**Market Agent**

Data

List<Order> orders;

CookAgent cook;

String name;

Timer timer;

Food steak, chicken, pizza, salad;

enum OrderState {Pending, Preparing, Done};

Map<String, Food> foods;

class Order {

CookAgent cook;

String food;

int amount;

OrderState state;

int amountPrepared;

}

class Food {

String type;

int prepTimer;

int inventory;

int low;

int capacity;

}

Messages

OrderFood(CookAgent cook, Map<String,Integer> lowFood) {

for (food : lowFood.keySet)

orders.add(new Order(cook, food, lowFood.get(food)));

this.cook = cook;

}

OrderReady(Order order) {

order.state = Done;

}

Scheduler

if Ǝ order in orders ϶ order.state = Done

giveOrderToCook(order)

if Ǝ order in orders ϶ order.state = Pending

tryToPrepareOrder(order)

Actions

TryToPrepareOrder(Order order) {

int totalPrepTime = 0;

int foodPrepTime = foods.get(order.food).prepTimer;

int inventory = foods.get(order.food).inventory;

int numWanted = order.amount;

int numToPrepare = 0;

boolean tooLow = false;

if (inventory >= order.amount { numToPrepare = order.amount; }

else { numToPrepare = inventory; tooLow = true; }

if (tooLow)

cook.OrderNotFulfilled(this, order.food, numWanted – numToPrepare);

foods.get(order.food).inventory -= numToPrepare;

totalPrepTime = numToPrepare \* foodPrepTime;

order.state = Preparing;

timer.schedule( run { OrderReady(order) }, totalPrepTime);

}

GiveOrderToCook(Order order) {

orders.remove(order);

cook.OrderFinished(this, order.food, order.amountPrepared);

}