Mikko Ylikangas

TEKOÄLYN SIMULAATION TOTEUTTAMINEN UNITY3D PELIYMPÄRISTÖSSÄ

Opinnäytetyö
KESKI-POHJANMAAN AMMATTIKORKEAKOULU
Mediatekniikan koulutusohjelma
Toukokuu 2011

KESKI-POHJANMAAN AMMATTIKORKEAKOULU
MELLERSTA ÖSTERBOTTENS YRKESHÖGSKOLA

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö	**Aika**	**Tekijä/tekijät**
Ylivieskan yksikkö	Toukokuu 2011	Mikko Ylikangas
Koulutusohjelma		
Mediatekniikka		
Työn nimi		
Tekoälyn simulaation toteuttaminen Unity3D peliympäristössä		
Työn ohjaaja		**Sivumäärä**
Hannu Puomio		17 + 24
Työelämäohjaaja		

Opinnäytetyön aiheena oli A* -algoritmin avulla tehdä reaaliaikainen reitihakuohjelma Unity3D -ympäristössä.

Kappaleessa kaksi on kerrottu, miten A* -algoritmi toimii teoriassa. Kappaleessa kolme on kerrottu ohjelmoinnin eri vaiheet suunnittelusta toteutukseen. Kappaleessa neljä on tulokset ja pohdinta.

Asiasanat

A*, A-star, Algoritmi, C#, C-sharp, Ohjelmointi, Reitinhaku, Tekoäly

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

CENTRAL OSTROBOTHNIA UNIVERSITY OF APPLIED SCIENCES	Date May 2011	Author Mikko Ylikangas
Degree programme Media technology		
Name of thesis Implementation of artificial intelligence simulation in Unity3D game environment		
Instructor Hannu Puomio		Pages 17 + 24
Supervisor		

The subject of the thesis was an A* algorithm based pathfinding program for Unity3D environment.

Chapter two contains basics of how the A* algorithm works in theory. Chapter three contains the various stages of the programming. Chapter four contains results and discussion.

Key words

A*, Algorithm, A-star, Artificial intelligence, C#, C-sharp, Pathfinding, Programming

1. JOHDANTO

Opinnäytetyön ideana oli kehittää ohjelmointitaitojani, koska en omannut suurta määrää käytännön ohjelmointikokemusta. Aihe oli oma ehdotukseni ja tarkoituksella hieman hankala riittävän haasteen takaamiseksi. Alustavasti tarkoituksena oli kehittää laajempi tekoälysimulaatio, mutta jo pitkäksi venyneen opiskeluajan takia päätimme supistaa aiheen pelkkään reitinhakualgoritmiin.

A* pohjaisia reitinhakualgoritmeja Unity3D alustalle oli saatavilla useampia, mutta päätin olla hyödyntämättä niitä omassa työssäni. Koko ohjelma on kirjoitettu käyttämällä apuna vain algoritmin toiminnasta kertovia ohjeita ilman suoria koodiesimerkkejä.

Ohjelma tehtiin Unity3D:llä ja kaikki koodi kirjoitettiin Microsoft Visual Studio 2008 professionalissa C# kielellä. Kaikki 3D-elementit luotiin Unity3D:ssä olevista primitiiveistä ja UI-elementteihin käytettiin peruspohjaa, joka Unity3D:ssä on vakiona.

Kappaleessa kaksi kerrotaan A-star algoritmin toiminnasta teoriassa. Kappaleessa selitetään reitinhakumekanismi, reitin hinnoittelu ja lopullisen reitin määritys. Kappaleessa kolme käydään läpi ohjelmoinnin eri vaiheet suunnittelusta toteutukseen, sekä ohjelman kaikkien eri luokkien toiminta ja tarkoitus. Kappale neljä sisältää tulokset ja pohdinnan

2. A-STAR ALGORITMIN TOIMINTA

Algoritmi pohjautuu soluihin, joihin koko alue jaetaan. Aloitussolusta voidaan liikkua mihin tahansa soluun, johon aloitussolu on yhteydessä. Jokaisella solulla on kolme arvoa: H (heuristiikka), F (lopullinen arvo) ja G (arvo aloitussolusta tähän soluun). F-arvo on H- ja G-arvon summa. Heuristiikalla lasketaan suorin reitti aloitussolusta kohdesoluun välittämättä reitillä olevista esteistä. Tutkittavan solun G-arvo lasketaan lisäämällä kiinteä liikkumishinta aktiivisen solun G-arvoon. Aktiivisesta solusta liikutaan aina siihen soluun, jonka F-arvo on pienin.

Algoritmissa käytetään kahta listaa: Avointa- ja suljettua listaa. Avoimessa listassa on kaikki solut, joita voidaan käyttää mahdollisena reittinä. Suljetussa listassa taas on kaikki solut, jotka on jo tarkistettu. (Lester 2005.)

2.1. Reitin haku

Reitin haku aloitetaan tarkistamalla aloitussolun viereiset solut. Tämän jälkeen tutkitaan kaikki solut aloitussolusta ulospäin kunnes kohdesolu löydetään.

Haku aloitetaan tekemällä seuraavat toimenpiteet:

1. Lisätään aloitussolu avoimelle listalle tutkittavaksi.
2. Tarkistetaan kaikki solut, jotka ovat yhteydessä aloitussoluun ja eivät sisällä seiniä, vettä tai muuta sisältöä, jossa ei voi liikkua. Kaikki solut joihin voi liikkua lisätään avoimelle listalle. Jokaiselle listaan laitetulle solulle määritetään, että ne ovat aloitussolun lapsisoluja. Merkintä on tärkeää, koska tällä tiedolla merkitään lopullinen reitti myöhemmin.
3. Poistetaan aloitussolu avoimelta listalta ja siirretään se suljetulle listalle, koska

kyseistä solua ei tarvitse tutkia enää.

Hakua jatketaan aloitussolun jälkeen hakemalla avoimesta listasta solu, jonka F-arvo on pienin. Hakua jatketaan seuraavilla toimenpiteillä:

4. Pienimmällä F-arvolla oleva solu määritetään tutkittavaksi soluksi, poistetaan avoimesta listasta ja lisätään suljettuun listaan.
5. Tarkistetaan kaikki solut, jotka ovat yhteydessä tutkittavaan soluun, välittämättä niistä soluista joiden kautta ei voi kulkea. Lisätään solut avoimeen listaan, jos ne eivät vielä ole siellä. Sekä merkitään solut tutkittavan solun lapsisoluiksi.
6. Jos joku viereisistä soluista on jo avoimella listalla tarkistetaan onko reitti soluun parempi. Toisin sanoen tarkistetaan onko solun G-arvo pienempi, jos tutkittavaa solua käytetään reittinä soluun. Jos arvo on suurempi ei tehdä mitään. Jos taas uuden reitin G-arvo on pienempi solun isännäksi merkitään tutkittava solu ja lasketaan F- ja G-arvot uusiksi. (Lester 2005.)

2.2. Reitin hinnoittelu

Oikean reitin löytäminen suoritetaan seuraavalla kaavalla:

F = G + H

- G = siirtymishinta aloitussolusta tiettyyn soluun kartalla seuraten sinne luotua reittiä
- H = arvioitu siirtymishinta kohdesolusta tiettyyn soluun. Tätä kutsutaan heuristiikaksi, koska se on arvaus. Lopullista matkaa kohdesoluun ei tiedetä ennen kuin reitti sinne on löydetty.

Reitti luodaan käymällä toistuvasti läpi avointa listaa ja valitsemalla sieltä solu, jonka F arvo on pienin.

G on siirtymishinta mikä täytyy maksaa, että voi liikkua aloitussolusta johonkin toiseen soluun käyttäen sinne luotua reittiä. Siirtymishinta määritellään vapaasti, mutta esimerkiksi

neliöistä luodussa kaksiulotteisessa matriisissa arvo voisi olla 10 pystysuoralle ja sivuttaiselle liikkumiselle ja 14 viistolle liikkumiselle. G-arvon laskeminen tietylle solulle onnistuu lisäämällä solun isäntäsolun G-arvoon kiinteän siirtymishinnan.

H voidaan arvioida monella eri tavalla. Yksi yksinkertaisimmista tavoista on Manhattan metodi, missä lasketaan pysty- ja vaakasuoraan liikuttavien solujen määrä kohdesolusta tutkittavaan soluun esteet huomioimatta. Tämän jälkeen kerrotaan solujen määrä siirtymishinnalla.

F lasketaan summaamalla G- ja H-arvot. (Lester 2005.)

2.3. Lopullisen reitin määrittäminen

Reitinhakua jatketaan niin kauan, kunnes kohdesolu päätyy suljetulle listalle tai avoin lista on tyhjä. Jos päädytään tilanteeseen, jossa avoin lista on tyhjä ja kohdesolua ei ole löydetty, reittiä kohdesoluun ei ole olemassa.

Lopullinen reitti määritetään tarkistamalla kohdesolun isäntäsolu ja siirtymällä solu kerrallaan takaisinpäin tarkistamalla aina seuraavan solun isäntäsolu kunnes päädytään aloitussoluun. (Lester 2005.)

3. OHJELMOINTI

3.1. Suunnittelu

Unity3D:lle oli valmiiksi muutamia reitinhakualgoritmeja. Näistä mainittavia olivat *X9* ja *AngryAnt*, joista kumpaakaan en käyttänyt tähän työhön.

Ennen ohjelmoinnin aloittamista tein suunnitelman eri luokista, joita ohjelmaan tarvittaisiin. Alkuperäinen suunnitelma sisälsi viisi eri luokkaa, joista yksikään ei toteutunut suunnitelman mukaisesti. Suunnitellut luokat olivat:

- Agent
  - Luokka, joka ohjaa kentällä liikkuvan agentin toimintaa.
- Pathfinder
  - Itse reitinhakuluokka, joka välittää agentille kuljettavan reitin.
- ScenePainter
  - Ruudunpäivittämistä varten. Piirtää kentän näytölle.
- Main
  - Pääluokka, joka sisältää käyttöliittymän ja muut päätoiminnot
- Gameboard
  - Kentän tiedot sisältävä luokka.

Alustavasti oli tarkoitus tehdä käyttöliittymä omana luokkana, mutta oli helpompaa kirjoittaa se *Main*-luokan sisään. Lopulliseen työhön tuli näiden viiden luokan lisäksi vielä kaksi luokkaa:

- RTSCamera

    - Kameranohjausluokka, jonka otin aiemmasta projektistani.
- SingleSquare
    - Luokka, joka sisältää kentällä olevan solun tiedot.

