**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЛЬВІВСЬКА ПОЛІТЕХНІКА**



**АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ**

**КОМП’ЮТЕРНИХ СИСТЕМ**

Task3 implement Server (HW) and Client (SW) parts of game (FEF)

Виконав :

Студент групи КІ-401

Онисько М.М.

Прийняв:

Федак П.Р

**2024**

**Завдання:**

1. Розробити серверну та клієнтську частину гри
2. Створити CI файл для автоматизованого тестування та компілювання
3. Оновити Readme.md
4. Додати тег про нову версію
5. Злити створену гілку до develop

**Теоретичні відомості:**

СІ (Continuous Integration, або CI) скрипт – це набір команд або сценарій, який автоматизує процеси інтеграції та тестування коду під час розробки. Його застосовують, щоб виявити помилки на ранніх етапах, забезпечити стабільність коду та автоматизувати робочі процеси. Часто CI-скрипти запускаються при кожній зміні коду (наприклад, при злитті pull request-ів у репозиторії), що дозволяє швидко виявляти і виправляти баги. Скрипти можуть виконувати такі дії, як:

1. **Тестування**: автоматичний запуск тестів для перевірки роботи коду.
2. **Збірка проекту**: компіляція або збірка проекту, особливо для мов, які цього потребують (наприклад, C++ або Java).
3. **Аналіз коду**: запуск статичного аналізу, щоб виявити помилки та потенційні проблеми.
4. **Деплоймент**: автоматичний деплой на сервер або у хмару, коли код проходить всі тести.

**Як написати та запустити CI-скрипт локально на Windows**

На Windows можна використовувати популярні CI-інструменти, наприклад, GitHub Actions, Jenkins, або GitLab CI/CD. Ось основні кроки:

1. **Створення CI-скрипта**:
   * Зазвичай CI-скрипти пишуться у форматі YAML. Наприклад, для GitHub Actions ви створите файл .yml у папці .github/workflows вашого проекту.
2. Структура файлу може виглядати так:

|  |
| --- |
| name: CI  on: [push, pull\_request]  jobs:  build:  runs-on: windows-latest  steps:  - name: Checkout code  uses: actions/checkout@v2  - name: Set up Python  uses: actions/setup-python@v2  with:  python-version: '3.x'  - name: Install dependencies  run: |  python -m pip install --upgrade pip  pip install -r requirements.txt  - name: Run tests  run: |  pytest |

**Запуск локально**:

* **З Jenkins**: Якщо у вас встановлений Jenkins, ви можете створити нове завдання, обрати тип "Pipeline", та вставити свій CI-скрипт у конфігурацію. Jenkins буде виконувати сценарій на вашій машині.
* **З GitHub Actions**: Локально його можна перевірити за допомогою інструмента [act](https://github.com/nektos/act), який дозволяє запускати GitHub Actions на локальному комп’ютері. Для цього:
  1. Встановіть act за допомогою команди:

scoop install act

* 1. Запустіть свій сценарій:

act

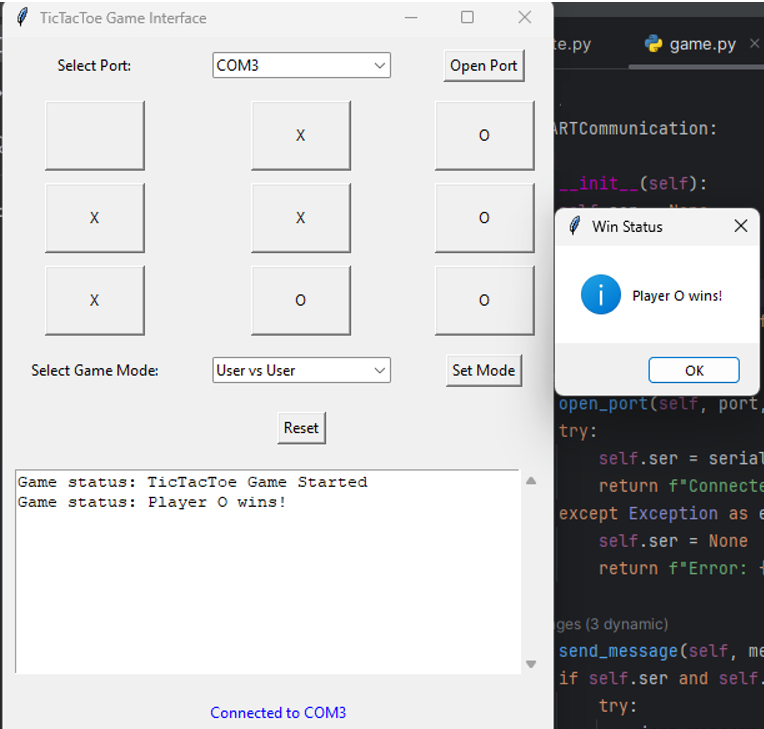
**Виконання**

**Код скрипту для HW частини**

|  |
| --- |
| #include <Arduino.h>  #include <ArduinoJson.h>  const int BOARD\_SIZE = 3;  char board[BOARD\_SIZE][BOARD\_SIZE];  char currentPlayer = 'X';  bool gameOver = false;  int gameMode = 0;  void initializeBoard() {  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  board[i][j] = ' ';  }  }  currentPlayer = 'X';  gameOver = false;  }  void sendJsonMessage(const char\* type, const char\* message) {  StaticJsonDocument<200> doc;  doc["type"] = type;  doc["message"] = message;  serializeJson(doc, Serial);  Serial.println();  }  void sendBoardState() {  StaticJsonDocument<300> doc;  doc["type"] = "board";  JsonArray boardArray = doc.createNestedArray("board");  for (int i = 0; i < BOARD\_SIZE; i++) {  JsonArray row = boardArray.createNestedArray();  for (int j = 0; j < BOARD\_SIZE; j++) {  row.add(String(board[i][j]));  }  }  serializeJson(doc, Serial);  Serial.println();  }  bool checkWin() {  for (int i = 0; i < BOARD\_SIZE; i++) {  if (board[i][0] == currentPlayer && board[i][1] == currentPlayer && board[i][2] == currentPlayer) return true;  if (board[0][i] == currentPlayer && board[1][i] == currentPlayer && board[2][i] == currentPlayer) return true;  }  if (board[0][0] == currentPlayer && board[1][1] == currentPlayer && board[2][2] == currentPlayer) return true;  if (board[0][2] == currentPlayer && board[1][1] == currentPlayer && board[2][0] == currentPlayer) return true;  return false;  }  bool checkDraw() {  for (int i = 0; i < BOARD\_SIZE; i++) {  for (int j = 0; j < BOARD\_SIZE; j++) {  if (board[i][j] == ' ') return false;  }  }  return true;  }  void aiMoveRandom() {  while (true) {  int row = random(0, BOARD\_SIZE);  int col = random(0, BOARD\_SIZE);  if (board[row][col] == ' ') {  board[row][col] = currentPlayer;  break; // Exit the loop after a valid move  }  }  }  void handleAiVsAi() {  while (!gameOver) {  if (checkDraw()) {  sendJsonMessage("win\_status", "It's a draw!");  gameOver = true;  return;  }  aiMoveRandom(); // AI makes a random move  if (checkWin()) {  String message = "Player " + String(currentPlayer) + " wins!";  sendBoardState();  sendJsonMessage("win\_status", message.c\_str());  gameOver = true;  return;  }  currentPlayer = (currentPlayer == 'X') ? 'O' : 'X'; // Switch players  sendBoardState(); // Send the board state after each move    }  }  bool makeMove(int row, int col) {  if (row >= 0 && row < BOARD\_SIZE && col >= 0 && col < BOARD\_SIZE && board[row][col] == ' ' && !gameOver) {  board[row][col] = currentPlayer;  if (checkWin()) {  String message = "Player " + String(currentPlayer) + " wins!";  sendJsonMessage("win\_status", message.c\_str());  gameOver = true;  } else if (checkDraw()) {  sendJsonMessage("win\_status", "It's a draw!");  gameOver = true;  } else {  currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';  }  return true;  }  return false;  }  void setup() {  Serial.begin(9600);  initializeBoard();  sendJsonMessage("info", "TicTacToe Game Started");  }  void loop() {  if (Serial.available() > 0) {  StaticJsonDocument<200> doc;  String input = Serial.readStringUntil('\n');  DeserializationError error = deserializeJson(doc, input);  if (!error) {  const char\* command = doc["command"];  if (strcmp(command, "MOVE") == 0) {  int row = doc["row"];  int col = doc["col"];  if (makeMove(row, col)) {  sendBoardState();  } else {  sendJsonMessage("error", "Invalid move.");  }  } else if (strcmp(command, "RESET") == 0) {  initializeBoard();  sendJsonMessage("game\_status", "Game reset.");  sendBoardState();  } else if (strcmp(command, "MODE") == 0) {  gameMode = doc["mode"];  String message = "Game mode set to " + String(gameMode);  sendJsonMessage("game\_mode", message.c\_str());  initializeBoard();  sendJsonMessage("game\_status", "Game reset.");  sendBoardState();  }  // AI move logic if applicable  if (gameMode == 1 && !gameOver && currentPlayer == 'O') {  aiMoveRandom(); // Make a random move for the AI  if (checkWin()) {  String message = "Player " + String(currentPlayer) + " wins!";  sendJsonMessage("win\_status", message.c\_str());  gameOver = true;  } else if (checkDraw()) {  sendJsonMessage("win\_status", "It's a draw!");  gameOver = true;  }  currentPlayer = 'X'; // Switch back to Player X  sendBoardState();  } else if (gameMode == 2 && !gameOver) {  handleAiVsAi(); // Handle AI vs AI  }  }  }  } |

