Projektityön dokumentti

Labyrintti

Niklas Lindroos, 218177 TIK, vk. 2010
25.4.2013

Yleiskuvaus

Tässä projektityössä olen toteuttanut keskivaikealla vaikeustasolla labyrintin. Tekemäni ohjelma käyttää graafista käyttöliittymää, joka on toteutettu PyQt:lla. Ohjelmaa ajettaessa se luo uuden ikkunan johon piirretään labyrintti ja sen keskelle pelinappula. Pelaajan tehtävänä on ohjata pelinappula maaliin. Labyrintin oikealla puolella on joukko nappuloita joita käyttämällä pelaaja voi ohjata pelinappulaa. Nappuloiden avulla voi myös aloittaa uuden pelin, näyttää nykyisen pelin ratkaisun, tallentaa labyrintin tiedostoon ja ladata labyrintin tiedostosta sekä näyttää pelin ohjeet.

Labyrintti on niin kutsuttua Weave-tyyppiä, jossa on siltoja jotka ylittävät käytäviä. Siltoja voi olla sekä vaakasuoraan että pystysuoraan. Hyvin pienissä labyrinteissa käytäviä ei aina ole. Olen myös määrittänyt ylärajan sille kuinka monta käytävää labyrintissa maksimissaan on. Labyrintti on nk. täydellinen labyrintti, jossa kahden eri tyhjän ruudun välillä on yksi ainut polku.

Ohjelmaa on testattu vain Windows 8 käyttöjärjestelmällä, eikä toimivuutta ole varmistettu muilla alustoilla. Pienin mahdollinen labyrintti on noin 5*5 -kokoinen, eikä ohjelma toimi tämän pienemmillä labyrinteilla. Suosittelen, että ohjelmaa ajettaisiin vähintään 30*30 -kokoisilla labyrinteilla. Leveyden ja korkeuden ei tarvitse olla sama.

Käyttöohje

Ohjelma käynnistetään ajamalla mainwindow-moduuli. Pelaajan pelinappula sijoitetaan peliä aloitettaessa labyrintin keskelle. Pelaaja voi nyt liikkua korkeintaan neljään suuntaan; ylös, alas, vasemmalle ja oikealle. Seinien läpi ja seinien päälle ei voi liikkua. Labyrintissa on erivärisiä ruutuja, joiden tarkoitus on selitetty alla:

Väri	**Selitys**
Musta	Seinä
Läpinäkyvä	Käytävä
Sininen	Pelinappula
Keltainen	Pystysuora käytävä
Syaani	Vaakasuora käytävä
Vihreä	Maali/Oikea reitti maaliin

Nuolinäppäinten lisäksi ikkunassa on näppäimet erilaisille toiminnoille. ’New Game’ kysyy käyttäjältä uuden labyrintin leveyden ja korkeuden ja aloittaa uuden pelin molempien arvojen ollessa kokonaislukuja. Jos käyttäjä antaa vääräntyyppiset luvut, ohjelma tulostaa virheviestin. ’Resign’ lopettaa nykyisen pelin ja

näyttää oikean ratkaisun. ’Load’ kysyy pelaajalta tiedoston nimeä josta labyrintti ladataan. Jos tiedostoa ei ole olemassa, tai siinä on virheitä, ohjelma tulostaa virheviestin. ’Save’ kysyy käyttäjältä tiedoston nimeä, johon labyrintti tallennetaan. Jos käyttäjän antamaa tiedostoa ei entisestään ole olemassa, ohjelma luo uuden tiedoston kyseisellä nimellä ja tallentaa labyrintin siihen. ’Help’ näyttää pelin ohjeet.

Kaikille näppäimille on myös omat pikakomentonsa, joten ohjelmaa voi käyttää myös vain näppäimistön avulla. Pikakomennot ovat seuraavat:

- Nuolinäppäimet: siirry ylös, alas, vasemmalle ja oikealle
- Ctrl + N: Uusi peli
- Ctrl + Q: Lopeta peli ja sulje ikkuna (ei kysy käyttäjän varmennusta)
- Ctrl + R: Lopeta nykyinen peli ja näytä ratkaisu (ei sulje ikkunaa)
- Ctrl + S: Tallenna nykyinen labyrintti tiedostoon
- Ctrl + L: Lataa labyrintti tiedostosta
- H: Näytä pelin ohjeet

Lisäksi ikkunasta löytyy valikko, josta voi valita pelin lopettamisen ja ikkunan sulkemisen. Aja salliessa olisin liittänyt valikkoon lisää toimintoja.

Ohjelman rakenne

Ohjelma käyttää pääasiassa kolmea moduulia; mainwindow, player ja labyrinth. Tietorakenteita varten olen toteuttanut kaksi luokkaa. Luokka Stack moduulissa stack kuvaa pinoa ja sen toiminnallisuutta. Pinon takana abstraktiotasoa alempana on linkitetty lista LinkedList linkedlist-moduulissa, jota pino käyttää tietorakenteenaan. Lisäksi olen käyttänyt testaukseen print_lab-moduulia.

Käyttöliittymä on toteutettu Window-luokassa mainwindow-moduulissa. Window-luokasta kutsutaan Player- ja Labyrinth-luokkaa ja käytetään näiden toiminnallisuuksia. Alla on lueteltu Window-luokan keskeiset metodit:

- initUI(self) – Luo uuden ikkunan ja sen nappulat.
- paintEvent(self, e) – Piirtää labyrintin ikkunaan draw_rectangle():n avulla.
- draw_rectangle(self, x, y, param) – Piirtää oikean värisen neliön painEvent():n määräämään paikkaan.
- button1Clicked(self) – ’New Game’-napin toiminnalisuus
- button2Clicked(self)-button3Clicked(self) – Nuolinäppäinten toiminnalisuus
- button6Clicked(self) – ’Resign’-napin toiminnalisuus
- button7Clicked(self) – ’Load’-napin toiminnalisuus
- button8Clicked(self) – ’Save’-napin toiminnalisuus
- button9Clicked(self) – ’Help’-napin toiminnalisuus
- center(self) – Siirtää ikkunan keskelle käyttäjän ruutua
- closeEvent(self, event) – Muuttaa oletusarvoista sulkemistoimintoa

Lisäksi mainwindow-moduulissa on main()-funktio joka luo Window-olion.

Labyrinth-luokan keskeiset metodit:

- __init__(self, size) – Ottaa parametrina kahden pituisen listan, jossa on uuden Labyrinth-olion leveys ja korkeus. Kutsuu generate()-metodia, joka luo labyrintin.
- generate(self) – Metodi, joka luo labyrintin depth-first searchin avulla.
- set_bridge(self) – generate()-metodin kutsuma metodi, joka luo sillan labyrinttiin kun se on mahdollista.
- solve(self, coordinates) – Ratkaisee labyrintin annettujen koordinaattien perusteella. Palauttaa ratkaistun kopion labyrintista.
- to_file(self, file_name) – Ottaa parametrina tiedoston nimen ilman päätettä. Tallentaa labyrintin kyseisellä nimellä olevaan tekstitiedostoon.
- from_file(self, file_name) – Lukee file_name.txt -tiedostosta labyrintin. Palauttaa 'True':n jos luku onnistui, muutoin 'False':n.

Player-luokan keskeiset metodit:

- move(self, direction, maze) – Liikuttaa, jos mahdollista, pelinappulaa annettuun suuntaan annetussa labyrintissa.
- at_exit(self) – Palauttaa totuusarvon riippuen siitä, onko pelaaja maalissa vai ei.
- update_movable_directions(self, maze) – Muuttaa self.movable-taulukkoa mahdollisten liikkumasuuntien mukaisesti. move()-metodi käyttää tätä metodia.

Käytetty luokkarakenne on omasta mielestäni melko yksinkertainen. Window-luokan olisi mahdollisesti voinut myös jakaa useampaan osaan. Osakokonaisuudet voivat olla melko laajoja, mutta mielestäni ohjelmaa ei kannata jakaa pienempiin osiin. Nykyisessä muodossaan luokkia on mielestäni helpointa käyttää.

