# Jezyk do gry Twilight Imperium IV
TKOM

Rafał Uzarowicz, 300282

Styczeń 2021

## 1 Opis

Implementacja prostego interpretera prostego jezyka programowania, który bedzie umożliwia łatwe zarzadzanie jednostkami i aktywacjami na hexagonalnej planszy w grze Twilight Imperium IV w celu ułatwiania pisania logiki botów do gry.

## 2 Funkcjonalności

- Możliwość obsługi samego poruszania statkami oraz żetonami na planszy co stanowi wycinek elementów rozgrywki.
- Obsługa siedmiu typów prostych - bool, int, string, color, unit, hex, planet.
- Obsługa deklaracje zmiennych oraz tablic wszystkich typów prostych.
- Obsługa zmiennych z zasiegiem.
- Obsługa niejawnego typu zmiennej - var.
- Obsługa zaawansowanych wyrażeń arytmetycznych i logicznych z implementacja porównań, mnożenia, dzielenia, dodawania, odejmowania, negacji, alternatywy, koniunkcji.
- Obsługa priorytetu operatorów dla wszystkich operatorów.
- Obsługa instrukcji warunkowej.
- Obsługa petli opartej na zakresie.
- Obsługa wbudowanych instrukcji do działania na planszy.
- Obsługa wbudowanych instrukcji sprawdzajacych stan planszy.
- Obsługa rzutowania typów.
- Obsługa definiowania własnych funkcji.

# 3 Założenia

- Interpreter wykonuje wszelkie operacje na podanej planszy i na koniec swojego działania zapisuje zmiany do tego samego pliku.
- W przypadku wyrażeń logicznych wartości typu int sa traktowane jako fałsz, jeśli beda równe 0 i jako prawda w pozostałych przypadkach, a wartości typu string sa fałszem w przypadku, gdy ciag znaków jest pusty.
- W przypadku wyrażeń arytmetycznych wartości bool sa traktowane jako 1 w przypadku prawdy i jako 0 w przypadku fałszu.
- Jezyk nie obsługuje pracy na wskaźnikach, operacji na fragmentach pamieci.
- Jezyk nie pozwala na tworzenie własnych typów.
- Cały kod programu mu znajdować sie w jednym pliku.
- Dane o planszy musza sie znajdować w jednym pliku.
- Plik musi zawierać funkcje z identyfikatorem "main", która zostanie wywołana jako poczatek programu.
- Funkcja "main" nie może przyjmować żadnych argumentów oraz musi zwrócić wartość typu int.
- Jezyk ignoruje w pełni białe znaki.
- Jezyk nie obsługuje komentarzy.
- Jezyk nie obsługuje rzutowania każdego typu na każdy typ - obsługuje tylko cześć z nich.

