Doodle Brawl

Team LUGNUTS: Connor Fair, Jon Rutan, Trevor Corcoran
AI and Machine Learning

Visit the site:

doodle.jfelix.space

faircw@vcu.edu, corcorantj@vcu.edu, rutanjf@vcu.edu

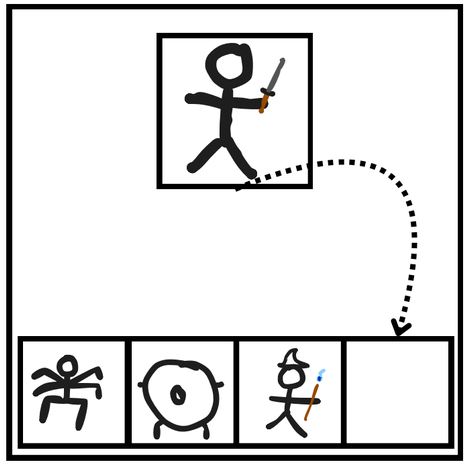
What is Doodle Brawl?

A website that brings your doodles to life in 1v1 prize fights

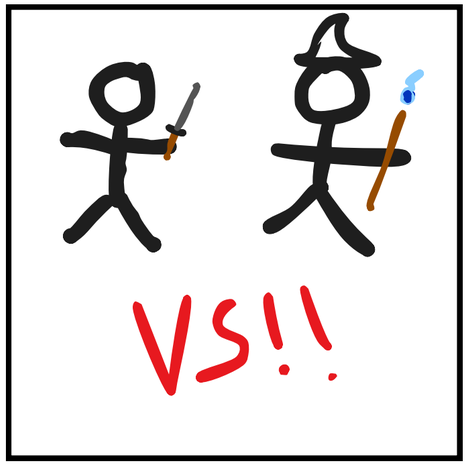
Doodle Brawl is a lighthearted combat sports webapp.
Draw a fighter to add them to the roster, then spectate the
(not-so) bloodsport.

Doodle Brawl uses the Gemini API for “billing” fighters and producing their matches all while providing color commentary on the action.

1. Create your fighter

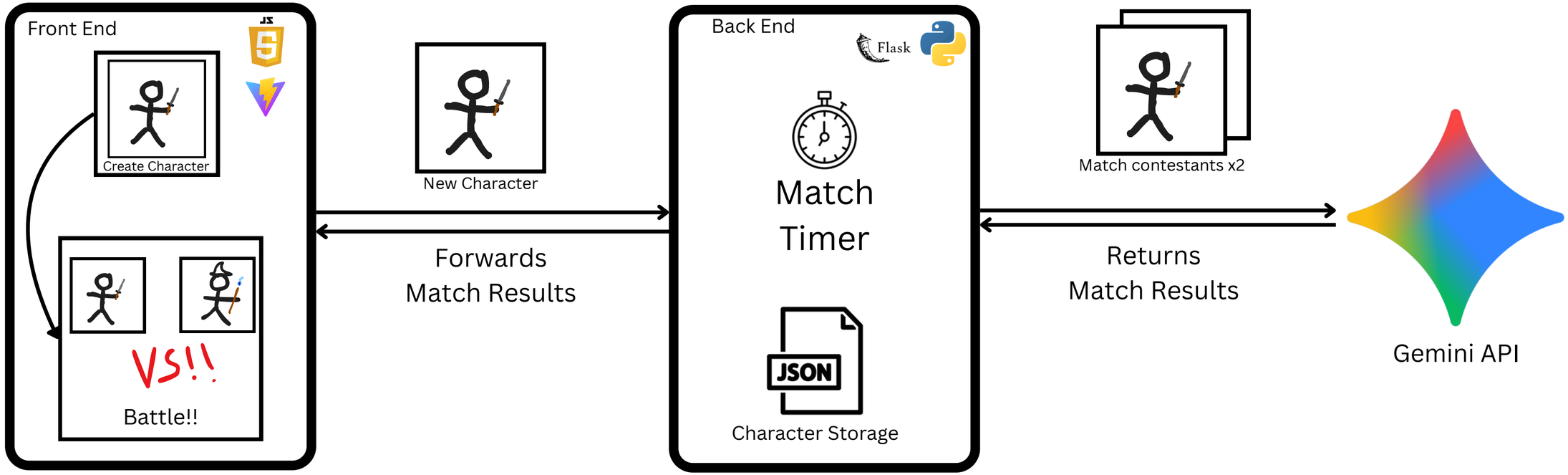


2. Submit them to the roster



3. Watch them **fight!**

How does it work?

