Tworzenie skryptu smoka jest stosunkowo proste. Musimy jednak pamiętać o pewnych wymaganiach co do pliku tekstowego w którym znajduje się kod źródłowy. Pierwsze dwie linijki muszą wyglądać następująco:

TreeBrain
Name: AttackDragon

Gdzie nazwa smoka jest to ciąg znaków zawierający litery łacińskiego alfabetu. Bez spacji ani innych separatorów. Później już następuję kod źródłowy smoków. Przypuśćmy że chcielibyśmy napisać smoka, który zauważając przeszkodę przed swoją głową skręci w którąś ze stron, tutaj np. w Lewo. Konstrukcja takiego smoka wyglądałaby następująco:

```
TreeBrain
Name: SimpleLeftTurner
if (getTile(Point2Di(0, 1)) == Empty) {
   # Jeżeli jest puste to smok idzie prosto,
   return Forward;
} else {
   # A jeżeli coś tam jest to skręci;
   return TurnLeft;
}
```

Wiem, niby taki prosty smok, a jednak niektórzy mogą pomyśleć że aby cokolwiek zrobić trzeba napisać tony skryptu. Tak po prawdzie to chcąc napisać to w cpp zrobilibyście mniej więcej tyle samo. Więc tłumaczymy co się tutaj tak naprawdę dzieje:
Po podstawowym nagłówku następuję konstrukcja **if**. Jest to instrukcja warunkowa, i posiada pewne reguły do których musimy się stosować. Po słowie kluczowym **if** może ale nie musi nastąpić nawias (a dlaczego to się za jakiś czas dowiemy), i później następuję właściwy warunek.
getTile pobiera pole Point2Di(0, 1), oraz przyrównuje tą wartość do Empty. Jeżeli pole to jest równe Empty to zostanie wywołany return Forward; jeżeli nie to return TurnLeft. Możliwości elementów do których możemy przyrównać jest dosyć dużo, są to:

Empty
Wall
DragonBody
DragonHead
DragonNeck
DragonTail

Oraz dla możliwości porównań mamy do dyspozycji nie tylko znak równości (podwójnej równości znany niektórym z C++), ale także znak „!=”, który równość zaprzecza. Ten sam przykład smoka można tym samym zapisać w następujący sposób:

```
TreeBrain
Name: SimpleLeftTurner
if (getTile(Point2Di(0, 1)) != Empty) {
   # Jeżeli przed nami nie ma pustego pola to skręcamy,
   return TurnLeft;

} else {
   # a jeżeli jest to idziemy prosto;
   return Forward;
}
```

Węzeł **if** oraz **getTile** nie powinny być dla was problemem. Możecie się mnie pytać czemu trzeba pisać Point2Di. Odpowiedzi na to nie posiadam, da się zrezygnować z tego słowa kluczowego, lecz w aktualnej chwili nie jest to dla mnie priorytet ☺. Wartości zawarte w nawiasach po tym słowie definiują współrzędne względne, względem głowy smoka.

W ten sposób:
Point2Di(0, 1) jest to pole bezpośrednio przed głową smoka.
Point2Di(0, -1) jest to pole za głową smoka (po paru pierwszych kolejkach powinna być to zawsze szyja)
Point2Di(1, 0) – pole na prawo od smoczej głowy,
Point2Di(1, 2) – pole dwa do góry i jeden w prawo.

W ten sposób można pobrać komendą **getTile** wartość dowolnego pola w dowolnie odległym otoczeniu głowy smoka, tudzież całej areny. Jeżeli będziecie chcieli pobrać pole które znajduje się poza areną np. Point2Di(300, 5400), to otrzymacie *Wall*.

Dobrym elementem języka jest możliwość umieszczania komentarzy. Po znaku #, zawartość linii jest ignorowana. W następnej linii już powinny znajdować się komendy, chociaż i tym nie musicie się przejmować, tzw. **białe** znaki są ignorowane.

