Rozdział 3.3
Stosowanie behawioralnych architektur robotycznych w tworzeniu gier

Hugo Pinto i Luis Otavio Álvares
hugo@hugopinto.net

Coraz więcej twórcy gier wykazują coraz większe zainteresowanie robotyką i architekturą robotyczną. Powodem tego jest uderzające podobieństwo gier 3D do środowisk, w jakich obecnie funkcjonują mobilne roboty.

Robot sam decyduje o każdym swym kroku, bazując wyłącznie na informacji uzyskanej z środowiska i na wbudowanej wewnętrznie wiedzy. Musi szybko podejmować decyzje, funkcjonować bez nadzoru, nieprzerwanie przez długie godziny i efektywnie rozstrzygać konflikty wynikające ze sprzeczności przyjętych celów. Działając w świecie rzeczywistym, musi szybko reagować na zmiany nieustannie w nim zachodzące. Z tym wszystkim musi sobie radzić, mając do dyspozycji ograniczoną moc obliczeniową.

Akcje zachodzące w świecie współczesnych gier można by opisać w podobny sposób. Agent usytuowany jest w trójwymiarowym świecie o właściwościach zbliżonych do świata rzeczywistego, zmieniających się w sposób ciągły. Wykonuje rozmaite akcje ze swego repertuaru i oddziaływuje z innymi obiektami w czasie rzeczywistym. Jego działanie sterowane jest procesorem o ograniczonej mocy obliczeniowej, zmuszony jest szybko wybierać między sprzecznymi celami — np. między atakowaniem wroga a troską o własne bezpieczeństwo. Jedna wszakże cecha odróżnia agenta od robota: w świecie gry wyniki obserwowania środowiska (detekcji) nie są obarczone zakłóceniami (szumami), a sama detekcja ma zdecydowanie prostszy charakter (chyba że projektant sam zadecyduje inaczej).

Agent zasługujący na zaufanie gracza to agent *uczciwy* (nieoszukujący). Przymiotnik ten oznacza, że agent posiada te same informacje, co gracz (i ani trochę więcej): zauważa wyłącznie to, co dzieje się w jego polu widzenia, nie słyszy zbyt odległych zjawisk

dźwiękowych i generalnie wie tylko tyle, ile sam może wydedukować z własnych serwacji, pytań, rozmów itp. Uczciwość agenta jest warunkiem koniecznym do tego, gra była grą realistyczną. Jak za chwilę zobaczymy, taki uczciwy agent może zbudowany na wzór robota: zostaje wyposażony w sensory i urządzenia wykonawcze (aktuatory), a jego mechanizm decyzyjny zostaje zaprogramowany w postaci odrębnego wątku. W świetle tej analogii bardzo dobrą wiadomością jest fakt, że robotyka jest dziedziną dojrzałą, obfitującą w sprawdzone, znakomicie przetestowane rozwiązania. Projektant gier 3D ma do dyspozycji niewyczerpane bogactwo metod i technik, które łatwo adaptować może do swoich potrzeb.

Przegląd dorobku naukowców i projektantów w dziedzinie owej adaptacji jest naprawdę zachęcający. [Yiskis03] i [Champandard03] prezentują wykorzystanie architektury subsumpcyjnej [Brooks86], z której zrobiono użytek przy tworzeniu gry *Quake II*. [Pr...] wykorzystał rozszerzone sieci behawioralne [Dorer04] do konstrukcji agentów gry [...] *Tournament*. W niniejszym rozdziale opiszemy szczegółowo te architektury i ich zastosowania, zaprezentujemy także kilka innych obiecujących technik i rozszerzeń opisywanych systemów.

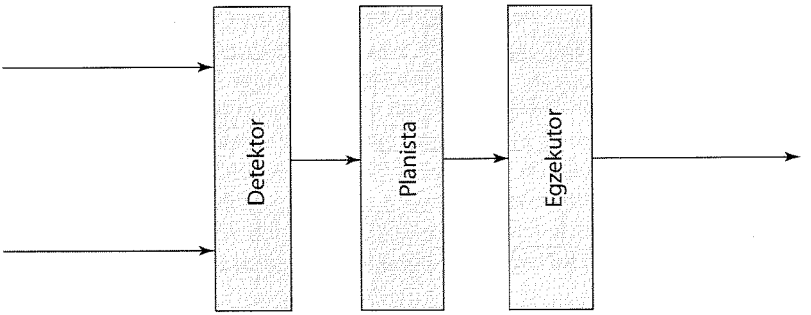
Architektura subsumpcyjna

Architektura subsumpcyjna [Brooks86] została zaprojektowana jako mechanizm sterujący działaniem autonomicznych robotów kierujących się wieloma celami, posiadających wiele możliwości detekcyjnych i zdolnych do radzenia sobie z zakłóceniami (błędami) detekcji oraz niewielkimi awariami systemu. Architekturę tę z powodzeniem wykorzystano do konstrukcji wielu rzeczywistych robotów, w tym robota poruszającego się w biurze i zbierającego puste puszki po lemoniadzie. W grze *Quake II* architektura subsumpcyjna stanowi osnowę działania agentów [Champandard03].

