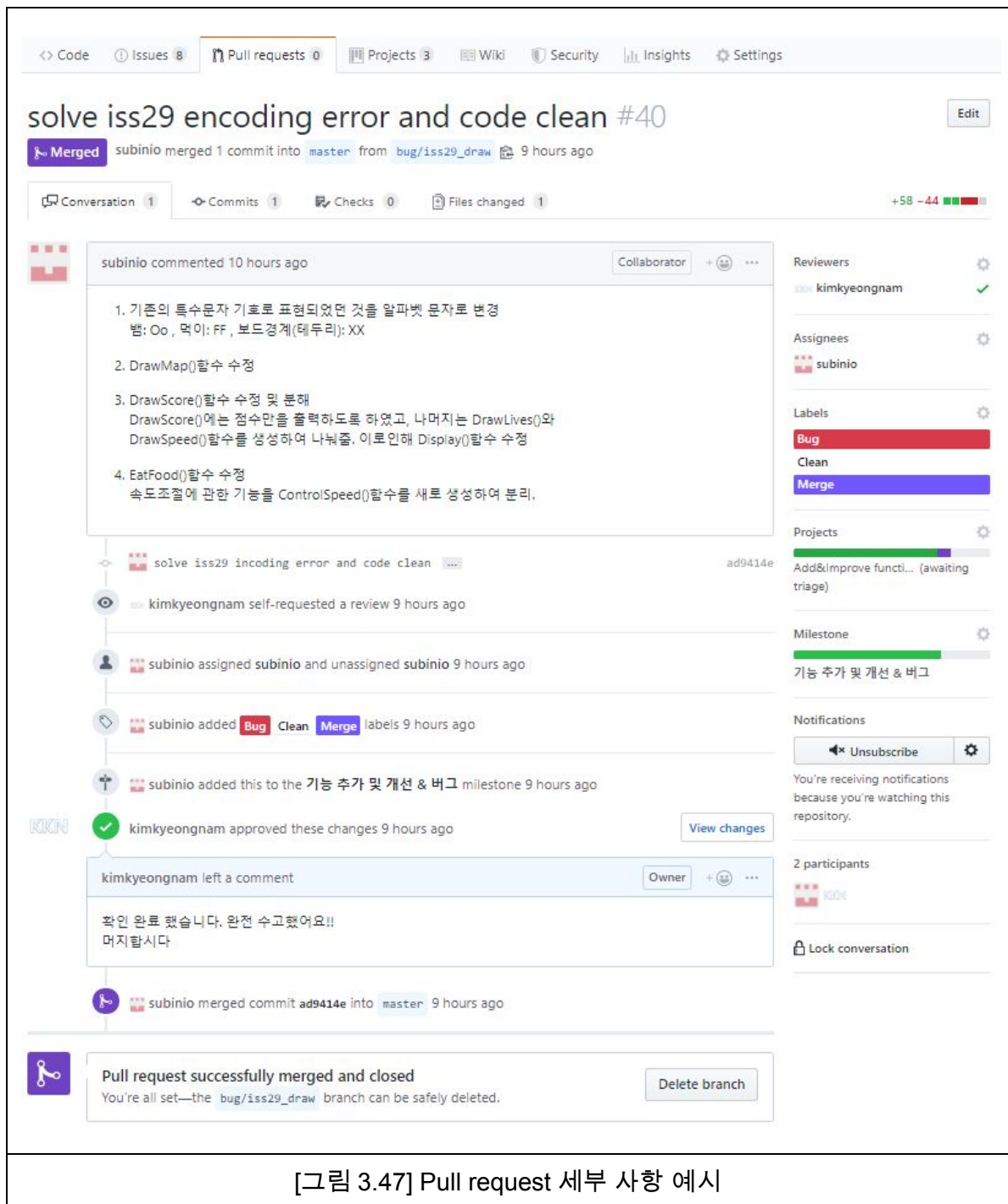
1) Pull requests 요청하기

Pull requests 메뉴에 접속하여 New pull request 버튼을 클릭한다. 그러면 [그림 3.46]과 같은 화면을 볼 수 있는데 왼쪽, 오른쪽 branch 설정을 통해 합병할 branch를 선택할 수 있다.



