<HW1>

과제 : turtlesim CLI 제어 패키지 제작 (패키지명 turtlesim_cli)

1. 과제 조건

- 터미널로 모드 설정 기능 (조종, 배경색 설정, 거북이 모양 설정, pen 설정)
- 조종 모드 : WASD 입력받아 거북이 조종(turtle_teleop_key 참고)
- 배경색 설정 모드 : 배경색 입력받아 변경(RGB, 색상코드, 텍스트 등 자유)
- 거북이 모양 설정 : 모드 실행 시 선택 가능한 거북이 모양 리스트 출력, 모드 선택시 거북이 모양 변경
- pen 설정 모드 : 거북이 경로 색, 굵기 등 변경 가능하게

2. 개발과정 및 프로그램 구조

2 - 1. 사용 라이브러리

```
#include <termios.h>
#include <stdio.h>
#include <memory.h>
#include <unistd.h>
#include <chrono>
#include <thread>
#include <vector>
```

- termios.h 라이브러리

UNIX 및 UNIX-like 운영 체제에서 터미널 입출력 설정을 관리하기 위한 라이브러리이다. 이 라이브러리는 터미널의 동작을 제어하는 다양한 구조체와 함수를 제공한다. 주로 비동기 키 입력 처리, 터미널 모드 변경, 시리얼 포트 설정 등에 사용된다. 이 과제에서는 터미널 입력을 비동기적으로 처리하기 위해 터미널을 원시 모드로 설정하기 위해 사용되었다.

***비동기** : 특정 코드가 끝날때 까지 코드의 실행을 멈추지 않고 다음 코드를 먼저 실행하는 것

***원시 모드** : 터미널 입력의 처리 방식 중 하나로, 사용자 입력이 프로그램에 즉시 전달되는 모드

- stdio.h 라이브러리

이 라이브러리는 여러 입력과 출력을 처리하는 부분에서 사용되며 모드를 선택하거나 설정하고 싶은 내용을 입력할때 사용되었다.

- memory.h 라이브러리

메모리 복사 및 초기화에 사용되는 라이브러리로, 터미널 설정을 복사할 때 사용되었다.

- unistd.h 라이브러리

UNIX 및 UNIX-like 운영 체제에서 제공하는 라이브러로 여러 시스템 호출과 함수들을 정의하여 프로세스 제어, 파일 입출력, 사용자 환경 관리 등의 기능을 제공한다. 이 과제에서는 시스템 호출을 통해 데이터 입력 및 출력을 처리하는 역할을 수행한다.

- chrono, thread 라이브러리

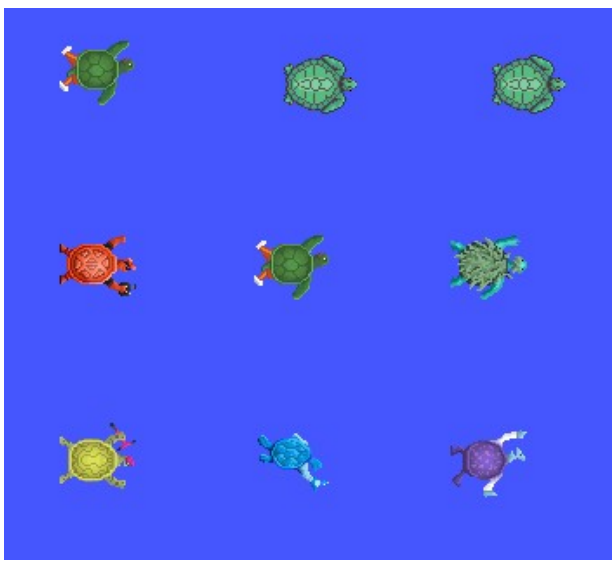
chrono 라이브러리는 시간 관련 기능을 제공하는 라이브러리로, 다양한 시간 단위와 시간 측정을 지원한다. thread 라이브러리는 멀티스레딩을 지원하는 C++ 표준 라이브러리이다. 이를 통해 스레드를 생성하고 관리할 수 있다. 이 두 라이브러리는 이 과제에서 시간을 지연하는 역할을 한다.