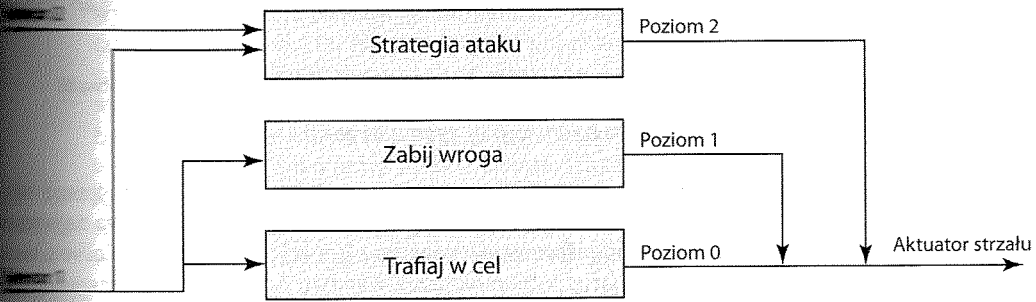
Uderzającą cechą architektury subsumpcyjnej jest sposób dekompozycji konstrukcji agenta między warstwy projektowe. Dekompozycja ta opiera się nie na tradycyjnie rozumianym przepływie informacji, lecz na klasach różnych zachowań agenta, które chcemy go wyposażyć — czyli na jego *kompetencjach*.

Załóżmy dla przykładu, że chcemy skonstruować agenta atakującego wrogów. Tradycyjna dekompozycja jego postaci, opierająca się na przepływie informacji, pokazana jest na rysunku 3.3.1. Na początku mamy moduł percepcyjny (detekcyjny), w którym informacja uzyskana z środowiska transformowana jest do postaci odpowiedniej dla modułu planisty. Wypracowany w module planisty plan działania przekazywany jest do modułu wykonawczego (egzekutora), uruchamiającego określoną sekwencję akcji. Podstawowym problemem wynikającym z takiej dekompozycji jest trudność w uwzględnianiu nowych kategorii obserwowanych zjawisk (co przekłada się na nowe sensory) oraz nowych akcji — rozmiary planisty rosną bowiem wówczas w tempie kombinatorycznym[1]. Ponadto wypracowany plan może okazać się nieadekwatny już w momencie jego realizowania, co wymaga stosowania zaawansowanych heurystyk w celu jego przepracowywania.

[1] Stopień złożoności planisty jest proporcjonalny do *iloczynu* liczby sensorów i liczby możliwych akcji — sensory i potencjalne akcje kombinowane są „każdy z każdym" — *przyp. tłum.*



Dla odróżnienia, dekompozycja opierająca się na kompetencjach agenta pokazana jest na rysunku 3.3.2. Dodawanie nowych sensorów nie stanowi tu problemu, bowiem każda z warstw wykorzystuje niewielką ich liczbę. Zmiana planu także nie jest problemem, bowiem cały system jest szybki, a jego warstwy funkcjonują równolegle.

