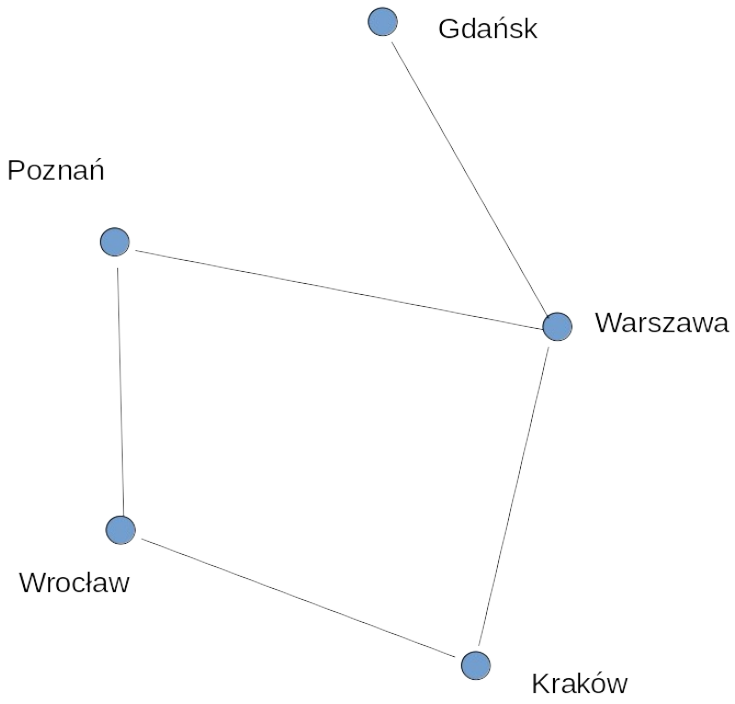
Dorban - dywagacje

Paweł Biernacki
Vantaa 18.01.2023

Mój program Svarog i jego demko Dorban.
https://github.com/pawelbiernacki/svarog
https://sourceforge.net/projects/dorban/

Jeden gracz, dyskretny czas, zadana funkcja wypłaty, stochastyczne otoczenie. Mam algorytm, który dla planowania do n-kroków naprzód znajduje optymalną akcję dla sytuacji określonej przez percepcję gracza oraz jego wiarę (przekonanie) co do wartości zmiennych ukrytych. Formalny opis algorytmu jest w dokumencie https://www.perkun.org/resources/pdf/triglav_0_0_0.pdf.

Jest sobie wampir, gracz (Dorban) oraz NPC (Pregor). Każdy z nich jest w jednym z pięciu miast. Tworzą one graf połączony krawędziami:



Dorban widzi następujące informacje:

- czy Dorban wygrał ostatnio walkę
- miasto, w którym jest Dorban
- czy Dorban widzi Pregora
- czy Dorban widzi, że Pregor jest żywy
- czy Dorban widzi wampira
- czy Pregor podąża za Dorbanem

Zmienne te (nazywane zmiennymi wejściowymi) mogą mieć następujące wartości:

Nazwa zmiennej	Możliwe wartości
czy Dorban wygrał ostatnio walkę	false, true, none
miasto, w którym jest Dorban	Warszawa, Kraków, Wrocław, Poznań, Gdańsk
czy Dorban widzi Pregora	false, true
czy Dorban widzi, że Pregor jest żywy	false, true, none
czy Dorban widzi wampira	false, true
czy Pregor podąża za Dorbanem	false, true

Oprócz tego, co Dorban widzi są również pewne zmienne “ukryte”, na temat których ma swoje zdanie (w każdym kroku uaktualniane zgodnie z moim algorytmem). Są to następujące zmienne ukryte:

- miasto, w którym jest wampir
- miasto, w którym jest Pregor
- czy Pregor żyje

Zmienne te mogą mieć następujące wartości:

Nazwa zmiennej	Możliwe wartości
miasto, w którym jest wampir	Warszawa, Kraków, Wrocław, Poznań, Gdańsk
miasto, w którym jest Pregor	Warszawa, Kraków, Wrocław, Poznań, Gdańsk
czy Pregor żyje	false, true

Dorban widzi wampira wtedy i tylko wtedy, gdy wampir i Dorban są w tym samym mieście.
Dorban widzi Pregora wtedy i tylko wtedy, gdy Pregor i Dorban są w tym samym mieście.
Jeśli Dorban nie widzi Pregora, wówczas zmienna “czy Dorban widzi, że Pregor jest żywy” ma wartość “none”. Jeżeli go widzi, wówczas zmienna ta musi mieć wartość “true” lub “false”.

