Luleå tekniska universitet
Skellefteå

Laborationsrapport

Kurs: S0006D, Datorspels AI

Labb 1

Morgan Nyman

mornym-9@student.ltu.se

Källkod: github.com

3 februari 2021

Luleå tekniska universitet
Skellefteå

Problemspecifikation **1**

Användarmanual **2**

Algoritmbeskrivning **2**

Systembeskrivning **3**

Lösningens begränsningar **4**

Diskussion **5**

Problemspecifikation

Iscensätt en värld med minst 4 agenter, dessa ska kunna göra följande:

1. jobba för pengar, 2 olika platser och former.
2. köpa mat och vatten.
3. sova.
4. äta.
5. dricka.
6. umgås socialt med de andra agenterna.

Varje agent är utrustad med förmågan att skicka sms för att meddela sina kamrater att de är välkomna att umgås.
som de använder för att kommunicera en träff. När en Agent är hand och meddelar därför varandra när det går att umgås.

Användarmanual

Projektet finns på https://github.com/mogggen/AI_Lab_1
Det finns 5 färgade kvadrater:
Grönt är affär där agenter kan handla.
Rosa är en park där agenter kan umgås.
Röd och blå är arbetsplatser där agenter väljer den närmaste av de två där det jobbar och arbetar.
Grå är lägenheter där agenterna vilar och äter.

Obs! Agenternas färger har inget samband med kvadraternas.

I Lab 1.py finns en “update()” function. Ändra decimaltalet i inuti parenteserna i “time.sleep(0)” för att ändra flödet av tiden.
Agenterna ges ett startvärde för varje respektive behov där startvärdet är mellan 6-99 inklusivt. Detta ger de olika agenterna olika förutsättningar. målet är att hålla alla behov, också känt som states, mellan ett värde på större än 0 och mindre än 100 samtidigt för alla states och per agent.

Algoritmbeskrivning

Varje agent bestämmer sina handlingar baserat på vad som är viktigast just nu. I fallet att någon av de andra behoven blir mer brådskande, avbryts den pågående handlingen.

Behovet utförs tills det finns ett annat brådskande behov blir mer brådskande. Algoritmen skapar ingen buffer, det vill säga, att agenten byter så fort det finns ett nytt lägsta, ignorerar sträckan mellan de olika platserna.

Systembeskrivning

En lista med fyra agenter skapas där **Agent.getState()** körs inuti **Agent.__init__()** för att välja start värden för alla **Agent.self.states**.
Agent.need() sätter **Agent.self.worst** till största behovet. I **Lab 1.update()** så förlorar alla agenter poäng i varje **Agent.self.states** med 1 varje **Lab 1.update()** anrop. **Agent.walk()** söker i en for-loop upp detta state för den valda agenten och kallar för den agenten att gå kortaste vägen dit genom att få **Agent.tur.lt()** att titta mot **Lab 1.dest** och börja gå mot platsen med hastigheten **Agent.walkspeed**.

Agent.walk() tar in en destination som en tuple (x, y) och när den kommer nog nära börjar den uppfylla det mest brådskande behovet. I samband med att **Lab 1.update()** alltid kollar att Agenten är på väg eller på plats för det mest brådskande utav behoven. Varje agent kan endas utföra ett behov åt gången. Om alla behov inom det givna inkasserings-värdet från att passera 100 för alla behov så gör agenten ingenting tills detta inte är fallet.

Lösningens begränsningar

Källkoden måste ändras före programmet körs för att få ny indata, det går inte att ändra under körning eller i början av programmet. Varje testfall måste användaren ändra värden i källkoden och sedan starta om.
programmet har ett eget vis att räkna tid och som inte används för att bestämma tid vid möte.
Den totala väntetiden beror också på den tid det tar att utföra beräkningar. vilket gör att tiden kan variera.

Diskussion

Balanseringen gick inte som planerat eftersom jag hade hoppats att lista ut hur mycket ett behov skulle öka så att de kunde ske en balans, men det krävdes mer balansering än vad jag kunde ha hoppats på.

Jag hade en bugg där jag inte förbättrade states när dem utfördes. Vilket tog ett tag att hitta så de har varit en komplikation.

Jag avvek från rekommendationerna för att skapa detta system. Eftersom turtle, enligt python dokumentation, hade så mycket användbart så tyckte jag att det var det mer praktiska alternativet.

Agenterna träffar inte alltid av målet utan kan börja arbete utanför jobbet till exempel, men tidsmässigt så är det rimligt i hur systemet balanserar restid mellan olika platser på kartan.

Testkörning

Efter en testkörning med 4 agenter, efter 100 steg, så såg alla **self.states** ut enligt nedan:

Agent/state	work	consume	eating	hydration	rest	socialize
A	42	56	34	56	34	39
B	55	70	70	50	51	56
C	61	48	41	41	39	42
D	64	61	51	58	60	48

Där målet var att hålla dem mellan 0 < state < 100.