ASANSÖR SİMÜLASYONU

Kocaeli Üniversitesi Bilgisayar Mühendisliği Yazılım Labaratuvarı 1 – 2. Proje

Emre Yelbey
Kocaeli Üniversitesi
Bilgisayar Mühendisliği
180202043
emre_yelbey@hotmail.com

Ege Özeren
Kocaeli Üniversitesi
Bilgisayar Mühendisliği
180202047
ege_ozeren@gmail.com

Hazırlanan program bir avmye girip çıkan müşterilerin taşınmasını sağlayan asansör sistemini simüle eder.

I. GIRIŞ

Program bir avmye belirli aralıklarla giriş ve çıkış yapan inanların, aansörlerle taşınmasını modeller.

Program kullanıcı tarafından müdahale edilmeksizin kod içerisinde belirlenmiş sabit değişken değerler doğrultusunda çalışır.

Programda birden fazla iş parçacığı aynı anda çalıştırılıp kullanılmıştır. Temelde 4 ana iş parçacığı bulunur. Bunlar "Login", "Exit", "Elevator" ve "Kontrol" classlarıdır.

II. TEMEL BILGILER

Program **C#** programlama dilinde geliştirilmiş olup geliştirme ortamı olarak **Visual Studio 2019 16.7.6** kullanılmıştır.

III. TASARIM

Asansör simülasyonu programının geliştirme aşamaları belirtilen başlıklar altında açıklanmıştır.

A. Algoritma

Projedeki problem temel olarak aynı anda meydana gelen birden fazla olayın yürütülüp kontrol edilmesidir.

Bu olaylar 4 başlık altında toplanabilir:

1. Login

Bu thread her 500 ms'de bir, bir grup insanın avmye giriş yapmasını sağlar. Bu class parametre olarak zemin katı alır ve Grup classından türetilen insan gruplarını zemin katın kuyruğuna ekler.

2. Exit

Bu thread her 1000 ms'de bir, bir grup insanın avmden çıkış yapmasını sağlar. Bu class parametre olarak tüm katların bulunduğu diziyi alır ve içerisinde insan bulunan katlardan rastgele birisinden, 1-5 arasında rastgele sayıda insanı çıkış yapmak üzere bulunduğu katın kuyruğuna ekler.

3. Kontrol

Bu thread tüm kuyruklarda oluşan insan yoğunluğuna göre asansörleri aktif veya deaktif duruma getirir. Başlangıçta tek asansör çalışır. Kuyruklardaki toplam insan sayısı mevcut asansör kapasitesinin 2 katına çıktığında yeni bir asansör devreye girer. Yoğunluk düştüğünde ise aksi şekilde bir asansör deaktif duruma getirilir.

4. *Elevator*

Bu thread programdaki en ağır yükü üstlenen ve en fazla kodu içeren classtır. İlk yaptığı iş zemin katta kuyruk oluşmuşsa kapasitesini aşmayacak şekilde mevcut insanları almaktır. Daha sonra aldığı bu grupların gitmek istediği katlara sırasıyla bakarak, bu katlardan en uzaktaki katı hedef olarak belirler ve harekete başlar.

Asansörün her kat arası ilerlemesi 200 ms kadar süre alır. Asansör yukarı yönde hareket yaparken her katta bırakılacak insan olup olmadığını kontrol eder. Eğer varsa insanları o katın içerisne bırakır ve kat bilgilerini günceller. Hedef olarak belirlenen en uzak kata ulaşıldığında asansör tekrar aşağı yönde harekete başlamadan önce üst katlarda kuyruk oluşup oluşmadığını kontrol eder. Eğer oluşmuşsa öncelikle o kattaki insanlar zemine gitmek üzere alınır ve aşağı yönde harekete başlanır. (Bu kontrol yapılmazsa giriş kuyruğunda rastgele oluşturulan gruplardan hiçbir zaman kuyruk oluşan kat gelmeyebilir ki bu durum da kuyruk oluşan katta yığılmaya sebep olur).

B. Yöntem

Proje yönteminde şu şekilde bir yöntem izlenmiştir:

Avmdeki katları temsil etmek üzere Floor adında bir class tanımlanmıştır. Bu classın en önemli yanı her katın içersindeki insan gruplarını tutan Grup objelerinden oluşan bir listeye ve çıkış yapmak üzere bekleyen insan gruplarını tutan Grup objelerinden oluşan bir kuyruk yapısına sahip olmasıdır.

