
Progetto di Sistemi Complessi: Modelli e Simulazione

SUPERMARKET SIMULATION

TONELLI LIDIA LUCREZIA - 813114

MARCO GRASSI - 829664

GIANLUCA GIUDICE - 830694

Contents

1	Introduzione	2
2	Stato dell'arte	3
3	Descrizione del modello	4
3.1	Framework adottato	4
3.2	Sistema	4
3.3	Agente di tipo Cliente	4
3.3.1	Workflow	5
3.4	Agente di tipo Cassa	6
3.4.1	Tipi di cassa	6
3.4.2	Workflow	9
3.5	Ambiente	13
3.5.1	Caratteristiche dell'ambiente	13
3.5.2	Interazione tra agenti	13
3.6	Considerazioni sul modello	13
3.7	Parametri del modello	13
3.7.1	Basket size	15
4	Implementazione dei comportamenti e delle strategie	16
4.1	Fase di scelta della coda	16
4.2	Fase di attesa in coda e jockeying	18
5	Simulazione	19
6	Analisi dei risultati	20
7	Conclusioni	21

Chapter 1

Introduzione

In questo progetto è stato costruito un modello basato su agenti con lo scopo di simulare il comportamento dei clienti all'interno di un supermercato durante la fase di scelta della coda relativa a diversi tipi di casse.

Un grande supermercato è composto da molte casse, ognuna di queste ha un comportamento diverso. Come vedremo in dettaglio nel capitolo 3, esistono diversi tipi di casse: la cassa standard, in cui si trova una cassiera che passa i prodotti provenienti dal nastro, la cassa self-service, in cui è il cliente stesso a scannerizzare i prodotti e la cassa self-scan, comparsa negli ultimi anni, in cui il cliente deve soltanto pagare, in quanto ha già effettuato la scannerizzazione dei prodotti man mano che li ha raccolti nel carrello. La configurazione delle casse nel supermercato può prevedere che ogni cassa abbia la sua coda dedicata, oppure che tante casse condividano la stessa coda.

Dal momento che in un grande supermercato sono presenti più clienti che casse, è fondamentale l'utilizzo di code per gestire la grande quantità di clienti. Un supermercato può essere considerato a tutti gli effetti un sistema complesso in quanto si verificano diversi aspetti che considerati contemporaneamente fanno emergere un comportamento complessivo difficilmente prevedibile, tra questi:

- Flusso di clienti in ingresso variabile (che influisce sulle persone in coda)
- Numero di prodotti che un cliente acquista durante la spesa (che influisce sul tempo passato in cassa, scatenando eventualmente il cambio della coda da parte di altri clienti)
- Numero di casse aperte contemporaneamente nel supermercato
- Strategia di scelta della coda dei clienti (un cliente può avere diversi criteri per la scelta della coda)
- Strategia di cambio della coda dei clienti

Lo scopo di questo lavoro è costruire un modello basato su agenti per simulare il comportamento del supermercato. Dopo una prima fase di validazione, sfruttando i pochi dati a disposizione, vengono considerati diversi casi "what-if" con lo scopo di sperimentare diverse configurazioni di casse e strategie di scelta della coda per una gestione ottimale del flusso dei clienti.

Chapter 2

Stato dell'arte

In questo capitolo viene introdotto il problema, il dominio di riferimento e l'approccio adottato per la risoluzione.

Chapter 3

Descrizione del modello

3.1 Framework adottato

Mesa è un framework in Python usato per la modellazione basata su agenti (ABM). Permette di creare il modello con componenti *built-in* come griglie spaziali e scheduler di agenti, e di visualizzare i componenti del modello con un'interfaccia browser. Sfruttando Mesa è possibile definire sia l'ambiente che gli agenti estendendo le relative classi messe a disposizione dal framework. Mesa infine comprende strumenti per l'analisi del modello creato.

Nel nostro modello di supermercato, il modello vero e proprio di Mesa è la classe **Supermarket** e gli agenti sono i clienti, classe **Customer**, e le casse, classe **CashDesk**.

3.2 Sistema

Supermercato non predicibile, presente una componente stocastica.

Il sistema è eterogeneo, due tipi di agenti. Sistema multiagente.

Come comunicano

Come avviene l'interazione

Descrizione dell'ambiente

3.3 Agente di tipo Cliente

I clienti sono gli agenti principali che compongono il modello e interagiscono con l'ambiente per raggiungere l'obiettivo di fare la spesa; per portare a termine questo compito l'agente esegue alcuni macro-step in cui è necessaria anche una fase di pianificazione e valutazione della bontà (*utility function*) della scelta.

Dal punto di vista dell'agente di tipo cliente, la fase più complicata è quella in cui ha raggiunto il target basket size e deve quindi mettersi in una coda per attendere il suo turno in cassa. A questo punto il **goal dell'agente** è: essere servito da una cassa nel minor tempo possibile, pur dovendo necessariamente passare per una coda. Questa fase richiede una pianificazione da parte dell'agente in quanto la coda viene scelta in modo ottimale cercando di minimizzare il tempo passato in coda, ciò avviene basandosi su diverse strategie. Ragionamenti analoghi vengono fatti nel momento in cui l'agente già in coda decide di fare jockeying, ovvero cambiare la coda attuale perchè questa azione porta al raggiungimento del goal.

