Progetto Sistemi Complessi: Modelli e Simulazione

Progetto per il corso di Sistemi Complessi: Modelli e Simulazione aa.2020/21

Artemisia Sarteschi 829677

Nicola Armas 816398

Indice

Capitolo 1

Introduzione

Il tema di questo progetto è basato sostanzialmente sullo studio esplorativo del software AnyLogic, utilizzando un problema del mondo reale come quello di ottimizzazione delle tempistiche di un pronto soccorso.

Siamo quindi andati valutare come questo strumento si comportasse nel modellare questa situazione per valutarne le funzionalità utilizzando un approccio ad agenti. All'interno di un pronto soccorso gravitano diverse figure e i pazienti possono arrivare in condizioni molto diverse.

Abbiamo scelto di utilizzare AnyLogic per conoscere un software differente da quelli proposti nel corso e capire se, essendo fornito ed utilizzato a livello commerciale, potesse fornire degli strumenti migliori per rendersi potenzialmente comprensibile ad un gruppo di persone interessate all'argomento ma non ferrate sulla simulazione.

Nei prossimi capitoli di questo documento descriveremo il software scelto, con un approfondimento sul modello ad agenti realizzato e simulato grazie ad esso; infine, analizzeremo i risultati ottenuti da alcune sperimentazioni eseguite, al fine di confrontarli con quello che è lo stato dell'arte attuale, ovvero gli articoli su cui ci siamo basati per fondare il nostro modello.

All'interno del repository è possibile scaricare il file della simulazione in AnyLogic e visualizzare i video che ne mostrano il funzionamento.

Capitolo 2

Stato dell'Arte

In un contesto complesso, dinamico e spesso riservato come quello di un pronto soccorso, può risultare molto arduo ottenere informazioni volte a realizzarne una rappresentazione, seppur in forma modellare, che si avvicini il più possibile al sistema reale.

Per questo motivo è spesso utile per un modellista basarsi su quello che è chiamato lo *"stato dell'arte"*: articoli scientifici, libri e strumenti che descrivono il massimo livello di sviluppo nella ricerca in un determinato ambito.

Nel caso in esame, abbiamo deciso di basarci sul lavoro di diversi ricercatori che avessero già approfondito il tema, tramite progetti riportati sotto forma di paper scientifici.

2.1 Paper

Gli articoli scientifici su cui si è basato il nostro modello sono i seguenti:

- García et al., «Reducing time in an emergency room via a fast-track»
- Espinoza et al., «Real-time simulation as a way to improve daily operations in an Emergency Room»

- Stainsby, Taboada e Luque, «Towards an Agent-Based Simulation of Hospital Emergency Departments»
- Wang, «An agent-based simulation for workflow in Emergency Department»

Il lavoro svolto in ciascuna di queste ricerche è servito principalmente a non dover sviluppare un nuovo modello partendo da zero: è risultato infatti difficile ottenere informazioni direttamente sul campo, a causa della riservatezza caratteristica dell'ambiente in oggetto.

In particolare, i paper *García et al., «Reducing time in an emergency room via a fast-track»*, *Stainsby, Taboada e Luque, «Towards an Agent-Based Simulation of Hospital Emergency Departments»* e *Wang, «An agent-based simulation for workflow in Emergency Department»* sono risultati utili a definire un workflow (vedi *Capitolo 4*) che potesse dare forma all'interazione degli agenti con l'ambiente, ai percorsi seguiti e alle risorse utilizzate.

Il paper *Espinoza et al., «Real-time simulation as a way to improve daily operations in an Emergency Room»* è servito a definire delle metriche per effettuare sperimentazioni e a confrontarne i risultati con quelli ottenuti nei lavori precedenti (vedi *Capitolo 5*).

Infine, consultando nuovamente il paper *García et al., «Reducing time in an emergency room via a fast-track»* e confrontandolo con le informazioni riportate sul sito della *Regione Lombardia* (per un maggiore adattamento a quello che è il modello locale), è risultato possibile ottenere una rappresentazione tabellare della gravità tipica dei pazienti ed annessa priorità.

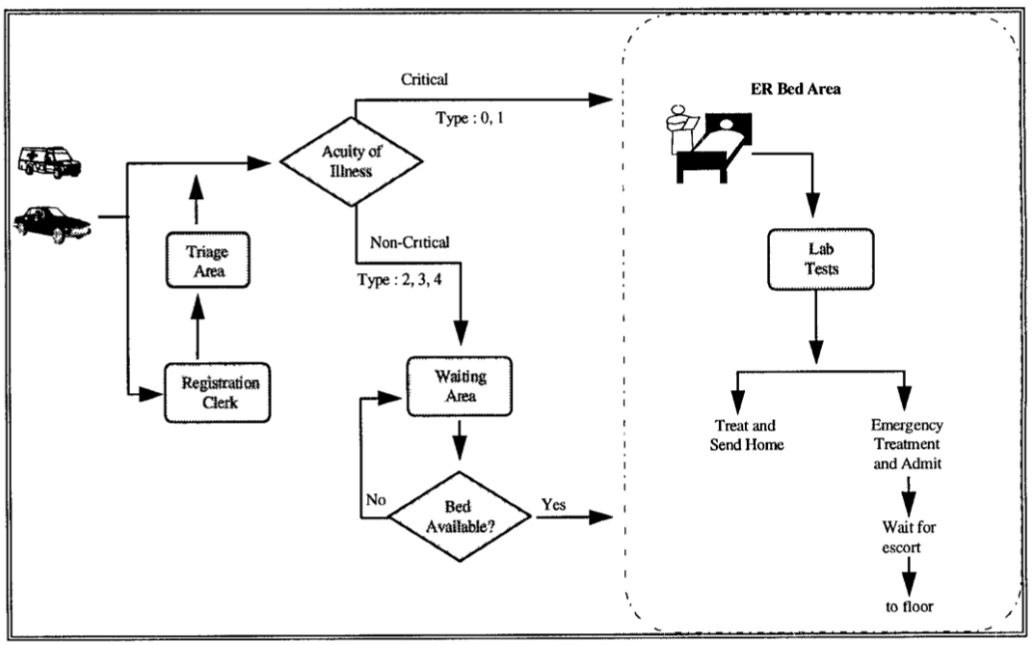

Figura 2.1: Workflow utilizzato nel paper *García et al.*, «*Reducing time in an emergency room via a fast-track*»

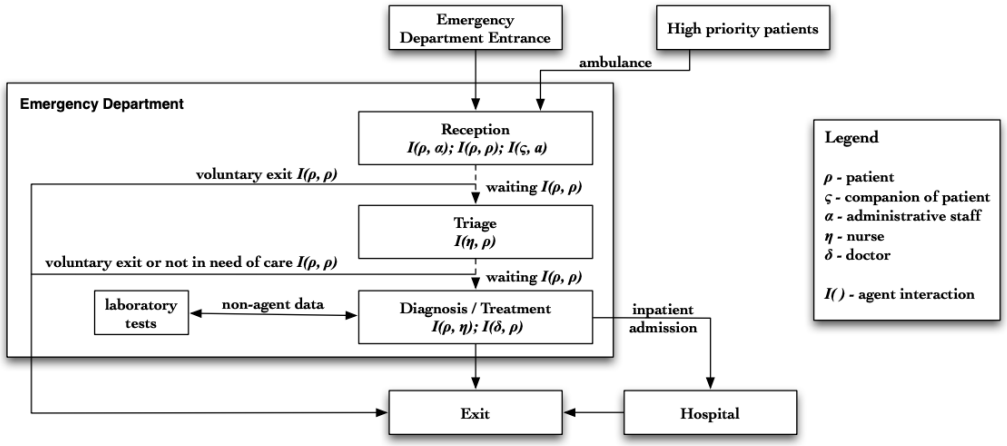

Figura 2.2: Workflow utilizzato nel paper *Stainsby, Taboada e Luque*, «*Towards an Agent-Based Simulation of Hospital Emergency Departments*»

Table 1: Patients Categories

Patient Category	Condition
Ambulance - Category 0	Cardiac, Hemorrhage, Respiratory, Neural, ortho, Trauma, Psyche, etc.
Emergent - Category 1	Cardiac, Hemorrhage, Respiratory, Neural, ortho, Trauma, Psyche, etc.
Urgent -Category 2	Abdominal pain, Emotionally disrupted behavior, Acute back pain, etc.
Non-Urgent -Category 3	Chronic Headache, Nerves, Sprains, Eye Infection, Abrasion, etc.
Stable -Category 4	Wound checks

Figura 2.3: Codici di priorità utilizzati nel paper *Stainsby, Taboada e Luque, «Towards an Agent-Based Simulation of Hospital Emergency Departments»*

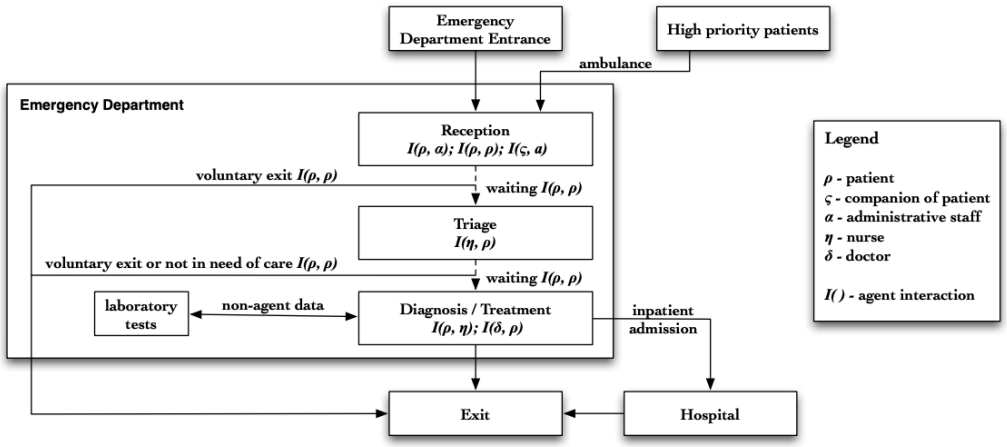

Figura 2.4: Codici di priorità reperibili dal sito della *Regione Lombardia*

2.2 Strumenti

Per completare l'approfondimento relativo a quello che è lo stato dell'arte, in questa sezione verranno descritti brevemente gli strumenti utilizzati in ciascuno dei paper consultati.