Algoritmit

Ohjelman keskeisimmät algoritmit ovat labyrintin luonti- ja ratkaisualgoritmi. Labyrintin luonnissa käytetään mukaelmaa depth-first search -algoritmista. Tämä algoritmi oli ehkä yksi helpoimmista implementoida mutta kuitenkin kohtuullisen tehokas. Nämä ovat pääsyyt miksi olen valinnut kyseisen algoritmin. Alla on selostettu algoritmin toiminta pseudokoodina korkealla abstraktiotasolla.

Depth-first search:

Valitse satunnainen aloitusruutu nykyiseksi ruuduksi ja merkitse se vierailluksi. Alusta uusi pino (olen toteutuksessani käyttänyt kahta pinoa).

while True{

Jos nykyisellä ruudulla on naapureita, joissa ei ole vierailtu, valitse yksi niistä satunnaisesti. Pistä nykyinen ruutu pinoon ja merkitse valittu naapuri nykyiseksi ruuduksi. Merkitse samalla naapuri vierailluksi. Poista seinä näiden kahden ruudun välistä.

Jos kaikki nykyisen ruudun naapurit ovat vierailtuja poista pinosta päällimmäinen ruutu ja aseta se nykyiseksi ruuduksi.

}

Koska omassa labyrintissani seinät ovat myös ruutuja, olen käyttänyt mukaelmaa depth-first searchista. Sen sijaan, että poistaisin kahden ruudun välisen seinän, poistankin seinän joka on nykyisessä ruudussa. Koska labyrintissani on käytävien lisäksi myös käytäviä ylittäviä siltoja, olen itse implementoinut set_bridge()-metodin, joka katsoo jokaisen ruudun kohdalla onko tästä ruudusta mahdollista asettaa silta johonkin suuntaan.

Toisenlainen luontialgoritmi olisi ollut esimerkiksi Kruskalin algoritmi. Kruskalin algoritmissa kaikki seinät ovat alussa paikoillaan, kuten depth-first searchissa. Kruskal kaivertaa käytäviä sieltä täältä satunnaisesti, kuitenkin siten, että lopputulos on nk. täydellinen labyrintti. Tämä eroaa DFS:stä, koska DFS kaivertaa aluksi yhden pitkän käytävän niin pitkälle kuin mahdollista, ja palaa tämän jälkeen siihen edelliseen ruutuun josta on mahdollista jatkaa toiseen suuntaan. Luotu labyrintti tulee olemaan erinäköinen näillä kahdella algoritmilla. Päädyin DFS:ään koska se oli kohtuullisen helppo implementoida ja ymmärtää ja luo oikeanlaisen labyrintin.

Uusia labyrintin luontialgoritmeja olisi kohtuullisen helppoa halutessa lisätä. Tällöin voisi esimerkiksi antaa käyttäjän valita mitä algoritmia käytetään.

Labyrintin ratkaisemisessa käytän left wall followeria. Tämä on ideana helppo algoritmi ja kohtuullisen helppo toteuttaa. Aluksi alustetaan taulukko risteysruuduille. Algoritmissa aloitetaan pelinappulan nykyisestä sijainnista ja edetään siitä johonkin suuntaan. Aina kun tullaan ruutuun, josta on mahdollista edetä useampaan suuntaan, lisätään se taulukkoon. Edettäessä käännytään vasemmalle aina kun sen on mahdollista. Jos ei ole mahdollista kääntyä vasemmalle, jatketaan suoraan eteenpäin. Jos tämäkään ei ole mahdollista, käännytään oikealle. Jos mahdollisia liikkumissuuntia ei ole, poistetaan taulukosta viimeisin ruutu ja jatketaan tästä eteenpäin.

Labyrintin ratkaisussa olisi ollut mahdollista käyttää myös rekursiota. Kokeilin myös rekursiota, mutta en saanut sitä kohtuullisella vaivalla toimimaan, joten päädyin left wall followeriin.

Tietorakenteet

Sekä luonti- että ratkaisualgoritmeissani tarvitaan tietorakennetta, johon voidaan tallentaa alkioita järjestyksessä ja josta on mahdollista poimia viimeisin lisätty alkio. Olen käyttänyt tässä sekä itse

toteuttamaani pinoa että taulukkoa. Luontialgoritmissa jokainen vierailtu ruutu laitetaan pinoon ja poistetaan sieltä tarvittaessa. Tähän tarkoitukseen tarvittiin siis LIFO-rakenne. Esimerkiksi jonoa ei olisi kannattanut käyttää, koska jono on perinteisesti FIFO-rakenne. Jos jonossa olisi ollut osoitin myös alkuun ja olisi poistettu aina viimeisin lisätty alkio, jonoakin olisi voinut käyttää pinon sijaan.

Ohjelmoin itse pinon toiminnan. Pino rakentuu linkitetyn listan päälle. Listassa on osoitin sekä ensimmäiseen että viimeiseen alkioon. Listaan voi lisätä alkioita mihin kohtaan tahansa. Koska käytin listaa pinon tietorakenteena, alkiot lisättiin aina listan alkuun ja poimittiin listan alusta.

Ratkaisualgoritmissa olen käyttänyt muuttuvamittaista taulukkoa, johon alkiot lisätään. Tässäkin olisi ollut mahdollista käyttää pinoa, kuten luontialgoritmissa, mutta halusin kokeilla molempia tietorakenteita.

Tiedostot

Käyttäjällä on mahdollisuus tallentaa labyrintti tekstitiedostoon haluamallaan nimellä. Tämä tehdään ohjelman ollessa käynnissä painamalla ’Save’-nappulaa. Ohjelma kysyy käyttäjältä tiedoston nimeä johon labyrintti tallennetaan. Tekstitiedoston ensimmäisellä rivillä on välilyönnillä erotettuna labyrintin leveys ja korkeus. Tämän jälkeen tulee joukko rivejä, joissa on numeroita välilyönnillä erotettuna. Jokainen numero kuvaa yhtä ruutua. Numeroiden merkitykset ovat seuraavat:

Numero	Selitys
0	Tyhjä ruutu/käytävä
1	Seinä
2	Silta pohjoiseen
3	Silta etelään
4	Silta itään
5	Silta länteen
7	Maali
8	Pelaaja
9	Ratkaisun näyttämisen jälkeen vierailtu ruutu

Labyrintin voi ladata tekstitiedostosta painamalla ’Load’-nappulaa. Tällöin ohjelma kysyy tekstitiedoston nimeä ilman päätettä (.txt). Oman kokemukseni mukaan ohjelma ei kaadu, vaikka ladatussa tekstitiedostossa olisi virheitä, mutta labyrintti ei tietenkään piirry oikealla tavalla.

Esimerkki 30*35 -kokoisesta tallennetusta labyrintista:

