Ročníkový projekt – zimný semester

Truc Lam, Bui January 13, 2017

Stolných hier je nespočetne veľa, no z našej skúsenosti vieme, že zdieľajú veľa spoločných vlastností. Konkrétne sme si všimli nasledovné.

1. Jednoduchosť pravidiel. Oproti videohrám sú veľmi jednoduché – jeden z dôvodov je ten, že pravidlá hry musí ovládať aspoň jeden z hráčov. A o pohyb figúrok a herných objektov sa takisto stará niektorý z hráčov. Vo videohrách sa o dodržiavanie pravidiel stará stroj – počítač, herná konzola, ... a tie túto "manuálnu" robotu vedia robiť lepšie a rýchlejšie.

2. Častokrát je stav hry jednoznačne určený tým, v akej "konfigurácii" sa nachádzajú herné objekty (tj. ako sú na sebe "naskladané"), a hráčom na ťahu.

   (a) Príkladom je napríklad šach. Môžeme sa na to pozerať takto: herné objekty sú jednotlivé šachové figúrky, a políčka šachovnice. Stav hry je určený tým, ktoré figúrky sú položené na ktorých políčkach (+ hráčom na ťahu). Teda ak máme dve partie, a v oboch sú objekty na sebe naskladané rovnako, tak ak je v oboch na ťahu ten istý hráč, tak sú to úplne identické partie.

   (b) Pri komplikovanejších hrách nám to už ale nemusí stačiť – napríklad sa môže stať, že množina možných ťahov hráča závisí od doterajšieho priebehu hry.

   (c) "Naskladanie" objektov zvykne mať stromovitú štruktúru.

3. Ťah hráča spočíva vo výbere nejakej množiny herných objektov. Prípadne vyberá číslo, iného hráča, ...

   (a) V šachu: vyberie figúrku, ktorou chce pohnúť, a vyberie políčko, na ktoré chce figúrku presunúť.

   (b) V *Magic the Gathering*: vyberie kartu, ktorú chce zahrať, a dodatočné ciele, módy, a iné parametre.

Na základe horeuvedeného od nášho jazyka očakávame minimálne toto.

1. Práca s hernými objektami.

   (a) Možnosť **vytvárať a rušiť herné objekty** s rôznymi parametrami. Napríklad na začiatku partie šachu chceme vytvoriť hernú dosku, políčka na nej (s parametrami riadok a stĺpec) a jednotlivé figúrky.

   (b) Možnosť **na seba objekty "ukladať"** – napríklad položiť pešiaka na `h6`. A tiež možnosť objekty **"odkladať"** – napríklad keď pohneme pešiakom z `h5` na `h6`, tak už nie je položený na políčku `h5`.

   (c) Možnosť **zistiť, ako sú na sebe objekty uložené** – teda napríklad pre dve objekty A, B vedieť zistiť, či A je "na" B.

   Ak by sme túto možnosť nemali, tak nám je úplne jedno, ako sú na sebe objekty uložené, lebo by sme sa podľa toho aj tak nevedeli rozhodovať (tj. či je ťah platný alebo nie, ...).

2. Všeobecné vymoženosti programovacieho jazyka.

   (a) Všimli sme si, že väčšinou hry nemajú pamäť – teda málokedy od nás požadujú, aby sme si zapamätali hodnotu $X, Y, \ldots$ (ktoré nejako vypočítame) na neskoršie využitie. (Na to slúži predsa konfigurácia herných objektov.)

   Pre jednoduchosť preto môže byť **funkcionálny** – žiadne premenné a pamäť (okrem argumentov volanej funkcie), jediný "stav" je stav hry.

3. Komunikácia s hráčmi – vedieť si **vypýtať "odpoveď" (ťah)** hráča.

   (a) V každom momente je práve jeden hráč "pri slove" (na ťahu). Hovoríme, že tento hráč má prioritu. Od jazyka očakávame, že nám umožňuje posúvať prioritu ďalším hráčom. (Inak hru hrá len jeden hráč ...)

   (b) Na hru sa môžeme pozerať ako na výpočet, v ktorom v rôznych okamihoch rozhodujú rôzne entity (hráči) o priebehu výpočtu. Od hráča (ktorý má aktuálne prioritu) si teda budeme pýtať hodnoty, ktoré chce "dosadiť" do premenných.

Vykonávanie hry si predstavujeme takto.

1. Na začiatku sa zavolá procedúra s názvom `init` (definovaná v popise hry). Tá nám vytvorí všetky potrebné herné objekty a počiatočný stav hry.

2. Potom sa až do konca hry bude volať procedúra `round`, ktorá odsimuluje jedno kolo hry (resp. nejakú ucelenú časť hry). Simulujeme výpočet, a keď to potrebujeme, tak si vypýtame hodnoty od hráčov. Hráčom sú samozrejme oznamované zmeny stavu hry.

3. Hra končí zavolaním špeciálnej procedúry `game_over`. Hráčom oznámime, ako sa im v hre darilo.

Poďme sa pozrieť na to, ako konkrétnejšie si predstavujeme popisovací jazyk.

1. Funkcie – `<navratovy_typ> <nazov_funkcie> <zoznam_argumentov> <telo_funkcie>` Návratový typ môže byť

   (a) `void` (ide o procedúru – nevraciame žiadnu hodnotu)

   (b) `bool` (predikát)

   (c) `int`

   (d) `<nazov_triedy>` – funkcia vráti odkaz na objekt z tej triedy

   (e) `set <zoznam_typov>` – funkcia vráti množinu alebo reláciu

2. Práca so stavom hry.

   (a) `class` – definuje triedu objektov (napríklad `policko`). V tele triedy definujeme parametre objektov (napríklad `int riadok`, `int stlpec`) a konštruktor, ktorý im dá konkrétne hodnoty.

