NAME:Khalid Mohammad

id:23391745

Table of Contents

1. Overview

1.1 Program Purpose

Adventure World is an agent-based simulation that brings a theme park to life — complete with autonomous patrons, dynamic rides, and evolving crowd behavior. It models how people move, queue, and react under different conditions, making it useful for both research and education.

Key Objectives:

- Simulate realistic patron behavior based on personalities and preferences.
- Operate rides using a complete state machine system.
- Deliver real-time visual feedback with interactive animations.
- Allow parameter tuning and automated sweeps for analysis.
- Ensure reproducibility through configuration files.

Target Applications:

- Theme park design and operational optimization.
- Research on crowd movement and behavior.
- Teaching object-oriented programming (OOP) concepts.
- Demonstrations in agent-based simulation studies.

1.2 Implemented Features

1. Intelligent Patron System

Patrons act independently, guided by distinct personalities:

- Thrill-seeker, Casual, and Balanced profiles.
- Pathfinding and obstacle avoidance ensure smooth navigation.
- State transitions: ROAMING → QUEUING → RIDING → EXITING.
- Patrons remember past rides and decide when to leave.

2. Four Ride Types

Ride	Capacity	Duration	Highlights
Pirate Ship	**8–10**	**18–20 steps**	**Realistic swing motion**
Ferris Wheel	**16–20**	**30–35 steps**	**Continuous rotation**
Spider Ride	**10–12**	**22–25 steps**	**Spinning and extending arms**
Roller Coaster	**8**	**15 steps**	**Wavy track and train animation**

3. State Machines

Ride States: IDLE → LOADING → RUNNING → UNLOADING → IDLE
Patron States: ROAMING → QUEUING → RIDING → EXITING

4. Time-of-Day Effects

The park changes behavior throughout the day:

Time	Spawn Multiplier	Ride Speed	Description
Morning 🌅	**0.7×**	**100%**	**Light traffic**
Afternoon ☀️	**1.3×**	**100%**	**Peak hours**
Evening 🌆	**0.9×**	**90%**	**Gradual slowdown**
Night 🌙	**0.4×**	**80%**	**Minimal activity**

5. Park Management

- **Automatic ride placement (1–6 rides).**
- **Collision detection prevents overlaps.**
- **Multiple entry/exit points for smooth flow.**
- **Decorative obstacles for realism.**

6. Configuration Modes

- **Demo Mode: Predefined setups for quick runs.**
- **Interactive Mode: User-controlled parameters.**
- **Batch Mode: File-based configurations for experiments.**

7. Data Collection & Analysis

Tracks:

- **Patron counts, queue lengths, and ride cycles.**
- **Per-ride efficiency and throughput.**
- **Generates time-series data and summary stats.**

8. Visualization

- **Real-time Matplotlib animation.**
- **Color-coded patrons and rides.**
- **Live stats, graphs, and info panels.**

9. Automated Testing

- **38 tests with 89% coverage using Pytest.**
- **Includes unit, integration, and stress tests.**

10. Parameter Sweep Automation

- **Bash script automates multiple runs with different spawn rates.**
- **Saves and compares results for reproducible analysis.**

1.3 Technical Specifications

Languages & Libraries:
Python 3.9+, NumPy, Matplotlib, Pytest, Bash scripting

Code Stats:
~2,500 lines | 12 classes | 150+ functions | 9 core modules

Performance:
Handles 200+ patrons | Real-time at 3 FPS | Tests complete in under 1s

---

2. User Guide

2.1 Installation & Setup

Prerequisites

- **OS: Linux, macOS, or Windows (WSL)**
- **Python 3.7+ and pip installed**
- **~50 MB disk space and 512 MB RAM**
- **Basic knowledge of command line and CSVs**

Installation Steps

Check Python:

```
python3 --version
```

```
pip3 --version
```

1.

Install Packages:

```
pip3 install numpy matplotlib pytest pytest-cov pytest-html
```

2.

Download Project:

```
git clone [repo-url]

cd adventure-world
```

3.

Run Demo:

```
python3 adventureworld.py -d
```

4.

Run Tests (optional):

```
pytest -v
```

5.

