

Joni Metsälä

# TEKOÄLY CRYENGINE-PELIMOOTTO- RISSA

Opinnäytetyö

Tietotekniikka / Peliohjelmointi

2017



**Kaakkois-Suomen  
ammattikorkeakoulu**

Tekijä/Tekijät Joni Metsälä	Tutkinto Insinööri	Aika Maaliskuu 2017
Opinnäytetyön nimi Tekoäly Cryengine-pelimoottorissa		36 sivua
Toimeksiantaja Gamelab, Kaakkois-Suomen ammattikorkeakoulu		
Ohjaaja Lehtori Niina Salmi		
Tiivistelmä		
<p>Opinnäytetyö käsittelee pelien tekoälyä ja sisältää Cryengine-pelimoottorilla toteutetun konseptin lipunryöstötekoälystä. Lipunryöstössä kaksi joukkuetta kilpailevat ja yrittävät viedä vastustajajoukkueen lipun. Cryengininen lisänä käytettiin sen virallista GameSDK-projektipohjaa.</p> <p>Teoriaosuudessa käsitellään yleisiä tekoälyn toteuttamiseen käytettyjä tapoja, kuten navigaation ja käyttäytymislogiikan toteutusta. Tekoäly on erittäin laaja kokonaisuus, joten kaikkia sen osia ei pystytty sisällyttämään työhön. Käsiteltyä teoriaa vastaavia tapoja sovellettiin toteutuksessa.</p> <p>Keskeisiä menetelmiä toteutuksen kannalta olivat Behavior Tree -rakenteen muodostaminen ja pelilogiikan muodostaminen Cryengininen visuaalisella ohjelmointijärjestelmällä. Behavior Tree kirjoitettiin XML-merkintäkielellä Notepad++ -ohjelmaa käyttäen. Pelimekaniikat ja logiikka tehtiin Cryengininen omalla Flow Graph -järjestelmällä. Rakenteita laajennettiin Cryengineen kuuluvia Signal Reference ja Tactical Point System -menetelmiä käyttäen.</p> <p>Projektille asetetut tavoitteet saavutettiin. Lopullisessa työssä on toimiva Behavior Tree -rakenne sekä hahmoja, jotka käyttävät sitä lipun siirtämiseen pisteiden välillä. Tekoälyn jatkokehitys on aina mahdollista, mutta projektin katsottiin edenneen riittävästi, kun peruslogiikat olivat toimivia ja työ vastasi asetettuja tavoitteita.</p>		
Asiasanat Cryengine, tekoäly, GameSDK, XML, AI, MBT		

Author (authors)	Degree	Time
Joni Metsälä	Bachelor of Engineering	March 2017
Thesis Title		
Artificial Intelligence in Cryengine		36 pages
Commissioned by		
Gamelab, South-Eastern Finland University of Applied Sciences		
Supervisor		
Niina Salmi, Senior Lecturer		
Abstract		
<p>This thesis covers artificial intelligence in games and contains concept of capture the flag game developed with Cryengine. Capture the flag is about two teams competing against each other and trying to steal the flag from the other team. Cryengine was used with an official GameSDK sample project platform.</p> <p>In the theory section, common artificial intelligence practices are examined, such as navigation and behavior logic. Artificial intelligence is a very wide subject. Therefore, not all of its parts were included in the thesis. Practices similar to the examined theory were used in the implementation</p> <p>Essential methods for the implementation were forming a behavior tree structure and game logic with a visual programming tool in Cryengine. The behavior tree was formed with XML markup language using Notepad++ program. Game mechanics and logic were made with Flow Graph tool in Cryengine. Structures were extended using Signal Reference and Tactical Point System included in Cryengine.</p> <p>The objectives set for the project were met. In the final version, the project contained a functional behavior tree structure and characters collecting flags using the structure. Possibilities for extending artificial intelligence are infinite, but the project was considered finished when the main logics were functional and the objectives set at the beginning were reached.</p>		
Keywords		
Cryengine, artificial intelligence, GameSDK, XML, AI, MBT		

# SISÄLLYS

LYHENTEET JA TERMIT .....	5
1 JOHDANTO .....	7
2 AI PELEISSÄ .....	7
2.1 Behavior Tree .....	7
2.2 Navigaatio .....	8
2.3 Tietoisuus .....	9
3 CRYENGINE .....	9
3.1 Cryenginen tekoäly .....	10
3.2 MBT .....	11
3.3 TPS-kysely .....	12
3.4 Lua ja ScriptBind .....	13
3.5 Signal Reference .....	13
3.6 Flow Graph .....	14
3.7 Navigation Mesh Cryenginessä .....	15
4 TOTEUTUS .....	17
4.1 Suunnitelma .....	17
4.2 GameSDK projektin pohjana .....	17
4.3 Versionhallinta ja ohjelmisto .....	18
4.4 Ympäristö .....	19
4.5 Hahmot ja behavior treet .....	20
4.5.1 MBT projektissa .....	20
4.5.2 Flow Graph .....	26
4.6 Virheidenetsintä .....	29
4.7 Projektin kokoaminen .....	31
5 JOHTOPÄÄTÖKSET .....	32
6 LOPUKSI .....	32
LÄHTEET .....	34
KUVALUETTELO .....	36

## LYHENTEET JA TERMIT

AI	Lyhenne englanninkielisestä termistä Artificial Intelligence. Suomeksi tekoäly. AI termiä voidaan käyttää tekoälystä yleensä, tai yksittäisestä itsenäisestä kokonaisuudesta.
Agent	Cryenginen kutsumanimi peleissä oleville hahmoille.
Asset	Pelin tai muun tietokoneohjelman sisältötiedosto, kuten 3D-malli tai äänitiedosto.
Behavior Tree	Tekoälyjen käyttämä logiikkarakenne.
Cryengine	Crytek-yrityksen tekemä pelimoottori, eli ohjelma jota käytetään pelien tekemiseen.
Faction	Ryhmä, jonka perusteella eri agentit voidaan erotella Cryengine-pelimoottorissa.
Flow Graph	Cryenginen visuaalinen ohjelmointijärjestelmä.
Flow Node	Flow Graph -järjestelmän yksittäinen osa, joista kasataan Flow Graph -kaaviot.
GameSDK	Cryenginen pelinkehityspohja. SDK on lyhenne englanninkielisestä termistä software development kit.
MBT	Lyhenne termistä Modular Behavior Tree. Cryenginen uudempi XML-tiedostotyyppiä käyttävä Behavior Tree -päätöksentekojärjestelmä. MBT-konsepti keskittyy nopeaan iterointiin ja uudelleenkäyttöön.

Navigation Mesh	Peleissä käytetty rakenne, joka kertoo tekoälylle liikumiseen käytettävissä olevat alueet.
TPS	Lyhenne termistä Tactical Point System. Cryenginien sijainnihakujärjestelmä.
XML	Merkintäkieli, jolla voidaan tallentaa tietorakenteita.

## 1 JOHDANTO

Opinnäytetyön aiheena on tekoäly, joka on yksi vauhdikkaasti kehittyvistä tietotekniikan aloista. Tekoälyä sovelletaan nykyaikana moniin kohteisiin, kaikilla itse ohjautuvien autojen ja automaattisten puhelinvastaaajien välillä. Uusimpia tekoälyn sovelluksia on neuroverkon avulla sairauksien tunnistaminen (Hansen 2016).

Pelimoottoriksi päätettiin tavanomaisten pelimoottorien Unreal Enginen ja Unityn sijaan ottaa Cryengine, koska työn aikana haluttiin tutustua mahdollisimman moniin uusiin asioihin. Toisena syynä Cryenginen valintaan olivat sen käyttäjäepäystävällisyydestä liikkuneet huhut. Ennen tätä päätöstä selvitettiin kuitenkin pienen taustatyön avulla, että Cryengine sisältää käyttökelpoisen pohjan sekä mahdollisuudet tekoälyn kehittämiseksi. Työ toteutettiin koulun työpisteellä sekä kotona. Projektitiedostoja päivitettiin työpisteiden kesken TortoiseSVN -versionhallinnan avulla.

Työ käsittelee tarkemmin Cryenginen AI-mahdollisuuksia ja implementointitapoja. Toteutuksen aikana kehitettiin skenaario, jossa tietokone pelaa lipunryöstöä itseään vastaan. Lipunryöstössä on kaksi joukkuetta, jotka yrittävät viedä vastustajajoukkueen lipun.

## 2 AI PELEISSÄ

Pelien tekoälyn idea on tehdä pelistä kiinnostavampi ja todenmukaisempi. Eri laisten pelien vaatimukset tekoälyn suhteen vaihtelevat paljon. Peleissä olevan tekoälyn tarkoitus on enemmänkin luoda tietty pelikokemus, eikä simuloida täysin realistista ihmisen toimintaa. Tavoiteltu pelikokemus vaihtelee pelikohtaisesti. (Dill 2014, 3 - 4.)

### 2.1 Behavior Tree

Behavior tree on tekoälyjen hallintaan vakiintunut tekniikka. Behavior Tree antaa paljon joustavuutta muiden tekniikoiden lisäämiseksi, mikä mahdollistaa paljon vaihtoehtoja käyttäytymisen ja suorituskyvyn hallintaan (Champanand

& Dunstan 2014, 73). Behavior treen ”behavior”-osa eli käyttäytyminen tarkoittaa toimintaa ja ehtoja, kuten oven avaaminen ja halutaanko oven toiselle puolelle ylipäättään päästä (Champandard & Dunstan 2014, 75). Ehtojen ja toimintojen avulla saavutetaan halutun kaltainen käyttäytyminen.

## 2.2 Navigaatio

Navigointi on olennaisessa osassa pelien tekoälyssä. Navigoinnin perusperiaate on hakea reitti kahden pisteen välillä. Toimiakseen navigointi tarvitsee tietoonsa käytettävissä olevan alueen sekä algoritmin, joka hakee reitin alueelta. Navigaatioalueen voi esittää usealla eri tavalla. Tavoista valitaan sellainen, joka kykenee edustamaan aluetta mahdollisimman optimaalisesti.

### **Navigation Mesh**

Navigation mesh on polygoneista muodostettu maan muotoja myötäilevä verkko. Kolmion muotoisia polygoneja yhdistämällä voidaan esittää monen muotoisia alueita, joten navigation mesh on käyttökelpoinen useimmissa kohteissa (Sturtevant 2014, 256). Suurempia alueita varten olisi hyvä käyttää algoritmia navigation meshin paikalleen laittamiseen, sillä manuaalisesti se on erittäin aikaa vievää (Sturtevant 2014, 257). Siksi useimmissa pelimootoreissa on integroitu työkalu tätä varten. Lisäksi joissakin pelimootoreissa, esimerkiksi Bethesdan Creation Engineissä, pystyy manuaalisesti hienosäätämään tai luomaan navigation mesh -verkkoa.

