Waiters

2013년 9월 17일 화요일
오후 11:16

Messages

```
// 2:
msgSitAtTable(cust, table) {
        customer.add(cust, table);
}

// 4:
msgReadyToOrder(cust) {
        If exists in customers such that customer.c = cust
                Then customer.state = readyToOrder;
}


// 6:
msgHereIsMyChoice(cust, choice) {
        If exists in customer such that cutomer c= cust
                Then
                        MyCustomer.state = waitingFood1;
                        Customer.choice = choice;
                        state = waiting;
}

// OutOfFood 3: OutOfFood()
// TheMarketAndCook 1: OutOfFood()
msgOrderIsOutOfStock(Order order) {
        menu_list.remove(order.choice);
        Order.cust.state = reOrder;
}


// 8:
msgOrderIsReady(cust, choice) {
        If exists in customer such that cutomer c= cust
                Then customer.state = FoodIsReady;
}

// Cashier 0: ReadyForcheck
msgReadyForCheck(CustomerAgent customer) {
        customer.state = MyCustomer.CustState.askingForCheck;
}

// Cashier 1b: HereIsCheck
public void msgHereIsCheck(Check check) {
        check.customer.state = checkIsReady;
        cust.check = check;
        state = AgentState.Waiting;
}

// 10: IAmDone(customer)
// Cashier 3: LeaveTable
msgLeavingTable(CustomerAgent customer) {
        cust.state = MyCustomer.CustState.leaving;
}

// WaiterOnBreak 0: message from gui when break box is checked (on break)
msgOnBreak() {
        Break = AgentBreak.askForBreak;
}

// WaiterOnBreak 2: ReplyBreak
msgReplyBreak(boolean breakPermission) {
        waiterGui.getReplyBreak(breakPermission);
        if(breakPermission) {
                Break = AgentBreak.onBreak;
        }
        else {
```

Scheduler

```
if state == waiting
        For every customer in MyCustomers
                if (customer.state == Waiting) {
                        state = AgentState.Serving;
                        SitAtTable(customer);
                }
                else if (customer.state == readyToOrder) {
                        state = AgentState.Serving;
                        WhatWouldYouLike(customer);
                }
                else if (customer.state == waitingFood1) {
                        state = AgentState.Serving;
                        HereIsAnOrder(this, customer);
                }
                else if (customer.state == reOrder) {
                        state = AgentState.Serving;
                        WhatWouldYouLikeAgain(customer);
                }
                else if (customer.state == foodIsReady) {
                        state = AgentState.Serving;
                        HereIsYourOrder(customer);
                }
                else if (customer.state == askingForCheck) {
                        state = AgentState.Serving;
                        AskForCheck(customer);
                }
                else if (customer.state == checkIsReady) {
                        state = AgentState.Serving;
                        HereIsCheck(customer);
                }
        }
If Break == AgentBreak.askForBreak)
        AskForBreak();
else if Break == AgentBreak.offBreak)
        OffBreak();

For every customer in MyCustomers
        if (customer.state == leaving) {
                TableIsCleared(customer);
        }
}
```