2) Pull requests 확인하기

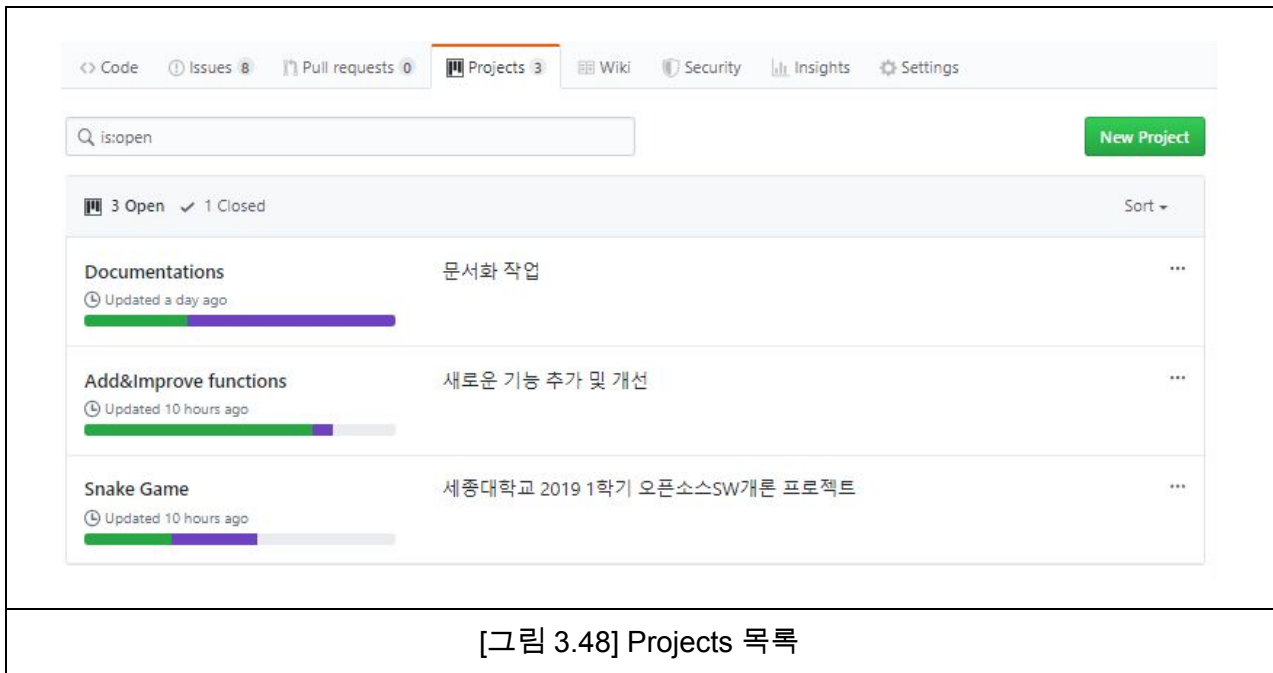
Issue와 마찬가지로 오른쪽 사이드바를 통해 Assignees, Labels, Projects, Milestone을 설정할 수 있다. Issue와 다른 점은 바로 Reviewers인데, Reviewers에 할당받은 사용자는 code review를 통해 해당 pull requests요청을 승인할 것인지, 아니면 추가할 내용에 대해 코멘트를 남길 수 있다. pull requests 요청자는 프로젝트 기여 규칙과 해당 코멘트를 참고하여 merge를 진행할 수 있다.



