Foundry VTT QR Token Tracker Setup Guide

This system allows you to use physical QR code tokens on a table surface to control digital tokens in Foundry VTT, creating a hybrid physical/digital tabletop gaming experience.

System Overview

- **Physical Layer**: QR codes on your gaming table tracked by Raspberry Pi camera
- **Digital Layer**: Foundry VTT tokens that mirror the physical token positions
- **Bridge**: Python script that connects the physical tracking to Foundry
- **Network**: Works across different machines - Pi tracks, Foundry runs elsewhere

Hardware Requirements

- Raspberry Pi 4 (recommended) with camera module
- Camera mount positioned above your gaming table
- QR code tokens (printed or generated)
- Gaming surface with clear boundaries
- **Network**: Both Pi and Foundry host on same network (WiFi/Ethernet)

Network Configuration

For Remote Foundry Setup:

- **Raspberry Pi**: Runs the tracking script
- **Foundry Host**: Separate machine running Foundry VTT
- **Network Requirements**: Both machines must be able to communicate
- **Ports**: WebSocket port (default 30001) must be accessible
- **Firewall**: Ensure port 30001 is open on both machines

Required Files

Raspberry Pi Files:

1. `foundry_qr_tracker.py` - Main tracking script
2. `network_test.py` - Network connectivity test script

Foundry VTT Module Files:

Create directory: `Data/modules/qr-tracker/`

1. **`module.json`** - Module manifest
2. **`qr-tracker.js`** - Module JavaScript code

Optional Helper Files:

1. **Corner markers** - QR codes with `CORNER_TL`, `CORNER_TR`, `CORNER_BL`, `CORNER_BR`
2. **Player tokens** - QR codes with player identifiers

Software Installation

1. Raspberry Pi Setup

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install required system packages
sudo apt install python3-opencv python3-numpy python3-pip

# Install Python packages
pip3 install picamera2 pyzbar websockets requests aiohttp opencv-python numpy
```

2. Foundry VTT Module Installation

1. Create folder: `Data/modules/qr-tracker/`
2. Place these files in the folder:
    - `module.json` (manifest file)
    - `qr-tracker.js` (main module code)
3. Restart Foundry VTT
4. Enable the "QR Token Tracker" module in your world

3. QR Code Generation

Create QR codes for your players/tokens. Each should contain a unique identifier:

- `PLAYER_1`, `PLAYER_2`, etc.
- `GOBLIN_1`, `WIZARD_BOSS`, etc.
- Any unique string identifier

Special Corner Markers (optional, for automatic calibration):

- `CORNER_TL` (Top-Left)
- `CORNER_TR` (Top-Right)
- `CORNER_BL` (Bottom-Left)
- `CORNER_BR` (Bottom-Right)

Setup Process

1. Physical Setup

1. Mount your camera above the gaming table
2. Ensure good lighting and clear view of the entire play surface
3. Place corner markers if using automatic calibration

2. Camera Calibration

Run calibration mode
python3 foundry_qr_tracker.py --foundry-url "http://localhost:30000" --scene-id "your-scene-id" --calibrate-only

Calibration Options:

- **Automatic**: Place corner marker QR codes and the system will detect them
- **Manual**: Click the four corners of your play area when prompted

3. Foundry Configuration

1. Open your Foundry world
2. Create or open the scene you want to use
3. Note the Scene ID (visible in the URL or scene configuration)
4. Configure module settings in Foundry:
    - **QR Tracker Host**: IP address of your Raspberry Pi (e.g., `192.168.1.100`)
    - **WebSocket Port**: Default 30001 (must match Python script)
    - **Auto-create tokens**: Enabled (recommended)
    - **Default token image path**: Set your preferred token image

4. Network Setup

Find Your Raspberry Pi's IP Address:

On the Raspberry Pi
hostname -I

```
# Example output: 192.168.1.100
```

Test Network Connectivity (Recommended):

```
# Run the network test script first
python3 network_test.py --foundry-host 192.168.1.50 --foundry-port 30000

# This will test:
# - Basic connectivity between machines
# - Foundry HTTP API accessibility
# - WebSocket port availability
```

Manual Network Tests:

```
# From Foundry host, test if Pi is reachable
ping 192.168.1.100

# Test if WebSocket port is open (after starting tracker)
telnet 192.168.1.100 30001
```

Firewall Configuration:

```
# On Raspberry Pi, open WebSocket port
sudo ufw allow 30001

# On Foundry host (if needed)
sudo ufw allow from 192.168.1.100
```

5. Start Tracking

For Remote Foundry (most common):

```
python3 foundry_qr_tracker.py \
  --foundry-url "http://192.168.1.50:30000" \
  --scene-id "abc123def456" \
  --surface-width 1000 \
  --surface-height 1000
```

For Local Foundry:

```
python3 foundry_qr_tracker.py \
```

```
--foundry-url "http://localhost:30000" \
--scene-id "abc123def456" \
--surface-width 1000 \
--surface-height 1000
```

Command Line Options:

- `--foundry-url`: Your Foundry server URL (IP:port)
- `--scene-id`: The scene to update (get from Foundry URL)
- `--surface-width/height`: Coordinate system size
- `--websocket-port`: WebSocket port (default: 30001)
- `--no-display`: Run without camera preview window

6. Verify Connection

1. **In Foundry**: Check the QR Tracker status button in scene controls
2. **Python Console**: Look for "Connected to Foundry WebSocket" message
3. **Test Connection**: Use the "Test Connection" button in Foundry status dialog

Usage During Games

Physical Tokens

- Place QR tokens on your surface within the calibrated area
- Move tokens around - Foundry tokens will follow in real-time
- Remove tokens from surface - they'll disappear from Foundry after 3 seconds

Foundry Interface

- Tokens auto-create when new QR codes are detected
- Green indicator = token connected to QR tracker
- Red indicator = token exists but no QR connection
- Use the QR status button in scene controls for diagnostics

Troubleshooting

Camera Issues:

```
# Test camera
python3 -c "from picamera2 import Picamera2; cam = Picamera2(); cam.start(); print('Camera working')"
```

Network/Connection Issues:

```
# Check if Foundry host is reachable
ping [foundry-host-ip]

# Test WebSocket port connectivity
telnet [foundry-host-ip] 30001

# Check if Python script can reach Foundry HTTP API
curl http://[foundry-host-ip]:30000/api/scenes
```

Common Network Problems:

- **"Connection refused"**: Check firewall settings and port availability
- **"Host unreachable"**: Verify both machines are on same network
- **"WebSocket disconnected"**: Check if Foundry module is enabled and configured
- **"No route to host"**: Check IP addresses and network configuration

Foundry Module Issues:

- Check Foundry F12 console for JavaScript errors
- Verify module is enabled in world settings
- Ensure QR Tracker Host setting points to correct Raspberry Pi IP
- Use "Test Connection" button in status dialog

Tracking Issues:

- Ensure good lighting
- Check QR codes are not damaged or too small
- Recalibrate surface if tracking seems offset
- Verify camera view includes entire play area

Quick Network Test Script

Create this script on your Raspberry Pi to test connectivity:

```
#!/bin/bash
# save as test_network.sh on Raspberry Pi

FOUNDRY_HOST="192.168.1.50"  # Replace with your Foundry host IP
FOUNDRY_PORT="30000"
WEBSOCKET_PORT="30001"

echo "Testing network connectivity to Foundry VTT..."
```

```
echo "Foundry Host: $FOUNDRY_HOST"
echo "Foundry Port: $FOUNDRY_PORT"
echo "WebSocket Port: $WEBSOCKET_PORT"
echo ""

echo "1. Testing basic connectivity..."
if ping -c 3 $FOUNDRY_HOST > /dev/null 2>&1; then
    echo "✓ Host is reachable"
else
    echo "✗ Host is NOT reachable - check network configuration"
    exit 1
fi

echo "2. Testing Foundry HTTP port..."
if curl -s --connect-timeout 5 http://$FOUNDRY_HOST:$FOUNDRY_PORT > /dev/null; then
    echo "✓ Foundry HTTP port is accessible"
else
    echo "✗ Foundry HTTP port is NOT accessible"
fi

echo "3. Testing WebSocket port..."
if timeout 5 bash -c "</dev/tcp/$FOUNDRY_HOST/$WEBSOCKET_PORT"; then
    echo "✓ WebSocket port is open"
else
    echo "✗ WebSocket port is closed or filtered"
fi

echo ""
echo "Network test complete!"
```

Example Configuration

Scenario: Foundry runs on a desktop PC, Raspberry Pi with camera tracks tokens

Network Setup:

- Desktop PC (Foundry): `192.168.1.50`
- Raspberry Pi (Tracker): `192.168.1.100`
- Router: `192.168.1.1`

Foundry Module Settings:

- QR Tracker Host: `192.168.1.100`

- WebSocket Port: `30001`
- Auto-create tokens: `true`

Python Command:

```
python3 foundry_qr_tracker.py \
  --foundry-url "http://192.168.1.50:30000" \
  --scene-id "your-scene-id-from-foundry" \
  --websocket-port 30001
```

Custom Token Creation

Modify the module to use specific token images:

```
// In qr-tracker.js, customize createTokenForQR function
const tokenData = {
    name: `Player_${qr_id}`,
    img: getTokenImageForQR(qr_id), // Custom function
    x: x,
    y: y,
    // ... other properties
};
```

Integration with Other Modules

The tracker can work alongside:

- Token health/status modules
- Combat trackers
- Dice rolling systems
- Chat integrations

Performance Tuning

```
# Adjust update frequency
--update-interval 0.05  # 20 FPS updates

# Change detection sensitivity
--confidence-threshold 0.8  # Higher = more reliable detection
```

API Reference

Python Script Events

The tracker emits these events to Foundry:

```
{
  "type": "token_update",
  "qr_id": "PLAYER_1",
  "x": 150,
  "y": 300,
  "confidence": 0.95,
  "scene_id": "abc123"
}
```

Foundry Module API

Access via browser console:

```
// Get connection status
game.modules.get('qr-tracker').api.getStatus()

// Force reconnection
game.modules.get('qr-tracker').api.reconnect()

// Get tracked tokens
game.modules.get('qr-tracker').api.getTrackedTokens()
```

Best Practices

QR Code Design

- Use high contrast (black on white)
- Minimum 2cm x 2cm size
- Laminate for durability
- Include visual identifier (player name/picture)

Table Setup

- Use consistent lighting
- Avoid reflective surfaces
- Keep camera lens clean
- Test different angles for best tracking

Gaming Workflow

1. Start Python tracker before session
2. Have players place their tokens
3. Verify all tokens appear in Foundry
4. Begin gaming with hybrid experience!

Support and Development

Common Issues

- **Tokens jumping**: Recalibrate surface
- **Delayed updates**: Check network connection
- **Missing tokens**: Verify QR codes are readable

Contributing

This system is designed to be extensible. Areas for improvement:

- Multi-camera support
- 3D token tracking
- Integration with more VTT platforms
- Mobile app for token management

License

This project is open source. Feel free to modify and distribute according to the license terms.