Actions

```
SitAtTable(cust) {
        State = Serving;
        customer.msgFollowMe(menu);
        DoSeatCutomer(cust);
        customer.state = MyCustomer.CustState.seated;
        state = Waiting;
}

DoSeatCustomer(cust) {
        Customer.state = seated;
        Customer.t.setOccupant(cust);
}

WhatWouldYouLike() {
        Customer.msgWhatWouldYouLike();
        customer.state = MyCustomer.CustState.askedForOrder;
        state = Waiting;
}
```

```
        if(breakPermission) {
                Break = AgentBreak.onBreak;
        }
        else {
                Break = AgentBreak.none;
        }
}

// WaiterOnBreak 0: message from gui when break box is unchecked (off
break)
msgOffBreak() {
        Break = AgentBreak.offBreak;
}


Data

List<MyCustomer> customers;

Cook cook;
Host host;

List<String> menu_list
Map<String, Double> menu

enum AgentState
        {Waiting, Serving}

enum AgentBreak
        {none, askForBreak, waitingForAnswer, onBreak, offBreak};



Class MyCustomer {
        Customer c;
        Table t;
        String choice ;
        Check check;

        enum CustState
        {Waiting, seated, readyToOrder, askedForOrder, waitingFood1,
        waitingFood2, foodIsReady, eating, askingForCheck,
        waitingForCheck, checkIsReady, getTheCheck, leaving,
        reOrder, reOrdering};
}
```

```
        customer.msgWhatWouldYouLike();
        customer.state = MyCustomer.CustState.askedForOrder;
        state = Waiting;
}

HereIsAnOrder(this.waiter, customer) {
        Cook.msgHereIsAnOrder(new order(waiter, customer));
        customer.state = MyCustomer.CustState.waitingFood2;
        State = Waiting;
}

WhatWouldYouLikeAgain (MyCustomer customer) {
        customer.state = MyCustomer.CustState.reOrdering;
        state = Waiting;
        customer.c.msgAskForOrderAgain(menu_list);
}

HereIsYourOrder(customer) {
        Customer.msgHereIsYourOrder();
        Customer.state = eating;
        State = Waiting;
}

 AskForCheck(MyCustomer c) {
        c.state = waitingForCheck;


        cashier.msgComputeBill(c.choice, c.c, this, c.t.tableNumber, menu);
        State = Waiting;
}

HereIsCheck(MyCustomer c) {
        c.state = getTheCheck;
        c.c.msgHereIsYourCheck(c.check);
        state = Waiting;
}

TableIsCleared(customer) {
        Customers.remove(customer);
        Host.msgTableIsCleared(customer.t);
}

AskForBreak() {
        Break = waitingForAnswer;
        host.msgCanIBreak(this);
}


OffBreak() {
        Break = AgentBreak.none;
        host.msgOffBreak();
}
```

Customers

2013년 9월 17일 화요일
오후 11:16

Messages

```
// 0:
IAmHungry() {
        Event = gotHungry
}

// message when tables are full from host
msgWhetherLeave() {
        event = AgentEvent.tableFull;
}

// 3:
msgFollowMe(menu) {
        This.menu= menu;
        Event = followHost;
}

// 5:
// OutOfFood 0: WhatWouldYouLike
msgWhatWouldYouLike() {
        Event = makeOrder;
}

// OutOfFood 4:WhatWouldYouLikeAgain
msgAskForOrderAgain(List<String> menu_list) {
        this.menu_list.clear();
        this.menu_list.addAll(menu_list); // deep copy
        event = AgentEvent.reOrder;
}

 // 9:
msgHereIsYourOrder() {
        Event = getFood;
}

// Cashier 2: HereIsCheck
msgHereIsYourCheck(Check check) {
        event = AgentEvent.getCheck;
        this.check = check;
}

// Cashier 5: Change
public void msgChange(Cash cash) {
        this.cash.addChanges(cash);
        event = AgentEvent.leaveRestaurant;
}
```

Data

```
String name;
HostAgent host;
Waiter wait;
private List<String> menu_list
Set<String> menu;
String choice;
Check check;
Cash cash

enum AgentState
```

Scheduler

```
if (state == AgentState.DoingNothing && event == AgentEvent.gotHungry ){
        state = AgentState.WaitingInRestaurant;
        goToRestaurant();
}
else if (state == AgentState.WaitingInRestaurant && event == AgentEvent.tableFull ){
        state = AgentState.TableFull;
        thinkWhetherLeave();
}
else if (state == AgentState.WaitingInRestaurant && event == AgentEvent.followHost ){
        state = AgentState.BeingSeated;
        SitDown();
}
else if (state == AgentState.BeingSeated && event == AgentEvent.seated){
        state = AgentState.Seated;
        ChooseMenu();
}
else if (state == AgentState.Seated && event == AgentEvent.makeOrder){
        state = AgentState.WaitingFood;
        HereIsMyChoice(choice);
}
else if (state == AgentState.WaitingFood && event == AgentEvent.reOrder){
        // Go back to previous step to go over the process again
        state = AgentState.Seated;
        ChooseMenu();
}
else if (state == AgentState.WaitingFood && event == AgentEvent.getFood){
        state = AgentState.Eating;
        EatFood();
}
else if (state == AgentState.Eating && event == AgentEvent.doneEating){
        state = AgentState.DoneEating;
        ReadyForCheck();
}
else if (state == AgentState.DoneEating && event == AgentEvent.getCheck){
        state = AgentState.LeavingTable;
        leaveTable();
}
else if (state == AgentState.LeavingTable && event == AgentEvent.payment) {
        state = AgentState.DonePayment;
        Payment();
}
else if (state == AgentState.DonePayment && event == AgentEvent.leaveRestaurant) {
        state = AgentState.LeavingRestaurant;
        exitRestaurant();
}
```

