

1. **(Flexible Manufacturing System)** Considerare un sistema flessibile di lavorazione formato da tre macchine  $m_a$ ,  $m_b$ ,  $m_c$  e da un sistema di handling, costituito da una macchina  $m_h$  per il carico dei pezzi dall'esterno e lo spostamento dei pezzi da una macchina all'altra o da una macchina verso lo scarico all'esterno. I tempi di lavoro delle macchine  $m_a$ ,  $m_b$ ,  $m_c$  sono supposti deterministici e dipendono dal tipo di pezzo da lavorare, mentre il tempo di lavoro della macchina  $m_h$  è distribuito secondo una densità gamma di media 0,2 minuti e devstd 0,05 minuti. Ogni pezzo in arrivo viene caricato e trasportato dalla macchina  $m_h$ , lavorato da una prima macchina, trasportato dalla macchina  $m_h$  verso una seconda macchina di lavorazione, lavorato sulla seconda macchina, eventualmente trasportato verso una terza macchina di lavorazione, scaricato all'esterno dalla macchina  $m_h$ . I pezzi di tipo  $p_1$ ,  $p_2$ ,  $p_3$  e  $p_4$  arrivano con tempo di interarrivo distribuito secondo una normale di parametri dipendenti dal tipo come segue:
  - $p_1$ : media 10 min devstd 2 min;
  - $p_2$ : media 3 min devstd 0.2 min;
  - $p_3$ : media 5 min devstd 1 min;
  - $p_4$ : media 7 min devstd 2 min;
 a partire dal tempo 0 e nelle seguenti quantità:
  - $p_1$ : 100 pezzi,
  - $p_2$ : 300 pezzi,
  - $p_3$ : 200 pezzi,
  - $p_4$ : 150 pezzi,
 e subiscono le seguenti lavorazioni (tempi costanti):
  - $p_1$ : 4 min su  $m_a$ , 3 min su  $m_b$ , 1 min su  $m_c$ ;
  - $p_2$ : 1 min su  $m_a$ , 2 min su  $m_b$ ;
  - $p_3$ : 5 min su  $m_a$ , 1 min su  $m_c$ ;
  - $p_4$ : 1 min su  $m_b$ , 1 min su  $m_c$ ;
 Costruire il modello e simularlo; si osserverà che la coda più consistente si avrà davanti alla macchina  $m_a$ ; uno sdoppiamento della macchina  $m_a$  porta lo spostamento della coda più consistente sulla macchina  $m_b$ . Come possiamo configurare il sistema per individuare ed eliminare colli di bottiglia?
2. **(Guasti – gestione eventi aggiutivi)** Considerare una macchina che riceve 1000 pezzi di semilavorato al ritmo costante di uno ogni 8 minuti e lavora un pezzo in un tempo distribuito secondo una normale di media 6 min e devstd 0,2 min. La macchina è soggetta a rotture che avvengono in modo casuale e indipendente con tempo di interarrivo pari a 200 minuti; dopo una rottura la macchina viene riparata in un tempo distribuito secondo una esponenziale con media 50 minuti, mentre il buffer si riempie. Simulare il funzionamento descritto.
3. **(Servente di riserva)** Considerare un'officina nella quale pezzi di semilavorato arrivano a ritmo costante di 10 pezzi l'ora per le prime due ore e 15 pezzi l'ora per le successive 8 ore. L'officina ha due macchine uguali capaci di lavorare un pezzo in un tempo distribuito normalmente con media 10 min e devstd 2 min, più una macchina aggiuntiva, capace di lavorare un pezzo in un tempo distribuito normalmente con media 12 min e devstd 3 min, che interviene solo quando entrambe le altre sono già occupate. Simulare e analizzare l'andamento della coda.

## Suggerimenti per la simulazione con AnyLogic

1. Usiamo quattro *source* diverse per i diversi tipi di pezzi e introduciamo un nuovo tipo di agente per registrare il tipo di pezzo in `agent.type` e altre informazioni che permettono di gestire percorsi differenziati (come nell'esercizio DE2.4).

**Soluzione 1:** Usiamo due attributi: `agent.nextMachine` per definire la successiva macchina che l'entità passante dovrà visitare, e `agent.nextProcessingTime` per memorizzare il relativo tempo di lavorazione. Tali attributi sono inizializzati dalla *source* con la prima macchina e il primo tempo, e aggiornati opportunamente in uscita dalle diverse macchine (eventi *onExit*). Notare che questa soluzione, sebbene relativamente semplice da implementare, distribuisce le istruzioni di controllo su più blocchi e potrebbe essere uno svantaggio qualora si volesse cambiare il processo e realizzare altre sequenze di lavorazione (bisognerebbe intervenire su più blocchi). Inoltre, alcune sequenze (ad esempio più passaggi sulla stessa macchina in fasi diverse) non sono direttamente realizzabili.

