Gruppbeteende - Formationer

TSBK03 Teknik för avancerade datorspel

Anton Lindskog, antli559@student.liu.se
William Malteskog, wilma366@student.liu.se

9 augusti 2022

1 Inledning

Detta är ett arbete där en AI ska fungera i grupp och tillsammans bilda formationer. Arbetet inkluderar metoder som detektion av andra AI, kollisionsundvikning och krav som prioritera platser i formationen. Tillvägagångsättet har mestadels utgått från egen implementering som stödjer sig på egen uppfattning och kunskap.

Arbetet kommer att utföras med hjälp av Unreal Engine 4 med dess visuella kodning i tanke om att se dess potential och lämplighet för implementation av gruppbeteende för AI.

2 Bakgrund

Unreal Engine 4 är en av de ledande spelmotorerna på marknaden idag och detta arbete sågs som en möjlighet att fördjupa kunskaper inom applikationen. Unreal Engine 4 kommer i fortsättningen att förkortas till *UE*.

2.1 Blueprints

Kod i UE skrivs i C++ men det finns ett verktyg för att skriva kod mer visuellt, så kallat *Blueprints*. Som kan ses i Figur 1 används ett nodsystem där variabler kopplas samman med linjer. Likt en typisk funktion i C++ är skriven så motsvarar den lila noden uppe till vänster funktionens parametrar och den andra lila noden dess returvärde. Operationer på värden kopplas sedan på dessa linjer, så som subtraktion och skalärprodukt.[1]

2.2 Beteendeträd

Ett *beteendeträd*, eller kort BT, bestämmer hur AI:n ska bete sig. Likt blueprints byggs ett BT upp av sammankopplade noder. Trädet kallar dess noder i ordning från roten och ned, från vänster till höger. Sedan kallas funktioner beroende på hur noderna är kopplade. Ett BT har ett par noder som bygger upp logiken. Till ett BT tillkommer en *Blackboard*, vilket kan ses som AI:ns minne där variabler som används i BT sparas.

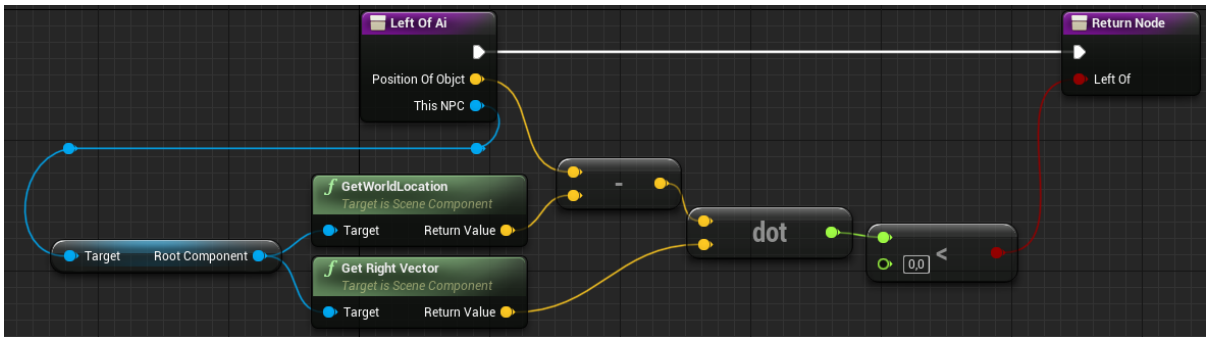
Decorator är noder med villkor som avgör om en enstaka nod eller gren ska verkställas, likt if-satser i generisk C++ kod. Dessa noder kan till exempel även vara i formen av en loop, där en nod eller gren i BT verkställs upprepande tills att villkoret inte längre gällller.

