Luleå tekniska universitet

Skellefteå

Laborationsrapport laboration 2

S0006D

Laboration 2

Pontus Stenlund

Ponste-5@student.ltu.se

S0006D AI/S0006D_Laboration_02_Ponste-5 i SVN

2017-02-27

Innehåll

Pontus Stenlund, ponste-5@student.ltu.se

1. Problemspecifikation

Uppgiften gick ut på att implementera ett AI som skulle utforska kartan, skaffa resurser och sedan producera träkol, järntackor etc.

Vi skulle använda oss av pathfinding från föregående laboration för att kunna navigera på kartan.

AI skulle bestå av 50 stycken basagenter (arbetare) som bara kan gå på den upptäckta delen av kartan.

Sedan kan arbetarna uppgraderas till upptäckare, hanterverkare och soldater.

Agenterna skulle röra sig med en viss hastighet när dom går och varje arbetsuppgift som de hade skulle ta en viss tid.

Hastigheten som de skulle röra sig med skulle vara 1 m/s. Och på träskland skulle de röra sig 0.5 m/s. Varje ruta på kartan representerar 10 m.

Jag anser att jag har uppnått lösningen för betyg 3.

2. Användarhandledning

Projektet finns på SVN i mappen som heter S0006D
AI/S0006D_Laboration_02_Ponste-5.zip

2.1 Köra lösningen

Autodesk stingray måste vara installerat. Sedan när man har laddat in projektet i stingray så trycker man ”Test level” eller ”Run project” så sköter sig AI ’et av sig själv.

3. Algoritmbeskrivning

3.1 A*

Varje nod har värden i sig som beskriver kostnaden för att gå till den noden.
Heuristiska värdet (H-värde): Hur långt det är till målet i steg från given nod. I min implementation av A* så har jag använt Manhattan Distance för att räkna ut kostnaden.
G-värdet: Är summan av alla förflyttningar man har gjort hittills från startpunkten.
F-värdet: Är totala kostnaden av H+G.
Genom att beräkna alla kostnader och sedan välja den nod som har lägst F kostnad så kan vi få ut den kortaste vägen från start till mål.
Algoritmen använder sig av en öppen och stängd lista för att hålla reda på den kortaste vägen. Den öppna listan innehåller alla noder vi har utvärderat.
Den stängda listan innehåller alla noden vi har gått till.

1. Vi adderar startnoden till den öppna listan.
2. Vi repeterar följande:
3. Vi kollar i den öppna listan med noden med lägst F-kostnad och sätter den som aktuell nod.
4. Vi flyttar den till den stängda listan.
5. Kollar de åtta noder som ligger runt oss. Så länge noden går att gå på och inte redan finns i den stängda listan så lägger vi till den i den öppna listan och sätter aktuell nod som förälder.

   Om den redan finns i den öppna listan så tar vi och jämför G-kostnaden och om den är lägre så ändrar vi föräldern till aktuell nod och räknar om G- värde och F- värde.
6. Vi stannar när lägger till målnoden till den öppna listan eller om den öppna listan är tom. Om vägen är funnen så går vi tillbaka från målnoden och kollar föräldrarna till vi har nått startnoden. Då har vi funnit vår väg.

Har även modifierat A* så att den har ännu en funktion som används av arbetarna. När vi tittar på giltiga grannar så kollar vi även om noden har en fog-of-war på sig. Om den har det så är det inte en giltig granne.

4. Systembeskrivning.

4.1 Project.lua

Är en fil som skapas automatiskt när man skapar projektet i Stingray.
I denna fil så har jag lagt till följande saker:

LoadMap(Path), laddar in kartan från den givna sökvägen. LoadMap anropar sedan ett plugin som är skrivet i C++. Detta plugin läser raderna från filen som anges och lägger till dem i ett lua table. Sedan går LoadMap igenom raderna och kollar på varje element i strängen och kollar ifall den sedan ska lägga till en vägg eller golv. Den sparar även positionerna på start och mål. Golven och väggarna läggs sedan till i separata tables för att kunna iterera dem senare.

InitAgents(), skapar 50 agenter och registrerar dem i Entity_Manager. Funktionen sätter även agenterna på en randomposition på kartan och tar bort fog-of-war på det området.

4.2 Entity_Manager.lua

Är en singleton som håller reda på all generell data som krävs av agenterna.