Ohjelman ensimmäisissä versioissa *Gameboard*-luokka sisälsi myös kaikki tiedot soluista, mutta yhden solun informaation määrä kasvoi niin suureksi, että niille oli pakko luoda oma luokka.

Varhaisissa versioissa kamera oli kiinteästi yhdessä kohdassa, mikä johti ongelmiin isommissa kentissä.

3.2. Luokat

3.2.1. Agent

Agent luokka ohjaa kentällä liikkuvaa agenttia eli hahmoa, joka hyödyntää reitinhakua. Kun agentti luodaan kentälle se tarkistaa, onko sillä kohdepistettä minne liikkua. Jos loppupistettä ei löydy agentti ottaa satunnaisen pisteen kentältä ja tarkistaa onko piste avoin. Jos piste on avoin luo agentti pisteestä kohdepisteen ja kutsuu *Pathfinder* luokkaa hakemaan suorimman reitin pisteeseen.

Agentti odottaa kunnes *Pathfinder* palauttaa sille listan koordinaatteja ja lähtee liikkumaan koordinaattien pohjalta. Kun agentti päätyy kohdepisteeseen aloittaa se koko toimenpiteen alusta.

3.2.2. Gameboard

Gameboard on pieni luokka, joka sisältää pelialueen. Luokalla ei ole kuin yksi funktio, jolla pelialue voidaan alustaa tyhjäksi.

3.2.3. Main

Main luokka on ohjelman pääluokka, joka jatkaa *MonoBehaviour* luokkaa. Jotta Unity3D:ssä voidaan suorittaa skriptejä niiden täytyy jatkaa *MonoBehaviour* luokkaa ja luokan täytyy sisältää *Start()*- ja *Update()*-funktiot. Tässä luokassa on myös *OnGUI()*-funktio, johon määritellään kaikki käyttöliittymään kuuluva koodi. Main luokkaan on myös koodattu kaikki muu käyttäjän ja ohjelman väliseen kanssakäyntiin tarvittava koodi.

Ohjelman käynnistyksessä suoritetaan *Start()*-funktio, joka alustaa *Gameboardin* ja *ScenePainterin*. Käynnistyksen yhteydessä ladataan myös kartta, ettei joka kerta tarvitse luoda tai ladata uutta karttaa.

Update()-funktiota kutsutaan jokaisella ruudunpäivityksellä. Tähän funktioon sisällytetään koodi, jossa käsitellään esimerkiksi hiiren klikkaukset pelikentälle. Main luokan *Update()*-funktioon on määritelty seinien lisäys ja poisto, sekä agentin lisääminen kentälle. Seinien poiston ja lisäyksen yhteydessä tarkistetaan myös, onko uusi seinä agentin reitillä tai poistetaanko seinä reitin läheltä. Jos joku näistä on tosi lasketaan agentin reitti uusiksi, koska on mahdollista, että reitti lyhentyy tai pitenee muutoksen takia.

OnGUI()-funktiossa on määritelty kaikki näytöllä näkyvät käyttöliittymäelementit eli painikkeet, ikkunat ja tekstit. Funktiossa on myös uuden kartan luonti ja kartan lataaminen tiedostosta. Käyttöliittymän eri tilat on delegoitu *GUIMethod()* tunnukseen. Määrittämällä tälle tunnukselle eri funktioita voidaan saada eri käyttöliittymän osat näkyviin.

3.2.4. Pathfinder

Pathfinder luokassa on itse reitinhakualgoritmi. *Pathfinderia* kutsutaan Agent luokan kautta aina, kun agentti tarvitsee uuden reitin itselleen.

Luokan alustuksessa määritellään kaikki mahdolliset suunnat mihin tietystä solusta voidaan liikkua. Kun reittiä pyydetään lähetetään kutsussa mukana agentin tiedot, *scenepainterin* tiedot ja *gameboardin* tiedot. Ennen reitinhakua alustetaan kaikki muuttujat ja etsitään sekä alku- että loppupiste. Reitinhaku aloitetaan tallentamalla alku- ja loppupisteiden koordinaatit ja lisäämällä aloituspiste avoimelle listalle.

Reitinhakuluuppia pyöritetään niin kauan, että avoin lista tyhjenee tai loppupiste päätyy suljetulle listalle. Jos reitti löytyi lähdetään loppupisteestä takaisinpäin solun parentID:n avulla ja tallennetaan jokainen solu listaan kunnes päädytään aloituspisteeseen. ReturnPath()-funktiolla palautetaan lista, jossa reitti on.

3.2.5. RTSCamera

RTSCamera luokka hoitaa kaiken kameran liikuttamiseen, sekä lähentämiseen ja loitontamiseen liittyvän toiminnan. Luokan avulla kenttää voidaan liikuttaa siirtämällä hiiri kentän reunoille, liikuttamalla hiirtä ALT-näppäin pohjassa tai keskimmäinen hiiren nappi pohjassa. Zoomaus tapahtuu hiiren rullalla tai + ja – näppäimillä. Home näppäimellä voi palauttaa kameran takaisin aloituspaikalle.

3.2.6. Scenepainter

Scenepainter määrittää kaikki kentän objektit ja piirtää ne näytölle aina, kun muutoksia tapahtuu. Alustusfunktiossa ladataan objektien 3D-mallit ja määritellään niiden värit. *Update()*-funktio poistaa kartan ja piirtää sen uudelleen. Muuttujista riippuen kartalle piirretään reitti ja muu reitinhakuun liittyvä informaatio.

3.2.7. SingleSquare

Tässä luokassa on määritelty solujen tilat, jotka ovat vapaa, seinä, alku tai loppu. Määriteltynä on myös kaikki muuttujat, jotka soluihin tallennetaan.

3.3. Toteutus

3.3.1. Kenttä ja sen piirtäminen

Ennen varsinaisen reitinhakualgoritmin ohjelmointia täytyi tehdä koko ohjelman runko, joka sisälsi ensimmäisen version kentästä ja käyttöliittymästä. Ensimmäisenä luotiin kenttä ja luokka, jolla kenttä saatiin piirrettyä näytölle.

Kenttä on kaksiulotteinen taulukko, jossa jokainen ruutu on yksi kentän solu. Taulukko luodaan tyhjänä, jonka jälkeen jokaiseen ruutuun tallennetaan alustettu solu. Kentän alustus suoritetaan kahdella sisäkkäisellä *for*-silmukalla. Kentän piirtäminen suoritetaan samanlaisilla sisäkkäisillä *for*-silmukoilla, jossa solun tyyppi määrää millainen objekti kentälle piirretään.

```
_map = new SingleSquare[mapSizeX, mapSizeY];
for (int i = 0; i < mapSizeX; i++)
{
	for (int j = 0; j < mapSizeY; j++)
	{
		_map[i, j] = new SingleSquare();
		_map[i, j].ID = id;
		id++;
		_map[i, j].LocX = i;
		_map[i, j].LocY = j;
	}
}
```

Aluksi koko kenttä on vapaasti kuljettavaa aluetta. Solun muuttaminen seinäksi tapahtuu klikkaamalla solua, kun seinäpainike on pohjassa. Solun klikkaaminen tunnistetaan fysiikkamoottorin avulla ampumalla kenttää kohti säde kohdasta, missä klikkaus tapahtui. Agentin lisääminen kentälle toimii samalla periaatteella, kuin seinänkin lisääminen. Valitaan vain agentpainike ja klikataan jotain solua.

```
if (Input.GetMouseButtonDown(0) && !disableMouseClicks)
	{
	ray = Camera.main.ScreenPointToRay(Input.mousePosition);
		if (Physics.Raycast(ray, out hitInfo))
		{
			switch (actionGridNumber)
			{
				case 0:
```

Viimeisessä versiossa ohjelmasta agentti luo loppupisteen automaattisesti satunnaiseen paikkaan kentällä. Kun loppupiste löytyy kutsuu agentti *pathfinderia*, jolle se välittää itsensä, *scenepainterin* ja *gameboardin*. Agentti pyörittää *while*-silmukkaa niin kauan, että *pathfindfer* saa laskettua reitin. Kun *pathfinder* ilmoittaa, että reitti on laskettu agentti hakee sen ja aloittaa liikkumisen.

3.3.2. Reitinhaku

Reitinhakusilmukka aloitetaan määrittämällä tutkittava solu, joka on avoimessa listassa matalimmalla F-arvolla oleva solu. Solu siirretään avoimelta listalta suljetulle listalle, jonka jälkeen ympärillä olevat solut tutkitaan.

```
while (!done)
{
	debugLoopCount++;
	//etsitään matalimmalla finalcostilla oleva blokki
	currentSquare = (SingleSquare)openList[0];
	foreach (SingleSquare square in openList)
	{
		if (square.F < currentSquare.F) currentSquare = square;
	}
	//poistetaan blokki openlististä
	openList.Remove(currentSquare);
	//listätään blokki closedlistiin
	closedList.Add(currentSquare);
	currentSquare.IsVisited = true;
```

Ympärillä olevat solut käydään läpi liikkumisvaihtoehtolistan avulla, jonka pohjalta lasketaan myös liikkumishinta.

```
//käydään läpi liikkumisvaihtoehdot
foreach (Vector3 move in OpenDirections(currentSquare.LocX,
currentSquare.LocY))
{
	int moveX = (int)move.x;
	int moveY = (int)move.z;
	//lasketaan onko liike viisto vai suora
	if (Math.Abs(map[moveX, moveY].LocX - currentSquare.LocX) +
Math.Abs(map[moveX, moveY].LocY - currentSquare.LocY) == 2)
	{
		moveCost = 14;
	}
	else moveCost = 10;
```

Kun liikkumishinta on saatu selville tarkistetaan onko kyseisellä suunnalla oleva solu suljetulla listalla tai seinä, jos näin on silmukka katkaistaan, ja siirrytään tutkimaan seuraavaa solua.

```
        //tarkistetaan onko blokki suljetulla listalla tai onko se seinä
        bool notInClosedList = true;
        if (map[moveX, moveY].Type == SingleSquare.BlockType.Wall)
        {
              notInClosedList = false;
        }
        foreach (SingleSquare square in closedList)
        {
        if (map[moveX, moveY].ID == square.ID)
        {
              notInClosedList = false;
              break;
        }
}
```