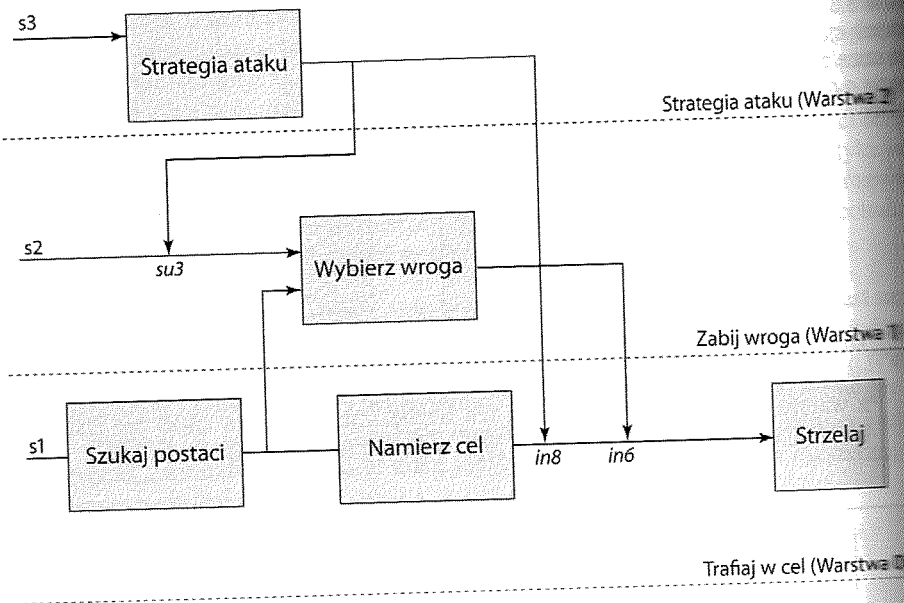


Rysunek 3.3.2. *Dekompozycja agenta-strzelca w oparciu o jego kompetencje*

Jak widzimy, warstwy na coraz wyższych poziomach odpowiadają coraz bardziej specyficznemu zachowaniu agenta, mieszczącemu się w kategoriach reprezentowanych przez warstwę niższą. I tak zabijanie wroga (Poziom 1) jest szczególnym przypadkiem trafiania do celu (Poziom 0), zaś szczegóły owej eksterminacji określają strategię ataku (Poziom 2). To, co reprezentuje warstwa wyższa, *zalicza się* więc (po angielsku — *subsumes*) do tego, co reprezentowane jest przez warstwę bezpośrednio niższą, stąd nazwa architektury.

Każda warstwa architektury subsumpcyjnej może być z kolei zdekomponowana na kilka modułów behawioralnych, reprezentujących poszczególne elementy kompetencji agenta. Na rysunku 3.3.3 kompetencja `Trafiaj w cel` składa się z trzech modułów: `Szukaj postaci`, `Namierz cel` i `Strzelaj`.

Moduły warstw wyższych mogą „podkładać" własną treść w charakterze informacji wejściowej dla modułów w warstwach niższych i (lub) zastępować informację wyjściową produkowaną przez nie. Warstwy wyższe mogą też otrzymywać informację wejściową od modułów z warstw niższych. Na rysunku 3.3.3 widzimy sposób połączenia warstwy `Zabij wroga` z warstwą niższą `Trafiaj w cel`, a także połączenie warstwy najwyższej (`Strategia ataku`) z dwiema warstwami niższymi. Zauważmy, że przedstawiona architektura obejmuje trzy sensory (`s1`, `s2` i `s3`) i jeden aktuator (`Strzelaj`). Warstwa



Rysunek 3.3.3. *Szczegóły architektury subsumpcyjnej agenta-strzelca*

`Zabij wroga` pobiera informację wejściową z sensora `s2` i (ewentualnie) nadpisuje ście z warstwy `Namierz cel` własną informację o położeniu wroga. Warstwa najwyższa (`Strategia ataku`) otrzymuje informację z sensora `s3`, a jej informacja wyjściowa nadpisywać zarówno informację pobieraną przez warstwę `Zabij wroga` z sensora `s2`, również informację produkowaną przez moduł `Wybierz wroga`. Całościowe funkcjonowanie tej architektury można podsumować następująco: w warstwie `Trafiaj` wybrany zostaje losowo cel strzału. W warstwie `Zabij wroga` sprawdza się, czy w polu dzenia znajduje się jakiś nieprzyjacielski wojownik — jeżeli tak, to w polu z informacją pochodzącą z sensora `s2` wybrany zostaje jako cel. Warstwa `Strategia ataku` może zmieniać działanie modułu `Wybierz wroga`, nadpisując informację w z jego kanałów wejściowych; może ona także nadpisywać informację produkowaną przez warstwę najniższą.

Trzy cechy projektowe architektury subsumpcyjnej warte są krótkiego komentarza. Po pierwsze, działanie poszczególnych modułów nie zmienia się po ich zaimplementowaniu — zmienia się jedynie ich informacja wejściowa i wyjściowa. Po drugie, moduły warstw wyższych zależne mogą być jednie od modułów warstw niższych, co automatycznie prowadzi do projektowania wstępującego (ang. *bottom-up*). Po trzecie, każdy z modułów działa w sposób asynchroniczny, dzięki czemu architektura ta nadaje się idealnie do środowisk wielowątkowych i wieloprocesorowych.

Zauważmy, że z organizacji tego systemu wynika jego niezawodność, bo jeśli moduły z wyższych warstw przestanie funkcjonować, jej funkcje mogą przejąć warstwy niższe. Jeżeli przykładowo któryś z agentów nie będzie w stanie (z jakiegokolwiek powodu) ułożyć strategicznego planu ataku, może atakować konsekwentnie każdego nieprzyjaciela, który znajdzie się w jego polu widzenia.