En annan nod är en *Task* vilket är en funktion som kallas. Det som är speciellt för BT är att en Task retunerar sant eller falsk beroende på om den lyckades utföra det den är designad till att göra. Till exempel kan en Task vara att AI:n ska gå till en given position och om den lyckas returnerar den sant. Men en Task kan också vara att utföra någon form av beräkning.

Ytterligare en nod är en *Selector*, den verkställer dess grenar och söker efter en nod som returnerar sant. Om en nod returnerar falsk går den vidare till nästa gren. Likt en Selector finns en nod *Sequence* som kallar sina grenar från vänster till höger tills att en nod returnerar falsk.[2]

2.3 NavMesh

För att en AI ska kunna manövrera en statisk värld måste den veta vilka punkter som är möjliga att nå



Figur 1: En enkel funktion för att avgöra om en punkt är sedd till höger eller vänster från en NPC, gjord i Blueprints i Unreal Engine 4.

och om det ligger objekt i dess väg. I UE finns det en redan implementerad metod en så kallad NavMesh. I dess mest simpla användning, placeras en fyrkant över det området i världen där AI:n ska kunna manövrera i. *NavMeshen* utför sedan beräkningarna som AI:n sedan kan nyttja. [3]

2.4 Seende

En av de mest använda förmågorna en AI kan ha i spel är seende, ett seende som som spelaren kan vänta sig och relatera till. Med andra ord ett seende likt människans. UE kommer med AI perception implementerat vilket ger användaren möjlighet att applicera flera sinnen till en AI. Så som hörsel, mottagning av skada och seende. För ett simpelt seende krävs inte mycket beräkning, AI:ns riktning och en *array* med möjligt synbara objekt gör att man kan beräkna vinkeln för synfältet och kontrollera om objekt befinner sig inom det. Fördelaktligen så begränsas även radien på synfältet. Det vill säga hur långt AI:n ser.

2.5 Gruppbeteende

Det mest klassiska exempel som dyker upp när man tänker på gruppbeteenden och grupprörelser är fiskstimm. Det som ska bemärkas med fiskstimm är att de inte har en bestämd ledare, utan fiskarnas beteenden beror på fiskarna runt om. Detta är det man kallar flockbeteende, *flocking* på engelska. Nyckelkomponenterna hos flockbeeteenden är att varje **enhet** drar sig mot varandra med samma hastighet och avstånd och samtidigt undvika kollision. Om gruppen istället följer en ledare eller ett grupp centrum kallas det istället för gruppbeteende. Det behöver inte nödvändigtvis finnas en tilldelad ledare för gruppbeteendet utan den kan vara dynamisk och bestämmas utifrån olika kriterier vart efter. Fördelar med detta är att den mest lämpliga enheten kan bestämmas i varje scenario när formationen ändras eller förflyttas samt om enheter läggs till eller tas bort.[4]

2.5.1 Formering

Ett annat klassiskt gruppbeteende är fåglars sätt att flyga i en V-formation, rättare sagt formering. I en formering finns det bestämda punkter och relationer som fåglarna söker sig till. Beroende på om fåglarna har en bestämd formationspunkt eller om dessa beräknas dynamiskt, skapas olika problem som kräver olika metoder för att lösas. Med olika krav som sätts för formationen, alltså om formationen ska fyllas på i en viss ordning, om fåglarna ska kunna ändra position dynamiskt, om det finns någon ledare i flocken osv, kan en lämplig positionsfördelning tas fram. [5]

2.5.2 Kollisionsundvikning

Vidare om fåglar med dess formering. Trots att fåglarna har en väl optimerad platsfördelning krävs att dom försöker undvika varandra för att motverka kollision vid grupprörelser. Idén bakom kollisionsundvikning är att fåglarna försöker styra iväg från en kommande kollision. Med hjälp av seendet vet fågeln vilka fåglar som ska undvikas och hur mycket

av kursen som krävs ändras för att undvika kollisionen kan då beräknas. Något som är värt att notera här är att detta inte nödvändigtvis påverkar de andra fåglarna.[6]

3 Implementation

3.1 Formationspunkter

För att en AI ska kunna bilda en formation behöver den veta hur formationen ska se ut. Att ge användaren redskap för att kunna flytta och vrida formationen under körning underlättar test av AI:n. En V-formation valdes då den är relativt enkel i formen.

