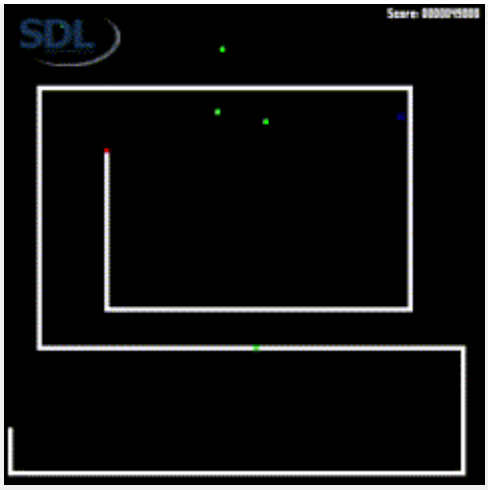
게임프로그래밍 기말대체과제

스네이크 게임 보고서

미디어기술콘텐츠학과 01921408 소병욱



플레이 영상 You Tube

How to Play?

```
↑ ↓ ← →  : 캐릭터 기본 이동
p        : 게임 일시정지
spacebar : 게임 클리어 치트키
```

```
Game Ready - 키보드 입력을 받으면 게임 실행
Game Over - 마우스 좌클릭으로 다시 시작 가능
Game Clear - 마우스 좌클릭으로 다시 시작 가능
```

```
게임 오버 조건
- Player Head 가 벽과 충돌한 경우
- Player Head 가 Player Body와 충돌한 경우

게임 클리어 조건
- 플레이어 캐릭터의 전체 크기가 ARR_SZIE^2 를 만족했을 경우 (화면을 모두 채운경우)
- 치트 아이템을 획득한 경우
- spacebar를 누른 경우
```

```
플레이어는 빨간색입니다
파란색 아이템을 먹으면 꼬리가 하나 추가됩니다. 먹기 전까지 새로 생성되지 않습니다.
```

```
초록색 아이템은 꼬리를 하나 제거합니다. N초에 N개씩 무작위로 생성됩니다.
주황색 아이템은 게임 클리어 치트 아이템입니다. N초에 N% 확률로 N초동안 하나 생성됩니다.
```

1. 게임 기본 설계

1.1 - 전체 격자 크기

- 100 x 100
- `Include.h` - `const int ARR_SIZE = WINDOW_SIZE / PLAYER_SIZE;`

  (윈도우 크기가 1000px x 1000px이고, 캐릭터의 크기가 10px x 10px 이니까 100 x 100)

1.2 - 뱀이 이동하는 속도

- 25FPS
- `Include.h` - `const int GAME_FRAME = 25;`
- `main.cpp`

```
Uint32 g_last_time_ms;
Uint32 g_frame_per_sec = GAME_FRAME;

if (cur_time_ms - g_last_time_ms < (1000 / g_frame_per_sec))
    continue;
```

  1초에 `g_frame_per_sec` 만큼 Update() 됩니다

- `GameLogic.cpp` - `Update()`

```
// ...

if (!g_game_paused) {
        // Player와 Item 업데이트
        for (auto player : g_Player) {
            if (player->getType() == Head) { // Head 타입의 Player만 입력 방향
설정
                player->setDir(g_input);
            }
            player->update();
        }
    }

// ...
```

일시정지 상태가 아닐때만 플레이어가 이동합니다 아래에서 더 설명하겠지만, 플레이어는 멤버변수인 mDir 방향으로 10px만큼 이동합니다

1.3 - 아이템을 하나 먹을 때 뱀이 늘어나는 길이

- 일반 아이템 : 1 칸 늘어남
- 함정 아이템 : 1 칸 줄어듦
- 치트 아이템 : 게임 클리어
- `GameLogic.cpp` - `void CreateNewTail(Player* player);`

```
void CreateNewTail(Player* player) {
    Player* newTail = new Player(player->getPrevX(), player->getPrevY(),
BODY, g_renderer);
    g_Player.push_back(newTail);
    cout << "Size : " << g_Player.size() << " / " << ARR_SIZE * ARR_SIZE <<
" | Score :  " << g_score << endl;
}
```

기본적으로 생성 자체는 Player의 이전 X값과 Y값을 불러와 해당 위치에 Player객체 Body타입을 생성시킨다

1.4 - 플레이어의 꼬리 움직임 구현

- `GameLogic.cpp` - `Update()`

```
// ...
        // 꼬리 Player 객체 움직임 처리
    for (auto it = ++g_Player.begin(); it != g_Player.end(); ++it) {
        Player* prevPlayer = *(prev(it));
        Player* currentPlayer = *it;
        currentPlayer->setX(prevPlayer->getPrevX());
        currentPlayer->setY(prevPlayer->getPrevY());
    }

// ...
```

