Uwaga, eksperyment Sperlinga

Konrad Miazga

Poznan University of Technology

Poznań, 2018

VISUAL-LOCATION0-0-1

VALUE	TEXT
COLOR	BLACK
HEIGHT	10
WIDTH	7
SCREEN-X	130
SCREEN-Y	156
DISTANCE	1080
KIND	TEXT
SIZE	0.19999999

Filtrowanie zapytań

```
< screen-y 166
```

screen-y mniejsze niż 166, a więc **powyżej** 166-ego rzędu pikseli.

```
>= screen-y 166
```

screen-y większe bądź równe 166, a więc **poniżej** (lub na) 166-ego rzędu pikseli.

```
> screen-x 50
```

screen-x większe bądź równe 50, a więc **na prawo** od 50-ej kolumny pikseli.

```
- screen-x 50
```

screen-x różne od 50, a więc nie w 50-ej kolumnie pikseli.

Filtrowanie zapytań

screen-y lowest

Prośba o obiekt z najniższym *screen-y*, a więc **najwyżej** w oknie.

screen-y highest

Prośba o obiekt z najwyższym *screen-y*, a więc **najniżej** w oknie.

screen-x highest

Prośba o obiekt z najwyższym *screen-x*, a więc najbardziej po prawej stronie w oknie.

width highest

Prośba najszerszy obiekt w oknie.

Filtrowanie zapytań

Gdy łączymy filtry obiektywne i relatywne (lowest/highest), filtry obiektywne zawsze zostaną rozpatrzone jako pierwsze, a następnie filtry relatywne w podanej kolejności.

```
+visual-location>
   width highest
   screen-x lowest
   color red
```

Powyższe zapytanie odfiltruje najpierw wszystkie obiekty (lokacje) które nie są czerwone (filtr obiektywny), następnie wybierze najszerszy z nich i jeśli jest więcej niż jeden taki obiekt (lokacja), wybierze ten który jest najbardziej po lewej stronie.

Filtrowanie zapytań

Wartość *current* pozwala nam się odnosić do obiektu na którym aktualnie się skupiamy (do którego jako ostatniego przenieśliśmy naszą uwagę za pomocą `cmd move-attention`).

```
+visual-location>
  screen-y current
```

Powyższa reguła poprosi zatem o lokację znajdującą się na tej samej wysokości co obecnie obsługiwany obiekt.

Wartość *current* można łączyć z modyfikatorami jak każdą inną wartość liczbową.

```
  >= screen-y current
```

Filtrowanie zapytań

Jeśli poprzedzimy wartość slotu symbolem & (np. &*value*), utworzymy zmienną. Zmienna ta w każdym swoim wystąpieniu wewnątrz zapytania musi przyjmować taką samą (acz dowolną) wartość.

```
+visual-location>
   isa visual-location
   height &height
   width &height
```

Powyższe zapytanie poprosi więc o lokację z obiektem którego wysokość i szerokość są sobie równe. Nie definiuje ono jednak jaka dokładnie powinna być jego szerokość / wysokość.

Jaka jest różnica między =*shoe* a &*shoe*?

Filtrowanie zapytań

Wartość *current* może być użyta również do **parametru** *:nearest*.

```
+visual-location>
   :nearest current
```

Powyższe zapytanie poprosi więc o lokację najbliższą naszej aktualnie obsługiwanej lokacji (odległość euklidesowa).

```
+visual-location>
   :nearest =some-location
```

Powyższe zapytanie poprosi o lokację najbliższą pewnej lokacji przechowywanej w zmiennej *=some-location*.

Filtrowanie zapytań

Zadanie 1: Polityka czytania tekstu

Zaproponuj reguły które zapewnią, iż model odczytywać będzie litery przedstawione na ekranie od lewej do prawej strony, przechodząc do nowego wiersza gdy konieczne.

