Automatiserad planering i spel

Niklas Luokkala
Kandidatavhandling i datavetenskap
Handledare: Marina Waldén
Fakulteten för naturvetenskaper och teknik
Åbo Akademi
2022

Referat

Automatiserad planering innebär att bygga planer som består av sekvenser av aktioner som leder till ett specificerat mål. Planer används för att styra beteendet hos så kallade agenter. Syftet med detta arbete är att analysera för- och nackdelar med planeringsteknikerna Goal-oriented Action Planner (GOAP) och Simple Hierarchical Ordered Planner (SHOP). Dessa planeringstekniker kunde översättas till målinriktad aktionsplanerare och enkel hierarkisk ordnad planerare. GOAP bygger upp planer genom att skapa kedjor av aktioner med en sökalgoritm medan SHOP bygger upp planer genom uppdelning av uppgifter.

Vid val av planeringsteknik är prestanda, modularitet och skalbarhet viktiga egenskaper att ta i beaktande. Jämförelsen i detta arbete inkluderar en analys av prestanda, skalbarhet, modularitet, prioritering, felsökning och beteende.

Nyckelord: Automatiserad planering, målinriktad aktionsplanerare, enkel hierarkisk ordnad planerare

Innehåll

1 Inledning

Artificiell intelligens (AI) i spel, enligt Millington och Funge, handlar om att få datorer att utföra tankeuppgifter som människor och djur är kapabla till [3]. Artificiell intelligens i spel kan till exempel handla om att fatta beslut, planera rutter eller bygga långsiktiga planer. I denna avhandling ligger fokus på planering, där en planerare skapar långsiktiga planer som styr artificiella intelligensens beteende.

Beslutsfattning inom AI i spel är i de flesta fall kort- eller långsiktig. Kortsiktig beslutsfattning handlar om att ta reda på vad AI bör utföra vid ett givet tillstånd. För mera avancerat beteende används planering, där man planerar för framtiden för att nå långsiktiga mål.

Enligt Ghallab et al. är automatiserad planering en beräkningsmässig process som består av val och organisering av aktioner genom att förutse deras förväntade resultat. Denna process används för att på bästa möjliga sätt komma fram till ett specificerat mål [6]. En planerare är ansvarig för att utföra denna process för att göra upp en plan. En plan består av en sekvens av aktioner eller uppgifter som utförs av en agent för att nå ett specifikt mål.

Viktiga egenskaper som man bör ta i beaktande när man väljer mellan olika algoritmer och tekniker till spel är prestanda, modularitet och skalbarhet. Målplattformen av ett spel är ofta definierade redan i början av utvecklingen. Detta betyder att processorkraften och minnesmängden är begränsad. Kraven på spelen brukar ändras under utvecklingen, vilket gör skalbarhet och modularitet till viktiga egenskaper att ta i beaktande.

Syftet med denna avhandling är att analysera och jämföra två planeringstekniker. De valda planeringsteknikerna är målinriktad aktionsplanerare (eng. Goal-oriented Action Planner, GOAP) och enkel hierarkisk ordnad planerare (eng. Simple Hierarchical Ordered Planner, SHOP). GOAP och SHOP valdes för jämförelse eftersom de ofta används i spel. Genom att lista för- och nackdelar av dessa planerare, kan en programmerare göra ett informerat val mellan dessa tekniker.

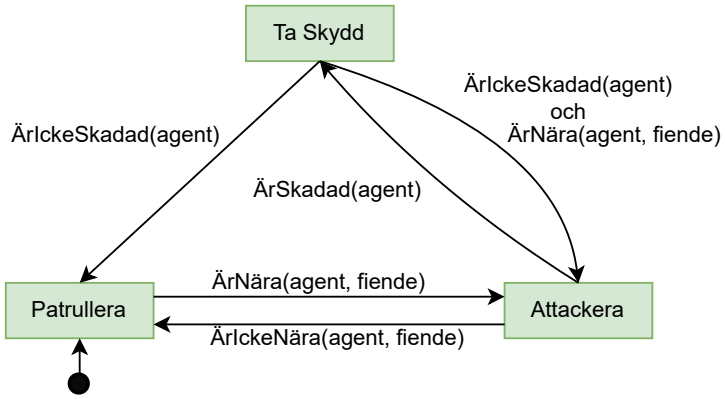
I avhandlingen beskrivs planeringsteknikerna GOAP och SHOP. Innehållet består av planering allmänhet samt bakgrund, förklarning och exempel av både GOAP och SHOP. I jämförelsedelen analyseras för- och nackdelar hos dessa planeringstekniker.

2 Planering

Beslutsfattning inom artificiell intelligens i spel syftar på agenternas förmåga att bestämma vad de borde göra. De flesta spelen använder enkla system för beslutsfattningen, till exempel ändliga tillståndsmaskiner och beslutsträd. Beteendeträd är också ett populärt verktyg för att skapa beslutsfattande agenter i spel [3]. För mer intelligent beteende kan planering användas.