Da queste considerazioni è possibile classificare l'agente di tipo Cliente come "utility-based", secondo la tassonomia proposta Russel-Norvig [?]; viene qui sotto riportata l'architettura di un agente "utility-based":

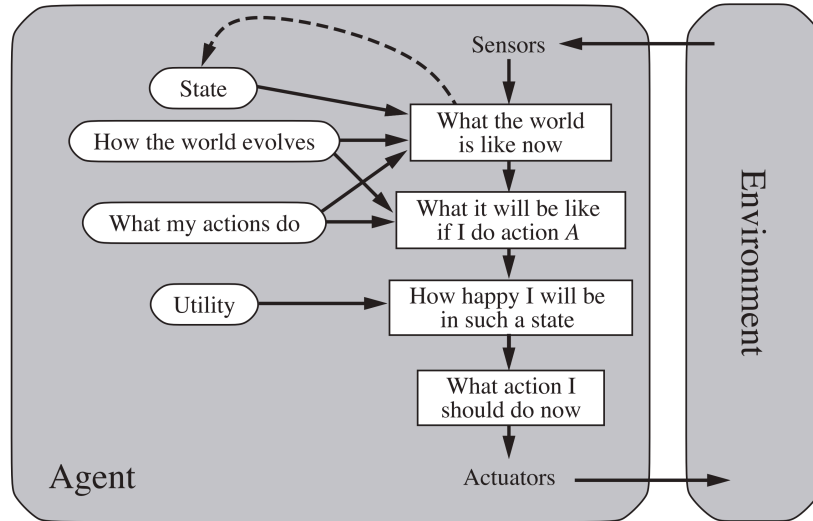


Figure 3.1: Agente di tipo utility-based.

Da questa immagine possiamo notare come ci sia una corrispondenza tra l'architettura proposta e quella dell'agente di tipo cliente. Infatti l'agente pianifica le sue azioni e valuta la bontà dell'azione di scegliere una coda per mezzo di un'utility function.

3.3.1 Workflow

I macro-step che l'agente deve seguire per il raggiungimento del goal, che coincidono con gli stati associati all'agente durante la simulazione, sono:

1. **Attesa all'entrata del supermercato:** in questo stato il cliente si mette in attesa finché non è possibile entrare nel supermercato.
2. **Fase di shopping:** il cliente è entrato e può iniziare a "fare la spesa", con questo si intende raggiungere il numero di prodotti desiderato, infatti nella simulazione viene considerato il *basket size*, un'astrazione del carrello modellato attraverso un numero interno. Il *basket size* è il target da raggiungere nella fase di shopping, a ogni step il numero di prodotti aumenta di una certa quantità (la velocità di shopping è un parametro del modello descritto nella sezione 3.7).

La *basket size* viene generata stocasticamente secondo una distribuzione esponenziale governata dal parametro $\lambda = 0.07361$. La scelta di questo parametro viene giustificata nella sezione 3.7.

3. **Scelta della coda:** in questo stato il cliente ha finito di fare la spesa e vuole mettersi in coda ad una cassa. Dal momento che sono disponibili più code e ognuna di queste avrà molto probabilmente altri clienti già in coda, il cliente deve scegliere in quale coda andare in base alla coda che lui considera migliore.

Formalmente viene definita una strategia di scelta della coda in base a una funzione: la *utility function*. Il cliente sceglie quindi la coda che minimizza questa funzione:

$$q^* = \underset{q \in Q}{\operatorname{argmin}} f(q) \quad (3.1)$$

Esperimenti
sul covid

dove Q è l'insieme delle code dedicate e f varia a seconda del tipo di strategia che il cliente può utilizzare, ognuna delle quali modella una diversa strategia per la scelta della coda.

4. **Attesa in coda e jockeying:** una volta che il cliente ha scelto la coda deve aspettare il proprio turno per essere servito. Nel lavoro di Tomasz Antczak e altri [?] viene suggerito come sviluppo futuro lo studio del fenomeno di *jockeying*: con questa espressione si intende cambiare la coda che è stata scelta inizialmente in quanto un'altra risulta essere più conveniente, ad esempio perchè più scorrevole.

Ad ogni timestamp l'agente considera le due code adiacenti (parametro spiegato nella sezione 3.7 e scelto a priori per il modello) alla propria e per ognuna di queste ricalcola la coda migliore secondo la 3.1. A questo punto l'agente valuta se per lui è conveniente cambiare la coda o rimanere in quella dove è già presente. Si noti come per effettuare questo confronto è necessario utilizzare l'approccio della 3.1 sulla coda in cui l'agente è in quel momento, andando ad escludere i clienti che si sono inseriti in coda dopo di lui. Una volta fatta questa valutazione, nel caso in cui sia per lui conveniente cambiare coda, il cliente la cambia in modo stocastico secondo una certa distribuzione di probabilità spiegata nella sezione 4, che è maggiore più il cambio di coda risulta conveniente.

La scelta modellistica di effettuare jockeying stocasticamente e in funzione dell'effettivo guadagno di tempo è stata fatta per cercare di avere una rappresentazione il più fedele possibile al mondo reale. Infatti nella realtà un cliente al supermercato potrebbe scegliere di non cambiare la coda nel momento in cui ha un guadagno minimo in quanto questo richiede uno sforzo fisico che non tutti vogliono spendere nella realtà. Inoltre questa operazione richiede che gli agenti controllino continuamente le casse vicine, anche questo non è necessariamente vero in quanto dispendioso di energia. Infatti nel lavoro di Tomasz Antczak e altri [?] viene fatto notare come questo comportamento non sia molto frequente, tuttavia seppur essendo poco frequente noi lo consideriamo un fattore importante da modellare, adottando le dovute precauzioni e sfruttando la stocasticità dell'azione.

5. **Attesa alla cassa:** in questa fase il cliente è arrivato alla cassa, essendo arrivato con lo stato precedente in prima posizione della coda. Da questo punto in poi verrà servito dall'agente di tipo cassa che avrà la responsabilità di processare il basket size del cliente. Dal momento che ci sono diversi tipi di casse, questa fase dipende dall'agente di tipo cassa e non dal cliente. Il cliente deve attendere la fine dell'elaborazione della spesa e quindi uscire dal negozio.

