***Detailed Conceptual Model***

***Structural Components***

|  |  |  |
| --- | --- | --- |
| Constants | | |
| Name | Description | Value |
| PARTYWAITINGLIST\_CAP | Room required for accommodating parties. | 2 |
| TOTAL\_TABLE\_NUM | Total number of tables counted by small tables (tables accommodated for one or two people). | 11 |
| Parameters | | |
| Name | Description | Value |
| RG.Tables.num4T | Number of large tables (accommodating large party of 3 or 4 people) at the restaurant. | 2, 3, 4 or 5 |
| RG.Waiters.numW | Number of waiters at the restaurant. | 2 or more |
| RG.Cooks.numC | Number of cooks at the restaurant. | 2 or more |

|  |  |
| --- | --- |
| Consumer Class : Party | |
| This consumer entity structure represents the party arrive at the restaurant, being served and paying and then leaving the restaurant. | |
| Attributes | Description |
| uSize | This input variable represents the size of the party (value is one of 1, 2, 3, or 4) as assigned via RVP.uPartySize(). |
| partyScale | The party scale represents the scale of the party. When the size of the party is more than 2 people, the value is LARGE, otherwise, the value is SMALL. |
| startWait | A timestamp used to determine waiting time. |
| waitTime | The time the party waiting for the service. |

|  |  |
| --- | --- |
| Resource Group Unary: Waiters | |
| This resource entity structure represents the waiters that are needed to seat the party, place and deliver the order for the party, deliver food, collect the fee and clean the table. | |
| Attribute | Description |
| numW | The number of available waiters (this is a parameter). |
| busyWaiters | The number of waiters in the busy status. |

|  |  |
| --- | --- |
| Resource Group Unary: Cooks | |
| This resource entity structure represents the cooks that are needed to prepare the food. | |
| Attribute | Description |
| numC | The number of available cooks (this is a parameter). |
| busyCooks | The number of waiters in the busy status. |

|  |  |
| --- | --- |
| Resource Group SET[2]: Tables | |
| This resource entity structure represents the tables that are needed to serve the party. LARGE used to identify the group of large tables and SMALL the group of small tables. | |
| Attribute | Description |
| numTables | The maximum number of table that is available. |
| num4T | Number of large tables (accommodating large party of 3 or 4 people) at the restaurant. |
| list | A list of the table entity instances those are available to accommodate the parties. |
| n | The number of tables used for accommodating the parties in one set. |

|  |  |
| --- | --- |
| Queue SET[2]: PartyWaitingList | |
| This queue entity structure represents the party waiting list in the counter area waiting to be seated by a waiter. LARGE and SMALL are identifiers for parties of ¾ and 2/1 respectively. | |
| Attribute | Description |
| list | A list of the Party entity instances in the restaurant’s list of parties that have arrived and are waiting for a waiter to seat them. |
| n | The number of entries in the list.(maximum values is numParty) |

|  |  |
| --- | --- |
| Queue Unary: PartyPayList | |
| This queue entity structure represents the party waiting-to-pay list after eating the food at the restaurant. | |
| Attribute | Description |
| list | A list of the Party entity instances in the restaurant’s list of parties that has already finished their food and waited to pay their fee. |
| n | The number of entries in the list. |

|  |  |
| --- | --- |
| Queue Unary: PartyOrderList | |
| This queue entity structure represents the party order list in the kitchen area waiting to be prepared by cooks. | |
| Attribute | Description |
| list | A list of the orders that placed by Party entity waiting for a cook to prepare them. |
| n | The number of entries in List. |

|  |  |
| --- | --- |
| Queue Unary: PartyDelFoodList | |
| This queue entity structure represents the party food list in the kitchen area waiting for the waiter to deliver them. | |
| Attribute | Description |
| list | A list of food that has been prepared by the cooks and waits to be delivered. |
| n | The number of entries in list. |

***Behavioural components***

Time units: minutes

Observation interval: Bounded Horizon Study ----- Starting from 0 minute to 360 minutes.