Onnistuneesti toteutettu navigation mesh ei erotu lopputuloksessa, koska kaikki toimii oletetusti. Epäonnistunut navigation mesh puolestaan ilmenee nopeasti pelin hahmojen omituisena tai kömpelönä reitinvalintana. Ääritapauksissa reitinhaku epäonnistuu täysin.

### **Ruudukko**

Ruudukkoa tai taulukkoa voidaan käyttää reitinhakuun yksinkertaisissa kaksulotteisissa peleissä. Ruudukkoa ei yleensä käytetä kolmiulotteisissa peleissä,



sillä pelialueet ovat liian monimutkaisen muotoisia. Yksinkertainen implementaatio navigaatoruudukosta sisältää taulukon vapaista ja estetyistä soluista. Edistyneemmässä versiossa soluissa voi olla painoarvoja, jotka edustavat hitaammin liikuttavissa olevaa aluetta, kuten mäkeä (Sturtevant 2014, 255). Hyvä esimerkki ruudukkoa käyttävästä pelistä on shakki, jossa nappulat liikkuvat kahden akselin suuntaisesti sekä vinosti.

### **A-Star**

A-Star, on vanha, 1968 kehitetty, mutta silti useimmissa tapauksissa tehokain ja optimaalisin, reitinhakualgoritmi (Mnemstudio 2012). A-Star on Dijkstran algoritmin ja Best-First-Search-algoritmin yhdistelmä (MIT Rajiv Eranki 2002). Dijkstran algoritmi löytää lyhyimmän reitin pisteiden välillä, mutta käy läpi suuren määrän vaihtoehtoja. Best-First-Search-algoritmi on nopea heuristisen arvioinnin ansiosta, mutta sen löytämä reitti ei ole kaikissa tapauksissa lyhyin (Stanford Amit's Thoughts on Pathfinding, 2009). Cryengin reitinhaku käyttää A-Star-algoritmia.

## 2.3 Tietoisuus

Jotta tekoäly pystyy tekemään ympäristöstä riippuvia päätöksiä, se tarvitsee jonkinlaisen tietoisuuden ja tapoja havainnoida. Yleisin ihmiseltä lainattu aisti peleissä on näkö. Muita havainnointitapoja voivat olla kuulo tai joissain tapauksissa jopa hajuaisti.

Aistien simulointi realistisesti olisi erittäin paljon suorituskykyä vaativaa, siksi tätä kierretään peleissä erilaisia tavoilla. Näköaistina voidaan käyttää yhtä tai useampaa sädettä kahden agentin välillä ja seurataan, osuuko säde esteeseen. Joskus pelaajaa saatetaan huijata ja pelaajan sijainti kerrotaan suoraan tekoälylle, vaikka näköyhteyttä ei välttämättä olisikaan.

## 3 CRYENGINE

Cryengine on saksalaisen Crytek-yhtiön kehittämä ja ylläpitämä pelimoottori. Tunnetuimpia Cryenginellä tehtyjä pelejä ovat Crysis-sarja, Ryse: Son of

Rome sekä Evolve. Cryengine on myös surullisen kuuluisa huonosta käyttäjävälisyydestään sekä puutteellisesta dokumentoinnista verrattuna suosittuihin pelimoottoreihin, kuten Unity ja Unreal Engine. Toisaalta Cryengine on myös usein kilpailijoihin suorituskkyisempi, mutta silti visuaalisesti tyrmäävä verrattuna kilpaileviin pelimoottoreihin (Mishra & Shrawankar 2016, 73 - 74).

### 3.1 Cryengininen tekoäly

Cryengininen tekoälyjärjestelmä on hyvin ammutapeleihin soveltuva. Järjestelmää tukevat XML-merkintäkielellä rakennetut behavior treet, Lua sekä pelimoottorin oma visuaalinen ohjelmointijärjestelmä FlowGraph. Ne sisältävät tähän tarkoitukseen soveltuvia funktioita. Toisaalta on myös otettava huomioon, että suuri osa tekoälyä käyttävistä peleistä on jonkinlaisia ammutapelejä. Vanhemmissa Cryengininen versioissa oli modular behavior treen sijaan vain behavior tree, joka ei ollut kykenevä keskeyttämään toimintaa tarvittaessa.

Cryenginissä on myös kohtalainen valikoima pelaajalle näkymättömiä objekteja. Ne helpottavat tekoällyn navigointia ja mahdollistavat eri toimintoja, kuten animaation ajamisen tai polun seuraamisen.

Agentit voidaan jakaa eri ryhmiin (Faction), esimerkiksi pelaajat, siviilit ja roistot. Ryhmien avulla määritellään ovatko agentit ystävällisiä, neutraaleja vai vihanielisiä keskenään. (Kuva 1)

```

<Factions>
  <Faction name="Players">
    <Reaction faction="Grunts" reaction="hostile" />
    <Reaction faction="Civilians" reaction="friendly" />
    <Reaction faction="Assassins" reaction="hostile" />
  </Faction>

  <Faction name="Civilians" default="neutral" />
  <Faction name="WildLife" default="neutral" />

  <Faction name="Grunts">
    <Reaction faction="Players" reaction="hostile" />
    <Reaction faction="Civilians" reaction="neutral" />
    <Reaction faction="Assassins" reaction="hostile" />
  </Faction>

  <Faction name="BlueTeam" default="neutral">
    <Reaction faction="RedTeam" reaction="hostile" />
  </Faction>

  <Faction name="RedTeam" default="neutral">
    <Reaction faction="BlueTeam" reaction="hostile" />
  </Faction>
</Factions>

```

Kuva 1. Ryhmien (Faction) määrittely

## 3.2 MBT

Modular Behavior Tree, eli ”modulaarinen käyttäytymispuu”, on XML-elementteistä koostuva tietorakenne. MBT mahdollistaa korkean tason käyttäytymisen luomisen ilman monimutkaista koodia monikäyttöisellä ohjelmointikielellä, kuten C++ (Cryengine MBT-dokumentaatio).

Cryenginessä ei ole MBT-työkalua tai -editoria. Aikaisemmin sellainen oli, mutta se poistettiin vuonna 2015, eikä uutta ole vielä julkaistu. Koska työkalua ei ole, MBT-editointi tehdään alusta lähtien tekstieditorilla. Tekstieditorilla isompien kokonaisuuksien hahmottaminen voi olla haastavaa verrattuna graafiseen editoriin.

Kuvan 2 yksinkertainen MBT hakee satunnaisen pisteen kuvassa 3 esitetyn TPS-kyselyn avulla, jonka jälkeen AI kävelee pisteeseen, odottaa yhden sekunnin ja toistaa saman uudelleen.

Oleellinen XML-elementti kuvassa 2 on *Loop*, jonka avulla logiikkaa toistetaan äärettömästi. Kuvassa 2 rivillä 19 *QueryTPS* hakee joka kerta uuden referenssipisteen. *Move*-elementillä tekoäly siirtyy ”to”-parametriin asetettuun referenssi pisteeseen. Lisäksi *Move* elementissä ovat parametrit *speed*,

*stance* ja *firemode*, joilla asetetaan tekoäly liikkumaan kävelyvauhtia rentoutuneesti. Näiden jälkeen on rivillä 22 *Wait*, jonka jälkeen aloitetaan alusta.

```
<BehaviorTree>
  <Variables>
    <Variable name="ExecuteSequence" />
    <Variable name="ExecuteInterruptibleSequence" />
  </Variables>

  <SignalVariables>
  </SignalVariables>

  <Timestamps>
  </Timestamps>

  <Root>
    <BehaviorTree>
      <Loop>
        <Sequence>
          <QueryTPS name="test_randomposition" register="RefPoint"/>
          <Move to="RefPoint" speed="Walk" stance="Relaxed"
            firemode="off"/>
          <Wait duration="1.0"/>
        </Sequence>
      </Loop>
    </BehaviorTree>
  </Root>
</BehaviorTree>
```

Kuva 2. Yksinkertainen MBT

### 3.3 TPS-kysely

Tactical Point System Queryn, lyhyemmin TPS-kyselyn, avulla AI hakee itselleen suotuisia pisteitä, joihin voidaan liikkua. TPS-kyselylle on vaikea keksiä hyödyllistä käyttöä taistelupelien ulkopuolella, jonka vuoksi sitä käytettiin tässäkin projektissa vain kokeilumielessä. Ammuntapeleissä AI voi hakeutua TPS-kyselyn avulla kentän tekijän määrittämiin pisteisiin tai jopa maaston muodon perusteella määrittyviin paikkoihin. Kyselyllä haettu piste voidaan painottaa, niin että kysely suosii pisteitä lähempänä tai kauempana pelaajaa, vihollista tai muuta referenssipistettä. (Cryengine TPS-dokumentaatio 2014) TPS-kyselyitä voi suorittaa Cryengin Lua- ja C++ -rajapintojen kautta.