**Код клієнтської частини**

|  |
| --- |
| import threading  import serial  import serial.tools.list\_ports  import json  import tkinter as tk  from tkinter import ttk, scrolledtext  from tkinter import messagebox  class UARTCommunication:  def \_\_init\_\_(self):  self.ser = None  def list\_ports(self):  return [port.device for port in serial.tools.list\_ports.comports()]  def open\_port(self, port, baud\_rate=9600):  try:  self.ser = serial.Serial(port, baud\_rate, timeout=1)  return f"Connected to {port}"  except Exception as e:  self.ser = None  return f"Error: {e}"  def send\_message(self, message):  if self.ser and self.ser.is\_open:  try:  json\_message = json.dumps(message)  self.ser.write((json\_message + "\n").encode())  return f"Sent: {json\_message}"  except Exception as e:  return f"Error: {e}"  return "Port not opened"  def receive\_message(self):  if self.ser and self.ser.is\_open:  try:  if self.ser.in\_waiting > 0:  response = self.ser.readline().decode().strip()  if response:  json\_response = json.loads(response)  return json\_response  except json.JSONDecodeError:  return "Error: Invalid JSON received"  except Exception as e:  return f"Error: {e}"  return "Port not opened"  def update\_game\_board(board, buttons):  for i in range(3):  for j in range(3):  buttons[i][j].config(text=board[i][j])  def send\_move(uart, row, col):  message = {"command": "MOVE", "row": row, "col": col}  uart.send\_message(message)  def set\_mode(uart, mode):  message = {"command": "MODE", "mode": mode}  uart.send\_message(message)  def reset\_game(uart):  message = {"command": "RESET"}  uart.send\_message(message)  def auto\_receive(uart, buttons, output\_text, root):  try:  if uart.ser and uart.ser.is\_open:  response = uart.receive\_message()  if response and response != "Port not opened":  if isinstance(response, dict):  if "board" in response:  update\_game\_board(response["board"], buttons)  else:  output\_text.insert(tk.END, f"Game status: {response['message']}\n")  if response.get("type") == "win\_status":  thread = threading.Thread(target=messagebox.showinfo, args=("Win Status",  response.get("message")))  thread.start()  else:  output\_text.insert(tk.END, f"Received: {response}\n")  output\_text.see(tk.END)  except Exception as e:  output\_text.insert(tk.END, f"Error: {str(e)}\n")  root.after(100, lambda: auto\_receive(uart, buttons, output\_text, root))  def start\_gui():  uart = UARTCommunication()  root = tk.Tk()  root.title("TicTacToe Game Interface")  port\_label = tk.Label(root, text="Select Port:")  port\_label.grid(row=0, column=0, padx=10, pady=10)  port\_var = tk.StringVar()  port\_combobox = ttk.Combobox(root, textvariable=port\_var, values=uart.list\_ports(), state="readonly")  port\_combobox.grid(row=0, column=1, padx=10, pady=10)  def open\_port\_callback():  status = uart.open\_port(port\_var.get())  status\_label.config(text=status)  if "Connected" in status:  auto\_receive(uart, buttons, output\_text, root)  else:  output\_text.insert(tk.END, f"Failed to connect: {status}\n")  open\_button = tk.Button(root, text="Open Port", command=open\_port\_callback)  open\_button.grid(row=0, column=2, padx=10, pady=10)  buttons = [[None for \_ in range(3)] for \_ in range(3)]  for i in range(3):  for j in range(3):  button = tk.Button(root, text=" ", width=10, height=3,  command=lambda row=i, col=j: send\_move(uart, row, col))  button.grid(row=i + 1, column=j, padx=5, pady=5)  buttons[i][j] = button  mode\_label = tk.Label(root, text="Select Game Mode:")  mode\_label.grid(row=4, column=0, padx=10, pady=10)  mode\_var = tk.StringVar(value="User vs User")  mode\_combobox = ttk.Combobox(root, textvariable=mode\_var,  values=["User vs User", "User vs AI", "AI vs AI"],  state="readonly")  mode\_combobox.grid(row=4, column=1, padx=10, pady=10)  def set\_mode\_callback():  mode\_index = mode\_combobox.current()  set\_mode(uart, mode\_index)  status\_label.config(text=f"Game mode set to {mode\_combobox.get()}")  mode\_button = tk.Button(root, text="Set Mode", command=set\_mode\_callback)  mode\_button.grid(row=4, column=2, padx=10, pady=10)  reset\_button = tk.Button(root, text="Reset", command=lambda: reset\_game(uart))  reset\_button.grid(row=5, column=1, padx=10, pady=10)  output\_text = scrolledtext.ScrolledText(root, width=50, height=10, wrap=tk.WORD)  output\_text.grid(row=6, column=0, columnspan=3, padx=10, pady=10)  status\_label = tk.Label(root, text="Status: Not connected", fg="blue")  status\_label.grid(row=7, column=0, columnspan=3, padx=10, pady=10)  root.mainloop()  if \_\_name\_\_ == "\_\_main\_\_":  start\_gui() |