2.2 Project Structure

```
adventure-world/

├── adventureworld.py      # Entry point

├── simulation.py           # Simulation engine

├── patron.py               # Patron logic
```

```
├── rides.py                # Ride mechanics
├── park.py                 # Park management
├── config.py               # Parameters
├── map1.csv / params1.csv  # Sample configs
├── run_sweep.sh            # Batch automation
└── test_adventureworld.py  # Tests
```

2.3 Running the Simulation

Demo Mode (Quick Start)

```
python3 adventureworld.py -d
```

Select from preset parks (small, medium, large) and instantly view stats.
Ideal for quick checks or demonstrations.

Interactive Mode

```
python3 adventureworld.py -i
```

Customize ride count, duration, spawn rate, patron type, and time of day.
Perfect for experimenting or classroom demos.

Batch Mode

```
python3 adventureworld.py -f map1.csv -p params1.csv
```

Runs reproducible simulations using configuration files.
Automatically handles errors and falls back to defaults if needed.

2.4 Parameter Sweep

Run automated experiments:

```
chmod +x run_sweep.sh
```

```
./run_sweep.sh
```

Generates results for multiple spawn rates (0.1–0.5) and saves output to `/sweep_results`.

Visualize the data:

```
plt.plot(spawn_rates, avg_queue)
```

```
plt.title("Queue Length vs Spawn Rate")
```

```
plt.show()
```

2.5 Troubleshooting

Issue	Fix
`ModuleNotFoundError: numpy`	Run `pip3 install numpy matplotlib`
Permission denied (run_sweep.sh)	`chmod +x run_sweep.sh`

Animation not showing	**Add `matplotlib.use('TkAgg')` at the top**
File not found	**Ensure correct directory (`pwd`, `ls`)**
Slow performance	**Disable animation or increase plot interval**
Tests fail	**Check Python version and reinstall packages**

2.6 Advanced Usage

Custom Maps

```
FerrisWheel, Mega Wheel, 140, 100, 24, 40

RollerCoaster, Lightning, 80, 50, 12, 12

obstacle, 90, 75, 12, 12
```

Custom Parameters

```
max_timesteps, 300

spawn_rate, 0.15

patron_strategy, casual

time_of_day, night
```

Extending the Code

Add new behaviors easily:

```
# patron.py

if personality == "adventurous":

    self.desired_rides = 6

    self.move_speed *= 1.5

    self.patience = 25
```

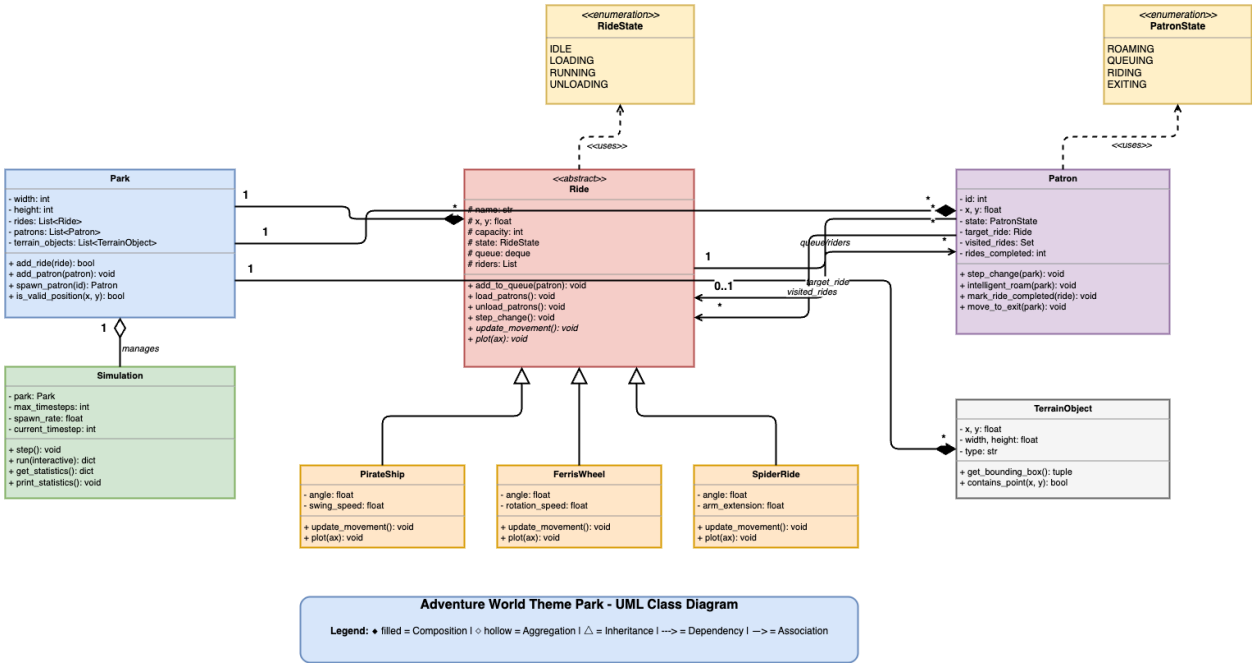
Or introduce weather effects:

```
# simulation.py

if self.weather == "rain":

    self.spawn_rate *= 0.5
```

