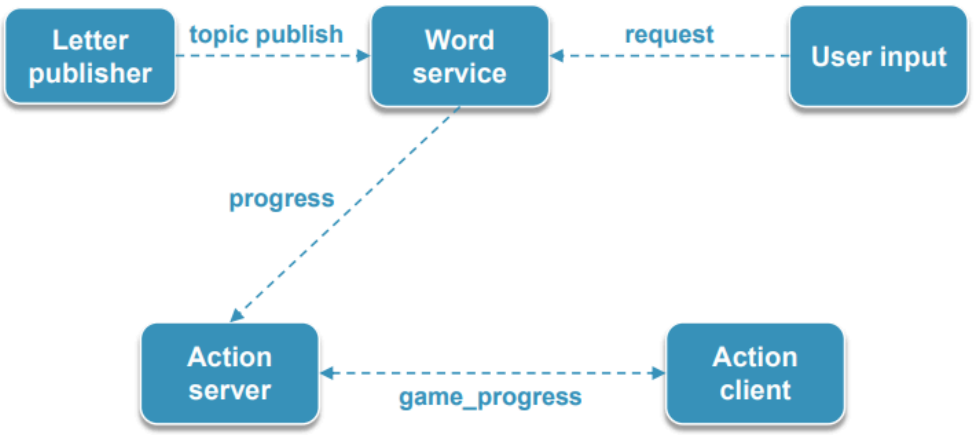
5.1 스터디 송주훈

주제: ROS2 관련 시험 대비 복습

Hang man 관련 분석 및 정리

행맨 구조도



- Letter Publisher: a부터 z까지 알파벳을 순서대로 publish
- Word Service: 임의의 단어를 선택하고 행맨 게임을 진행함. 진행 상황을 publish
- Action Client: Goal 설정, action server와 상호작용을 통해 게임의 진행 상황 업데이트
- Action Server: 사용자의 진행 상황을 관리. 진행 상태 추적. 게임 결과를 클라이언트에게 전달
- User Input: Enter key를 입력받으면 send_request 메소드를 호출하여 현재 요청을 서비스로 보냄

인터페이스 정리

- hangman_interfaces/srv/CheckLetter.srv

```
# Empty request
---
string updated_word_state
bool is_correct
string message
```

update_word_state: 현재 상태 ex)py__on

is_correct: user_input으로 들어온 글자가 선택된 단어 내에 존재하는지 확인

message: 맞으면 correct 틀리면 wrong을 띄움

- hangman_interfaces/msg/Progress.msg

```
string current_state
int32 attempts_left
bool game_over
bool won
```

current_state : 현재 상태 ex) p y _ _ o n
attempts_left : 목숨(남은 시도 횟수)
game_over : 목숨이 다 소진 되었는지
won : 게임에서 이겼는지(목숨 소진 이전에 정답을 맞추었는지)

- hangman_interfaces/action/GameProgress.action

```
# Goal
# Empty since the client doesn't need to send any data
---
# Result
bool game_over
bool won
---
# Feedback
bool game_over
```

game_over : 목숨이 다 소진 되었는 지
won : 게임에서 이겼는지(목숨 소진 이전에 정답을 맞추었는지)

Letter_publisher.py

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class LetterPublisher(Node):

    def __init__(self):
        super().__init__('letter_publisher')
        self.publisher_ = self.create_publisher(String, 'letter_topic', 10)
        self.timer = self.create_timer(1.0, self.publish_letter)
        self.current_letter = ord('a')

    def publish_letter(self):
        msg = String()
        msg.data = chr(self.current_letter)
        self.publisher_.publish(msg)
        self.get_logger().info(f'Publishing: {msg.data}')
        self.current_letter += 1
        if self.current_letter > ord('z'):
            self.current_letter = ord('a')

def main(args=None):
    rclpy.init(args=args)
    letter_publisher = LetterPublisher()
    rclpy.spin(letter_publisher)
    letter_publisher.destroy_node()
    rclpy.shutdown(

if __name__ == '__main__':
    main()
```

publisher_letter를 통해 하나의 알파벳씩 순서대로 publish한다.

word_service.py