RegisterEntity(entity_id, new_entity), lägger till agentens pekare i entityList.
RemoveEntity(entity), tar bort agenten från entityList.
GetEntityFromId(id), returnerar pekaren till den entity som man begär.
GetTree(), returnerar indexet som ett träd befinner sig på.
DeepCopy(orig), kopierar kartans noder till en ny table som används för att göra A*. Anledningen varför jag gör så är att jag inte vill ändra data som redan finns i original tablen.
ScoutMap(agent), kopierar kartans noder, kör A* och sätter vägen som den finner till upptäckarens egna table currentPath.
GoToTree(agent, treeIndex), kör A* för att hitta en väg utan fog-of-war till trädet.
SetWorldMap(map), sätter Entity_Managers worldMap till map.
UpgradeToScout(), uppgraderar 10 stycken arbetare till upptäckare och sätter deras tillstånd till Upgrade_State.
CallWorker(nodeIndex), går igenom listan med alla agenter för att hitta en arbetare som inte arbetar för tillfället och kallar GoToTree(agent, nodeIndex) för att hitta vägen till trädet. Om alla arbetare är upptagna och nodeIndex inte finns i treeList så läggs den till.
CheckTree(nodeIndex), kollar ifall nodeIndex finns i treeList och returnerar en bool värde.

4.3 Worker.lua

FSM_ChangeState(new_state), sätter workers state machine's current state till new_state och state machine's previous state till current state.

FSM_SetCurrentState(new_state), sätter workers state machine's current state till new_state
Update(delta_time), kör state machine's update().
SetPosition(startX, startY), sätter workers position.

4.4 State Machine

SetCurrentState(state_type), sätter current state till state_type.
SetPreviousState(state_type), sätter previous state till state_type.
Update(delta_time), kör tillståndets Execute() och skickar sin ägare och delta time som argument.
ChangeState(newState), sätter previous state till current state och kör sedan previous state's Exit(). Sedan sätter den current state till newState och kör dess Enter().
RevertToPreviousState(), anropar ChangeState() med previous state som argument.

4.5 De olika tillstånden

Alla tillstånd har dessa tre metoder i sig som utför olika saker beroende på vilket tillstånd det är

Enter(agent), sätter i de flesta fall agent.waitTime till os.clock() för att kunna räkna ut hur lång tid vissa saker har tagit.
Execute(agent, delta_time), utför det specifika arbetet för tillståndet för agent.
Exit(agent), gör oftast ingenting

De olika tillstånden jag har gjort är:
Idle, upgrade, build, scout, make coal, work lumber och walk.

5. Lösningens begränsningar

5.1 Begräsningar

Nu kör jag laborationen i 10x hastighet. Detta är för att annars skulle det ta flera timmar för att nå målet som är att samla 200 träkol.

En annan begränsning som jag vet är att jag har hårdkodat antalet upptäckare som man uppgraderar till.

Jag har dessutom ingen ny klass för upptäckare, hanterverkare etc. utan jag tar bara och sätter om materialet på arbetarens modell. Detta gör att alla agenter har variabler och funktioner som den kanske inte behöver vilket i sin tur tar lite mer minne.

Jag har hela min karta i en enda stor table. Så för denna laboration så är den fylld med 10 000 element. Detta gör att loopar igenom den tar lite tid. Dessutom kan man inte lägga break-points för att debugga.

5.2 Hur kunde begränsningarna undvikas?

I fallet med hastighetsändringen så kunde jag bara ha kört programmet och sedan gjort annat för att få ut exakt tid.

Och för att enkelt ändra på hur många upptäckare och dylikt som jag vill uppgradera till så kunde jag bara ha lagt till några variabler längst upp i scriptet som gör att det blir lätt att ändra.

I fallet med att uppgradera en arbetare så skulle jag bara ha kunnat hämta ut agentens id och sparat undan den temporärt och sedan skapat en ny instans med den andra klassen och sedan registrerat den igen i entity managern.

6. Diskussion

Jag har stött på ett par problem.

Det största problemet som har varit är att jag är inte van vid lua som språk. Detta gör att implementationerna har varit tidskrävande och omständliga.

Och att försöka göra en objektorienterad lösning har varit ett litet problem genom att jag en längre stund försökte använda mig av arv. Men efter ett par timmar så insåg jag att det var helt onödigt.

Sedan har jag haft väldiga problem med att implementera hastighetsförändringen mellan kvadraterna.

Men allt som allt tycker jag laborationen har varit väldigt rolig och givande. Den har gett bra insikt i hur man ska strukturera upp en AI med pathfinding.

Det har även varit roligt att börja nudda på ämnet om optimering t.ex. ska alla agenter anropa A* själva eller ska man han ha en manager som hanterar all pathfinding åt dem.

Jag har tagit information och inspiration från:

http://help.autodesk.com/view/Stingray/ENU/?guid=__lua_ref_ns_stingray_html

https://www.lua.org/pil/contents.html

http://lua-users.org/

Jag vill tacka alla mina klasskamrater som har gett mig goda råd och för alla givande diskussioner som vi har haft angående labben.

7. Testkörningar

De testkörningar jag har gjort har gett mig en tid på 1729.565 sekunder, dvs 28 minuter. Detta beror mycket på att eftersom startpositionen är slumpad så tar det olika tid för programmet att skaffa 200 träkol.