Kuvan 3 Query luo hahmon ympärille ruudukon pisteitä kahden metrin välein ja enintään 18 metrin etäisyydellä hahmosta. Näistä pisteistä valitaan satunnaisesti yksi, jonka kysely palauttaa.

```

AI.RegisterTacticalPointQuery({
    Name = "test_randomposition",
    {
        Parameters =
        {
            density = 2
        },
        Generation =
        {
            grid_around_puppet = 18.0
        },
        Conditions =
        {
            isInNavigationMesh = true
        },
        Weights =
        {
            random = 1.0
        },
    },
});

```

Kuva 3. Lähdekoodiin lisätty uusi Lua TPS kysely

### 3.4 Lua ja ScriptBind

Lua-skriptien kirjoittaminen on nopeampaa kuin C++:n, sillä Lua-koodin voi ladata ajon aikana, eikä koko pelin koodia joudu kääntämään uudelleen jokaisen muutoksen jälkeen (Dawe 2014. 94). ScriptBind on Cryengin rajapinta C++ -lähdekoodin ja Lua-koodin välillä. ScriptBind mahdollistaa monipuolisen pelimoottorin käsittelyn ja sen avulla voi myös manipuloida MBT arvoja Lua-koodia käyttäen. (Mastering CryENGINE. 2014)

### 3.5 Signal Reference

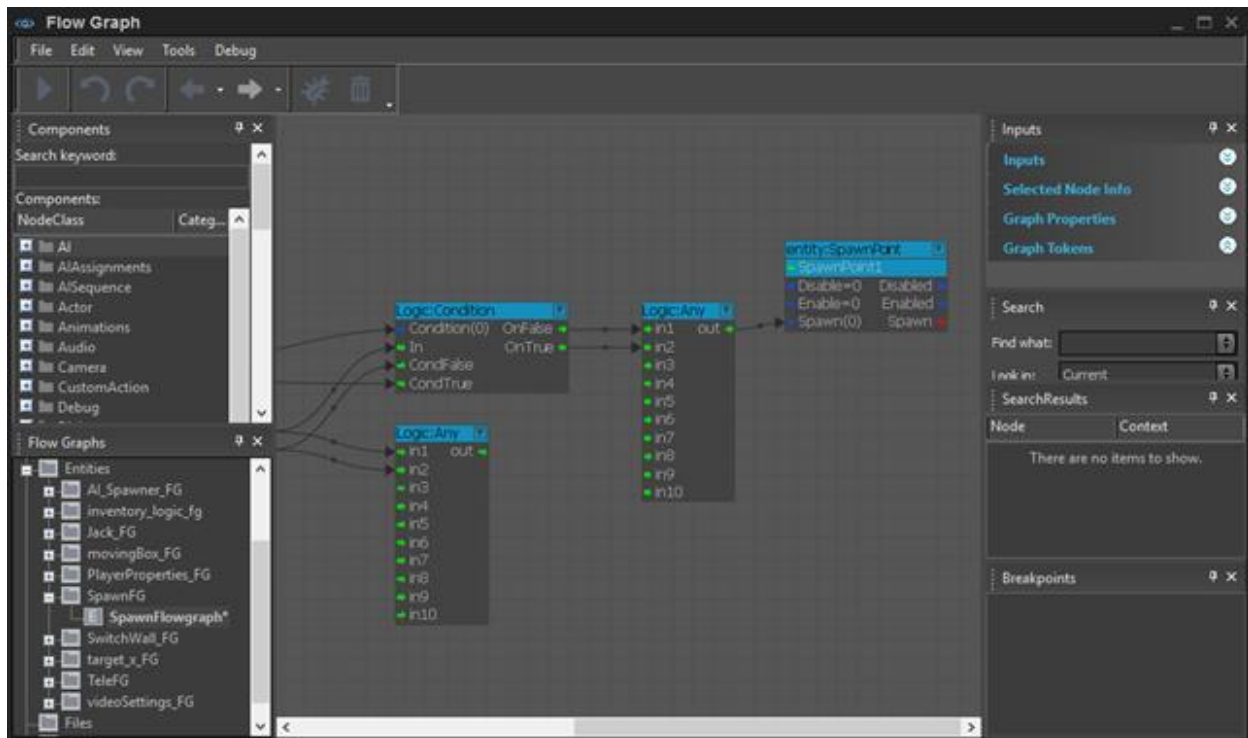
Signal Reference on ryhmä valmiiksi määritellyjä havaintokykyjä. Signal Reference viittausten avulla voidaan asettaa MBT aikaleima tai muuttujan arvo. Signaalit asetetaan XML-tiedoston alussa *SignalVariables*- ja *Timestamps*-ryhmissä riippuen siitä, kumpaan tarkoitukseen signaalia halutaan käyttää. (Cryengine SignalReference dokumentaatio. 2013)

Taulukko 1. Signal Reference

<b>Havainto</b>	<b>Havaintoon liittyvien signaalien kuvaukset</b>
No Target	Kun huomionkohde on menetetty.
Sound	Epäilyttävä, uhkaava tai mielenkiintoinen ääni, mutta kohde ei ole nähtävissä.
Visual	Uhka, vihollinen tai esine näkökentässä.
Awareness of Player	Pelaaja on lähellä tai kulkee poispäin. Pelaaja katsoo kohti tai poispäin.
Awareness of Attention Target	Kohde lähestyy tai pakenee. Uusi huomionkohde. Ei kohdetta näkyvissä.
Weapon Damage	Vahingon ottaminen aseeseen aiheuttamana.
Pathfinding	Reitti päämäärään löytyy tai ei löydy.

### 3.6 Flow Graph

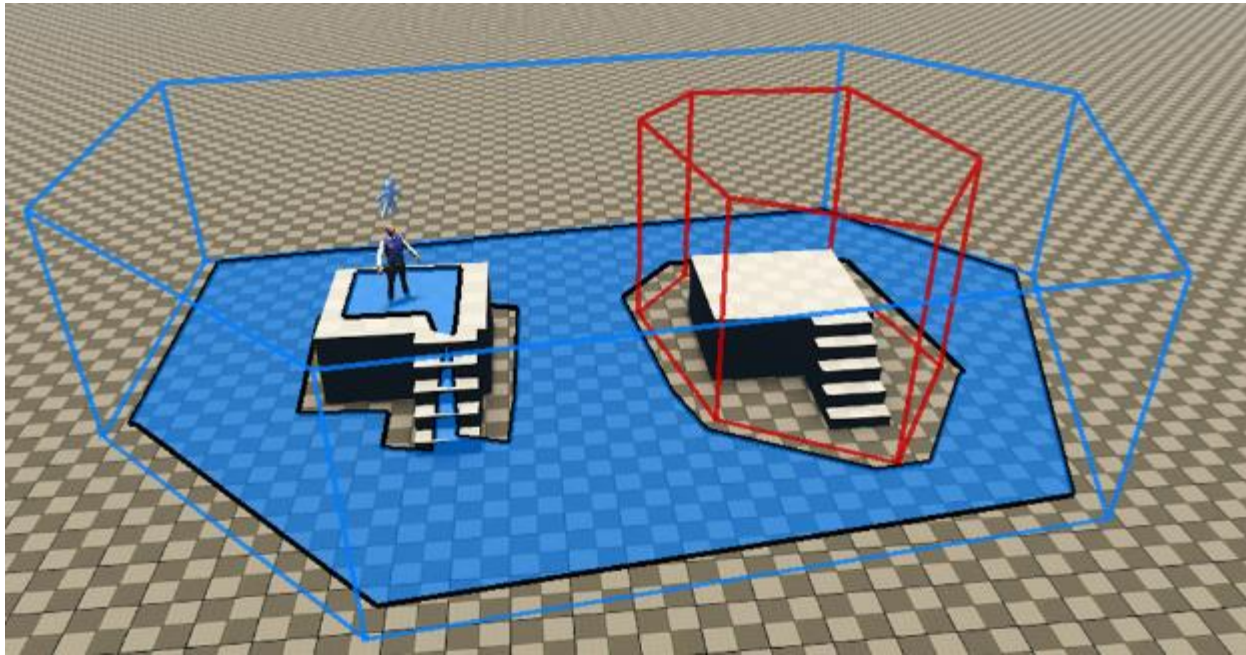
Cryengininen Flowgraph on visuaalinen ohjelmointijärjestelmä, jota vastaava on kilpailevan pelimoottorin Unreal Enginen Blueprint-järjestelmä. Flowgraphilla ohjelmointi on käytännössä logiikkarakenteiden muodostamista Flownodeksi kutsuttuja osia yhdiställä. Flowgraphin avulla voidaan muodostaa nopeita prototyyppijä tai yksinkertaisia pelin perusmekaniikkoja. Suurempienkin kokonaisuuksien ohjelmointi Flowgraphin avulla on mahdollista, mutta rakenteesta voi helposti tulla liian sekava suuren Flownode-määrän vuoksi. (Kuva 4)



Kuva 4. Flow Graph muokkaustyökalun perusnäkyä

### 3.7 Navigation Mesh Cryenginissä

Yksinkertainen tapa agenttien navigaation toteuttamiseksi on navigation mesh. Editorilla voidaan määrittää alue, jonka sisälle pelimoottori laskee navigaatioverkoston. Aluetta pystyy myös rajaamaan tarpeen mukaan. Kuvassa 5 on esitetty navigation mesh -esimerkki, jossa sininen alue on navigoitavissa ja punaisella on rajattu pois estetty alue. Useimmiten pois halutaan rajata alueita, joissa on hyvin epätasainen maasto, tai jotakin muuta mikä haittaa agenttien etenemistä.



Kuva 5. Navigation Mesh esimerkki

Navigation meshin generointiin pystyy myös vaikuttamaan muuttamalla konfiguraatiotiedostoa. Konfiguroinnilla voidaan muuttaa, kuinka kaukaa seinistä navigaatio alue alkaa, tai kuinka korkeista aukoista voi kulkea. Mikäli pelissä on erikokoisia agenteja, voidaan niille määrittää omat, erilaiset, navigointi alueet (kuvassa 6 *MediumSizedCharacters* ja *VehicleMedium*).

```
<Navigation version="6" >
  <AgentTypes>

    <AgentType   name="MediumSizedCharacters"
                 voxelSize="0.125, 0.125, 0.125"
                 radius="4"
                 height="16"
                 climbableHeight="3"
                 maxWaterDepth="8" >
      <SmartObjectUserClasses>
        <class name="Human" />
      </SmartObjectUserClasses>
    </AgentType>

    <AgentType name="VehicleMedium"
                 voxelSize="0.20, 0.20, 0.20"
                 radius="14"
                 height="10"
                 climbableHeight="4"
                 maxWaterDepth="5" >
    </AgentType>

  </AgentTypes>
</Navigation>
```

Kuva 6. Navigation Mesh konfiguraatiotiedosto



## 4 TOTEUTUS

Projekti tehtiin Cryengine V:llä GameSDK-projektipohjaa käyttäen. Tavoitteena oli luoda simulaatio, jossa tietokone pelaa lipunryöstöä itseään vastaan. Vaatimukset tämän saavuttamiseksi olivat: kenttä, jossa hahmot voivat liikkua, mekaniikat lipun siirtämiseen sekä tekoäly, joka tietää mihin mennä ja mitä tehdä.

Toteutus sisältää XML-merkintäkielellä kirjoitetun MBT-rakenteen, mallin TPS-kyselyllä haetusta sijainnista ja Flow Graph -järjestelmällä ohjattuja tilan muutoksia. MBT rakenteen korvaava toiminta olisi voitu tehdä myös Flow Graph -logiikoiden avulla, mutta lopputulos olisi ollut äärimmäisen epämiellyttävä kehittämisen ja jatkokehittämisen kannalta. Flow Graph -komennoilla tehtiin lipunryöstö -mekaniikkoja, kuten lipun ottaminen vastustajajoukkueen puolelta. Eri joukkueet erotettiin Faction-järjestelmää käyttäen, jonka avulla ne voidaan tunnistaa eri pelimekaniikkoja varten. Navigaatio toteutettiin käyttäen navigation mesh -järjestelmää ja tagpoint pisteitä.