Skoro mamy już pojęcie o tym, czym jest architektura subsumpcyjna, jak funkcjonuje i jak można ją projektować, zastanówmy się, jakie wypływają stąd wnioski praktyczne dla projektantów gier. [Yiskis03] sugeruje budowanie niższych warstw z intencją osiągania celów doraźnych i implementowanie celów długofalowych w warstwach wyższych. Proponuje konsekwentne wymuszanie reguły, zgodnie z którą kompetencje wysokopoziomowe realizowane są wyłącznie poprzez wymianę komunikatów z modułami niższych warstw, wykonującymi niskopoziomowe rodzaje zachowania. [Champandard03] formułuje natomiast wskazówkę, by kompetencje niskopoziomowe angażowane były jako domyślne, zaś wysokopoziomowe traktowane były jako wyjątki od domyślnej reguły.

[Yiskis03] podaje przykład zastosowania architektury subsumpcyjnej do sterowania animacją postaci. Prezentowana architektura składa się z czterech warstw, o nazwach (poczynając od najniższej warstwy) `Movement` (ruch), `Action` (akcja), `Behavior` (zachowanie) i `Strategy` (strategia). Warstwa najniższa — `Movement` — odpowiedzialna jest za bieżące odgrywanie animacji i odzwierciedlanie najważniejszych ograniczeń fizycznych; jeśli (przykładowo) agent, biegnąc, niespodziewanie uderza w jakąś przeszkodę, wyświetlane jest najpierw jego zatrzymanie (ang. *stop*), a następnie „odbicie" od przeszkody (ang. *go back*). Na warstwę `Action` składają się kompletne bloki ruchów podstawowych, implementowanych w warstwie `Movement`; obowiązuje tu (wspomniana wcześniej) żelazna reguła, że warstwa `Action` komunikuje się z systemem animacji wyłącznie poprzez warstwę `Movement`, nigdy bezpośrednio. Złożone akcje warstwy `Action` wykonywane są tylko przez krótkie chwile. Elementy kolejnej warstwy — `Behavior` — to zachowania wykonywane w sposób ciągły, w większości będące cyklicznymi powtórzeniami działań z warstwy `Action`. Wreszcie, warstwa najwyższa — `Strategy` — reprezentuje „wysokopoziomowe" plany strategiczne.

Ponieważ warstwa najniższa (`Movement`) pełni rolę swoistego interfejsu między systemem animacji a warstwami wyższymi, zwiększając liczbę tych ostatnich, nie musimy w system animacji w ogóle ingerować.

[Champandard03] użył architektury subsumpcyjnej do zbudowania robota występującego w grze *Quake II*. Jego konstrukcja składała się z siedmiu warstw, o sugestywnych nazwach (w kolejności od warstwy najniższej): `Explore` (eksploracja), `Investigation` (badanie), `Gather` (wnioskowanie), `Attack` (atakowanie), `Hunt` (pościg), `Evade` (uniki) i `Retreat` (odwrót) — można podejrzewać, że robot ów raczej nie grzeszy odwagą. Szczegóły jego implementacji, wraz z kompletnym kodem źródłowym, znajdą czytelnicy w cytowanej publikacji.

Modularność i efektywność architektury subsumpcyjnej oraz jej odporność na błędy czynią ją doskonałym kandydatem dla implementowania akcji odgrywanych w czasie rzeczywistym. Głównym jej ograniczeniem, w porównaniu z innymi architekturami kategorii behawioralnej, jest konieczność jawnego wyspecyfikowania wszystkich połączeń między modułami oraz sztywny system hierarchicznych powiązań między kompetencjami. W dalszym ciągu rozdziału pokażemy, jak można uporać się z tym ograniczeniem.