Aynı şekilde Elevator classından türetilen asansörlerde de her asansörün içerisindeki katlar arasında taşınacak insanları tutan Grup objelerinden oluşan bir liste bulunmaktadır.

Programın yaptığı iş algoritmik ifadeyle en temel şekilde bu insan gruplarını katlar ve asansörler arasında sorunsuz şekilde transfer etmektir.

C. Kullanılan Bazı Metodlar ve Classlar

- ***private void MusteriAl( ):***

Bu metod asansörün gereken durumlarda katlardan müşteri almasını sağlar.

- ***public void MusteriBirak( ):***

Bu metod asansörün gerek durumlarda katlara müşteri bırakmasını sağlar.

- ***public void HedefBelirleveHareketEt( ):***

Bu metod asansörün içindeki müşterilerin gitmek istediği katları göz önünde bulundurarak bir hedef belirler ve asansörün hedefe ilerlemesini sağlar.

- ***public void CikisKuyrukKontrolu( ):***

Bu meted asansörün yukarı yöndeki hareketi bittikten sonra üst katlarda kuyruk oluşup oluşmaması üzerine asansörün gideceği yönü belirler.

- ***public class Elevator:***

Bu class avmdeki asansörleri temsil eder.

- ***public class Login:***

Bu class avmye giriş yapmak üzere her 500 ms'de bir, bir grup insan oluşturur.

- ***public class Exit:***

Bu class avmden çıkış yapmak üzere her 1000 ms'de bir, bir grup insan oluşturur.

D. Karşılaşılan Bazı Sorunlar

- Bu proje ile ilk kez birden fazla çekirdekle çalışmayı gerektiren bir probleme odaklandık. Multithreading mantığını kavrayıp proje içerisinde kullanırken oldukça zorlandık. Birden fazla threadin aynı yapıya aynı anda müdahil olması programın işleyişini bozup exeption fırlatmasına sebep oluyordu.

- Asansörlerin aktiflik durumlarını kontrol eden threadi yazarken threadlerin durdurulup başlatılması konusu bir süre karışıklığa sebep oldu.

E. Kazanımlar

- Birden fazla thread ile çalışmayı zor da olsa öğrendik.

- Queue ve List yapılarını daha efektif şekilde kullanmayı öğrendik.

- Algoritma becerimize pozitif katkıda bulundu.

KAYNAKÇA

[1] https://docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/keywords/lock-statement

[2] https://docs.microsoft.com/tr-tr/dotnet/api/system.invalidoperationexception?view=net-5.0

