Objektorienterad programmering i C++
Örjan Sterner

”Simulerade hamburgare”

Ett exempel på

Diskret händelsestyrd simulering

Simulerade hamburgare

Hamburgerkedjan *McSmack* planerar en etablering i utkanten av Östersund. En viktig del av förberedelserna är att dimensionera verksamheten för att optimera vinsten utifrån vissa antaganden om kundernas beteende. Ett sätt att göra detta är att försöka simulera en dag i stekoset vilket detta exempel demonstrerar. Metoden som används kallas Diskret händelsestyrd simulering (”Discrete even-driven simulation”) och implementeras i ett framework som kan anpassas till, och integreras i en annan applikation.

Diskret händelsestyrd simulering
I en diskret händelsestyrd simulering finns det en ’klocka’ i form av en räknare som håller reda på aktuell *simuleringstid* och en *händelsekö* vilket är en kö av händelser (”event”) ordnad efter tidpunkterna då händelserna i kön skall inträffa. Varje händelse har en tidsstämpel och är inplanerad att inträffa vid en viss tidpunkt i simuleringens framtid. Nästa händelse som inträffar är alltså den som har det lägsta tidsvärdet och därmed ligger närmast i framtiden. Man använder lämpligen en *prioritetskö* där elementet med det lägsta tidsvärdet är högst prioriterat. Simulerings-tiden startar på 0 och ökar allt eftersom simuleringen fortskrider. Före starten av simuleringen schemalägger (schedulerar) man de initiala händelser som ska utgöra utgångsläget för simuleringen genom att placera dem i händelsekön.

När en händelse inträffar så *processas* den genom att den kod som är förknippad med händelsen exekveras. Detta kan innebära att andra objekt i simuleringen anropas och även att *nya händelser skapas och scheduleras för processande vid senare tidpunkter*.

Simuleringen drivs framåt genom en loop där nästa händelse i tidsordning plockas från kön och processas så länge det finns händelser kvar i kön.

Att simuleringsmetoden kallas diskret beror på att simuleringstiden alltid antar diskreta värden, nämligen de värden som motsvarar tidsstämplarna på de schedulerade händelserna. Simuleringstiden flyter alltså inte kontinuerligt fram som vår vanliga ’väggklocketid’ utan hoppar framåt i större eller mindre intervall. På så sätt kan alltså händelser över en lång tid simuleras på mycket kort tid i datorn.

Exempelmodellen
Den modell som simuleras i exemplet avspeglar verksamheten hos en hambugerbar, McSmack, och kan beskrivas enligt följande.

- Hungriga kunder anländer i grupper om 1 till 10 personer. Tiden mellan gruppernas ankomster är 1 till 4 minuter. Om det vid ankomsttillfället inte finns lediga stolar åt alla i gruppen vänder de direkt i dörren och går till någon konkurrent till McSmack (McSmack är en hamburgerbar med stil: inga kunder får äta stående). Annars går de och ställer sig i kön för att beställa.

- Det tar mellan 1 och 5 minuter att få göra sin beställning. Alla beställer det enda som finns på menyn, nämligen en smaskig BigSmack.

- När beställningen är gjord står gruppen kvar och väntar snällt på sina beställningar, det tar mellan 1 och 5 minuter innan alla i gruppen får sina burgare.

- När de fått sina fettdrypande fläskbiffar letar de lediga stolar vilket tar 1 min. Om det inte finns lediga stolar till alla så väntar hela gänget i 1 minut och försöker igen.

När alla i gruppen sitter på en stol med sina burgare påbörjas smaskandet som tar mellan 5 och 15 minuter. Därefter lämnar man McSmack ljudligt rapande och drömmer om mammas mat...

Efter 8 timmar får inga nya kunder komma in men de som är inne hos McSmack får naturligtvis fullfölja sina beställningar och äta klart.

Vinsten per serverad BigSmack uppskattas till 6:- och kostnaden för varje stol beräknas till 150:- per dag, löner och hyra mm inräknat. Filen SimConst.h innehåller alla konstanter för simuleringen.
Frågan som simuleringen besvarar är hur restaurangen ska dimensioneras:

Vilket antal stolar ger den största vinsten per dag under de givna förutsättningarna?

Genom att köra upprepade simuleringar där antalet stolar varieras kan vi få en ganska bra uppfattning om detta. I implementationen av modellen finns sex typer av händelser.

- ArriveEvent - en ny grupp anländer och försöker få plats. Om det vid det tillfället finns lediga stolar till alla så går de in och köar för att beställa och ett OrderEvent scheduleras. I annat fall vänder de och går.

- OrderEvent - en beställning görs, ett ServiceEvent scheduleras.

- ServiceEvent - burgarna serveras och ett SeatedEvent scheduleras

- SeatedEvent - gruppen försöker hitta stolar till alla. Om det misslyckas scheduleras ett nytt SeatedEvent, annars scheduleras ett LeaveEvent för gruppen.

