FastReaction Game Documentation

Contents

Quick Start

Running the Game

```
# Install dependencies
pip install -r requirements.txt

# Run the application
python Fast_Reaction_V2.1.5.py
```

Environment Variables

```
export FAST_REACTION_API_BASE_URL="https://api.example.com/"
export FAST_REACTION_TIMER_VALUE="30000"  # milliseconds
export FAST_REACTION_SERIAL_PORT="/dev/ttyUSB0"
export FAST_REACTION_SERIAL_ENABLED="true"
```

File Structure

```
FastReaction/
├── Fast_Reaction_V2.1.5.py          # Main application
├── config.py                         # Configuration
├── api/
│   └── game_api.py                   # API client
├── utils/
│   ├── logger.py                     # Logging
│   └── audio_service.py              # Audio system
├── *_backend.py                      # QML backends (4 files)
├── *.ui.qml                          # QML UI screens (5 files)
├── Assets/                           # Fonts, images, audio
```

```
├── teams/                          # JSON backups
├── logs/                           # Log files
└── docs/                           # Documentation
```

Overview

FastReaction is a PyQt5-based reaction-time game that integrates with serial hardware (optional), MQTT messaging, and a cloud API. Players react to visual stimuli by pressing buttons or triggering serial inputs, with scores based on reaction speed and accuracy.

Key Features

- Real-time reaction tracking via serial or button inputs
- Score tracking with correct/wrong/miss counters
- Cloud API for team management and leaderboards
- MQTT control for game state
- QML-based UI with 4K support
- Audio service for sound effects
- Automatic JSON backup of team sessions

Game Flow

1. Authentication - Connect to cloud API
2. Polling Initialization - Wait for game admin to create session
3. Polling Start - Wait for admin to start the game
4. Gameplay - Real-time reaction tracking and scoring
5. Score Submission - Submit final scores to API
6. Leaderboard - Display top teams