### 4.1 Suunnitelma

Suunnitteluvaiheessa hahmoteltiin behavior tree jota AI tulisi käyttämään. Hahmottelun aikana tutkittiin GameSDK pohjaan sisällettyä SDK\_Grunt behavior tree -rakennetta sekä etsittiin virallisesta dokumentaatiosta soveltuvia solmukkeita (Cryengine MTB node dokumentaatio).

Ennen varsinaisen projektin aloitusta tehtiin myös Flow Graph -prototyyppejä. Prototyypeillä kokeiltiin muun muassa, kuinka AI saadaan seuraamaan ennalta määritettyä reittiä tai kulkemaan tiettyyn pisteeseen reitinhaun avulla.

### 4.2 GameSDK projektin pohjana

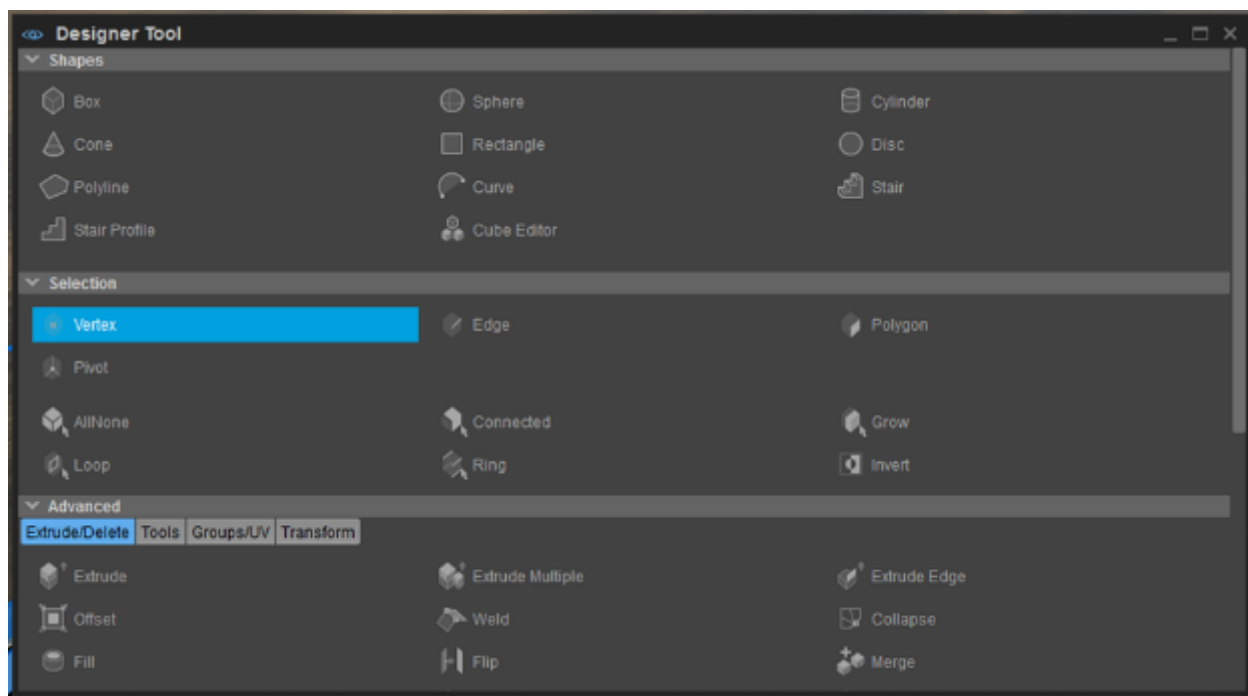
GameSDK on Cryengininen näyteprojekti, jota kehittäjät voivat käyttää apuna omaa projektia tehdessä. GameSDK sisältää useiden 3D-mallien ja tekstuurien lisäksi paljon oleellisia kooditiedostoja. GameSDK-pohjassa on käytetty Scaleform GfX -väliohjelmistolla tehty flashiin perustuva graafinen käyttöliittymä ja päävalikko. Niiden muokkaamiseen vaaditaan Autodeskin maksullinen

Scaleform-ohjelma. Kaiken tämän lisäksi mukana on kaksi esimerkkikenttää, joiden on tarkoitus olla apuna uusien kenttien kehittämiseen. Tässä projektissa edellä mainittu sisältö karsittiin pois tilan säästämiseksi ja jäljelle jäi GameSDK-lähdekoodi.

### 4.3 Versionhallinta ja ohjelmisto

Pääasiallisesti työtä tehtiin Cryenginellä ja Notepad++ -ohjelmalla. Cryengin versio projektia aloittaessa oli 5.1, mutta se päivitettiin 5.2.1-versioon. Tästä aiheutui päänvaivaa väliaikaisesti, sillä GameSDK on osin riippuvainen pelimoottorin omista tiedostoista, jotka olivat muuttuneet versioiden välillä. Ongelma saatiin korjattua kopiaamalla manuaalisesti oikeat tiedostot paikasta toiseen.

Mikäli projektissa olisi käytetty omia 3D-malleja, olisi siihen tarvinnut 3ds Maxia. Ajan ja taidon puutteen vuoksi käytettiin kuitenkin GameSDK:n mukana olevia hahmoja, sekä tarpeen mukaan mallinnettiin Cryengin sisäisellä Designer Tool työkalulla (Kuva 7) suhteellisen yksinkertaisia elementtejä, kuten silta ja seiniä.

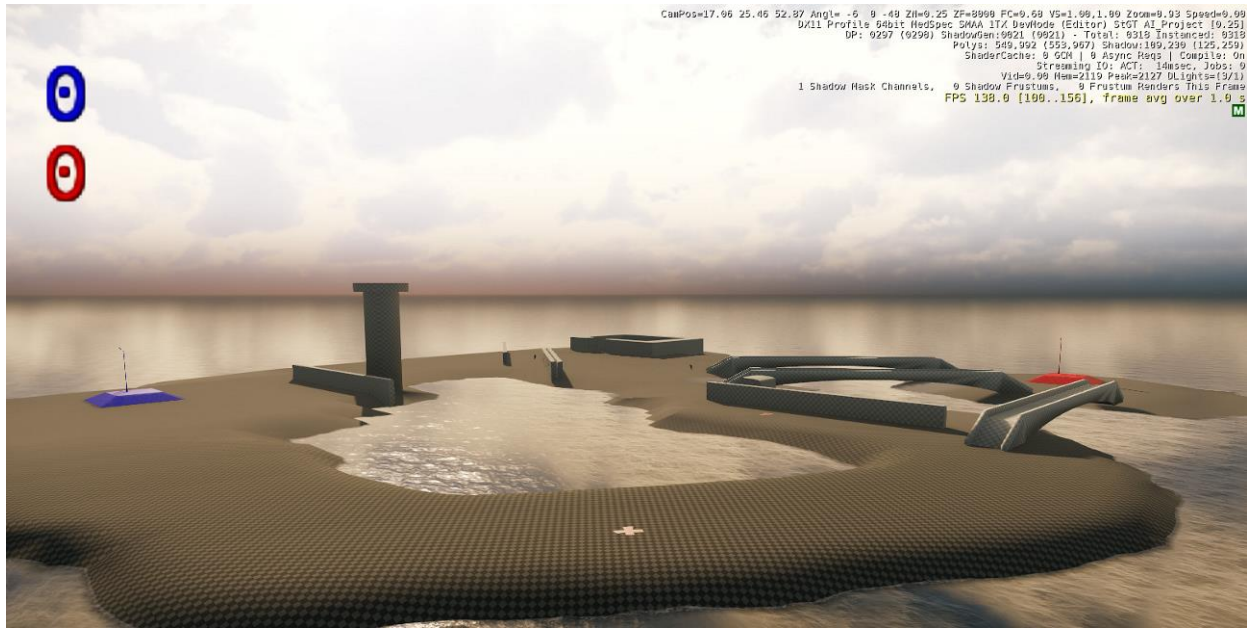


Kuva 7. Designer Tool perusnäkö

Versionhallinnassa käytettiin TortoiseSVN-ohjelmaa, jonka *repository* kansio oli Google Drivessä. Tällä yhdistelmällä projektin viimeisemmän version saa käyttöön millä tahansa tietokoneella, johon on asennettu nämä kaksi ohjelmaa, eikä muita ulkoisia palvelimia tarvita. Versionhallinnan käyttö on kätevää jos työskentely tapahtuu aina samoilla tietokoneilla, jolloin versionhallintaohjelmaa ei tarvitse joka kerta asentaa uudelleen. Esimerkiksi muistitikun käyttö projektin säilytyspaikkana voi olla riskialtista, sillä muistitikku voi kadota tai hajota. Suurin etu TortoiseSVN-ohjelmalla on mahdollisen virheen tai ongelmalanteen sattuessa aiempaan versioon palaaminen, tai nykyisen ja vanhemman version vertailu keskenään. Ainoa ongelma oli Google Drive -pilvipalvelun kanssa, joka saattoi kaatua uutta versiota tallentaessa, mikäli sitä ei pysäytetty siksi aikaa. Vaihtoehtoisen pilvipalvelun, Dropboxin, kanssa tätä ongelmaa ei olisi ollut, mutta työn aikana päädyttiin kuitenkin käyttämään Google Drive -pilvipalvelua sen suuremman ilmaisen tallennustilan vuoksi.

#### 4.4 Ympäristö

Projektia varten luotiin yksinkertainen ympäristö, johon käytettiin minimaalisesti aikaa, sillä graafinen ulkoasu ei ollut olennainen projektin kannalta. Kuva 8 on yläviistosta kuvattu näkymä alueesta. Kuvan oikeassa laidassa on punaisen joukkueen ja vasemmassa sinisen joukkueen alue. Kuvassa näkyy myös yksinkertain graafinen käyttöliittymä, joka esittää joukkueiden pisteet ruudun vasemmassa yläkulmassa.