Head를 제외한 Player 객체들은 이전 Player 객체의 위치로 이동 이전 Player 객체의 위치를 저장 X 재 Player 객체의 위치를 이전 Player 객체의 위치로 설정 Y 현재 Player 객체의 위치를 이전 Player 객체의 위치로 설정

플레이어의 이동은 mDir에 의해서만 결정되며, Body의 움직임은 자신의 바로 앞에 Player객체의 mDir을 상속받아서 자연스러운 이동로직을 구현했다.

1.5 - 플레이어의 꼬리가 생성될때 Head 와 Body 구분

- 플레이어의 Head와 Body를 구분하기 위해 `getType`이라는 인터페이스 함수를 사용하여 구현하였다.

```
int getType() const { return mType; }
```

- `getType`을 통해 얻은 데이터에 따라 다음과 같은 동작을 구분하였다

    1. 플레이어 객체 생성 시, **Head가 아닌 경우에만 점수를 추가하도록 하였음.** 이를 통해 최초 실행 시 100점으로 실행하는 작은 버그를 해결함
    2. 플레이어 객체 렌더링 시, Head라면 빨간색으로, 꼬리라면 흰색으로 렌더링함
    3. 게임 로직에서 플레이어 업데이트 시, Head 타입의 플레이어만 입력 방향을 설정하도록 함 이를 통해 Head만 사용자의 입력에 반응하고, 꼬리는 Head를 따라가도록 구현함
    4. 아이템과의 충돌 감지 시, Head 타입의 플레이어만 아이템과 충돌 여부를 확인함 이를 통해 Head가 아이템을 먹었을 때만 꼬리가 생성되고, 점수가 증가하도록 구현했음

- `obj_Player.cpp`

```
Player::Player(int x, int y, int defineType, SDL_Renderer* renderer) :
Obj(x, y, renderer) {
    mType = defineType;
    if (mType != Head) {
        g_score += mScore; // Head가 아닌 경우에만 점수 추가
    }
    // ...
}

void Player::render(SDL_Renderer* renderer) const {
    if(mType == Head) {
        SDL_SetRenderDrawColor(renderer, 255 , 0, 0, 255); // 머리라면 빨간색
    } else {
        SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255); // 꼬리라면 흰
색
    }

    SDL_RenderFillRect(renderer, &mRect);
}
```

    1. 플레이어 객체 생성 시, **Head가 아닌 경우에만 점수를 추가하도록 하였음.**
    2. 플레이어 객체 렌더링 시, Head라면 빨간색으로, 꼬리라면 흰색으로 렌더링함

- `GameLogic.cpp` - `void Update()`

```
void Update() {
    if (!g_game_paused) {
        // Player와 Item 업데이트
        for (auto player : g_Player) {
            if (player->getType() == Head) { // Head 타입의 Player만 입력 방향
설정
```

