

## ESERCIZI DI SIMULAZIONE DISCRETA 2

1. **(Coda Prioritaria)** Considerare una fila d'attesa con priorità in cui gli utenti arrivano in modo casuale e indipendente con frequenza pari a due al minuto. Il 25% hanno priorità nella coda per un servizio prestato da tre serventi con tempi distribuiti secondo i seguenti casi:
  - a. una gamma con media 120 secondi e deviazione standard 30 secondi per tutti gli utenti;
  - b. una gamma con media 120 secondi e deviazione standard 30 secondi per gli utenti prioritari e 240 secondi e 60 secondi per i non prioritari.Si simulino 100 arrivi.
  
2. **(Produzione con Schedule e Table Function)** In un reparto di lavorazione i pezzi arrivano in modo casuale e indipendente con le seguenti frequenze:
  - dalle 8 alle 10: 6 pezzi/h;
  - dalle 10 alle 14: 10 pezzi/h;
  - dalle 14 alle 20: 4 pezzi/h.I pezzi vengono lavorati da  $n$  macchine uguali fra loro dove:
  - dalle 8 alle 10:  $n=2$ ;
  - dalle 10 alle 14:  $n=4$ ;
  - dalle 14 alle 20:  $n=1$ .Il tempo di lavorazione è distribuito secondo una normale di media 12 min. e dev. std. 2 min. Calcolare tempi di permanenza per fasce orarie.
  
3. **(Tipi di agenti differenziati)** Due tipi di pezzi arrivano in modo casuale e indipendente con frequenze rispettivamente:
  - tipo 1: 10 pezzi/h,
  - tipo 2: 12 pezzi/h,per 8 h. I pezzi vengono lavorati dalla stessa unica macchina con tempo distribuito secondo una normale con:
  - tipo 1: media 2 min. e deviazione standard 0.1 min.
  - tipo 2: media 3 min. e deviazione standard 0.1 min.I pezzi di tipo 2 hanno priorità nella coda davanti alla macchina. Calcolare i tempi di coda e permanenza per i due tipi di pezzi.
  
4. **(Percorsi differenziati con ritorno)** Il servizio di ritiro referti è organizzato con 4 sportelli. Gli utenti arrivano a intervalli distribuiti, nelle ore di punta, secondo un'esponenziale di media 2 minuti. All'ingresso, ogni utente ritira un biglietto numerato che determina il turno per il servizio al primo sportello libero. Allo sportello, dopo un tempo distribuito secondo un'esponenziale di media 7 minuti (lo stesso per tutti gli sportelli), l'utente riceve i dati per il pagamento del ticket, e si mette in coda presso una macchina self-service: il tempo per pagare il ticket è distribuito secondo una normale di media 5 minuti e deviazione standard 1 minuto. Pagato il ticket, l'utente torna agli sportelli iniziali per il ritiro dei referti, che dura mediamente 3 minuti. Si tenga conto che, la seconda volta, l'utente non deve rifare la coda, ma passa davanti a tutti gli altri utenti che non hanno ancora pagato il ticket. Si vuole determinare il tempo medio speso dagli utenti per ritirare i referti e la lunghezza media delle varie code, considerando anche scenari alternativi rispetto al numero di sportelli e di macchine self-service.

## Suggerimenti per la simulazione con AnyLogic

1. Per la realizzazione della **coda prioritaria**, impostare nel blocco *queue* l'attributo *Queueing* a *Prioriy-based* e definire la priorità a 2 nel 25% dei casi e a 1 nel restante 75% (le entità con valore di priorità più alto passano davanti a quelli con valore più basso), impostando il campo *Agent priority* (usiamo un'espressione condizionale tipica di Java e di molti linguaggi di programmazione). Se si apre un probe (sonda) sulla coda (per aprire una probe su un blocco in fase di simulazione, fare click sul blocco stesso), si vede come talvolta entità con identificativo più alto (entrate dopo) passino davanti alle altre.

Si noti che in questo modo l'informazione sulla priorità non viene mantenuta e, pertanto, non può essere utilizzata per simulare il caso b. Allo scopo dobbiamo fare in modo che la priorità diventi una caratteristica dell'entità passante che possa essere letta dal servente per determinare il tempo di servizio.

