Laporan Tugas
Kelompok Kecerdasan Artifisial
Dikumpul tanggal : 30 Juni 2024

Nama Tim : Acht (Kelompok 8)

Anggota :

- Andrew Agustio Winata – 221110989
- Elroy Saputra Duha – 221110066
- Frenz Luiz – 221111135
- Leonardo Ade Hendrawan – 221110179
- Louis – 221110738

Deskripsi Tugas :

Project kami mencakup pembuatan Chatbot Translation di aplikasi Telegram yang mampu menerima input dari pengguna dalam Bahasa Indonesia. Setelah menerima input dalam bentuk teks atau gambar, Chatbot akan memberikan hasil terjemahan dan cara pengucapan kepada user dengan bahasa yang telah dipilih pengguna sebelumnya.

Selain fokus pada pengembangan chatbot Translation, proyek kami juga mencakup pembuatan bot lainnya. Ini mencakup implementasi Perceptual Interface Hand Gesture Drawing yang mampu merespon aksi berupa gerakan tangan yang diberikan oleh pengguna, serta pembuatan gamebot Pacman yang dibangun menggunakan Unity engine.

Progres :

Tiga bot telah berhasil kami kembangkan dengan baik sesuai dengan rencana awal, menunjukkan dedikasi dan komitmen tim.

Cara menjalankan program :

Semua cara menjalankan program sudah tersedia di file “readme.txt” masing-masing folder project. (Note : Pastikan perangkat terhubung dengan internet yang stabil untuk menghindari gangguan dalam menjalankan aplikasi yang memerlukan akses internet.)

Pembagian Tugas :

Pengembangan Chatbot Translator
- Andrew Agustio Winata

Pengembangan Gamebot Pacman
- Elroy Saputra Duha
- Louis

Pengembangan Perceptual Interface Hand Gesture Drawing
- Frenz Luiz
- Leonardo Ade Hendrawan

Video Dokumentasi
- Elroy Saputra Duha

Laporan Tugas
- Andrew Agustio Winata
- Frenz Luiz

Lampiran Chatbot Translator

1. Main code (index.js)

```
const { Telegraf, Markup } = require("telegraf");
const translator = require("translation-google");
const gtts = require("gtts");
const config = require("./config.json");
const fs = require("fs");
const https = require("https");
const Tesseract = require("tesseract.js");

const bot = new Telegraf(config.BOT_TOKEN);

const languages = {
    English: "en",
    Mandarin: "zh-CN",
    Japanese: "ja",
    "South Korean": "ko",
    Russian: "ru",
    Tagalog: "tl",
};

let selectedLanguage = "";
let inputMethod = "";

function generateLanguageMarkup() {
    const buttons = Object.keys(languages).map((language) =>
        Markup.button.callback(language, `select_language ${language}`)
    );
    return Markup.inlineKeyboard(buttons, { columns: 2 });
}

bot.start((ctx) => {
    ctx.replyWithHTML(
        "✨ <b>Welcome GAN</b> ✨!\nSelamat Datang di Bot Translation by.Kel8 🚀🚀.\n\nSilakan pilih bahasa yang ingin dijadikan terjemahan :\n(Oh iya, untuk sementara kita hanya menerima Bahasa Indonesia
        generateLanguageMarkup()
    );
});
```

```
bot.action(/select_language (.+)/, (ctx) => {
    selectedLanguage = languages[ctx.match[1]];
    ctx.replyWithHTML(
        `Kamu memilih <b>${ctx.match[1]}</b>. Bagaimana kamu ingin mengirim teks yang ingin diterjemahkan?`,
        Markup.inlineKeyboard([
            Markup.button.callback("Teks", "text_input"),
            Markup.button.callback("Gambar", "image_input"),
        ])
    );
});

bot.action("text_input", (ctx) => {
    inputMethod = "text";
    ctx.replyWithHTML(
        `Silakan kirim teks yang ingin diterjemahkan ke <b>${getKeyByValue(
            languages,
            selectedLanguage
        )}</b> :`
    );
});

bot.action("image_input", (ctx) => {
    inputMethod = "image";
    ctx.reply(
        "Silakan kirim gambar yang berisi teks yang ingin diterjemahkan."
    );
});

bot.on("text", async (ctx) => {
    if (inputMethod !== "text") return;

    const text = ctx.update.message.text;

    if (!selectedLanguage) {
        ctx.reply("Dipilih dulu yaa bahasa yang diinginkan🤔🤔");
        return;
```

```
        }

        const loadingMessage = await ctx.reply("Sedang menerjemahkan...");

        const translation = await translator(text, {
            from: "auto",
            to: selectedLanguage,
        });

        await ctx.telegram.deleteMessage(ctx.chat.id, loadingMessage.message_id);

        ctx.replyWithHTML(
            `<b>Teks yang diterima</b> :\n${text}\n\n<b>Hasil translate (${getKeyByValue(
                languages,
                selectedLanguage
            )})</b> :\n${translation.text}`
        );

        const gttsInstance = new gtts(translation.text, selectedLanguage);
        const voiceFilePath = `./voice_${Date.now()}.mp3`;

        gttsInstance.save(voiceFilePath, (err) => {
            if (err) {
                console.error(err);
                ctx.reply("Terjadi kesalahan saat membuat voice note.");
            } else {
                ctx.replyWithVoice({ source: voiceFilePath }).then(() => {
                    fs.unlinkSync(voiceFilePath);
                    askForContinue(ctx);
                });
            }
        });
    });
```

```
    bot.on("photo", async (ctx) => {
        if (inputMethod !== "image") return;

        const loadingMessage = await ctx.reply("Sedang menerjemahkan...");

        const fileId = ctx.message.photo[ctx.message.photo.length - 1].file_id;
        const fileUrl = await bot.telegram.getFileLink(fileId);
        const imagePath = `./image_${Date.now()}.jpg`;

        await downloadFile(fileUrl, imagePath);

        const {
            data: { text },
        } = await Tesseract.recognize(imagePath, "eng");

        await ctx.telegram.deleteMessage(ctx.chat.id, loadingMessage.message_id);

        if (text.trim().length === 0) {
            ctx.reply("Tidak ada teks yang terdeteksi di gambar.");
            fs.unlinkSync(imagePath);
            return;
        }

        const joinedText = text.replace(/\n/g, " ");

        const translation = await translator(joinedText, {
            from: "auto",
            to: selectedLanguage,
        });