Ale zauważyłem jeszcze jeden problem (który być może już któregoś z was zadziwił). W instrukcji **if** musi znaleźć się coś za słowem kluczowym **else**, oraz samo to słowo musi wystąpić. Przykra sprawa, ale nawet aktualnie nie mam pomysłu jak to obejść dla aktualnej struktury języka (chociaż już mi coś tam świta). Konstrukcje w stylu:

```
if (getTile(Point2Di(0, 1)) == Wall) {
   return Forward;
}
```

czy:

```
if (getTile(Point2Di(0, 1)) != Wall) {
   return TurnLeft;
} else;
```

Są nieprawidłowe. Pierwsza ponieważ musi wystąpić słowo **else** a druga ponieważ za słowem tym musi być co najmniej pusty blok **{ }.** Jeżeli dla kogoś będzie bardziej czytelne to naszego skonstruowanego smoka można zapisać w postaci:

```
TreeBrain
Name: SimpleLeftTurner

{

if (getTile(Point2Di(0, 1)) != Empty) {
   # Jeżeli przed nami nie ma pustego pola to skręcamy,
   return TurnLeft;
} else { }

if (getTile(Point2Di(0, 1) == Empty) {
   # a jeżeli jest to idziemy prosto;
   return Forward;
} else { }

}
```

Wszystko byłoby ok, gdyby nie to że niektórzy z was zauważą: “Co robię te przeklęte nawiasy klamrowe na początku i na końcu algorytmu.” Jest to oczywiście instrukcja blokowa, problem jest w tym że węzeł główny algorytmu jest jeden. Więc jeżeli chcecie stworzyć dwa następujące po sobie ify. (Albo jakąkolwiek konstrukcję wielokrotną na poziomie korzenia algorytmu) to trzeba opakować wszystko w blok. Niby nic... a niektórzy z was będą zdenerwowani jak wasz algorytm wykrzacza arenę. ☺

Już nie będę męczył dalej tego przykładu, tylko skonstruuję na potrzeby waszej edukacji kolejny (i będę pisał bardzo rozwlekle aby jak najwięcej mechanizmów języka umieścić w przykładzie). Tamten smok był całkiem dobry, ale chciałbym dodać do niego nowy element. Jeżeli pole przed nami nie jest puste, a jest tak część mojego (aktulanie programowanego) smoka to chciałbym skręcić w prawo (a nie w lewo jak to było dotychczas). Napiszę wtedy następujący fragment:

```
TreeBrain
Name: SimpleTurner

if (getTile(Point2Di(0, 1)) != Empty) {
   # Jeżeli przed nami nie ma pustego pola to,
   if (isSelf(Point2Di(0, 1)) {
      # jeżeli jest tam segment mojego ciała,
      return TurnRight;
   } else {
      # inny smok, lub ściana
      return TurnLeft;
   }
} else {
   # a jeżeli jest to idziemy prosto;
   return Forward;
}
```

Nowy zaprezentowany tutaj węzeł to **isSelf**. Jest to test czy na danym polu jest segment ciała tego smoka, czy nie. Prosty test, który chyba każdy z nas jest w stanie wykonać (trudno w końcu pomylić naszą rękę z inną :D).

Nasz smok i tak ciągle jest głupi. Jeżeli wejdzie w róg areny, to nieumyślnie może skręcić w złą stronę. Chcielibyśmy sprawdzać czy skręt jest możliwy. Zapiszę algorytm, choć będzie on długi, to mam nadzieję że czytelny:

```
{

if (getTile(Point2Di(0, 1)) == Empty) {
   return Forward;
} else { }

if (getTile(Point2Di(0, 1)) != Empty & getTile(Point2Di(-1, 0)) == Empty) {
   return TurnLeft;
} else { }

if (getTile(Point2Di(0, 1)) != Empty & getTile(Point2Di(1, 0)) == Empty) {
   return TurnRight;
} else { }

}
```

Przykład napisany rozwlekle, przyznaję. Prezentuje on operator logiczny w węźle **if**. Aktualnie są jego dwa warianty **&** równoznaczny z logicznym AND oraz | czyli logiczne OR.

Warunek drugi będzie prawdziwy jeżeli pole przed nie jest puste, oraz pole na lewo od nas jest puste. Oraz przy użyciu tych operatorów możliwe jest zagnieżdżanie warunków (korzystając przy okazji z nawiasów okrągłych). Konstrukcja,

```
if (getTile(Point2Di(0, 1)) == Empty & ( getTile(Point2Di(1, 0)) == Empty |
getTile(Point2Di(-1, 0)) == Empty)) {
   #jeżeli prawidłowy to…
} else {

}
```

Jest prawidłowa, oraz jeżeli warunek jest prawdziwy to pole na wprost głowy smoka jest puste oraz co najmniej jedno z pól na lewo bądź na prawo jest puste.

Informacje zawarte w tym dokumencie są wystarczające aby rozpocząć z powodzeniem pisanie smoków... tym samym...

POWODZENIA