30 35
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1 0 1
1 1 0 1 0 1 1 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 0 1
1 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 1 1 0 1 0 1

```
1 0 1 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1
1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 0 1
1 1 1 0 1 0 1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 1 1
1 0 0 0 1 1 0 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 0 1 0 1
1 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1
1 0 0 0 0 1 1 1 0 0 1 0 0 4 0 5 0 1 0 0 1 0 1 1 1 1 1 1 0 1
1 0 1 1 1 0 0 0 0 1 1 0 1 0 0 1 3 0 1 0 1 0 0 0 0 0 0 0 0 1
1 0 0 0 4 0 5 0 1 0 0 0 4 0 5 0 0 0 1 0 0 1 0 1 1 0 1 1 0 1
1 0 1 1 1 0 0 1 0 0 1 1 1 0 0 0 2 0 0 1 0 0 1 1 0 0 1 0 0 1
1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 1
1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1
1 0 1 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 1 1 1 0 1 0 0 0 1
1 0 0 0 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1
1 0 4 0 5 0 0 0 1 1 0 0 0 1 0 1 8 0 0 1 1 0 0 0 1 1 0 1 0 1
1 0 1 0 1 0 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 4 0 5 0 0 0 1 0 1
1 1 0 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 4 0 5 0 1
1 0 0 1 0 1 0 1 0 0 1 3 0 1 0 4 0 5 0 0 0 1 1 1 1 0 0 1 0 1
1 0 1 0 0 1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 1
1 0 0 0 1 0 1 0 0 0 0 2 1 0 0 0 1 0 1 1 1 0 1 1 0 0 1 0 0 1
1 1 0 3 0 0 0 1 1 1 3 0 1 1 0 3 1 0 0 1 0 0 0 0 1 0 0 0 1 1
1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1 0 1
1 0 1 2 3 0 0 1 0 1 2 1 1 0 1 2 0 0 1 0 0 0 0 3 0 0 0 1 0 1
1 0 1 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 1 1 1 0 0 0 1 0 1 0 1
1 0 1 0 2 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 2 1 1 0 0 0 1
1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 0 0 1 0 1 1
1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 0 1
1 0 1 0 1 0 1 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 1 1
1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1
1 0 0 1 0 0 0 1 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 1 1 1 0 1 1
1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Testaus

Ohjelmaa on testattu ajamalla sitä erisuuruisilla labyrinteilla. Labyrintin luontialgoritmi on testeissäni aina luonut nk. täydellisen labyrintin, eli labyrintin, jossa on yksi ainut polku minkä tahansa kahden pisteen välillä. Tämä onkin dept-first search -algoritmin tarkoitus. Pienillä labyrinteilla voi syntyä reunojen viereen pieniä seinärykelmiä, joissa ei ole käytävää, mutta tämä ei itsessään ole virhe.

Olen itse testannut ohjelmaa print_lab-moduulin avulla toteutuksen kaikissa vaiheissa. print_lab-moduulissa olen tehnyt esimerkkitulostuksia Ascii-merkistöllä. Kun olin toteuttanut Window-luokan, testasin kaikki sen toiminnot perusteellisesti.

Ohjelman tunnetut puutteet ja viat

Kun on ohjannut pelinappulan loppuruutuun ja painaa ’Resign’-nappia, ohjelma näyttää kaatuvan tuntemattomasta syystä. Tämä liittyy ilmeisesti ratkaisualgoritmin toimintaan, ja voisi olla helposti korjattavissa katsomalla onko ratkaisun aloitusruutu pelin loppuruutu. Toinen ongelma on nappuloiden sijoitus. Ohjelmaa käynnistäessä napit ovat aina oikeassa paikassa, vaikka ruutua voikin joutua suurentamaan nähdäkseen kaikki nappulat. Kun kuitenkin käyttää ’New Game’-toimintoa ja valitsee suuremman labyrintin kuin aikaisemmin, nappulat voivat joutua labyrintin päälle. Kaikki nappulat ja pikakomennot toimivat kuitenkin oikealla tavalla. Ajan salliessa asian olisi voinut ratkaista ehkä liittämällä nappulat QToolBaariin, joka aina olisi ikkunan oikeassa reunassa labyrintin ulkopuolella.

Kun labyrintti tallennetaan tiedostoon ja sama labyrintti ladataan tiedostosta, pelaaja aloittaa uudestaan labyrintin keskeltä. Pelinappulan sijainti näyttää kuitenkin tallentuvan tiedostoon ja näin ollen näkyy myös seuraavan kerran sinisenä ruutuna kun peliä ladataan. Tämä ei oikeastaan ole virhe, vaan tästä pelaaja myös näkee mihin kohtaan hän edellisellä kerralla jäi. Tämän voisi kuitenkin halutessaan muuttaa siten, että pelaaja voi jatkaa labyrintin ratkaisemista samasta paikasta jossa hän oli peliä tallentaessa.

Jos ’New Game’ -toiminnolle annetaan pienempiä lukuja kuin 5*5, ohjelma ei luultavasti osaa tulostaa labyrinttia ja joko kaatuu tai tulostaa jotain roskaa.

3 parasta ja 3 heikointa kohtaa

Parasta ohjelmassani on omasta mielestäni melko selkeä ulkoasu, luontialgoritmin tehokkuus ja toimivuus sekä pelillinen toimivuus. En ole kertaakaan onnistunut löytämään tilannetta, jossa ohjelma olisi tulostanut vääränlaisen labyrintin. Jos ladattavaa tiedostoa on muokattu, labyrintti ei luonnollisesti tulostu oikealla tavalla. Luontialgoritmi on kohtuullisen tehokas myös suurille tiedostoille, tosin pelattavuus ei ole paras mahdollinen koska ikkunan sisältöä ei voi liikuttaa. Hyvin suurilla labyrinteilla peli on hieman hidas. Myös pelinappulan toiminta on testeissäni aina toiminut oikealla tavalla, eikä ole mahdollista kävellä seinien läpi tai muuta laitonta.

Heikoin kohta ohjelmassani on luultavasti ratkaisualgoritmi. Algoritmi ei kaikissa tapauksissa näytä lyhintä ratkaisua, vaan voi kiertää labyrinttia turhan paljon. Tämän voisi ratkaista lisäämällä ratkaisualgoritmiin osion, jossa katsotaan ollaanko tultu umpikujaan. Jos ollaan umpikujassa, pitäisi muuttaa kaikki ruudut takaisin käytäviksi edelliseen risteykseen asti. Tällöin ainoastaan lyhyin polku piirtyisi ratkaisuksi. Olisin tehnyt tämän lisäyksen ajan salliessa.

Toinen ohjelmani heikko kohta on nappien sijoitus. Napit voivat joskus joutua labyrintin päälle. Kuten edellisessä Ohjelman puutteet ja viat -kohdassa kerrotaan, tämän olisi voinut korjata esimerkiksi QToolBarin avulla.

Poikkeamat suunnitelmasta

En ole pitänyt kirjaa työajasta. Tiedon etsimiseen ja algoritmien toteutukseen kului yllättävän paljon aikaa. Ikkunan toteutukseen PyQt:n avulla kuitenkin sujui jopa odotettua nopeammin, vaikka en ennen ollut tutustunut PyQt:een. Toteutin eri luokat ja funktiot suunnittelemassani järjestyksessä. Aikaa arvioisin käyttäneeni ainakin yli 50 tuntia, ehkä yli 60 tuntia kokonaisuudessaan.

Toteutunut työjärjestys ja aikataulu

Projektin alussa toteutin labyrintin luontialgoritmin. Tämä on luonnollisesti yksi keskeisimmistä ohjelman asioista ja toteutuksen jälkeen oli mahdollista heti nähdä työn tulos. Tämän jälkeen toteutin funktiot tiedostoon tallentamiseen ja tiedostosta lataamiseen. Pinon käyttämät Stack- ja LinkedList-luokat olin tehnyt jo aiemmin, mutta muokkasin niitä projektin edetessä. Player-luokka syntyi myös pikkuhiljaa projektin edetessä.

Aikomuksenani oli toteuttaa ratkaisualgoritmi rekursion avulla, mutta lopputuloksesta ei tullut aivan toimiva. Päädyin toteuttamaan ratkaisualgoritmin left wall followerina. Lopussa määräaika alkoi lähestyä ja aika kävi melko vähiin. Toteutin lopussa graafisen käyttöliittymän. Jos aikaa olisi ollut enemmän, olisi käyttöliittymää voinut hioa hienommaksi ja käyttäjäystävällisemmäksi.

Arvio lopputuloksesta

Mielestäni ohjelmakoodi on melko helppolukuinen ja looginen. Poikkeuksena tähän on ehkä Labyrinth-luokan generate() ja set_bridge() -luokkien pitkät if-lauseet. Nämä on mahdollista helposti korvata sisäkkäisillä if-lauseilla, mutta on makuasia kumpi on parempi ratkaisu. Ohjelmaa voisi laajentaa melko helposti lisäämällä uusia labyrintin luontialgoritmeja. Käyttäjä voisi valita näiden algoritmien välillä saadakseen erityyppisiä labyrintteja.

Ohjelmaa voisi parantaa usealla saralla. Suurilla labyrinteilla olisi mahdollista nähdä vain pelinappulan ympärillä oleva osuus labyrintista, tai vaihtoehtoisesti labyrinttia voisi selata ylöspäin ja alaspäin vierityspalkeilla. Myös näppäimet voisi sijoittaa paremmin ’New Gamen’ yhteydessä. Ratkaisualgoritmissa

olisi myös parantamisen varaa aiemmin mainitulla tavalla. Voisi myös miettiä seinien ja siltojen piirtämistä tyylikkäämmällä tavalla. Esimerkiksi sillat voisi piirtää kahtena rinnakkaisena viivana sillan aloitusruudusta sillan loppuruutuun. Ehto labyrintin pienimmälle sallitulle koolle olisi yksi mahdollisesti implementoitava asia.

Yleisesti algoritmien luonti sujui kohtuullisen mutkattomasti. Myös käyttöliittymän luonti sujui sutjakasti ja odotettua helpommin. Lopputulos on mielestäni hyvä.

Viitteet

http://www.cse.hut.fi/fi/opinnot/t1061215/2013_external/projekti/aiheet2013/101ed.html

http://weblog.jamisbuck.org/2011/3/4/maze-generation-weave-mazes

http://weblog.jamisbuck.org/2011/3/17/maze-generation-more-weave-mazes#article_body

http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=python_10

http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=python_11

http://zetcode.com/tutorials/pyqt4/

http://www.astrolog.org/labyrnth/algrithm.htm

http://www.mazeworks.com/mazegen/mazetut/index.htm

http://en.wikipedia.org/wiki/Maze_generation_algorithm

http://code.activestate.com/recipes/252127-maze-generator/

http://www.daniweb.com/software-development/python/threads/238326/python-maze#

http://weblog.jamisbuck.org/2011/2/7/maze-generation-algorithm-recap

http://www.cs.bu.edu/teaching/alg/maze/

http://ogun.stanford.edu/~bnayfeh/mazealgo.html

http://www.commandprompt.com/community/pyqt/c7391

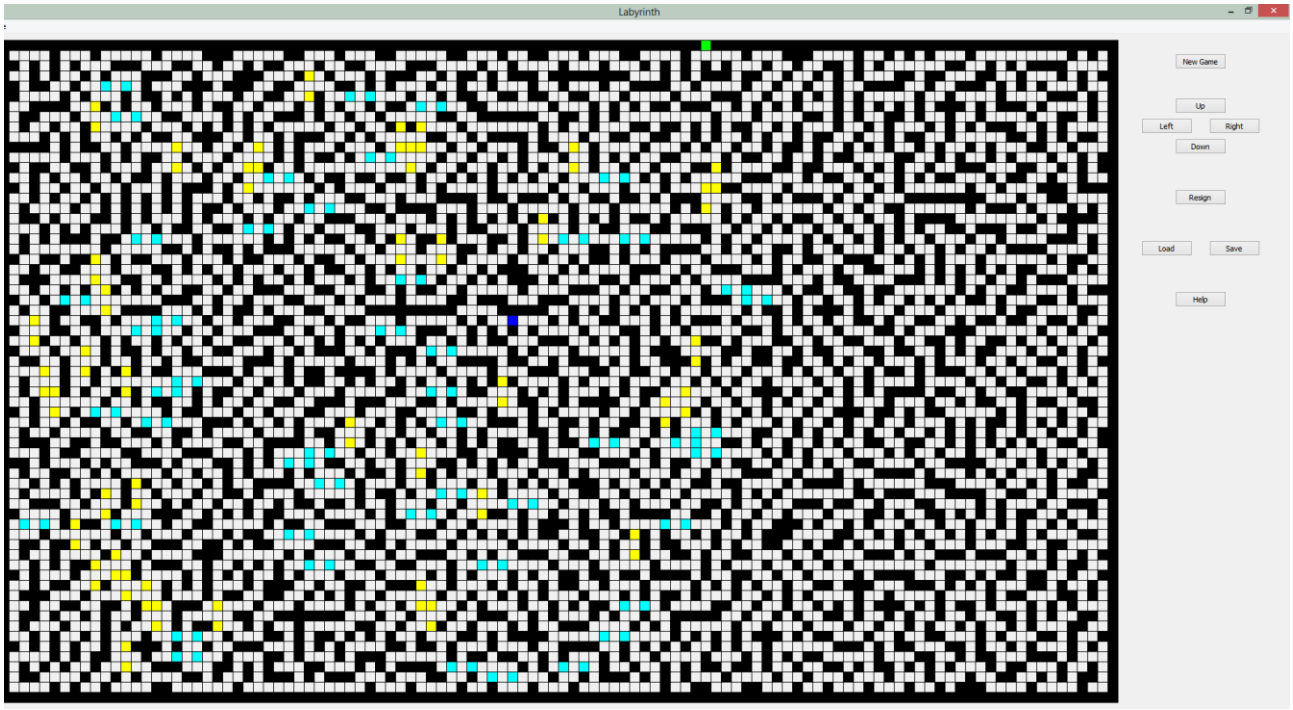
http://www.rkblog.rk.edu.pl/w/p/qgraphicsview-and-qgraphicsscene/

http://qt-project.org/doc/qt-4.8/qcolor.html

http://www.textfiles.com/programming/maze-faq

Liitteet

Alla on esimerkki 110*65 -kokoisesta labyrintista.



Alla on moduulikohtainen ohjelmakoodi.

mainwindow:

```
import sys
from PyQt4 import QtGui, QtCore
from player import Player
from labyrinth import Labyrinth


class Window(QtGui.QMainWindow, QtGui.QWidget):

    def __init__(self, x_size, y_size):
        self.ended = 0
        self.changed = 1
        self.x_size = x_size*15+250
        self.y_size = y_size*15+75
        self.qp = QtGui.QPainter()
        self.lab = Labyrinth([x_size, y_size])

        if self.lab.matrix[self.lab.x_size/2][self.lab.y_size/2]:
            x = self.lab.x_size/2+1
            y = self.lab.y_size/2
            self.player = Player([x, y])
            self.lab.matrix[self.lab.x_size/2+1][self.lab.y_size/2] = 8
```

```
        else:
            x = self.lab.x_size/2
            y = self.lab.y_size/2
            self.player = Player([x, y])
            self.lab.matrix[self.lab.x_size/2][self.lab.y_size/2] = 8

        super(Window, self).__init__()
        self.initUI()


    def initUI(self):
        exitAction = QtGui.QAction('Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.triggered.connect(QtGui.qApp.quit)
        self.statusBar()
        menubar = self.menuBar()
        fileMenu = menubar.addMenu('&File')
        fileMenu.addAction(exitAction)

        btn1 = QtGui.QPushButton('New Game', self)
        btn1.setShortcut('Ctrl+N')
        btn1.setToolTip('Generates a new maze.')
        btn1.resize(btn1.sizeHint())
        btn1.move(self.x_size-150, 50)
        btn1.clicked.connect(self.button1Clicked)

        btn2 = QtGui.QPushButton('Up', self)
        btn2.setShortcut('Up')
        btn2.setToolTip('<b>Up</b>')
        btn2.resize(btn2.sizeHint())
        btn2.move(self.x_size-150, 115)
        btn2.clicked.connect(self.button2Clicked)

        btn3 = QtGui.QPushButton('Down', self)
        btn3.setShortcut('Down')
        btn3.setToolTip('<b>Down</b>')
        btn3.resize(btn3.sizeHint())
        btn3.move(self.x_size-150, 175)
        btn3.clicked.connect(self.button3Clicked)

        btn4 = QtGui.QPushButton('Right', self)
        btn4.setShortcut('Right')
        btn4.setToolTip('<b>Right</b>')
        btn4.resize(btn4.sizeHint())
        btn4.move(self.x_size-100, 145)
        btn4.clicked.connect(self.button4Clicked)

        btn5 = QtGui.QPushButton('Left', self)
        btn5.setShortcut('Left')
        btn5.setToolTip('<b>Left</b>')
        btn5.resize(btn5.sizeHint())
        btn5.move(self.x_size-200, 145)
        btn5.clicked.connect(self.button5Clicked)

        btn6 = QtGui.QPushButton('Resign', self)
        btn6.setShortcut('Ctrl+R')
        btn6.setToolTip('Ends the game and shows the solution.')
        btn6.resize(btn6.sizeHint())
        btn6.move(self.x_size-150, 250)
        btn6.clicked.connect(self.button6Clicked)
```