Il workflow dell'agente di tipo cliente viene astratto e riassunto da questa immagine:

3.4 Agente di tipo Cassa

3.4.1 Tipi di cassa

Gli agenti di tipo cassa hanno l'unica funzione di servire i clienti che si mettono in coda per pagare la spesa; sono di 4 tipi e differiscono per modalità di elaborazione della spesa del cliente e per velocità.

1. **Cassa standard:** questa cassa rappresenta la normale cassa che si trova solitamente in tutti i supermercati, all'interno della quale lavora un cassiere che si occupa di passare i prodotti del carrello del cliente sul nastro.

Il tempo di servizio di un cliente per questo tipo di cassa è formato da due tempi diversi: il **transaction-time**, che è il tempo vero e proprio durante il quale avviene

Qua parlare dell'architettura delle casse (tipo di agente, immagine ecc..)

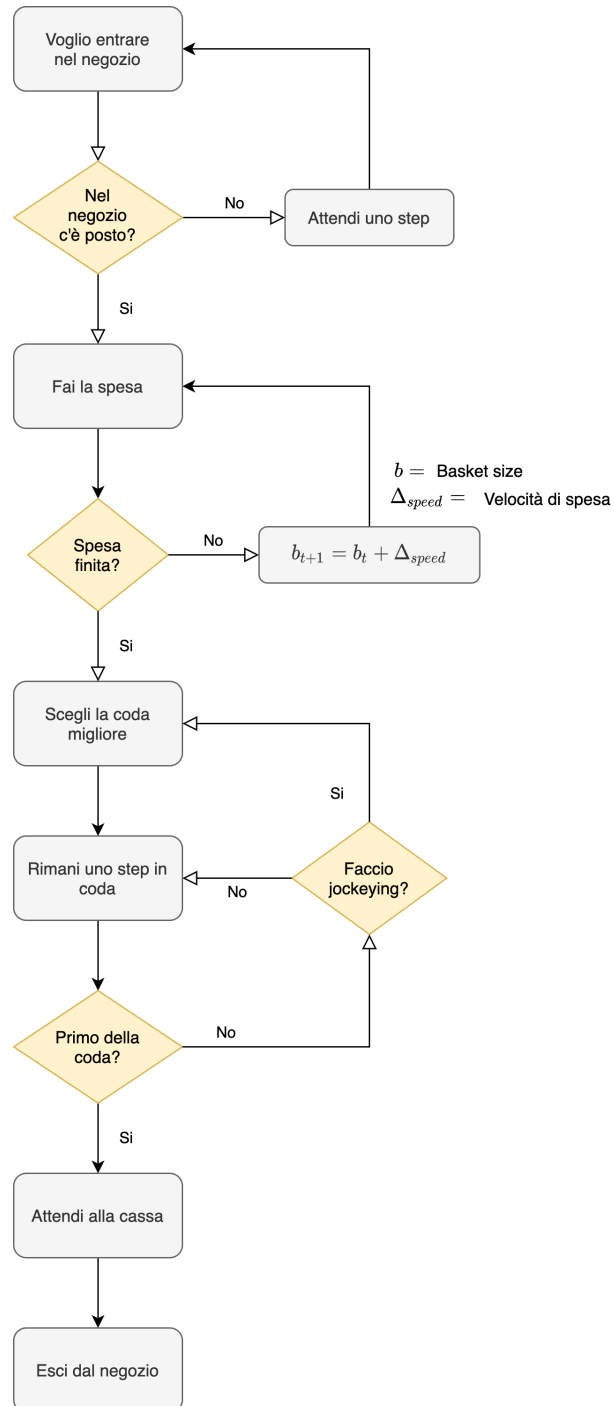


Figure 3.2: Workflow dell'agente di tipo customer.

la transazione, e il **break-time**, che è il tempo che passa tra il servizio di un cliente e di un altro ed include anche la fase in cui il cliente insacchetta i prodotti acquistati. Il transaction-time è stato stimato con una regressione di potenza e il break-time con una distribuzione Gamma. In particolare, per un cliente $c_i \in q$, i due tempi sono calcolati in questo modo:

$$\text{transaction-time}_i = e^{a \log(\text{basket-size}(c_i)) + b} \quad (3.2)$$

$$\text{break-time}_i = \frac{\beta^\alpha \text{basket-size}(c_i)^{\alpha-1} e^{-\beta \text{estimate-basket-size}(c_i)}}{\Gamma(\alpha)} \quad (3.3)$$

Dove Γ è la funzione **gamma** e corrisponde a:

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx \quad \forall z \in \mathbb{C} \quad (3.4)$$

Queste stime sono state prese dall'articolo [?] e in particolare i parametri $a, b, \alpha, \beta \in \mathbb{R}$ sono:

$$a = 0.6984, \quad b = 2.1219, \quad \alpha = 3.074209, \quad \beta = \frac{1}{4.830613} \quad (3.5)$$

2. **Cassa self-service:** alla cassa self-service il cliente provvede da solo al passaggio dei prodotti del carrello allo scanner, per cui non è presente un cassiere. Si suppone che il cliente non sia veloce quanto un cassiere esperto a passare i prodotti sullo scanner, per questo la velocità di processing per questa cassa è stata divisa per un fattore 1.5, uno dei parametri descritti nella sezione 3.7. A causa della velocità ridotta, di solito nei supermercati è posto un limite di *basket-size* per accedere alla cassa self-service; nel nostro modello questo limite è posto a 15, per cui se un cliente ha più di 15 elementi nel carrello e non va nel self-scan, deve per forza andare alla cassa standard.