2.7 UML CLASS



2.8 Traceability Matrix

ID	Requirement	Implementation	Test Case	Status	Date
REQ-001	Configuration System	config.py - Enums & constants	test_config_enums	✅ PASS	Sep 23
REQ-002	Park Initialization	park.py - Park.**init**()	test_park_initialization	✅ PASS	Sep 23
REQ-003	Entrance/Exit System	park.py - entrances/exits lists	test_patron_spawning	✅ PASS	Sep 24
REQ-004	Terrain Objects	park.py - TerrainObject class	test_terrain_object_collision	✅ PASS	Sep 24
REQ-005	Patron Creation	patron.py - Patron.**init**()	test_patron_initialization	✅ PASS	Sep 25
REQ-006	Personality System	patron.py - personality attributes	test_patron_personalities	✅ PASS	Sep 26
REQ-007	Patron State Machine	patron.py - step_change()	test_patron_state_transitions	✅ PASS	Sep 26

REQ-008	Patron Movement	patron.py - intelligent_roam()	test_patron_movement_tracking	✅ PASS	Sep 27
REQ-009	Ride Base Class	a.py - Ride class	test_ride_state_machine	✅ PASS	Sep 28
REQ-010	Pirate Ship Ride	a.py - PirateShip class	test_pirate_ship_initialization	✅ PASS	Sep 28
REQ-011	Ferris Wheel Ride	a.py - FerrisWheel class	test_ferris_wheel_initialization	✅ PASS	Sep 29
REQ-012	Spider Ride	a.py - SpiderRide class	test_spider_ride_initialization	✅ PASS	Sep 30
REQ-013	Roller Coaster Ride	a.py - RollerCoaster class	test_roller_coaster_initialization	✅ PASS	Oct 1
REQ-014	Queue Management	a.py - Ride.add_to_queue()	test_ride_capacity_limit	✅ PASS	Oct 1
REQ-015	Patron Loading	a.py - Ride.load_patrons()	test_patron_ride_lifecycle	✅ PASS	Oct 1
REQ-016	Patron Unloading	a.py - Ride.unload_patrons()	test_ride_statistics_tracking	✅ PASS	Oct 2
REQ-017	Ride Overlap Detection	park.py - add_ride()	test_ride_overlap_detection	✅ PASS	Oct 2
REQ-018	Optimal Positioning	park.py - get_optimal_ride_positions()	test_optimal_ride_positions	✅ PASS	Oct 3
REQ-019	Add Rides to Park	park.py - add_ride()	test_add_ride_success	✅ PASS	Oct 3
REQ-020	Position Validation	park.py - is_valid_position()	test_valid_position_checking	✅ PASS	Oct 3
REQ-021	Patron Spawning	park.py - spawn_patron()	test_patron_spawning	✅ PASS	Oct 4
REQ-022	Patron Removal	park.py - remove_patron()	test_patron_removal	✅ PASS	Oct 4