È doveroso riportare che solo in uno di questi lavori non è stata effettuata una simulazione né sono stati adoperati strumenti informatici di supporto: in particolare, nel paper *Stainsby, Taboada e Luque, «Towards an Agent-Based Simulation of Hospital Emergency Departments»* è stato realizzato unicamente il modello matematico per il raggiungimento di un modello ad agenti simulabile (il titolo recita "*towards*", per l'appunto).

2.2.1 Garcia et al., 1995: *SIMAN*

Lo strumento utilizzato in questo paper per simulare il modello è un linguaggio chiamato SIMAN (SIMulation ANalysis), presentato nel 1982 nell'articolo *Pegden e Ham, «Simulation of Manufacturing Systems Using SIMAN»*.

È utilizzato per modellare sistemi combinati discreto-continui, tramite l'impiego di diversi approcci: per il discreto, si orienta verso l'utilizzo di *processi* oppure *eventi*. La parte continua è modellata tramite *algebra, differenze* o *equazioni differenziali*.

2.2.2 Espinoza et al., 2014: *FlexSim HC*

FlexSim HC (HealthCare) è un ambiente di simulazione sviluppato da FlexSim, adoperato a livello aziendale nell'ambito della salute. In maniera simile ad AnyLogic, permette di modellare un ambiente 3D ed un workflow per orientare pazienti e staff all'interno del detto ambiente.

Sul sito Internet dedicato è possibile scaricarne una versione di prova gratuita per testarne le funzionalità.

2.2.3 Wang, 2009: *NetLogo*

NetLogo è un linguaggio di programmazione e IDE per la modellazione basata su agenti. Presentato nel 1999, è particolarmente efficace per l'esplorazione di *fenomeni emergenti*, di cui include una vasta libreria di modelli al suo interno.

NetLogo è open-source e scaricabile gratuitamente dal suo sito Internet.

Capitolo 3

AnyLogic

AnyLogic è uno strumento per creare modelli di simulazione sviluppato da The AnyLogic Company. Viene utilizzato nel contesto industriale in molti ambiti, come ad esempio:

- Healthcare
- Manifattura
- Pedoni e traffico
- Trasporti
- Catene di montaggio

Questo software include un linguaggio di modellazione grafico che si interfaccia con il linguaggio Java, offrendo all'utente svariate opzioni di personalizzazione e impostazione.

L'utilizzo di AnyLogic per questo progetto è stato considerato e successivamente messo in atto per diversi motivi: primo fra tutti, la varietà degli ambiti di utilizzo possibili, molto elevata e adatta al caso in esame.
Un altro motivo importante è la possibilità di simulare modelli in ambienti anche tridimensionali, che rende questo software user-friendly anche per neofiti, obiettivo raggiunto grazie anche alla simulazione a blocchi, che permette una rappresentazione dei modelli chiara e funzionale.

In ultimo, oltre ai vantaggi sopra elencati, la possibilità di ottenere analisi statistiche in tempo reale su vaste popolazioni di agenti e la facoltà di potersi interfacciare con molti sistemi di gestione dati esterni hanno fatto ricadere la scelta su questo potente strumento.

3.1 Edizioni di AnyLogic e scelta della nostra

AnyLogic offre una versione di prova, per un periodo di 30 giorni, delle sue versioni non gratuite (Professional e University Researcher).

La scelta di usare la versione di prova dell'edizione Professional per questo progetto è dettata dalle limitazioni della versione gratuita PLE (Personal Learning Edition), in particolare la limitazione del tempo di simulazione a 60 minuti. Nonostante si sia deciso di utilizzare un'edizione Professional, tuttavia, la sua versione di prova presenta a sua volta delle grosse limitazioni: un esempio è quello di avere un limite massimo di 35 blocchi per workflow. Per il presente progetto, ciò è particolarmente arginante poiché lo scopo è quello di simulare un ambiente con molte interazioni e spostamenti.

Alcuni possibili espedienti per risolvere questo problema possono essere: la divisione degli ambienti in più sotto-ambienti (progetti AnyLogic complementari), l'utilizzo della versione PLE con scala adatta delle tempistiche oppure, più drasticamente, l'acquisto della versione Professional completa.

Figura 3.1: Versioni di AnyLogic

3.2 Piattaforma Cloud

La piattaforma Cloud viene principalmente utilizzata per condividere i modelli realizzati con altri utenti oppure, in caso di contesti aziendali, con i propri clienti e dipendenti. Permette inoltre di integrare dati operazionali e di creare dashboard altamente personalizzabili in base al contesto (analisi, sperimentazione, gestione, ...).
Una caratteristica molto utile è quella di poter effettuare simulazioni distribuite: si possono partizionare modelli pesanti a livello di risorse in modelli più piccoli e, tramite l'utilizzo di API (JS, Java, Python), agevolare lo scambio di dati e la sincronizzazione tra essi per un'esecuzione più veloce. AnyLogic Cloud è offerto in due soluzioni: Public Cloud e Private Cloud.

Mentre la prima è online, gratuita e disponibile a chiunque per il caricamento e la condivisione di modelli, la seconda opzione è un'infrastruttura vera e propria per organizzazioni e aziende, che viene installata in-house presso un data center o provider PaaS. Ciò permette di avere un controllo completo su dati e operazioni, integrando l'offerta di AnyLogic direttamente all'interno della propria azienda.

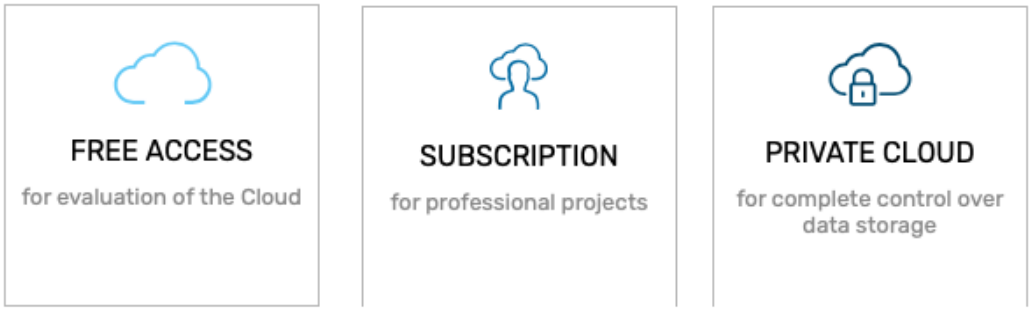

Figura 3.2: Versioni del Cloud

3.3 AI e ML

AnyLogic fornisce un ambiente altamente strutturato per sviluppare simulazioni di ambienti reali con differenti approcci *(system dynamics*, *discrete event simulation* e *agent based*).
In riferimento all'ultimo approccio citato, ovvero *agent based*, la piattaforma fornisce la possibilità di creare e progettare agenti capaci di imparare dalla loro esperienza: in particolare possono essere modellati l'intera gamma di agenti con comportamento cooperativo, competitivo o gerarchico. Abbiamo quindi la possibilità di impiegare la piattaforma per addestrare l'ambiente impiegato negli schemi di reinforcement learning multi agente. Inoltre, AnyLogic Company mette a disposizione una piattaforma Cloud che può fornire una esecuzione parallela del modello, una facilità di condivisione e una RESTful API. AnyLogic appoggiandosi e/o collaborando con altre aziende fornisce al cliente tre possibilità per inserire l'intelligenza artificiale e più in generale varie forme di machine learning all'interno dei loro progetti.

3.3.1 Project Bonsai

Project Bonsai è un progetto nato dalla collaborazione tra The AnyLogic Company e Microsoft. L'obbiettivo di Bonsai è permettere agli esperti di dominio, anche privi di una conoscenza pregressa di *Intelligenza Artificiale* (AI) di incorporare la loro esperienza attraverso il *Machine Teaching* (MT) direttamente nei modelli e con l'aiuto della potenza del *Deep Reinforcement Learning* (DRL) ottimizzare e automatizzare i sistemi reali.
La collaborazione tra le due aziende si realizza con un connettore che permette di utilizzare i modelli sviluppati in AnyLogic come simulatori della piattaforma Bonsai. Ovvero per semplificare la conversione di questi modelli di simulazione (AnyLogic) in simulatori (Bonsai) è stato creato "RLExperiment" che permette di esportare un modello con tutti i meccanismi integrati necessari per comunicare senza problemi con la piattaforma Project Bonsai. Nel caso in cui volessimo utilizzare questo strumento i passi sono i seguenti e nel caso si volesse un esempio di approfondimento lo si può trovare al seguente link.

3.3.1.1 Creare un Brain

1. Creare un account o collegarsi Bonsai
2. Cliccare su `Create brain` selezionare `Empty brain`.
3. Nominarlo con un nome a scelta.

A questo punto visualizziamo la schermata in figura 3.3: se già possediamo un file `.ink` possiamo inserirlo all'interno della finestra `Teach` oppure scrivere in linguaggio proprietario *Inkling* il codice per il MT, questo codice verrà anche chiamato *curriculum*.

Nella parte destra della schermata (*Graphing panel*) è possibile avere anche una rappresentazione grafica interattiva del processo di apprendimento iterativo definito dal codice Inkling.