2.1 Tillståndsmaskin

Ändliga tillståndsmaskiner, tillsammans med hårdkodat beteende, utgör majoriteten beslutsfattande system som används i spel idag. En agent som styrs av en tillståndsmaskin är endast i ett tillstånd åt gången. Ett tillstånd är oftast förknippat med en specifik beteende eller aktion. Agenten fortsätter att utföra aktionen så länge en den befinner sig i samma tillstånd. Transitioner innehåller villkor, och de fungerar som kopplingar mellan tillstånden. Tillståndsmaskinen byter från ett tillstånd till ett annat om villkoren i transitioner mellan nuvarande tillstånd och ett annat tillstånd uppfylls. En tillståndsmaskin, som beskrivs ovan, kallas ofta för en ändlig tillståndsmaskin (eng. finite-state machine, FSM) inom AI i spel. [3]



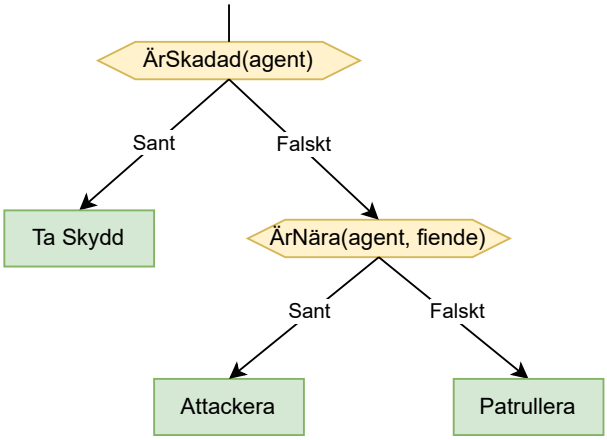
Figur 1: Ett exempel på en ändlig tillståndsmaskin.

Den ändliga tillståndsmaskinen i figur 1 har 3 tillstånd (*Ta Skydd, Attackera* och *Patrullera*) och transitioner i form av villkor. Tillståndsmaskinen är i initialtillståndet *Patrullera*. Tillståndet ändras om ett villkor i en närliggande transition uppfylls. Tillståndsmaskinen övergår, till exempel, från *Patrullera* till *Attackera* om villkoret *ÄrNära* uppfylls.

2.2 Beslutsträd

Beslutsträd är effektiva och lätta att implementera. De består av beslutspunkter som är kopplade i en trädstruktur. Vid varje punkt fattas ett beslut bland alternativen. Ett beslut involverar ofta

kontrollering av ett enda värde och innehåller inte boolesk logik (till exempel operatorer OCH eller ELLER). Traversering av trädet börjar från roten av trädet, vilket också är en beslutspunkt. Agenten gör ett beslut vid varje beslutspunkt, vilket styr traverseringen längs en gren. Beslut fattas utifrån agentens kunskap. Algoritmen fortsätter nerför trädet tills den når ett löv, vilket är en aktion som utförs direkt. [3]



Figur 2: Ett exempel på ett beslutsträd.

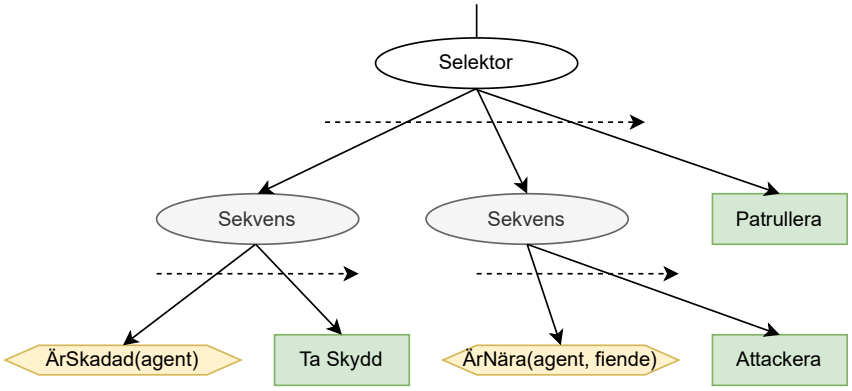
Beslutsträdet i figur 2 har 2 beslutspunkter (*ÄrSkadad* och *ÄrNära*) och 3 aktioner (*Ta Skydd, Attackera* och *Patrullera*). Algoritmen börjar med att kontrollera villkoret ÄrSkadad. Aktionen *Ta Skydd* utförs om villkoret uppfylls, annars fortsätter algoritmen till villkoret *ÄrNära*. Aktionen *Attackera* utförs om villkoret *ÄrNära* uppfylls. Om inte villkoret uppfylls, utförs aktionen *Patrullera*.

2.3 Beteendeträd

Beteendeträd är ett populärt verktyg för att skapa agenter i spel. De är trädstrukturer som har uppgifter som noder. Dessa uppgifter kan vara aktioner, villkor eller kompositer. Varje uppgift returnerar en statuskod som beskriver resultatet av den utförda uppgiften. Aktioner är löven av trädet och de används för att ändra tillståndet av spelet. Villkor används för att kontrollera någon egenskap i spelet, vilket sedan styr traverseringen av trädet. De huvudsakliga kompositnoderna är selektionsnoden och sekvensnoden. De evaluerar sina barnnoder och returnerar en statuskod enligt på beteendet av barnnoderna. Selektionsnoden returnerar en statuskod för framgång om en av dess barnnoder också returnerar en statuskod för framgång. Om selektionsnoden har inga barnnoder som lyckas, returnerar den en statuskod för misslyckande. En sekvensnod returnerar en statuskod för framgång om alla barnnoder lyckas, och en statuskod för misslyckande genast om en barnnod misslyckas. [3]

De ovannämnda uppgifter är ihopkopplade i en hierarkisk struktur, vilket tillåter uppdelning av

trädet i delträd. Beteendeträd har en egenskap som kallas reaktiv planering (eng. reactive planning). Detta uppnås med hjälp av kompositnoder och villkor. Selektionsnoder kan till exempel försöka utföra något och sedan gå tillbaka om det misslyckas. [3]



Figur 3: Ett exempel på ett beteendeträd.