Anche in questo caso il tempo di servizio è diviso in transaction-time e break-time, solo che, a differenza della cassa normale, il break-time risulta più lungo in quanto il cliente inizia ad insacchettare solamente dopo aver passato i prodotti allo scanner, invece nella cassa standard può iniziare già mentre il cassiere passa i prodotti. In questo caso, dunque, il transaction-time e il break-time sono stati stimati entrambi con una regressione di potenza:

$$\text{transaction-time}_i = e^{a \log(\text{basket-size}(c_i)) + b} \quad (3.6)$$

$$\text{break-time}_i = e^{c \log(\text{basket-size}(c_i)) + d} \quad (3.7)$$

I parametri in questo caso sono:

$$a = 0.6725, \quad b = 3.1223, \quad c = 0.2251, \quad d = 3.5167 \quad (3.8)$$

3. **Cassa self-scan:** la cassa self-scan si differenzia totalmente dalle prime due descritte, il cliente che voglia utilizzarla infatti, deve deciderlo già al momento dell'entrata nel supermercato, in quanto deve scannerizzare i prodotti man mano che fa la spesa; una volta arrivato alla cassa self-scan, egli deve semplicemente effettuare il pagamento, saltando quindi le fasi di scanner e insacchettamento che caratterizzano le casse standard e self-service. La velocità di elaborazione della spesa in questo caso è praticamente

nulla, in particolare un cliente che inizia il pagamento a una cassa self-scanner, lo termina allo step successivo; il tempo di pausa per questa cassa è ugualmente nullo, dato che un cliente paga ed entra immediatamente nella fase di uscita dal negozio.

Chiaramente per evitare che vengano rubati prodotti, i supermercati decidono di effettuare dei controlli random sui clienti che scelgono di usare le casse self-scan. A questo scopo è stata implementata la cassa di tipo riservato.

4. **Cassa riservata:** questa cassa è identica nel funzionamento a una cassa standard, ciò che cambia è che è riservata alla riletatura della spesa dei clienti self-scan, che avviene in maniera random. Nel momento in cui un cliente si avvicina ad una cassa self-scan, viene estratto un numero che determina la **rilettura parziale** o la **rilettura totale** della sua spesa; la riletatura parziale consiste nell'estrazione di 10 prodotti dal carrello e nella verifica che essi appartengano alla lista contenuta nello scanner del cliente, invece la riletatura totale consiste appunto nella riletatura completa della spesa di esso. Le probabilità di riletatura sono parametri del modello ed attualmente corrispondono allo 7.5% per la riletatura parziale e allo 2.5% per la riletatura totale.

Il funzionamento e i tempi di servizio sono esattamente gli stessi della cassa standard, considerando un basket-size completo per la riletatura totale e un basket-size di soli 10 elementi per la riletatura parziale.

Tutti i tipi di cassa attraversano 3 fasi: fase di attesa di un nuovo cliente, fase di elaborazione della spesa del cliente, fase di completamento della transazione.

Code Nel modello vengono considerati i casi di:

- **Code parallele:** per questa disposizione ad ogni cassa è associata una coda. Nel momento in cui la cassa è libera il cliente in cima alla coda viene spostato alla cassa per procedere con la fase di elaborazione della spesa.
- **Coda condivisa:** nota anche come *N-fork* [?], più casse condividono la stessa coda, nel momento che una qualsiasi delle casse risulta libera il cliente in posizione utile procede alla cassa. Utilizzando questa disposizione non viene ammesso jockeying.

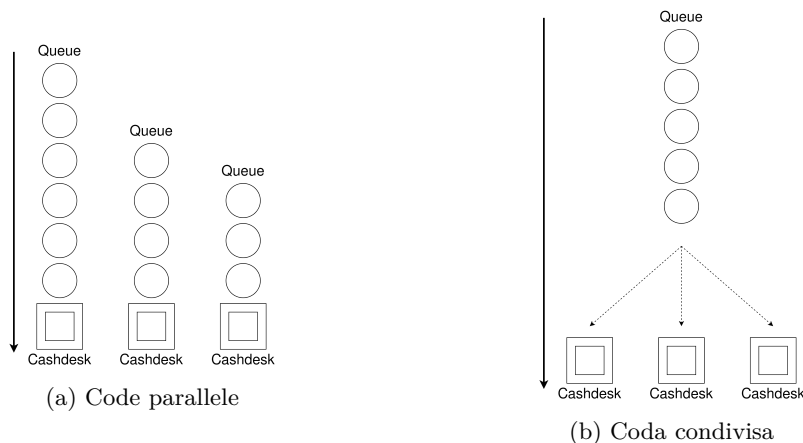


Figure 3.3: Tipi di code

3.4.2 Workflow

Cassa standard All'inizio della simulazione, e in generale quando la cassa ha terminato la transazione di un cliente o quando non ci sono clienti in coda, la cassa si trova nella **fase**

di attesa di un nuovo cliente. L'agente cassa chiede alla coda a cui è associata il primo cliente in fila, se ce n'è, quindi compie 4 passi:

- Cambia lo stato al cliente che sta servendo, impostandolo in fase di attesa alla cassa
- Muove il cliente di fianco alla propria postazione, spostandolo quindi dalla coda
- Cambia il proprio stato impostandolo in fase di elaborazione
- Fa avanzare tutti i clienti che erano in coda dopo il cliente che viene servito

Nella **fase di elaborazione della spesa del cliente**, la cassa simula il passaggio dei prodotti sullo scanner, diminuendo il *basket-size* del cliente ad ogni passo di una quantità che dipende dalla velocità di processing, parametro del modello. Quando la transazione è completata, ovvero quando il *basket-size* del cliente è nullo, fa uscire il cliente dal negozio e cambia il proprio stato, entrando nella **fase di completamento della transazione**.

