# Emergency Room Simulation
# Milestone 2: Initial Implementation

Aashna Suthar, Ash Arul
CS 4632 Modeling and Simulation
Section W01 Fall 2025

## I. Project Overview

This document reports progress on Milestone 2 for our Emergency Room Simulation project for the Modeling and Simulation class. In Milestone 1, most of the work was focused on the planning stage: writing up the proposal, making UML diagrams, and explaining how the system would work in theory. Now with Milestone 2, we've moved from design into real code and can actually show the simulation running. The main goal here is to demonstrate that the system is being built step by step, the logic is starting to come together, and the project is following the plan we outlined in M1.

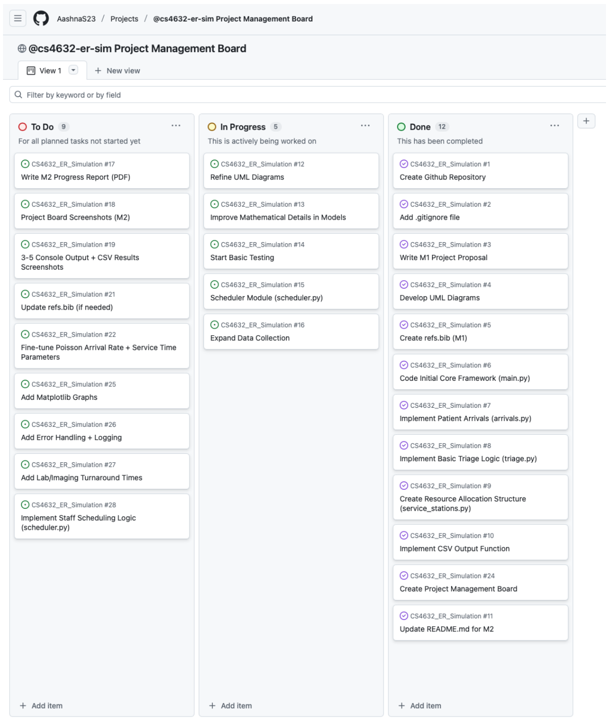
So far, we have the basics of the simulation working. Patients arrive randomly based on a Poisson process, which mimics how unpredictable ER visits really are. Each patient is given a severity level using the Emergency Severity Index (ESI), and that determines their priority in the queue. Nurses handle triage, doctors provide treatment, and rooms are required for patients to actually be seen. All of these resources have limits, so when demand is high, patients are forced to wait. The simulation tracks these waits, service times, and how busy the staff are. The results are stored in CSV files, which lets us review the data and compare different runs.

The main purpose of this milestone is to confirm that the simulation framework is set up correctly and that progress is being made toward the final project. At this stage, the ER system is already showing useful patterns, like how resource bottlenecks appear when capacity is low. This progress gives us confidence that we're on track for M3, where more advanced features like scheduling, labs/imaging delays, and visualization will be added. Overall, this milestone serves as a checkpoint to make sure the design and code are aligned and the project is moving forward in the right direction.

## II. Project Management Board Status

We are keeping track of our work using a GitHub Project Board. This board helps us stay organized by laying out all the tasks in one place and sorting them into different categories. It makes it clear what has already been done, what is currently being worked on, and what we still need to focus on next. By using the board, our progress feels more structured and it is easier to show exactly where the project stands at this point in time.

The board is split into three main columns: *To Do*, *In Progress*, and *Done*. Tasks start in the *To Do* column and are moved to *In Progress* once we begin working on them. When a task is completed, it gets moved into the *Done* column. This setup helps us visualize our workflow and ensures nothing slips through the cracks. For this milestone, it also gives us a good way to show the instructor how the work is being managed step by step.


AashnaS23 / Projects / @cs4632-er-sim Project Management Board
View 1
New view
Filter by keyword or by field
Add item
Add item
Add item

@cs4632-er-sim Project Management Board

To Do 9
For all planned tasks not started yet

- CS4632_ER_Simulation #17
  Write M2 Progress Report (PDF)
- CS4632_ER_Simulation #18
  Project Board Screenshots (M2)
- CS4632_ER_Simulation #19
  3-5 Console Output + CSV Results Screenshots
- CS4632_ER_Simulation #21
  Update refs.bib (if needed)
- CS4632_ER_Simulation #22
  Fine-tune Poisson Arrival Rate + Service Time Parameters
- CS4632_ER_Simulation #25
  Add Matplotlib Graphs
- CS4632_ER_Simulation #26
  Add Error Handling + Logging
- CS4632_ER_Simulation #27
  Add Lab/Imaging Turnaround Times
- CS4632_ER_Simulation #28
  Implement Staff Scheduling Logic (scheduler.py)

In Progress 5
This is actively being worked on

- CS4632_ER_Simulation #12
  Refine UML Diagrams
- CS4632_ER_Simulation #13
  Improve Mathematical Details in Models
- CS4632_ER_Simulation #14
  Start Basic Testing
- CS4632_ER_Simulation #15
  Scheduler Module (scheduler.py)
- CS4632_ER_Simulation #16
  Expand Data Collection

