Luleå tekniska universitetet

Skellefteå

Laboration 2: Pathfinding algoritmer i Python 3

S0006D, Datorspels AI

Laboration: 2

Oscar Östryd

Oscstr-9@student.ltu.se

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

23/02-2021

Innehållsförteckning

Problemspecifikation

Denna uppgift gick ut på att skapa fyra olika algoritmer för att ta reda på kortaste vägen eller snabbaste vägen att beräkna. Algoritmerna som skulle testas var a* algoritmen, brute force algoritmerna för bredden först och djupet först (bfs och dfs) samt en egen definierad algoritm. Dessa algoritmer skulle testas på tre olika kartor där tiden togs för varje algoritm för varje karta så att det lätt gick att jämföra vilken algoritm som var snabbast. Den funna vägen skulle även ritas ut på kartan så att det gick att se vilken väg algoritmen funnit. Jag anser att mitt arbete uppnått betyg 3.

Användarhandledning

För att kunna test köra laborationen behöver du först köra kommandot 'pip install pygame' i din kommandotolk för att ladda ner pygame. Efter det är det bara att köra filen Main.py i ett program som kan köra python filer och testa programmet. Du kommer få några prompts som frågar dig om vad du vill testa, när den frågar "What algorithm should be checked?: " kan du välja mellan: dfs, bfs, a*, custom och all. När du väljer en specifik algoritm får du välja vilken karta du vill testa, den kommer då bara testa algoritmen en gång. Om du väljer all så kommer du få ännu en prompt som frågar hur många checkar som ska göras. Efter det kommer varje algoritm testas på alla kartor lika många gånger som du angett. En genomsnittstid kommer att printas ut för den/de algoritmer du testat samt vägarna som algoritmerna tagit.

Algoritmbeskrivning

De olika algoritmerna som används för att hitta en okej väg mellan start och slut på en vald karta är:

A*

A* fungerar på så sätt att den använder sig av en heuristisk kalkylation för att bestämma värdet för ett steg där ett diagonalt steg kostar lite mer och utifrån det väljer den billigaste vägen mellan start och slut. Det finns många olika heuristiska metoder som fungerar för att finna en väg mellan start och slut för a* där alla kan ge olika resultat och bör användas till olika situationer. Eftersom denna labb tillät diagonal rörelse är den heuristiska metoden: diagonal distance mest passande för att den fungerar bäst när man har 8 olika riktningar att röra sig genom. Till en början använde jag mig av manhattan metoden men bestämde mig att byta då jag fann att manhattan passar bäst för situationer då man endast har 4 riktningar att röra sig genom.

Depth first search

Depth first search eller dfs går ut på att hela tiden arbeta sig i en vald riktning och när det tar stop byter man riktning till det att man funnit målet. Min dfs är rekursiv och arbetar sig med prioritet ner mot det högra hörnet. Algoritmen sätter även hela tiden sin nuvarande position till samma "state" som en vägg vilket gör att den inte går på samma yta flera gånger ifall att den åker fast.

Breadth first search

Breadth first search eller bfs är den lösning som söker av mest av kartan för att hitta målet vilket gör den till den långsammaste lösningen, däremot hittar den alltid den kortaste vägen. Bfs fungerar genom att hela tiden checka alla grannar min nuvarande position har och sedan checka deras grannar och så vidare till det att man har hittat slutet. Om en del av bfs algoritmen fastnar ger den bara upp på den delen och fortsätter med de delar som inte är fast än och kan i värsta fall söka igenom hela kartan.

Egen lösning

Min egen lösning är lite av en blandning av tidigare lösningar där jag använder mig av en bfs i mitten av kartan för att hitta den första gå-bara ytan närmast mitten som jag sätter till min startposition. Efter det använder jag mig av en ett söksätt som liknar en iterativ variant av en dfs för att hitta vägen mellan mitten och start samt mitten och mål så att jag sedan kan lägga ihop dessa vägar för att skapa en komplett väg genom hela kartan. Min största motivation till att göra på det här sättet var att jag var nyfiken över ifall det skulle vara snabbt att köra två halvor av vägen på två olika trådar

Systembeskrivning

Main filen är den som körs för att starta programmet då den skapar kartan genom att parsa en text fil. Kartan som den skapats tillsammans som de inputs användaren givit skickas sedan till Pathfinder filen där den beräknar den bästa vägen beroende på vilken algoritm som blivit vald. Vägen blir sedan returnerad tillsammans med tiden det tog att köra algoritmen tillbaka till Main filen. Tiden printas ut direkt medan vägen skickas till min egen Pygame fil som tar hand om allting med pygame. Där skapar pygame först blir initierad så att den korrekta skärmstorleken kan bestämmas beroende på storleken på kartan. Sedan ritar Pygame filen ut en grafisk variant av kartan som blev parsad från textfilen och den eller de vägar som skickats till den så att användaren kan se vilken väg algoritmen tog för att komma från start till mål.

Lösningens begränsningar

De olika algoritmerna fungerar bra till olika saker men generellt sätt skulle jag säga att a* är den bästa att använda. Min egen lösning förväntade jag mig inte att den skulle vara bättre vilket den i slutändan inte var och då genom att använda fler trådar till en väg skapade sina egna problem och kommer vara väldigt ineffektivt ifall man skulle beräkna ett flertal vägar för olika agenter i exempelvis ett spel, däremot var det intressant experiment.

Diskussion

Denna labb har varit väldigt intressant då jag tidigare inte kände till något speciellt sätt för att bestämma hur en agent skulle kunna ta sig från punkt a till punkt b. Jag är även ganska nöjd med min egen insatts då jag gjort det mesta utan någon hjälp, fast de gånger som jag körde fast fick jag bra hjälp ifrån mina klasskamrater Magnus Lindh och Morgan Nyman.

Labben överlag var rolig att arbeta på men den kändes ganska stor och tog därför väldigt långt tid vilket är varför jag skulle önskat lite mer tid eller alternativt en lite kortare labb där att göra sin egen lösning kanske skulle kunnat vara en bonus uppgift eller liknande.

Testkörningar

Under testkörningarna av min egen lösning märkte jag hur på den ibland fastnade och ibland blev kön av grannar tom innan vägen funnit mål, men efter ett flertal tester på olika kartor insåg jag att jag började få race conditions då båda trådarna använde sig av samma lista av vägar de gått på och murade därför in varandra, lösningarna till detta saktade ner algoritmen märkbart.