Jos solu ei ole seinä eikä sitä löydy suljetulta listalta tutkitaan onko se avoimella listalla. Jos solu löytyy avoimelta listalta tarkistetaan onko aktiivisen solun G-arvon ja liikkumishinnan summa pienempi kuin tutkittavan solun. Jos näin on poistetaan solu avoimelta listalta, koska nykyisen solun kautta liikkuminen on halvempaa. Jos taas G-arvo on suurempi lasketaan kaikki arvot uusiksi ja merkitään aktiivinen solu tutkittavan solun isännäksi.

```
//jos blokki ei ole closedlistillä tai seinä
if (notInClosedList)
{
//tarkistetaan onko se openlistillä
       bool notOnOpenList = true;
       foreach (SingleSquare square in openList)
       {
//jos löytyy tarkistetaan G ja verrataan sitä entiseen tulokseen.
              if (map[moveX, moveY].ID == square.ID)
              {
//jos nykyisen palikan G + movecost on pienempi kun palikan G arvo poistetaan
palikka openlistiltä koska nykyisen palikan kautta liikkuminen on halvempaa
                     if (currentSquare.G + moveCost < square.G)
                     {
                            openList.Remove(square);
                            break;
                     }
//jos G:n arvo on pienempi lasketaan arvot uusiksi ja merkataan currentBlock
tämän blokin parentiksi.
                     else if (map[moveX, moveY].G < square.G)
                     {
                            map[moveX, moveY].ParentID = currentSquare.ID;
                            map[moveX, moveY].G = currentSquare.G + moveCost;
                            map[moveX, moveY].F = map[moveX, moveY].G +
map[moveX, moveY].H;
                     }
                     notOnOpenList = false;
                     break;
              }
       }
```

Jos solu ei ole seinä eikä sitä löydy kummaltakaan listalta lasketaan sille kaikki arvot, merkitään aktiivinen solu sen isännäksi ja lisätään se avoimelle listalle. Heuristiikkaa laskettaessa lasketaan myös niin sanottu *tie breaker*, joka nostaa solujen H-arvoa sitä enemmän mitä kauempana ne ovat suoralta linjalta alku- ja loppupisteiden välistä. Arvoa nostetaan erittäin pieniä määriä, mutta se riittää tekemään linjalla olevat solut halutummaksi kuin muualla olevat solut.

```
        if (notOnOpenList) {
    //TieBreakerin laskeminen
                float dx1 = moveX - endX;
                float dy1 = moveY - endY;
                float dx2 = startX - endX;
                float dy2 = startY - endY;
                float cross = Math.Abs(dx1 * dy2 - dx2 * dy1);
    //Lasketaan heuristiikka
                int xDistance = Math.Abs(map[moveX, moveY].LocX - endX);
                int yDistance = Math.Abs(map[moveX, moveY].LocY - endY);
                if (xDistance > yDistance) {
                        map[moveX, moveY].H = 14 * yDistance + 10 * (xDistance -
    yDistance);
                        map[moveX, moveY].H += cross * 0.001f;
                }
                else {
                        map[moveX, moveY].H = 14 * xDistance + 10 * (yDistance -
    xDistance);
                        map[moveX, moveY].H += cross * 0.001f;
                }
                map[moveX, moveY].G = currentSquare.G + moveCost;
                map[moveX, moveY].F = map[moveX, moveY].G + map[moveX, moveY].H;
                map[moveX, moveY].ParentID = currentSquare.ID;
                openList.Add(map[moveX, moveY]);
                map[moveX, moveY].IsInOpenList = true;
                }
        }
    }
```

Reitinhakusilmukkaa ajetaan niin kauan, että loppupiste päätyy suljettuun listaan, jolloin reitti löytyi. Jos avoin lista tyhjenee ennen reitin löytymistä ei reittiä loppupisteeseen ole, missä tapauksessa silmukan ajaminen lopetetaan.

```
    //jos reitti löytyi
    foreach (SingleSquare square in closedList)
    {
            if (square.Type == SingleSquare.BlockType.End)
            {
                    done = true;
                    pathFound = true;
            }
    }
    //jos reitti ei löytynyt
    if (openList.Count == 0)
    {
            done = true;
            pathFound = false;
            sp.Update(gb);
    }
```

Kun reitti on löydetty luodaan lista, johon merkitään koko reitti lopusta alkuun solujen isäntätiedon avulla.

```
if (pathFound)
{
	wayPoints = new ArrayList();
	pathMarked = false;
	map[endX, endY].IsPath = true;
	int pathID = map[endX, endY].ParentID;
	wayPoints.Add(new Vector3((float)map[endX, endY].LocX, 0f,
(float)map[endX, endY].LocY));
	while (!pathMarked)
	{
		for (int i = 0; i < mapSizeX; i++)
		{
			for (int j = 0; j < mapSizeY; j++)
			{
				if (map[i, j].ID == pathID)
				{
					map[i, j].IsPath = true;
					pathID = map[i, j].ParentID;
//tallennetaan kaikki reittinäolevat pisteet arraylistiin
					Vector3 point = new Vector3((float)map[i,
j].LocX, 0f, (float)map[i, j].LocY);
					wayPoints.Add(point);
				}
				if (map[i, j].Type == SingleSquare.BlockType.Start
&& map[i, j].IsPath == true) pathMarked = true;
			}
		}
		yield return null;
	}
}
```

Agentti voi tämän jälkeen pyytää reitin kutsumalla *pathfinderin ReturnPath()*-funktiota, joka palauttaa listan, jos reitti on löytynyt.

```
public ArrayList ReturnPath()
{
	if (pathFound && pathMarked)
	{
		return wayPoints;
	}
	else return null;
}
```

4. TULOKSET JA POHDINTA

4.1. Tulokset

Toimiva algoritmi saatiin tehtyä, vaikka sen nopeus ei vastaa tehtävänannossa määrättyä nopeutta. A-Star pohjaisen reitinhakualgoritmin ohjelmoiminen ei ollut niin vaikeaa kuin alun perin oletin. Tämän projektin ansiosta olen oppinut tekoälyn ohjelmoinnista paljon.

4.2. Pohdinta

Työssä tavoitteena oli ohjelmoida Unity3D alustalla toimiva reaaliaikainen (30fps) A* reitinhakualgoritmi. Toimiva algoritmi saatiin kirjoitettua, mutta se ei todellakaan ole reaaliaikainen. Reitinhaku kestää yhdellä agentilla reitin pituudesta riippuen 0.5 – 20 sekuntia 30x30 kokoisessa kentässä.

Unity3D on skriptipohjainen kehitysalusta, jota ei niinkään ole suunniteltu niin sanotun ”puhtaan” koodin kirjoittamiseen. Unity3D:llä voi kirjoittaa, joko javalla tai C#:lla, mutta C# tuki on hieman vajaa. Esimerkiksi luokkien *constructor*-funktioita ei voi käyttää, koska Unity3D suorittaisi kyseiset funktiot kehitystilassakin. *Constructorin* tilalla käytin *Init()*-funktiota, jota täytyi kutsua aina kun haluttiin alustaa uusi instanssi luokasta.

Unity3D ei myöskään antanut käyttää *SortedList* luokkaa, joten kaikki listat ohjelmassa on tehty tavallisilla *ArrayListeillä*. Tämä on yksi suurimmista syistä miksi reitin haku on niin hidasta. Jos aikaa olisi ollut enemmän olisin kirjoittanut koko ohjelman alusta uusiksi ja luonut oman listan, joka olisi toiminut *Binary Heap* menetelmällä. *Binary Heap*-lista olisi toiminut huomattavasti tehokkaammin ja nopeuttanut reitinhakua tämän ohjelman tapauksessa niin paljon, että reitin hakua olisi voitu väittää reaaliaikaiseksi.

Olen jo suunnitellut tekeväni uuden reitinhakuohjelman C++ kielellä, johon tulen käyttämään tästä projektista saatua tietoa ja taitoa.

LÄHTEET

Lester Patrick 2005. A* Pathfinding for Beginners. Saatavissa: http://www.policyalmanac.org/games/aStarTutorial.htm. Luettu 12.11.2010.