|  |  |
| --- | --- |
| Action : Initialise | |
|  | |
| TimeSequence | <0> |
| Event SCS | RG.Cooks.busyCooks🡨0  RG.Waiters.busyWaiters🡨0  RG.Tables[LARGE].n🡨0  RG.Tables[SMALL].n🡨0  RG.Tables[LARGE].numTables 🡨 RG.Tables.num4T  RG.Tables[SMALL].numTables 🡨 11 - RG.Tables.num4T\*2  NumOfParty 🡨 RVP.NumOfParty // indicate the number of parties arriving at restaurant per day  Q.PartyWaitingList[LARGE].n🡨 0  Q.PartyWaitingList[SMALL].n🡨 0  Q.PartyPayList.n🡨0  Q.PartyOrderList.n🡨0  Q.PartyDelFoodList.n🡨0  SSOV.numBalkParty🡨0  SSOV.numLongWait🡨0  SSOV.profit🡨-(60\*RG.Waiters.numWaiters + 100\*RG.Cooks.numCooks + 300) |

***Output***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OUTPUTS | | | | |
| Trajectory Sequences | | | | |
| Name | | Description | | |
| TRJ[RG.Cooks.busyCooks] | | This trajectory sequence records the values of the attribute RG.Cooks.busyCooks which represents the number of cooks being busy. | | |
| TRJ[RG.Waiters.busyWaiters] | | This trajectory sequence records the values of the attribute RG.Waiters.busyWaiters which represents the number of waiters being busy. | | |
| Sample Sequences | | | | |
| Name | | Description | | |
| PHI[iCG.Party.waitTime] | | Each value in PHI[iCG.Party.waitTime] is of the form (tk, yk) where yk is the value for some party instance and tk is the time when the waiter begins to serve the party. Its value is equal to t – iC.Party.startWait. ( delete this, no use in the project) | | |
| Derived Scalar Output Variables (DSOV’s) | | | | |
| Name | Description | | Output Sequence Name | Operation |
| waiterUtilization | The utilization of the waiters. | | TRJ[RG.Waiters.busyWaiters] | AVG/ RG.Waiters.numW |
| cookUtilization | The utilization of the cooks. | | TRJ[RG.Cooks.busyCooks] | AVG/ RG.Cooks.numC |
| Simple Scalar Output Variable (SSOV’s) | | | | |
| profit | | The total profit per day of the restaurant. First, it is initialized to the daily cost of the restaurant (60\*RG.Waiters.numWaiters + 100\*RG.Cooks.numCooks + 300) and then add profit from each party when a party pays (RVP.uPartyProfit). | | |
| numBalkParty | | The number of party who leave without eating. | | |
| numLongWait | | The number of party who wait for more than 30 minutes. | | |
| perPartyWait | | numLongWait/ NumOfParty | | |
| perBalkParty | | numBalkParty/NumOfParty | | |

***User Defined Procedures***

|  |  |
| --- | --- |
| User Defined Procedures | |
| UDP.CanStartPreService() | This procedure provides the information whether we can start the pre-service for the party. In the restaurant, if  1) the large table is available, and there is a large party waiting in the queue for the service, meanwhile, there is an available waiter, then it return true.  2) the small table is available, and there is a small party waiting in the queue for the service, meanwhile, there is an available waiter, then it return true.  3) otherwise, it returns false. |
| UDP.CanStartPaying() | This procedure provides the information whether we can start collecting fee and clean the table. In the restaurant, if  1) the UDP.CanStartPreService() = false and  2) the Q.PartyPayList.n > 0 and  3) the waiter is available,  then this function return true, otherwise return false. |
| UDP.UpdateSSOVs(iCG.Party) | This procedure is to update the SSOVs: numLongWait, perPartyWait.  If the waiting time of the party is more than 30, then the SSOV.numLongWait increases one.  Then we can obtain the percentage of the party who waiting more than 30 minutes through the formulas: SSOV.numLongWait/SSOV.numParty. (Delete) |

