

Laborationsrapport

S0006D, Datorspels AI

Laboration 2



February 28, 2016

Anton Christoffersson

antchr-4@student.ltu.se

Innehåll

[Problemspecifikation 2](#_Toc444445855)

[Användarhandledning 3](#_Toc444445856)

[Algoritmbeskrivning 4](#_Toc444445857)

[Systembeskrivning 5](#_Toc444445858)

[Pathfinding 5](#_Toc444445859)

[Statemachine 5](#_Toc444445860)

[Autonomrörelse 6](#_Toc444445861)

[Settings Loader 6](#_Toc444445862)

[Lösnings begränsningar 7](#_Toc444445863)

[Diskussion 8](#_Toc444445864)

[Referenser 9](#_Toc444445865)

# Problemspecifikation

I denna rapport var uppgiften att slå samman det vi gjort i tidigare labbar till ett litet AI RTS spel som gick ut på att samla 200 träkol så snabbt som möjligt.

Uppgiften baserades mycket på att balansering och prioritering.

# Användarhandledning

För att köra programmet så behöver det bara kompileras. Det finns en settings.in fil ibland källkoden som har en dal olika värden som kan ändras för att ändra bland annat max faten på gubbarna osv.

När man kört programmet så kan man se FPS:n och under den så ser man hur mycket träkol som har producerats.

# Algoritmbeskrivning

# Systembeskrivning

Doxygen html mapp finns inkluderad i inlämingen för beskrivning av class samband och lättare översikt.

## Pathfinding

Systemet är uppbyggt så att pathfinder är det centrala objektet där allting kopplas samman. Pathfinder har en funktion Load(char\* filename) som laddar in en .txt karta och fyller en member graf som pathfinder har med kartans innehåll. Först läggs alla noder till i grafen och markerar samtidigt varje nod med en ”flag” om den är unwalkable, walkable, start eller goal för att man ska veta vad för sorts tile det är senare. Sedan läggs edges till genom att kolla om dom är valid för noden och så att dom är walkable. Pathfinder har även två member variablar ”startNode” och ”goalNode” som anger vilken nod man startar och slutar på. Under inladdningen av kartan sparas även kartan/grafens höjd och längd.

GraphEdge och GraphNode är båda basklasser till edges eller nodes så att man lätt kan implementera olika sorters edges/nodes till sin graf

NavEdge är en subklass till GraphEdge och innehåller inte något speciellt då det inte behövs i denna labb.

NavNode är en subklass till grapEdge och innehåller en flag som anger om den är walkable eller liknande samt en position som beskriver nodens ”fysiska” position.

SparseGraph är själva grafen som håller noderna och dess edges. Den har en vector member som inehåller alla noder och en vector som inehåller en lista med edges så att man lätt kan komma åt en nods edges genom att kolla vilka edges som finns i EdgeListvectorn på samma index som noden har i sin vector.

## Statemachine

**Entity** är en abstrakt klass som är basen till olika enyitys, i denna labb finns det bara en sorts entity och det är ”MovingEntitys”, det finns dock flera olika sorters moving entitys som är själva agenterna. Entity klassen specificerar att varje agent måste ha en update loop, ett ID, en position, nästa position som agenten ska söka sig till, en Unit\* som används för att rita ut och ett sätt att hantera meddelanden på.

**EntityManager** sparar alla agenter i en ”std::map” tillsammans med deras ID så att dom är lättåtkomliga. Den innehåller även två listor som håller koll på vilka noder som har blivit besökta, en som används för att rita ut scouternas path som töms varje utritning så att man inte ritar ut samma sak flera gånger. Den ska även fungera som ett hivemind så den håller koll på hur många workers, socuts, builders och kilners som finns samt hur mycket kol/träd man har. Den sköter också hanteringen av uppgradering av dom olika agenterna.

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

**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 movingEntity och är själva basarbetaren som sedan kan uppgraderas till dom andra olika enheterna som fungerar på samma sätt men har olika states och prioriteter.

## Autonomrörelse

Den autonoma rörelsen fungerar genom att ha en klass MovingEntity som är basen till alla entitys som ska röra sig och ärver utav Entity. MovingEntity ger alla entitys en hastighet, en massa, en max fart och en max kraft(force).

SteeringBehavior klassen inehåller alla olika autonoma beteenden som entitys kan använda, i denna labb så används bara seek men den här klassen gör det lätta att implementera en till om det skulle behövas. Steeringbehavior räknar ut kraften som en agent ska röra sig med beroende på vart den ska någonstans och returnerar den i Calculate().

## Settings Loader

För att ladda in inställningar ifrån en fil så har jag en klass ”LoaderBase” som öppnar en och har en funktion som läser in en rad och parsear bort allt förutom float värdet och sedan castar om det ifrån en char till en string. Detta används i settings loader för att läsa in alla inställningar stegvis. Så inläsnings ordningen i settingsLoader måste vara densamma som i settings.ini filen annars kommer inte rätt variabel få rätt värde.

# Lösnings begränsningar

Just nu är inte spelet så speciellt smart eller optimerat för att producera träkol så snabbt som möjligt. Det byggs till exempel bara en kol mila och den byggs alltid precis bredvid basen. Det skickas inte bara 5 arbetare till varje trädnod vilket skulle vara betydligt smartare mm.

Lösningen har även bara ett autonomt beteende och det är seek. Var lite oklart i labb specifikationen om agenterna skull undvika varandra och träd osv men jag tror det var menat att det skulle göras.

En annan begränsning är att det laggar lite i början innan det har hittats några träd då alla workers söker igenom alla noder som upptäckts efter träd varje frame.

# Diskussion

Labben var väldigt rolig men tiden som var utsatt för att göra klart labben var väldigt felkalkylerad. Jag har nog suttit över 70 timmar med den här labben och det har varit väldigt stressigt att bli klar med den och även ha kvar nebula att göra samtidigt. Det är därför programmet har några bristningar som att agenterna inte ställer sig bredvid träd eller undviker andra agenter när dom går osv.

Annars har jag tyckt att labben har varit väldigt givande och hade gärna lagt ner mer tid på den och förfinat allt om jag hade haft tid till det.

Följde boken väldigt nära på autonomt beteende och settings loader.

# Referenser

Boken Game AI by Example.