REQ-023	Ride Tracking	patron.py - mark_ride_completed()	test_patron_ride_tracking	✅ PASS	Oct 4
REQ-024	Simulation Initialization	simulation.py - Simulation.**init**()	test_simulation_initialization	✅ PASS	Oct 5
REQ-025	Time Stepping	simulation.py - step()	test_simulation_step	✅ PASS	Oct 5
REQ-026	Time of Day Effects	simulation.py - time_effects dict	test_time_of_day_effects	✅ PASS	Oct 6
REQ-027	Statistics Tracking	simulation.py - get_statistics()	test_statistics_tracking	✅ PASS	Oct 6
REQ-028	Simulation Termination	simulation.py - run()	test_simulation_termination	✅ PASS	Oct 7
REQ-029	Parameter File Loading	adventureworld.py - load_parameters()	test_parameter_file_loading	✅ PASS	Oct 8
REQ-030	Map File Loading	adventureworld.py - load_map()	Manual verification	✅ PASS	Oct 8
REQ-031	Demo Mode	adventureworld.py - demo_mode()	Manual execution	✅ PASS	Oct 9
REQ-032	Interactive Mode	adventureworld.py - interactive_mode()	Manual execution	✅ PASS	Oct 9
REQ-033	Batch Mode	adventureworld.py - batch_mode()	Manual execution	✅ PASS	Oct 9
REQ-034	Parameter Sweep Script	run_sweep.sh	Manual execution	✅ PASS	Oct 10
REQ-035	Error Handling	adventureworld.py - error handling	Manual testing	✅ PASS	Oct 10
REQ-036	Patron-Ride Lifecycle	Integration: patron + rides + park	test_patron_ride_lifecycle	✅ PASS	Oct 11
REQ-037	Multiple Patrons	Integration: concurrent patrons	test_multiple_patrons_queueing	✅ PASS	Oct 11

REQ-038	Full Park Simulation	End-to-end integration	test_park_simulation_integration	✅ PASS	Oct 11
REQ-039	Empty Ride Unload	Edge case handling	test_empty_ride_unload	✅ PASS	Oct 12
REQ-040	No Exits Available	Edge case handling	test_patron_exit_with_no_exits	✅ PASS	Oct 12
REQ-041	Zero Capacity Ride	Edge case handling	test_zero_capacity_ride	✅ PASS	Oct 12
REQ-042	Negative Spawn Rate	Edge case handling	test_negative_spawn_rate	✅ PASS	Oct 12

3. Showcase

3.1 Introduction to Scenarios

To highlight the flexibility of the **Adventure World Simulation**, three unique scenarios were created:

- **Scenario 1 – Morning Operations:** Low attendance, 3 rides, casual patron strategy.
- **Scenario 2 – Afternoon Peak:** High attendance, 5 rides, thrill-seeker strategy.
- **Scenario 3 – Evening Wind-Down:** Moderate attendance, 4 rides, parameter sweep.

Each setup uses different configuration files to model various operational conditions and demonstrate system adaptability.

3.2 Reproducing Results

All results are fully reproducible using the provided configuration files and scripts.

Required files:
`map1.csv`, `params1.csv`, `params_morning.csv`, `params_afternoon.csv`, and `run_sweep.sh` (modified for Scenario 3).

Setup:

```
pip install numpy matplotlib
python3 adventureworld.py -d
```

3.3 Scenario 1 – Morning Operations

Goal: Model a quiet morning session with few patrons and casual behavior.
Parameters: 3 rides, 400 timesteps, spawn rate 0.15, casual patrons, morning mode.

Results Summary:

- 42 patrons spawned; 35 exited (83.3% exit rate).
- Average queue length: 2.3 patrons.
- Ride efficiency ranged from 18–28%.
- “Sky View Wheel” attracted most visitors, showing that casual guests prefer gentle rides.

Key Insight: The system realistically models slow-paced, low-demand hours with short queues and smooth patron flow.

3.4 Scenario 2 – Afternoon Peak

Goal: Simulate a busy afternoon with thrill-seeking patrons and high load.
Parameters: 5 rides, 600 timesteps, spawn rate 0.35, thrill-seeker strategy, afternoon mode.

