Politecnico di Torino

POLITECNICO DI TORINO

Corso di Laurea
in Ingegneria Matematica

Homework di Business Analytics

Cristina Baitan 343428
Cecilia Bergamini 343341

Anno Accademico 2024-2025

Contents

Abstract

In questo report si presenta la costruzione di una simulazione a eventi discreti seguendo un approccio orientato alla programmazione a oggetti. Sono state sviluppate classi astratte e generiche che definiscono l'infrastruttura base della simulazione e che vengono poi specializzate per modellare un caso di studio ispirato al contesto ospedaliero.

Tale contesto concreto presenta un flusso dei pazienti all'interno di un ospedale, a partire dal loro arrivo, il passaggio attraverso la visita medica, l'eventuale ricovero post-visita, fino alla dimissione. Vengono inoltre considerate dinamiche reali come la gestione delle priorità, la presenza di buffer limitati, la limitata disponibilità di risorse, la compatibilità tra entità e risorse, e la possibilità di guasti temporanei nelle risorse.

Verranno eseguite diverse simulazioni del modello, dalle quali saranno estratte statistiche come il numero di pazienti rifiutati e il tempo medio di attesa in coda, al fine di valutare l'efficienza e il comportamento del sistema ospedaliero nel tempo.

Capitolo 1

1.1 Sviluppo di simulazione a eventi

1.1.1 Classi astratte per la simulazione a eventi

La classe *SimulationManager* rappresenta l'ambiente centrale della simulazione ad eventi discreti, gestendo il tempo, gli eventi e le risorse coinvolte. Possiede come attributi principali il clock di simulazione e la lista degli eventi futuri.
I suoi compiti sono i seguenti:

- Tiene traccia degli oggetti della simulazione (come entità, risorse e code) grazie al metodo *register*.
- Pianifica e gestisce eventi futuri e aggiorna lo stato della simulazione: crea e inserisce gli eventi nella lista degli eventi futuri nel metodo *create_event_and_insert* e estrae l'evento più prossimo in *extract_event*.
- Contiene metodi per il reset e per l'inizializzazione della simulazione.

La classe *Event* rappresenta l'evento. Possiede un metodo astratto *event_manager* che verrà dettagliato nell'evento specifico.

La classe *Entity* rappresenta un'entità che attraversa il sistema simulato. Tiene traccia della coda o dello store in cui si trova e del suo stato attuale ("active", "wait" e "processing").

La classe *Resource* rappresenta una risorsa generica nel sistema che processa entità. Possiede lo stato operativo della risorsa ("idle", "busy", "failed" e "reserved") come attributo.

La classe *Queue* implementa una coda prioritaria con capacità massima e inserimento ordinato per priorità e tempo.

La classe *QueueMember* rappresenta un nodo in una coda, contiene attributi quali: l'entità rappresentata e i riferimenti al nodo predecessore e successore, formando una lista doppiamente collegata. Quest'ultima classe è stata definita in modo da poter gestire le operazioni sulla coda come inserimento e rimozione: quando un'entità viene inserita nella coda con insert_in_queue(), viene creato un nuovo oggetto QueueMember e lo si posiziona correttamente nella struttura aggiornando i nodi predecessori e successori.

La classe *Store* rappresenta un luogo con una capacità limitata, dove possono essere trattenute entità. Ci sono attributi che indicano la capacità massima e quella disponibile.

La classe *Distribution* è una classe astratta che rappresenta una generica distribuzione di probabilità. Sono state implementate in sottoclassi la distribuzione normale e esponenziale con metodo

che permette di generare campioni dalla distribuzione specifica.

La classe *DataCollection* e le sue sottoclassi forniscono un'interfaccia per la raccolta e gestione di dati all'interno di una simulazione. Supporta tre modalità principali:

- raccolta di dati temporali (*DataTime*),
- raccolta di dati non associati al tempo (*DataWithoutTime*),
- gestione di statistiche aggregate (*DataStat*).

Ogni sottoclasse implementa metodi specifici per l'aggiornamento, il reset e la stampa dei dati. In particolare, la classe *DataTime* include anche funzionalità per il calcolo della media integrale, utile per analizzare variabili che variano nel tempo con andamenti a gradini.

1.1.2 Il caso studio

Dopo aver definito le classi astratte si sono costruite sottoclassi specifiche per un esempio di simulazione di contesto ospedaliero.
Si considera la seguente situazione: si vuole simulare la permanenza di pazienti all'interno dell'ospedale. Ogni paziente, al suo arrivo, si mette in coda in attesa di essere visitato da uno dei medici disponibili. Una volta effettuata la visita, se vi sono posti liberi, il paziente viene trasferito in una delle stanze post visita, dove riceve assistenza da parte degli infermieri. Dopo un periodo di permanenza variabile, specifico per ciascun paziente, egli lascia definitivamente l'ospedale.
Tale schema si ispira al problema presentato in *https://ihtc2024.github.io*.
Sono state fatte le seguenti ipotesi:

1. Ogni paziente ha una priorità di visita differente e quindi la coda terrà conto della priorità prima che del tempo di arrivo. Quindi a livello di modellizzazione la lista di attesa è una coda a priorità FIFO

2. Gli arrivi dei pazienti segue un processo di Poisson e il tempo di servizio da parte dei dottori è anch'esso esponenzialmente distribuito. Inoltre, al termine della coda il paziente potrà essere visitato da uno dei dottori presenti all'ospedale. Abbiamo quindi una coda M/M/k con k$\geq$1 numero di dottori.

3. I dottori non sono indistinguibili poichè modellizzati con un attributo "capacità". Un dottore con capacità più alta può processare pazienti con capacità richiesta più alta. Sarà necessario un certo matching tra capacità richiesta e fornita.

4. La coda ha posti limitati quindi un paziente non viene fatto mettere in coda se la coda è arrivata a capacità massima.

