**C++프로그래밍**

**프로젝트**

|  |  |
| --- | --- |
| 프로젝트 명 | *Snake\_Game* |
| 팀 명 | *호수 (HoSu)* |
| 문서 제목 | 결과보고서 |

|  |  |
| --- | --- |
| **Version** | 1.7.1 |
| **Date** | 19 |

|  |  |
| --- | --- |
| **팀원** | 정호용(팀장) |
| 구수연 |
|  |

|  |
| --- |
| **CONFIDENTIALITY/SECURITY WARNING**  이 문서에 포함되어 있는 정보는 국민대학교 소프트웨어융합대학 소프트웨어학부 및 소프트웨어학부 개설 교과목 C++프로그래밍 수강 학생 중 프로젝트 “Snake game”를 수행하는 팀 “HoSu”의 팀원들의 자산입니다. 국민대학교 소프트웨어학부 및 팀 “HoSu”의 팀원들의 서면 허락없이 사용되거나, 재가공 될 수 없습니다. |

**문서 정보 / 수정 내역**

|  |  |
| --- | --- |
| **Filename** | C++ HoSu 스네이크게임\_보고서.docx |
| **원안작성자** | 구수연, 정호용 |
| **수정작업자** | 구수연, 정호용 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 수정날짜 | 대표수정자 | Revision | 추가/수정 항목 | 내 용 |
| 2023-06-04 | 정호용 | 1.0 | 최초 작성 |  |
| 2023-06-05 | 구수연 | 1.1 | 2.2.2 일부 작성  2.2.5 사진 변경 | 1,5단계 작성, 3단계 일부 작성,  게임화면 수정 |
| 2023-06-06 | 구수연 | 1.2 | 개요, 부록, 2.1, 2.2, 2.3자기평가 작성 | 개요, 부록, 2.1, 2.2의 2단계 작성,  2.3 작성 |
| 2023-06-06 | 정호용 | 1.3 | 2.2.4 작성  자기평가 작성 | 2.2.4 한계점 및 해결방안 3가지 작성 , 자기평가 작성 |
| 2023-06-06 | 정호용 | 1.4 | 2.1및2.2.1 작성 | 2.1 목표 작성 및 2.2.1 개발 내용 작성 |
| 2023-06-16 | 구수연 | 1.5 | 업데이트 현황 작성  자기평가 수정 | 버전 별 업데이트 현황 작성 및  자기평가 내용 보충 |
| 2023-06-16 | 정호용 | 1.6 | 오타, 형식 수정 |  |
| 2023-06-16 | 정호용 | 1.7 | 2.1, 2.2, 3 작성 및 형식 통일 | 3,4단계 중심으로 2.1 목표,  2.2.1~2.2.4, 3 작성 및 형식 통일 |
| 2023-06-19 | 정호용 | 1.7.1 | 부록, 참고문헌, 소감 작성 |  |
|  |  |  |  |  |
|  |  |  |  |  |

**목 차**

[1 개요 4](#_Toc106362480)

[2 개발 내용 및 결과물 6](#_Toc106362481)

[2.1 목표 6](#_Toc106362482)

[2.2 개발 내용 및 결과물 9](#_Toc106362483)

[2.2.1 개발 내용 9](#_Toc106362484)

[2.2.2 시스템 구조 및 설계도 14](#_Toc106362485)

[2.2.3 활용/개발된 기술 43](#_Toc106362486)

[2.2.4 현실적 제한 요소 및 그 해결 방안 44](#_Toc106362487)

[2.2.5 결과물 목록 45](#_Toc106362488)

[3 자기평가 49](#_Toc106362489)

[4 참고 문헌 51](#_Toc106362490)

[5 부록 51](#_Toc106362491)

[5.1 사용자 매뉴얼 51](#_Toc106362492)

[5.2 설치 방법 52](#_Toc106362493)

# 개요

**저희 팀은 개발을 진행하기 앞서 개발 진행 계획을 세웠습니다.**

**첫 번째**로 ncurses라이브러리를 사용하여 기본적인 게임의 화면 예를 들어 각 게임스테이지, 미션창, 점수창 등을 구성과 디자인합니다.

**두 번째**로 뱀이 스테이지에 나타나고 움직이는 것을 구현합니다.

**세 번째**로 게임 내 요소인 아이템과 게이트의 등장을 구현합니다.

**네 번째**로 뱀이 아이템을 먹고 게이트를 통과할 때 나타나는 뱀의 변화와 점수를 구현합니다.

**다섯 번째**로 각 스테이지의 미션을 완료 후 다음 스테이지로 넘어가는 것을 구현합니다.

**마지막**으로 기타 오류와 버그를 수정 또는 보완합니다.

개발하는 동안 구글 드라이브를 통해 각자 구현한 파일을 공유하며 협업을 하였습니다. github의 commit처럼 오픈채팅을 통해 각자가 수정하고 구현한 내용 또는 버그나 오류 내용 등을 해당 버전을 드라이브에 업로드함과 동시 실시간으로 공유하였습니다.

snakegame을 개발하면서 사용한 외부 라이브러리는 ncurses입니다. 리눅스 우분투 개발환경에서 사용하였으며 해당 라이브러리의 획득 및 설치 방법은 터미널에 **「sudo apt-get install libncurses5-dev libncursesw5-dev**」 입력하여 설치하였습니다. 그 후 main.cpp파일에 #include <ncurses.h>를 작성하여 진행하였습니다. 또한 makefile을 활용하여 「**g++ -std=c++11 -o main main.cpp -lncursesw**」 을 컴파일 하여 실행하였습니다.

snakegame 구조는 map.h, game.h, gate.h의 3개의 헤더 파일과 main.cpp, map.cpp, gate.cpp, snake.cpp의 4개의 소스파일 그리고 makefile로 구성되어 있습니다. map.h파일에서는 각 스테이지의 맵을 0과 1로 이루어진 2차원 배열 형태로 구성하며 각 스테이지를 배열를 통해 관리합니다. game.h 파일에서는 게임이 진행되는 main.cpp파일과 게임 화면을 출력하는 map.cpp파일의 초기 설정값과 변수, 함수들이 선언하여 관리합니다. gate.h는 gate.cpp의 헤더 파일로 뱀의 위치와 이동, 변화를 구현한 snake.cpp과 연결되어 게이트 통과 후 뱀의 위치 설정에 영향을 줍니다.

main.cpp에는 ncurses, vector, time.h, cstdlib같은 라이브러리와 map.h, map.cpp, gate.cpp, snake.cpp 파일 등이 include 됩니다. main.cpp의 game()함수에서는 map.cpp, gate.cpp, snake.cpp의 함수들을 이용하여 게임화면과 스테이지의 세팅, 뱀의 조작, 아이템과 게이트의 발생 등을 설정하고 관리하여 게임을 진행시킵니다. map.cpp파일에서 색깔, 배경, 스테이지, 점수와 미션 창, 초기화 등을 구현합니다. gate.cpp파일에서는 게이트의 발생 위치와 게이트 통과 후 뱀의 위치에 대해 설정합니다. snake.cpp 파일에서는 뱀의 위치, 이동, 아이템, 뱀의 몸 길이 변화 등을 구현하고 관리합니다.

다음으로는 구글 드라이브에 올렸던 파일들이 어떤 과정을 거쳐 업데이트 되었는지를 나타내는 간략한 표 입니다. 조원 모두가 게임 구현에 골고루 참여했음을 알 수 있습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **폴더명** | **업로드 일시** | **최종**  **작성자** | **변경 내용** | **비고 (버그 등)** |
| Main | 23.05.26 | 정호용 | 메인 화면 출력 구현  실행 부분 |  |
| Map | 23.05.26 | 구수연 | 맵 헤더파일 생성  맵 동작 구현 |  |
| Map | 23.05.26 | 정호용 | 맵 편집 및 맵 병합 |  |
| Snake | 23.05.28 | 구수연 | 일정 시간마다 뱀 이동 |  |
| Item | 23.05.28 | 구수연 | 아이템 생성 |  |
| Snake | 23.05.28 | 정호용 | 스네이크 구현 및 스네이크 이동 수정  성장 아이템, 독 아이템 생성 및  랜덤 생성 구현 |  |
| Snake | 23.05.28 | 구수연 | 아이템 획득 시  스네이크 몸 길이 변경 구현 |  |
| Snake | 23.05.28 | 정호용 | 스네이크 길이 변경 및 움직임 수정 |  |
| Score | 23.06.04 | 정호용 | 아이템 획득 시 스코어 보드 반영  (미완성) | 변수 오류인지 0으로 뜨는 버그 발생 |
| Gate | 23.06.04 | 구수연 | 게이트 통과 구현  (미완성)  gate.cpp 분리 | 게이트 통과 시 segmentation fault 오류 발생 |
| Gate | 23.06.04 | 정호용 | 게이트 통과 구현 (완성) |  |
| Snake | 23.06.04 | 구수연 | 벽에 충돌 시 게임오버 구현 반대방향으로 이동 시 게임오버 구현 | 게이트 통과 후 우측으로 빠져나올 때 위로 향하면 게임오버가 되는 버그 발생 |
| Score | 23.06.04 | 정호용 | 스코어 보드 반영  (완성) |  |
| Map | 23.06.05 | 정호용 | 통과 조건 충족 시 다음 맵으로 이동, 모든 맵 통과 시 올 클리어 창 뜨고 끌 수 있게 수정 | 다음 맵으로 넘어갈 때 뱀이 맵에 걸쳐서 나오는 경우 맵이 깨지는 버그 있음  다음 맵으로 넘어갈 때 뱀 길이가 8,12 등으로 임의 고정됨 |
| Map | 23.06.05 | 구수연 | 맵 변경 시 스네이크 몸 길이 초기화, 스네이크 출현 위치 수정 | 모든 맵 통과 시 segmentation fault 오류 발생 |
| Map | 23.06.05 | 정호용 | 모든 맵 통과 시 segmentation fault 오류 해결 |  |
| Map | 23.06.06 | 구수연 | 맵 속 불필요 immune wall 삭제, game rule #6 의 최대길이 구현 | 맵 통과 시 마지막에 획득한 아이템의 체크가 안되는 버그 발생 |
| Map | 23.06.06 | 구수연 | 맵 별 소요시간 구현, 맵별 점수 구현, 미션 수정, 맵 통과 시 마지막에 획득한 아이템 체크 되게 수정 |  |
| Main | 23.06.06 | 정호용 | 게임화면 자동크기 조절 기능 구형 |  |
| Main | 23.06.19 | 정호용 | 게임 처음 시작 시 안내 문구 터미널창에 출력되도록 수정 / 최종 업데이트 |  |
|  | | | | |

# 개발 내용 및 결과물

## 목표

|  |  |  |
| --- | --- | --- |
| **적용단계** | **내용** | **적용 여부** |
| 1단계 | Map의 구현 | 적용 |
| 2단계 | Snake 표현 및 조작 | 적용 |
| 3단계 | Item 요소의 구현 | 적용 |
| 4단계 | Gate 요소의 구현 | 적용 |
| 5단계 | 점수 요소의 구현 | 적용 |

**1 단계. Map의 구현**

4개의 서로 다른 맵을 구성합니다. 맵은 3차원 배열로 이루어져 있는데 맵을 구분하는 번호, 맵의 가로, 세로로 설정합니다. 맵의 구성요소는 색깔로 구분하는데, map[selmap][i][j] 에서 1은 wall, 2는 immune wall, 3은 뱀의 머리, 4는 뱀의 몸체, 5는 성장 아이템, 6은 독 아이템, 7은 게이트로 구분합니다.

**2 단계. Snake 표현 및 조작**

snake를 표현하기 위해 좌표값을 갖는 snakebody 객체를 선언하여 vector에 저장함으로써 snake를 구성합니다. vector에서 첫 번째 요소는 head로 나머지는 body로 구별되며 map에서는 head는 3으로 body는 4로 나타냅니다.

snake는 스테이지 맵에서 벽을 제외한 위치 3곳에서 랜덤하게 생성되며 각 스테이지마다 생성 위치가 다릅니다. snake는 사용자가 입력하는 방향키의 방향으로 이동하며 사용자가 방향키를 입력하지 않으면 진행방향대로 계속 이동합니다. 그리고 진행방향과 반대의 방향을 입력 시 게임오버됩니다.

snake의 이동은 입력받은 진행방향을 토대로 snake의 head를 이동시키고 나머지 body가 따라 움직입니다. snake의 이동은 아이템을 먹거나, 게이트를 통과하거나, 벽을 만나는 상황에 따라 그 효과가 구현됩니다.