Moduł słuchowy (Aural)

- Modeluje ośrodki słuchowe mózgu.
- Podobnie jak moduł wizualny, posiada dwa bufory:
  - aural location („gdzie” jest dźwięk),
  - aural („czym” jest dźwięk).
- Może rozpoznawać różne rodzaje dźwięku (ton, litera, słowo).
- Informacja dźwiękowa dostępna jest z opóźnieniem które zależy od rodzaju dźwięku (np. szybciej otrzymamy informację o literze niż o słowie).

Eksperyment Sperlinga

Badanym na 50ms (0.05s) prezentowana jest siatka liter (4x3), tak jak poniżej:

Y	M	C	A
R	X	P	O
Z	N	G	T

Następnie z opóźnieniem (0s, 0.15s, 0.3s albo 1s po pojawieniu się siatki) emitowany jest dźwięk o jednej z trzech częstotliwości. W zależności od wysokości dźwięku (500Hz, 1000Hz, 2000Hz), badany ma podać zapamiętane litery z jednego z trzech wierszy.

Eksperyment Sperlinga – implementacja

Eksperyment zaimplementowany w dla modelu ACT-R różni się od oryginału kilkoma detalami:

- Nie posiada on wersji do testowania ludzi. Wynika to z trudności z zapewnieniem właściwego brzmienia odpowiednich tonów na różnych komputerach i w różnych wersjach Lispa.
- Moduł wizualny ACT-R nie posiada wizualnej pamięci ikonicznej. W związku z tym, symuluje ją pokazując modelowi ekran przez chwilę dłużej – przez losowy czas, od 0.9 do 1.1 sekund.

Eksperyment Sperlinga

Zadanie 2: Uruchomienie zadania Sperlinga

Uruchom zadanie Sperlinga dla sygnału dźwiękowego pojawiającego się z opóźnieniem wynoszącym 0.15s: *(do-sperling-trial .15)*

Plik z modelem:
...\ACT-R Standalone Environment\tutorial\unit3\sperling.lisp

Przebieg modelu

Przebieg modelu można przedstawić w następujących krokach:

1. Zapamiętuj losowe litery.
2. DŹWIĘK!
3. Zapamiętuj litery tylko z odpowiedniego rzędu.
4. OBRAZ ZNIKA / ZAPAMIĘTANO CAŁY RZĄD
5. Wypisz litery z odpowiedniego rzędu
6. Wciśnij spację aby zakończyć eksperyment.

Chunk startowy

```
(goal
      ISA read-letters
      state attending
      upper-y 0
      lower-y 300
)
```

Krok 1a: Zapamiętuj losowe litery

Za znajdowanie liter odpowiadają trzy reguły produkcji: *attend-low*, *attend-medium* i *attend-high*. Działają one w analogiczny sposób, więc skupimy się na jednej z nich:

```
(P attend-medium
=goal>
   ISA read-letters
   state attending
=visual-location>
   ISA visual-location
   > screen-y 154
   < screen-y 166
?visual>
   state free
==>
=goal>
   location medium
   state encode
+visual>
   cmd move-attention
   screen-pos =visual-location
)
```

Krok 1a: Zapamiętuj losowe litery

```
=visual-location>
   ISA visual-location
   > screen-y 154
   < screen-y 166
```

- Używamy tutaj modyfikatorów ($>$, $<$) aby dokonać porównań na wartościach liczbowych slotu *screen-y*
- `> screen-y 154` oznacza sprawdzenie czy wartość slotu *screen-y* bufora visual-location jest większa niż 154.
- Działanie na wartościach liczbowych ma sens w tym kontekście! Sprowadza się bowiem do sprawdzenia czy obserwowany obiekt znajduje się na pewnym obszarze ekranu!
- Niepoprawnym byłoby natomiast zastosowanie takiego porównania do konceptów liczb przechowywanych w pamięci.

Buffer stuffing

Zauważ, że reguła produkcji *attend-medium* została odpalona już w chwili 0.000, gdy jeszcze nie kierowaliśmy żadnego zapytania do bufora *visual-location*!

```
0.000 VISION SET-BUFFER-CHUNK VISUAL-LOCATION
VISUAL-LOCATION1-0 REQUESTED NIL
```

Wynika to z mechanizmu zwanego *buffer suffing* (nie mylić z *visual re-encoding*). Gdy bufor *visual-location* jest pusty, może on samoistnie znajdować nowe lokacje!