[그림 3.47] Pull request 세부 사항 예시

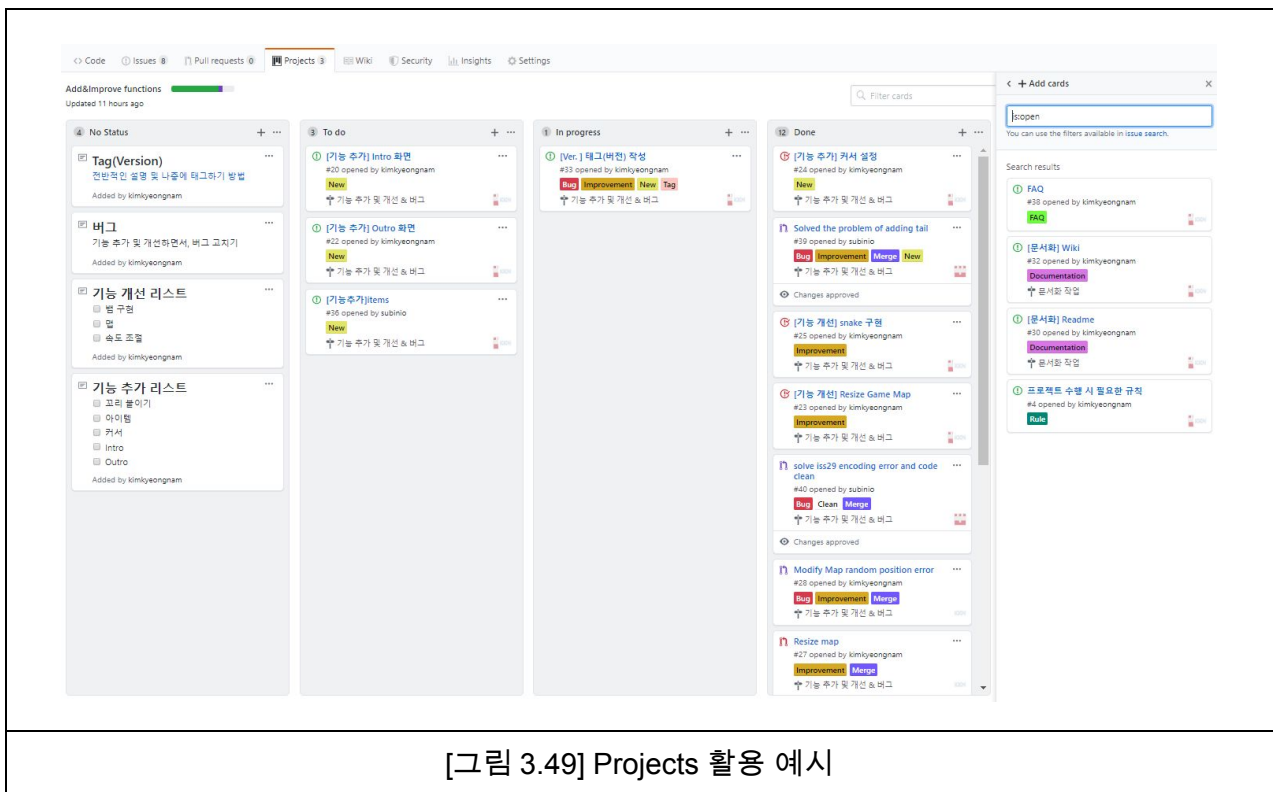
3-6-3. Projects 활용하기

프로젝트의 전반적인 일정 및 진행 내역을 확인할 수 있다. 초록색 게이지는 완료한 항목, 보라색 게이지는 현재 진행 중인 항목을 의미한다. 간단하고 사용하기 쉬운 인터페이스로 구성되어 있으며 한 눈에 들어오는 대시보드로 프로젝트 진행도와 목록을 확인할 수 있다.



[그림 3.48] Projects 목록

만약 Issue와 Pull requests에서 Project를 선택했으면 cards에 해당 Issue와 Pull requests항목이 뜬다. 생성된 card를 이동시켜 해당 Issue나 Pull requests상태가 예정된 일정(To do)인지, 진행 중(In progress)인지, 끝(Done)인지 파악할 수 있다.

