슬롯 머신 게임기

프로젝트 기술서

목차

「 학적 사항 」

- **학과** : 컴퓨터과학부
- **학번** : 2019920057
- **이름** : 최명재

「 주제 선정 」

현대의 디지털 슬롯 머신은 플레이어가 버튼을 누른 순간 RNG(Random Number Generator)로 숫자를 생성하여 모든 릴(reel)의 결과를 곧바로 정한다. 이러한 방식에서는 순전히 운에 따라 당첨 여부가 결정되므로, 옛날 기계식 슬롯 머신의 동작 방식에 착안하여, 좀 더 플레이어가 결과에 개입할 수 있는 슬롯 머신 게임기를 만들어 보고자 하였다.

「 기능 구현 」

https://prod-files-secure.s3.us-west-2.amazonaws.com/2f4a1174-4ae9-43c0-82eb-495e9ea7ee4a/6a8ccef7-6ad2-455d-a4a7-a300badae08d/IMG_0959.mov

영상1. 게임 플레이 영상

1. 고정 속도 업데이트 적용

- 초 단위로 애니메이션을 재생하고, 음악을 재생하기 위해서는 고정된 시간 주기로 상태를 갱신할 필요가 있었다.
- 각각의 객체가 동기화되어 작업을 진행할 필요가 있었기 때문에, 각 객체를 일관성 있게 관리하는 관리자가 필요했다.
- 그에 따라 FixedRateUpdater 클래스를 활용하여 고정 속도(60프레임)로 등록된 객체들을 호출할 수 있도록 기능을 구현했다.
- 속도는 동적으로 변경이 가능하며, Time 클래스를 사용하여 다른 클래스에서도 쉽게 시간 주기를 계산할 수 있도록 하였다.

2. 싱글톤 관리 및 의존성 주입

- GameManager, FixedRateUpdater, BuzzerController 등 전역적으로 사용되는 객체들을 관리하기 위해 싱글톤 패턴을 사용하였다.
- 또한 GameState 타입 객체들이 싱글톤 객체와 너무 큰 결합도를 갖지 않도록, GameState 타입 객체 생성 시 각각의 싱글톤 객체를 주입받도록 설계하였다.

3. FND 애니메이션 동시 재생

- FND 화면의 4개 디지트를 부분적으로 할당받아 특정 애니메이션을 재생할 수 있도록 설계하였다.
- 재생 디지트가 겹치지 않는 선에서 여러개의 애니메이션을 동시에 재생할 수 있도록 하여 보다 다양한 화면 연출이 가능하도록 하였다.
- 재생 중인 애니메이션의 재생 상태를 추적하여, 애니메이션 재생이 끝나면 playingAnimations 리스트에서 제거되도록 설계하였다.

4. 벡터를 활용한 음악 및 효과음 관리

- 음계 주파수를 상수로 정의하여 활용하는 데에 있어 가독성을 확보하였다.
- 음계 상수를 활용하여 각각의 게임 음악과 효과음을 벡터 컨테이너에 담아 관리하였다.
- BuzzerController에서 음악 벡터에 담긴 음계를 반복자로 순차적으로 읽어가며 출력하였다.
- 음악 벡터에는 각각의 음계가 재생되는 시간이 포함되어 있어, 적절한 박자로 음악을 출력할 수 있었다.

5. 스위치 입력 성능 최적화

- 스위치 입력 인터럽트 발생 시, 인터럽트 함수에서 큰 작업을 맡지 않도록 최적화를 진행하였다.
- 인터럽트 함수에서는 스위치 입력 변수를 토글하여 true 상태로 변경하고, SwitchController의 메인 루프에서 해당 토글을 인식하고 순차적으로 GameManager에 스위치 입력을 알리는 방식을 사용하였다.