Lokacje znalezione metodą *buffer stuffing* są automatycznie usuwane z bufora, jeśli nie zostały użyte przez ostatnie (domyślnie) pół sekundy.

Z jakim zjawiskiem związany jest *buffer stuffing*?

Krok 1b: Zapamiętuj losowe litery

Gdy znajdziemy już literę, należy ją odkodować, zapamiętać i zlecić poszukiwania następnej:

```
(P encode-row-and-find
=goal>
   ISA read-letters
   location =pos
   upper-y =uy
   lower-y =ly
=visual>
==>
=visual>
   status =pos
-visual>
=goal>
   location nil
   state attending
+visual-location>
   :attended nil
   > screen-y =uy
   < screen-y =ly
)
```

Krok 1b: Zapamiętuj losowe litery

```
=visual>
   status =pos
-visual>
```

- W slocie *location* naszego celu, przechowujemy informację o etykiecie rzędu związanego z obecnie obsługiwaną literą.
- Do chunka z informacją o literze dodajemy slot *status* o wartości przechowywanej aktualnie w slocie *location* naszego celu.
- Zapytaniem `-visual>` czyścimy bufor visual.
- Jak pamiętamy, chunki usuwane z buforów trafiają do pamięci deklaratywnej. w ten sposób zapamiętaliśmy informację o literze i związanym z nią rzędzie.

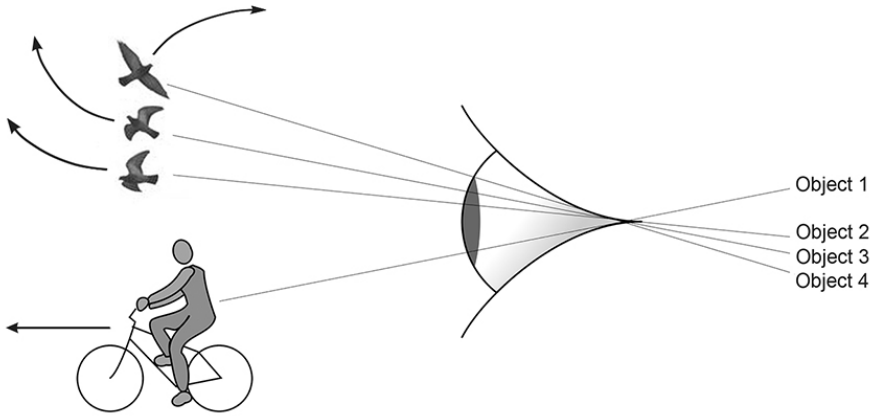
Krok 1b: Zapamiętuj losowe litery

```
+visual-location>
   :attended nil
   > screen-y =uy
   < screen-y =ly
```

- `:attended nil` oznacza, iż prosimy o dostarczenie nam lokacji na ekranie która jeszcze nie była przez nas obsługiwana.
- Następnie precyzujemy, iż interesuje nas litera z zakresu zapamiętanego w naszym chunku celu w slotach *upper-y* i *lower-y*.
- Początkowo zakres ten obejmuje całe okno.

Finsty

ACT-R implementuje teorię FINST opracowaną przez Zenona Phylyshyna. Teoria ta zakłada, iż układ wzrokowy może automatycznie „obsługiwać" (czyli np. śledzić ruch) jedynie ograniczonej liczby obiektów naraz. Jeden **finst** (FINger of INSTanatiation) może być interpretowany jako palec podążający za obiektem na którym został położony.



Liczba oraz „czas życia" finstów są ograniczone i definiowane parametrami *:visual-num-finsts* oraz *:visual-finst-span* (domyślnie 4 finsty i 3 sekundy). Lokalizacje obsługiwane przez finsty będą przyjmować wartość *:attended t*.

Uwaga na temat finstów...