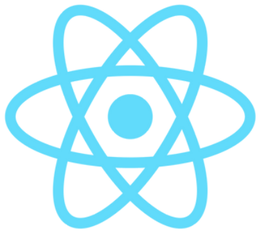


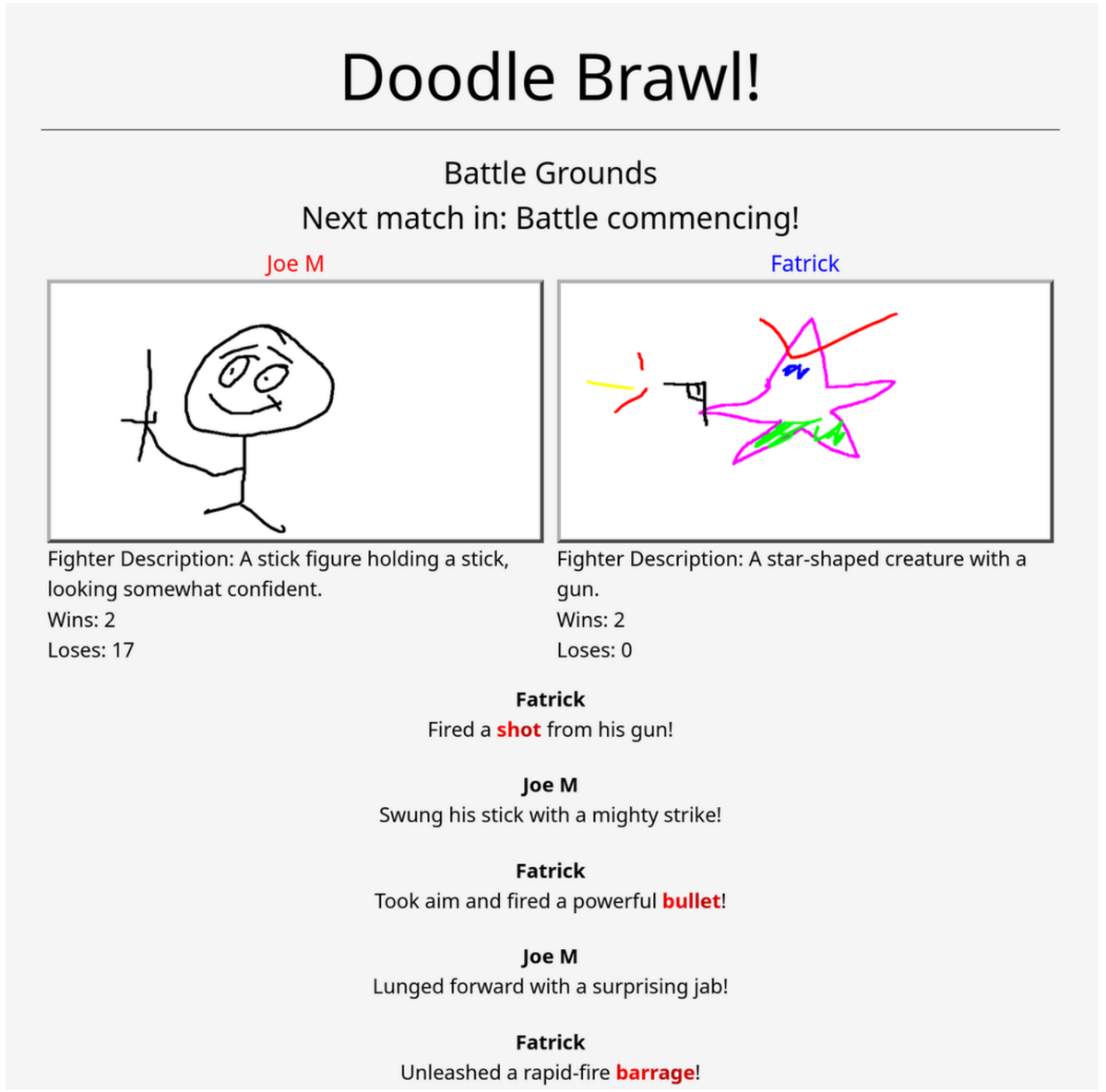
Frontend

Built with React & Vite

- App split in two sections: Battle grounds and Drawing canvas

```
c/BattleView.jsx  App.css                                                    X
121 21      );
122 20    }
123 19
124 18  return (
125 17    <div class='root'>
126 16      <h2>Next match in: {timer}</h2>
127 15      <div class='row'>
128 14
129 13        <div class='column'>
130 12          <p class='fighter-name fighter-1'>{battleState.fighters[0].name}</p>
131 11          <div class='fighter-img'>
E 132 10          {battleState && <ImageViewer base64={battleState.fighters[0].image_file} />}
133  9           </div>
134  8           <div class='stats'>
135  7             <p>Fighter Description: {battleState.fighters[0].description}</p>
136  6             <p>Wins: {battleState.fighters[0].wins}</p>
137  5             <p>Loses: {battleState.fighters[0].losses}</p>
138  4           </div>
139  3         </div>
140  2
141  1         <div class='column'>
142  0           <p class='fighter-name fighter-2'>{battleState.fighters[1].name}</p>
143  1           <div class='fighter-img'>
E 144  2          {battleState && <ImageViewer base64={battleState.fighters[1].image_file} />}
145  3           </div>
146  4           <div class='stats'>
147  5             <p>Fighter Description: {battleState.fighters[1].description}</p>
148  6             <p>Wins: {battleState.fighters[1].wins}</p>
149  7             <p>Loses: {battleState.fighters[1].losses}</p>
150  8           </div>
151  9         </div>
152 10
153 11      </div>
154 12      <div class='logs'>
155 13        <ul>
156 14          {logState.map((log, index) => (
157 15            <li class='one-log' key={index}>
158 16              <span class='log-name'>
159 17                {log.actor}
160 18              </span>
161 19              <div dangerouslySetInnerHTML={{ __html: log.description }} />
162 20            </li>
163 21          ))}
164 22        </ul>
165 23
166 24        <p class='summary'>{summaryState}</p>
167 25      </div>