6. 상태 패턴 활용 및 단계적 관리

- GameManager에서는 상태 패턴을 사용하여 ReadyState, OpeningState 등 다양한 상태를 관리할 수 있도록 하였다.
- GameState 인터페이스를 정의하고, 구체적인 상태 클래스가 해당 인터페이스를 구현하도록 설계하여 의존성을 역전시켰다.
- 상태 객체에서는 FIRST_PHASE, SECOND_PHASE 등을 정의하여 일련의 동작들이 단계적으로 실행되도록 설정하였다.

7. 당첨 결과 확인 알고리즘

릴 결과	당첨
7777	1st
1111	
1234	2nd
2143	
3337	3rd
1345	
else	Fail

그림1. 당첨 결과표

- 위 당첨 결과표에 따라 당첨 결과를 확인하고 출력하도록 알고리즘을 작성하였다.
- 2143과 같이 정렬되지 않은 연속된 숫자에 대해서도 당첨을 인식하도록 하였다.

「 진행 과정 」

1. 요구 분석

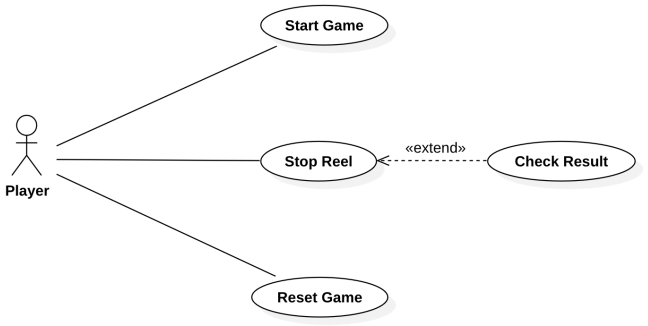

그림2. 유즈케이스 다이어그램

▼ Start Game (게임 시작하기)

[기본 흐름]

1. FND 디스플레이에 'PLAY' 글자가 표시 되었다가 사라지기를 반복한다. 경쾌한 음악이 출력되고 있다.
2. 플레이어가 스위치1을 누른다.
3. 게임 시작 효과음이 출력된다.
4. 'PLAY' 글자가 짧은 시간 빠르게 깜빡거리다가 사라진다.
5. 왼쪽 릴부터 차례대로 점등되며 0~9 사이의 숫자가 서로 다른 회전 속도로 출력된다.
6. 오른쪽으로 갈수록 회전 속도가 빨리지고, 빨라지는 속도에 맞춰 릴의 회전 효과음이 출력된다.
7. 모든 릴이 점등 되었으면, FND 전체 디스플레이가 깜빡이며 게임 플레이 시작 효과음이 출력된다.

▼ Stop Reel (릴 멈추기)

[제약 조건] 현재 회전하고 있는 릴이 1개 이상 존재한다.

[기본 흐름]

1. 플레이어가 스위치1을 누른다.

2. 클릭 효과음이 출력된다.
3. 현재 회전 중이던 릴 중 가장 왼쪽의 릴이 멈추기 시작한다.
4. 멈추는 릴의 회전 속도가 점점 느려지며, 숫자가 바뀔 때마다 릴의 회전 효과음이 출력된다.
5. 일정 시간 이후 릴이 완전히 멈추면, 회전 정지 효과음이 출력되고 릴의 숫자가 두 번 깜빡인다.

▼ **Check Result (결과 확인하기)**

[활성화 조건] 마지막 4번째 릴까지의 숫자가 확정되었다.

[기본 흐름]

1.정지된 모든 릴의 숫자 조합을 확인한다.

2.당첨일 경우 릴의 숫자가 왼쪽에서 오른쪽으로, 다시 오른쪽에서 왼쪽으로 점등되었다가 마지막에 전체적으로 두 번 깜빡이며 당첨 효과음이 출력된다.

3. 모든 숫자가 동일할 경우 '1st' 가, 숫자가 오름차순이나 내림차순으로 정렬되었을 경우 '2nd'가, 3개 이상 동일하거나 정렬되었을 경우 '3rd'를 FND 디스플레이에 출력한다.

4. 당첨이 아닐 경우 릴 숫자가 천천히 사라지고 'FAIL'이 천천히 FND 화면에 표시된다. 동시에 실패 효과음이 출력된다.