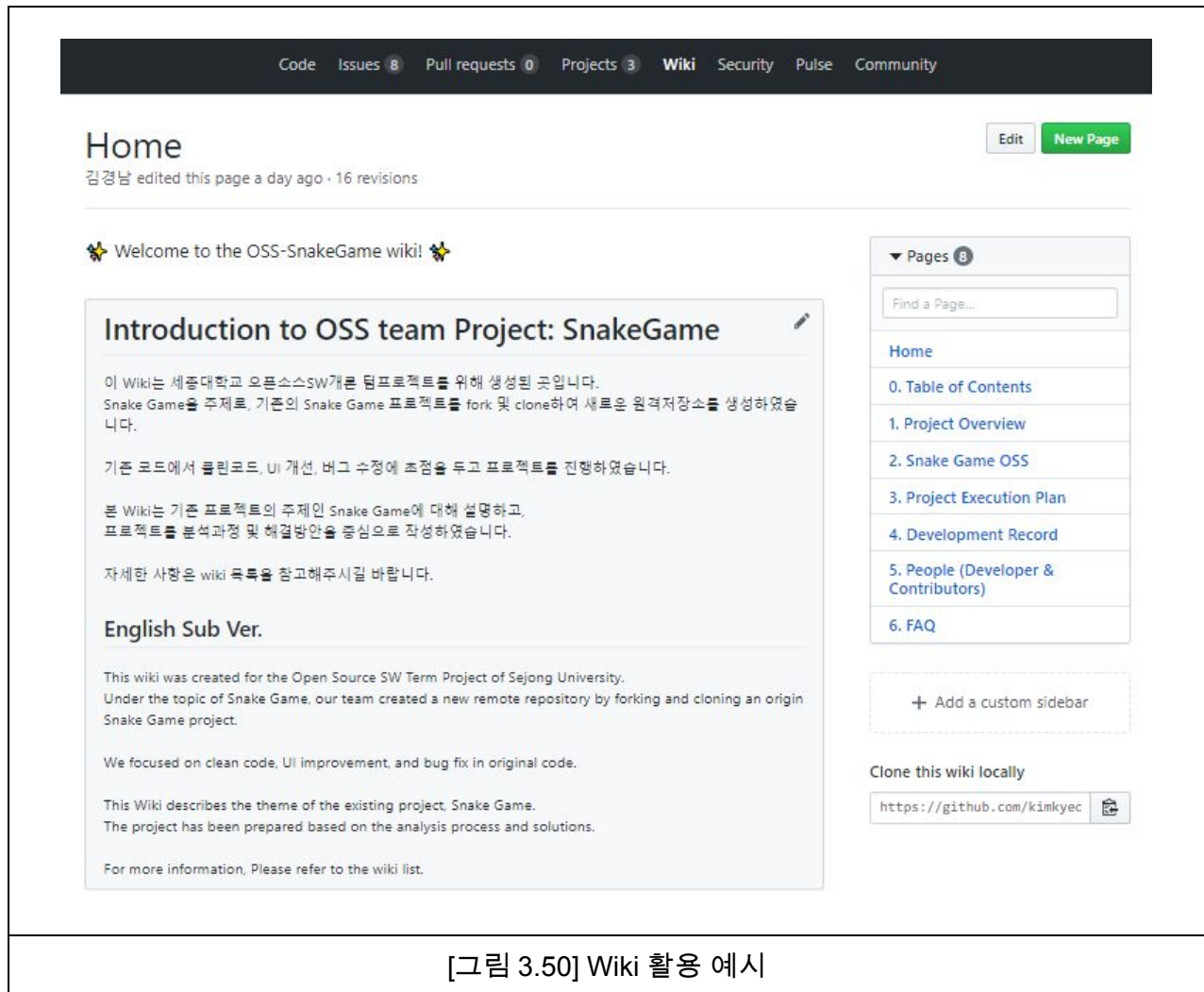


[그림 3.49] Projects 활용 예시

3-6-4. Wiki 활용하기

프로젝트의 내용에 관해서는 README.md를 참고하면 되지만, README.md에 세부사항을 작성하기엔 한계가 있다. 이렇게 프로젝트의 세부 내용에 관련해 작성할 시 Wiki를 사용한다.

Wiki에는 다른 Github 사용자가 해당 repository를 접했을 시, repository 활용법이나 프로젝트 기여자 정보 등 프로젝트에 관련된 전반적인 사항을 작성한다.