Beteendeträdet i figur 3 har 2 villkor (*ÄrSkadad* och *ÄrNära*), 3 aktioner (*Ta Skydd, Attackera* och *Patrullera*) samt 2 sekvensnoder och en selektionsnod. Algoritmen börjar med att evaluera uppgiften i trädets rot, det vill säga selektionsnodet. Kompositnoderna evalueras i ordning från vänster till höger.

2.4 Planeringstekniker

STanford Research Institute Problem Solver (STRIPS) är en problemlösare som söker efter en världsmodell (eng. world model) som uppfyller ett givet mål [8]. GOAP fungerar på liknande sätt, men är specifikt anpassat för spel [5]. F.E.A.R. [5] och Middle-earth : Shadow of Mordor [9] är exempel på spel som använder GOAP.

Hierarkisk uppgiftsnätverk (eng. Hierarchical Task Network, HTN) är en planeringsteknik som skapar planer genom att dela upp av uppgifter i mindre delar. SHOP fungerar på samma sätt som HTN, men ordningen i vilket hierarkin traverseras i är annorlunda. Transformers: Fall of Cybertron [11] och Killzone 2 [1] är exempel på spel som använder HTN.

3 Målinriktad aktionsplanerare

Orkins utvecklade planeringstekniken målinriktad aktionsplanerare, vilket är baserat på STRIPS. STRIPS bygger en kedja av aktioner som modifierar världstillståndet tills en specificerad måltillstånd uppnås. GOAP fungerar på liknande sätt men skiljer sig från STRIPS i hur den hanterar världstillståndet och hur kedjan av aktioner byggs upp. [5]

3.1 Metod

Enligt Orkins [4] består GOAP av aktioner, mål och en ändlig tillståndsmaskin. I planeringen används ett världstillstånd (en samling av variabler) för att kontrollera om förvillkor i aktioner är giltiga. En aktion har effekter, förvillkor och en kostnad. Effekter består av en lista av variabler som appliceras till världstillståndet. För att effekter ska kunna appliceras måste aktionens förvillkor uppfyllas. Varje aktion har en kostnad som används av planeraren för att välja mellan flera aktioner som har likadana effekter. Sökalgoritmen använder denna kostnad för prioritering när den ska välja mellan flera aktioner. Ett mål definieras som ett specificerat världstillstånd. En ändlig tillståndsmaskin används för att utföra uppgifter och aktioner används som övergångar mellan tillstånden. En aktion kan beskrivas på följande sätt:

Aktion: Kör Till X
Kostnad: 1
Förvillkor:
Är In I En Bil
Effekter:
Är Vid X

Orkins utvecklade planeringstekniken GOAP specifikt för spel. Den skiljer sig från STRIPS med kostnader som används för att prioritera mellan flera aktioner. GOAP använder sig också av förvillkor i form av procedurer som tillåter användningen av funktioner för att kolla om förvillkoren är giltiga. Effekter i form av procedurer används också i GOAP. Dessa tillåter applicering av effekter vid en senare tidpunkt. GOAP skiljer sig också från STRIPS i hur förvillkor och effekter är representerade. STRIPS använder två olika listor, en för att lägga till och en för att ta bort effekter och förvillkor. GOAP använder en lista av bestämd storlek för både förvillkor och effekter. [5]

Planeringen utgår från ett måltillstånd där det finns ett villkor som ska uppfyllas. Sökalgoritmen letar efter aktioner som satisfierar detta villkor och väljer en aktion att fortsätta med. Den valda aktionens förvillkor blir då det nya målet. Planeraren fortsätter med att skapa en kedja av aktioner tills en aktion utan förvillkor hittas eller tills måltillståndet uppnås. Planeraren returnerar en lista av alla valda aktioner ifall planeringen lyckas. Sökalgoritmen A* används ofta för sökningen, vilket innebär att varje aktion har en kostnad samt en estimerad kostnad. Den estimerade kostnaden kan beräknas utgående från antalet onådda mål. Den totala mängden aktioner kan delas upp till mindre aktionssamlingar, så att endast de relevanta aktionerna används vid sökningen. [4]

Sökalgoritmen A* letar efter den kortaste stigen i en graf som består av noder. Algoritmen traverserar grafen genom att alltid välja den kortaste stigen från en kö av alternativa stigar. Den

nuvarande stigen överges om kostnaden av stigen är högre än en alternativ väg. En stig med lägre kostnad väljs då. En heuristik beräknas för varje alternativ stig, det vill säga för varje intilliggande nod. Alla alternativa vägar läggs till i en kö. Detta fortsätter tills målet nås [7].