Kameravyn är vinklad uppifrån för att ge en övergripande bild av gruppen. När användaren klickar med höger musknapp utförs en *Linetrace* det vill säga en linje sänds i z-axels riktning och söker efter en statisk *mesh* att kollidera med. När den gör det sparas kollisionspunkten i världkoordinater och denna punkt utgör V-formationens spets.

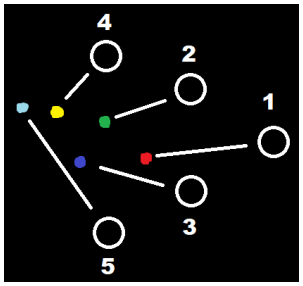
Då formationen är ett V behövs en riktning bestämmas, detta löstes genom att även ge användaren kontroll över en annan punkt, denna punkt bestäms med vänster musklick. Således kan en riktning dras mellan dessa två punkter. Denna riktning används sedan för resterande punkter i formationen, genom att addera en roterad vektor relativt till riktningen beräknas punkterna. Beroende på vektorns längd och graden av rotation kan V-formationen modifieras efter tycke.

För formationspunkterna gjordes en ny datastruktur, där formationspunktens världskoordinater, prioritering i formationen, referens till den NPC(AI) som tagit punkten och en boolean för att indikera om en punkt blivit tagen eller ej sparas.

3.2 Platsfördelning

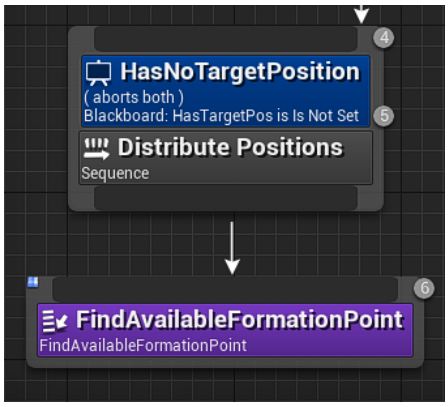
För V-formationen beslöts det att formationen bör fyllas på framifrån, därav prioriteringarna. Platsfördelningen i formationen behövde även vara dynamisk, vilket betyder att NPC:erna inte tilldelas någon bestämd punkt, utan dom delas ut och byts ut under körningen beroende på deras positioner.

Eftersom att formationen ska fyllas på framifrån, är målet med platsfördelningen att positionerna tilldelas till den NPC som är närmst, sett utifrån prioritering. Alltså, punkten i spetsen, som också har högst prioritering, kommer att tilldelas till den NPC som är närmst just den punkten. Därefter tilldelas den näst högst prioriterade punkten på samma sätt osv. Värt att nämna är att det i själva fallet är NPC:n som söker en punkt och inte tvärt om.



Figur 2: Platsfördelning med prioriterade platser med kortast avstånd.

Processen av positionssökningen av en NPC inleder med att kolla på den högst-prioriterade punkten. Om punkten är ledig tar den punkten, annars jämförs avståndet till punkten mellan den NPC som redan har den tilldelad. Den som har kortast avstånd får behålla punkten och den andra får undersöka vidare mellan punkterna av lägre prioritet. Detta upprepas parallellt mellan NPC:erna tills att alla NPC:er har en punkt tilldelad. Efter att processen är fullbordad bör platsfördelningen se ut likt Figur 2. Som vi ser på bilden så har punkt 1, som har högst prioritering, fått den röda NPC:n tilldelad som är den närmsta NPC:n för den punkten, trots att NPC:n är närmre punkt 2.



Figur 3: Platsfördelningen i beteende trädet.