Rozszerzone sieci behawioralne

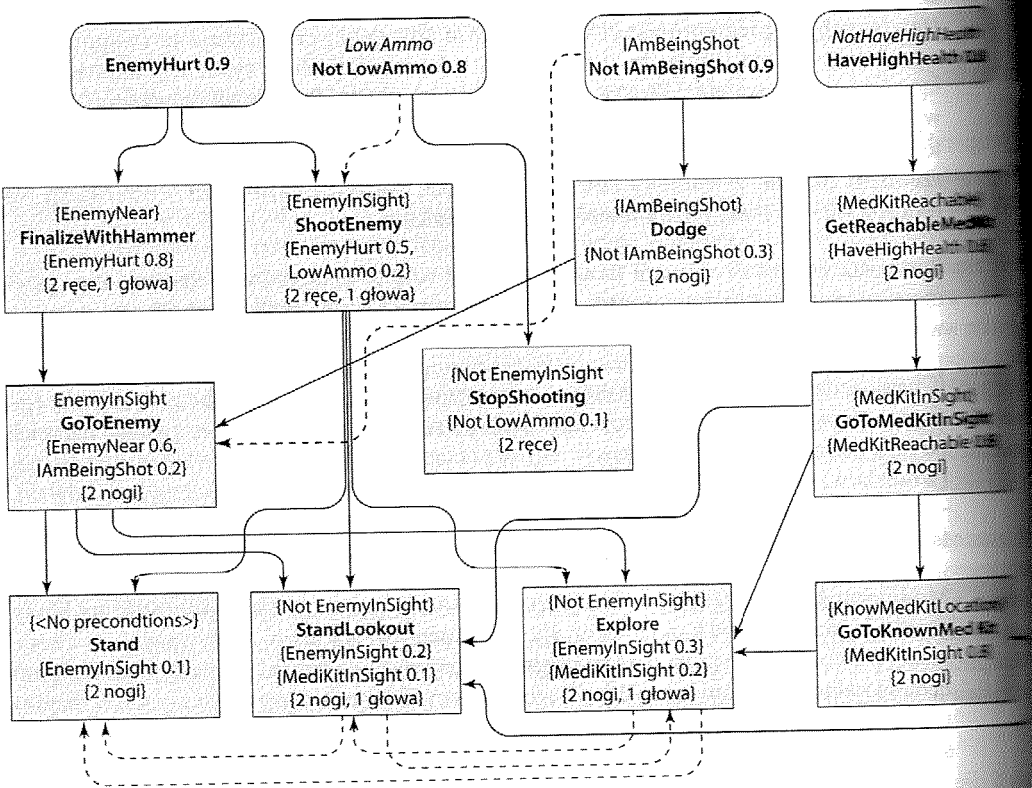
Rozszerzone sieci behawioralne [Dorer04] (ang. *Extended Behavior Networks — EBN*) zaprojektowane zostały jako mechanizm selekcyjny akcji robota futbolowego RoboCup; jego drużyna zdobyła wicemistrzostwo swej ligi [Dorer99], a w następnym roku również

odnosiła sukcesy w rozgrywkach [Dorer00]. Oprócz sukcesów na polu rozgr
bolowych, rozszerzone sieci behawioralne znalazły zastosowanie (z obie
rezultatami) w grze *Unreal Tournament* [Pinto05-b].

Rozszerzone sieci behawioralne to najnowszy rezultat ewolucji sieci beha
[Maes89]; inne kierunki tej ewolucji opisane są w pracach [Rhodes96], [Goetz97] i [

Rozszerzona sieć behawioralna może być rozpatrywana jako zbiór połączonych
i celów, pomiędzy które rozpraszana jest energia aktywacyjna — poczynając
i przepływając przez poszczególne moduły. W każdym kroku symulacji do wy
nia wybierane są moduły posiadające najwyższy zasób energii i najwyższą
walność", pod warunkiem dostępności niezbędnych do tego zasobów.

Na strukturę rozszerzonej sieci behawioralnej składają się: zbiór modułów be
nych, zbiór celów, zbiór połączeń modułów z celami i innymi modułami, zbiór
i zbiór parametrów sterujących. Na rysunku 3.3.4 przedstawiony jest schemat
hawioralnej sterującej zachowaniem botów[2] w grze *Unreal Tournament*.



Rysunek 3.3.4. *Sieć behawioralna sterująca zachowaniem botów w grze Unreal Tournament*

[2] Patrz *http://pl.wikipedia.org/wiki/Bot_%28program%29 — przyp. tłum.*

Cel definiowany jest poprzez zbiór trzech elementów: propozycji pewnego warunku, którego spełnienia się oczekuje, potencjału (ang. *strength* — wyrażanego w postaci liczby rzeczywistej) i propozycji alternatywnej, zwanej warunkiem adekwatności (ang. *relevance condition*). Potencjał odzwierciedla istotność celu niezależnie od kontekstu, podczas gdy warunek adekwatności wyraża jego istotność w kontekście aktualnej sytuacji. Wyrażenie istotności celu jako pary dwóch wartości — statycznej i dynamicznej — pozwala dostosowywać tę istotność do sytuacji, w której aktualnie znajduje się agent. Efektywna istotność celu jest iloczynem tych dwóch wartości. Na rysunku 3.3.4 cel `HaveHighHealth` posiada potencjał 0,8 i warunek alternatywny `NotHaveHighHealth`. Cel ten staje się tym bardziej istotny, im słabsza staje się kondycja robota. Wartościowanie istotności za pomocą dziedziny ciągłej, jaką są liczby rzeczywiste, pozwala na zastosowanie koncepcji logiki rozmytej.