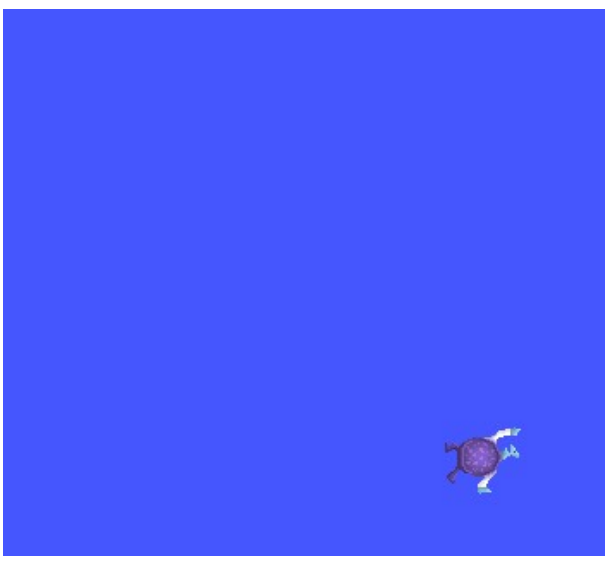
- vector 클래스

vector 는 C++ 표준 라이브러리에서 제공하는 컨테이너 클래스로, 동적 배열을 구현한다. 동적으로 크기가 조정 가능한 배열을 제공하여, 원소의 추가 및 삭제가 용이하다. 이 클래스는 STL(표준 템플릿 라이브러리)의 일부로, 효율적인 메모리 관리를 통해 다양한 데이터 구조를 처리할 수 있도록 돕는다. 이 과제에서는 변수를 관리하는데 사용이 되었다.

2 – 2 거북이 생성, 소멸

과제 조건이 "모드 실행 시 선택 가능한 거북이 모양 리스트 출력, 모드 선택시 거북이 모양 변경"이었다. 때문에 처음 개발할 때에는 처음에 있는 거북이를 삭제하고 원하는 거북이의 이름을 입력해 거북이를 재생성 시키는 방식으로 구현하려 했었다. 하지만 거북이의 이미지를 지정해서 원하는 이미지의 거북이를 생성하는데 어려움이 있었다. (거북이를 생성할때 생성되는 이미지가 랜덤이었기 때문)
그래서 처음에 다음과 같이 9 개의 거북이 이미지를 생성해두고 원하는 거북이를 제외한 거북이들을 kill 시키는 방향으로 사용자가 원하는 거북이를 선택할 수 있도록 구현했다.
(하지만 중복되는 거북이 이미지를 해결할 방법을 찾지 못했다)





거북이 생성, 소멸과 관련한 코드 구조는 다음과 같다.

- 거북이 생성

거북이 생성은 spawnAllTurtles 함수로 처리하였다. spawnAllTurtles 함수는 위 사진처럼 9 개의 거북이를 생성하는 역할을 하는데, 각 거북이는 앞서 정의된 2 차원 배열의 좌표대로 생성된다.
이 함수의 작동 순서는 먼저 9 개의 거북이 이름을 각각 벡터에 추가한다. 그 후에 spawn 서비스가 사용할 수 있을때까지 대기한다. 후에는 앞서 말한 2 차원 배열대로 각 거북이의 위치와 이름을 설정하고 이 요청을 전송한다.

- 거북이 선택

거북이 선택은 selectTurtle 함수로 처리하였다. 사용자가 입력한 번호에 따라 해당 거북이를 제외한 모든 거북이를 삭제하며, while 문을 활용해 선택한 번호의 거북이와 이름이 일치하지 않으면 삭제되는 식으로 kill 서비스를 통해 제거한다. 그 후 그 거북이를 활성화 시켜 조정 가능한 상태로 만든다.

2 – 3 배경색 설정

배경색 설정과 관련한 코드는 setBackground 함수로 처리했다. 이 함수는 r,g,b 세개의 매개변수를 받아오고, 이는 배경색의 rgb 를 의미한다. 이 매개변수를 활용해 각각의 rgb 값에 대한 파라미터를 설정한다.
그리고 파라미터 클라이언트가 이렇게 설정된 값을 노드에 전달한다.

2 - 4 거북이 경로 설정

거북이 경로(Pen)에 관련한 코드는 setPen 함수에서 처리된다 이 함수는 r,g,b,width 이렇게 네개의 매개변수를 사용한다. 이 매개변수들을 사용해서 색상과 굵기 요청을 설정하고, 이 정보를 노드에 비동기적 요청을 전송해 거북이 경로에 관한 설정을 반영한다.

2 -5 거북이 조작

https://github.com/sukha-cn/turtlesim-ros2/blob/master/tutorials/teleop_turtle_key.cpp
거북이 조작은 위 깃허브 소스코드를 참고해 제작했다.

거북이 조작과 관련한 코드 구조는 다음과 같다.

- 원시 모드 설정

입력한 키를 즉시 읽을 수 있도록 원시모드를 설정한다.

- 키 입력

키 입력을 받아온다.