▼ **Reset Game (게임 초기화하기)**

[기본 흐름]

1. 플레이어가 게임 플레이 도중에 스위치 2를 클릭했다.
2. 재시작 효과음이 출력된다.
3. FND 화면이 정지되고, 두 번 깜빡인 후 왼쪽 디지트부터 오른쪽으로 지워지듯이 사라진다.
4. FND 디스플레이에 'PLAY' 글자가 표시되며 플레이 대기 상태로 돌아간다.

2. 시스템 설계

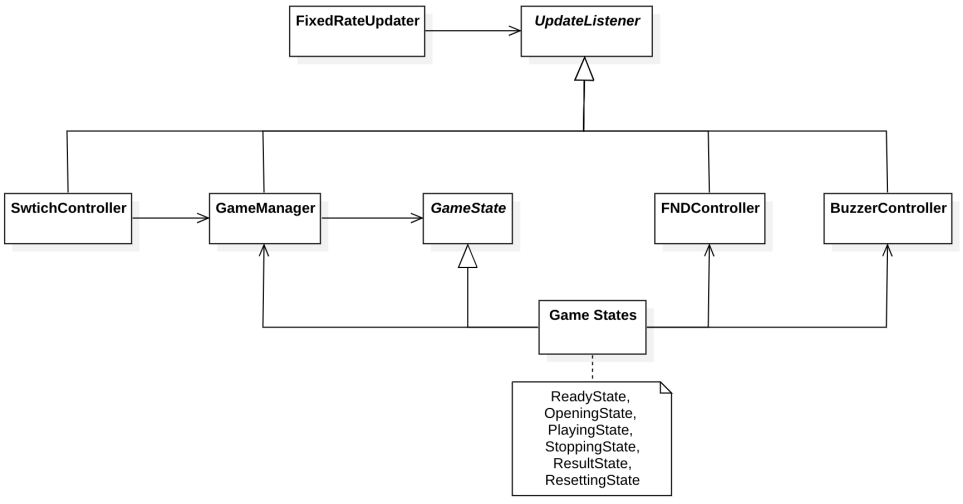

그림3. 클래스 다이어그램

- **FixedRateUpdater** : 타이머 인터럽트를 설정하여, 지정된 프레임레이트(기본값 60)로 매 프레임마다 등록된 UpdateListener들의 Update 함수를 호출한다.
- **UpdateListener** : 매 프레임마다 Update 함수를 호출 받아, 객체의 상태를 갱신할 수 있도록 하는 인터페이스 클래스이다.

- **SwitchController** : 사용자로부터 스위치1, 스위치2의 입력을 감지하고 이를 GameManager에 알려준다.
- **GameManager** : 외부 스위치 입력이나 내부 상태의 변화에 따라 다양한 GameState를 관리한다.
- **GameState** : GameManager로부터 받은 상태 시작, 갱신 등의 요청이나 스위치 입력 이벤트에 대하여 다형성을 보이기 위해 정의된 인터페이스이다.
- **Game States** : GameState 인터페이스를 구현한 구체적인 클래스들로, ReadyState, OpeningState 등 GameManager의 다양한 상태들을 의미한다. 게임 화면 출력을 위해 FNDController를 참조하고, 게임 음악 및 효과음을 출력하기 위해 BuzzerController를 참조한다.
- **FNDController** : FND 화면에 숫자, 글자 등을 출력하며, Flicker, Swipe 등 다양한 애니메이션을 갖고 있다.
- **BuzzerController** : 부저를 사용하여 게임 음악 및 효과음을 출력한다.