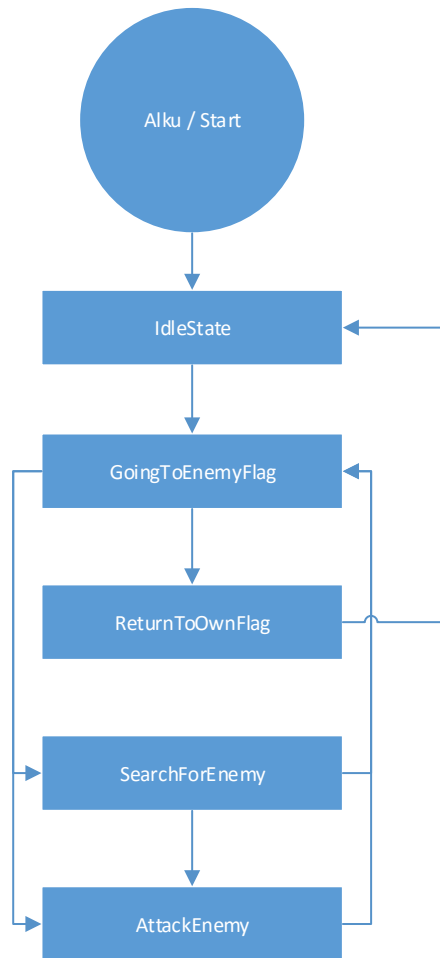
Kuva 8. Pelialue

## 4.5 Hahmot ja behavior treet

Hahmoina käytettiin GameSDK *grunt* -hahmoja ja animaatioita. Hahmojen takkien tekstuurin värit vaihdettiin vastaamaan joukkueen väriä. Hahmojen näkökenttää laajennettiin 80 asteesta 120 asteeseen, muuta sen etäisyys pienennettiin 70 metristä 30 metriin, jotta hahmot eivät näe vastustajia kovin kaukaa. Näitä muutoksia lukuun ottamatta hahmot ovat alkuperäiskuntoisia.

### 4.5.1 MBT projektissa

Hahmoille tehtiin lipunryöstö- behavior tree, jossa on tilakone viidellä tilalla. Tilat ovat *IdleState*, *GoingToEnemyFlag*, *ReturningToOwnFlag*, *SearchForEnemy* ja *AttackEnemy* (Kuva 9). Transitiot eri tilojen välillä tapahtuvat joko Flow Graph -komennolla tai tilan itsensä kutsumalla signaalilla.



Kuva 9. Pelkistetty hahmotelma MBT logiikan toiminnasta

Tekoäly liikkuu referenssi pisteisiin, jotka asetetaan Lua-koodin avulla *combat-Move-* tai *holdPosition* -välimuuttujista. Välimuuttujien arvot asetetaan Flow Graph -käskyllä hakemalla kenttään sijoitettujen *tagPoint*-pisteiden sijainti kolmelukuisena vektorina.

IdleState on ensimmäinen tila, johon lipunryöstö-MBT etenee (Kuva 10). Tilaa käytettiin työn alussa erilaisiin kokeiluihin. Yksi kokeiluista oli TPS-kysely, jonka seurauksena tilaan jätettiin QueryTPS-node ilman varsinaista syytä. Kokonaisuudessaan tilan aikana AI liikkuu lyhyen matkan, odottaa puolitoista sekuntia ja siirtyy seuraavaan tilaan nimeltä "GoingToEnemyFlag".

```

<State name="IdleState">
  <Transitions>
    <Transition to="GoingToEnemyFlag" onEvent="goToEnmFlag"/>
  </Transitions>

  <BehaviorTree>
    <Loop>
      <Sequence>
        <QueryTPS name="test_randomposition" register="RefPoint"/>
        <Move to="RefPoint" speed="run" stance="Relaxed" firemode="off"/>
        <Wait duration="1.5"/>

        <Signal name="ResetPassedMidwaypoint" />

        <SendTransitionSignal name="goToEnmFlag"/>
      </Sequence>
    </Loop>
  </BehaviorTree>
</State>

```

Kuva 10. MBT tilakoneen IdleState niminen tila

GoingToEnemyFlag-tilan aikana AI kulkee combatMove- ja holdPosition-pisteisiin. Pisteet haetaan Lua-koodin avulla. HoldPosition-pisteen luona Flow Graph -järjestelmä lähettää ACT\_ALERTED siirtymäsignaalin, mikäli lippu on otettavissa. ACT\_ALERTED signaalilla tilakone siirtyy ReturningToOwnFlag tilaan. Jos lippua ei ole, ACT\_ALERTED signaalia ei tule, jolloin NoFlagHere siirtymä vaihtaa tilaksi SearchForEnemy. (Kuva 11).

```

<State name="GoingToEnemyFlag">
  <Transitions>
    <Transition to="AttackEnemy" onEvent="OnEnemySeen"/>
    <Transition to="AttackEnemy" onEvent="OnNewAttentionTarget"/>
    <Transition to="AttackEnemy" onEvent="OnBulletRain"/>
    <Transition to="SearchForEnemy" onEvent="NoFlagHere"/>
    <Transition to="ReturningToOwnFlag" onEvent="ACT_ALERTED"/><!-- took
enemy flag, called with FG -->
  </Transitions>
  <BehaviorTree>

    <Sequence>

      <Bubble message="going to enemy flagpoint" duration="4"/>
      <SuppressFailure>
        <IfCondition condition="HasNotPassedMidwaypoint">
          <Sequence>

            <ExecuteLua code="AI.SetRefPointPosition(entity.id,
entity.AI.combatMove.position) " /><!-- midwaypoint
position -->
            <Move to="RefPoint" speed="Sprint" stance="Relaxed "
firemode="off"/>
            <Signal name="PassedMidwaypoint" />

          </Sequence>
        </IfCondition>
      </SuppressFailure>

      <ExecuteLua code="AI.SetRefPointPosition(entity.id,
entity.AI.holdPosition.position) " /><!-- enemy flag pos -->
      <Move to="RefPoint" speed="Sprint" stance="Relaxed " firemode="off"
stopWithinDistance="1"/>
      <Wait duration="1.5"/>

      <!-- this is reached if AI is at point and flowgraph has not called
ACT_ALERTED -->
      <SendTransitionSignal name="NoFlagHere"/>

    </Sequence>

  </BehaviorTree>
</State>

```

Kuva 11. MBT tilakoneen GoingToEnemyFlag niminen tila

ReturningToOwnFlag tilan aikana AI palaa välietapin kautta omalle lippupisteelle välittämättä vihollisjoukkueesta. Tämän tilan aikana käytetty liikkumisnopeus on *Run* eikä *Sprint*, kuten muissa tiloissa, eli AI liikkuu hieman hitaammin lipun kanssa. Tilan lopussa tilakone siirtyy IdleState tilaan ja lipunhakusilmukka alkaa alusta. (Kuva 12).

```

<State name="ReturningToOwnFlag">
  <Transitions>
    <Transition to="IdleState" onEvent="FlagReturned"/>
  </Transitions>
  <BehaviorTree>

    <Sequence>
      <Bubble message="returning to own flagpoint" duration="4"/>

      ExecuteLua code="AI.SetRefPointPosition(entity.id,
entity.AI.combatMove.position) " /><!-- midwaypoint position -->
      <Move to="RefPoint" speed="Run" stance="Relaxed " firemode="off"/>

      <ExecuteLua code="AI.SetRefPointPosition(entity.id,
entity.AI.holdPosition.position) " /><!-- own flag pos -->
      <Move to="RefPoint" speed="Run" stance="Relaxed " firemode="off"
stopWithinDistance="1"/>

      <Signal name="ResetPassedMidwaypoint" />
      <SendTransitionSignal name="FlagReturned"/>

    </Sequence>

  </BehaviorTree>
</State>

```

Kuva 12. MBT tilakoneen ReturningToOwnFlag niminen tila

Tilakoneessa on kaksi tilaa, jotka eivät liity lipun hakemiseen, vaan vihollisjoukkueen estämiseen. SearchForEnemy tilan (Kuva 13) aikana AI käy puoli-matkapisteellä etsimässä vihollista ja AttackEnemy tilassa (Kuva 14) AI pyrkii 10 metrin päähän vihollisesta jonka jälkeen hyökätään.



```

<State name="SearchForEnemy">
  <Transitions>
    <Transition to="AttackEnemy" onEvent="OnEnemySeen"/>
    <Transition to="AttackEnemy" onEvent="OnNewAttentionTarget"/>
    <Transition to="AttackEnemy" onEvent="OnBulletRain"/>
    <Transition to="GoingToEnemyFlag" onEvent="DoneSearching"/>
    <Transition to="ReturningToOwnFlag" onEvent="ACT_ALERTED"/><!-- took
enemy flag, called with FG -->
  </Transitions>
  <BehaviorTree>

    <Sequence>
      <Signal name="ResetPassedMidwaypoint" />

      <Bubble message="Searching for enemies" duration="4"/>

      <ExecuteLua code="AI.SetRefPointPosition(entity.id,
entity.AI.combatMove.position)" /><!-- midwaypoint position -->
      <Move to="RefPoint" speed="Sprint" stance="Relaxed "
firemode="off"/>

      <Wait duration="2.5"/>

      <QueryTPS name="test_randomposition" register="RefPoint"/>
      <Move to="RefPoint" speed="run" stance="Relaxed" firemode="off"/>
      <Wait duration="1.5"/>

      <!-- end searching -->
      <SendTransitionSignal name="DoneSearching"/>

    </Sequence>

  </BehaviorTree>
</State>

```

Kuva 13. MBT tilakoneen SearchForEnemy niminen tila

```

<State name="AttackEnemy">
    <Transitions>
        <Transition to="GoingToEnemyFlag" onEvent="StopAttacking"/>
        <Transition to="ReturningToOwnFlag" onEvent="ACT_ALERTED"/><!-- took
enemy flag, called with FG -->
    </Transitions>
    <BehaviorTree>
        <Sequence>
            <SetAlertness value="2"/>
            <Stance name="Alerted"/>
            <ExecuteLua code="entity:SelectPrimaryWeapon()"/>
        </Sequence>
        <Loop>
            <Sequence>
                <SuppressFailure>
                    <Move to="Target" speed="Run" stance="Stand"
                    avoidDangers="0" stopWithinDistance="10"/>
                </SuppressFailure>
                <Shoot at="Target" fireMode="Burst" stance="Stand"
                duration="2.0"/>
                <!-- keep fighting or transition back to "GoingToEnemyFlag" -->
                <Selector>
                    <AssertCondition condition="HasTarget"/>
                    <SendTransitionSignal name="StopAttacking"/>
                </Selector>
            </Sequence>
        </Loop>
    </BehaviorTree>
</State>