Agent.cs

```
using UnityEngine;
using System.Collections;
//sisältää ukkelit jokka juoksee kartalla.

public class Agent : MonoBehaviour
{
    //kartan koko
    private int _mapSizeX = -1;
    public int MapSizeX { get { return _mapSizeX; } set { _mapSizeX = value; } }
    private int _mapSizeY = -1;
    public int MapSizeY { get { return _mapSizeY; } set { _mapSizeY = value; } }
    //itse kartta
    private SingleSquare[,] _map;
    public SingleSquare[,] Map { get { return _map; } set { _map = value; } }
    //reittipisteet
    private ArrayList _waypoints = new ArrayList();
    public ArrayList Waypoints { get { return _waypoints; } set { _waypoints =
value; } }
    //agentin nykyinen paikka kartalla
    private Vector3 _coordinates = new Vector3();
    public Vector3 Coordinates { get { return _coordinates; } set { _coordinates =
value; } }
    //seuraava piste reitillä
    private Vector3 _nextWaypoint = new Vector3();
    public Vector3 NextWaypoint { get { return _nextWaypoint; } set { _nextWaypoint =
value; } }
    //agentin malli
    private GameObject _model;
    public GameObject Model { get { return _model; } set { _model = value; } }

    private bool _findingPath = false;
    public bool FindingPath { get { return _findingPath; } set { _findingPath = value; }
}
    //onko agentilla reitti
    private bool _hasPath = false;
    public bool HasPath { get { return _findingPath; } set { _findingPath = value; } }
    //liikkuuko agentti
    private bool _isMoving = false;
    public bool IsMoving { get { return _isMoving; } set { _isMoving = value; } }
    //liikkuuko agentti automaattisesti
    private bool _autoMove = true;
    public bool AutoMove { get { return _autoMove; } set { _autoMove = value; } }

    //loppukoordinaatit
    private Vector3 endCoordinates;
    private GameObject mdl;
    //pathfindingin tietoja joita välitetään main classille agentin kautta
    private bool _pathFound = false;
    public bool PathFound { get { return _pathFound; } set { _pathFound = value; } }
    private bool _pathMarked = false;
    public bool PathMarked { get { return _pathMarked; } set { _pathMarked = value; } }
    private float _loopTime = 0f;
    public float LoopTime { get { return _loopTime; } set { _loopTime = value; } }

    //onko agentti while luupissa odottamassa että se saa reitin
    private bool _waitingPath = false;
    public bool WaitingPath { get { return _waitingPath; } set { _waitingPath = value; }
}
    private int _loopCount = 0;
    public int LoopCount { get { return _loopCount; } set { _loopCount = value; } }

    public IEnumerator Draw(Scenepainter sp, Gameboard gb)
    {
        Pathfinder pathfind = new Pathfinder();
        endCoordinates = new Vector3();
        bool pathfindStarted = false;

        bool endFound = false;
        //tarkistetaan löytyykö kentältä loppupiste agentille
        for (int x = 0; x < _mapSizeX; x++)
        {
            for (int y = 0; y < _mapSizeY; y++)
            {
                if (_map[x, y].Type == SingleSquare.BlockType.End)
                {
                    endFound = true;

                    //_map[x, y].Type = SingleSquare.BlockType.Free;
```

```
                    //_map[x, y].AgentID = -1;
                }
            }
        }
        //jos loppua ei löydy luodaan satunnaisesta pisteestä sellainen
        while (!endFound)
        {
            int x = (int)Random.Range(0f, _mapSizeX);
            int y = (int)Random.Range(0f, _mapSizeY);
            if (_map[x, y].Type == SingleSquare.BlockType.Free)
            {
                _map[x, y].Type = SingleSquare.BlockType.End;
                endFound = true;
            }
            yield return null;
        }

        //luupataan tyhjää niin kauan että reitti löytyy
        while (!pathfind.pathMarked && !_isMoving)
        {
            _waitingPath = true;
            if (!pathfindStarted)
            {
                pathfind.Init();
                StartCoroutine(pathfind.Pathfind(this, sp, gb));
                pathfindStarted = true;
            }
            //Debug.Log("Waiting for path.");
            yield return new WaitForSeconds(0.5f);
        }
        _waitingPath = false;
        //kun reitti on löytynyt haetaan se
        if (pathfind.ReturnPath() != null)
        {
            _waypoints = pathfind.ReturnPath();
            _hasPath = true;

            mdl = (GameObject)Resources.Load("AgentModel");

            StartCoroutine(Move(sp, gb));
        }
        _pathFound = pathfind.pathFound;
        _pathMarked = pathfind.pathMarked;
        _loopTime = pathfind.loopTime;
        _loopCount = pathfind.debugLoopCount;
    }

    private IEnumerator Move(Scenepainter sp, Gameboard gb)
    {
        //etsitään aloituskoordinaatit ja luodaan agentti jos sitä ei ole
        //muussa tapauksessa siirretään agentti aloituskoordinaatteihin
        bool startFound = false;
        bool endFound = false;
        for (int i = 0; i < _mapSizeX; i++)
        {
            for (int j = 0; j < _mapSizeY; j++)
            {
                if (_map[i, j].Type == SingleSquare.BlockType.Start)
                {
                    startFound = true;
                    if (_model == null) _model = (GameObject)GameObject.Instantiate(mdl,
new Vector3(i + 0.5f, 0f, j + 0.5f), Quaternion.identity);
                    else
                    {
                        float lerpTime = 0f;
                        float rate = 2f;
                        while (lerpTime < 1.0f)
                        {
                            lerpTime += Time.deltaTime * rate;
                            if (_model != null) _model.transform.position =
Vector3.Lerp(_model.transform.position, new Vector3(i + 0.5f, 0f, j + 0.5f), lerpTime);
                            else Debug.Log("Notice: Agentmodel is null");
                            yield return null;
                        }
                    }
                }
                if (_map[i, j].Type == SingleSquare.BlockType.End)
                {
                    endFound = true;
                    endCoordinates = new Vector3(_map[i, j].LocX, 0f, _map[i, j].LocY);
```

```
                    Debug.Log("end coordinates: " + endCoordinates);
                }
            }
        }
        //jos alku- ja loppukoordinaatit löytyvät alotetaan liikkumisluuppi
        if (startFound && endFound)
        {
            Debug.Log("Agent moving!");
            //luupataan niin kauan kun agentilla on reitti
            while (_hasPath)
            {
                //jos agentti ei vielä liiku eli kun alku- ja loppukoordinaatit on juuri
laitettu
                if (!_isMoving)
                {
                    //etsitään alkukoordinaatit
                    for (int i = 0; i < _mapSizeX; i++)
                    {
                        for (int j = 0; j < _mapSizeY; j++)
                        {
                            if (_map[i, j].Type == SingleSquare.BlockType.Start)
                            {
                                //tallennetaan ne
                                _coordinates = new Vector3(i + 0.5f, 0f, j + 0.5f);
                                //jos reittipisteitä löytyy listasta
                                if (_waypoints.Count > 0)
                                {
                                    //otetaan niistä viimeinen ja tallennetaan seuraavan
pisteen koordinaatit
                                    _nextWaypoint = (Vector3)_waypoints[_waypoints.Count
- 1];
                                    //Debug.Log("Next waypoint ID: " + s.ID);
                                    _nextWaypoint += new Vector3(0.5f, 0f, 0.5f);
                                    //poistetaan piste listalta
                                    _waypoints.RemoveAt(_waypoints.Count - 1);
                                }
                                _coordinates = _nextWaypoint;
                                float lerpTime = 0f;
                                float rate = 2f;
                                //luupataan lerppiä niinkauan että agentti on liikkunut
seuraavaan pisteeseen
                                while (lerpTime < 1.0f)
                                {
                                    lerpTime += Time.deltaTime * rate;
                                    if(_model != null) _model.transform.position =
Vector3.Lerp(_model.transform.position, _coordinates, lerpTime);
                                    else Debug.Log("Notice: Agentmodel is null");
                                    yield return null;
                                }
                                _isMoving = true;
                            }
                        }
                    }
                }
                else
                {
                    //jos agentti liikkuu jo
                    if (_isMoving)
                    {
                        if (_waypoints.Count > 0)
                        {
                            _nextWaypoint = (Vector3)_waypoints[_waypoints.Count - 1];
                            //Debug.Log("Next waypoint ID: " + s.ID);
                            _nextWaypoint += new Vector3(0.5f, 0f, 0.5f);
                            _waypoints.RemoveAt(_waypoints.Count - 1);
                        }
                        _coordinates = _nextWaypoint;
                        float lerpTime = 0f;
                        float rate = 2f;
                        while (lerpTime < 1.0f)
                        {
                            lerpTime += Time.deltaTime * rate;
                            if(_model != null) _model.transform.position =
Vector3.Lerp(_model.transform.position, _coordinates, lerpTime);
                            else Debug.Log("Notice: Agentmodel is null");
                            yield return null;
                        }
                        for (int i = 0; i < _mapSizeX; i++)
                        {
                            for (int j = 0; j < _mapSizeY; j++)
```

```
                            {
                                if (_map[i, j].Type == SingleSquare.BlockType.Start)
_map[i, j].Type = SingleSquare.BlockType.Free;
                            }
                        }
                        _map[(int)_nextWaypoint.x, (int)_nextWaypoint.z].Type =
SingleSquare.BlockType.Start;
                    }
                    if (_waypoints.Count == 0 && !_autoMove)
                    {
                        _isMoving = false;
                        _hasPath = false;
                    }
                    else if (_waypoints.Count == 0 && _autoMove)
                    {
                        endFound = false;
                        for (int i = 0; i < _mapSizeX; i++)
                        {
                            for (int j = 0; j < _mapSizeY; j++)
                            {
                                if (_map[i, j].Type == SingleSquare.BlockType.End)
_map[i, j].Type = SingleSquare.BlockType.Free;
                            }
                        }
                        while (!endFound)
                        {
                            int x = (int)Random.Range(0f, _mapSizeX);
                            int y = (int)Random.Range(0f, _mapSizeY);
                            if (_map[x, y].Type == SingleSquare.BlockType.Free)
                            {
                                _map[x, y].Type = SingleSquare.BlockType.End;
                                endFound = true;
                            }
                            yield return null;
                        }
                    }
                    for (int i = 0; i < _mapSizeX; i++)
                    {
                        for (int j = 0; j < _mapSizeY; j++)
                        {
                            //tarkistetaan onko loppukoordinaatit vaihtunut
                            if (_map[i, j].Type == SingleSquare.BlockType.End)
                            {
                                //jos ei liikutaan seuraavaan pisteeseen
                                if (_map[i, j].LocX == (int)endCoordinates.x && _map[i,
j].LocY == (int)endCoordinates.z)
                                {

                                }
                                //muuten alotetaan uusi luuppi ja katkaistaan nykyinen
                                else
                                {
                                    Debug.Log("End coordinates changed.");

                                    _hasPath = false;
                                    _isMoving = false;
                                    _waypoints = new ArrayList();

                                    StartCoroutine(this.Draw(sp, gb));

                                    yield break;
                                }
                            }
                        }
                    }
                }
                yield return null;
            }
        }
    }

    public void RemoveAgent()
    {
        GameObject.Destroy(this._model);
        _hasPath = false;
        _isMoving = false;
    }
}
```

GameBoard.cs

```
using UnityEngine;
using System.Collections;
//sisältää pelialueen
public class Gameboard
{
    public int mapSizeX = -1;
    public int mapSizeY = -1;

    private SingleSquare[,] _map;
    public SingleSquare[,] Map { get { return _map; } set { _map = value; } }

    //Alustaa kartan joka ruudun tyhjäksi
    public void Init(int X, int Y)
    {
        int id = 0;
        mapSizeX = X;
        mapSizeY = Y;
        _map = new SingleSquare[mapSizeX, mapSizeY];
        for (int i = 0; i < mapSizeX; i++)
        {
            for (int j = 0; j < mapSizeY; j++)
            {
                _map[i, j] = new SingleSquare();
                _map[i, j].ID = id;
                id++;
                _map[i, j].LocX = i;
                _map[i, j].LocY = j;
            }
        }
    }
}
```