Per creare delle entità passanti con attributi speciali (la priorità in questo caso), **definiamo un nuovo tipo di agente**: lo possiamo fare agevolmente dal blocco *source* con *new agent – create a custom type* (corrispondente a *File – New – Agent type*) e seguendo le istruzioni (in particolare per inserire l'attributo *priority* come un *new parameter*).

I nuovi attributi così creati potranno essere letti o modificati durante la simulazione. Ogni entità passante creata, infatti, sarà accessibile tramite l'oggetto *agent* disponibile in ogni blocco attraversato. I diversi attributi di *agent* sono disponibili tramite la sintassi "*agent.nomeattributo*".

Nel nostro caso, impostiamo *agent.priority* tramite un'espressione condizionale attivata quando un'entità passante esce dal *source* (azione *on exit* del blocco *source*).



A questo punto possiamo usare *agent.priority* nel blocco *queue* per modificare le modalità di accodamento, e anche nel blocco *delay* per impostare un tempo di servizio differenziato. In particolare, per realizzare la coda prioritaria in base al valore di *agent.priority*, impostiamo, nel blocco *queue*, *Queueing* a *Agent comparison*, e il campo *Agent 1 is preferred to Agent 2* a *agent1.priority > agent2.priority* (*agent1* è l'entità entrante, *agent2* è un'entità già in coda).

**ATTENZIONE:** il procedimento indicato dall'Help, che imposta *Queueing* a *Prioriy-based* e *Agent priority* a *agent.priority*, sarebbe più diretto ma sembra non funzionare correttamente.

Nell'esercizio svolto, abbiamo anche differenziato i percorsi di uscita per le entità prioritarie tramite un blocco *SelectOutput* controllato da condizione (opzione *if condition is true*).

2. Per ottenere medie dei tassi di arrivo differenziate per fascia oraria, impostiamo *Arrivals defined by a Rate Schedule* (possiamo usare *rate* perché gli arrivi sono casuali e indipendenti, altrimenti si poteva usare *Arrival Schedule* nei modi descritti dall'Help). In questo modo leggiamo il tasso di arrivo da un apposito oggetto *Schedule* (palette *Agent*). Tale oggetto è impostato in modo da definire il tasso orario (*rate per hour*) degli arrivi come descritto nella consegna (vedi *Help* su blocchi *Source* e su oggetti *Schedule* per maggiori dettagli).

Per ottenere un numero di macchine variabile usiamo una *TableFunction* (palette *Agent*) che definisce una funzione tramite una tabella di punti in un piano cartesiano. Nel nostro caso consideriamo una funzione che ha in ascisse il tempo (considerato in ore, visto che abbiamo scelto ore come unità di misura del tempo del modello), e in ordinata il numero di macchine. Abbiamo impostato *Interpolation* a *Step* e *Out of range* a *Nearest* per ottenere la funzione a gradino, come richiesto. Per accedere ai valori della tabella usiamo la funzione *time()* per definire l'ascisse, ossia il tempo di simulazione corrente.

**ATTENZIONE:** molti degli attributi dei blocchi visti finora (ad esempio *Interarrival Time* in *source*, oppure *delay time* in *delay*) sono **dinamici**, cioè l'espressione ad essi associata viene automaticamente aggiornata quando richiamata. Abbiamo sfruttato questa caratteristica nell'Esercizio 2.1 per aggiornare il tempo di servizio a fronte di entità passanti diverse. Gli attributi dinamici sono indicati con la parola *dynamic* nella descrizione dei blocchi (e individuati dall'apposito simbolo  nell'interfaccia). Invece la *capacity* di un *delay* è un **attributo statico** (*static* ) , cioè non viene aggiornato automaticamente, ma viene valutato solo all'inizio della simulazione: il suo cambiamento deve essere pertanto forzato esplicitamente durante

l'esecuzione di un evento: noi abbiamo scelto *on enter* dello stesso *delay* (in modo che ogni entità passante trovi il giusto numero di serventi, a seconda dell'ora di arrivo nel *delay*).

Per misurare il tempo di permanenza nel sistema abbiamo scelto un metodo più flessibile rispetto a quello descritto nell'esercizio 1.2: definiamo un nuovo tipo di agente che registri i tempi di ingresso e di uscita dal sistema come parametri; impostiamo tali tempi *on exit* dal *sink* e *on exit* dal *delay*; creiamo tre oggetti *Variable* (palette *Agent*) che accumulino i tempi totali delle entità appartenenti alle tre fasce orarie; separiamo i *sink* per le tre fasce; aggiorniamo le variabili *on enter* del rispettivo *sink*; visualizziamo i grafici delle medie calcolate in base alle tre variabili e ai valori dei diversi *sink.count()*.