***Input Constructs***

|  |  |  |
| --- | --- | --- |
| Random Variate Procedures | | |
| Name | Description | Data Model |
| RVP.uNumParty | Provide the values of the number of party instance who arrive at restaurant each day. | The data model is UNIFORM(Min, Max), where Min = 30 and Max = 50. |
| RVP.DuParty() | Provides the values of the arrival times of party instance. | Exponential(X) where X is:   |  |  | | --- | --- | | time | mean | | 0 to 60 | 60min/(0.1Num) | | 60 to 120 | 60min/(0.2Num) | | 120 to 240 | 120min/(0.55Num) | | 240 to 300 | 60min/(0.1Num) | | 300 to 360 | 60min/(0.05Num) |   where the Num is the number of parties for each day( RVP.uNumParty). |
| RVP.uPartySize() | Returns the size of a party entity instance; value is one of 1, 2, 3 or 4. | The data model is UNIFORM(Min, Max), where Min = 1 and Max = 4. |
| RVP.uPreServiceTime() | The returned value is the duration of waiter entity instance seating a party entity instance, writing the order and delivering the order for the party entity. | The data model is NORMAL(Mean, Variance), where Mean = 7 min, Variance = 1.7 min. |
| RVP.uPreFoodTime() | The procedure provides values for the duration of cook entity instance to prepare the food for the party entity instance. | The data model is NORMAL(Mean, Variance), where Mean = 7 min, Variance = 1.5 min. |
| RVP.uDelFoodTime() | The procedure provides values for the duration of waiter entity instance to deliver the food. | The data model is NORMAL(Mean, Variance), where Mean = 2 min, Variance = 0.5 min. |
| RVP.uEatingTime() | The procedure provides values for the duration of party entity instance to eat the food. | The data model is NORMAL(Mean, Variance), where Mean = 10 min, Variance = 2 min. |
| RVP.uPostServiceTime() | The procedure provides values for the duration of waiter entity instance to clean the table and collect the fee. | The data model is NORMAL(Mean, Variance), where Mean = 3 min, Variance = 0.8 min. |
| RVP.uPartyProfit( uPartySize) | The procedure provides value of the bill that each party need to pay. | For each individual, the bill is uniform distributed UNIFORM(Min, Max) where Min = 10, Max = 16. And the total value of the party profit is to sum the individual bill within each party. |

|  |  |
| --- | --- |
| Action : PartyArrivals | |
| Provides the input entity stream of arriving party entity instances; includes the associated derive operation and attribute value assignment. | |
| TimeSequence | RVP.DuParty() |
| Event SCS | iCG.Party🡨SP.Derive(Party)  iCG.Party.uSize🡨RVP.uPartySize()  IF(iCG.Party.uSize > 2)  partyScale 🡨 LARGE  ELSE  partyScale 🡨SMALL  END IF  iCG.Party.startWait🡨t  iCG.Party.waitTime🡨0  IF (Q.PartyWaitingList[iCG.partyScale].n> 2)  SSOV.numBalkParty + 🡨 1;  SSOV.perBalkParty 🡨 SSOV.numBalkParty/RVP.uNumParty ( Delete)  ENDIF  SP.InsertQue(Q.PartyWaitingList[iCG.Party.PartyScale], iCG.Party) |