```

```
        ctx.replyWithHTML(
            `<b>Teks yang diterima dari gambar</b> :\n${joinedText}\n\n<b>Hasil translate (${getKeyByValue(
                languages,
                selectedLanguage
            )})</b> :\n${translation.text}`
        );

        const gttsInstance = new gtts(translation.text, selectedLanguage);
        const voiceFilePath = `./voice_${Date.now()}.mp3`;

        gttsInstance.save(voiceFilePath, (err) => {
            if (err) {
                console.error(err);
                ctx.reply("Terjadi kesalahan saat membuat voice note.");
            } else {
                ctx.replyWithVoice({ source: voiceFilePath }).then(() => {
                    fs.unlinkSync(voiceFilePath);
                    askForContinue(ctx);
                });
            }
        });

        fs.unlinkSync(imagePath);
    });

    function askForContinue(ctx) {
        ctx.reply(
            "Mau lanjut terjemahkan dengan bahasa yang sama atau ubah bahasa nih?",
            Markup.inlineKeyboard([
                Markup.button.callback("Lanjut", "again"),
                Markup.button.callback("Pilih bahasa lain", "different"),
            ])
        );
    }
```

```
    bot.action("again", (ctx) => {
        ctx.replyWithHTML(
            `Bagaimana kamu ingin mengirim teks yang ingin diterjemahkan ke <b>${getKeyByValue(
                languages,
                selectedLanguage
            )}</b> :`,
            Markup.inlineKeyboard([
                Markup.button.callback("Teks", "text_input"),
                Markup.button.callback("Gambar", "image_input"),
            ])
        );
    });

    bot.action("different", (ctx) => {
        selectedLanguage = "";
        ctx.reply(
            "Pilih bahasa yang ingin dijadikan terjemahan :",
            generateLanguageMarkup()
        );
    });

    function getKeyByValue(object, value) {
        return Object.keys(object).find((key) => object[key] === value);
    }

    bot.launch();
```

```
function downloadFile(url, filePath) {
    return new Promise((resolve, reject) => {
        const file = fs.createWriteStream(filePath);
        https
            .get(url, (response) => {
                response.pipe(file);
                file.on("finish", () => {
                    file.close(resolve);
                });
            })
            .on("error", (err) => {
                fs.unlink(filePath);
                reject(err);
            });
    });
}
```