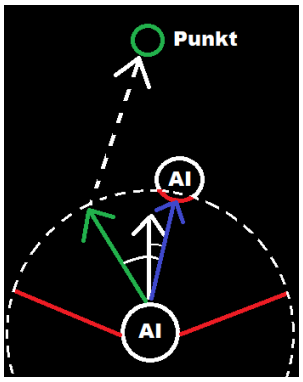
I Figur 3 visas platsfördelningen ser ut i beteendet trädet där. Den blå rutan motsvarar villkoret att NPC:n inte har någon formationspunkt och då kan exekvera den lila noden som motsvarar funktionen för punktsökningen.

3.3 Seende

Först användes UE egna implementation av AI seende, ett problem uppstod dock att när AI:n väl registrerat en annan NPC i dess synfält så registrerar den inte om ytterligare en skulle gå in i fältet. Istället för att ödsla tid på att lösa detta gjordes en egen implementation av seende baserat på teorin som tidigare nämnts.

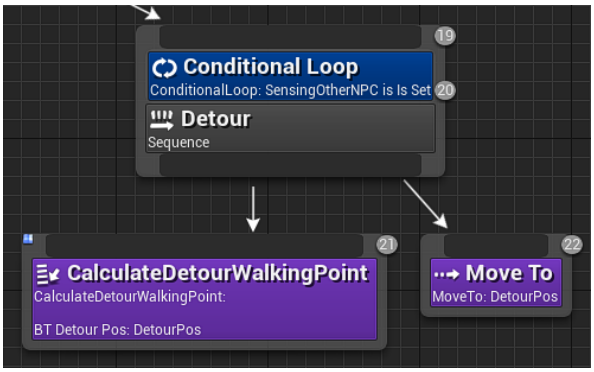
3.4 Undvikning

Efter att platsfördelningen var implementerad så uppstod situationer där NPC:er kolliderar när de exempelvis korsar varandras väg, vilket var väntat. För att förhindra detta implementerades en funktion för kollisionsundvikning. Värt att notera är att det finns en redan implementerad funktion för en AI att undvika kollisioner med varandra i UE, men då kollisionsundvikningen utgör en stor del av gruppförelser valdes istället att göra en egen implementation.



Figur 4: Platsfördelning med prioriterade platser med kortast avstånd.

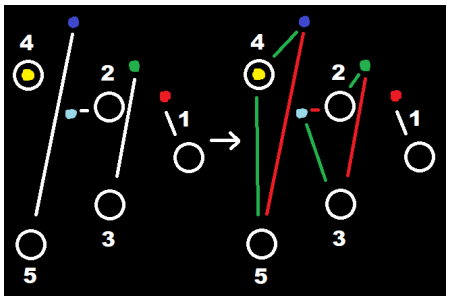
Undvikningsmetoden har som mål att utnyttja seendet för att beräkna en optimal rutt runt NPC:er som hindrar dess väg och i bästa fall undvika att ta stora omvägar. Först går NPC:n igenom alla NPC:er som betraktats inom synfältet och beräknar för var och en vinkeln mellan färdriktningen och de andra NPC:erna. Eftersom minsta möjliga omväg sökes sparas den minsta vinkeln tillsammans med referensen till motsvarande NPC:n samt vilken sida den befinner sig av. Undvikningsrutten sätts då till att vara riktad 90 grader åt motsatt sida från den betraktade NPC:ns riktning.



Figur 5: Kollisionsundvikningen i beteende trädet.

Exempel på kollisionsundvikningen ser vi på Figur 4 där den vita pilen är NPC:ns riktning, riktningen mot den sedda NPC:n är den blå och undvikningsrikntningen är den gröna. NPC:n kommer fortsätta att gå undvikningriktningen ända tills att

ingen NPC befinner sig i synfältet, då den återtar riktningen rakt mot punkten, som kan ses på den streckade pilen. I beteende trädet i Figur 5 kan motsvarande ses där beräkningen av undvikningsriktningen utförs, följt av att den går mot dess riktning, vilket enligt villkoret kommer att fortsätta exekveras så länge en NPC är inom dess synfält.