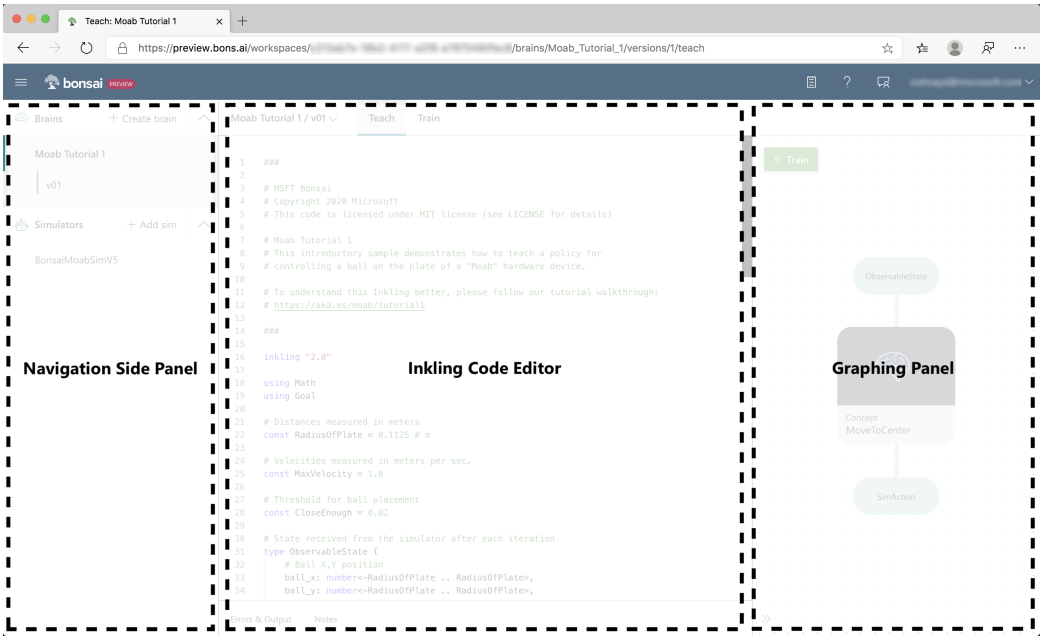

Figura 3.3: Interfaccia di Bonsai

3.3.1.2 Esecuzione del Modello

A questo punto per eseguire il modello si può scegliere di farlo con un animazione: click destro su `AnimatedExperiment`, poi `Run` e dopo si visualizzerà in console la riuscita della registrazione alla piattaforma Bonsai tornare nella schermata dove è stato creato il brain. A questo punto si fa partire l'effettivo `train` del modello (azione diretta nel caso si sia scelto di eseguire un esperimento non animato), si vedrà quindi comparire un simulatore con il nome corrispondente su cui bisognerà cliccare.
Nel caso fosse la prima volta che il brain viene eseguito potrebbero essere necessari diversi minuti per generare i parametri adatti e connettersi al simulatore. Si può far addestrare il brain per un lasso di tempo variabile, ma viene consigliato di tenere le iterazioni sotto le 1000, quando si ritiene siano sufficienti si cliccherà su `Stop Training`.

3.3.1.3 Esportare e scalare il modello di Anylogic

Il modello creato in AnyLogic necessita di essere esportato e compresso (`.zip`) per poter essere caricato su Bonsai.
Nel caso in cui si utilizzi *AnyLogic Professional* sarà sufficiente selezionare nella barra dei menù `File > Export...> standalone Java application`, se si incontrassero delle difficoltà è disponibile il supporto di Anylogic.

Dopo questo passaggio è necessario importare il modello di Anylogic all'interno di Bonsai e per farlo sarà sufficiente selezionare su `Add sim` sull'interfaccia di Bonsai. Si aprirà quindi la finestra di dialogo in figura 3.3.1.3 dove sarà sufficiente cliccare su AnyLogic, caricare il file `.zip` precedentemente creato e selezionare `Create simulator`. Il simulatore appena creato verrà visualizzato nella sezione `Simulators`.

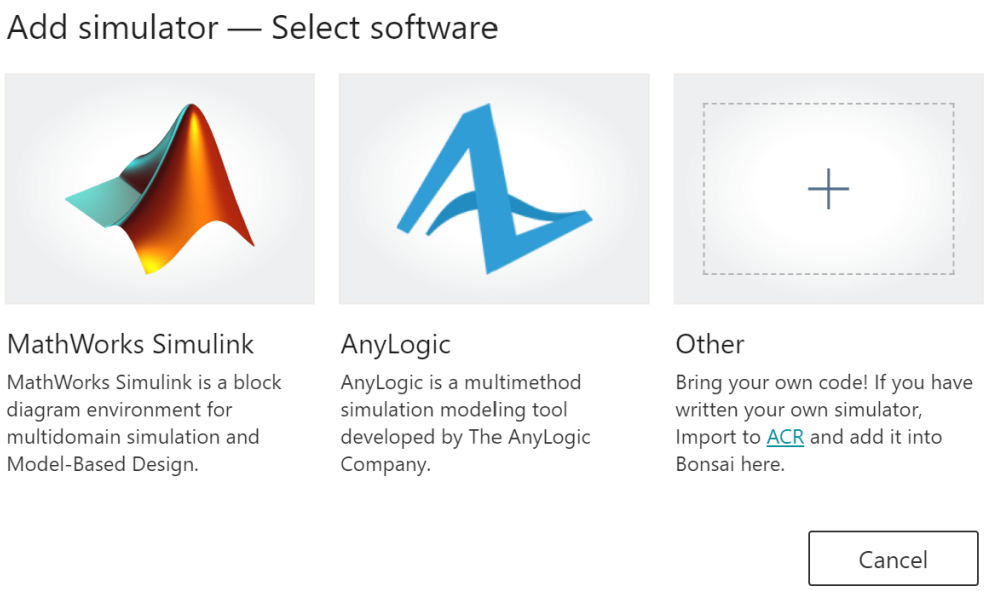

Figura 3.4: Finestra per la scelta del simulatore

All'interno della scheda `Teach`, nella definizione del modello, è necessario aggiungere le seguenti linee di codice:

```
simulator Simulator(action: Action, config: SimConfig): SimState {
        package "<simulator_name_from_upload>"
}
```

Si potrà quindi selezionare `Train` e dopo alcuni minuti si potranno vedere diversi simulatori collegati che addestrano il Brain.

3.3.2 H20.ai

AnyLogic è partner di H2O.ai, una delle principali piattaforme di automatic machine learning. L'AutoML è una branca del ML che permette di applicare le tecniche di Ml ai problemi del mondo reale usando l'automazione. In particolare, automatizza la selezione, la composizione e la parametrizzazione dei modelli di machine learning. L'obiettivo di integrare questi due sistemi è fornire la possibilità ai data scientist di testare le loro soluzioni in uno spazio sicuro e ai modellatori di accedere a una maggior quantità di input data-driven.

Le informazione per come utilizzare questa piattaforma con AnyLogic non sono accessibili al pubblico ma solo mettendosi in contatto con l'azienda. Le indicazioni che si possono ricavare dal sito di Anylogic è la possibilità di scaricare da *H2O.ai* il modello già addestrato per inserirlo all'interno delle simulazioni di AnyLogic ed utilizzarlo come fosse una funzione in grado di restituire delle predizioni basate sugl'input che vengono passati dinamicamente alla simulazione durante l'esecuzione.

3.3.3 Pathmind

La terza possibilità per inserire degli elementi di AI in AnyLogic è *Pathmind*. Pathmind è una piattaforma SaaS che permette di avere accesso a delle tecniche di deep reinforcement learning e cloud computation per applicarle a degli scenari del mondo reale senza la necessità di avere esperienza di data science.

Pathmind a differenza dei servizi precedentemente presentati si va ad inserire all'interno di AnyLogic con Pathmind Helper attraverso cui è possibile agire sul modello.

Il processo da seguire per utilizzare Pathmind è il seguente:

1. Creare un modello in AnyLogic che simuli un problema del mondo reale.
2. Usare Pathmind Helper per aggiungere il reinforcement learning all'interno della simulazione di AnyLogic.
3. Caricare la simulazione di AnyLogic in Pathmind, nel cui cloud avviene l'addestramento.
4. Scaricare i risultati ottenuti e validarli in AnyLogic.

E' possibile trovare informazioni aggiuntive su Pathmind nel help mantenuto dall'azienda.

3.3.3.1 Pathmind Helper

All'interno di ogni agente che vogliamo dotare di AI abbiamo bisogno di creare un diagramma di stato e inserire Pathmind Helper come nell'esempio in figura 3.5.

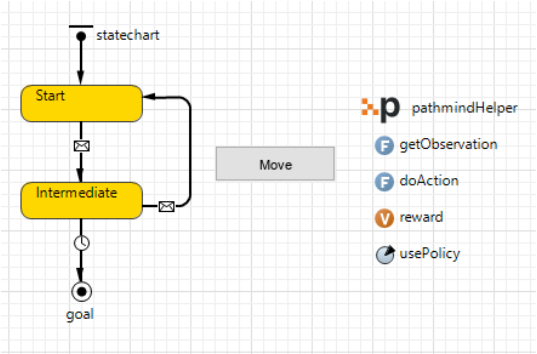

Figura 3.5: Agente tipo

Aprendo Pathmind Helper possiamo controllare i seguenti elementi del RL:

- **Number of Agents**: il numero di agenti "controllati" presenti nel modello.
- **Observations**: contengono tutte le informazioni sullo stato corrente dell'ambiente.
- **Metrics**: sono le metriche che vengono utilizzate all'interno della funzione di *reward* per determinare se un'azione è stata buona o cattiva, a volte contiene anche il costo di essa.
- **Actions**: definisce tutte le azioni che un agente può svolgere.
- **Done**: dal momento che tutte le simulazioni hanno necessità di un punto di arrivo, questo viene settato o dopo un lasso di tempo prestabilito o al raggiungimento di una determinata condizione.
- **Event Trigger**: indicano a Pathmind quando inizia l'azione successiva, dopo un lasso di tempo o al verificarsi di una determinata condizione.

All'interno di Pathmind Helper è possibile testare il modello, prima di esportarlo, spuntando la `Debug Mode` che permetterà di aprire il `Developer Panel` al cui interno, in caso in cui fosse tutto impostato correttamente, verrà stampata a video ogni azione che l'agente performa (figura 3.6).

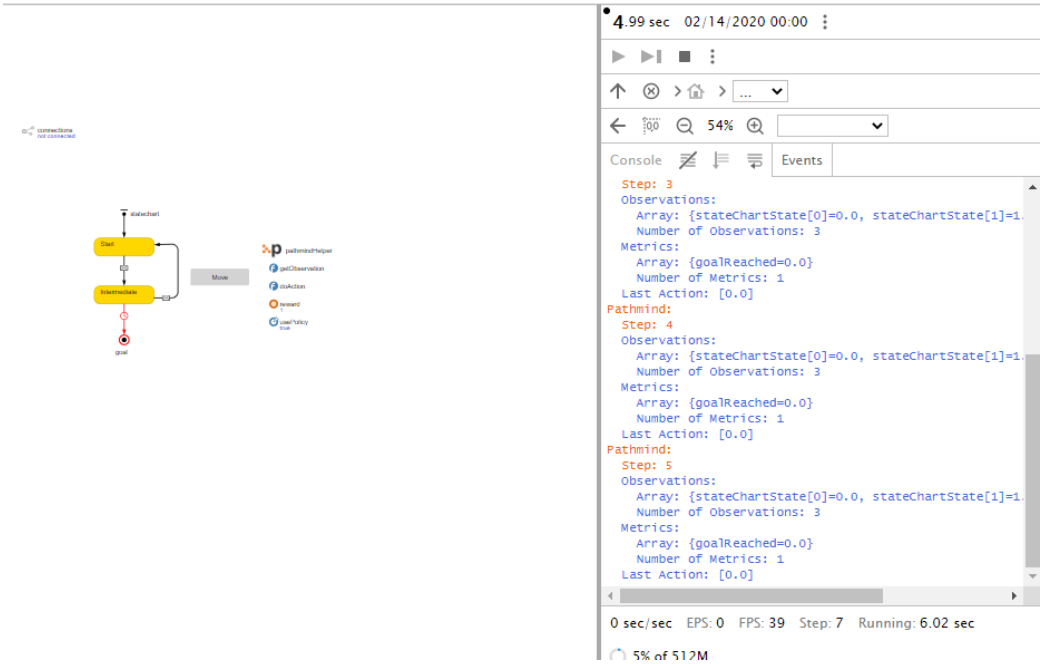

Figura 3.6: Agente tipo

Dopo che il modello è stato configurato è possibile esportarlo nell'applicazione web di Pathmind per l'addestramento. All'interno di questa web app è possibile configurare la funzione di reward ed addestrare con essa il modello in circa 10 minuti.