3. 상태 머신 설계

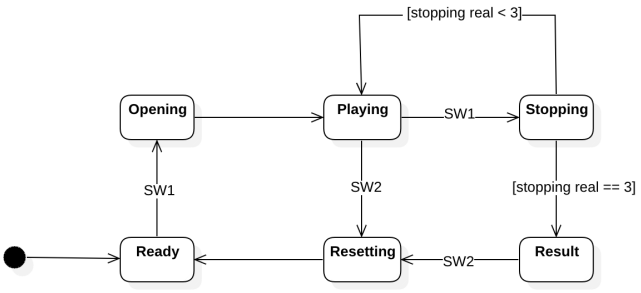

그림4. 스테이트 차트 다이어그램

- **Ready :** 게임을 처음 시작하면 돌입하는 상태로, 스위치1을 클릭하면 Opening 상태로 이동한다.
- **Opening** : 게임 플레이를 시작하기 전 여러 애니메이션 효과를 재생하는 상태이다. 오프닝 애니메이션이 끝나면 자동으로 Playing 상태로 넘어간다.
- **Playing** : 게임을 실제 플레이하는 상태이다. 스위치1을 클릭하여 Stopping 상태로 넘어간다.
- **Stopping** : 릴이 정지되는 애니메이션이 출력되는 상태이다. 아직 회전 중인 릴이 남아있을 경우 Playing 상태로, 모든 릴이 정지 되었을 경우 Result 상태로 넘어간다.
- **Result** : 릴들의 결과를 확인하고 최종 당첨 여부를 표시하는 상태이다. 스위치2를 눌러 Resetting 상태로 넘어간다.
- **Resetting** : 게임 초기화 작업이 진행되는 단계로, Playing 상태와 Result 상태에서 스위치2를 눌러 이동할 수 있다. 게임 초기화 작업이 끝나면 자동으로 Ready 상태로 다시 돌아간다.

「 사용 부품 」

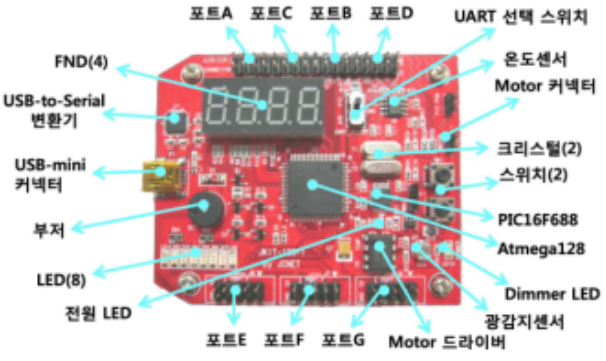

그림5. JKIT-128-1 구조

JKIT-128-1

내장된 모듈 중 부저, FND 인터페이스, 스위치를 사용하였다. 부저를 통해 게임 노래, 효과음을 출력하고, FND를 사용해 게임 화면을 출력한다. 스위치를 사용하여 게임의 진행 및 초기화를 실행한다.

「 소스 코드 」

1. 파일 구조

```
include/
├── game_enums.h
├── note_frequency.h
├── README
└── sounds.h

lib/
├── input/
│   ├── switch_controller.cpp
│   └── switch_controller.h
├── manager/
│   ├── game_states/
│   │   ├── game_states.h
│   │   ├── opening_state.cpp
│   │   ├── playing_state.cpp
│   │   ├── ready_state.cpp
│   │   ├── resetting_state.cpp
│   │   ├── result_state.cpp
│   │   └── stopping_state.cpp
│   ├── fixed_rate_updater.cpp
│   ├── fixed_rate_updater.h
│   ├── game_manager.cpp
│   └── game_manager.h
└── output/
    ├── buzzer_controller.cpp
    ├── buzzer_controller.h
    ├── fnd_animations.cpp
    ├── fnd_animations.h
    ├── fnd_controller.cpp
    ├── fnd_controller.h
    └── write_animation.cpp

src/
└── main.cpp
```