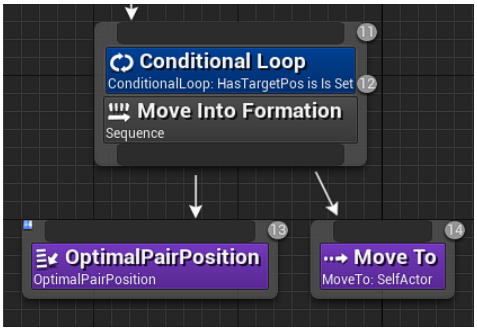


Figur 6: Platsfördelningen innan och efter paroptimeringen.

3.5 Paroptimering

Det blev snabbt tydligt att platsfördelningen var varken optimerad eller tillräcklig. Detta berodde bland annat på att det inte togs till hänsyn att varje punkt-par från spetsen ligger lika långt bort och därför borde ha samma prioritering och jämföras vid platsfördelningen. Då platsfördelningen endast ger positionerna den NPC med kortast avstånd från prioritering, vilket inte inte nödvändigtvis är den närmsta punkten för NPC:n skapas det även problem som att NPC:er som hinner till sina tilldelade punkter och blockerar vägen till andra. Ännu ett hinder var att då platsfördelningen endast sker initialt och inte uppdateras vid körtid, gör det att NPC:erna inte kunde byta plats under körning.

För att göra kontinuerliga optimeringar till platsfördelningen och motverka oönskade hinder infördes en funktion som optimering mellan parpunkterna med samma prioritering. Målet med paroptimeringen är att se till att bägge NPC:er går enskilt kortast avstånd. Detta görs genom att beräkna det största avståndet för NPC:erna i med punkt-par till respektive tilldelade punkt och sedan jämföra med det största avståndet mellan varandras punkt. Om de tilldelade platserna har större max-avstånd byter NPC:erna punkt. Detta är för att det fall med minst max-avstånd är det fallet som går snabbast eftersom båda går med konstant hastighet, vilket gör att båda blir klara snabbast.



Figur 7: Paroptimeringen i beteende trädet.

På vänstra delen i Figur 6 syns en instans där platsfördelnignen har utförts och är markerade med streck. I detta fall hamnar två NPC:erna i blått och grönt bakom de andra, vilket tvingar dom att gå runt. Den högra delen är resultatet av paroptimeringen där de röda strecken är de gamla tilldelningarna och de nya är markerade med grönt. Här har den gula NPC:n bytt punkt med den mörkblå med samma punkt-par, trots att den gula redan befinner sig på en position. Anledningen till detta är för att den tidigare sträckan för den mörkblå är längre än de båda enskilda nya sträckorna. Tillsammans kommer de alltså att färdas en längre sträcka men den längsta sträckan en NPC behöver ta är kortare och eftersom de går samtidigt är de på plats snabbare.

Då beteende träden för NPC:erna körs parallellt mellan alla NPC:er uppstod många problem med referenser och tilldelningen av variabler eftersom en funktion som exekveras av den ena NPC:n kan ändra värden på en annan NPC medans den själv kör en funktion. Detta problem förekom i större del

vid paroptimeringen då denna jämför och eventuellt byter punkter mellan NPC:er. Detta problemet löstes genom att helt enkelt välja att endast första NPC:n utför funktionen och låta den ändra värden för båda punkterna och NPC:erna i paret.