# 4 Gramatyka

```
#(* Main program parts. *)
module grammar;

program : functionDefinition {functionDefinition};

functionDefinition : type identifier "(" parameters ")" block;

parameters : [type identifier {"," type identifier}];

block : "{" [statement {statement}] "}";

statement : conditional | loop | functionCallStatement |
    ↪ varDeclaration | arrayDeclaration | assign | print | break
    ↪  | return | continue | block | boardChange;
```

```
#(* Changes to game state. *)
boardChange : whichPlayer playerAction ";";

playerAction : unitsAction | activation;
unitsAction : moveUnits | addUnits | removeUnits;
moveUnits : "move" unitsList fromWhere toWhere;
addUnits : "add" unitsList toWhere;
removeUnits : "remove" unitsList fromWhere;
activation : activateHex | deactivateHex;
activateHex : "activate" "(" value ")";
deactivateHex : "deactivate" "(" value ")";

fromWhere : "from" "(" value ")";
toWhere : "to" "(" value ")";

unitsList : "(" unitAmount {"," unitAmount} ")";
unitAmount : value ":" value;

#(* Checking game state. *)
boardStateCheck : playerStateCheck | planetOrHexStateCheck |
    ↪ hexActivationCheck;

hexActivationCheck : whichPlayer "activated" "(" value ")";

planetOrHexStateCheck : (whichPlanet|whichHex) "has" "(" value ")
    ↪ ";

playerStateCheck : whichPlayer "has" playerUnitsCheck;
playerUnitsCheck : "(" value ")" "at" "(" value ")";

whichPlayer : "player" "(" value ")";
whichPlanet : "planet" "(" value ")";
whichHex : "hex" "(" value ")";

#(* Specific values for unit, hex and planet. *)
unit : ships | ground | structures;
ships : "Fighter" | "Destroyer" | "Carrier" | "Cruiser" | "
    ↪ Dreadnought" | "WarSun" | "Flagship";
ground : "Infantry";
structures : "SpaceDock" | "PDS";

color : "Red" | "Yellow" | "Green" | "Blue" | "Purple" | "Black"
    ↪ | "NoColor";

hex : "h" "0'-'50";
```

```
planet : "p" "0'-'58";

#(* Basic statements. *)
conditional : "if" "(" orCondition ")" block ["else" block];
loop : "foreach" "(" typeOrVar identifier ":" identifier ")"
    ↪ block;

varDeclaration : typeOrVar identifier "=" orCondition ";";
arrayDeclaration : typeOrVar "[" "]" identifier "=" type "["
    ↪ nonZeroNumber "]" ["{" [value {{"," value}}] "}" ] ";";
typeOrVar : type | "var";
assign : identifier ["[" number "]"] "=" orCondition ";";
functionCallStatement : identifier "(" arguments ")" ";";
arguments : [orCondition {"," orCondition}];
print : "print" "(" [orCondition {"," orCondition}] ")" ";";
break : "break" ";";
continue : "continue" ";";
return : "return" orCondition ";";

#(* Operators' order. *)
orCondition : andCondition { or andCondition };
andCondition : equalityCondition { and equalityCondition };
equalityCondition : relationCondition { equality
    ↪ relationCondition };
relationCondition : addExpression { relation addExpression };

addExpression : multiplyExpression { add multiplyExpression };
multiplyExpression : unaryExpression { multiply unaryExpression};

unaryExpression : expression | notUnaryExpression |
    ↪ negativeUnaryExpression;
notUnaryExpression : not unaryExpression;
negativeUnaryExpression : negative unaryExpression;

expression : value | "(" orCondition ")";

#(* Basic operators. *)
or : "||";
and : "&&";
relation : ">" | "<" | "<=" | ">=";
equality : "==" | "!=";

add : "+" | "-";
multiply : "*" | "/";

not : "!";
```

```
negative : "-";

#(* Variables and literals. *)
value : literal | functionCall | variableReference |
    ↪ boardStateCheck;
functionCall : identifier "(" arguments ")";
variableReference : identifier ["[" value "]"];
type : "int"|"string"|"bool"|"unit"|"color"|"hex"|"planet";
identifier : ((underscore (letter | digit | underscore)) |letter)
    ↪  {letter | digit | underscore};

literal : number | string | bool | unit | color | hex | planet;

number : nonZeroNumber | "0";
nonZeroNumber : nonZeroDigit {digit};
string : "\"" {character} "\"";
bool : "true" | "false";

character : letter | digit | special;
letter : 'a'-'z' | 'A'-'Z';
digit : "0" | nonZeroDigit;
special : underscore | "." | "," | "-" | " " | "/" | "\\";
nonZeroDigit : '1'-'9';
underscore : '_';
```

# 5 Moduły

## 5.1 Moduł analizatora leksykalnego

Jest to moduł, który na podstawie danych odebranych od modułu źródła tworzy tokeny na podstawie gramatyki jezyka i przekazuje je dalej do modułu analizatora składniowego.

## 5.2 Moduł analizatora składniowego

Moduł ten pobiera tokeny od modułu analizatora leksykalnego i na ich podstawie buduje drzewo programu zgodnie z gramatyka jezyka. Drzewo to zostaje przekazane do modułu interpretera. Wykorzystywany jest analizator rekursywny zstepujacy.

## 5.3 Moduł interpretera

Jest to moduł zajmujacy sie samym wykonywaniem programu. Pobiera on drzewo programu od modułu analizatora składniowego i rozpoczynajac od funkcji

”main” wykonuje instrukcje zapisane w kodzie. Dokonuje on także analizy semantycznej.

### 5.4 Moduł planszy

Jest to moduł, który odczytuje/zapisuje plansze z/do pliku oraz jest wykorzystywany przez moduł interpretera podczas wykonywania kodu. Przechowuje on stan planszy.