Actions

```
goToRestaurant() {
        Host.msgIWantFood(this.cust);
}

thinkWhetherLeave() {
        if(random) {
                exitRestaurant();
        }
        else {
                state = AgentState.WaitingInRestaurant;
                event = AgentEvent.decidedToWait;
```

```
Cash cash

enum AgentState
        {DoingNothing, WaitingInRestaurant, TableFull, BeingSeated,
        Seated, WaitingFood, Eating, DoneEating, LeavingTable,
        DonePayment, LeavingRestaurant};

public enum AgentEvent
        {none, gotHungry, tableFull, decidedToWait, followHost, seated,
        callWaiterToOrder, makeOrder, getFood, doneEating,
        askForCheck, getCheck, payment, leaveRestaurant, doneLeaving,
        reOrder};

Class Cash {
                int twentyDollar;
                int tenDollar;
                int fiveDollar;
                int oneDollar;
                int coins;
}
```

```
        }
        else {
                state = AgentState.WaitingInRestaurant;
                event = AgentEvent.decidedToWait;
        }
        host.msgDecision(this);
}

SitDown() {
        cusotmerGui.dotogoseat();
}

ChoseMenu() {
        Choice = choices.get(random);
        event = AgentEvent.callWaiterToOrder;
}

ReadyToOrder(cust) {
        Wait.msgReadyToOrder(this.cust);
}

HereIsMyChoice(cust, choice) {
        Wait.msgHereIsMyChoice(this.cust, choice);
}

EatFood() {
        State = Eating;
        Timer.start( doneEating() );
}

ReadyForCheck() {
        wait.msgReadyForCheck(this);
}

leaveTable() {
        wait.msgLeavingTable(this);
}

Payment() {
        Cash payment;

        if(cash.totalAmount() >= check.price) {
                payment = cash.payCash(check.price);
                cashier.msgPayment(check, payment);
        }
        else {
                state = AgentState.DoingNothing;
                exitRestaurant();
        }
}
```

Host

2013년 9월 17일 화요일
오후 11:16

Messages

```
// 1:
msgIWantFood(cust){
        WaitingCustomer.add(cust);
}

// 11:
msgTableIsCleared (table) {
        Table.unoccupied();
}

// WaiterGoOnBreak 0: CanIHaveBreak
msgCanIBreak(WaiterAgent w) {
        stateChanged();
}

// WaiterGoOnBreak 2: OffBreak
msgOffBreak() {
        stateChanged();
}
```

Data

```
List<Customer> WaitingCustomers;
List<Table> Tables;
List<Waiter> Waiters;

Class Table() {
        Int tableNumber;
        Customer occupiedby;
}
```

Scheduler

```
If exits a table in tables such that table is empty
        If exits waiter in waiters such that waiter.break; = waitingForAndwer
                chkIfWaiterCanBreak(waiter);

        If exits customer in waitingCustomers
                Then tellWaiter(customer, table);
                If tables are full
                        askCustomerWhenFull();
```

Actions

```
askCustomerWhenFull() {
        For all customer in customers
                c.msgWhetherLeave();
}

tellWaiter(customer, table) {
        wait.msgSitAtTable(cust, table);
        WaitingCustomer.remove(0);
}

chkIfWaiterCanBreak(WaiterAgent w) {
        if( (waitingCustomers.isEmpty()) && (numberOfNoBreakWaiters > 1)) {
                breakPermission = true;
        }
        else {
                breakPermission = false;
        }
        w.msgReplyBreak(breakPermission);
}
```

Cook

2013년 9월 17일 화요일
오후 11:16

Messages

```
// 7
msgHereIsAnOrder(order) {
        Orders.add(order);
}

// TheMarketAndCook 3: OrderFulfillment
msgOrderFulfillment(Procure procure) {
        foods.get(procure.getFood()).amount +=
        foods.get(procure.getFood()).batchSize;
}
```

Scheduler

```
If there exits in Order
        If state == pending
                Order.state = cooking;
                CookOrder(order);
        Else if state == cooked
                OrderIsReady(order);
        Else if state == outOfStock
                OrderIsOutOfStock(order)
```

Data

```
List<Order> orders;
List<MarketAgent> markets
Map<String, Food> foods
List<String> menu_list


Class Order {
        Waiter w;
        Customer cust;
        Food choice;

        Enum OrderStatus:Pending, Cooking, Cooked;
}

Class Food {
        String name;
        Int time;
}
```