****

**Рис.1 – Виконання клієнтської частини**

**Автоматизація**

**GitHub actions**

**ci.yml**

|  |
| --- |
| name: CI Workflow  on:  push:  branches:  - develop  - feature/develop/task3  pull\_request:  branches:  - develop  jobs:  build:  runs-on: ubuntu-latest  steps:  # Checkout the repository  - name: Checkout code  uses: actions/checkout@v3  # Set up Python environment  - name: Set up Python 3.x  uses: actions/setup-python@v4  with:  python-version: '3.x'  # Install Python dependencies  - name: Install dependencies  run: |  python -m pip install --upgrade pip  pip install -r TicTacToeSWPart/requirements.txt  pip install pytest    # Run Python tests  - name: Run tests  run: |  python -m pytest --junitxml=test-reports/results.xml TicTacToeSWPart/tests.py    # Install Arduino CLI  - name: Set up Arduino CLI  run: |  wget https://downloads.arduino.cc/arduino-cli/arduino-cli\_latest\_Linux\_64bit.tar.gz  tar -xvf arduino-cli\_latest\_Linux\_64bit.tar.gz  sudo mv arduino-cli /usr/local/bin/  arduino-cli config init  # - name: Add Arduino AVR package URL  # run: |  # arduino-cli config set board\_manager.additional\_urls https://downloads.arduino.cc/packages/arduino/hardware/avr/1.8.5/avr-1.8.5.json  - name: Install Arduino AVR core  run: |  arduino-cli core update-index  arduino-cli core install arduino:avr # Install the Arduino AVR core  - name: Install ArduinoJson library  run: |  arduino-cli lib install ArduinoJson  - name: Compile Arduino Sketch for Arduino Nano (Old Bootloader)  run: |  mkdir -p build # Ensure the build directory exists  arduino-cli compile --fqbn arduino:avr:nano:cpu=atmega328old --output-dir build HWPart/TicTacToe/TicTacToe.ino  # Collect binaries as artifacts  - name: Upload binaries  uses: actions/upload-artifact@v3  with:  name: compiled-files  path: build/\*.bin  # Collect test results as artifacts  - name: Upload test reports  uses: actions/upload-artifact@v3  with:  name: test-reports  path: test-reports/results.xml |

