Tic-Tac-Toe (3x3) report

Jack Hao (Ruojie Hao)
CS506_01_IN: Programming for Computing, City University of Seattle,
haoruojie@cityuniversity.edu

1. INTRODUCTION

Tic-Tack-Toe is often used as an introductory case study for Computer area in System Design, OOP and AI learning. To fulfill the requirements, I implement the core logic of the game by providing both single-player (versus AI) and two-player game modes. The back-end uses Python with Flask to build a WebSocket protocol (I initially use POST method but with high latency). While the front-end uses WebGPU technology to realize stereoscopic rendering effect, making the game interface more interactive.

In addition, the game records the player/s actions in the tictactoe.txt file every time the player takes a step to ensure the traceability of the game steps. If an AI opponent (default) is used, the Negamax algorithm is used to calculate the optimal pace of the opponent to make the game a little more interesting. The following section describes the technical design, system architecture, and usage of the project.

Abstract

This report introduces a Tic-Tac-Toe (3x3) game based on Object Oriented Programming (OOP) implementation. The game uses Python as the back-end language, combines Flask-SocketIO for real-time bidirectional communication, and utilizes the EasyAI suite to implement Negamax-based AI pairs. Player steps are recorded.

The front-end uses HTML, CSS, JavaScript and WebGPU technology to present a 3D rendered game interface. Some of the more difficult computer graphic parts (such as the pipeline, rotation, translation, and shapes) were generated by AI and adapted by me based on experience and finally integrated into the code.

Keywords
Game, Python, JavaScript, OOP, WebGPU, WebSocket, Negamax

2. TECHNICAL DESIGN AND SYSTEM ARCHITECTURE

System Overview

The system is divided into two parts:

- Back-end (Python): It is responsible for game logic, AI decision making, step logging and real-time communication. A game class is responsible for managing the game progress based on user input from front-end or AI decision.

- Front-end (Web): Implement the visualization interface using HTML, CSS and JavaScript, and present 3D rendering effect by WebGPU technology, and exchange data with the back-end through WebSockets.

Back-end Design

1. Main class design

The core of this project is the game class, its main functions include:

- Game initialization and status management: Decide the current game mode (single or double).

- Judgement of win/loss: Judge whether a player has achieved the win condition by comparing the predefined win combinations (8 in total).

- Steps Record: Record the player's selected position to tictactoe.txt after each walk.

- AI opponent: When the game mode is single player (1P mode), the Negamax algorithm is used to determine the AI's moves.

2. WebSockets Connection

Real-time communication is established based on WebSocket protocol, to simplify this process in Python, I first tried to use the simplest POST method to implement the communication part between front-end and back-end, after implementing the basic game functionality, I used Flask-SocketIO to establish a WebSocket connection.

When a player clicks on the board, the front-end sends corresponding information (e.g. position) to the back-end, and it processes the position based on the internel logic and then sends the updated information back to the front-end. The webSocket provide real-time, bidirectional communication with lower latency and overhead compared to the request-response nature of POST.

Front-end Design

HTML5 Canvas is used as the graphic rendering area, and WebGPU is used to render the board and pieces. I used 2 shapes: cylinders and rings, to achieve all the 3D modeling.

At the beginning of the game, 9 positions are deployed with pieces that are hidden. When the user clicks on the corresponding position of the board, WebSocket communication ensures that the game status is updated to the back-end for computation. The result of it then sent back to the front-end, including:

- Position of the pieces
- Win/Loss
- Whether played by AI (affects the visual effect of Win/Loss)

The WebGPU shader applies per-instance transformations, projects vertices using camera matrices, calculates diffuse lighting based on the dot product of transformed normals and a light direction derived from the mouse position, and modulates fragment color accordingly. But the blinking of the winning pieces is based on a 2D light and shadow effect realized by CSS filter and animation. The combination of the two allows it to achieve the best visual effect in the simplest way.

3. Conclusion

This project successfully integrates OOP, WebGPU, and WebSockets to realize a full-featured and visually rich Tic-Tac-Toe game. I spent a lot of time optimizing the 3D effects, as I envision the potential of WebGPUs. In the future, I can further expand this technique into a deeper area with the selfless help of AI. Such as physical collisions, ray tracing, or better AI algorithms.