Każdy moduł behawioralny stanowi kombinację listy warunków wstępnych, akcji, listy efektów i listy zasobów. Pierwsza ze wspomnianych list stanowi koniunkcję warunków wstępnych niezbędnych do wykonania modułu; lista efektów to koniunkcja warunków (będących zwykle negacjami), jakich spełnienia oczekujemy w wyniku wykonania modułu. Elementy listy zasobów mają z kolei postać par (zasób, ilość) odzwierciedlających ilości konkretnych zasobów, jakich agent potrzebuje do swego działania.

Połączenia w sieci behawioralnej opierają się na wspomnianych efektach, warunkach wstępnych i celach. Połączenia *wsteczne* (ang. *predecessor links*) to połączenia skierowane od modułu lub celu B do modułu A, wiążące elementy wspólne listy efektów modułu A z listą warunków wstępnych modułu B, przy czym łączone elementy mają tę samą wartość logiczną (`true` albo `false`). Przykładem tego jest połączenie celu `EnemyHurt` z modułem `ShootEnemy` na rysunku 3.3.4. Analogiczne połączenie między elementami o różnej wartości logicznej nazywa się połączeniem *kolizyjnym* (ang. *conflict link*); przykładem takiego połączenia na rysunku 3.3.4 jest połączenie celu `NotLowAmmo` z modułem `ShootEnemy`. Połączenia kolizyjne reprezentują wyczerpywanie energii ze swych modułów docelowych, podczas gdy połączenia wsteczne — przeciwnie — wzbogacają energię swych modułów docelowych. W ten sposób dany moduł lub cel może zwiększać lub zmniejszać szansę wykonywania poszczególnych modułów, zależnie od warunków zachodzących w środowisku zewnętrznym.

Na schematach, takich jak przedstawiony na rysunku 3.3.4, cele reprezentowane są przez prostokąty z zaokrąglonymi narożnikami, zaś modułu — przez prostokąty z narożnikami ostrymi. Połączenia wsteczne reprezentowane są przez linie ciągłe, zaś połączenia kolizyjne — przez linie przerywane; łącza zasobowe reprezentowane są przez linie w kształcie łuków. W dwóch pierwszych przypadkach kierunki połączeń odzwierciedlają przepływ aktywacji, w trzecim wskazują na zależność modułu od określonych zasobów.

Wybór (selekcja) wykonywanych akcji opiera się na ciągu przeplatających się wzbudzeń (ang. *excitations*) i wstrzymywań (ang. *inhibitions*) wynikających z przepływu energii aktywacyjnej pomiędzy węzłami sieci. Do wykonywania wybierane są mianowicie moduły o najwyższym priorytecie wykonawczym, pod warunkiem dostępności (w wystarczającej ilości) zasobów niezbędnych do ich wykonania; ów priorytet wykonawczy jest iloczynem wartości wyrażających energię aktywacji i wykonywalność modułu (co odpowiada koniunkcji w logice rozmytej). Szczegółom tego mechanizmu przyjrzymy się dokładniej w rozdziale 3.5, tutaj poprzestając jedynie na opisie właściwości sieci EBN.

Sieci EBN mają te same zalety, co architektury subsumpcyjne: są szybkie, modułowe, łatwo poddają się projektowaniu przyrostowemu (ang. *incremental design*) i są [illegible] niezawodne. Ponadto selekcja akcji odbywa się równolegle i asynchronicznie, co [illegible] turalny sposób sprzyja efektywnej implementacji wielowątkowej i wieloprocesorowej. Jednak EBN przewyższają pod kilkoma względami architektury subsumpcyjne i radzą sobie z ich niektórymi ograniczeniami.

Ponieważ architektura EBN jest de facto architekturą połączeń między warunkami wstępnymi i warunkami końcowymi wykonań modułów, nie jest już zdeterminowana na etapie projektowania, jak miało to miejsce w architekturach subsumcyjnych. Projektanci muszą się tylko troszczyć o zaimplementowanie poszczególnych celów i modułów dualnych zachowań; interakcje między nimi wyznaczane są przez wartości poszczególnych warunków i algorytm selekcyjny.

Sieci EBN są też bardziej odporne na błędy. Na gruncie architektury subsumpcyjnej odporność ta wynikała z faktu, że awaria w którejś z wyższych warstw nie paraliżowała działania warstw niższych; jednak awaria w warstwie najniższej automatycznie paraliżowała zdolność działania wszystkich warstw. Stawia to przed projektantami i programistami niebagatelne wymagania dotyczące przewidywania wielu różnych sytuacji w celu zapobiegania takim awariom. Następstwem awarii jakiegoś modułu w sieci EBN jest natomiast automatyczny wybór innego modułu, którego wykonanie prowadzi do spełnienia tego samego warunku docelowego.