3.6 Dynamiskt under körtid

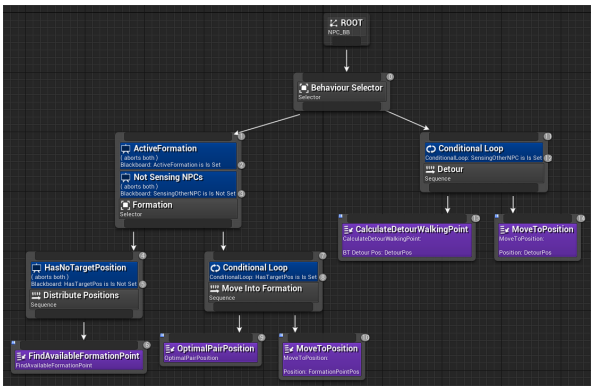
Till en början användes UE implementation för att få AI:n att gå till en given punkt i scenen. Denna funktion användes i BT som en Task, likt alla Tasks returnerar de sant eller falsk beroende på om det funktionen är avsedd att göra lyckas eller ej. I det här fallet returnerar denna implementationen sant först när AI:n når den angivna punkten. Detta orsakar problem om man vill dynamiskt ändra till vilken punkt AI:n ska gå till under körtid. Då det inte går att ge en ny punkt att gå till förrän AI:n nått den tidigare givna punkten. Detta löstes genom att implementera en egen Task som gör att AI:n börjar gå mot en given punkt, men som returnerar sant direkt. Det vill säga att funktionen lyckats sätta AI:n i rörelse mot punkten. Det gör så att BT inte låser sig tills AI:n når punkten och istället kan nya punkter anges dynamiskt under körning.

Att kunna ta bort och lägga till NPC:er i världen dynamiskt under körtid är ett rimligt krav på en applikation som riktar sig till spel. Att ta bort eller lägga till NPC:er är ett enkelt problem, UE kommer med fullt implementerade metoder för detta. Problemet ligger i att alla NPC:er utför beräkningar kontinuerligt under körtid som beror på andra NPC:er. Då en NPC tas bort kommer övriga NPC:er vars beräkningar berör den bortagna att få konstiga resultat. Detta löstes genom att låsa beteendeträden för samtliga NPC:er i scenen så att beräkningar ej längre görs. Sedan kan en NPC tas bort eller en ny läggas till. Därefter låses beteendeträden upp igen för att fortsätta med beräkningarna.

4 Slutprodukt

Arbetet resulterade i en applikation där man kan kontrollera formationen som en grupp NPC:er ska hålla. Det begränsades till en typ av formation, V-formation, vilkets position och riktning kan bestämmas under körtid. Applikationen stödjer också ett obestämt antal NPC:er, färre eller fler än de existerande formationspositionerna, samt att det går att dynamiskt kan lägga till och tar bort NPC:er under körtid.

AI:n hanterar sökningen av formationspunkter på ett rimligt vis, paroptimeringen körs kontinuerligt under körtid och gör så att formationen kan bildas snabbare samt att den minskar antalet kollisioner.

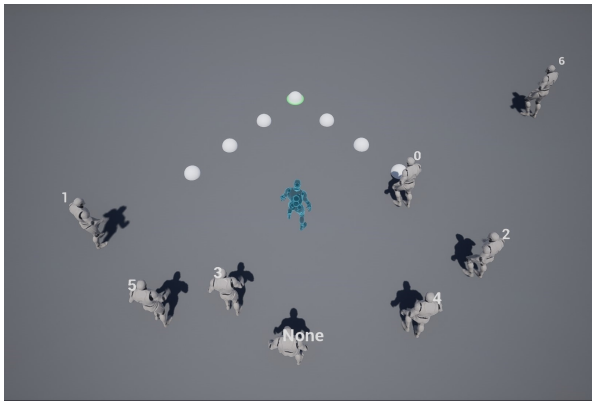


Figur 8: Applikationens beteendeträd.

