

Laborationsrapport

S0006D, Datorspels AI

Laboration 1a



January 31, 2016

Anton Christoffersson

antchr-4@student.ltu.se

Contents

[Problemspecifikation 2](#_Toc442023077)

[Användarhandledning 3](#_Toc442023078)

[Algoritmbeskrivning 4](#_Toc442023079)

[Systembeskrivning 5](#_Toc442023080)

[Lösningens begränsningar 6](#_Toc442023081)

[Diskussion 6](#_Toc442023082)

[Testkörningar 6](#_Toc442023083)

[Referenser 6](#_Toc442023084)

# Problemspecifikation

I denna rapport var uppgiften att vi skulle skapa en simpel AI igenom en FSM som simulerar livet av fyra stycket karaktärer i en värld.

# Användarhandledning

Programmet behöver bara kompileras, inga parametrar behövs. Om man vill se vad som händer längre fram kan man kommentera bort ”Sleep()” funktionerna och pausa programmet istället.

# Algoritmbeskrivning

En icke-trival del som gjordes var hur agenterna beter sig när dom vill gå till ett jobb som redan är ockuperat av en annan agent. Det finns en bool ”occupied” i alla jobb states som sätts till TRUE när en agent går in i statet och FALSE när dom lämnar det. Boolen kollas när en agent kommer kör ett states ”Enter()” funktion och om occupied redan är true så skickas agenten vidare till nästa state. Eftersom det finns 4 jobb och 4 agenter så kommer alla alltid ha något att göra.

# Systembeskrivning

**Entity** är en abstrakt klass som är basen till Workers som är själva agenten den säger att varje agent måste ha en update loop, ett ID, en position och ett sätt att hantera meddelanden på.

**EntityManager** sparar alla agenter i en ”std::map” tillsammans med deras ID så att dom är lättåtkomliga.

**EntityStates** innehåller alla states och deras implementeringar.

**Locations** inehåller en enum som beskriver dom olika platserna som agenterna kan vara på.

**MessageDispatcher** tar hand om att skicka meddelanden till rätt agenter.

**MessageTypes** inehåller en enum som beskriver vad för sorts meddelande som kan skickas.

**State** är basklassen till alla states och säger att alla states måste vara singletons, ha en ”StateEnter()”, en ”Execute()”, en ”StateExit()” och en ”OnMessage()”

**StateMachine** tar han om en agents states. Har även agenters förra state så att man lätt kan gå tillbaka till der förra statet. Gjordes till en template så att man senare kan a flera olika sorters agenter.

**Telegram** är en mall för hur ett meddelande ska se ut.

**State** är basklassen till alla states och säger att alla states måste vara singletons, ha en ”StateEnter()”, en ”Execute()”, en ”StateExit()” och en ”OnMessage()”

**Workers** ärver av Entity och håller en agents olika värden som tex; törst, hunger osv.

# Lösningens begränsningar

Svårt att veta vad som händer då allting bara är text och svårt att hänga med på allt bara genom att ha det i huvet. Men när man skrev ut allas värden och namn så gjorde det allt lite lättare att hantera. Ingen riktigt timer bara en int som beskriver tid och vilken dag det är.

# Diskussion

Ett problem som stöttes på var om ett jobb var upptaget eller inte men det löstes ganska lätt genom att ha en bool ”occupied” och två olika change states, ett som inte kör ”StateExit()”.

Labben har varit väldigt rolig och intressant men jag hade velat kunna lägga ner mer tid på den så att jag kunde fixa del 4 och 5 för att få ut det grafiska vilket också verkade kul att göra.

# Testkörningar

Testkörningar innehåller att jag tagit bort sleep() i loopen så att man snabbt kan komma längre fram i ”tiden” för att se om något gått fel och om någon har dött eller har konstiga värden. Detta hjälpte i balanseringen av statesen.

# Referenser

Boken Game AI by Example.