```
        btn7 = QtGui.QPushButton('Load', self)
        btn7.setShortcut('Ctrl+L')
        btn7.setToolTip('<b>Loads</b> the game from a file.')
        btn7.resize(btn7.sizeHint())
        btn7.move(self.x_size-200, 325)
        btn7.clicked.connect(self.button7Clicked)

        btn8 = QtGui.QPushButton('Save', self)
        btn8.setShortcut('Ctrl+S')
        btn8.setToolTip('<b>Saves</b> the game to a file.')
        btn8.resize(btn8.sizeHint())
        btn8.move(self.x_size-100, 325)
        btn8.clicked.connect(self.button8Clicked)

        btn9 = QtGui.QPushButton('Help', self)
        btn9.setShortcut('H')
        btn9.setToolTip('Shows the instructions for the game.')
        btn9.resize(btn9.sizeHint())
        btn9.move(self.x_size-150, 400)
        btn9.clicked.connect(self.button9Clicked)

        self.resize(self.x_size, self.y_size)
        #self.center()   # Centers the window on the screen
        #self.setGeometry(200, 200, self.x_size, self.y_size)
        self.setWindowTitle('Labyrinth')

        self.show()


    def paintEvent(self, e):
        '''
        Draws a new maze in the window.
        '''
        if self.ended==0:
            self.qp.begin(self)
            j = 0
            height = 15
            while j<self.lab.y_size:
                width = 15
                height+=15
                for k in self.lab.matrix:
                    if k[j]==1:
                        self.draw_rectangle(width, height, 1)
                    elif k[j]==2:
                        self.draw_rectangle(width, height, 2)
                    elif k[j]==3:
                        self.draw_rectangle(width, height, 3)
                    elif k[j]==4:
                        self.draw_rectangle(width, height, 4)
                    elif k[j]==5:
                        self.draw_rectangle(width, height, 5)
                    elif k[j]==7:
                        self.draw_rectangle(width, height, 7)
                    elif k[j]==8:
                        self.draw_rectangle(width, height, 8)
                    else:
                        self.draw_rectangle(width, height, 10)

                    width+=15
                j+=1
            self.qp.end()
```

```
        else:
            self.qp.begin(self)
            j = 0
            height = 15
            while j<((self.y_size-75)/15):
                width = 15
                height+=15
                for k in self.new_lab:
                    if k[j]==1:
                        self.draw_rectangle(width, height, 1)
                    elif k[j]==2:
                        self.draw_rectangle(width, height, 2)
                    elif k[j]==3:
                        self.draw_rectangle(width, height, 3)
                    elif k[j]==4:
                        self.draw_rectangle(width, height, 4)
                    elif k[j]==5:
                        self.draw_rectangle(width, height, 5)
                    elif k[j]==7:
                        self.draw_rectangle(width, height, 7)
                    elif k[j]==8:
                        self.draw_rectangle(width, height, 8)
                    elif k[j]==9:
                        self.draw_rectangle(width, height, 9)
                    else:
                        self.draw_rectangle(width, height, 10)

                    width+=15
                j+=1
            self.qp.end()


    def draw_rectangle(self, x, y, param):
        if param==1:
            self.qp.setBrush(QtGui.QColor(QtCore.Qt.black))
            self.qp.drawRect(x, y, 15, 15)
        elif param==2 or param==3:
            self.qp.setBrush(QtGui.QColor(QtCore.Qt.yellow))
            self.qp.drawRect(x, y, 15, 15)
        elif param==4 or param==5:
            self.qp.setBrush(QtGui.QColor(QtCore.Qt.cyan))
            self.qp.drawRect(x, y, 15, 15)
        elif param==7:
            self.qp.setBrush(QtGui.QColor(QtCore.Qt.green))
            self.qp.drawRect(x, y, 15, 15)
        elif param==8:
            self.qp.setBrush(QtGui.QColor(QtCore.Qt.blue))
            self.qp.drawRect(x, y, 15, 15)
        elif param==9:
            self.qp.setBrush(QtGui.QColor(QtCore.Qt.green))
            self.qp.drawRect(x, y, 15, 15)
        else:
            self.qp.setBrush(QtGui.QColor(0, 0, 0, 0))
            self.qp.drawRect(x, y, 15, 15)



    def button1Clicked(self):
        '''
        Calls a new generation of a maze.
        '''
```

```
        x_value, ok = QtGui.QInputDialog.getText(self, 'New game',
            "Enter the width of the new maze:")

        if ok:
            y_value, ok = QtGui.QInputDialog.getText(self, 'New game',
            "Enter the height of the new maze:")
            try:
                x_size = int(x_value)
                y_size = int(y_value)
                self.statusBar().showMessage("")
                self.lab = Labyrinth([x_size, y_size])
                if self.lab.matrix[self.lab.x_size/2][self.lab.y_size/2]:
                    x = self.lab.x_size/2+1
                    y = self.lab.y_size/2
                    self.player = Player([x, y])
                    self.lab.matrix[self.lab.x_size/2+1][self.lab.y_size/2] = 8
                else:
                    x = self.lab.x_size/2
                    y = self.lab.y_size/2
                    self.player = Player([x, y])
                    self.lab.matrix[self.lab.x_size/2][self.lab.y_size/2] = 8
                self.x_size = self.lab.x_size*15+250
                self.y_size = self.lab.y_size*15+75
                self.ended = 0
                self.resize((x_size*15+250), (y_size*15+75))
                self.update()
            except ValueError:
                self.statusBar().showMessage("One of the values was incorrect.")


    def button2Clicked(self):
        '''
        Moves up.
        '''
        if not self.player.at_exit() and not self.ended:
            self.changed = 1
            self.player.move(0, self.lab.matrix)
            self.update()
            if self.player.at_exit():
                self.statusBar().showMessage("Game ended.")
        else:
            self.statusBar().showMessage("Game ended.")


    def button3Clicked(self):
        '''
        Moves down.
        '''
        if not self.player.at_exit() and not self.ended:
            self.changed = 1
            self.player.move(2, self.lab.matrix)
            self.update()
        else:
            self.statusBar().showMessage("Game ended.")


    def button4Clicked(self):
        '''
        Moves right.
        '''
```

```
        if not self.player.at_exit() and not self.ended:
            self.changed = 1
            self.player.move(1, self.lab.matrix)
            self.update()
        else:
            self.statusBar().showMessage("Game ended.")


    def button5Clicked(self):
        '''
        Moves left.
        '''
        if not self.player.at_exit() and not self.ended:
            self.changed = 1
            self.player.move(3, self.lab.matrix)
            self.update()
        else:
            self.statusBar().showMessage("Game ended.")


    def button6Clicked(self):
        '''
        Ends the game.
        '''
        if not self.ended:
            self.ended = 1
            self.new_lab = self.lab.solve(self.player.get_position())
            self.update()
            self.statusBar().showMessage("Game ended.")