5. Conseguentemente alla visita/operazione dal dottore il paziente può spostarsi in una stanza dell'ospedale. Si immagini che al paziente, appena conclusa l'operazione dal dottore, venga assegnata una stanza; il posto per tale paziente sarà riservato. Tuttavia, se non sono disponibili stanze con posti liberi sufficienti, il paziente viene dimesso e mandato a casa.

6. C'è una corrispondenza univoca tra infermieri e stanze: un infermiere per stanza e una stanza per infermiere.

7. C'è la possibilità che gli infermieri o i dottori si debbano assentare e che quindi non possano più processare pazienti durante il tempo di assenza. Tale possibilità è contemplata solo nel caso in cui questo tempo di assenza inizi quando il medico o l'infermiere non stiano già processando pazienti.

Per rappresentare queste situazioni abbiamo definito delle classi figlie delle classi astratte. Nella classe *Resource* sono state aggiunte le sottoclassi:

- *Doctor*: con attributo relativo alla capacità massima che può fornire.

- *Nurse*: con attributi relativi al suo skill level e alla stanza in cui si trova.

Nella classe *Entity* è stata aggiunta la sottoclasse:

- *Patient*: con attributi relativi alla capacità richiesta al dottore, allo skill level richiesto all'infermiere e al tempo di permanenza nella stanza.

La classe *Queue* gestisce la coda all'ingresso dell'ospedale, mentre la classe *Store* modella le stanze di degenza.
Nella classe *Event* sono state aggiunte le sottoclassi relative all'evento specifico:

- *Arrival*: gestisce l'arrivo di pazienti
- *StartProcessDoctor*: gestisce l'inizio del processamento di pazienti da parte dei dottori
- *EndProcessDoctor*: gestisce la fine del processamento di pazienti da parte dei dottori
- *StartProcessNurse*: gestisce l'inizio del processamento di pazienti da parte degli infermieri
- *EndProcessNurse*: gestisce la fine del processamento di pazienti da parte degli infermieri
- *Failure*: gestisce i failure sulle risorse (dottori e infermieri)
- *Recovery*: gestisce i recovery sulle risorse (dottori e infermieri)

Il metodo astratto *event_manager* viene implementato in ciascuna sottoclasse e definisce il comportamento del sistema al verificarsi di un evento. In particolare, aggiorna lo stato delle entità e delle risorse coinvolte, interagisce con code e stanze, e programma eventuali eventi futuri.
Lo scheduling di eventi futuri da parte di eventi presenti è raffigurato approssimativamente nell'event graph in Figura 1.1.

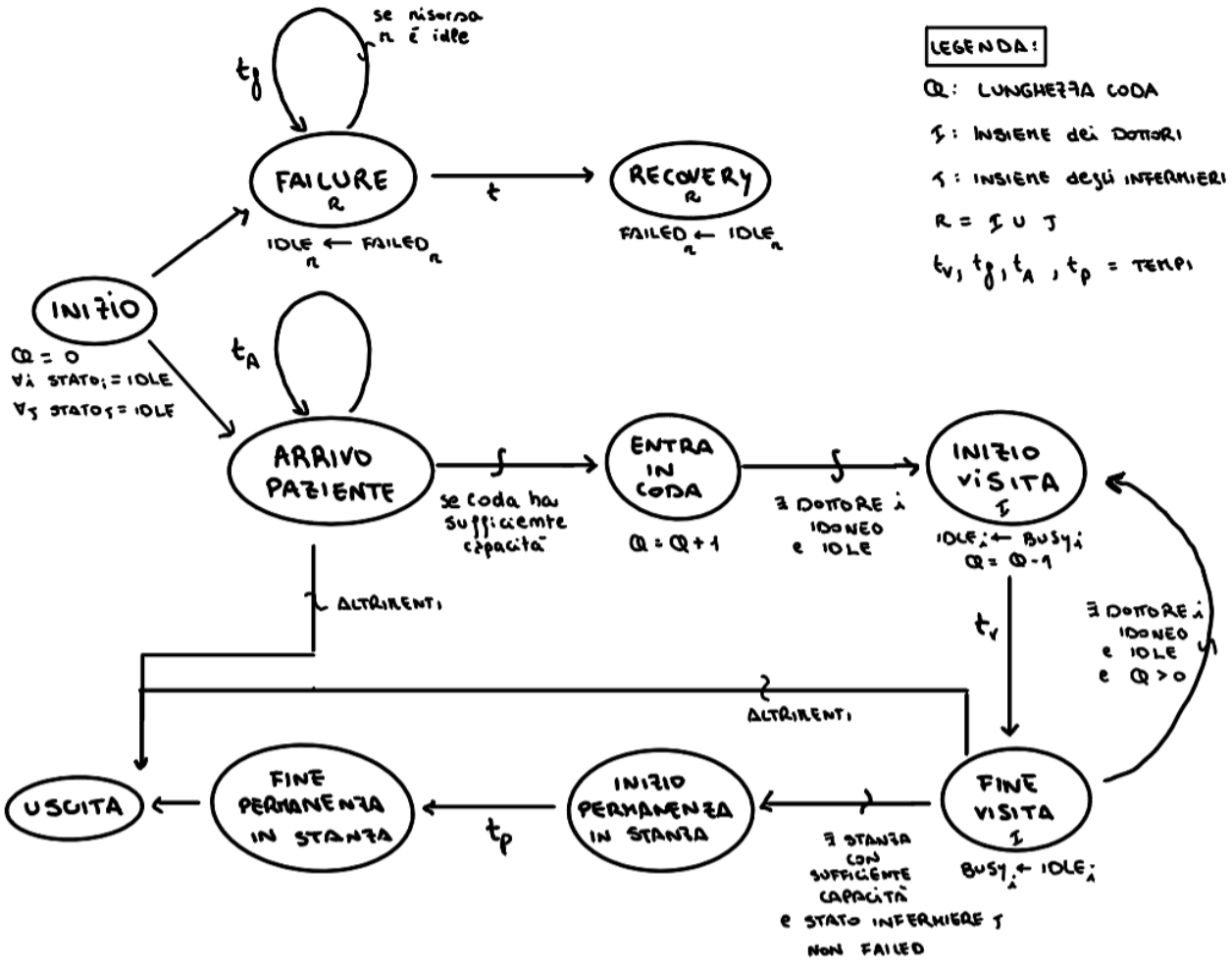

Figure 1.1: Event graph del contesto ospedaliero

Capitolo 2

2.1 Risultati e output analysis

Nelle simulazioni realizzate si sono considerati i seguenti componenti:

Tipo Risorsa	Numero Risorse	Capacità per Risorsa
Stanze	3	capacity_max = [1, 3, 2]
Infermieri	3	skill_level = [3,2,4]
Dottori	2	capacity_max = [1, 1]

I risultati seguenti sono stati ottenuti impostando $seed = 42$, $num\_sim = 1$ in aggiunta ai parametri illustrati in Tab.2.1.

period_max	par_arrival	par_recovery	par_failure	par_process_1	par_process_2	cap_max_queue1
45	0.6	0.9	0.0003	0.5	[3, 1]	6

Table 2.1

Dove $par\_recovery$ è il parametro di intertempo esponenziale che intercorre tra failure e guarigione e $par\_failure$ il parametro di intertempo esponenziale fra due failure.

Si riporta in Fig.(2.1) la lunghezza della coda nel tempo. Si osserva che non viene mai raggiunta la capacità massima di 6 e che la lunghezza massima raggiunta nel tempo è 2. In Fig.(2.2) si riporta invece il tempo di attesa in coda per ogni entità. Le Fig.(2.3), (2.4) riportano il numero di pazienti rispettivamente nelle stanze 1 e 2 durante il periodo di simulazione.

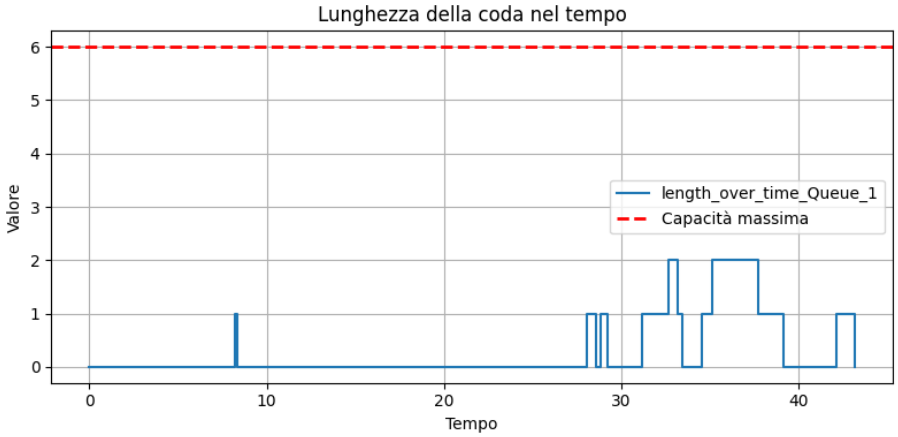

Figure 2.1: Lunghezza della coda di attesa durante il periodo considerato.

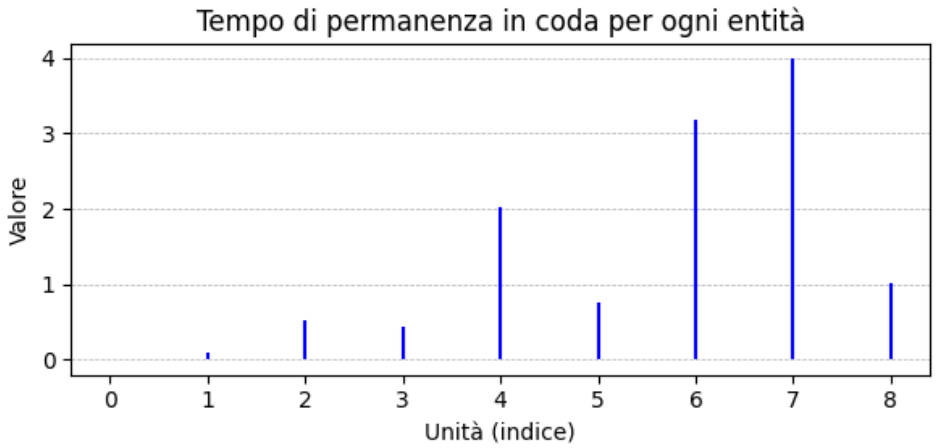

Figure 2.2: Tempo di permanenza in coda per ogni entità.

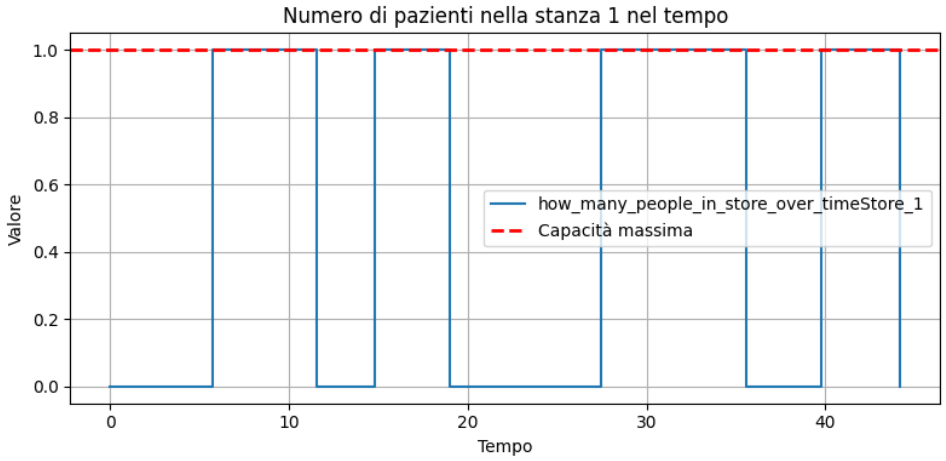

Figure 2.3: Numero di pazienti presenti nella stanza 1 durante il periodo considerato.