In questa ultima fase la cassa ha terminato il servizio di un cliente, quindi si rimette nel primo stato, in fase di attesa di un nuovo cliente.

Il workflow della cassa standard viene riassunto nell'immagine 3.4.

Cassa self-service Il workflow della cassa self-service è uguale in tutto a quello della cassa standard, le uniche differenze sono che esiste una coda condivisa ogni 8 casse self-service (che però funziona allo stesso modo della coda della cassa standard) e che la velocità di elaborazione della spesa è più piccola di 1.5, un parametro del modello.

Cassa self-scan Nella **fase di attesa di un nuovo cliente**, se ci sono clienti nella coda condivisa, la cassa self-scan prende il primo cliente della coda; estrae quindi un numero e , con lo 7.5% di probabilità lo manda alla cassa riservata per una **rilettura parziale**, con lo 2.5% di probabilità lo manda alla cassa riservata per una **rilettura totale** e invece con il 90% di probabilità gli fa fare il pagamento normale. Nel caso di rilettura parziale o totale, la cassa sposta il cliente alla cassa riservata, fa avanzare i clienti in coda e rimane nella fase di attesa di un nuovo cliente, pronta per prendere il nuovo cliente della coda.

Nel caso non debba avvenire una rilettura la cassa compie queste 4 azioni:

- Cambia lo stato al cliente che sta servendo, impostandolo in fase di attesa alla cassa
- Muove il cliente di fianco alla propria postazione, spostandolo quindi dalla coda
- Cambia il proprio stato impostandolo in fase di elaborazione
- Fa avanzare tutti i clienti che erano in coda dopo il cliente che viene servito

Nella **fase di elaborazione della spesa del cliente**, la cassa effettua la vera e propria transazione, la differenza con la cassa standard è la velocità di elaborazione della spesa, che in questo caso è uguale alla *basket-size* del cliente; il cliente infatti deve soltanto pagare, in quanto ha già scannerizzato i prodotti durante la sua spesa.

Nell'ultima fase la cassa ha terminato il servizio di un cliente, quindi si rimette nel primo stato, in fase di attesa di un nuovo cliente.

Il workflow della cassa self-scan viene riassunto nell'immagine 3.5.

Cassa riservata Nella **fase di attesa di un nuovo cliente** la cassa riservata è pronta a servire i clienti in coda, però prima controlla che ci siano in coda clienti a cui è stata assegnata una rilettura parziale della spesa, dandogli la precedenza in quanto la rilettura parziale è più veloce di quella totale.

Scelto quindi il cliente, questa cassa passa per le stesse fasi della cassa standard.

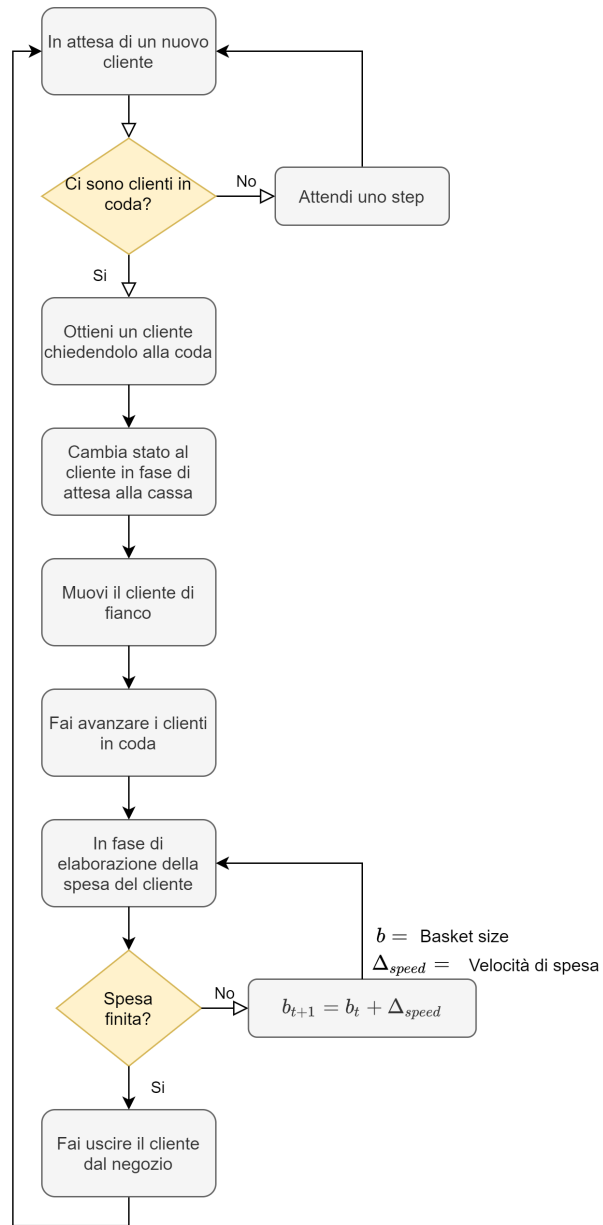


Figure 3.4: Workflow dell'agente di tipo cassa standard.