168 26    </div>
169 27  );
170 28 }
NORMAL  main  2 1  BattleView.jsx  4.6k   eslint  javascriptreact  83%  142:1
"App.jsx" 74L, 2332B written
```





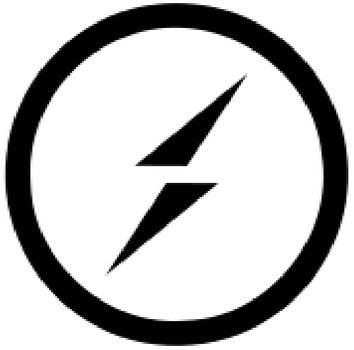
Frontend

Built with React & Vite

Frontend

Built with React & Vite

- Communicates with backend via **Socket.io**
  - Images encoded with base64
  - Lost connection is handled seamlessly

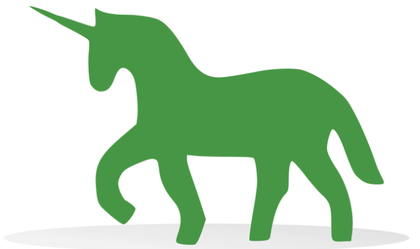


socket.io

```
c/BattleView.jsx  App.css                                                    X
      setTimer("Battle commencing!")
    }
    else {
      setTimer("Bookie is working on the next match...")
    }
    console.log(data);
  }

  // Listen for and load battles from backend
  useEffect(() => {
    // Register listeners
    socket.on('match_scheduled', handleSchedule);
    socket.on('match_result', handleResult);
    socket.on('timer_update', handleTimerUpdate);

    // Get initial fighter info from scheduled battle
    fetch('/card')
      .then(response => {
        if (!response.ok) throw new Error('Network response was not ok');
        return response.json();
      })
      .then(json => {
        console.log("Got Fresh Fighter Data");
        processFightData(json);
        setLoading(false);
      })
      .catch(err => {
        setError(err.message);
        setLoading(false);
      });


    // De-register listeners for cleanup
    return () => {
      socket.off('match_scheduled', handleSchedule);
      socket.off('match_result', handleResult);
      socket.off('timer_update', handleTimerUpdate);
    }
  }, []);


  if (loading) return <div class='net-loading'>Loading...</div>;
  if (error) return <div class='net-error'>Error: {error}</div>;
  if (!battleState || !battleState.fighters || battleState.fighters.length < 2) { return (
        <div className='root waiting-screen'>
            <h1>Waiting for Next Match...</h1>
            {timer && <h2>Next Match in: {timer}s</h2>}
        </div>
      );
    }

NORMAL  main  ⊗ 2 △ 1  BattleView.jsx  4.6k        eslint ✓  javascriptreact  53%  91:1
"App.jsx" 74L, 2332B written
```