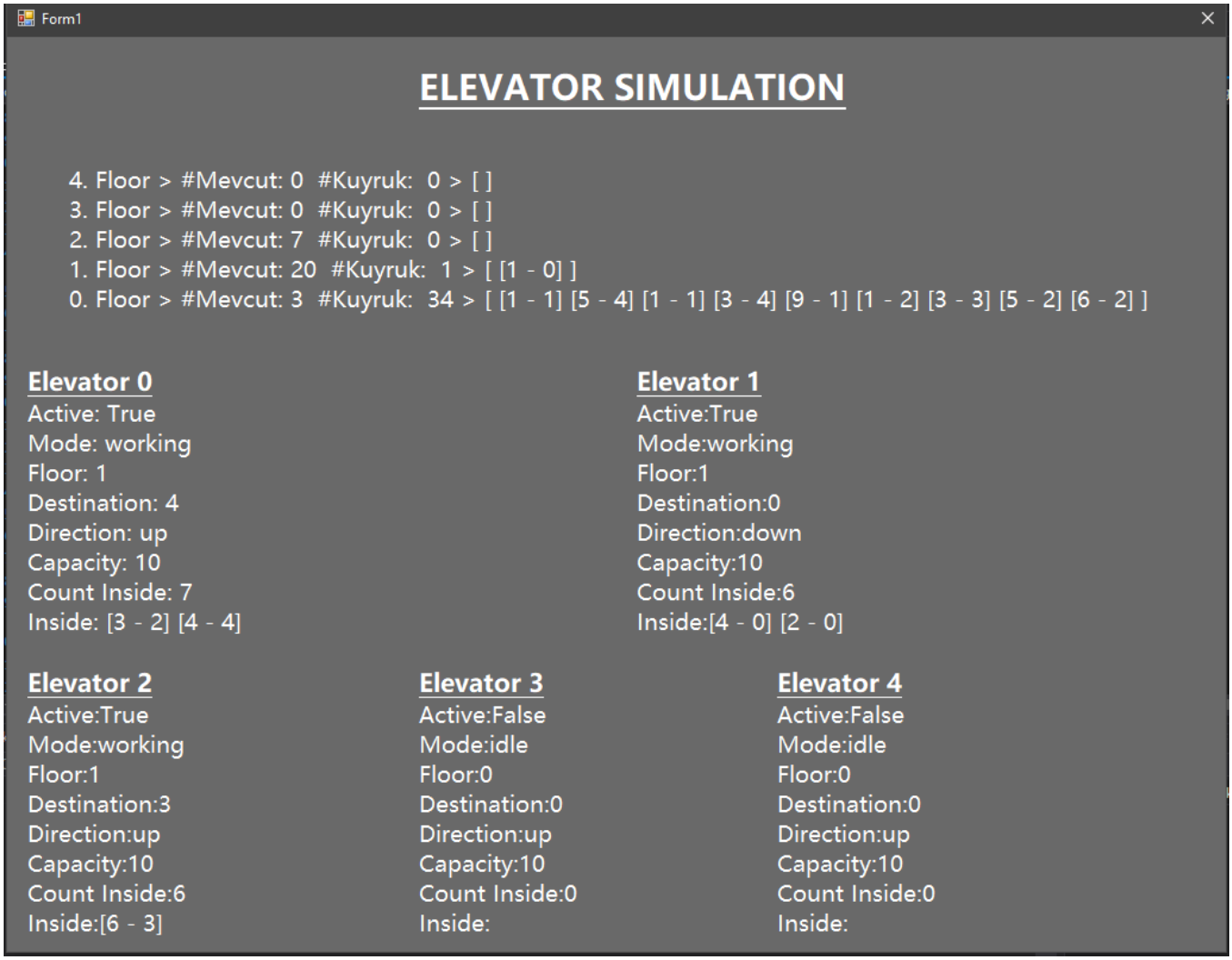
[3] https://stackoverflow.com/questions/541194/c-sharp-version-of-javas-synchronized-keyword