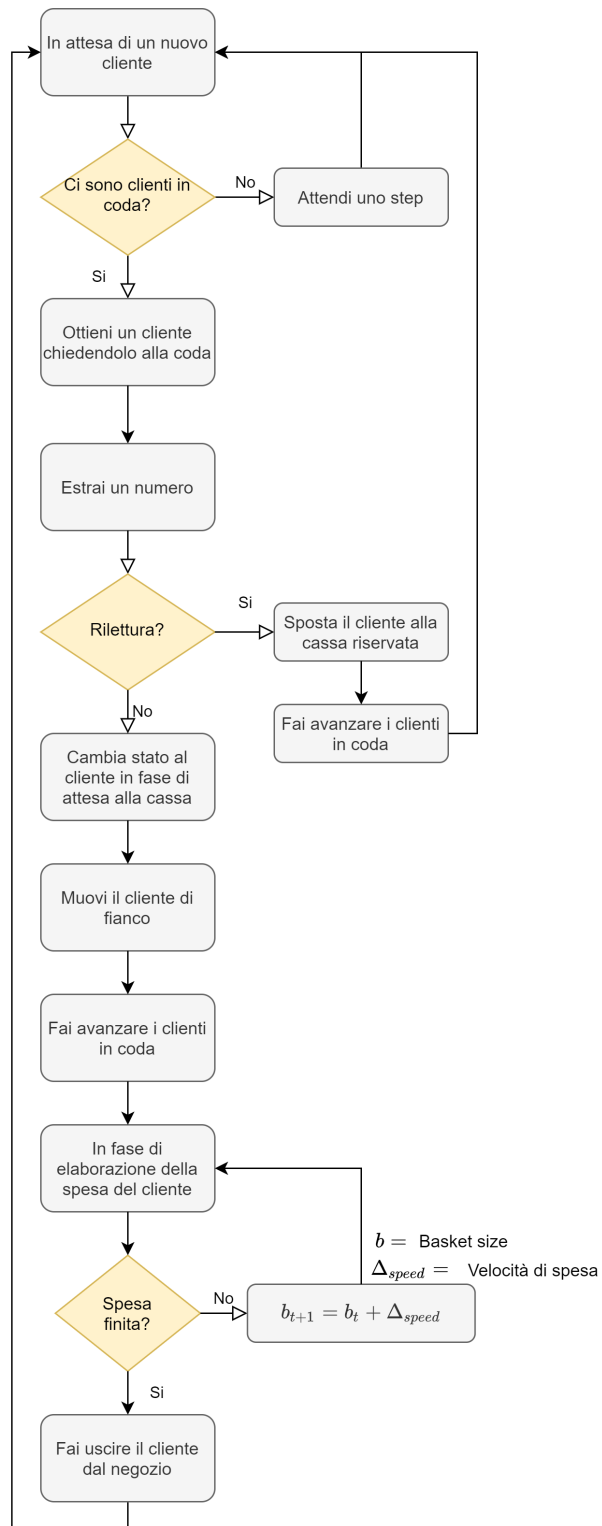


Figure 3.5: Workflow dell'agente di tipo cassa self-scan.

Il workflow della cassa riservata viene riassunto nell'immagine 3.6.

3.5 Ambiente

La classe **Supermarket** si occupa di inizializzare la griglia che verrà poi mostrata in fase di simulazione sull'interfaccia, inizializzare le casse, l'ambiente e i clienti.

Per inizializzare la griglia, l'ambiente viene diviso in zone: zona d'entrata, zona di shopping, zona casse normali, zona casse self-service, zona casse self-scan. Questa divisione permette una gestione più semplice dello spazio e dei movimenti degli agenti. Ogni zona ha come parametri la dimensione o il numero di casse che deve contenere, questi parametri vengono inizializzati nella classe **main**, come si vedrà nella prossima sezione.

Ogni zona è responsabile della propria costruzione, ovvero del proprio collocamento nella griglia dell'interfaccia in base alle proprie dimensioni ed eventualmente del posizionamento delle casse che contiene. Inoltre ogni zona è responsabile dei movimenti dei clienti: se un cliente vuole muoversi da una zona all'altra oppure mettersi in coda nelle casse di una zona, è la zona di destinazione che fornisce il metodo per posizionarsi correttamente in essa.

Ad ogni step della simulazione, il modello crea dei clienti secondo la distribuzione data (si veda la sezione 3.7) e li posiziona nella **Entering Zone** del supermercato in coordinate random, dunque si occupa della attivazione e disattivazione delle casse standard in base al numero di clienti presenti nel negozio in quello step, secondo dei parametri che si vedranno nella prossima sezione. Quindi il modello chiama gli scheduler degli agenti, clienti e casse, e fa eseguire i loro step.

Parlare di quanto dura uno step e quanto dura una simulazione

3.5.1 Caratteristiche dell'ambiente

- Accessibile vs. Inaccessibile - Deterministico vs. Non-deterministico - Episodicità vs. Non-episodicità - Statico vs. Dinamico - Discreto vs. Continuo

3.5.2 Interazione tra agenti

3.6 Considerazioni sul modello

Il modello è stato sviluppato in modo da essere flessibile per permettere con semplicità future espansioni e cambiamenti. Gli aspetti più dinamici del modello, quindi il comportamento degli agenti e le varie modalità di scelta di una coda e la capacità di un agente Cliente di effettuare *jockeying* sono stati realizzati lasciando all'utilizzatore piena possibilità di modifica, ampliamento o di non utilizzo. A livello di architettura del software questo si traduce nell'implementazione dei pattern:

- **Strategy**: usato per definire diverse modalità di scelta della coda e diversi comportamenti di *jockeying*. Nuovi comportamenti possono essere aggiunti rispettando un'interfaccia standard. Permette inoltre se necessario cambiamenti di comportamento a run time.
- **State**: usato per definire tramite una macchina a stati finiti i comportamenti degli agenti Cliente e Cassa. Nuovi stati possono essere aggiunti con facilità e collegati ad altri stati tramite transizioni. La divisione in stati rende semplice ragionare sul comportamento di un agente.