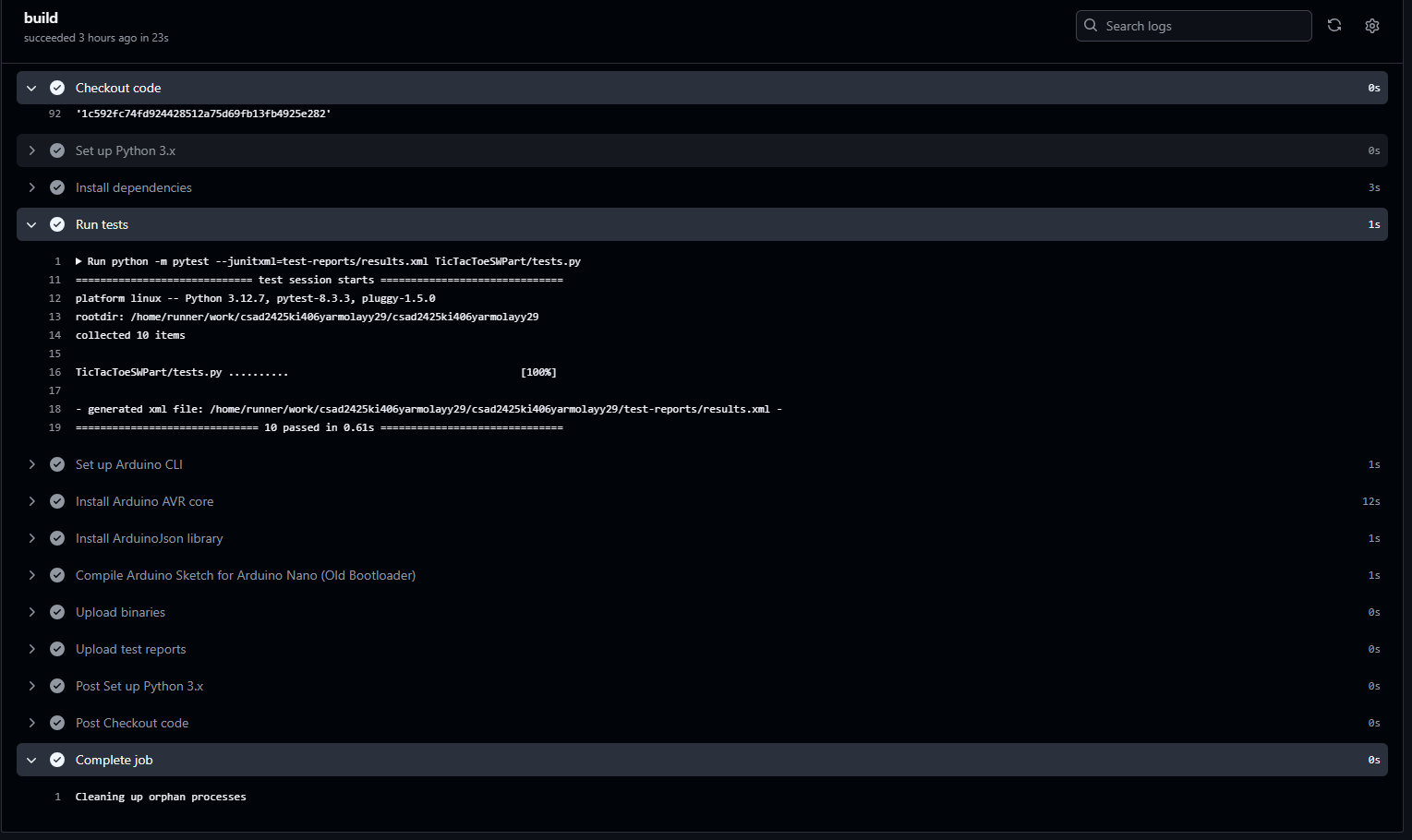


Рис.2 – Виконання GitHub actions скрипту

**ArduinoTest.ps – Лістинг СІ скрипту**

|  |
| --- |
| # ------------------------- CONFIGURABLE VARIABLES -----------------------------------------  $board = "arduino:avr:nano:cpu=atmega328old"  $baudRate = 9600  $sketch = "C:\Users\onisk\Desktop\csad2425ki401onyskomm15\Server\TicTacToe\TicTacToe.ino"  $serialLog = "serial\_output.log"  # ------------------------------------------------------------------------------------------  function Check-ArduinoCLI {  if (-not (Get-Command arduino-cli -ErrorAction SilentlyContinue)) {  Write-Output "arduino-cli не знайдено. Будь ласка, встановіть його."  exit 1  }  }  function Select-ArduinoPort {  Write-Output "Доступні порти:"  $ports = & arduino-cli board list | Select-Object -Skip 1 | ForEach-Object { $\_.Split(" ")[0] }  $ports | ForEach-Object { Write-Output "$([array]::IndexOf($ports, $\_)) - $\_" }  $portNumber = Read-Host -Prompt "Виберіть номер порту для вашої плати Arduino Nano"  $global:port = $ports[$portNumber]  if (-not $global:port) {  Write-Output "Невірний вибір порту."  exit 1  }  Write-Output "Обраний порт: $global:port"  }  function Compile-Sketch {  Write-Output "Компілляція скетчу..."  & arduino-cli compile --fqbn $board $sketch  if ($LASTEXITCODE -ne 0) {  Write-Output "Помилка компіляції."  exit 1  }  Write-Output "Компіляція успішна."  }  function Upload-Sketch {  Write-Output "Завантаження скетчу на плату Arduino Nano через порт $global:port..."  & arduino-cli upload -p $global:port --fqbn $board $sketch  if ($LASTEXITCODE -ne 0) {  Write-Output "Помилка завантаження."  