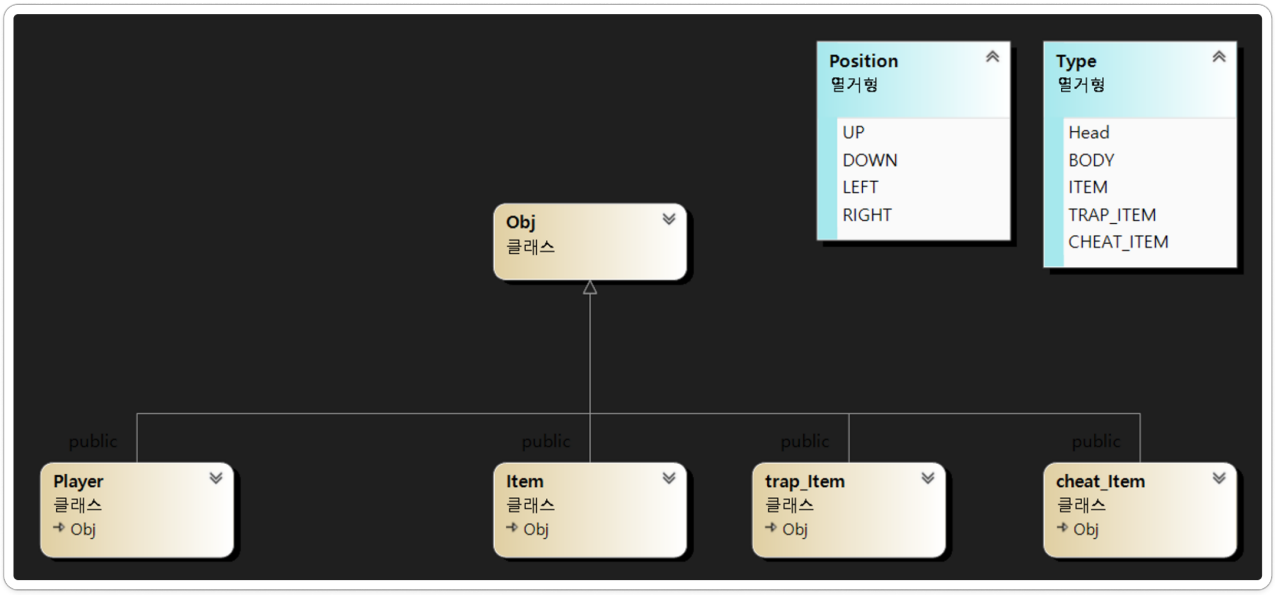
```
                player->setDir(g_input);
            }
            player->update();
        }
    }
    // ...
}

//! Player와 Item 충돌 감지
for (auto player : g_Player) {
    if (player->getType() == Head) {
        //~ 일반 Item 충돌 감지
        for (auto it = g_Item.begin(); it != g_Item.end();) {
            if (player->getX() == (*it)->getX() && player->getY() == (*it)-
>getY()) {
                delete* it;
                it = g_Item.erase(it);
                CreateNewTail(player);
                CreateNewItem();
                break;
            }
            else {
                ++it;
            }
        }
        // ...
    }
}
```

3. Head만 사용자의 입력에 반응하고, 꼬리는 Head를 따라가도록 구현함
4. 아이템과의 충돌 감지 시, Head 타입의 플레이어만 아이템과 충돌 여부를 확인

2. 기타 설계사항

2.1 - 클래스 다이어그램

- 클래스 및 열거형 구조



게임의 클래스 구조 및 열거형 타입은 다음과 같이 구성했다. 기본적으로 플레이어와 아이템은 같은 Obj 라는 순수 가상 클래스 부모를 상속받는다.

2.2 - 입력 로직 구현

- `GameLogic.cpp` - `void HandleEvents()`

    1. 뱀의 이동방향은 현재 방향의 반대방향으로 이동할 수 없다
        1. (반대 방향 전환 시 Head가 Body에 닿아 게임오버 되는것을 막기 위함)
    2. spacebar 누르면 바로 게임 클리어 화면으로 이동
    3. p 누르면 게임 일시정지

```
// ...

else if (event.type == SDL_KEYDOWN && g_game_started && !g_game_over &&
!g_game_clear) {
    switch (event.key.keysym.sym)
    {
    case SDLK_LEFT: if (g_input != RIGHT && !g_game_paused) { g_input =
LEFT; } break;
    case SDLK_RIGHT: if (g_input != LEFT && !g_game_paused) { g_input =
RIGHT; } break;
    case SDLK_UP: if (g_input != DOWN && !g_game_paused) { g_input = UP; }
break;
    case SDLK_DOWN: if (g_input != UP && !g_game_paused) { g_input = DOWN; }
break;
    case SDLK_SPACE: g_game_clear = true; g_game_paused = true;  break;
    case SDLK_p: g_game_paused = !g_game_paused; break;
    default: break;
    }

// ...
```

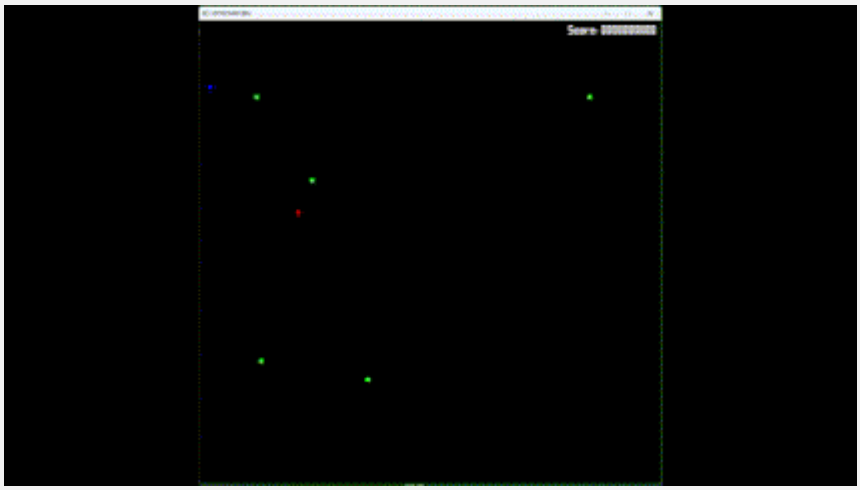
키가 입력되었고 게임이 플레이중인 상태이고 게임오버가 아니고 게임클리어가 아닐때

