

LAborationsrapport

S0006D, Datorspels AI  
Laboration 1b



February 13, 2016

Luleå tekniska universitet

Anton Christoffersson

antchr-4@student.ltu.se

Innehåll

[Problemspecifikation 2](#_Toc443145139)

[Användarhandledning 3](#_Toc443145140)

[Algoritmbeskrivning 4](#_Toc443145141)

[Systembeskrivning 5](#_Toc443145142)

[Diskussion 6](#_Toc443145143)

[Testkörningar 7](#_Toc443145144)

# Problemspecifikation

Denna laboration gick ut på att implementera fyra olika sökalgoritmer som skulle kunna hitta ifrån start till mål i tre olika kartar som gavs och sedan jämföra hur snabba dom olika algoritmerna var jämfört med varandra.

# Användarhandledning

För att köra programmet så behöver det bara kompileras men om man vill ändra algoritm eller map så måste man gå in i ” myWindow.cpp” och ändra där det är angivet i koden att man ska ändra.

# Algoritmbeskrivning

Algoritmen Shaggy search är byggd på Breadth first search men lägger till ett element som gör att den söker sig emot målets punkt. Algoritmen har två stycken köer, en som prioriteras över den andra. I sökningen så läggs edges som går emot målet till i den prioriterade kön medans edges som inte gör det läggs i den vanliga kön. När man sedan hämtar vägen till målet så hämtar man i första hand edges som går emot målet och om den prioriterade kön är tom så använder man sig av den vanliga kön.

# Systembeskrivning

Doxygen html finns inkluderat i zip filen för beskrivning av class samband och lättare översikt.

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.

# Lösningens begränsningar

En begränsning som programmet har är att A\* algoritmen går fel på ett ställa i map1. Den går en kostsammare väg vid början istället för att bara gå rakt på. Har försökt hitta felet till detta men inte lyckats hitta det.

# Diskussion

Labben var ganska svår och det var svårt att hinna då det var trott att vi skulle göra en navmesh också men det var väldigt intressant att använda sökningsalgoritmerna för att hitta en väg på en karta.

Har följt boken ganska nära då jag tycker att han förklarar problemen och implementationen bra oftast.

Det hade varit kul att göra uppgifterna för betyg 4 och 5 men tiden fanns inte för det.

# Testkörningar

Tider för testkörning:

BFS:

* Map1: 4.23 s
* Map2: 1.48 s
* Map3: 76.99 s

DFS:

* Map1: 0.51 s
* Map2: 0.7 s
* Map3: 14.16 s

A\*:

* Map1: 0.0025 s
* Map2: 0.0011 s
* Map3: 0.0079 s

Shaggy Search:

* Map1: 5.62 s
* Map2: 2.28 s
* Map3: 83.27 s

## Resultat

Alla algoritmer förutom DFS går snabbare när om noderna har färre edges som i map2 där noderna inte har lika många edges då kartan är formad som en labyrint. Men DFS är ändå snabbare än BFS i det fall som testats.

A\* är överlägset mycket snabbare än alla dom andra testade algoritmerna.