[4] https://www.gencayyildiz.com/blog/cta-lock-anahtar-sozcugu/

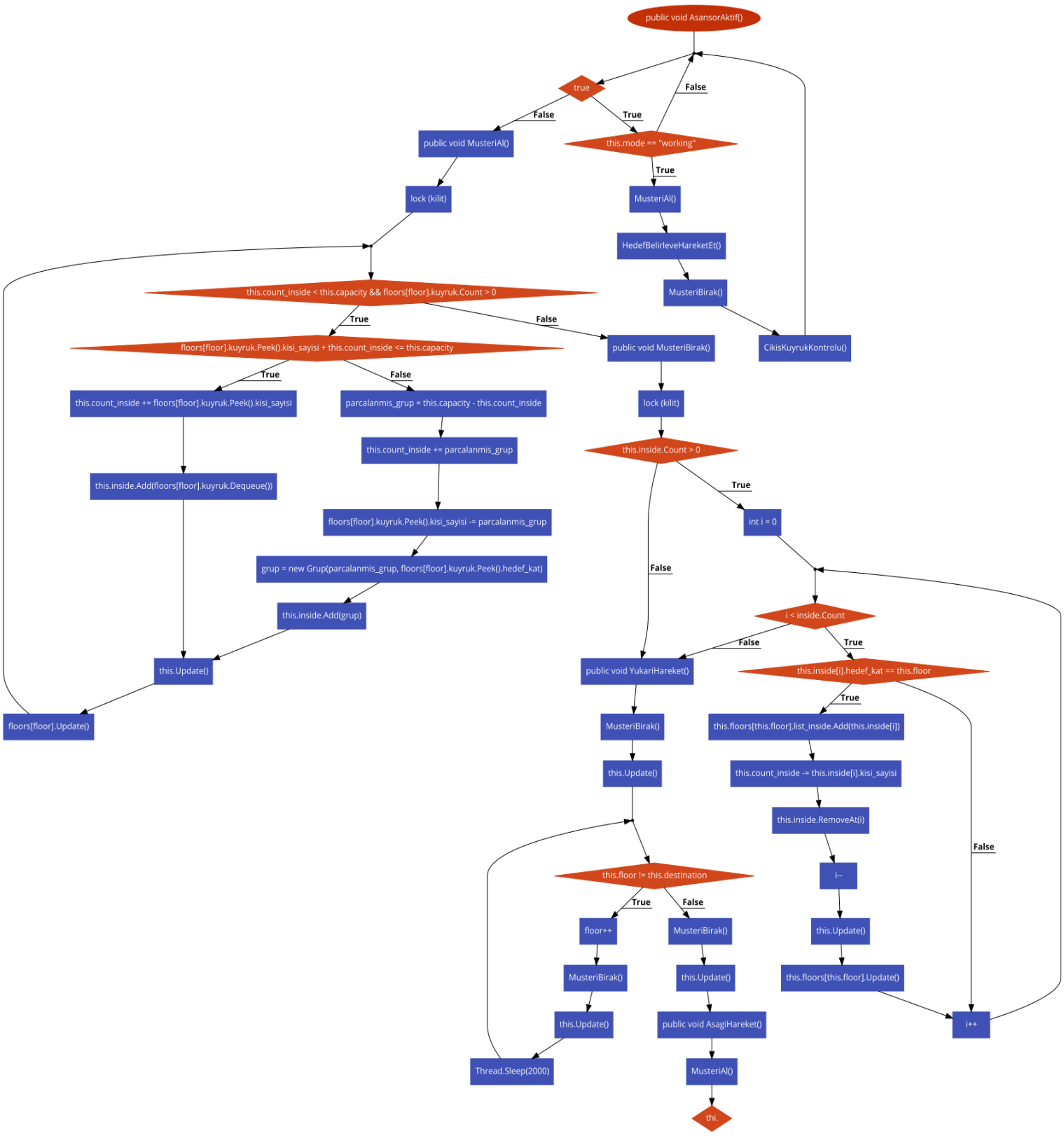
[5] https://www.youtube.com/watch?v=7ENFeb-J75k

[6] https://www.youtube.com/watch?v=TCd8QIS-2KI

[7] https://www.c-sharpcorner.com/article/introduction-to-multithreading-in-C-Sharp/



AKIŞ ŞEMASI


public void AsansorAktif()
true
False
True
public void MusteriAl()
this.mode == "working"
lock (kilit)
True
MusteriAl()
HedefBelirleveHareketEt()
this.count_inside < this.capacity && floors[floor].kuyruk.Count > 0
MusteriBirak()
True
False
floors[floor].kuyruk.Peek().kisi_sayisi + this.count_inside <= this.capacity
public void MusteriBirak()
CikisKuyrukKontrolu()
True
False
this.count_inside += floors[floor].kuyruk.Peek().kisi_sayisi
parcalanmis_grup = this.capacity - this.count_inside
lock (kilit)
this.count_inside += parcalanmis_grup
this.inside.Count > 0
this.inside.Add(floors[floor].kuyruk.Dequeue())
True
floors[floor].kuyruk.Peek().kisi_sayisi -= parcalanmis_grup
int i = 0
grup = new Grup(parcalanmis_grup, floors[floor].kuyruk.Peek().hedef_kat)
False
this.inside.Add(grup)
i < inside.Count
this.Update()
False
True
public void YukariHareket()
this.inside[i].hedef_kat == this.floor
floors[floor].Update()
True
MusteriBirak()
this.floors[this.floor].list_inside.Add(this.inside[i])
this.Update()
this.count_inside -= this.inside[i].kisi_sayisi
this.inside.RemoveAt(i)
False
this.floor != this.destination
i--
True
False
floor++
MusteriBirak()
this.Update()
MusteriBirak()
this.Update()
this.floors[this.floor].Update()
this.Update()
public void AsagiHareket()
i++
Thread.Sleep(2000)
MusteriAl()
thi.

UML DİYAGRAMI