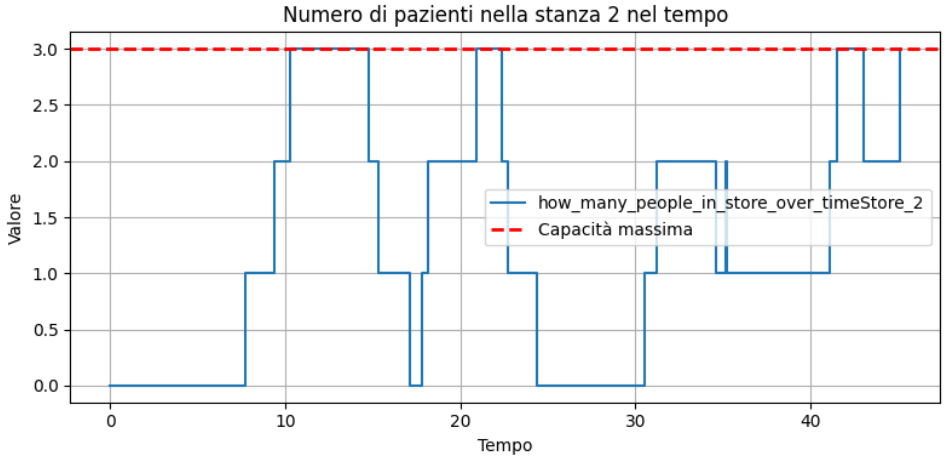

Figure 2.4: Numero di pazienti presenti nella stanza 2 durante il periodo considerato.

Si illustra la differenza positiva di skill level tra paziente e infermiere 1, Fig.(2.5), e paziente e infermiere 2, Fig.(2.6). Nello specifico, per l'infermiere 1 la differenza di skill level rispetto al paziente 4 è positiva e pari a 1 poichè l'infermiere ha un livello inferiore di tale quantità rispetto a quello richiesto dal paziente 4.

Figure 2.5: Differenza di skill level per il primo infermiere

Figure 2.6: Differenza di skill level per il secondo infermiere

Nella Tabella 2.2 si riportano le statistiche calcolate in termini di proporzione del numero di pazienti persi sul totale perchè raggiunta la capienza massima della coda nel momento del loro arrivo e la proporzione del numero di dimissioni a causa di assenza di posti disponibili nelle stanze. Nello specifico, sono state eseguite 20 repliche e si è presa la media campionaria dei valori ottenuti.

proporzione media di entità perse prima di mettersi in coda	proporzione media di entità perse per la capienza stanze
0.0005025	0.03043

Table 2.2: Statistiche su pazienti persi

Si evince che il numero medio di entità non ammesse nella prima coda è lo 0.05% del to-

tale, mentre nelle stanze rappresenta il 3% dei pazienti totali. Questa prima analisi suggerisce quindi che, più che aumentare la capacità massima della coda, risulta prioritario intervenire sull'ampliamento della capienza delle stanze o sul miglioramento dell'efficienza del processamento da parte degli infermieri. È importante sottolineare che questi risultati sono fortemente influenzati dalla probabilità di failure: infatti, incrementando il parametro $par\_failure$, stabilendo ad esempio $par\_failure = 0.8$, aumenta sensibilmente la percentuale media di entità non ammesse in coda, che diventa circa il 7%.

proporzione media di entità perse prima di mettersi in coda	proporzione media di entità perse per la capienza stanze
0.07323313529089522	0.015306122448979591

Table 2.3: Statistiche su pazienti persi aumentando la probabilità di failure

Sempre considerando i parametri illustrati in Tab. 2.1, le statistiche relative alle medie temporali (medie integrali) rispetto alla lunghezza della coda e al numero di persone per ciascuna stanza sono riportate in Tab. 2.4.

media integrale lunghezza della coda	media integrale dimensione store1	media integrale dimensione store2	media integrale dimensione store3
0.278	0.5095	1.241	0.395

Table 2.4: Media integrale relativa a lunghezza della coda e numero di pazienti per stanza.

Tali medie integrali sono state calcolate tramite il metodo *calculate_integral_mean* presente nella sottoclasse *DataTime*, comprendente le statistiche legate al tempo.

2.1.1 Stati transienti

Nella casistica in cui il rate di arrivo (par_arrival) dei pazienti sia maggiore rispetto alla somma dei rate di processo dei dottori (par_process_1) la coda è transiente e la sua lunghezza non si stabilizza nel tempo, anzi ha un andamento di crescita; non esiste una distribuzione stazionaria. Per valutare le statistiche rispetto alla lunghezza della coda e al tempo di permanenza in coda sono stati impostati i seguenti parametri:

period_max	par_arrival	par_recovery	par_failure	par_process_1	par_process_2	cap_max_queue1
10000	1.5	0.9	0.0003	0.6	[3, 1]	10000

Table 2.5: Parametri utilizzati nella simulazione.

Si osserva che la probabilità di failure è stata impostata in modo tale che l'evento failure fosse altamente improbabile.

