April 14, 2026 · 12 min read · By Yiğit Konur

Pokémon Red'i çok oyunculu paylaşımlı bir arcade'e dönüştürdüm — işte nasıl

software-engineering

 Canlı oyna → (https://yigitkonur.github.io/wasm-pokemon-red/) — tek bir örnek, herkes birlikte oynuyor.

fikir basitti aslında: siteyi ziyaret eden insanlar Pokémon Red'i birlikte oynayabilse, kumandayı aralarında geçirseler — Twitch Plays benzeri ama bir blog üzerine gömülü, düzgün kayıt durumu olan ve maliyeti yok denecek kadar az bir şey olsa nasıl olurdu?

bu, o yapının tam inşa günlüğü.

---

ne yapıyor

tarayıcıda tek bir Pokémon Red kopyası çalışıyor. sayfayı açan herkes katılabiliyor. bir seferde tek oyuncu kumandayı elinde tutuyor; beş saniye hareketsizlik sonrası sıra kuyruktaki bir sonraki kişiye geçiyor. etrafta kimse yoksa bir AI botu devreye giriyor, overworld'de dolaşıyor ya da savaşıyor. oyunun yanında canlı bir sohbet var. kayıt durumu oturumlar arası korunuyor: sekmeyi kapatın, yarın geri gelin — oyun bıraktığınız yerde.

-

adım 1 — ROM'u kaynaktan derleme

her şey pret/pokered (https://github.com/pret/pokered) ile başlıyor; bu, Pokémon Kırmızı ve Mavi'nin tamamen tersine mühendislikle elde edilmiş disassembly'si. onlarca yıllık topluluk emeğiyle 1 MB'lık bir Game Boy kartuşu okunabilir, açıklamalı assembly'ye dönüştürüldü.

```bash
git clone https://github.com/pret/pokered.git
cd pokered
make
# pokered.gbc üretilir
sha1sum pokered.gbc   # roms.sha1 ile doğrula
```

rgbds (https://rgbds.gbdev.io) (≥0.9) gerekiyor. assembler, linker ve fix aracının tamamı dahil. `make` , `pokered.gbc` üretiyor — doğrulanmış bir kartuş imajı. SHA1, `roms.sha1` 'de

sabitleniyor; bu ROM'u kamuya açık biçimde dağıtacaksanız tekrarlanabilir derleme önemli.

---

adım 2 — emülatörü WebAssembly'ye derleme

binjgb (https://github.com/nicknassar/binjgb), C ile yazılmış, döngü doğruluklu bir Game Boy emülatörü. küçük, iyi yapılandırılmış ve Emscripten derlemesi için doğru kancaları sunuyor.

```bash
git clone https://github.com/nicknassar/binjgb.git
source ~/emsdk/emsdk_env.sh

emcc binjgb/src/emulator.c binjgb/src/joypad.c binjgb/src/apu.c \
     web/binjgb/wrapper.c \
     -o dist/binjgb.js \
     -s EXPORTED_FUNCTIONS=@web/binjgb/exported.json \
     -s MODULARIZE=1 \
     -s ALLOW_MEMORY_GROWTH=1 \
     -O3
```

`web/binjgb/exported.json`, tarayıcı kabuğunun ihtiyaç duyduğu sekiz C fonksiyonunu listeliyor. `wrapper.c` ise binjgb kaynağını değiştirmeden kare-mükemmel 60fps canvas döngüsü ekleyen ve ses tamponunu JavaScript'e açan bir kaplama katmanı. derleme çıktısı iki dosya: `binjgb.js` (çalışma zamanı ve bağlantı kodu) ile `binjgb.wasm` (asıl bytecode).

---

adım 3 — tarayıcı kabuğunu bağlama

`web/player.html` bağımsız bir HTML sayfası. çerçeve yok, paket yöneticisi yok. `binjgb.js`'yi yükleyip emülatörü ROM baytlarıyla başlatıyor, sonra `requestAnimationFrame` döngüsünde ilerlettiriyor.

-

klavye girişi, `player.js` içindeki ince bir dağıtıcıdan geçiyor; bu dağıtıcı DOM `KeyboardEvent` kodlarını sekiz Game Boy düğmesine eşleyip `Module._joypad_set`'i çağırıyor. canvas, piksel sanatının net kalması için en-yakın-komşu interpolasyonuyla 3x büyütülüyor.

---

adım 4 — çok oyunculu sıra sistemi

en ilginç mühendislik, sıra sistemi. iki dosyada yaşıyor: `web/multiplayer.js` (tarayıcı istemcisi) ve Railway'de çalışan `server/server.js` (Node.js WebSocket sunucusu).

protokol minimal:

`join`		
`input`		
`state_sync`		
`turn_granted`		
`turn_revoked`		
`chat`		
`ai_granted`		
`ai_revoked`		

sunucu tek bir `activeMode` takip ediyor: `idle`, `human` veya `ai`. yalnızca o anki mod sahibinin girişleri emülatöre iletiliyor. diğer herkes izliyor.