- 키 입력에 따른 동작 결정

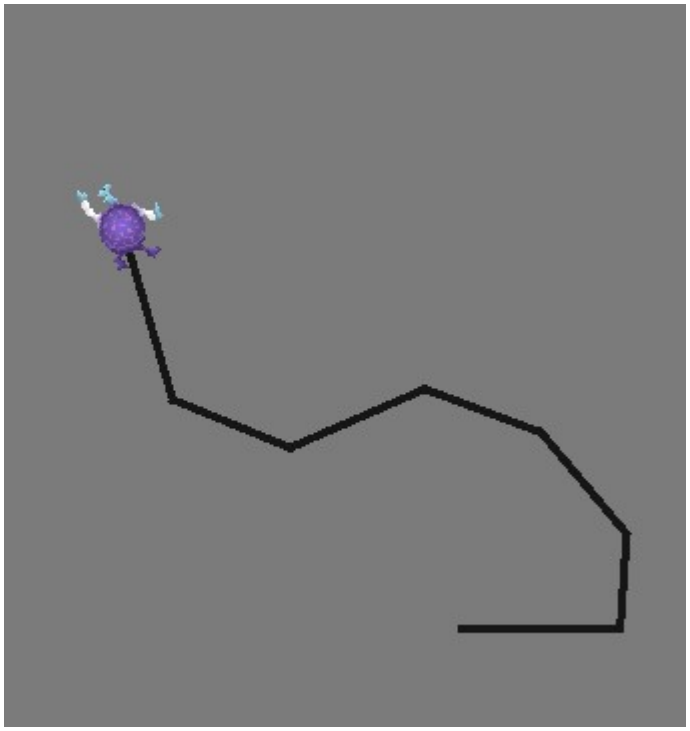
코드에 미리 키에 맞는 값을 정의해두고 그 값과 키입력 받은 값이 같은 경우에 각속도와 선형 속도를 설정한다.

- 메세지 보내기

설정된 각속도와 선형속도를 이용해 메세지를 퍼블리시해 거북이를 이동시킨다.

3. 실행

```
모드를 선택하세요(1. 시작, 2. 배경색 설정, 3. 거북이 모양 설정, 4. 펜 설정) :
3
9개의 거북이를 생성합니다...
조종할 거북이를 선택하세요 (1~9): 1
모드를 선택하세요(1. 시작, 2. 배경색 설정, 3. 거북이 모양 설정, 4. 펜 설정) :
3
9개의 거북이를 생성합니다...
조종할 거북이를 선택하세요 (1~9): 3
모드를 선택하세요(1. 시작, 2. 배경색 설정, 3. 거북이 모양 설정, 4. 펜 설정) :
4
펜 색상 RGB 값을 차례로 입력하세요 (0-255):
22 22 22
펜 굵기를 입력하세요 (1~10): 4
모드를 선택하세요(1. 시작, 2. 배경색 설정, 3. 거북이 모양 설정, 4. 펜 설정) :
2
RGB 값을 차례로 입력하세요 (0-255):
123 123 123
모드를 선택하세요(1. 시작, 2. 배경색 설정, 3. 거북이 모양 설정, 4. 펜 설정) :
1
```



실행 화면은 위와 같다.

4. 배운점

이 과제가 토픽, 서비스, 파라미터 세 가지 주요 ros 통신 방식이 사용된 과제였기 때문에 각 통신방법의 차이와 사용하는 방식을 알게 되었고, 깃허브에 올라와 있는 소스 파일들을 이해하는 과정에서 이런 소스파일들을 활용하는 법이나 공부하는 방법들을 익힐 수 있었다.

<HW2>

과제 : turtlesim 으로 그림 그리기 (package 이름 turtlesim_draw)

1. 과제 조건

- CLI 입출력을 통해 옵션 설정
- 삼각형, 원, 사각형 선택
- 크기 선택 (변 길이, 지름 등)
- pen 색상, 굵기 선택

2. 개발과정 및 프로그램 구조

2-1 사용 라이브러리

```
#include "turtlesim_draw/draw_node.hpp"
#include <chrono>
#include <thread>
```

사용한 라이브러리는 위와 같다.

chrono 라이브러리와 thread 라이브러리를 이용해서 시간을 지연시킬 수 있고 이는 반복문과 함께 원하는 도형을 그리기 위해 사용된다. 예를 들어 삼각형이면 그에 맞는 각도와 적절한 속도를 위에서 정의해주고 그것을 퍼블리시하면 거북이에게 설정된 속도와 회전이 전달된다. 이를 두 라이브러리를 통해 다음 명령어가 실행될때까지 지연시켜주면 거북이가 그 속도를 유지한채 움직이는 것처럼 보이게 만들 수 있다.