### 5.5 Moduł źródła

Jest do moduł który podaje do analizatora leksykalnego kolejne znaki kodu oraz zaznacza koniec źródła.

### 5.6 Moduł wyjatków

Jest to moduł zawierajacy implementacje wszystkich wyjatków wykorzystywanych na różnych etapach interpretacji.

### 5.7 Moduł dodatków

Jest to moduł zawierajacy dodatkowe funkcje, struktury danych oraz słowniki, które sa używane przez pozostałe moduły.

## 6 Struktury danych

### 6.1 Token

Jest to struktura utworzona przez analizator leksykalny na podstawie kolejnych znaków ze źródła i przekazywana do analizatora składniowego.

### 6.2 Pozycja

Jest to struktura przechowujaca pozycje w kodzie. Jest ona używana przy wyświetlaniu pozycji błedu.

### 6.3 Drzewo programu

Jest to struktura tworzona przez analizator składniowy na podstawie tokenów odebranych od analizatora leksykalnego. Na podstawie wielu klas reprezentuje ona pełne drzewo rozbioru interpretowanego programu. Ta struktura przekazywana jest do interpretera. Na jej podstawie beda wykonywane instrukcje.

### 6.4 Strumień wyjściowy

Jest to struktura, która interpreter wykorzystuje jako swoje standardowe wyjście. Struktura ta zależnie od implementacji interfejsu może wypisywać albo zapisywać wyjście interpretowanego programu.

## 7 Obsługa błedów

Na różnych poziomach działania program różnie reaguje na błedy. W analizatorze leksykalnym zostanie wyświetlona dokładna pozycja nierozpoznanego znaku/niezgodnego z gramatyka napisu. W analizatorze składniowym zostaje wyświetlona informacja z dokładnym miejsce w kodzie, w którym nastapił problem ze zgodnościa składniowa z gramatyka jezyka. Na poziomie interpretera błedy wyświetlaja informacje z linijka, w której wystapił problem w trakcie wykonywania.

## 8 Testy

Dla modułów planszy, źródła, dodatków, analizatora leksykalnego i składniowego zostały napisane testy jednostkowe.

Dla modułu interpretera zostały napisane testy integracyjne.

Wszystkie testy łacznie zapewniaja całkowite pokrycie kodu powyżej 90%.

## 9 Sposób realizacji

Program został napisany w jezyku Java z użyciem dodatkowej zewnetrznej bilioteki json-simple w celu wygodnego wczytywania/zapisywania stanu planszy.

## 10 Struktura pliku planszy

Plik planszy jest plikiem w formacie JSON, który zawiera w sobie dwie tablice - jedna dla planet, a druga dla heksagonów. W ramach każdej planety znajduje sie informacja o liczbie jednostek danego gracza na danej planecie, a w ramach heksagonów poza informacja o jednostkach jest także informacja o tym czy dany gracz aktywował ten heksagon. Jeśli w pliku nie znajduja sie informacje o danej planecie/heksagonie, to jest to traktowane tak jakby nie było tam jednostek. Plik przechowuje tylko informacje o jednostkach jeśli jest ich dodatnia liczba.

## 11 Sposób uruchomienia

Program jest uruchamiany w konsoli. Programowi trzeba podać ścieżke do pliku z kodem źródłowym oraz ścieżke do pliku przechowujacego stan gry. Program

wyświetla ewentualne komunikaty o błedach oraz na końcu zapisuje zaktualizowany stan rozgrywki do pobranego wcześniej pliku.

Program uruchamia sie wykorzystujac plik typu jar.

Komenda do uruchamiania:
java -jar ITI4.jar ”code file path” ”board file path”