**3 단계. Item 요소의 구현**

Item은 뱀의 몸 길이를 늘리는 Growth item과 뱀의 몸 길이를 축소시키는 Poison item으로 구성되어 있습니다. Growth item은 5로, Poison item은 6으로 구분되어 map에 생성되며, 각각 초록색과 빨간색으로 출력됩니다. map에는 총 세 개의 아이템이 생성될 수 있으며, Growth item의 개수를 기준으로 Poison item의 수가 결정됩니다. 이때 Growth item의 개수는 0~3의 수 중 랜덤하게 결정됩니다. 이러한 아이템들은 8초 간격으로 새로 생성되는데, 뱀이 usleep() 함수로 0.17초마다 한 칸 씩 이동하는 것을 고려해 cnt 변수로 Item 생성 시점을 계산합니다.

**4 단계. Gate 요소의 구현**

**Gate의 구성 및 생성 기준**

Gate는 in\_gate와 out\_gate로 구성되어 있습니다. Map의 wall 부분 중 임의의 위치에 한 쌍의 Gate가 생성되는데, 생성된 두 개의 Gate 중 뱀이 먼저 통과한 Gate가 in\_gate가 되며, 나머지 한 개의 Gate가 out\_gate가 됩니다. Gate는 7로 구분되어map에 생성되며, Yellow 색으로 출력됩니다. 또한, 게임 시작 13초 이후에 처음으로 생성되며, 13초 간격으로 랜덤한 위치에 새로운 Gate가 생성됩니다. Item 생성과 마찬가지로 뱀이 usleep() 함수로 0.17초마다 한 칸씩 이동하는 것을 고려해 cnt2 변수로 Gate 생성 시점을 계산합니다.

Item 생성과 달리, Gate 생성에는 추가적 메커니즘이 요구됩니다. 뱀이 Gate를 통과하는 동안은 Gate가 사라지거나 다시 생성될 수 없습니다. 이러한 조건 구현을 위해 gatecnt 변수로 뱀이 Gate를 완전히 통과했는지 여부를 확인합니다. 이때 gatecnt 변수값과 현재 뱀의 몸 길이 값이 같아지면 뱀이 Gate를 완전히 통과했음을 의미합니다. 이러한 조건 만족 시 Gate가 새로운 장소에 임의적으로 다시 생성됩니다. 즉, Gate는 1) 게임 시작 13초 이후 13초 간격으로, 2) 뱀이 Gate를 완전히 통과했을 때의 두 가지 조건 중 하나를 만족했을 때 새로 생성됩니다.

**Gate 통과**

두 개의 Gate 중 하나의 Gate에 뱀이 진입하게 되면 다른 Gate로 진출하게 되는데, 이때 뱀의 진행 방향에 있어 일련의 규칙이 적용됩니다. 뱀은 벽이 없는 빈 공간으로만 진출이 가능하며, 만약 생성된 out\_gate가 Map의 가장자리 부분에 위치한 경우, 뱀의 진출 방향은 항상 Map의 안쪽 방향으로 고정됩니다. 예를 들어, out\_gate가 상단 벽에 있을 경우, 뱀의 진출 방향은 아래 방향이 되고, out\_gate가 하단 벽에 있을 경우 뱀의 진출 방향은 위쪽 방향이 되는 것입니다. 마찬가지로 out\_gate가 좌측 벽에 있으면 오른쪽 방향으로, out\_gate가 우측 벽에 있으면 뱀의 진출 방향은 왼쪽 방향이 됩니다. 이를 구현하기 위해 snake\_direction 변수 값을 조정함으로써 뱀 머리의 다음 위치를 정의하도록 합니다.

만약 생성된 out\_gate가 Map의 중앙에 있다면 뱀의 진출 방향은 진입 방향에 의해 결정되는데, 진입 방향과 일치하는 방향이 우선됩니다. 그러나 뱀은 벽이 없는 빈 공간으로만 진출이 가능하기 때문에 out\_gate 주변에 장애물이 있는지 확인해야 합니다. 만약 진출 방향에 장애물이 있다면 현재의 진출 방향에서 시계방향으로 진출 방향을 수정하며, 수정된 방향에 장애물이 있는지 검사합니다. 이러한 과정을 진출 방향에 장애물이 없을 때까지 반복합니다. 장애물이 없는 진출로를 찾아낸 경우, 그 방향으로 뱀이 진출합니다. 예를 들어, out\_gate가 맵 중앙에 있고, 위쪽으로 진입했는데 out\_gate의 위쪽에 Wall 등 장애물이 있는 경우, 시계방향으로 돌아서 장애물이 없는지 검사하고, 진출이 가능하다면 오른쪽 방향으로 진출하게 되는 것입니다.

**5 단계. 점수 요소의 구현**

게임 화면의 하단에 각각 미션 보드와 스코어 보드를 구현합니다. 미션 보드에는 각 스테이지별로 획득해야 하는 성장 아이템과 독 아이템의 개수, 게이트 통과 횟수를 표현하고 달성해야 하는 뱀의 몸길이를 표현합니다. 이는 2차원 배열로 구현합니다. 배열의 인덱스에 접근해서 인덱스보다 수가 크면 달성한 것으로 간주합니다. 모든 요구사항이 충족되었을 때, 그 스테이지는 통과된 것이며 통과했다는 결과창을 띄웁니다

## 2.2 개발 내용 및 결과물

### 2.2.1 개발 내용

**0 단계. 실행 전 작업**

game.h 파일에 게임에서 쓰이는 다양한 변수 및 함수들을 선언합니다. 창을 띄우는 WINDOW\* win부터 score,reset함수, 스네이크, 시간, 아이템과 관련된 변수들이 이에 해당합니다.

**1 단계. Map의 구현**

맵은 총 4개로 이루어져 있으며, 마지막 맵으로 갈수록 난이도가 증가합니다. 맵의 크기는 세로 30, 가로 60으로 설정했습니다. 맵의 모양은 map.h의 int map의 배열을 읽어들이는 것으로 구현했습니다. 맵은 색상별로 구분하되, map[selmap][i][j] 의 값을 기준으로 합니다. 1은 wall, 2는 immune wall, 3은 뱀의 머리, 4는 뱀의 몸체, 5는 성장 아이템, 6은 독 아이템, 7은 게이트로 구분합니다. selmap은 맵을 구별하는 변수입니다. main.cpp와 map.cpp 에서는 게임 타이틀과 맵,미션,스코어 나타내는 윈도우 등을 설정합니다. 2중 반복문을 이용해 맵에 필요한 wall, 스네이크, 아이템 등을 출력합니다. 출력 시 map.h의 int map의 배열은 0,1,2의 정수로 구성 되어 있는데, 이 값을 토대로 빈 화면, wall, immune wall을 각각 출력하게 됩니다.

**2 단계. Snake 표현 및 조작**

**snake의 표현**

snake는 map에서 좌표값을 갖는 snake\_body객체로 구성되며 그 객체들은 snake vector에 저장되어 관리됩니다. snake는 map.h의 배열에서 벽을 나타내는 1을 제외한 0의 값을 갖는 좌표 3곳에서 랜덤하게 출현합니다. 출현 시 배열에서 snake의 head는 3, body는 4로 표현됩니다. snake는 헤드와 2개의 body로 이루어져 있으며 총 길이는 3입니다. 사용자의 방향키 입력이 없으면 진행방향으로 계속 이동합니다. snake는 0.25초 한 칸씩 이동합니다.

**snake의 조작**

snake의 초기 방향은 진행 방향으로 설정되어 있고 사용자의 입력을 받아 head의 진행 방향이 결정됩니다. 이때 기존의 진행방향과 반대의 방향을 입력할 경우 게임이 오버됩니다. head는 사용자의 입력을 받아 새 좌표로 이동하는데 head가 벽에 닿거나 빨간색 아이템을 먹어 뱀의 길이가 3보다 작아지면 게임 오버됩니다. snake의 head가 초록색 아이템을 먹으면 길이가 증가하고 빨간색 아이템을 먹으면 길이가 감소합니다. head가 gate를 만나면 반대 쪽 게이트로 head의 좌표가 이동하고 body도 따라 이동합니다.

**3 단계. Item 요소의 구현**

make\_item() 함수를 통해 아이템이 새로 생성됩니다. 일정 시간 간격으로 아이템 생성 시 기존 맵에 존재하던 아이템을 없애주어야 하므로 make\_item() 함수 실행 뒤 가장 먼저 initialize\_item() 함수를 통해 맵을 초기의 상태와 같도록 만들어줍니다. Initialize\_item() 함수에서는 이중 for문을 사용해 맵 전체의 데이터를 확인해 growth\_item을 의미하는 5 혹은 poison\_item을 의미하는 6을 만나면 빈 공간을 나타내는 0으로 바뀌도록 합니다.

맵에 나타날 수 있는 아이템의 최대 개수는 3개이므로, 먼저 srand()와 time()함수를 함께 사용해 난수가 생성되도록 시드값을 줍니다. num\_growth 변수에 growth\_item의 수를 0~3 사이로 랜덤하게 결정한 뒤, 그에 따라 poison\_item의 수가 결정되도록 했습니다. 아이템의 생성 위치를 정하기 위해서는 rand() 함수로 임의의 수를 생성하고, 맵의 가로, 세로 크기가 30인 것과 가장자리의 벽을 고려해 28로 나눈 나머지에 1을 더해주어 1~28 사이의 수로 새로 생성될 아이템의 임시 좌표값을 생성합니다.

이때 생성된 좌표값의 맵 데이터가 0이 아니라면, 즉 빈 공간이 아니라면 다시 새로운 좌표값을 구하도록 했습니다. 만약 생성된 좌표값에 아이템 생성이 가능하면 현재 시점까지 생성된 아이템 개수가 위에서 결정된 num\_growth보다 작을 때까지는 해당 좌표값의 맵 데이터를 5로 수정함으로써 growth\_item을 생성하고, 총 아이템 개수가 3이 될 때까지 해당 과정을 반복하며 생성된 좌표값의 맵 데이터를 6으로 수정함으로써 poison\_item을 생성합니다.

또한, 뱀이 game() 함수에서의 usleep() 함수를 통해 0.25초마다 이동하는 것을 고려해 cnt 변수로 아이템 생성 빈도를 조정합니다. 8초마다 아이템이 새로 생성되게 하기 위해 cnt가 32으로 나눠질 때마다 make\_item() 함수가 호출되어 아이템이 새로 생성되도록 합니다.

만약 새로운 뱀의 머리가 아이템을 만나면, 아이템의 종류에 따라 조건에 맞게 뱀의 몸 길이가 조절되도록 합니다. 만약 growth\_item을 마주치면 snake 벡터에 요소를 하나 추가함으로써 몸 길이를 1만큼 늘리고, poison\_item을 마주치면 snake 벡터에서 요소를 제거하고 마지막 snake body에 해당하던 맵 데이터 값을 0 (빈 공간)으로 바꿔줌으로써 뱀의 몸 길이를 1만큼 줄입니다.

**4 단계. Gate 요소의 구현**

**Gate의 구성 및 생성 기준**

make\_gate() 함수를 통해 Gate가 생성됩니다. 함수 첫 라인의 cnt2 변수는 시간에 따른 Gate의 생성 빈도를 조정하는 데 사용되는 변수인데, 만약 make\_gate() 함수가 호출되면 처음부터 시간을 체크하도록 하기 위해 cnt2를 0으로 초기화해줍니다. 또한, 아이템 생성시 기존에 생성된 아이템을 제거하기 위해 initialize\_item() 함수를 호출해주었던 것과 마찬가지로, make\_gate() 함수 내부에서 기존의 Gate를 제거하기 위해 initialize\_gate() 함수가 가장 먼저 호출됩니다. Initialize\_gate() 함수에서는 이중 for문을 사용하여 맵 전체의 데이터를 확인해 Gate를 의미하는 7을 만나면 벽을 나타내는 1로 데이터 값을 바꿉니다.