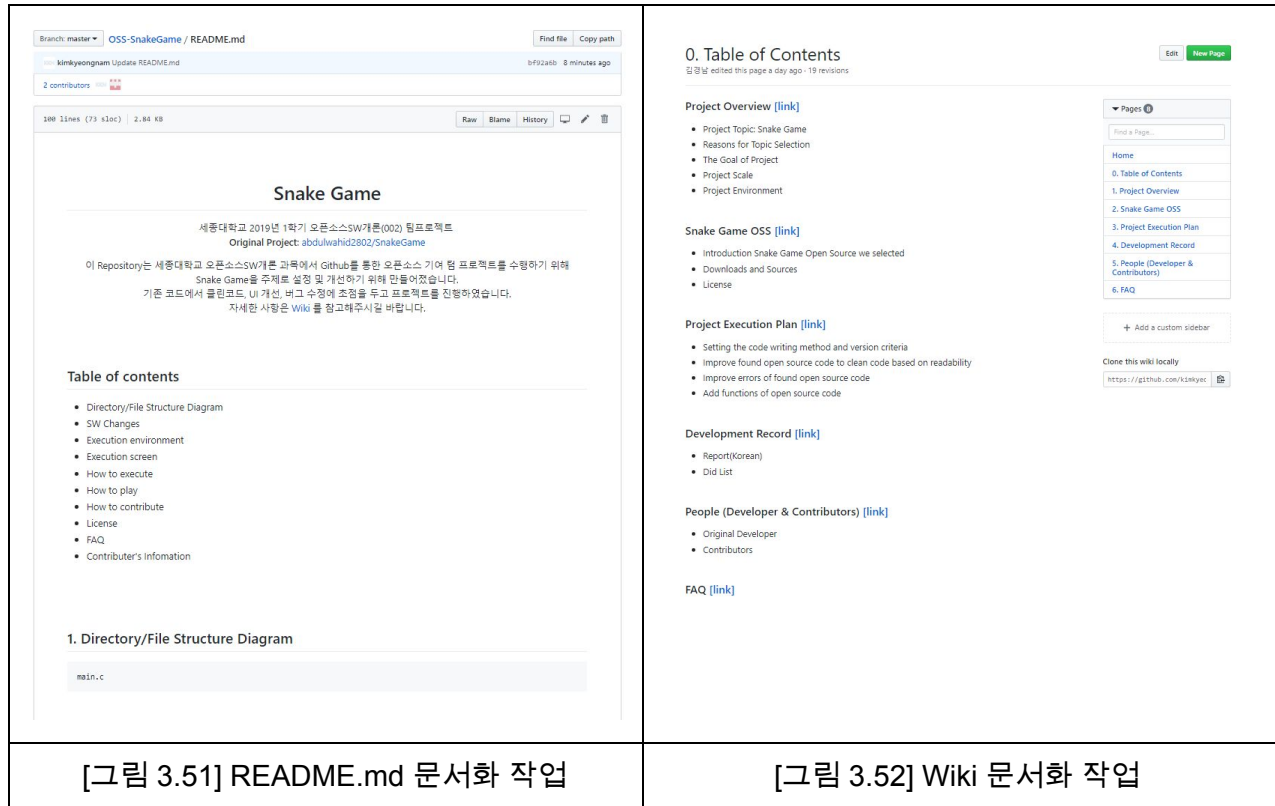
[그림 3.50] Wiki 활용 예시

3-7. 문서화 작업하기

문서화는 효과적인 위험 완화나 업무 명확성 등 효율적인 공동작업을 진행할 수 있게 가이드라인을 제시한다. 하지만 너무 많은 문서는 오히려 개발자들에게 혼란을 안겨줄 수 있다. 따라서 현재 또는 미래에 기여를 할 때 실제로 도움이 될 수 있는지의 여부와 항상 최신 버전으로 업데이트할 수 있는지를 고려해 효율적으로 문서화 작업을 진행하였다.

그 결과 프로그램 소개 및 활용 범위, 디렉토리/파일 구조도, 프로그램 설치 방법, SW옵션 및 사용법, 운영에 관련된 FAQ, SW변경사항, 소스코드 저작권을 중심으로 문서화를 진행하기로 결정했다.

README.md에는 전반적인 프로젝트 설명을 작성했고, 세부 내용은 Wiki를 통해 다른 Github 사용자가 볼 수 있도록 설정했다.



4. 역할 분담

4-1. 코드분석, 개선, 기능추가 역할 분담

이름	코드 분석 (함수)	코드 개선 및 수정	기능 추가	문서화 작업 및 발표 준비	기타
김경남	InitMap() DrawMap() GetInput()	커서 함수 구현 및 기능 추가 및 개선 아이디어 구성	Wiki, README, 보고서 작성	PPT 제작 및 발표 준비	Project 관리 Milestone 관리 Issue 관리 Pull requests 관리 Wiki 관리 Branch 관리 Tag 관리 Label 관리
진수빈	EatFood() Die() main()	Snake 관련 함수 개선 및 Intro 구현	README 아이디어 구성 및 보고서 작성		

선택한 Snake Game 코드는 총 6개의 함수(InitMap, DrawMap, GetInput, EatFood, Die, main)로 구성되어 있고, 해당 함수들 중 관계가 있는 함수들로 묶고 둘로 나눠 코드를 분석하였다.

분석 이후 진행될 코드 개선 및 수정 역시 각자 분석했던 코드를 바탕으로 프로젝트 수행 내용에 맞춰 개발하도록 할 것이다.

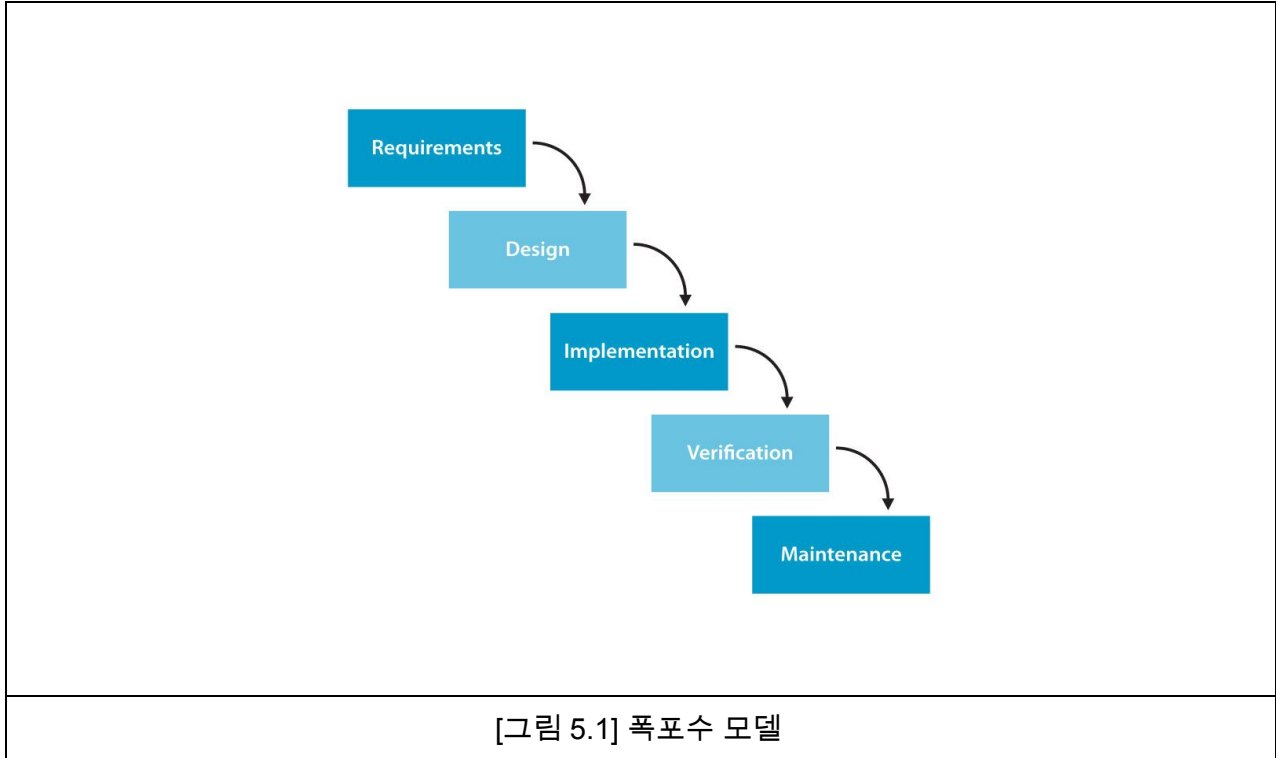
보고서 및 발표자료 준비는 매주 수요일마다 정기회의를 통해 서로의 진도 및 해야할 목록을 피드백하여 과제 기여도 균형을 유지하도록 하며, 각각의 목차에 대해 서로의 생각을 정리하고 작성하도록 하였다.

