INF-2101 Algoritmer

Obligatorisk oppgave 2

Universitetet I Tromsø, NT-fakultet
Counter Logic Programming
Håvard H. Johansen og Peter H. Haro
10/3/2012

Dette er en gjennomgang av en sti søkings program brukt for å fine fram i en labyrint. Labyrinten kan være luket med ingen åpne rom eller den kan være et stort åpent rom. Hvor målet kan være mitt i et rom eller ved en veg. Dermed er veldig mange av programmene som eksisterer ubrukelige. Dette programmet takler overgangene mellom åpne rom og lukede labyrinter.

1. Introduksjon

Programmet skal finne en vei gjennom en tilfeldig generert graf ved bruk av så få trekk som mulig.

Programmet skal ikke være raskest i CPU tid for å finne veiene. Det skal derimot utforske høyst mulig antall felter på færrest mulig trekk. Programmet skal fortsatt kunne finne neste trekk innen en rimelig tid.

1.2 Teknisk bakgrunn

1.2.1 Dijkstras[1]

Dijsktras algoritme finner en sti med lavest vekt mellom to vilkårlige noder i en graf, dvs. den korteste stien. Algoritmen forutsetter at kantene mellom nodene ikke har negativ vekt, og at dersom det ikke foreligger en direkte kant er stien i utgangspunktet satt med uendelig vekt. Etter å ha satt start noden, (noden man ønsker å traversere fra), til vekt 0, markerer man restene av nodene som ubesøkt. Deretter regner man ut de tentative distansene mellom noden vi står på nå, og alle nodene, den har kant til. Dernest går man til noden med lavest vekt, og legger den inn i besøkt listen, og anvender identisk logikk til listen av ubesøkte noder er 0. Implementasjonsspesifikkt utnytter man seg av trekantulikheten for å optimalisere denne prosessen. Videre følger det etter induktivt bevis at alle sub-grafer av en korteste sti er korteste-stier, dermed finner Dijkstra ikke bare korteste sti fra a, til b, men alle korteste stier fra a til alle noder som man kan gå til fra a.

1.2.2 Labyrinter

Det er to hovedtyper labyrinter lukede labyrinter som består kun av korridorer hvor en alltid har en veg ved siden av seg så fremst man ikke står i et krys. Den andre har mange åpne rom hvor en ikke har noen veier i nærheten av seg.

2. Design

Algoritmen må kunne takle overgangen fra åpne rom til lukede områder. På grunn av disse begrensingene valgte vi å la algoritmen gå i en «tilfeldig retning» hver gang den skal bevege seg fra en node, men hvis den skulle ende på en node uten noen ikke utforskede noder rundt seg vil den bruke en Dijkstras algoritme for å finne den nærmeste ikke komplette noden rundt seg, det vil si den nærmeste noden som ikke er linket til 4 andre noder. Dijkstras trenger et mål å gå til så dermed må det implementeres en måte å finne et mål å gå til. Dette målet vil da være den nærmeste noden så vil Dijkstras finne den korteste veien til denne node.

2.2 2.1 Graph

Graph klassen holder en liste med noder som er et kart over den delen av labyrinten som spilleren har registrert. Hvilke noder som liger ved siden av hverandre og hvorvidt kontakten er mellom to tile noder eller wall node. Den inneholder også en dijkstra funksjon.

1

Dijkstra's algorithm. (2012, 10 5). Hentet fra wikipedia.org: http://en.wikipedia.org/wiki/Dijkstra's_algorithm

2.3 Player

Denne klassen er selve spilleren som beveger seg gjennom labyrinten. Klassen inneholder et labyrint objekt som kommuniserer med serveren. Et Graph objekt for å holde kontroll på hvor spilleren har hvert. Uten om dette holder den bare kontroll på hvor spilleren er og hvilke noder som er kant noder.

3. Implementasjon

I praksis er å implementere en metode for å finne den nærmeste noden effektivt og samtidig garantere at det faktisk er den nærmeste uten å lage en ekstremt komplisert algoritme. Dermed benytter dette programmet seg av en liste over nyeste ikke fulle noder. Det går så igjennom og sjekker at de faktisk fortsatt er en ikke full node. Dermed har en fått den tilnærmet nærmeste noden. Det kan være noen nærmere, men ikke så ofte at det er verdt det ekstra arbeidet.

4. Diskusjon

Den beste måten å løse denne typen labyrinter er å bruke genetiske algoritmer og gi den mange tusen labyrinter slik at den vil kunne gjøre statistiske valg. Ved ekstrem oppgraderinger av denne vil den kunne gjenkjenne mønstre og kunne se hvor de største områdene er maksimere effektivitet av look funksjonen. Spesielt når den koster 1 min per bruk. Men selv ikke denne vil aldri kunne gjøre noe annet enn en «kvalifisert gjetning». Dermed vil den noen ganger kunne bli slått av en random algoritmer som går i en tilfeldig retning og tilfeldig vis ender på rett sted. Uansett så er statistisk tilnærming den eneste effektive måten å takle åpne rom og look funksjonen.

Algoritmen brukt her kunne hvert effektivisert kraftig den enkleste måten å gjøre dette på ville være å legge inn noder som ikke har blidt besøkt enda og istedenfor å gå til nærmeste node som har en ikke søkt rute ved siden av seg, så går algoritmen til nærmeste ikke besøkte node. Det betyr at med bruk av look funksjonen kan en øke effektiviteten i en åpen labyrint med en faktor på 2.5-3 avhenge i størrelsen dvs. hvor mange ganger den må snu. Hvis desto større labyrinten er desto større blir effektiviseringen. Sånn den er i dag er look nesten ubrukt siden en uansett i de fleste tilfeller vil gå over rutene som look viste oss.

Utregningen av Dijkstras algoritmen brukt i dette programmet kan effektiviseres ved bruk av en Fibonacci heap. Det vil ikke ha noen effekt på resultatet. Så antall trekk som må gjøres vil ikke synke dermed er det ikke noe poeng i å legge det inn i dette programmet.

Konklusjon

Algoritmen klarer å søke seg igjennom labyrintene og klarer å finne Toby i de labyrint eksemplene som er gitt, den er langt ifra den mest effektive som kunne vert implementert, men den er god nok til å gå igjennom en labyrint. Den takler åpne rom, men er ikke effektiv på dette. Algoritmen er ikke veldig rask i utregningene men den er langt ifra treg, mer en rask nok i utregningene for denne oppgaven.