- LeaveEvent - gruppen har ätit klart och lämnar McSmack

- ClosingEvent - McSmack stänger sin dörr för nya gäster

Implementation
Förutsättningarna för simuleringen bestäms av konstanter som finns i filen SimConst.h.

```
const int COST_PER_CHAIR = 150;  // Daily cost for one chair
const int MIN_ARRIVAL_DELAY = 1; // Min time between two arrivals
const int MAX_ARRIVAL_DELAY = 4; // Max time between two arrivals
const int MIN_GRP_SIZE = 1;     //Min nr of persons for each arrival
const int MAX_GRP_SIZE = 10;    //Max nr of persons for each arrival
const int MIN_ORDER_WAIT = 1;   //Min wait time for ordering
const int MAX_ORDER_WAIT = 5 ;  //Max wait time for ordering
const int MIN_SERVICE_WAIT = 1; //Min time to service after ordering
const int MAX_SERVICE_WAIT = 5; //Max time to service after ordering
const int SEAT_SEARCH_TIME= 1;  //Time to search for chairs
const int MIN_EATING_TIME = 5;  //Min time to finish meal after seated
const int MAX_EATING_TIME = 15; //Max time to finish meal after seated
const int BURGERS_PER_PERSON = 1; //Nr of burgers ordered by a person
const int PROFIT_PER_BURGER = 6.0; // Profit for each sold burger
```

```
const int SIM_TIME = 480; // Duration of simulation
```

Funktionen `rand_between(low,high)` ger dig ett slumptal inom intervallet [low..high] (RandomInt.h).

Varje händelsetyp implementeras som en deriverad klass till den abstrakta klassen Event:

```
class Event {
public:

    Constructor...
    Destructor...

    virtual void processEvent()=0; // Process event

    unsigned int getTime() const { // Time for this event
        return time;
    }

protected:
    // Time for this event
    unsigned int time;
};  // Event

// Compare two Events with respect to time
class EventComparison {
public:
    bool operator() (Event * left, Event * right) {
        return left->getTime() > right->getTime();
    }
};
```

Kärnan i simuleringen finns i klassen Simulation:

```
class Simulation {
public:
    Simulation () : eventQueue(), currentTime(0) { }

    // Add a new event to event queue.
    void scheduleEvent (Event * newEvent);

    int getTime() const { return currentTime; }

    void run();

private:
    int currentTime; // Time for last processed event

     /* The event queue. Always sorted with respect to the times
        for the events. The event with the 'smallest' time is always
        placed first in queue and will be processed next. */

   priority_queue<Event*,vector<Event*>,EventComparison> eventQueue;
};
```

Hamburgerbaren implementeras genom en instans av klassen BurgerBar:

```
class BurgerBar {
public:
   BurgerBar(Simulation *sim, int chairs);

   // True if enough free chairs, no chairs are taken.
   bool tryEnter (int nPeople);

   void order(int nBurgers);

   // True if enough free chairs, in that case the chairs are taken
   bool getChairs(int nPeople);

   void serve(int nBurgers);
   void leave(int nPeople);
   void closeDown();
   int getFreeChairs() const { return freeChairs; }
   double getProfit() { return profit; }
private:
   Simulation *theSim;
   int freeChairs;
   int nChairs; // Nr of chairs
   double profit;
};
```

Event-klasserna kommunicerar både med Simulation och BurgerBar genom att pekare skickas till dessa objekt via konstruktorerna. Hur detta ser ut i praktiken framgår av huvudprogrammet (McSmackSim.cpp):

```
...
int chairs;
cout << "How many chairs? ";
cin >> chairs;

Simulation *theSim = new Simulation;
BurgerBar *theBar = new BurgerBar(theSim,chairs);

int simTime = 0; // Simulation time starts at time 0
int groupSz;

// Load event queue with a number of initial arrival events
while (simTime < SIM_TIME) {
   // Advance simulation time some random minutes...
   simTime += randBetween(MIN_ARRIVAL_DELAY,MAX_ARRIVAL_DELAY);

   // Decide size of group to arrive
   groupSz= randBetween(MIN_GRP_SIZE,MAX_GRP_SIZE);

   if(simTime>=SIM_TIME) // Schedule the close down
      theSim->
      scheduleEvent(new ClosingEvent(theSim,theBar,SIM_TIME));
   else // Schedule a new arrival
      theSim->
      scheduleEvent(new ArriveEvent(theSim,theBar,simTime,groupSz));
}
```

```
// Run simulation and get profits!
theSim->run();

cout << "Total profits " << theBar->getProfit() << endl;
delete theSim;
delete theBar;
...
```

Exempel på utskrifter från en körning:

```
8 chairs
Time 3: Group of 3 customers arrives. In queue to order.
Time 4: Group of 2 customers arrives. In queue to order.
Time 4: 3 BigSmack ordered.
Time 5: Group of 3 customers arrives. In queue to order.
Time 5: 2 BigSmack ordered.
Time 7: Served 3 BigSmack, searching for chairs.
Time 7: Served 2 BigSmack, searching for chairs.
Time 8: 3 customers found chairs, seated. Free chairs: 5
Time 8: 2 customers found chairs, seated. Free chairs: 3
Time 8: Group of 6 customers arrives. No free chairs, they leave.
Time 9: 3 BigSmack ordered.
Time 10: Served 3 BigSmack, searching for chairs.
Time 11: 3 customers found chairs, seated. Free chairs: 0
Time 12: Group of 7 customers arrives. No free chairs, they leave.
Time 13: Group of size 2 leaves. Free chairs: 2
...
Time 479: Group of size 3 leaves. Free chairs: 5
Time 479: Group of 10 customers arrives. No free chairs, they leave.
Time 480: CLOSING DOWN - no entrance
Time 490: Group of size 3 leaves. Free chairs: 8
Time 490: 7 customers found chairs, seated. Free chairs: 1
Time 505: Group of size 7 leaves. Free chairs: 8
Time 505: 6 customers found chairs, seated. Free chairs: 2
Time 514: Group of size 6 leaves. Free chairs: 8
Chairs: 8 Profit: 198
```