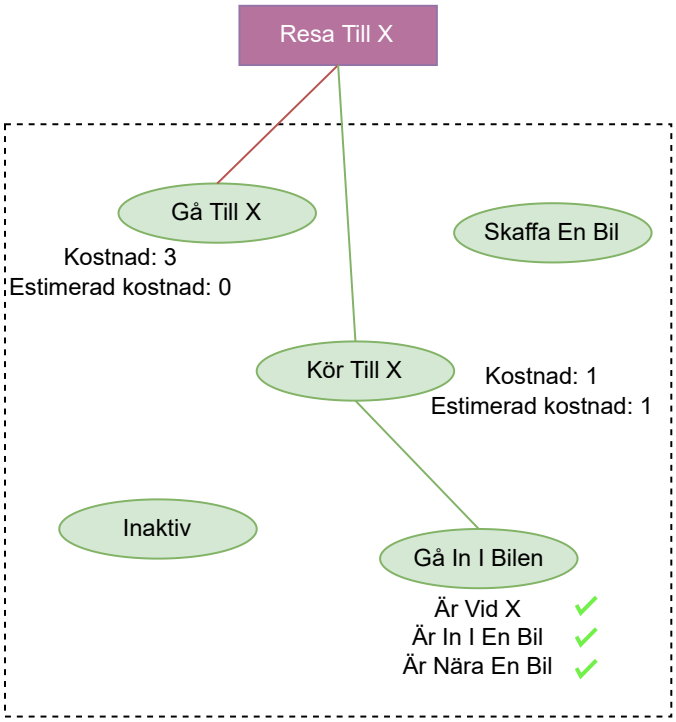
3.2 Exempel

I figur 4 finns tabeller med information om världstillstånd, mål och aktioner till vänster. Till höger finns ett exempel på hur planeraren bygger en plan för att komma till målet *Resa Till X*, som har villkoret *Är Vid X*. I början av planeringen innehåller världstillståndet informationen *Är Nära En Bil* och *Är Inaktiv*.

Världstillstånd
Är Nära En Bil
Är Inaktiv

Mål	Villkor
Resa Till X	Är Vid X
Göra ingenting	Är Inaktiv
Söka Efter En Bil	Är Nära En Bil

Aktion	Effekter	Förvillkor
Gå Till X	Är Vid X	
Kör Till X	Är Vid X	Är In I En Bil
Gå In I Bilen	Är In I En Bil	Är Nära En Bil
Skaffa En Bil	Är Nära En Bil	
Inaktiv	Är Inaktiv	



Figur 4: Ett exempel på målinriktad aktionsplanering. Inspirerad av [9].

Planeringen börjar med att sökalgoritmen letar efter aktioner som satisfierar villkoret *Är Vid X*, varvid den hittar två aktioner; *Gå Till X* och *Kör Till X*. Aktionen *Gå Till X* har kostnaden 3 och en estimerad kostnad 0 som ger totala kostnaden 3. Aktionen *Kör Till X* har kostnaden 1 och en estimerad kostnad 1 som ger totala kostnaden 2. Sökalgoritmen väljer aktionen med billigare total kostnad, i detta fall *Kör Till X*. Denna aktion har förvillkoret *Är In I En Bil*, som blir det nya målet. Sökalgoritmen söker då efter en aktion som satisfierar detta mål och hittar aktionen *Gå In I En Bil*, vars förvillkor *Är Nära En Bil* blir nästa mål. Det sista villkoret *Är Nära En Bil* kan uppfyllas med aktionen *Skaffa En Bil*. Detta är dock inte nödvändigt eftersom villkoret *Är Nära En Bil* finns redan i världstillståndet. Alla tre villkor (som är under aktionen Gå In I Bilen i figur 4) kan nu uppfyllas. Planeringen har således kommit fram till ett resultat. Den slutliga

planen består av aktionerna *Kör Till X* och *Gå In I Bilen* i omvänd ordning.

4 Enkel hierarkisk ordnad planerare

Enkel hierarkisk ordnad planerare [13] är en form av hierarkiskt uppgiftsnätverk. Hierarkiska uppgiftsnätverk bygger upp planer genom att rekursivt dela upp uppgifter till mindre delar som kan direkt utföras av agenten. Skillnaden mellan SHOP och HTN är ordningen i vilken komponenterna utvärderas. Uppgifter i SHOP uppdelas i totalordning vilket innebär att planerna består av uppgifter som är ordnade i samma ordning som dom utförs.

4.1 Metod

Som beskrivits av Neufeld et al [13] består SHOP av uppgifter i en hierarkisk struktur. Det finns två typer av uppgifter; sammansatta uppgifter (eng. compound tasks) och primitiva uppgifter (eng. primitive tasks).

Sammansatta uppgifter delas upp till deluppgifter med hjälp av metoder. Deluppgifterna kan vara endera sammansatta eller primitiva uppgifter. En sammansatt uppgift som är kopplad till flera metoder kan delas upp på många olika sätt, vilket ger planeraren möjligheten att utföra en uppgift på flera olika sätt. En sammansatt uppgift kan beskrivas på följande sätt:

Sammansatt uppgift: Resa Till X
    Metoder:
        Resa Till Fots
        Resa Med Cyckel
        Resa Med Bil

En metod innehåller en beskrivning för hur en sammansatt uppgift uppdelas till deluppgifter samt ett visst antal förvillkor. För att en metod ska kunna dela upp sammansatta uppgifter krävs att alla förvillkor (eng. preconditions) i metoden uppfylls. Förvillkoren är specifika för varje metod. En metod kan beskrivas på följande sätt:

Metod: Resa Med Bil
    Uppgift:
        Resa Till X
    Förvillkor:
        ÄrNära(agent, bil)
    Deluppgifter:

Gå In I Bilen
Kör Till X

Primitiva uppgifter är löven i det hierarkiska nätverket, vilket innebär att de inte kan delas upp. De innehåller en operatör som modifierar det interna världstillståndet. Operatören består av förvillkor och effekter. Effekterna är förändringar som appliceras till det interna världstillståndet ifall alla förvillkor uppfylls. En operatör kan beskrivas på följande sätt:

Operatör: Gå In I Bilen
Förvillkor:
ÄrNära(agent, bil)
Effekter:
Position(agent) = bil

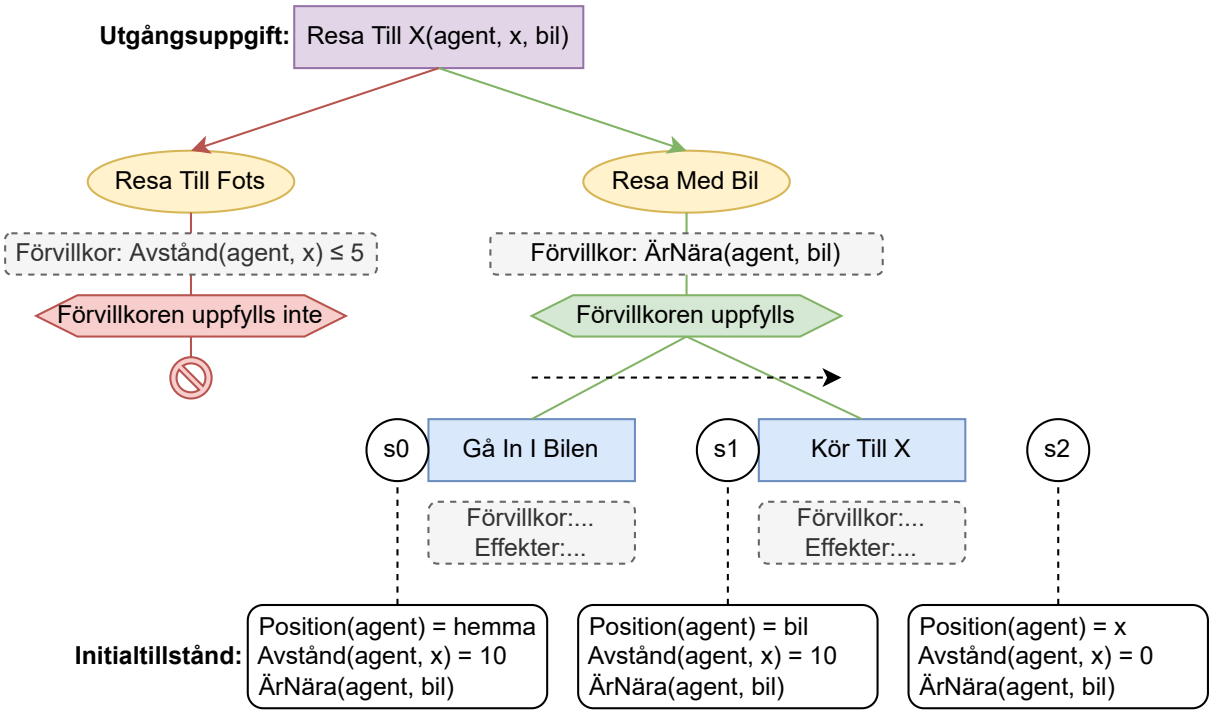
Planeringen börjar med en utgångsuppgift och ett initialtillstånd. Planeraren delar upp uppgiften tills den hittar en gren där alla förvillkor i primitiva uppgifter uppfylls. Primitiva uppgifter ändrar tillståndet vilket betyder att följande primitiva uppgifter och metoder använder sig av uppdaterat information i sina förvillkor. Planeraren går igenom metoder i samma ordning som de presenteras. Planeringen har lyckats om en gren hittas där alla förvillkor i primitiva uppgifter uppfylls. Om ett förvillkor inte uppfylls, måste planeraren gå tillbaka till den senaste sammansatta uppgiften och fortsätta med nästa metod. Förutom detta, måste alla förändringar upphävas som gjorts till interna världstillståndet sedan den senaste sammansatta uppgiften. Planeringen misslyckas om planeraren inte hittar en gren där alla förvillkor uppfylls. [2]

Delplanering är möjligt hos SHOP eftersom planen skapas i samma ordning som uppgifterna utförs. Detta betyder att planeraren inte behöver skapa en fullständig plan, utan kan dela planeringen till flera steg. [11]

4.2 Exempel

I figur 5 börjar uppdelningen från den sammansatta utgångsuppgiften *Resa Till X*. Planeringen har initialtillståndet *s0*, som innehåller information om agentens position, avståndet mellan agenten och x samt att agenten är i närheten av bilen.

Utgångsuppgiften är kopplat till två metoder; *Resa Till Fots* och *Resa Med Bil*. Planeraren går igenom dessa metoder i logisk ordning (från vänster till höger). Förvillkoren i *Resa Till Fots* uppfylls inte eftersom avståndet mellan agenten och x är längre än 5. Detta leder till att planeraren måste fortsätta med nästa metod det vill säga *Resa Med Bil*. Dess förvillkor uppfylls eftersom agenten är nära bilen. Sedan utförs primitiva uppgiften *Gå In I Bilen*, vilket innebär att dess effekter appliceras och världstillståndet ändras från *s0* till *s1*. Efter detta utförs den anda



Figur 5: Ett exempel på SHOP. Inspirerad av [2] och [13].

primitiva uppgiften, *Kör Till X*, vilket ändrar världstillståndet från *s1* till *s2*. I det slutliga världstillståndet *s2* är agenten vid positionen x, avståndet mellan agenten och x är 0 och agenten är fortfarande i närheten av bilen.