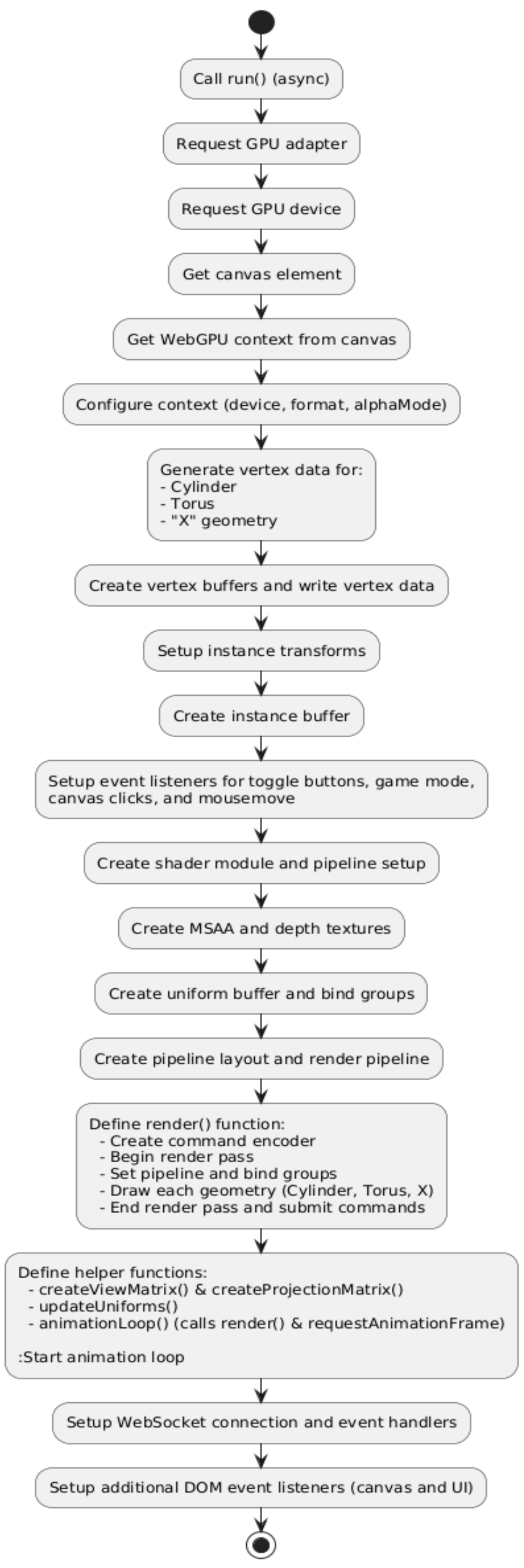
4. REFERENCES

easyAI. easyAI 1.0.0.2 documentation. (n.d.). https://zulko.github.io/easyAI/

W3C. (2025, January 31). WebGPU. https://www.w3.org/TR/webgpu/

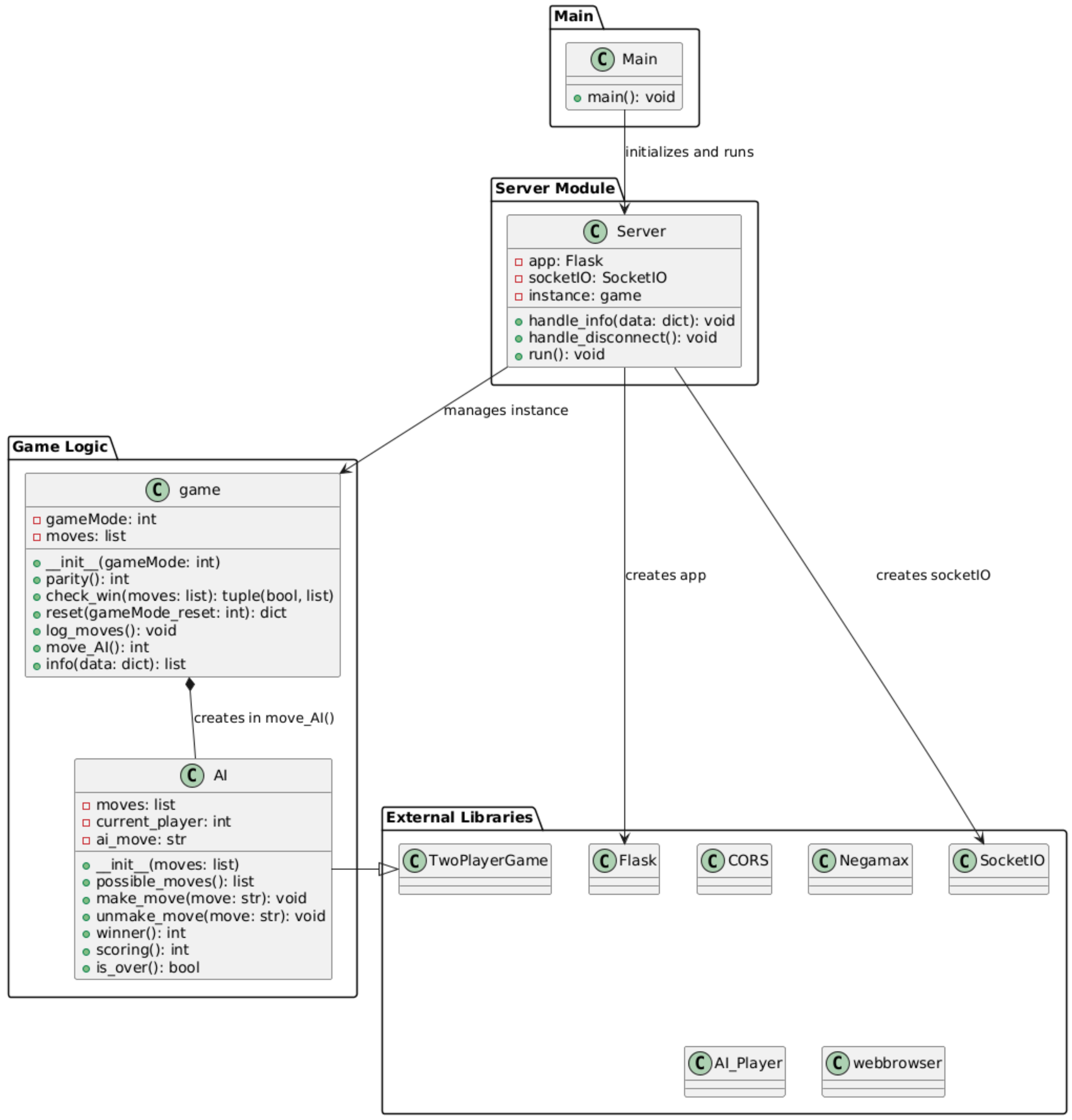
Flask-SocketIO. Flask-SocketIO. documentation. (n.d.). https://flask-socketio.readthedocs.io/en/latest/

OpenAI. (2025, February 6). ChatGPT (v4.0) [Large language model]. https://chat.openai.com


Call run() (async)
Request GPU adapter
Request GPU device
Get canvas element
Get WebGPU context from canvas
Configure context (device, format, alphaMode)
Generate vertex data for: - Cylinder - Torus - "X" geometry
Create vertex buffers and write vertex data
Setup instance transforms
Create instance buffer
Setup event listeners for toggle buttons, game mode, canvas clicks, and mousemove
Create shader module and pipeline setup
Create MSAA and depth textures
Create uniform buffer and bind groups
Create pipeline layout and render pipeline
Define render() function: - Create command encoder - Begin render pass - Set pipeline and bind groups - Draw each geometry (Cylinder, Torus, X) - End render pass and submit commands
Define helper functions: - createViewMatrix() & createProjectionMatrix() - updateUniforms() - animationLoop() (calls render() & requestAnimationFrame) :Start animation loop
Setup WebSocket connection and event handlers
Setup additional DOM event listeners (canvas and UI)

Flow chart of index.js (front-end)

Appendices and Annexures



UML diagram of index.py (back-end code)