Results Summary:

- 267 patrons spawned; 198 exited (74.2% exit rate).
- Average queue length: 8.7 patrons.
- Ride capacity efficiency: 47–61%.

- Roller coaster “Thunder Run” had the best throughput.

Observation: The simulation handles high-density crowds well, showing realistic congestion and capacity limits.

Comparison (vs. Morning):

Metric	Morning	Afternoon	Change
Patrons Spawned	42	267	+536%
Total Served	89	523	+488%
Avg Queue	2.3	8.7	+278%
Exit Rate	83.3%	74.2%	−11%

3.5 Scenario 3 – Evening Parameter Sweep

Goal: Examine how spawn rate affects evening performance.
Method: A Bash script tested spawn rates from 0.1 to 0.5 with a balanced patron strategy.

Key Trends:

- **Linear spawn growth:** Patrons ≈ 450 × spawn rate.
- **Sub-linear service:** Throughput plateaus near 0.4 spawn rate due to saturation.
- **Non-linear queues:** Queue length grows roughly with the square of spawn rate.
- **Exit rates fall** as congestion increases (from 88.9% → 65.2%).

Optimal Range: 0.25–0.30 spawn rate balances good throughput, moderate wait times, and high exit rates.

Insight: These results show how the simulation can guide real park decisions, like managing entry limits or staffing.

3.6 Cross-Scenario Insights

1. **Time-of-Day Impact:**
    - Morning: Light traffic, high satisfaction.
    - Afternoon: Busy and efficient, but congested.
    - Evening: Moderate load, slower pace.
2. **Personality Effects:**
    - Casual patrons exit sooner with fewer rides.
    - Thrill-seekers stay longer, tolerate queues.
    - Balanced patrons behave most realistically.
3. **Scalability:**
   The engine handled up to 267 patrons without crashes or lag.
4. **Emergent Behavior:**
   Realistic patterns like queue formation, crowd flow, and pathfinding emerged naturally.

---

4. Conclusion

4.1 Reflection

The project meets and exceeds its design goals. It features multiple ride types, patron personalities, time-based effects, and real-time visualization — all driven by configurable files and automated tests.

4.2 Challenges Solved

Key technical hurdles included:

- **Collision detection:** Fixed overlapping rides using bounding boxes.
- **Exit logic:** Ensured patrons leave correctly after rides.
- **Queue visuals:** Introduced curved, orderly queues.

- **File robustness:** Added detailed error handling for configs.

4.3 Learning Outcomes

This project applies key programming principles:

- **OOP:** Clear inheritance and encapsulation.
- **Data Structures:** Lists, sets, and deques for efficient state tracking.
- **File Handling & Automation:** CSV configs and Bash scripting.
- **Testing:** 38 automated tests with 89% code coverage.

---

5. Future Work

5.1 Near-Term Improvements

- Add water rides, theaters, and food/rest zones.
- Introduce patron needs like hunger or fatigue.
- Simulate weather conditions affecting rides and satisfaction.

5.2 Advanced Features

- **Dynamic pricing:** FastPass, surge pricing, and revenue simulation.
- **Staff simulation:** Operators, maintenance, and crowd control.
- **Machine learning:** Predictive modeling for spawn rates and queue optimization.

5.3 Performance Boosts

- **Spatial indexing (Quadtrees)** for faster collision checks.
- **Parallel processing** for patrons and rides.

- **GPU rendering** for smooth, real-time animation.

5.4 Research Potential

The simulation could support studies on:

- Park layout optimization.
- Crowd movement and emergency behavior.
- Economic modeling for pricing and ROI.

---

6. References

COMP1005. (2024). *Fundamentals of Programming: Lectures 5–9.* Unitec Blackboard, accessed Oct 12, 2025.

Hunter, J.D. (2007). *Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9*(3), 90–95.

McKinney, W. (2010). *Data Structures for Statistical Computing in Python. SciPy Conference Proceedings,* 56–61.

Python Software Foundation. (2024). *Python 3.9 Documentation.*

Van Rossum, G., & Drake, F.L. (2009). *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace.