05/07/2020

**PIAZZA\_SPRINT 3**

## Sprint Goal

Essere in grado di gestire più clienti contemporaneamente, seppur in maniera "non ottimizzata".

## Requisiti

-I requisiti sono gli stessi elencati nel file [TFBO20ISS.pdf](../../../../DOCUMENTS/TFBO20ISS.pdf) con **alcune assunzioni**.

-Assunzioni:

1) Si rilassa il vincolo de "un solo cliente in sala". Ora possono arrivare tutte le richieste di ingresso che si vuole. Come da requisiti, la stanza potrà accogliere fino a N=2 clienti contemporaneamente.

2) Al barman, di conseguenza, potrà essere richiesto di lavorare alla preparazione di più ordini, anche in parallelo. Potrà lavorare al massimo su N=2 preparazioni in parallelo.

3) Il tempo di preparazione di un ordine è sempre lo stesso, a prescindere da cosa è stato ordinato.

4) I task del waiter **non** sono interrompibili.

## Analisi dei Requisiti

COSA deve essere il sistema? Come mi aspetto che si comporti al termine di questo Sprint il sistema?

Dai requisiti è evidente come la decisione di accettare o meno una richiesta di ingresso dipenda dalla situazione attuale dei tavoli. In questa tabella che ho realizzato vengono mostrati i 6 possibili scenari significativi in cui possono trovarsi i due teatable nel momento in cui arriva una richiesta.

|  |  |  |  |
| --- | --- | --- | --- |
| **teatable2**  **teatable1** | tableclean | busy | dirty |
| tableclean | 1 – R | 2 | 3 |
| busy | 2 – R | S | 4 – R | S | 5 |
| dirty | 3 – S | C | 5 – S | C | 6 – C |

**Scenario 1-2-3**

Arriva una richiesta di ingresso 🡪 Il waiter la prende in consegna il prima possibile[1] e accoglie il cliente.

**Scenario 4**

Arriva una richiesta di ingresso al waiter 🡪 Il waiter esegue il task *inform.*

**Scenario 5-6**

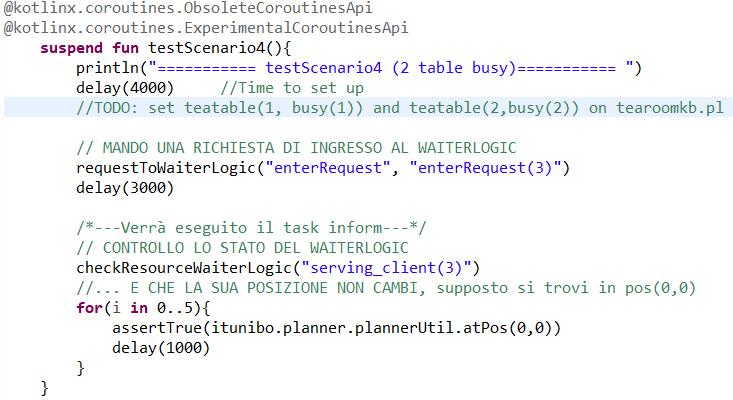
Arriva una richiesta di ingresso al waiter 🡪 Il waiter dovrà accettare lo stesso la richiesta di ingresso, ma dovrà pulire il tavolo prima di accoglierlo.

[1] dal momento che i task del waiter non sono interrompibili se il waiter è nello stato *serving\_client* oppure *cleaning* porta a termine ciò che sta facendo. Solo dopo valuterà le nuove richieste, in ordine di arrivo. Se si trova in stato *rest* invece processerà la prima richiesta estraendola dalla coda delle richieste.

### Bozze di Test Plan

Simulare gli scenari enunciati sopra e testare il comportamento del robot. Lo scenario 1 è già stato testato nello Sprint2.

Segue un esempio di come può essere impostato il test di uno scenario:



## Analisi del Problema

### Problematiche riprese dall’overview iniziale e da affrontare in questo sprint

“the waiter should reduce as much as possible the waiting time of the requests coming from each client”:Nello Sprint 2, una volta terminato il task *convoyToExit,* il waiter si metteva subito a pulire il tavolo appena usato e solo dopo tornava ad ascoltare altre richieste di ingresso. In un’ottica di ridurre il tempo di attesa delle richieste dei clienti, come richiesto dal requisito, ora si metterà sin da subito in ascolto di nuove richieste.

Da questo derivano due osservazioni:

1. Se siamo nello scenario 5 o 6 e arriva una richiesta di ingresso, il waiter dovrà eseguire una sorta di task *“accept but clean first”*: risponde **subito** e **in modo affermativo** al cliente che desidera entrare, in modo che riceva subito un feedback. Poi però, pulirà un tavolo[2] prima di raggiungere l’ingresso ed accogliere il cliente.

ATTENZIONE! In questo caso, mentre il waiter pulisce il tavolo, si trova nello stato logico *serving\_client* e non *cleaning* in quanto l’azione di pulizia è subordinata alla richiesta del cliente.

1. Il waiter non è più ‘obbligato’ a pulire i tavoli al termine del task *convoyToExit.* Se si limitasse a pulire i tavoli solo nel caso specificato al punto precedente, ritarderebbe sempre l’accoglienza dei clienti 🡪 è bene che se si trova nello stato logico *rest(X,Y)* e non ha ricevuto nessun tipo di richiesta controlli se ci sono dei tavoli dirty. Se così è il waiter andrà a pulirlo e solo dopo tornerà ad ascoltare le richieste.

Ovviamente, se ci sono dei tavoli da dover pulire, il waiter non potrà eseguire il task *rest*.

Attesa dovuta al task clean-Caso peggiore: il waiter decide di pulire il tavolo poiché fino a quel momento non aveva richieste da servire e appena entra nel task *clean* gli arriva una richiesta. Questa richiesta deve aspettare prima di essere presa in consegna. 🡪 Dopo averne discusso con il committente è emerso che il task clean può essere scomposto in 3 sottotask ciascuno della durata di .

Questi nuovi task li chiamiamo:

- clean1 (consisterebbe nello sparecchiare).

- clean2 (consisterebbe nel sanificare).

- clean3 (consisterebbe nell’apparecchiare).

Se dopo ogni sottotask facciamo controllare al waiter se ci sono nuove richieste e nel caso gliele facciamo prendere in consegna, il caso peggiore di attesa dovuta al task clean si riduce a . Ovviamente sarà compito del waiter ricordarsi lo stato di pulizia raggiunto per ogni tavolo, in modo da riprendere a pulirli da dove era rimasto.

Stato dei teatable*:* Ecco come risulta alla luce di ciò la transizione di stato dei tavoli.

**tableclean**

*(free and clean) clean*

**busy(clientId)**

**dirty(1)**

*Ingresso di un cliente che occupa il tavolo*

*Uscita del cliente*

*clean1*

**dirty(2)**

**dirty(3)**

*clean2*

*clean3*

~~Stima del maxWaitingTime: cosa fare se arriva un cliente e tutti i due tavoli sono dirty (free but not clean)? In questo caso si può pensare ad una stima diversa da quella pensata nell’overview e che sia esattamente uguale al tempo di pulizia di un tavolo. Se uno è busy e uno è dirty? In questo caso maxWaitingTime = min(clean\_time, maxStayTime+ LongestPreparationTime)~~

Countdown del maxStayTime*:* Avendo rilassato il vincolo di ‘un solo cliente in sala’ **nasce una nuova problematica che nello scorso Sprint non avevamo**. Cosa fare se un cliente dice “Sono pronto a ordinare”, oppure “sono pronto a pagare” ma il waiter sta servendo un altro cliente o sta pulendo un tavolo? **Potrebbe scadere il maxStayTime mentre il waiter sta finendo di servire l’altro cliente**. Il cliente non ha colpa…tuttavia non sarebbe giusto secondo me dare a lui la responsabilità di informare direttamente il maxstaytimeobserver. Potremmo conservare la modalità di ora. Semplicemente è il cliente a mandare il currentTime del momento in cui ha fatto la richiesta come payload del messaggio readytoX.

Al massimo devo introdurre un nuovo attore che faccia le veci del waiter per tenere sotto controllo solo la situazione maxstaytime.

Preparazione ordini concorrente: Il barman deve gestire più ordini contemporaneamente. Fino a 2.

Dotarlo di una propria knowledge base?!

Simulare più clienti contemporaneamente: fino a 2

Nell’ottica di avere un feedback più chiaro ed efficiente con il committente, il progettista valuti l’opportunità di fare un’interfaccia grafica con alcuni pulsanti che possano permettere di “suonare il campanello” tutte le volte che voglio, e ogni volta viene mandata una ring.

Volendo si possono mettere anche dei pulsanti che permettano al cliente di dire: sono pronto ad ordinare, sono pronto a pagare. In questo modo non devo simulare i tempi e impazzire a sincronizzare. Decido io con un pulsante quando il cliente prende iniziativa.

Attivare / disattivare i pulsanti per controllare l’input del cliente in base alla situazione attuale. Es: scade il maxStayTime. Nell’interfaccia di quel cliente si disattiveranno sia il pulsante readyToOrder sia readyToPay.

Poi si può fare un’interfaccia dove faccio vedere lo stato della stanza.

Si segnala al progettista di realizzare il tutto come interfaccia Web. Questa, infatti, essendo portabile su ogni tipo dispositivo dotato di browser (smartphone, pc, tablet ecc…) rende il prodotto più appetibile e facilmente vendibile sul mercato.

### Architettura Logica

### Modello eseguibile

Si veda il modello eseguibile ProblemAnalysisModel\_sprint3.qak che è una revisione del modello eseguibile prodotto nello Sprint2.

## Test Plan

## Progetto

**SPRINT 2 – REVIEW**