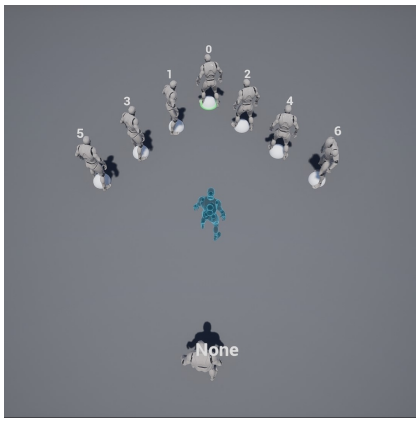
I Figur 8 visualiseras det slutgiltiga beteendeträdet för AI:n. Trädet kallar dess noder uppifrån och ned, vänster till höger. Först kontrolleras det i vänstra grenen med en Decorator node om det finns aktiva formationspunkter för AI:n att söka sig till. Om det finns, så kontrolleras sedan att det inte finns någon annan NPC i vägen. Gör det inte det, så går selectorn alltså till nod nummer 3 som är uppmärkt i sagd figur, till vänster gren under sig, decorator noden 'HasNoTargetPositio', där kontrolleras om AI:n har en punkt att gå till, har den inte det så kallas en Task under som tilldelar en punkt. Därefter kommer nästa gren under selectorn att kallas, där en *Condition Loop* körs, denna loop kommer köras så länge som AI:n har en formationspunkt att söka sig till. I denna loop kallas funktionerna 'OptimalPairPosition' som optimerar hur punkter har valt paren emellan och sedan funktionen 'MoveToPosition' som gör att NPC:n går mot formationspunkten. Skulle en annan NPC upptäckas i synfältet kommer BT istället gå till den vänstra grenen, det vill säga nod nummer 11. Då beräknar en lämplig punkt utifrån upptäckta NPC:er med funktionen 'CalculateDetourWalkingPoint' att gå till för att undvika dem med 'Mo-

veToPosition', vilket upprepas tills att ingen NPC längre är sedd.

I Figur 9 visas platsfördelningen till NPC:erna där deras tilldelade position är numrerade över respektive NPC. Punkterna är numrerade mellan 0-6 och utgår från spetsen och fortsätter därefter varannan punkt först från vänster och sedan till höger, sett med spetsen uppåt. Detta är tydligare visat på Figur 10. Notera att det 8 NPC:er som ska fördela platser mellan 7 punkter, vilket medför att en NPC inte får en position och därav 'None'.

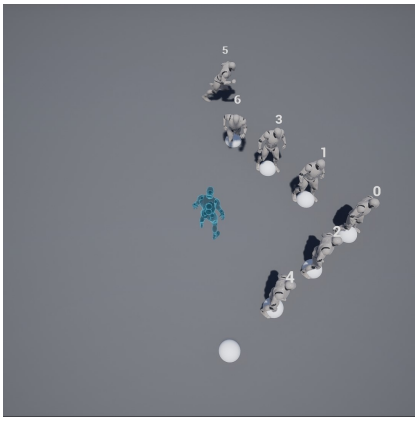


Figur 9: Platsfördelning mellan NPC:erna innan de börjat gå.



Figur 10: NPC:erna som har intagit deras position.

I Figur 11 visas platsfördelningen som sker direkt när en NPC har lagts till då det finns en ledig punkt i formationen. NPC:n som har lagts till befinner sig högst upp i bilden. Här visas resultatet av paroptimeringen då den tillagda NPC:n har tilldelats samma punktpar som NPC:n på punkt 5, men då den tllagda NPC:n är längre bort från båda punkterna byter dessa deras punkter. Liknande resultat fås även då en NPC tas bort ur formationen, vilket ses på Figur 12. Här har NPC:n på punkt 1 just tagits bort, vilket gör att NPC:erna på plats 3 och 5 flyttar framåt mot spetsen medans de övriga NPC:erna står kvar vid deras tidigare positioner.



Figur 11: Platsfördelning efter en NPC har blivit tillagd med en ledig plats innan de börjat gå.

Figur 12: Platsfördelning efter slumpvis NPC har tagits bort innan de börjat gå.