코드 분석 및 클린코드로 개선 이후 Issue를 통해 해당 소스를 개선시키기 위한 방법을 구상하고, 협의 후 해당 기능을 구현도록 결정했다.

5. 추진 방법 및 수행 일정

5-1. 추진 방법

효율적이고 효과적인 프로젝트 결과물 산출을 위해 소프트웨어 개발 생명주기(Software Development Life Cycle, SDLC)모델 중 폭포수 모델(Waterfall Model)을 통해 프로젝트를 추진한다.



폭포수 모델은 분석, 설계, 구현, 확인(테스트), 유지보수의 과정을 순차적으로 진행하는 모델로 Snake Game 오픈소스 개발 단계에 적합하다. 기존의 Snake Game 오픈소스 코드를 분석하고, 새로운 기능이나 수정이 필요한 부분을 다시 설계하여 구현하고 유지보수를 반복하는 것이다.

모델에 맞춰 개발 계획을 미리 세워 놓고 프로젝트를 추진해나갈 것이다. 수행일정은 다음과 같다.

5-2. 수행 일정

날짜	3.25 ~ 3.31	4.1 ~ 4.7	4.8 ~ 4.14	4.15 ~ 4.19	4.22 ~ 4.28	4.29 ~ 5.5	5.6 ~ 5.12	5.13 ~ 5.19	5.20 ~ 5.26	5.27 ~ 5.31
요구 분석					중간 고사					
설계										
구현										
테스트										
유지 보수										

요구 분석 기간에는 프로젝트의 구조 파악 및 flow-chart를 작성했다.

설계 기간에는 해당 프로젝트에서 기능 추가 및 구현에 대한 아이디어를 도출하기로 정했고, 구현 및 테스트 단계에서는 아이디어를 실제로 프로그래밍 및 구현하면서 생긴 오류를 고치는 것을 중점으로 두었다.

유지보수 기간에는 최종마무리 및 문서화 작업에 중점을 두어 프로젝트 수행을 진행하였다.

6. 회의록

일시	2019. 03. 27 16:30 ~ 18:00	장소	센B112
작성자	김경남	작성일	2019. 03. 30
참석자	김경남, 진수빈		
안건	오픈소스 게임별 특징 찾기 및 정기 회의 날짜 결정		

■ 회의내용

- 회의내용
 - [오픈소스] 오픈소스 각 주제별 특징 찾기
 - [정기회의] 정기 회의 날짜 결정
- 결정사항
 - [보고서] 과제 3번 양식 작성 완료(3/27)
 - [오픈소스] 다음 회의까지 주제별 게임 특징 정보 수집 및 GitHub 코드 검색
 - [정기회의] 매주 수요일 16:30 - 18:30에 진행하기로 결정
이외의 회의는 상시에 진행
- 향후일정
 - [정기회의] 2019.03.27 센B112에서 정기 회의 진행 예정

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] Tetris game, Car game, Pacman game, Snake game, Tic Tac Toe game의 게임별 특징 정보 수집 및 GitHub 코드 검색	김경남	~ 04.10	회의록 작성
[오픈소스] 2048 game, Bricks game, Chess for two humans, Mine Sweeper game, Egg game의 게임별 특징 정보 수집 및 GitHub 코드 검색	진수빈	~ 04.10	

일시	2019. 04. 03 16:30 ~ 18:30	장소	센B112
작성자	김경남	작성일	2019. 04. 03
참석자	김경남, 진수빈		
안건	프로젝트 일정 조율, 오픈소스 게임 및 코드 선택		