***Behavioural Constructs***

|  |  |
| --- | --- |
| Activity : PreService | |
| This activity represents the Waiters entity leading the Party which is first on the PartyWaitingList from the waiting area to the table, writing down the Party’s order and deliver the order to the kitchen. | |
| Precondition | UDP.CanStartPreService() = true |
| Event SCS | iCG.Party🡨SP.RemoveQue(Q.PartyWaitingList)  iCG.Party.waitTime 🡨 t – iCG.Party.startWait  IF(iCG.Party.waitTime ≥ 30)  SSOV.numLongWait+🡨1  ENDIF (adding this)  SP.Put(PHI[iCG.Party.waitTime], iCG.Party.waitTime) ( delete this, not used in the project)  RG.Waiters.busyWaiter +🡨 1  SP.InsertGrp(RG.Tables[partyScale], iCG.Party)  RG.Tables[partyScale].n +🡨 1 |
| Duration | RVP.uPreServiceTime() |
| Event SCS | SP.InsertQue(Q.PartyOrderList, iCG.Party)  RG.Waiters.busyWaiter - 🡨1 |

|  |  |
| --- | --- |
| Activity : PrepareFood | |
| This activity represents the Cooks entity preparing the food according to the order list and bringing out the food for the Waiter entity to deliver it to the Party entity. | |
| Precondition | RG.Cooks.busyCooks < RG,Cooks.numCooks AND Q.PartyOrderList.n≠0 |
| Event SCS | iCG.Party 🡨 SP.RemoveQue(PartyOrderList)  RG.Cooks.busyCooks + 🡨1 |
| Duration | RVP.uPreFoodTime() |
| Event SCS | RG.Cooks.busyCooks - 🡨1  SP.InsertQue(PartyDelFoodList, iCG.Party) |

|  |  |
| --- | --- |
| Activity : DeliverFood | |
| This activity represents the Waiters entity delivering food to the Party entity. | |
| Precondition | RG.Waiters.busyWaiters < RG.Waiters.numWaiters AND Q. PartyDelFoodList.n ≠0 AND UDP.CanStartPaying() = false |
| Event SCS | iCG.Party🡨 SP. RemoveQue(Q.PartyDelFoodList)  RG.Waiters.busyWaiters + 🡨 1 |
| Duration | RVP.uDelFoodTime() |
| Event SCS | SP.Start(Eating, iCG.Party) |

|  |  |
| --- | --- |
| Activity : Eating | |
| This activity represents the Party enjoying their food in the restaurant. This is a sequel activity and is initiated at the completion of the DelFood activity. | |
| Causal | (iCG.Party) |
| Event SCS |  |
| Duration | RVP.uEatingTime() |
| Event SCS | SP.InsertQue(Q. PartyPayList, iCG.Party) |

|  |  |
| --- | --- |
| Activity : PostService | |
| This activity represents the Party paying their fees to the Waiters entity after finishing eating and Waiters entity cleaning the table. It is a conditional activity which requires availability of the Waiters entity. | |
| Precondition | UDP.CanStartPostService() = true |
| Event SCS | iCG.Party🡨SP.RemoveQue(Q. PartyPayList)  RG.Waiters.busyWaiters + 🡨 1  SSOV.profit+🡨 RVP.uPartyProfit( uPartySize) |
| Duration | RVP.uPostServiceTime() |
| Event SCS | SP.RemoveGrp(RG.Tables[partyScale], iCG.Party)  RG.Tables[partyScale].n -🡨 1  SP.Leave(iCG.Party)  RG.Waiters.busyWaiters - 🡨1 |

***Design of Validation Experimentation***

Given the simplicity of the model, it is possible to validate the model using a trace log as described below. The model shall be validated for base case and alternative case.

***Trace Logging***

The state of the simulation model is monitored by tracking the utilization of waiters and cooks( DSOV.waiterUtiliztion, DSOV.cookUtilization), the profit(SSOV.profit), and some statistical value of party( SSOV.numBalkParty, SSOV.numLongWait, SSOV.perPartyWait, SSOV.perBalkParty). These values are presented on a single line as follows:

Clock: xxxxx; DSOV.waiterUtilization: xxxxx; DSOV.cookUtilization: xxxxx; SSOV.profit: xxxx; SSOV.numBalkParty: xxxx; SSOV.numLongWait: xxxx; SSOV.perPartyWait: xxxx; SSOV.perBalkParty: xxxx;

where the xxx’s are replace with current values of the clock and attributes accordingly.