해당 키 입력 이벤트가 작동함

2.3 - 세부 아이템 구현

> - Item, trap_Item, cheat_Item 공통 생성조건 - 생성조건 : 객체들은 Player Head 와 Body 가 위치한 곳에 절대로 생성될 수 없다. - 생성조건 : 객체들은 다른 연결리스트에 존재하는 객체더라도, 겹쳐서 생성될 수 없다

2.3.a - 일반 아이템

> 일반 아이템 : 1 칸 늘어남 `Obj_Item.h` - `class Item : public Obj`
>
> 

- `GameLogic.cpp` - `void CreateItem()`

```
void CreateItem() {
    int itemX, itemY;
    do {
        itemX = rand() % WINDOW_SIZE / 10 * 10;
        itemY = rand() % WINDOW_SIZE / 10 * 10;
    } while (itemX == g_Player.front()->getX() && itemY == g_Player.front()-
>getY());

    Item* item = new Item(itemX, itemY, g_renderer);
    g_Item.push_back(item);
}
```

게임 시작 시 초기 아이템을 생성하는 함수입니다. 아이템의 위치는 랜덤으로 결정되며, 플레이어의 초기 위치와 겹치지 않도록 설계하였습니다. 생성된 아이템은 `g_Item` 리스트에 추가됩니다.

- `GameLogic.cpp` - `void CreateNewItem()`

```
void CreateNewItem() {
    int newItemX, newItemY;
    bool isValidPosition;
    do {
        newItemX = rand() % WINDOW_SIZE / 10 * 10;
```

```
            newItemY = rand() % WINDOW_SIZE / 10 * 10;
            isValidPosition = true;
            for (auto p : g_Player) {
                if (newItemX == p->getX() && newItemY == p->getY()) {
                    isValidPosition = false;
                    break;
                }
            }
        } while (!isValidPosition);

        Item* newItem = new Item(newItemX, newItemY, g_renderer);
        g_Item.push_back(newItem);
    }
```

플레이어가 아이템을 먹었을 때 새로운 아이템을 생성하는 함수입니다. 새로운 아이템의 위치는 랜덤으로 결정되며, 플레이어의 현재 위치와 겹치지 않도록 설계하였습니다.

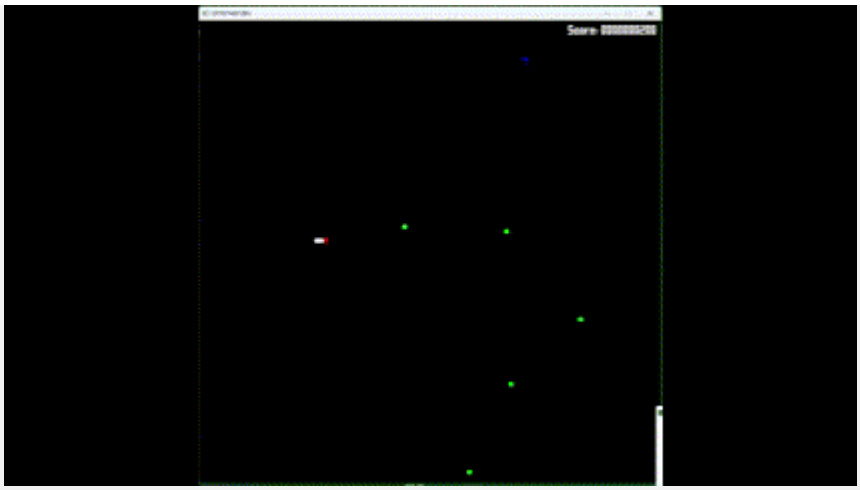
- `GameLogic.cpp` - `void Update()`

```
// ...
    //! Player와 Item 충돌 감지
    for (auto player : g_Player) {
        if (player->getType() == Head) {
            //~ 일반 Item 충돌 감지
            for (auto it = g_Item.begin(); it != g_Item.end();) {
                if (player->getX() == (*it)->getX() && player->getY() ==
(*it)->getY()) {
                    delete* it;
                    it = g_Item.erase(it);
                    CreateNewTail(player);
                    CreateNewItem();
                    break;
                }
                else {
                    ++it;
                }
            }
            // ...
        }
    }
// ...
```