Al termine dell'addestramento è possibile esportare la policy risultante e inserirla in Anylogic all'interno del Pathmind helper sotto `Policy File`. Seguendo la policy inserita l'agente convergerà all'obbiettivo nel minor numero di passi possibile.

Capitolo 4

Modello

4.1 Ambiente costruito

Prima di andare a definire gli agenti che operano nel nostro modello, abbiamo raccolto informazioni sulla possibile struttura dell'ambiente di un pronto soccorso, al fine di costruire una piantina che avesse le caratteristiche generiche di un pronto soccorso comune.

In particolare, le stanze che abbiamo infine deciso di realizzare sono le seguenti:

- Sala d'attesa
- Accettazione: 2 file
- Triage: 2 stanze
- Sala visita: 4 stanze
- Chirurgia: 2 sale operatorie
- Degenza breve: 1 con posti letto normali, 1 con posti letto di emergenza

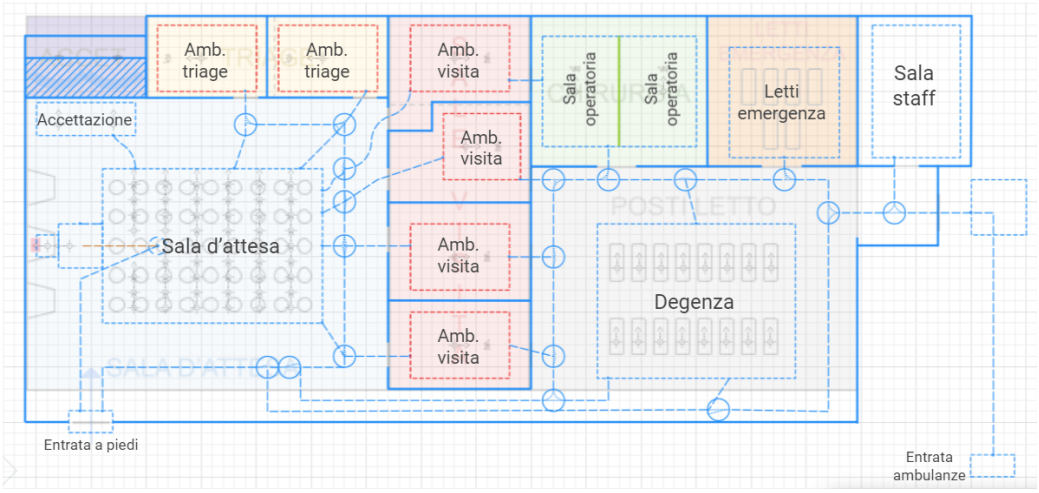

Figura 4.1: Layout 2D dell'ambiente costruito

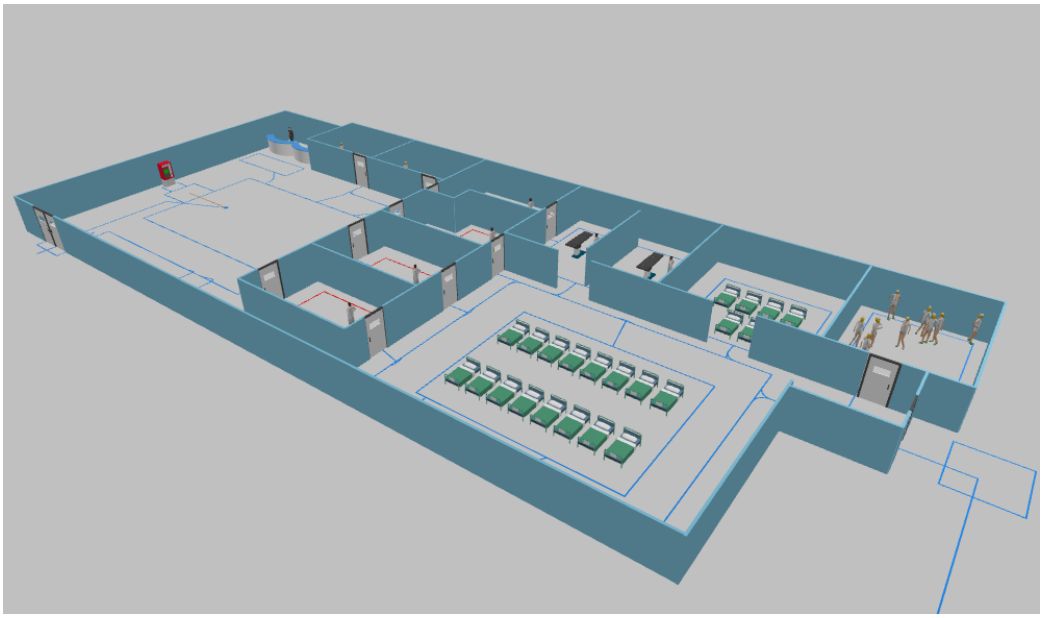

Figura 4.2: Layout 3D dell'ambiente costruito

4.2 Agenti realizzati

Gli agenti che abbiamo realizzato per impostare il funzionamento del modello sono Patient, Ambulance, AmbulancePatient e Nurse. Mentre i primi tre sono generati in base ad uno specifico tasso di input, gli infermieri appartengono ad una pool di risorse per cui vengono effettuate delle richieste all'occorrenza.

4.2.1 Patient

Questo agente è il più importante del modello: rappresenta infatti il paziente che si reca al pronto soccorso. E' caratterizzato da alcuni parametri che ne definiscono il comportamento all'interno del workflow:

appearance Definisce l'agente a livello puramente estetico tramite un'estrazione da una distribuzione uniforme discreta, con tre possibili opzioni.

priority Definisce tramite un numero intero la priorità di ogni paziente, data la gravità della sua situazione e rispetto agli altri pazienti che sono arrivati; viene impostato a zero nel momento della creazione dell'agente e verrà modificato durante il triage, sulla base del parametro *sickness*.

sickness È un parametro di tipo Illness, ovvero una Option List che contiene al suo interno le tipologie di urgenze (rosso, giallo, verde o bianco) e viene istanziato secondo una distribuzione ad hoc (*IllnessDistr*), che segue un andamento pressoché realistico:

- Codice rosso: 10% dei pazienti
- Codice giallo: 25% dei pazienti
- Codice verde: 40% dei pazienti
- Codice rosso: 25% dei pazienti

surgery È un parametro di tipo NeedSurgery, ovvero una Option List (con opzioni *yes* oppure *no*) che descrive il bisogno o meno di ricorrere alla chirurgia per un determinato paziente.

bed È un parametro di tipo NeedBed, ovvero una Option List (con opzioni *yesB* oppure *noB*) che viene utilizzato per segnalare la necessità di un letto per il paziente.

hospital È un parametro booleano che viene impostato a *true* quando il periodo di degenza supera le 20 ore: in questo caso il paziente viene immediatamente trasferito in ospedale.

newVisit È un parametro booleano che viene impostato a *true* quando un paziente ha bisogno di un'ulteriore visita in seguito ad un periodo in degenza.

leave, leaveProb, leaveEst() Rispettivamente un parametro, una variabile ed una funzione che permettono di implementare il meccanismo di noia durante l'attesa e conseguente uscita volontaria dal PS (meccanismo descritto nel seguito).

cameByAmb È un parametro booleano che definisce se il paziente è arrivato in ambulanza o meno. Tale valore è importante poiché permette di definire la transizione dallo stato *initial* allo stato *other*, possibile solamente per i pazienti che arrivano in ambulanza e non hanno dunque bisogno di aspettare in sala d'attesa.

entryTime, waitingEntryTime, waitingTime Sono variabili di tipo double che permettono di definire il tempo (inteso come timestamp) di entrata del paziente, di entrata nello stato *Waiting* e il tempo che passa in quest'ultimo. Sono utili perlopiù per definire le statistiche descritte nel capitolo 5.

4.2.1.1 Meccanismo di uscita volontaria

All'interno dei pronto soccorso, date le elevate attese, molti pazienti scelgono di andarsene prematuramente. Essendo quindi una situazione possibile ed ampiamente frequente abbiamo deciso di dotare ogni agente di questa peculiarità che può portarli o meno ad abbandonare prematuramente l'ospedale, sfruttando la possibilità offerta da AnyLogic che permette di implementare diagrammi di stato all'interno delle classi che rappresentano gli agenti.

In questi diagrammi, ogni blocco identifica uno stato in cui l'agente può trovarsi: nello specifico, un agente di tipo Patient si trova inizialmente nello stato Initial, da cui passa in Waiting a condizione che l'agente si trovi nella sala d'attesa.

L'agente rimane in questo stato finché non viene visitato oppure se viene chiamato (accettazione, triage, visita, che gli permettono di passare nello stato Other); se invece non viene chiamato entro 60 minuti, passa nello stato WantsToLeave, al cui ingresso viene valutata la decisione di andarsene (parametro leave) tramite la funzione leaveEst(), con una probabilità del 15% (definita tramite la costante leaveProb).

Nel caso in cui l'agente decidesse di non andarsene, dopo altri 30 minuti, se ancora in attesa, invocherà nuovamente la funzione di decisione.

Nello stato Leaving il paziente si accingerà a muoversi verso la porta di uscita; nello stato Left, eliminerà se stesso dall'ambiente.