2. 주요 코드

main.cpp

```
#include <fixed_rate_updater.h>
#include <game_manager.h>
#include <fnd_controller.h>
#include <buzzer_controller.h>
#include <switch_controller.h>
#include <game_states/game_states.h>
```

```
#include <fnd_animations.h>

void AddGameStates();
void AddAnimations();
void AddUpdateListeners();
void DestroySingleTones();

int main(void)
{
    // Initial settings
    AddGameStates();
    AddAnimations();
    AddUpdateListeners();

    // Mute buzzer when testing
    //BuzzerController::GetInstance().MuteBuzzer(true);

    // Start game with ready state
    GameManager::GetInstance().SetGameState(State::READY);

    // Update modules every frame
    FixedRateUpdater& updater = FixedRateUpdater::GetInstance();
    while(true) updater.CallListeners();

    // Destory all single tone objects
    DestroySingleTones();

    return 0;
}

void AddGameStates()
{
    GameManager& gm = GameManager::GetInstance();
    FNDController& fnd = FNDController::GetInstance();
    BuzzerController& buzzer = BuzzerController::GetInstance();

    gm.AddGameState(State::READY, new ReadyState(gm, fnd, buzzer));
    gm.AddGameState(State::OPENING, new OpeningState(gm, fnd, buzzer));
    gm.AddGameState(State::PLAYING, new PlayingState(gm, fnd, buzzer));
    gm.AddGameState(State::RESETTING, new ResettingState(gm, fnd, buzzer));
    gm.AddGameState(State::STOPPING, new StoppingState(gm, fnd, buzzer));
    gm.AddGameState(State::RESULT, new ResultState(gm, fnd, buzzer));
}

void AddAnimations()
{
    FNDController& fnd = FNDController::GetInstance();

    fnd.AddAnimation(Animation::NONE, new NoAnimation());
    fnd.AddAnimation(Animation::SWIPE, new SwipeAnimation());
    fnd.AddAnimation(Animation::FLICKER, new FlickerAnimation());
    fnd.AddAnimation(Animation::ROUND_ROBIN, new RoundRobinAnimation());
    fnd.AddAnimation(Animation::WRITE_FAIL, new WriteFailAnimation());
    fnd.AddAnimation(Animation::WRITE_1ST, new Write1stAnimation());
    fnd.AddAnimation(Animation::WRITE_2ND, new Write2ndAnimation());
```

```
    fnd.AddAnimation(Animation::WRITE_3RD, new Write3rdAnimation());
}

void AddUpdateListeners()
{
    FixedRateUpdater& updater = FixedRateUpdater::GetInstance();

    updater.AddListener(&FNDController::GetInstance());
    updater.AddListener(&BuzzerController::GetInstance());
    updater.AddListener(&SwitchController::GetInstance());
    updater.AddListener(&GameManager::GetInstance());
}

void DestroySingleTones()
{
    FixedRateUpdater::DestroyInstance();
    SwitchController::DestroyInstance();
    GameManager::DestroyInstance();
    BuzzerController::DestroyInstance();
    FNDController::DestroyInstance();
}
```

switch_controller.cpp

```
#include <avr/interrupt.h>
#include "switch_controller.h"

const int SWITCH_ONE = 1;
const int SWITCH_TWO = 2;

SwitchController::SwitchController() :
        isSwitchOneClicked(false), isSwitchTwoClicked(false)
{
    // Set switch interrupt
    EICRB = (1 << ISC41) | (1 << ISC51);
    EIMSK = (1 << INT4) | (1 << INT5);
    PORTE = (1 << PE4) | (1 << PE5);
}

SwitchController& SwitchController::GetInstance()
{
    if (!instance) instance = new SwitchController();
    return *instance;
}

void SwitchController::DestroyInstance()
{
    delete instance;
    instance = nullptr;
}


void SwitchController::OnSwitchClick(Switch sw)
{
    switch (sw)
```

```
    {
    case Switch::ONE:
        isSwitchOneClicked = true;
        break;
    case Switch::TWO:
        isSwitchTwoClicked = true;
        break;
    default:
        break;
    }
}

void SwitchController::Update()
{
    if(isSwitchOneClicked)
    {
        GameManager::GetInstance().SwitchClick(Switch::ONE);
        isSwitchOneClicked = false;
    }
    else if(isSwitchTwoClicked)
    {
        GameManager::GetInstance().SwitchClick(Switch::TWO);
        isSwitchTwoClicked = false;
    }
}

SwitchController* SwitchController::instance = nullptr;



// Get switch 1 input
ISR(INT4_vect)
{
    static SwitchController& switchController = SwitchController::GetInstance();

    if (!(PINE & (1 << PE4)))
        switchController.OnSwitchClick(Switch::ONE);
}


// Get switch 2 input
ISR(INT5_vect)
{
    static SwitchController& switchController = SwitchController::GetInstance();

    if (!(PINE & (1 << PE5)))
        switchController.OnSwitchClick(Switch::TWO);
}
```