Main.cs

```
using UnityEngine;
using System.Collections;
using System.Xml;
using System.IO;
using System;
//pääskripti käyttää muita ja sisältää GUI osat
public class Main : MonoBehaviour
{
    //muuttujien määrittelyt
    private delegate void GUIMethod();
    private GUIMethod currentGUIMethod;
    private bool disableMouseClicks = false;

    private string statusBoxText = "";
    private string secondStatusBoxText = "";
    private Ray ray;
    private RaycastHit hitInfo;

    private int actionGridNumber = 0;
    private string[] actionGridItems = { "Wall", "Agent", "Select" };

    private string location = "";
    private DirectoryInfo dI = null;
    private Vector2 scrollPosition;
    private Vector2 scrollMax;

    string newMapSizeX = "";
    string newMapSizeY = "";

    public int mapSizeX;
    public int mapSizeY;

    private Gameboard gb = new Gameboard();
    private Scenepainter sp = new Scenepainter();

    ArrayList agents = new ArrayList();
    int selectedAgent = 0;
    //Agent a;
    // Use this for initialization
    void Start()
    {
        this.currentGUIMethod = DefaultGUI;

        gb.Init(mapSizeX, mapSizeY);
        sp.Init(gb);
        StartCoroutine(OpenMapFile(Application.dataPath + "//Maps//defaultmap.xml"));
        //Lisätään käynnistyksen yhteydessä yksi agentti
        //testaamista varten vain
        /*
        a = (Agent)gameObject.AddComponent<Agent>();
        a.MapSizeX = gb.mapSizeX;
        a.MapSizeY = gb.mapSizeY;
        a.Map = gb.Map;
        agents.Add(a);
         */
    }

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0) && !disableMouseClicks)
        {
            ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray, out hitInfo))
            {
                switch (actionGridNumber)
                {
                    //tarkistetaan mikä nappi on painettuna
                    //Lisätään tai poistetaan seinä
                    case 0:
                        if (gb.Map[(int)hitInfo.point.x, (int)hitInfo.point.z].Type ==
SingleSquare.BlockType.Free)
                        {
                            gb.Map[(int)hitInfo.point.x, (int)hitInfo.point.z].Type =
SingleSquare.BlockType.Wall;
                            foreach (Agent a in agents)
                            {
                                if (a.Map[(int)hitInfo.point.x,
```

```
(int)hitInfo.point.z].Type == SingleSquare.BlockType.Free) a.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].Type = SingleSquare.BlockType.Wall;
                            }
                            //pävitetään pathfinding vain jos uusi seinä tuli reitin
päälle


                            foreach (Agent a in agents)
                            {
                                if (a.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].IsPath == true) StartCoroutine(a.Draw(sp, gb));
                            }

                        }
                        else if (gb.Map[(int)hitInfo.point.x, (int)hitInfo.point.z].Type
== SingleSquare.BlockType.Wall)
                        {
                            gb.Map[(int)hitInfo.point.x, (int)hitInfo.point.z].Type =
SingleSquare.BlockType.Free;
                            bool isCloseToPath = false;
                            foreach (Agent a in agents)
                            {
                                if (a.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].Type == SingleSquare.BlockType.Wall) a.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].Type = SingleSquare.BlockType.Free;

                                //tarkistetaan jos seinä tulee lähelle reittiä
                                //jos seinä on lähellä reittiä lasketaan reitti uusiksi
koska seinän poisto saattaa lyhentää reittiä
                                try
                                {
                                    if (a.Map[(int)hitInfo.point.x + 1,
(int)hitInfo.point.z].F != -1)
                                    {
                                        isCloseToPath = true;
                                    }
                                    else if (a.Map[(int)hitInfo.point.x - 1,
(int)hitInfo.point.z].F != -1)
                                    {
                                        isCloseToPath = true;
                                    }
                                    else if (a.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z + 1].F != -1)
                                    {
                                        isCloseToPath = true;
                                    }
                                    else if (a.Map[(int)hitInfo.point.x + 1,
(int)hitInfo.point.z - 1].F != -1)
                                    {
                                        isCloseToPath = true;
                                    }
                                }
                                catch (IndexOutOfRangeException e)
                                {
                                    Debug.Log(e);
                                }
                            }

                            if (isCloseToPath)
                            {
                                foreach (Agent a in agents)
                                {
                                    StartCoroutine(a.Draw(sp, gb));
                                }
                            }
                        }
                        sp.Update(gb);
                        break;
                    case 1:
                        Agent ag = (Agent)gameObject.AddComponent<Agent>();
                        ag.MapSizeX = gb.mapSizeX;
                        ag.MapSizeY = gb.mapSizeY;
                        ag.Map = new SingleSquare[gb.mapSizeX, gb.mapSizeY];
                        int id = 0;
                        //täytetään kartta tyhjillä palikoilla
                        for (int i = 0; i < ag.MapSizeX; i++)
                        {
                            for (int j = 0; j < ag.MapSizeY; j++)
                            {
                                ag.Map[i, j] = new SingleSquare();
```

```
                                if (gb.Map[i, j].Type == SingleSquare.BlockType.Wall)
ag.Map[i, j].Type = SingleSquare.BlockType.Wall;
                                if (i == (int)hitInfo.point.x && j ==
(int)hitInfo.point.z) ag.Map[i, j].Type = SingleSquare.BlockType.Start;
                                ag.Map[i, j].ID = id;
                                id++;
                                ag.Map[i, j].LocX = i;
                                ag.Map[i, j].LocY = j;
                            }
                        }
                        agents.Add(ag);

                        foreach (Agent a in agents)
                        {
                            StartCoroutine(a.Draw(sp, gb));
                        }
                        //sp.Update(gb);

                        break;
                    case 2:
                        break;
                    default:
                        Debug.Log("Error: Selection Grid # is wrong!");
                        break;
                }
                sp.Update(gb);
            }
            else Debug.Log("Notice: Cursor outside of gamearea!");
        }
    }

    //UI koodi alkaa
    void OnGUI()
    {
        this.currentGUIMethod();
    }

    public void DefaultGUI()
    {
        statusBoxText = "Selected button # " + actionGridNumber + "\n"
                        + "Coordinates: " + (int)hitInfo.point.x + " " +
(int)hitInfo.point.z + "\n\n"
                        + "BlockID: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].ID + "\n"
                        + "Block type: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].Type + "\n\n"
                        + "Is path: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].IsPath + "\n"
                        + "Is in openlist: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].IsInOpenList + "\n"
                        + "Is visited: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].IsVisited + "\n\n"
                        + "Block H-value: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].H + "\n"
                        + "Block G-value: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].G + "\n"
                        + "Block F-value: " + gb.Map[(int)hitInfo.point.x,
(int)hitInfo.point.z].F + "\n";
        GUI.Label(new Rect(0, 0, 200, 400), statusBoxText);
        //if (agents.Count > 0)
        //{
        //    Agent agent = (Agent)agents[0];
        //    secondStatusBoxText = "IsMoving: " + agent.IsMoving + "\n"
        //                    + "HasPath: " + agent.HasPath + "\n"
        //                    + "NextWaypoint: " + agent.NextWaypoint + "\n\n"
        //                    + "PathFound: " + agent.PathFound + "\n"
        //                    + "PathMarked: " + agent.PathMarked + "\n\n"
        //                    + "LoopTime: " + agent.LoopTime + "\n\n"
        //                    + "WaitingPath: " + agent.WaitingPath + "\n"
        //                    + "AutoMove: " + agent.AutoMove;
        //    GUI.Label(new Rect(Screen.width - 200, 0, 200, 400), secondStatusBoxText);
        //}
        Agent agent;
        if (agents.Count > 0)
        {
            agent = (Agent)agents[selectedAgent];
            secondStatusBoxText = "SelectedAgent: " + selectedAgent + "\n"
                        + "IsMoving: " + agent.IsMoving + "\n"
                        + "HasPath: " + agent.HasPath + "\n"
                        + "NextWaypoint: " + agent.NextWaypoint + "\n\n"
```

```
                            + "PathFound: " + agent.PathFound + "\n"
                            + "PathMarked: " + agent.PathMarked + "\n\n"
                            + "LoopCount: " + agent.LoopCount + "\n"
                            + "LoopTime: " + agent.LoopTime + "\n\n"
                            + "WaitingPath: " + agent.WaitingPath;
            GUI.Label(new Rect(Screen.width - 200, 0, 200, 400), secondStatusBoxText);

            if (GUI.Button(new Rect(Screen.width - 50, 200, 50, 25), "-->"))
            {
                if (agents.Count > 0 && selectedAgent < agents.Count - 1)
                {
                    selectedAgent++;
                }
                else selectedAgent = 0;
            }
            if (GUI.Button(new Rect(Screen.width - 100, 200, 50, 25), "<--"))
            {
                if (agents.Count > 0 && selectedAgent > 0)
                {
                    selectedAgent--;
                }
                else selectedAgent = agents.Count - 1;
            }
        }
        sp.drawRoute = GUI.Toggle(new Rect(5, 200, 150, 25), sp.drawRoute, "Draw
route");

        GUI.BeginGroup(new Rect(0, Screen.height - 75, 140, 70), "");

        GUI.Box(new Rect(0, 0, 140, 70), "");
        actionGridNumber = GUI.SelectionGrid(new Rect(5, 5, 130, 60), actionGridNumber,
actionGridItems, 2);

        GUI.EndGroup();

        GUI.BeginGroup(new Rect(Screen.width - 145, Screen.height - 75, 140, 70), "");
        GUI.Box(new Rect(0, 0, 140, 70), "");
        if (GUI.Button(new Rect(5, 5, 60, 25), "Open"))
        {
            location = Application.dataPath;
            dI = new DirectoryInfo(location);

            scrollPosition = new Vector2();
            scrollMax = new Vector2();
            this.currentGUIMethod = OpenFileGUI;
            disableMouseClicks = true;

        }
        if (GUI.Button(new Rect(5, 40, 60, 25), "Clear"))
        {
            for (int i = 0; i < gb.mapSizeX; i++)
            {
                for (int j = 0; j < gb.mapSizeY; j++)
                {
                    gb.Map[i, j].Type = SingleSquare.BlockType.Free;
                    gb.Map[i, j].IsPath = false;
                    gb.Map[i, j].IsVisited = false;
                    gb.Map[i, j].IsInOpenList = false;
                }
            }
            foreach (Agent a in agents)
            {
                a.AutoMove = false;
                a.RemoveAgent();
                a.IsMoving = false;

            }
            agents.Clear();
            sp.Update(gb);
        }
        if (GUI.Button(new Rect(70, 40, 60, 25), "Quit"))
        {
            Application.Quit();
        }
        if (GUI.Button(new Rect(70, 5, 60, 25), "New"))
        {
            disableMouseClicks = true;
            this.currentGUIMethod = NewMapGUI;
        }
        GUI.EndGroup();
```

```
    }
    //uuden kartan luonti
    public void NewMapGUI()
    {
        GUI.BeginGroup(new Rect(5, 20, 200, Screen.height - 40), "");
        GUI.Box(new Rect(0, 0, 155, 120), "Enter new map size.");
        GUI.Label(new Rect(5, 30, 80, 20), "X size: ");
        GUI.Label(new Rect(5, 60, 80, 20), "Y size: ");

        newMapSizeX = GUI.TextField(new Rect(100, 30, 50, 20), newMapSizeX, 5);
        newMapSizeY = GUI.TextField(new Rect(100, 60, 50, 20), newMapSizeY, 5);

        if (GUI.Button(new Rect(5, 90, 70, 25), "Cancel"))
        {
            disableMouseClicks = false;
            this.currentGUIMethod = DefaultGUI;
        }

        if (GUI.Button(new Rect(80, 90, 70, 25), "Done"))
        {
            try
            {
                gb.mapSizeX = Convert.ToInt32(newMapSizeX);
                gb.mapSizeY = Convert.ToInt32(newMapSizeY);
                if (gb.mapSizeX > 0 && gb.mapSizeY > 0)
                {
                    gb.Map = new SingleSquare[gb.mapSizeX, gb.mapSizeY];
                    int id = 0;
                    for (int i = 0; i < gb.mapSizeX; i++)
                    {
                        for (int j = 0; j < gb.mapSizeY; j++)
                        {
                            gb.Map[i, j] = new SingleSquare();
                            gb.Map[i, j].ID = id;
                            id++;
                            gb.Map[i, j].LocX = i;
                            gb.Map[i, j].LocY = j;
                        }
                    }
                    disableMouseClicks = false;
                    this.currentGUIMethod = DefaultGUI;
                }
            }
            catch (FormatException e)
            {
                Debug.Log(e);
            }
            foreach (Agent a in agents)
            {
                a.MapSizeX = gb.mapSizeX;
                a.MapSizeY = gb.mapSizeY;
                a.Map = gb.Map;
                StartCoroutine(a.Draw(sp, gb));
            }
            sp.Update(gb);
        }
        GUI.EndGroup();
    }

    //uuden kartan avaus tiedostosta
    public void OpenFileGUI()
    {
        GUI.Box(new Rect(10, 20, Screen.width / 3, Screen.height - 40), "Open map");

        if (dI.FullName != "/" && GUI.Button(new Rect(20, 40, Screen.width / 3 - 20,
Screen.height / 9), "Parent folder"))
        {
            dI = dI.Parent;
        }

        GUILayout.BeginArea(new Rect(20, 50 + Screen.height / 9, Screen.width / 3 - 20,
Screen.height - (100 + Screen.height / 9)));
        GUILayout.Label(dI.FullName + " :");
        scrollMax = GUILayout.BeginScrollView(scrollPosition);
        // Kansio
        foreach (DirectoryInfo item in dI.GetDirectories())
        {
            if (GUILayout.Button(new GUIContent(item.Name,
(Texture2D)Resources.Load("folder-icon"))))
            {
```

```
                dI = item;
            }
        }
        // Tiedosto
        foreach (FileInfo item in dI.GetFiles())
        {
            if (GUILayout.Button(new GUIContent(item.Name)))
            {
                if (item.Extension == ".xml")
                {
                    disableMouseClicks = false;
                    StartCoroutine(OpenMapFile(item.FullName));
                    this.currentGUIMethod = DefaultGUI;
                }
            }
        }
        GUILayout.EndScrollView();
        GUILayout.EndArea();

        //peruutusnappi
        if (GUI.Button(new Rect(20, Screen.height - 50, 75, 25), "Cancel"))
        {
            disableMouseClicks = false;
            this.currentGUIMethod = DefaultGUI;
        }
    }
    //karttatiedoston lukeminen
    private IEnumerator OpenMapFile(string p)
    {
        Debug.Log("OpenMapFile running");
        XmlDocument mapFile = new XmlDocument();
        //jos polun pituus on 0 ei polkua saatu ja mitään ei voi tehdä
        if (p.Length != 0)
        {
            Debug.Log("p.Length: " + p.Length + "\n" + "Path = " + p);
            //luodaan uusi www polulla joka saatiin UIn kautta
            WWW www = new WWW("file:///" + p);
            while (!www.isDone)
            {
                //odotetaan että www saa ladattua tiedoston
                Debug.Log("WWW is loading");
                yield return www;
            }
            //tallennetaan wwwn lataama teksti xmldociksi
            mapFile.LoadXml(www.text);
            Debug.Log("Mapfile loaded");
            //käydään xml läpi
            XmlNode mapData = mapFile.SelectSingleNode("mapData");
            foreach (XmlNode data in mapData.ChildNodes)
            {
                //luodaan uusi kartta xml:stä saaduilla koolla
                if (data.Attributes.GetNamedItem("sizeX").Value != null) gb.mapSizeX =
Convert.ToInt32(data.Attributes.GetNamedItem("sizeX").Value);
                if (data.Attributes.GetNamedItem("sizeY").Value != null) gb.mapSizeY =
Convert.ToInt32(data.Attributes.GetNamedItem("sizeY").Value);

                gb.Map = new SingleSquare[gb.mapSizeX, gb.mapSizeY];
                int id = 0;
                //täytetään kartta tyhjillä palikoilla
                for (int i = 0; i < gb.mapSizeX; i++)
                {
                    for (int j = 0; j < gb.mapSizeY; j++)
                    {
                        gb.Map[i, j] = new SingleSquare();
                        gb.Map[i, j].ID = id;
                        id++;
                        gb.Map[i, j].LocX = i;
                        gb.Map[i, j].LocY = j;
                    }
                }

                Debug.Log("Map size (x,y): " + gb.mapSizeX + "," + gb.mapSizeY);
                //if (data.Attributes.GetNamedItem("mapData").Value != null)
                //luetaan itse karttadata
                if (data.InnerText.ToString() != null)
                {
                    char[] c = new char[1];
                    StringReader sr = new StringReader(data.InnerText.ToString());

                    while (sr.Peek() >= 0)
```

```
                    {
                        for (int j = gb.mapSizeY - 1; j >= 0; j--)
                        {
                            for (int i = 0; i < gb.mapSizeX; i++)
                            {
                                sr.Read(c, 0, 1);
                                string si = new string(c);
                                //Debug.Log(si);
                                switch (si)
                                {
                                    case "0":
                                        //Debug.Log("Block type = " + gb.Map[i,j].Type);
                                        gb.Map[i, j].Type = SingleSquare.BlockType.Free;
                                        break;
                                    case "1":
                                        //Debug.Log("Block type = " + gb.Map[i,
j].Type);
                                        gb.Map[i, j].Type = SingleSquare.BlockType.Wall;
                                        break;
                                    case "S":
                                        //Debug.Log("Block type = " + gb.Map[i,
j].Type);
                                        gb.Map[i, j].Type =
SingleSquare.BlockType.Start;
                                        break;
                                    case "E":
                                        //Debug.Log("Block type = " + gb.Map[i,
j].Type);
                                        gb.Map[i, j].Type = SingleSquare.BlockType.End;
                                        break;
                                    default:
                                        Debug.Log("Error: Unidentified character in map-
file!");
                                        gb.Map[i, j].Type = SingleSquare.BlockType.Free;
                                        break;
                                }
                            }
                        }
                        //tuhotaan streamreader
                        sr.Dispose();
                    }
                }
            }
            //käsketään kaikkia agentteja hakemaan uusi reitti
            foreach (Agent a in agents)
            {
                a.MapSizeX = gb.mapSizeX;
                a.MapSizeY = gb.mapSizeY;
                a.Map = gb.Map;
                StartCoroutine(a.Draw(sp, gb));
            }
            sp.Update(gb);
        }
    }
```