exit 1  }  Write-Output "Завантаження успішне."  }  function Run-Tests {  Write-Output "Виконання тестів..."  $serialPort = new-Object System.IO.Ports.SerialPort $global:port, $baudRate  $serialPort.Open()  Start-Sleep -Seconds 2 # Час на перезавантаження Arduino і початок виводу  $serialPort.WriteLine('{"command":"RESET"}')  Start-Sleep -Seconds 1  $serialPort.WriteLine('{"command":"MOVE","row":0,"col":0}')  Start-Sleep -Seconds 1  $serialPort.WriteLine('{"command":"MOVE","row":1,"col":1}')  $output = ""  $stopwatch = [System.Diagnostics.Stopwatch]::StartNew()  while ($stopwatch.Elapsed.TotalSeconds -lt 5) {  if ($serialPort.BytesToRead -gt 0) {  $output += $serialPort.ReadExisting()  }  }  $serialPort.Close()  $output | Out-File -FilePath $serialLog  if ($output -match "TicTacToe Game Started" -and $output -match '"type":"board"') {  Write-Output "Тести пройдені успішно."  } else {  Write-Output "Тести не пройдені. Перевірте лог виводу серійного порту."  exit 1  }  }  Check-ArduinoCLI  Select-ArduinoPort  Compile-Sketch  Upload-Sketch  Run-Tests |

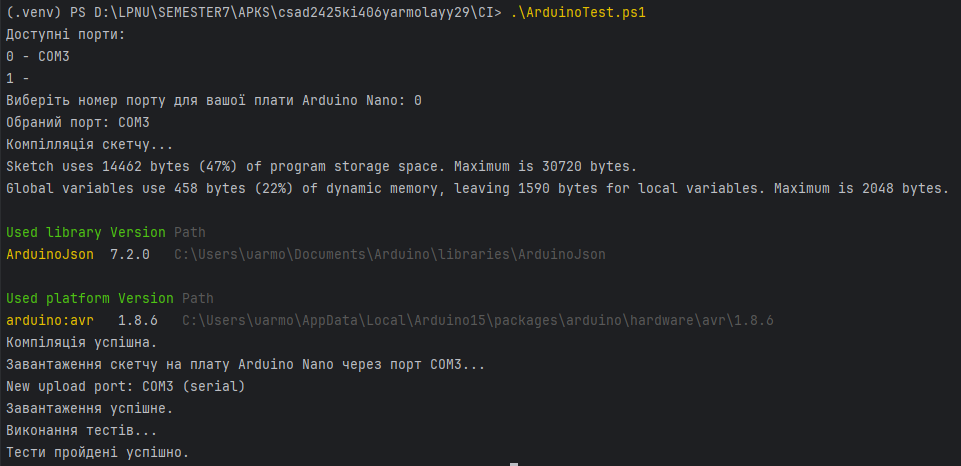
yuri

Рис.3 – Виконання СІ скрипту

**Висновки**

На лабораторній роботі я повністю розробив серверну та клієнтську частину гри. Також розробив СІ скрипт для автоматизації компілювання скрипту та завантаження його у плату для подальшого тестування та використання.

**Посилання**

1. <https://www.arduino.cc/>

2. <https://uk.wikipedia.org/wiki/UART>

3. <https://docs.python.org/uk/3/library/tkinter.html>

4. <https://pyserial.readthedocs.io/en/latest/pyserial.html>

5. <https://medium.com/@kaikok/using-gitlab-ci-to-automate-daily-tasks-c18f45c49378>