    def button7Clicked(self):
        '''
        Loads a new game.
        '''
        file_name, ok = QtGui.QInputDialog.getText(self, 'Load maze',
            "Enter the name of the file to be read without the filename
extension (.txt):\ne.g. game1")
        if ok:
            if self.lab.from_file(file_name):
                self.statusBar().showMessage("Load successful.")
                if self.lab.matrix[self.lab.x_size/2][self.lab.y_size/2]:
                    x = self.lab.x_size/2+1
                    y = self.lab.y_size/2
                    self.player = Player([x, y])
                    self.lab.matrix[self.lab.x_size/2+1][self.lab.y_size/2] = 8
                else:
                    x = self.lab.x_size/2
                    y = self.lab.y_size/2
                    self.player = Player([x, y])
                    self.lab.matrix[self.lab.x_size/2][self.lab.y_size/2] = 8
                #self.resize((self.lab.x_size*15+250), (self.lab.y_size*15+75))
                self.x_size = self.lab.x_size*15+250
                self.y_size = self.lab.y_size*15+75
                self.ended = 0
                self.update()

            else:
                self.statusBar().showMessage("Load failed.")
```

```
    def button8Clicked(self):
        '''
        Saves the maze.
        '''
        if not self.ended:
            file_name, ok = QtGui.QInputDialog.getText(self, 'Save',
                "Enter the name of the file to save to without the filename
extension (.txt):\ne.g. game1")
            if ok:
                self.changed = 0
                self.lab.to_file(file_name)
                self.statusBar().showMessage("Saved the maze to file " +
file_name + ".txt")
        else:   # Game ended
            self.statusBar().showMessage("Unable to save. The game has ended.")


    def button9Clicked(self):
        '''
        Shows the help window with instructions.
        '''
        QtGui.QMessageBox.question(self, 'Message',
                "Your objective is to steer the blue square (player) to
the\ngreen square (end). \
Empty squares are passages and\nfilled squares are walls.\nYellow squares are
north-southbridges and \
cyan\nsquares are north-south bridges. The resign button\nshows a path to the
end from the current square.", QtGui.QMessageBox.Ok, QtGui.QMessageBox.Ok)


    def center(self):
        '''
        Function for centering the window on the screen.
        '''
        qr = self.frameGeometry()
        cp = QtGui.QDesktopWidget().availableGeometry().center()
        qr.moveCenter(cp)
        self.move(qr.topLeft())


    def closeEvent(self, event):
        '''
        Changes the default close event.
        '''
        if self.changed:
            reply = QtGui.QMessageBox.question(self, 'Message',
                "Are you sure to quit without saving?", QtGui.QMessageBox.Yes |
                QtGui.QMessageBox.No, QtGui.QMessageBox.No)

            if reply == QtGui.QMessageBox.Yes:
                event.accept()
            else:
                event.ignore()


def main():

    app = QtGui.QApplication(sys.argv)
    window = Window(30, 35)
    sys.exit(app.exec_())
```

```
if __name__ == '__main__':
    main()
```

labyrinth:

```
import random
from stack import Stack


class Labyrinth(object):

    def __init__(self, size):
        self.size = [size[0], size[1]]
        # self.size = [x-value, y-value]
        self.x_size = self.size[0]
        self.y_size = self.size[1]
        #random.seed(13)
        self.x = random.randint(1, self.x_size-2)    # x-coordinate for the
current square. Does not choose perimeter squares
        self.y = random.randint(1, self.y_size-2)    # y-coordinate for the
current square. Does not choose perimeter squares
        self.bridge_count = 0

        self.total_cells = self.x_size*self.y_size
        self.matrix = list(list(1 for i in range(self.y_size)) for j in
range(self.x_size))

        '''
        matrix = columns*[len(column)*1]
        The matrix is a list containing the columns
        The columns are lists containing len(column) number of 1's
        To manipulate (2,3) in matrix: matrix[2][3]
        7 means exit, 0 means passage, 1 wall, 2 bridge north, 3 south, 4 bridge
east, 5 bridge west and 8 means player
        '''
        self.visited = list(list(0 for i in range(self.y_size)) for j in
range(self.x_size))
        self.matrix = self.generate()
        #self.matrix_copy = self.matrix


    def generate(self):
        '''
        Depth-first search

        1. Pick a random starting location as the "current cell" and mark it
           as "visited".  Also, initialize a stack of cells to empty (the stack
           will be used for backtracking).  Initialize VISITED (the number of
           visited cells) to 1.
        2. While VISITED < the total number of cells, do the following:
             If the current cell has any neighbors which haven't yet been
visited,
```

```
                pick one at random.  Push the current cell on the stack and set
the
                current cell to be the new cell, marking the new cell as visited.
                Knock out the wall between the two cells.  Increment VISITED.
              If all of the current cell's neighbors have already been visited,
then
                backtrack.  Pop the previous cell off the stack and make it the
current
                cell.
        '''

        i = 0
        while i<len(self.visited[0]):
            self.visited[0][i] = 1   # First column
            self.visited[self.x_size-1][i] = 1   # Last column
            i+=1
        i = 0
        while i<len(self.visited):
            self.visited[i][0] = 1   # First row
            self.visited[i][self.y_size-1] = 1   # Last row
            i+=1

        stack = Stack()
        x_stack = Stack()   # Contains x-values for squares in stack
        y_stack = Stack()   # Contains y-values for squares in stack
        total_bridges = self.total_cells/80  # total_bridges = (maximum amount
of bridges in the maze - 1)
        current = self.matrix[self.x][self.y]
        visited_count = 1
        self.visited[self.x][self.y] = 1

        while 1:
            choices = []
            while True:
                # Checks that square not out of bounds, not visited and no
passage next to the square
                if (self.y-1)>=1 and not self.visited[self.x][self.y-1] and
self.matrix[self.x][self.y-2] and self.matrix[self.x-1][self.y-1] and
self.matrix[self.x+1][self.y-1]:
                    choices.append(1)   # North
                if (self.x+1)<=(self.x_size-1) and not
self.visited[self.x+1][self.y] and self.matrix[self.x+1][self.y-1] and
self.matrix[self.x+1][self.y+1] and self.matrix[self.x+2][self.y]:
                    choices.append(2)   # East
                if (self.y+1)<=(self.y_size-1) and not
self.visited[self.x][self.y+1] and self.matrix[self.x][self.y+2] and
self.matrix[self.x-1][self.y+1] and self.matrix[self.x+1][self.y+1]:
                    choices.append(3)   # South
                if (self.x-1)>=1 and not self.visited[self.x-1][self.y] and
self.matrix[self.x-2][self.y] and self.matrix[self.x-1][self.y-1] and
self.matrix[self.x-1][self.y+1]:
                    choices.append(4)   # West
                if len(choices)>0:
                    choice = random.choice(choices)
                if len(choices)==0:
                    break
                if choice==1:   # North
                    stack.push(current)
                    x_stack.push(self.x)
                    y_stack.push(self.y)
                    current = self.matrix[self.x][self.y-1]
```

```
                    self.visited[self.x][self.y-1] = 1 # Marks the cell as
visited
                    self.matrix[self.x][self.y-1] = 0 # Removes the wall in the
square
                    self.y = self.y-1 # Updates the y-value
                    visited_count+=1

                elif choice==2: # East
                    stack.push(current)
                    x_stack.push(self.x)
                    y_stack.push(self.y)
                    current = self.matrix[self.x+1][self.y]
                    self.visited[self.x+1][self.y] = 1 # Marks the cell as
visited
                    self.matrix[self.x+1][self.y] = 0 # Removes the wall in the
square
                    self.x = self.x+1 # Updates the x-value
                    visited_count+=1

                elif choice==3: # South
                    stack.push(current)
                    x_stack.push(self.x)
                    y_stack.push(self.y)
                    current = self.matrix[self.x][self.y+1]
                    self.visited[self.x][self.y+1] = 1 # Marks the cell as
visited
                    self.matrix[self.x][self.y+1] = 0 # Removes the wall in the
square
                    self.y = self.y+1 # Updates the y-value
                    visited_count+=1

                elif choice==4: # West
                    stack.push(current)
                    x_stack.push(self.x)
                    y_stack.push(self.y)
                    current = self.matrix[self.x-1][self.y]
                    self.visited[self.x-1][self.y] = 1 # Marks the cell as
visited
                    self.matrix[self.x-1][self.y] = 0 # Removes the wall in the
square
                    self.x = self.x-1 # Updates the x-value
                    visited_count+=1
                choices = []
                if self.bridge_count<=total_bridges:
                    self.set_bridge()   # Checks for, and sets possible bridges

            current = stack.pop()
            if current:
                self.x = x_stack.pop()
                self.y = y_stack.pop()

            else:
                break


        # Adds the exit to the labyrinth
        exit_square = random.randint(1, len(self.matrix)-2)
        while self.matrix[exit_square][1]:  # Loops while not passage south of
exit_square
            exit_square = random.randint(1, len(self.matrix)-2)
        self.matrix[exit_square][0] = 7
```