2. Configuration

```
{
    "BOT_TOKEN": "7397635228:AAFNMZltGlhLTcrGGMRFfqiXJBFLW6FaDE8"
}
```

3. Dependencies

```
{
    "name": "bot_translation_js",
    "version": "1.0.0",
    "lockfileVersion": 3,
    "requires": true,
    "packages": {
        "": {
            "name": "bot_translation_js",
            "version": "1.0.0",
            "license": "ISC",
            "dependencies": {
                "gtts": "^0.2.1",
                "nodemon": "^3.1.0",
                "telegraf": "^4.16.3",
                "tesseract.js": "^5.1.0",
                "translation-google": "^0.2.1"
            }
        },
```

Lampiran Perceptual Interface Hand Gesture Drawing

1. Main code

```python
import cv2
import mediapipe as mp
from collections import deque
import numpy as np

# Initialize MediaPipe Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1, min_detection_confidence=0.7)
mp_drawing = mp.solutions.drawing_utils

bpoints = [deque(maxlen=1024)]
gpoints = [deque(maxlen=1024)]
rpoints = [deque(maxlen=1024)]
ypoints = [deque(maxlen=1024)]

blue_index = 0
green_index = 0
red_index = 0
yellow_index = 0

colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (0, 255, 255)]
colorIndex = 0

paintWindow = np.zeros((471, 636, 3)) + 255

paintWindow = cv2.rectangle(paintWindow, (40, 1), (140, 65), (0, 0, 0), 2)
paintWindow = cv2.rectangle(paintWindow, (160, 1), (255, 65), colors[0], -1)
paintWindow = cv2.rectangle(paintWindow, (275, 1), (370, 65), colors[1], -1)
paintWindow = cv2.rectangle(paintWindow, (390, 1), (485, 65), colors[2], -1)
paintWindow = cv2.rectangle(paintWindow, (505, 1), (600, 65), colors[3], -1)

cv2.putText(paintWindow, "CLEAR", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(paintWindow, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (150, 150, 150), 2, cv2.LINE_AA)

cv2.namedWindow('Paint', cv2.WINDOW_AUTOSIZE)

cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.flip(frame, 1)
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Process the frame and get hand landmarks
    result = hands.process(frame_rgb)

    index_finger_tip = None
    draw = False
    if result.multi_hand_landmarks:
        for hand_landmarks in result.multi_hand_landmarks:
            mp_drawing.draw_landmarks(frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
            index_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
            thumb_tip = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
            middle_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
            ring_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TIP]
            pinky_tip = hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP]

            # Convert landmark to pixel coordinates
            h, w, _ = frame.shape
            cx, cy = int(index_finger_tip.x * w), int(index_finger_tip.y * h)
            thumb_y = int(thumb_tip.y * h)
            middle_y = int(middle_finger_tip.y * h)
            ring_y = int(ring_finger_tip.y * h)
            pinky_y = int(pinky_tip.y * h)
```

```
            # Check if the index finger is up and others are down (hand open)
            if (index_finger_tip.y < thumb_tip.y and
                index_finger_tip.y < middle_finger_tip.y and
                index_finger_tip.y < ring_finger_tip.y and
                index_finger_tip.y < pinky_tip.y):
                draw = True

            if cy <= 65:
                if 40 <= cx <= 140:
                    bpoints = [deque(maxlen=512)]
                    gpoints = [deque(maxlen=512)]
                    rpoints = [deque(maxlen=512)]
                    ypoints = [deque(maxlen=512)]

                    blue_index = 0
                    green_index = 0
                    red_index = 0
                    yellow_index = 0

                    paintWindow[67:, :, :] = 255

                elif 160 <= cx <= 255:
                    colorIndex = 0
                elif 275 <= cx <= 370:
                    colorIndex = 1
                elif 390 <= cx <= 485:
                    colorIndex = 2
                elif 505 <= cx <= 600:
                    colorIndex = 3
            elif draw:
                if colorIndex == 0:
                    bpoints[blue_index].appendleft((cx, cy))
                elif colorIndex == 1:
                    gpoints[green_index].appendleft((cx, cy))
                elif colorIndex == 2:
                    rpoints[red_index].appendleft((cx, cy))
                elif colorIndex == 3:
                    ypoints[yellow_index].appendleft((cx, cy))

        else:
            bpoints.append(deque(maxlen=512))
            blue_index += 1
            gpoints.append(deque(maxlen=512))
            green_index += 1
            rpoints.append(deque(maxlen=512))
            red_index += 1
            ypoints.append(deque(maxlen=512))
            yellow_index += 1
```

```
        points = [bpoints, gpoints, rpoints, ypoints]
        for i in range(len(points)):
            for j in range(len(points[i])):
                for k in range(1, len(points[i][j])):
                    if points[i][j][k - 1] is None or points[i][j][k] is None:
                        continue
                    cv2.line(frame, points[i][j][k - 1], points[i][j][k], colors[i], 2)
                    cv2.line(paintWindow, points[i][j][k - 1], points[i][j][k], colors[i], 2)

        # Draw the color selection options on the tracking frame
        cv2.rectangle(frame, (40, 1), (140, 65), (0, 0, 0), 2)
        cv2.rectangle(frame, (160, 1), (255, 65), colors[0], -1)
        cv2.rectangle(frame, (275, 1), (370, 65), colors[1], -1)
        cv2.rectangle(frame, (390, 1), (485, 65), colors[2], -1)
        cv2.rectangle(frame, (505, 1), (600, 65), colors[3], -1)

        cv2.putText(frame, "CLEAR", (49, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 2, cv2.LINE_AA)
        cv2.putText(frame, "BLUE", (185, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
        cv2.putText(frame, "GREEN", (298, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
        cv2.putText(frame, "RED", (420, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
        cv2.putText(frame, "YELLOW", (520, 33), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (150, 150, 150), 2, cv2.LINE_AA)

        cv2.imshow("Tracking", frame)
        cv2.imshow("Paint", paintWindow)

        if cv2.waitKey(4) & 0xFF == ord('q'):
            break

cap.release()
cv2.destroyAllWindows()
```