Inne zalety sieci EBN to kontekstowe uwarunkowanie celów, wartościowanie propozycji za pomocą liczb rzeczywistych i natura interakcji między modułami. Uzależnienie kontekstowe celów pozwala na łatwiejsze partycjonowanie przestrzeni zdarzeń (sytuacji) i bardziej naturalne podejście do projektowania agentów. Możemy śmiało formułować cele, które niekoniecznie są najważniejsze, lecz zyskują na istotności jedynie w ściśle określonych warunkach. Przykładowo w konfiguracji przedstawionej na rysunku 3.3.4 agent powinien zatroszczyć się o wysoki poziom swej kondycji (cel `HaveHighHealth`), jeśli aktualny jej poziom jest nienaturalnie niski. Podobnie ma się rzecz z amunicją (cel `NotLowAmmo`) — agent nie musi myśleć o amunicji tak długo, jak długo ma jej pod dostatkiem.

Nie istnieje także ścisła hierarchia celów i zachowań, co stanowi o szerokich możliwościach adaptacyjnych sieci EBN — to, który moduł wybrany zostanie do wykonania, zależne jest tylko od rozkładu poziomu „wykonywalności" i zakumulowanej energii aktywacji wśród poszczególnych modułów. Energia aktywacji jest funkcją celów agenta, jego bieżącej sytuacji oraz (ogólnie mówiąc) istnienia innych modułów. Hierarchiczna zależność modułów zmienia się stosownie do bieżącej aktywności agenta. Wartościowanie warunków za pomocą liczb rzeczywistych pozwala w naturalny sposób na odzwierciedlanie wielkości ciągłych z natury, takich jak odległość, siła i kondycja.

Budując sieć EBN dla danego agenta, rozpoczynamy od zdefiniowania jego celów. Następnie tworzymy moduły realizujące te cele, a ostatnim etapem jest strojenie parametrów decydujących o rozprowadzaniu energii aktywacji, w celu osiągnięcia ogólnego charakteru pożądanego zachowania. Dwa pierwsze etapy opisane są szczegółowo w rozdziale 3.4, trzecim zajmiemy się w rozdziale 3.5.

Sieci EBN mają ponadto tę interesującą własność, że możemy drastycznie przekształcić sposób zachowania się agenta, zmieniając wartości zaledwie kilku parametrów sterujących dystrybucją energii aktywacyjnej — czyli jej ilością zakumulowaną w poprzednim

...ku i minimalną jej ilością niezbędną do wykonywania poszczególnych modułów. ...atwia to pracę projektantowi, który manipulując wartościami poszczególnych parametrów, wygodnie wprowadzać może każdą z pożądanych zmian z osobna. Szerzej ...mówimy tę kwestię w rozdziale 3.5, zainteresowani nią czytelnicy mogą też przeczytać pracę [Pinto05-b].

...ykorzystując sieć EBN na potrzeby *Unreal Tournament* [Pinto05-a], przeprowadziliśmy szereg eksperymentów mających na celu ocenę jakości akcji selekcyjnej i efektywności implementacji agenta. Jeśli chodzi o wspomnianą efektywność, porównaliśmy agenta zaimplementowanego na bazie EBN z interaktywnym agentem o strukturze ściśle hierarchicznej. Ten pierwszy okazał się znacznie efektywniejszy ze względu na lepszą politykę wyboru akcji do wykonania — politykę uwidaczniającą niezawodność, reaktywność, łańcuchowanie akcji i właściwą kombinację akcji zrównoleglonych. Eksperymenty te ...mówiliśmy szczegółowo w rozdziale 3.5.

...sja

Skoro opisaliśmy rozmaite zalety architektury subsumpcyjnej oraz sieci EBN i właściwe im obu ograniczenia, nie możemy nie wspomnieć o bardzo istotnym ograniczeniu charakterystycznym wyłącznie dla sieci EBN — nieobecności zmiennych.

Niemożność używania zmiennych prowadzi natychmiast do znacznie mniejszej ogólności i elastyczności modułów — wprowadzając do scenariusza (przykładowo) nowy rodzaj uzbrojenia, musimy utworzyć odrębny moduł. Nie ma jednak tego złego, co by na dobre nie wyszło: ograniczenie to jednocześnie sprawia, że moduły są szybsze, i zmniejsza ich zapotrzebowanie na zasoby. Moduły bardziej ogólne, parametryzowane za pomocą zmiennych, w znacznie większym stopniu obciążałyby procesor(y).