C'è infine la possibilità che un paziente precedentemente nello stato Other ritorni in WantsToLeave nel caso in cui si trovasse ancora una volta in sala d'attesa.

4.2.2 Ambulance e AmbulancePatient

Ambulance e AmbulancePatient sono le due classi di agenti definite per rappresentare i pazienti che giungono al pronto soccorso in ambulanza. Nell'ambiente hanno un'entrata prioritaria che permette loro di saltare le fasi di accettazione, triage e visita.

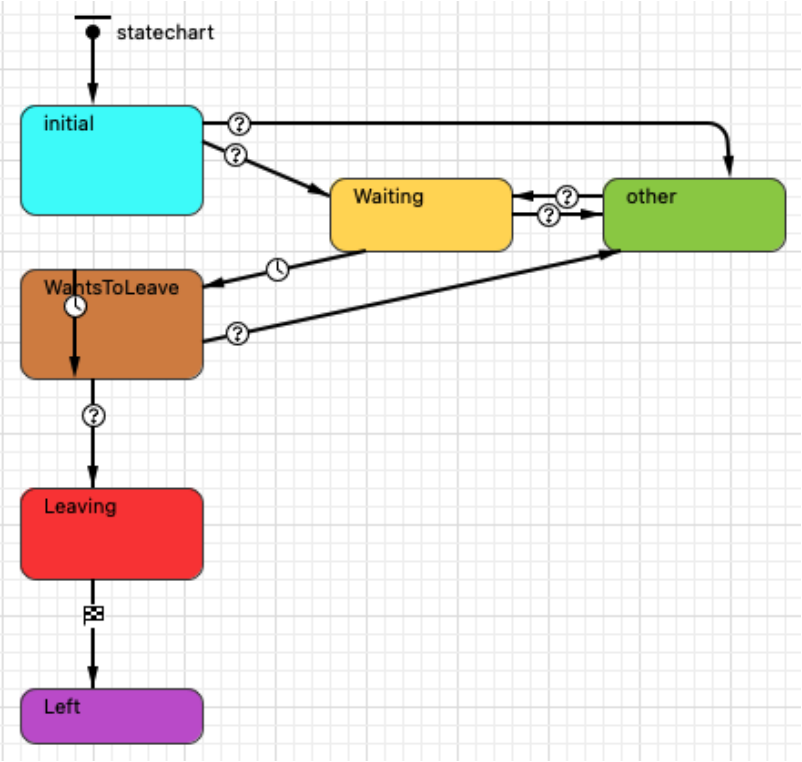

Figura 4.3: *Statechart* dell'agente Patient

Ambulance è definito ai fini del modello solo come mezzo di trasporto "contenitore" per i pazienti di tipo AmbulancePatient. Viene generato uno di questi agenti all'occorrenza, ovvero quando un agente di tipo AmbulancePatient ha bisogno di essere trasportato in ambulanza fino al pronto soccorso.

AmbulancePatient è un'estensione della classe che definisce l'agente Patient e mantiene dunque i suoi parametri descritti in precedenza; per questo agente viene però impostato di default il bisogno di un letto per la degenza, mentre gli altri parametri sono impostati come segue nel blocco di generazione:

- Priority: il valore viene generato da una distribuzione uniforme discreta con valori compresi tra 6 e 10.
- Surgery: viene impostato un valore casuale tra yes e no.
- Sickness: per gli arrivi in ambulanza viene sempre impostato a Red.

4.2.3 Nurse

È l'agente che rappresenta gli infermieri: questi sono generati nel modello in un numero variabile in base all'impostazione del pool di risorse che li contiene. La classe che definisce l'agente non presenta parametri, funzioni o stati particolari ma serve a definire solo a livello estetico gli infermieri. Questi agenti sono utilizzati nel workflow del sistema esclusivamente tramite blocchi di tipo *seize* e *release*, che permettono di "riservarli" per un tempo utile ad eseguire una determinata azione (nel nostro caso, accompagnare i pazienti in varie stanze qualora ce ne sia bisogno).

4.3 Workflow

Il workflow è la parte focale della costruzione del sistema tramite il software AnyLogic e viene mostrato interamente nella figura 4.4.
Permette infatti di far interagire agenti ed ambiente secondo determinate regole e condizioni, che possono essere specificate tramite la modellazione a blocchi e, per ciascun blocco, estese con linguaggio Java.

In questa sezione verrà descritta ogni fase del workflow realizzato, al fine di far comprendere al lettore non solo il funzionamento del modello in sè, ma anche quali sono alcune delle funzionalità offerte da Anylogic.

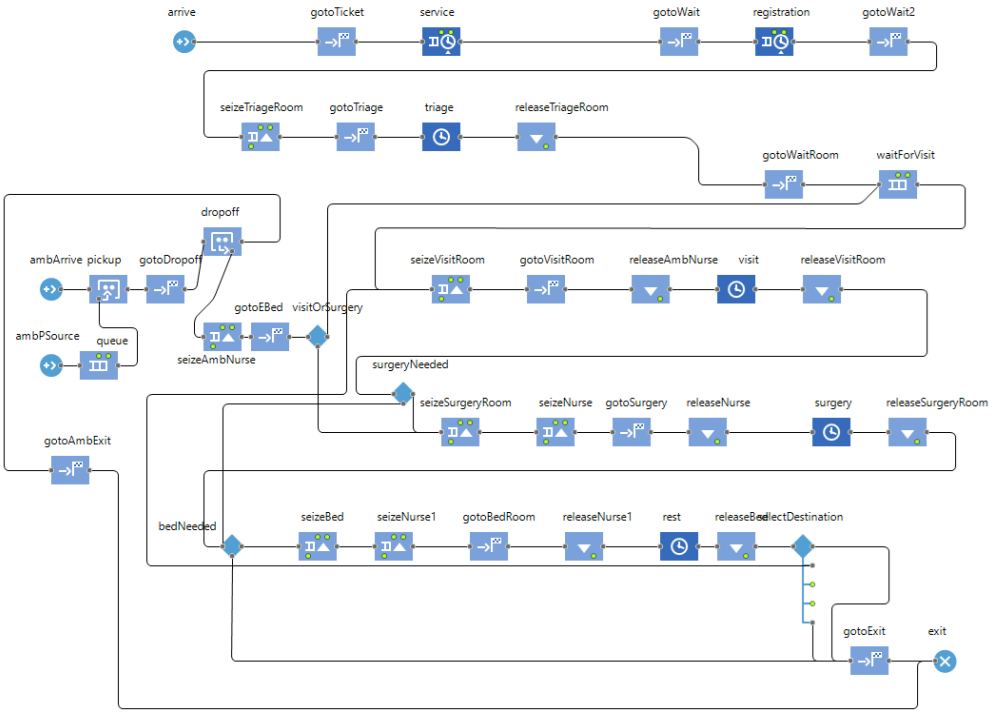

Figura 4.4: Workflow complessivo del sistema

4.3.1 Strumenti utilizzati nel Workflow

Nell'implementazione della simulazione abbiamo avuto necessità di utilizzare degli strumenti, tra cui le *resource pool* che ci hanno permesso di definire degli insiemi di unità che potevano essere riservate (*seized*) e rilasciate (*released*) dagli agenti che utilizzavano i seguenti tipi di blocco: `Seize`, `Release`, `Delay` e `Service`.

In particolare abbiamo definito i seguenti insiemi: infermieri, biglietteria, accettazione (Registrars), triage, sala visita, chirurgia e letti disponibili.

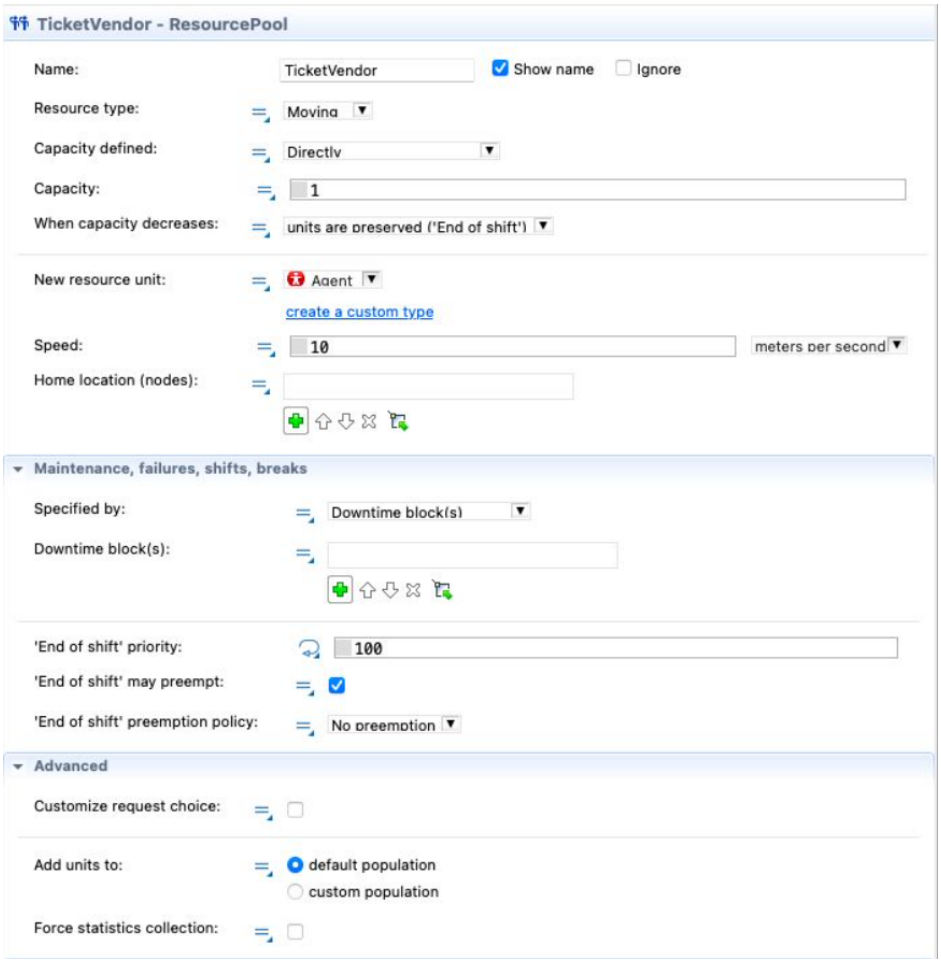

Figura 4.5: Interfaccia Resource Pool

All'interno della simulazione è stato necessario inserire parametri e funzioni; i primi ci hanno permesso di definire delle caratteristiche statiche e le seconde sono state fondamentali per calcolare valori a runtime a seconda del percorso seguito dal paziente.

