Luleå tekniska universitetet

Skellefteå

Laboration 3: Strategiskt AI i Python 3

S0006D, Datorspels AI

Laboration: 3

Oscar Östryd

Oscstr-9@student.ltu.se

GitHub - oscstr-9/Datorspels-AI---S0006D

15/03-2021

Innehållsförteckning

Problemspecifikation

Uppgiften i denna laboration var att skapa ett strategiskt AI som kunde med hjälp av agenter med olika roller upptäcka en värld, fälla träd och plocka mineraler, bygga byggnader och arbeta i en byggnad för att skapa olika produkter. Applikationen var även tvungen att följa vissa designkrav samt använda sig av pathfindingalgoritmen A*.

De olika betygskraven var:

Betyg 3 - Uppgiften är att så snabbt som möjligt skapa 200 st Träkol

Betyg 4 - Uppgiften är att så snabbt som möjligt skapa 20 st Järntackor

Betyg 5 - Uppgiften är att så snabbt som möjligt skapa 20 st Soldater

Jag anser att mitt program når upp till betyg 4 då mina agenter fokuserar på att så snabbt som möjligt skapa just 200 träkol och 20 järntackor samt följer alla designkrav specificerade.

Användarhandledning

För att testa mitt program är det bara att öppna och köra Main.py i ett passande program. Användaren kommer att få några prompts som hen behöver svara på, där första frågan är om hur snabbt spelet bör gå och den andra om användaren vill se lite extra information under tiden som applikationen körs. Efter det kommer agenterna gå till arbete och scenariot spelas ut.

Algoritm- och Systembeskrivning

Programmet är uppbyggt på ett objektorienterat sätt med olika klasser och datastrukturer för olika delar.

Agenterna är sin egen datastruktur som beskriver vad de har för roll, jobb, state, bas, inventory, position, path, id, en låsnings variabel och en lokal timer. Agenterna används av stateManager klassen samt baseManager klassen och är en central del i hela programmet.

StateManager filen innehåller alla olika states som i sig är olika klasser som agenterna kan ha såsom, explore och locateMaterials. State klassen använder sig av några globala variabler från andra filer exempelvis kartan och tidsmultiplikatorn, men även data från statParser filen. En agent är alltid i något state. Varje state har en execute som kallas på när agenten ska utöva sitt state. Alla execute metoder tar in samma parametrar så oavsett vilket state en agent är i kommer den att kunna exekveras.

StatParser filen körs i början av programmet och används för att läsa in alla specifika värden som används för att uppgradera agenter, bygga byggnader och hantera material från en separat text fil. Funktionerna i denna fil används både av baseManager och stateManager klasserna då de använder dessa värden för att hantera agenterna.

BaseManager filen tar hand om allt har med basen att göra. Det vill säga hur mycket material som basen har för nuvarande, vart basen befinner sig, vilka byggnader som kan byggas samt vilka byggnader som är byggda och del av basen. Byggnaderna är i sig egna klasser och fungerar på liknande sätt som de olika states som en agent kan vara i där byggnadens work metod kallas på från agenten. Liksom de olika states en agent kan vara i tar även work metoderna in samma parametrar så att inga ändringar behöver göras beroende på vad en agent ska jobba med.

Kartan och fog of war har som förväntat en koppling mellan varandra där fog of war tar in kartan via en global variabel för att skapa en lika stor 2D array som den fyller med True eller False bools beroende på om en yta har blivit hittad än. Dessa används även i Pygame filen vilket tar hand om allt som behöver ritas ut på skärmen, det vill säga kartan, fog of war och agenterna.

Agenterna använder sig även av A* för att hitta vägen till sina destinationer. Koden för A* algoritmen som används i den här labben är mer eller mindre den samma som den jag använde i lab 2, där den stora skillnaden är att den även prioriterar att undvika sumpmark och att den även inte får gå över vatten.

För mer information rekommenderar jag att läsa kommentarerna som beskriver varje metod och klass i varje fil.

Lösningens begränsningar

Eftersom allting i simulationen var tvunget att följa en verklig tid tar det väldigt långt tid att debugga om man inte snabbar upp alla beräkningar lite. Genom att göra detta kan programmet börja lagga en del och om hastigheten sätts för hög kan det till och med crasha. Eftersom applikationen vanligtvis ska köras i ett långsamt tempo då detta inte är ett problem ansåg jag att detta inte var ett alldeles för stort problem.

Diskussion

I denna laboration har jag stött på en mängd problem som jag löst med mycket testande, såsom att arbetarna plockade upp fel föremål, byggarna byggde på fel ställen och byggnaderna skapade mycket fler produkter än vad det fanns material för.

Överlag tycker att labben har varit lärorik och intressant att arbeta med men jag tycker att den var alldeles för stor med för många små krav vilket har gjort att det tagit väldigt långt tid att utföra.

Jag planerade en hel del innan jag började med labben men stötte ändå på en del saker som jag inte kunnat förutse och efter en mängd sådana tillfällen tog jag lite mer av en bruteforce approach i mitt kodande vilket jag fått städa upp i efterhand.

Men i det stora hela är jag ändå nöjd med min insatts då jag har planerat mer än vanligt och lärt mig en hel del.