2-2 Teleop Turtle 클래스

```
class TeleopTurtle
{
public:
    TeleopTurtle(std::shared_ptr<rclcpp::Node> node_);
    void spawnTurtle(std::string name);
    void setPen(int r, int g, int b, int width);
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr twist_publication;

private:
    std::shared_ptr<rclcpp::Node> node_;
    std::string current_turtle_name_;
    rclcpp::Client<turtlesim::srv::Spawn>::SharedPtr spawn_client_;
    rclcpp::Client<turtlesim::srv::SetPen>::SharedPtr pen_client_;
};
```

- spawnTurtle 메서드

말그대로 거북이를 생성시키는 메서드이며, 미리 생성할 위치, 각도를 설정해두어, 이름만 입력하면 생성되도록 구성했다. 이 메서드 안에 Twist 메세지를 퍼블리시하는 퍼블리셔와 거북이의 펜 속성을 변경할수 있게 해주는 서비스 클라이언트가 같이 생성되어 있다. 이 메서드가 호출되면서 원하는 거북이의 이름을 입력하면, 그 거북이가 정해진 위치와 방향으로 생성되며, 그 거북이를 제어하기 위한 퍼블리셔와 거북이의 경로색을 변경할 수 있는 클라이언트도 함께 생성된다. 요청을 보내면 거북이가 생성된다. 여기서 pen_client 도 같이 생성하는 이유는 다음 setPen 메서드를 사용할 때에 용이성을 위함이다.

- setPen 메서드

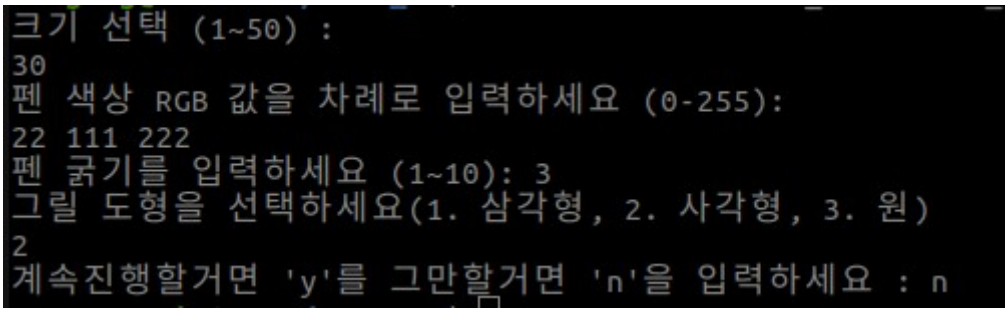
거북이의 경로(Pen)을 설정하는 메서드이며, 경로의 색(r,g,b 값)과 굵기(width)를 매개변수로 사용한다.

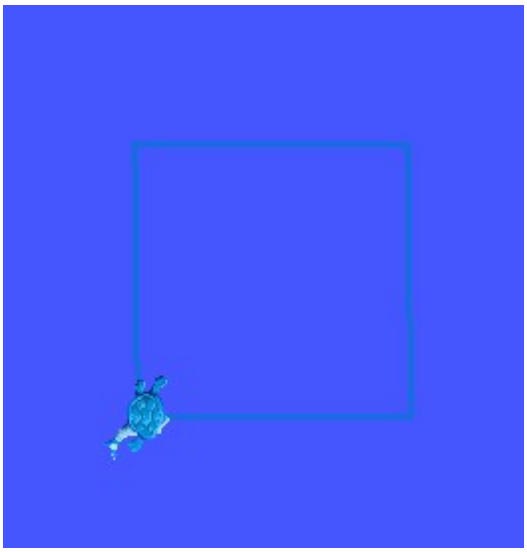
이 메서드에서 SetPen 서비스 클라이언트를 생성해 요청을 보낼 수 있도록 준비한후, 값을 설정해 서비스를 요청하면, 설정한 펜의 속성 거북이에 반영된다.

- TeleopTurtle 생성자

이 생성자에는 거북이를 생성하는 클라이언트를 미리 정의해 두었다.

2-3 실행





실행화면은 위와 같다.

경로의 크기는 적절한 값을 곱해 도형을 그릴때 너무 많은 경로를 이동하거나 너무 적은 경로를 이동하지 않게 해두었으며, 색상의 RGB 값, 굵기를 차례로 받고, 그릴 도형을 선택하면 그리도록 되어있으며,
while 문을 이용해 사용자의 선택으로 프로그램의 진행여부를 결정할 수 있다.