Aby uniknąć problemów związanych z nadpisywaniem finstów (domyślnie tylko 4), w naszym modelu ustawimy wysoką liczbę finstów. **Nie jest to realistyczne rozwiązanie** ale upraszcza nasz model, w związku z czym się na to godzimy...

Krok 2: DŹWIĘK!

Obsługa sygnału dźwiękowego przeprowadzana będzie w dwóch krokach. Najpierw odpalona zostanie reguła *detected-sound*...

```
(P detected-sound
=aural-location>
?aural>
   state free
==>
+aural>
   event =aural-location
)
```

Krok 2: DŹWIĘK!

...a następnie jedna z trzech reguł: *sound-respond-low*, *sound-respond-medium* lub *sound-respond-high*. Są one do siebie analogiczne, więc skupimy się tylko na jednej z nich:

```
(P sound-respond-low
=goal>
   ISA read-letters
   tone nil
=aural>
   ISA sound
   content 500
==>
=goal>
   tone low
   upper-y 205
   lower-y 215
)
```

Zmiana wartości slotów *upper-y* i *lower-y* oznacza, iż od tego momentu interesują nas jedynie litery z tego właśnie, zawężonego zakresu!

Opóźnienia

Zwróć uwagę na opóźnienia występujące przy reagowaniu na sygnał dźwiękowy:

```
0.135 VISION SET-BUFFER-CHUNK VISUAL TEXT0
0.185 PROCEDURAL PRODUCTION-FIRED ENCODE-ROW-AND-FIND
0.185 VISION SET-BUFFER-CHUNK VISUAL-LOCATION VISUAL-LOCATION3-0
0.200 AUDIO SET-BUFFER-CHUNK AURAL-LOCATION AUDIO-EVENT0 REQUESTED NIL
0.235 PROCEDURAL PRODUCTION-FIRED ATTEND-HIGH
0.285 PROCEDURAL PRODUCTION-FIRED DETECTED-SOUND
...
0.570 AUDIO SET-BUFFER-CHUNK AURAL TONE0
0.605 PROCEDURAL PRODUCTION-FIRED ATTEND-HIGH
0.655 PROCEDURAL PRODUCTION-FIRED SOUND-RESPOND-MEDIUM
0.690 VISION SET-BUFFER-CHUNK VISUAL TEXT3
0.740 PROCEDURAL PRODUCTION-FIRED ENCODE-ROW-AND-FIND
```

Pomimo tego, że dźwięk zabrzmiał w chwili 0.150, pierwszy wpływ na zachowanie modelu (pomijając samą obsługę zdarzenia w chwili 0.570) można zauważyć dopiero w chwili 0.690!

Krok 3: Zapamiętuj litery tylko z odpowiedniego rzędu

Zauważ, iż po zmienieniu wartości *upper-y* oraz *lower-y*, reguła *encode-row-and-find* będzie poszukiwała liter jedynie w interesującym nas wierszu (zapytanie +visual-location>):

```
(P encode-row-and-find
=goal>
   ISA read-letters
   location =pos
   upper-y =uy
   lower-y =ly
=visual>
==>
=visual>
   status =pos
-visual>
=goal>
   location nil
   state attending
+visual-location>
   :attended nil
   > screen-y =uy
   < screen-y =ly
)
```

Krok 4: Wypisz litery z odpowiedniego rzędu

Aby rozpocząć wypisywanie liter z odpowiedniego rzędu, skorzystamy z poniższej reguły:

```
(P start-report
=goal>
   ISA read-letters
   tone =tone
?visual>
   state free
==>
+goal>
   ISA report-row
   row =tone
+retrieval>
   status =tone
)
```

Kiedy odpali się powyższa reguła?

Użyteczność reguł

Z regułą *start-report* jest pewien problem... może ona przedwcześnie zacząć wypisywać litery!

Aby temu przeciwdziałać, zdefiniujemy dla naszych reguł **użyteczności**.