# 12 Przykładowe programy

```
int factorial( int n ){
    if( n <= 1 ){
        return 1;
    }else{
        return n * factorial(n-1);
    }
}

int main(){
    int[] arr = int[10];
    int i = 0;
    foreach( var x : arr ){
        x = i;
        i = i + 1;
    }
    foreach( var x : arr ){
        x = factorial(x);
        print(x, " ");
    }
    print("end");
    return 0;
}
```

```
int increment(int x){
       print("Before incrementation: ", x);
       return x+1;
}

color whoOccupy(hex h, color p){
       if( player(p) activated (h) ){
              return p;
       }
       return NoColor;
}
```

```
int main(){

        color currentPlayer = Black;
        unit u = Carrier;
        bool isActivated = player(Red) activated(h2);
        int fightersCount = player(Green) has (u) at (h17);

        if( isActivated && fightersCount > 2 || planet(p4) has(
            ↪ Fighter) ){
                player(currentPlayer) activate (h12);
                player(currentPlayer) add(Fighter:3, Carrier:1,
                    ↪ Flagship:1) to (h12);
                player(currentPlayer) move(Destroyer:2, WarSun:1)
                    ↪ from (h9) to (h12);
        }else{
                int destroyedFighters = player(Red) has(u) at (h17)
                    ↪ ;
                if(increment(destroyedFighters) <= 1){
                        player(Black) remove(Fighter:
                            ↪ destroyedFighters) from (h17);
                }
        }

        var[] players = color[3]{Black, Red, Blue};
        var[] units = unit[2];
        units[0] = Fighter;
        units[1] = Infantry;

        var sourceHex = h3;

        var enemy = whoOccupy(h0, Red);
        foreach(var p: players ){
                enemy = whoOccupy(h0, p);
                print(" ", p);
    }
        if(enemy == Red){
                player(Green) move(units:1) from(sourceHex) to(h29)
                    ↪ ;
        }

        return 0;
}
```

```
int increment(int x){
```

```
       print("Before incrementation: ", x);
       return x+1;
}

int main(){
    color[] players = color[2]{Red, Blue};
       if( planet(p4) has(Fighter) ){
              foreach(var p:players){
                     print(p);
       }
       }else{
       print("Tak");
       }
       var[] units = unit[2];
       units[0] = Fighter;
       units[1] = Infantry;

       print(" ", units[0], " ", units[1]);

       return 0;
}
```

```
int factorial( int n ){
    if( n <= 1 ){
       return 1;
    }else{
       return n * factorial(n-1);
    }
}

int main(){
    int[] arr = int[3];
    int i = 0;
    foreach( var x : arr ){
       x = i;
       i = i + 1;
    }
    hex h = h7;
    color p = Red;
    unit u = Fighter;
    foreach( var x : arr ){
       x = factorial(x);
       player(p)add(u:x, Carrier:1)to(h);
    }
    print(player(p)has(u)at(h), " ");
```

```
    print(player(Red)has(u)at(h), " ");
    print(player(p)has(Fighter)at(h), " ");
    print(player(p)has(u)at(h7), " ");
    print(player(p)has(Carrier)at(h7), " ");
    print("end");
    return 0;
}
```

```
int factorial( int n ){
    if( n <= 1 ){
        return 1;
    }else{
        return n * factorial(n-1);
    }
}

int main(){
    int[] arr = int[4];
    int i = 0;
    foreach( var x : arr ){
        x = i;
        i = i + 1;
    }
    hex h = h7;
    color p = Red;
    player(p)activate(h);
    foreach( var x : arr ){
        x = factorial(x);
        if(x>1){
            if(x>=6){
                print(x, "if2 ");
            }else{
                print(x, "else2 ");
            }
        }else{
            print(x, "else1 ");
        }
    }
    print(player(Red)activated(h7), " ");
    print(player(Black)activated(h7), " ");
    print("end");
    return 0;
}
```

```
int fibtorial( int n ){
```

```
    if( n <= 0 ){
        return 0;
    }
    if( n == 1 ){
        return 1;
    }
    return factnacci( n - 1 ) + factnacci( n - 2 );
}

int factnacci( int n ){
    if( n <= 1 ){
        return 1;
    }else{
        return n * fibtorial(n-1);
    }
}

int main(){
    int[] arr = int[5];
    int i = 0;
    foreach( var x : arr ){
        x = i;
        i = i + 1;
    }
    foreach( var x : arr ){
        x = factnacci(x);
        print(x, " ");
    }
    print("end");
    return 0;
}
```

```
int pow( int x, int n ){
    if( n == 0 ){
        return 1;
    }
    return x * pow( x, n-1 );
}

int main(){
    int[] arr = int[5];
    int i = 0;
    foreach( var x : arr ){
        x = i;
        i = i + 1;
```

```
    }
    foreach( var x : arr ){
        x = pow(x, 3);
        print(x, " ");
    }
    print("end");
    return 0;
}
```