■ 회의내용

- 회의내용
 - [오픈소스] 각자 조사해 온 게임 정보 교환 및 최종 주제 결정
- 결정사항
 - [일정] 프로젝트 수행 일정 조율
 - [오픈소스] 게임주제: Snake game으로 결정
GitHub코드는 좀 더 조사 후 선택하기로 결정
- 향후일정
 - [정기회의] 2019.04.10 센B112에서 정기 회의 진행 예정
 - [오픈소스] 선택한 Snake game의 코드를 나누어 분석할 예정

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] GitHub에서 Snake game 코드 조사	김경남	~4월 10일	회의록 작성
[오픈소스] GitHub에서 Snake game 코드 조사	진수빈	~4월 10일	

일시	2019. 04. 10 16:30 ~ 18:30	장소	센B112
작성자	진수빈	작성일	2019. 04. 10
참석자	김경남, 진수빈		
안건	Snake game 코드 결정 및 역할분담		

■ 회의내용

- 회의내용
 - [오픈소스] 조사한 Snake game 코드 공유 및 최종 선택
선택한 코드 분리 및 분석할 부분 선택
- 결정사항
 - [오픈소스] 코드: process/Snake로 결정 (<https://github.com/process/Snake>)
역할분담: 선택한 함수 코드 분석 및 flow-chart를 작성하기로 결정
- 향후일정
 - [정기회의] 2019.04.17 센B112에서 정기 회의 진행 예정
 - [오픈소스] 각자의 flow-chart 확인 및 전체 flow-chart 작성 예정

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] InitSnake(), UpdateSnake(), SetPos(), UpdateScreen(), FreshDraw() 함수 분석 및 flow-chart 작성	김경남	~4월 17일	
[오픈소스] GameLoop(), GameOver(), PrintCenteredText(), main(), DrawFood() 함수 분석 및 flow-chart 작성	진수빈	~4월 17일	회의록 작성

일시	2019. 04. 13 19:30 ~ 2019. 04. 14 03:30	장소	- (카카오톡을 통한 온라인 회의)
작성자	김경남	작성일	2019. 04. 13
참석자	김경남, 진수빈		
안건	Snake game 코드 재조사, 결정 및 역할분담		

■ 회의내용

※ [긴급회의] 기존의 분석했던 코드가 C++인 것을 깨달아 긴급회의 진행

- 회의내용
 - [오픈소스] 게임주제: Snake game으로 유지할지 회의
 - [일정] 프로젝트 일정 재조율
- 결정사항
 - [오픈소스] 게임주제: Snake game으로 주제를 유지하기로 결정
코드: 새로운 소스 코드 조사 및 최종 결정
(<https://github.com/abdulwahid2802/SnakeGame>)
역할분담: 선택한 함수 코드 분석
 - [일정] 프로젝트 수행 일정 재조율 완료
- 향후일정
 - [정기회의] 2019.04.17 21:00부터 카카오톡을 통해 정기 회의 진행 예정
 - [오픈소스] 각자 맡은 부분의 flow-chart를 작성할 예정

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] InitMap(), DrawMap(), GetInput() 함수 분석	김경남	~4월 17일	회의록 작성
[오픈소스] EatFood(), Die(), main() 함수 분석	진수빈	~4월 17일	

일시	2019. 04. 17 21:00 ~ 22:30	장소	- (카카오톡을 통한 온라인 회의)
작성자	진수빈	작성일	2019. 04. 18
참석자	김경남, 진수빈		
안건	forest 및 tree level의 flow-chart 제작		

■ 회의내용

- 회의내용
 - [오픈소스] 각자 맡은 코드분석 확인 및 flow-chart 작성
- 결정사항
 - [오픈소스] flow-chart: 각자 맡은 코드를 tree level의 flow-chart로 제작 후, 팀장이 forest level의 flow-chart를 제작하기로 결정
- 향후일정
 - [정기회의] 2019.04.27 센B112에서 회의 진행 예정
(04.24는 중간고사 기간이므로 회의 취소)
 - [오픈소스] 보고서(과제3) 작성에 초점을 맞출 예정

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] 1. InitMap(), DrawMap(), GetInput()의 코드 주석 및 flow-chart 작성 2. 전체 flow-chart 작성	김경남	1. ~4월 18일 2. ~4월 19일	
[오픈소스] EatFood(), Die(), main()의 코드 주석 및 flow-chart 작성	진수빈	~4월 18일	회의록 작성

일시	2019. 04. 29 16:30 ~ 19:30	장소	학술정보원 4층 창의토론라운지
작성자	김경남	작성일	2019. 04. 27
참석자	김경남, 진수빈		
안건	보고서 작성		