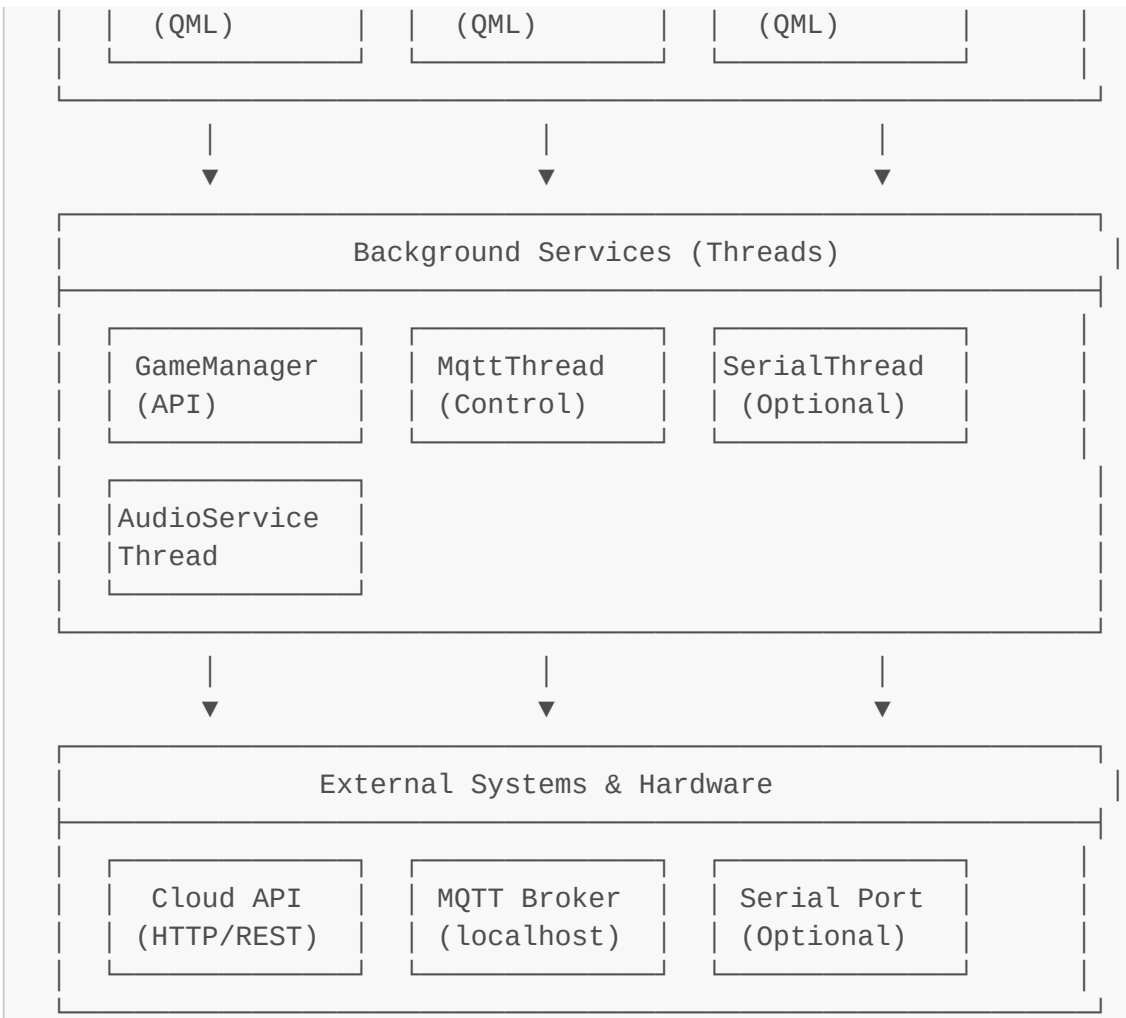
Scoring System

- Correct reactions: +1 point (with correct sound)
- Wrong reactions: -1 point (with wrong sound)
- Missed reactions: -1 point (with miss sound)
- Final score: Sum of all reactions within time limit
- Serial input supported for automated scoring

Architecture

System Overview

```
┌──────────────────────────────────────────────────────────┐
│                    MainApp (Main Window)                 │
├──────────────────────────────────────────────────────────┤
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐    │
│  │ Home Screen  │  │ Active Screen│  │ Final Screen │    │
```



Threads

- Main Thread - UI rendering, user interaction
- GameManager Thread - API communication, game flow control
- MQTT Thread - Message broker for game control
- Serial Thread - Serial communication (optional)
- AudioService Thread - Background audio playback

Data Flow

Game Initialization

```
Admin creates game → API polls (status=initiated) → Display team name →
Wait for start
```

Gameplay

```
Player reacts → Input detected → Update score → Play audio feedback →
Update UI
```

Score Submission

```
Timer ends → Save to CSV → Save to JSON → Submit to API → Show leaderboard
```

Main Components

1. Main Application (Fast_Reaction_V2.1.5.py)

Main game application that orchestrates all components.

Classes:

- `MainApp` - Primary application window and coordinator
- `GameManager` - Thread that manages API communication and game flow
- `Active_screen` - Active gameplay screen with real-time updates
- `Final_Screen` - Score display and submission screen
- `SimpleSerialThread` - Threaded serial communication (optional)
- `MqttThread` - MQTT message broker for game control
- `Home_screen` - Home screen with leaderboard display
- `TeamMember_screen` - Team entry screen (uses QML backend)

Global variables:

- `scored` - Current game score
- `correct_count` - Number of correct reactions
- `miss_count` - Number of missed reactions
- `wrong_count` - Number of wrong reactions
- `RemainingTime` - Countdown timer value (seconds)
- `teamName` - Current team name
- `gameStarted` - Boolean indicating if game is active
- `gamefinished` - Boolean indicating if game has ended
- `serial_scoring_active` - Boolean for serial-based scoring

2. QML UI Screens

All QML files support 4K resolution (3840x2160) with responsive scaling.