```

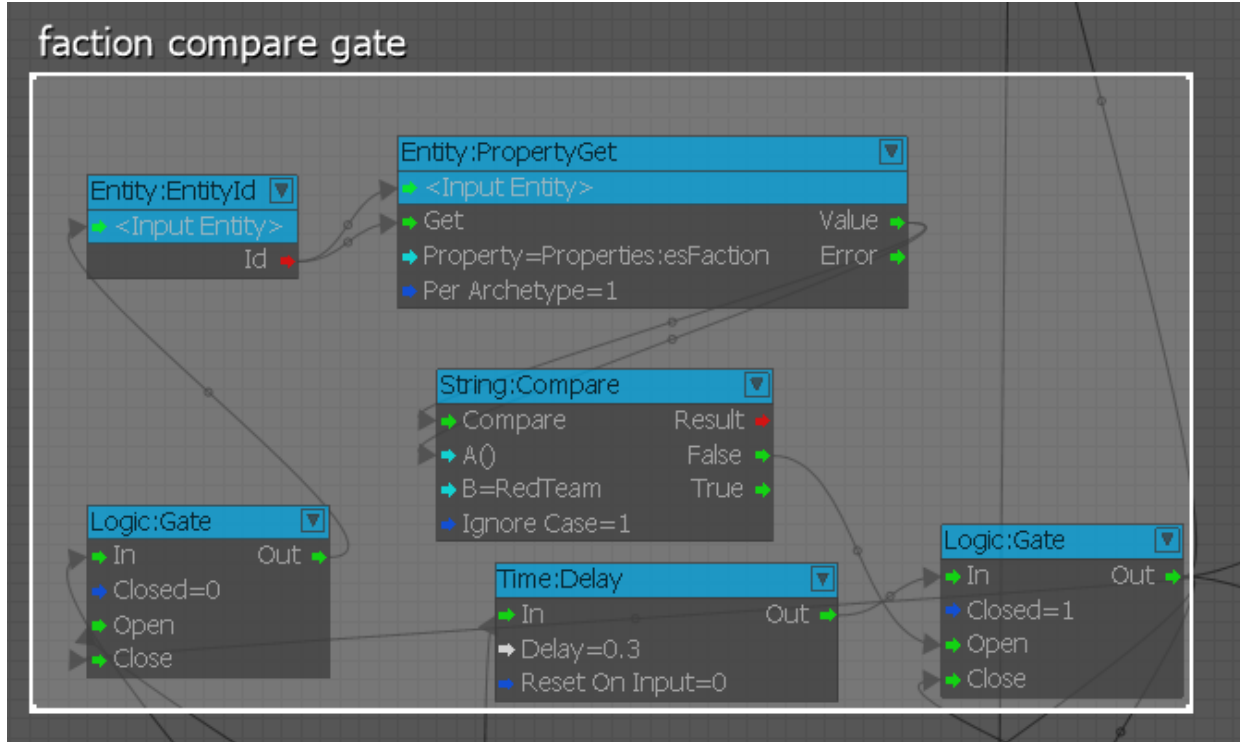
Kuva 14. MBT tilakoneen AttackEnemy niminen tila

## 4.5.2 Flow Graph

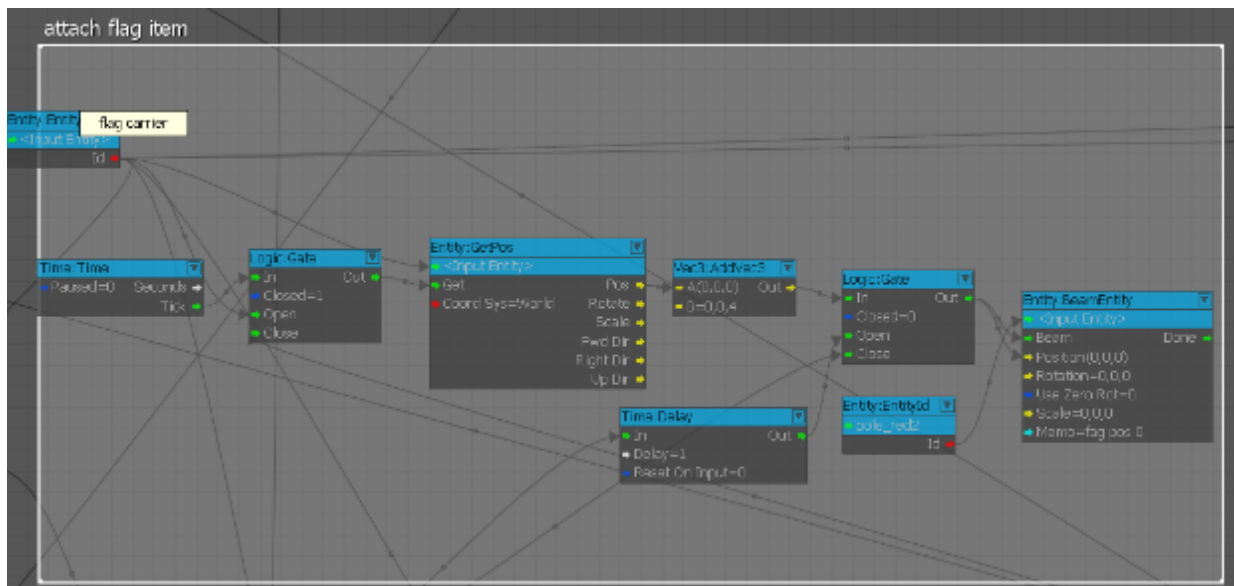
Työssä toteutettiin Flow Graph -järjestelmää käyttäen lipunryöstön sekä teko-  
 älyn ydinmekaniikkoja. Olennaisten funktioiden lisäksi Flow Graph -komen-  
 noilla tehtiin huomaamattomampia taustatehtäviä, kuten GameSDK alkuperäi-  
 sen graafisen käyttöliittymän piilottaminen ja *motion blur* asetuksen käytöstä  
 poistaminen.

Kun AI saapuu lipun alueelle, sen joukkue tarkistetaan (Kuva 15). Mikäli ky-  
 seessä on vastapuolen joukkue, suoritetaan lisätoimenpiteitä, kuten lipun lii-  
 kuttaminen (Kuva 16). Kuvassa 16 Time:Time Flow node kutsuu sitä seuraa-

via nodeja joka kerta kun ruutu päivittyy. Jokaisella ruudunpäivityksellä haetaan lippua kantavan hahmon sijainti ja siirretään lippu neljä mittayksikköä hahmon yläpuolelle BeamEntity Flow noden avulla.



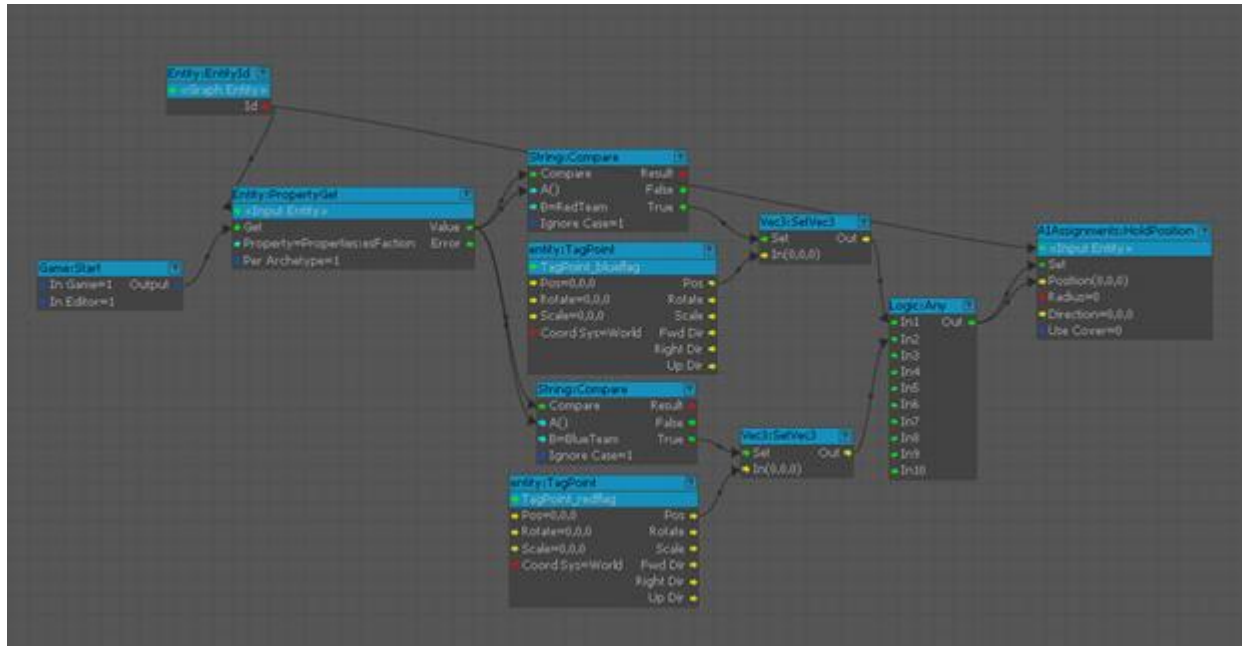
Kuva 15. Flow Graph järjestelmällä agentin joukkueen tarkistus lipuin luona



Kuva 16. Lipun liikuttaminen hahmon mukana Flow Graph -logiikka

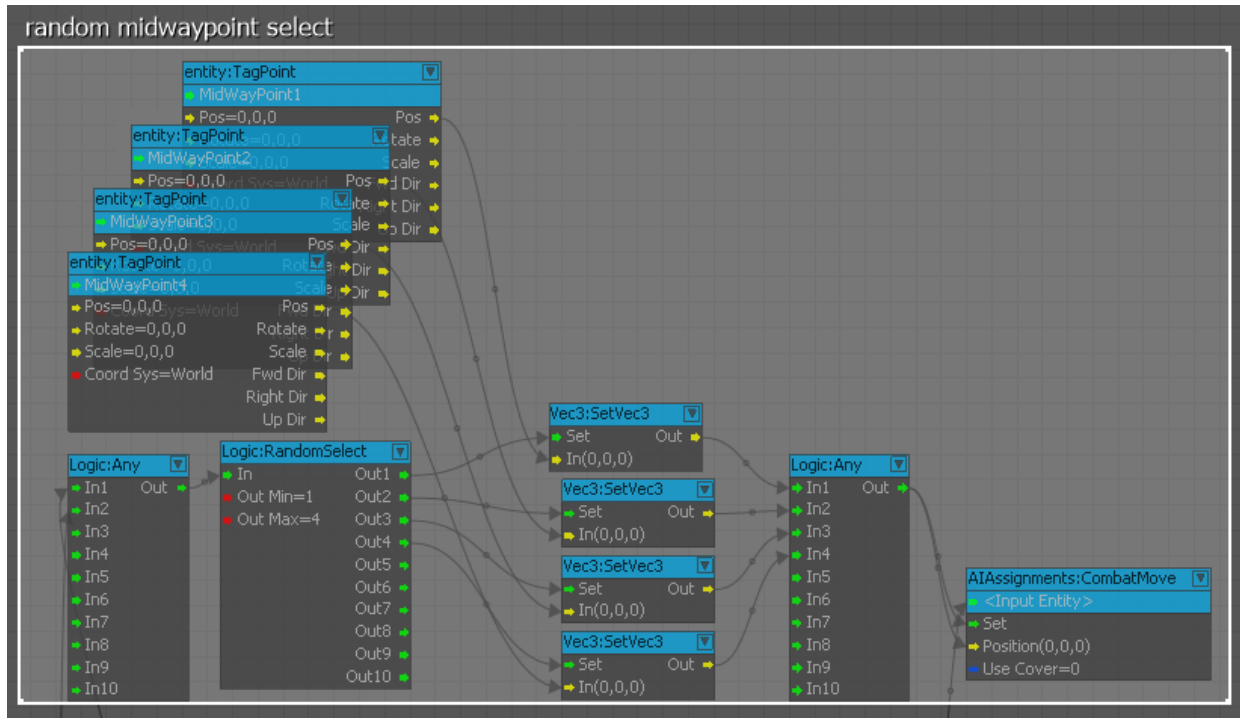
MBT vaatii toimiakseen holdPosition- ja combatMove-muuttujat heti pelin alussa (kuvat 17 ja 18). Kuvassa 17 asetetaan joukkueen perusteella holdPosition muuttuja, jota AI käyttää lipun sijaintina. Nämä molemmat muuttujat ovat