3. 배운점

여러 자료들을 보면 서비스 클라이언트를 미리 생성해두어 처리하는 경우가 많아서, 왜 저렇게 하는지 의문이었는데, 이번 과제를 하면서 그 궁금증이 해결 되었고, 서비스 통신 방법을 다루는 방법을 알게되었다.

<HW3>

과제 : demo_node_cpp 변경 (pakcage 이름 chatter_cli)

1. 과제 조건

- 노드 두개(talker, listener), 패키지 한개
- talker 노드에서 터미널로 퍼블리쉬 할 std_msgs/String 의 값 입력 받기
- 입력받은 내용 퍼블리쉬
- std_msgs/Int64 사용해서 몇번째 퍼블리쉬 인지 따로 퍼블리쉬
- listener 노드에서 talker 노드에서 퍼블리쉬 한 토픽 2 개 서브스크라이브 후 터미널로 출력. ex) hello, 1 -> Subscribed "hello" "1" / hi, 2 -> Subscribed "hi" "2"
- 토픽명 /chatter_cli, /chatter_count 로 고정

***demo_node_cpp(토픽)의 소스파일 주소는 아래와 같다.**

https://github.com/ros2/demos/tree/rolling/demo_nodes_cpp/src/topics

2. 개발과정 및 프로그램 구조

2 – 1 talker 노드

talker 노드의 작동 방식은 다음과 같다.

- 노드 생성 및 초기화

먼저 노드를 생성하고, 몇번 퍼블리시 했는지 셀 count 변수를 초기화 한다.

또한 setvbuf 함수를 사용해 즉시 출력되도록 설정한다.

- 퍼블리셔 생성

std_msgs::msg::String 과 std_msgs::msg::Int64 메세지를 topic, count 토픽에 각각 퍼블리시한다.

- 타이머 설정

1 초마다 publish_message 라는 람다 함수를 호출하는 타이머를 생성한다.

*람다함수 : 람다 함수는 C++에서 정의된 익명 함수(즉, 이름이 없는 함수)로, 주로 짧은 함수 표현식을 정의할 때 사용된다.

- 퍼블리시 함수 정의

먼저 사용자로부터 입력을 받고 string_msg_와 count_msg_를 생성한후에 입력 받은 문자열과 카운트를 퍼블리시 한다. 그 후에 퍼블리시된 내용을 로그로 출력한다.

- main 함수

여기에는 ros2 시스템을 초기화하고, publisher 클래스의 인스턴스를 생성하고 실행해 이벤트 루프를 시작하는 부분과 ros2 시스템을 정리하는 코드가 포함되어 있다.

2 – 2 listener 노드

listener 노드의 작동 방식은 다음과 같다.

- 노드 생성 및 초기화

노드를 생성하고, 몇번 메세지를 받았는지 기록할 received_count 변수를 초기화한다.

- 구독자 생성

std_msgs::msg::String 메시지를 수신하기 위한 구독자를 topic 토픽에 생성하고,

std_msgs::msg::Int64 메시지를 수신하기 위한 구독자를 count 토픽에 생성한다.

- 콜백 함수 정의

각 구독자에 대해 람다 형식의 콜백 함수를 정의한다. 첫 번째 콜백 함수는 topic 토픽에서 수신된

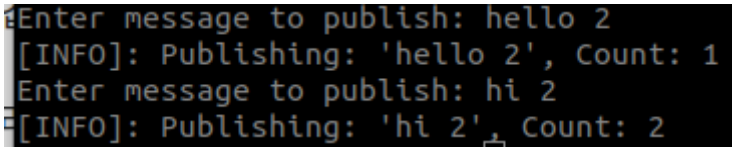
문자열 메시지와 received_count 값을 로그에 출력한다. 두 번째 콜백 함수는 count 토픽에서 수신된 Int64 메시지를 받아 received_count 변수에 저장한다.

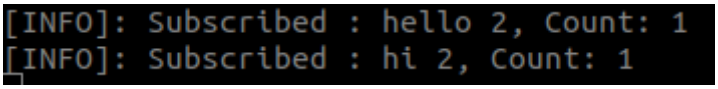
- main 함수

main 함수는 ROS2 시스템을 초기화하고, Subscriber 클래스의 인스턴스를 생성하여 실행해 이벤트 루프를 시작한다. 이와 함께 ROS2 시스템을 정리하는 코드도 포함되어 있다.

3. 실행

실행화면은 다음과 같다.





4. 배운점

토픽에 구조에 대해 자세히 알 수 있는 시간이었고, demo 코드 같은 것들이 있는지 이 과제를 하면서 처음 알게 되어서, 앞으로 공부할때 관련 자료를 찾아보면서 공부할 수 있을 것 같다.