5 Diskussion

Något som i början lades mycket tanke på men sedan kom till att vara nyckeln till många av de förekommande problemen var 'MoveTo' funktionen. Detta är en funktion som är lätt att tänka bort då dess enda uppgift är att be AI:n att gå till en specifik position. Då det anmärktes att den implementerade funktionen i UE obesrtört exekverar denna funktion tills dess att punkten har blivit nådd blev det uppenbart att detta i själva fallet var huvudanledningen till att inte våra funktioner fungerade som väntat. Det spenderades stor tid att felsöka funktionerna utan någon lycka, när problemet var enklare än så. Detta gjorde dock att

Koden för undvikning är simpel och till en början fungerade den den inte särskilt bra. Den främsta bristen berodde på att UE egna implementation för att få NPC:n att gå inte fungerade så som det var tänkt, men detta löstes genom att implementera en ny som tidigare nämnts. Det återstår mycket som kan förbättras när det kommer till specifikt undvikning. En förbättring hade varit att ta hänsyn till om två eller fler NPC:er upptäcks, om det finns en lucka mellan dem som AI:n kan ta sig igenom. I nuläget är det ett scenario som helt överses. AI:n har inte heller förmågan att backa, så skulle den fastna har den inget sätt att ta sig ur situationen. En implementation hade möjligen varit att om AI:n inte förflyttat sig en viss sträcka på en viss tid, så kan det vara en fråga om att den fastnat och då gå mot motsatt riktning eller en slumpvis vald. På grund av tid hanns detta inte med.

Så som undvikningen är implementarad så undviker varje NPC andra NPC:er. Det hade varit intressant att vikta undvikning baserat på prioritet i formationen. Alltså att en AI med högre prioritet hade manat andra NPC:er att ge plats så att den kan nå sin plats effektivare.

Det är viktigt att nämna att formationsbeteendet inte är baserat på någon forskning, utan är byggt på vad vi tycker ser bra ut och vad vi anser vara rimligt. Mänskligt beteende var inte huvudfokus, istället var optimering det stora fokuset. Detta ledde till att formationen ibland ser omänsklig ut. Dels beror detta på att undvikningen är bristande stundvis, även om den oftast löser formationen tillslut. Det hade gynnat arbetet om jämförelse med andra implementationer gjorts för att identifiera eventuella brister och fördelar med implementationen.

Vidare värt att nämna är att *Blueprints* är trevligt för mindre kodstycken, men ju mer komplex koden blir ju sämre lämpar sig Blueprints. Det som var tänkt att vara lätt att tyda för att överskådligt förstå kod har istället blivit mycket svårt att tyda. Något att ta med sig till framtida projekt. En stor fördel med UE är dock den stora mängd färdigimplementerade funktioner och verktyg som finns tillgängliga samt dess informationsrika och pedagogiska dokumentation.

6 Slutsats

Arbetet resulterade i en AI som fungerar i grupp och som löser en V-formation. En egen implementation av gruppbeteende, seende och undvikning gjordes. Gruppbeteendet baseras på vår egna uppfattning av vad som fungerar bra och vad som ser trovärdigt ut. Det finns mycket att finslipa på men den fundamentala grunden är lagd.

Referenser

[1] Unreal Engine, Epic Games, *Blueprints*, accessed: 2021-12-31

[2] Unreal Engine, Epic Games, *Beteendeträd*, accessed: 2021-12-31

[3] Unreal Engine, Epic Games, *Navmesh*, accessed: 2021-12-31

[4] *Ai for Game Developers*Glenn Seemann, David M Bourg, *Gruppbeteende*, accessed: 2021-12-31

[5] *Implementing Coordinated Movement*, Dave Pottinger, *Formering*, accessed: 2021-12-31

[6] *Flocks, Herds, and Schools: A Distributed Behavioral Model*, Craig W. Reynolds, *Kollision*, accessed: 2021-12-31