TODO forse aggiungere link a pattern GOF, non so come citare correttamente

3.7 Parametri del modello

Casse adiacenti considerate

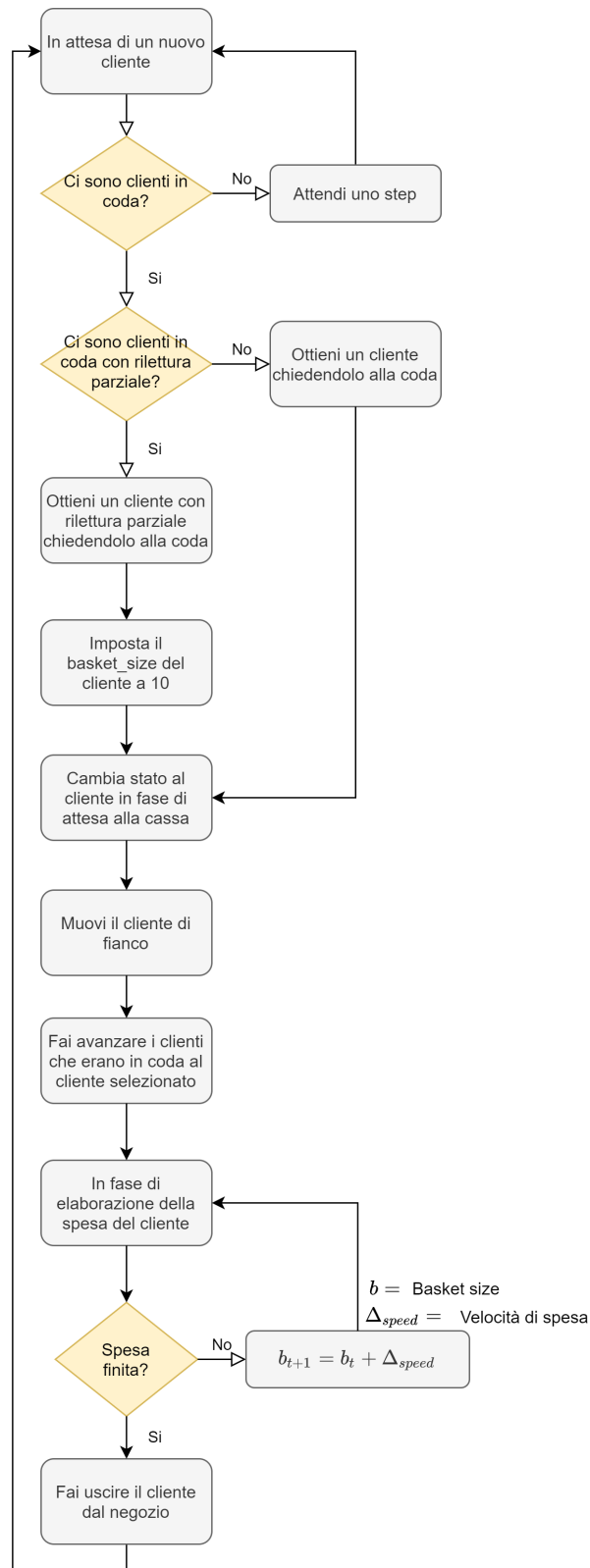


Figure 3.6: Workflow dell'agente di tipo cassa riservata.

Inizializzazione della griglia e parametri per creare casse (numero casse, coda condivisa o no, velocità di processing e parametro per rallentare la velocità della self-service), customer, basket size (analisi dei dati di Gianluca).

3.7.1 Basket size

Citare paper con i dati (distribuzione, parametro lambda scelto).

Chapter 4

Implementazione dei comportamenti e delle strategie

In questo capitolo verranno analizzate le strategie di scelta della coda da parte del cliente. In particolare queste strategie determinano la coda che il cliente deciderà di seguire durante la fase di scelta della coda e durante la fase di attesa in coda e jockeying.

Come introdotto all'inizio della relazione, obiettivo di questo progetto è analizzare le varie configurazioni di casse all'interno del supermercato, al variare del tipo di casse, della quantità di clienti presenti nel negozio e alla strategia di scelta della coda dei clienti, pertanto è necessario disporre di strategie che sfruttano calcoli basati su variabili diverse: le variabili in gioco saranno il numero di elementi nel carrello, il numero di persone e il tempo medio di attesa.

Forse qui serve parlare dei paper (quello dei polacchi e quello delle fork)

Se un cliente entra nel supermercato con l'intenzione di usare le casse self-scan, chiaramente non sarà dotato di strategie di scelta della coda e di jockeying, dal momento che le casse self-scan hanno sempre un'unica coda condivisa. Anche nel caso in cui le casse standard abbiano un'unica coda condivisa, come capita in molti negozi, il cliente non avrà una strategia di scelta della coda o di jockeying. Al contrario invece, dopo la fase di shopping il cliente dovrà scegliere la coda tra le code disponibili delle casse standard e delle casse self-service; una volta che esso si accoda in una cassa standard, può decidere di cambiare coda e quindi effettuare il jockeying, se ritiene, con la propria strategia, che nelle casse a lui più vicine ci sia un tempo minore di attesa.

Lo scopo del progetto è indagare sulla configurazione di casse migliori, quindi ad ogni simulazione tutti i clienti avranno la stessa strategia di scelta della coda e di jockeying per creare una netta distinzione tra simulazioni diverse.

4.1 Fase di scelta della coda

Alla fine della *fase di shopping*, il cliente che non vuole andare alla cassa self-scan, deve scegliere quale coda seguire tra quelle disponibili, ovvero tra le code associate a casse aperte.

Una strategia è una scelta della coda q che minimizza una certa funzione $f(q)$, come riportato nel capitolo precedente alla 3.1.