Dla każdego stanu percepcji (czyli sygnału na wejściu) jest sobie pewien zbiór możliwych stanów gry. Każdy stan gry jest trójką postaci:

```
{miasto_w_którym_jest_wampir=>...,
miasto_w_którym_jest_Pregor=>..., czy_Pregor_żyje=>...}
```

Na przykład:

```
{miasto_w_którym_jest_wampir=>Warszawa,
miasto_w_którym_jest_Pregor=>Kraków, czy_Pregor_żyje=>true}
```

Ilość tych stanów jest równa $5^2*2=50$. Rozkład prawdopodobieństwa nad zbiorem tych stanów jest “wiarą” (przekonaniem) Dorbana.

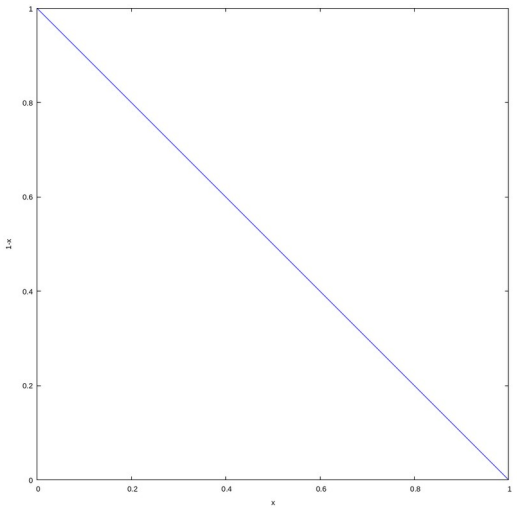
Mimo, że wszystkich stanów jest 50, to stan percepcji (stan wejściowy) zawęża faktyczną ilość możliwych stanów do liczby mniejszej lub równej 50. Np. jeśli Dorban jest w Warszawie, widzi wampira i nie widzi Pregora, ale wie, że Pregor żyje, wówczas rozkład ten mógłby mieć postać:

```
{miasto_w_którym_jest_wampir=>Warszawa,
miasto_w_którym_jest_Pregor=>Kraków, czy_Pregor_żyje=>true}    10%
{miasto_w_którym_jest_wampir=>Warszawa,
miasto_w_którym_jest_Pregor=>Wrocław, czy_Pregor_żyje=>true}  20%
{miasto_w_którym_jest_wampir=>Warszawa,
miasto_w_którym_jest_Pregor=>Poznań, czy_Pregor_żyje=>true}   30%
{miasto_w_którym_jest_wampir=>Warszawa,
miasto_w_którym_jest_Pregor=>Gdańsk, czy_Pregor_żyje=>true}   40%
```