GameSDK lähdekoodista peräisin. Projektia olisi helpottanut, jos sijaintina käytettäviä muuttujia olisi ollut enemmän kuin kaksi, mutta muuttujien lisääminen osoittautui hyvin monimutkaiseksi. Jotta muuttujat olisi saatu toimimaan MBT- ja Flow Graph ympäristöissä, lähdekoodiin olisi vaadittu paljon muutoksia. Tämän vuoksi päädyttiin käyttämään vain valmiiksi olemassa olleita muuttujia.



Kuva 17. Flow Graph holdPosition pisteiden asettaminen

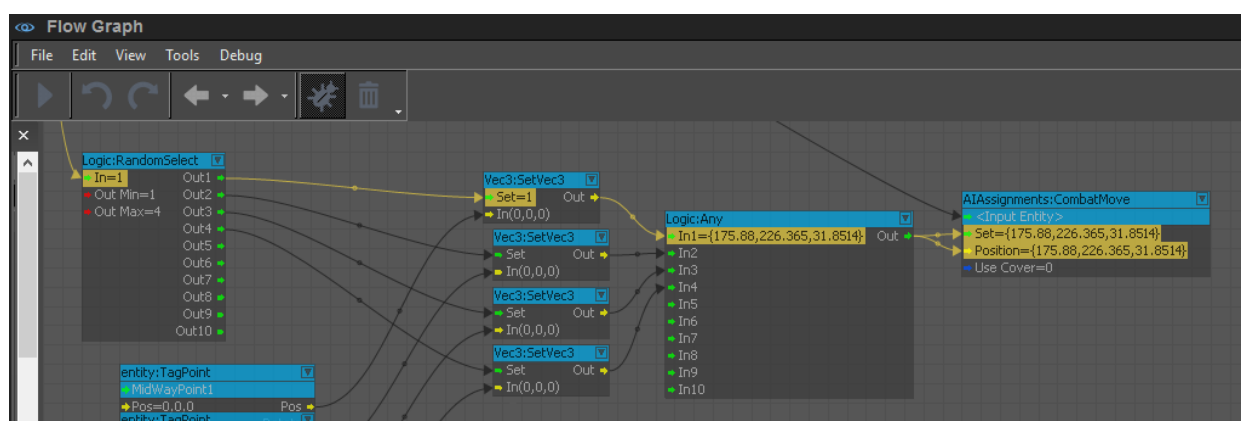
Jokaiselle hahmolle asetetaan alussa välietappipiste, jonka kautta se kulkee lipulle. Piste asetetaan uudelleen joka kerta kun AI vaihtaa päämäärää. Kuvassa 18 RandomSelect Flow node valitsee satunnaisen SetVec3 Flow noden, jolla välietapin sijainti asetetaan CombatMove-nimiseen muuttujaan. Välietapit ovat tärkeässä osassa kokonaisuutta, sillä ilman niitä AI kulkisi joka kerta lyhintä reittiä lippujen välillä. Koska lyhin reitti ei muutu ikinä, joukkueet tulisivat aina toisiaan vastaan, eikä kukaan pääsisi vastakkaiselle puolelle. Välietappien sopivaksi määräksi arvioitiin neljä kentän ja joukkueiden kokojen perusteella.



Kuva 18. Satunnaisen välietappipisteen asettaminen Flow Graph -järjestelmällä

#### 4.6 Virheidenetsintä

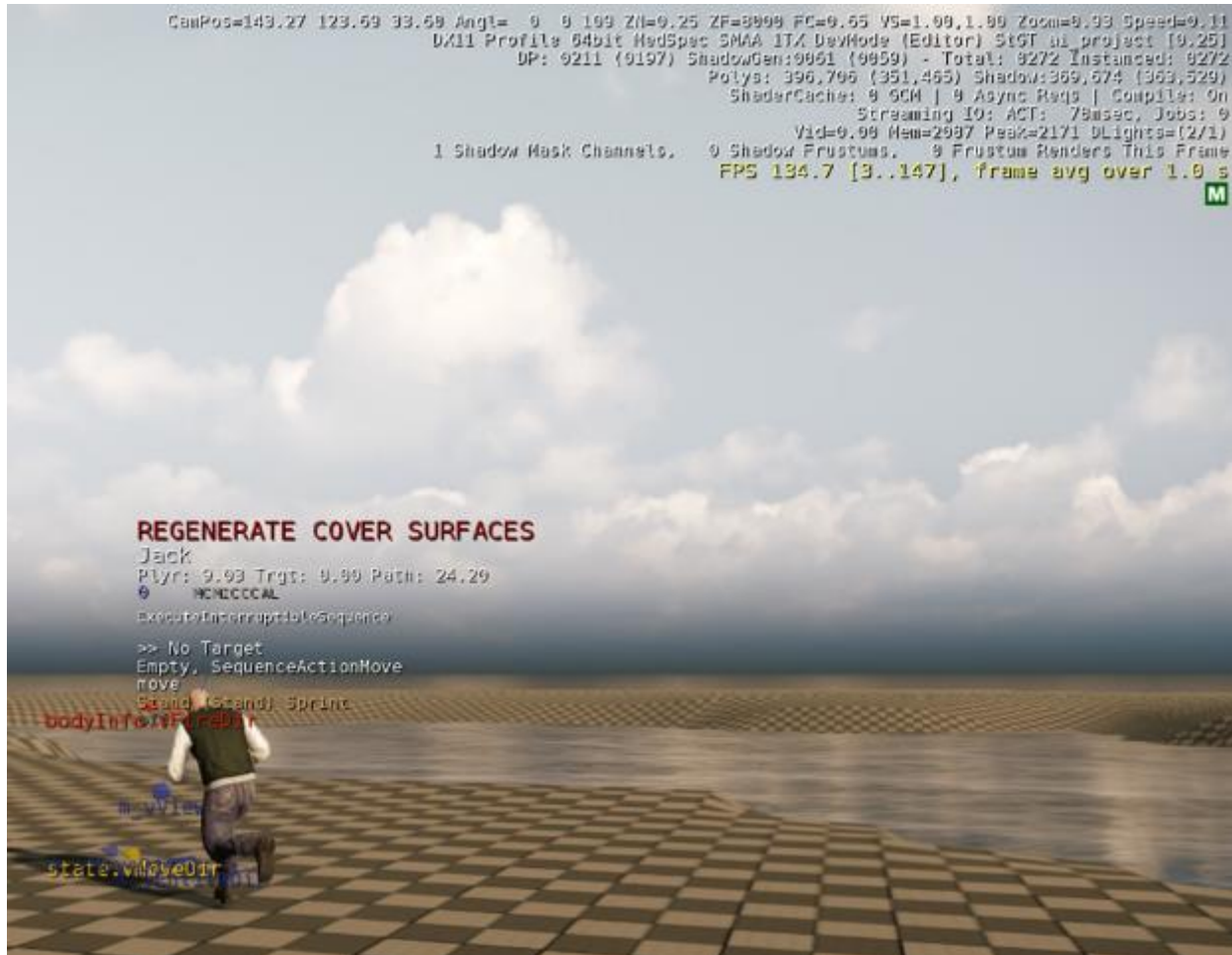
Flow Graph työkalu on mahdollista laittaa *debug*-tilaan, jolloin työkalu piirtää keltaisella logiikan kulkeman reitin ja näyttää siinä käytettyjen muuttujien arvot (Kuva 19). Flow Graph virheiden etsintä auttaa lähinnä logiikkavirheiden havaitsemisessa. Mikäli Flow Graph sisältää syntaksivirheen, pelimoottori useimmiten kaatuu.



Kuva 19. Flow Graph virheidenetsintä

Virheidenetsintä, eli "debuggaus", on Cryenginissä suhteellisen ongelmattonta. Editorissa on konsoli, joka varoittaa mahdollisista virheistä. Pelitilassa on toinen konsoli, jonka avulla voi tarpeen vaatiessa tarkistaa tai muuttaa eri

muuttujien arvoja. Kuvassa 21 näkyy osa muuttujista, joiden arvon voi vaihtaa nollasta yhdeksi, jolloin kyseiset tiedot kirjoittuvat pelin aikana konsoliin, pelin päälle kiinteään kohtaan näytöllä tai kolmiulotteiseen avaruuteen esimerkiksi hahmon kohdalle (Kuva 20).