All'interno del nostro *Main* possiamo trovare tre funzioni e due parametri:

- `arrivalRate`: il numero di pazienti che arrivano per ora, che nel nostro caso è settato a 8 pazienti/ora.
- `ambulanceArrRate`: il numero di pazienti che arrivano in ambulanza ogni ora (default 4 pazienti/ora).

- `staffSpeed`: restituisce la velocità dello staff del pronto soccorso.
- `patientSpeed`: restituisce la velocità dei pazienti.
- `restTime`: permette di calcolare il tempo che i pazienti necessitano di passare a letto, in base alla priorità assegnata.

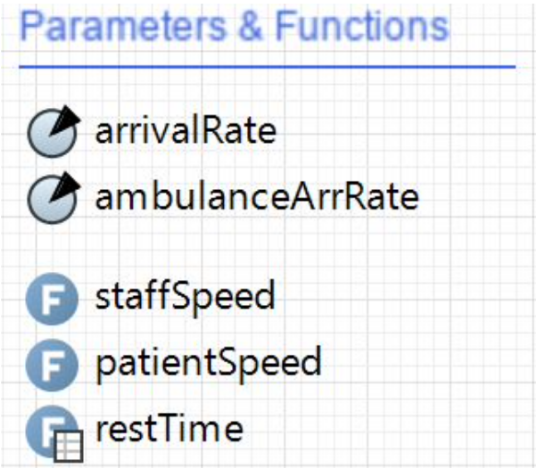

Figura 4.6: Parametri e Funzioni implementati

4.3.2 L'arrivo del paziente

Il nostro workflow ha due diversi punti d'inizio: il primo che andremo a trattare è il caso in cui il paziente arriva all'ospedale autonomamente, deve prendere il biglietto per poter sedersi in sala d'aspetto e successivamente essere chiamato per l'accettazione. Nel paragrafo 4.3.7 verrà trattato anche il caso in cui il paziente arrivi trasportato in ambulanza.

Questa parte del workflow comprende il blocco `arrive` da cui tutto il workflow ha inizio. All'interno di essp è possibile definire il rate di arrivo degli agenti (parametro `arrivalRate`) e la velocità che avranno nel modello (funzione `patientSpeed()`). Inoltre in questo blocco, come in tutti quelli successivi, è possibile definire la zona della simulazione in cui avviene l'azione e quale agente specifico è coinvolto (Patient in questo caso).

Il paziente arriva e va a prendere il biglietto con il blocco di movimento `gotoTicket` che segnala all'agente in che punto della simulazione recarsi.

Il blocco successivo `service` invece è quello designato per prendere il biglietto che sarà necessario per aspettare il turno dell'accettazione nella sala d'aspetto. Questo blocco è particolare in quanto permette di prendere delle risorse da una `resource pool` (Ticket Machine), creare una coda di grandezza prestabilita e settare un delay: per entrambe le cose è possibile definire il nodo in cui le azioni devono avvenire.

Dopo che l'agente ha preso il numero, con il blocco di movimento `gotoWait` si sposta in sala d'aspetto fino a che non verrà chiamato per effettuare l'accettazione.

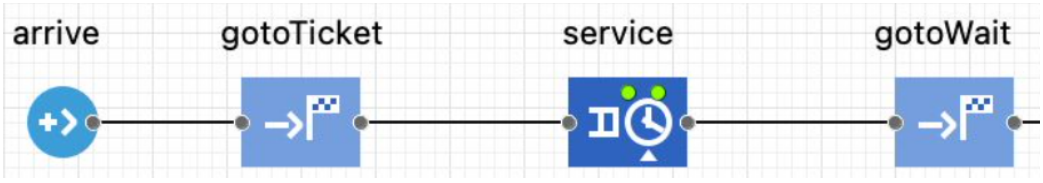

Figura 4.7: Workflow dell'arrivo del paziente

4.3.3 Triage

L'agente entra nel blocco `registration` che preleva una persona, se libera, dalla resource pool `Registrars` che avrà il compito di fare l'accettazione al paziente (ne sono presenti due); inoltre è settato un delay secondo una distribuzione triangolare, che sarà il tempo impiegato per effettuare l'accettazione. Infine, vengono delineate due zone: quella per aspettare in coda che l'accettazione sia libera e la zona in cui effettivamente effettuarla.
Dopo che il paziente effettua l'accettazione, tramite il blocco di movimento `gotoWait2` torna in sala d'aspetto.

Il blocco successivo `seizeTriageRoom` crea la coda di persone che nella `WaitingRoom` aspettano di venir chiamati per andare a fare il triage, richiede alla resource pool `TriageRooms` se ci sia una sala libera e nel caso ci fosse la riserva per il paziente; tramite il blocco `gotoTriage` l'agente ci si dirige, altrimenti rimane nella coda ad aspettare una sala libera.

Il triage viene eseguito dal blocco `triage` che ha la particolarità di creare un delay in cui verosimilmente viene effettuato un triage: da questo blocco il paziente uscirà con la diagnosi e gli verrà assegnata una priorità.

Alla fine di queste operazioni, tramite il blocco `release TriageRoom` viene liberata la sala per il triage che ritornerà disponibile nella resource pool `TriageRooms`.

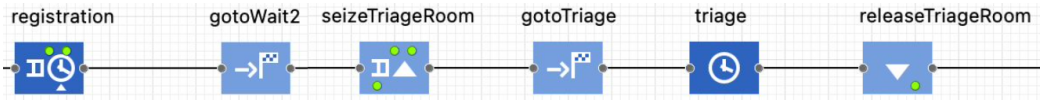

Figura 4.8: Workflow del triage

4.3.4 Visita Specialistica

La priorità affidata al paziente sulla base della diagnosi sarà la discriminante per mandarlo direttamente alle sale visita (in casi di codice rosso) oppure ad aspettare in sala d'attesa.

Nel caso in cui il paziente non è troppo grave viene quindi mandato nella sala d'attesa (`gotoWaitRoom`) e si mette in fila (`wairForVisit`), quest'ultima gestita secondo una politica priority-based (a seconda della priorità di ciascun paziente).

Quando il paziente è il primo della coda, viene controllato se nella resource pool `VisitRooms` vi sia una sala visita libera e nel caso vi sia viene mandato a fare la visita (`gotoVisitRoom`).

Durante la visita viene valutato se sia necessaria la chirurgia e/o un letto. Al termine della visita viene rilasciata la sala visita (`releaseVisitRoom`) che torna nella resource pool `VisitRooms`.

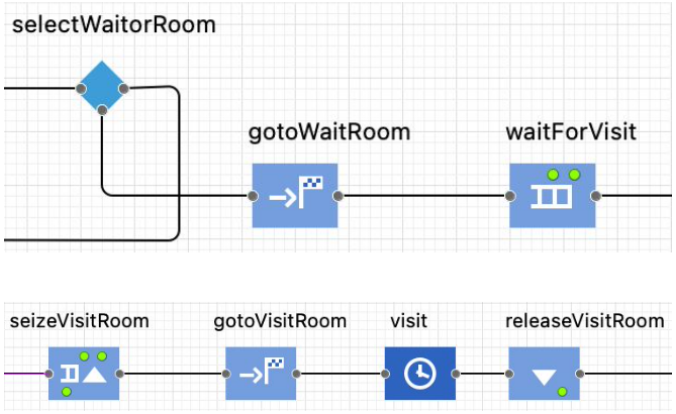

Figura 4.9: Workflow di una visita specialistica

4.3.5 Chirurgia

Nel caso in cui il paziente avesse bisogno di un intervento in chirurgia (`surgeryNeeded`) viene verificato che ci siano delle sale operatorie (`seize SurgeryRoom`) e infermieri (`seizeNurse`) liberi.
Se queste due condizioni sono verificare il paziente viene spostato in chirurgia con l'aiuto dell'infermiere (`gotoSurgery`) che viene rilasciato (`releaseNurse`) nel momento in cui il paziente arriva nella sala, tornando al nodo `staffRoom`.
L'intervento (`surgery`) dura per un tempo variabile che segue una distribuzione triangolare e nel momento in cui finisce la sala viene liberata (`release SurgeryRoom`).

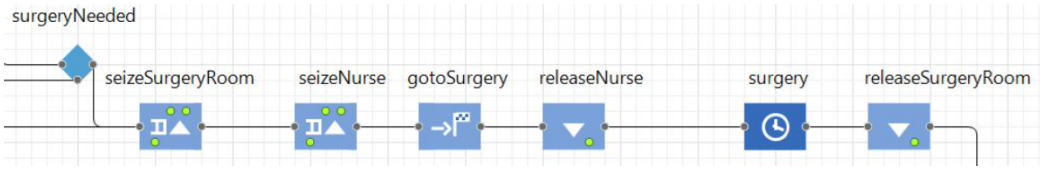

Figura 4.10: Workflow di un intervento chirurgico

4.3.6 Degenza e uscita

Il paziente può aver bisogno di un letto in due casi: se ha avuto un intervento chirurgico oppure ha bisogno di essere genericamente monitorato.

Viene quindi controllato che ci siano un letto disponibile (`seizeBed`) e un infermiere libero (`seizeNurse1`) per accompagnare il paziente nella degenza (`gotoBedRoom`), infermiere che torna a disposizione dopo aver portato il paziente (`releaseNurse1`).

Durante la degenza (`rest`) i pazienti possono aver bisogno di ulteriori visite (seconda uscita del blocco `selectDestination`) oppure i problemi possono risolversi ed essere dimessi (prima e ultima uscita, ovvero quella di default). Dopo un delay variabile a seconda della gravità del paziente questo parametro viene aggiornato.

Nel caso in cui non fosse necessaria un'ulteriore visita il paziente può uscire (`gotoExit`) liberando il letto (`releaseBed`).