Backend

Built with Flask, Gemini API, and Python



- Server uses Flask sockets for client/server updating and endpoints for data retrieval
- Uses a highly configured API call to Google's **genai** library.
- Website is run using gunicorn to create a production-level flask environment

```
#################################
#          GEMINI API           #
#################################
client = genai.Client(api_key=API_KEY)
SYSTEM_PROMPT = """
You are the "Doodle Brawl" Game Engine. Your goal is to simulate a tur
You'll also need to act as the color commentator of the matches, givin

### PHASE 1: STAT GENERATION
Analyze the provided images for both fighters.
IF a fighter has `fight_count: 0` (stats are empty/null), you MUST gen
1.  **HP (50-200):** Low for small/fragile, High for big/armored. (Avg
2.  **AGILITY (1-10):** Low for heavy/clunky, High for sleek/athletic.
3.  **POWER (1-20):** Low for weak, High for dangerous/weapon-wielding
4.  **DESCRIPTION:** A one-sentence combat-sport introduction (e.g. "T

### PHASE 2: COMBAT SIMULATION
Simulate the fight turn-by-turn until one reaches 0 HP. A "favorabilit
* **Agility Rule:** If Agility > 6, that fighter has a 20% chance to p
* **Move Types:**
    * STANDARD:     `ATTACK` : Standard hit (Power +/- variance).
    * STANDARD:     `RECOVER`: Recover HP (HP +/- variance).
    * IF POWER>=15: `SLAM`   : Large hit (Power(+5) +/- variance).
    * IF AGILITY>=7:`DIVE`   : Dive from off ropes (Agility + Power +/

### PHASE 3: MATCH SUMMARY AND WINNER
You'll end off by declaring the winner, and providing an exciting, but
```

GEMINI API PROMPT

```
#################################
#        SERVER HANDLERS        #
#################################

@app.route('/card')
def return_current_card():
    global NEXT_MATCH
    current_match = NEXT_MATCH
    if current_match is None:
        return jsonify({
            'fighters': [],
            'starts_in': 0,
            'status': 'waiting'
        })
    try:
        fighters_data = [c.to_dict() for c in current_match]

        return jsonify({
            'fighters': fighters_data,
            'starts_in': BATTLE_TIMER,
            'status': 'scheduled'
        })
    except Exception as e:
        print(f"!-- ERROR SERVING CARD: {e} --!")
        return jsonify({'error': str(e)}), 500
```

SERVER HANDLERS

```
def run_scheduled_battle():
    global NEXT_MATCH
    if not NEXT_MATCH:
        return

    p1, p2 = NEXT_MATCH
    print(f"!-- RUNNING BATTLE: {p1.name} vs {p2.name} --!")
    favorability = random.randint(1,100)                    #add some randomness to outcome
    #battle information to be sent to gemini API
    request_content = [
        f"FAVORABILITY: {favorability}",
        f"""
        FIGHTER 1:
        ID: {p1.id}
        Name: {p1.name}
        Current Stats: {p1.stats} (If empty, generate them based on attached image)
        Fight Count: {p1.wins + p1.losses}
        """,
        get_image_part_from_base64(p1.image_file), #fighter 1 drawing

        f"""
        FIGHTER 2:
        ID: {p2.id}
        Name: {p2.name}
        Current Stats: {p2.stats} (If empty, generate them based on attached image)
        Fight Count: {p2.wins + p2.losses}
        """,
        get_image_part_from_base64(p2.image_file)  #fighter 2 drawing
    ]
```

FIGHT GENERATOR

Live Demo

Visit the site:

doodle.jfelix.space