DraWar

Joc interactiv cu recunoaștere AI

Proiect Informatică Aplicată 4

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

https://github.com/OctaVianu8/DraWar

Echipa:

Nițu Eriko-Laurențiu
Crețoiu Teodora-Elena
Stănescu Matei-Octavian

Anul 2025

Cuprins

1 Introducere

Despre proiect

DraWar este o aplicație web interactivă care transformă desenarea într-o competiție captivantă între doi jucători și inteligența artificială. Conceptul este simplu, dar provocator: desenează mai rapid și mai clar decât adversarul tău pentru ca AI-ul să îți ghicească primul desenul!

Fiecare rundă începe cu un cuvânt secret afișat pe ecranele ambilor jucători. Aceștia desenează simultan folosind un canvas interactiv, iar desenele lor sunt analizate în timp real de un model AI. Primul jucător al cărui desen este recunoscut corect de AI câștigă punctul rundei.

1.1 Motivația proiectului

Am vrut să explorăm modurile de a integra într-un mod elegant și inovativ tehnologiile moderne, precum inteligența artificială, într-o experiență user-friendly și inedită de gaming. Proiectul îmbină:

- ✔ Interfață web intuitivă și responsivă
- ✔ Procesare grafică în timp real pe bază de computer vision
- ✔ Arhitectură simplă, implementată în Python

1.2 Viziunea pe termen lung

În versiunile viitoare, ne imaginăm o platformă extinsă care include:

- Sistem de conturi și autentificare
- Lobby pentru găsirea adversarilor
- Clasamente și statistici
- Bază de date extinsă cu sute de obiecte recunoscute
- Salvarea progresului și a istoricului jocurilor

2 Funcționalități principale

2.1 Fluxul de joc

Experiența de joc:

1. **Start rundă** – Sistemul generează automat un cuvânt secret
2. **Desenare** – Ambii jucători desenează simultan pe canvas-uri separate
3. **Analiză live** – Desenele sunt trimise continuu către backend pentru recunoaștere
4. **Ghicire AI** – Primul desen corect identificat oprește runda
5. **Punctaj** – Câștigătorul rundei primește punctul
6. **Următoarea rundă** – Procesul se repetă cu un nou cuvânt

2.2 Caracteristici tehnice

Generare cuvinte Sistemul alege aleator cuvinte din baza de date pentru fiecare rundă

Canvas interactiv Zonă de desen responsivă, funcțională pe orice dispozitiv

Comunicare real-time Transmitere instantanee a desenelor către server (HTTP/WebSockets)

Model AI optimizat Recunoaștere rapidă bazată pe arhitecturi de tip QuickDraw

Detecție automată Identificarea momentului exact când AI ghicește corect

Gestionare rundă Timer de 30 secunde, scoruri actualizate automat

Interfață prietenoasă Design simplu, intuitiv, accesibil

Rezultate clare Afișare câștigător, scoruri finale, statistici rundă

3 Arhitectura aplicației

Arhitectură client-server

Aplicația folosește o arhitectură simplă și eficientă, în care browserul comunică direct cu un server Flask care orchestrează întreaga logică de joc și integrarea cu modelul AI.

3.1 Frontend

Frontend-ul este realizat cu tehnologii web standard și include:

- **HTML & CSS** – Interfață grafică
- **Canvas API** – Zonă de desen interactivă pentru fiecare jucător
- **JavaScript** – Logică client-side pentru capturarea desenelor și comunicare cu serverul

3.2 Backend

Backend-ul este implementat integral în Python și gestionează:

- **Flask Server** – Servirea aplicației și rutarea cererilor
- **Game Logic** – Orchestrarea rundelor, regulilor și progresului
- **Word Generation** – Selectarea aleatorie a cuvintelor pentru fiecare rundă
- **Image Processing** – Recepționarea și preprocesarea desenelor
- **Computer Vision** – Trecerea desenelor prin modelul de recunoaștere
- **Score Management** – Actualizarea și transmiterea scorurilor
- **WebSocket Communication** – Comunicare în timp real între jucători folosind Flask-SocketIO

3.3 Modelul AI

Modelul de inteligență artificială este antrenat să recunoască obiecte comune din desene simple, similar cu arhitectura Google QuickDraw. Caracteristici:

- Preprocesare automată a imaginilor (resize, normalizare, apply kernels)
- Suport pentru mai multe tipuri de desene (diferite obiecte)
- Implementare în PyTorch cu arhitectură CNN (Convolutional Neural Network)

4 Instalare și rulare

4.1 Cerințe preliminare

Dependențe necesare:

- Python 3.x
- Flask & Flask-SocketIO (pentru server și comunicare real-time)
- PyTorch (pentru modelul AI)
- FastAPI & Uvicorn (pentru AI inference server)
- NumPy, Pillow (pentru procesare imagini)

4.2 Pornirea aplicației

Pasul 1: Deschideți un terminal în directorul proiectului

Pasul 2: Porniți serverul Flask:

```
python3 app.py
```

Pasul 3: Deschideți browserul și accesați:

`localhost:5003`

Pasul 4: GLHF!