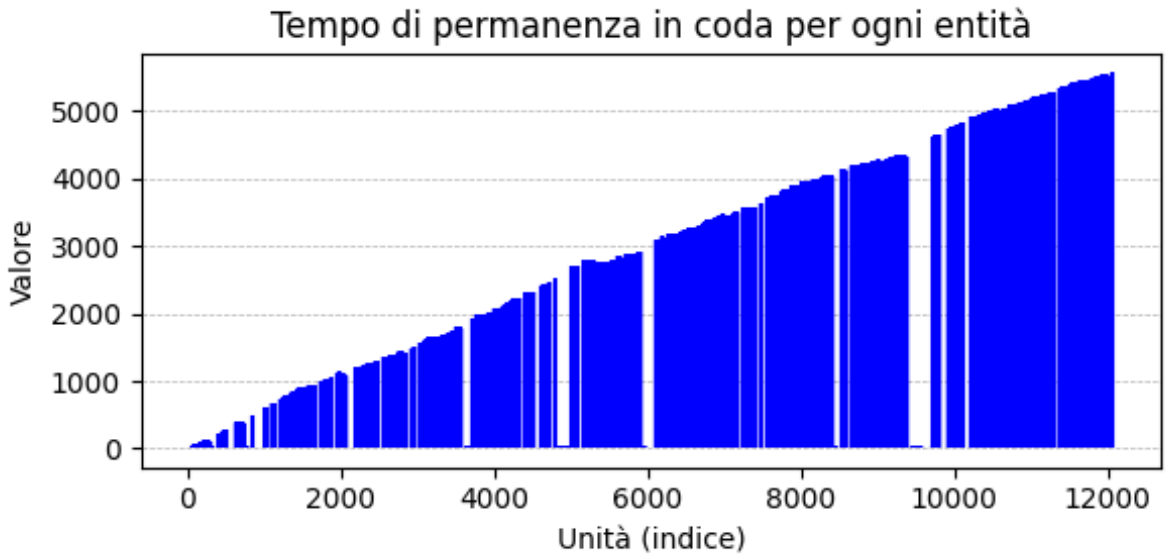

Figure 2.7: Tempo di permanenza in coda per ogni entità.

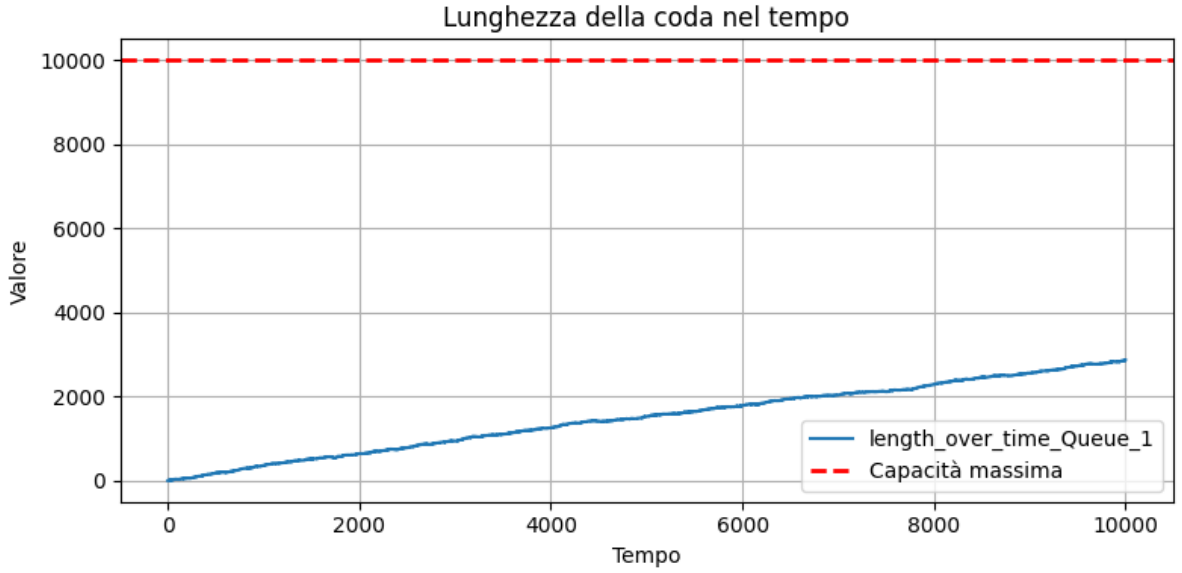

Figure 2.8: Lunghezza della coda nel tempo.

Sembrerebbero configurarsi stati transienti anche nel caso in cui si pone una probabilità di failure non trascurabile sebbene $par\_arrival < \#doctors \cdot par\_process\_1$:

period_max	par_arrival	par_recovery	par_failure	par_process_1	par_process_2	cap_max_queue1
10000	0.6	0.9	0.03	0.5	[3, 1]	15