Done 12
This has been completed

- CS4632_ER_Simulation #1
  Create Github Repository
- CS4632_ER_Simulation #2
  Add .gitignore file
- CS4632_ER_Simulation #3
  Write M1 Project Proposal
- CS4632_ER_Simulation #4
  Develop UML Diagrams
- CS4632_ER_Simulation #5
  Create refs.bib (M1)
- CS4632_ER_Simulation #6
  Code Initial Core Framework (main.py)
- CS4632_ER_Simulation #7
  Implement Patient Arrivals (arrivals.py)
- CS4632_ER_Simulation #8
  Implement Basic Triage Logic (triage.py)
- CS4632_ER_Simulation #9
  Create Resource Allocation Structure (service_stations.py)
- CS4632_ER_Simulation #10
  Implement CSV Output Function
- CS4632_ER_Simulation #24
  Create Project Management Board
- CS4632_ER_Simulation #11
  Update README.md for M2

Fig. 1. Current Project Board layout and task distribution

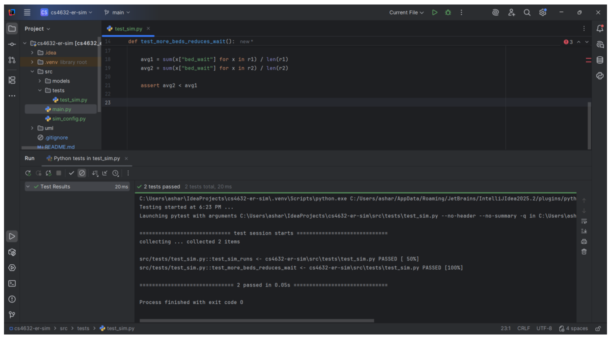
Right now, the **Done** column shows things like the repository setup, core modules (arrivals, triage, service), README updates, and the .gitignore file. These were important first steps to get the foundation in place. The **In Progress** column has tasks like writing automated tests, refining UML diagrams, and updating documentation. These are being worked on actively and will likely move to **Done** soon. The **To Do** column still has larger upcoming tasks like adding scheduling logic, modeling labs/imaging, and building out plots and visualizations.

Overall, the board makes it easy to see that steady progress is being made. It also lets us add new tasks when ideas come up, so the project can grow in a more organized way instead of being all over the place. For example, when we realized visualization would be useful for results, we added it as a new task under *To Do*. Having this system keeps us accountable and makes the project management side of things much more professional.

## III. Implementation Evidence

The screenshots below show that the project is moving from just the planning stage into actual working code. The first figure shows that our automated tests are running and passing, which means the core pieces of the code are working like they should. One of the tests even checks if adding more beds actually lowers the wait times, and it passed, so that's a good sign that the logic is doing what we expected and that the simulation is behaving consistently with our design.

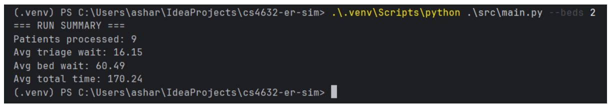
The console output is also an important piece of evidence that the simulation is running correctly. When the program is executed from `main.py`, the terminal prints a live summary of the run. This includes the number of patients that entered the system, how many were processed, and the average wait times for both triage and beds. By looking at this output, we can confirm that the logic is working because the numbers change realistically when parameters like bed capacity are adjusted. For example, with only 2 beds, the console showed that only 9 patients could be processed with long wait times, while runs with 5 and 10 beds showed more patients being processed but highlighted new bottlenecks in triage. This demonstrates the core functionality of the simulation and proves that it is following the design we outlined in M1. The other figures show runs of the simulation with different numbers of beds. Each run gives a short summary of how many patients were processed and what the average wait times looked like. When there are fewer beds, only a small number of patients can be processed and the average wait time goes up. When there are more beds, the ER can handle more patients, but then the bottleneck shifts to other areas like triage. These results show that the simulation is behaving realistically and responding to changes in capacity the way we would expect.



```
def test_more_beds_reduces_wait():

    avg1 = sum(x["bed_wait"] for x in r1) / len(r1)
    avg2 = sum(x["bed_wait"] for x in r2) / len(r2)

    assert avg2 < avg1
```

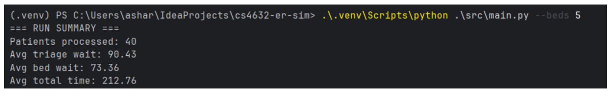
```
✓ 2 tests passed  2 tests total, 20 ms
C:\Users\ashar\IdeaProjects\cs4632-er-sim\.venv\Scripts\python.exe C:/Users/ashar/AppData/Roaming/JetBrains/IntelliJIdea2025.2/plugins/pyth
Testing started at 6:23 PM ...
Launching pytest with arguments C:\Users\ashar\IdeaProjects\cs4632-er-sim\src\tests\test_sim.py --no-header --no-summary -q in C:\Users\ash

============================= test session starts ==============================
collecting ... collected 2 items

src/tests/test_sim.py::test_sim_runs <- cs4632-er-sim\src\tests\test_sim.py PASSED [ 50%]
src/tests/test_sim.py::test_more_beds_reduces_wait <- cs4632-er-sim\src\tests\test_sim.py PASSED [100%]

============================== 2 passed in 0.05s ===============================

Process finished with exit code 0
```

Fig. 2. Automated tests running with `pytest` (`pytest_passed.png`). Both tests passed successfully, including the one that checks if adding more beds reduces wait times. This shows that not only is the code executing without errors, but it is also meeting the specific behavior we designed in the logic. Passing tests at this stage gives us confidence that the basic simulation framework is correct, stable, and ready to be built on in later milestones.