Actions

```
CookOrder(order) {
        DoCooking(order);
        foods.get(order.choice).amount --;
        if (foods.get(order.choice).amount == 1 || foods.get(order.choice).amount == 0) {
                BuyFood(order.choice, foods.get(order.choice).batchSize);
        }
        else {
                order.state = Order.OrderState.outOfStock;
                BuyFood(order.choice, foods.get(order.choice).batchSize);
        }
}

DoCooking(order) {
        state = cooking;
        Time.start { Done(Order) };
        state = cooked;
}

OrderIsReady(Order order) {
        order.waiter.msgOrderIsReady(order);
        orders.remove(order);
}

OrderIsOutOfStock(Order order) {
        order.waiter.msgOrderIsOutOfStock(order);
        orders.remove(order);
}

BuyFood(String food, int batchSize) {
        for(MarketAgent m : markets) {
                if(m.msgBuyFood(new Procure(food, batchSize))) {
                        marketAvailable = true;
                        Do("Ordered " + food + " to " + m.getName());
                        break;
                        }
                }
                if(!marketAvailable) {
                        Do("There is no market that has a stock for " + food);
                }
        }
}
```

Market

2013년 10월 13일 일요일
오전 11:30

Messages

```
// TheMarketAndCook 2: BuyFood()
msgBuyFood(Procure procure) {
	// check availability for the procure order
	if(inventory.get(procure.food).amount < procure.batchSize) {

	}
	else {
		procures.add(procure);
		inventory.get(procure.food).amount -= procure.batchSize;
	}
}
```

Scheduler

```
If exits a procure in procures
	If procure.state == pending {
		procure.state = delivering
		DeliverOrder(procure)
	}
	else if procure.state == done {
		OrderFulfillment(procure)
	}
}
```

Data

```
List<Procure> procures
Map<String, Food> inventory
List<String> food_list

class Procure {
		private String food;
		int batchSize;

		public enum ProcureState
		{Pending, Delivering, Done, outOfStock};
}
```

Actions

```
DeliverOrder(Procure procure) {
	DoDeliver(procure);
}

DoDeliver(Procure procure) {
	Timer timer = new Timer();
	Time.start { Done(Procure) };
	Procure.state = done; // will be implemented with gui and timer
}

OrderFulfillment(Procure procure) {
	cook.msgOrderFulfillment(procure);
	procures.remove(procure);
}
```

Cashier

2013년 10월 13일 일요일
오전 11:30

Messages

```
// Cashier 1a: ComputeBill
	public void msgComputeBill(choice, cust, wait, table #) {
		checks.add(new Check(choice, cust, wait, table#));
	}

// Cashier 4: Payment
	public void msgPayment(check, cash) {
		for(Check c : checks) {
			if(c == check) {
				c.cash = cash
				c.state = Check.CheckState.receivedCash;
			}
		}
	}
```

Data

```
List<Check> checks
private Map<String, Double> menu

class Check {
		String choice;
		CustomerAgent customer;
		WaiterAgent waiter;
		int tableNumber;
		double price;

		Cash cash; // from customer

		public enum CheckState
		{nothing, waitingToBePaid, receivedCash, paid};
}
```

Scheduler

```
If exits a check in checks
	If check.state == nothing {
		Check.state = watingToBePaid;
		ComputeBill(check);
	}
	else if check.state == receivedCash {
		Check.state = paid;
		returnChange(check);
	}
}
```

Actions

```
ComputeBill(Check c) {
	c.setPrice(foods.get(c.choice).price);
	c.waiter.msgHereIsCheck(c);
}

returnChange(Check c) {
	int twentyDollar = 0;
	int tenDollar = 0;
	int fiveDollar = 0;
	int oneDollar = 0;
	int coins = 0;

	double change = c.cash.totalAmount() - c.price;
	Cash Change;

	twentyDollar = (int) change/20;
	change -= 20*twentyDollar;

	tenDollar = (int) change/10;
	change -= 10*tenDollar;

	fiveDollar = (int) change/5;
	change -= 5*fiveDollar;

	oneDollar = (int) change/1;
	change -= 1*oneDollar;

	coins = (int) (100*((double)(change)+0.0001));

	Change = new Cash(twentyDollar, tenDollar,
	fiveDollar, oneDollar, coins);
```

```
            checks.remove(c);
            c.customer.msgChange(Change);
        }
```