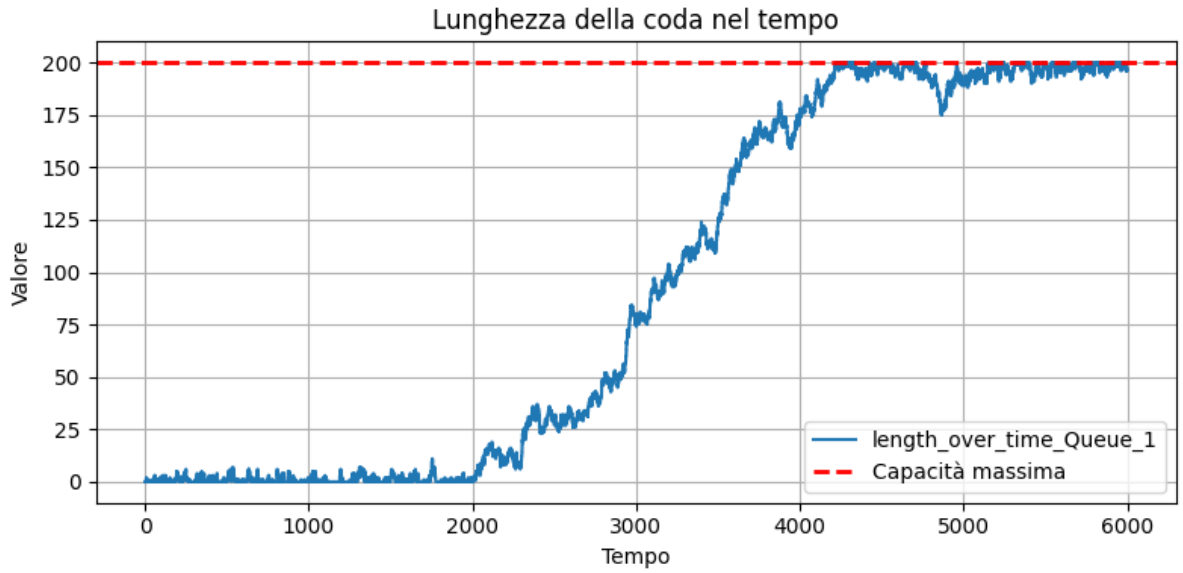

Figure 2.9: Lunghezza della coda nel tempo.

2.1.2 Stati positivo ricorrenti

Nel caso in cui il rate degli arrivi (par_arrival) sia inferiore alla somma dei rate di processo (par_process_1) la coda viene smaltita in modo sufficiente da non crescere indefinitamente, tende a stabilizzarsi, raggiungendo una distribuzione stazionaria. La probabilità di failure e la capacità massima sono state impostate in maniera tale da essere trascurabili.
Parametri settati per i seguenti risultati:

period_max	par_arrival	par_recovery	par_failure	par_process_1	par_process_2	cap_max_queue1
10000	0.6	0.9	0.00003	0.5	[3, 1]	15

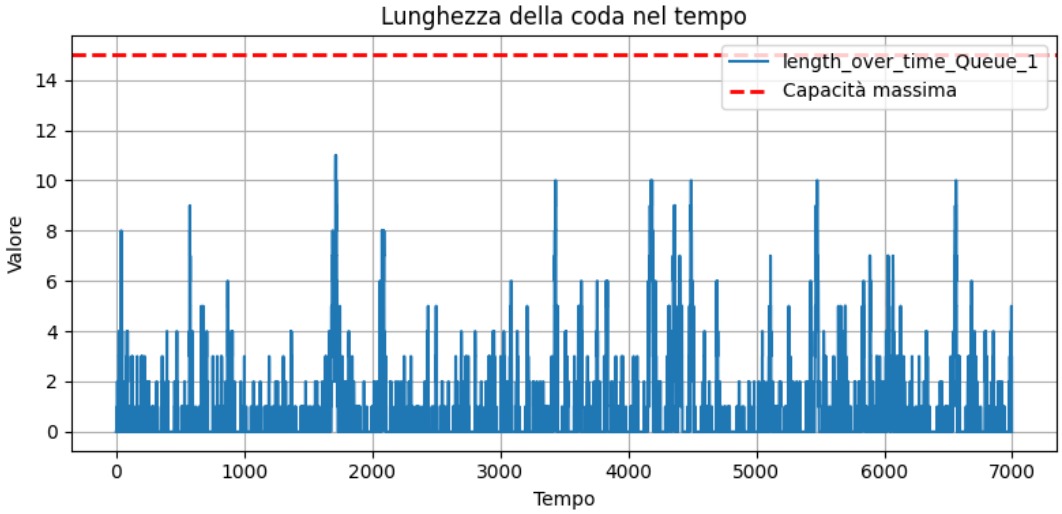

Figure 2.10: Lunghezza della coda nel tempo

In questo scenario è possibile analizzare il tempo di attesa in coda, che ci si aspetta rimanga limitato, a differenza di quanto accade nel caso transientes precedentemente osservato, in cui tende a crescere indefinitamente. Per studiare statisticamente tale tempo di attesa si svolgono molteplici repliche e di queste si scartano le prime osservazioni per rimuovere almeno in parte il periodo iniziale di transitorio. Per ognuna delle repliche si svolge la media campionaria e infine si formula l'intervallo di confidenza per la media della distribuzione del tempo di attesa a regime stazionario. In effetti le repliche sono indipendenti mentre l'ipotesi di normalità viene assunta.
Con $n\_sim = 40$ e $burn\_in = 500$ si ottiene:

Media campionaria	2.5357
Intervallo di confidenza al 95%	(2.4412, 2.6303)

Quindi ci si aspetta che, dati i parametri fissati inizialmente e per tempi sufficientemente lontani, mediamente il tempo di attesa in coda sarà di circa 2.5357.