QML File	Backend	Purpose
Active_screen_FastReaction.ui.qml	FastReactionBackend	Real-time score, timer, team name
Teamname_screen_new.ui.qml	TeamNameBackend	Bilingual team entry, 4 player avatars
Teamscore_screen.ui.qml	TeamScoreBackend	Final score display

QML File	Backend	Purpose
Leaderboard_screen_enhanced.qml	EnhancedLeaderboardBackend	Top 5 teams, podium
CircularTimer_enhanced.qml	CircularTimerBackend	Circular countdown timer

3. Backend Modules

Each QML screen has a Python backend using PyQt5 signals:

- `FastReactionBackend` - Active game screen updates (score, timer, team name)
- `EnhancedLeaderboardBackend` - Leaderboard data management
- `TeamScoreBackend` - Score display
- `TeamNameBackend` - Team entry management

See BACKEND_MODULES.md for detailed backend documentation.

4. API Client (api/game_api.py)

Handles all HTTP REST API communication.

Key methods:

- `authenticate()` - Login and obtain bearer token
- `poll_game_initialization()` - Wait for game creation
- `poll_game_start_continuous()` - Wait for game start
- `submit_final_scores()` - Submit team scores
- `get_leaderboard()` - Fetch top teams

See API_MODULE.md for detailed API documentation.

5. Configuration (config.py)

Centralized settings with environment variable support.

Classes:

- `APIConfig` - API endpoints, credentials, timeouts
- `GameConfig` - Timer values, scoring parameters
- `SerialConfig` - Serial port settings (optional)
- `MQTTConfig` - MQTT broker configuration
- `UIConfig` - UI and display settings

See CONFIG_MODULE.md for detailed configuration documentation.

6. Serial Thread (SimpleSerialThread)

Optional serial communication for hardware integration.

Features:

- Automatic reconnection on disconnect

- Simple text command parsing
- Thread-safe data handling
- Connection status monitoring
- Error handling and recovery

Serial data format:

- Numeric values: "5" (adds 5 to score)
- "hit" or "point": adds 1 to score
- "win": triggers game end
- Custom commands can be added

7. Utilities

Logger (utils/logger.py)

- Automatic timestamped log files
- Console and file logging
- Configurable log levels
- Location: `logs/fast_reaction_YYYYMMDD_HHMMSS.log`

Audio Service (utils/audio_service.py)

- Threaded audio playback using pygame.mixer
- Pre-loaded sounds for zero-latency
- Supports: continuous, inactive_game, active_game, correct, wrong, miss
- Thread-safe audio control

API Reference

Authentication

```
api = GameAPI()
if api.authenticate():
    print("Authenticated")
```

Game Polling

```
# Wait for game creation
game_data = api.poll_game_initialization()
team_name = game_data['name']
game_id = game_data['id']

# Wait for game start
game_data = api.poll_game_start_continuous(game_id, ...)
if game_data.get('status') == 'playing':
    start_game()
```

Score Submission

```
scores = [
    {"userID": "user1", "nodeID": 1, "score": 50},
    {"userID": "user2", "nodeID": 2, "score": 50}
]
api.submit_final_scores(game_id, scores)
```

Leaderboard

```
top_teams, last_played = api.get_leaderboard()
for rank, (name, score, weighted) in enumerate(top_teams, 1):
    print(f"{rank}. {name}: {score}")
```

Configuration

API Settings

```
# Production (config.py)
base_url = "https://prod-eaa25-api.azurewebsites.net/"
email = "sas@uxe.ai"
password = "6u52fkKJl"
game_id = "fb21ca1b-bac0-4a3d-914d-258ea5b2f8ba"
game_name = "Fast Reaction"
```

Game Settings

```
timer_value = 30000  # 30 seconds in milliseconds
final_screen_timer = 10000  # 10 seconds
```

Serial Settings (Optional)

```
enabled = False  # Set to True to enable serial
port = "/dev/ttyUSB0"  # Linux
# port = "COM3"  # Windows
baudrate = 9600
timeout = 1
```

Environment Override

```
export FAST_REACTION_API_BASE_URL="https://dev-api.example.com/"
export FAST_REACTION_TIMER_VALUE="60000"
export FAST_REACTION_SERIAL_ENABLED="true"
export FAST_REACTION_SERIAL_PORT="/dev/ttyUSB1"
```

Deployment

See DEPLOYMENT.md for complete deployment guide including:

- Hardware requirements
- Software installation
- Systemd service setup
- Configuration
- Testing procedures