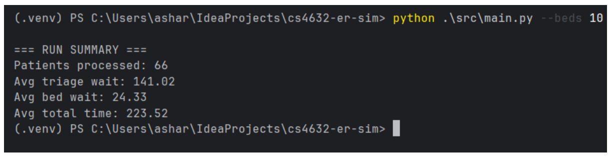
```
(.venv) PS C:\Users\ashar\IdeaProjects\cs4632-er-sim> .\.venv\Scripts\python .\src\main.py --beds 2
=== RUN SUMMARY ===
Patients processed: 9
Avg triage wait: 16.15
Avg bed wait: 60.49
Avg total time: 170.24
(.venv) PS C:\Users\ashar\IdeaProjects\cs4632-er-sim>
```

Fig. 3. Simulation run with 2 beds (`results_beds2.png`). Only 9 patients were processed in total, and the average bed wait time was high (around 60 minutes). This result makes sense because limited capacity forces patients to wait much longer before being treated. It also highlights that with fewer resources, the system quickly gets overwhelmed, which reflects real ER problems when staffing or space is low. This demonstrates how the simulation can mirror real-world issues by showing how under-resourced setups lead to heavy delays.



```
(.venv) PS C:\Users\ashar\IdeaProjects\cs4632-er-sim> .\.venv\Scripts\python .\src\main.py --beds 5
=== RUN SUMMARY ===
Patients processed: 40
Avg triage wait: 90.43
Avg bed wait: 73.36
Avg total time: 212.76
```

Fig. 4. Simulation run with 5 beds (`results_beds5.png`). In this case, 40 patients were processed. The average bed wait time improved compared to the 2-bed scenario, but triage wait jumped to about 90 minutes. This shows how adding resources in one area helps, but then creates a new bottleneck somewhere else, in this case triage. It demonstrates how the simulation helps us see the tradeoffs and system-wide effects of changing capacity. Even though more patients were treated, the delays simply shifted to the start of the process.



```
(.venv) PS C:\Users\ashar\IdeaProjects\cs4632-er-sim> python .\src\main.py --beds 10

=== RUN SUMMARY ===
Patients processed: 66
Avg triage wait: 141.02
Avg bed wait: 24.33
Avg total time: 223.52
(.venv) PS C:\Users\ashar\IdeaProjects\cs4632-er-sim>
```

Fig. 5. Simulation run with 10 beds (`results_beds10.png`). The ER processed 66 patients, and bed wait times became almost zero. However, the triage wait time went way up, hitting over 140 minutes. This confirms that once beds are no longer the issue, triage becomes the main bottleneck. It shows how the simulation can uncover hidden capacity issues and how fixing one part of the system doesn't automatically fix the whole flow. By scaling one resource too far ahead of others, the system creates imbalances that cause new choke points, which is exactly the type of insight simulation is meant to provide.

## IV. Status Summary

The Status Summary goes over:

- Whats Implemented
- Key Accomplishments since M1
- Challenges
- Next Steps

Since Milestone 1, the project has moved from just being an idea to having actual working code. Patients now arrive randomly using a Poisson process, get a severity level with the Emergency Severity Index, and move through triage, doctors, and rooms, which are all modeled as limited resources. The simulation collects key metrics like wait times, length of stay, and staff utilization, and writes them into CSV files for later analysis. The repository is also more organized now with separate files for arrivals, triage, and resources, plus an updated README, .gitignore, and setup instructions that make it easier for anyone else to clone and run the project. These changes show clear progress from a design only stage to a functioning system that already demonstrates useful insights.

One of the biggest accomplishments since Milestone 1 is that the simulation now behaves realistically when parameters change. The bed capacity experiments showed that wait times react the way we expect, and the tests confirm that the logic is sound. Another accomplishment is setting up a GitHub Project Board, which keeps our work structured and transparent by tracking what's done, what's in progress, and what's still to come. Challenges so far have mainly been around deciding which features to prioritize and how much detail to add at this stage. For example, staff scheduling is still tricky to implement because it adds complexity with shifts and staggered times, so we decided to push that into the next stage. The plan moving forward is to finish scheduling, add labs and imaging delays, and build graphs so that results are easier to visualize. We'll also expand test coverage to make sure the code stays reliable as we add more features. Overall, the project is on track, and the progress so far gives us a strong foundation for Milestone 3.

## V. Repository & Board Links

- Github Repository Link: cs4632-er-sim
- Project Management Board: cs4632-er-sim Project Management Board