게임 업데이트 함수에서 플레이어와 아이템의 충돌을 감지하는 부분입니다. 플레이어의 머리(Head)가 아이템의 위치와 겹칠 경우, 해당 아이템을 삭제하고 플레이어의 꼬리를 생성하는 `CreateNewTail` 함수를 호출합니다. 그리고 `CreateNewItem` 함수를 호출하여 새로운 아이템을 생성합니다.

2.3.b - 함정 아이템

> 함정 아이템 : 1 칸 줄어듦 `Obj_Item.h` - `class trapItem : public Obj`



- GameLogic.cpp - void CreateTrapItem()

```
void CreateTrapItem() {
    int itemX, itemY;
    bool isValidPosition;
    do {
        itemX = rand() % WINDOW_SIZE / 10 * 10;
        itemY = rand() % WINDOW_SIZE / 10 * 10;
        isValidPosition = true;

        // Player와 겹치는지 확인
        for (auto player : g_Player) {
            if (itemX == player->getX() && itemY == player->getY()) {
                isValidPosition = false;
                break;
            }
        }

        // 다른 Item과 겹치는지 확인
        for (auto item : g_Item) {
            if (itemX == item->getX() && itemY == item->getY()) {
                isValidPosition = false;
                break;
            }
        }
        for (auto item : g_TrapItem) {
            if (itemX == item->getX() && itemY == item->getY()) {
                isValidPosition = false;
                break;
            }
        }
        for (auto item : g_CheatItem) {
            if (itemX == item->getX() && itemY == item->getY()) {
                isValidPosition = false;
                break;
            }
        }
    } while (!isValidPosition);
```

```
    trap_Item* item = new trap_Item(itemX, itemY, g_renderer);
    g_TrapItem.push_back(item);
}
```

함정 아이템을 생성하는 함수입니다. 함정 아이템의 위치는 랜덤으로 결정되며, 플레이어와 다른 아이템들의 위치와 겹치지 않도록 설계하였습니다.

- GameLogic.cpp - void Update()

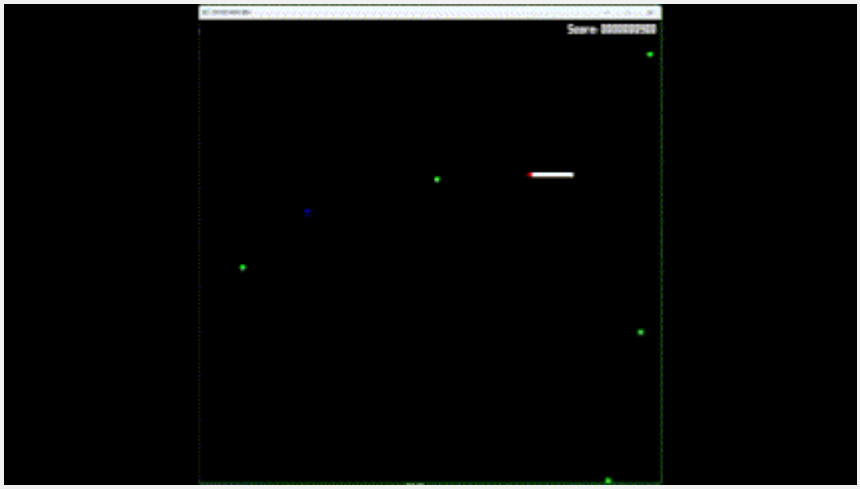
```
// ...

    //~ trap_Item 생성 // 10초마다 trap_Item 생성
    if (SDL_GetTicks() - g_last_trap_item_time >= 10000) {
        // 기존의 trap_Item 객체들 삭제
        for (auto it = g_TrapItem.begin(); it != g_TrapItem.end();) {
            delete* it;
            it = g_TrapItem.erase(it);
        }

        for (int i = 0; i < 5; ++i) {
            CreateTrapItem();
        }
        g_last_trap_item_time = SDL_GetTicks();
    }

// ...
```

게임 업데이트 함수에서 함정 아이템을 생성하는 부분입니다. 10초마다 함정 아이템을 생성하도록 설계하였습니다. 기존의 함정 아이템들은 삭제하고, 새로운 함정 아이템을 5개 생성합니다.

```
// ...
    //~ trap_Item 충돌 감지
    for (auto it = g_TrapItem.begin(); it != g_TrapItem.end();) {
        if (player->getX() == (*it)->getX() && player->getY() == (*it)-
>getY()) {
            delete* it;
            it = g_TrapItem.erase(it);
            if (g_Player.size() > 1) {
                Player* tail = g_Player.back();
                g_Player.pop_back();
                delete tail;
            }
            break;
        }
        else {
            ++it;
        }
    }
// ...
```

게임 업데이트 함수에서 플레이어와 함정 아이템의 충돌을 감지하는 부분입니다. 플레이어의 머리(Head)가 함정 아이템의 위치와 겹칠 경우, 해당 함정 아이템을 삭제하고 플레이어의 꼬리를 하나 제거합니다. 단, 플레이어의 크기가 1보다 클 경우에만 꼬리를 제거하도록 하였습니다. 즉, Head만 존재하는경우 trapItem을 먹어도 아무 효과가 없다는 의미입니다.

2.3.c - 치트 아이템

> 치트 아이템 : 게임 클리어 `Obj_Item.h` - `class cheatItem : public Obj`
>
> 

- `GameLogic.cpp` - `void CreateCheatItem()`

```
void CreateCheatItem() {
    int itemX, itemY;
    bool isValidPosition;
    do {
        itemX = rand() % WINDOW_SIZE / 10 * 10;
        itemY = rand() % WINDOW_SIZE / 10 * 10;
        isValidPosition = true;

        // Player와 겹치는지 확인
        for (auto player : g_Player) {
            if (itemX == player->getX() && itemY == player->getY()) {
                isValidPosition = false;
                break;
            }
        }

        // 다른 Item과 겹치는지 확인
        for (auto item : g_Item) {
            if (itemX == item->getX() && itemY == item->getY()) {
                isValidPosition = false;
                break;
            }
        }
        for (auto item : g_TrapItem) {
            if (itemX == item->getX() && itemY == item->getY()) {
                isValidPosition = false;
```

```
                break;
            }
        }
        for (auto item : g_CheatItem) {
            if (itemX == item->getX() && itemY == item->getY()) {
                isValidPosition = false;
                break;
            }
        }
    } while (!isValidPosition);

    cheat_Item* item = new cheat_Item(itemX, itemY, g_renderer);
    g_CheatItem.push_back(item);
}
```

치트 아이템을 생성하는 함수입니다. 치트 아이템의 위치는 랜덤으로 결정되며, 플레이어와 다른 아이템들의 위치와 겹치지 않도록 설계하였습니다.

- `GameLogic.cpp` - `void Update()`

```
// ...
    //~ cheat_Item 생성 // 30초마다 10% 확률로 cheat_Item 생성
    if (!g_is_cheat_item_active && SDL_GetTicks() - g_last_cheat_item_time
>= 30000) {
        if (rand() % 100 < 10) {
            CreateCheatItem();
            g_is_cheat_item_active = true;
            g_last_cheat_item_time = SDL_GetTicks();
        }
    }

    //~ cheat_Item 소멸 // 5초 후 cheat_Item 소멸
    if (g_is_cheat_item_active && SDL_GetTicks() - g_last_cheat_item_time >=
5000) {
        for (auto it = g_CheatItem.begin(); it != g_CheatItem.end();) {
            delete* it;
            it = g_CheatItem.erase(it);
        }
        g_is_cheat_item_active = false;
    }
// ...
```

게임 업데이트 함수에서 치트 아이템을 생성하고 소멸시키는 부분입니다. 30초마다 10%의 확률로 치트 아이템을 생성하도록 설계하였습니다. 치트 아이템이 생성되면 `g_is_cheat_item_active` 변수를 `true`로 설정하여 활성화된 상태로 표시합니다. 그리고 5초 후에는 치트 아이템을 소멸시키고 `g_is_cheat_item_active` 변수를 `false`로 설정하여 비활성화된 상태로 표시합니다.

```
// ...
    //~ cheat_Item 충돌 감지
```

```
    for (auto it = g_CheatItem.begin(); it != g_CheatItem.end();) {
        if (player->getX() == (*it)->getX() && player->getY() == (*it)-
>getY()) {
            delete* it;
            it = g_CheatItem.erase(it);
            g_score = 9999999999;
            g_game_clear = true;
            g_game_paused = true; // 게임 일시 정지 상태로 설정
            break;
        }
        else {
            ++it;
        }
    }
// ...
```

게임 업데이트 함수에서 플레이어와 치트 아이템의 충돌을 감지하는 부분입니다. 플레이어의 머리(Head)가 치트 아이템의 위치와 겹칠 경우, 해당 치트 아이템을 삭제하고 게임을 클리어 상태로 만듭니다. 이를 위해 `g_score`를 매우 높은 값으로 설정하고, `g_game_clear` 변수를 `true`로 설정합니다. 또한, `g_game_paused` 변수를 `true`로 설정하여 게임을 일시 정지 상태로 만듭니다. 이를 통해 치트 아이템을 먹으면 즉시 게임을 클리어할 수 있도록 구현하였습니다.

2.4 - 점수 구현

- `GameLogic.cpp` - `void RenderScore()`

  점수 구현은 Player의 생성자 및 소멸자가 호출 될 때, GameLogic 에 존재하는 g_score의 값을 수정하는 방향으로 구현했다.

  단순하게 SDL_TTF 를 이용했다

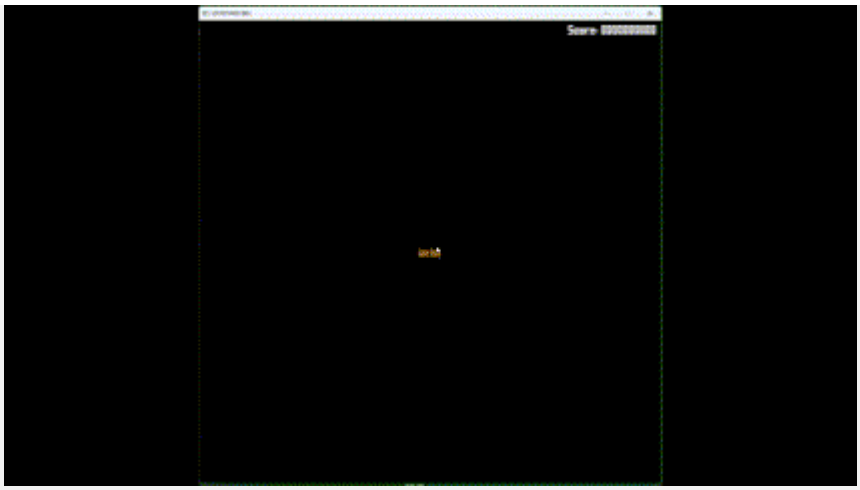
2.5 - 게임 실행 상태

- `Include.h` , `GameLogic.cpp`

```
//! 게임 실행 상태
bool g_game_started = false;    // 게임 시작 여부
bool g_game_over = false;       // 게임 오버 여부
bool g_game_clear = false;      // 게임 클리어 여부
bool g_game_paused = false;     // 게임 일시정지 여부
```

변수들을 설정하여 게임의 현재 상태를 나타내었고 관련 함수와 게임 로직에서 해당 변수들을 사용하여 게임의 흐름을 제어합니다. 각 변수의 값에 따라 게임 화면 렌더링, 업데이트 등의 동작이 결정됩니다.

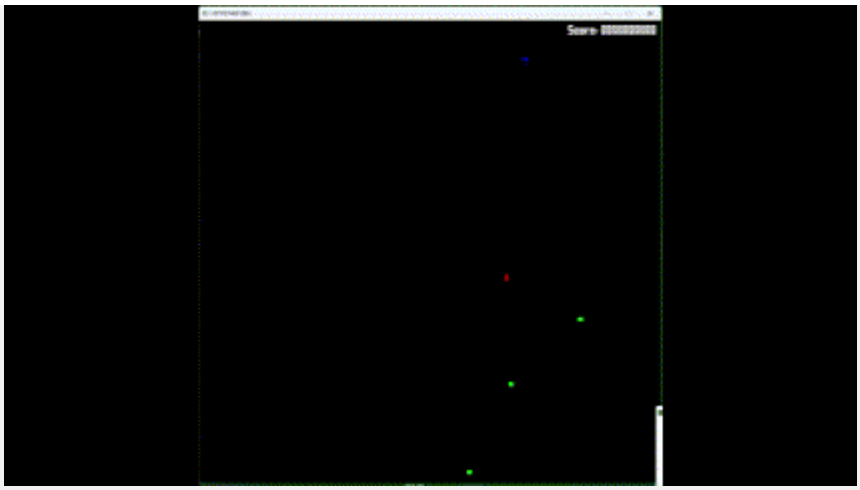
1. `Include.h` - `g_game_started` // 게임 시작 여부를 나타내는 변수



```
void HandleEvents() {
// ...
    else if (event.type == SDL_KEYDOWN && !g_game_started && !g_game_over &&
!g_game_clear) {
        g_game_started = true;
    }
// ...
}
```

`HandleEvents()` 함수에서 게임이 시작되지 않은 상태에서 키 입력이 발생하면 `true`로 설정됩니다.

2. `Include.h` - `g_game_over` // 게임 오버 여부를 나타내는 변수

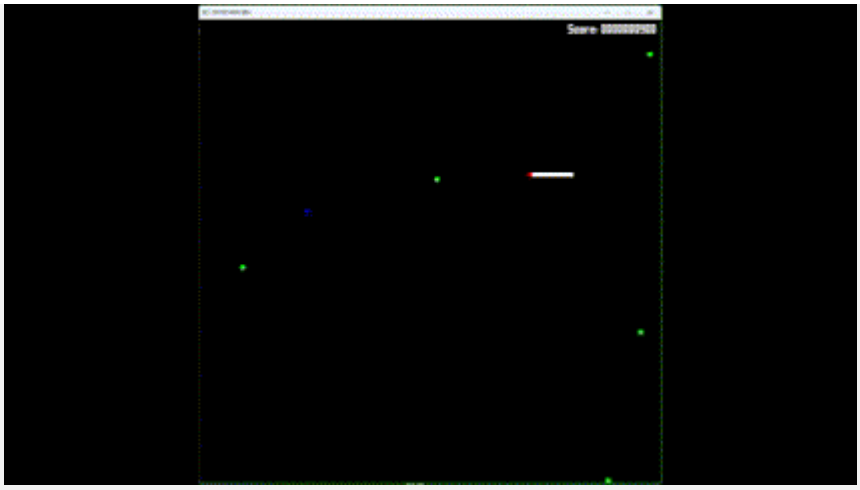


```
// ... update()
    void CheckGameOver() {
    // 조건 2: Player가 다른 Player 객체에 닿았을 경우 (게임 오버)
        for (auto it = ++g_Player.begin(); it != g_Player.end(); ++it) {
            Player* body = *it;
            if (head->getX() == body->getX() && head->getY() == body-
>getY()) {
                cout << "Game Over - Collided with body" << endl;
                g_game_over = true;
                g_game_paused = true;// 게임 오버 시 일시 정지 상태로 설정
```

```
                break;
            }
        }

    // 조건 3: Player가 벽에 닿았을 경우 (게임 오버)
        if (head->getX() < 0 || head->getX() >= WINDOW_SIZE || head->getY()
< 0 || head->getY() >= WINDOW_SIZE) {
            cout << "Game Over - Collided with wall" << endl;
            g_game_over = true;
            g_game_paused = true;// 게임 오버 시 일시 정지 상태로 설정
        }
    }
// ...
```

`CheckGameOver()` 함수에서 게임 오버 조건을 만족하면 `true`로 설정됩니다.

3. `Include.h` - `g_game_clear` // 게임 클리어 여부를 나타내는 변수



```
// ... update()

    void CheckGameOver() {
    // ...
            // 조건 1: g_Player 크기가 ARR_SIZE*ARR_SIZE와 같을 경우 (게임 클리
어)
        if (g_Player.size() == ARR_SIZE * ARR_SIZE) {
            cout << "Game Clear!" << endl;
            g_game_clear = true;
        }
    // ...
    }
// ...
```

`CheckGameOver()` 함수에서 게임 클리어 조건을 만족하면 `true`로 설정됩니다.

4. `Include.h` - `g_game_paused` // 게임 일시 정지 여부를 나타내는 변수

```
void HandleEvents() {
// ...
    else if (event.type == SDL_KEYDOWN && g_game_started && !g_game_over &&
!g_game_clear) {
        switch (event.key.keysym.sym) {
// ...
            case SDLK_p: g_game_paused = !g_game_paused; break;
// ...
        }
    }
// ...
}
```

`HandleEvents()` 함수에서 `p` 키가 입력되면 `g_game_paused` 값을 토글합니다.

```
void Update() {
    if (!g_game_paused) {
// Player와 Item 업데이트
        for (auto player : g_Player) {
            if (player->getType() == Head) {// Head 타입의 Player만 입력 방향
설정
                player->setDir(g_input);
            }
            player->update();
        }
    }
// ...
}
```

`Update()` 함수에서 `g_game_paused`가 `true`인 경우 플레이어와 아이템 업데이트를 수행하지 않습니다.