Lampiran Gamebot Pacman

1. Game Controller

```
using System.Collections;
using System.Collections.Generic;
using System.Runtime.InteropServices;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;

namespace Pacman
{
    public class PacmanGameController : MonoBehaviour
    {
        [SerializeField] private PacmanCharacter pacmanCharacter = default;
        [SerializeField] private TextMeshProUGUI currentScoreTMP = default;
        [SerializeField] private PathFinding pathFinding = default;
        [SerializeField] private Blinky blinky = default;
        [SerializeField] private Transform pelletTransform = default;
        [SerializeField] private TextMeshProUGUI gameStatusTMP = default;
        [SerializeField] private GameObject RestartGO = default;
        public static PacmanGameController Instance { get; private set; } = null;
        public bool hasStartGame { get; private set; } = false;
        private int score = 0;
        public const float MOVEMENT_SPEED_MULTIPLIER = 5.5f;
        public bool hasLose { get; private set; } = false;

        private void Update()
        {
            if (Input.anyKeyDown)
            {
                hasStartGame = true;
            }
        }

        private void Awake()
        {
            Instance = this;
        }
        private void Start()
        {
            StartGame();
        }
        private void StartGame()
        {
            if (hasStartGame)
                return;
            RestartGO.SetActive(false);
            RestartGame();
```

```
            blinky.Init(pathFinding, pacmanCharacter);
        }

        public void AddScore(int score)
        {
            this.score += score;
            currentScoreTMP.text = $"Score: {this.score}";

            CheckGame();
        }
```

```
58          private bool hasWinGame()
59          {
60              Pellet[] pellets = pelletTransform.GetComponentsInChildren<Pellet>();
61
62              foreach(Pellet pellet in pellets)
63              {
64                  if (pellet.gameObject.activeSelf)
65                      return false;
66              }
67
68              return true;
69          }
70
71          private void CheckGame()
72          {
73              if (hasWinGame())
74              {
75                  gameStatusTMP.text = "WIN";
76                  RestartGO.SetActive(true);
77                  UpdatePlayerGameStatus();
78              }
79
80              if (hasLose)
81              {
82                  RestartGO.SetActive(true);
83                  gameStatusTMP.text = "LOSE";
84              }
85          }
86
87          public void UpdatePlayerGameStatus()
88          {
89              hasLose = true;
90              CheckGame();
91          }
92
93          public void RestartGame()
94          {
95              RestartGO.SetActive(false);
96              hasLose = false;
97              hasStartGame = false;
98              score = 0;
99              currentScoreTMP.text = $"Score: {score}";
100             blinky.ResetPos();
101             gameStatusTMP.SetText("");
102             pacmanCharacter.ResetGameCharacter();
103
104             Pellet[] pellets = pelletTransform.GetComponentsInChildren<Pellet>(true);
105             foreach (Pellet pellet in pellets)
```

```
106             {
107                 pellet.gameObject.SetActive(true);
108             }
109
110         }
111
112         private void OnDestroy()
113         {
114             Instance = null;
115         }
116     }
117 }
```

