

Język programowania do definiowania zasad gier

Jakub Stromski

W jaki sposób zdefiniować grę w postaci tekstowej?

Teoria gier to matematyczna teoria rozwiązywania sytuacji konfliktu i kooperacji.

Cechy gry:

- W grze bierze udział przynajmniej dwóch graczy.
- Każdy gracz posiada dostępne strategie, czyli algorytmy podejmowania decyzji w zależności od sytuacji.
- W zależności od wyniku gry gracze dostają wypłatę opisaną liczbowo.

Teoria gier c.d.

Cechy graczy:

- Każdy gracz dąży do zmaksymalizowania swojej wypłaty.
- Każdy gracz z założenia postępuje racjonalnie.

Podział gier:

- Liczba graczy - gry jedno i wieloosobowe
- Kooperatywność - gry kooperacyjne pozwalają na komunikację i kooperację graczy
- Dostęp do informacji - w grach z pełną informacją gracze mają dostęp do wszystkich informacji w grze, a w grach z informacją niekompletną tylko do jej części
- Czas wykonania ruchu - w grach turowych ruchy wykonywane są na zmianę, a w grach symultanicznych jednocześnie

Systemy ekspercie

System ekspercki to system służący do rozwiązywania złożonych problemów przy pomocy zestawu faktów. Ma w założeniu naśladować działania ludzkiego eksperta. System składa się z dwóch głównych elementów:

- Baza wiedzy - zawiera fakty i reguły wnioskowania
- Silnik wnioskujący - używa bazy wiedzy, dedukując kolejne fakty, w celu rozwiązania zadanego problemu

Systemy eksperckie były jednymi z pierwszych użytecznych systemów opartych na sztucznej inteligencji.

Uniwersalny model definiowania gier:

- pozwoli na łatwą implementację gry i użycie jej w dowolnym projekcie
- umożliwi symulowanie rozgrywki w badaniu skuteczności strategii (w badaniach dotyczących teorii gier)
- jest ciekawym i wymagającym zadaniem

Zakres projektu

Celem projektu jest stworzenie gramatyki języka i interpretera, który z pliku źródłowego stworzy obiekt reprezentujący grę.

Obiekt gry reprezentuje pojedynczą rozgrywkę, symuluje wszystkie procesy odbywające się w grze, wchodzi w interakcję z graczami i pozwala im wpływać na stan poprzez wykonywane ruchy. Kontroluje poprawność wykonywanych ruchów, aktualizuje stan i wydaje wypłaty po zakończeniu rozgrywki.

Kryteria akceptacji

Kryteria akceptacji:

- Gramatyka daje możliwość zdefiniowania prostych gier turowych z pełną informacją.
- Obiekt gry pozwala na przeprowadzenie symulacji rozgrywki, gracz może podejrzeć stan gry i wykonać ruch w swojej turze.
- Obiekt kontroluje poprawność wykonywanych ruchów i wystąpienie któregoś ze stanów końcowych gry.

Propozycja rozwiązania

Projekt składa się z dwóch głównych komponentów: analizatora i symulatora.

Program analizatora otrzymuje na wejściu plik tekstowy zawierający definicję gry, a na wyjściu otrzymujemy obiekt gry, czyli symulator.

Symulator jest odpowiedzialny za przebieg rozgrywki. Zajmuje się komunikacją z graczami, sprawdza poprawność ruchów, aktualizuje stan gry itd.

Analizator

Analiza pliku źródłowego odbywa się w kilku krokach:

- Analiza leksykalna rozpoznaje podstawowe elementy języka (symbole końcowe) takie jak słowa kluczowe, operatory, stałe, identyfikatory, czy nawiasy. Sprawdza czy każdy znak znajduje swoje miejsce w którymś z wzorców.
- Analiza składniowa jest odpowiedzialna za budowę drzewa składniowego z symboli końcowych na podstawie odpowiednich reguł składniowych. Sprawdza czy plik wejściowy jest poprawny pod kątem składni języka.
- Analiza semantyczna buduje obiekt gry na podstawie drzewa składniowego. Na tym etapie odbywa się łączenie komponentów za pomocą identyfikatorów, w trakcie tego procesu weryfikowana jest zgodność nazw i poprawność wszystkich komponentów składowych gry.

Symulator reprezentuje grę za pomocą czterech głównych komponentów:

- Gracze - określa liczbę i rodzaj graczy
- Stan - przechowuje informacje o stanie rozgrywki
- Zasada główna - definiuje możliwe wyniki i warunki zakończenia gry
- Ruchy - definiuje możliwe ruchy dla każdego rodzaju graczy

Gracze

Komponent określa klasy graczy i liczbę graczy z każdej klasy (nie każdy gracz musi być taki sam)

```
PLAYERS
[
    players[4];
    masters[2];
]
```

```
players:
  players_list:
    player:
      identifier: players
      integer: 4
  players_list:
    player:
      identifier: masters
      integer: 2
```

Stan gry jest przechowywany jako zbiór zmiennych. Dane wszystkich typów przechowywane są w pojedynczej tablicy. Odwołanie do konkretnej danej odbywa się za pośrednictwem mapy, która łączy identyfikator z adresem danej w tablicy i jej typem.

```
STATE
[
  INT a = 1;
  BOOL b = false;
  INT c = 5;
]
```

```
state:
  data_set:
    var_list:
      var_declaration:
        var_type:
          kw_int: INT
          identifier: a
          var_definition:
            integer: 1
      var_list:
        var_declaration:
          var_type:
            kw_bool: BOOL
            identifier: b
            var_definition:
              boolean: false
        var_list:
          var_declaration:
            var_type:
              kw_int: INT
              identifier: c
              var_definition:
                integer: 5
```

Zasada główna

Zasada główna określa listę możliwych wyników. Każdy wynik posiada warunek zaistnienia i funkcje wypłat dla każdej klasy graczy. Zasada główna jest aktywowana po każdej zmianie stanu gry i sprawdza wystąpienie każdego wyniku po kolei.

Zasada główna

```
MAIN_RULE
[
  r1
  ({ RETURN true; })
  players
  {RETURN 1;}
  masters
  {a = 2; RETURN a;}
]
```

```
main_rule:
  m_rule_list:
    m_rule:
      identifier: r1
      instruction_block:
        instruction_list:
          return_instr:
            assign_instr:
              expr_literal:
                var_definition:
                  boolean: true
          payoff_list:
            payoff:
              identifier: players
              instruction_block:
                instruction_list:
                  return_instr:
                    assign_instr:
                      expr_literal:
                        var_definition:
                          integer: 1
```

```
payoff_list:
  payoff:
    identifier: masters
    instruction_block:
      instruction_list:
        instruction:
          assign_instr:
            identifier: a
            expr_literal:
              var_definition:
                integer: 2
          instruction_list:
            return_instr:
              assign_instr:
                expr:
                  identifier: a
```


Każdy zdefiniowany ruch posiada zakres graczy, którzy mogą go wykonać, informacje niesione przez ruch w postaci zbioru danych (takiego jak stan), warunku sprawdzającego poprawność ruchu i funkcję aktualizującą stan.

Po wprowadzeniu przez gracza ruchu symulator sprawdzi jego poprawność, wywoła funkcję aktualizującą stan, wywoła zasadę główną i przekaże ruch kolejnemu graczowi.

Zasada główna

```
MOVES
[
  m1<*> [ INT a = 0; ] ({ RETURN; })
  {
    RETURN;
  }
  %
  m2<masters> [ BOOL b = false; ] ({ RETURN; })
  {
    a = (4 + s) * 7 + 9 % 5;
  }
]
```

```
moves:
  move_list:
    move:
      identifier: m1
      players_scope:
      data_set:
      var_list:
      var_declaration:
      var_type:
      kw_int: INT
      identifier: a
      var_definition:
      integer: 0
      instruction_block:
      instruction_list:
      return_instr:
      return_instr:
      instruction_block:
      instruction_list:
      return_instr:
      return_instr:
```

Zasada główna

```
move_list:
  move:
    identifier: m2
    players_scope:
      identifier_list:
        identifier: rrr
    data_set:
      var_list:
        var_declaration:
          var_type:
            kw_bool: BOOL
            identifier: b
          var_definition:
            boolean: false
    instruction_block:
      instruction_list:
        return_instr:
          return_instr:
```

```
instruction_block:
  instruction_list:
    instruction:
      assign_instr:
        identifier: a
      expr_add:
        expr_mul:
          expr:
            expr_add:
              expr_literal:
                var_definition:
                  integer: 4
            expr:
              identifier: s
          expr_literal:
            var_definition:
              integer: 7
      expr_mod:
        expr_literal:
          var_definition:
            integer: 9
      expr_literal:
        var_definition:
          integer: 5
```

Zbiór danych - implementacja

Zbiór danych wykorzystywany jest do definiowania stanu gry, ruchów oraz zmiennych lokalnych w funkcjach. Składa się z dwóch elementów:

- Jednolitej tablicy przechowującej wszystkie dane
- Mapy, która umożliwia odwołanie do zmiennej po jej nazwie. Mapa odwzorowuje nazwę zmiennej na jej pozycję w tablicy i typ

Funkcje - implementacja

Funkcje to ciągi instrukcji mogące zwracać pojedynczą wartość. Wykorzystywane są do kontroli poprawności ruchów, sprawdzania wystąpienia stanu końcowego, aktualizacji stanu gry oraz realizacji wypłat dla graczy.

Funkcje mogą posiadać zbiór danych definiujący zmienne lokalne.

Funkcje realizowane są jako graf skierowany, gdzie rozwidleniami są instrukcje skoku warunkowego.

Analiza leksykalna - Flex

Analiza składniowa - Bison

Implementacja

Język programowania - C++, flex, bison

Środowisko - MS Visual Studio

Repozytorium - GitLab

Flex, Bison, edycja pliku wejściowego - Notepad++

Pytania?