```python
import rclpy
from rclpy.node import Node
from hangman_interfaces.srv import CheckLetter
from hangman_interfaces.msg import Progress
from std_msgs.msg import String
import random


class WordService(Node):

    def __init__(self):
        super().__init__("word_service")
        self.service = self.create_service(
            CheckLetter, "check_letter", self.check_letter_callback
        )
        self.subscription = self.create_subscription(
            String, "letter_topic", self.letter_callback, 10
        )
        self.progress_publisher = self.create_publisher(Progress, "progress", 10)

        data = Progress()
        data.current_state = ""
        data.attempts_left = 20
        data.game_over = False
        data.won = False
        self.progress_publisher.publish(data)

        self.current_letter = ""
        self.word_list = ["python", "hangman", "robot", "ros", "interface"]
        self.word = random.choice(self.word_list)
        self.word_state = ["_"] * len(self.word)
        self.get_logger().info(f"The word has {len(self.word)} letters.")
        self.attempts_left = 20  # Max attempts
```

```python
def letter_callback(self, msg):
    self.current_letter = msg.data

def check_letter_callback(self, request, response):
    letter = self.current_letter
    if letter in self.word:
        for idx, char in enumerate(self.word):
            if char == letter:
                self.word_state[idx] = letter
        response.is_correct = True
        response.message = "Correct!"
    else:
        self.attempts_left -= 1
        response.is_correct = False
        response.message = "WRONG"
    response.updated_word_state = "".join(self.word_state)
    self.get_logger().info(f"Received letter: {letter}")
    self.get_logger().info(f"Current word state: {response.updated_word_state}")

    # Publish progress
    progress_msg = Progress()
    progress_msg.current_state = response.updated_word_state
    progress_msg.attempts_left = self.attempts_left
    progress_msg.game_over = "_" not in self.word_state or self.attempts_left <= 0
    progress_msg.won = "_" not in self.word_state
    self.progress_publisher.publish(progress_msg)
    return response
```

```python
def main(args=None):
    rclpy.init(args=args)
    letter_publisher = LetterPublisher()
    rclpy.spin(letter_publisher)
    letter_publisher.destroy_node()
    rclpy.shutdown(

if __name__ == '__main__':
    main()
```

letter_callback: letter_topic을 통해 수신된 메세지를 처리하고 저장

check_letter_callback: 수신한 문자가 현재 단어에 포함되어있는지 확인하고 상태 업데이트. 업데이트된 상태는 Progress.msg를 통해 퍼블리시

user_input.py

```python
# hangman_game/user_input.py
import rclpy
from rclpy.node import Node
from hangman_interfaces.srv import CheckLetter
import threading

class UserInput(Node):

    def __init__(self):
        super().__init__("user_input")
        self.cli = self.create_client(CheckLetter, "check_letter")
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info("Service not available, waiting...")
        self.req = CheckLetter.Request()
        self.get_logger().info("Press Enter to check the current letter.")
        threading.Thread(target=self.input_thread, daemon=True).start()

    def input_thread(self):
        while True:
            input("Press Enter to input the current letter.")
            self.send_request()

    def send_request(self):
        future = self.cli.call_async(self.req)

def main(args=None):
    rclpy.init(args=args)
    user_input = UserInput()
    rclpy.spin(user_input)
    user_input.destroy_node()
    rclpy.shutdown()

if __name__ == "__main__":
    main()
```

input_thread: Enter 키 입력을 기다리며 입력이 들어오면 send_request 함수를 호출함

send_request: CheckLetter 서비스에 요청을 보내 현재 선택된 문자가 단어에 포함되어 있는지 확인

progress_action_client.py

```python
import rclpy
from rclpy.node import Node
from hangman_interfaces.action import GameProgress
from rclpy.action import ActionClient


class ProgressActionClient(Node):

    def __init__(self):
        super().__init__("progress_action_client")
        self._action_client = ActionClient(self, GameProgress, "game_progress")
        self.result_received = False
        self.send_goal()

    def send_goal(self):
        self.get_logger().info("Action Client: Waiting for action server...")
        self._action_client.wait_for_server()
        goal_msg = GameProgress.Goal()
        self.get_logger().info("Action Client: Sending goal request...")
        self._send_goal_future = self._action_client.send_goal_async(
            goal_msg, feedback_callback=self.feedback_callback
        )
        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def feedback_callback(self, feedback_msg):
        feedback = feedback_msg.feedback
        if feedback.game_over:
            self.get_logger().info("Action Client: Game over detected in feedback")
```

