ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ № 1 (2)

За курсом «Прикладний системний аналіз»

Студента групи ПА-19-2

Ільяшенко Єгора Віталійовича

Кафедра комп’ютерних технологій, ДНУ

Завдання

Потрібно побудувати імітаційну модель роботи каси з продажу квитків за таких параметрів роботи. Відвідувачі приходять до каси через 10 ... 30 сек. Оглядають приміщення – 0 ... 15 сек. і займають чергу. У касі працюють два касири. Кожен касир витрачає на обслуговування відвідувача однаковий час – приблизно 15 ... 25 сек. Каса працює 5 год.

Результати

Text

Description automatically generated

Через достатньо високу швидкість обслуговування, черга не встигала накаплюватися і більшість відвідувачів пройшла через першого касира.

Код C#

new MyProgram();

while (true)

Console.ReadKey();

interface IIterational

{

void Iteration();

}

public class Customer : IIterational

{

(int, int) staringIntoSpaseDelayBounds = (0, 15);

int staringIntoSpaseDelay = 0;

Action<Customer> toQueue;

public Customer(Action<Customer> toQueue)

{

staringIntoSpaseDelay = MyProgram.rnd.Next(

staringIntoSpaseDelayBounds.Item1,

staringIntoSpaseDelayBounds.Item2);

this.toQueue = toQueue;

}

public void Iteration()

{

if (staringIntoSpaseDelay > 0)

{

--staringIntoSpaseDelay;

return;

}

MyProgram.freeCustomers.Remove(this);

toQueue(this);

}

~Customer()

{

}

}

public class CustomerGenerator : IIterational

{

bool isReady = true;

(int, int) generationDelayBounds = (10, 30);

int generationDelay = 0;

Action<Customer> toQueue;

public CustomerGenerator(Action<Customer> toQueue)

{

this.toQueue = toQueue;

}

public int generatedCounter = 0;

public void Iteration()

{

if (!isReady)

{

--generationDelay;

if (generationDelay > 0)

return;

isReady = true;

}

else

{

isReady = false;

MyProgram.freeCustomers.Add(new Customer(toQueue));

++generatedCounter;

generationDelay = MyProgram.rnd.Next(generationDelayBounds.Item1, generationDelayBounds.Item2);

}

}

}

public class Worker : IIterational

{

(int, int) processingDelayBounds = (15, 25);

int processingDelay = 0;

bool isOccupied = false;

public Worker()

{

}

void RefreshProcessingDelay()

{

processingDelay = MyProgram.rnd.Next(

processingDelayBounds.Item1,

processingDelayBounds.Item2);

}

public int passedCounter = 0;

public void Iteration()

{

if (isOccupied)

{

--processingDelay;

if (processingDelay == 0)

{

++passedCounter;

isOccupied = false;

}

}

if (!isOccupied)

if (MyProgram.customers.Count > 0)

{

MyProgram.customers.Dequeue();

isOccupied = true;

RefreshProcessingDelay();

return;

}

}

}

public class MyProgram

{

public static List<Customer> freeCustomers = new();

public MyProgram() => StartSimulation();

const int simulationTime = 5 \* 60 \* 60; //h -> m -> s

int currentTime = simulationTime;

List<Worker> workers = new()

{

new(),

new()

};

static public Random rnd = new();

public static Queue<Customer> customers = new Queue<Customer>();

CustomerGenerator customerGenerator;

public void AddCustomerToQueue(Customer customer)

{

customers.Enqueue(customer);

}

void StartSimulation()

{

customerGenerator = new(AddCustomerToQueue);

ProgramClock();

}

void ProgramClock()

{

while (currentTime > 0)

ClockIteration();

GenerateLog();

}

void GenerateLog()

{

Console.WriteLine("Log");

Console.WriteLine("");

Console.WriteLine($"Simulation Time: {simulationTime}");

Console.WriteLine($"Current Time: {currentTime}");

Console.WriteLine("");

Console.WriteLine($"Generated customers: {customerGenerator.generatedCounter}");

Console.WriteLine("");

Console.WriteLine($"First worker handled: {workers[0].passedCounter} customers");

Console.WriteLine($"Second worker handled: {workers[1].passedCounter} customers");

Console.WriteLine($"Overall customers handled: {workers[0].passedCounter + workers[1].passedCounter}");

Console.WriteLine("");

}

void ClockIteration()

{

--currentTime;

customerGenerator.Iteration();

List<Customer> buffer = new();

foreach (Customer customer in freeCustomers)

buffer.Add(customer);

foreach (Customer customer in buffer)

customer.Iteration();

foreach (Worker worker in workers)

worker.Iteration();

}

}