fixed_rate_updater.cpp

```
#include <avr/interrupt.h>
#include <util/delay.h>
#include "fixed_rate_updater.h"
```

```
/*** Define Fixed Rate Updater ***/

FixedRateUpdater::FixedRateUpdater() : frameRate(60), updateFlag(false)
{
    // Initialize timer1 for fixed update
    TCCR1A = 0;
    TCCR1B = (1 << WGM12) | (1 << CS12);
    OCR1A = 1040; // 60fps
    TIMSK |= (1 << OCIE1A);
    sei();
}

FixedRateUpdater& FixedRateUpdater::GetInstance()
{
    if(!instance) instance = new FixedRateUpdater();
    return *instance;
}

void FixedRateUpdater::DestroyInstance()
{
    delete instance;
    instance = nullptr;
}

void FixedRateUpdater::AddListener(UpdateListener* listener)
{
    listeners.push_back(listener);
}

void FixedRateUpdater::CallListeners()
{
    // Update only when update flag is set
    if (updateFlag)
    {
        for (auto& listener : listeners)
            listener->Update();
        updateFlag = false;
    }
}

void FixedRateUpdater::SetFrameRate(int rate)
{
    frameRate = rate;

    OCR1A = ( F_CPU / (256 * frameRate)) - 1;

    if (OCR1A < 1) {
        OCR1A = 1; // Minimum possible value
    }
}

int FixedRateUpdater::GetFrameRate()
{
    return frameRate;
}
```

```
void FixedRateUpdater::SetUpdateFlag()
{
    updateFlag = true;
}

FixedRateUpdater* FixedRateUpdater::instance = nullptr;

float Time::DeltaTime()
{
    static float frame = FixedRateUpdater::GetInstance().GetFrameRate();
    return 1 / frame;
}


/*** Timer Interrupt ***/

ISR(TIMER1_COMPA_vect)
{
    // Set fixed update flag
    static FixedRateUpdater& updater = FixedRateUpdater::GetInstance();
    updater.SetUpdateFlag();
}
```

game_manager.cpp

```
#include <game_manager.h>

GameManager& GameManager::GetInstance()
{
    if (!instance) instance = new GameManager();
    return *instance;
}

void GameManager::DestroyInstance()
{
    delete instance;
    instance = nullptr;
}

void GameManager::AddGameState(State state, GameStatePtr statePtr)
{
    gameStateMap[state] = statePtr;
}

void GameManager::SetGameState(State state)
{
    if(currentState) currentState->EndState();
    currentState = gameStateMap[state];
    currentState->StartState();
}

void GameManager::SwitchClick(Switch sw)
{
    if(currentState) sw == Switch::ONE ?
```

```
            currentState->SwitchOne() : currentState->SwitchTwo();
}

void GameManager::Update()
{
    if (currentState)
        currentState->UpdateState();
}

GameManager::~GameManager()
{
    for (auto state : gameStateMap)
        delete state.second;
}

GameManager* GameManager::instance = nullptr;
```