5 Jämförelse

Att ha tillgång till en jämförelse av planeringsteknikerna är nödvändigt för att kunna göra informerade val bland de olika alternativen. I detta kapitel finns för- och nackdelar hos både GOAP och SHOP. De viktigaste kriterierna är prestanda, skalbarhet och modularitet. Prestanda har en stor inverkan på användarupplevelsen medan skalbarhet och modularitet krävs för att utveckling och underhåll ska gå till så smidigt som möjligt.

5.1 Prestanda

Institute of Electrical and Electronics Engineers (IEEE) definierar prestanda som graden i vilken ett system eller en komponent utför sina funktioner med givna begränsningar såsom hastighet, noggrannhet eller minnesanvändning [10]. Spel är realtidssystem och har således tillgång till en begränsad mängd processorkraft. Prestanda påverkar användarupplevelsen och om spelet överhuvudtaget kan spelas.

I GOAP påverkas prestanda av vilken sökalgoritm som används. Ett val mellan progressiv och regressiv sökning bör också göras. Progressiv sökning innebär att utföra sökningen från nuvarande tillståndet till målet medan i regressiv sökning utförs sökningen från målet till nuvarande tillståndet. Enligt Orkins är regressiv sökning mera effektiv och intuitivt [4]. Malmberg gjorde en undersökning på ordningen av sökningen och kom fram till att regressiv sökning var effektivare än progressiv sökning [12].

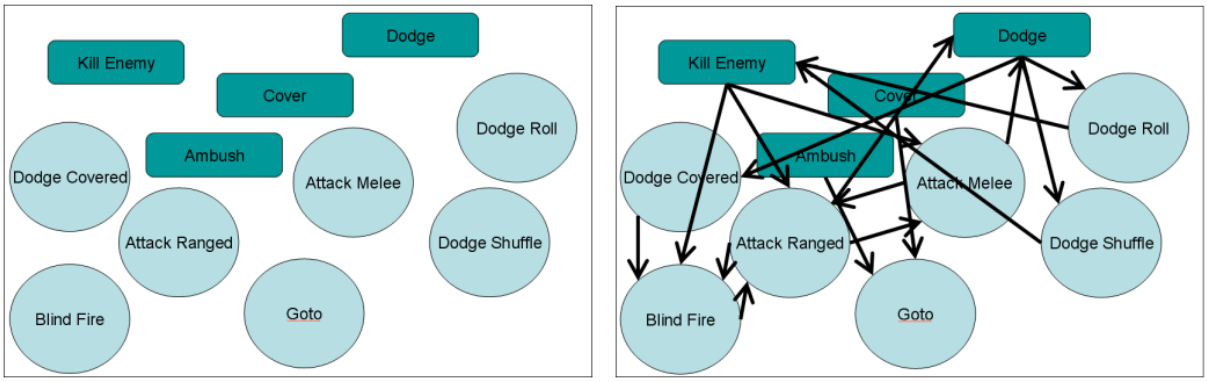
SHOP kan använda sig av delplanering och metoder för att minska på mängden datorresurser som används för planeringen. SHOP behöver inte evaluera varje gren i hierarkin på grund av metoderna som finns i sammansatta uppgifter. En plan kan skapas utan att traversera hierarkin fullständigt.[11]

Delplanering har fördelen att kunna dela upp planeringen till flera olika steg, där varje planeringssteg kan utföras då det krävs. Det lönar sig inte att planera alltför långt framåt i tiden eftersom världstillståndet kan ändras, vilket leder till omplanering och bortkastade datorresurser [11]. Att dela upp arbetet i mindre delar som utförs vid olika tidpunkter minskar på sannolikheten att spelet fryser till under planeringen.

Enligt Rabin är HTN-planeraren som används i spelet Transforers: Fall of Cybertron betydligt mera effektiv än GOAP systemet som används i spelet Transformers: War for Cybertron. Orsaken till detta är att HTN använder inte heuristik eller kostnader så som sökalgoritmen A*, vilket betyder att sortering inte krävs. [11]

5.2 Skalbarhet

Med skalbarhet avses svårighetsgraden att lägga till eller att ta bort aktioner eller uppgifter från en planerare. Detta är en viktig aspekt att ta i beaktande eftersom kraven tenderar att förändras under utvecklingen.



Figur 6: Ett exempel på komponenter som är oberoende (figur till vänster) och beroende (figur till höger) av varandra. Tagen från [5].

Orkins förklarar att tillägg av krav innebär att definiera mera aktioner och att lägga till mera förvillkor till relaterade aktioner [4]. Detta är ett enkelt sätt att lägga till funktionalitet eftersom aktioner inte direkt är kopplade till varandra. Nya krav kan dock innebära att ta bort funktionalitet. Borttagning av aktioner kan göras på liknande sätt, det vill säga genom att ta bort aktioner och uppdatera förvillkor hos relaterade aktioner.

Ett problem som Orkins stötte på under utvecklingen av No One Lives Forever 2 var brist på skalbarhet hos systemet som användes för att styra agenter i spelet. Tillägg av krav betydde att återbesöka flera komponenter och försöka infoga nya beteendet [5]. Bilden till höger i figur 6 visualiserar detta problem. GOAP har inte detta problem eftersom aktionerna är oberoende av varandra. Tillägg av nya aktioner är således enkelt. Bilden till vänster i figur 6 visualiserar detta.