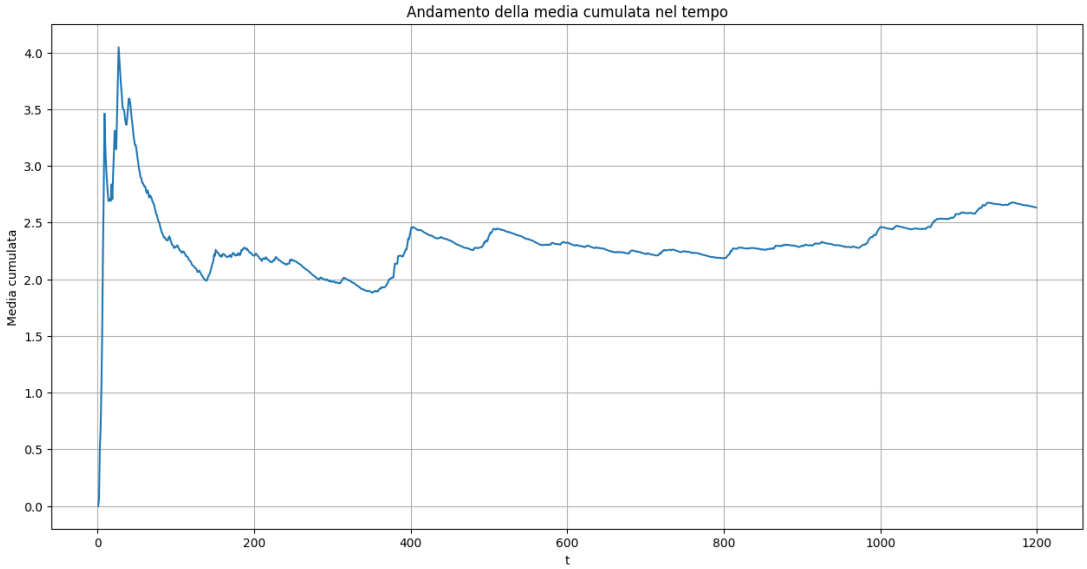

Figure 2.11: Cumulata per ogni tempo del tempo di permanenza in coda per la prima replica

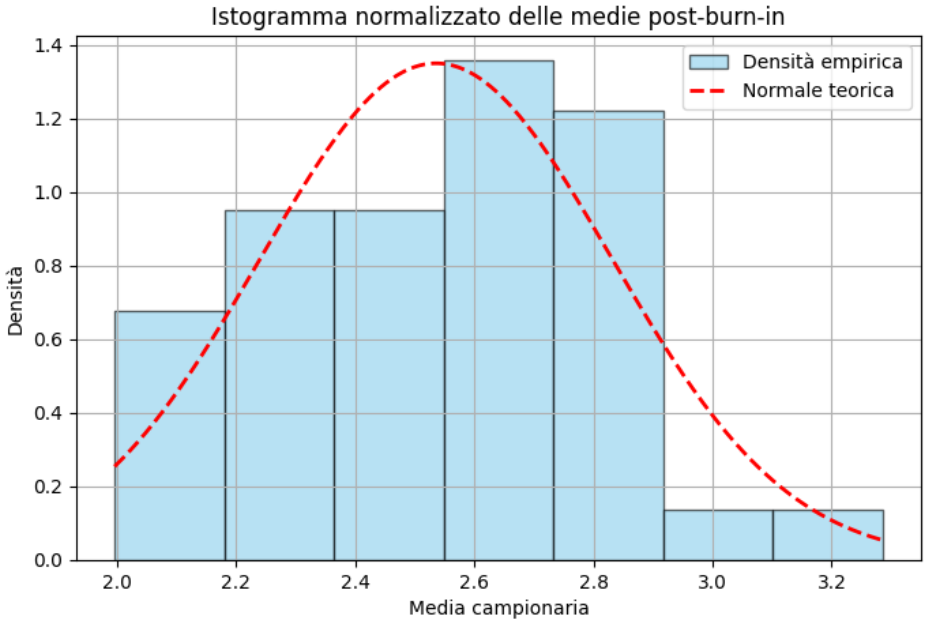

Figure 2.12: Istogramma tempo di attesa in coda confrontato con la distribuzione normale teorica sotto ipotesi nulla

Capitolo 3

3.1 Esempi di simulazione

3.1.1 Comportamento della coda dei pazienti

In questa sezione viene illustrata la dinamica della coda: i pazienti arrivano in seguito a un evento "Arrivo" e vengono disposti in coda in base alla loro priorità seguendo una logica FIFO con priorità.

Arrivo e inserimento in coda del paziente: Patient_7 con priorità P1

```
Queue Tail -> [P1 Patient_7] -> [P2 Patient_8] -> Queue Head
                                                  ↓
                                                  Doctors Processing Patients:
                                                  Doctor_1 [busy]: Patient_5
                                                  Doctor_2 [busy]: Patient_6
                                                  ↓
                                                  Patients transitioning from
                                                  doctors to nurses
                                                  Nurse_1 : Patient_3, Patient_4
                                                  Nurse_2 : Patient_2
                                                  Nurse_3 : No patient
                                                  ↓
                                                  Nurses Processing Patients:
                                                  Nurse_1 [busy]: Patient_1
                                                  Nurse_2 [reserved]: No patient
                                                  Nurse_3 [idle]: No patient
```

Arrivo e inserimento in coda del paziente Patient_9 con priorità P3

```
Queue Tail -> [P1 Patient_7] -> [P2 Patient_8] -> [P3 Patient_9] -> Queue Head
                                                                    ↓
                                                                    Doctors Processing Patients:
                                                                    Doctor_1 [busy]: Patient_5
                                                                    Doctor_2 [busy]: Patient_6
                                                                    ↓
                                                                    Patients transitioning
                                                                    from doctors to nurses
                                                                    Nurse_1 : Patient_3, Patient_4
                                                                    Nurse_2 : Patient_2
                                                                    Nurse_3 : No patient
                                                                    ↓
                                                                    Nurses Processing Patients:
                                                                    Nurse_1 [busy]: Patient_1
                                                                    Nurse_2 [reserved]: No patient
                                                                    Nurse_3 [idle]: No patient
```

3.1.2 Gestione dei pazienti da parte dei dottori

In questa sezione viene illustrata l'evoluzione degli eventi relativi alle risorse dottori: inizio processo sul paziente e fine processo sul paziente. *Doctor_1 è idle*

```
Queue Tail -> [P1 Patient_6] -> [P1 Patient_5] -> Queue Head
                                                  ↓
                                                  Doctors Processing Patients:
                                                  Doctor_1 [reserved]: Idle
                                                  Doctor_2 [busy]: Patient_2
                                                  ↓
                                                  Patients transitioning from doctors to nurses
                                                  Nurse_1 : Patient_1, Patient_3, Patient_4
                                                  Nurse_2 : No patient
                                                  Nurse_3 : No patient
                                                  ↓
                                                  Nurses Processing Patients:
                                                  Nurse_1 [reserved]: No patient
                                                  Nurse_2 [idle]: No patient
                                                  Nurse_3 [idle]: No patient
```