buzzer_controller.cpp

```
#include <avr/interrupt.h>
#include <util/delay.h>
#include "buzzer_controller.h"

BuzzerController::BuzzerController()
    : currentSound(nullptr), currentNote(nullptr), isMute(false)
{
    // Set buzzer pin
    DDRB |= (1 << PB4);
    sei();
}

BuzzerController& BuzzerController::GetInstance()
{
    if (!instance) instance = new BuzzerController();
    return *instance;
}

void BuzzerController::DestroyInstance()
{
    delete instance;
    instance = nullptr;
}

void BuzzerController::StartSound(const BuzzerSound* sound,
        float speed, bool loop)
{
    if(isMute) return;

    currentSound = sound;
    playSpeed = speed;
    loopSound = loop;
    playTime = 0;

    // Play first note
    if (currentSound)
```

```
    {
        currentNote = currentSound->begin();
        PlayNote(currentNote->first);
    }
}

void BuzzerController::Update()
{
    if (!currentSound) return;

    playTime += playSpeed * Time::DeltaTime();
    if (playTime >= currentNote->second)
    {
        currentNote++;

        if (currentNote == currentSound->end())
        // Playing complete
        {
            if (loopSound)
                StartSound(currentSound, playSpeed, true);
            else
            {
                currentSound = nullptr;
                PlayNote(NOTE_REST);
            }
        }
        else
        // Play next note
        {
            PlayNote(currentNote->first);
            playTime = 0;
        }
    }
}

void BuzzerController::PlayNote(double frequency)
{
    // Stop buzzer
    if (frequency == 0)
    {
        TCCR2 = 0;
        TIMSK &= ~(1 << OCIE2);
        PORTB &= ~(1 << PB4);
        return;
    }

    // Activate timer2 with 256 prescaler
    OCR2 = static_cast<uint8_t>((F_CPU / (256.0 * frequency)) - 1);
    TCCR2 = (1 << WGM21) | (1 << CS22);
    TIMSK |= (1 << OCIE2);
}

bool BuzzerController::IsSoundPlaying()
{
    return currentSound;
```

```
}

void BuzzerController::MuteBuzzer(bool value)
{
    isMute = value;
}

BuzzerController* BuzzerController::instance = nullptr;


// Play buzzer with timer2 interrupt
ISR(TIMER2_COMP_vect)
{
    PORTB ^= (1 << PB4);
}
```