W przypadku wielu pasujących reguł, wybrana zostanie ta o wyższej wartości użyteczności. Domyślnie, wszystkie reguły mają użyteczność równą 0. Regule *start-report* ustawimy użyteczność -2.

```
(spp start-report :u -2)
```

Ustawiając niską wartość użyteczności, upewniamy się, iż reguła ta będzie użyta tylko wtedy, gdy nie ma nic innego do zrobienia.

Użyteczność reguł

Zauważ, że regułom do reagowania na dźwięk ustawiliśmy wysoką użyteczność. Ma to na celu upewnienie się, iż odpalą się one tak szybko jak to możliwe.

```
(spp detected-sound :u 10)
(spp sound-respond-low :u 10)
(spp sound-respond-medium :u 10)
(spp sound-respond-high :u 10)
```

Krok 4: Wypisz litery z odpowiedniego rzędu

Aby wypisać litery, użyjemy reguły:

```
(P do-report
=goal>
   ISA report-row
   row =tone
=retrieval>
   status =tone
   value =val
?manual>
   state free
==>
+manual>
   cmd press-key
   key =val
+retrieval>
   status =tone
   :recently-retrieved nil
)
```

Kolejna uwaga na temat finstów...

W regule *do-report* korzystamy z parametru *:recently-retrieved* który jest odpowiednikiem *:attended* w module wizualnym. Podobnie jak tam, posiadamy ograniczoną liczbę finstów (domyślnie 4) i ograniczony czas ich utrzymania.

Liczba deklaratywnych finstsów nie będzie tu problemem (cztery wystarczą) ale mogą one zostać zbyt szybko zapomniane, a więc zwiększamy czas ich trwania do 10 sekund (domyślnie trzy sekundy).

```
(sgp :v t :declarative-finst-span 10)
```

Krok 5: Wciśnij spację aby zakończyć eksperyment

Aby zakończyć eksperyment, użyjemy reguły:

```
(P stop-report
=goal>
   ISA report-row
   row =row
?retrieval>
   buffer failure
?manual>
   state free
==>
+manual>
   cmd press-key
   key space
-goal>
)
```

Dopasowywanie do danych

Zadanie 3: Dopasowanie do danych

Porównaj dopasowanie naszego modelu do rzeczywistych wyników z eksperymentów Sperlinga. W tym celu, uruchom polecenie:
(run-sperling 100)

Aby sprawić, iż każdy przebieg będzie inny, musimy zakomentować w pliku eksperymentu (poprzez dodanie ';' na początku linii) linię zawierającą stałe ziarno losowości:
;(sgp :seed (100 0))

Ponadto, aby przyśpieszyć symulację stu przebiegów eksperymentu, ustaw parametr *:v* na wartość *nil*
(sgp :v nil :declarative-finst-span 10)
oraz wyłącz wypisywanie odpowiedzi na ekran:
*(setf *show-responses* nil)*

Dopasowanie do danych

Zadanie 3: oczekiwane wyniki

CORRELATION: 0.997
MEAN DEVIATION: 0.115

Condition	Current Participant	Original Experiment
0.00 sec.	3.207	3.03
0.15 sec.	2.43	2.40
0.30 sec.	2.17	2.03
1.00 sec.	1.56	1.50

Wysoka korelacja czasów rzeczywistych i modelowanych, wydaje się potwierdzać trafność naszego modelu.

Dopasowywanie do danych

Zadanie 4: Liczenie obiektów

Plik z szkieletem modelu:
...\ACT-R Standalone Environment\tutorial\unit3\subitize.lisp

a) Uruchom eksperyment *subitize* w trybie interakcji z użytkownikiem, aby się z nim zapoznać:
(subitize 'human)

b) Stwórz model rozwiązujący problem liczenia obiektów (X-ów) na ekranie. Możesz podnieść liczbę dostępnych finstów aby uprościć zadanie.

c) Stwórz model rozwiązujący problem liczenia obiektów (X-ów) na ekranie. Tym razem nie zmieniaj domyślnych parametrów finstów (4 finsty, 3 sekundy).

Za każdym razem porównaj uzyskane przez model czasy z czasami uzyskiwanymi przez ludzi.