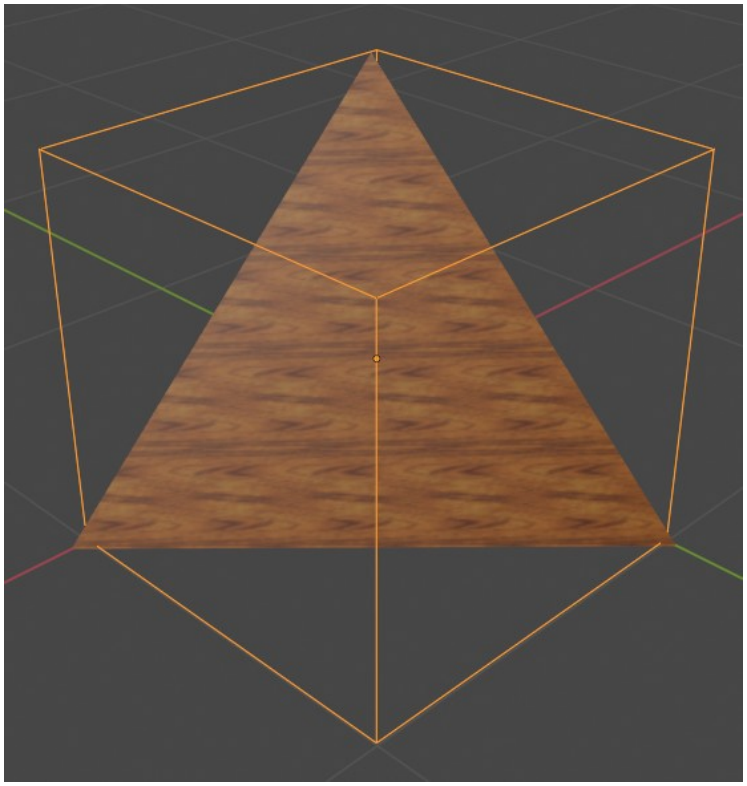
Wówczas Dorban wie na pewno o mieście, w którym jest Pregor, że nie jest to Warszawa, gdyż Dorban nie widzi Pregora, a sam wie zawsze gdzie jest.

Okazuje się, że optymalna decyzja do maksymalizacji funkcji wypłaty w/g mojego algorytmu zależy zarówno od stanu percepcji gracza jak i od jego wiary. Ilość możliwych stanów percepcji wynosi $3*5*2*3*2*2=360$. Dla każdego stanu percepcji mamy w realnej grze pewien rozkład prawdopodobieństwa nad zbiorem możliwych stanów. Zbiór tych rozkładów prawdopodobieństwa jest zbiorem ciągłym, w związku z tym nie możemy obliczyć po prostu optymalnych decyzji dla wszystkich możliwości. Możemy go jednak zdyskretyzować.

Dla n stanów rozkład odpowiada tablicy wartości $[x_1, x_2, \ldots, x_n]$ przy czym $x_i \in [0,1]$ oraz $\sum_{x_i} x_i = 1$. Jeżeli n=1 wówczas jest to trywialnie tablica $[1]$. Dla n=2 jest to zbiór będący wykresem następującej funkcji:



Dla n=3:



Każdy ze stanów gry odpowiadający naszemu stanowi percepcji dodaje do naszej kostki jeden wymiar. Wprowadźmy parametr który nazwiemy "granulacją": $\gamma \in N$. Niech będzie on równy 2. Dla granulacji 2 generujemy wszystkie możliwe n-ki o wartościach ze zbioru $\{0,1\}$.

$(0,1)$
$(1,0)$
$(1,1)$

Następnie normalizujemy te n-ki, które zawierają przynajmniej jedną niezerową wartość. N-ka o wartościach (0,0) nie da się znormalizować (bo suma jest zero). Ale pozostałe można znormalizować.
Otrzymujemy:
$(0,1)$
$(1,0)$
$(0.5,0.5)$
Pomysł polega na tym, by obliczyć z algorytmu optymalne akcje jedynie dla tych trzech przypadków, a następnie już w trakcie gry dla dowolnej wiary postaci (x_1, x_2) sprawdzać który z obliczonych trzech punktów jest jej najbliższy, w sensie odległości euklidesowej. Po zidentyfikowaniu najbliższego punktu decydujemy wykonać optymalną akcję dla niego (zamiast liczyć z algorytmu jaka jest prawdziwa optymalna akcja).

Na przykład dla wiary $(0.3,0.7)$ odległość euklidesowa pomiędzy nią a punktem $(0,1)$ wynosi $\sqrt{(0.3-0)^2+(0.7-1)^2}$ czyli 0.4242640687119286.
Odległość euklidesowa pomiędzy wiarą $(0.3,0.7)$ a punktem $(1,0)$ wynosi $\sqrt{(0.3-1)^2+(0.7-0)^2}$ czyli 0.9899494936611665.
Odległość euklidesowa pomiędzy wiarą $(0.3,0.7)$ a punktem $(0.5,0.5)$ wynosi $\sqrt{(0.3-0.5)^2+(0.7-0.5)^2}$ czyli 0.282842712474619.