```python
    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info("Action Client: Goal rejected")
            self.result_received = True
            return

        self.get_logger().info("Action Client: Goal accepted")
        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result().result
        if result.won:
            self.get_logger().info("Action Client: Congratulations! You won!")
        else:
            self.get_logger().info("Action Client: Game Over. You lost.")
        self.result_received = True


def main(args=None):
    rclpy.init(args=args)
    action_client = ProgressActionClient()
    while rclpy.ok():
        rclpy.spin_once(action_client)
        if action_client.result_received:
            break
    action_client.destroy_node()
    rclpy.shutdown()


if __name__ == "__main__":
    main()
```

send_goal: 액션 서버에 목표를 전송하고 피드백콜백과 완료 콜백을 설정해 서버 응답 처리

feedback_callback: 서버에서 수신한 피드백 메세지를 처리

goal_response_callback: 서버가 목표를 수락했는지 확인. 수락했으면 최종 결과를 비동기적으로 요청하며 결과 콜백을 설정

get_result_callback: 최종 결과를 확인 후 승리/패배 따라 출력

progress_action_server.py

```python
import rclpy
from rclpy.node import Node
from hangman_interfaces.action import GameProgress
from hangman_interfaces.msg import Progress
from rclpy.action import ActionServer
from rclpy.executors import MultiThreadedExecutor
import time
import threading


class ProgressActionServer(Node):

    def __init__(self):
        super().__init__("progress_action_server")
        self._action_server = ActionServer(
            self, GameProgress, "game_progress", self.execute_callback
        )
        self.current_progress = Progress()
        self.progress_received_event = threading.Event()

        # Subscribe to the 'progress' topic to get game updates
        self.subscription = self.create_subscription(
            Progress, "progress", self.progress_callback, 10
        )
        self.subscription

        self.get_logger().info("Action Server Initialized")
        self.get_logger().info(f"GAME OVER: {self.current_progress.game_over}")
        self.get_logger().info(f"WON: {self.current_progress.won}")
```

```python
    def progress_callback(self, msg):
        self.current_progress = msg
        self.get_logger().info(
            f"Progress updated: {self.current_progress.current_state}"
        )

    def execute_callback(self, goal_handle):
        self.get_logger().info("Action Server: Received goal request")
        feedback_msg = GameProgress.Feedback()
        update_rate = 1.0  # seconds

        while not self.current_progress.game_over:
            # Publish feedback
            feedback_msg.game_over = self.current_progress.game_over
            goal_handle.publish_feedback(feedback_msg)
            self.get_logger().info(
                f"Current State: {self.current_progress.current_state}"
            )
            self.get_logger().info(
                f"Attempts Left: {self.current_progress.attempts_left}"
            )

            time.sleep(update_rate)

        # Game is over
        result = GameProgress.Result()
        result.game_over = self.current_progress.game_over
        result.won = self.current_progress.won

        if self.current_progress.won:
            self.get_logger().info("Action Server: Congratulations! You won!")
        else:
            self.get_logger().info("Action Server: Game Over. You lost.")

        goal_handle.succeed()
        self.get_logger().info("Action Server: Goal succeeded")
        return result
```

```python
def main(args=None):
    rclpy.init(args=args)
    action_server = ProgressActionServer()

    executor = MultiThreadedExecutor()
    executor.add_node(action_server)
    try:
        executor.spin()
    except KeyboardInterrupt:
        pass
    finally:
        action_server.destroy_node()
        rclpy.shutdown()


if __name__ == "__main__":
    main()
```

*여기서는 Multi Thread Executor 사용

progress_callback: progess 토픽으로부터 수신한 메세지를 처리.

execute_callback: 클라이언트의 목표 요청 수신 / 게임 진행 상황 피드백 / 게임 종료시 결과 반환. 주기적으로 클라이언트에게 게임 상태를 전달