Pathfinder.cs

```
using UnityEngine;
using System.Collections;
using System;
//sisältää pathfinding koodin.

public class Pathfinder
{
    public int debugLoopCount = 0;

    public bool done = false;
    public bool pathFound = false;
    public bool pathMarked = false;

    public float loopTime = 0f;
    private float startTime = 0f;

    private Vector3[] possibleDirections;

    public ArrayList openList = new ArrayList();
    public ArrayList closedList = new ArrayList();
    public ArrayList wayPoints = new ArrayList();

    private SingleSquare[,] map;
    private int mapSizeX;
    private int mapSizeY;

    //alustusfunktio
    public void Init()
    {
        //luodaan array jossa on kaikki suunnat
        possibleDirections = new Vector3[]
        {
            new Vector3(-1f,0f,-1f),
            new Vector3(1f,0f,-1f),
            new Vector3(1f,0f,1f),
            new Vector3(-1f,0f,1f),
            new Vector3(0f,0f,-1f),
            new Vector3(1f,0f,0f),
            new Vector3(0f,0f,1f),
            new Vector3(-1f,0f,0f)
        };
    }

    //pää pathfind funktio.
    public IEnumerator Pathfind(Agent a, Scenepainter sp, Gameboard gb)
    {
        //tallennetaan funktion aloitusaika, jota käytetään pathfindingin keston
määrittämiseen
        startTime = Time.time;
        //tallennetaan muuttujia
        map = a.Map;
        mapSizeX = a.MapSizeX;
        mapSizeY = a.MapSizeY;
        debugLoopCount = 0;
        //nollataan kartta
        ResetMap();

        //Debug.Log("Pathfind starting");
        Vector3 start = new Vector3();
        Vector3 end = new Vector3();
        SingleSquare currentSquare;

        int startX = 0;
        int startY = 0;
        int endX = 0;
        int endY = 0;

        openList = new ArrayList();
        closedList = new ArrayList();

        float moveCost = 10;
        done = false;

        start = FindBlockType(SingleSquare.BlockType.Start);
        end = FindBlockType(SingleSquare.BlockType.End);
        //tarkistetaan onko sekä alku- että loppupiste kartalla. jos ei keskeytetään
funktio
        if (start == new Vector3(-1f, -1f, -1f) || end == new Vector3(-1f, -1f, -1f))
```

```
        {
            done = true;
            pathFound = false;
        }
        // tallennetaan alku- ja loppupisteiden koordinaatit
        startX = (int)start.x;
        startY = (int)start.z;

        endX = (int)end.x;
        endY = (int)end.z;

        //asetetaan alkupiste tutkittavaksi palikaksi, määritetään sen F arvo nollaksi
ja lisätään se avoimeen listaan
        try
        {
            map[startX, startY].F = 0;
            currentSquare = map[startX, startY];
            openList.Add(currentSquare);
        }
        catch (IndexOutOfRangeException e)
        {
            Debug.Log(e);
        }
        //pääluuppi jota pyöritetään niinkauan kun openlistillä on palikoita tai
loppupiste päätyy closedlistiin
        while (!done)
        {
            debugLoopCount++;
            //Debug.Log("While loop starting. Loop #: " + debugLoopCount);
            //etsitään matalimmalla finalcostilla oleva blokki
            currentSquare = (SingleSquare)openList[0];
            foreach (SingleSquare square in openList)
            {
                if (square.F < currentSquare.F) currentSquare = square;
            }
            //Debug.Log("CurrentSquare: " + currentSquare.ID);
            //poistetaan blokki openlististä
            openList.Remove(currentSquare);
            //listätään blokki closedlistiin
            closedList.Add(currentSquare);
            currentSquare.IsVisited = true;
            //käydään läpi liikkumisvaihtoehdot
            foreach (Vector3 move in OpenDirections(currentSquare.LocX,
currentSquare.LocY))
            {
                int moveX = (int)move.x;
                int moveY = (int)move.z;
                //lasketaan onko liike viisto vai suora ja muutetaan moveCost sen mukaan
                if (Math.Abs(map[moveX, moveY].LocX - currentSquare.LocX) +
Math.Abs(map[moveX, moveY].LocY - currentSquare.LocY) == 2)
                {
                    moveCost = 14;
                }
                else moveCost = 10;
                //Debug.Log("Move cost: " + moveCost);
                //tarkistetaan onko blokki suljetulla listalla tai onko se seinä
                bool notInClosedList = true;
                if (map[moveX, moveY].Type == SingleSquare.BlockType.Wall)
                {
                    notInClosedList = false;
                }
                foreach (SingleSquare square in closedList)
                {
                    if (map[moveX, moveY].ID == square.ID)
                    {
                        notInClosedList = false;
                        break;
                    }
                }
                //jos blokki ei ole closedlistillä tai seinä
                if (notInClosedList)
                {

                    //Debug.Log("Item was not in closedList.");
                    //tarkistetaan onko se openlistillä
                    bool notOnOpenList = true;
                    foreach (SingleSquare square in openList)
                    {
                        //jos löytyy tarkistetaan G ja verrataan sitä entiseen
tulokseen.
```

```
                        if (map[moveX, moveY].ID == square.ID)
                        {
                            //Debug.Log("Item is in openList.");
                            //if (currentSquare.G < map[moveX, moveY].G + moveCost)
openList.Remove(map[moveX, moveY]);
                            //jos nykyisen palikan G + movecost on pienempi kun palikan
G arvo poistetaan palikka openlistiltä
                            //koska nykyisen palikan kautta liikkuminen on halvempaa
                            if (currentSquare.G + moveCost < square.G)
                            {
                                openList.Remove(square);
                                break;
                            }
                            //jos G:n arvo on pienempi lasketaan arvot uusiksi ja
merkataan currentBlock tämän blokin parentiksi.
                            else if (map[moveX, moveY].G < square.G)
                            {
                                map[moveX, moveY].ParentID = currentSquare.ID;
                                map[moveX, moveY].G = currentSquare.G + moveCost;
                                map[moveX, moveY].F = map[moveX, moveY].G + map[moveX,
moveY].H;
                            }
                            notOnOpenList = false;
                            break;
                        }
                    }
                    //jos ei ole openlistillä merkataan currentBlock tämän blokin
parentiksi, lasketaan costit ja lisätään blokki openlistiin
                    if (notOnOpenList)
                    {
                        //Debug.Log("Item was not in openList.");
                        //TieBreakerin laskeminen
                        //tiebreaker nostaa palikoiden H arvoa sitä enemmän mitä
kauempana ne ovat suoralta linjalta alku- ja loppupisteiden välillä

                        float dx1 = moveX - endX;
                        float dy1 = moveY - endY;
                        float dx2 = startX - endX;
                        float dy2 = startY - endY;
                        float cross = Math.Abs(dx1 * dy2 - dx2 * dy1);

                        //Lasketaan heuristiikka
                        //heuristiikalla arvioidaan halvin reitti alkupisteestä
loppupisteeseen
                        int xDistance = Math.Abs(map[moveX, moveY].LocX - endX);
                        int yDistance = Math.Abs(map[moveX, moveY].LocY - endY);
                        if (xDistance > yDistance)
                        {
                            map[moveX, moveY].H = 14 * yDistance + 10 * (xDistance -
yDistance);
                            map[moveX, moveY].H += cross * 0.001f;
                        }
                        else
                        {
                            map[moveX, moveY].H = 14 * xDistance + 10 * (yDistance -
xDistance);
                            map[moveX, moveY].H += cross * 0.001f;
                        }

                        //Lisätään liikkumisarvo currentSquaren G arvoon ja tallennetaan
se tämän ruudun G arvoksi
                        map[moveX, moveY].G = currentSquare.G + moveCost;
                        //Lasketaan F
                        map[moveX, moveY].F = map[moveX, moveY].G + map[moveX, moveY].H;
                        //Tallennetaan parent arvo
                        map[moveX, moveY].ParentID = currentSquare.ID;
                        //Lisätään blokki openlistille
                        openList.Add(map[moveX, moveY]);
                        map[moveX, moveY].IsInOpenList = true;
                    }
                }
            }
            //jos reitti löytyi
            foreach (SingleSquare square in closedList)
            {
                if (square.Type == SingleSquare.BlockType.End)
                {
                    done = true;
                    pathFound = true;
```

```
                    //Debug.Log("Done. Path found.");
                }
            }
            //jos reitti ei löytynyt
            if (openList.Count == 0)
            {
                done = true;
                pathFound = false;
                //Debug.Log("Done. Path not found.");
                sp.Update(gb);
            }
            //string openListDebug = "";
            //string closedListDebug = "";
            //foreach (SingleSquare debug in openList)
            //{
            //    openListDebug = openListDebug + debug.ID + " ";
            //}
            //foreach (SingleSquare debug in closedList)
            //{
            //    closedListDebug = closedListDebug + debug.ID + " ";
            //}
            //Debug.Log("OpenList: " + openListDebug + "\n" + "ClosedList: " +
closedListDebug);
            yield return null;
            loopTime = Time.time - startTime;
        }
        //jos reitti löytyi merkitään se
        if (pathFound)
        {
            wayPoints = new ArrayList();
            //Debug.Log("Marking the path.");
            pathMarked = false;
            map[endX, endY].IsPath = true;
            int pathID = map[endX, endY].ParentID;
            wayPoints.Add(new Vector3((float)map[endX, endY].LocX, 0f, (float)map[endX,
endY].LocY));
            while (!pathMarked)
            {
                for (int i = 0; i < mapSizeX; i++)
                {
                    for (int j = 0; j < mapSizeY; j++)
                    {
                        if (map[i, j].ID == pathID)
                        {
                            map[i, j].IsPath = true;
                            pathID = map[i, j].ParentID;
                            //Debug.Log("Adding waypoint with ID: " + map[i, j].ID);
                            //tallennetaan kaikki reittinäolevat pisteet arraylistiin
                            Vector3 point = new Vector3((float)map[i, j].LocX, 0f,
(float)map[i, j].LocY);
                            wayPoints.Add(point);
                        }
                        if (map[i, j].Type == SingleSquare.BlockType.Start && map[i,
j].IsPath == true) pathMarked = true;
                    }
                }
                yield return null;
            }
            //näytetään reitti sekunnin ajan
            if (sp.drawRoute)
            {
                for (int i = 0; i < mapSizeX; i++)
                {
                    for (int j = 0; j < mapSizeY; j++)
                    {
                        if (map[i, j].IsPath == true) gb.Map[i, j].IsPath = true;
                        if (map[i, j].IsVisited == true) gb.Map[i, j].IsVisited = true;
                        if (map[i, j].IsInOpenList == true) gb.Map[i, j].IsInOpenList =
true;
                    }
                }
                sp.Update(gb);
                yield return new WaitForSeconds(1f);
                for (int i = 0; i < gb.mapSizeX; i++)
                {
                    for (int j = 0; j < gb.mapSizeY; j++)
                    {
                        if (gb.Map[i, j].IsPath == true) gb.Map[i, j].IsPath = false;
                        if (gb.Map[i, j].IsInOpenList == true) gb.Map[i, j].IsInOpenList
= false;
```

```
                        if (gb.Map[i, j].IsVisited == true) gb.Map[i, j].IsVisited =
false;
                    }
                }
            }
            sp.Update(gb);
        }
    }

    //palauttaa arraylistin jossa on reitti
    public ArrayList ReturnPath()
    {
        if (pathFound && pathMarked)
        {
            return wayPoints;
        }
        else return null;
    }

    //tarkistaa onko tietyt koordinaatit kentän rajojen sisällä
    private bool ValidCoordinates(float x, float y)
    {
        if (x < 0) return false;
        if (x >= mapSizeX) return false;
        if (y < 0) return false;
        if (y >= mapSizeY) return false;
        return true;
    }

    //etsii tietyn blokin
    private Vector3 FindBlockType(SingleSquare.BlockType blockType)
    {
        foreach (Vector3 point in AllBlocks())
        {
            if (map[(int)point.x, (int)point.z].Type == blockType)
            {
                return new Vector3(point.x, 0f, point.z);
            }
        }
        return new Vector3(-1f, -1f, -1f);
    }

    //tarkistaa onko blokki seinä
    private bool BlockOpen(float x, float y)
    {
        if (map[(int)x, (int)y].Type == SingleSquare.BlockType.Wall) return false;
        else return true;
    }

    //palauttaa kaikki palikat järjestyksessä
    private IEnumerable AllBlocks()
    {
        for (int i = 0; i < mapSizeX; i++)
        {
            for (int j = 0; j < mapSizeY; j++)
            {
                yield return new Vector3(i, 0f, j);
            }
        }
        //return new Vector3(-1f, -1f, -1f);
    }

    //palauttaa kaikki mahdolliset suunnat
    private IEnumerable OpenDirections(float x, float y)
    {
        foreach (Vector3 moveDirection in possibleDirections)
        {
            float newX = x + moveDirection.x;
            float newY = y + moveDirection.z;
            if (ValidCoordinates(newX, newY) && BlockOpen(newX, newY))
            {
                yield return new Vector3(newX, 0f, newY);
            }
        }
        //return new Vector3(-1f,-1f,-1f);
    }

    //nollaa kartan arvot
    public void ResetMap()
    {
```

```
            for (int i = 0; i < mapSizeX; i++)
            {
                for (int j = 0; j < mapSizeY; j++)
                {
                    map[i, j].G = -1;
                    map[i, j].H = -1;
                    map[i, j].F = -1;
                    map[i, j].ParentID = -1;
                    map[i, j].IsPath = false;
                    map[i, j].IsVisited = false;
                    map[i, j].IsInOpenList = false;
                }
            }
        }
    }
```