Gate 관련 데이터들을 관리하기 위해 뱀의 몸체를 표현하기 위해 생성되었던 gate의 이름을 가진 snake\_body 타입의 벡터를 사용합니다. 기존에 저장되어있는 Gate 정보를 지우기 위해 gate.clear() 함수로 내용을 지워주고 새로운 Gate를 생성합니다. Item을 생성했던 것과 같이 srand() 함수와 time() 함수를 활용해 난수 생성의 시드값을 지정해줍니다. in\_gate\_row, in\_gate\_col, out\_gate\_row, out\_gate\_col의 네 가지 변수에 rand() 함수로 생성된 난수값을 30으로 나눈 나머지값을 저장합니다. 즉, 각 변수는 0~29의 값을 갖게 됩니다.

Gate는 wall 부분에서만 생성될 수 있습니다. 따라서 각 좌표값이 지정되면, while문 내부에서 맵에서의 각 좌표값에 해당하는 맵 데이터가 wall을 의미하는 1인지를 확인합니다. 만약 wall 부분에 해당한다면, 생성된 좌표의 데이터 값을 7로 수정함으로써 in\_gate를 먼저 생성합니다. 같은 논리로 out\_gate도 생성한 후, gate 벡터에 지금까지 생성된 좌표값들을 in\_gate와 out\_gate의 데이터로 구분해 넣어줍니다. 이때의 in\_gate와 out\_gate의 변수명은 임시 별칭입니다.

Gate는 1) 시간, 2) Gate 통과의 두 가지 조건 중 하나를 만족했을 때 새로 생성됩니다. 1) 시간 조건의 경우, 게임 시작 13초 이후부터 13초 간격으로 Gate가 새로 생성됩니다. 이를 만족하기 위해 뱀이 game() 함수에서의 usleep() 함수를 통해 0.25초마다 이동하는 것을 고려해 cnt2 변수로 Gate 생성 빈도를 조정합니다. 13초마다 Gate가 생성되게 하기 위해 cnt2가 52으로 나누어떨어질 때 make\_gate() 함수가 호출되도록 합니다.

2) Gate 통과 조건의 경우, 뱀이 Gate를 완전히 통과했을 때 Gate가 새로 생성되며, 뱀이 Gate를 통과하고 있는 중에는 Gate가 사라지거나 새로 생성될 수 없습니다. 이러한 조건을 구현하기 위해 뱀이 Gate를 통과하는 중인지 확인해야 하는데, 이때 gatecnt 변수와 gate\_crossing 변수를 활용합니다. 만약 생성된 새로운 뱀의 머리가 Gate를 마주치면 in\_gate와 out\_gate가 무엇인지 정의한 후 cross\_gate() 함수를 호출합니다. 이후 Gate\_crossing 변수 값을 true로 바꿔줌으로써 뱀이 Gate를 통과하고 있음을 표현하고, cross\_cnt의 값을 1 증가시킴으로써 뱀이 Gate를 통과하는 중임을 의미하도록 합니다. 이때 만약 cross\_cnt가 snake의 몸 길이와 같아지면 뱀이 Gate를 완전히 통과했음을 의미합니다. 완전히 통과했다면, 조건문을 활용해 cross\_cnt는 0으로, gate\_crossing은 false로 초기화하고 make\_gate 함수를 호출합니다.

**Gate 통과**

만약 생성된 새로운 뱀의 머리가 Gate를 마주치면 맵에서의 해당 좌표값을 확인해 앞서 생성된 gate벡터의 두 좌표 중 어떤 것과 일치하는지 확인합니다. 일치하는 좌표를 가진 벡터의 요소를 진입 Gate, 즉 in\_gate로, 나머지 하나의 벡터 요소를 out\_gate로 지정합니다. 이후 뱀의 게이트 통과를 위해 각 Gate의 좌표값을 인자로 주고 cross\_gate() 함수를 호출합니다. 이때 뱀의 진행 방향에 있어 일련의 규칙이 적용되는데, cross\_gate() 함수가 뱀의 다음 진행 방향을 결정하는 역할을 합니다.

뱀은 벽이 없는 빈 공간으로만 진출이 가능하며, 만약 생성된 out\_gate가 Map의 가장자리 부분에 위치한 경우, 뱀의 진출 방향은 항상 Map의 안쪽 방향으로 고정됩니다. 인자로 받은 좌표값을 확인해 out\_gate의 위치를 파악하고, 만약 맵의 가장자리에 위치한다면 switch-case문에서 규칙에 따른 진출 방향에 맞게 snake\_direction 변수값을 조정함으로써 다음 뱀의 머리 위치가 올바른 곳에 생성될 수 있도록 합니다. 이때 0,1,2,3은 각각 위, 오른쪽, 아래, 왼쪽을 의미합니다. 예를 들어, out\_gate가 상단 벽에 있을 경우, 뱀의 진출 방향은 아래 방향이 되고, out\_gate가 하단 벽에 있을 경우 뱀의 진출 방향은 위쪽 방향이 되도록 합니다.

만약 생성된 out\_gate가 Map의 중앙에 있다면 뱀의 진출 방향은 진입 방향에 의해 결정되는데, 진입 방향과 일치하는 방향이 우선시됩니다. 또한, 뱀은 벽이 없는 빈 공간으로만 진출이 가능하기 때문에 시계방향 순서로 진출로 방향에 장애물이 있는지 확인하며 장애물이 없는 진출로의 방향으로 진출하게 됩니다. 이러한 기능을 구현하기 위해 switch-case문을 활용해 snake\_direction을 조건으로 넣어 진출로 방향의 장애물 여부를 확인하고, 장애물이 있다면 진출 방향을 시계방향으로 회전시킨 후 다시 조건문을 실행합니다. 만약 진출로에 장애물이 없다면, status 변수에 1을 저장해 new\_head의 좌표값이 변했음을 의미하도록 하고, out\_gate의 좌표를 기준으로 새로운 뱀 머리의 위치를 지정한 뒤 반복문을 빠져나와 함수 실행이 종료되도록 합니다.

예를 들어, out\_gate가 맵 중앙에 있고, 위쪽으로 진입했는데 out\_gate의 위쪽에 Wall 등 장애물이 있는 경우, 시계방향으로 돌아서 장애물이 없는지 검사하고, 진출이 가능하다면 오른쪽 방향으로 진출하게 되는 것입니다.

**5 단계. 점수 요소의 구현**

미션과 스코어를 보여주는 윈도우를 띄웁니다. 미션은 2차원 배열을 이용해 각 맵 별로 다르게 설정했습니다. 순서대로 스네이크 몸길이, 성장 아이템, 독 아이템, 게이트 사용 횟수입니다. 저희 팀은 아이템과 게이트 달성 횟수를 나름의 기준을 세워 점수로 변환하는 것도 구현해 봤습니다. 스코어를 보여주는 윈도우에서 각 조건 달성 여부를 체크박스 형태로 볼 수 있고, 맨 아래에 스테이지 점수를 볼 수 있습니다. 또한 마지막 맵을 통과할 경우 각 맵별 점수를 결과 화면에 띄워줍니다. 조건 달성 시 해당 맵은 통과했다고 결과창을 띄워주며, 소요시간을 계산해 점수와 함께 결과창에서 볼 수 있습니다. 다중 조건문을 이용해 모든 스테이지를 통과했을 때의 결과창을 구현할 수 있었습니다. 또한 다음 맵을 시작하기 전에 모든 아이템 변수 및 뱀길이를 초기화합니다.

### 2.2.2 시스템 구조 및 설계도

**0단계. 게임에 기본이 되는 변수 설정**

|  |
| --- |
| **<game.h>**  #include <time.h>  WINDOW\* win1; // game board  WINDOW\* win2; // misson board  WINDOW\* win3; // score board  ->게임에서 사용될 3개의 윈도우 창을 각각 선언해 줍니다.  void score();  ->점수 및 미션 띄우는 윈도우를 설정합니다.  int itemcnt =0;  int poisoncnt = 0;  int gatecnt=0;  ->성장 아이템과 독 아이템을 먹었을 때 개수와 게이트 지나간 횟수를 세어주는 변수입니다.  int selmap = 0;  ->맵 번호 관련된 변수입니다.  int head\_dir = 2;  int MaxSize = 0;  int size = 0;  ->뱀의 머리방향, 최대 길이, 몸길이 나타내는 변수입니다.  int starttime,endtime,gettime;  -> 각각 시작 시간, 끝난 시간, 그리고 총 소요시간 나타내는 변수입니다.  int stagescore;  -> 각 스테이지별 점수입니다.  int scoreset[4] = {0};  -> 스테이지별 점수를 저장하는 배열입니다.  void result(int num);  -> 결과창 띄우는 함수입니다. 인자로 어떤 수가 들어가느냐에 따라 스테이지 클리어, 스테이지 올 클리어, 게임오버 3가지로 나뉩니다.  void reset();  -> 맵이 바뀔때마다 성장 아이템, 독아이템, 게이트 통과 횟수 초기화하는 함수입니다.  void game();  -> 게임 시작하는 함수입니다. |

