Testausdokumentti.

Automaattista testausta on toteutettu Javan JUnit työkalulla, joten ne ovat helposti toistettavissa. Automaattinen testaus kattaa 93% algoritmien koodista ja 93% tietorakenteiden koodista.

Käyttöliittymätesteissä testataan pääasiassa sitä, että käyttöliittymän Nodet toimivat oikein ja ne maalataan oikeilla väreillä oikeassa kohdassa. Käyttöliittymätesteissä on myös testejä, joissa hyödynnytetään Javan Robot oliota simuloimaan käyttäjän syötteitä graafisessa käyttöliittymässä. Tässä testissä kokeillaan siirtää robotin kursori tiettyyn pisteeseen ruutua ja piirretään seinää, jonka jälkeen tarkistetaan AssertTrue metodilla, että Noden väri muuttui oletetuksi. Muita keskeisiä testejä käyttöliittymälle ovat kaikkien getterien testit, joilla varmistetaan, että metodit antavat oikeat paluuarvot, ja että nodet ovat oikein synkronoituna itse käyttöliittymän taulukkoon.

Astarin automatisoiduissa testeissä annetaan algoritmille erilaisia tilanteita. Osassa joissa tilanne on melko normaali, ja toisissa esimerkiksi on alkupiste loppupisteen päällä. Tietyissä Astar testeissä on myös timeout, jonka sisällä algoritmin on pakko suorittaa tehtävä. Tämä pääasiassa sitä varten, että algoritmi ei ajautuisi ikuiseen silmukkaan, joihin se testauksen aikana aina välillä pääsi erikoisemmissa testeissä.

JumpPointSearch:in automatisoiduissa testeissä testataan pääasiassa JPS:n eri metodien toimintaa yksittäin metodi kerrallaan. Metodit testataan pääasiassa ohjelman oletussyötteillä, joiden kuitenkin tulisi päteä yleisellä tasolla.

JumpPointSearch:in pruneNeighbors metodi testataan neljään kertaan, kerran ilman vanhempaa, kerran vino suunnassa, kerran pystysuunnassa ja kerran vaakasuunnassa.

Tietorakenteita on testattu pääasiassa yksinkertaisilla getteri ja setteri testeillä, joilla varmistetaan, että näiden arvot saadaan asetettua ja haettua oikein. Testeissä, kuten aiemmissakin hyödynnetään monien metodien boolean palautusarvoa, jonka avulla tiedetään, että metodi on suorittanut loppuun asti. Gettereitä ja settereitä yleensä testataan yhdessä, jolloin asetetaan arvo setterillä ja haetaan se getterillä. Tämän jälkeen verrataan getterin palauttamaa arvoa alkuperäiseen arvoon.

Iso osa testeistä liittyen siihen, kuinka ohjelma todellisuudessa toimii, on tapahtunut kuitenkin empiirisesti. Algoritmien toimintaa on pääasiassa testattu maalamalla käyttöliittymään seiniä ja sen jälkeen ajamalla algoritmi etsimään polku maaliin. Tämä on toistettavissa melko yksinkertaisesti käynnistämällä ohjelma ja piirtämällä tietyn tyyliset seinät ruudukkoon ja ajamalla algoritmit. Testauksen perusteella ohjelma tuottaa aina saman tuloksen samankaltaisilla arvoilla.

Testaus on ollut melko kattavaa ja algoritmeilla on empiirisesti testattu niin yksinkertaisia kuin monimutkaisempiakin kenttiä, esimerkiksi Astar algoritmilla on testattu rakentaa mahdollisimman monimutkainen ”labyrintti” ruudukkoon ja tämän jälkeen ajamalla itse algoritmi. Tämän jälkeen on tutkittu polku, jota algoritmi päätti käyttää, ja selvitetty minkä takia kyseinen polku on valittu.

Testit ovat muunmoassa paljastaneet sen, että JPS algoritmi ei ensimmäisissä versioissa nähnyt ollenkaan tiettyjen seinien taakse, vaan nämä seinät jäivät niin sanottuihin sokeisiin pisteisiin. Tämä ongelma on kuitenkin saatu jokseenkin ratkaistua jälkeenpäin.

Testejä on myös suoritettu vertailutesteinä, jolloin samalle kentälle on ajettu molemmat algoritmit ja verrattu, kuinka ne eroavat toisistaan.

Suorituskyky testausta varten tein pienen erillisen käyttöliittymän joka sisältää painikkeen, joka ajaa algoritmin satatuhatta kertaa ja laskee suoritusten keskimääräisen ajan. Algoritmit ajavat kaikki ajot, jonka jälkeen vasta graafinen käyttöliittymä päivittyy, joten ne toimivat niin sanotusti päättöminä, jolloin graafiset päivitykset eivät sekoita tuloksia.

Vertailutesteissä kaikki testit on suoritettu 32x32 ruudukossa, siten että alkupiste on ylä-oikealla ja loppupiste ala-vasemmalla.

Tyhjällä kentällä A* saa keskiarvoksi: 0.99366ms, ja JPS: 1.01324ms, joten A* toimi hieman paremmin.

Yksinkertaisen seinän kanssa A* saa keskiarvoksi: 1.43248ms, ja JPS: 4.35333ms, joten voidaan todeta, että A* toimii kohtuullisen paljon paremmin kuin JPS, kun kentällä on esteitä, ja reitti ei ole suora. Yksittäisten syötteiden kanssa JPS toimii kuitenkin näennäisesti nopeammin koska JPS:n suorituksessa on vähemmän graafisia päivityksiä, jotka A* algoritmissa vievät oman aikansa.

Monimutkaisemmassa kentässä jossa algoritmit joutuvat kiemurtelemaan useamman seinän ympäri saa A* algoritmi keskiarvoksi: 1.39076ms ja JPS: 3.17719ms. A* ei hirvesti lähtenyt etsimään reittejä väärästä suunnasta, mutta JPS törmäsi seinään aika monta kertaa, joka selittää minkä takia JPS:n keskiarvo on sen verran korkeampi kuin A*:n

Voidaan todeta, että JPS hidastuu huomattavan määrän, kun reitille lisätään esteitä eikä reitti ole suora. Toisaalta, kun kenttä on tyhjä niin JPS ja A* toimivat lähes samassa ajassa.

A* algoritmi puolestaan näyttää toimivan melko vakiollisessa ajassa oli esteet minkälaiset tahansa. Kokeillaan täten tehdä A* algoritmille mahdollisimman pitkä ”labyrintti” jossa algoritmin on pakko käydä läpi jokainen node kentällä, tällä syötteellä A* sai keskiarvoksi 2.38510ms.

Nostin ruudukon resoluution 128x128 nodeen, ja ajoin algoritmit. Piirsin kentälle myös ruudukon keskelle kiemurtelevan seinän, jotta algoritmi ei pääse suoraan vinosti aloituspisteestä maaliin, vaan että, joutuu kiertämään sen joko ylhäältä tai alhaalta.

A* sai ajaksi 29.3343ms ja JPS sai keskiarvoksi 44.6149ms. Arvot näyttäisivät vastaavan samaa jakaumaa kuin 32x32 resoluutiolla. Tämä vahvistaa sitä, että aikaisemmat keskiarvojen osoittamat tulokset pätevät myös muilla resoluutioilla eikä ole vain erikoistapaus kyseisellä koolla.