RTSCamera.cs

```
using UnityEngine;
using System.Collections;
//kamerakoodi
public class RTSCamera : MonoBehaviour
{
    private float mPosX = 0f;
    private float mPosY = 0f;
    private float scrollAreaX = 0f;
    private float scrollAreaY = 0f;
    //private float maxZoomIn = 2f;
    //private float maxZoomOut = 40f;
    private float defaultZoomLevel = 20f;

    public bool useEdgeScrolling;

    public float scrollSpeed = 15f;
    public float dragSpeed = 0.5f;
    public float zoomSpeed = 50f;

    public Transform cameraTarget;
       // Use this for initialization
       void Start ()
    {
         //cameraTarget = gameObject.transform;
       }

       // Update is called once per frame
       void Update ()
    {
         mPosX = Input.mousePosition.x;
         mPosY = Input.mousePosition.y;

         scrollAreaX = Screen.width;
         scrollAreaY = Screen.height;

         // Do camera movement by mouse position
         if (useEdgeScrolling)
         {
             if (mPosX <= scrollAreaX)
             {
                 cameraTarget.Translate(Vector3.right * -scrollSpeed * Time.deltaTime,
Space.World);
             }
             if (mPosX >= Screen.width - scrollAreaX)
             {
                 cameraTarget.Translate(Vector3.right * scrollSpeed * Time.deltaTime,
Space.World);
             }
             if (mPosY <= scrollAreaY)
             {
                 cameraTarget.Translate(Vector3.forward * -scrollSpeed * Time.deltaTime,
Space.World);
             }
             if (mPosY >= Screen.height - scrollAreaY)
             {
                 cameraTarget.Translate(Vector3.forward * scrollSpeed * Time.deltaTime,
Space.World);
             }
         }
         // Do camera movement by keyboard
         cameraTarget.Translate(new Vector3(Input.GetAxis("Horizontal") * scrollSpeed *
Time.deltaTime, 0, Input.GetAxis("Vertical") * scrollSpeed * Time.deltaTime),
Space.World);

         // Do camera movement by middle mouse or alt key
         if ((Input.GetKey("left alt") || Input.GetKey("right alt")) ||
Input.GetMouseButton(2))
         {
             cameraTarget.Translate(-(new Vector3(Input.GetAxis("Mouse X") * dragSpeed,
0, Input.GetAxis("Mouse Y") * dragSpeed)), Space.World);
         }

         //Do camera zoom by mouse wheel

         if (Input.GetAxis("Mouse ScrollWheel") < 0 || Input.GetKey("[-]"))// &&
transform.position.y < maxZoomOut) // back
         {
             cameraTarget.Translate(Vector3.up * zoomSpeed * Time.deltaTime,
```

```
Space.World);
        }
        if (Input.GetAxis("Mouse ScrollWheel") > 0 || Input.GetKey("[+]"))// &&
transform.position.y > maxZoomIn) // forward
        {
            cameraTarget.Translate(Vector3.up * -zoomSpeed * Time.deltaTime,
Space.World);
        }

        //Reset zoom

        if (Input.GetKey("home"))
        {
            DefaultZoom();
        }
      }

    public void DefaultZoom()
    {
        cameraTarget.transform.position = new Vector3(transform.position.x,
defaultZoomLevel, transform.position.z);
    }
}
```