```
        return self.matrix


    def set_bridge(self):

        bridges = [2, 3, 4, 5]  # List containing numbers representing bridges

        # North-south bridges
        # Check that not out of bounds and no passage on or next to this square
        #if (self.y-4)>=1 and self.matrix[self.x][self.y] not in bridges and
self.matrix[self.x-1][self.y] not in bridges and self.matrix[self.x+1][self.y]
not in bridges and self.matrix[self.x][self.y+1] not in bridges:
        if (self.y-4)>=1 and self.matrix[self.x][self.y-1] not in bridges and
self.matrix[self.x-1][self.y-1] not in bridges and self.matrix[self.x+1][self.y-
1] not in bridges and self.matrix[self.x][self.y] not in bridges:
            # Wall north, horizontal passage, wall and empty square
            if self.matrix[self.x][self.y-4]==0:    # Empty square
                if self.matrix[self.x][self.y-1]==1 and
self.matrix[self.x][self.y-3]==1:   # Walls
                    if self.matrix[self.x][self.y-2]==0 and (self.matrix[self.x-
1][self.y-2]==0 or self.matrix[self.x+1][self.y-2]==0): # Crosses over
horizontal passage
                        self.matrix[self.x][self.y-1] = 2     # Sets passage
north
                        self.matrix[self.x][self.y-3] = 3   # Sets passage south
                        self.visited[self.x][self.y-1] = 1  # Sets wall as
visited
                        self.visited[self.x][self.y-3] = 1  # Sets wall as
visited
                        self.bridge_count+=1

        # East-west bridges
        # Check that not out of bounds and no passage on or next to this square
        #if (self.x+4)<=(len(self.matrix)-2) and self.matrix[self.x][self.y] not
in bridges and self.matrix[self.x][self.y-1] not in bridges and
self.matrix[self.x][self.y+1] not in bridges and self.matrix[self.x-1][self.y]
not in bridges:
        if (self.x+4)<=(len(self.matrix)-2) and self.matrix[self.x+1][self.y]
not in bridges and self.matrix[self.x+1][self.y-1] not in bridges and
self.matrix[self.x+1][self.y+1] not in bridges and self.matrix[self.x][self.y]
not in bridges:
            if self.matrix[self.x+4][self.y]==0:    # Empty square
                if self.matrix[self.x+1][self.y]==1 and
self.matrix[self.x+3][self.y]==1:   # Walls
                    if self.matrix[self.x+2][self.y]==0 and
(self.matrix[self.x+2][self.y-1]==0 or self.matrix[self.x+2][self.y+1]==0): #
Crosses over vertical passage
                        self.matrix[self.x+1][self.y] = 4     # Sets passage
east
                        self.matrix[self.x+3][self.y] = 5   # Sets passage west
                        self.visited[self.x+1][self.y] = 1  # Sets wall as
visited
                        self.visited[self.x+3][self.y] = 1  # Sets wall as
visited
                        self.bridge_count+=1



    def solve(self, coordinates):
        '''
```

```
        Solves the maze, beginning from the coordinates the player is located
at.
        'coordinates' is a list of length 2 with the current x-coordinate
        at index 0 and the current y-coordinate at index 1.
        Solves the maze by following the wall on the left hand side.
        Always turns left if it is possible. Second choice is going straight.
        Third choice turning right. Not possible to go back, but will pop an
element
        from the list if at a dead-end.
        '''

        matrix_copy = self.matrix
        x_intersect = [] # Stack containing the x-coordinates
        y_intersect = [] # Stack containing the y-coordinates
        forbidden_list = [1, 2, 3, 4, 5, 9]
        directions = []
        moving_counter = 0  # Counter used in retracking
        moving_dir = 0   # Current moving direction
        #TODO: Change the initial moving_dir
        x = coordinates[0]
        y = coordinates[1]
        while matrix_copy[x][y]!=7: # While not at the end. This loop appends
the possible moving directions to the list 'directions'.
            if matrix_copy[x][y-1] not in forbidden_list:
                directions.append(0)    # North
            if matrix_copy[x+1][y] not in forbidden_list:
                directions.append(1)    # East
            if matrix_copy[x][y+1] not in forbidden_list:
                directions.append(2)    # South
            if matrix_copy[x-1][y] not in forbidden_list:
                directions.append(3)    # West
            matrix_copy[x][y] = 9
            if len(directions)>1:   # Intersection
                x_intersect.append(x)
                y_intersect.append(y)
                '''
                moving_dir = moving_dir%3-1
                while moving_dir not in directions:
                    moving_dir = moving_dir%3+1
                '''

                if moving_dir==0: # Previously moved north
                    if matrix_copy[x][y-1]==7:
                        moving_dir = 0
                    elif 3 in directions:
                        moving_dir = 3
                    elif 0 in directions:
                        moving_dir = 0
                    elif 1 in directions:
                        moving_dir = 1
                elif moving_dir==1: # Previously moved east
                    if 0 in directions:
                        moving_dir = 0
                    elif 1 in directions:
                        moving_dir = 1
                    elif 2 in directions:
                        moving_dir = 2
                elif moving_dir==2: # Previously moved south
                    if 1 in directions:
                        moving_dir = 1
                    elif 2 in directions:
```

```
                    moving_dir = 2
                elif 3 in directions:
                    moving_dir = 3
            elif moving_dir==3: # Previously moved west
                if matrix_copy[x][y-1]==7:
                    moving_dir = 0
                elif 2 in directions:
                    moving_dir = 2
                elif 3 in directions:
                    moving_dir = 3
                elif 0 in directions:
                    moving_dir = 0
            # Updates the x- or y-value
            if moving_dir%2==0:
                if moving_dir==0:
                    y-=1
                else:
                    y+=1
            else:
                if moving_dir==1:
                    x+=1
                else:
                    x-=1
        elif len(directions)==0 and len(x_intersect):
            #TODO: Add backtracking to remove the marked cells from a dead
end
            #matrix_copy[x][y] = 0
            x = x_intersect.pop()
            y = y_intersect.pop()
        elif len(directions)==1:   # One possible direction to move in
            # Updates the x- or y-value
            moving_dir = directions[0]
            if moving_dir==0:
                y = y-1
            elif moving_dir==1:
                x = x+1
            elif moving_dir==2:
                y = y+1
            elif moving_dir==3:
                x = x-1
        directions = []
    return matrix_copy
```

```
    def to_file(self, file_name):
        '''
        file_name = the file's name without the ending (.txt).
        If the file doesen't exist, a new file with the name 'file_name' is
created.
        '''
        file = open(file_name+'.txt', 'w+')
        word = str(self.x_size) + ' ' + str(self.y_size) + '\n'
        file.write(word)
        i = 0
        word = ''
        while i<self.y_size:
            for j in self.matrix:
                word+=str(j[i])
```

```
                word+=' '
            if self.y_size-i!=1:    # If not the last line
                word+='\n'
            file.write(word)
            word = ''
            i+=1
        file.close()


    def from_file(self, file_name):
        '''
        file_name = the file's name without the ending (.txt).
        Returns True at a successful load and False if something went wrong.
        '''

        try:
            file = open(file_name+'.txt', 'r')
            line = file.readline()
            line = line.rstrip()
            line = line.split(' ')
            self.x_size = int(line[0])
            self.y_size = int(line[1])
            self.matrix = list(list(0 for i in range(self.y_size)) for j in
range(self.x_size))

            j = 0
            z = 1
            while z<=self.y_size:
                line = file.readline()
                line = line.rstrip()
                line = line.split(' ')
                k = 0
                while k<self.x_size:
                    self.matrix[k][j] = int(line[k])
                    k+=1
                j+=1
                z+=1
            file.close()

            return True

        except:
            return False
```

player:

```
class Player(object):

    def __init__(self, start_position):

        self.position = [start_position[0], start_position[1]]   # position =
[x-value, y-value]
        self.x = self.position[0]
        self.y = self.position[1]
        self.earlier = 0
```

```
        self.movable = [False, False, False, False] # North, east, south, west
        self.directions = ['north', 'east', 'south', 'west']


    def move(self, direction, maze):
        '''
        direction: 0 north, 1 east, 2 south, 3 west
        '''
        self.update_movable_directions(maze)
        if self.movable[direction]:   # can move
            if direction==0:
                if maze[self.x][self.y-1]==2:
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x][self.y-4]
                    self.position[1]-=4
                    self.y-=4
                    maze[self.x][self.y] = 8
                else:   # maze[self.x][self.y-1]==0
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x][self.y-1]
                    self.position[1]-=1
                    self.y-=1
                    maze[self.x][self.y] = 8
            elif direction==1:
                if maze[self.x+1][self.y]==4:
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x+4][self.y]
                    self.position[0]+=4
                    self.x+=4
                    maze[self.x][self.y] = 8
                else:
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x+1][self.y]
                    self.position[0]+=1
                    self.x+=1
                    maze[self.x][self.y] = 8
            elif direction==2:
                if maze[self.x][self.y+1]==3:
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x][self.y+4]
                    self.position[1]+=4
                    self.y+=4
                    maze[self.x][self.y] = 8
                else:
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x][self.y+1]
                    self.position[1]+=1
                    self.y+=1
                    maze[self.x][self.y] = 8
            elif direction==3:
                if maze[self.x-1][self.y]==5:
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x-4][self.y]
                    self.position[0]-=4
                    self.x-=4
                    maze[self.x][self.y] = 8
                else:
                    maze[self.x][self.y] = self.earlier
                    self.earlier = maze[self.x-1][self.y]
                    self.position[0]-=1
```

```
                    self.x-=1
                    maze[self.x][self.y] = 8
            return maze
        else:
            return False

    def get_position(self):
        return self.position

    def at_exit(self):
        if self.position[1]==0:
            return True
        else:
            return False

    def get_movable_directions(self):
        directions = []
        for i in self.movable:
            if i==True:
                directions.append(self.directions[i])
            i+=1
        return directions


    def update_movable_directions(self, maze):
        '''
        The players' position cannot be on a perimeter tile, unless the player
has completed the maze.
        '''
        x = self.position[0]
        y = self.position[1]
        north = maze[x][y-1]
        if north==1 or north ==3 or north==4 or north==5:    # North: x, y-1
            self.movable[0] = False
        else:
            self.movable[0] = True
        east = maze[x+1][y]
        if east==1 or east==2 or east==3 or east==5:    # East: x+1, y
            self.movable[1] = False
        else:
            self.movable[1] = True
        south = maze[x][y+1]
        if south==1 or south==2 or south==4 or south==5:    # South: x, y+1
            self.movable[2] = False
        else:
            self.movable[2] = True
        west = maze[x-1][y]
        if west==1 or west==2 or west==3 or west==4:    # West: x-1, y
            self.movable[3] = False
        else:
            self.movable[3] = True
```

stack:

```
from linkedlist import LinkedList


class Stack:

    def __init__( self ):
        self.stack = LinkedList()


    def push( self, item ):
        self.stack.addFirst(item)


    def pop( self ):
        removed = self.top()
        self.stack.removePosition(0)
        return removed.data


    def top( self ):
        return self.stack.getFirst()
        #return self.stack.getPosition(self.stack.getSize()-1)


    def isEmpty( self ):
        size = self.stack.getSize()
        if size<=0:
            return True
        else:
            return False
```

linkedlist:

```
class ListNode:

          def __init__(self, data, previous, next1):
                    self.data = data
                    self.prev = previous
                    self.next = next1


class LinkedList:

          def __init__(self):
                    self.first = None
                    self.last = None
                    self.size = 0


          def getFirst(self):
```

```
            return self.first


    def addFirst(self, item):
            if self.size==0:
                    self.first = ListNode(item, None, None)
                    self.last = self.first
                    self.size+=1

            else:
                    temp = self.first
                    self.first = ListNode(item, None, temp)
                    temp.prev = self.first
                    self.size+=1


    def addLast(self, item):
            if self.size==0:
                    self.first = ListNode(item, None, None)
                    self.last = self.first
                    self.size+=1

            else:
                    temp = self.last
                    self.last = ListNode(item, temp, None)
                    temp.next = self.last
                    self.size+=1


    def addPosition(self, n, item):
            temp = self.first
            if n==0:
                    self.addFirst(item)
            elif n==(self.size-1):
                    self.addLast(item)
            else:
                    while n>0:
                            temp = temp.next
                            n-=1
                    new = ListNode(item, temp.prev, temp)
                    temp.prev.next = new
                    temp.prev = new
                    self.size+=1


    def removePosition(self, n):
            temp = self.first
            if self.size==1:
                    self.first.data = None
                    self.last = self.first
            elif n==(self.size-1):
                    self.last.prev.next = None
            elif n==0:
                    self.first = self.first.next
                    self.first.prev = None
            else:
                    while n>0:
                            temp = temp.next
                            n-=1
                    temp.prev.next = temp.next
                    temp.next.prev = temp.prev
```

```
                    self.size-=1


        def getPosition(self, n):
                    temp = self.first
                    while n>0:
                                temp = temp.next
                                n-=1
                    return temp.data


        def getSize(self):
                    return self.size
```

print_lab:

```
from labyrinth import Labyrinth
from player import Player

#111, 49
lab  = Labyrinth([20, 20])

if lab.matrix[lab.x_size/2][lab.y_size/2]:
    x = lab.x_size/2+1
    y = lab.y_size/2
    player = Player([x, y])
    lab.matrix[lab.x_size/2+1][lab.y_size/2] = 8
else:
    x = lab.x_size/2
    y = lab.y_size/2
    player = Player([x, y])
    lab.matrix[lab.x_size/2][lab.y_size/2] = 8

i = 0
word = ''
while i<lab.y_size:
    for j in lab.matrix:
        word+=str(j[i])
    print word
    word = ''
    i+=1
'''
i = 0
while i<lab.y_size:
    print ' X' *lab.x_size
    i+=1
'''
'''
while not player.at_exit():
    print ' '*40
    word = ''
    j = 0
    while j<lab.y_size:
        for k in lab.matrix:
```

```
            if k[j]==1:
                word+=' X'
            elif k[j]==2:
                word+=' |'
            elif k[j]==3:
                word+=' |'
            elif k[j]==4:
                word+=' >'
            elif k[j]==5:
                word+=' <'
            elif k[j]==7:
                word+=' E'
            elif k[j]==8:
                word+=' P'
            else:
                word+=' .'
        print word
        word = ''
        j+=1
    dir = int(raw_input("Direction:"))
    player.move(dir, lab.matrix)
print ' '*40'''
'''
lab.to_file('haba')
value = lab.from_file('haba')
if value:
    print 'Success!'
else:
    print 'Not that successful..'

i = 0
word = ''
while i<lab.y_size:
    for j in lab.matrix:
        word+=str(j[i])
    print word
    word = ''
    i+=1
'''
solved = lab.solve([x, y])

print ' '*40
#print mat
i = 0
word = ''
while i<lab.y_size:
    for j in solved:
        word+=str(j[i])
    print word
    word = ''
    i+=1

print ' '*40
word = ''
j = 0
while j<lab.y_size:
    for k in solved:
        if k[j]==1:
            word+=' X'
        elif k[j]==2:
            word+=' |'
```

```
        elif k[j]==3:
            word+=' |'
        elif k[j]==4:
            word+=' >'
        elif k[j]==5:
            word+=' <'
        elif k[j]==7:
            word+=' E'
        elif k[j]==8:
            word+=' P'
        elif k[j]==9:
            word+=' S'
        else:
            word+=' .'
    print word
    word = ''
    j+=1
```