fnd_controller.cpp

```
#include <avr/interrupt.h>
#include <util/delay.h>
#include "fnd_controller.h"


/*** Const Variables ***/
const vector<unsigned char> number = {0x3F, 0x06, 0x5B, 0x4F,
        0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
const vector<unsigned char> play = {0x73, 0x38, 0x77, 0x6E};
const vector<unsigned char> fail = {0x71, 0x77, 0x06, 0x38};
const vector<unsigned char> _1st = {0x00, 0x30, 0x6d, 0xf8};
const vector<unsigned char> _2nd = {0x00, 0x5b, 0x54, 0xde};
const vector<unsigned char> _3rd = {0x00, 0x4f, 0x50, 0xde};


FNDController::FNDController()
    : originalDisplay(4, 0), outputDisplay(4, 0)
{
    // Set FND pin
    DDRC = 0xFF;
    DDRG = 0x0F;

    // Set timer0 for FND
    TCCR0 = (1 << WGM01) | (1 << CS02);
    OCR0 = 124; // 2ms
    TIMSK |= (1 << OCIE0);
}

FNDController& FNDController::GetInstance()
{
    if (!instance) instance = new FNDController();
    return *instance;
}

void FNDController::DestroyInstance()
{
    delete instance;
```

```
    instance = nullptr;
}

unsigned char FNDController::GetOutputDigit(int digit)
{
    return outputDisplay[digit];
}

void FNDController::SetDisplay(Letter letter, bool consecutive)
{
    // Set original display
    switch (letter)
    {
    case Letter::NONE:
        std::fill(originalDisplay.begin(), originalDisplay.end(), 0);
        break;
    case Letter::PLAY:
        originalDisplay = play;
        break;
    case Letter::FAIL:
        originalDisplay = fail;
        break;
    case Letter::_1ST:
        originalDisplay = _1st;
        break;
    case Letter::_2ND:
        originalDisplay = _2nd;
        break;
    case Letter::_3RD:
        originalDisplay = _3rd;
        break;
    default:
        break;
    }

    // Empty output display
    if (!consecutive)
        std::fill(outputDisplay.begin(), outputDisplay.end(), 0);
}

void FNDController::SetDisplay(int num, bool consecutive, int start, int end)
{
    // Set original display
    for (int i = start; i <= end; i++) {
        int divisor = 1;
        for (int j = 0; j < (3 - i); j++) divisor *= 10;
        originalDisplay[i] = number[(num / divisor) % 10];
    }

    // Empty output display
    if (!consecutive)
        std::fill(outputDisplay.begin() + start, outputDisplay.begin() + end + 1, 0);
}

void FNDController::AddAnimation(Animation animation, FNDAnimation* fndAnimation)
```

```
{
    animationMap[animation] = fndAnimation;
}

void FNDController::StartAnimation(Animation animation, float speed, int start, int end)
{
    // Start animation
    FNDAnimation* startingAnimation = animationMap[animation];
    startingAnimation->StartAnimation(speed, start, end);
    playingAnimations.push_back(startingAnimation);
}

void FNDController::StopAnimations()
{
    // Clear every animations
    playingAnimations.clear();
}

bool FNDController::IsAnimationPlaying()
{
    bool playing = false;
    for (auto ani : playingAnimations)
        if(ani->IsAnimationPlaying())
            playing = true;
    return playing;
}

void FNDController::Update()
{
    for (auto iter = playingAnimations.begin(); iter != playingAnimations.end(); )
    {
        if((*iter)->IsAnimationPlaying())
        {
            (*iter)->PlayAnimation(originalDisplay, outputDisplay);
            iter++;
        }
        else
        {
            iter = playingAnimations.erase(iter);
        }
    }
}

FNDController::~FNDController()
{
    for (auto a : animationMap)
        delete a.second;
}

FNDController* FNDController::instance = nullptr;

void FNDAnimation::StartAnimation(float spd, int start, int end)
{
    speed = spd;
    playTime = 0;
```

```
    startDigit = start;
    endDigit = end;
    isAnimationPlaying = true;
}

bool FNDAnimation::IsAnimationPlaying()
{
    return isAnimationPlaying;
}



// Show FND display on every timer interrupt
unsigned char fnd_select[4] = {0x08, 0x04, 0x02, 0x01};
volatile int currentDigit = 0;

ISR(TIMER0_COMP_vect)
{
    static FNDController& fndController = FNDController::GetInstance();

    PORTC = fndController.GetOutputDigit(currentDigit);
    PORTG = fnd_select[currentDigit];

    currentDigit++;
    if (currentDigit >= 4) {
        currentDigit = 0;
    }
}
```

3. 전체 코드

< 프로젝트 깃허브 >

https://github.com/DdingJae418/micro-SlotMachine

「 결과 및 고찰 」

1. 구현 완료 기능

- 타이머 인터럽트를 활용하여 고정 속도로 업데이트를 진행하였다.
- 다양한 FND 애니메이션의 출력과 음악 및 효과음 출력 기능을 성공적으로 구현하였다.
- 스위치 클릭을 통해 게임을 진행하고, 초기화하는 기능을 성공적으로 구현하였다.
- 최종 결과를 확인하고, 릴의 조합에 따라 다른 결과를 표시하는 알고리즘을 구현하였다.
- 결과적으로 초기에 계획한 모든 기능의 구현에 성공하였다.

2. 아쉬운 점

- 멀티 스레드를 사용하여 프로그램을 관리하고 싶었으나, 마이크로 프로세서의 성능 한계 및 적절한 라이브러리가 제공되지 않아 적용하지 못했다.
- 스마트 포인터를 사용하여 메모리 관리를 수행하고 싶었으나, 마찬가지로 적절한 라이브러리가 제공되고 있지 않아 적용하지 못했다.
- 정식 STL 이 아닌 ArduinoSTL을 활용하여 STL 기능을 활용 하였는데, 모든 알고리즘이 제공되고 있지 않아 불편함이 있었다.