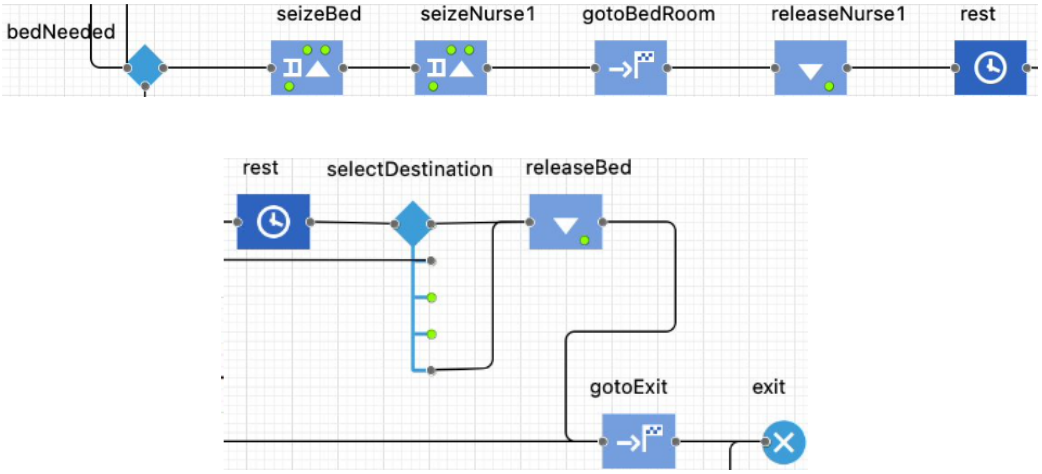

Figura 4.11: Workflow della degenza e dll'uscita

4.3.7 Arrivo in Ambulanza

Un paziente oltre a poter arrivare al pronto soccorso da solo può arrivare in ambulanza.

Le ambulanze vengono generate dal blocco `ambArrive` ogni qualvolta un paziente ha bisogno del loro intervento.

I pazienti bisognosi vengono creati dal blocco `ambPSource`, dal quale poi vengono inviati in una coda per attendere l'arrivo dell'ambulanza. All'entrata nel blocco di coda, viene invocata la funzione `inject()` sul blocco di generazione delle ambulanze per chiamarne una all'occorrenza.

L'ambulanza funge da container e tramite il blocco `pickup` carica il paziente e lo scarica (tramite `dropoff`) all'entrata per ambulanze del pronto soccorso, dove gli viene riservato un infermiere per portarlo ad effettuare una visita o, nei casi più urgenti, una chirurgia.

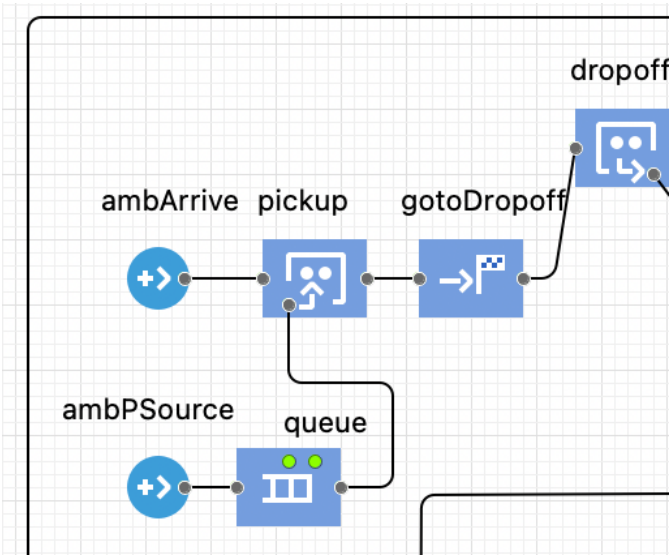

Figura 4.12: Workflow dell'arrivo in ambulanza

Capitolo 5

Sperimentazioni e Risultati

Le potenzialità di AnyLogic ci permettono di fare varie sperimentazioni sul modello che abbiamo costruito. In particolare ci andremo a concentrare sulla distribuzione della gravità dei pazienti, basandoci sui codici con cui arrivano al pronto soccorso, per valutare la risposta della nostra simulazione.

Valuteremo i seguenti casi:

- **Distribuzione Uniforme**: utilizziamo il random su option list fornita da AnyLogic.
- **Distribuzione Personalizzata**: utilizziamo la nostra distribuzione personalizzata in cui i codici si presentano con le seguenti percentuali:
  - 25% codici bianchi
  - 40% codici verdi
  - 25% codici gialli
  - 10% codici rossi

Inoltre, avendo la possibilità di personalizzare i parametri dei nostri agenti è stato possibile:

- `appearence = uniform_discr(1, 3)`: dare un indicazione di come avremo voluto che apparissero
- `priority = 0` e `surgery = no`: il livello di priorità iniziale e la necessità di intervento che vengono impostati a zero e successivamente modificati durante il blocco di `triage`.
- `bed = yesB`: si pensa che le persone arrivino genericamente con una condizione che necessiterà di una breve degenza (anche di minuti), anche questo può venir modificato durante il workflow.
- `initial speed = 5`: la velocità iniziale del paziente che è stata impostata a 5km all'ora in accordo con i risultati medi dei paper consultati.

All'interno del *Main* della simulazione sono presenti anche delle funzioni e parametri che sono stati precedentemente presentati nel capitolo 4.3.1 che abbiamo impostato con i seguenti valori:

- `arrivalRate = 8`: le persone che arrivano in ospedale autonomamente sono 8 all'ora.
- `ambulanceArrRate = 4`: il numero di ambulanze che arrivano in ospedale sono 4 all'ora.
- `staffSpeed()`: ritorna una distribuzione uniforme `uniform(3.3, 4.2)`.
- `patientSpeed()` ritorna una distribuzione uniforme `uniform(2.5, 3.5)`.
- `restTime()`: è una funzione costruita per simulare il tempo necessario nella degenza per un paziente, è modellata come in figura 5.1.

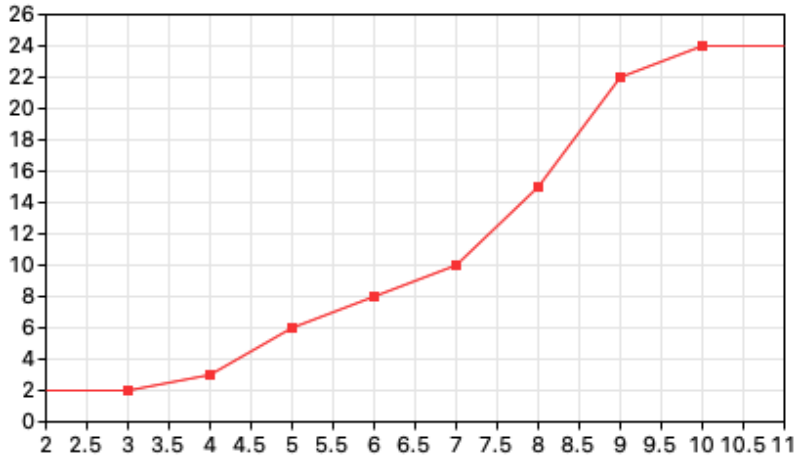

Figura 5.1: restTime()

Le *resource pool* hanno anch'esse dei valori che possono essere impostati e nel nostro modello ci siamo soffermati in particolare sul numero di elementi che componevano ogni pool che riportiamo:

- `Nurses = 10`
- `Registrars = 2`
- `TriageRooms = 2`: viene impostata in base alle stanze per il triage effettivamente esistenti, nel nostro caso due.
- `TicketMachine = 1`
- `VisitRooms = 4`: viene impostata in base alle sale visita effettivamente esistenti, nel nostro caso quattro.
- `SurgeryRooms = 2`: viene impostata in base alle sale operatorie effettivamente esistenti, nel nostro caso due.
- `Beds = 10`

5.1 Sperimentazioni e Parametri utilizzati

Per poter analizzare propriamente le conseguenze del cambiamento delle variabili sopra descritte, abbiamo definito alcune statistiche che ci permettessero di visualizzare alcune delle metriche più importanti:

- *Tempo totale di permanenza, per paziente (minuti)*
- *Numero di pazienti presenti nel pronto soccorso, nel tempo*
- *Tempo di attesa medio (minuti)*

I risultati ottenuti sono stati successivamente confrontati con quelli ottenuti dagli autori dei lavori precedenti (*Espinoza et al., ≪Real-time simulation as a way to improve daily operations in an Emergency Room≫*).

Tutte le sperimentazioni sono state eseguite per 500 minuti.

5.1.1 Prima sperimentazione

I valori dei parametri, per questo primo tentativo, sono stati mantenuti come descritti nei paragrafi precedenti, ovvero arrivano 8 pazienti e 4 ambulanze all'ora. Questa sperimentazione sarà utilizzata nel seguito come "caso di default" per i confronti.

Quello che possiamo vedere dall'osservazione di questa iterazione sono i seguenti valori medi riportati in figura 5.2 anche sotto riportati:

- *Tempo medio di permanenza:* 163 minuti.
- *Numero di pazienti presenti nel pronto soccorso, nel tempo*: 54 pazienti.
- *Tempo di attesa medio:* 91 minuti.

Possiamo vedere che il numero di pazienti andrebbe ad aumentare così come il tempo medio, e questo potrebbe portare ad una saturazione del pronto soccorso.