Hos SHOP är tillägg och borttagning av funktionalitet enkelt på grund av dess hierarkiska struktur. Metoder gör det enkelt att lägga till och ta bort grenar från hierarkin. Enligt Ghallab et al. är den huvudsakliga nackdelen hos HTN-planerare att tillägg av funktionalitet innebär att man inte bara skapar operatörer utan också metoder [6]. Förutom detta måste alla dessa komponenter kopplas ihop, vilket gör modifiering av en SHOP-hierarki mera tidskrävande jämfört med GOAP.

5.3 Modularitet

IEEE definierar modularitet som graden i vilken ett system eller datorprogram består av skilda komponenter, där förändringar till en komponent har en minimal inverkan på andra komponenter [10]. Modularitet hos en planerare syftar således på dess förmåga att uppdelas i komponenter och återanvändning av dessa komponenter. Detta har betydelse för hur lätt det är att skapa nya, modifiera och återanvända planerarens olika delar.

Enligt Orkins är strukturen av GOAP ideal för att skapa varierande karaktärer med olka beteende. Återanvänding av komponenter som styr beteendet är lätt eftersom planeraren använder en samling av aktioner för att skapa plan. Karaktärerna kan använda olika delar av den totala aktionssamlingen, vilket leder till varierande beteende mellan karaktärerna [4]. Eftersom aktioner inte är beroende av andra komponenter, kan de läggas till och tas bort ur samlingen av aktioner utan att söndra planeraren. Förvillkor och effekter hos aktionerna måste dock definieras så att planeraren har möjligheten att skapa nyttiga plan.

I en SHOP-planerare kan både enskilda komponenter och större delar av hierarkin återanvändas. Grenar i hierarkin kan betraktas som fristående delar och kan därmed återanvändas i andra uppsättningar. Metoden *Resa Till Fots* i figur 5 kan återanvändas av till exempel en mus och en soldat. Att återanvända sammansatta uppgifter är inte alltid möjligt eftersom agenter kan ha

olika förmågor. Sammansatta uppgiften *Resa Till X* i figur 5 används förmodligen inte hos både en mus och en soldat, eftersom en mus inte kan köra en bil.

5.4 Prioritering

Enligt IEEE syftar prioritering på nivån av betydelse som tilldelas ett föremål [10]. Prioritering hos en planerare syftar på dess förmåga att prioritera mellan uppgifter. I detta avsnitt tas också svårighetsgraden att definiera prioriteringen hos GOAP och SHOP i beaktande. Definiering av prioritering är en egenskap som påverkar utvecklingen samt resultatet av planeringen.

Prioriteringen i GOAP bestäms av kostnader som är definierade i aktioner. En designer ger varje aktion en kostnad för att få det önskade resultatet. Detta har dock två huvudsakliga nackdelar; brist på skalbarhet och återanvändning. Med ökande antal aktioner kan det bli svårt att definiera kostnader för nya aktioner. I värsta fall kan det leda till en kedjereaktion av förändringar i kostnader i aktioner. En aktion som återanvänds i en annan aktionssamling borde kanske ha ett annat värde på kostnad. Detta är inte dock möjligt utan att skapa en kopia av den ursprungliga aktionen.

SHOP använder metoder och traverserar hierarkin i total ordning som erbjuder en inbyggd metod att prioritera mellan uppgifter. Metoder leder till alternativa gren och total ordning bestämmer ordningen i vilken metoderna evalueras. Hierarkin tillsammans med total ordning gör att prioriteringen är enkelt att visualisera och felsöka. Justeringar i prioriteringen kan dock leda till stora förändringar i hierarkin ifall man vill återanvända en hierarki och samtidigt ändra på prioriteringen. Omvandlingsprocessen är dock enkel eftersom hierarkin lätt kan uppdelas i mindre delar och återanvändas.

5.5 Felsökning

Enligt IEEE handlar felsökning (eng. debugging) om att upptäcka, spåra och korrigera fel i en datorprogram [10]. Felsökning hos planerare syftar således på svårighetsgraden att hitta och korrigera fel. Spel utvecklas ofta under begränsad tid. Felsökning och möjliga misstag som sker under utvecklingen kan ta mer tid än förväntat. Därför är det viktigt att välja en planeringsteknik som är lätt att felsöka. Planerarens begänhet för misstag tas också i beaktande i detta avsnitt.

Orkins påstår att GOAP alltid bygger upp giltiga plan på grund av förvillkoren i aktionerna. En programmerare som kodar för hand kan göra misstag och koda en sekvens av aktioner som inte kan följa varandra [4]. Problem med detta påstående är att en programmerare också kan göra misstag i definiering av förvillkoren och effekterna. Detta kan leda till att planeringen

misslyckas eller att planeraren producerar oönskade resultat.

GOAP har kostnader som styr prioriteringen. En designer måste definiera kostnader manuellt, vilket kan leda till misstag. I SHOP däremot styrs prioritering av ordningen av komponenterna i hierarkin. Hierarkiska strukturen gör att prioriteringen är lätt att definiera och visualisera, vilket gör SHOP mindre benägen för misstag när det kommer till prioritering.

GOAP kan anses vara enklare att felsöka än SHOP eftersom GOAP har endast aktioner och mål som måste upprätthållas. SHOP har fler komponenter, men på grund av dess hierarkiska struktur är planeringsprocessen tydligare än hos GOAP. Sökalgoritmen som används i GOAP kan göra felsökningen svår eftersom tillståndet av planeringen inte är uppenbart.

