Lake Overview

# Koncepcja działania

W **Lake** koncepcja działania oparta jest o kilka prostych zasad:

1. Istnieje instancja Clisp’a dla świata – tylko jedna
2. Istnieje wiele instancji Clisp’a dla aktorów – po jednej instancji dla każdej instancji aktora
3. Istnieją ustalone formaty wejść/wyjść świata oraz aktorów
   1. W przypadku aktorów formaty wyjść/wejść mogą zostać rozszerzone
   2. Zarówno wyjściowym i wejściowym odpowiadają odpowiednie klasy w Java, których format musi odpowiadać obiektom w Clisp
      1. Klasy opisujące wejścia/wyjścia muszą implementować metodą pozwalające na odczyt/zapis zmiennych z faktu
4. Program działa, jako gra turowa
   1. Każdy z aktorów implementuje odpowiedni interfejs i posiada referencję do środowiska clispa. Dzięki temu każdy z aktorów może działać we własny indywidualny sposób.
   2. Ważne jest, aby po zakończeniu logiki aktora, zacomittować wszystkie zmiany do clispa
      1. Zmiana położenia
         1. Zacomittowania aktora
         2. Zacomiitowanie pola
      2. Zmiany we polach aktora
5. Na początku działania programu odbywa się proces **bootstrap**, który polega na załadowaniu danych o świecie, wybraniu w sposób losowy aktorów tak, aby wybrany został, co najmniej 1 aktor danego typu
   1. Informacje o danym, jakie typy są ładowane umieszczone zostały w pliku **lake.properties**
   2. Struktura pliku wygląda następująco
      1. ***lake.types*** *=* lista a prefix mapujących na konfiguracje konkretnych aktorów
      2. lake.types.{prefix}.clazz – nazwa pakietowa klasy implementującej aktora w klasie Java
      3. lake.types.{prefix}.clisp – ścieżka do pliku Clisp danego aktora
      4. lake.types.{prefix}.initData – ścieżka do pliku properties z danymi wejściowymi danego aktora

# Plansza

Plansza w aplikacji składa się z pól, które opisane są w sposób umożliwiający ich późniejsze wykorzystanie przez świat oraz aktorów.

# Aktorzy

Aktorzy w aplikacji zostali zdefiniowani zgodnie z wymaganiami świata, tj. posiadają oni odpowiednie pola oraz metody pozwalające im na komunikację z aplikacją. Niemniej nie zostali wyposażeni w funkcjonalność niezbędną do wymiany danych.

## Wiedza aktorów

Aktorzy w k-tej chwili posiadają wiedzę o sobie oraz o otaczającym ich świecie. Innymi słowy struktura danych aktora wygląda w sposób następujący:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Bird** | **Fish** | **FishP** | **Poacher** | **Forester** | **Angler** |
| meta | Id | X | X | X | X | X | X |
| type | X | X | X | X | X | X |
| location | atField | X | X | X | X | X | X |
| toField | X | X | X | X | X | X |
| generic-abilities | catAttack | X |  | X | X | X | X |
| canFly | X |  |  |  |  |  |
| canSwim |  | X | X |  |  |  |
| generic-properties | caught |  | X | X | X | X | X |
| hp | X | X | X | X | X | X |
| visionRange | X | X | X | X | X | X |
| attackRange | X |  | X | X | X | X |
| attackPower | X |  | X | X | X | X |
| moveRange | X | X | X | X | X |  |
| attack-properties | targetId | X |  | X | X | X | X |
| targetDown | X |  | X | X | X | X |
| corruption-properties | cash |  |  |  | X |  | X |
| corruptionThreshold |  |  |  |  | X |  |
| validId |  |  |  | X | X | X |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

W razie potrzeby można zdefiniować format danych dla danego aktora oraz napisać własną logikę odpowiedzialną za odczyt i modyfikację tychże danych.

## Ładowanie aktorów

Aktorzy do poprawnego działania wymagają:

* Zdefiniowanej klasy w Java, można użyć jednej z abstrakcyjnych klas, które opisują danego aktora.
* Implementacji metod z interfejsu **ClispActor**, które pozwalają realizować funkcjonalność aktora.
* Pliku \*.properties z danymi wejściowymi dla danego aktora, które zostaną załadowane, jako wejściowa struktura w pierwszej iteracji

## Wymagania implementacyjne aktorów

1. Aktorzy muszą implementować logikę odpowiedzialną za odczyt/zapis informacji ze struktur wejściowych/wyjściowym. Jest to ważne z tego powodu, że następnie informacja o tych danych są serializowane do Stringa (java.lang.String) i przekazywane do instancji Clisp’a danego aktora
2. Aktorzy muszą implementować logikę pozwalająca na losowe przeprowadzenie następnej tury, tj. informacje zależne losowo (kierunek przemieszczenie, czy losowo wybrać kierunek, czy przyjąć łapówkę) muszą zostać ustalone na podstawie danych ze struktur wejściowych na poziomie logiki danego aktora.
3. Implementacja aktorów konkretnych osób należy umieścić w osobnych paczkach
4. W paczce, które zawiera implementacja aktorów powinien również znaleźć się plik \*.properties, który zawiera dane wejściowe do pierwszej tury gry. Format pliku musi odpowiadać strukturze danych wejściowych, aby program mógł go odczytać i przeprowadzić inicjalizację instancji aktora w prawidłowy sposób.

Powyższe wymagania można zrealizować implementując dla swojego aktora interfejs **org.kornicameister.sise.lake.types.ClispType**, który wymusza implementacją odpowiednich metod.

# Jak działa tura

Aplikacja trzyma załadowane deskryptory aktorów. Deskryptor jest w tym miejscu takim obiektem, który wiąże ze sobą środowiska Clisp oraz instancję aktora po stronie Javy.

Tura jest kontrolowana przez świat, który uruchamia aktorów.

Jeśli te informacje są niewystarczające to proszę odwołać się do klasy **org.kornicameister.sise.lake.clisp.ClispRunner**