Najbliższy wierze $(0.3,0.7)$ jest więc punkt $(0.5,0.5)$, ponieważ odległość od danej wiary do tego punktu jest najmniejsza. W takiej sytuacji wykonamy więc akcję optymalną dla $(0.5,0.5)$.

Dla granulacji γ tworzymy n-ki o wartościach ze zbioru $\{0,1,2,...,\gamma-1\}$.

Pewnym ulepszeniem tej metody jest następujący algorytm:

Dla trzech punktów otrzymanych w procesie dyskretyzacji przestrzeni wiar (w danym przykładzie dla n=2 i granulacji $\gamma=2$) liczymy z algorytmu planowania nie tylko optymalną akcję, ale i wartość oczekiwaną funkcji odpłaty.

Znormalizowana n-ka	Optymalna akcja	Wartość oczekiwana funkcji wypłaty
$(0,1)$	A	10.0
$(1,0)$	B	5.0
$(0.5,0.5)$	B	2.0

Dla każdej akcji liczymy sumę ważoną wartości oczekiwanej funkcji wypłaty z wagami zależnymi np. odwrotnie proporcjonalnie od odległości aktualnej wiary (załóżmy jak poprzednio $(0.3,0.7)$).

Dla akcji A suma ta będzie równa $10.0/0.4242640687119286=23.57022603955158$

Dla akcji B suma ta będzie równa
$5.0/0.9899494936611665+2.0/0.282842712474619=12.12183053462653$

Widać, że w danym przypadku ważona wartość oczekiwana funkcji wypłaty jest większa dla decyzji A niż dla decyzji B, chociaż ta ostatnia obejmuje dwa przypadki. Innymi słowy gracz będzie stawiał raczej na $(0,1)$.

Odnośnie zależności wagi od odległości – nie musi to być funkcja $1/x$. Możemy w mianowniku dodać jakąś niewielką stałą, tak, by uniknąć dzielenia przez zero w przypadku niewielkich odległości.

Dyskretyzacja w przestrzeni trójwymiarowej jest nieco bardziej spektakularna.

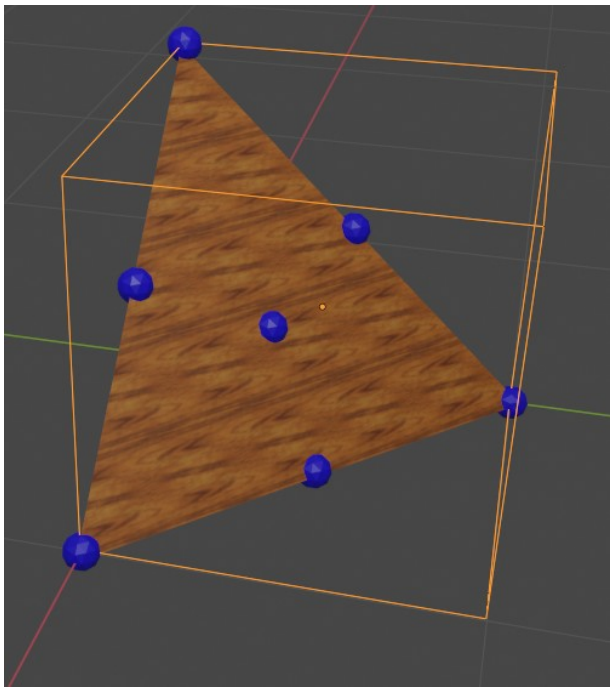
Wygenerujmy 2^3-1 a zatem 7 możliwych trójek do normalizacji:

$(0,0,1)$
$(0,1,0)$
$(0,1,1)$
$(1,0,0)$
$(1,0,1)$
$(1,1,0)$
$(1,1,1)$

Po normalizacji otrzymamy z nich:

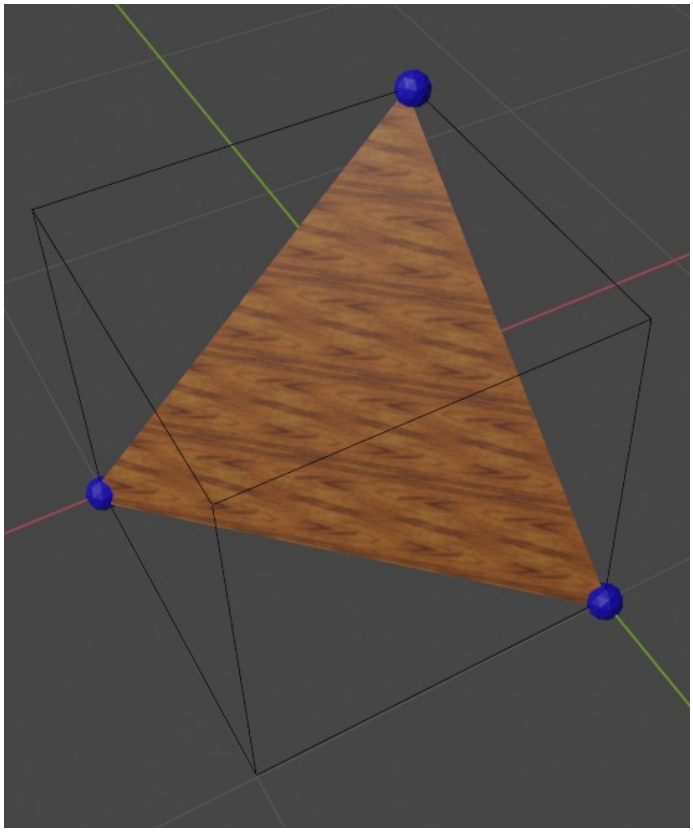
$(0,0,1)$
$(0,1,0)$
$(0,0.5,0.5)$
$(1,0,0)$
$(0.5,0,0.5)$
$(0.5,0.5,0)$
$(0.3333,0.3333,0.3333)$

Oto wizualizacja 3D powyższego zbioru punktów:



Problem z dyskretyzacją pojawia się w przypadku, gdy mamy wiele wymiarów dyskretyzowanej przestrzeni. Z warunków gry w programie Dorban wynika, że ilość stanów gry odpowiadających danemu stanowi percepcji waha się od 1 do 32 (a więc nigdy nie jest to 50). Nawet dla granulacji $\gamma=2$ ilość punktów do normalizacji i obliczenia optymalnej akcji wyniesie $2^{32}-1=4294967295$, a więc ponad cztery miliardy. To bardzo dużo. Gdyby obliczenie optymalnej akcji trwało tylko jedną sekundę skończylibyśmy obliczenia po 138 latach. Dla dużych ilości wymiarów musimy postąpić inaczej.
Wygenerujmy dla n możliwych stanów wszystkie wektory bazowe z bazy kanonicznej. Jest ich n, a więc dużo mniej niż 2^n-1 . Obliczmy zarówno optymalną akcję jak i wartość oczekiwaną funkcji wypłaty dla tych n przypadków.

W przypadku n=3 zysk nie będzie bardzo spektakularny – zamiast 7 punktów mamy 3:

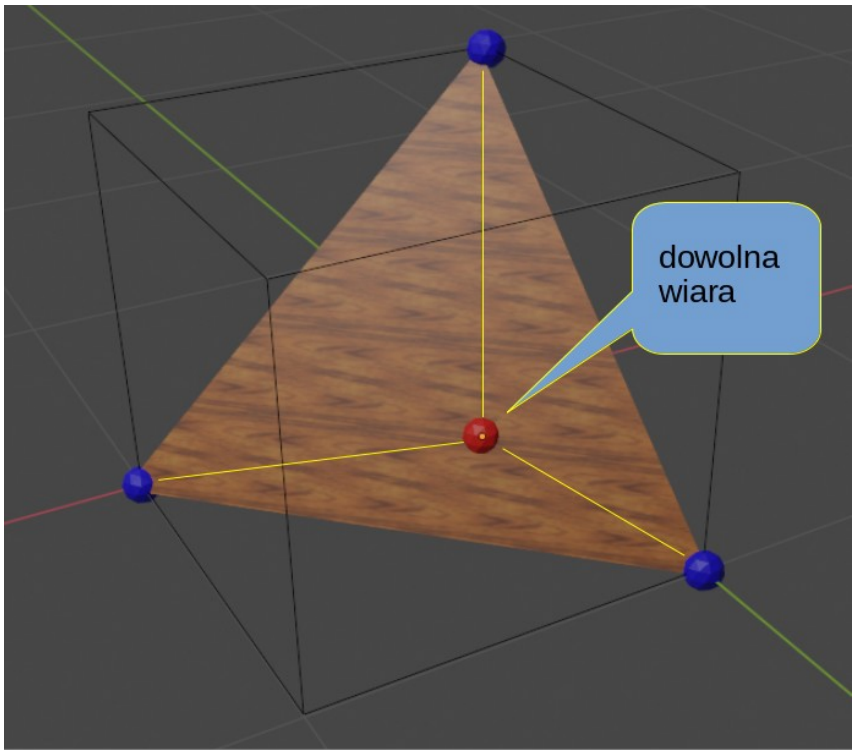


Jednak już w przypadku n=10 mamy 10 punktów zamiast 1023, a więc ponad 100 razy mniej!

Dla wszystkich tych punktów odpowiadających wektorom bazy kanonicznej w naszej n-wymiarowej przestrzeni prawdopodobieństw obliczamy:

- optymalną akcję
- wartość oczekiwaną funkcji wypłaty

Następnie możemy postępować zgodnie z opisanym już algorytmem:



- dla każdej aktualnej wiary w trakcie gry liczymy odległości do wiar z bazy kanonicznej
- zależnie od tych odległości wyznaczamy wagi (np. jako $1/(0.001+\delta)$, gdzie δ jest odległością)
- dla każdej optymalnej akcji (z tych prekalkulowanych) obliczamy sumę ważoną wartości oczekiwanej
- wybieramy ostatecznie tą akcję, dla której obliczona suma ważona jest maksymalna

Według tego algorytmu działa Dorban – demko Svaroga napisane w C++. Po rozpakowaniu tarballa w katalogu svarog znajdziemy plik dorban_with_precalculated_knowledge.svarog. Plik ten zawiera nie tylko definicje napisane ręcznie ale i wiedzę prekalkulowaną. Ma on wielkość 2286468 bajtów. W linii 694 tego pliku jest deklaracja:

```
precalculated (4,2)
```

Pierwsza liczba oznacza głębokość planowania (do czterech kroków naprzód), a druga granulację.

Następnie dla każdego ze stanów percepcji mamy deklarację:

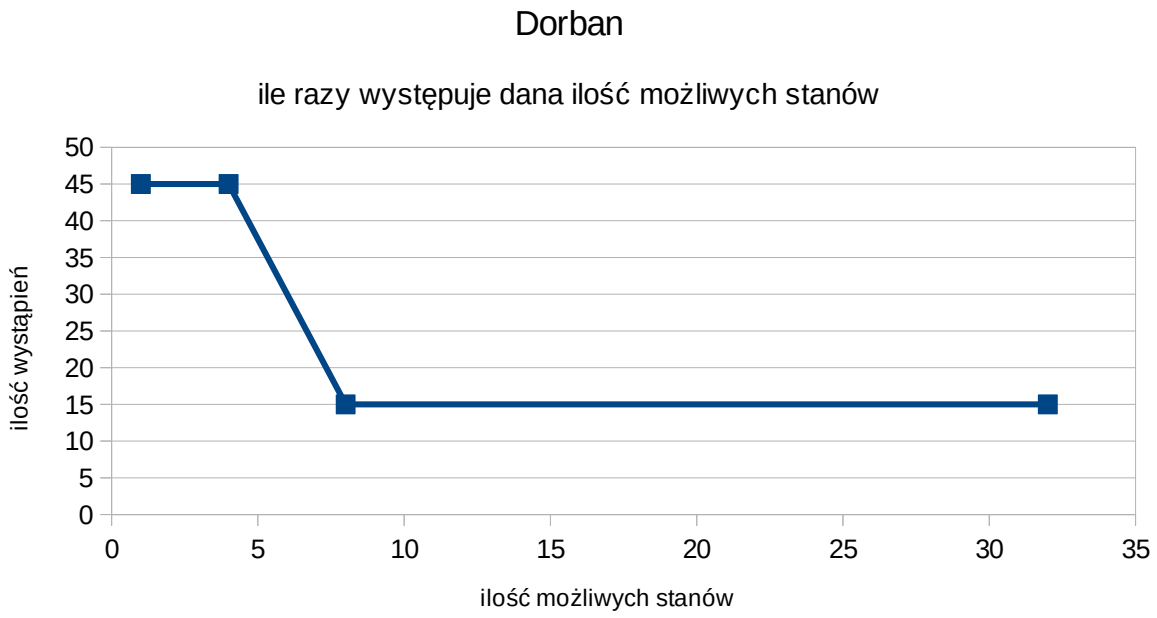
```
on visible state
```

Łatwo sprawdzić, że w całym pliku deklaracja ta występuje 120 razy. Oznacza to, że mamy prekalkulowane optymalne akcje dla 120 różnych stanów percepcji. Dlaczego nie dla wszystkich 360? Otóż pewne stany percepcji są niemożliwe, co wynika z charakterystyki gry.

W każdej deklaracji on_visible_state mamy następujące informacje:

```
amount_of_possible_states = ...;
max_amount_of_beliefs = ...;
```

Okazuje się, że ilość możliwych stanów dla tej konkretnej gry może przybierać jedynie wartości 1,4,8 lub 32. Dla jedynki oraz czwórki jest to 45 wystąpień, dla ósemki oraz trzydziestki dwójki 15 wystąpień:



`max_amount_of_beliefs` jest równa $2^{\alpha}-1$
gdzie α - `amount_of_possible_states`

Jeżeli max_amount_of_beliefs przekracza pewną ustaloną wartość (1000), wówczas w następnej linii występuje deklaracja "`too complex;`" i dalej następuje pewna ilość (w naszym wypadku 32) deklaracji `on belief`. Odpowiadają one "wiarom" dla wektorów bazowych przestrzeni wiar, dlatego jest ich tylko tyle, co wymiarów, czyli w naszym wypadku 32.

Łatwo sprawdzić, że deklaracja `too complex;` występuje w omawianym pliku 15 razy, to jest dokładnie wtedy, gdy ilość wektorów bazowych równa jest 32.

Co się dzieje, jeżeli ilość wymiarów jest mniejsza niż 32, czyli gdy przybiera wartości 1, 4 lub 8? Wówczas dokonujemy pełnej dyskretyzacji przestrzeni wiar, a więc liczymy optymalną akcję odpowiednio dla 1, 15 lub 255 przypadków.

Optymalne akcje są podawane w klauzuli `action` pod deklaracją `on belief`. Łatwo sprawdzić, że jest ich wszystkich 5025. Po symbolu ":" następuje wartość oczekiwana funkcji wypłaty, np.

```
action {optimal_action=>attack_vampire } :    198.967;
```

Zaletą takiej reprezentacji jest szybkość. Dorban podejmuje decyzje w czasie mierzonym w setkach milisekund na desktopowej maszynie (12 x AMD Ryzen 5 4600G with Radeon Graphics). Wady są przynajmniej trzy:

1. spory rozmiar pliku zawierającego wiedzę prekalkulowaną – 2286468 bajtów
2. długi czas prekalkulacji (rzędu kilku dni)
3. Dorban nie potrafi w żaden sposób uzasadnić swojej decyzji

Dwie pierwsze wady wydają się rozsądnym kosztem w naszej sytuacji, zwłaszcza, że rozmiar pliku nie rośnie wraz z rosnącą głębokością planowania. Ponadto prekalkulacja musi być dokonana tylko raz.
Trzecia wada jest poważniejsza. Dorban nie dysponuje żadnym wyjaśnieniem dlaczego coś należy zrobić. Ma tylko optymalną akcję oraz wartość oczekiwaną funkcji wypłaty. Wydaje się, że Svaroga (tj. program będący sercem Dorbana) można jeszcze ulepszyć, tak by nasz bohater mógł "rozumować", tj. obliczać sobie optymalną akcję operując na pewnym poziomie abstrakcji.