Autonomous Goal-Driven Agent - Quick Start Guide

🚀 Get Started in 5 Minutes

1. Run the Example (Recommended First Step)

```
cd /home/ubuntu/powerhouse_b2b_platform/backend
python examples/autonomous_agent_example.py
```

What you'll see:
- Agent initialization
- Real-time metric recording
- Autonomous goal creation
- Goal execution tracking
- Learning insights
- Comprehensive reporting

Duration: ~2 minutes

2. Start the Server

```
cd /home/ubuntu/powerhouse_b2b_platform/backend
python start_with_autonomous_agent.py
```

Server runs on: `http://localhost:5003`

Features:
- Autonomous agent active and running
- RESTful API for monitoring
- Real-time goal tracking
- Predictive analytics

3. Monitor via API

Get Agent Status:

```
curl http://localhost:5003/api/autonomous/status
```

View Active Goals:

```
curl http://localhost:5003/api/autonomous/goals
```

See Predictions:

```
curl http://localhost:5003/api/autonomous/predictions?horizon_hours=24
```

Comprehensive Report:

```
curl http://localhost:5003/api/autonomous/report
```

4. Basic Usage in Code

```python
from core.goal_driven_agent import GoalDrivenAgent

# Create agent
agent = GoalDrivenAgent(
    agent_config={"autonomous_mode": True}
)

# Start autonomous behavior
agent.start()

# Record metrics (agent uses these for predictions)
agent.record_metric("cpu_usage", 75.0)
agent.record_metric("memory_usage", 60.0)
agent.record_metric("latency", 150.0)

# Record events (for pattern recognition)
agent.record_event("api_request", metadata={"endpoint": "/users"})

# Check status
status = agent.get_agent_status()
print(f"Active Goals: {status['active_goals']}")

# View goals
overview = agent.get_goal_overview()
for goal in overview['goals']:
    print(f"{goal['description']}: {goal['progress']:.1%}")

# Stop when done
agent.stop()
```

5. Register Custom Actions

```
# Define your custom action
def optimize_database(params, goal_id):
    """Custom database optimization action."""
    # Your optimization logic here
    print(f"Optimizing database for goal {goal_id}")

    # Perform optimization
    # ...

    # Return result
    return {
        "success": True,
        "impact": {
            "query_time": -0.20,  # 20% improvement
            "throughput": 0.15    # 15% increase
        }
    }

# Register with agent
agent.register_action_handler("optimize_database", optimize_database)

# Now when a goal includes "optimize_database" action,
# your custom handler will be called automatically
```

🎯 What Happens Autonomously

Without Any Commands, the Agent:

1. **Monitors** - Continuously tracks system metrics
2. **Predicts** - Forecasts future states (CPU, memory, latency, etc.)
3. **Detects** - Identifies patterns in historical data
4. **Decides** - Sets proactive goals based on predictions:
   - "Prevent CPU bottleneck in 6 hours"
   - "Optimize memory usage to avoid threshold"
   - "Adapt to daily traffic spike pattern"
5. **Plans** - Creates detailed execution plans
6. **Executes** - Autonomously runs actions to achieve goals
7. **Learns** - Records outcomes and improves strategies
8. **Adapts** - Adjusts approach based on what works

Example Autonomous Cycle

```
12:00 PM - Agent predicts CPU will reach 95% at 6:00 PM
12:01 PM - Automatically creates goal: "Prevent CPU bottleneck"
12:01 PM - Plans execution: analyze → optimize → monitor
12:02 PM - Executes: analyzes CPU-intensive processes
12:05 PM - Executes: implements optimization
12:10 PM - Measures: CPU usage reduced to 70%
12:10 PM - Goal achieved! Records success
12:10 PM - Learns: "CPU optimization" action works well

(Continues autonomously 24/7)
```

📊 Key Metrics to Monitor

Agent Health

- `uptime_seconds` - How long agent has been running
- `goals_created` - Total goals set autonomously
- `goals_achieved` - Goals successfully completed
- `active_goals` - Currently pursuing

Execution Performance

- `success_rate` - Percentage of successful executions
- `average_execution_time` - Time to complete actions
- `total_actions_executed` - Actions taken

Learning Progress

- `action_success_rates` - Success rate per action type
- `best_strategies` - Optimal strategies per goal type
- `common_failures` - Actions that frequently fail

🔧 Configuration Tips

For Development/Testing

```
config = {
    "autonomous_mode": True,
    "goal_sync_interval_seconds": 30,      # Check often
    "analysis_interval_minutes": 15        # Analyze frequently
}
```

For Production

```
config = {
    "autonomous_mode": True,
    "goal_sync_interval_seconds": 60,      # Balanced
    "analysis_interval_minutes": 60        # Standard interval
}
```

Conservative (Start Here)

```
config = {
    "autonomous_mode": False,              # Manual mode first
    "goal_sync_interval_seconds": 120,     # Less frequent
    "analysis_interval_minutes": 120       # Less frequent
}
```

📝 Common Patterns

Pattern 1: Monitor and React

```
agent.start()

while True:
    status = agent.get_agent_status()
    if status['active_goals'] > 5:
        print("⚠️  Many active goals, system busy")
    time.sleep(60)
```

Pattern 2: Manual Trigger

```
# Let agent run normally, but force analysis when needed
agent.start()

# When you want immediate analysis
result = agent.force_analysis()
print(f"Analysis complete: {result['timestamp']}")
```

Pattern 3: Learning Review

```
# Periodically review what the agent has learned
insights = agent.executor.get_learning_insights()

print("Action Success Rates:")
for action, data in insights['action_success_rates'].items():
    print(f"  {action}: {data['rate']:.1%}")
```

Pattern 4: Goal Monitoring

```
# Track specific goals
overview = agent.get_goal_overview()

critical_goals = [
    g for g in overview['goals']
    if g['priority'] == 'critical'
]

print(f"Critical Goals: {len(critical_goals)}")
for goal in critical_goals:
    print(f"  - {goal['description']}: {goal['progress']:.1%}")
```

🧪 Testing

Run Unit Tests

```
cd /home/ubuntu/powerhouse_b2b_platform/backend
python -m pytest tests/test_autonomous_agent.py -v
```

Test Specific Component

```
python -m pytest tests/test_autonomous_agent.py::TestGoalDrivenAgent -v
```

📚 Documentation

- **Full Guide**: `AUTONOMOUS_BEHAVIOR_README.md`
- **Implementation Details**: `AUTONOMOUS_GOAL_DRIVEN_IMPLEMENTATION_SUMMARY.md`
- **Code Examples**: `examples/autonomous_agent_example.py`

🐛 Troubleshooting

No Goals Being Created?

```python
# Check if metrics are being recorded
status = agent.get_agent_status()
print(f"Predictions made: {status['statistics']['total_predictions']}")

# Force analysis
agent.force_analysis()
```

Goals Not Executing?

```python
# Check executor is running
executor_stats = agent.executor.get_statistics()
print(f"Active executions: {executor_stats['active_executions']}")
print(f"Scheduled plans: {executor_stats['scheduled_plans']}")
```

Want More Verbose Output?

```python
import logging
logging.basicConfig(level=logging.INFO)
# Now you'll see detailed logs
```

⚡ Quick Commands Reference

```
# Run example
python examples/autonomous_agent_example.py

# Start server
python start_with_autonomous_agent.py

# Run tests
python -m pytest tests/test_autonomous_agent.py -v

# Check imports
python -c "from core.goal_driven_agent import GoalDrivenAgent; print('✓ OK')"
```

🎓 Next Steps

1. ✅ Run the example
2. ✅ Start the server
3. ✅ Monitor via API
4. ✅ Register custom actions for your domain
5. ✅ Review learning insights
6. ✅ Gradually enable full autonomy

💡 Key Concepts

Autonomous: Agent acts without external commands
Proactive: Prevents issues before they occur
Goal-Driven: Works towards specific objectives
Learning: Improves strategies over time
Predictive: Uses forecasting for decisions

🏁 Success Criteria

Your autonomous agent is working correctly when:

1. ✅ Agent starts without errors
2. ✅ Goals are created automatically
3. ✅ Goals show execution progress
4. ✅ Learning insights show data accumulation
5. ✅ Status shows increasing uptime
6. ✅ API endpoints return valid data

Ready to go? Start with:

```
python examples/autonomous_agent_example.py
```

Questions? Check `AUTONOMOUS_BEHAVIOR_README.md` for detailed docs.