5 Echipa

Stănescu Matei-Octavian:
Contribuții:

- Implementarea claselor de bază (Game, Player, Round, Lobby)
- Configurarea serverului Flask și a comunicării WebSocket
- Antrenarea modelului CNN pentru recunoașterea desenelor
- Integrarea serviciului AI cu backend-ul principal
- Sistem de efecte sonore și feedback vizual

Dificultăți întâmpinate:

- Odată ce am introdus WebSocket-uri am început să avem race conditions și jocul fie se bloca, fie era într-o stare la mine și altă stare la Eriko/Teo. **Rezolvare:** Crearea unei unici surse de adevăr în server care are verificări la fiecare pas.
- De aceea am încercat să folosesc State Machine Design Pattern, dar m-am lovit de bug-uri precum Player în Lobby dar nu în Game. **Rezolvare:** Multe print-uri cu starea curentă.
- Inițial am vrut să integrăm Gemini API pentru recunoasterea desenelor, dar am realizat repede că nu e fezabil, pentru ca noi aveam nevoie de ceva care sa proceseze desenul și sa dea un output aproape instant. Așa ca am folosit un model local de rețea neurală convoluțională, antrenat local pe dataset-ul Google QuickDraw.
- La integrarea AI-ului cu backend-ul, am avut probleme cu formatul datelor: canvas-ul din browser trimitea imagini PNG base64 de 400x400 pixeli color, dar modelul CNN aștepta array-uri NumPy de 28x28 grayscale normalizate între 0 și 1. **Rezolvare:** Am creat (Eriko a creat) un pipeline de preprocesare în image_processor.py care face resize și normalizare înainte de a trimite datele către serviciul AI.

Crețoiu Teodora-Elena:
Contribuții:

- Dezvoltarea interfeței utilizator (HTML/CSS)
- Implementarea canvas-ului interactiv pentru desenare
- Logica client-side în JavaScript (comunicare WebSocket, events)
- Testarea și debugging-ul aplicației

Dificultăți întâmpinate:

- Au fost necesare numeroase modificări CSS pentru a asigura o experiență cât mai bună atât pe desktop, cât și pe dispozitive mobile. Acest proces a implicat mult trial and error pentru a obține un layout stabil și ușor de utilizat.
- Crearea unui canvas responsive a fost dificilă din cauza diferențelor dintre coordonatele mouse-ului și cele ale evenimentelor touch. Pe mobil apăreau frecvent probleme precum scroll-ul paginii în timpul desenului.
- Utilizarea variabilelor globale precum lobbyState și isDrawing a îngreunat procesul de debugging. Resetarea completă a stării la finalul jocului sau la ieșirea din lobby a necesitat o atenție suplimentară.
- Sincronizarea scorurilor, listei de jucători și a timerelor în timp real a fost dificil de gestionat. Evenimentele primite rapid de la server puteau duce la inconsistențe vizuale în interfață.
- Coordonarea mai multor jucători conectați simultan prin WebSocket a generat probleme în timpul testării. Atunci când mai mulți utilizatori trimiteau desene în același timp, apăreau situații de tip race condition și erori în lanț.
- Administrarea tranzițiilor dintre fazele jocului și a timerelor pentru runde a fost considerabil complexă. Timer-ele trebuiau anulate corect pentru a evita declanșarea multiplă a acelorași evenimente.

Nițu Eriko-Laurențiu:
Contribuții:

- Preprocesarea datelor din datasetul QuickDraw
- Implementarea AI inference server cu FastAPI
- Optimizarea modelului și deploy pe Hugging Face Spaces
- Integrarea componentelor și deploy pe Render

Dificultăți întâmpinate:

- Inițial am încercat să rulăm modelul AI direct în backend-ul Flask, dar încărcarea modelului PyTorch bloca serverul la startup și inferența încetinea răspunsurile WebSocket pentru toți jucătorii. **Rezolvare:** Am separat AI-ul într-un microserviciu independent cu FastAPI (deploy pe Hugging Face Spaces), iar backend-ul comunică cu el prin HTTP POST, trimițând imaginea ca un array JSON flatten-uit de 784 valori float.
- Serverul de AI adoarme când nu e folosit, iar prima cerere dădea mereu eroare. **Rezolvare**: Am crescut timeout-ul jocului la 15 secunde pentru a lăsa AI-ul să se trezească.
- PyTorch este imens și îngreuna serverul. **Rezolvare**: Am folosit versiunea "CPU-only" a PyTorch, care este mult mai mică și rapidă pentru servere fără placă video.
- Pentru deployment planurile gratuite nu au destul RAM pentru a rula și jocul și AI-ul în același loc.
- Ecranele de "Winner" și "Countdown" se suprapuneau greșit pe iPhone. **Rezolvare**: Le-am dat prioritate maximă de afișare (z-index). **Rezolvare**: Am separat proiectul: Backend-ul pe Render (consum mic) și AI-ul pe Hugging Face (optimizat pentru ML).

6 Concluzii și perspective

6.1 Realizări

Am reușit să creăm o aplicație completă și funcțională care demonstrează:

- ✔ Integrarea cu succes a unui model AI într-o aplicație web interactivă
- ✔ Comunicare în timp real între jucători folosind WebSocket-uri
- ✔ Suport pentru până la 10 jucători simultan într-un lobby
- ✔ Recunoașterea a peste 45 de categorii de obiecte desenate
- ✔ Interfață intuitivă cu feedback vizual și sonor
- ✔ Arhitectură scalabilă cu separarea AI server-ului de backend
- ✔ Deploy funcțional pe platformele cloud (Render și Hugging Face Spaces)

6.2 Dezvoltări viitoare

Următoarele etape pe care le avem în vedere:

1. **Multiplayer extins** - Support pentru mai mult de 2 jucători simultan
2. **Sistem de conturi** - Autentificare, profiluri, istoric jocuri
3. **Lobby și matchmaking** - Găsirea automată a adversarilor
4. **Bază de date extinsă** - Sute de cuvinte și categorii tematice
5. **Clasamente globale** - Top jucători, achievements, badges
6. **Versiune mobilă** - Aplicație nativă iOS/Android

Mulțumim pentru interesul acordat!