Kuva 20. hahmon debug tietoja



```

Create Object X Console X
File Edit Help
CAIManager: navigation world monitor state: Started -> Stopped
CActor::NetReviveAt: Dude
[CONSOLE] Executing console command 'memReplayLabel spawnPlayer'
<Lua> self.Properties.Switch: Props_Interactive.switches.small_switch
[Warning] AI: Variable 'ExecuteSequence' missing from targetA3's Behavior Tree.
[Warning] AI: Variable 'ExecuteInterruptibleSequence' missing from targetA3's Behavior Tree.
[Warning] AI: Variable 'ExecuteSequence' missing from targetA2's Behavior Tree.
[Warning] AI: Variable 'ExecuteInterruptibleSequence' missing from targetA2's Behavior Tree.
<Lua> self.Properties.Switch: Props_Interactive.switches.small_switch
ai_DebugDrawHideSpotSearchRays (0)
ai_DebugDrawLightLevel (0)
ai_DebugDrawNavigation (0)
ai_DebugDrawNavigationWorldMonitor (0)
ai_DebugDrawPhysicsAccess (0)
ai_DebugDrawPlayerActions (0)
ai_DebugDrawReinforcements (-1)
ai_DebugDrawStanceSize (0)
ai_DebugDrawVegetationCollisionDist (0)
ai_DebugDrawVisionMap (0)
ai_DebugDrawVisionMapObservables (1)
ai_DebugDrawVisionMapObservers (1)
ai_DebugDrawVisionMapObserversFOV (0)
ai_DebugDrawVisionMapStats (1)
ai_DebugDrawVisionMapVisibilityChecks (1)
ai_DebugDrawVolumeVoxels (0)
ai_DebugGlobalPerceptionScale (0)
ai_DebugHideSpotName (0)
ai_DebugInterestSystem (0)
ai_DebugMovementSystem (0)
ai_DebugMovementSystemActorRequests (0)
ai_DebugPathfinding (0)
ai_DebugPerceptionManager (0)
ai_DebugPressureSystem (0)
ai_DebugRangeSignaling (0)
ai_DebugSearch (0)
ai_DebugSignalTimers (0)
ai_DebugTacticalPoints (0)
ai_DebugTacticalPointsBlocked (0)
ai_DebugTargetSilhouette (0)
75 items match, arrows to navigate, press TAB to complete
ai_DebugMovementSystem

```

Kuva 21. Konsolin ennakoivan tekstinsyötön näyttämät muuttujat virheenkorjaustilojen aktivoimiseksi

#### 4.7 Projektin kokoaminen

Cryengin editorilla ei voi luoda lopullista tuotosta uuteen polkuun kuten voisi olettaa, vaan editorin itse poistetaan, jolloin jäljelle jäävät moottori ja asset-tiedostot. Asset-tiedostot voidaan pakata .pak-tiedostoihin, jotka ovat itseasiassa vain .zip tiedostoja eri päätteellä.

## 5 JOHTOPÄÄTÖKSET

Cryengine on selvästi tarkoitettu suuremmille projekteille. Pelimoottorin sujuva ja tehokas käyttö vaatii hyvin paljon tutustumista ja opettelua. Kokemus muista pelimoottoreista on hyödyksi Cryengininen käytön opettelussa. Projektia pystyisi varmasti jatkamaan äärettömästi, mutta se saatiin opinnäytetyön toteutuksen aikana siihen pisteeseen, että konsepti todettiin riittävän toimivaksi ja määränpää saavutetuksi.

Jälkeenpäin pohdittuna Flow Graph -järjestelmällä toteutetut asiat olisi voitu tehdä myös C++ -rajapintaa käyttäen. Tämä olisi kuitenkin vaatinut, että työn alussa olisi käytetty huomattavasti enemmän aikaa Cryengininen C++ -rajapintaan tutustumiseen. C++ -versiossa ainakin lipunpoimimislogiikka olisi saatu luotettavammaksi ja selkeämmäksi käyttämällä *boolean* muuttujaa Flow Graph Gate -logiikan sijaan. Flow Graph -järjestelmä on kuitenkin soveltuva nopeaan kehittämiseen ja prototyypin luomiseen, joten sen käyttämisessä ei ollut tuloksen kannalta liian suuria haittapuolia.

## 6 LOPUKSI

Tämä projekti oli pääpiirteittäin mielekäs. Cryengine sisälsi paljon uusia ominaisuuksia. Paljon aikaa kului uusien asioiden opetteluun Cryengininen erilaisuuden ja vähäisen dokumentoinnin vuoksi. Projektin aikana todettiin oletus Cryengininen käyttäjäepäystävällisyydestä todeksi useiden kaatumisten ja muiden pienempien ongelmien perusteella.

Toteutukselle asetettu päämäärä saavutettiin ongelmista huolimatta. Lopullisessa työn versiossa tietokone pelaa lipunryöstöä ilman kummempia virheitä. Testauksen aikana havaittiin usein, että sininen joukkue saa pisteitä hieman punaista joukkuetta useammin. Piste-ero voi johtua kentän muotoilusta, tai todennäköisyydet olivat vain punaista joukkuetta vastaan. Noin kolmenkymmenen minuutin yhtäjaksoisen testin aikana sininen joukkue sai 12 pistettä ja punainen joukkue 9 pistettä. Pisteiden kertymistä ei testattu enempää, johtuen juurikin mahdollisesti epäreilusta kentän muotoilusta, jonka seurauksena pisteiden määrää ei nähty oleellisena toteutuksen kannalta.



Projektin aikana opittiin visuaalisen ohjelmoinnin periaatteita, joita voi Flow Graph -järjestelmän lisäksi tulla vastaan Unreal Engine pelimoottorin Blueprint järjestelmää käyttäessä. Myös Behavior Tree rakenteiden muodostamisesta saatiin selkeämpi kuva. Selväksi tuli myös se, että jos ohjelmistolla on pieni käyttäjäyhteisö, ei käytännön esimerkkejä ja malleja ole helposti saatavilla. Vaikka tämän pitäisi olla itsestään selvä johtopäätelmä, ohjeiden vähyys oli kuitenkin niin äärimmäistä, että se pääsi hieman yllättämään työtä aloittaessa. Jatkossa Cryengine ei luultavasti ole ensisijainen pelimoottorivalinta sen pienien, mutta epämiellyttävien, vikojen takia.

## LÄHTEET

Champanand, A., Dunstan, P. 2014. The Behavior Tree Starter Kit. Teoksessa Rabin, Steve (ed.) Game AI Pro. Collected wisdom of game AI professionals. London: CRC Press, 73–91.

Hansen T. 2016. Startup Uses Deep Learning to Detect Disease from Medical Scans. WWW-dokumentti. Saatavissa: <https://blogs.nvidia.com/blog/2016/02/09/deep-learning-3/> [viitattu 14.3.2017].

Mishra, P., Shrawankar, U. 2016 Comparison between Famous Game Engines and Eminent Games. Saatavissa: [http://www.ijimai.org/journal/sites/default/files/files/2016/06/ijimai20164\\_1\\_13\\_pdf\\_63301.pdf](http://www.ijimai.org/journal/sites/default/files/files/2016/06/ijimai20164_1_13_pdf_63301.pdf) [viitattu 14.3.2017].

Cryengine dokumentaatio. 2014. Crytek. Saatavissa: <http://docs.cryengine.com/display/SDKDOC4/Home> [viitattu 30.11.2016].

Cryengine MBT-dokumentaatio. Crytek. WWW-dokumentti. Saatavissa: [docs.cryengine.com/download/attachments/1933332/Modular%20Behavior%20Tree%20Documentation.pdf?version=1&modificationDate=1366896427000&api=v2](http://docs.cryengine.com/download/attachments/1933332/Modular%20Behavior%20Tree%20Documentation.pdf?version=1&modificationDate=1366896427000&api=v2) [viitattu 14.11.2016]

Cryengine SignalReference dokumentaatio. 2013. Crytek. WWW-dokumentti. Saatavissa: <http://docs.cryengine.com/display/SDKDOC4/Signal+Reference> [viitattu 4.10.2016].

Cryengine TPS-dokumentaatio. 2014. Crytek. WWW-dokumentti. Saatavissa: <http://docs.cryengine.com/display/SDKDOC4/Tactical+Point+System> [viitattu 4.10.2016].

Cryengine yhteisöfoorumi. 2016. Crytek. WWW-dokumentti. Saatavissa: <https://www.cryengine.com/community/viewforum.php?f=314> [viitattu 4.10.2016].

Dawe M. 2014 Real-World Behavior Trees In Script. Teoksessa Rabin, Steve (ed.) Game AI Pro. Collected wisdom of game AI professionals. London: CRC Press, 93–98.

Designing Artificial Intelligence for Games. 2009. Intel. WWW-dokumentti. Saatavissa: <https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1> [viitattu 30.11.2016].

Dill, K. 2014. What Is Game AI? Teoksessa Rabin, Steve (ed.) Game AI Pro. Collected wisdom of game AI professionals. London: CRC Press, 3–10.

Mastering CryENGINE. 2014. WWW-dokumentti. Saatavissa: [apprize.info/game/cryengine/7.html](http://apprize.info/game/cryengine/7.html) [viitattu 20.3.2017].

Mnemstudio. 2012. WWW-dokumentti. Saatavissa <http://mnemstudio.org/path-finding-a-star.htm> [viitattu 30.11.2016].

MIT Rajiv Eranki. 2002. Massachusetts Institute of Technology WWW-dokumentti. Saatavissa <http://web.mit.edu/eranki/www/tutorials/search/> [viitattu 30.11.2016].

Stanford Amit's Thoughts on Pathfinding. 2009. Verkköjulkaisu. Saatavissa <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html> [viitattu 30.11.2016].

Sturtevant, N. 2014. Choosing a Search Space Representation. Teoksessa Rabin, Steve (ed.) Game AI Pro. Collected wisdom of game AI professionals. London: CRC Press, 253–258.

## KUALUETTELO

Kuva 1. Ryhmien (Faction) määrittely

Kuva 2. Yksinkertainen MBT

Kuva 3. Lähdekoodiin lisätty uusi Lua TPS kysely

Kuva 4. Flow Graph muokkaustyökalun perusnäkö

Kuva 5. Navigation Mesh esimerkki

Kuva 6. Navigation Mesh konfiguraatitiedosto

Kuva 7. Designer Tool perusnäkö

Kuva 8. Pelialue

Kuva 9. Pelkistetty hahmotelma MBT logiikan toiminnasta

Kuva 10. MBT tilakoneen IdleState niminen tila

Kuva 11. MBT tilakoneen GoingToEnemyFlag niminen tila

Kuva 12. MBT tilakoneen ReturningToOwnFlag niminen tila

Kuva 13. MBT tilakoneen SearchForEnemy niminen tila

Kuva 14. MBT tilakoneen AttackEnemy niminen tila

Kuva 15. Flow Graph järjestelmällä agentin joukkueen tarkistus lipuin luona

Kuva 16. Lipun liikuttaminen hahmon mukana Flow Graph -logiikka

Kuva 17. Flow Graph holdPosition pisteiden asettaminen

Kuva 18. Satunnaisen välietappipisteen asettaminen Flow Graph -järjestelmällä

Kuva 19. Flow Graph virheidenetsintä

Kuva 20. hahmon debug tietoja

Kuva 21. Konsolin ennakoivan tekstinsyötön näyttämät muuttujat virheenkoraustilojen aktivoimiseksi

Taulukko 1. Signal Reference