Mimo to prowadzone są badania, których celem jest próba pogodzenia elastyczności wynikającej z używania zmiennych z najbardziej istotnymi własnościami systemów behawioralnych. Architektura zaproponowana w [Horswill99] dopuszcza używanie zmiennych w ograniczonym stopniu; nie mamy informacji dotyczących jej zastosowania w tworzeniu gier, jakkolwiek na pewno spróbujemy ją wykorzystać w przyszłych wersjach *Unreal Tournament*. Podobne rozwiązanie opisane jest także w [Rhodes96]; łącząc je z modelem sieci EBN, otrzymujemy model znacznie bardziej elastyczny.

...kluzja

W niniejszym rozdziale opisaliśmy zastosowanie podczas tworzenia gier dwóch mechanizmów z dziedziny robotyki — architektury subsumpcyjnej i rozszerzonych sieci behawioralnych (EBN). Mechanizmy te cechują się dużą niezawodnością, reaktywnością, szybkością i modularnością; ułatwiają też projektowanie przyrostowe, implementację modularną i zrównoleglenie obliczeń.

W ramach architektury subsumpcyjnej zmuszeni jesteśmy a priori wyspecyfikować wszystkie połączenia między modułami i warstwami; w sieci behawioralnej interakcje między modułami sterowane są całkowicie przez jej architekturę.

Hierarchia zachowań, ściśle zdeterminowana na gruncie architektury subsumpcyjnej, w sieci behawioralnej ma naturę adaptacyjną. Ceną płaconą za tę elastyczność jest dłuższy

czas wyboru akcji, choć sieci EBN i tak są mechanizmem szybkim, zwłaszcza w porównaniu z systemami planistycznymi[3].

Wyróżniającą cechą sieci EBN jest możliwość wbudowywania w zachowanie agenta podstawowych cech osobowości za pomocą prostych operacji konfiguracyjnych. Ich wielkie zapotrzebowanie na zasoby czyni systemy behawioralne interesującymi z punktu widzenia przetwarzania informacji i podejmowania decyzji w czasie rzeczywistym.

Literatura cytowana

[Brooks86] R. Brooks „A Robust Layered Control System for a Mobile Robot". *Journal of Robotics and Automation*, t. RA-2, nr 1, 1986.

[Champandard03] A. Champandard, *AI Game Development*. New Riders Publishing, 2003.

[Dorer99] K. Dorer „Extended Behavior Networks for the Magma Freiburg Team". RoboCup-99 Team Descriptions for the Simulation League.

[Dorer00] K. Dorer „Concurrent Behavior Selection in Extended Behavior Networks". Team Description for RoboCup2000, Melbourne, 2000.

[Dorer04] K. Dorer „Extended Behavior Networks for Behavior Selection in Dynamic and Continuous Domains". *Proceedings of the ECAI Workshop on Agents in Dynamic and Real-time Environments*. 22/23 sierpnia 2004, Valencia, Hiszpania.

[Goetz97] P. Goetz „Attractors in Recurrent Behavior Networks". Praca doktorska, University of New York, Buffalo 1997.

[Horswill99] I. D. Horswill i R. Zubek „Robot architectures for believable game agents". AAAI Spring Symposium on Artificial Intelligence and Computer Games, AAAI Technical Report SS-99-02, 1999.

[Maes89] P. Maes „How to do the Right Thing". *Connection Science Journal*, t.1, nr 3, 1989.

[Nebel03] B. Nebel i Y. Babovich „Goal-Converging Behavior Networks and Self-Solving Planning Domains, or: How to Become a Successfull Soccer Player". s.l. *IJCAI03*, 2003.

[Pinto05-a] Hugo Pinto „Designing Autonomous Agents for Computer Games with Extended Behavior Networks: An Investigation of Agent Performance, Character Modelling and Action Selection in Unreal Tournament". Praca magisterska, Universidade Federal do Rio Grande do Sul, Brazylia, 2005.

[Pinto05-b] H. Pinto i L. O. Alvares „Extended Behavior Networks and Agent Personality: Investigating the Design of Character Stereotypes in the Game Unreal Tournament". *Proceedings of the Fifth International Working Conference on Intelligent Virtual Agents*, Kos, Grecja, 2005.

[Rhodes96] Bradley Rhodes „PHISH-Nets; Planning Heuristically in Situated Hybrid Networks". Praca magisterska, Massachusetts Institute of Technology, Boston, 1996.

[Yiskis03] E. Yiskis „A Subsumption Architecture For Character-Based Games". *AI Game Programming Wisdom II*, Charles River Media, 2003.

[3] Patrz *http://pl.wikipedia.org/wiki/Advanced_Planning_System — przyp. tłum.*