   (b) `new` – vytvorí objekt danej triedy, a jeho parametre nastaví podľa toho, aké hodnoty dáme konštruktoru. Napríklad `new policko (1, 3)`.

(c) `attach (A, B)` – položí objekt `A` na objekt `B`. Ak to nie je možné (napríklad by sa porušila stromovitá štruktúra), tak vyhodíme chybu a končíme. V korektne naprogramovanej hre by sa to nemalo stať.

(d) `detach A` – objekt `A` sa stane voľný, a už nie je na ničom položený.

(e) `pass_to P` – posunieme prioritu hráčovi `P`.

3. Vstavané predikáty, logické spojky, ...

(a) `A > B`, `A < B`, `A >= B`, `A <= B` na porovnávanie premenných typu `int`.

(b) `A and B`, `A or B`, `not A`

(c) `A = B` – odkazujú `A` aj `B` na ten istý objekt?

4. Vstavané funkcie.

(a) Práca s celými číslami. `A + B`, `A - B`, `A * B`, `A / B`, `min (A, B)`, `max (A, B)`

(b) `parent A` – vráti odkaz na objekt, na ktorom je položené `A`. Ak je `A` voľné, vráti `null`.

(c) Práca s množinami.

i. `collect <zoznam_argumentov_dlzky_k> from <mnozina> with <k-arny_predikat>` – vráti podmnožinu k-tic (reláciu), v ktorej sú práve tie k-tice objektov, ktoré spĺňajú predikát.

ii. `size_of A` – vráti počet prvkov množiny

iii. `A times B` – vráti karteziánsky súčin

iv. `A union B`, `A except B`, `A intersect B`

v. `range (A, B)` – vráti množinu čísel $\{A, A+1, \ldots, B\}$

5. Riadiace štruktúry.

(a) `repeat <int> <telo>` – `A`-krát za sebou vykonaj telo.

(b) `if <bool> <telo>`, `while <bool> <telo>`

(c) `for <zoznam_argumentov_dlzky_k> from <mnozina> with <k-arny_predikat> <telo>` – vykoná telo pre každú vyhovujúcu k-ticu. V tele sa môžeme odkazovať na argumenty.

6. Príkazy, ktoré nie sú výpočty – "niekto zvonka" nám dodá hodnotu.

(a) `random from <mnozina>` – vráti náhodný prvok (resp. k-ticu ak to je k-árna relácia) z množiny.

(b) `chosen <zoznam_argumentov_dlzky_k> from <mnozina> with <k-arny_predikat>` – spýtame sa hráča s prioritou na k-ticu objektov. Pritom ale táto k-tica musí mať nejaké vlastnosti (aby z toho bol platný ťah).

(c) `setup <zoznam_argumentov_dlzky_k> from <mnozina> with <k-arny_predikat>` – spýtame sa **game-mastera** na k-ticu objektov s nejakou vlastnosťou. Užitočné v prípravnej fáze hry – čo ak niektorý z hráčov chce hrať za biele figúrky? Dohodne sa s game-masterom.

Ďalej sa pozrieme na niektoré obmedzenia.

1. V jazyku sa zatiaľ dajú popisovať iba hry s úplnou informáciou, teda každý hráč vie kompletný stav hry. Napríklad poker.

2. Problémom sú tiež hry, kde stav hry závisí aj od doterajšieho priebehu. V našom jazyku sa totiž nedá "pozerať do minulosti". Napríklad v *Magic the Gathering* schopnosť *echo*.

3. Nedajú sa meniť parametre objektov po ich vytvorení. Opäť ako príklad uvedieme *Magic the Gathering*: creature karta v ruke má úplne iné parametre, ako creature karta na battlefielde.

4. Hry, kde sa veľa odohráva v hlavách hráčov. (Teda je v nich nejaká informácia, ktorá nie je zachytená v tom, ako sú na sebe objekty naskladané.)

5. ...

Čo sa týka vizualizácie hry klientským programom: klient vie, ktorý objekt je položený na ktorom objekte. Podľa toho vie celkom dobre zobraziť stav hry. Na vizualizáciu slúži druhý jazyk, ktorého ciele sú nasledovné:

1. Pre každý objekt na základe jeho parametrov načíta obrázok. Napríklad keď máme šach, a v ňom `policko` s `riadok = 5` a `stlpec = 3`, tak načítame obrázok čierneho políčka. Pre nepárne `riadok + stlpec` by sme načítali obrázok bieleho políčka.

2. Majme objekt A, ktorý je položený na B. Podľa parametrov týchto objektov určíme, kde má byť vyobrazené A relatívne k obrazu B.

Ohľadne komunikácie klienta so serverom, server klientovi posiela zmeny v stave hry (`attach (A, B)`, `detach A`, `new`, `pass_to <player>`, ...) a vždy, keď server od klienta požaduje "dosadenie" (`chosen <...> from <...> with <...>`), tak sa ho spýta (pričom mu bližšie špecifikuje, aké vlastnosti musí odpoveď spĺňať).

Lepšiu predstavu o detailoch (technických či netechnických) budeme mať po implementácii.