Quick Deployment

```
# Install dependencies
sudo apt update
sudo apt install python3 python3-pip python3-pyqt5 qml-module-qtquick*

# Install Python packages
pip3 install -r requirements.txt

# Install systemd service
cd services
sudo ./install-service.sh

# Start service
sudo systemctl start fastreaction
```

Troubleshooting

Serial Communication Fails

- Check `/dev/ttyUSB0` availability: `ls -l /dev/ttyUSB*`
- Verify baudrate (9600)
- Add user to dialout group: `sudo usermod -a -G dialout $USER`
- Set `FAST_REACTION_SERIAL_ENABLED=false` to disable serial

API Connection Fails

- Verify internet connectivity

- Check API credentials in config.py
- Review logs: `tail -f logs/fast_reaction_*.log`

MQTT Not Working

- Ensure MQTT broker is running: `sudo systemctl status mosquitto`
- Check localhost connection
- Verify firewall settings

Audio Not Playing

- Check audio files exist in `Assets/mp3/`
- Verify pygame.mixer initialization
- Test system audio: `aplay /usr/share/sounds/alsa/Front_Center.wav`

Service Not Starting

```
# Check service status
sudo systemctl status fastreaction

# View logs
sudo journalctl -u fastreaction -n 50

# Check permissions
ls -la /home/eaa25-game1/FastReaction/

# Restart service
sudo systemctl restart fastreaction
```

JSON Backup System

The application automatically saves team session data to `teams/` directory before API submission.

File Format

Filename: `{timestamp}_{team_name}.json`

```
{
  "team_name": "Team Alpha",
  "timestamp": "2025-11-13 10:29:32",
  "total_score": 100,
  "backend_payload": {
    "gameResultID": "uuid-game-id",
    "individualScore": [
      {"userID": "user1", "nodeID": 1, "score": 50},
      {"userID": "user2", "nodeID": 2, "score": 50}
    ]
```

```
  }
}
```

Recovery

If API submission fails:

1. Find JSON file in `teams/` directory
2. Extract `backend_payload`
3. Use `external_csv_submitter.py` to resubmit

CSV Logging

FastReaction.csv

Main game results log

```
team_name, final_score, timestamp
Team Alpha, 100, 2025-01-15 14:30:00
```

FastReaction_Individual_Players_Log.csv

Per-player score tracking

```
timestamp, game_result_id, user_id, node_id, individual_score,
submission_success, status
```

FastReaction_Pre_Submission_Backup.csv

Pre-API submission backup

```
timestamp, game_result_id, total_players, total_score, player_ids,
individual_scores_json, status
```

Development Notes

Thread Safety

- All Qt signals are thread-safe (Qt's meta-object system)
- Dedicated threads for serial, API, MQTT, and audio
- Proper cleanup methods in all screens

QML Integration

Python signals to QML:

```
# Python
self.scoreChanged.emit(str(new_score))

# QML
Connections {
    target: backend
    onScoreChanged: { score_value.text = scoreValue }
}
```

Testing Checklist

- [ ] Serial port available (if enabled)
- [ ] MQTT broker running
- [ ] API endpoint accessible
- [ ] Audio files present
- [ ] QML files found
- [ ] Permissions correct

Version History

V2.1.5 (Current)

- Integrated new GUI with QML
- Enhanced serial thread with auto-reconnect
- Improved audio service
- Better error handling and cleanup
- Fixed resource cleanup on close

Dependencies

Python Packages

- PyQt5 - UI framework and widgets
- pygame - Audio playback (via audio_service)
- paho-mqtt - MQTT messaging
- requests - HTTP API communication
- pyserial - Serial communication (optional)
- numpy - Numerical operations

System Requirements

- Python 3.8+
- Qt5 and QML
- MQTT broker (mosquitto)

- Serial port access (optional)
- Network connectivity

Additional Documentation

- API_MODULE.md - Detailed API client reference
- CONFIG_MODULE.md - Configuration guide
- BACKEND_MODULES.md - QML backend reference
- DEPLOYMENT.md - Deployment guide

Last Updated: November 13, 2025
Application Version: V2.1.5