3. Vengono utilizzati due blocchi *source* distinti per generare in modo diverso i due tipi di pezzi. Per caratterizzare le entità passanti associate ai diversi tipi di pezzi, utilizziamo, per i due *source*, un nuovo tipo di agente che memorizza, in uscita dagli stessi *source*, il tipo di pezzo in *agent.type*. Questo viene utilizzato per gestire la priorità nella coda. Inoltre sono registrati diversi tempi (ingresso nel sistema, uscita dalla coda, uscita dal servizio) usati alla fine dei percorsi (*sink*) per calcolare diversi tempi medi di permanenza. Si usa un *Time Stack Chart* per visualizzare gli utenti in coda distinti per tipo: a tal fine le *Variable* *qu1* e *qu2* sono inizializzate a 0 e aggiornate in ingresso e in uscita dalla coda, a seconda del tipo dell'entità passante (letta da *agent.type*).

Per interrompere gli arrivi dopo 8 ore, potremmo usare un oggetto *Schedule* come nell'esercizio precedente, ma procediamo un modo più semplice (e generale). Usiamo un oggetto *Event* dalla palette *Agent*, che introduce nella simulazione degli eventi definiti in modo diretto dall'utente (vedi Help). In questo caso, introduciamo un evento dopo 8 ore che serve per interrompere la generazione di nuove entità. Impostiamo l'evento *eventStopArrivals* in modo che scatti dopo 8 ore (*Trigger type* = *Timeout*, *Occurrence time* = 8) e non si ripeta (*Mode* = *Occurs once*). L'azione collegata all'evento (*Action*) permette di "congelare" le due *source* limitando gli arrivi a quelli già avvenuti. A questo punto, possiamo lasciare che gli eventi si esauriscano e interrompere a mano la simulazione (avendo posto *Simulation – Model time – Stop* pari a *Never*), oppure interrompere automaticamente quando tutte le entità sono arrivate nei *sink*. Allo scopo introduciamo una *Function* dalla palette *Agent*: il codice inserito nel *Function body* di *checkFinishSimulation* controlla se le sorgenti sono state congelate e tutte le entità sono arrivate nei *sink* e, in questo caso, usa il comando di AnyLogic *finishSimulation()* per interrompere la simulazione.

4. I percorsi delle entità sono controllati dall'attributo *agent.status*, aggiunto in un nuovo tipo di agente appositamente creato. Inizialmente *agent.status* è posto a 1 dal *source*, e poi a 2 in uscita dal *delay* relativo al POS. Il valore di questo campo determina la priorità nella coda allo sportello, il relativo tempo di servizio, e il percorso seguito dall'entità.

Per considerare scenari alternativi direttamente tramite l'interfaccia, si usano dei controlli dalla palette *Controls*. In particolare, si usano delle *slider* associate al numero di sportelli e al numero di POS. In questo caso si imposta il campo *Link to* in modo da collegare direttamente il valore della *slider* alla *capacity* dei rispettivi *delay*. Si usa ancora una *slider* per variare il tempo di servizio medio nel POS: in questo caso non possiamo sfruttare un collegamento diretto tramite *Link to*, in quanto il tempo di servizio non è direttamente determinato dal valore della *slider*, ma usato nell'espressione che lo calcola (una normale in questo caso). Pertanto si collega il valore della *slider* a una *variable* di comodo, e la variabile è usata nell'espressione del tempo di servizio. Notare che, essendo il tempo di servizio un attributo dinamico, la sua espressione verrà automaticamente valutata all'occorrenza, rispecchiando immediatamente il nuovo valore della media impostato, tramite la *variable*, con la *slider*.

Notare che la variabile *ticket* non è utilizzata, visto che gli utenti sono automaticamente serviti nell'ordine di arrivo tramite la coda (gestione FIFO all'interno delle classi di priorità). La variabile poteva essere omessa, ma è stata mantenuta nel disegno del modello al solo fine di evidenziare visivamente la presenza dei biglietti numerati utilizzati nel sistema reale per indicare l'ordine di arrivo e facilitare la realizzazione della politica FIFO nella gestione degli utenti.