Inizio processo del Pazient_5 da parte del Doctor_1

```
Queue Tail -> [P1 Patient_6] -> Queue Head
                        ↓
                        Doctors Processing Patients:
                        Doctor_1 [busy]: Patient_5
                        Doctor_2 [busy]: Patient_2
                        ↓
                        Patients transitioning from doctors to nurses
                        Nurse_1 : Patient_1, Patient_3, Patient_4
                        Nurse_2 : No patient
                        Nurse_3 : No patient
                        ↓
                        Nurses Processing Patients:
                        Nurse_1 [reserved]: No patient
                        Nurse_2 [idle]: No patient
                        Nurse_3 [idle]: No patient
```

Fine processo del Pazient_2 da parte del Doctor_2

```
Queue Tail -> [P1 Patient_6] -> Queue Head
                                ↓
                                Doctors Processing Patients:
                                Doctor_1 [busy]: Patient_5
                                Doctor_2 [reserved]: Idle
                                ↓
                                Patients transitioning from doctors to nurses
                                Nurse_1 : Patient_1, Patient_3, Patient_4, Patient_2
                                Nurse_2 : No patient
                                Nurse_3 : No patient
                                ↓
                                Nurses Processing Patients:
                                Nurse_1 [reserved]: No patient
                                Nurse_2 [idle]: No patient
                                Nurse_3 [idle]: No patient
```

3.1.3 Gestione dei pazienti da parte degli infermieri

Similmente a quanto visto per i dottori si espongono gli eventi relativi alla risorsa "infermiere".

```
Queue Tail -> [P3 Patient_9] -> Queue Head
                                ↓
                                Doctors Processing Patients:
                                Doctor_1 [busy]: Patient_5
                                Doctor_2 [busy]: Patient_6
                                ↓
                                Patients transitioning from doctors to nurses
                                Nurse_1 : No patient
                                Nurse_2 : Patient_2
                                Nurse_3 : No patient
                                ↓
                                Nurses Processing Patients:
                                Nurse_1 [busy]: Patient_1, Patient_3, Patient_4
                                Nurse_2 [reserved]: No patient
                                Nurse_3 [idle]: No patient
```

Pazient_3 finisce di essere processato dal Nurse_1 ed esce dalla simulazione

```
Queue Tail -> [P3 Patient_9] -> Queue Head
                                ↓
                                Doctors Processing Patients:
                                Doctor_1 [busy]: Patient_5
                                Doctor_2 [busy]: Patient_6
                                ↓
                                Patients transitioning from doctors to nurses
                                Nurse_1 : No patient
                                Nurse_2 : Patient_2
                                Nurse_3 : No patient
                                ↓
                                Nurses Processing Patients:
                                Nurse_1 [idle]: Patient_1, Patient_4
                                Nurse_2 [reserved]: No patient
                                Nurse_3 [idle]: No patient
```

Pazient_1 finisce di essere processato dal Nurse_1 ed esce dalla simulazione

```
Queue Tail -> [P3 Patient_9] -> Queue Head
                                ↓
                                Doctors Processing Patients:
                                Doctor_1 [busy]: Patient_5
                                Doctor_2 [busy]: Patient_6
                                ↓
                                Patients transitioning from doctors to nurses
                                Nurse_1 : No patient
                                Nurse_2 : Patient_2
                                Nurse_3 : No patient
                                ↓
                                Nurses Processing Patients:
                                Nurse_1 [idle]: Patient_4
                                Nurse_2 [reserved]: No patient
                                Nurse_3 [idle]: No patient
```

3.1.4 "Guasti" e "ripristino" delle risorse (dottori e infermieri)

In questa sezione si mostrano degli esempi di eventi di "failure" delle risorse e il successivo evento di "recovery".

Evento di failure per Doctor_2: stato = failed

```
Queue Tail -> Queue Head
                  ↓
                  Doctors Processing Patients:
                  Doctor_1 [idle]: Patient_1
                  Doctor_2 [failed]: Idle
                  ↓
                  Patients transitioning from doctors to nurses
                  Nurse_1 : No patient
                  Nurse_2 : No patient
                  Nurse_3 : No patient

                  ↓
                  Nurses Processing Patients:
                  Nurse_1 [idle]: No patient
                  Nurse_2 [idle]: No patient
                  Nurse_3 [idle]: No patient
```

Evento di recovery per Doctor_2: stato = idle

```
Queue Tail -> Queue Head
                  ↓
                  Doctors Processing Patients:
                  Doctor_1 [idle]: Patient_1
                  Doctor_2 [idle]: Idle
                  ↓
                  Patients transitioning from doctors to nurses
                  Nurse_1 : No patient
                  Nurse_2 : No patient
                  Nurse_3 : No patient
                  ↓
                  Nurses Processing Patients:
                  Nurse_1 [idle]: No patient
                  Nurse_2 [idle]: No patient
                  Nurse_3 [idle]: No patient
```

Codice

Codice Python presente nella repository:
https://github.com/cristina210/hwBA

Bibliografia

M. Roma, *Simulazione*, Università degli Studi di Roma "La Sapienza".
Disponibile su: `http://www.diag.uniroma1.it/~roma/didattica/SSS13-14/parteE.pdf`

Salabim discrete event simulation package.
Disponibile su: `https://www.salabim.org/`

Integrated Healthcare Timetabling Competition 2024.
Disponibile su: `https://ihtc2024.github.io/`

Arianna Alfieri, *Slide del corso di Simulazione dei Sistemi Gestionali*, Politecnico di Torino.