**1 단계. Map의 구현**

|  |
| --- |
| **<map.cpp>**  #include "game.h"  void set\_color(){    start\_color();    init\_pair(10, COLOR\_BLACK, COLOR\_MAGENTA);    init\_pair(11, COLOR\_BLACK, COLOR\_WHITE);    init\_pair(12, COLOR\_WHITE, COLOR\_WHITE); // null box    init\_color(COLOR\_GRAY, 270, 270, 270);      // Gray    init\_pair(1, COLOR\_GRAY, COLOR\_GRAY);       // Wall    init\_color(COLOR\_BLACK, 0, 0,0);    init\_pair(2, COLOR\_BLACK, COLOR\_BLACK); // Immune Wall    init\_pair(3, COLOR\_CYAN, COLOR\_CYAN);       // Snake Head    init\_pair(4, COLOR\_BLUE, COLOR\_BLUE);       // Snake Body    init\_pair(5, COLOR\_GREEN, COLOR\_GREEN);       // Growth item    init\_pair(6, COLOR\_RED, COLOR\_RED);     //Poison item    init\_color(COLOR\_MAGENTA, 1000, 250, 500);    init\_pair(7, COLOR\_YELLOW,COLOR\_YELLOW); // gate  -> 성장 아이템, 독 아이템, 게이트, wall, immune wall등에 필요한 색상들을 설정합니다.  }  void back\_setting(){    bkgd(COLOR\_PAIR(10));    attron(COLOR\_PAIR(11));    border('|', '|', '-', '-', '.', '.', '.', '.');  mvprintw(1, 47, "#SNAKE GAME made by HoSu#");  ->게임 타이틀을 출력합니다.  attroff(COLOR\_PAIR(11));  attroff(COLOR\_PAIR(10));    mvprintw(36, 15, "<<Press arrow keys to move snake!>>");  -> 간단한 게임 방식을 출력해줍니다.    attroff(COLOR\_PAIR(10));    refresh();  }  void window\_setting(WINDOW\* win1){    wbkgd(win1, COLOR\_PAIR(10));    wattron(win1, COLOR\_PAIR(10));    wborder(win1, '|', '|', '-', '-', '.', '.', '.', '.');  }  -> 맵을 표시할 윈도우를 띄웁니다.  **…중략…**  // color apply function-------------------------------------------------------------  void color\_apply(int selmap,WINDOW\* win1){    // color apply    for (int i = 0; i < 30; i++) { //map print      int temp=0; //temporary x positon. to equalize the map size        for (int j = 0; j < 30; j++) {          // null box is white            if (map[selmap][i][j] == 0) {                wattron(win1, COLOR\_PAIR(12));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]); //space in behind                wattroff(win1, COLOR\_PAIR(12));                temp+=2; //update x position            }  -> 아무것도 없는 빈 화면을 윈도우 1에 출력합니다. 색상은 0번이며, 흰색입니다. 여기서는 스네이크가 자유롭게 이동할 수 있습니다.            //Wall is gray            else if (map[selmap][i][j] == 1) {                wattron(win1, COLOR\_PAIR(1));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]);                wattroff(win1, COLOR\_PAIR(1));                temp+=2;            }  -> 일반 벽을 윈도우 1에 출력합니다. 색상은 1번이며, 회색입니다. 여기서는 충돌 시 게임 오버되며, 게이트가 생성될 수 있는 벽입니다.            // Immune Wall is black            else if (map[selmap][i][j] == 2) {                wattron(win1, COLOR\_PAIR(2));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]);                wattroff(win1, COLOR\_PAIR(2));                temp+=2;            }  -> 게이트가 만들어질 수 없는immune wall을 윈도우 1에 출력합니다. 색상은 2번이며, 검정색입니다. 주로 스네이크가 접근할 수 없는 맵의 모서리 부분 혹은 맵 속의 벽 중 스네이크가 접근할 수 없는 부분이 해당됩니다.            // Snake Head is cyan            else if (map[selmap][i][j] == 3) {                wattron(win1, COLOR\_PAIR(3));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]);                wattroff(win1, COLOR\_PAIR(3));                temp+=2;            }  -> 스네이크의 머리를 윈도우 1에 출력합니다. 색상은 3번이며, cyan 색입니다. 이 스네이크의 머리 부분은 1개를 넘을 수 없습니다.            // Snake Body is blue            else if (map[selmap][i][j] == 4) {                wattron(win1, COLOR\_PAIR(4));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]);                wattroff(win1, COLOR\_PAIR(4));                temp+=2;            }  -> 스네이크의 몸체를 윈도우 1에 출력합니다. 색상은 4번이며, 파란색입니다. 성장 아이템을 획득할 시 몸체 부분이 증가하고, 독 아이템 획득 시 몸체 부분이 감소합니다.            // Growth item is green            else if (map[selmap][i][j] == 5) {                wattron(win1, COLOR\_PAIR(5));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]);                wattroff(win1, COLOR\_PAIR(5));                temp+=2;            }  -> 성장 아이템을 윈도우 1에 출력합니다. 색상은 5번이며, 초록색입니다. 위치와 개수가 랜덤이고 출현 시간은 일정하며, 스네이크의 머리 부분의 좌표와 아이템의 배열이 일치할 때 획득한 것으로 간주됩니다.            //Poison item is red            else if (map[selmap][i][j] == 6) {                wattron(win1, COLOR\_PAIR(6));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]);                wattroff(win1, COLOR\_PAIR(6));                temp+=2;            }  -> 독 아이템을 윈도우 1에 출력합니다. 색상은 6번이며, 빨간색입니다. 위치와 개수가 랜덤이고 출현 시간은 일정하며, 스네이크의 머리 부분의 좌표와 아이템의 배열이 일치할 때 획득한 것으로 간주됩니다.            // Gate is Bright pink            else if (map[selmap][i][j] == 7) {                wattron(win1, COLOR\_PAIR(7));                mvwprintw(win1, i, temp, "%d ", map[selmap][i][j]);                wattroff(win1, COLOR\_PAIR(7));                temp+=2;            }        }  }  -> 게이트를 윈도우 1에 출력합니다. 색상은 7번이며, 노란색입니다. 위치가 랜덤이며, 스네이크의 머리 부분의 좌표와 아이템의 배열이 일치할 때 획득한 것으로 간주됩니다.    wrefresh(win1);  }  -> 0,1,2는 map.h 에 저장된 배열의 인덱스에 속해있는 정수값을 토대로 구현됩니다. 예를 들어, map[selmap][i][j] == 1이라면, 1에 해당되는 색상으로 윈도우 1에 출력하게 됩니다.  temp를 2씩 증가시켜 가로 세로 비율을 1:1로 맞춰줍니다.  **<main.cpp>**      starttime = time(NULL);  -> 게임이 시작될 때 시작 시간을 측정합니다.    //background setting---------------------------------------------------      back\_setting();      win1 = newwin(30, 60, 3, 30); //game board      win2 = newwin(15, 35, 38, 26); //mission board      win3 = newwin(15, 35, 38, 61); //score board  -> 게임 내 띄울 윈도우 설정 : 각각 스테이지, 미션, 점수를 표시하는 윈도우입니다. 맵의 크기는 30,60이고 미션과 점수 윈도우의 크기는 각각 15,35 입니다.      window\_setting(win1);      reset();  -> 게임 시작 시 항상 초기화함수를 사용합니다. |

**2 단계. Snake 표현 및 조작**

|  |
| --- |
| <game.h>  struct snakebody{  int r, c;  snakebody(int row, int col);  snakebody();  };  snakebody::snakebody(int row, int col){  r=row;  c=col;  }  snakebody::snakebody(){  r=0;  c=0;  }  vector<snakebody> snake; |

2차원 배열인 map에서 좌표값(row,col)을 갖는 snakebody 구조체를 생성합니다.

|  |
| --- |
| <snake.cpp>  void snake\_make(vector<snakebody>& snake, int selmap){  //possible location (map, row, colum)  int snake\_positon[4][3][2]={  {{4, 23}, {6, 8}, {18, 16}},  {{15, 14}, {21, 9}, {2, 13}},  {{4, 15}, {27, 18}, {16, 20}},  {{26, 22}, {6, 12}, {3, 10}}  };  // make random number-> random location and snake location setting  srand((unsigned)time(0));  int sel\_r = rand()%3;  //head setting ->snake <vector> has head positon and body position  //######################  // #####Important######  //#######################  //ex) row ==4 colum==23  for(int i=0; i<3; i++){  snake.push\_back(snakebody(snake\_positon[selmap][sel\_r][0], snake\_positon[selmap][sel\_r][1]+i));}  map[selmap][snake[0].r][snake[0].c] = 3; //snake head is 3  // body setting  for(int i=1; i<3; i++){  map[selmap][snake[i].r][snake[i].c] = 4;} //snake body is 4  } |

snake\_make()함수는 snake vector,와 스테이지의 숫자를 뜻하는 selmap를 인자로 받습니다. snake\_position의 3차원 배열은 각 스테이지의 뱀이 나타날 위치를 저장하고 있습니다. 첫 번째 배열의 요소들은 각 스테이지를 나타내고 그 안에 두 번째 배열의 요소 뱀이 나타날 좌표를 나타냅니다. 마지막으로 세 번째 배열의 각 요소들은 map에서의 row과 column 값을 나타냅니다. 각 스테이지에서 뱀이 나타날 위치는 rand()%3를 통해 랜덤하게 선택되어 sel\_r변수에 저장됩니다. 그리고 설정된 좌표를 가진 snakebody객체가 for문을 통해 snake vector에 차례로 pushback되며 그때 마다 snakebody의 c의 값은 i만큼 증가하여 저장됩니다.

|  |
| --- |
| <snake.cpp>  // input directon function---------------------------------------------------------  int input\_Direction(int input\_key, int\* snake\_direction){  switch(input\_key) {  case KEY\_UP:  if(\*snake\_direction == 2){  result(3);  delwin(win1);  exit(0);  }  \*snake\_direction =0;  return 0;  break;  case KEY\_RIGHT:  if(\*snake\_direction == 3){  result(3);  delwin(win1);  exit(0);  }  \*snake\_direction =1;  return 1;  break;  case KEY\_DOWN:  if(\*snake\_direction == 0){  result(3);  delwin(win1);  exit(0);  }  \*snake\_direction =2;  return 2;  break;  case KEY\_LEFT:  if(\*snake\_direction == 1){  result(3);  delwin(win1);  exit(0);  }  \*snake\_direction =3;  return 3;  break;  }  return \*snake\_direction; //if input is null  } |

input\_Direction() 함수는 새로 입력 받은 키인 input\_key와 snake의 진행방향인 snake\_direction를 인자로 받습니다. input\_Direction() 함수는 사용자에게서 받은 방향키의 입력을 통해 snake\_direction를 바뀝니다. 키보드 방향키의 입력이 올 때 switch case문을 통해 KEY\_UP, KEY\_LEFT, KEY\_RIGHT, KEY\_DOWN를 각각 처리하여 snake의 방향을 설정해주었고 함수는 변경된 snake\_direction를 리턴합니다. 만약 방향키의 입력이 인자로 받은 기존의 snake\_direction와 비교하여 반대 방향인 경우 result(3)함수를 통해 게임오버 화면을 불러오며 게임이 종료됩니다.

|  |
| --- |
| <snake.cpp>  void snake\_move(vector<snakebody>& snake, vector<snakebody>& gate, int\* snake\_direction, bool\* gate\_crossing, int\* cross\_cnt, int\* cnt2){  int new\_head\_row=snake[0].r;  int new\_head\_col=snake[0].c;  int new\_body\_row=snake[snake.size()-1].r;  int new\_body\_col=snake[snake.size()-1].c;  int out\_gate\_row, out\_gate\_col;  switch (\*snake\_direction) {  case 0: //up  new\_head\_row--;  break;  case 1: //right  new\_head\_col++;  break;  case 2: //down  new\_head\_row++;  break;  case 3: //left  new\_head\_col--;  break;  default: //another key  break;  }  if (map[selmap][new\_head\_row][new\_head\_col]==5){ //encounter growth item  snake.push\_back(snakebody(new\_body\_row, new\_body\_col)); //body\_length+=1  itemcnt++; //item\_count +=1  ~~usleep(17000); // if snake gets growth item, snake's speed will be faster.~~  //아이템 획득 시 화면이 버벅이는 문제로 제거하였습니다.  }  else if (map[selmap][new\_head\_row][new\_head\_col]==6){ //encounter poison item  snake.pop\_back(); //body\_length-=1  map[selmap][snake[snake.size()].r][snake[snake.size()].c] = 0;  poisoncnt++; //poison\_count +=1  if (snake.size() < 3){ //if snake size is shorter than 3  result(3);  delwin(win1);  exit(0);  }  }  else if (map[selmap][new\_head\_row][new\_head\_col]==7){ //encounter gate  if (new\_head\_row==gate[0].r&&new\_head\_col==gate[0].c){ //gate[0] is in\_gate  out\_gate\_row=gate[1].r; out\_gate\_col=gate[1].c; //gate[1] is out\_gate  }  else{ //gate[1] is in\_gate  out\_gate\_row=gate[0].r; out\_gate\_col=gate[0].c; //gate[0] is out\_gate  }  cross\_gate(out\_gate\_row, out\_gate\_col, &new\_head\_row, &new\_head\_col, snake\_direction, selmap); //set head location through the gate  gatecnt++;  \*gate\_crossing=true;  }  else if (map[selmap][new\_head\_row][new\_head\_col]==1){ //encounter wall  result(3);  delwin(win1);  exit(0);  }  else if (map[selmap][new\_head\_row][new\_head\_col]==2){ //encounter immune wall  result(3);  delwin(win1);  exit(0);  }  snake.insert(snake.begin(), snakebody(new\_head\_row, new\_head\_col)); //insert new head  snake.pop\_back(); //delete last snake body //body\_length-=1  for(int i=snake.size()-1; i>=0; i--){  if(i==snake.size()-1){ //remove last snake body  map[selmap][snake[snake.size()].r][snake[snake.size()].c] = 0;  continue;  }  else if(i!=0){ //move snake body  map[selmap][snake[1].r][snake[1].c] = 4;  continue;  }  map[selmap][snake[0].r][snake[0].c] = 3; //move snake head  }  if(\*gate\_crossing){ //if snake is crossing the gate  if(\*cross\_cnt==snake.size()){  \*cross\_cnt=0;  make\_gate(selmap, gate, cnt2); //new gate generates when snake crossed the gate  \*gate\_crossing=false;  }  \*cross\_cnt+=1;  }  refresh();  } |

이번 2단계에서는 snake\_move()함수의 아이템과 게이트와 관련된 부분은 각각 3단계와 4단계에서 설명함으로 해당 부분은 제외하고 설명합니다. 2단계에서 snake\_move()함수는 snake vector와 snake\_direction를 인자로 받습니다. snake의 이동을 구현하기 위해서는 새로운 snake head가 필요함으로 새 head의 좌표값을 저장할 new\_head\_row와 new\_head\_col를 선언하고 기존 헤드의 좌표를 할당해놓습니다. 그리고 switch문을 통해 인자로 받은 snake\_direction의 값에 따라 new\_head\_row와 new\_head\_col 값을 재설정 합니다. 만약 재설정한 새로운 head의 좌표가 map에서 1 또는 2와 같다면 snake의 head가 벽에 닿은 것으로 result(3)를 통해 게임오버 화면을 출력하고 게임오버 됩니다. 만약 게임오버가 되지 않았다면 snake vector의 snake.begin()위치에 새 head의 좌표값을 가진 snakebody객체를 insert합니다. 그리고 snake vector의 마지막 요소 값을 pop합니다. 그리고for문을 snake.size()-1값부터 0까지 돌며 map에 바뀐 snake vector의 snakebody객체의 좌표를 적용시킵니다. 이 과정에서 pop를 통해 삭제된 snake vector 요소의 좌표 위치의 값을 0으로 바꿉니다. 그리고 새로 추가된 head의 좌표 위치의 값을 4로 바꾼 다음 나머지 snake vector 요소들의 좌표 위치의 값을 3으로 설정합니다. 마지막으로 refresh()함수를 통해 변경사항을 win1에 반영합니다.

|  |
| --- |
| <main.cpp>  //snake\_positon---------------------------------------------------------------------------  snake\_make(snake,selmap);  make\_item(selmap);  color\_apply(selmap, win1);  refresh();  //Snake moving --------------------------------------------------------------------------------------  int snake\_direction =1;  int cnt=0;  int cnt2=0;  bool gate\_crossing=false;  int cross\_cnt=0;  while(TRUE){  score();  stagescore = itemcnt\*200 - poisoncnt\*100 + gatecnt\*100;  // directon decide  keypad(stdscr, TRUE);  nodelay(stdscr, TRUE); //to move snake without input  int input\_key=getch();  if(input\_key==KEY\_F(1)){  break;  }  snake\_direction=input\_Direction(input\_key, &snake\_direction);  snake\_move(snake, gate, &snake\_direction, &gate\_crossing, &cross\_cnt, &cnt2);  color\_apply(selmap, win1);  usleep(180000); //move snake per 0.18 sec  } |

게임이 진행되는 main.cpp에서 snake의 표현과 조작 부분을 보면 snake\_make()함수에 snake vector과 selmap를 인자로 전달하여 snake를 생성하였습니다. 그후 snake\_direction의 값을 1로 초기화한 상태에서 while문을 통해 계속 사용자로부터 입력을 받고 snake가 움직이게 합니다. kepad()는 사용자 키보드로부터 특수문자를 입력받게 하며 nodelay()는 입력이 없을 때 snake를 움직이게 합니다. getch()를 통해 사용자로부터 입력 받은 키 값을 input\_key에 할당합니다. 그리고 input\_key의 값이 KEY\_F(1)일 때 반복문을 빠져 나오며 게임을 종료합니다. snake\_direction은 input\_key와 snake\_direction를 인자로 받는 input\_Direction()함수를 통해 재설정되고 빠뀐 sanke\_directiond값을 인자로 받은 snake\_move()함수가 snake를 움직입니다. 움직이고 그 사이에 unsleep(180000) 매서드를 추가하여 0.18초마다 움직이게 합니다.

**3 단계. Item 요소의 구현**

<snake.cpp> line 76~98 : 아이템 생성

void make\_item(int selmap){

    initialize\_item(selmap); //initialize map

    srand((unsigned) time(0)); //seed

    int num\_growth = rand() % 4; //0~3 growth item can be generated

    int num=0,tmpX=0,tmpY=0; //num of total item, location

    while(num<3){ //while lower than maximum item num

      tmpX = rand()%28+1; //1~29 (except boundary)

      tmpY = rand()%28+1; //1~29

      if(map[selmap][tmpY][tmpX] != 0) //can place item only if it's blank

        continue;

      else if(num<num\_growth){

        map[selmap][tmpY][tmpX] = 5; //set the growth item location

        num++; //total\_item\_num

        continue;

      }

      map[selmap][tmpY][tmpX]=6; //set the poison item location

      num++; //total\_item\_num

    }

}

make\_item() 함수 실행 시 처음에 initialize\_map() 함수가 실행되어 기존에 생성된 아이템을 없앱니다. Srand()함수로 시드값을 정해주고, rand()함수로 생성된 임의의 정수를 4로 나눈 나머지를 growth\_item의 개수로 정합니다. 따라서 0~3의 수 중 하나로 growth\_item의 개수가 정해집니다. 변수 num, tmpX, tmpY는 각각 총 아이템의 개수와 임시 X좌표, 임시Y좌표의 값을 저장합니다.

총 3개의 아이템을 생성할 수 있으므로, 지금까지 생성된 아이템의 개수가 3보다 작을 때까지 while문을 실행합니다. 이때 맵의 가장자리에는 아이템이 생성될 수 없으므로 tmpX, tmpY는 1~29 사이의 값이 저장되도록 합니다. 만약 해당 좌표에 대한 맵의 데이터값이 0이 아니면, 즉 빈 공간이 아니면 아이템을 생성할 수 없으므로 건너뛰고 다시 랜덤 위치를 배정받습니다. 만약 생성된 새로운 좌표의 데이터가 0(빈 공간)이라면, 현재까지 생성된 아이템의 개수가 num\_growth보다 작을 때까지는 growth\_item을 의미하는 5를, 이외의 경우에는 poison\_item을 의미하는 6을 맵 데이터의 해당 위치에 저장합니다. 이로서 맵의 빈 공간 중 랜덤한 위치에 총 3개의 아이템이 무작위로 조합되어 맵에 생성됩니다.

<map.cpp> line 210~217: 맵에서 아이템 초기화

//initialize map function---------------------------------------------------

void initialize\_item(int selmap){

  for (int i=0; i<30; i++){

    for(int j=0; j<30; j++){

      if (map[selmap][i][j]==5||map[selmap][i][j]==6) //growth/poison\_item

        map[selmap][i][j]=0; //remove

    }

  }

}

Initialize\_item() 함수는 기존에 생성된 아이템을 맵에서 없애는 역할을 합니다. 이를 위해 이중 for문으로 맵 전체의 데이터를 확인하며 growth\_item과 poison\_item을 의미하는 5,6을 만나면 빈 공간을 의미하는 0으로 바꿔줍니다.

<main.cpp> line 52~90 in game(): 아이템 출현 빈도 조정

//Snake moving -------------------------------------------------------------------------------------

    int snake\_direction =1;

    int cnt=0;

    int cnt2=0;

    bool gate\_crossing=false;

    int cross\_cnt=0;

    while(TRUE){

      score();

      stagescore = itemcnt\*200 - poisoncnt\*100 + gatecnt\*100;

    // directon decide

      keypad(stdscr, TRUE);

      nodelay(stdscr, TRUE); //to move snake without input

      int input\_key=getch();

      if(input\_key==KEY\_F(1)){

        break;

    }

      snake\_direction=input\_Direction(input\_key, &snake\_direction);

      snake\_move(snake, gate, &snake\_direction, &gate\_crossing, &cross\_cnt, &cnt2);

      color\_apply(selmap, win1);

      usleep(180000); //move snake per 0.18 sec

      cnt+=1;

      cnt2+=1;

      if (cnt%32==0){ //generate item per 8sec

        make\_item(selmap);

        cnt=0;

      }

      if(cnt2%52==0){ //gate appears after 13sec

        make\_gate(selmap, gate, &cnt2);

      }

      color\_apply(selmap, win1);

      refresh();

  }

Main.cpp 파일의 game()함수 내부의 while문에서 cnt값을 통해 아이템의 생성 빈도를 조절합니다. Usleep()함수를 통해 0.18초마다 뱀의 이동이 이루어지므로, 이를 고려해 cnt가 32으로 나누어 떨어지면 make\_item() 함수를 호출함으로써 8초마다 새로운 아이템이 생성되도록 합니다. 이때 cnt=0으로 다시 초기화 해줍니다.

<snake.cpp>: line 101~140 in snake\_move(): 뱀이 growth\_item을 만나면 길이 증가, poison\_item을 만나면 길이 감소

int new\_head\_row=snake[0].r;

  int new\_head\_col=snake[0].c;

  int new\_body\_row=snake[snake.size()-1].r;

  int new\_body\_col=snake[snake.size()-1].c;

  int out\_gate\_row, out\_gate\_col;

**--중략--**

  if (map[selmap][new\_head\_row][new\_head\_col]==5){ //encounter growth item

      snake.push\_back(snakebody(new\_body\_row, new\_body\_col)); //body\_length+=1

      itemcnt++; //item\_count +=1

    }

  else if (map[selmap][new\_head\_row][new\_head\_col]==6){ //encounter poison item

      snake.pop\_back(); //body\_length-=1

      map[selmap][snake[snake.size()].r][snake[snake.size()].c] = 0;

      poisoncnt++; //poison\_count +=1

      if (snake.size() < 3){ //if snake size is shorter than 3

        result(3);

        delwin(win1);

        exit(0);

      }

    }

Snake.cpp 파일의 snake\_move() 함수 내부에서 조건문을 사용해 새로운 뱀 머리의 위치가 아이템을 만나면 몸 길이가 조정되도록 합니다. 만약 growth\_item을 만나면 snake 벡터에 new\_body\_row와 new\_body\_col 값을 가진 snakebody 요소를 추가함으로써 뱀의 몸 길이가 1만큼 증가되도록 합니다. Poison 아이템을 만나면 snake 벡터의 꼬리인 마지막 요소를 pop\_back()해서 없애주고, 꼬리가 위치했던 좌표의 데이터값을 0(빈 공간)으로 바꿔줌으로써 뱀의 몸 길이를 1 감소시킵니다.

**4 단계. Gate 요소의 구현**

<gate.cpp> line 1~29: Gate 생성

void make\_gate(int selmap, vector<snakebody>& gate, int\* cnt2){

    \*cnt2=0; //reference for time

    initialize\_gate(selmap); //initialize gate of map

    gate.clear(); //remove elements

    srand((unsigned) time(0));

    int in\_gate\_row=0, in\_gate\_col=0, out\_gate\_row=0, out\_gate\_col=0;

    in\_gate\_row=rand()%30; //0~29

    in\_gate\_col=rand()%30;

    out\_gate\_row=rand()%30;

    out\_gate\_row=rand()%30;

    while(TRUE){

      if(map[selmap][in\_gate\_row][in\_gate\_col] ==1) break; //possible to make in\_gate

      in\_gate\_row=rand()%30;

      in\_gate\_col=rand()%30;

    }

    map[selmap][in\_gate\_row][in\_gate\_col]=7; //set in\_gate

    while(TRUE){

      if(map[selmap][out\_gate\_row][out\_gate\_col] ==1) break; //possible to make out\_gate

      out\_gate\_row=rand()%30;

      out\_gate\_col=rand()%30;

    }

    map[selmap][out\_gate\_row][out\_gate\_col]=7; //set out\_gate

    gate.push\_back(snakebody(in\_gate\_row,in\_gate\_col)); //gate[0] is in\_gate

    gate.push\_back(snakebody(out\_gate\_row,out\_gate\_col)); //gate[1] is out\_gate

}

Make\_gate() 함수 실행시 뱀이 Gate를 완전히 통과한 후 make\_gate() 함수가 호출되었을 경우를 고려해 처음에 cnt2 값을 0으로 초기화해주고, initialize\_gate() 함수로 기존에 생성된 Gate들을 없애줍니다. 생성된 Gate는 gate 벡터의 요소로 관리되기 때문에 gate 벡터 내부의 기존의 Gate 요소들도 clear() 함수를 통해 전부 없애줍니다.

Gate의 위치도 랜덤으로 정해지기 때문에 srand()와 time() 함수를 활용해 난수 생성 시 시드값을 지정합니다. Gate는 in\_gate와 out\_gate의 한 쌍으로 구성되어 있으므로 in\_gate\_row, in\_gate\_col, out\_gate\_row, out\_gate\_col의 네 변수로 0~29 사이의 랜덤한 좌표값을 저장합니다. Gate는 map의 wall 부분에만 생성될 수 있으므로 map에서 새로 생성된 좌표값에 해당하는 데이터가 wall 해당하는지 확인 후 해당하지 않으면 다시 좌표값을 생성하고, gate 생성이 가능하면 해당 위치의 데이터를 7로 수정함으로써 in\_gate를 먼저 생성합니다. 같은 과정을 한 번 더 반복해 out\_gate도 생성하고, In\_gate와 out\_gate가 전부 생성되면 각 좌표값을 요소로 한 snakebody 객체를 생성해 gate 벡터에 삽입합니다. 이 때 in\_gate와 out\_gate는 임시 별칭에 해당합니다.

<main.cpp> line 51~90 in game(): Gate 출현 빈도 조정

//Snake moving -------------------------------------------------------------------------------------

    int snake\_direction =1;

    int cnt=0;

    int cnt2=0;

    bool gate\_crossing=false;

    int cross\_cnt=0;

    while(TRUE){

      score();

      stagescore = itemcnt\*200 - poisoncnt\*100 + gatecnt\*100;

    // directon decide

      keypad(stdscr, TRUE);

      nodelay(stdscr, TRUE); //to move snake without input

      int input\_key=getch();

      if(input\_key==KEY\_F(1)){

        break;

    }

      snake\_direction=input\_Direction(input\_key, &snake\_direction);

      snake\_move(snake, gate, &snake\_direction, &gate\_crossing, &cross\_cnt, &cnt2);

      color\_apply(selmap, win1);

      usleep(180000); //move snake per 0.18 sec

      cnt+=1;

      cnt2+=1;

      if (cnt%32==0){ //generate item per 8sec

        make\_item(selmap);

        cnt=0;

      }

      if(cnt2%52==0){ //gate appears after 13sec

        make\_gate(selmap, gate, &cnt2);

      }

      color\_apply(selmap, win1);

      refresh();

  }

Item의 생성 빈도를 cnt 변수로 조정했듯이 Main.cpp 파일의 game()함수 내부의 while문에서 cnt2값을 통해 Gate의 생성 빈도를 조절합니다. Usleep()함수를 통해 0.18초마다 뱀의 이동이 이루어지므로, 이를 고려해 cnt2가 52으로 나누어 떨어지면 make\_gate() 함수를 호출함으로써 13초마다 새로운 Gate가 생성되도록 합니다. 이때 cnt2는 make\_gate 함수 내에서 0으로 다시 초기화 됩니다.

<map.cpp> line 219~226 : 맵에서 Gate 초기화

void initialize\_gate(int selmap){

  for (int i=0; i<30; i++){

    for(int j=0; j<30; j++){

      if (map[selmap][i][j]==7)

        map[selmap][i][j]=1;

    }

  }

}

Initialize\_gate() 함수는 기존에 생성된 Gate를 맵에서 없애는 역할을 합니다. 이를 위해 이중 for문으로 맵 전체의 데이터를 확인하며 gate를 의미하는 7을 만나면 wall을 의미하는 1로 바꿔줍니다.

<snake.cpp> line 142~153 in snake\_move(): 뱀이 Gate를 만났을 때

  else if (map[selmap][new\_head\_row][new\_head\_col]==7){ //encounter gate

    if (new\_head\_row==gate[0].r&&new\_head\_col==gate[0].c){ //gate[0] is in\_gate

      out\_gate\_row=gate[1].r; out\_gate\_col=gate[1].c; //gate[1] is out\_gate

    }

    else{ //gate[1] is in\_gate

      out\_gate\_row=gate[0].r; out\_gate\_col=gate[0].c; //gate[0] is out\_gate

    }

    cross\_gate(out\_gate\_row, out\_gate\_col, &new\_head\_row, &new\_head\_col, snake\_direction, selmap);  //set head location through the gate

    gatecnt++;

    \*gate\_crossing=true;

    }

Snake.cpp 파일의 snake\_move()함수 내부 조건문을 통해 뱀이 Gate를 만났다면 맵에 생성되어있는 두 개의 Gate 중 어떤 것이 in\_gate/out\_gate인지 결정되도록 합니다. 마주친 Gate의 좌표와 gate 벡터 내부 요소의 값들을 비교해 현재 마주친 Gate를 in\_gate로, 나머지 하나의 Gate를 out\_gate로 정의합니다. In\_gate와 out\_gate가 정의됐으면, cross\_gate() 함수를 통해 뱀의 다음 진출 방향을 설정합니다. 이때 앞서 정의된 out\_gate의 좌표값들과 새로운 뱀 머리의 좌표값들이 함수의 인자로서 주어지고, 뱀이 Gate를 통과하는 동안 Gate가 새로 생성되지 않도록 통제하기 위한 변수 두 가지의 값이 정의됩니다. 이때 Gatecnt는 Gate를 통과한 뱀의 몸 길이를, gate\_crossing은 현재 뱀이 Gate를 통과중인지 여부를 저장합니다.

<snake.cpp> line 179~186 in snake\_move()

: 뱀이 Gate 완전히 통과했는지 여부 확인 및 Gate 생성

  if(\*gate\_crossing){ //if snake is crossing the gate

    if(\*cross\_cnt==snake.size()){

      \*cross\_cnt=0;

      make\_gate(selmap, gate, cnt2); //new gate generates when snake crossed the gate

      \*gate\_crossing=false;

    }

     \*cross\_cnt+=1;

}

뱀이 Gate를 완전히 통과했는지 확인하고, 완전히 통과했다면 Gate를 새로 생성해주기 위해 snake\_move() 함수 하단에서 조건문을 넣어주었습니다. 만약 gate\_crossing 변수 값이 true이면서

Cross\_cnt 값이 snake의 길이와 같지 않다면, 즉 뱀이 아직 완전히 Gate를 통과하지 못했다면 cross\_cnt값을 1 늘려줍니다. 이때 만약 cross\_cnt 값과 뱀의 길이가 같아지면 뱀이 Gate를 완전히 통과했음을 의미하므로 cross\_cnt와 gate\_crossing 변수를 초기값으로 초기화해주고, 새로운 gate가 생성되도록 make\_gate() 함수를 호출합니다.

<gate.cpp> line 31~98: Gate 통과시 조건에 맞게 진출 방향 지정

void cross\_gate(int out\_gate\_row, int out\_gate\_col, int\* new\_head\_row, int\* new\_head\_col, int\* snake\_direction, int selmap){

  if (out\_gate\_row==0){ //out gate at top

    \*snake\_direction=2;

    \*new\_head\_row=1; //downside

    \*new\_head\_col=out\_gate\_col;

  }

  else if (out\_gate\_row==29){ //out gate at bottom

    \*snake\_direction=0;

    \*new\_head\_row=28; //upside

    \*new\_head\_col=out\_gate\_col;

  }

  else if (out\_gate\_col==0){ //out gate at left

    \*snake\_direction=1;

    \*new\_head\_col=1; //to rightside

    \*new\_head\_row=out\_gate\_row;

  }

  else if (out\_gate\_col==29){ //out gate at right

    \*snake\_direction=3;

    \*new\_head\_col=28; //to leftside

    \*new\_head\_row=out\_gate\_row;

  }

  else{ //out gate not in boundary

    int status=0; //not modified

    while(status==0){ //until snake head position changes

      switch(\*snake\_direction){

        case 0: //up

          if(map[selmap][out\_gate\_row-1][out\_gate\_col]==1||map[selmap][out\_gate\_row-1][out\_gate\_col]==2){ //wall|immune wall in upside

            \*snake\_direction+=1; //turn right

            continue;

          }

          status=1;

          \*new\_head\_row=out\_gate\_row-1;

          \*new\_head\_col=out\_gate\_col;

          break;

        case 1: //right

          if(map[selmap][out\_gate\_row][out\_gate\_col+1]==1||map[selmap][out\_gate\_row][out\_gate\_col+1]==2){ //wall|immune wall in rightside

            \*snake\_direction+=1;

            continue;

          }

          status=1;

          \*new\_head\_row=out\_gate\_row;

          \*new\_head\_col=out\_gate\_col+1;

          break;

        case 2: //down

          if (map[selmap][out\_gate\_row+1][out\_gate\_col]==1||map[selmap][out\_gate\_row+1][out\_gate\_col]==2){

            \*snake\_direction+=1;

            continue;

          }

          status=1;

          \*new\_head\_row=out\_gate\_row+1;

          \*new\_head\_col=out\_gate\_col;

          break;

        case 3: //left

          if (map[selmap][out\_gate\_row][out\_gate\_col-1]==1||map[selmap][out\_gate\_row][out\_gate\_col-1]==2){

            \*snake\_direction=0;

            continue;

          }

          status=1;

          \*new\_head\_row=out\_gate\_row;

          \*new\_head\_col=out\_gate\_col-1;

          break;

      }

    }

  }

}

Cross\_gate() 함수에서는 게임 규칙에 맞게 out\_gate에서의 뱀의 진출 방향을 결정합니다. Out\_gate\_row와 out\_gate\_col의 값이 0이나 29에 해당하면 해당 out\_gate는 맵의 가장자리에 위치한 것이므로 맵의 안쪽 방향으로 진출 방향을 설정해야 합니다. 이를 구현하기 위해 조건문을 활용해 각 경우에 맞게 snake\_direction 값을 변경해주고, 새로운 뱀 머리의 좌표도 설정해줍니다. 이때 snake\_direction에서 0,1,2,3은 각각 위, 오른쪽, 아래, 왼쪽을 의미합니다.

만약 out\_gate가 맵의 가장자리에 위치하지 않은 경우, 뱀의 진출 방향은 진입 방향에 의해 결정되는데, 진입 방향과 일치하는 방향이 우선시됩니다. 또한, 뱀은 벽이 없는 빈 공간으로만 진출이 가능하기 때문에 시계방향 순서로 진출로 방향에 장애물이 있는지 확인하며 장애물이 없는 진출로의 방향으로 진출하게 됩니다. 이러한 기능을 구현하기 위해 switch-case문을 활용해 snake\_direction을 조건으로 넣어 진출로 방향의 장애물 여부를 확인하고, 장애물이 있다면 진출 방향을 시계방향으로 회전시킨 후 다시 조건문을 실행합니다. 만약 진출로에 장애물이 없다면, status 변수에 1을 저장해 new\_head의 좌표값이 변했음을 의미하도록 하고, out\_gate의 좌표를 기준으로 새로운 뱀 머리의 위치를 지정한 뒤 반복문을 빠져나와 함수 실행이 종료되도록 합니다.

**5 단계. 점수 요소의 구현**

|  |
| --- |
| **<map.cpp>**  void score(){  int goal[4][4] = {{4, 1, 1, 1}, {5, 1, 1, 2}, {6, 1, 2, 1}, {7, 3, 1, 2}};  -> 각 스테이지 별로 필요한 조건들의 2차원 배열입니다. 순서대로 뱀의 몸길이, 성장 아이템, 독 아이템, 게이트 사용 횟수입니다.    //============mission board========    wbkgd(win2, COLOR\_PAIR(11));    wattron(win2, COLOR\_PAIR(11));    mvwprintw(win2, 1, 13, "[MISSION]");    wborder(win2, '|', '|', '-', '-', '.', '.', '.', '.');    mvwprintw(win2, 3, 8, "B : %d", goal[selmap][0]);    mvwprintw(win2, 5, 8, "+ : %d", goal[selmap][1]);    mvwprintw(win2, 7, 8, "- : %d", goal[selmap][2]);    mvwprintw(win2, 9, 8, "G : %d", goal[selmap][3]);  -> 미션을 알려주는 윈도우입니다. 순서대로 뱀의 몸길이, 성장 아이템, 독 아이템, 게이트 사용 횟수를 출력합니다. 윈도우의 색상은 초록색입니다.  int current\_size = snake.size();  -> 현 스테이지의 뱀의 몸길이를 알려주는 변수입니다. snake라는 구조체의 크기를 size()함수를 통해 알아냅니다.    MaxSize = max(current\_size,size);  -> 한 스테이지가 끝났을 때, 뱀의 최대 몸길이를 알려주는 변수입니다.  //===========score board===========    wbkgd(win3, COLOR\_PAIR(11));    wattron(win3, COLOR\_PAIR(11));    mvwprintw(win3, 1, 13, "<<SCORE>>");    wborder(win3, '|', '|', '-', '-', '.', '.', '.', '.');    mvwprintw(win3, 3, 2, "B : %d / %d", current\_size, MaxSize);    mvwprintw(win3, 5, 2, "+ :   %d   ", itemcnt);    mvwprintw(win3, 7, 2, "- :   %d   ", poisoncnt);    mvwprintw(win3, 9, 2, "G :   %d   ", gatecnt);    mvwprintw(win3, 11, 2, "=======================");    mvwprintw(win3, 13, 2, "S :   %d   ",stagescore);  -> 점수판입니다. 순서대로 뱀의 몸길이, 성장 아이템, 독 아이템, 게이트 사용 횟수를 출력하며, 맨 아랫부분에서는 현 스테이지의 점수를 알려주는 변수를 출력합니다. 윈도우의 색상은 초록색입니다.  //==========checkbox=============    if(current\_size < goal[selmap][0]){      mvwprintw(win3, 3, 14, "(   )");    }    else{      mvwprintw(win3, 3, 14, "( V )");  wrefresh(win3);    }    if(itemcnt < goal[selmap][1]){      mvwprintw(win3, 5, 14, "(   )");      mvwprintw(win3, 6, 1, "[+200 points per item]");    }    else{      mvwprintw(win3, 5, 14, "( V )");      mvwprintw(win3, 6, 1, "[+200 points per item]");  wrefresh(win3);    }    if(poisoncnt < goal[selmap][2]){      mvwprintw(win3, 7, 14, "(   )");      mvwprintw(win3, 8, 1, "[-100 points per poison]");    }    else{      mvwprintw(win3, 7, 14, "( V )");      mvwprintw(win3, 8, 1, "[-100 points per poison]");  wrefresh(win3);    }    if(gatecnt< goal[selmap][3]){      mvwprintw(win3, 9, 14, "(   )" );      mvwprintw(win3, 10, 1, "[+100 points per gate]" );    }    else{      mvwprintw(win3, 9, 14, "( V )");      mvwprintw(win3, 10, 1, "[+100 points per gate]");  wrefresh(win3);    }  -> 점수판에 들어갈 체크박스 및 아이템별 점수 설명입니다. 현재 조건을 달성했을 경우를 조건문으로 만들어 해당하는 경우 괄호 안에 체크를 출력합니다. wrefresh(win3)함수로 인해 조건 충족 후 결과창이 뜰 때 마지막에 획득한 아이템이 체크 표시가 되지 않았던 버그를 해결했습니다.  //if player get all of requirements, that stage should be finished    if(current\_size>=goal[selmap][0])      if(itemcnt>=goal[selmap][1])        if(poisoncnt>=goal[selmap][2])          if(gatecnt>=goal[selmap][3]){            if(selmap==3){              result(2);              delwin(win1);              exit(0);            }            else{            result(1);            selmap++;            reset();            game();           }          }    wrefresh(win2);  wrefresh(win3);  -> 모든 조건(성장 아이템, 독 아이템, 게이트, 몸길이) 충족 시 다음 스테이지로 넘어갑니다.  [result(1)] 다음으로 넘어가면서 각 아이템을 세어주는 변수는 초기화를 해주고, 만약 모든 스테이지를 통과했을 경우 모든 스테이지를 통과했다는 결과창을 띄워주는 함수를 호출합니다. [result(2)]  }  //when the stage changes, all of the counts should be set to 0.  void reset(){    itemcnt = 0;    poisoncnt = 0;    gatecnt = 0;  -> 성장 아이템, 독 아이템, 게이트 통과 횟수를 초기화해주는 함수입니다.  **…중략…**  void caltime(){    endtime = time(NULL);    gettime = endtime - starttime;  }  -> 스테이지를 통과하는 데 까지 걸린 시간을 계산해주는 함수입니다. 종료 시간에서 시작 시간을 빼서 구하며, 단위는 초(s) 입니다. 이를 구현하기 위해 time.h 를 초반에 선언해줍니다.  void result(int resultnum){    wclear(win1);    if (resultnum == 1){      caltime();      mvwprintw(win1,15,25,"Stage%d Clear!!",selmap+1);      mvwprintw(win1,16,14,"Press Any Key to Move Next Stage...");      mvwprintw(win1,17,11,"You spent %d seconds to clear this stage",gettime);      mvwprintw(win1,18,18,"Your current score is : %d ",stagescore);      scoreset[selmap] = stagescore;      size = snake.size();  -> 1,2,3 스테이지를 통과했을 때 나오는 결과창입니다. resultnum이라는 정수를 함수 인자로 받으며, 스테이지가 종료되었을 때 출력되는 함수이므로 먼저 총 소요시간을 계산합니다. 이를 토대로 걸린 시간, 점수를 함께 알려줍니다. 추가적으로 통과한 스테이지의 점수를 scoreset 이라는 배열에 저장합니다. 아무 키나 누르면 다음 스테이지로 이동하며 한 스테이지가 끝났을 때 최종 몸길이를 저장합니다.    }    else if (resultnum == 2){      caltime();      scoreset[selmap] = stagescore;      mvwprintw(win1,10,25,"Stage All Clear!!");      mvwprintw(win1,12,16,"Press Any Key to Exit the Game...");      mvwprintw(win1,14,11,"You spent %d seconds to clear this stage",gettime);      mvwprintw(win1,15,16,"===========<Score Board>==========");      mvwprintw(win1,16,27,"[Stage 1 : %d]",scoreset[0]);      mvwprintw(win1,17,27,"[Stage 2 : %d]",scoreset[1]);      mvwprintw(win1,18,27,"[Stage 3 : %d]",scoreset[2]);      mvwprintw(win1,19,27,"[Stage 4 : %d]",scoreset[3]);  -> 4스테이지 까지 모두 통과했을 때 나오는 결과창입니다. 스테이지 1,2,3과 동일하게 resultnum이라는 정수를 함수 인자로 받으며, 스테이지가 종료되었을 때 출력되는 함수이므로 먼저 총 소요시간을 계산하고 4스테이지의 점수를 scoreset이라는 배열에 저장합니다. 이를 토대로 걸린 시간을 알려주며, scoreset 배열을 출력함으로써 1~4스테이지까지의 각각의 점수를 띄워줍니다.    }    else if (resultnum == 3){      mvwprintw(win1,15,25,"Game Over!!");      mvwprintw(win1,16,16,"Press Any Key to Exit the Game...");  -> 게임오버 됐을 때 나오는 결과창입니다. 통과한 것이 아니기 때문에 점수 저장, 시간 출력 등 별다른 동작을 하지 않습니다. 아무 키나 누르면 게임을 종료합니다.    }    wrefresh(win1);    nodelay(stdscr, false);    getch();    wclear(win1);  }  **<main.cpp>**       while(TRUE){  score();  -> 점수 및 미션 나타내는 함수입니다. while문으로 인해 점수와 미션에 변동이 생기면 항상 최신화됩니다.        stagescore = itemcnt\*200 - poisoncnt\*100 + gatecnt\*100;  -> 각 스테이지의 점수 계산 – 아이템 : +200점, 독 : -100점, 게이트 : +100점으로 설정했습니다. |

### 2.2.3 활용/개발된 기술

snakegame에 사용된 라이브러리는 **<ncurses>, <vector>, <cstdlib>, <time.h>, <cloclale>**입니다.

**<ncurses>**는 snakegame에 활용되는 3개의 window(win1,win2,win3)의 레이아웃과 백그라운드을 제작하고 cli환경에서 출력하는데 사용하였으며 keypad(), getch()을 통해 사용자로부터 방향키를 입력받도록 하였습니다. 또한 usleep()을 이용하여 snake가 0.25초 마다 이동하겠끔 구현하였습니다.

**<vector>**는 게임 내에서 계속 변하는 snake의 길이를 효율적으로 구현하기 위해 map내의 좌표값을 가지는 snakebody객체를 배열형태로 저장하고자 사용하였습니다. insert(), pop\_back(), push\_back()를 이용하여 vector에서의 효율적인 삽입과 삭제를 가능하게 하여 가변적인 snake의 길이를 구현하였습니다.

**<cstlib>**은 게임 내에서 난수를 생성하여 랜덤한 위치에서 snake, item, gate가 출현하도록 하였습니다. rand()를 이용하고 그 뒤에 %n를 붙여 snake에서는 막 스테이지의 지정된 생성 위치들이 저장되어 있는 배열에서 랜덤하게 선택하게 하였습니다. 그리고 item은 wall과 snake이외의 빈 곳에서 랜덤하게 생성되고 생성되는 아이템의 종류도 랜덤하게 결정됩니다. 마지막으로 gate또한 wall인 곳에서 랜덤하게 호출되게 하였습니다.

**<time.h>**는 게임 내 난수 발생과 실행 시간을 측정하는데 사용되었습니다. time()를 이용하여 시작 시간과 종료 시간을 측정하고 두 시간간의 차이를 구하여 실행시간을 구하였습니다.

**<clocale>**은 setlocale(LC\_ALL, “”)을 통해 game을 시작할 때 화면 맞춤을 하기 위해서 사용되었습니다.

추가적으로, 리눅스에서 터미널을 실행했을 때 화면이 너무 작아 원활한 플레이를 위해서는 화면을 수동으로 늘려줘야 했는데, xterm이라는 프로그램을 설치하고, resize -s 라는 명령어를 사용해서 크기를 자동으로 늘린 뒤 게임을 시작하도록 추가했습니다.

<Main.cpp> 중 main함수

 system("echo Please connect to the network to run this game...");

system("sleep 3");

// 충분한 시간 동안 일시정지를 해서 사용자가 충분히 안내 글을 읽을 수 있게 함.

system("echo If you already installed xterm, then just press ctrl+c to skip the installation...");

// 이미 xterm을 설치했다면 비밀번호 입력 창에서 ctrl+c를 누르면 바로 게임을 시작함을 안내함

  system("sleep 1");

  system("echo instead of entering password...");

  system("sleep 1");

  system("echo If you press ctrl+c, the game will start right away...");

  system("sleep 1");

  system("echo Or system will install xterm within 5 seconds...");

  system("sleep 5");

// 5초 일시정지 후 프로그램 설치

system("sudo apt install xterm");

// 프로그램 설치는 sudo 명령어를 사용하므로 비밀번호를 입력해야 하고, 프로그램을 인터넷에서 설치하기 때문에 네트워크에 반드시 연결되어야 함. 네트워크 연결해야 한다는 안내는 맨 처음에 출력함.

  system("resize -s 75 120");

  game(); // 설치가 끝났다면 화면을 조정하고, 게임 함수를 실행함.

### 2.2.4 현실적 제한 요소 및 그 해결 방안

**1-1)**

각각의 스테이지를 통과했을 때, 최종적으로 획득한 아이템의 체크박스가 작동하지 않는 버그가 있었습니다. 이를 고민하던 중 wrefresh() 함수를 사용함으로써 해결했습니다.

**1-2)**

스테이지 통과 조건을

 If((current\_size>=goal[selmap][0]) && (itemcnt>=goal[selmap][1]) &&

(poisoncnt>=goal[selmap][2]) && (gatecnt>=goal[selmap][3]))

와 같이 AND 연산자 (&&) 를 이용하여 한 줄의 조건문으로 작성했더니 스코어 및 미션을 나타내는 윈도우가 아예 뜨지 않았습니다. 이를 해결하기 위해

if(current\_size>=goal[selmap][0])

    if(itemcnt>=goal[selmap][1])

      if(poisoncnt>=goal[selmap][2])

        if(gatecnt>=goal[selmap][3]){

위와 같이 조건들을 분리해서 4중 if 문으로 만들었습니다. 이 부분을 작성한 사람으로서 왜 이렇게 조치했을 때만 정상적으로 작동하는지 그 이유를 찾지 못한 것이 이번 프로젝트의 한계점이라고 생각합니다**.**

**1-3)**

1,2,3,4 스테이지를 모두 통과했을 때,if(selmap>3) 을 단일 조건문으로 걸었더니 계속 segmentation fault (core dumped) 오류가 떴습니다.

첫 번째 조치는

if(current\_size>=goal[selmap][0])

    if(itemcnt>=goal[selmap][1])

      if(poisoncnt>=goal[selmap][2])

        if(gatecnt>=goal[selmap][3]){

여기에if(selmap>3) 이 조건문을 들여쓰기로 넣었습니다. 그래도 문제가 발생했습니다. 한참을 고민한 결과 맵을 구분하는 selmap이라는 변수가 3을 넘어가면 map.h에서 구현한 배열에서 인덱스의 범위가 맞지 않다는 것을 깨달았습니다.

int map[4][30][30]

즉, 여기에서 selmap에 해당하는 인덱스 범위는 0,1,2,3 이므로 selmap>3 일 경우 인덱스의 범위를 초과하게 됩니다.

또 한가지 깨달은 것은 이렇습니다. :

스테이지를 통과할 때 마다 selmap이 1씩 증가되는데, 0->1->2 순으로 증가되고, 2->3으로 증가될 때는 이미 3스테이지를 마치고 마지막인 4스테이지로 넘어가는 순간입니다. 그러므로 스테이지를 모두 통과했을 때를 나타내는 if문이 selmap>3일 필요는 없다는 것입니다. 즉, selmap ==3 이어도 조건문이 성립하는 데 전혀 문제가 없다는 것을 깨달았습니다.

따라서if(selmap == 3) 으로 변경했더니 제대로 작동했습니다.

**1-4)**

Snake.cpp의 snake\_move() 함수에서 뱀의 머리 부분이 poison item을 만났을 경우, snake 벡터에서 pop\_back()을 해줌으로써 요소를 제거하고, 뱀의 몸 길이가 1만큼 감소한 효과를 나타냈습니다.

구현 초반에는 사라진 요소 위치의 데이터값을 수정하지 않았었는데, 그랬더니 뱀이 꼬리 자르기를 한 것 마냥 마지막 요소가 맵에서 사라지지 않고 그대로 출력되었습니다. 하지만 이후 해당 위치의 데이터값을 0으로 수정해줌으로써 이러한 문제를 해결할 수 있었습니다.

  else if (map[selmap][new\_head\_row][new\_head\_col]==6){ //encounter poison item

      snake.pop\_back(); //body\_length-=1

      map[selmap][snake[snake.size()].r][snake[snake.size()].c] = 0; //error in here!

      poisoncnt++; //poison\_count +=1

      if (snake.size() < 3){ //if snake size is shorter than 3

        result(3);

        delwin(win1);

        exit(0);

      }

    }

**2-1)**

게임을 시작하기 위해 터미널을 열어 명령어를 입력하였을 때 터미널 창의 크기가 랜덤하게 정해져 게임을 설정해놓은 크기대로 실행을 못한다는 어려움이 있었습니다. 이를 고민하던 중 system()을 이용하여 해결했습니다. System() 함수를 찾아보니, 리눅스의 터미널에서 쓰이는 명령어를 C++ 코드 내에 삽입하여, C++로 작성된 프로그램 실행 시 터미널 명령어를 사용할 수 있게 하는 함수입니다.

**2-2)**

먼저 터미널 창의 크기 문제 부분에서는 화면 조정을 game()부분에서

  resize\_term(75,120);

을 통해 조정을 반영하고자 하였습니다. 그러나 조정된 값은 반영이 되지 않았습니다.

**2-3)**

화면 조정을 하는 명령을 main()부분에 넣기로 변경을 하고 맞는 명령어를 검색해보았습니다.

검색 결과 system()이라는 함수를 통해 터미널 창의 크기 자체를 조절하는 방법을 찾아냈습니다.

**2-4)**

main부분에서 system()을 통한 게임 실행을 위한 xterm의 설치를 진행한 후 사이즈 조절하는 부분을 추가하고 게임을 다 진행한 후에는 reset을 통해 게임 종료 화면과 터미널의 명령어가 겹치는 문제를 해결하였습니다.

이 부분에서 깨달은 점은 터미널창의 조절을 c++파일 내에 있는 system()을 통해 할 수 있다는 점과 ncurses게임 종료 화면과 터미널의 명령어를 겹치지 않게 하기 위해서는 main()부분에서 game()이후 system(reset)을 넣어야 한다는 점이었습니다.

-> 다시 확인해 보니 여전히 게임 종료 후 명령어 출력 부분과 겹치는 문제가 있었고, 해결하지 못했습니다.

system(”sudo apt install xterm”);

system(”resize -s 75 120”);

game();

system(”reset”);

**2-5)**

Xterm 과 system 함수를 이용해서 xterm이 설치되었을 때는, 프로그램 설치 없이 바로 게임을 실행하도록 if문을 넣으려고 했으나, system함수로 터미널 명령어를 실행한 뒤에 나오는 반환값의 타입을 알 수 없어 적용하지 못했습니다.

### 2.2.5 결과물 목록

**1) 게임구조**

|  |
| --- |
| snakegame1.1.7  ├── Makefile  ├── map.h  ├── game.h  ├── gate.h  ├── main.cpp  ├── map.cpp  ├── snake.cpp  └── gate.cpp |

**2) 파일 설명**

|  |  |
| --- | --- |
| **snakegame1.1.7** | |
| **파일** | **설명** |
| **Makefile** | make명령어를 통해 main 오브젝트 생성 |
| **map.h** | stage1,2,3,4의 맵 |
| **game.h** | map.cpp의 초기 설정 및 함수선언, snake 객체 구조체 선언 |
| **gate.h** | gate.cpp 헤더 파일 |
| **main.cpp** | 게임이 실행되는 메인 파일 |
| **map.cpp** | 게임 맵 및 화면 설정 및 출력 |
| **gate.cpp** | 게이트 위치 및 통과 관리 |
| **snake.cpp** | 뱀의 위치, 이동 및 길이, 아이템 관리 |

**3) 게임 진행 화면**

**(1) 게임 맵**

|  |  |
| --- | --- |
| **stage 1** | **stage 2** |
|  |  |
| **stage3** | **stage4** |
|  |  |

**(2) 안내 화면**

|  |  |
| --- | --- |
| **스테이지 클리어** | **모든 스테이지 클리어** |
|  |  |

# 3 자기평가

**구수연:**

스네이크 게임에서 item의 표현과 조작, gate 생성 등을 담당하여 구현하였고 초기 맵 세팅에 기여를 하였습니다. 프로젝트를 진행하면서 점점 구현해야 될 코드도 길어지고 복잡해지면서 나중에는 여러 함수와 변수들을 관리하고 기억하는 것이 어려웠습니다. 또한 알고리즘을 구현하고 모르는 명령어를 찾아내고 구현하는 것이 익숙하지 않아 어려움을 겪었습니다.

팀원과 협력하여 프로젝트를 진행한 것이 이번이 처음이었는데 주석의 중요성과 팀원의 중요성을 알게 되는 경험이었습니다. 최대한 의사소통하며 코딩을 하고 있음에도 불구하고 서로 이해가 되지 않는 부분은 반드시 존재하였는데 주석을 통해 그 어려움을 가장 효율적으로 해결할 수 있음을 알게 되었습니다. 또 서로만의 다른 코딩 방법을 합쳐야 하는 과정에서는 코딩식이 깔끔해야 그 진행이 빠름을 알 수 있었습니다. 알고리즘을 짜는 부분에서 팀원의 도움을 많이 받을 수 있었는데 아이디어가 떠오르지 않을 때 필요한 키워드를 던져주셔서 포기하지 않고 해결할 수 있었습니다.

프로젝트를 시작할 때 각 단계별로 구현한 것에 집중한 나머지 전체적인 프로그램의 구조를 짜고 사용할 변수와 구현할 함수를 미리 구체적으로 정하지 않았습니다. 개발 초기에는 문제가 없었지만 나중에는 전체적인 코드를 파악하기가 쉽지 않았습니다. 그래서 프로젝트 초기에 더 구체적으로 프로그램의 구조를 짜고 변수와 함수를 미리 정의하여 진행하였으면 좋았을 것이라 생각합니다.

**정호용:**

저는 조장과 1단계,3단계 일부,5단계 구현 및 버그 수정을 했습니다. 교수님께서 제시하신 강의 자료를 보고 게임의 틀을 구성한 뒤 정말 막막하다고 생각이 들었으나, 팀원들이 다같이 격려하며 하다 보니 완성도 높은 게임을 만들 수 있었다고 생각합니다. 함수와 클래스 부분이 학습이 덜 되어 처음에는 다른 팀원들이 작성한 코드 조차도 이해를 잘 못했지만, 기말고사 공부를 조금씩 하며 헤더를 왜 써야 하는지, 클래스에 어떻게 접근하고 사용하는지 알게 되었습니다. 그래도 큰 틀이 잡히고 어느 정도 C++에 대해 숙지하고 나니 점수판을 만들고, 아이템을 만들고, 버그를 해결하는 것이 좀 더 수월했습니다.

특히나 이번 프로젝트에서 도움이 많이 되었던 것은 주석 달기 였습니다. 저희는 나중에 보고서 작성할 때를 위해서 코드를 작성할 때 이 코드가 어떤 것을 의미하는지 주석을 달기로 했습니다. 게임 제작이 끝난 후 이는 빛을 발하였습니다. 주석을 작성하지 않았더라면 어떤 코드인지 기억을 못 했을 것입니다. 이를 통해 주석 달기가 정말 중요하다는 것을 알았습니다. 또한 구글 드라이브에 업로드 시 변경사항이 무엇인지를 같이 공유하니 저희 팀이 어떤 것을 구현했고 무엇을 놓치고 있는지 한 눈에 보여 다음 작업 시 어떤 것을 목표로 구현할 지 생각하는 게 편했습니다. 개선이 필요한 점은 팀원 모두가 적극적이고 협동적이어서 크게 느끼지는 못했습니다.

# 4 참고 문헌

|  |  |  |
| --- | --- | --- |
| 번호 | 출저 | 참고 사항 |
| 1 | https://psman2.tistory.com/entry/ncurses-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D | ncurses의 키 입력 시 참고 |
| 2 | https://www.linuxjournal.com/content/about-ncurses-colors-0 | ncurses 색깔 설정 시 참고 |
| 3 | https://docs.microsoft.com/ko-kr/cpp/standard-library/vector-class?view=msvc-170 | vector클래스 활용 시 참고 |
| 4 | [https://sroongzi.tistory.com/entry/c-%EC%8B%A4%ED%96%89%EC%8B%9C%EA%B0%84-%EC%B8%A1%EC%A0%95-%EB%B0%A9%EB%B2%95-time-clock](https://sroongzi.tistory.com/entry/c-ì¤íìê°-ì¸¡ì -ë°©ë²-time-clock) | 시간 측정 시 참고 |
| 5 | <https://thisisvegetable.tistory.com/19> | 키보드 이용한 플레이어 동작 구현 시 참고 |
| 6 | https://steemit.com/kr-dev/@codingman/c-20-system--1563143796300 | System 명령어 사용법 |
| 7 | https://www.w3big.com/ko/linux/linux-comm-resize.html#gsc.tab=0 | 리눅스 터미널 창 크기 조절 명령어 |

# 5 부록

## 5.1 사용자 매뉴얼

1) 실행: 터미널에서 snakegame1.1.7 파일로 이동후 ./main 명령어 실행합니다.

2) xterm을 설치했다면 패스워드 입력 창에서 ctrl+c를 눌러 바로 게임을 시작합니다. (자동으로 화면크기 조정 뒤 게임이 시작됩니다.)

아직 xterm을 설치하지 않았다면, 인터넷 연결 후 패스워드를 입력하여 xterm을 설치합니다.

설치 후 자동으로 게임을 시작합니다.

3) 뱀의 키보드 방향키로 움직입니다.

3) 초록색 아이템을 먹으면 + 1개 획득하고 점수는 200증가하며 뱀의 길이는 1증가합니다.

4) 빨간색 아이템을 먹으면 - 1개 획득하고 점수는 100감소하며 뱀의 길이는 1 감소합니다.

5) 노란색 게이트가 게임 시작 10초 후 생성되고 이후 10초마다 위치가 갱신됩니다. 노란색 게이트를 통과하면 G 1개 획득하고 점수는 100증가합니다.

6) 노란색 게이트는 한 쌍으로 한 쪽 게이트로 들어가면 다른 쪽 게이트로 나옵니다.

7) Mission 화면의 조건을 달성하면 다음 스테이지로 이동합니다. 스테이지는 총 4개가 있고 모두 클리어시 승리

8) 스테이지 클리어 이후 화면에서 획득한 점수와 시간이 표시됩니다.

9) Score 항목 설명 B: 뱀의 길이/최대길이, +: 초록색 아이템 , -: 빨간색 아이템 , G: 게이트 통과 S: 점수합계 입니다.

10) 뱀의 진행방향과 반대방향 입력 시, 뱀의 머리가 벽에 닿을 시, 미션 충족을 못 할 시 게임오버됩니다.

11) f1을 누르면 게임을 중간에 종료할 수 있습니다.

## 5.2 설치 방법

1)터미널에서 snakegame1.1.7 파일로 이동 후 make 명령어 입력

g++ -std=c++11 -o main main.cpp -lncursesw 실행 후 main파일 생성

1-1)혹은

혹은 g++ -std=c++11 -o main main.cpp -lncursesw

**입력해서 컴파일 해도 가능합니다.**

**2) ./main으로 실행**