-

adım 5 — Redis ile kayıt durumunu kalıcı hale getirme

kalıcılık olmadan oyun, sunucu her yeniden başladığında ya da son oyuncu ayrıldığında sıfırlanırdı. çözüm: her sıra devri sırasında emülatörün tam RAM'ini serileştirmek ve Redis'e kaydetmek.

```javascript
// sıra devri sırasında (server.js)
const stateBlob = emulator.serializeState();      // ~8 KB base64
await redis.set('pokemon:state', stateBlob);

// yeni oyuncu katıldığında
const saved = await redis.get('pokemon:state');
if (saved) ws.send(JSON.stringify({ type: 'state_sync', state: saved }));
```

upstash redis sunucusuz: kalıcı bağlantı yok, provizyon yok. REST API'si Railway'in geçici konteyner modeliyle mükemmel uyuşuyor. ücretsiz katman günlük ~10.000 istek kapsıyor — hobi ölçeği için fazlasıyla yeterli.

adım 6 — AI botu

`autoplay.js`, bouletmarc/PokeBot (https://github.com/bouletmarc/PokeBot) projesinin basitleştirilmiş bir uyarlaması. bot ekran görüntüsü analizi yapmıyor; emülatörün dışa aktarılan bellek işaretçisi üzerinden Game Boy RAM adreslerini doğrudan okuyor.

-

```javascript
// savaşta olup olmadığını kontrol et
const BATTLE_ACTIVE = 0xD057;
const inBattle = readRam(BATTLE_ACTIVE) !== 0;

if (inBattle) {
  battleRoutine();   // FIGHT seç, en yüksek PP'li hamleyi al
} else {
  overworldWalk();   // duvar algılama ile rastgele yürüyüş
}
```

bot `multiplayerAllowed` koşuluna bağlı: yalnızca sunucu `ai_granted` mesajıyla sıra verdiğinde etkinleşiyor. bir insan kumandayı tutarken asla devreye girmiyor.

---

adım 7 — gömme ya da kendi altyapınızda çalıştırma

paylaşımlı arcade'i gömün (tek satır):

```html
<iframe src="https://yigitkonur.github.io/wasm-pokemon-red/"
        width="100%" height="640"
        style="border:none;border-radius:12px"
        allow="autoplay">
</iframe>
```

tüm sistemi kendi altyapınızda çalıştırın:

1. yigitkonur/wasm-pokemon-red (https://github.com/yigitkonur/wasm-pokemon-red) reposunu klonlayın ve ROM ile WASM'ı derleyin (`make`)
2. `server/server.js` 'yi Railway'e deploy edin — `railway.toml` ile tek tıklama
3. Upstash Redis örneği oluşturun, `UPSTASH_REDIS_REST_URL` ve `UPSTASH_REDIS_REST_TOKEN` değerlerini Railway ortam değişkenlerine ekleyin
4. `web/multiplayer.js` 'deki `WS_URL` değerini kendi Railway deployment'ınıza güncelleyin

-

5. `web/` klasörünü herhangi bir statik barındırma servisine gönderin (GitHub Pages, Netlify, Vercel, S3)

hobi ölçeğinde toplam bulut maliyeti: **$0**.

---

bundan sonra ne var

mantıksal bir sonraki adım daha fazla ROM. binjgb emülatörü herhangi bir Game Boy kartuşunu çalıştırıyor; ROM dosyasını değiştirip yeniden deploy etmek beş dakikalık bir işlem. sıra sistemi, Redis kalıcılığı ve AI botu tamamen ROM'dan bağımsız.

ilginç tasarım sorusu şu: her ROM ayrı bir sayfa ve ayrı bir sunucu olarak mı çalışmalı, yoksa sunucuyu her odanın bir ROM olduğu çok odalı bir modele mi genişletmeli? ikincisi daha zarif ve ziyaretçilerin oyunlar arasında geçiş yapmasına — küçük bir arcade gibi — izin verir.

Tetris, açık ilk ek. Sonra Link's Awakening. Sonunda ana sayfada bir lobi: bir oyun seç, kuyruğa gir, beş saniyen için oyna.

---

bağlantılar

- canlı arcade: yigitkonur.github.io/wasm-pokemon-red (https://yigitkonur.github.io/wasm-pokemon-red/)
- kaynak kod: github.com/yigitkonur/wasm-pokemon-red (https://github.com/yigitkonur/wasm-pokemon-red)
- pret/pokered: github.com/pret/pokered (https://github.com/pret/pokered)
- binjgb: github.com/nicknassar/binjgb (https://github.com/nicknassar/binjgb)

-

- PokeBot: github.com/bouletmarc/PokeBot (https://github.com/bouletmarc/PokeBot)

SOURCES

pret/pokered — Pokémon Kırmızı/Mavi disassembly (https://github.com/pret/pokered)

nicknassar/binjgb — Game Boy emülatörü (C) (https://github.com/nicknassar/binjgb)

Emscripten — C'den WebAssembly'ye araç zinciri (https://emscripten.org)

rgbds — Game Boy assembler ve linker (https://rgbds.gbdev.io)

bouletmarc/PokeBot — RAM okuyan Pokémon botu (https://github.com/bouletmarc/PokeBot)

Upstash Redis — sunucusuz Redis (https://upstash.com)

Railway — WebSocket sunucusunu deploy etmek için (https://railway.app)

Kaynak kod ve self-host kılavuzu (https://github.com/yigitkonur/wasm-pokemon-red)

-