■ 회의내용

- 회의내용
 - [오픈소스] 과제4 피드백 확인
 - [보고서] 목차 세부사항 설정
- 결정사항
 - [보고서] 목차 세부사항 결정
각자 맡은 부분을 작성하기로 결정
- 향후일정
 - [오픈소스] 코드를 클린코드로 개선 및 기능 추가
 - [보고서] 보고서(과제3) 제출

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
보고서 목차 3, 4, 5, 7번 작성	김경남	~5월 3일	회의록 작성
보고서 목차 1, 2, 3, 7번 작성	진수빈	~5월 3일	

일시	2019. 05.08 16:30 ~ 18:00	장소	센B112
작성자	진수빈	작성일	2019. 05. 08
참석자	김경남, 진수빈		
안건	문서화 작업 및 규칙 정리, Github 기능 사용		

■ 회의내용

- 회의내용
 - [오픈소스] 문서화 작업 목록 확인
branch 및 commit 규칙 확인
Github 기능 확인
 - [보고서] 피드백 확인
- 결정사항
 - [오픈소스] 문서화 작업 및 코드 개선: 5월 12일까지 완료
 - [보고서] 피드백 바탕으로 목차 및 내용 수정
- 향후일정
 - [오픈소스] 코드를 클린코드로 개선 및 기능 추가
 - [정기회의] 2019.05.15 센B112에서 회의 진행 예정

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] 각자 맡은 부분 클린코드로 수정 기능 추가 및 개선 아이디어 생각해오기	김경남	~5월 12일	
[오픈소스] 각자 맡은 부분 클린코드로 수정 기능 추가 및 개선 아이디어 생각해오기	진수빈	~5월 12일	회의록 작성

일시	2019. 05.15 16:00 ~ 18:00	장소	센B112
작성자	김경남	작성일	2019. 05. 15
참석자	김경남, 진수빈		
안건	문서화 작업 및 규칙 정리, Github 기능 사용		

■ 회의내용

- 회의내용
 - [오픈소스] 클린코드 및 버그 수정
 - [보고서] 서로의 피드백 확인
- 결정사항
 - [오픈소스] 코드개선 및 버그 수정: 일요일까지
 - [보고서] 목차 구성 완료
- 향후일정
 - [오픈소스] 코드를 클린코드로 개선 및 기능 추가
 - [정기회의] 2019.05.22 센B112에서 회의 진행 예정

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] 각자 맡은 부분 클린코드로 수정 기능 추가 및 개선 아이디어 생각해오기	김경남	~5월 19일	회의록 작성
[오픈소스] 각자 맡은 부분 클린코드로 수정 기능 추가 및 개선 아이디어 생각해오기	진수빈	~5월 19일	

일시	2019. 05.22 20:00 ~ 21:00	장소	- (카카오톡을 통한 온라인 회의)
작성자	진수빈	작성일	2019. 05. 15
참석자	김경남, 진수빈		
안건	문서화 작업 및 규칙 정리, Github 기능 사용		

■ 회의내용

- 회의내용
 - [오픈소스] 클린코드 및 버그 최종 확인
- 결정사항
 - [오픈소스] 코드개선 및 버그 수정 최종 완료
기능 개선 및 추가에 중점을 두기로 결정
Github 규칙 재정의 및 확인
 - [보고서] 코드 수정을 하면서 수정한 부분 캡처
- 향후일정
 - [오픈소스] 기능 개선 및 추가, tag를 붙여 최종 완료 예정
 - [보고서] 작성 완료

■ 업무분담 담당리스트

업무사항	담당자	기한	비고
[오픈소스] 문서화 작업 및 커서, map 구현	김경남	~5월 31일	
[오픈소스] Snake 구현, UI 구현	진수빈	~5월 31일	회의록 작성

7. 참고문헌 및 사이트

- [그림1] 출처 : CoolmathGames, “snake game image”, <https://www.coolmathgames.com/0-snake>
- MIT 라이선스 주요내용: 오픈소스SW 라이선스 종합정보시스템, “MIT 라이선스 주요내용”, <https://www.olis.or.kr/license/Detailselect.do?lId=1006&mapCode=010006>
- [그림14] 출처 : SOFTWARE TESTING TUTORIALS - MANUAL AND AUTOMATION QUESTIONS ANSWERS, “waterfall model”, <http://jobsandnewstoday.blogspot.com/2013/04/what-is-waterfall-model.html>
- 프로젝트 유형에 따른 SDLC 선정(폭포수 모델): 봉봉이 아빠의 관심사, “폭포수 모델 프로젝트 규모”, <https://bongbonge.tistory.com/category/%EC%A0%95%EB%B3%B4%EA%B4%80%EB%A6%AC%EA%B8%B0%EC%88%A0%EC%82%AC/%EB%8F%84%EB%A9%94%EC%9D%B82.%20%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4%20%EA%B3%B5%ED%95%99?page=3>, (2015.07.13)