Det som kan göra felsökningen svårt för både GOAP och SHOP är världstillståndet. Båda planeringsteknikerna använder sig av liknande världstillstånd för planeringen. Felsökning kan bli svårt om världstillståndet är globalt, så att flera agenter har tillgång till den. Då kan flera agenter och även spelaren påverka världstillståndet, vilket gör det svårt att kontrollera vem som har ändrat världstillståndet.

5.6 Beteende

I detta arbete syftar beteendet av en planerare på dess egenskaper som tillåter olika typer av beteende. Det lönar sig i ett tidigt skede reda ut om planeringstekniken som används stöder kraven på beteendet för agenterna. I detta avsnitt tas gruppbeteende och variation i beteende bland agenter i beaktande.

GOAP har aktioner som kan användas som en central samling som används av alla agenter i spelet. Detta innebär att alla agenter har liknande beteende i liknande situationer. För att skapa variation mellan beteendet hos agenter kan den totala aktionssamlingen delas upp i mindre aktionssamlingar. Detta ger möjlighet till varierat beteende mellan agenter. En aktionssamling kan också återanvändas för att ge en viss grupp av agenter liknande beteende.

I SHOP kan alla hierarkier användas gemensamt av alla agenter, vilket innebär att alla agenter använder sig av samma planerare. På samma sätt som i GOAP, kan samlingen av alla hierarkier i SHOP delas upp till mindre samlingar. Dessa delsamlingar kan då tilldelas grupper eller individuella agenter för att få variation i beteendet.

Enligt Neufeld et al. bör agenter ha ett sätt att kommunicera med varandra för att uppnå gruppbeteende. Detta kan uppnås till exempel med ett separat system som hanterar utbytet av information bland agenter och som bestämmer målen för varje agent. Agenter kan då använda sina egna individuella planerare eller en centraliserad planerare för att skapa plan åt sig själva [13].

Både GOAP och SHOP är kapabla till detta.

6 Sammanfattning och diskussion

Både GOAP och SHOP är goda alternativ för att hantera beteendet av agenter i spel. Även om GOAP är utvecklat specifikt för spel, finns det flera spel som har lyckats integrera SHOP eller HTN.

GOAP och SHOP kan anses vara skalbara och modulära. GOAP har aktioner som är oberoende av varandra medan SHOP har en hierarkisk struktur. Dessa egenskaper tillåter enkla tilläggnignar och borttagningar av komponenter. Återanvändning av komponenter är också enkelt eftersom ändringar till dessa planerare har minimal inverkan på andra komponenter.

Prioritering kan anses vara tydligare hos SHOP än hos GOAP eftersom SHOP har en hierarkisk struktur där prioriteringen kan enkelt visualiseras. Prioritering i GOAP kan däremot bli jobbigt eftersom en kostnad måste manuellt definieras för varje aktion.

Referenser

[1] Alex Champandard, Tim Verweij, Remco Straatman. *Killzone Multiplayer Bots*. 2009. URL: https://www.guerrilla-games.com/media/News/Files/GAIC09_Killzone2Bots_StraatmanChampandard.pdf.

[2] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Héctor Muñoz-Avila, J. William Murdock. *Applications of SHOP and SHOP2*. 2004. URL: https://apps.dtic.mil/sti/pdfs/ADA447982.pdf.

[3] John Funge Ian Millington. *Artificial intelligence for games*. Artificial intelligence for games. Morgan Kaufmann, 2009.

[4] Jeff Orkin. *Applying Goal-Oriented Action Planning to Games*. 2003. URL: https://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf.

[5] Jeff Orkin. *Three States and a Plan: The A.I. of F.E.A.R.* 2006. URL: https://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf.

[6] Malik Ghallab, Dana Nau, Paolo Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[7] Masoud Nosrati, Ronak Karimi, Hojat Allah Hasanvand. *Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches*. 2011. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.685.6608&rep=rep1&type=pdf.

[8] Nils J. Nilsson, Richard E. Fikes. *STRIPS: A NEW APPROACH TO THE APPLICATION OF THEOREM PROVING TO PROBLEM SOLVING*. 1970. URL: https://apps.dtic.mil/sti/pdfs/ADA459687.pdf.

[9] Peter Higley. *Goal-Oriented Action Planning: Ten Years Old and No Fear!* 2015. URL: https://ubm-twvideo01.s3.amazonaws.com/o1/vault/gdc2015/presentations/Higley_Peter_Goal-Oriented_Action_Planning.pdf.

[10] The Institute of Electrical and Electronics Engineers. *IEEE standard glossary of software engineering terminology - IEEE Std 610.12-1990*. 1990. URL: http://www.mit.jyu.fi/OPE/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf.

[11] Troy Humphreys. *Exploring HTN Planners through Example*. Utg. av Steve Rabin. Game AI Pro 360: Collected wisdom of game ai professionals. CRC Press, 2013. URL: http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter12_Exploring_HTN_Planners_through_Example.pdf.

[12] William Olofsson Malmberg. *UTVÄRDERING AV SÖKRIKTNINGAR I GOAL-ORIENTED ACTION PLANNING*. 2018. URL: https://www.diva-portal.org/smash/get/diva2:1218831/FULLTEXT01.pdf.

[13] Xenija Neufeld, Sanaz Mostaghim, Dario L. Sancho-Pradel, Sandy Brand. *Building a Planner: A Survey of Planning Systems Used in Commercial Video Games*. 2017. URL: https://www.is.ovgu.de/is_media/Research/Publications/IEEE_ToG_2018_Neufeld-p-4450.pdf.