**Soluzione 2:** Per evitare di distribuire l'intelligenza del sistema su più blocchi (e avere la possibilità di effettuare controlli più flessibili), possiamo centralizzare il controllo in un solo blocco, in questo caso il selettore a 5. In questo caso abbiamo scelto di selezionare l'uscita direttamente con un numero (impostare *Use: Exit number*) che viene calcolato dalla funzione `setProcessingInfo` che riceve come parametro un agente e, in base al valore dei suoi attributi, stabilisce quale è la macchina successiva e il tempo di lavorazione. A tal fine gli agenti riportano come attributi `agent.type`, `agent.stepCurrent` (indica qual è il numero d'ordine del passo di lavorazione corrente) e `agent.processingTime` (relativo tempo di lavorazione): la funzione incrementa di uno l'attributo `stepCurrent` (per indicare che siamo pronti per la fase di lavorazione successiva) e, in base al valore di `type` e `stepCurrent`, stabilisce l'uscita corretta (cioè la macchina su cui passare) e il relativo tempo di lavorazione.

**Attenzione!** La funzione `setProcessingInfo` modifica il valore `agent.stepCurrent` rendendo la scelta dell'uscita *volatile*, cioè, la scelta dipende da un fattore che viene modificato nel corpo della funzione. Dalla versione 8.9.8, è quindi necessario, nel selettore a 5, selezionare la casella *Choice is stochastic or volatile*. Infatti, AnyLogic potrebbe, per motivi interni, eseguire più volte la funzione in corrispondenza dello stesso passaggio dello stesso agente, portando a un comportamento indeterminato (nel nostro caso, il valore `agent.stepCurrent` sarebbe aumentato di più di un'unità ad ogni passaggio). In alternativa, si potrebbe spostare l'incremento del valore `agent.stepCurrent` fuori dalla funzione, ad esempio in corrispondenza dell'evento *on enter* del delay `mH` corrispondente all'handler (in questo caso la scelta non è più *volatile* e pertanto non serve selezionare *Choice is stochastic or volatile*).

Per variare il numero di serventi e effettuare le analisi richieste, usiamo caselle di testo legate alla capacità degli oggetti `mA` e `mB` (in alternativa all'uso di slider visto in altri esercizi): potremo osservare che il sistema diventa bilanciato (lunghezza delle code simili sulle varie macchine) con 2 serventi su `mA` e su `mB`.

2. Proponiamo due possibili implementazioni.

Nella **prima implementazione**, i guasti sono simulati come un'entità passante speciale, generata da un apposito blocco *source* e immesso nella stessa coda di lavorazione. Si usa un tipo di agente personalizzato dotato di attributo `agent.isGuasto` impostato a `true` dal *source* dei guasti e a `false` dal *source* dei pezzi: le entità passanti che rappresentano i guasti hanno priorità nella coda e tengono impegnata la macchina per un tempo opportuno.

Nella **seconda implementazione** facciamo uso dell'oggetto *Event* dalla palette *Agent* e dell'oggetto *Hold* della palette *PML* (non sarà necessario l'uso di uno speciale tipo di agente). L'oggetto *hold* viene usato per bloccare, in modo condizionato, gli ingressi alla macchina (vogliamo bloccare l'accesso all'occorrenza di un guasto per la durata della riparazione). Usiamo quindi un primo *event* chiamato *guasto* che scatta ogni *n* minuti, con *n* distribuito secondo quanto richiesto. Allo scopo impostiamo:

- *Trigger type* = *Timeout* (l'evento scatta al termine di un tempo stabilito, con un conto alla rovescia);
- *Mode* = *cyclic* (il conto alla rovescia si ripete ciclicamente);
- *First recurrence time* e *recurrence time* impostati in modo da ottenere la distribuzione richiesta.

Notare che, visto che abbiamo arrivi casuali e indipendenti possiamo usare, in modo equivalente ma non estensibile ad altre distribuzioni dei tempi di interarrivo dei guasti, *Trigger type* = *rate* con *rate* =  $1/200$ .

Allo scattare dell'evento *guasto*, si chiude l'oggetto *hold* con `hold.block()` e si abilita un altro evento, l'evento *fineGuasto*, con `fineGuasto.restart()`. L'evento *fineGuasto* è impostato con:

- *Trigger type* = *Timeout* (l'evento scatta al termine di un tempo stabilito);
- *Mode* = *user control* (il conto alla rovescia inizia su comando esplicito, quando viene richiamata la funzione `restart()`);
- *Timeout* impostato in modo da ottenere la distribuzione richiesta per il tempo di intervento su guasto. Allo scattare dell'evento, la *action* associata riabilita l'accesso alla macchina con `hold.unblock()`.