Scenepainter.cs

```
using UnityEngine;
using System.Collections;
//sisältää kartan ja muun piirtämisen näytölle
public class Scenepainter
{

    private GameObject freeBlock;
    private GameObject occupiedBlock;
    private GameObject endBlock;
    private GameObject startBlock;
    private GameObject freeBlockPath;
    private GameObject visitedBlock;
    private GameObject openListBlock;

    public bool drawRoute = false;

    //alustusfunktio jolla ladataan modelit sunmuu
    public void Init(Gameboard gb)
    {
        freeBlock = (GameObject)Resources.Load("FreeBlock");
        freeBlockPath = (GameObject)Resources.Load("FreeBlockPath");
        occupiedBlock = (GameObject)Resources.Load("OccupiedBlock");
        endBlock = (GameObject)Resources.Load("EndBlock");
        startBlock = (GameObject)Resources.Load("StartBlock");
        visitedBlock = (GameObject)Resources.Load("VisitedBlock");
        openListBlock = (GameObject)Resources.Load("OpenListBlock");

        occupiedBlock.renderer.material.color = Color.red;
        endBlock.renderer.material.color = Color.black;
        startBlock.renderer.material.color = Color.magenta;
        freeBlockPath.renderer.material.color = Color.cyan;
        freeBlock.renderer.material.color = Color.white;
        visitedBlock.renderer.material.color = Color.yellow;
        openListBlock.renderer.material.color = Color.green;

        Update(gb);
    }

    //ruudunpäivitysfunktio piirtää kartan näytölle
    public void Update(Gameboard gb)
    {
        RemoveBlocks();
        for (int i = 0; i < gb.mapSizeX; i++)
        {
            for (int j = 0; j < gb.mapSizeY; j++)
            {
                if (gb.Map[i, j].Type == SingleSquare.BlockType.Free)
                {
                    //jos drawroute == true piirretään reitinlaskun kaikki eri vaiheet
omilla väreillä kartalle
                    if (drawRoute)
                    {
                        if (gb.Map[i, j].IsPath) GameObject.Instantiate(freeBlockPath,
new Vector3(i + 0.5f, -0.5f, j + 0.5f), Quaternion.identity);
                        else if (gb.Map[i, j].IsVisited)
GameObject.Instantiate(visitedBlock, new Vector3(i + 0.5f, -0.5f, j + 0.5f),
Quaternion.identity);
                        else if (gb.Map[i, j].IsInOpenList)
GameObject.Instantiate(openListBlock, new Vector3(i + 0.5f, -0.5f, j + 0.5f),
Quaternion.identity);
                        else GameObject.Instantiate(freeBlock, new Vector3(i + 0.5f,
-0.5f, j + 0.5f), Quaternion.identity);
                    }
                    else GameObject.Instantiate(freeBlock, new Vector3(i + 0.5f, -0.5f,
j + 0.5f), Quaternion.identity);
                }
                else if (gb.Map[i, j].Type == SingleSquare.BlockType.Wall)
                {
                    GameObject.Instantiate(occupiedBlock, new Vector3(i + 0.5f, 0f, j +
0.5f), Quaternion.identity);
                }
                else if (gb.Map[i, j].Type == SingleSquare.BlockType.Start)
                {
                    GameObject.Instantiate(startBlock, new Vector3(i + 0.5f, -0.5f, j +
0.5f), Quaternion.identity);
                }
                else if (gb.Map[i, j].Type == SingleSquare.BlockType.End)
                {
```

```
                    GameObject.Instantiate(endBlock, new Vector3(i + 0.5f, -0.5f, j +
0.5f), Quaternion.identity);
                }
                else
                {
                    Debug.Log("Error: Map array has wrong stuff in it!");
                }
            }
        }
    }

    //funktio jolla ruutu tyhjennetään
    private void RemoveBlocks()
    {
        GameObject[] clones;
        clones = GameObject.FindGameObjectsWithTag("FreeBlock");
        foreach (GameObject i in clones)
        {
            GameObject.Destroy(i);
        }
        clones = GameObject.FindGameObjectsWithTag("OccupiedBlock");
        foreach (GameObject i in clones)
        {
            GameObject.Destroy(i);
        }
        clones = GameObject.FindGameObjectsWithTag("EndBlock");
        foreach (GameObject i in clones)
        {
            GameObject.Destroy(i);
        }
    }
}
```

SingleSquare.cs

```
using UnityEngine;
using System.Collections;
//määrittää kentän ruudut
public class SingleSquare
{

    public enum BlockType
    {
        Free,
        Start,
        End,
        Wall
    };
    //palikan tyyppi
    private BlockType _type = BlockType.Free;
    public BlockType Type { get { return _type; } set { _type = value; } }
    //Lopullinen liikkumisarvo
    private float _f = -1;
    public float F { get { return _f; } set { _f = value; } }
    //Arvo alusta pisteeseen
    private float _g = -1;
    public float G { get { return _g; } set { _g = value; } }
    //Heuristiikka eli arvioitu liikkumisarvo
    private float _h = -1;
    public float H { get { return _h; } set { _h = value; } }
    //koordinaatit
    private int _locX = -1;
    public int LocX { get { return _locX; } set { _locX = value; } }
    private int _locY = -1;
    public int LocY { get { return _locY; } set { _locY = value; } }
    //palikan tunnus
    private int _iD = -1;
    public int ID { get { return _iD; } set { _iD = value; } }
    //parent id pathfindingia varten
    private int _parentID = -1;
    public int ParentID { get { return _parentID; } set { _parentID = value; } }

    //reitin piirtämistä varten muutama muuttuja
    private bool _isPath = false;
    public bool IsPath { get { return _isPath; } set { _isPath = value; } }
    private bool _isVisited = false;
    public bool IsVisited { get { return _isVisited; } set { _isVisited = value; } }
    private bool _isInOpenList = false;
    public bool IsInOpenList { get { return _isInOpenList; } set { _isInOpenList =
value; } }
}
```

defaultmap.xml

```
//karttapohja

<?xml version="1.0" encoding="utf-8"?>
<mapData>
	<data sizeX="30" sizeY="30">
111111111111111111111111111111100000000000000000000000000001101111111111111101111111111
011000000000000000101000000010110000000000000001000000000010110000000000000001000000000
010110000000000000001111111111110110000000000000000000000000001100000000000000010000000
000001100000000000000010000000000011000000000000001000000000001100000000000000100000
000000011000000000000000100000000000111111111111111110000000000011000000000000000011
111111111110000000000000000010000000001111111111111111101000000000110000000000000000
100000000001100000000000000000010000000001100000000000000000000000011000000000000000
001111111111111111111111111111101000000000110000000000000001010000000001100000000000
001011111111110110000000000000010000000000011000000000000001000000000001100000000000
000000000000000001100000000000000100000000000110000000000000010000000000011111111111
11111111111111111111
	</data>
</mapData>
```