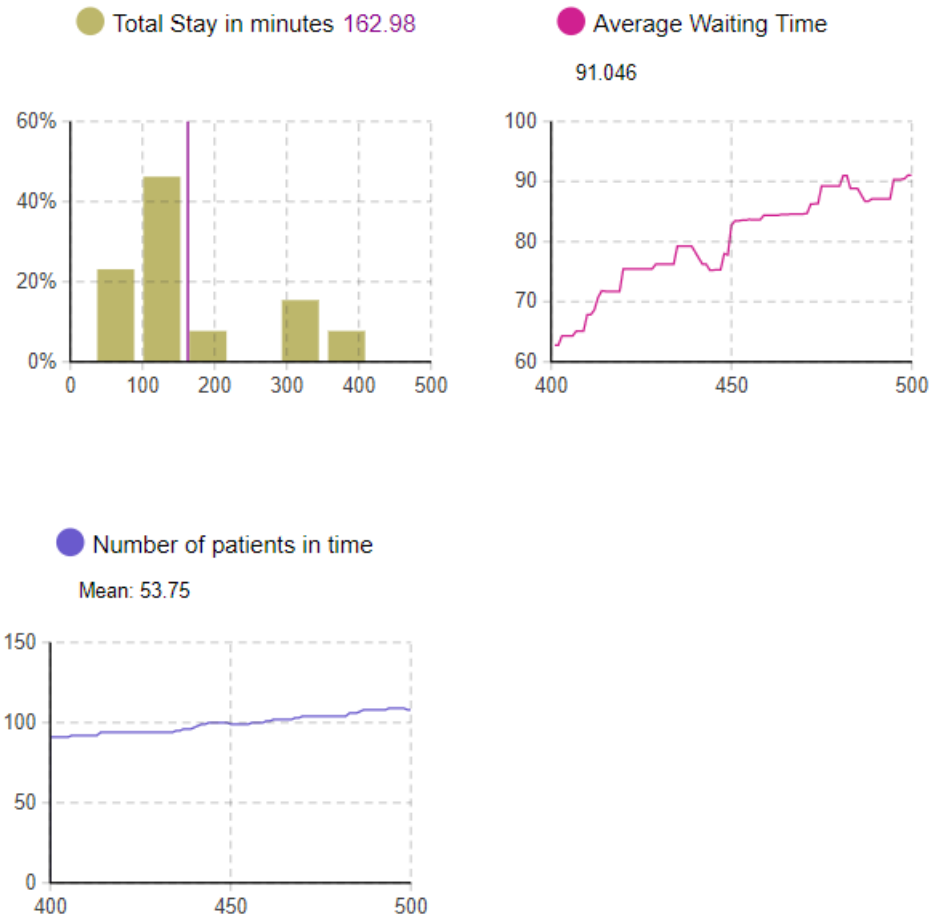

Figura 5.2: Grafici delle metriche della prima sperimentazione

5.1.2 Seconda Sperimentazione

I valori dei parametri, in questa seconda configurazione, sono stati raddoppiati rispetto alla prima sperimentazione, ovvero arrivano 16 pazienti e 8 ambulanze all'ora.

Per questa sperimentazione troviamo come valori delle metriche:

- *Tempo medio di permanenza:* 221 minuti.
- *Numero di pazienti presenti nel pronto soccorso, nel tempo*: 84 pazienti.
- *Tempo di attesa medio:* 98 minuti.

5.1.3 Terza Sperimentazione

Per eseguire il terzo esperimento abbiamo mantenuto i valori di default per quanto riguarda il tasso di arrivo di ambulanze e pazienti (8 pazienti/ora, 4 ambulanze/ora), modificando però la distribuzione del parametro *sickness* (dei pazienti arrivati a piedi) con una distribuzione uniforme, invece che basarla su percentuali più realistiche come impostato in precedenza.

Con questa configurazione i risultati ottenuti sono i seguenti:

- *Tempo medio di permanenza:* 143 minuti.
- *Numero di pazienti presenti nel pronto soccorso, nel tempo*: 46 pazienti.
- *Tempo di attesa medio:* 78 minuti.

5.1.4 Quarta Sperimentazione

Per l'esecuzione del quarto esperimento abbiamo scelto di modificare il tempo di attesa nel triage abbassandolo (`triangular(2, 20, 5)` → `triangular(2, 10, 5)`) e riportare le percentuali di *sickness* su una distribuzione realistica ed abbiamo ottenuto i seguenti risultati.

- *Tempo medio di permanenza:* 156 minuti.

- *Numero di pazienti presenti nel pronto soccorso, nel tempo*: 53 pazienti.
- *Tempo di attesa medio:* 76 minuti.

5.1.5 Quinta Sperimentazione

L'ultima sperimentazione effettuata ha previsto l'aumento del numero di infermieri disponibili, da 10 a 20, mantenendo i tassi di arrivo di default.

I risultati ottenuti sono i seguenti:

- *Tempo medio di permanenza:* 142 minuti.
- *Numero di pazienti presenti nel pronto soccorso, nel tempo*: 53 pazienti.
- *Tempo di attesa medio:* 89 minuti.

5.2 Risultati

In seguito alle sperimentazioni effettuate con i parametri impostati come precedentemente presentato nel paragrafo 5.1, abbiamo potuto osservare e trarre le seguenti conclusioni:

1. A parità di tasso di arrivo, una distribuzione uniforme dei codici dei pazienti (per quanto irrealistica) migliora di molto l'efficienza in tutte e tre le metriche analizzate, come prevedibile. Tuttavia non si avrà mai questo caso nella realtà;
2. La diminuzione del tempo massimo di triage ha un leggero impatto sul tempo di attesa medio, che passa da 91 minuti del caso di default a 76 minuti;
3. Il raddoppio dei tassi di arrivo adottato nella seconda sperimentazione influisce negativamente in maniera sostanziale sul tempo medio di permanenza, mentre non impatta troppo il tempo di attesa medio;

4. L'aumento del numero di infermieri disponibili non migliora quasi per nulla l'efficienza rispetto al caso di default. Questo può essere dovuto al fatto che gli infermieri, nel nostro modello, hanno mansioni di solo accompagnamento che difficilmente portano ad una saturazione delle risorse disponibili.

In conclusione, da quanto osservato, possiamo dire che per aumentare il più possibile l'efficienza del modello e di conseguenza quella di un caso reale, non essendo evidentemente possibile diminuire sempre il tempo massimo di visite e triage o prevedere una distribuzione uniforme della gravità dei pazienti, si potrebbe pensare di aumentare la capacità del pronto soccorso con nuove stanze, permettendo così il trattamento di più pazienti in parallelo.

È tuttavia da sottolineare che nel presente modello non sono state considerate varie problematiche reali come ad esempio il cambio di turno dello staff o la sua non-disponibilità, che potrebbero peggiorare di molto le metriche prese in considerazione.

5.3 Confronto con lavori precedenti

I risultati ottenuti nei paper di riferimento per la costruzione del modello non sono perfettamente comparabili con i risultati da noi osservati poiché essi utilizzavano varie distinzioni di scenari maggiormente complessi ed articolati rispetto al nostro lavoro.

Possiamo però sottolineare come in *Espinoza et al., «Real-time simulation as a way to improve daily operations in an Emergency Room»* vengano messi in evidenza il numero di accessi al pronto soccorso durante il giorno al variare del giorno della settimana considerato. Valutando una media degli accessi giornalieri possiamo vedere una media che si mantiene tra i 6 e gli 8 pazienti all'ora, con picchi fino a 12 nelle ore più congestionare e 2 nelle ore notturne. Nella nostra simulazione abbiamo provato a simulare un momento di particolare congestione nel paragrafo 5.1.2 arrivando a sedici pazienti e quattro ambulanze all'ora. Da notare però che il tempo medio di attesa in tutte queste condizione rimane sostanzialmente inferiore a quello dai noi osservato

(dai 10 ai 37 minuti rispetto ai 76-98 del nostro modello). In merito però a questi risultati possiamo però dire che la simulazione presentata nel paper ha delle differenze sostanziali nella costruzione del pronto soccorso basandosi su una struttura esistente ed i tempi non derivano da una simulazione ma dall'osservazione reale del fenomeno.

Per questo motivo, se pure si è cercato di imitare un contesto realistico e vicino alla realtà, abbiamo comunque scelto di generalizzare piuttosto che andare nello specifico di un singolo pronto soccorso esistente e per questo motivo i dati dei paper non coincidono perfettamente con i nostri.

Capitolo 6

Conclusioni

In conclusione, date le osservazioni ricavate dalle sperimentazioni effettuate possiamo affermare che il software AnyLogic si dimostra uno strumento utile per supportare la simulazione di scenari che hanno bisogno di essere ottimizzati, valutarne possibili estensioni o semplicemente per rispondere a domande del tipo *"what if...?"*.

La palette di tool disponibili, dalla libreria per la modellazione del movimento di pedoni a quella per la modellazione del movimento di fluidi, lo rende uno strumento versatile ed utilizzabile in moltissimi ambiti.

La modellazione a blocchi rende questo strumento easy to learn, ma le molte funzionalità e il fatto di essere basato completamente su Java lo rendono hard to master; inoltre la possibilità di integrare algoritmi di learning è interessante, ma non è compresa direttamente nel software sebbene con *Pathmind* sia possibile avere un Helper integrato.

Nel caso della nostra simulazione è risultato difficile integrare algoritmi di learning: gli agenti che abbiamo modellato seguivano un workflow preciso ed ognuno di essi non perseguiva un obiettivo, ma si muoveva in base al prossimo blocco presente nel flusso.

Bisogna menzionare che è sempre tuttavia possibile osservare o modificare le variabili a runtime, tramite controlli, per vedere come reagisce il modello. Questo ci permette di avere un controllo maggiore sulla simulazione e di poter

effettuare anche degli *"stress test"*. Molto interessante anche la possibilità di inserire statistiche per le popolazioni di agenti ed osservarne l'andamento tramite grafici, per una miglior comprensione dell'andamento.

AnyLogic è uno strumento molto ampio e sarebbe necessaria una quantità di tempo più elevata per realizzare e simulare modelli più complessi, ma ciò nonostante si è presentato pratico e funzionale per la realizzazione del modello scelto.

Bibliografia

[1] Marelys L. García et al. «Reducing time in an emergency room via a fast-track». In: *Proceedings of the 27th conference on Winter simulation*. WSC '95. USA: IEEE Computer Society, dic. 1995, pp. 1048–1053. ISBN: 978-0-7803-3018-4. DOI: 10.1145/224401.224771. URL: https://doi.org/10.1145/224401.224771.

[2] Camila Espinoza et al. «Real-time simulation as a way to improve daily operations in an Emergency Room». In: *Proceedings of the Winter Simulation Conference 2014*. ISSN: 1558-4305. Dic. 2014, pp. 1445–1456. DOI: 10.1109/WSC.2014.7019998.

[3] Hayden Stainsby, Manel Taboada e Emilio Luque. «Towards an Agent-Based Simulation of Hospital Emergency Departments». In: *2009 IEEE International Conference on Services Computing*. Set. 2009, pp. 536–539. DOI: 10.1109/SCC.2009.53.

[4] Lu Wang. «An agent-based simulation for workflow in Emergency Department». In: *2009 Systems and Information Engineering Design Symposium*. Apr. 2009, pp. 19–23. DOI: 10.1109/SIEDS.2009.5166148.

[5] Dennis Pegden e Inyong Ham. «Simulation of Manufacturing Systems Using SIMAN». en. In: *CIRP Annals* 31.1 (gen. 1982), pp. 365–369. ISSN: 0007-8506. DOI: 10.1016/S0007-8506(07)63329-0. URL: https://www.sciencedirect.com/science/article/pii/S0007850607633290.