Al fine di definire le strategie, è importante notare che la funzione $\text{basket-size}(c_i)$ non deve essere deterministica, come già introdotto nel capitolo precedente: il cliente fa una stima del numero di elementi nel carrello degli altri elementi, non ne è però certo, per questo questa funzione ha una probabilità di errore che rende la stima più veritiera, e si può ridefinire:

$$\text{estimate-basket-size}(c_i) = \text{basket-size}(c_i) + e_i \quad (4.1)$$

dove e_i è l'errore commesso nella stima e può essere sia positivo che negativo. L'errore dipende dalla grandezza del carrello, in particolare più elementi ci sono nel carrello più l'errore aumenta; intuitivamente in questo modello si è deciso di far variare l'errore in modo logaritmico rispetto alla grandezza del carrello, in base alla formula:

$$e_i = \log_b(\text{basket-size}(c_i)) \quad (4.2)$$

dove $b \in \mathbb{R}$, $b > 1$. La quantità $\text{estimate-basket-size}$ diventa dunque una variabile aleatoria normale con media la basket-size reale e deviazione standard l'errore commesso sulla stima.

Le 4 strategie di scelta della coda prese in considerazione per il modello sono:

1. **Minimo numero di elementi:** viene scelta la coda con il minimo numero di elementi nei carrelli di tutte le persone in attesa. La funzione da minimizzare è quindi per una coda $q \in Q$ dove Q è l'insieme di tutte le code disponibili:

$$f(q) = \sum_{i=1}^N \text{estimate-basket-size}(c_i) \quad (4.3)$$

dove $c_i \in q$ è un cliente in coda, $\text{estimate-basket-size}(c_i)$ è il numero di elementi che ha nel suo carrello e $N = |q|$ è il numero di clienti in coda in q .

2. **Minimo numero di persone:** viene scelta la coda con il minimo numero di persone accodate. La funzione da minimizzare per $q \in Q$ è:

$$f(q) = |q| \quad (4.4)$$

3. **Minimo tempo d'attesa in base al tempo di servizio medio:** viene scelta la coda con il tempo d'attesa minimo, calcolato in base al tempo di servizio medio. Il *tempo di servizio totale di una coda* è calcolato come somma del tempo di servizio per ogni cliente della coda. Il tempo di servizio per un cliente comprende il **tempo di transazione** e il **tempo di pausa** tra un cliente e un altro, e variano a seconda del tipo di cassa, come illustrato nella sezione 3.4. Le formule per il calcolo dei tempi per la cassa standard 3.2 e 3.3, comprendendo la stima della grandezza del carrello, diventano dunque:

$$\text{transaction-time}_i = e^{a \log(\text{estimate-basket-size}(c_i)) + b} \quad (4.5)$$

$$\text{break-time}_i = \frac{\beta^\alpha \text{estimate-basket-size}(c_i)^{\alpha-1} e^{-\beta \text{estimate-basket-size}(c_i)}}{\Gamma(\alpha)} \quad (4.6)$$

e per la cassa self-service, le 3.6 e 3.7 diventano:

$$\text{transaction-time}_i = e^{a \log(\text{estimate-basket-size}(c_i)) + b} \quad (4.7)$$

$$\text{break-time}_i = e^{c \log(\text{estimate-basket-size}(c_i)) + d} \quad (4.8)$$

da cambiare
con quelle
self-service

I tempi di servizio per ogni cliente di una coda sono quindi sommati per calcolare il tempo servizio totale per quella coda. Il tempo di servizio totale di una coda q_j , $j = 1, \dots, M$, dove M è il numero totale di code del supermercato, è pertanto:

$$\text{total-service-time}(q_j) = \sum_{i=1}^N (\text{transaction-time}_i + \text{break-time}_i) \quad (4.9)$$

Il tempo di servizio medio per le code è la somma dei tempi totali di servizio divisa per il numero di code. Viene scelta la coda con il tempo totale minimo, mettendo insieme le 4.5, 4.6, ??, 4.8 e 4.9 si ottiene la funzione da minimizzare:

$$f(q) = |q| * \frac{1}{M} \sum_{j=1}^M (\text{total-service-time}(q_j)) \quad (4.10)$$

4. **Minimo tempo d'attesa in base alla *power regression*:** viene scelta la coda con il tempo d'attesa minimo, calcolato in base al tempo di transazione e il tempo di pausa medi per quella coda. Il tempo di servizio totale per una coda è calcolato anche qui in base alla 4.9. La funzione da minimizzare è:

$$f(q) = |q| * \text{total-service-time}(q) \quad (4.11)$$

4.2 Fase di attesa in coda e jockeying

Quando il cliente è nella *fase di attesa in coda* ha la possibilità di cambiare la propria scelta se nota che nelle code adiacenti il tempo di attesa è minore che nella propria; le code adiacenti implicate nel calcolo sono quelle che hanno distanza minore o uguale del **parametro di adiacenza**, descritto nel capitolo precedente, rispetto a dove il cliente è già accodato. Una coda è distante 1 da un'altra coda se fisicamente nella griglia della simulazione esse compaiono una di fianco all'altra.

Le strategie seguite nel jockeying sono due, e coincidono con le prime due strategie di scelta della coda, descritte nella sezione precedente. Anche in questo caso il cliente stima le grandezze dei carrelli degli altri clienti in base alle 4.1 e 4.2.

1. **Minimo numero di elementi:** viene scelta la coda con il minimo numero di elementi nei carrelli di tutte le persone in attesa.
2. **Minimo numero di persone:** viene scelta la coda con il minimo numero di persone accodate.

Se il cliente, grazie alla sua strategia, calcola che nelle code adiacenti ci sia un tempo di attesa minore rispetto alla propria, secondo una distribuzione di probabilità, la quale aumenta più risulta conveniente cambiare coda, il cliente si muove o meno.

che distribuzione?

Chapter 5

Simulazione

Chapter 6

Analisi dei risultati

Mettere grafico fondamentale, ma prima definire densità e flusso per come li abbiamo calcolati nella classe Supermarket.

Chapter 7

Conclusioni

Ciao