2. Pathfinding Tile

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Tilemaps;

namespace Pacman
{
    public class PathFinding : MonoBehaviour
    {
        [SerializeField] public Tilemap tilemap = default;

        private List<Node> openList = new List<Node>();
        private List<Node> closedList = new List<Node>();
        private Dictionary<Vector2Int, Node> allNodes = new Dictionary<Vector2Int, Node>();

        public List<Node> FindPath(Vector3Int start, Vector3Int target)
        {
            Node startNode = GetNode(new Vector2Int(start.x, start.y));
            Node targetNode = GetNode(new Vector2Int(target.x, target.y));

            openList.Clear();
            closedList.Clear();

            openList.Add(startNode);

            while (openList.Count > 0)
            {
                Node currentNode = openList[0];
                foreach (Node node in openList)
                {
                    if (node.F < currentNode.F || node.F == currentNode.F && node.H < currentNode.H)
                    {
                        currentNode = node;
                    }
                }

                openList.Remove(currentNode);
                closedList.Add(currentNode);

                if (currentNode == targetNode)
                {
                    return RetracePath(startNode, targetNode);
                }

                foreach (Node neighbor in GetNeighbors(currentNode))
                {
                    if (closedList.Contains(neighbor))
                    {
                        continue;
                    }

                    int newMovementCostToNeighbor = currentNode.G + GetDistance(currentNode, neighbor);
                    if (newMovementCostToNeighbor < neighbor.G || !openList.Contains(neighbor))
                    {
                        neighbor.G = newMovementCostToNeighbor;
                        neighbor.H = GetDistance(neighbor, targetNode);
                        neighbor.Parent = currentNode;

                        if (!openList.Contains(neighbor))
                        {
                            openList.Add(neighbor);
                        }
                    }
                }
            }

            return null;
        }
```

```
69          private List<Node> RetracePath(Node startNode, Node endNode)
70          {
71              List<Node> path = new List<Node>();
72              Node currentNode = endNode;
73
74              while (currentNode != startNode)
75              {
76                  path.Add(currentNode);
77                  currentNode = currentNode.Parent;
78              }
79
80              path.Reverse();
81              return path;
82          }
83
84          private int GetDistance(Node nodeA, Node nodeB)
85          {
86              int dstX = Mathf.Abs(nodeA.GridPosition.x - nodeB.GridPosition.x);
87              int dstY = Mathf.Abs(nodeA.GridPosition.y - nodeB.GridPosition.y);
88
89              return 10 * (dstX + dstY);
90          }
91
92          private List<Node> GetNeighbors(Node node)
93          {
94              List<Node> neighbors = new List<Node>();
95
96              Vector2Int[] directions = new Vector2Int[]
97              {
98                  new Vector2Int(0, 1),
99                  new Vector2Int(0, -1),
100                 new Vector2Int(-1, 0),
101                 new Vector2Int(1, 0)
102             };
103
104             foreach (var direction in directions)
105             {
106                 Vector2Int neighborPos = node.GridPosition + direction;
107                 if (tilemap.HasTile((Vector3Int)neighborPos))
108                 {
109                     if (node.Parent == null || (neighborPos - node.Parent.GridPosition).magnitude > 0.1f)
110                     {
111                         neighbors.Add(GetNode(neighborPos));
112                     }
113                 }
114             }
115
```

```
115
116             return neighbors;
117         }
118
119         private Node GetNode(Vector2Int gridPosition)
120         {
121             if (!allNodes.ContainsKey(gridPosition))
122             {
123                 allNodes[gridPosition] = new Node(gridPosition);
124             }
125             return allNodes[gridPosition];
126         }
127     }
128 }
```