3. Per modellare il sistema, usiamo un'unica coda e un blocco *selectOutput* che devia l'entità passante in uscita dalla coda sulla terza macchina solo se le prime due sono entrambe occupate. Notare che la coda blocca l'uscita delle entità passanti se tutti i *delay* a valle del selettore sono occupati, pertanto se un'entità arriva al *selectOutput*, siamo sicuri che almeno una macchina a valle è libera<sup>1</sup>. Notiamo che la condizione di funzionamento del selettore è *volatile/stocastica* in quanto potrebbe succedere che nello stesso istante di simulazione ci sono eventi di uscita dalla coda e di uscita dai *delay*, pertanto il comportamento del sistema dipende dall'ordine in cui gli eventi simultanei vengono eseguiti nel *run time* che non è ben definito e potrebbe variare: è necessario quindi selezionare *Choice is stochastic or volatile*.

### Esercizio di riepilogo

Uno studio dentistico riceve due tipi di pazienti. Il primo tipo riguarda pazienti che hanno bisogno di otturazioni, il secondo riguarda pazienti che hanno bisogno di revisioni ortodontiche. Lo studio riceve a partire dalle 9:30 e fino alle 19:00. Di solito si lavora su appuntamento. Tuttavia i pazienti non sono solitamente puntuali e, inoltre, la maggior parte dei pazienti preferisce gli orari compresi tra le 12:00 e le 14:00 e tra le 16:00 e le 19:00 (ore di punta). Si è osservato che gli arrivi dei pazienti del primo tipo avvengono a intervalli distribuiti secondo un'esponenziale di media 15 minuti nelle ore di punta e 30 minuti nelle restanti ore; gli arrivi dei pazienti del secondo tipo avvengono secondo una distribuzione esponenziale di media 30 minuti nelle ore di punta e 45 minuti nelle restanti ore. Tutti i pazienti sono curati da un solo dentista, operativo dalle 10 fino al massimo alle ore 23:00. All'arrivo, i pazienti sono accolti da un'infermiera in accettazione, che li serve in ordine FIFO con tempi distribuiti secondo una normale di media 4 minuti e deviazione standard 30 secondi (minimo 30 secondi), e individua alcuni pazienti (mediamente il 15%) che non possono essere curati subito, ma sono rimandati ad un altro giorno. Per i restanti pazienti, quelli del primo tipo ricevono cure con durata distribuita secondo una normale di media 12 minuti e deviazione standard di 5 minuti. Inoltre il 30% di questi pazienti viene sottoposto a un trattamento con lampada polimerizzante della durata di 10 minuti. Il numero di lampade è sufficiente per trattare tutti i pazienti parallelo, e dopo il trattamento i pazienti si rimettono in attesa che il dentista si liberi per la seconda fase delle cure, ma questa volta hanno priorità sugli altri pazienti. I pazienti del secondo tipo ricevono cure con durata distribuita secondo una gamma di media 8 minuti e varianza 2. Il 20% dei pazienti del secondo tipo, hanno cure molto veloci distribuite secondo una triangolare con valore minimo di 1 minuto, valore massimo di 4 minuti e moda di 2 minuti: in questo caso i pazienti hanno priorità. Si fornisca un modello di simulazione del sistema descritto e si dica come si può utilizzare per poter studiare mediante simulazione il tempo mediamente speso nel sistema dai pazienti del primo tipo nei due casi, dai pazienti del secondo tipo "normali" e dai pazienti del secondo tipo con priorità, nonché il numero dei pazienti in coda, in particolare dopo le 20:00, considerando la possibile assunzione di altri dentisti. Descrivere l'eventuale possibilità di utilizzare grafici e controlli a tempo di simulazione che possano facilitare l'analisi richiesta.

---

<sup>1</sup> Attenzione: nella versione 6.5.0 di AnyLogic questo blocco automatico non avviene: se l'uscita di una coda è collegata a un *selectOutput*, un'entità in uscita dalla coda non viene *mai* bloccata perché viene *persa* la relazione con i *delay* a valle: quindi, se entrambi i *delay* oltre il *selectOutput* sono occupati, l'entità passante (ricevuta comunque nel *selectOutput*) cerca di uscire dal *selectOutput* (che non offre funzionalità di buffering, chi entra esce immediatamente) ma si trova il percorso chiuso per l'indisponibilità delle macchine, e viene generato un errore di runtime che interrompe la simulazione. Questo complica la modellazione di sistemi in cui si ha un'unica coda a monte di più *delay*, forzando, anche per sistemi semplici come quello in analisi, l'utilizzo di blocchi specializzati (in particolare *Service* in combinazione con *Resource Pool*, vedere l'*Help* in caso di necessità) opportunamente personalizzati.